Smart Home Planning Programs

Riccardo De Masellis, Claudio Di Ciccio, Massimo Mecella, Fabio Patrizi

Dipartimento di Informatica e Sistemistica
SAPIENZA – Università di Roma, Italy
{demasellis,cdc,mecella,patrizi}@dis.uniroma1.it

ABSTRACT

In pervasive (ubiquitous) computing an increasing amount of devices are embedded and interconnected in the user's environment, e.g., a smart house. The system needs to adapt to the user's varying contexts and goals. The aim is to provide transparent services, reacting to input from the users and to the state of the environment. As user's requirements increase and new devices are inserted, new services need to be dynamically created. We present a technique that allows the user to express planning programs (i.e., procedures allowing to go through different states of the environment) and to have it realized through automatic service composition techniques.
Keywords: embedded services, automatic service composition, planning programs, smart homes

1. Introduction

Embedded devices are specialized computers used in large systems to control equipments, such as automobiles or home appliances. Such a pervasiveness is particularly evident in *ubiquitous* realities, i.e., scenarios where embedded systems collaborate with human users by providing information and reacting to service requests. Examples of such scenarios are digital libraries, eTourism, automotive, next generation buildings, eHealth and domotics.

Human-service collaboration poses new challenges to current technologies. Indeed, devices such as sensors, appliances and actuators, offering *services*, are no longer combined *statically*, as in classical networks, (e.g., for environmental monitoring and management or surveillance), but need to be *dynamic*, as the overall distributed system is required to adapt to different contexts, user habits, wishes, etc. Moreover, the number of resources required to really immerse the user in the system is large, at least an order of magnitude more than in the current systems. For instance, current best-in-class smart houses count up to one hundred resources, while next generation's are expected to involve several hundreds.

This paper comes from the experience of SM4ALL: a project aimed at studying and developing a platform for collaboration between human- and software-based services in immersive, person-centric, environments. [1] In particular, this is carried on in the scenario of private homes and buildings, in presence of users with different abilities and needs (e.g., young, elderly or disabled people), where three major goals are pursued: *(i) person-centric awareness*, i.e., humans are at the heart of ubiquitous environments; *(ii) global distribution and service-centric functionalities*, i.e., middleware and infrastructure services (e.g., storage and retrieval of service descriptions, communication) should be managed in a widely distributed manner to guarantee dynamism, scalability and dependability; *(iii) openness and maximum-reuse*, i.e., a generic embedded middleware capturing all common aspects of pervasive scenarios is needed to obtain scalability, reusability and extensibility.

[1] http://www.sm4all-project.eu/

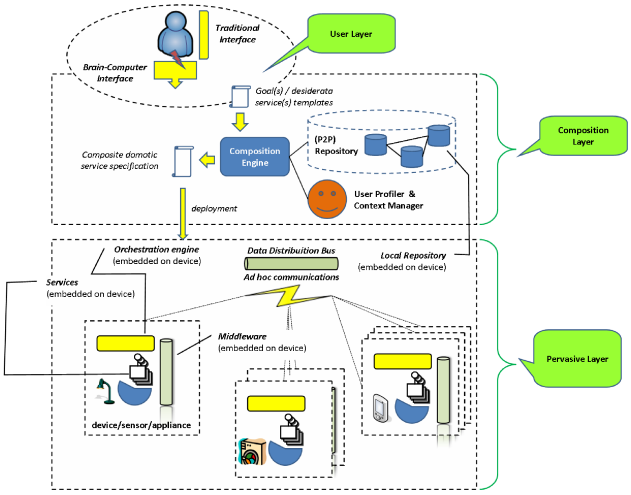


Fig. 1: The SM4ALL architecture

Figure 1 shows the SM4ALL architecture. Devices (i.e., actuators and sensors) available in the house constitute the basis of the system: actuators drive the home *hardware*, including doors, lights, media or security appliances; while sensors measure physical quantities, and can range from simple thermometers to self-calibrating, satellite-carried, radiometers. Sensors and actuators can closely interact: a device opening a window blinds can change the luminosity value sensed by a sensor.

Both sensors and actuators make their functionalities available according to the service-oriented paradigm [2]. In order to be dynamically configurable and composable, embedded services need to expose semantically rich service descriptions, comprising *(i)* interface specifications, *(ii)* specifications of the externally visible behaviors, *(iii)* offered QoS and *(iv)* security requirements. Moreover, human actors in the environment can be abstracted as services, and actually "wrapped" by a semantic description (e.g., a nurse offering medical services). This allows them to be involved service compositions and to collaborate with devices, to reach certain goals.

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by DOI: 10.1109/ICSSSM.2010.5530212

Embedded devices are inter-connected in a wireless ad-hoc fashion, and a specific middleware needs to be available to enable service collaboration. The collaboration of services is carried out by the *orchestration engine*. When the user expresses a high-level goal (e.g., in form of a state of the house she would like to have realized), the *composition engine* automatically synthesizes the right orchestration of services able to satisfy the goals. Users can interact with the house through different kinds of *interfaces*, either centralized (e.g., in a home control station) or distributed, and embedded in specific interface devices. Brain Computer Interfaces (BCIs) allow also people with disabilities to interact with the system. Using such interfaces, users issue specific goals to the system, which is, in turn, expected to react and satisfy the request.

In this context, the present paper considers two prominent challenges, namely: *(i)* the issue of describing the environment and the devices that users, respectively, live in and interact with; and *(ii)* the problem of automatically reacting to user requests, by *orchestrating* the devices so as to fulfill user desires.

In the last years, several works appeared which address the above problems from different viewpoints. Largely, the basic interoperability problems for service composition have been resolved (e.g., standards such as WS-BPEL and WS-CDL exist and are widely supported in order to compose services), and designing programs, known as *orchestrators*, able to execute compositions by coordinating available services according to the description they expose, has become the *bread and butter* of the service programmer. The availability of abstract descriptions of services has been instrumental to devising automatic techniques for synthesizing service compositions and orchestrators.

In the literature, we find works focused on data-oriented services, which bind service composition to the work on data integration [9]. Others have looked at process-oriented services, in which operations executed by services affect, in general, the environment that such services act in. Among these approaches, several consider stateless, or *atomic* services, where the operations offered clients do not depend on the past history, as services do not retain any information about past interactions. Much of this work relies on the literature on Planning in AI [16].

On the other hand, we have approaches that consider *stateful* services, which impose constraints on the possible sequences of interactions (a.k.a., conversations) that a client can engage with the service. Stateful service composition raises additional challenges, as the composite service should be correct w.r.t. the possible conversations allowed by the component ones. This is exactly the approach of this work: we focus on composition of process-oriented stateful services, in particular extending the framework for service composition adopted in [4], often referred to as *Roman Model*, where services are abstracted as transition systems and the composition objective is to obtain a *composite* service that preserves a desired interaction, expressed as a (virtual) target service. Here, we consider a model extension where *planning programs* [6] are used, instead of target services, to specify user goals. Such programs can be thought of as *routines* built from high-level goals, which allow users to specify some desired tasks to be carried out, by possibly requiring user decisions, at particular points of choice.

The rest of the paper is organized as follows. Sections 2 and 3 introduce, respectively, the concepts of *smart home environment* and *planning programs*. Section 4 presents the notion of *smart services*, provides a formal definition of the problem we deal with, and shows the main complexity results. Section 5 presents a complete example. Section 6 outlines a possible way of concretely realizing the idea, whereas Section 7 concludes the paper and highlights future developments.

2. Smart Home Environment

A *Smart Home* is a domotic environment, equipped with *smart* devices, distributed all over the house as *embedded systems*. Such devices can be divided into *actuators* (e.g., shutter controllers, refrigerators...), able to actually modify the home status, and *sensors* (e.g., barometers, thermometers...), that can access the current state of the home and measure some properties. Hence, the former class contains those devices that can have a side-effect on the home, whereas the latter contains devices able to monitor the environment status. Data expressed by sensors may be raw, and need to be collected, filtered, refined, analyzed and exposed so that a *reasoning core*, as an *environment* controller, can actually use them to build an abstract knowledge base of the world it acts in. Actuators have different levels of capability (from simple, e.g., a fan, to complex, e.g., a DVD recorder) and are typically dedicated to restricted specific tasks. Hence the *reasoning core*, as a domotic management *system*, has to properly instruct such a vast community, in order to possibly achieve an overall result able to cover more complex and even structured tasks.

It is important to separately consider the *environment* from the *system*: indeed, the environment can evolve not only because of system interventions (e.g., the shutter might be lifted up because of a human actor pulling down the mechanical cord, even without involving the shutter controller activation), nevertheless the *Smart Home* must be aware of such a change in the status.

Formalization We formalize the notion of home *environment* as a (possibly nondeterministic) *dynamic domain* $\mathcal{D}$, which provides a symbolic abstraction of the world that the user acts in (as usually done in planning and reasoning about actions [8, 13]).

Definition 1 (Dynamic Domain). A *dynamic domain*, or *environment*, is a tuple $\mathcal{D} = \langle P, A, S_0, \rho \rangle$, where:

- $P = \{p_1, \ldots, p_n\}$ is a finite set of *domain propositions*. A *state* is a subset of 2^P;
- $A = \{a_1, \ldots, a_r\}$ is the finite set of *domain actions*;
- $S_0 \in 2^P$ is the *initial state*;
- $\rho \subseteq 2^P \times A \times 2^P$ is the *transition relation*. We freely interchange notations $\langle S, a, S' \rangle \in \rho$ and $S \xrightarrow{a} S'$ in $\mathcal{D}$. ■

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by DOI: 10.1109/ICSSSM.2010.5530212

Intuitively, a dynamic domain models the evolution of a dynamic system (i.e., the home environment), whose state is described by a set of boolean propositions. For instance, the state of a room can be defined by the light being on or off (proposition on) and the door being open or close ($open$). We adopt the convention that if a proposition is in the current state of $\mathcal{D}$, then it evaluates to $\top$ (true), otherwise it is $\bot$ (false). So, the state of the room when the light is off and the door is open can be represented by state $\{open\}$, while $\emptyset$ captures the situation where the light is off and the door is closed. As for state changes, relation ρ captures the (possibly nondeterministic) state transitions an action yields. Finally S_0 is the single domain initial state. A propositional formula φ over P *holds* in a $\mathcal{D}$ state $S \in 2^P$, if φ evaluates to $\top$ when all of its propositions are replaced by $\top$ iff they occur in S.

3. Planning Programs

When in use, a *Smart Home* can assist the user to achieve her everyday life tasks at home: she may describe the desired goals to be fulfilled through a given formalism. Expressions of such a formalism are evaluated over the *environment* (i.e., over the *status variables* that define the environment), and it is up to the *Smart Home system* to coherently produce strategies that achieve the desired condition.

Goals may have several forms, and different goals require, in general, different solution approaches. A very natural form of goal is *reachability*: a condition the user desires to be achieved, e.g., *window W is open* or *room R is warm*. Also, we have *maintenance* goals, when a user desires a condition to be maintained from a given point on, e.g., *maintain room R warm*. Notice that in the former case nothing is said about what should happen after the condition has been fulfilled. Of course, above forms can be combined, obtaining goals like: achieve *room A is warm* while maintaining *window W is open*.

Here, we refer to a more general form of goals, referred to as *planning programs* [6], capturing high-level routines, where the user can request, in a step-by-step fashion, one among some *reachability* goals, plus a *maintenance* property ψ, constraining the system to compute only solutions satisfying ψ. Planning programs capture *routine-task*, possibly allowing the user to make some decisions.

Formalization A planning program $\mathcal{T}$ is a transition system, whose states represent *choice points*, and whose transitions specify pairs of *maintenance* and *achievement* goals that the user can request at each step.

Definition 2 (Planning Program). A *planning program* for a dynamic domain $\mathcal{D}$ is a tuple $\mathcal{T} = \langle T, \mathcal{G}, t_0, \delta \rangle$, where:

- $T = \{t_0, \ldots, t_q\}$ is the finite set of *program states*;
- $\mathcal{G}$ is a finite set of goals of the form *achieve* ϕ *while maintaining* ψ, denoted by pairs $g = \langle \psi, \phi \rangle$, where ψ and ϕ are propositional formulae over P;
- $t_0 \in T$ is the *program initial state*;
- $\delta \subseteq T \times \mathcal{G} \times T$ is the *transition relation*. We freely interchange notations $\langle t, g, t' \rangle \in \delta$ and $t \xrightarrow{g} t'$ in $\mathcal{T}$. ■

A planning program $\mathcal{T}$ combines achievement and maintenance goals so that they can be requested (and hence fulfilled) according to a specific temporal arrangement. An instance of such programs is depicted in Figure 5.

In order to understand planning programs' semantics, i.e., how they are *realized*, the notion of *plan over a domain* $\mathcal{D}$ is required. A plan π over a domain $\mathcal{D}$ is essentially a program (recursively) built by using instructions of the form **if** φ **then do** a, where φ is a propositional formula over P and $a \in A$ is an action on $\mathcal{D}$. When plans are executed on a domain, they result in a domain evolution, namely one state is traversed after each action is executed. We say that a plan π *achieves* a goal ϕ if it guarantees that a $\mathcal{D}$ state where ϕ holds is reached after the execution of last π action. Similarly, we say that π *maintains* a property ψ if it guarantees ψ to hold in all the states $\mathcal{D}$ traverses during π execution. Then, a plan π *achieves a goal* ϕ *while maintaining a property* ψ if, when executed, a state where ϕ holds is eventually reached by $\mathcal{D}$ and, at the same time, ψ holds in all the states $\mathcal{D}$ traversed. Formal details about plans, their executions, and goal achievement/maintenance can be found in [6].

Now, we can describe planning program realization. When a planning program is realized, a typical execution is as follows: at any point in time, the planning program is in a state t and the environment in a state $S \in 2^P$ (initially, t_0 and S_0, respectively); the user requests a transition $t \xrightarrow{\langle \psi, \phi \rangle} t'$ in $\mathcal{T}$; then, a plan π is executed from S which eventually leads the environment to a state that satisfies achievement goal ϕ, while only traversing states satisfying maintenance goal ψ; upon plan completion, the planning program moves to t' and the user can request a new transition among those outgoing from t', i.e., $t' \xrightarrow{\langle \psi', \phi' \rangle} t''$ in $\mathcal{T}$, and so on. Notice that, at any point in time, all possible choices available in the planning program must be guaranteed by the system—every legal request needs to be satisfied.

We provide here an intuitive notion of *planning program realization*, referring the reader to [6] for formal details. Given a planning program $\mathcal{T}$ for $\mathcal{D}$, a $\mathcal{T}$*-realization* can be thought of as a function f that, given the set of states traversed by $\mathcal{D}$ up to the current point (i.e., a $\mathcal{D}$*-history*), and a current transition $t \xrightarrow{\langle \phi, \psi \rangle} t'$ requested by the user, returns a plan π that: *(i)* achieves ϕ while maintaining ψ, when executed from current $\mathcal{D}$ state; and *(ii)* leads $\mathcal{D}$ to a new state such that all the $\mathcal{T}$ transitions outgoing from t', can be served by a new plan π', returned by f itself, based on the history traversed and the new transition.

4. Smart Home Services

So far, we implicitly assumed that all actions are made available by $\mathcal{D}$, that is, any of the actions *executable* in the current state of $\mathcal{D}$ can be potentially performed. However, this assumption might not be fulfilled. In the smart home context, actions are provided by several devices, each having its own internal logic. Thus, action executability, besides depending on $\mathcal{D}$'s current state, also depends device

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by DOI: 10.1109/ICSSSM.2010.5530212

logics. For instance, a device might require to be switched on before exporting its actions, a fact that is not captured by $\mathcal{D}$ alone. Of course, when an action is allowed by some device, in order to be actually executed, it needs to be executable in $\mathcal{D}$, as well. In this section, we introduce the formal notions needed to capture such devices.

The devices installed in a smart home can be either *sensors* or *actuators*. As we said, in general, they have their internal logic, actually implemented using particular technologies, and act as *action providers*, in the sense that offer the possibility of executing actions –that can affect the state of the home. Here, we abstract from low-level implementation details, and, adopting a *service-oriented* perspective (see, e.g., [2]), look at them simply as *(embedded) services* [1, 3], focussing on their "conversational" capabilities. This allows the *Smart Home* system to effectively *reason* about device behaviors.

Considering the *Smart Home* as a *service community* enables known service composition techniques (see, e.g., [4, 7, 5, 14, 10]) to be used for automatically fulfilling user requests (cf., 2).

In practice, several standards are adopted to realize the service-oriented approach. A popular implementation is *SunSPOT*[2], i.e., Java-programmable hardware interfaces built to be integrated as embedded devices in domestic environments, and to communicate through radio signals with a central processing unit: through this technology, devices may act as services, distributed all over the house. We remark here that such an abstraction overcomes the difference between sensors and actuators, as to any device. At the extreme, even human actors could be modeled as services.

Formalization We abstract devices as *finite-state services*. For instance, an air conditioner is a service required to be turned on before cooling the air. That is, action $coolAir$ can be executed only if the service state is on, but is not allowed in state off. Observe there can be actions, such as $switchOn$ not affecting the domain's state.

Definition 3. A *service* over a dynamic domain $\mathcal{D}$ is a tuple $\mathcal{B} = \langle B, O, b_0, \varrho \rangle$, where: *(i)* B is the finite set of service states; *(ii)* O is the finite set of service actions s.t. $O \cap A \neq \emptyset$; *(iii)* $b_0 \in B$ is the service initial state; *(iv)* $\varrho \subseteq B \times O \times B$ is the service transition relation. We freely interchange notations $\langle b, a, b' \rangle \in \varrho$ and $b \xrightarrow{a} b'$ in $\mathcal{B}$. ■

The idea, as depicted in Figures 3 and 4, is that, in each state, a service offers a set of possible actions from O. The system can interact with the environment $\mathcal{D}$ *only* by means of available devices, i.e., services. When a service is instructed to perform an action, it is executed and both the device in question and the domain evolve, *synchronously* and possibly *nondeterministically*, to their successor states, according to their respective transition relations. So, for an action to be carried out, it needs to be both compatible with the domain and (currently) available in some service.

[2]Sun Small Programmable Object Technology

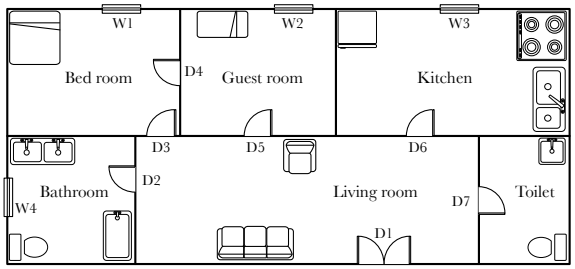
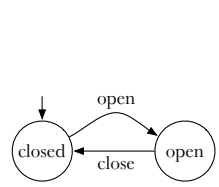
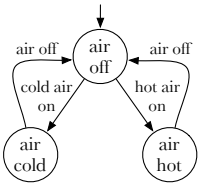


Fig. 2: Smart home plan.



(a) Door and windows behavior.



(b) Toilet device manager and central heating system beahavior.

Fig. 3: Devices behavior.

4.1. The Problem

Next, we combine the notions introduced so far to define the problem of interest. We model the smart home as a tuple $\mathcal{S} = \langle \mathcal{D}, \mathcal{B}_1, \ldots, \mathcal{B}_n \rangle$, where $\mathcal{D}$ is a dynamic domain, abstracting the smart home environment, and $\mathcal{B}_1, \ldots, \mathcal{B}_n$ are the set of services over $\mathcal{D}$, corresponding to the smart home devices that the environment accommodates and which allow actions to be actually executed. Assuming a user issues a request to the system, in the form of a planning program $\mathcal{T}$, the problem of concern is: *can $\mathcal{T}$ be realized in $\mathcal{S}$, through a realization that delegates actions to available services?*

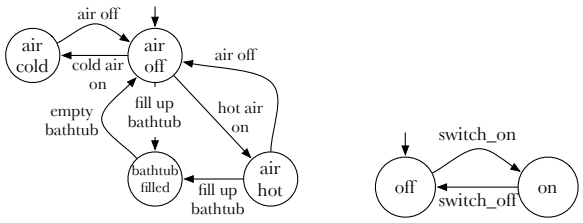
Such a question can be automatically answered and, in detail, the problem is proven EXPTIME-complete [6]. Section 6 provides some details about the solution approach actually adopted.

5. Case Study

In the following, we provide a running example showing some details of our approach at work. Figure 2 shows the smart home where Bob and Alice live. Bob needs a wheelchair to move around. The home is provided with some devices, that help Bob and Alice in their everyday life. As already said, devices are abstracted as transition systems. Precisely:

- all doors, $D1, \ldots D7$ and windows $W1, \ldots, W4$ can be open or closed by an actuator, whose abstraction is depicted in fig. 3(a);
- the house is provided with a central heating system for all the rooms (fig. 3(b));
- the toilet is provided with a device manager that controls its air temperature (fig. 3(b));
- the bathroom is controlled by a device manager able to change the air temperature and the water level of the bathtub (fig. 4(a)) as to avoiding the bathtub to be

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by DOI: 10.1109/ICSSSM.2010.5530212



(a) Bathroom device manager behavior.

(b) Kitchen ventilation system behavior.

Fig. 4: Devices behavior.

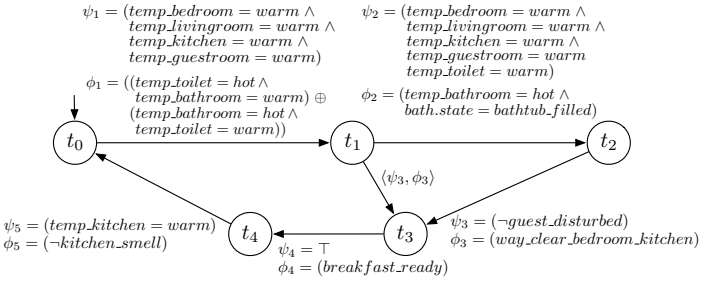


Fig. 5: Morning planning program.

filled if the cold air is switched on, in order to preserve user's health;

- the kitchen is provided with a ventilation system (fig. 4(b)).

All this devices can be remotely controlled by a user interface, e.g., a touchscreen. Even if some devices behave in the same way – for instance the toilet device manager and the central heating system. It is important to notice that their actions can produce different effects: a device's transition system basically describes *which* actions can be performed from the service current state, but *how* actions affect the world is described in the *Dynamic domain*. Let us illustrate our home domain. For the sake of readability, we consider the set P of the dynamic domain $\mathcal{D}$ as a set of *state variables* instead of boolean proposition, recalling that we can easily ground the domain of a variable, i.e., represent it using boolean propositional variables only, provided that, as in our case, the domain is finite. Bob and Alice have a guest, Trudy, who lives in the guest room. Trudy is sleepyhead, so she would not be disturbed during early morning. To the contrary, Alice and Bob are in the habit of waking up at eight o'clock, washing themselves and taking their breakfast. The set P of (interesting) variables that describe the house includes:

- a set of state variables $temp\_room = \{cold, warm, hot\}$, one for each room;
- the boolean proposition $guest\_disturbed$ that is true if Trudy has been disturbed for some reason;
- the boolean proposition $kitchen\_smell$ that is true if there is smell in the kitchen, typically after cooking;
- the boolean self-explanatory proposition $breakfast\_ready$.

Moreover, we can check for devices' status, i.e. every device has a state variable $device\_status = \{set\ of\ states\}$. In this example we model Bob as a service that can perform only the action $cook$. The result of actions, is not only a change in the service status, but also in the domain's:

- opening a windows decreases the temperature of the respective room to $cold$;
- opening or closing guest room's doors ($D1$ and $D5$) or windows cause the guest to be disturbed;
- turning on the central heating system cause all rooms' temperature to become to hot;
- turning on the hot (resp. cold) air in the bath or in the toilet causes the bath's or toilet's temperature to rise up (lower down) to hot ($cold$);
- cooking causes the breakfast to be ready and the kitchen to smell;
- switching on the kitchen's ventilation causes the smell to go out.

Alice and Bob would not to change their morning's habits, but they also would not to disturb Alice. A *planning program* for the morning can be formalized as in figure 5. In the initial program state t_0, the home is in a (starting) state s_0 where the temperature of all rooms is $warm$, Trudy is blissfully sleeping in the guest room ($\neg guest\_disturbed$), Alice and Bob are supposed to be in their bed room, there is not smell in the kitchen ($\neg kitchen\_smell$) and all devices are in their initial state, so all doors and windows are closed, the ventilation system is switched off, the air in the toilet and the central heating system are turned off, and finally the air in the bathroom is turned off and the bathtub is empty. The two outgoing edges from t_1 represent our incomplete knowledge about the users' will: Bob (or Alice) can either decide if he want to go to the bathroom $t_0 \xrightarrow{\langle \psi_1, \phi_1 \rangle} t_2$ and then in the kitchen $t_2 \xrightarrow{\langle \psi_3, \phi_3 \rangle} t_3$, or go to the bathroom $t_0 \xrightarrow{\langle \psi_1, \phi_1 \rangle} t_1$ and have a bath $t_1 \xrightarrow{\langle \psi_2, \phi_2 \rangle} t_2$. We thus don't know what will be the next user's choice, and this situation force us to take into account every possible evolutions. When Bob goes to the bathroom, he wants hot temperature within one among the bathroom or the toilet. This is expressed through formula ϕ_1, where the symbol $\oplus$ stands for the logical XOR. Contemporaneously he would maintain a $warm$ temperature within all other room, as expressed by the ψ_1 formula. If Bob will resolve also to have a bath, the house has to reach a state where the bathroom is hot and the bathtub is filled (ϕ_2), while maintaining all other room $warm$ (ψ_2). This two maintenance formulae force the system to choose the bathroom or toilet device manager in order to rise the temperature, otherwise, i.e., switching on the central heating system, will falsify them. As aforementioned, planning a sequence of action for reaching ϕ_1 is not trivial: we actually have to take into account that the next request would be to reach ϕ_2 maintaining ψ_2. This means, roughly speaking, that the sequence of actions needed to reach ϕ_1 will never switch on the toilet hot air, because this will avoid to satisfy ψ_2 that could be asked by the user as second request. In fact, if the plan organizes to switch on the toilet hot air, in t_1 the proposition $temp\_toilet$ will be hot, which is explicitly forbidden by ψ_2. The next goal $\phi_3 = way\_clear\_bedroom\_kitchen$

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by DOI: 10.1109/ICSSSM.2010.5530212

is a shortcut to denote that, since Bob's will is to get to the kitchen, the doors in the path from his bedroom to the kitchen have to be open: $way\_clear\_bedroom\_kitchen = (D4.state = open \wedge D5.state = open \wedge D6.state = open) \vee (D3.state = open \wedge D6.state = open)$. Moreover, recall that Bob, while going to the kitchen, does not want to disturb Trudy, e.g. by passing through the guest room (ψ_3). Then, the plan is supposed not to open $D4$ nor $D5$. Afterwards, Bob presumably cooks some eggs with bacon (ϕ_4) and then wants to ventilate the kitchen (ϕ_5), while maintaining the temperature constant (ψ_5), so he expects that the system will switch on the ventilation system instead of opening the window.

6. Implementation

In this Section, we sketch out the solution approach adopted to compute a realization of a planning problem, given a smart home environment and a set of available devices. Full details are available in [6].

The solution approach is based on reducing the problem to the *synthesis* of a Linear-time Temporal Logic (LTL) formulae (see [12]) by *Model Checking* over *Game Structures*. LTL is a well-known logic used to specify temporal properties of programs (see, e.g., [15]), whose formulas are built from a set P of atomic propositions and are closed under boolean and temporal operators, i.e., $\bigcirc$ (*next*), $\Diamond$ (*eventually*), $\Box$ (*always*), $\mathcal{U}$ (*until*).

Our problem is encoded as a *Game Structure*, that is a mathematical representation of a game played by two opponents, namely *Environment* and *System*. Both players control some game variables in P. The *Environment* controls, i.e., assigns values to, the set $\mathcal{X} \in P$, while the *System* controls $\mathcal{Y} = P \setminus \mathcal{X}$. Morover, an LTL $goal$ formula over P abstracts the game's purpose: the aim of the *Environment* is falsifying the $goal$, whereas the objective of the *System* is verifying it. At each turn, the *Environment* and the *System* choose a move according to their respective transition relations, consisting in the assignment of truth values to their variables. The *System* wins if the game is infinite and the LTL formula $goal$ (over P) holds, otherwise the game is winning for the *Environment*. In other words the problem can be stated as: *can the system always control the values of $\mathcal{Y}$ so that for all possible values the environment can assign to $\mathcal{X}$, the LTL goal is satisfied?* We are looking for a *strategy* (if any), that, by acting only on $\mathcal{Y}$, makes the formula $goal$ true, regardless of the environment choices. The *synthesis* task consists in computing such a function, which turns out to be an actual realization of the program. As for computational considerations, even though the synthesis is 2EXPTIME-complete for arbitrary LTL specifications [12], a well-behaved class, namely GR(1) [11], here suffices, for which the synthesis task is proven to be EXPTIME-complete.

7. Conclusions

In this paper we have presented the concept of planning programs as a technical solution for modeling smart houses and being able to perform automatic service composition in such scenarios. The implementation of a prototype realizing the techniques presented in Section 6 is currently ongoing, and is expected to be ready by Summer 2011; in that occasion, the case study will be effectively demonstrated by running the prototype in a real smart house, that in the context of the SM4ALL project is currently under deployment at some partner's premises.

Acknowledgements. This work has been partially funded by the IST Programme of the Commission of the European Communities as project number FP7-224332 "SM4All". The authors would also like to acknowledge Giuseppe De Giacomo for his important suggestions and technical support.

REFERENCES

[1] M. Aiello and S. Dustdar. Are our homes ready for services? a domotic infrastructure based on the web service stack. *Pervasive and Mobile Computing*, 4(4):506–525, 2008.

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

[3] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of e-Services that Export their Behavior. In *Proc. of ICSOC 2003*, pages 43–58, 2003.

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

[5] 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.

[6] G. De Giacomo, F. Patrizi, and S. Sardiña. Agent Programming via Planning Programs. In *Proc. of AAMAS'10*, 2010. To appear.

[7] G. De Giacomo and S. Sardiña. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI 2007*, pages 1866–1871, 2007.

[8] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[9] 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.

[10] F. Patrizi. *Simulation-based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA – Università di Roma, 2009.

[11] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, pages 364–380, 2006.

[12] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proc. of POPL*, pages 179–190, 1989.

[13] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, September 2001.

[14] S. Sardiña, G. De Giacomo, and F. Patrizi. Behavior Composition in the Presence of Failure. In *Proc. of KR'08*, 2008.

[15] M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, volume 1043 of *LNCS*, pages 238–266, 1996.

[16] D. Wu, B. Parsia, E. Sirin, J. A. Hendler, and D. S. Nau. Automating DAML-S Web Services Composition Using SHOP2. In *Proceedings of The Semantic Web - ISWC 2003, Second International Semantic Web Conference*, 2003.

Pre-print copy of the manuscript published by IEEE (available at ieeexplore.ieee.org)
identified by DOI: 10.1109/ICSSSM.2010.5530212

This document is a pre-print copy of the manuscript
(De Masellis et al. 2010)
published by IEEE (available at ieeexplore.ieee.org).

The final version of the paper is identified by DOI: 10.1109/ICSSSM.2010.5530212

References

De Masellis, Riccardo, Claudio Di Ciccio, Massimo Mecella, and Fabio Patrizi (2010). "Smart Home Planning Programs". In: *ICSSSM*. Ed. by Jian Chen. IEEE, pp. 377–382. DOI: 10.1109/ICSSSM.2010.5530212.

BibTeX

```
@InProceedings{   DeMasellis.etal/ICSSSM2010:SmartHomePlanning,
  author        = {De Masellis, Riccardo and Di Ciccio, Claudio and Mecella,
                   Massimo and Patrizi, Fabio},
  title         = {Smart Home Planning Programs},
  booktitle     = {ICSSSM},
  year          = {2010},
  pages         = {377-382},
  publisher     = {IEEE},
  crossref      = {ICSSSM2010},
  doi           = {10.1109/ICSSSM.2010.5530212},
  keywords      = {embedded services, automatic service composition, planning
                   programs, smart homes}
}
@Proceedings{     ICSSSM2010,
  title         = {7th International Conference on Service Systems and
                   Service Management, ICSSSM 2010, Tokyo, Japan, June 28-30,
                   2010},
  year          = {2010},
  editor        = {Chen, Jian},
  publisher     = {IEEE}
}
```