The Journal of Systems and Software 162 (2020) 110496

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

In Practice

A framework for pervasive computing applications based on smart objects and end user development

Christos Goumopoulos[a,c,*], Irene Mavrommati[b]

[a] Department of Information & Communication Systems Engineering, University of the Aegean, Greece
[b] Hellenic Open University, School of Applied Arts, Greece
[c] Computer Technology Institute and Press, Diophantus, DAISy Research Unit, Patras, Greece

ARTICLE INFO

Article history:
Received 16 April 2019
Revised 9 December 2019
Accepted 12 December 2019
Available online 13 December 2019

Keywords:
Framework
End User Development
Pervasive computing
Smart objects
Component-based system engineering
User evaluation

ABSTRACT

Pervasive Computing (PerComp) research remains to this date technology-centric, requiring more focus on utilizing human and societal intelligence. To bridge this gap we motivate the need for a conceptual framework that provides a vision for enabling end-users to participate actively in the design of PerComp applications. The framework in particular provides guidance for the development of tools that allow end-users to configure their own smart environments. It emphasizes the benefits of using Smart Objects (SOs) as components of PerComp applications under a system engineering perspective and the benefits of using affordances as dynamic connectable capabilities under a user experience perspective. A generic application model is developed within the framework that manifests the concepts governing the structure and operation of applications composed by connecting SOs together and adhering to a rule-based behavior. A multi-layered software mediator is defined to provide a platform for the runtime support of such PerComp applications. Theoretical foundations for the key concepts adopted regarding the interaction design and system engineering perspectives are discussed and the results of a user evaluation study focusing on the perceived usability of the high-level conceptual models and the developed tools are reported.

1. Introduction

Pervasive Computing (PerComp) environments constitute complex computing environments; they can sometimes be modelled and understood as ecologies of Smart Objects (SOs), services, people and infrastructure (Goumopoulos and Kameas 2009). Smart environments with such complexity can be effectively approached as component-based systems (Papadopoulou et al., 2012; Bergesio et al., 2017), namely systems with loose coupling of interchangeable parts, which communicate via interfaces (Heineman and Councill 2001). In this context, a PerComp application can be seen as an ecology of interconnected SOs and services. From an engineering perspective this approach is advantageous for scaling and is extensible, in terms of production, by many different manufacturers of different components.

SOs in the context of this research are physical objects enhanced with Information and Communication Technology capabilities in order to determine the context of their surroundings, communicate with peer SOs, and make decisions upon affecting their environment. Such SOs, because of their inherent connectivity, can support additional 'augmented' functions, more tailored to their owners' lifestyle, making user empowerment an important factor in their sustainable adoption. For example, a technology-augmented mug could be used by its owner for drinking, but also in cooperation with other objects can form a system assigned to carry out health monitoring. Or, moved by a new owner to a different environment, the mug could be part of a different aggregation of objects, this time fulfilling an application that facilitates ordinary tasks, such as in case the tea level drops, it might assign to a smart kettle the task of switching itself on, for making more tea.

Significant progress has been made in terms of hardware and software research and development affecting the PerComp vision (Satyanarayanan 2001), so we now find ourselves surrounded by smart environments (Cook and Das 2007), almost as seamlessly as Marc Weiser envisioned when comparing it to "a walk in the woods…" (Weiser 1991). On the other hand, PerComp applications need to adapt their behavior to dynamic context, to the characteristics of diverse environments and to circumstances unanticipated by application developers. A reported solution is to enable users to restructure their PerComp applications (Kameas and Mavrommati 2005; Gross and Marquardt 2007; Barricelli and Valtolina 2017; Chen and Li 2017). This approach has several benefits: (a) users are

* Corresponding author.
E-mail address: goumop@aegean.gr (C. Goumopoulos).

https://doi.org/10.1016/j.jss.2019.110496
0164-1212/

able to proactively design their own augmented environments and experiences, but also may be able to better understand and safeguard their security and privacy needs; (b) applications are better adapted to users' idiosyncratic needs and requirements; (c) users can progressively improve their applications in a creative way.

The need to let users shape their PerComp experiences is not new but it is with the advent of the Internet of Things (IoT) and the increasing amounts of commercially available smart devices that these challenges have scaled up, and the research area of End User Development (EUD) for smart environments is now starting to be focused on (Markopoulos et al., 2017). To address this challenge further and bridge part of the gap between lab-produced-technology and users, we motivate the need for a conceptual framework for PerComp applications that provides a vision for enabling end-users to participate actively in applications design. Adding human and societal intelligence as part of application development is a way to explore emergent uses and potentially benefit from coupling the system's intelligence with the human intelligence. A framework should therefore consider such systems in a holistic approach, which includes people, conceptual models, principles, methodologies, design patterns, intelligent mechanisms and supporting tools, as interconnected elements that evolve synergistically through time.

In this paper we provide the rationale of such a broad conceptual framework which builds on well understood theories, methodologies and research emanated in domains such as cognitive science and educational psychology, while adaptively applied in the field of Human Computer Interaction. The framework identifies a concerted structure of concepts, methodologies and tools that on the one hand pertains the different elements of PerComp systems integration from an engineering perspective and on the other hand allows end-users to have an active role in the design of PerComp applications from a user experience perspective. An objective of this framework is to facilitate the communication between members of multidisciplinary development teams and to guide the development of end-user supporting tools. Such tools should provide selective transparency and 'seamfulness' (as in: exposing the seams) into the working of PerComp applications, by providing methods, conceptual models and interface mechanisms, which can help people to gain a greater degree of understanding of, and ease in handling the issues in the smart environment they live in.

The main contributions can be summarized as follows:

- A framework that addresses the design challenge of EUD in PerComp domain, in a holistic way by integrating different perspectives in its structure, i.e. system engineering aspects as well as user interaction and system design principles; A generic application model is developed within the framework that manifests the concepts governing the structure and operation of applications composed by connecting SOs together and adhering to a rule-based behavior.
- EUD tools that embrace the framework principles and provide high-level metaphors for composing PerComp applications and expressing their rule-based logic enabling thus the creative participation of end-users in the development process; A multi-layered software mediator is defined to provide a platform for the runtime support of such PerComp applications.
- An evaluation study of the developed EUD approach focusing on the perceived usability of the high-level conceptual models and the EUD tools indicating that the resources provided to the users are adequate to enable them successfully complete development tasks in PerComp environments.
- A set of design guidelines that can be used as a basis for designers and developers of EUD tools and applications in PerComp domain.

The rest of the paper is organized as follows. First related research is discussed, followed by a reference on previously developed applications in different domains such as smart home, precision agriculture and ambient-assisted living, as their design considerations were influencing in exploring elements of the framework while shaping its versatile and generic characteristic. The next section introduces the proposed EUD framework in PerComp domain. First, the rationale of the framework is provided, followed by the elaboration on the different perspectives that are integrated in its structure. A discussion is provided, next, regarding the benefits of using SOs as components of PerComp applications with dynamic connectable capabilities, results of a user evaluation study, an outlook of design guidelines for PerComp EUD domain experts and open issues that call for future research. Finally, conclusions from this research are outlined.

2. Related work

EUD, although has been extensively studied as a research area in Human Computer Interaction, for PerComp and smart environments is a currently upcoming research area (Desolda et al., 2017); EUD research specifically targeted to the domain of smart spaces has been suggested (Mavrommati and Darzentas 2007; Fogli et al., 2016; Markopoulos et al., 2017) and reported in systematic mapping studies (Barricelli et al., 2019), while broader theoretical foundations or frameworks targeting EUD in PerComp environments are not yet formulated. EUD frameworks exist, but are either generic for the application domains of EUD (Fischer 2012) or have a focus in virtual environments (Mugellini et al., 2009). Specific PerComp frameworks on the other hand address only the component-based or service-oriented architecture aspects of the system (Becker et al., 2004; Kalofonos and Wisner 2007; Kameas et al., 2009). A broader framework approach, that addresses the mutual co-evolution between people, EUD tools (mental or physical), and PerComp technology, is not the norm in literature.

A design framework for smart environments based on the Ecological Approach is presented in (Kymäläinen 2015). The framework provides a do-it-yourself (DIY) setting that enables and motivates users to build their own smart experiences employing remote and immediate design strategies. Three case studies in the realm of ambient assisted living are discussed as proof of concept prototypes. The approach draws upon the DIY communities of practice which aim to create and share projects, a process which is boosted by the capabilities offered by social networking platforms and ubiquitous Web. The emphasis was given to user experience research in terms of determining the design steps required to engage users in the development process. Our framework has a broader scope providing relations to theories and methods that underpin EUD tasks, distinguishing between end-user design and end-user programming.

A framework for the rapid creation of smart environments by mushing up off-the-shelf smart devices is discussed in (Kubitza and Schmidt 2015). The concept is to facilitate the prototyping of new applications based on existing SOs by abstracting the complexity and heterogeneity of the different platforms that may be found in smart environments. In this way the users can concentrate more on the implementation of the application logic. The framework enables the dynamic alteration of an application's behavior by providing tools to add new context aware rules. A web-based tool has been designed to assist in the management of different components of an application such as devices, triggers in the form of conditions and events, using JavaScript to express the application logic. Consequently, the current version of the developed toolkit is more useful to programmers rather to typical end-users

who may not have programming experience as is the case with our approach.

Application composition in smart environments can be done either automatically with the support of intelligent services or interactively where end-users are explicitly involved in the configuration of their smart environments with the mediation of EUD tools. A thorough list of such EUD tools and related interfaces are discussed in (Davidyuk et al., 2015). In the automated composition user involvement is negligible since the system organizes and delivers most of the required functionality autonomously. However, there are limitations and an assurance cannot be really provided that the resulting services will always meet the expectations of the users. In the interactive composition approach, which is the strategy followed in the presented research, people are in control of the application composition albeit of some extra effort that may be required on their side. To assist end-users in the composition task, EUD tools are endorsed with cognitive clues in the form of metaphors which map programming constructs to concepts that are well known in the physical world and thus users can reasonably learn and apply. Various metaphors have been recommended in the relevant literature such as jigsaw puzzle, pipeline, join the dots and timeline (Danado and Paternò, 2014; Davidyuk et al., 2015). The high-level conceptual model presented here, is a form of the join the dots metaphor which is characterized by its simplicity regarding the visual representation of devices' capabilities and link possibilities in the context of an editor tool. Additionally the proposed framework allows the definition of emergent properties with a rule-based approach which enables the programming of more complex application behaviors grounded on the application and rule models presented in this paper.

Rule-based EUD empowers people with the ability to adapt the original behavior of smart devices conceived at design time at the time of their actual usage. In this way users can receive personalized services and feel more pleasure when using PerComp applications created out of context-aware SOs. The trigger-action programming style which is based on event-condition-action (ECA) rules has been adopted by a number of frameworks to handle the configuration of smart spaces (Huang and Cakmak 2015; Ghiani et al., 2017). Rule-based EUD prototypes have been demonstrated successfully also in domestic environments (Ur et al., 2014; Coutaz and Crowley 2016). A popular platform in this category is IFTTT (IF This Then That) which provides end-users with a web-based environment to define simple ECA rules, called recipes, following a wizard approach (IFTTT 2019). Each rule involves a single service that is tracked by the platform for detecting the occurrence of a specific event and another service that executes a certain action by reacting to the triggered event. IFTTT favors simplicity over expressiveness and thus does not handle multiple events or actions in the same rule. Compared to composition models like IFTTT and puzzle our approach demonstrates more expressive power by allowing the handling of multiple events in a structured manner as a means to empower end-users to customize the composition of meaningful applications. The puzzle metaphor, on the other hand, is confined to a functionality composition that depends on what service the specific puzzle piece can provide. Moreover, puzzle pieces provide only a bounded number of sides (i.e. interfaces), thus limiting the set of potential emerging expressions.

A design challenge is to protect users from confusion when they have to differentiate between similar concepts, for example to discriminate between the editing tasks of conditions and actions. The rule editor included in the proposed EUD tools uses a visual representation of conditions and actions blocks making a clear separation between them and provides a structured representation of related notions (e.g. parameters, states, events, actions) in the context of visual editors to facilitate rule building. Whereas, other approaches that use operators and natural language constructs to differentiate between similar concepts (e.g. events and state conditions) report user difficulties in their attempt to choose the proper option in all cases (Ghiani et al., 2017).

Another high-level EUD strategy that has been applied in smart environments is programming by example (Dey et al., 2004; Chen and Li 2017). The user in this programming style demonstrates a number of interactions on a specific example and the system infers what the corresponding application behavior is. The generalized logic deduced by the example can then be re-used in analogous situations. Generalization is usually supported by intelligent agents that observe interactions in the PerComp environment and can subsequently infer patterns that are converted to applications as demonstrated in (Chin et al., 2006). In interactive systems a design challenge however is the provision of meaningful examples by inexperienced users in order to infer the correct control logic. In the present work end-users are empowered to specify rules using context parameters and change the behavior of services provided in smart environments.

As mentioned previously, existing EUD methodologies consider smart environments mainly as technology enhanced spaces that can be manipulated by user-friendly tools often based on the ECA programming model. This technology-focused consideration, however, disregards the intrinsic complexity of the real world in terms of the human and societal factor, which accordingly reduces the opportunities for conceptualizing prospective design features of smart environments for user action. The proposed EUD framework alleviates this gap, as will be discussed in Section 3.2, by considering Activity Theory and other related conceptual frameworks as the underlying base for grounding EUD in PerComp domain. This resulted in the supporting of a range of abstraction levels and application specification methods to address the variety of end-users requirements and skills.

2.1. Background

The conceptual and applied parts of the proposed framework have been progressively developed over several years to form a new EUD method in the PerComp domain. In this vein, the development of PerComp applications in various domains and primarily their design processes were influencing in exploring elements of the framework while shaping its versatile and generic nature. In the following a reference to such background work is provided.

Applications in the domain of smart home for everyday living support (Hagras et al., 2012). The common characteristic of these applications is the adoption of the concept of component architectures into smart environments, whereby the components are smart devices which are purposefully composed into smart home applications. The emphasis, however, in this background work is the use of intelligent agents to support adaptive task realization and enhanced human-machine interaction for deciding which modalities should be instantiated by the use of which devices and how - i.e. planning, allocation and instantiation. On the other hand, the software mediator discussed in the current work (Section 3.4.4) inherits particular components for service management and invocation as developed in the ATRACO communication middleware (Hagras et al., 2012). This provides a baseline for the application and interaction layers described here.

Awareness applications that collect awareness information from sensors and devices in smart environments to convey it to remote users using a multitude of awareness objects found at the user space (Goumopoulos et al., 2012); *Precision agriculture applications to monitor and manage resources* (e.g. *water) in crop cultivation* (Goumopoulos et al., 2014); *Ambient assisted living applications for monitoring vital parameters and performing medical protocols* (Goumopoulos and Lappa 2017). In all these cases, appli-

cation domain specific models were conceptualized, as, for example, the ASTRA awareness model to describe awareness applications based on domain specific concepts such as the focus/nimbus concepts for modeling social aspects of awareness systems and the façade concept for handling information filtering (Goumopoulos et al., 2012). End-users have also exploited the functionality of preliminary versions of rule editors to express the rule logic of domain-specific applications as in the case of the PLANTS project (Goumopoulos et al., 2014). The current paper extends our previous work by providing a theoretical underpinning of the conceptual models governing the interaction design aspects of the EUD framework (Section 3.2 and 3.3) and by proposing a mapping of the inspired concepts hierarchies into a generic application model regarding the system engineering aspects (Section 3.4.2). In this way, the developed EUD tools (Section 3.4.3) are able to capture the PerComp characteristics of applications without being limited to the prescribed set of class hierarchies used in domain specific applications. Furthermore, the mapping of such conceptual hierarchies into the interaction layer of the proposed software mediator give EUD tool implementers new insights on the representations of the hierarchical concept categories required to decrease the cognitive effort related to the programming of application logic.

3. EUD framework in percomp domain

3.1. Rationale

A basic assumption embraced by the proposed framework for EUD in PerComp domain is that design and usage of systems reciprocally form each other in continual socio-technical processes. This is in contrast to the traditional approach where engineering experts predetermine the user needs by creating artifacts at design time that users have to put up with at usage time. Software engineering methodologies based on user-centered and participatory design (Sanders 2002), alleviate such limitations by enabling the active participation and social inclusion of end-users at the design process with benefits measured on the improved acceptability of the developed artifacts. Even though such methodologies are valuable, still, PerComp environments and applications are dynamic systems that are characterized by evolution at use time in order to meet new requirements, cover varying tasks, manage different users, contexts and changing needs, as well as incorporating new technologies. Therefore, these characteristics of PerComp domain justify the consideration of a framework which in addition to the technical/technological dimension embraces human and societal intelligence. Consequently, an EUD framework that addresses the mutual co-evolution between people, tools (mental or physical), and PerComp technology will be of a broad nature and draw influences from different scientific domains for its theoretical foundations.

The conceptual part of the proposed framework is addressed to the different disciplines that collaboratively (but not always smoothly) work to create PerComp systems and tools such as System Developers, Engineers, User Experience Designers, User Interface and Interaction Designers, and Experts from Human Sciences (i.e. Psychology, Sociology). The proposed framework, in particular, attempts to give to engineers and designers of software systems, an overview of the necessary theoretical foundations to support, under the articulated basic assumption, PerComp EUD system engineering, as well as a toolbox of conceptual and methodological means, to facilitate common understanding between the different disciplines that are involved. Moreover, it provides to them insights on how the approach taken here has enabled the modeling of design experience and interaction problems in effective ways and has allowed the end-users to actively participate in EUD activities under the support of socio-technical methodologies. Besides that, the applied part of the framework led to the deployment of the developed EUD tools and the supporting software mediator in realistic settings which enabled the end-user evaluation study reported in this paper. To the best of our knowledge, there have been no reports in the scientific literature on frameworks with such characteristics for EUD in the PerComp domain, although frameworks exist for the general-purpose EUD systems (Fischer et al. 2012).

The proposed framework pertains PerComp environments that support EUD and have SOs as their application's components. The framework takes a broad view specifying the fundamental components which are integrated in its structure under different perspectives. The *interaction design perspective* focuses on concepts and models for addressing people's interaction with such mixed ecosystems as the PerComp applications that involve SO components. The *system engineering perspective* examines the architectural aspects of such dynamic systems and the associated tools supporting their realization. Key concepts of related *underlying theories* are reflected in adopted interaction design models and system engineering constructs, whereas concepts that can act as bridges between the different perspectives are proposed. Fig. 1 synthesizes the structure of the framework reflecting also the structure of the following sections that discuss its components.

3.2. Underlying theories

Theoretical foundation is divided into three broad categories key concepts of which provide a structuring basis for the proposed conceptual framework and a guiding support for the design of EUD tools in PerComp domain.

3.2.1. Activity theory

Activity Theory (AT) originated from Russian psychology (Leontiev 1978) and through the years it has been adapted in the research of multiple domains including educational research (Engeström 1987) and Human Computer Interaction (Kaptelinin 1996). Recently, the appropriateness of AT principles has been examined for the design of EUD tools in smart homes (Demeure et al., 2014).

AT is a broad conceptual framework, viewing activities as the focal point of consideration for collaborating entities in social settings. This collaboration is further enabled with the use of physical and/or mental tools (Bannon and Bødker 1989). AT considers in particular human activity and consciousness that is shaped through practice in the related environmental context. Social factors are seen as catalytic for forming interactions between people and their environments. Shaping principles of AT are object-orientedness, the dual concepts of internalization / externalization e.g. home automation alters the movement of inhabitants as well as their mental model of home (Yarosh and Zave 2017), tool mediation, hierarchical structure of activity, and continuous development (Kaptelinin and Nardi 1997). Tool mediation is a central concept in AT since it shapes the way people interact with the world. Other people's experiences are reflected on their tools, through a process of adaptation and modification, so as to make them more efficient for the specific tasks they were aimed for. This process is aggregated in the tools properties (structure, form, etc.) and in the usage awareness of a tool.

The principles of AT guide the formation of the proposed EUD framework in several ways:

- First, since activities are key notions that are motivated by people needs (e.g. study a homework) they should be represented directly in the EUD structure.

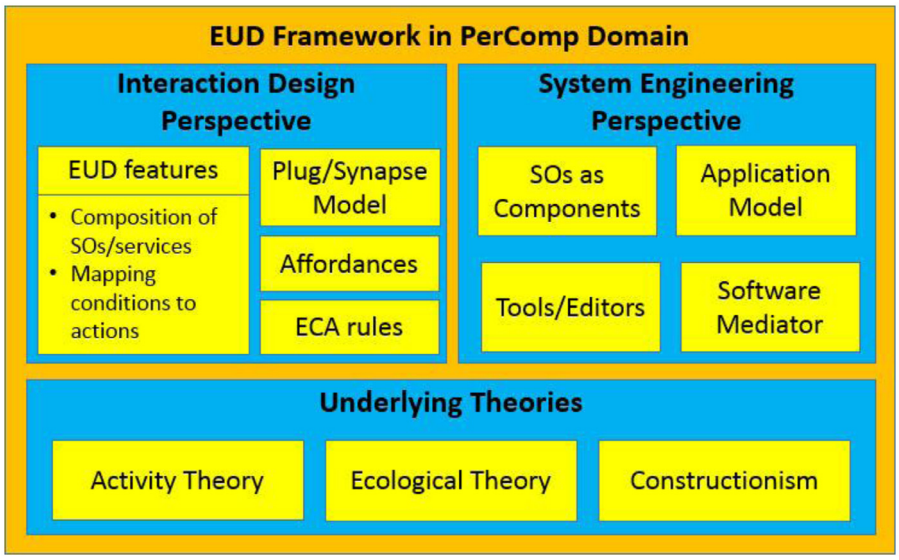

Fig. 1. EUD framework in PerComp domain overview.

- Second, since activities follow a hierarchical structure (e.g. an activity is comprised of goals and each goal is comprised of conditions), EUD can be considered in different levels likewise. In that sense, EUD is analyzed in design and programming tasks as will be elaborated later. The former is associated with the expression of relationships between an activity and its constituent goals, whereas the latter is associated with the expression of conditions under which a goal is attained.
- Third, since an activity represents a purposeful context for its constituent elements, it determines the way physical and digital resources involved are used. For example, accessing of a parlor display service and lights is relevant for an activity "viewing holiday photos" and less relevant for an activity "dinning with friends". In other words, resources in a smart environment (e.g. services, devices, sensors) should be represented and accessed in several ways, based on the supported activity.
- Fourth, since activities are mediated by tools, EUD gives emphasis to the significance of providing support for cooperation, development, and tool appropriation. Moreover, EUD should enable users to build their own tools (e.g. connect a calendar service with a robotic sweeping machine to attain the housekeeping goal). With the progress of time this leads to a co-evolution of PerComp systems with their users according to their changing requirements, their evolved knowledge and perceived experiences as well as with the availability of new devices. Potential uses of tool mediation include the finding and customizing collaboratively created applications from communal application pools.
- Fifth, since human cognition and intelligence is not only individual but lies within the social and physical environment, the notion of community is an essential extension of AT (Vygotsky 1980; Engeström 1987). Within a social group (e.g. a family), different members share cognitive processes. Time has an impact on cognition too, as it is the earlier events that form the later ones. Cognition is affected by the coordination of all these parameters (people, processes, artifacts, time). For example, the coordination of a common activity like "preparing dinner" will be mediated in the form of sub goals division. EUD, as well as the actual use of a PerComp system, can be addressed by considering users at the center of this coordination. It emphasizes the cooperative and participatory design of PerComp applications, for a social group, who could explicitly or implicitly mediate to the development, by employing various social tools, such as cooperation, antagonism and reciprocation.

3.2.2. Ecological theory

PerComp vision calls for new interaction models that will assist users to shape the context-dependent interactions in SO ecologies. Situated Cognition (SCog) theory argues that knowing and doing are inseparable since all knowledge is situated in activity bound to social, cultural and physical contexts (Brown et al., 1989; Greeno and Moore 1993). Following this insight one can model SOs based on the assumption that they are situated and used in the context of a SO ecology which allows for the reduction of interaction complexity within the ecology and allows for a reasonable cognitive effort by the users on their attempt to control and manage such an environment.

Ecological theory has influenced SCog theory and more specifically Gibson approach on visual perception which posits that a large amount of information is viewed selectively (Greeno 1994; Gibson 2014). Perceptions are geared by people's intentions, society playing a role also, and they evolve through time. It is exactly these perceptions that define the possible usage of an object. Taking such a stance permits the replacement of conventional input/output approaches by more appropriate to PerComp environments concepts such as the *perception and action cycle* (Gibson 2014). For example, in smart environments several activities may concurrently take place some of them overlapping and other performing independently. Each activity may require simultaneously multiple inputs regarding perception and involve several outputs for actuation. On the other hand, the sequential processing of traditional input/output would have been inconvenient to cope with such requirements.

In particular the Ecological approach to perception offers the concept of *affordances*, introduced by Gibson (1979), a popular design construct that describes the opportunity for action perceived in a situation. The theory of affordances argues that this construct is directly perceived rather than being a mental representation; affordances are not determining people's behavior but are increasing the possibilities of goal-specific actions to occur with the assistance of technical objects. The term effectivities is used to refer to the user's capabilities that enable physical actions resulting in some kind of interaction. Affordances and effectivities in unity determine the action and perception cycle (Gibson 1979; Greeno 1994). Two elaborations of the concept affordances were

developed: "perceived affordances" by Norman (2013) and "technology affordances" by Gaver (1991). The former relates the concept to everyday objects, indicating how it is possible to handle them. The later relates the concept to interactive systems, indicating functionality that may not be always visible.

The concept of affordances has led to the meditation of a SO, as having to somehow manifest its invisible characteristics, in such a way that users are aware of the possible uses of the SO in different combinations. This in turn has led this research to support the formation of the plug/synapse model, described in Section 3.3.2, as an appropriate one for assisting end-users to understand and handle PerComp applications as compositions of SOs abstracting what is fundamentally a publish/subscribe model in software engineering.

3.2.3. Constructionism

Constructionism theory underpins the "programming" part of EUD. The assumption is that people without programming knowledge can draw upon experiences from other types of construction to comprehend the way PerComp applications are synthesized out of SO components. In addition, sharing of constructs produced by community creativity allows their continuous use and adaptation without putting unnecessary effort. These notions inform also the design and development of the EUD supporting tools.

Constructionism is a learning theory developed by Papert (1986) promoting that it is through making things, such as physical and manipulative objects, and having hands-on experience that people learn more efficiently. Papert who was inspired from the experiential learning theory of Jean Piaget describes Constructionism as Sabelli (2008):From constructivist theories of psychology we take a view of learning as a reconstruction rather than as a transmission of knowledge. Then we extend the idea of manipulative materials to the idea that learning is most effective when part of an activity the learner experiences as constructing a meaningful product.

Constructionism has given inspiration to this framework, addressing end-users as active learners via the shaping of their PerComp environments which they construct by combining SOs (i.e. learning-by-doing in PerComp domain). Subsequently steps have been made to support people in order to actively learn about their environment, via mental constructs and tools for editing/creating PerComp applications on their own. Constructionism has also motivated the approach of treating tangible objects as application components that can act independently, but can also be associated so as to interact and influence each other's behavior. Methodologies and tools described further in this framework, such as composability and functional autonomy, stem from adopting the constructionism approach.

Social constructivism suggests that learning and consequently the knowledge is constructed by interactions and communications between members of a social group (Vygotsky 1980). Social constructivism levitated the need of community support tools, for sharing applications between communities, that can sustain and build on each-others knowledge. Such tools can promote and even boost a maker's culture for PerComp applications. Since the point is about sharing a common culture, and therefore making efforts for enabling common understanding of notions and relations, the developed framework suggests that this is facilitated by giving a unified vocabulary to end-users, but also to the developers of technology in respect to their mutual communication. In particular, by enabling a common glossary and shared metaphors used for application composition, comes as a response to the need of addressing an evolving community, and providing it with mental and construction tools so that they can share a common 'makers' culture.

3.3. Interaction design perspective

Principles, concepts, tools and user interface elements that can contribute to the sustainment of EUD in PerComp environments are described in this section. Concepts and tools introduced here are inspired from the theories mentioned in Section 3.2.

3.3.1. EUD features

EUD embraced in the proposed framework follows a few basic principles: involve users in the design process, provide EUD tools that support high-level tasks (e.g. composition of SOs/services, mapping conditions to actions), and build the supporting software mediator upon widely deployed technologies (e.g. Arduino, Raspberry Pi, etc.).

In particular, EUD indicates a process further to a simple customization, a process in which people become designers, by being able to envision, create, and adapt an application/environment. This process can be facilitated by open application repositories which represent a kind of common community knowledge. It also provides the possibility to configure a set of cooperating SOs by proper selection of options so that a given PerComp application can be customized. This involves mechanisms for collaborative sharing of applications which can be readapted to another environment.

EUD can use multimodal interfaces and be approached by multiple representations. Predefined stories and visualizations and other scenario based design techniques are used to support end-users to describe applications. A cascading degree of complexity, from the interfaces of the design functions (Figs. 2, 3, 9), to a more detailed programming (e.g. rules editing), is suggested (Fig. 11). EUD in PerComp environments can be further supported by visual representations, or designed models that correspond to the functional models of the system. These designed mental models (as in Fig. 2) help users grasp the configurable parts of the environment, and the means through which they can be linked.

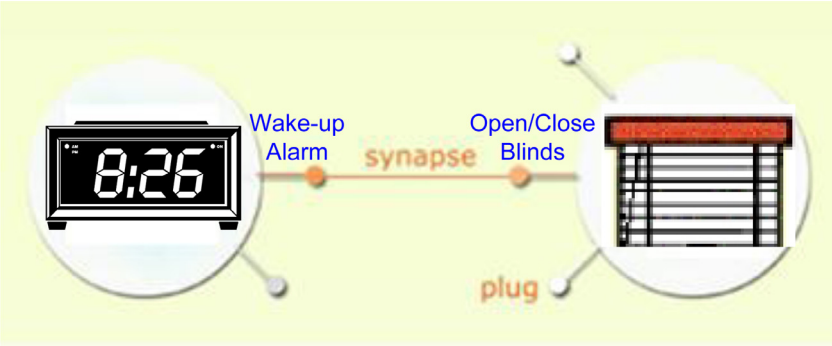

Fig. 2. Designing PerComp applications with high-level concepts via an EUD tool.

Fig. 3. Users involved in defining PerComp applications.

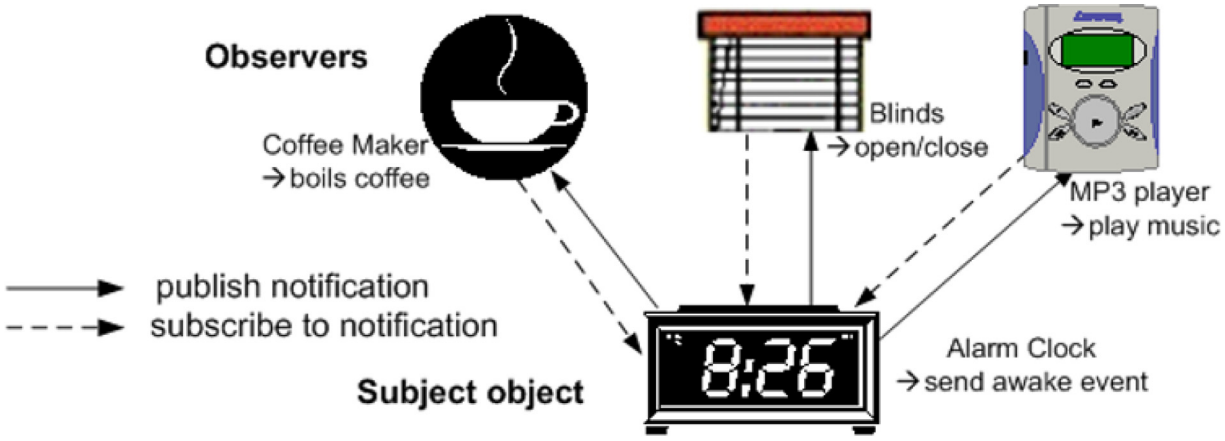

Fig. 4. Publish-subscribe design pattern to implement synapses.

EUD enables also end-users to coordinate the behavior of SOs and services by defining a sequence of conditional expressions for triggering actions that modify the state of the involved entities. Such sequences typically specify ECA rules which obey to the pattern *condition(s)* ⇒ *effect(s)*, where the *conditions* represent the events generated by SOs/services and the *effects* represent actions operated by the same or other entities. Regarding rule specification, the proposed conceptual model and the associated tool aims to strike a balance between ease of use and expressiveness, which can empower end-users to shape effectively their own smart environments.

For rule editing visual iconic editors (e.g. using maps or a node connection model) for the syntactic expression of the application logic in the form of rules are qualified. Syntax provided can be further supported by semi-automated methods for rule creation and use of natural language for rule composition (Coutaz and Crowley 2016) or explanation (Ghiani et al., 2017). Scaffolding mechanisms can provide an overview, starting from design concepts, and progress in particular association rules and programming syntax. An editing tool that enables typical users to understand and manage the rule conditions as logical expressions allows for the specification of more complex application behaviors. According to studies, theories on mental models dictate that end-users can conceptualize rule conditions more easily when using disjunction of conjunctions with respect to other analogous logical expressions (Metaxas and Markopoulos 2017). To decrease the cognitive effort related to the programming of multipart logical formulas a tool was designed that provides a visual interface for the expression of the application logic in terms of logical conditions and navigation support through hierarchical concept categories for appropriate feature selection (Fig. 11).

3.3.2. Plug/synapse model

The plug/synapse model provides conceptual abstractions which lead to a high-level composition approach that can be understood and adopted through supportive tools, even by end-users, for handling, in different detail, the features and functionality of PerComp applications. Fig. 2 shows an example of using the plug/synapse model to design a PerComp application. In this example each SO has certain plugs that offer specific services. Setting a synapse between the Wake-up Alarm plug and the Open/Close Blinds plug a simple application can be designed specifying that when the user sets an alarm time, the alarm will trigger the blinds opening.

The clock SO can be connected through synapses with other SOs too (Fig. 4). The end-user essentially is configuring her environment by creating synapses. At the implementation level synapses are manifestations of the observer design pattern, also known as publish/subscribe communication model (Eugster et al., 2003). The clock is responsible to issue the notification for the event of an alarm, and the other objects (the observers) will automatically receive the notification since they would have subscribed through the creation of the synapse. The clock does not need to know which are the observers and so it is feasible to make associations of objects at run time which offers a great flexibility in PerComp application configuration.

In particular, the key concepts of this model are outlined as follows:

Plugs are concepts representing SO properties and events in the digital space and are implemented as software objects. Plugs are characterized by their data and flow type. The flow of a plug can be a) output in case the SO property can be used by other entities (i.e. signifies a service provision), b) input in case the associated SO property is linked with data provided by other SOs (i.e. signi-

fies a service consumption), and c) input/output in case either of the previous flows can happen. Plugs, are therefore constructs that make their properties visible to the external world of the holding SO (e.g. end-users, agents and other SOs). These properties of SOs need to be somehow showed to people. One way to do this is via a special purpose software mechanism that can run in several interface modalities and devices. Such mechanism/tool can be generally referred to as the 'Editor' (Fig. 9).

Synapses are representing established functional associations between two plugs. The synapse creation process gives rise to the following capabilities: a) allows the formation of functional SO configurations that exhibit collective behavior, and b) provides designers and developers with the means to create, alter or otherwise handle the structure and behavior of PerComp applications (create, destroy, alter, "edit", copy/replicate, transfer, etc.).

3.3.3. Affordances as dynamic connectable capabilities

The design approach implied from the previous discussion is to model a PerComp application and its components as virtual artifacts, where parameters and trigger conditions (input events) are exposed as input plugs, and output state, together with outcome events (resulting functionality) are exposed as output plugs. While software engineers and programmers would be at ease with such a conceptualization of a PerComp application (potentially complex enough to be comprised of several, even nested, sub-applications), the average end-user would have difficulty to reason about and constructively manipulate a non-tangible entity. Moreover this type of representation would naturally cause an asymmetry between the number of SOs perceived by the user in their surrounding environment, and the representations displayed in the context of an editor's interface, since the editor would have to display the application and sub-applications virtual artifacts as well.

A higher level, but more user friendly approach towards PerComp application conceptualization by end-users, is suggested, where plugs of virtual artifacts corresponding to sub-systems are hosted as dynamic plugs of tangible augmented physical objects (i.e. SOs) of the application. Such plugs are dubbed as 'dynamic' since they stay attached to the respective tangible objects only in the context of their participation in the specific (sub-) application. This approach maintains symmetry between the tangible end-user environment and the virtual system context, enabling more fluent direct manipulation in editor interfaces and allowing for natural interaction paradigms such as programming by example. The proposed dynamic plugs correspond to affordances of the augmented SOs in the context of the specific PerComp application (sub-system / virtual artifact).

To explain the above concept, the following scenario is considered. In an augmented home, a sofa is alternatively used as a bed for guests. Aided by a camera-based recognition system that watches how the sofa is being used, a PerComp application is defined that exposes the affordance of a bed as virtual plugs on the sofa. If the sofa is moved to another location, not accessible by the vision system, the respective dynamic plugs are automatically removed from the sofa-SO. So, when the sofa is being used as a bed, other PerComp applications can be set up to trigger appropriate actions (like automatic dimming of the lights, or adjusting the room temperature).

3.4. System engineering perspective

To address the complexity and the dynamicity of PerComp environments we need to provide an extensible distributed system architecture that allows users to program their own smart environments and software mediators (middleware and tools) which provide PerComp applications with a component framework to support their runtime operation. Key elements that are adopted under the system engineering perspective of the EUD framework are outlined below.

3.4.1. SOs as components

At the heart of this component framework lies the concept of the SO. Subsequently we have everyday objects that are enhanced with information and communication technology to acquire new capabilities and services. Some of the questions posed in the context of the proposed framework are:

- How to model and manage context in terms of SOs and their state in the world?
- How to coordinate SOs so they can do something useful and meaningful together?
- How to "program" applications?
- How to interact with the user?

A SO-centric approach is followed for context awareness. This approach is based on the assumption that the context can be deconstructed into events that we can observe through sensors and then we can dynamically compose the context through the cooperation of many SOs.

Fig. 5 illustrates the SO architecture. This is a logical and to some extent an abstract architecture that will be reflected in specific modules of the software mediator which implements it. This architecture is comparable to a general agent architecture and is independent of any particular implementation platform (i.e. not

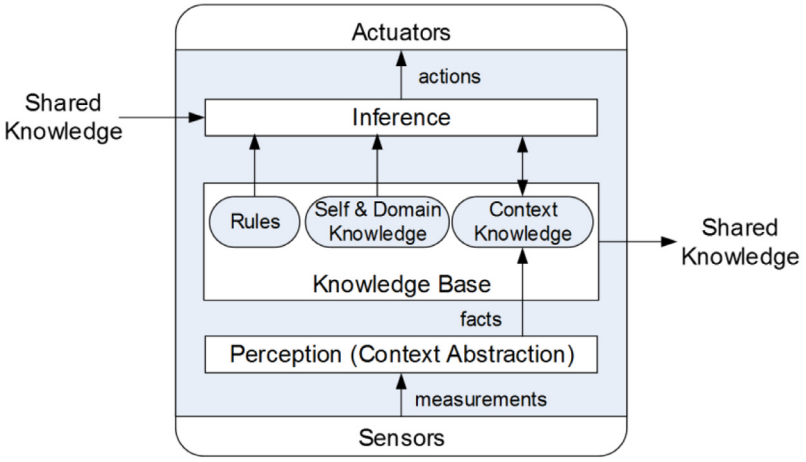

Fig. 5. Context-aware SO architecture.

limited by a specific micro-controller or a specific programming language used).

The key components of the architecture are:

Sensors. SOs use sensors to observe phenomena in the physical world. The sensors produce measurements that may be constant values (e.g. ambient temperature) or events (e.g. information about temperature changes over 1 °C).

Perception. Refers to a mechanism that performs context abstraction and is equivalent to the capacity of the system to understand events that are relevant to the application domain by using sensor measurements. For example, a fact may be that someone sits on a chair interpreting the measurements received from pressure sensors that are embedded into the chair. The conversion of measurements into events may involve the application of interpretation, transformation or aggregation mechanisms. For example, to calculate the photosynthetic activity of a plant we need to combine the measurements from different sensors by applying specific computational processes.

Knowledge base. A basic principle of the approach being considered is that the knowledge that is associated with a SO is stored and processed within the SO. This knowledge is built on facts and rules. The reasoning mechanism then in practice can be a simple mechanism of a Prolog-like interpreter with backward chaining. The knowledge base contains some SO self-description information such as its physical properties (identity, colour, weight, size, capacity, compute power, memory), its capabilities (services, access rights, certifying keys), interests (objectives, preferences, habits), and domain knowledge. It also contains dynamic knowledge (context knowledge) about its status in the world. Finally, we have the knowledge that arises from the reasoning mechanism that a SO may possess.

Inference. The reasoning mechanism is not necessary in every SO. When it is present it can process the knowledge of SO as well as the knowledge that comes from collaborating SOs in order to decide what actions can be performed in order to influence the physical world. For example, an action could be to turn on a light to illuminate a space, or open a faucet to water a garden. Another action could be calling a web service to update a remote system. The actions will occur when the conditions described in the form of rules permit it.

Actuators. The actuators are the means to bring changes in the real world.

In practice the proposed model is based on the cooperation of SOs through knowledge sharing. Knowledge (or a part of knowledge) that is stored in a SO becomes available to other SOs for use in their reasoning process for decision-making that individually the SOs would not be able to achieve on their own. So the end result is that we have a shared knowledge base which can be used by every SO for its reasoning.

The ways that a SO can be used is usually determined by its affordances that stem from certain aspects of its characteristics (physical or digital). Affordances describe the perceived properties of a SO, that determine how the users will handle it and the tasks they will perform with it, depending on their own effectivities like experience and skill, or any disabilities they may have. Application models and tools for configuring the cooperation of collections of SOs are required so that end-users can define PerComp applications.

3.4.2. Application model

As it was argued in the introduction of this paper on the one hand it is not feasible to predesign all the possible combinations of SOs in a smart environment and on the other hand the provision of a methodology that enables end-users to connect SOs and services together and define the rule-based application behavior can trigger their creative participation in the development process. In this section a generic model is presented that leverages on the plug/synapse model discussed previously in order to manifest the concepts governing the structure and operation of such applications. The end-users however are not exposed to the complexities and technical details of this model but instead would handle the high level metaphors visualized in the developed tools.

Instead of formalizing a specific context model that explicitly defines the components of smart environments (e.g. specific devices, services, places) the model aims to describe the key concepts involved in the realization of the targeted applications in a more generic way. In this view, the associated tools can support a wide range of application scenarios given the proper interfacing with the middleware which is responsible for managing the technical details of the smart environment. Furthermore, the binding process of this model with the middleware details can be based on an ontological approach as proposed by Seremeti et al. (2009) and by Corno et al. (2019), in contrast to ad hoc methods, so that ontology-based matching operations could facilitate full consistency of mappings between application general concepts and specific devices and services found in the smart space. Although there are existing models that semantically describe smart environments, especially in the context of the IoT domain, like the SSN ontology (Compton et al., 2012) and the IoT-A model (Bassi et al., 2013), these solutions are characterized by high complexity and large processing time as they include many concepts that are not essential for our purposes.

Fig. 6 illustrates in UML class diagram notation the generic application model. The model views a *PerCompApplication* as an aggregation of *Entity* components, each having an *EntityType* (e.g. chair, desk, lamp, refrigerator, plant, etc.). An existing application can be also a component of a more elaborated application. Each entity is characterized by its properties (*Property*) i.e. typically feature name and value pairs, actions (*Action*) i.e. typically operations that an entity can perform, and events (*Event*) that can generate. Data specification is given in JSON format, as it is simple, lightweight, and provides capabilities close to XML without demanding the processing overhead of XML due to its verbosity. Thus, a smart space is composed of entity instances that abstract the data (e.g. sensor values) and the behavior of items that populate the space. A *SO* or a *Service* are kinds of such entities. SOs represent everyday objects enhanced with ICT capabilities and electronic devices. Services are typically web services such as a weather service, a service for posting a message or a photo to a social network, an on-line shopping service, etc. Other entity types representing virtual objects (e.g. place, user) can also be defined in order to model particular features of the smart space with properties whose values are obtained by *PropertyDefinition* classifiers that provide the proper access, derivation, aggregation or interpretation mechanisms. Event types generated in the smart space are likewise cited in the model. A SO, for example, that monitors the temperature level of a plant could generate an event with the name "TemperatureRaised" which would provide the information that the temperature is higher than a specific threshold, together with the current temperature value. Similarly, an event called "AirPollutionAlert" could be raised this time by a more elaborated *EventDefinition* mechanism that aggregates and interprets values from different sensors attached to a SO. In the case of a *Service*, notification events can be generated such as a storm forecast from a weather service.

A *Synapse* is defined to connect two entities through their input/output *Plugs*. Synapses can wire events to effects and their operation is comparable to the operational semantics of event handlers used in graphical user interface applications. As soon as an entity generates an event, a synapse receives the output data, through the corresponding plug, and flows it through an *Invocation* classifier to an operation of the associated entity by passing the

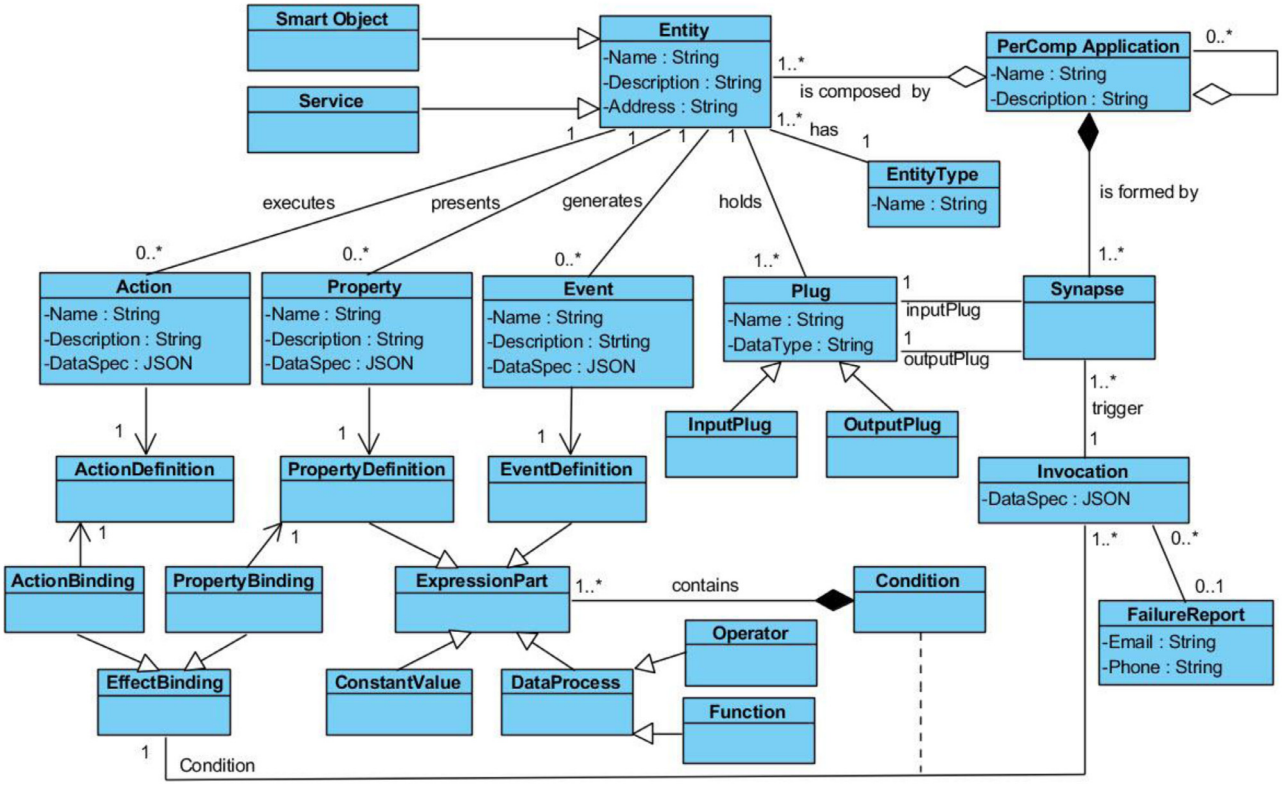

Fig. 6. Application model for connecting SOs and services together.

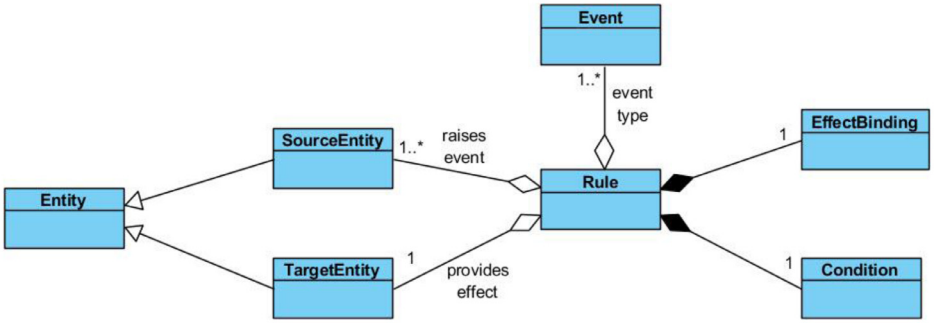

Fig. 7. Rule specification model.

data as parameters. The operation is decided by an *EffectBinding* mechanism which can determine either an action call (*ActionBinding*) or a property assignment (*PropertyBinding*). The binding determines also how the parameters of the operation should be handled, for example, properties or action attributes could be calculated according to assignment expressions (*PropertyDefinition* and *ActionDefinition* respectively). An invocation of an action operation may fail due to an entity's inaccessibility (e.g. any SO or Service unavailability). Such a failure may be of interest to the end-user and therefore a *FailureReport* can be defined. The report specifies the notification message to the user and the means of its communication (i.e. email and SMS).

The association between *Invocation* and *EffectBinding* may be controlled through the *Condition* concept which limits the invocation of an effect to specific circumstances. For example, a SO could provide notifications, through an SMS service, for air pollution in a room, only when a certain threshold is reached. A condition is formed by combining *ExpressionPart* elements which can be a data process (*DataProcess*), a constant value (*ConstantValue*) or a variable (*PropertyDefinition* or *EventDefinition*). The *DataProcess* class represents the presence of either *Operator* or *Function* items in expressions, illustrated as visual frames in the editing tool. These classes provide generic templates which are specialized with specific operator and function classes (not shown in the model) specifying the implementation of each supported logical operator (e.g. AND, OR, etc.) and function (e.g. MIN, MAX, NEARBY, etc.), the input parameters and the output outcome. This modeling allows easy extension of the visual language with additional and powerful data processors when required.

To provide adequate power to the expression of rules a distinct model is considered for rule specification using concepts defined in the application model. The model enables the specification of rule-based application behavior by providing the *Rule* concept (Fig. 7). The rule specification model aims to expand the *Synapse-Condition-EffectBinding* scheme which is linked to the specific entities participating in a specific synapse to a more general scheme involving multiple events and entities. A rule includes one or more events (*Event*) depending on the number of synapses that have been established from source entities (*SourceEntity*) to the target entity (*TargetEntity*). Moreover, a condition expression (*Condition*) is in-

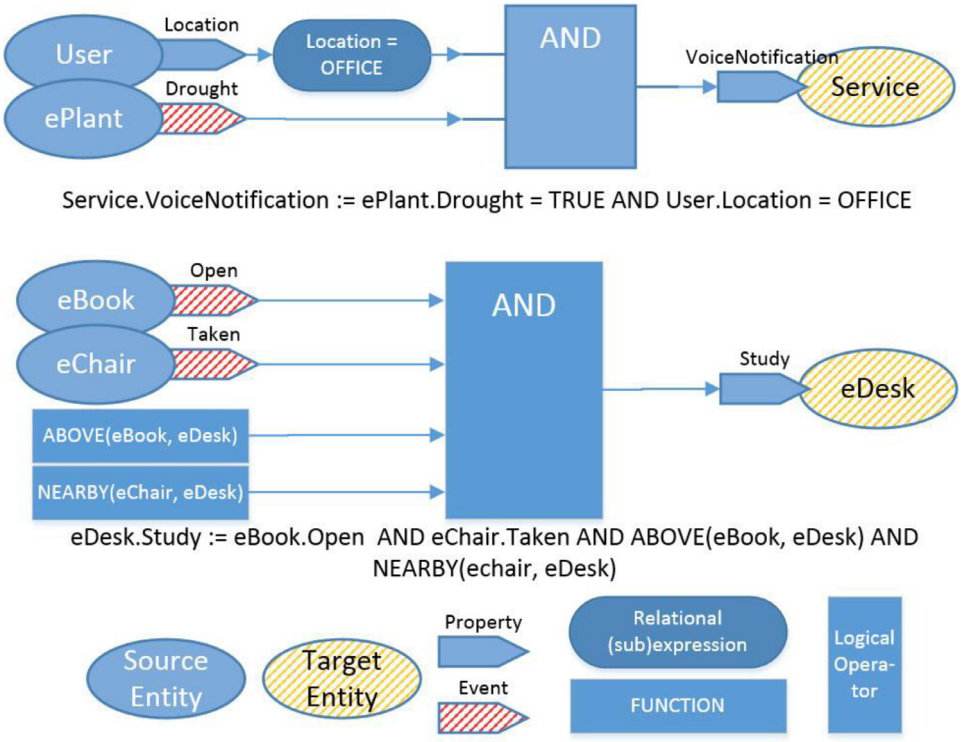

Fig. 8. Examples of rule oriented assignment expressions.

cluded to model a logical expression that governs the rule activation. Leveraging on the generic *Entity* concept the model enables the specification of rule conditions that could handle collectively the same kind of entities (e.g. activate all SOs in a smart space with a light type). The semantics of the *EffectBinding* is identical to the homonymous classifer of the application model.

The rule modeling described above enables the specification of assignment expressions in the form *targetProperty:= condition* to be part of the rule base. Such a rule modelling allows the development of a rule editor that manages assignment expressions in a visual way avoiding thus the need to write code in the typical text form. Fig. 8 illustrates examples of such rule oriented assignment expressions indicating the basic building blocks that will need to be visually represented in the rule editing tool. End-users thus can specify synapses and rules in the form of assignment expressions using the visual metaphors provided in the tools that will be described in the following section.

3.4.3. Tools

A toolset accompanies the framework to support efficiently the EUD for PerComp applications. Figs. 9–11 show snapshots of two tools that have been created and used in the context of different PerComp-focused domains. The first one is called PerComp Application Editor (PAE) (Fig. 9) and its purpose is to realize the design features of the EUD methodology. This is an end-user tool that adopts the plug/synapse model to manage the configuration of PerComp applications graphically through the connection of properties and services from various individual SOs, aiming to produce a specific behavior. The PAE tool offers functionality and services to end-users in order to accomplish the design of PerComp applications. The key services offered include:

- Discovering SOs and registered services and their associated plugs (Fig. 10a).
- Providing information about the explored SOs and their properties.
- Supporting end-users to set, manage and remove synapses between plugs.
- Designing PerComp applications as collections of synapses as inspired by the object-orientedness principle of Activity Theory; an application is represented as a graph where links between the nodes correspond to synapses between the plugs of the SOs.
- Supporting application operation (enable, disable, delete) and management (testing and configuration) – as inspired by Constructionism.

As shown in Fig. 9, the user interface of the PAE tool consists of two primary areas. On the left side the list of SOs discovered in the smart environment is visualized using a suitable icon. The list may also contain registered web services (e.g. Weather, Facebook), considered as virtual SOs, and provided that specific adapters have been defined to accommodate the plug/synapse model. In case the number of displayed SOs is large a scrollbar appears on the left sidebar. Other facilitating mechanisms for the efficient management of long SOs lists may be included such as logical groupings of SOs providing similar services, filtering options based on selection criteria and using an intelligent agent that proposes SOs depending on the application needs. On the right side lies the workspace area of the editor where the user composes the application. The user first must select the SOs that will participate in the application which are then placed into the workspace with a visual icon that is enhanced with one or more small pipes representing plugs, which signify the connections points of the synapses; the later essentially wire events to effects. Consider a SO called eCarpet holding a plug named ‘Position’ and a SO called eLamp holding a plug named ‘LampColour’. The two SOs are connected by creating a synapse from the ‘Position’ plug which publishes position events, to the ‘LampColour’ plug that will provide an operation. The type of operation is determined by the *EffectBinding* mechanism of the application model (Fig. 6) and in this case it is bound to an action (i.e. change the color of the lamp depending on the position data). The exact mappings between the different position events

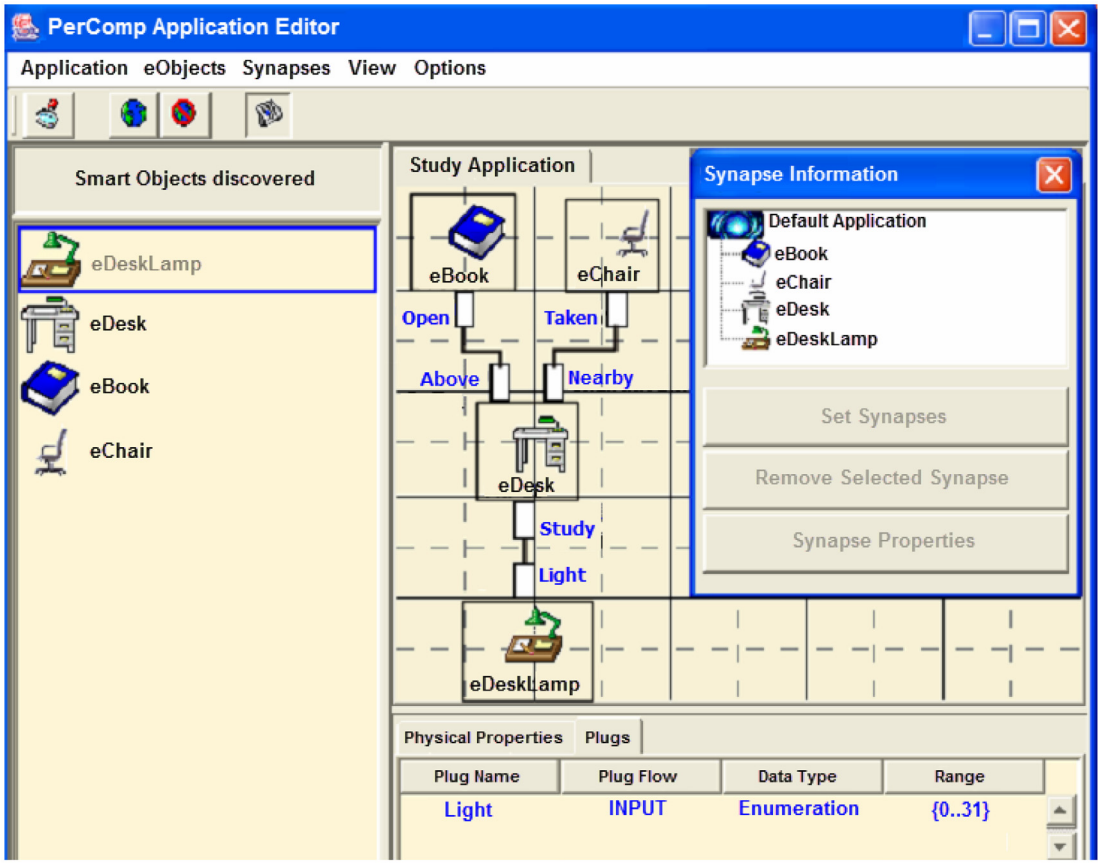

Fig. 9. PerComp Application Editor (PAE) a tool to design graphically an application by combining SOs discovered in a PerComp environment using the plug/synapse model.

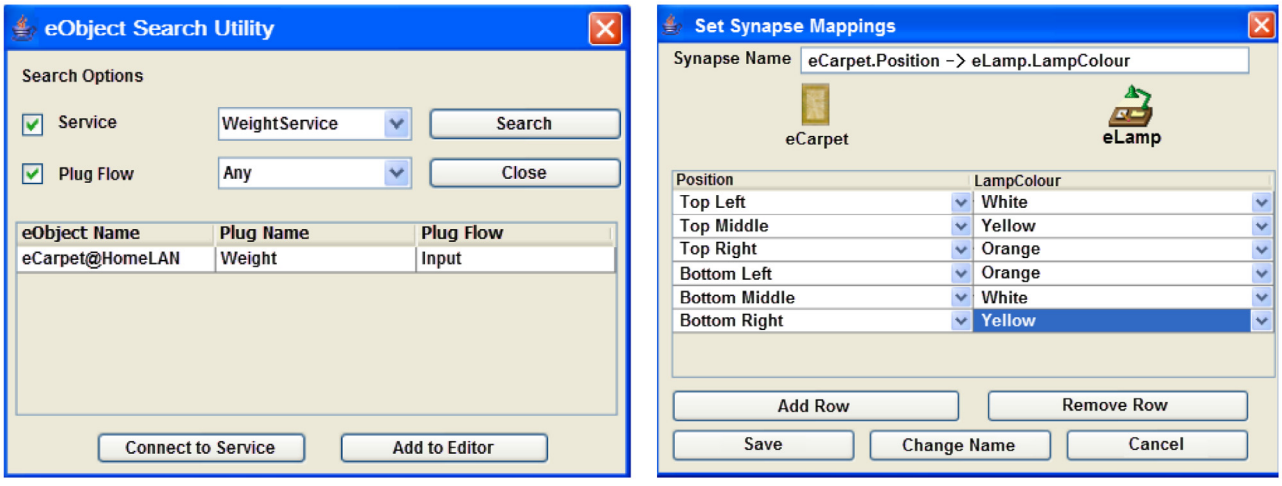

Fig. 10. (a) SO discovery (b) Setting synapse mappings.

and the color of the lamp are defined as rules which here take the form of synapse mappings (Fig 10b). The example shown in Fig. 9 demonstrates a more complex use case with four SOs that have been placed into the workspace. Two of them (i.e. eBook and eChair) are connected to the same SO (i.e. eDesk) through a pair of synapses. The 'Open' plug of the eBook and the 'Taken' plug of the eChair provide the corresponding events. The receiving plugs of the eDesk SO called 'Above' and 'Nearby' represent special functions and the operation triggered at eDesk is bound to a property calculation (eDesk.Study). The latter, when activated, is provided as an event from the eDesk SO to the eDeskLamp SO which upon receiving the study event is expected to provide the appropriate action.

In particular, to create a synapse the user has to select two SOs from the workspace area and press the button "Set Synapses" in the Synapse Information dialog. A two-dimension grid defined by the plugs of the selected SOs is displayed. In each cell of the grid link buttons appear for all possible combinations of synapses. The user by clicking on these buttons can activate or deactivate a synapse. Using PAE the user can also modify the mappings of a synapse by selecting "Synapse Properties". In this case the Set Synapse Mappings window appears and displays the existing map-

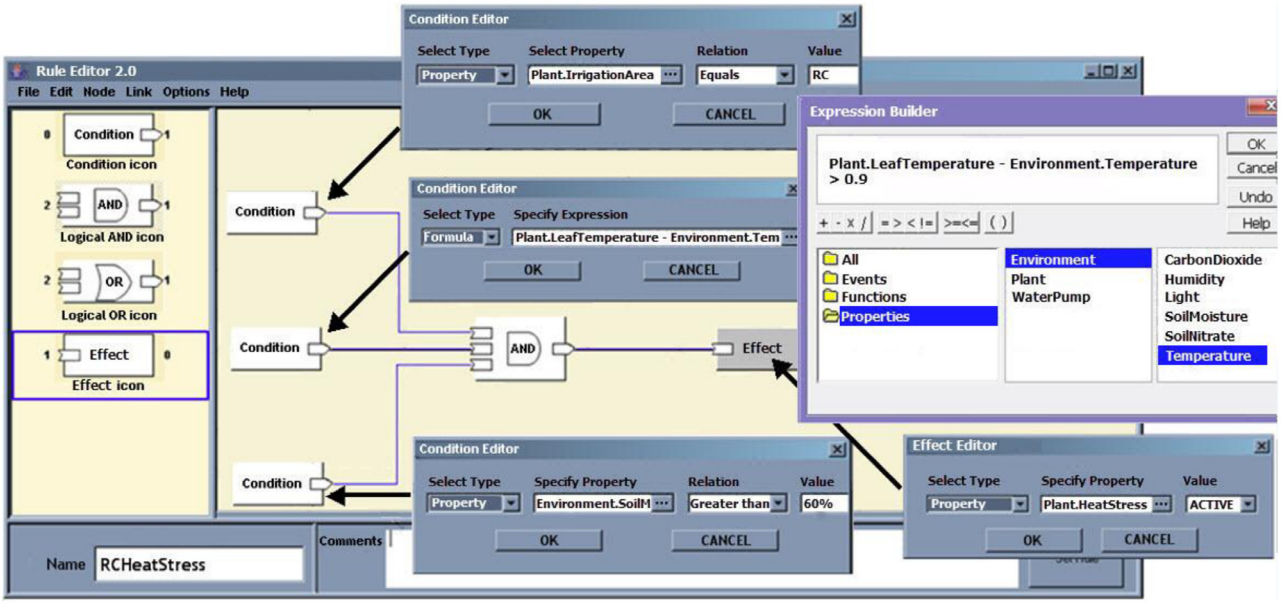

Fig. 11. A rule editor employed to let the user set up an irrigation application in a PerComp agricultural environment (adopted and revised from Goumopoulos et al. (2014)).

pings (Fig. 10b). The user then can modify or remove existing mappings, and add new ones.

The second tool is called Application Logic Editor (ALE) and its purpose is to realize the programming features of the EUD methodology, as inspired by Constructionism but also principles from Ecological Theory, such as visibility. The main functionalities of ALE are:

- Monitoring selected SO parameters.
- Managing the rule base of SOs and PerComp applications.

Although the wiring of SOs via the PAE tool enables the specification of event-driven applications, to enrich the expressiveness of the application behavior description, a distinct rule model was defined. This model expands the scope of rule expressions in order to address any entity in the smart environment instead of being confined to the specific SOs of a synapse. In this way, the rules could be defined in a more general way so that diverse entities (SOs and services) could interpose. The rule editor implements in practice the rule model described in the previous section (Fig. 7) and essentially supports the specification of assignment expressions as the examples given in Fig. 8 by utilizing a visual node-connection model.

A user interacting with the rule editor (Fig. 11) initially aims to design the rule structure in a visual way by selecting the proper building blocks located in the upper left part of the user interface and overlaying them to the design area in the upper right part using drag and drop movements. A rule design is comprised by a condition part and an effect part. The former includes one or more *Condition* blocks which can be combined with the logical *AND* and *OR* blocks in a composition that can be as elaborate as required by the application needs and as can be supported by end-user skills. The latter includes an *Effect* block representing a property or an action definition. Different blocks are wired by connecting output pins with input pins. Point and click movements in the design area provide further block customization and rule definition capabilities. Every block placed in the design area upon a right mouse click activates a drop down menu with available commands such as editing, renaming, deleting and copying. For example, a *Condition* block can be renamed to have a more representative name reflecting the defined expression or the number of input pins of a logical block can be modified from the default value. Editing a *Condition* block activates the Condition Editor which provides facilities, like the expression builder. For the condition editing the user selects the properties (representing property definitions and features of SOs), events (identified by the plugs of SOs) and special functions to be used as elements in forming the logical expressions. The user navigates through the concept categories associated to the SOs and services found in the smart environment for the appropriate variable selection. Editing an *Effect* block is supported similarly by the Effect Editor. Rule definition includes the specification of a name and optional comments and is finalized by pressing the Save Rule button.

Fig. 11 shows, as an example, the programming of the “Heat Stress” rule for a PerComp application in the precision agriculture domain using the Rule Editor component of ALE. SOs in this case include water pumps and plant/environmental parameters monitoring devices modeled as WaterPump, Plant and Environment respectively. The rule includes three conditions which if satisfied the stress status property of the specific plant zone becomes active. Such rules for the assessment of plant stated in different cultivation zones can be configured by experts in plants biology without having to know details of the operation of the system. Consequently, end-users in this application domain can alter the rules without any programming background knowledge.

The tools presented foster the supporting of a range of abstraction levels and application specification methods to address the variety of end-users requirements and skills. One viewpoint represents the implementation of the high level plug/synapse model, inspired by the concept of affordances, according to which a SO manifests its characteristics in such a way that users can be aware of the possible uses of the SO in different combinations. This model aims to assist end-users to comprehend and handle PerComp applications as compositions of SOs abstracting in effect the trigger/action programming paradigm. An elaboration of this level is to specify synapse mappings which provide simple to specify but limited one-to-one mappings between the involved pair of plugs. To enhance expressiveness, the option to describe in detail logical conditions using operators and connecting subexpressions, under a more general scheme involving multiple events and entities, is also accommodated in the toolset.

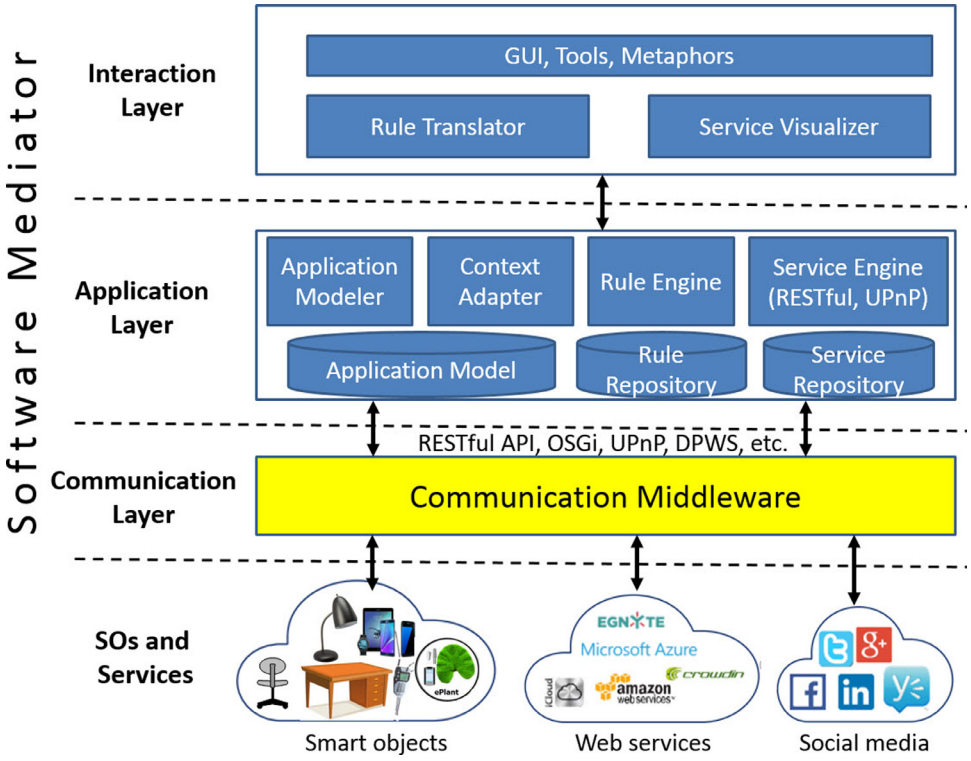

Fig. 12. Multi-layered software mediator.

The tools have been developed using the Java programming language and the NetBeans integrated development environment. The Swing GUI library was used for building the user interface along with the JGraphX library which provided the functionality for visualization and interaction with node-edge graphs.

3.4.4. Software mediator

The distributed system architecture for PerComp environments outlined in this section is supported by a software mediator that realizes the abstractions of the framework and provides the interfaces to abstract the details of the underlying data communication and accessing of sensors and actuators so that a PerComp application is easily defined and appeared as an integrated computational service. The software mediator architecture is organized in layers, each one handling a distinct aspect (Fig. 12): the communication layer, the application layer and the interaction layer. SOs and services that can be combined in order to realize component-based and cooperative applications in PerComp environments are located below the communication middleware.

The communication middleware aims at simplifying the communication between the application layer and the heterogeneous SO ecologies, which provide context information and execute the application rules. To communicate with real SOs and services, any communication middleware system that addresses the heterogeneity challenges of different devices, communication networks, and access technologies can be exploited, ranging from smart space gateways to vendor middleware systems such as Microsoft Azure IoT suite.[1] A number of technologies have been developed in order to address this kind of challenges such as RESTful web services, UPnP (Universal Plug and Play), DPWS (Devices Profile for Web Services) and OSGi (Open Services Gateway initiative). For the evaluation study performed in this work the ATRACO communication middleware for accessing UPnP devices and web services in smart spaces was adopted (Togias et al., 2010; Hagras et al., 2012). This middleware, in particular, provides an abstraction of devices and services by means of an ontology and also provides support for service registration and discovery. The ATRACO middleware is based on the REST (Representational State Transfer) API (Application Programming Interface) creating an end-point (i.e. a URL), through which the application components can have access to the services and context of a device (a SO in this case). Each URL consists of the address and the name of the SO in order to create a main access point to it. URL's general syntax is:

"*protocol://IP_address:port_no/SO_name/resources*"

For example, a valid address to access the resources of a SO called ePlant would be "http://localhost:8080/ePlant". Data exchange is realized by sending JSON messages via HTTP requests (e.g. GET, POST, DELETE). For example, the commands "*GET /events*" and "*GET /actions*" return the list of supported events and actions by the SO.

A key role of the ATRACO middleware is to maintain the status and address of all available SOs and services, via service discovery and service registration mechanisms acting as a smart space gateway. The gateway is assigned a static IP address in order to be visible by the SOs and the tools. When a SO is activated, it is registered to the gateway. Thus when the PAE tool needs to find the available SOs it sends a look up request to the gateway. Similarly, if a SO needs to communicate with another SO through a synapse, it finds its address through the gateway before establishing a direct communication with it. Also, the gateway periodically communicates with all the SOs registered to guarantee their availability. Thus the communication layer provides the ability to manage the discovery of SOs and services so as to support the creation of appropriate associations between them. Furthermore, different technologies (e.g. OSGi) can be smoothly integrated as the structuring of the ATRACO middleware allows the addition of different classes of adapters to handle the access of other API technologies.

[1] Microsoft Azure. Azure IoT suite. [Online] 2019 [cited 2019 August]. Available from: https://azure.microsoft.com/en-us/suites/iot-suite/.

```
(defrule NotifyUserViaVoice
(declare (salience ?PRIORITY_VALUE))
?param<-(ePlant(name ?name) User(name ?name) Service(name ?name) (ePlantDrought ?ePlantDrought)
(UserLocation ?UserLocation) (ServiceVoiceNotification FALSE) )
( test (and (eq ?ePlantDrought TRUE) (eq ?UserLocation "OFFICE")))
=>
(retract ?param)
 (assert (ePlant(name ?name) User(name ?name) Service(name ?name) (ePlantDrought ?ePlantDrought)
(UserLocation ?UserLocation) (ServiceVoiceNotification TRUE) ))
(send-to-java "ServiceVoiceNotification =TRUE")
)

(defrule IdentifyStudyActivity
(declare (salience ?PRIORITY_VALUE))
?param<-(eDesk(name ?name) eChair(name ?name) eBook(name ?name) (eBookOpen ?eBookOpen) (eChairTaken
?eChairTaken) (eDeskStudy FALSE) )
( test (and (eq ?eBookOpen TRUE) (eq ?eChairTaken TRUE) (ABOVE ?eBook ?eDesk) (NEARBY ?eChair ?eDesk)))
=>
(retract ? param)
 (assert (eDesk(name ?name) eChair(name ?name) eBook(name ?name) (eBookOpen ?eBookOpen) (eChairTaken
?eChairTaken) (eDeskStudy TRUE) ))
(send-to-java "eDeskStudy =TRUE")
)
```

Fig. 13. JESS/CLIPS code excerpt generated for example rules.

The application layer manages the operational aspects of the PerComp applications designed by means of connecting SOs and services together and programmed by defining rules. It contains four main modules: the *Application Modeler*, the *Context Adapter*, the *Rule Engine* and the *Service Engine*. The Application Modeler implements the application model (Fig. 6) and supports the operation of PerComp applications through the cooperation of autonomous SOs by managing the interaction with the rest of the modules in the layer. In this context it undertakes the responsibility of creating the required logical communication channels between the SOs and the management of their interaction based on a publish/subscribe mechanism where event objects are associated with publishing services and action objects are associated with subscription services. The Context Adapter is responsible for the interfacing with the actual context of the smart space delivered by the communication layer. In particular, the Context Adapter initially gathers real time context from actual devices and services, for example, sensor values, SOs status and service information. Afterwards, it transforms the collected information into instances of the application model key concepts such as events, properties and actions. In this respect, the consolidation between the application model and the context data coming from the communication middleware is materialized through a specific Java component that utilizes the APIs exported by the ATRACO middleware.

The Rule Engine, on the other hand, is responsible for managing the association of the defined rules onto actual devices and services and is actively supported by the Application Modeler since the concepts that emerge in the rules are supplied by the instantiated application model. The module is based on the JESS (Java Expert System Shell) rule engine (Friedman-Hill 2003). The choice of the rule engine was informed by its characteristics (powerful, fast and lightweight) and the prospect to execute the system in both resourceful and embedded devices (Mukherjee, 2018). The main components of the JESS architecture are: a) the working memory storing the facts (i.e. the data operated by the inference engine) which are associated with indexes to speedup searching; b) the rule base containing all the rules; c) the inference engine deciding which rules to fire based on the Rete pattern matching algorithm (Forgy 1982); and d) the execution engine executing the rules fired by the inference engine. The execution of this module starts based on the initial facts provided by the Context Adapter and the rules stored in the Rule Repository using CLIPS (C Language Integrated Production System) format (Riley 1991). The JESS rule engine supports forward chaining inference that activates a rule when all conditions based on the available facts are hold. The activation of a rule causes the execution of actions which can result in the modification of a fact or calling a function to produce an effect. JESS offers a rule conflict resolution mechanism using rule priorities through the salience JESS property. Salience is an integer value (default is 0) and higher salience implies higher priority. Active rules with the same priority are fired based on their activation order, which is identified by their specification order. This mechanism allows end-users to associate a specific priority value to each rule while JESS enforces the proper rule triggering order. JESS represents key components of the rule engine such as facts, rules, results and actions in standard Java classes and methods facilitating the integration with the rest of the modules. For example, each rule object cooperates with the Application Modeler module for materializing an event-action model based on the publish/subscribe pattern. Fig. 13 gives part of the JESS/CLIPS code that implements the rule examples presented in Fig. 8. Although the syntax of the rules is Lisp-like, still, the end-user is protected by such complexities and can use the rule editor to specify rules via simple assignment expressions as mentioned previously.

The Service Engine provides the operational environment to invoke a web service or a UPnP device/service as an effect of an activated rule. The specification of the service invocation is provided in a JSON service description file stored in the Service Repository. In the case of a RESTful web service, for example, the attributes that are specified include: *service name, service type* (i.e. action, event), *service url, service icon, application ID, application key, authentica-*

tion method (e.g. OAuth2), *invocation method* (e.g. POST, GET) and *response type* (e.g. XML, JSON).

The interaction layer provides the Graphical User Interface (GUI) and manages the visual composition of applications by end-users. Also, end-users interact with the GUI to compose rules using concepts defined in the application model and high level metaphors visualized in the developed tools. The GUI is supplied with events and actions defined in the application layer, therefore enabling the usage of high-level abstractions. Moreover, the usage of generic concepts and in particular those of events and actions without a specific technology dependency allows the interaction layer to decrease the volume of exposed information. Separation of concerns through layering provides the possibility of implementing diverse user interfaces to render the application model representation, ranging from typical window-based applications to mobile and web-based interfaces. Furthermore, the interaction layer includes two modules, the *Rule Translator* and the *Service Visualizer*. The Rule Translator takes as input the visual form of the rules created by the end-users and generates the corresponding JESS/CLIPS specification. As part of the translation process, for each rule defined by the user, a complementary rule is also generated to revert the effects of the original rule once the involved condition part fails. In this way, the interaction layer exports low-level code that can be read by the JESS rule engine kernel. The Service Visualizer is responsible for visualizing in the user interface the parameters of the services as defined in the Service Repository. This module is activated when users wish to define a rule by adding events or actions. To create the user interface of the registered services, the Service Visualizer interacts with the Service Engine to acquire a JSON file with the description of accessible services in terms of parameters such as service name, URL address, properties, events and actions. This approach makes the interaction layer independent of the technical details of the available services.

Finally, a plug-in mechanism may be employed to extend the core functionality of the application layer. We can use, for example, ontologies via an Ontology Handler plug-in. One of the objectives of such ontology is the semantic description of SOs' capabilities and services to support the SO discovery mechanism in a way that helps to manage the heterogeneity of the environmental context and models of the real world (Seremeti et al., 2009). In this respect the handler provides methods to query the ontology for services provided by a SO as well as for SOs providing specific services. By using the discovery mechanism and the ontology categorization SOs that offer semantically similar services can be identified and recommended for replacing non-available services.

4. Discussion

4.1. Framework insights

The approach of building PerComp applications using SOs as components has much in common with well-established software engineering approaches like object-oriented and component-based system development. Embracing these principles in the proposed framework and with the mediation of EUD tools end-users are enabled to handle SO collections and configure smart applications in creative and unforeseen ways. Under a system engineering perspective, the advantage of this composition process is that the general SO architecture encapsulates already part of the system design. In addition, by abstracting the development complexity via the implemented conceptual model allows end-users to instantiate, even after they have been left alone with an existing PerComp system, other use cases of that system. Consequently adaptation and evolution is possible as a system could be arranged in different ways to satisfy new user needs. The reuse of SOs for different goals, possibly not foreseen at design time, creates opportunities for evolving exploitation of existing devices. This idea is in harmony with the Activity Theory view on PerComp technology systems treated as evolving ecologies.

Table 1
Overview of the underling theories impact on EUD tools design and implementation.

Theory	Influence
Activity Theory	*Tool mediation* informed the provision of EUD tools to support shaping of PerComp environments; *mental tools* informed the adoption of metaphors in the interaction models implemented by the tools for simplifying application composition and rule editing tasks. The *object-orientedness* principle informed the designing of PerComp applications as collections of synapses. The *hierarchical structure of activities* informed the adoption of an hierarchical concept categories representation and different levels of abstraction in EUD tools design in order to decrease the cognitive effort related to the programming of application logic. The *socio-cultural object development* principle informed tool features (structure, form, etc.) supporting the reuse of SOs for different goals/applications, thus creating opportunities for evolving exploitation of existing devices.
Ecological Theory	The concept of *affordances* inspired the formation of the high level plug/synapse model. Likewise, it has led to the meditation of a SO in such a way that users are aware of the possible uses of the SO in different combinations; Tools expose visually such affordances on reusable SOs through their corresponding plugs.
Constructionism	SOs as *components* of PerComp applications; Tools visualize SOs as basic entities that can be discovered and combined together to form an application. *Properties* characterize SOs; Tools support the definition of emergent properties with a rule-based approach and provide information about the explored SOs and their properties. *Sharing gallery* as open application repositories representing common community knowledge; Tools support application creation, management and sharing.

The use of affordances as dynamic connectable capabilities, that can be considered as 'hidden' affordances (Gaver, 1991), signify perceived potential usage of the SO in a given situation or environment (context of use). Drawn up on the basis of Constructionism model and the Ecological theory, by exposing such affordances on reusable SOs, the end-user applies end-user design in practice, allowing for more comprehensive repurposing of the given SOs by people interacting in this environment. According to Activity Theory this allows the system to evolve in human and societal development space. Table 1 summarizes the way the underlying theories, discussed in Section 3.2, have informed the design and implementation of the EUD approach.

When such SOs are detached from the context of the PerComp application they stop hosting these dynamic plugs and thus they no longer present their respective affordances, since they can only be perceived in the specific PerComp environment. However, this approach also enables the case where a SO itself carries with it one or more persisted applications that are portable. Such applications only have this SO as a member. These applications can extend permanently the respective SO by exposing more affordances on it, irrespective of the environment of use. Persistence of the respective logic can be done on the SO itself, provided enough internal data storage, or be hosted at a cloud based repository with the SO holding a unique reference ID to those applications in the cloud.

Grounding rule handling on top of JESS provides a powerful way to formulate rules in a machine-readable form and an efficient and scalable manner to evaluate the sequence of triggering events for actuating an appropriate action or altering a property state. The adoption of a powerful event processing engine is justified by the complexity of PerComp and IoT environments which are characterized by an ever increasing number of devices and generated events and by application requirements to support data analysis on the fly for purposes such as detecting critical situations and identifying user intentions. In such device and data rich settings it is becoming essential to formulate advanced application logic and to support decision management, in terms of facts and rules, for improving automation, efficiency and safety in a personalized way. In terms of performance, the JESS engine used in this work implements an improved version of the Rete pattern matching algorithm, in terms of applying adaptive indexing techniques, which achieves a linear computational complexity in the size of the working memory (Friedman-Hill 2008). That is a complexity in the order of *O(RFP)*, where *R* is the number of rules, *P* is the average number of patterns (i.e. conditions) per rule, and *F* is the number of facts on the working memory. Tests reported in the literature testify the high execution performance of JESS and its capability of firing up to several thousands of rules and performing hundreds of thousands of pattern matching operations in the range of milliseconds in typical personal computing platforms provided that the rule file has been loaded and the Rete network has been formed (Friedman-Hill 2008). The time required to perform the latter operations is higher but is met only at the initial setup of the rule system. The scalability of the JESS reasoning process in the presence of several hundred of devices in smart space applications within the same time latency has been also reported (Gilman et al., 2010). In our case, the average elapsed time for rule evaluation is only 5 milliseconds for a configuration of 10 rules with about two patterns per rule. Every new SO is modeled with about 5 facts, effecting a low overhead (in the range of nanoseconds) to integrate the associated state in the rule system and requiring approximately 20 bytes of additional memory. The scalability of the rule system is also sustained by implementing a publish/subscribe mechanism for the context of interest associated to the facts in the working memory. JESS, finally, supports integration with Java based applications in a stable manner by adhering to the Java Specification Request (JSR) 94, which provides a detailed description of a standard way to access the rule engine through a Java Rule Engine API (Toussaint 2003).

4.2. User evaluation

4.2.1. Study design

A usability evaluation study focusing on the EUD dimension of the proposed framework took place in a lab environment established for testing small scale PerComp systems. The testbed is a single room office equipped with network infrastructure, a room control point and server, a multitude of sensors and an ecology of SOs. The evaluation targeted the perceived usability of the high-level conceptual model and the tools supporting the EUD dimensions of the framework. Twenty volunteers (n = =20, 62% female and 38% male, average age 37.4, std. dev. 9.6) took part in the evaluation which lasted 4 days with five individuals distinctively attended per day. The recruitment of participants was done using electronic-mail lists available to the authors' associated research institute and they were administrative and technical staff as well as acquaintances of the employees. None of them had any relation to our research and had no familiarity with EUD in smart environments. None of the participants had any knowledge of a general purpose programming language, though ten (10) have used spreadsheet formulas and one (1) had experience on UNIX scripts for administration purposes. All the participants signed a typical consent form for attending the evaluation study.

Table 2
Scenarios provided to the users in the evaluation study.

ID	Description
S1	Your environment is equipped with a number of smart objects (eObjecs) that can be combined with the support of the PerComp Application Editor to define suitable services using the plug/synapse model. You are requested to define **a smart light application** that is composed of four smart objects: a desk, a lamp, a book and a chair. The application logic is that when the chair is taken and is nearby the desk and the book is open above the desk the application infers that a study activity takes place and as service the application regulates the light depending on the brightness sensed on the book. According to the above description your goal is to define the proper synapses between smart objects using the editing tool. You should also provide a suitable name for the application.
S2	As an extension of the application described in S1 you are requested to define with the support of the Rule Editor an **awareness application** with an aim to implicitly notify your friends on social media that you are studying. In particular, your goal is to define a rule that examines whether the user study property is active and define as an effect the activation of the Facebook (FB) service notification. You should navigate through the tool facilities for the appropriate property/action selection.
S3	Your environment is further equipped with sensors to detect context information such as parameters of the environment, the presence and location of various entities, etc. Your goal in this case is to define a **"talking" plant application** which monitors the temperature and soil moisture of a home smart plant and notifies you about the state of the plant in a form that depends on your location. In particular, the plant state should be defined as drought (i.e. the drought property is active) if the difference between the plant temperature and the room temperature is greater than 1 °C or the soil moisture property of the plant is less than 50%. If the presence of your smart keys is detected (i.e. the presence property is active) the effect is to set the user location to office. If the plant state is drought and the user location is in the office, the effect is the activation of the voice notification service. If the plant state is drought but the user location is not in the office, the effect is the activation of the sms notification service. The behavior of this application requires the definition of more than one rules.

The interaction of the participants with the PerComp environment and the EUD tools was performed under the observation of a member of the research team with the responsibility to record and respond to any matter that would arise during the evaluation and to log the time required by each participant to complete each task. In the beginning, key concepts of the framework were presented and examples of using the EUD tools were demonstrated in a 30 min preparation session. The participants were given two training scenarios for familiarizing them with the tool facilities, one involving the use of the plug/synapse model and the PAE tool, and one that required the editing of rules with the use of the Rule Editor. During the training session the participants were able to request freely the help of the mediator researcher while the training material had different SOs and services involved with respect to the experimental material. Upon completion of the training session, the participants were invited to design and program various PerComp applications based on three scenarios (experimental material) which are shown in Table 2.

In the case of S1, participants were asked to create only the synapses between the plugs of the SOs using the PAE tool to design

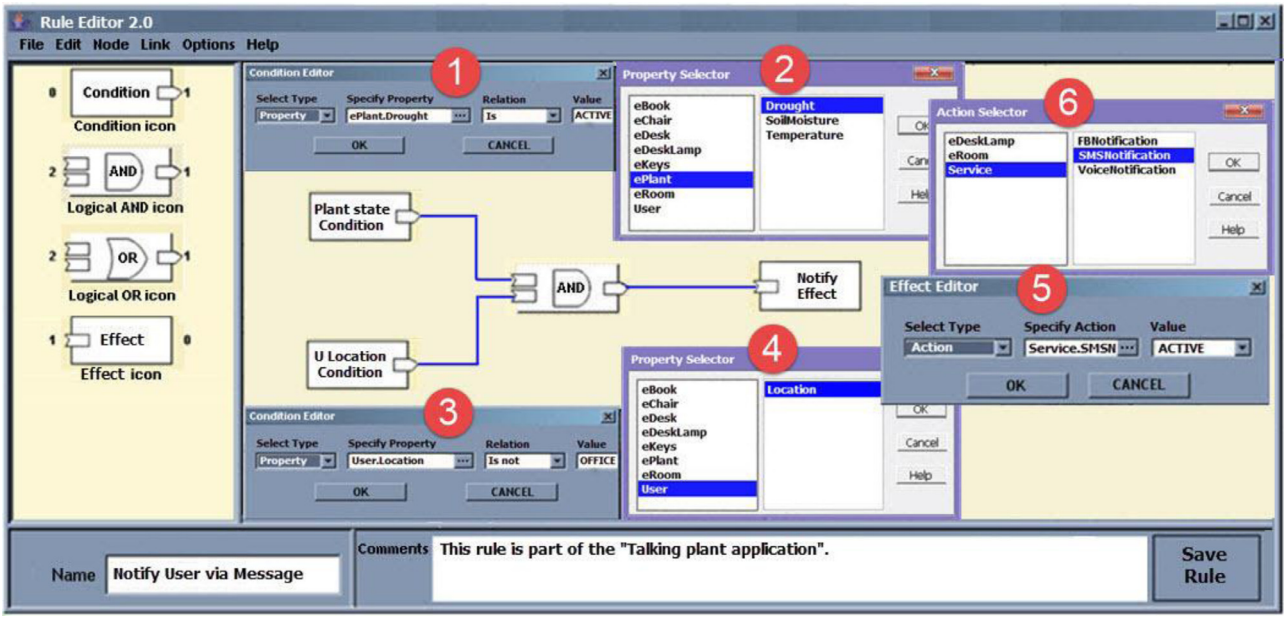

Fig. 14. User interface for editing rule 5.

Table 3
Rules defining application logic in S2 and S3.

No	Symbolic Name	Definition
1	Study Awareness	**Condition:** `User.Study is ACTIVE` **Effect:** `Activate Service.FBNotification`
2	Plant Drought	**Condition:** `ePlant.Temperature - eRoom.Temperature > 1 OR ePlant.SoilMoisture < 50%` **Effect:** `Set ePlant.Drought to ACTIVE`
3	Location Office	**Condition:** `eKeys.Presence is ACTIVE` **Effect:** `Set User.Location to OFFICE`
4	Notify User via Voice	**Condition:** `ePlant.Drought is ACTIVE AND User.Location is OFFICE` **Effect:** `Activate Service.VoiceNotification`
5	Notify User via Message	**Condition:** `ePlant.Drought is ACTIVE AND User.Location is not OFFICE` **Effect:** `Activate Service.SMSNotification`

the application. The user interface employed is the one depicted in Fig. 9. The use of the high level plug/synapse abstraction, implies that the application logic is partly encapsulated in the design of the SOs with respect to the rationale explained in Section 4.2.3. In the case of S2, the application logic is defined by rule 1 in Table 3. In this case the inferred study activity triggers a mobile application that interfaces with the Facebook platform. Facebook provides a REST API which makes possible the posting of specific messages to a defined group. In the case of S3, the application logic is defined by rules 2–5 in Table 3. The programming of all the rules was performed by using the rule editing tool presented in Section 3.4.3.

Fig. 14 depicts as an example the user interface with which participants had to interact to specify rule 5. The rule design includes two condition blocks combined with the logical AND function and one effect block. Blocks have been renamed to better represent their content. For each block the corresponding editor is shown which in turn is assisted by context dependent selectors. The property selectors classify properties with respect to the available entity instances (SOs, services, user etc.) found in the smart environment. Such a representation adheres to the generic application model specification discussed in Section 3.4.2. Accordingly, an action selector classifies operations that an entity can perform for an easy access. The numeric annotation in the figure indicates the step-wise rule definition. In such a way, rule building is facilitated by providing a clear and visual separation between conditions and action blocks and by providing a structured representation of related notions (e.g. properties, events, actions) in terms of associated entities. The rest of the rules are defined in a similar way, whereas rule 2 requires additionally the use of the expression builder as shown in Fig. 11, in an analogous manner.

For the smart light application sensors from Phidgets (www.phidgets.com) were used interfaced through the Phidgets 8/8/8 interface kit and a USB connection to a Windows-based laptop running the software mediator modules. The devices employed were resistive force sensors for detecting the occupancy of the chair, a pair of ultrasonic sensors for proximity detection between the chair and the desk, a flex sensor placed into the spine of the book to detect open/close status, a light sensor and a RFID sensor to detect the book on the desk. A temperature sensor measuring ambient temperature was also provided for other applications.

For the talking plant application a soil moisture probe and a thermistor attached to the leaf of the plant were connected to a Raspberry Pi 3 Model B device (https://www.raspberrypi.org/) with Raspbian OS and JVM installed. This configuration permits an autonomous SO operation that exports context and plant status to other SOs, when required, through wireless communication, while invoking the services required for this application scenario: a web SMS service provided by a telecommunication provider and accessed through a REST API, and a voice notification service implemented using the HTML5 Web Speech API.

Finally, a SO called eKeys was defined which is 'aware' whether it is in the smart space or not. The assumption is that the user always has the keys in place, thus the presence of the keys implies user presence too. In this case, the user presence at office is detected by a proximity beacon placed in the user's keychain. When the user is at office, any signal from the beacon can be detected by the base transceiver device and identified as presence.

Considering that the aim of the evaluation study was primarily the perceived usability assessment of the developed conceptual

Fig. 15. SOs prototypes utilized during the user evaluation study.

Table 4
Rule editing success/fail proportions.

Editing	Rule No				
	1	**2**	**3**	**4**	**5**
Success	20	18	20	19	18
Fail	0	2	0	1	2

model and its manifestation in the developed EUD tools, the use of low fidelity prototyping techniques like mockups, Wizard of Oz or simulations of the SOs functionality would have been an option. However, additional value was seen on employing real software and hardware prototypes in order to provide a richer experience to the participants and to monitor, under realistic conditions, how potential users grasp the concepts and whether they can create and test example applications and then conceive their own applications. Fig. 15 illustrates part of the SOs used during the user evaluation study.

Upon completion of the application design and programming tasks the participants provided their perceived usability of the proposed EUD approach by means of a validated technology acceptance rating scale. Furthermore, they were also requested to suggest use cases from their daily environment that in their opinion could be realized with the support of the evaluated EUD approach. Such use cases indicate the potential for the creative participation of end-users to configure their own smart environments by wiring SOs together given their empowerment with proper EUD tools. Other qualitative data were collected in terms of the observations logged during the development tasks referring to participants' expressed comments or noteworthy behavior.

4.2.2. Results

Regarding the application design dimension all the participants were able to successfully use the PAE tool to wire the required sensing and actuating components and test the application behavior. This outcome validated the view that an architectural approach where users act as composers of predefined components is beneficial.

The easy perception of the application logic and the navigation support of the rule editing tool through hierarchical concept categories for appropriate feature selection enabled users to complete 95% of the total rules they requested to define (Table 4). In a few cases some users have difficulty to successfully form composite conditions and expressions or select the relation that corresponds to inequality requiring the effective intervention of the observer. In particular, a few errors were committed on editing the condition part concerning the use of the correct logical operator (i.e. using *AND* instead of *OR*) and specifying the wrong relational operator (i.e. using *Equal*/*Is* instead of *Not equal*/*Is not* and using *Equal*/*Is* instead of *Greater than*). On the effect part an error identified was the selection of the Property concept category instead of the Action concept category to activate a service.

The editing times for each one of the rules defined in Table 3 are summarized in Fig. 16.

Rules 1, 3 and rules 4, 5 have similar structure and therefore on average these rule pairs required comparable times. Rule 2 was the most demanding in time as the programming of its logic required using several sub-editors (condition and effect editor, expression builder). Clearly the rule structure complexity is the crucial factor of the editing performance. From the data analysis and our own observations there is also an indication that the editing performance is improved as users practice more with the tools. In particular, by calculating the average time required to complete editing of the first three rules and comparing to the average time required to complete editing of the last two rules we found that twelve users (60%) have improved their performance.

The Technology Acceptance Model 3 (TAM3) (Venkatesh and Bala 2008) was used as the conceptual basis for the usability evaluation. Participants scored a total of twenty two items with a rating range 1–7 (1 = strongly disagree - 7 = strongly agree), covering six determinants of the TAM3 model (Table 5). TAM3 statements were adjusted to the context of the proposed EUD approach to make them relevant (see the Appendix).

The TAM3 evaluation results are summarized in Table 6. All six categories of statements were checked for statistical consistency, as indicated by the Cronbach's alpha reliability coefficient, and the metric found above the lower threshold value (0.70). Given the PU mean value, the participants moderately agree that using the framework's tools provides them an advantage regarding the definition of PerComp applications. The PEOU dimension on the other hand scores even higher showing that the concepts of the model are well perceived by the participants and the tools are easy to use. The high mean values of CSE and PEC categories reveal that the participants are satisfied with the resources provided to support the tasks assigned to them on the evaluation session. The ENJ has a positive score reflecting the degree of feeling pleasance and fun when using the EUD tools to build such applications. As a determinant of PU the OUT category scores similarly.

Complementary to the above quantitative data the study provided also useful qualitative data. The participants were asked to suggest application scenarios that could be realized with the evaluated EUD approach. Some of the responses indicated general daily life application categories such as home automation and security, smart gardening, elderly and child monitoring. There were also responses that provided explicit scenario descriptions in more detail demonstrating that end-users are able to conceive suitable applications as connections of SOs and services found in smart environments given the support of conceptual metaphors and EUD tools.

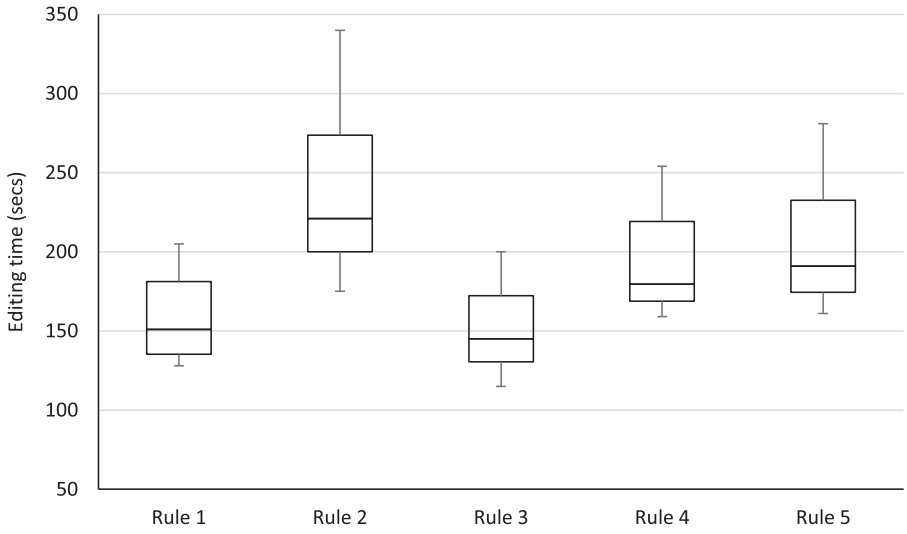

Fig. 16. Rule editing performance as a boxplot depicting min, max and percentile values.

Table 5
Determinants of the TAM3 model used for the evaluation of the EUD approach and tools.

Determinant	Abbr.	Items	Refers to
Perceived Usefulness	PU	4	the degree to which the framework enhances the development of PerComp applications
Perceived Ease of Use	PEOU	4	the degree of comfort related to the use of the key components of the framework i.e. the conceptual model and the associated tools
Computer Self-Efficacy	CSE	4	the degree to which an individual believes that has the ability to perform a specific task (e.g. programming a rule) using the tools of the framework
Perceptions of External Control	PEC	4	the degree to which an individual believes that organizational and technical resources (knowledge, consultant and technical support) exist to support the use of the framework
Perceived Enjoyment	ENJ	3	the degree to which the activity of using the framework is perceived to be enjoyable in its own right
Output Quality	OUT	3	the degree to which the framework tools operate in a helpful way assisting through the provided information to the addressing of the required tasks

Table 6
TAM3 evaluation results.

Category	Mean	Std dev	Cronbach's α	Category	Mean	Std dev	Cronbach's α
PU	5.88	1.15	0.92	PEC	5.96	1.27	0.80
PEOU	5.97	1.05	0.89	ENJ	5.30	1.10	0.85
CSE	6.01	1.35	0.83	OUT	5.91	1.18	0.81

A representative sample of the answers collected are provided below:

- *When I'm lying down in bed and have set a time to the sleep time indicator of the clock, then if the time is equal to that time the house blinds are automatically closed. Respectively, when I'm lying down in bed and have set a time to the alarm time indicator of the clock, then if the time is equal to that time the blinds are opened and fresh coffee is prepared.*
- *When the medication reminder is set to notify my parents to take their pills in collaboration with a smart cabinet which can detect if they have taken the medication or not then an indication is given in what position they can find the proper pill.*
- *When the alarm rings and I have gotten up from the bed the boiler switches on automatically to heat up the water.*
- *When I'm at work (9.00am to 5.00pm) and lights or appliances are on at home they automatically switch off.*
- *When my smart exercise floor detects the calories burnt my daily/weekly diet schedule is adapted.*
- *When my elderly parents, wearing their smart bracelet, fall I get a notice in my smartphone.*
- *When my child is far from the house I get an alarm notification.*
- *When my child watches the TV in the bedroom and it is after 09.00 pm then parental control is applied.*

Useful feedback was provided by the participants in the form of comments on favorable elements or on difficulties encountered when using the tools and as suggestions for improving the applicability of the EUD approach. Positive comments were made about the simplicity of the join the dots metaphor which combined with the handy user interface of the PAE tool (Fig. 9) triggered approving characterizations on the usefulness of the application designing process. Additionally, the clear editing separation between the rule condition and action parts and the structuring of events in relevant categories were highlighted as key enablers for successfully defining the requested application logic. As an exception, the expression of composite logical expressions as in rules 2 and 5 for which one had to combine multiple condition nodes with a Boolean logic was demanding for a couple of users even though a performance improvement was observed as they were getting more familiar with the system. Participants appreciated the on line help provided by the tools but also emphasized that additional resources, like training videos, would be useful especially for managing more difficult tasks like setting synapse mappings. A few users suggested that

instead of explicitly writing a description of a rule an automatic documentation of the rule logic in natural language should be generated by the system. Another suggestion was to include a wizard mode to support the editing of composite rules.

4.2.3. Reflection

In line with the TAM3 determinants (Table 5), PU and PEOU are the major factors determining the intention to use a specific type of technology. The PEOU as well as the PU, since PEOU has a causal effect on PU, are influenced by system characteristics, therefore affecting the system actual usage. In this context, the evaluation study results suggest a high degree of PU, PEOU as well as positive attitudes towards using the proposed EUD approach. In accordance with the high TAM3 scores especially in the categories CSE and PEC, the participants in their majority commented that the tools were intuitive and the displayed information was sensible and comprehensible.

Successful rule-based tools deployed in smart environments often advocate simplicity over expressiveness in their design. IFTTT, for example, which is used for mashing up smart things and web services does not support the use of logical operators in the definition of rule conditions and allows only the specification of a single triggering event and a single action. Similarly, AppsGate an EUD prototype deployed in smart home environments (Coutaz and Crowley 2016) does not support the use of event chains in logical expressions. In this context, multiple rules must be defined to handle composite cases with a negative impact on editing performance. Our user evaluation study shows that end-users without programming background can cope, in their majority, with more composite rules provided that proper interaction models are available. This observation is in line with other studies on rule-based tools arguing that non-technical users can master quickly the creation of rules with multiple events and actions if provided with adequate interaction mechanisms (Ur et al., 2014; Ghiani et al., 2017).

In EUD platforms, application definition can be assisted by system mechanisms that navigate users to explore components of the environment and simplify rendering of robust and correct application logic (Cappiello et al., 2015). The proposed framework provides such mechanisms both at operational and conceptual level. At the operational level, for example, the discovering of SOs and their associated services (Fig. 10a) facilitate the design of applications. At the conceptual level end-users can take advantage of available existing application configurations which can subsequently be tailored based on user preferences and actual needs. An example of this pattern can be seen in the configuration of the first and second application of the user evaluation study. The actual rules of the first application were predefined in the design of the eDesk component and the application behavior was emerging when wiring the four SOs composing the application. Namely, the user has not to explicitly define a rule in this case, but instead relies on the design of SOs which encapsulate already part of the logic and manifest it through their provided affordances i.e. plugs. Applying the plug/synapse model the user composes the application in a high level manner that abstracts the difficulty of a composite logic specification. Nevertheless, in the absence of component synapsing, it would be still possible to define the application logic directly as a composite rule based on the specification outlined in Fig. 8. The second application represented an extension of the first application by leveraging on a service (i.e. study event detection) that had been already configured and then was used as a trigger in a new user-defined rule to activate some additional functionality (i.e. colleague awareness). The use of predefined rules as well as rule templates matching the needs of common use cases and their approval by end-users has been validated by others researchers (Desolda et al., 2017). The extensive and successful use of predefined recipes in the IFTTT platform argues in the same direction.

The variety of end-users regarding their skills and capabilities in using EUD tools as well as the variety of application composition requirements entails designing tools that can accommodate a range of abstraction levels to compensate for this context variance (Paternò and Wulf 2017). Following the discussion above, users could be offered with the option of a fully preconfigured system with predefined rules, or at the opposite extreme, with the option to describe in detail logical conditions using operators and connecting subexpressions by their own. Both approaches have been explored in the evaluation study and were positively assessed by the participants.

In conclusion, the user evaluation showed that the understanding of the high-level conceptual model was satisfactory as the vast majority of the users were able to design and program simple applications using the models and tools of the proposed framework. Finally, it should be noted that the framework concepts and tools are currently used by postgraduate students of a PerComp curriculum (Engineering of Pervasive Computing Systems) to complete part of their assignments (Goumopoulos et al., 2017). Laboratory exercises are mandatory and hands-on experience is provided by using the development environment in order to design and implement PerComp applications. This usage provides an opportunity for continuous feedback and identification of possible future improvements.

4.3. Design guidelines

Based on the developed conceptual models and tools as well as the analysis of their design implications, a set of design guidelines was formed that can be used as a baseline for designers and developers of EUD tools and applications in PerComp domain.

EUD tools in PerComp domain should be designed for emerging behavior. Since it is not feasible to predesign all the possible combinations of SOs in a smart environment, PerComp environments should define design spaces that enable end-users to compose by their own applications at usage time and even explore situations that were not envisaged at the time of design. Therefore, EUD tools in PerComp domain should provide EUD spaces that define and create the socio-technical infrastructure which can interpret situations emerging at the time of use.

EUD tools for PerComp applications should be developed under a multidisciplinary effort and by adopting a broader systemic view. Designing EUD tools for PerComp applications is a complex problem which requires knowledge that is typically dispersed between multiple domain experts. In principle, different disciplines are collaboratively involved to create PerComp systems and tools such as System Developers, Engineers, User Experience Designers, User Interface and Interaction Designers and Experts from Human Sciences. A framework that bridges the gaps in the communication between these disciplines and provides a unified overview, rather than disconnected views and frameworks separate for each discipline is motivated. A unified agile process for system design and development might be followed, characterized by multidisciplinary face to face cooperation, and incremental, iterative development, with progress achieved through working system versions. Getting insights from the proposed framework, in the first phases of conceptualization the process takes input from the theoretical foundations (e.g. Activity Theory, Ecological Theory, Constructionism, Situated Cognition), that abridge understanding of the long term implications of the system, the end-user interaction design aspects and the system design and development aspects of it. As the phases get into more detail, elements of the relevant methodologies (e.g. EUD for SO Ecologies) are used and worked into further detail into system deployment as a functional prototype.

Employ proper metaphors in graphical user interfaces and visual methods to hide programming complexities. The component-based metaphor, adopted for example in the proposed framework, advocates SOs as basic entities for composing PerComp applications and their definition is not confined to associated events and actions, that mostly are modeled in IoT platforms, but includes also properties that can be defined by experts in a field for applying semantics to SOs. A software mediator then should be enacted to operate as a platform that defines the interfaces and the protocols that govern the cooperation between the SOs. This platform hides the internal implementation details of the SO architecture and provides the interfaces for composing an application in a high-level manner. Moreover, by abstracting the publish/subscribe model with the join the dots metaphor allows for simplicity regarding the visual representation of devices' capabilities and link possibilities in the context of an editor tool. Similarly, the visual editing of the application rules provides the advantage to the end-users of employing a reasonable mechanism to handle rule elements (i.e. operators, variables, expressions and actions) as single components that can be just wired instead of using an equivalent but hard to use text-based form.

Rule-based EUD tools should be designed aiming for conceivable expressive power. Single trigger and single action rules may be adequate for simple scenarios but specifying more advanced application behavior could benefit from more expressive rule models. The evaluation study showed that an editing tool that enables typical users to understand and manage the rule conditions as logical expressions allows for the specification of meaningful applications. Furthermore, a tool providing a visual interface for the expression of the application logic in terms of logical conditions and navigation support through hierarchical concept categories for appropriate feature selection can decrease the cognitive effort related to the programming of multipart logical formulas. The issue of natural event composition in logical expressions has been studied and theories on mental models dictate that end-users can conceptualize rule conditions more easily when using an OR of AND's with respect to other equivalent logical schemes (Metaxas and Markopoulos 2017).

Provide supportive mechanisms to explore and use resources and pre-configured elements in the smart environment. Such supportive mechanisms simplify the definition of rules by guiding people to discover event and actions that are provided by the entities (i.e. SOs and services) in the smart environment in order to define robust and correct application logic. Furthermore, as mentioned in the reflection of the user evaluation study, users can exploit a class of pre-configured application logic rules reflecting standard use cases. This is in particular, useful, when different, applications, share some common rules. An extension of the pre-configuration notion can be seen on a design level. A pre-configuration of the SOs to be connected at the use time may assist in avoiding excessive difficulty. Experienced designers and engineers may provide an initial system set-up by configuring the SO properties at design time that are actually helpful at the use time. Experience designers can subsequently specify methods or abstractions that intermediate between the native functionality of the SO and the configuration features that are opened to the end-users. In this way, the structure of an application can be configured by end-users, while the internal design of SOs and their properties are defined by experienced designers and engineers. This principle, has been also advocated by the meta-design framework for general-purpose EUD systems (Fischer et al. 2012).

EUD tools should be designed providing different levels of programming capabilities supporting the different levels of the technical competency of end-users. The analysis of the collected material from the user evaluation and other informal validation assessments in the context of project assignments in a PerComp curriculum (Goumopoulos et al., 2017) revealed that users with a technical background were able to use additional features of the editing tools and express more composite configurations of the applications. Therefore, the design of PerComp application building tools should provide diverse levels of programming capabilities to end-users, to support their varied level of technical competence and their willingness to add more complex logic to an application. For the inexperienced end-users the most relevant target is the formation and reformation of SO collections for quick application design, instead of programming the application logic. In this direction the user attempts an approach of trying various possibilities and learning by the errors made rather than a typical programming process.

EUD should embrace conceptual models that describe PerComp applications in a generic way and delegate associations to real devices and services by interfacing to the proper middleware layer. For example, the existence and operation of PerComp applications as SO ecologies entails the generation of events by the constituent entities causing the state of SO ecologies to evolve through operations triggered by the instantiation of the specified rules. Since the exact devices required for sustaining the SO ecologies operation may vary between different environments or deployments having a generic application model satisfies such required flexibility. Formalizing these notions in a generic way offers further benefits. For example, the introduction of new data-gathering mechanisms or new sensors is facilitated allowing the flexible long-term maintenance and evolution of the system.

Support EUD via a layered software mediator to allow for diverse user interaction schemes. Applying separation of concerns through software mediator layering and adopting software design patterns such as Model-View-Controller (MVC) provides the possibility of implementing diverse user interfaces and interaction modalities to render the application model representations. This is essential since the right fit between the users' semantic and programming view can accommodate many approaches of representation and configuration, to be effective. Multimodal interaction, including spoken language, alternative visualizations of programming structures, or programming by example techniques can be valid, non-exclusive approaches for end-users to design or program their environment. Platforms that seek to support EUD for PerComp applications need to be modular enough to allow this flexibility in user interaction. In addition, by separating the user interaction layer from the other layers of the software mediator (Fig. 12) facilitates the development of different user interfaces for a variety of devices (e.g. traditional PCs, mobile devices, ultramobiles).

4.4. Open issues

In articulating an application, a person will express his or her own goals and intentions, rather than how the information is organized or presented. For example: '*I would like to have music in the house when I come in*'. In contrast, the plug/synapse model requires the human goal to be decomposed into available SOs and their capabilities, and then synthesized to form a collective SO behavior: i.e. 'Player>start playing music>when>ID Anna> steps on doormat'.

However adept human beings are at using systems and notations, the above line does not come instinctively as human thought. Employing multiple representations for editing, including natural language for expressing intentions, and then defining the specifics by help of an expert system, could ease this gap between natural expression of intentions and the application configuration. The model still remains valid as a way to explain current configurations of the system or express evolving ones, and being able to reason between multiple representations.

An important issue that needs to be addressed in evolvable component-based PerComp applications is that of uploading new functionality to SOs. In low complexity SOs, such as sensors, the 'linkable' possibilities they provide can be their mere output readings. Adding higher level 'connectivities' that relate to the SOs affordances or observed usage can be assisted by intelligent systems and ontologies. Software developers should also be able to port new connectivities in SOs, automatically (in the form of software upgrades), or created intentionally and injected into SOs.

Appropriation of PerComp applications into different environments and debugging of applications are two very important issues for EUD in the PerComp domain. Appropriation may be assisted by ontologies, and tweaked by end-user decisions and debugging mechanisms. How to know of errors and avoid them, identify possible rule conflicts, recovery and debugging are crucial issues for EUD that are not easily addressed in PerComp applications (Coutaz and Crowley 2016). Multiple representations in a step by step design-creation-explanation approach can help with reasoning about possible errors. Yet, it is not easy to test the applications created, given that the conditions that apply for their triggering are not easy to reproduce. Many actions in the real world cannot be undone or presimulated. Simulations and debugging tools are an open area that can be investigated in detail, addressed to the specifics of this domain. Aspects of error tolerance and handling errors in PerComp application creation is an open interaction design issue that has to be investigated in time, in its own merit.

5. Conclusion

In this work we have elaborated on the rationale of a framework that addresses the design and use of PerComp applications. The proposed framework considers concepts, methodologies and tools in two broad viewpoints: interaction design and system engineering. The perspective of iterative evolution and development through time, for systems, people, and EUD tools, can draw knowledge from Activity Theory and Ecological Theory. The framework adheres to the component-based system engineering paradigm adapted to the use of SOs and the tool mediation principle of AT. EUD is considered as an integral part of PerComp applications enabling their co-evolution with end-users so that the latter can be empowered to shape effectively their own smart environments. To capture these principles a SO architecture has been conceived and a distributed system platform has been developed to support various application examples.

The specific contribution of this work in the EUD research in PerComp domain can be summarized as follows: (a) The proposed EUD approach is grounded on a broader conceptual framework, that addresses the mutual co-evolution between people, EUD tools (mental or physical), and PerComp technology, which is not the norm in literature; (b) The conceptual models and EUD tools developed can be used by typical end-users who may not have programming experience; (c) The multi-layered software mediator defined provides a platform for the runtime support of PerComp applications composed under the principles of the conceptual framework in realistic settings; (d) With respect to other rule-based systems and composition models the proposed EUD approach demonstrates more expressive power in terms of handling multiple events in a structured manner and allowing the definition of properties with a rule-based approach grounded on the application and rule models developed within the framework; and (e) The design guidelines formulated can serve as a baseline for designers and developers of EUD tools and applications in PerComp domain.

Declaration of Competing Interests

None.

CRediT authorship contribution statement

Christos Goumopoulos: Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing, Investigation, Formal analysis, Visualization, Supervision. **Irene Mavrommati:** Conceptualization, Methodology, Data curation, Validation, Writing - review & editing.

Acknowledgement

The authors would like to thank all the fellow researchers in the DAISy Research Unit for their valuable support in the performed research and the anonymous reviewers for their insightful and constructive comments.

Appendix

Measurement items used in the evaluation study.

Category	Item
PU1	Using the EUD approach and tools improves my performance as creator of PerComp applications.
PU2	Using the EUD approach and tools for my assigned tasks increases my productivity.
PU3	Using the EUD approach and tools enhances my effectiveness as creator of PerComp applications.
PU4	I find the EUD approach and tools to be useful in my role as creator of PerComp applications.
PEOU1	My interaction with the EUD tools is clear and understandable.
PEOU2	Interacting with the EUD tools does not require a lot of my mental effort.
PEOU3	I find the EUD tools to be easy to use.
PEOU4	I find it easy to get the EUD tools to do what I want it to do.
	I could complete the tasks using the EUD tools ...
CSE1	... if there was no one around to tell me what to do as I go.
CSE2	... if I had just the built-in help facility for assistance.
CSE3	... if someone showed me how to do it first.
CSE4	... if I had used similar packages before this one to do the same task.
PEC1	I have control over using the EUD tools.
PEC2	I have the resources necessary to use the EUD tools.
PEC3	Given the resources, opportunities and knowledge it takes to use the EUD tools, it would be easy for me to use the EUD tools.
PEC4	The EUD tools are compatible with other systems I use.
ENJ1	I find using the EUD approach and tools to be enjoyable.
ENJ2	The actual process of using the EUD tools is pleasant.
ENJ3	I have fun using the EUD tools.
OUT1	The quality of the output I get from the EUD tools is high.
OUT2	I have no problem with the quality of the EUD tools' output.
OUT3	I rate the results from the EUD tools to be excellent.

References

Bannon, L.J., Bødker, S., 1989. Beyond the interface: encountering artifacts in use. DAIMI Rep. Ser. 18 (288).

Barricelli, B.R., Cassano, F., Fogli, D., Piccinno, A., 2019. End-user development, end-user programming and end-user software engineering: a systematic mapping study. J. Syst. Softw. 149, 101–137.

Barricelli, B.R., Valtolina, S., 2017. A visual language and interactive system for end-user development of internet of things ecosystems. J. Visual Lang. Comput. 40, 1–19.

Becker, C., Handte, M., Schiele, G., Rothermel, K., 2004. Pcom-a component system for pervasive computing. In: In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications*, pp. 67–76.

Bergesio, L., Bernardos, A.M., Casar, J.R., 2017. An object-oriented model for object orchestration in smart environments. Procedia Comput Sci 109, 440–447.

Bassi, A., Bauer, M., Fiedler, M., Kranenburg, R.V., 2013. Enabling Things to Talk. Springer-Verlag GmbH.

Brown, J.S., Collins, A., Duguid, P., 1989. Situated cognition and the culture of learning. Educ. Res. 18 (1), 32–42.

Cappiello, C., Matera, M., Picozzi, M., 2015. A UI-centric approach for the end-user development of multidevice mashups. ACM Trans. Web 9 (3), 11.

Chen, X.A., Li, Y., 2017. Improv: an input framework for improvising cross-device interaction by demonstration. ACM Trans. Comput.-Hum. Interaction (TOCHI) 24 (2), 15.

Chin, J.S.Y., Callaghan, V., Clarke, G., 2006. An end-user programming paradigm for pervasive computing applications. In: ICPS, 6, pp. 325–328.

Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Huang, V., 2012. The SSN ontology of the W3C semantic sensor network incubator group, 17, pp. 25–32.

Cook, D.J., Das, S.K., 2007. How smart are our environments? An updated look at the state of the art. Pervasive Mob. Comput. 3 (2), 53–73.

Corno, F., De Russis, L., Roffarello, A.M., 2019. A high-level semantic approach to end-user development in the internet of things. Int. J. Hum. Comput. Stud. 125, 41–54.

Coutaz, J., Crowley, J.L., 2016. A first-person experience with end-user development for smart homes. IEEE Perv. Comput. 15 (2), 26–39.

Danado, J., Paternò, F., 2014. Puzzle: a mobile application development environment using a jigsaw metaphor. J. Visual Lang. Comput. 25 (4), 297–315.

Davidyuk, O., Milara, I.S., Gilman, E., Riekki, J., 2015. An overview of interactive application composition approaches. Open Comput. Sci. 5 (1).

Demeure, A., Caffiau, S., Coutaz, J., 2014. Activity based end-user-development for smart homes: relevance and challenges. In: Intelligent Environments (Workshops), pp. 141–152.

Desolda, G., Ardito, C., Matera, M., 2017. Empowering end users to customize their smart environments: model, composition paradigms, and domain-specific tools. ACM Trans. Comput.-Hum. Interact. (TOCHI) 24 (2), 12.

Dey, A.K., Hamid, R., Beckmann, C., Li, I., Hsu, D., 2004. a CAPpella: programming by demonstration of context-aware applications. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 33–40.

Engeström, Y., 1987. Learning By expanding: An activity-Theoretical Approach to Developmental Research. Orienta-Konsultit Oy, Helsinki, Finland.

Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M., 2003. The many faces of publish/subscribe. ACM Comput. Surv. (CSUR) 35 (2), 114–131.

Fischer, G., 2012. End user development and meta-design: foundations for cultures of participation. In: End-User Computing, Development, and Software Engineering: New Challenges. Hershey: IGI Global, pp. 202–226.

Fogli, D., Lanzilotti, R., Piccinno, A., 2016. End-user development tools for the smart home: a systematic literature review. In: In *Proceedings of the International Conference on Distributed, Ambient, and Pervasive Interactions*. Springer, Cham, pp. 69–79.

Forgy, C.L., 1982. Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif. Intell. 19 (1), 17–37.

Friedman-Hill, E., 2003. Jess in Action: Rule-Based Systems in JAVA. Manning Publications.

Friedman-Hill, E. (2008) Jess, the rule engine for the java platform. http://www.jessrules.com/jess/index.shtml.

Gaver, W.W., 1991. Technology affordances. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 79–84.

Ghiani, G., Manca, M., Paternò, F., Santoro, C., 2017. Personalization of context-dependent applications through trigger-action rules. ACM Trans. Comput.-Hum. Interact. (TOCHI) 24 (2), 14.

Gibson, J.J., 1979. The theory of affordances. The People, Place, and Space Reader 56–60.

Gibson, J.J., 2014. The Ecological Approach to Visual Perception: Classic Edition. Psychology Press.

Gilman, E., Sánchez, I., Saloranta, T., Riekki, J., 2010. Reasoning for smart space application: comparing three reasoning engines CLIPS, jess and win-prolog. In: In *International Conference on Computer and Information Technology*. IEEE, pp. 1340–1345.

Goumopoulos, C., Kameas, A., 2009. Ambient ecologies in smart homes. Comput. J. 52 (8), 922–937.

Goumopoulos, C., Kameas, A., Berg, E., 2012. A framework for developing Pervasive Awareness Systems in smart environments. Int. J. Ad Hoc Ubiquitous Comput. 9 (3), 142–158.

Goumopoulos, C., Lappa, A., 2017. Home-Based multi-parameter analysis for early risk detection and management of a chronic disease. In: International Conference on Information and Communication Technologies for Ageing Well and e-Health. Springer, Cham, pp. 46–68.

Goumopoulos, C., Nicopolitidis, P., Gavalas, D., Kameas, A., 2017. A distance learning curriculum on pervasive computing. Int. J. Continuing Eng. Educ. Life Long Learn. 27 (1–2), 122–146.

Goumopoulos, C., O'Flynn, B., Kameas, A., 2014. Automated zone-specific irrigation with wireless sensor/actuator network and adaptable decision support. Comput. Electron. Agric. 105, 20–33.

Greeno, J.G., 1994. Gibson's affordances. Psychol. Rev. 101 (2), 336–342.

Greeno, J.G., Moore, J.L., 1993. Situativity and symbols: response to VERA and SIMON. Cognit. Sci. 17 (1), 49–59.

Gross, T., Marquardt, N., 2007. CollaborationBus: an editor for the easy configuration of ubiquitous computing environments. In: *Parallel, Distributed And Network-Based Processing, 2007. PDP'07. 15th EUROMICRO International Conference on*, pp. 307–314 IEEE.

Hagras, H., Wagner, C., Kameas, A., Goumopoulos, C., Meliones, A., Seremeti, L., Minker, W., 2012. Symbiotic ecologies in next generation ambient intelligent environments. Int. J. Next-Gener. Comput. 3 (1), 52–86.

Heineman, G.T., Councill, W.T., 2001. Component-Based Software Engineering: Putting The Pieces Together. Addison-Wesley.

Huang, J., Cakmak, M., 2015. Supporting mental model accuracy in trigger-action programming. In: In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 215–225 ACM.

IFTTT (2019) IFTTT helps your apps and devices work together. https://ifttt.com/ (Accessed August 2019).

Kalofonos, D.N., Wisner, P., 2007. A framework for end-user programming of smart homes using mobile devices. In: *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*. IEEE, pp. 716–721.

Kameas, A., Goumopoulos, C., Hagras, H., Callaghan, V., Heinroth, T., Weber, M., 2009. An Architecture That Supports Task-Centered Adaptation in Intelligent environments. In *Advanced Intelligent Environments*. Springer, Boston, MA, pp. 41–66.

Kameas, A., Mavrommati, I., 2005. Extrovert gadgets. Commun. ACM 48 (3), 69.

Kaptelinin, V., 1996. Activity theory: implications for human-computer interaction. In: Context and Consciousness: Activity Theory and Human-Computer Interaction, 1, pp. 103–116.

Kaptelinin, V., Nardi, B.A., 1997. Activity theory: basic concepts and applications. In: CHI'97 Extended Abstracts on Human Factors in Computing Systems. ACM, pp. 158–159.

Kubitza, T., Schmidt, A., 2015. Towards a toolkit for the rapid creation of smart environments. In: International Symposium on End User Development. Springer, Cham, pp. 230–235.

Kymäläinen, T., 2015. The design methodology for studying smart but complex do-it-yourself experiences. J. Ambient Intell. Smart Environ. 7 (6), 849–860.

Leontiev, A.N., 1978. Activity, Consciousness, and Personality. Prentice-Hall, Englewood Cliffs, NJ.

Markopoulos, P., Nichols, J., Paternò, F., Pipek, V., 2017. End-User development for the internet of things. ACM Trans. Comput.-Hum. Interact. (TOCHI) 24 (2), 9.

Mavrommati, I., Darzentas, J., 2007. End User Tools For Ambient Intelligence environments: an overview. In: *International Conference on Human-Computer Interaction*. Springer, Berlin, Heidelberg, pp. 864–872.

Metaxas, G., Markopoulos, P., 2017. Natural contextual reasoning for end users. ACM Trans. Comput.-Hum. Interact. (TOCHI) 24 (2), 13.

Mugellini, E., Khaled, O.A., Pierroz, S., Carrino, S., Drissi, H.C., 2009. Generic framework for transforming everyday objects into interactive surfaces. In: *International Conference on Human-Computer Interaction*. Springer, Berlin, Heidelberg, pp. 473–482.

Mukherjee, C., 2018. Which rules engine is best for building smart applications? In: Build Android-Based Smart Applications. Apress, Berkeley, CA, pp. 3–14.

Norman, D., 2013. The Design of Everyday things: Revised and Expanded Edition. Basic Books, (AZ.

Papadopoulou, E., Gallacher, S., Taylor, N.K., Williams, M.H., 2012. A personal smart space approach to realising ambient ecologies. Pervasive Mob. Comput. 8 (4), 485–499.

Papert, S., 1986. Constructionism: A New Opportunity For Elementary Science Education. Institute of Technology, Media Laboratory, Epistemology and Learning Group, Massachusetts.

Paternò, F., Wulf, V., 2017. New Perspectives in End-User Development. Springer International Publishing.

Riley, G., 1991. CLIPS: an expert system building tool. In: N*ational Technology Transfer Conference and Exposition*, pp. 149–158.

Sabelli, N., 2008. Constructionism: a new opportunity for elementary science education. In: DRL Division of Research on Learning in Formal and Informal Settings, pp. 193–206.

Sanders, E.B.N., 2002. From user-centered to participatory design approaches. In: Design and the Social Sciences, pp. 18–25.

Satyanarayanan, M., 2001. Pervasive computing: vision and challenges. IEEE Pers. Commun. 8 (4), 10–17.

Seremeti, L., Goumopoulos, C., Kameas, A., 2009. Ontology-based modeling of dynamic ubiquitous computing applications as evolving activity spheres. Pervasive Mob. Comput. 5 (5), 574–591.

Togias, K., Goumopoulos, C., Kameas, A., 2010. Ontology-based representation of upnp devices and services for dynamic context-aware ubiquitous computing applications. In: International Conference on Communication Theory, Reliability, and Quality of Service, pp. 220–225.

Toussaint, A., 2003. Java rule engine api: Jsr-94. Java Community Process, pp. 8–33.
Ur, B., McManus, E., Pak Yong Ho, M., Littman, M.L., 2014. Practical trigger-action programming in the smart home. In: SIGCHI Conference on Human Factors in Computing Systems, pp. 803–812.
Venkatesh, V., Bala, H., 2008. Technology acceptance model 3 and a research agenda on interventions. Decis. Sci. 39 (2), 273–315.
Vygotsky, L.S., 1980. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.
Weiser, M., 1991. The computer for the twenty-first century. Sci. Am. 265 (3), 94–104.
Yarosh, S., Zave, P., 2017, May. Locked or not?: Mental models of IOT feature interaction. In: CHI Conference on Human Factors in Computing Systems. ACM, pp. 2993–2997.

Christos Goumopoulos is an Assistant Professor at the Department of Information and Communications Systems Engineering at Aegean University. He is also a cooperating professor in the Hellenic Open University in the Mobile and Pervasive Computing Systems postgraduate program. His research interests include pervasive computing system architectures, design and programming of pervasive computing systems, programming models and frameworks, middleware design, ontological knowledge representation, distributed systems, efficient resource scheduling systems, ambient assisted living systems and precision agriculture. He has more than 70 publications to his credit in international refereed journals, books and conferences (e.g. IEEE, ACM, Elsevier, Springer).

Irene Mavrommati is an Associate Professor at the Hellenic Open University, School of Applied Arts. She cooperates with Research Academic Computer Technology Institute as an experience/interaction design researcher, with roles in research projects conceptualization, acquisition and consortium coordination projects. She has previously been with Philips Design, the Netherlands, as a senior interaction designer/project manager (for AmI, ItTV, Car Navigation systems, etc.). She participates in art exhibitions with interactive installations, as well as technology exhibitions with research demo prototypes; she has authored several book chapters and research articles, with a focus in interaction within ambient computing environments.