ICST Transactions
on Ambient Systems

Research Article ICST.ORG

The homes of tomorrow: service composition and advanced user interfaces☆

Claudio Di Ciccio[1], Massimo Mecella[1,*], Mario Caruso[1], Vincenzo Forte[1], Ettore Iacomussi[1], Katharina Rasch[2], Leonardo Querzoni[1], Giuseppe Santucci[1], Giuseppe Tino[1]

[1]Dipartimento di Informatica e Sistemistica, University of Rome "La Sapienza", via Ariosto 25, I-00185 Rome, Italy; [2]KTH Royal Institute of Technology, School of Information and Communication Technology Forum 120, 16440 Kista, Sweden

Abstract

Home automation represents a growing market in the industrialized world. Today's systems are mainly based on *ad hoc* and proprietary solutions, with little to no interoperability and smart integration. However, in a not so distant future, our homes will be equipped with many sensors, actuators and devices, which will collectively expose services, able to smartly interact and integrate, in order to offer complex services providing even richer functionalities. In this paper we present the approach and results of SM4All- Smart hoMes for All, a project investigating automatic service composition and advanced user interfaces applied to domotics.

Keywords: advanced user interfaces, domotics, smart devices, smart homes, service composition, SM4All

Received on 24 May 2011; accepted on 11 July 2011

doi: 10.4108/trans.amsys.2011.e2

1. Introduction

Embedded systems, i.e. specialized computers used in larger systems in order to control the bundled equipments, are nowadays pervasive in immersive realities. For instance, they are widely adopted in those scenarios where both explicit and implicit interactions with human users are needed, in order to (i) provide continuous sensed information and (ii) react to service requests from the users themselves. Examples are digital libraries, eTourism applications, automotive appliances, next-generation buildings and domotics. Sensors/devices/appliances/actuators offering services are no more static, as in classical networks (e.g. for environmental monitoring and management or surveillance), but they form an overall distributed system that needs to continuously adapt instead. Such a task can be achieved by adding, removing and composing basic elements, i.e. the offered services.

This paper intends to outline some insights stemming from the European-funded project SM4All (Smart hoMes for All—http://www.sm4all-project.eu/), started on 1 September 2008 and finished on 31 August 2011. SM4All aims at studying and developing an innovative platform for software smart embedded services in immersive environments, based on a service-oriented approach and composition techniques. This is applied to the challenging scenario of private homes in the presence of users with different abilities and needs (e.g. young, elderly or disabled people).

In order to introduce the novel idea of services underlying SM4All, the reader should consider the following scenario: a person is at home and decides to take a bath. He/she would like to simply express this to the house; then, the available services should collaborate in order to move the house itself to a new state in which the desired situation holds. The temperature in the bathroom should be raised through the heating service, the

☆This work has been partly supported by the EU project FP7-224332 SM4All (http://www.sm4all-project.eu/).
*Corresponding author. Email: mecella@dis.uniroma1.it

wardrobe in the sleeping room opened in order to offer the bathrobe, the bath filled up with 37 °C water, etc. Some services, nonetheless, cannot be directly automated. If we consider a disabled user, the act of helping her to get to the bath can be considered an action offered by a service, though implemented, so to speak, by a human being, such as the nurse. We might suppose that she is notified (e.g. through her smartphone/tablet, while doing her job in another room) to go into the bath and help the patient at the right moment. This service could also be realized by the son of the patient (or any other person), living in a nearby house: thus, if the nurse is not at home, he is in turn asked to help the patient. The scenario draws the idea of a system of services, some of which are offered by completely automated systems (such as sensors, appliances or actuators), while the others are realized through the collaboration of other humans. Clearly, as in all the complex systems, there are tradeoffs to be considered: for instance, the goal of the person willing a relaxing bath could be in contrast with the availability of the nurse/son offering the 'help' service.

The rest of the paper is organized as follows: Section 2 provides a background on the current state of the art for home automation systems and other relevant techniques adopted in our approach, e.g. service composition, Section 3 gives the reader an overview of the SM4ALL system architecture. For sake of space, the remainder of the paper focuses only on some components, namely (i) the Pervasive Layer (Section 4), (ii) the Service Repository and the common service and data models used by all the components (Section 5), (iii) the Composition Layer (Section 6) and (iv) the User Interface (Section 7), which are among the most innovative ones produced by the project. Finally, Section 8 draws some conclusions.

2. Relevant work

Presently, we are assisting at a blooming of research projects on the usage of smart services at home and domotics, in particular for assisting people with physical or mental disabilities.

For instance, at Georgia Tech, a domotic home has been built for the elder adult with the goals of compensating physical decline, memory loss and supporting communication with relatives [1]. This work also considers issues of acceptability of domotics identifying key issues for the adoption of the technology by the end user. Acceptability, dangers and opportunities are also surveyed in [2]. Having a reliable system is a primary concern for all users.

At Carnegie Mellon, people's behavior is studied by automatic analysis of video images [3]. This is fundamental in detecting anomalies and pathologies in a nursing home where many patients live. Pervading the environment with active landmarks, called Cyber Crumbs, aims at guiding the blind by equipping him/her with a smart badge [4]. A number of projects to give virtual companions to people, to monitor people's health and behavioral patterns, and to help Alzheimer patients are presented in [5]. The Gator Tech Smart House [6] is a programmable approach to smart homes targeting the elder citizen. The idea is to have a service layer based on Open Services Gateway Initiative, in order to enable service discovery and composition. This work is close to what we propose as for the service-based approach, though it does not commit to any open standard or XML-based technology; no reference is made to the communication model adopted in the home and, most notably, there is no specific attention toward user interfaces.

Finally, in [7], the current adoption of service technologies for smart energy systems, including domotic ones, is discussed.

As far as service composition is concerned, there have been in the last years several works addressing it from different points of view. So far, the work on services has largely resolved the basic interoperability problems for service composition (e.g. standards such as WS-BPEL and WS-CDL exist and are widely supported in order to compose services, even if their applicability in embedded systems is still to be demonstrated), and designing programs, called orchestrators, that execute compositions by coordinating available services according to their exported description is the bread and butter of the service programmer [8, 9].

The availability of abstract descriptions of services has been instrumental in devising automatic techniques for synthesizing service compositions and orchestrators. Some works have concentrated on data-oriented services, by binding service composition to the work on data integration [10]. Other works have looked at process-oriented services, in which operations executed by the service have explicit effects on the system. Among these approaches, several consider *stateless* (also known as atomic) services, in which the operations that can be invoked by the client do not depend on the history of interactions, as services do not retain any information about the state of such interactions. Much of this work relies on the literature on Planning in AI [11–13]. Others consider *stateful* services which impose some constraints on the possible sequences of operations (also known as conversations) that a client can engage with the service. Composing stateful services poses additional challenges, as the composite service should be correct with regard to the possible conversations allowed by the component ones. Relevant approaches span over different areas, including research on Reasoning about Actions and Planning in AI, and research about Verification and Synthesis in Computer Science [14–17].

In this paper, we focus on composition of process-oriented stateful services. In particular, we have considered, extended and realized the framework for service composition adopted in [18–23], sometimes referred to as the

'Roman Model' [24, 25]. In the Roman Model, services are represented as transition systems (i.e. focusing on their dynamic behavior) and the composition aims at obtaining an actual composite service that preserves such an interaction. The composite service is expressed as a (virtual) target service specifying a desired interaction with the client.

Several research activities dealt with the idea of automatically generating different interfaces according to different variables, i.e. users, context and devices. The problem of generating different interfaces for different devices, having a single common application, is often indicated as the problem of creating *plastic interfaces* [26], i.e. creating user interfaces that adapt to devices characteristics.

Many ideas come from past research in *model-based* user interface design [27], where the designer is supposed to design an interactive system by editing and manipulating abstract models (e.g. task models) that describe the system's behavior and where the system is supposed to automatically generate the final application code.

The same idea of heavily exploiting formal models to design interactive applications comes from research on data-intensive web design, as illustrated in [28], that stems from past research on model-based hypermedia design, like RMM [29], and that has a major focus on data modeling.

The Abstract Adaptive Interface of SM4All diverges from these approaches because it does not have the goal of generating directly an interface. Instead, it provides all the pieces of information needed to design an interface that is suitable for the actual home status and for user preferences, in a parametric way. In particular, looking at the status of the available services, it provides the list of possible user actions together with associated icons and textual commands; the list is then ordered according to the home status and user preferences (cf. Section 7). Once such information is provided, it is possible to implement interfaces targeting different hardware platforms (e.g. smartphones or laptops) and interaction styles (e.g. icon based or menu based).

Finally, we would like to point out that some projects (e.g. EU-PUBLI.com [30] in an e-Government context and WORKPAD [31] in the field of emergency management) considered the issue of collaborating services, when some of which are not actually classical software applications, but human operators executing actions. As assumed in this work as well, they were all abstracted by the system as services and therefore seamlessly integrated into a general architecture.

3. The SM4All architecture

The goal of the SM4All architecture, shown in Figure 1, is to seamlessly integrate devices, in order to simplify the access to the services that they expose, and dynamically compose such services in order to offer the end users more complex functionalities and a richer experience with the domotic environment. There is an ever-increasing variety of devices, such as controlling parts of the home (doors, lights), media devices, etc. Sensors are devices for measuring physical quantities, ranging from simple thermometers to self-calibrating satellite-carried radiometers. Sensors and devices have an inherent connection, e.g. a device for opening the window blinds can change the luminosity value detected by a sensor. In SM4All, all the devices make their functionalities available according to the service-oriented paradigm. Due to the different technologies employed by the devices that are expected to interact within SM4All, the architecture relies on abstracting them as SOAP-based services, according to a riche service model (cf. Section 5.2) consisting not only of the service interface specification, but also of its conversational description, of the related graphical widgets (i.e. icons) to be presented in the user layer (by means of graphical interfaces, BCIs,...), etc. *Proxies* are indeed the software components offering such services by 'wrapping' and abstracting the real devices offering the functionalities. Services are not necessarily offered by hardware devices, but could also be realized through a human intervention; in this case, the proxy exposes a SOAP-based service to the platform above, whereas it interacts with the service provider (i.e. the human) by means of a dedicated graphical user interface, when executing the requested operations.

One particular service is the Localization Service, built on top of a subsystem for localizing persons[1] inside the home. It is in charge of tracking users in order to provide the location of each. The granularity of the provided information is at the level of presence inside a room (i.e. the service is able to state, e.g. whether 'Massimo is in the kitchen', though it cannot recognize his position within, e.g. in front of the oven, rather than on the chair nearby the table). Because of (i) the current advancements of indoor localization technologies, (ii) the requirements of the project and (iii) the consideration that indoor localization is a subject worthy of a research project *per se* (therefore, the research on this topic is out of the scope of SM4All), it is a sufficient and viable solution for the project.

During their run time, services continuously change their status, both in terms of values of sensed/actuating variables (e.g. a service wrapping a temperature sensor reports the current detected temperature, a service wrapping windows blinds report whether the blinds are open, closed, half-way, etc.) and in terms of their conversational state. The definition of the sensed/actuating variables, representing the 'state' of the domotic environment, is

[1]This subsystem is realized by adapting a commercial tool, namely the Ekahau Real-Time Location System (RTLS)—http://www.ekahau.com/.

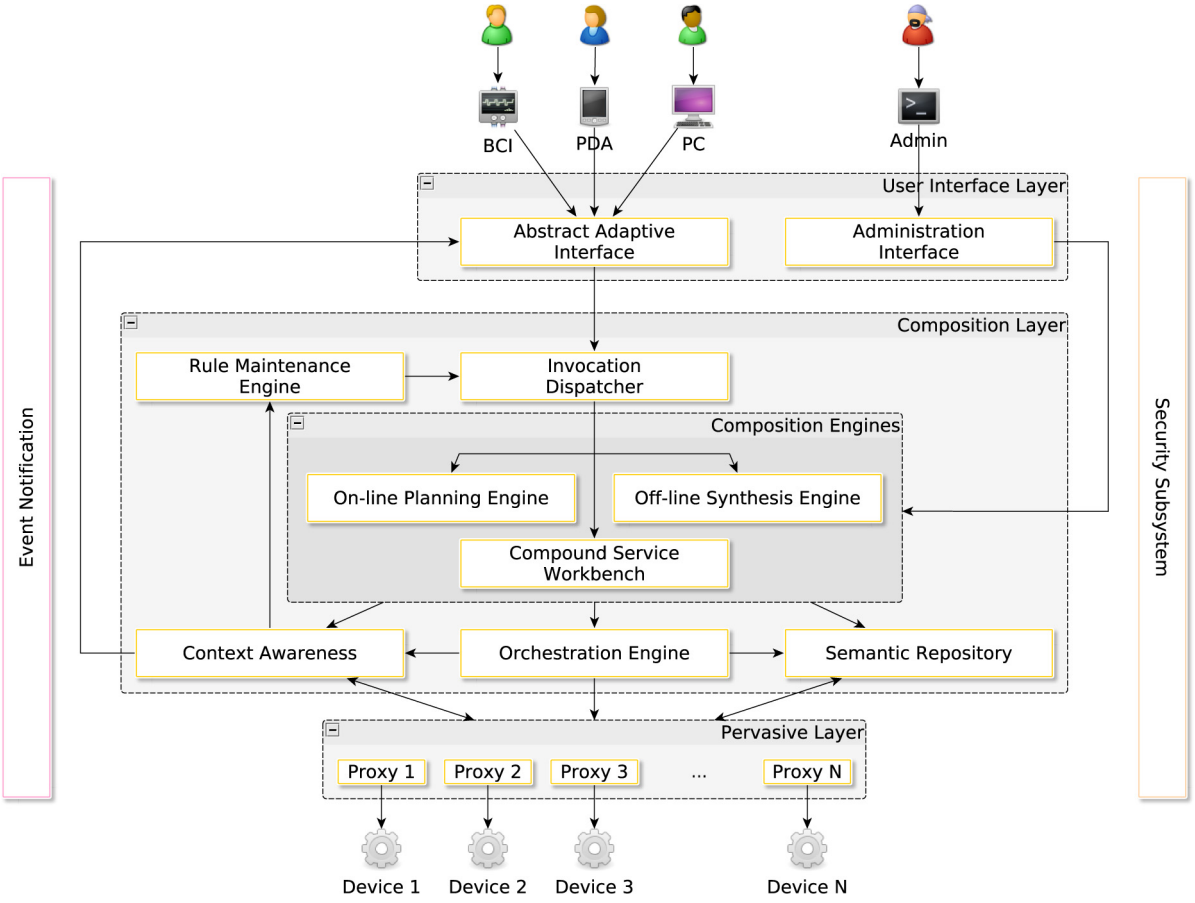

Figure 1. The SM4All Architecture.

performed in accordance with the *data model*. The data model, as well the service model, is designed to be 'additive', i.e. each new domotic device, plugged in the home, is expected to add new pieces to these models in order to register itself to the system.

A *Pervasive Controller* and a *Discovery Framework* are in charge, when a new device joins the system, to dynamically load and deploy the appropriate service, and to register all the relevant information into the *Service Semantic Repository*. All of the status information, both in terms of (i) service conversational states and (ii) values of the environmental variables, are kept available in the *Context Awareness Manager*, through a publish and subscribe mechanism.

Proxies, Pervasive Controller and Discovery Framework, together with the data and service models, constitute the *Pervasive Layer* of the SM4All architecture.

On the basis of the service descriptions, *Composition Engines* are in charge of providing complex services by suitably composing the available ones. In SM4All, three different types of approaches are provided, each providing different functionalities and therefore complementing one another, in order to provide a rich and novel environment to the users:

Off-line synthesis (provided through the Off-line Synthesis Engine). In the off-line mode, at design/deployment time of the house, a desiderata (i.e. not really existing) target service is defined, as a kind of complex routine, and the synthesis engine synthesizes a suitable orchestration of the available services realizing the target one. Such an orchestration specification is used at execution time (i.e. when the user chooses to invoke the composite/desiderata service) by the Orchestration Engine in order to coordinate the available services (i.e. to interact with the user on one hand and to schedule service invocations on the other hand). In this approach, the orchestration specification is synthesized off-line (i.e. not triggered by user requests, at run time) and executed on-line as if it were a real service of the home. The off-line mode is based on the Roman Model: it will be further detailed in Section 6. The Off-line Synthesis Engine produces what in SM4All is referred to as a *routine*.

On-line planning (provided through the On-line Planning Engine). The user, during its interaction with the home, may decide not to invoke a specific service (either available/real or composite), but rather to

ask the home to realize a *goal*; in such a case, the engine, on the basis of specific planning techniques [32], synthesizes and executes available service invocations in order to reach such a goal.

Visual design of complex services (provided through the Compound Service Workbench). A skilled user may want to define a *compound service*, by visually composing services offered by proxies, in a way similar to what currently happens in technologies like WS-BPEL. The compound service offers an aggregated operation, which is the result of the proper orchestration of operations offered by other services. Also in this case, the synthesis is performed off-line, but differently from the previous case, it is not supported by automatic techniques, but by a visual workbench. Both routines and compound services fall under the category of 'composite services'.

The *Orchestration Engine* interprets the specification of a composite service (either synthesized automatically, through the Off-line Synthesis Engine, or visually by the user, through the Compound Service Workbench) and consequently orchestrates the set of component services. In the case of the On-line Planning Engine, due to the need of continuously planning and monitoring services during plan executions, the Orchestration Engine is bypassed and services are directly invoked by the planner itself.

The *Rule Maintenance Engine* manages the automatic firing of actions when a predetermined situation occurs, i.e. the rules activated when given conditions hold. Rules can be defined by administrators through the User Layer. The triggering of rules to be applied, since they are automatic, is caused by changes in the environment and is therefore indirectly caused by the Context Awareness Manager when the predefined conditions are verified. Its output is the request to fulfill a (simple/composite) service or goal, as if the requests were generated by a user.

The *Invocation Dispatcher* is in charge of concretely invoking services (both simple or composite) and plans. All service invocations go through the Invocation Dispatcher. For example, if the Orchestration Engine, or the On-line Planning Engine, needs to invoke a service, the request is forwarded to the proper component through the Invocation Dispatcher. This is needed in order to differently manage requests for plans, basic and composite services, and allows a simple form of concurrency control (as it will be further clarified in the following).

Composition Engines, Invocation Dispatcher, Rule Maintenance Engine, Orchestration Engine, Context Awareness Manager and Semantic Service Repository constitute altogether the *Composition Layer* of the SM4ALL architecture.

Users are able to interact with the home and the platform through different kinds of user interfaces, e.g. a home control station accessible through a touchscreen in the living room. In particular, Brain–Computer Interfaces (BCIs) [33] allow also people with disabilities to interact with the system [34]. Of course, users can still control the home equipment as if there were not the SM4ALL platform. For example, a user is obviously allowed to switch the living room light on directly from the manual switcher on the wall, without using any BCI and/or touchscreen; in such a case, the platform, through the specific proxy wrapping the light/switcher as a service, is notified of the specific variable value change. Therefore, all needed actions are undertaken. De facto, the event is equivalent, due to the engineering of the platform, to the one of clicking a specific button on the touchscreen and/or selecting the icon on the BCI.

Users are able, through the interfaces, to invoke actions offered by services (either simple of composite) and to achieve goals, in order to reach specific situations that they would like to be realized in the home. Moreover, through the interfaces, they receive the feedback about state changes in the home, as well as requests for further inputs (in case additional parameters are needed for some actions to be executed), notifications about action/service completions, etc.

In order to abstract over the specific interfaces, the platform provides a unique façade component, namely the *Abstract Adaptive Interface,* which is in charge of (i) forwarding requests to the underlying layers and (ii) receiving messages from the latter, to be dispatched further to the proper interface. In such a way, the whole platform is unaware of the specific interfaces adopted by real users: particular implementation details are thus hidden. Indeed, there are specific algorithms needed to properly arrange icons on a BCI screen, while others are used for touch devices, and so forth.

Aside from the user interfaces, which are used by inhabitants to control and interact with their home, a specific *Administration Interface* is provided, in order to execute complex tasks, including (but not limited to):

- the definition of a target composite service to be synthesized off-line; this is performed by the domotic engineer at design/deployment time of the home, when her work is to define routines the users would like to run afterwards;
- the definition of rules;
- the definition, by a very skilled user, of a compound service trough the Compound Service Workbench.

As a rule of thumb, whichever task requiring the interaction with a user interface, though not strictly related to the direct control of the home, appears in the Administration Interface control panel.

Abstract Adaptive Interface and the various user interface modules are collected in the *User Layer* of the SM4ALL architecture.

Finally, an underlying *Event Notification Service* (e.g. for publishing updates of environmental variables) and a *Security Subsystem* (e.g. for AAA of users) complete the architecture.

3.1. Dealing with concurrency

Systems like SM4ALL are meant to be part of the environment where humans live, and are thus designed to allow interaction with multiple users at the same time. Moreover, the actions users can fulfill with these systems can be both limited in time or last for several tenths of minutes. As a consequence, several actions involving different devices can be in execution at the same time and this can easily lead to concurrency issues. As an example, suppose that Alice wants to bake a cake and this requires four eggs from the refrigerator. At the same time Bob wants to prepare fresh pasta that requires two eggs. However, the refrigerator currently contains a total of five eggs. There is clearly a contention among a set of limited resources (the eggs) that are needed to fulfill some goals. At the lower level physical devices part of the system can be used with different purposes by several actors at the same time. The SM4ALL typical scenario assumes that several humans act in the house together with the system. The system cannot take complete control of the house (as users should be free to act without necessarily interacting with the system), thus it can happen that a device involved in the execution of a specific procedure by the system is also maneuvered directly by a user.

At a higher level it can happen that a single specific device is considered part of the execution of several different procedures enacted by the SM4ALL system. These procedures can interact in different ways with the device, but nevertheless require exclusive access to it in order to avoid possible inconsistencies due to the interleaving among atomic actions pertaining to the executions of concurrent procedures. At an even more abstract level, but with strong practical implications in the system execution, is concurrency taking place in the physical environment where SM4ALL is deployed. Some devices, in fact, act by changing some aspects of the house global status (think about a heater that, when turned on, increases the temperature of the room where it is installed). These changes can negatively impact the execution of actions by other devices or change the way other concurrent procedures should be enacted. The possibilities seem to be infinite. Controlling concurrency at all the levels in order to prevent unexpected behaviors represents thus an extremely ambitious goal whose attainment appears far beyond the objectives of the SM4ALL project.

While a complete solution of the aforementioned issues goes beyond the scope of this project, nevertheless, we were faced with their practical implications during the design of the SM4ALL architecture. We decided to provide a practical solution that, even if far from being optimal from the performance and resource utilization point of view, is able to reduce the implications of concurrency issues during system usage.

At first level concurrency is handled directly on the device. Given the one-to-one mapping between each hardware device and a corresponding proxy deployed in the Pervasive Layer, the proxy handles concurrent request by serializing them. Concurrency can also be limited at the User Layer by enforcing single-action interactions with users: each user is allowed to issue a single command at a time to the system. If the execution of the issued command takes time, the user will not be able to issue other commands. This kind of interaction is enforced through visual elements in the user interface. Concurrent commands issued by independent users are allowed. A third point of synchronization is realized within the Composition Layer. Each command issued at the User Layer is passed down for plan execution (here we refer to 'plan execution' both in the case of direct invocation of a service and in the case of invocation of a routine or complex service, as well as of a goal). Two different paths are followed in this phase: either the command is related to a a simple action or an action previously synthesized (i.e. an action for which an orchestration exists), or the action is passed to the On-line Planning Engine to prepare a plan for it. Whichever the case is, the resulting invocations are passed to the Invocation Dispatcher. It checks if another execution plan is currently in execution. If this is the case, the Dispatcher checks if the plan that must be executed includes calls to services that are already considered by the plan currently in execution. If both checks are true, the Dispatcher enqueues the plan for later execution in order to avoid any possible clash with other concurrent executions.

4. The Pervasive Layer

The Pervasive Layer is in charge of communicating with home devices. It is able to manage the invocations coming from the upper layers, on one hand, and notify about the events which are generated by sensors, on the other hand. Nowadays, many home automation systems are commercially available, e.g. KNX, LonWorks and X10 are among the most common. Each is based on its own communication protocol, thus making it very difficult for them to be integrated inside a unique domotic system. However, the functionalities offered by the different home automation systems and their interworking models are basically the same. The Pervasive Layer is the middleware in charge of offering the home devices functionalities to the upper layers through a common standardized way. Every physical device installed in the house is managed by a specific software module, which is responsible for interacting with it and keeping track of its current state; this module contains all of the logic needed to communicate with the device. It is named *proxy*

in the SM4All architecture: in the SM4All context, it acts like a driver for an operating system. In order to provide high modularity and dynamicity, proxies are implemented as OSGi bundles; each bundle has its own life cycle which is completely independent from the others. Therefore, each proxy can be installed, started, stopped or removed from the system at run time: no system reboot or temporary stop is needed.

The skeleton of a proxy, with its SOAP-based communication stack, is generated automatically from the service interface. The developer of the proxy is in charge of implementing service details, according to the specific automation protocol. Every proxy exposes not only the interface descriptors and the related data types in use, but also behavioral descriptors (see Section 5.2) and user interface configuration details. When a proxy is started and plugged in the system, it registers itself to the Service Repository by means of a package containing all of the aforementioned details. From then on, the service offered by the proxy can be invoked from the upper layers and the current conversational state, as well as the controlled environmental variables values, can be retrieved at any point in time. At the beginning of its life cycle, the proxy also registers itself to the Context Awareness Manager as a publisher, so that SM4All components can be notified of the changes of states in an asynchronous way. In the current SM4All prototype, proxies are based on KNX.

The approach described above, which basically virtualizes devices, can potentially present some performance issues; hence, we conducted several tests, aiming at measuring the Total Round Trip Time (TOT RTT) of service/proxy invocations by a client (e.g. the user layer). The Total Round Trip Time includes (i) Proxy/KNX Round Trip Time (PK RTT), i.e. the time needed by the proxy for sending the command to the KNX bus and receiving the acknowledgment message back and (ii) proxy processing time (PPT), i.e. the time needed by the proxy to compute and refresh its own data structures.

Three main cases were identified and tested:

1. the client establishes its first connection to the proxy, does its first invocation and the proxy serves its first request (case 1 in Figure 2, right bars);
2. the client establishes its first connection to the proxy, does its first invocation served by a proxy which has already responded to some previous requests (case 2 in Figure 2, central bars);
3. the client invokes an operation offered by a proxy which has previously served some other request, on top of an already established connection between the two (case 3 in Figure 2, left bars).

We executed the tests on a local area network, in which the Pervasive Layer (i.e. the proxies and the KNX software layer used by all of them) is deployed on a EEE Box with an Intel Atom N330 dual core processor, 2 GB RAM and an Ubuntu distribution with the EIB/KNX demon. Each case was executed 100 times, and the average TOT RTTs and PK RTTs (over the 100 executions) for each case are depicted in Figure 2.

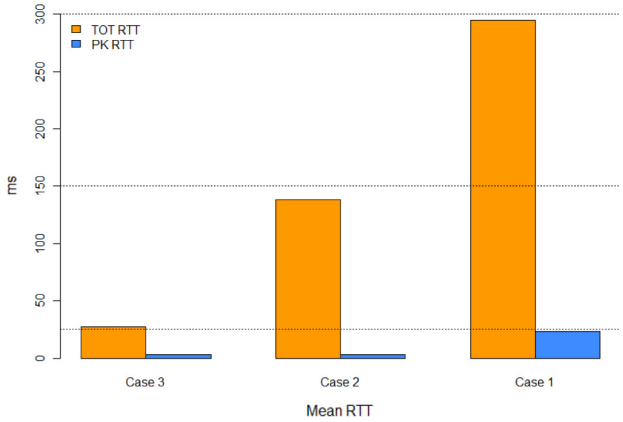

Figure 2. Performance tests.

As the reader can appreciate, in the typical scenario (case 3) the overall overhead due to the approach is about 25 ms, mostly due to internetworking time, which is absolutely not perceivable by a human (with respect to the case in which she switches on the light through a standard switcher on the wall, instead of using the SM4All system). In the worst case (case 3, which happens only after a full reset and restart of the system) the delay is of 300 ms, still acceptable.

5. The Service Repository and the models

5.1. Service Repository

The main functionality of the Service Repository is to manage the available services in the smart home, i.e. to register new services, unregister removed services and provide an interface for service retrieval. The Service Repository also serves additional information necessary for reasoning over the service descriptions, such as the data model which is used for describing environment variables.

When a new device is inserted into the system and the corresponding proxy is deployed into the Pervasive Layer, the proxy registers a template service description, which details the functionalities of the service, but is not yet fitted to the smart home it is installed in. Specifically, the template may refer to variables that only exist in the scope of the template and may not yet have been set up in the house. During service registration, the Service Repository therefore adapts the template description to reflect the actual house set-up. This includes registering new variables published by a service, e.g. the conversational status,

with the Context Awareness Manager, thereby making them available. The instantiated service description is from then on available for retrieval by other components. They are in particular the Composition Engines which need to query the Service Repository for services that can be used in compositions/plans. Typical queries for service matching concern the effects of the services, i.e. are aimed at finding those services which can change the user context in the desired way. Especially for the On-line Planning Engine, it is crucial that these requests are executed as fast as possible, so that planning can be performed without long waiting times for the user. In order to fulfill these requirements, the Service Repository uses a novel service indexing method and query algorithm, which was developed for SM4All and is described in detail in [35]. The basic idea of the indexing structure is to model context as a multi-dimensional space, where each environment variable corresponds to one context dimension. Service effects are described using projections of this space into a lower-dimensional space, which assigns value ranges to those context dimensions which are changed by the service.

During the registration of a new service, the service's effects are added to the in-memory index. Service matching requests are similarly transformed into low-dimensional projections representing the desired service effect. In the matching algorithm, first all services whose effects have no dimensions in common with the desired effect are filtered out using a fast bit-set operation. Only for the remaining services, it is then checked if the service effects conform to the desired effects and the matching services are returned. Using the described mechanisms, we have found that matching requests can be executed in less than 100 ms even for 1000 available services and 100 different context dimensions.

5.2. Service and data models

Service and composition models. The service model focuses on the behavior of services, in terms of conversational states that they traverse during the execution of the exposed actions, as well as on the way they (i) affect the environment and (ii) are inhibited (allowed) in the execution by the environment (respectively, by the expression of post-conditions and pre-conditions on top of the variables—see Section 5.2).

The smart home environment is populated by many deployed *service instance*s, which are actual occurrences of given *service types* (also *service*s for sake of brevity). Indeed, a developer can produce many instances showing the same behavior: e.g. many lamps of the same product series, installed in different rooms, are different instances of the same service type. Every service instance can be identified by one or more *properties*, which are deployment characteristics (such as the location in the house, the power consumption, etc.). Figures 3 and 4 show an

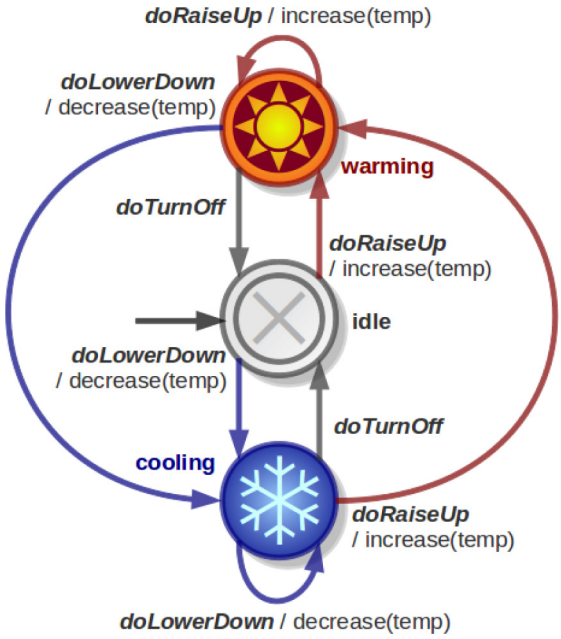

Figure 3. The `airConditionerService` transition system.

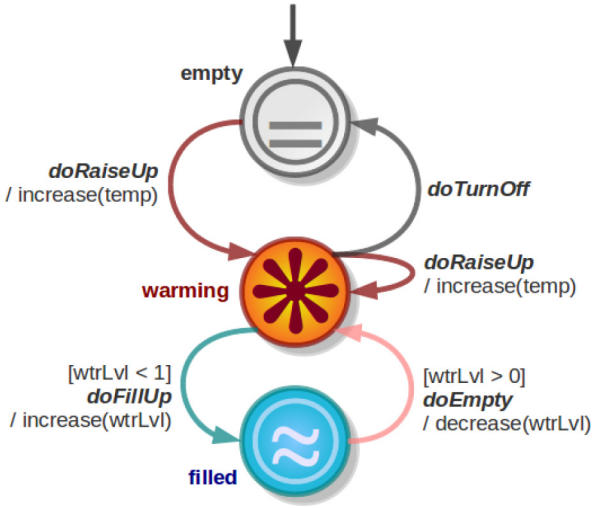

Figure 4. The `bathroomManagementService` transition system.

example with two service types: `airConditionerService` is supposed to be an air conditioner, while `bathroomManagementService` is a software manager of the bathroom.

Each service is represented by a *transition system*, which is a behavioral representation consisting in *state*s connected by *transition*s. The state is a break-point in the execution of a service (depicted as a node in the graphs of Figures 3 and 4), which new transitions (the arcs) can be fired from, through the invocation of an *action* (the emphasized part of the label). Action names are intended to correspond to the operation names offered

by the service interfaces (i.e. to the 'name' attribute in 'operation' nodes of WSDL files).

Transitions can be constrained by *pre-conditions* and *post-conditions* (effects) to verify, respectively, before and after the related action is executed. Such pre- and post-conditions are written as logic formulae over the set of home variables. In Figures 3 and 4, pre-conditions are written, when expressed, before the action name, between square brackets; post-conditions are put after the slash following the action name. The meaning of increase and decrease is graphically represented by Figures 5 and 6.

As further explained in the following, a user can ask the system to realize a given behavior, specified by a *target service* (or simply, target), as the one depicted in Figure 7. In particular, the Off-line Synthesis Engine returns a *composition*, i.e. an imperative program which, given the current target state and its next action to invoke, specifies which is the service instance to call the next action on, according to any possible coherent environmental status (in terms of both variables and service instances' states). Hence, it verifies the realizability of the target service by analyzing the actions, i.e. whether the paths admitted by transitions lead to consistent states with respect to the available services and the constraints set by pre- and post-conditions. Target services are described through the same syntax and semantics of any other service type.

The Compound Service Workbench works off-line as well. It allows the construction of new services (namely compound services), by means of a graphical toolkit. Whereas the target service must be specified as a transition system where each action corresponds to one of the actions that some available service offers, each compound service action can be a new action, defined as a structured sequence of the ones exposed by services. Its main utility is the gained ease in composing new target services:

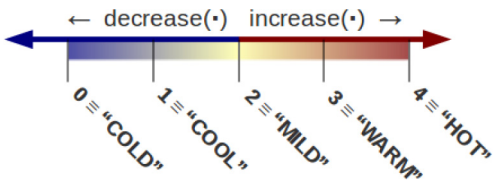

Figure 5. `temperatureLevel`.

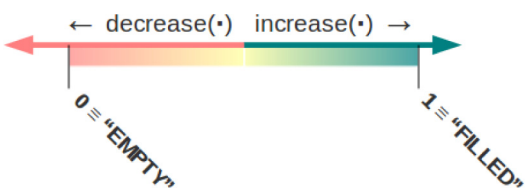

Figure 6. `waterLevelInTub`.

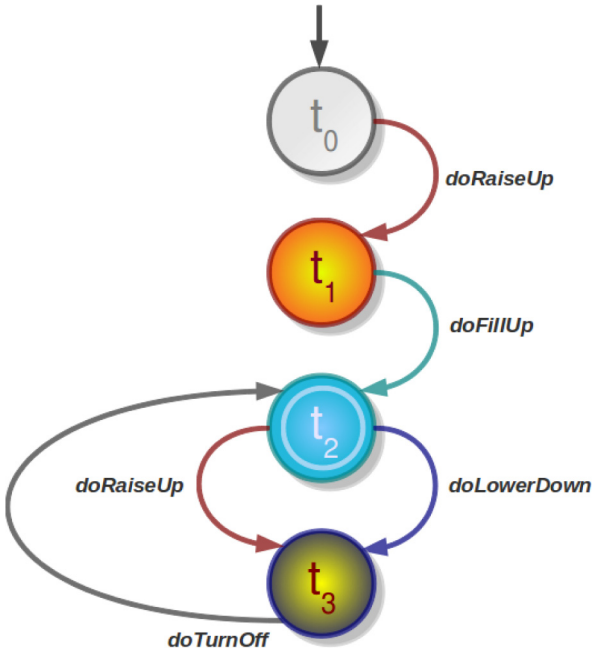

Figure 7. A target service, based on the component services of Figures 3 and 4.

indeed, the outcome of a synthesis can be wrapped in a single action and made part of another complex structure of automated invocations.

In case the user requires a goal to be immediately fulfilled, she can express the desired objective by means of formulae over the variables, and the On-line Planning Engine returns a plan, in terms of a structured path of invocations over actions, able to lead the environment to the desired status. Hence, being a planner, it builds the solution by considering the chain of effects and pre conditions (respectively, post- and pre-conditions) and thus establishing the actions to invoke in the proper order.

The service model is an XML standard for service descriptors. More than reported here, the XML Schemata which this model is built upon are published on-line at http://www.dis.uniroma1.it/~cdc/sm4all/proposals/servicemodel/latest, so to publicly show it as up to date with the latest version. Pre- and post-conditions are the natural link to the data model, described in Section 5.2.

Data model. The *data model* is an extensible framework of variable types. It concerns the specific environmental information used by reasoning engines only. That is, free parameters such as, e.g. name in an operation `cheers (name: string): string` may not adhere to the data model. Nevertheless, in case the developer wants (i) to describe the effects on the environment once a service action is invoked (post-condition), or (ii) to express the conditions that must hold in the context for an action to take place (pre-condition), she has to write statements

formulated on top of variables whose type is coherent with the data model.

This is due to the fact that both the Composition Layer and the User Layer must be able to cope with a predefined uniform set of common data types, so that the interaction with the environment is clear, despite the service developer. We call *variable types* (or simply *types*) the types, and *variables* are the entities whose type is a *variable type*. The data model is an XML standard, i.e. it is based on XML Schemata to define value spaces. Each service developer can define her own types, provided (i) they are described in XML Schema documents identified by a unique *namespace* and (ii) they extend, directly or indirectly, the SM4ALL *base types*.

The base types are identified by the http://www.sm4all-project.eu/datamodel/base namespace. They are published and kept up to date with the evolution of the standard on a public URL, namely http://www.dis.uniroma1.it/~cdc/sm4all/proposals/datamodel/latest. Indeed, types in the data model are derived by XML Schema native ones, and are designed to be extended by SM4ALL system service designers. The data model allows XML Schema simple types only as SM4ALL variable types, according to the XML Schema definition: complex types are not considered.

Common variable types are enumerations on top of the `numeric` type. This allows the ordering over the possible values, as inherited from the basic integer type (see Figures 5 and 6 for a visual representation of it). In such cases, the insertion of a `documentation` tag for each enumerated value provides also a human-readable form. The `documentation` node is intended to contain the information to show the users through the User Layer. That is to say: if, e.g. a variable of type `temperatureLevel` reaches the value 3, the reasoning engines are informed of it, whereas the users are notified of a new 'warm' status. Having enumerations over variables with finite sets of possible values makes feasible and effective the reasoning tasks of the composition engines.

6. Composition

The On-line Planning Engine performs service compositions at run time, i.e. as the user asks for a new plan it must return an orchestration to be invoked immediately after. As previously introduced, the representation of services is based upon their pre and post-conditions, i.e. logic formulae on top of environmental variables; goals as well are logic formulae on top of the same environmental variables, which the user expects to become true due to the enactment of the synthesized plan. The reasoning core is a planner that, as described in [32, 36], solves the underlying planning problem through the reduction of it into a CSP (Constraint Satisfaction Problem—see [37]).

In the Off-line Synthesis Engines, services are described as Transition Systems (TSs). Goals are in this case target TSs which the engine must realize by simulation, on top of the set of available services. Pre- and post-conditions are expressed as constraints over the TS transitions (see [38, 39]), on top of environmental variables. Once the orchestration is computed, the target itself is stored into the Service Repository: it can be invoked at any time in the future, like any other service. The returned orchestration is different from the On-line Planning Engine output. Indeed, it is a relation that, given the current target state and the next action to be fired, indicates which services can be invoked in order to enact it, according to any of the possible (i.e. coherent with the realizability of the goal) services' and environmental variables states. The solution approach is based on reducing the problem to the synthesis of Linear-time Temporal Logic formulae (see [40]) by Model Checking over Game Structures.

7. The User Layer

The home can be controlled from the user through different kinds of interfaces (BCIs, remote controls, touchscreens, keyboards, voice recognition, etc.). The *AAI* (Abstract Adaptive Interface) represents the core of the SM4ALL User Layer. It retrieves status information from the Context Awareness Manager and service descriptions from the Service Repository. It organizes the whole in order to correctly show the available actions to the user, depending on the interaction mode she is currently making use of (i.e. visual, aural, BCI, etc.). Indeed, the AAI is intended to be put as an abstraction layer among the multiple user interface devices and the underlying composition layer.

Its main novelty is represented by the ability to manage many different user interface models with a unique adaptable algorithm, able to change itself on the basis of the interaction device characteristics (speech/aural, visual/touch, handheld, brain-controlled ...) and on the basis of the user preferences, automatically gathered, analyzed and synthesized on top of the previous interactions with the system.

Through a message screen the user can see notifications coming from the system. The room actions' screen shows the list of actions that can be invoked, gathered up by groups which are built according to the rooms where the services offering the actions are actually located. The number of available services in the home can be very high, and a service can offer many actions; on the other hand, the icons that can be shown on a screen are limited. A pagination of the information, though useful and indeed exploited in many prototypes, is not sufficient to provide an effective interaction, since it would introduce a huge effort for the user to find the desired element among the big amount of items, navigating back and

forth. Hence, in SM4ALL, the AAI integrates a mechanism for grouping and smartly ordering the icons in order to improve the ease of interaction. An icon may represent either a service or an action. Sometimes, only a few actions, among the ones offered by a given service, are available, e.g. a 'bedroom light' service offers a 'turn off' and a 'turn on' action, but only the first (or the second, conversely) can be triggered when the lamp is switched on (off). In such a case, there is no need to show the service icon, as the only available action is enough.

When the user can fire more than one action, related to a single service, a clustering is needed. It is realized by initially showing the service icon; once activated, all of the other items are hidden and only the available related actions are displayed.

Another way to reduce the number of displayed icons is to divide services themselves into groups represented by a given type (e.g. 'Multimedia' for televisions, MP3 players, etc.). The idea is almost the same: at first, the menu shows only a type which many services belong to, and then, after the type is selected, all of the other items are hidden and the only related services are displayed (see Figures 8 and 9).

Beyond grouping, the AAI exploits the possibility to order the items according to their importance, with respect to the preferences of each user. This way, the actions which are known to be more relevant for the user will be displayed on the first screen, in order to appear at a first glimpse, while the others are going to be shown next. Two algorithms are offered: a *static* one and a *dynamic* one. The user can select which one she prefers through

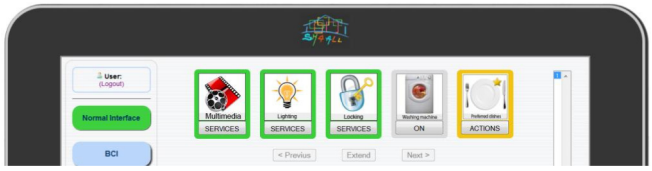

Figure 8. An example of the user interface grouping services by the type they belong to (here, before activating the 'Multimedia' services type selector—first icon, on the left of the horizontal list).

Figure 9. An example of the user interface grouping services by the type they belong to (here, after the 'Multimedia' services type selector is activated).

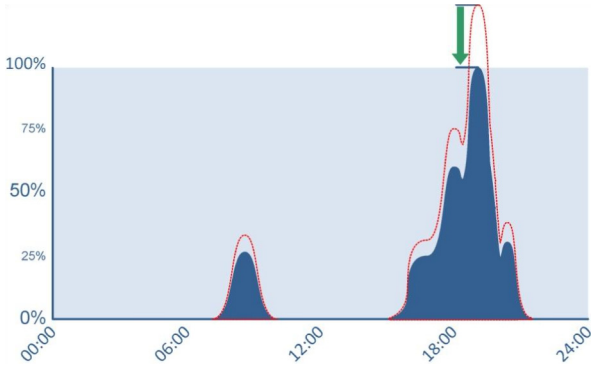

Figure 10. An example of automatic normalization of the parametric score.

an administration menu. The static algorithm makes use of explicitly defined user settings to identify her preferences. Each preference is constituted by (i) a set of conditions, representing the state of the environment which enables the action, (ii) a time frame in which the preference has to be considered (Always, Morning, Afternoon, Evening, Night) and (iii) a usage expectation degree (certain, highly probable, very probable, probable). The dynamic algorithm orders the actions according to the probability that each one is going to be executed, on the basis of the current environment status and previous invocations: the higher the probability, the higher the priority of the associated icon in the list (*partial order*). The home environment status consists of several parametric values related to the execution (e.g. the time of invocation). Each parameter is associated to a *relevance* (*weight*), manually tunable by administrators. At every call, the parametric value is computed and its *incidence* (*score*) re-calculated. Indeed, it is taken from a run-time updated graph, i.e. a normalized sum of Gaussian curves: at each execution, a new Gaussian centered in the parameter value which is associated to the call (e.g. the time of invocation) is added to the previous graph. In order to tune the evolution of the curve, norm and variance of the Gaussians are both customizable. If the global peak overcomes the maximum admissible value (100%), a normalization is automatically performed (see Figure 10). The probability is thus the *sum* of the *weighted scores* (Σ_i *relevance*$_i$ × *incidence*$_i$) of all the parameters, related to the current home environment status.

8. Concluding remarks

Throughout this paper, we presented the pervasive intelligent home system SM4ALL, and we focused, among the others, on the service composition techniques and on the self-adapting ones of the User Layer: they are the most involved in the challenge of hiding the heterogeneity of used hardware devices to the other software modules, which is a very important requirement in the field of dom-

otics, where a lack of standardization still holds. Currently, we have developed a running prototype interfaced with real KNX devices actually installed in a house set up on purpose in Rome, hosted by Fondazione Santa Lucia. A showcase has been demonstrated in October 2011; this has shown the wide acceptability of the system by both normal-bodied users and disabled ones. However, here we showed some performance tests demonstrating the feasibility of the approach.

References

[1] Mynatt, E., Melenhorst, A., Fisk, A. and Rogers, W. (2004) Understanding user needs and attitudes. *IEEE Pervasive Comput.* **3**(2): 36–41.

[2] Roberts, J. (2006) Pervasive health management and health management utilizing pervasive technologies: synergy and issues. *J.UCS* **12**(1): 6–14.

[3] Hauptmann, A., Gao, J., Yan, R., Qi, Y., Yang, J. and Wactlar, H. (2004) Automatic analysis of nursing home observations. *IEEE Pervasive Comput.* **3**(2): 15–21.

[4] Ross, D. (2004) Cyber crumbs for successful aging with vision loss. *IEEE Pervasive Comput.* **3**(2): 30–35.

[5] Joseph, A. (2004) Successful aging. *IEEE Pervasive Comput.* **3**(2): 36–41.

[6] Helal, S., Mann, W.C., El-Zabadani, H., King, J., Kaddoura, Y. and Jansen, E. (2005) The Gator Tech Smart House: a programmable pervasive space. *IEEE Comput.* **38**(3): 50–60.

[7] Paradiso, J., Dutta, P., Gellersen, H. and Schooler, E. (2011) Smart energy systems. Special issue. *IEEE Pervasive Comput.* **10**.

[8] Alonso, G., Casati, F., Kuno, H. and Machiraju, V. (2004) *Web Services. Concepts, Architectures and Applications* (Springer).

[9] Papazoglou, M. (2008) *Web Services: Principles and Technology* (Pearson Education).

[10] Michalowski, M., Ambite, J.L., Knoblock, C.A., Minton, S., Thakkar, S. and Tuchinda, R. (2004) Retrieving and semantically integrating heterogeneous data from the Web. *IEEE Intell. Syst.* **19**(3): 72–79.

[11] Blythe, J. and Ambite, J.L. [eds.] (2004) In *Proceedings of ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services.*

[12] Cardoso, J. and Sheth, A. (2004) Introduction to semantic Web services and Web process composition. In *Proceedings of SWSWPC 2004* .

[13] Wu, D., Parsia, B., Sirin, E., Hendler, J.A. and Nau, D.S. (2003) Automating DAML-S Web services composition using SHOP2. In *Proceedings of ISWC 2003.*

[14] Bultan, T., Fu, X., Hull, R. and Su, J. (2003) Conversation specification: a new approach to design and analysis of eService composition. In *Proceedings of WWW 2003.*

[15] Gerede, C.E., Hull, R., Ibarra, O.H. and Su, J. (2004) Automated composition of eServices: lookaheads. In *Proceedings of ICSOC 2004.*

[16] McIlraith, S.A. and Son, T.C. (2002) Adapting GOLOG for composition of semantic Web services. In *Proceedings of KR 2002.*

[17] Pistore, M., Traverso, P. and Bertoli, P. (2005) Automated composition of Web services by planning in asynchronous domains. In *Proceedings of ICAPS 2005.*

[18] Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M. and Mecella, M. (2003) Automatic composition of eServices that export their behavior. In *Proceedings of ICSOC 2003.*

[19] Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M. and Mecella, M. (2005) Automatic service composition based on behavioral descriptions. *Int. J. Coop. Inf. Syst.* **14**(4): 333–376.

[20] Berardi, D., Calvanese, D., De Giacomo, G. and Mecella, M. (2005) Composition of services with nondeterministic observable behavior. In *Proceedings of ICSOC 2005.*

[21] Berardi, D., Cheikh, F., De Giacomo, G. and Patrizi, F. (2008) Automatic service composition via simulation. *Int. J. Found. Comput. Sci.* **19**(2): 429–451.

[22] Muscholl, A. and Walukiewicz, I. (2007) A lower bound on Web services composition. In *Proceedings of FOSSACS 2007.*

[23] Sardina, S., Patrizi, F. and De Giacomo, G. (2008) Behavior composition in the presence of failure. In *Proceedings of KR'08.*

[24] Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M. and Patrizi, F. (2008) Automatic service composition and synthesis: the Roman model. *IEEE Data Eng. Bull.* **31**(3): 18–22.

[25] Hull, R. (2005) Web services composition: a story of models, automata, and logics. In *Proceedings of IEEE ICWS.*

[26] Thevenin, D. and Coutaz, J. (1999) Plasticity of user interfaces: framework and research agenda. In *Proceedings of Interact'99.*

[27] Puerta, A. and Eisenstein, J. (1999) Towards a general computational framework for model-based interface development systems. In *Proceedings of 4th International Conference on Intelligent User Interfaces.*

[28] Fraternali, P. (1999) Tools and approaches for developing data-intensive Web applications: a survey. *ACM Comput. Surv.* **31**(3): 227–263.

[29] Isakowitz, T., Stohr, E. and Balasubramanian, P. (1995) RMM: a methodology for structured hypermedia design. *Commun. ACM* **38**(8): 34–44.

[30] Contenti, M., Mecella, M., Termini, A. and Baldoni, R. (2005) A distributed architecture for supporting e-Government cooperative processes. In *Proceedings of TCGOV 2005.*

[31] Catarci, T., de Leoni, M., Marrella, A., Mecella, M., Salvatore, B., Vetere, G., Dustdar, S. *et al.* (2008) Pervasive software environments for supporting disaster responses. *IEEE Internet Comput.* **12**: 26–37.

[32] Kaldeli, E., Lazovik, A. and Aiello, M. (2009) Extended goals for composing services. In *Proceedings of ICAPS 2009.*

[33] McFarland, D. and Wolpaw, J. (2011) Brain–computer interfaces for communication and control. *Commun. ACM* **54**(5): 60–66.

[34] Aloise, F., Schettini, F., Aricò, P., Bianchi, L., Riccio, A., Mecella, M., Babiloni, F. *et al.* (2010) Advanced brain–computer interface for communication and control. In *Proceedings of AVI 2010*.

[35] Rasch, K., Li, F., Sehic, S., Ayani, R. and Dustdar, S. (2011) Context-driven personalized service discovery in pervasive environments. *World Wide Web*: http://dx.doi.org/10.1007/s11280-011-0112-x.

[36] Kaldeli, E. (2009) Using CSP for adaptable Web service composition. Tech. Rep. 2009-7-01, University of Groningen. www.cs.rug.nl/~eirini/tech_rep_09-7-01.pdf.

[37] Do, M. and Kambhampati, S. (2000) Solving planning-graph by compiling it into CSP. In *Proceedings of AIPS'00*.

[38] De Masellis, R., Di Ciccio, C., Mecella, M. and Patrizi, F. (2010) Smart home planning programs. In *Proceedings of ICSSSM 2010*.

[39] Patrizi, F. (2009) *Simulation-based techniques for automated service composition*. Ph.D. thesis, Department of Systems and Computer Science and Engineering, SAPIENZA—Università di Roma, Rome, Italy.

[40] Pnueli, A. and Rosner, R. (1989) On the synthesis of a reactive module. In *Proceedings of POPL'89*.