Service Ecologies for Home/Building Automation

Mario Caruso * **Claudio Di Ciccio** * **Ettore Iacomussi** *
Eirini Kaldeli ** **Alexander Lazovik** ** **Massimo Mecella** *

* *Sapienza Università di Roma*
Dipartimento di Ingegneria Informatica Automatica e Gestionale, Italy
e-mail: {caruso/cdc/iacomussi/mecella}@dis.uniroma1.it
** *University of Groningen*
Johann Bernoulli Institute, The Netherlands
email: {e.kaldeli/a.lazovik}@rug.nl

Abstract: Service ecologies are networks of services pervasively embedded in everyday environments, e.g., smart homes, where they are composed and orchestrated in order to provide advanced functionalities. In this paper, we show how the interplay of off-line and on-line composition of services can improve flexibility and adaptiveness.

Keywords: smart homes, automatic service composition, planning, sensors and actuators,

1. INTRODUCTION

Robotic ecologies are networks of heterogeneous robotic devices pervasively embedded in everyday environments, where they cooperate in applications such as security and ambient assisted living. Robotic ecologies are an emerging paradigm, which crosses the borders between the fields of robotics, sensor networks, and ambient intelligence. Instances of this paradigm include network robot systems, smart environments, sensor-actuator networks, ubiquitous robotics, etc. often referred to as physically embedded intelligent systems (PEISs). Common to these systems is the fact that the term "robotic device" is taken in a broad sense, including mobile robots, static sensors or actuators, and automated home appliances. One interesting challenge in such robotic ecologies is how to make them self-adaptive, so as to reduce the amount of preparation and pre-programming required for the deployment in real world applications.

An interesting solution to the problem of coordinating heterogeneous devices and generating/executing automatic plans is proposed in Lundh et al. [2007]; in this work, robotic devices, including home appliances, are able to share their functionalities through a distributed P2P middleware, in order to plan and perform complex tasks. The work is extended by Gritti et al. [2007], in which a more dynamic environment is considered, including robot crashes and failures: whenever dynamic changes of the environment lead to an inconsistent situation, a technique is re-triggered in order to restore the system into a correct configuration.

On the other hand, Service Oriented Computing (SOC) is a computing paradigm whose basic elements are services that can be used as building blocks to devise other services. In the last years, many approaches have investigated automated techniques for service composition and adaptiveness, through the use of different techniques (Calvanese et al. [2008]). The availability of abstract descriptions of services has been instrumental to 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 (Michalowski et al. [2004]). 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* (a.k.a. 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 (Wu et al. [2003], Blythe and Ambite [2004], Cardoso and Sheth [2004]). Others consider *stateful* services which impose some constraints on the possible sequences of operations (a.k.a. conversations) that a client can engage with the service. Composing stateful services poses additional challenges, as the composite service should be correct w.r.t. 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 (Bultan et al. [2003], Pistore et al. [2005], Gerede et al. [2004], McIlraith and Son [2002]).

Particularly relevant is the framework for service composition adopted in Berardi et al. [2003, 2005a,b], Sardina et al. [2008], Muscholl and Walukiewicz [2007], Berardi et al. [2008], sometimes referred to as the "Roman Model" (Hull [2005], Calvanese et al. [2008]). Some works (e.g., De Giacomo and Sardiña [2007], Sardina et al. [2008]) have already pointed out how service composition is an instance of a more general problem of composition of agents belonging to a given community/ecology.

Presently, we are assisting at a blooming of research projects on the usage of smart services at home and

Pre-print copy of the manuscript published by International Federation of Automatic Control
identified by DOI: 10.3182/20120905-3-HR-2030.00191

domotics, in particular for assisting people with physical or mental disabilities, e.g., at Georgia Tech (Mynatt et al. [2004]), at Carnegie Mellon (Hauptmann et al. [2004]), at the Gator Tech Smart House (Helal et al. [2005]), just to name a few.

The SM4All (Smart hoMes for All) EU research project, recently and successfully concluded [1], has studied and developed an innovative middleware platform for interworking of smart embedded services in immersive and person-centric environments, such as private homes in presence of users with different abilities and needs (e.g., young, able-bodied, elderly and disabled people) [2]. In particular, it considered home automation service networks, i.e., pervasive environments consisting of several sensors and actuators able to interact in order to provide a better user experience in everyday life, allowing users to perform their tasks efficiently and unobtrusively. To this end, the SM4All system is able to build more complex services by applying automatic composition algorithms, on top of simple services, offered by home automation devices. Here, we will discuss how the interplay of two different techniques (namely, off-line synthesis and dynamic planning) allows these compositions to be context-aware and self-adaptable, so as to be able to dynamically adjust to changes in the environment and recover from failures.

The reader should note that the concepts discussed in this paper are not strictly tied to services, but can be applied to computational entities in general, such as robots, agents, etc. Such concepts, illustrated through the case of SM4All, demonstrate that a layered architecture can be effective in gaining flexibility and adaptiveness. In such a conceptual architecture, basic entities at the bottom take care of simple tasks. They expose rich descriptions of their behaviors, both in terms of conversations and effects on the environment. On top of them, more complex entities are in charge of composing and orchestrating the behaviors of the single entities, adaptable to environmental changes. In particular, we will present a framework for centralized composition of entities exposing non-deterministic behaviors within a distributed environment. Such behaviors are not intended to directly depend on the behavior of any other entity; i.e., there is no action directly enabling or disabling the execution of other entites' actions. There are implicit links only, which are *(i)* the unique name assumption (two actions identified by the same name are considered the same action), and *(ii)* the side-effects of actions on the physical shared world that appliances are in.

[1] cf. news on major international televisions: Brazilian Globo TV – `http://video.globo.com/Videos/Player/Noticias/0,,GIM1751401-7823-CASA+INTELIGENTE+E+MOVIDA+A+PENSAMENTO+NA+ITALIA,00.html`, Channel 1 Russia – `http://www.1tv.ru/news/other/191509`, Italian Rai3 – `http://www.youtube.com/watch?v=a9F72_E4mT0` and `http://rai.it/dl/tg3/rubriche/PublishingBlock-79554b45-1e4c-41a8-a474-ad3e22ab750f.html#`, Ability Channel – `http://www.abilitychannel.tv/video/casa-domotica-sm4all/`

[2] In the follow-up project Greener Buildings, among other research issues, the authors are applying similar techniques of service composition for energy saving in buildings, on the basis of a similar architecture.

The remainder of the paper is organized as follows. Section 2 provides an overview of the SM4All architecture. Section 3 discusses basic services, including their semantic model. The automatic composition is described in Section 4, along with a motivating scenario that demonstrate the role of each technique, and finally the paper concludes with Section 5.

2. ARCHITECTURE

The goal of the SM4All architecture is to seamlessly integrate devices, in order to simplify the access to the services that they expose. Moreover, services are dynamically composed, 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 home appliances. We can conceptually divide them into *actuators* and *sensors*. Actuators are the controlling parts of the home (doors, lights), media devices, etc. Sensors are those devices that measure physical quantities, ranging from simple thermometers to self-calibrating satellite-carried radiometers. Sensors and actuators have an inherent connection: e.g., a device for opening the window blinds can change the level of luminosity 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 the abstraction of them as SOAP-based services, according to a rich *service model* consisting not only of the service interface specification, but also of its conversational description, of pre-conditions and effects of the operations, of the related graphical widgets (i.e., icons) to be presented in the user interfaces.

Proxies are the software components offering such services by "wrapping" and abstracting the real devices that provide the functionalities. Services are not necessarily offered by hardware devices only, but could also be realized through a human intervention; in this case, the proxy exposes a SOAP-based service, whereas it interacts with the service provider (i.e., the human) by means of a dedicated GUI, when executing the requested operations.

During the 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 performed in accordance with the *data model*.

Services register all the specifications 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&subscribe mechanism.

On the basis of the service descriptions, *Composition Engines* are in charge of providing complex services by suitably composing the available ones. In SM4All, different types of approaches are provided, each providing different

Pre-print copy of the manuscript published by International Federation of Automatic Control
identified by DOI: 10.3182/20120905-3-HR-2030.00191

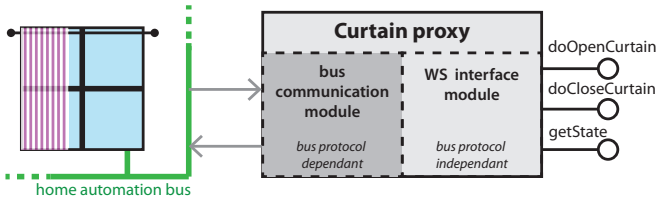

Fig. 1. Curtain proxy

functionalities and therefore complementing one another, in order to provide a rich and novel environment to the users:

Off-line synthesis (provided by 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, and the synthesis engine synthesizes a suitable orchestration of the available services realizing the target. Such an orchestration specification is used at execution-time (i.e., when the user chooses to invoke the routine) 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 at run-time as if it were a real service of the home. The off-line mode is based on the "Roman Model". The Off-line Synthesis Engine produces what in SM4All is referred to as *composite service*, or simply *routine*.

On-line synthesis (provided through the Dynamic Planner). 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*, i.e., a situation to hold in the house; in such a case, the Dynamic Planner synthesizes and immediately enacts the available service invocations in order to reach such a goal, on the basis of specific planning techniques described in Kaldeli et al. [2009]. The sequence of service invocations is called *plan*.

Users are able to interact with the home and the platform through different kinds of user interfaces. Of course, users can still control the home equipment as if there were not the SM4All platform. E.g., a user is obviously allowed to switch the living room light on directly from the manual switcher on the wall, without using any touchscreen; in such a case, the platform is notified of the specific change in the value of the variable, by the specific proxy that wraps the light-switcher as a service. The event is equivalent, de facto, to the one of clicking a specific button on the touchscreen. Users are able to either invoke actions offered by services (either real or composite), or to achieve goals. Moreover, they can 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. through the different user interfaces which the house is equipped with.

3. SERVICE ECOLOGIES

A proxy is a software component that acts as an intermediary between an home automation device and the rest of the system. Usually a proxy is related to a single specific actuator/sensor, e.g., a lamp proxy simply commands a binary actuator connected to the light bulb. Nonetheless, there are cases in which a proxy abstracts a home device by managing more than one actuator/sensor, e.g., the curtain proxy manages a motor actuator needed to move the curtain and two magnetic sensors used as limit switches. As depicted in Figure 1, a proxy consists of a Web service interface module that is completely independent of the home automation protocol, thus hiding its complexity and specificity. The interface contains all of the actions that can be invoked on the device and a method used to synchronously retrieve its current state. The rest of the proxy (bus communication module) is in charge of translating Web service invocations in actuation commands, that are further sent to the device, according to the specific automation protocol.

Proxies are intended to be connected on a field bus that they continuously monitor for retrieving environmental information. As soon as a raw event involving the bounded device is captured, the proxy is demanded to translate it into a common shared language and throw the information up to the Context Awareness Manager.

A proxy provides only a Web service with a static description of its interface encoded in WSDL (Web Services Description Language), i.e., in terms of the operations it offers, along with the accepted arguments and return values. Thanks to this kind of description, other components can properly invoke the operation it provides. However, this description is syntactic, and does not capture what each operation really entails, i.e., what its expected effect on the environment is, and what the restrictions and requirements that have to be met before the invocation can safely take place are. In order to specify these aspects, service descriptions must be enhanced with appropriate semantics. These extra annotations are necessary so that the automatic composition engines can perform complex reasoning on top of the services.

3.1 Service Behavior

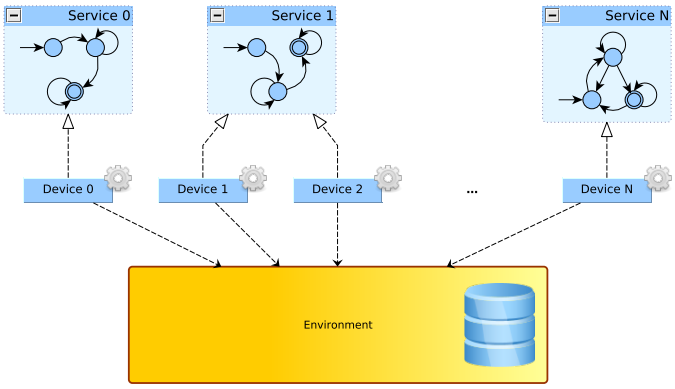

Fig. 2. The interaction of services with the home environment

Pre-print copy of the manuscript published by International Federation of Automatic Control
identified by DOI: 10.3182/20120905-3-HR-2030.00191

Each service is associated with a rich behavioral description, at two levels: *(i) conversational*, i.e., representing the service as a transition system that specifies the states it traverses during the execution of the exposed operations (labelling the transitions) when invoked by a client, and *(ii) declarative*, i.e., describing *pre-conditions* and *effects* of the operations.

Pre-conditions and effects are propositional formulas expressed on variables that represent the home environment status. In other words, they are the link between the service as a software artifact and the real world, which the underlying device is physically in (see Figure 2).

Figure 3 shows an example for an automated curtain. Each state in the transition system is a break-point in the execution of a service, and the transitions represent the operations that can be fired from a given state. Transitions are constrained by pre-conditions and effects, which are respectively verified before and after the execution of the related action. In Figure 3, pre-conditions are written above the transitions' names whereas effects are put below. In the example, we consider the window to open inwards. Therefore, it might interfere with the casements when attempting to close it. To this aim, the `doCloseCurtain` action has an extra pre-condition that the window should be closed (i.e., `windowStatus == 0`). This semantics are expressed in XML. As an example, the *LivingRoomCurtain* service behavior is encoded like this:

```
<ts>
  <state name="closed" type="initial-final">
    <transition action="doOpenCurtain">
      <target state="opened"/>
    </transition>
    <transition action="doCloseCurtain">
      <target state="closed"/>
    </transition>
  </state>
  <state name="opened" type="initial-final">
    <transition action="doOpenCurtain">
      <target state="opened"/>
    </transition>
    <transition action="doCloseCurtain">
      <target state="closed"/>
    </transition>
  </state>
</ts>

<declare>
  <action name="doOpenCurtain">
    <pre>
      <eq-val var="curtainStatus" value="0"/>
    </pre>
    <assign-val var="curtainStatus" value="1"/>
  </action>
  <action name="doCloseCurtain">
    <pre>
      <eq-val var="curtainStatus" value="1"/>
      <eq-val var="LivingRoomWindow_windowStatus" value="0"/>
    </pre>
    <assign-val var="curtainStatus" value="0"/>
  </action>
</declare>
```

The XML file is enriched with a header containing the variables considered in pre-conditions and effects. The declaration of the variables must be in accordance with the data types defined in the data model (see Section 3.2). In both the `<ts>` and the `<declare>` part, the action names correspond to the operation names defined in the WSDL of the service (see Figure 1). The `<ts>` part represents the transition system shown in Figure 3. In the `<declare>` part, pre-conditions and effects are expressed at the level of actions-operations, independently of the conversational state the service is currently in. These two kinds of service descriptions cover different needs and have different levels of flexibility.

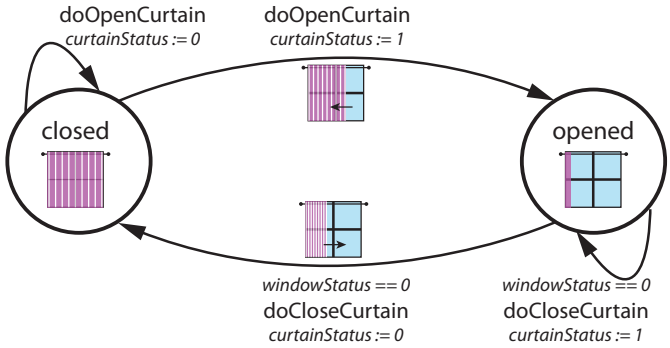

Fig. 3. Curtain service transition system

3.2 Data Model

The data model describes the variable types that are used by the services. They either represent sensed and actuating variables, or inputs and outputs of a service operation. The data model is encoded in XML: the types in the data model are derived by XML Schema native ones and are designed to be further extended. The most common variable types are enumerations on top of the `xs:numeric` type, thus allowing the ordering over the possible values, as inherited from the basic integer type. Having enumerations over variables with finite sets of possible values makes the reasoning tasks of synthesis engines feasible and effective. In the following, we show how the `curtainStatus` type is defined. It can be easily extended to support more values. For instance, if the curtain service supports more operations than just *open* and *close*, an intermediate value (e.g., `semi-closed`) could be added to the enumeration.

```
<xs:schema ...>
  <xs:simpleType name="curtainStatus">
    <xs:annotation>
      <xs:documentation>Specifies the values that a curtain can
          assume (opened/closed)</xs:documentation>
    </xs:annotation>
    <xs:restriction base="sm4base:numeric">
      <xs:enumeration value="0">
        <xs:annotation>
          <xs:documentation>Closed</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
      <xs:enumeration value="1">
        <xs:annotation>
          <xs:documentation>Opened</xs:documentation>
        </xs:annotation>
      </xs:enumeration>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

4. AUTOMATIC COMPOSITION

The SM4All architecture is based on the interplay of two composition engines: the off-line synthesis engine (see Section 4.1), which is based on pre-specified, reusable automaton-based composite services, and the dynamic planner (see Section 4.2), which performs continual planning on-demand, based on AI techniques.

Being computed off-line, routines are specifications that offer several orchestrations for enacting the target: each

Pre-print copy of the manuscript published by International Federation of Automatic Control
identified by DOI: 10.3182/20120905-3-HR-2030.00191

follows a different path, according to the current state of the service ecology and the environment, during the execution. Every legal state of the involved services and every coherent state of the environment is foreseen. Composite services are stored in the Service Semantic Repository, like any other service: indeed, they are reusable. Being made on-line, plans from the dynamic planner are single orchestrations starting from the current context and stopping in a reachable situation where the goal is fulfilled. Plans are not stored anywhere: their life cycle ends as they are executed.

In order to clarify the interplay of the two composition engines, let us assume that a *Relax Scenario* routine was already computed by the Off-line Synthesis Engine. The desired service specifies that the lights in the living room should be switched off, the curtains should be closed, and then the stereo should be turned on. Mike, an inhabitant of the smart house, makes the system launch the routine. Although, the service operation for closing the curtains in the living room fails to execute properly because his sister is opening the window in the meanwhile. The failure is caught by the Orchestration Engine. In turn, it invokes the Dynamic Planner to solve the issue: a declarative goal, equivalent to the final state of the *Relax Scenario* routine, is passed to it. The planner, then, generates a recovery plan, which first closes the window, then closes the curtains, and finally switches on the stereo. Although the computation of this simple plan might be a matter of less than a second, resorting to the dynamic planner may be time-consuming in case of complex compositions, since the Dynamic Planner always starts the computation from scratch. Therefore, the general framework is to have target services corresponding to the user's requests: off-line synthesized compositions realize the routines, and the Dynamic Planner is left as a recovery mechanism, if any failure occurs during the execution.

4.1 Off-line Synthesis

The off-line synthesis is the process of composing orchestrations that enact a given target service through the delegation of the operations to the ecology of available services. It is based on a conversational description of the available services and the target service, both represented as finite transition systems. The solution is based on the reduction of the problem to the synthesis of Linear-time Temporal Logic (LTL) formulae by Model Checking over Game Structures (Piterman et al. [2006]).

The composition returned by the off-line synthesis, called *routine*, can be seen as an imperative program which, given the current state of the target service and the next operation to invoke, specifies which service can realize that action, taking into account the current conditions in the home environment and the conversational states of services. 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, while respecting the constraints imposed by any pre-condition and effect associated with these transitions. Target services are described through the same syntax and semantics of the simple services. The target service is specified and processed at deployment time, i.e., during the installation of the system, hence before run-time.

4.2 Dynamic Planning

The Dynamic Planner (or *planner*, for short) bases upon an AI domain-independent planner, which ultimately models the home domain as a Constraint Satisfaction Problem (CSP). The planner accommodates for a high level language for expressing extended goals (Kaldeli et al. [2011]). The nature of the goal language is declarative, decoupled from the procedural and operational details of the services. The service operations that fulfill the properties prescribed in the goal are synthesized by the planner automatically and at run-time, depending on the current state of the environment and the devices available at the moment. More details about the planner and the techniques it uses can be found in Kaldeli et al. [2011].

The planner takes the following ingredients as input: *(i)* the representation of the home in the form of a planning domain, i.e., the description of the available service operations in terms of preconditions and effects; *(ii)* the description of the current state of the home; *(iii)* a goal prescribing a set of properties to be achieved;

Given the goal and the description of the domain instance, the planner computes a plan, i.e., a partially ordered set of actions which have the potential to satisfy the goal. The planner executes the plan step-by-step, by requesting the services to execute the respective operations and waiting to be informed about its outcome. If the response indicates a failure, then the planner will perform re-planning, i.e., it will come up with an updated plan starting from the new initial state. If the failure indicates that a service is permanently out of order, then the plan will look whether the same goal can be satisfied by an alternative plan, without using the defective appliance.

The encoded goal in the relax scenario looks like this:

```
<achieve-maint>
  <eq-val var="livingRoom_LivingRoomCurtain_curtainStatus" value=
       "0"/>
 <eq-val var="livingRoom_LivingRoomLight_lightStatus" value="0"/>
 <eq-val var="livingRoom_Stereo_stereo" value="1"/>
</achieve-maint>
```

The `achieve-maint` construct prescribes that the propositions which follow should be satisfied in some future state and remain true till the end of the plan. A description of the constructs offered by the goal language, which support the expression of temporal aspects, maintainability properties and others, along with examples that are of relevance in a smart home environment, are presented in Kaldeli et al. [2010].

5. CONCLUSIONS

In this paper, we outlined a layered architecture for realizing a highly dynamic pervasive environment for smart homes, based on ecologies of services. We discussed how the interplay of an off-line composition engine and a dynamic planner is a feasible solution to address scenarios which call for compositions that are able to adapt themeselves to environmental changes and recover from failures, while accommodating for efficiency and reusability. The

Pre-print copy of the manuscript published by International Federation of Automatic Control
identified by DOI: 10.3182/20120905-3-HR-2030.00191

architecture has been fully implemented in an apartment with real devices at the premises of Fondazione Santa Lucia in Rome, and evaluated by end-users (Aiello et al. [2011]). The results prove that the combination of planning and model checking techniques is a feasible solution to address the issues that arise from the dynamic nature of a home environment. We argue that similar concepts and approaches can be adopted also in robotic and agent ecologies.

Directions for future work touch upon different levels of the architecture. At the user layer, the support of a tool that assist users and home designers to express their requests as target services or declarative goals in a graphical way is indispensable. Several extensions are required to the dynamic composition engine, in combination with the other components, to effectively support many users that interact concurrently with the home, and resolve possible contradictions that may arise when their goals and activities interfere.

Acknowledgements. This work was partly supported by the EU Commission through the projects SM4Aall – FP7-224332 and Greener Buildings – INFSO-ICT-258888.

REFERENCES

M. Aiello, F. Aloise, R. Baldoni, F. Cincotti, C. Guger, A. Lazovik, M. Mecella, P. Pucci, J. Rinsma, G. Santucci, and M. Taglieri. Smart homes to improve the quality of life for all. In *Proc. EMBC 2011*.

D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *Proc. ICSOC 2003*.

D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *International Journal of Cooperative Information Systems (IJCIS)*, 14(4):333–376, 2005a.

D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Composition of services with nondeterministic observable behavior. In *Proc. ICSOC 2005*.

D. Berardi, F. Cheikh, G. De Giacomo, and F. Patrizi. Automatic service composition via simulation. *International Journal of Foundations of Computer Science*, 19 (2):429–451, 2008.

J. Blythe and J.L. Ambite, editors. *Proc. ICAPS 2004 Workshop on Planning and Scheduling for Web and Grid Services*, 2004.

T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *Proc. WWW 2003*.

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

J. Cardoso and A. Sheth. Introduction to semantic web services and web process composition. In *Proc. SWSWPC 2004*.

G. De Giacomo and S. Sardiña. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. IJCAI 2007*.

C.E. Gerede, R. Hull, O.H. Ibarra, and J. Su. Automated composition of e-services: lookaheads. In *Proc. ICSOC 2004*.

M. Gritti, M. Broxvall, and A. Saffiotti. Reactive self-configuration of an ecology of robots. In *Proc. ICRA 2007 Workshop on Network Robot Systems*.

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

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

R. Hull. Web services composition: A story of models, automata, and logics. In *Proc. ICWS 2005*.

E. Kaldeli, A. Lazovik, and M. Aiello. Extended goals for composing services. In *Proc. ICAPS 2009*.

E. Kaldeli, E.U. Warriach, J. Bresser, A. Lazovik, and M. Aiello. Interoperation, composition and simulation of services at home. In *Proc. ICSOC 2010*.

E. Kaldeli, A. Lazovik, and M. Aiello. Continual planning with sensing for web service composition. In *Proc. AAAI 2011*.

R. Lundh, L. Karlsson, and A. Saffiotti. Plan-based configuration of an ecology of robots. In *Proc. ICRA 2007*.

S.A. McIlraith and T.C. Son. Adapting golog for composition of semantic web services. In *Proc. KR 2002*.

M. Michalowski, J.L. Ambite, C.A. Knoblock, S. Minton, S. Thakkar, and R. Tuchinda. Retrieving and semantically integrating heterogeneous data from the web. *IEEE Intelligent Systems*, 19(3):72–79, 2004.

A. Muscholl and I. Walukiewicz. A lower bound on web services composition. In *Proc. FOSSACS 2007*.

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

M. Pistore, P. Traverso, and P. Bertoli. Automated composition of web services by planning in asynchronous domains. In *Proc. ICAPS 2005*.

N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Proc. VMCAI 2006*.

S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proc KR 2008*.

D. Wu, B. Parsia, E. Sirin, J.A. Hendler, and D.S. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *Proc. ISWC 2003*.

Pre-print copy of the manuscript published by International Federation of Automatic Control identified by DOI: 10.3182/20120905-3-HR-2030.00191

This document is a pre-print copy of the manuscript
(Caruso et al. 2012)
published by International Federation of Automatic Control.

The final version of the paper is identified by DOI: 10.3182/20120905-3-HR-2030.00191

References

Caruso, Mario, Claudio Di Ciccio, Ettore Iacomussi, Eirini Kaldeli, Alexander Lazovik, and Massimo Mecella (2012). "Service Ecologies for Home/Building Automation". In: *SyRoCo*. Ed. by Ivan Petrovic and Peter Korondi. Vol. 10. Robot Control. IFAC, pp. 467–472. ISBN: 978-3-902823-11-3. DOI: 10.3182/20120905-3-HR-2030.00191. URL: http://www.ifac-papersonline.net/Robot_Control/10th_IFAC_Symposium_on_Robot_Control/index.html.

BibTeX

```
@InProceedings{   Caruso.etal/SYROCO2012:ServiceEcologiesHome,
  author        = {Caruso, Mario and Di Ciccio, Claudio and Iacomussi, Ettore
                   and Kaldeli, Eirini and Lazovik, Alexander and Mecella,
                   Massimo},
  title         = {Service Ecologies for Home/Building Automation},
  booktitle     = {SyRoCo},
  year          = {2012},
  pages         = {467--472},
  publisher     = {IFAC},
  crossref      = {SYROCO2012},
  doi           = {10.3182/20120905-3-HR-2030.00191},
  keywords      = {smart homes, automatic service composition, planning,
                   sensors and actuators}
}
@Proceedings{     SYROCO2012,
  title         = {10th International {IFAC} Symposium on Robot Control,
                   SyRoCo 2012, Dubrovnik, Croatia, September 5-7, 2012},
  year          = {2012},
  editor        = {Petrovic, Ivan and Korondi, Peter},
  publisher     = {International Federation of Automatic Control},
  series        = {Robot Control},
  volume        = {10},
  doi           = {10.3182/20120905-3-HR-2030.00191},
  isbn          = {978-3-902823-11-3},
  url           = {http://www.ifac-papersonline.net/Robot_Control/10th_IFAC_Symposium_on_Robot_Control/index.html}
}
```