Designing a goal-oriented smart-home environment

Javier Palanca[a], Elena del Val[a], Ana Garcia-Fornes[a], Holger Billhardt[c], Juan Manuel Corchado[b], Vicente Julián[a]

[a]*Departamento de Sistemas Informáticos y Computación Universitat Politècnica de València, Spain*
[b]*Department of Computer Science, University of Salamanca, Spain*
[c]*CETINIA, Universidad Rey Juan Carlos, Spain*

Abstract

Nowadays, systems are growing in power and in access to more resources and services. This situation makes it necessary to provide user-centered systems that act as intelligent assistants. These systems should be able to interact in a natural way with human users and the environment and also be able to take into account user goals and environment information and changes. In this paper, we present an architecture for the design and development of a goal-oriented, self-adaptive, smart-home environment. With this architecture, users are able to interact with the system by expressing their goals which are translated into a set of agent actions in a way that is transparent to the user. This is especially appropriate for environments where ambient intelligence and automatic control are integrated for the user's welfare. In order to validate this proposal, we designed a prototype based on the proposed architecture for smart-home scenarios. We also performed a set of experiments that shows how the proposed architecture for human-agent interaction increases the number and quality of user goals achieved.

Keywords: Multi-Agent Systems, Smart-home environments, Adaptive systems, Goal-oriented systems, Service-oriented systems

1. Introduction

Collaboration is an important factor in achieving success in any type of work or project. In general, any task with hints of complexity requires the collaboration of more than one individual. Technology should be capable of supporting these collaboration processes through the formation and management of groups or coalitions of entities that which can be humans or software agents. These groups or coalitions can arise in a spontaneous or planned manner in order to maximize the expected utility or profit of the individuals. Agent technology enables the development of applications that support the formation and management of such organizations dynamically. Applications of this kind are possible through the use of a goal-oriented architecture for human-agent societies, where the traditional notion of application disappears. Rather than developing software applications that accomplish computational tasks for specific purposes, the goal-oriented approach in these human-agent societies is based on the immersion of users in self-adaptive environments that facilitate the achievement of their goals in an automated way.

This new way of envisioning applications requires new methods and techniques that support the integration of humans and software agents, considering agents as service/resource providers. Taking this into consideration, one of the main problems is how to show all the available services and resources to users in an appropriate way. As the size of available services increases, it is more difficult for users to determine which service or set of services is the most suitable for achieving their goals. Moreover, users usually know what they want to do, but they do not know how to do it. Since users know their goals, it will be easier to help them with a goal-oriented architecture. Considering that agents are intelligent entities that have social capabilities, they fit properly in a goal-oriented architecture where services are provided and consumed in order to achieve their goals. In this paper, we present a goal-oriented architecture that is based on the SOC (Service-Oriented Computing) concepts [1] and their use in the design and implementation of a goal-oriented smart-home environment. The purpose of the architecture is to find solutions to reach user goals through the composition and execution of services offered by agents. In this architecture, agents provide services in a ubiquitous environment where users (human or non-human) express their goals and the system determines the set of actions that fulfils the goal in way that is a completely transparent to the user (i.e., the user does not see the translation process from goals to actions).

The goal-oriented architecture is designed to work in environments with Service-Oriented Architectures, Grid architectures, or business process architectures. In these environments, there are many interconnected nodes where providers offer a multitude of services. This work focuses on a goal-oriented smart-home environment, where a user declares a goal with certain constraints and the smart-home system tries to carry out a set of actions in order to deal with the user request expressed as a goal. The smart environment considers services offered by different nodes in a pervasive or ubiquitous way [2, 3] as well as services provided by external entities.

On the basis of the contents of the preceding paragraphs, in this work we propose a goal-oriented architecture for smart-home environments. The main contributions of this proposal are: (i) a formal model for an architecture that not only has a fixed set of plans to deal with the self-adaptation of the system, but one that is also able to create new plans and refine or repair running plans; (ii) the architecture includes case-based reasoning techniques to learn from previous plans used in specific situations and reuse them according to new requirements; (iii) semantic information in the definition of services and user goals in order to facilitate the automatic service discovery and composition of complex services; (iv) a negotiation process is included to check the availability of service providers and to establish temporal commitments to ensure the execution of the services within a time frame.

The rest of the paper is structured as follows: section 2 describes the related work; section 3 introduces the approach of Distributed Goal-Oriented Computing architecture to develop human-agent systems; section 4 presents the design of a goal-oriented smart-home environment based on the previous architecture. Finally, section 5 presents the conclusions of the paper.

2. Related Work

One of the areas where the use of human-agent technologies play a key role is smart-home environments. The area of smart-homes can be considered as a branch of ubiquitous computing that integrates ambient intelligence and automatic control into living spaces for comfort, healthcare, safety, security, and energy conservation [4]. Smart-home environments have been researched for nearly two decades [5]. The research in the area of smart-home environments has since tackled different technical issues such as heterogeneity in devices and technologies, context awareness, and security in order to facilitate the implementation of intelligent environments.

Although several technical challenges have been achieved, there are still open issues. The increase in the number of devices and the proliferation of services that are locally or remotely available (i.e., services in local smart-home environments and services in the cloud) and the dynamism of smart-home environments (i.e., appearance of new entities, temporal unavailability of components, changes in the environment conditions or user preferences) make it necessary to include mechanisms that facilitate reconfiguration in smart-home environments. Self-adaptation enables a system to reason and to adapt itself in order to achieve user goals when uncertainties and changes in the environment appear. There are previous works in the context of smart-home environments that try to deal with user requirements using pre-defined plans established at design time [6, 7]. The use of pre-defined plans makes the reconfiguration of the system difficult when unforeseen events occur. A better approach to this problem is to dynamically generate new plans or reuse and adapt plans that were previously used in similar contexts to deal with unforeseen events, as we propose in this work.

Other works based on Multi-Agent Systems (MAS) have been proposed to deal with self-configuration in smart-home environments to deal with user goals. MAS are considered to be a suitable tool for the study of complex adaptive systems, especially for those that are distributed and dynamic [8, 9]. A smart-home environment can be viewed as a MAS where software agents and users interact.

For instance, Iftikhar et al. [10] propose a formal approach for self-adaptation. This approach is based on a feedback loop that consists of four adaptation components: monitor, analyze, plan, and execute. The main contribution of this approach is that it ensures that the goals (which were verified offline) are guaranteed at runtime. It also supports adaptation to changing goals. A goal-driven approach based on agents and semantics for automatic service discovery and control is presented in [11]. The authors describe a layered architecture (device, connectivity, service, and semantic agents layer). The semantic agents layer contains an agent for each of the goals of the smart-home environment (i.e., energy, comfort, safety, and security) and a coordinator agent that solves conflicts when agents have competing goals. Although it is an interesting proposal, details about how the coordination and the self-adaptation of the system is carried out by the agents are not provided. Ayala et al. [12] propose an agent-based Ambient Assisted Living system that incorporates self-configuring tasks. In this approach, the self-configuring process is inside agents. There is a control loop that starts analyzing the context to check if something has changed in the system. Then, the agent determines if the current situation requires reconfiguring the system according to a plan. These plans are stored in a plan library and can be accessed by the goal. After this, the scheduler executes the set of actions of the plan. The main drawback in this approach is that the plan library seems to be a set of predefined plans at design time; therefore, if new services or unforeseen situations appear in the system they would not be considered in the plan library.

Kucher and Weyns propose a self-adaptive software system to support elderly care [13]. The architecture proposed consists of four modules: autoconfigurator, context-adaptor, sensor infrastructure, and communication infrastructure. The modules that are directly related to the self-configuration are the *autoconfigurator*, which supports the discovery of new services and configures the system based on user requirements, and the context-adaptor which detects changes in the environment and adapts services based on user preferences. However this proposal lacks specific details about how to build this solution, namely, how the system adapts itself to changes in the environment.

Loseto et al. present a multi-agent approach that is based on service discovery and orchestration in smart-homes [14].They introduce semantic information in services and user profiles to facilitate the negotiation of the services that are most suitable according to user preferences. The agent-based framework presented is populated by a home agent, a user agent, an interface agent, and a device agent. The home agent plays a key role in the self-adaptation process of the system. The home agent facilitates the service discovery and orchestration and also mediates between user agents and device agents to maximize the overall utility. Although this proposal is interesting, it does not consider temporal constraints when services are scheduled for their execution. The lack of this feature would make the accomplishment of goals uncertain when their execution must finish before a deadline.

According to this analysis, the theoretical basis of our proposal tackles the self-adaptation of the smart-home environments in order to deal with user goals using a goal-oriented architecture. This initial formulation of the problem we want to solve is based on the gaps and proposals founded in the analysed literature (see Table 1). Therefore, this proposal improves previous approaches in the following ways:

- the proposed architecture includes an on-line planner that not only can create a new plan, but that can also refine existing plans or repair running plans. This behaviour solves the problem of having a static set of predefined plans as other proposals do.

- the on-line planner uses case-based reasoning techniques to learn from previous plans used in specific situations and reuse them according to new user requirements. This learning capability allows the adaptation of the proposal and represents an improvement on existing proposals.

- services and user goals are semantically annotated in order to facilitate automatic service discovery and composition of complex services while maintaining temporal constraints to avoid unpredictability.

- negotiation techniques are included to check the availability of service providers in real-time and to establish temporal commitments to ensure the execution of the services within a time frame. This behaviour assures the execution of complex services before the deadline established by the client.

Approach	Dynamic adaptation	User's goal-oriented	Availability of services	Service discovery and composition	Temporal constraints
[6, 7]	✗	✓	✗	✗	✗
[10, 11]	✓	✓	✗	✗	✗
[12]	✓	✗	✗	✗	✗
[13]	✓	✓	✗	✓	✗
[14]	✓	✓	✗	✓	✗
DGOC	✓	✓	✓	✓	✓

Table 1: Summary of differences between our approach and other existing systems.

3. Formal model for Distributed Goal-Oriented Computing architecture

In this section, we present the model that defines the Distributed Goal-Oriented Computing architecture (DGOC). In this work, we define the concept of Distributed Goal-Oriented Computing as the paradigm where heterogeneous agents can express their desires through goals [15]. Agents have their own goals and take actions to fulfill these goals. In order to achieve their goals, agents can use automatic service composition mechanisms considering internal and/or external services. Internal services are those services provided by the agents in the system. External services are those provided by external entities in the cloud.

The proposed architecture incorporates abstractions of *agent*, *knowledge base*, *services*, *goals*, and *plans* (compositions of services) [16]. Some of these abstractions are taken from the BDI agent model [17]. The model of this architecture aims to define a new runtime support on an operating system kernel where the basic execution entities are services instead of processes.

First, we formally define an agent a_i by the following tuple:

$$a_i = \langle KB_i, S_i, Pl_i, G_i \rangle, \quad (1)$$

where:

- KB_i represents the Knowledge Base of the agent a_i. The KB_i stores information about the states of the agent. A state consists of a set of facts that the agent believes to be true. For example, its own representation of the environment.

- S_i represents the set of services offered by the agent. The agent uses its services to achieve its own objectives and it also may offer them to other agents to help them in the achievement of their goals. The services are described using the OWL-S ontology. Therefore, an important part of a service description will be its preconditions (i.e., service preconditions and inputs) and postconditions (i.e., service postconditions and outputs).

- Pl_i represents a set of pre-compiled plans provided by the agent for the achievement of its goals. A plan is an ordered sequence of services where the post-condition of a service s_j ($Q(s_j)$) is equal to the precondition of a service s_{j+1} ($P(s_{j+1})$), where $s_j, s_{j+1} \in S$, with $S = \bigcup_{a_i} S_i$ being the domain of the set of services offered by all the agents in the system. Thus, we can define the set of pre-compiled plans as $Pl_i \subseteq Pl$, where Pl represents the domain of the total set of possible plans, that is, the set of all possible service sequences:

$$\begin{aligned} Pl = \{(s_1..s_n / \forall i \in 1..n, s_i \in S \wedge \\ \forall j \in 1..n-1, , s_j \in S \wedge i \neq j, P(s_{j+1}) = Q(s_j))\} \end{aligned}$$

The pre-compiled plans are created in order to optimize the process of composing new plans at runtime.

- G_i represents the set of goals that the agent wants to achieve. When a goal is reached, the agent marks it as an entry in its KB_i, which means that the agent believes that the facts associated to the goal are true.

Once the formal definition of an agent is presented, the model of the Distributed Goal-Oriented Computing architecture is defined as follows:

$$DGOC = \langle A, \gamma_g, \kappa_p, \delta_p \rangle \tag{2}$$

where:

- A represents the set of agents that are in the system: $A = \{a_i, a_j, ..., a_n\}$.
- $\gamma_g : 2^A \to G$ is the goal selection function, where $G = \bigcup_{a_i} g_i$ represents the domain of the set of goals of all the agents in the system and $g_i = \{g_{ij}, ..., g_{in}\}$ represents a set of goals of the agent a_i. Therefore, g_{ij} refers to the j-th goal of the agent a_i.
- $\kappa_p : G \times A \to 2^{Pl}$ is the function for the composition of new plans. This function is used when there is no plan in the set of pre-compiled plans of the agent (Pl_i). The κ_p function creates a set of service compositions taking into account the knowledge base KB_i of the agent a_i and the goal g_{ij}

$$\begin{aligned} \kappa_p(g_{ij}, a_i) = & \{(s_1..s_n / \forall i \in 1..n, s_i \in S \wedge \\ & \forall j \in 1..n-1 s_j \in S \wedge i \neq j, P(s_{j+1}) = Q(s_j)) \\ & \wedge P(s_1) \in KB_i \wedge (Q(s_n) = g_{ij})\} \end{aligned}$$

- $\delta_p : G \times 2^{Pl} \to Pl$ is the plan selection function. This function selects a plan to be executed in order to reach a selected goal. To do this, the δ_p function considers the set of pre-compiled plans Pl_i and the set of plans generated by the function $\kappa_p(g_{ij}, a_i)$. It selects a valid plan, which is a plan whose preconditions are satisfied in the KB_i of the agent a_i that activates the goal and whose post-conditions match the goal to be reached. The invocation of this function will be: $\delta_p(g_{ij}, \{p_1..p_n\} \cup \kappa_p(g_{ij}, a_i))$

The algorithm follow by the DGOC architecture is shown in Algorithm 1. Initially, a new goal is selected from the set of potential goals using the γ_g function. Then, the algorithm checks the reachability and consistency of the goal with regard to the other activated goals. After selecting a goal, the selection function δ_p is used to find a plan that will try to reach the goal. After this, the plan is executed. In case of failure, a new plan will be selected or a different existing plan could be used or adapted to deal with the goal. Finally, the correct execution of the plan is checked by analyzing the postconditions. If all the postconditions are fulfilled, the goal is marked as reached. For any uncontrolled case, the goal will be marked as non-reachable.

```
repeat
    $g_i \leftarrow \gamma_g(\{a_1..a_n\})$ ;
    if IsPossible($g_i$) $\wedge$ IsConsistent($g_i$) then
        $p_i \leftarrow \delta_p(g_i, \{p_1..p_n\}, \kappa_p(g_i, a_j))$;
        while $\neg$ IsFinished($p_i$) do
            Execute($p_i$);
            if HasFailed($p_i$) then
                $p_i \leftarrow \delta_p(g_i, \{p_1..p_n\} \cup \kappa_p(g_i, a_j))$;
            end
        end
        if CheckPostCondition($p_i$) $==$ True then
            GoalPursued($g_i$);
        else
            GoalNotPursued($g_i$);
        end
    else
        GoalNotPursued($g_i$);
    end
until True;
```

Algorithm 1: DGOC main algorithm

This algorithm presents a generalized view of the DGOC execution. Different components of BDI agents were used for this definition, such as the plan selection function, or common planning techniques in Artificial Intelligence and Case-Based Planning for the composition of new plans. The basic execution component in the architecture is the service. To describe a service in this formal model, we employ the OWL-S service ontology [18]. OWL-S is a well-defined standard that provides enough power to semantically describe all the functionality provided by agents in the DGOC architecture.

Based on the formal model for the DGOC architecture, there is a goal-oriented execution framework that provides an implementation of the model. Figure 1 shows the main components of the framework: the *Deliberation Engine*, the *Runtime Engine*, and a set of agents in which there should be at least one *Operating System Agent*. We briefly describe each component, but for more details about the components we refer the reader to [15].

The *Deliberation Engine* is responsible for deciding the order and how plans are executed. This engine is permanently running in the background, evaluating the goals that agents want to achieve and selecting them for their completion. This component contains the *On-line Planner* (i.e., the entity that composes new plans or repairs existing plans when there are no pre-compiled plans), and the *Commitment Manager* (i.e., the entity that negotiates with service providers and is able to estimate the services that offer the best temporal conditions taking into account time constraints).

The *Runtime Engine* takes plans provided by the *On-line planner* or libraries of pre-compiled plans. Then, it manages the plan execution by transferring the execution of the services included in the plan to its scheduler. This component is able to locate services that are in other nodes through a discovery protocol and include them in the plan that is in execution.

The *Operating System Agent* is composed of the following elements: a goal set to carry out the tasks associated with the operating system, a knowledge base that represents the beliefs of the operating system, and a set of services to provide the basic low-level functionality to the agents and a plan library.

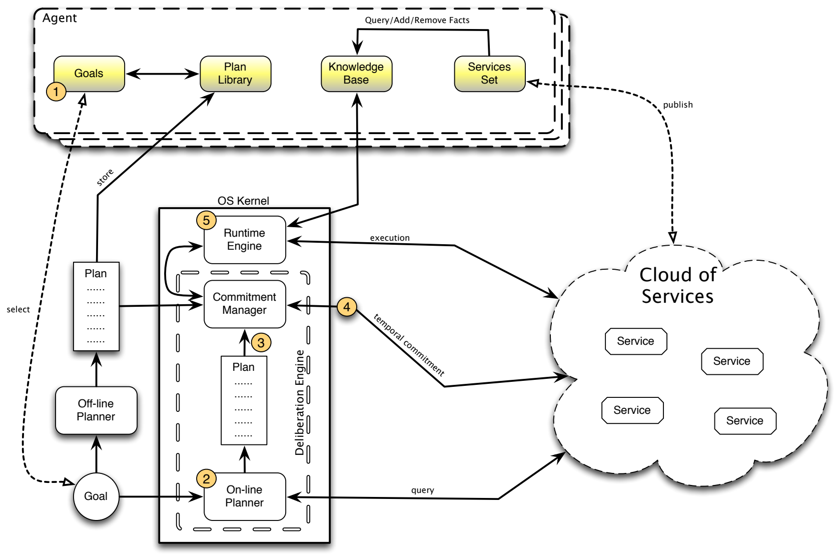


Figure 1: Execution framework based on DGOC formal model.

4. Designing a Goal-Oriented System for a Smart-Home Environment

Smart-home environments are a suitable scenario to apply the proposed formal model for the DGOC architecture. We aim to achieve a complete immersion of the user in the environment (i.e., we want human users to be a component of the system that interact with the smart-home in a natural way expressing their goals). Therefore, the use of our goal-oriented architecture model for smart-home environments represents this level of required immersion while remaining minimally invasive. A user only has to express his/her goals, and the framework based on the formal model of the DGOC architecture is responsible for finding the best way to accomplish them. The system makes most of the decisions and is able to adapt its plans to deal with unexpected situations (i.e., services that are not available or environment conditions that change).

In this section, we describe the application of the framework based on the formal model of the DGOC architecture presented in Section 3 for a smart-home by means of an example. In this smart-home, a user can express his/her desire to watch a movie when he/she gets home and communicates it to the execution framework in one of the following ways: sending an mobile message, using an iPad application, or through a voice interface. In any case, the application receives the user's intention and transforms it into a goal. This goal is introduced in the framework based on the DGOC architecture model that controls the smart-home. Although there are different options for watching a movie (i.e., renting it in an on-line shop, downloading via P2P protocols, or using video streaming services), the user only expresses the wish to watch a film. When the user expresses the goal, he/she can also provide information about restrictions and input parameters. As an example, an input parameter would be the title of the movie that the user wants to watch. Another possible input parameter would be that the user may not want to pay more than a certain amount of money to watch the movie and does not want to use a specific payment provider. There are other parameters that would also be important to take into consideration in order to reach the goal, such as the required quality of the film, or the device on which the user wants to watch the movie (i.e., a smart TV , an iPad, a PC, etc.). The entity responsible for making the best choice of services to deal with user's goals is the framework, specifically, the *Deliberation Engine*. The aim of the following example is to describe in detail how the user's goal is defined in the framework, how the plan to deal with the goal is built, and how the services that accomplish the user's goal are selected by the framework.

4.1. Scenario Description

The proposed scenario has been implemented over the DGOC framework described above (see Figure 1). The Figure 2 illustrates the initial situation of the example where the user employs a mobile interface to introduce the goal to be accomplished. This goal is sent to the DGOC framework which starts the process to reach the goal. In the implemented framework, the different rooms and devices are controlled by different intelligent agents in charge of different tasks. Thus, the *personal assistant agent* allows the user to introduce the goal into the system. Moreover, different agents that incorporate the internal services needed to solve the problem have been implemented. The *TV agent* has the capability to play a movie on a specific device (in this case, a smart TV). The *Scene Agent* has the capability to control the lights and the curtain motor of a specific room. The *Operating System Agent* allows the tasks associated with the operating system to be carried out, such as downloading the needed files and moving those files to a specific player device. The framework also includes the possibility to use different external services such as streaming or paying services, which are added (if needed) into the plans created to accomplish the goal.

For the services needed in the proposed example, we considered the services described in Tables 1-9. These tables show the available services, their preconditions (P)

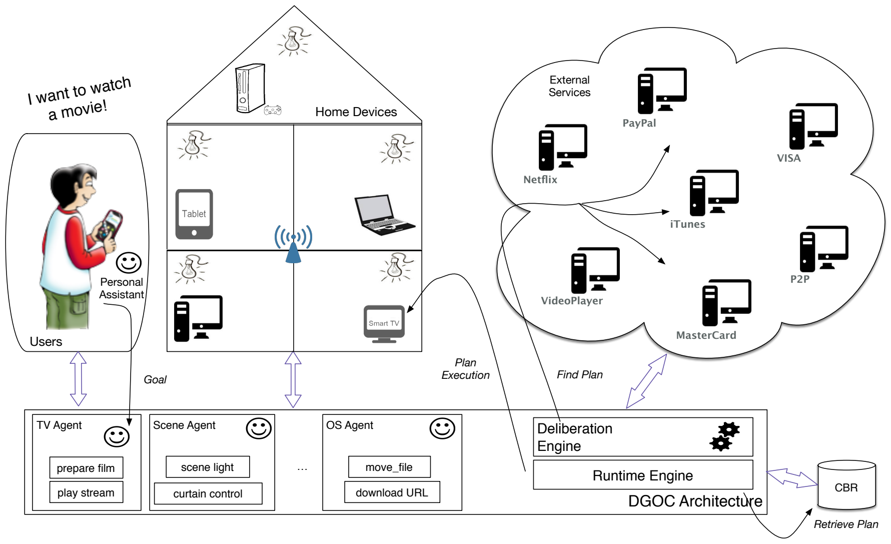


Figure 2: General view of the proposed use case.

and postconditions (Q), their probability of success (PS), and their worst-case execution time (T). There are payment service providers such as Visa (Table 5), MasterCard (Table 6), or PayPal (Table 7). There are also movie providers that rent and sell such as Netflix (Table 2) and iTunes (Table 3) and services related to the ambient conditions of the room (Table 10). There are other service providers such as P2P file sharing, film playback (Table 4), and conversion services like VideoPlayer (Table 9). In addition, the operating system provides its own services which are associated to the *Operating System Agent* (Table 8).

4.2. Execution trace

We have considered that a user wants to see a movie when he/she arrives home, with a minimum quality of 720p and paying no more than 5 euros. The user also prefers to avoid Paypal as a payment method. The user employs an external application (which acts as an interface agent) to introduce the user's desires as an active goal into the DGOC framework. An electronic interface that employs the OpenMind Commonsense knowledge base [19] and ConceptNet [20] is used to transform the user's desires and preferences into goals of the framework [21]. There are also other interfaces available that employ natural language processing (like Siri on Apple devices) or gesture recognition (like Kinect on XBOX from Microsoft) that could be used by users to introduce their goals.

With regard to our design, once user requirements are captured and transformed into a framework goal, we obtain an XML-based representation as shown in Listing

ID	Name	Conditions	PS	T

A	`film:search_film`	`P: ?film:title`
		`P: ?film:quality`
		`Q: ?film:ID`
		`Q: ?bank:price`
			$0,9$	8

B	`film:buy_film`	`P: ?film:ID`
		`P: ?bank:price`
		`Q: ?bank:objectToPurchase`
			$0,9$	3

C	`film:prepare_download`	`P: ?film:ID`
		`P: ?bank:objectPurchased`
		`Q: ?os:url`
			$0,93$	5

Table 2: Services offered by *Netflix agent*.

ID	Name	Conditions	PS	T

D	`film:search_film`	`P: ?film:title`
		`P: ?film:quality`
		`Q: ?film:ID`
		`Q: ?bank:price`
			$0,7$	7

E	`film:buy_film`	`P: ?film:ID`
		`P: ?bank:price`
		`Q: ?bank:objectToPurchase`
			$0,8$	3

F	`film:prepare_download`	`P: ?film:ID`
		`P: ?bank:objectPurchased`
		`Q: ?os:url`
			$0,81$	4

Table 3: Services offered by *iTunes agent*.

ID	Name	Conditions	PS	T

G	`film:search_film`	`P: ?film:title`
		`P: ?film:quality`
		`Q: ?film:ID`
			$0,4$	7

H	`film:prepare_donwload`	`P: ?film:ID`
		`Q: ?os:url`
			$0,8$	23

Table 4: Services offered by *P2P agent*.

ID	Name	Conditions	PS	T

I	`bank:pay`	`P: ?bank:price`
		`P: ?bank:objectToPurchase`
		`Q: ?bank:objectPurchased`
		`Q: ?bank:confirmationMethod`
		`Q: ?bank:paymentMethod(VISA)`
			$0,95$	4

Table 5: Services offered by *VISA agent*.

ID	Name	Conditions	PS	T

J	`bank:pay`	`P: ?bank:price`
		`P: ?bank:objectToPurchase`
		`Q: ?bank:objectPurchased`
		`Q: ?bank:confirmationMethod`
		`Q: ?bank:paymentMethod(MC)`
			$0,63$	6

Table 6: Services offered by *MasterCard agent*.

ID	Name	Conditions	PS	T

K	`bank:pay`	`P: ?bank:price`
		`P: ?bank:objectToPurchase`
		`Q: ?bank:objectPurchased`
		`Q: ?bank:confirmationMethod`
		`Q: ?bank:paymentMethod(PayPal)`
			$0,61$	7

Table 7: Services offered by *PayPal agent*.

ID	Name	Conditions	PS	T

L	`os:download_url`	`P: ?os:url`
		`Q: ?os:file`
			$0,76$	345

M	`os:move_file`	`P: ?os:file`
		`P: ?os:where`
		`Q: ?os:file_available`
			$0,99$	112

Table 8: Services offered by the *OS agent*.

ID	Name	Conditions	PS	T

N	`video:prepare_film`	`P: ?os:file_available`
		`P: ?os:file`
		`Q: (or`
		`(?video:ready)`
		`(?video:need_encode)`
		`)`
			$0,89$	3

O	`video:encode`	`P: ?os:file_available`
		`P: ?os:file`
		`P: ?video:need_encode`
		`Q: ?video:ready`
			$0,78$	156

P	`video:play`	`P: ?video:ready`
		`Q: ?video:viewed(?film:title)`
			$0,97$	412

Q	`video:play_stream`	`P: ?os:url`
		`Q: ?video:viewed(?film:title)`
			$0,71$	412

Table 9: Services offered by *VideoPlayer agent*.

ID	Name	Conditions	PS	T

R	`ambient:shutdown_lights`	`P: ?ambient:lights_on`
		`Q: ambient:lights_off`
			$0,99$	2

S	`ambient:close_curtains`	`P: ?ambient:curtains_opened`
		`Q: ?ambient:curtains_closed`
			$0,90$	30

Table 10: Services offered by *Scene agent*.

Listing 1: Use Case Goal.

```
<goal type='achieve' retry='true' retrydelay='0'
        recur='false' exclude='when_failed'
        name='ViewFilm'>
    <parameter name='film:title'>Casablanca</parameter>
    <parameter name='film:quality'>720p</parameter>
    <targetcondition>
        (video:viewed ?film:title)
    </targetcondition>
    <softcondition>
     (>= (film:current-quality ?film:title) ?film:quality)
    </softcondition>
    <contextcondition>
     (> (begin (video:ready ?film:title)) "21:15:00")
    </contextcondition>
    <dropcondition>
      (and
       ( > bank:price 5)
       ( = bank:paymentMethod 'PayPal')

       ( <= (end (video:viewed ?film:title))
"23:59:59" )
      )
    </dropcondition>
    <deliberation>
            <inhibits rel='ScreenSaver'/>
    </deliberation>
</goal>
```

1. The goal representation language is an extension of the language used in JadeX[1] with the addition of the temporal requirements needed in our system. To do this, we have added the temporal operators *begin* and *end*, which allow us to express whether a condition must be true before or after a specific time point.

Once the user's goal is selected by the *Deliberation Engine*, the *On-line Planner* tries to locate those plans that reach the goal. Taking into account the available services (see Tables 1-9), the *On-line Planner* will return a set of plans as a plan graph that contains all possible paths (i.e., set of services) that can be executed to accomplish the goal (see Figure 3). In the plan graph, there are several alternatives depending on the film provider, the payment method, or the movie playback. The planner found three possible movie search services from different providers: *Netflix* and *iTunes*, which are paid

[1]http://www.activecomponents.org

services, and a *P2P downloading service*, which is free. Also, since the video playback service requires some ambient preconditions (i.e., to turn off the lights and close the curtains), the ambient services related to the lights and curtains should be executed sequentially before the playback of the movie. In the pre- and post-condition predicates of the services, there are five different ontologies: *film* for movie search management and rental, *bank* for the bank payment management, *ambient* for the domotic services that manage the scenes of the smart-home, *os* for services related to the *operating system* (these services are depicted with a dotted line), and *video* for video playback and encoding services.

The *On-line Planner* performs the plan composition following a backward strategy. Starting from the goal, it searches for services whose postconditions match the conditions of the goal. During this process, the *On-line Planner* connects services until a known state is reached (i.e., a fact or set of facts that are known by the agents). To do this, the *On-line Planner* queries the knowledge base of the agent that represents the user, taking into account the input parameters of the goal. The *On-line Planner* does not return a single fixed plan; it returns a graph with different options and information to be instantiated at runtime. This is because some variables cannot be solved until runtime and it is better to keep alternatives just in case a condition fails during execution (instead of restarting the composition process of a new plan). During the composition process, if there are alternative paths, the *On-line Planner* uses a CHOICE node. Similarly, if the *On-line Planner* finds logical operators (such as OR), it may introduce conditional nodes such as IF-THEN-ELSE to follow one path or another. At this point, the *Deliberation Engine* decides which path is the best to take during the plan execution.

For example, Figure 4 shows the shortest path of the plan graph and, initially, the more advantageous. The plan has the lowest number of services, and it does not need to pass through the payment provider, which ensures the condition of not paying more than a certain price for the film. Also, the plan avoids the download of the film using the streaming playback, which presumably saves time. During the execution, at the second CHOICE node of the plan (i.e., where the system must choose between downloading or streaming the movie), the *Deliberation Engine* detects that the *video:play_stream* service has the effect: `decrease 1000mAh ?battery` (i.e., a constraint that appears in execution time). Therefore, the *Deliberation Engine* will not select this service because it has the effect of decreasing the battery, which will result in not having enough battery power to complete the execution.

Another situation that may occur if the same execution path is selected is the following. The *P2P film service* provider is the cheapest one since it is based on the content sharing among users of the network. However, the quality is usually not optimal in P2P networks. The quality, which is an input of the goal, can only be solved after the execution of the service. Therefore, it will be at that time when the *Deliberation Engine* selects the next execution step. If the condition of quality is a *dropcondition* (i.e., a compulsory condition that implies the immediate cancelation of the goal, if it is not satisfied), the *Deliberation Engine* must find an alternative path using *Netflix* or *iTunes* service providers. However, since it is a *softcondition* (i.e., a desirable condition, that is not compulsory to satisfy), the plan execution could continue. In the case of an empty search, the plan execution fails and forces an execution restart that, in the worst case

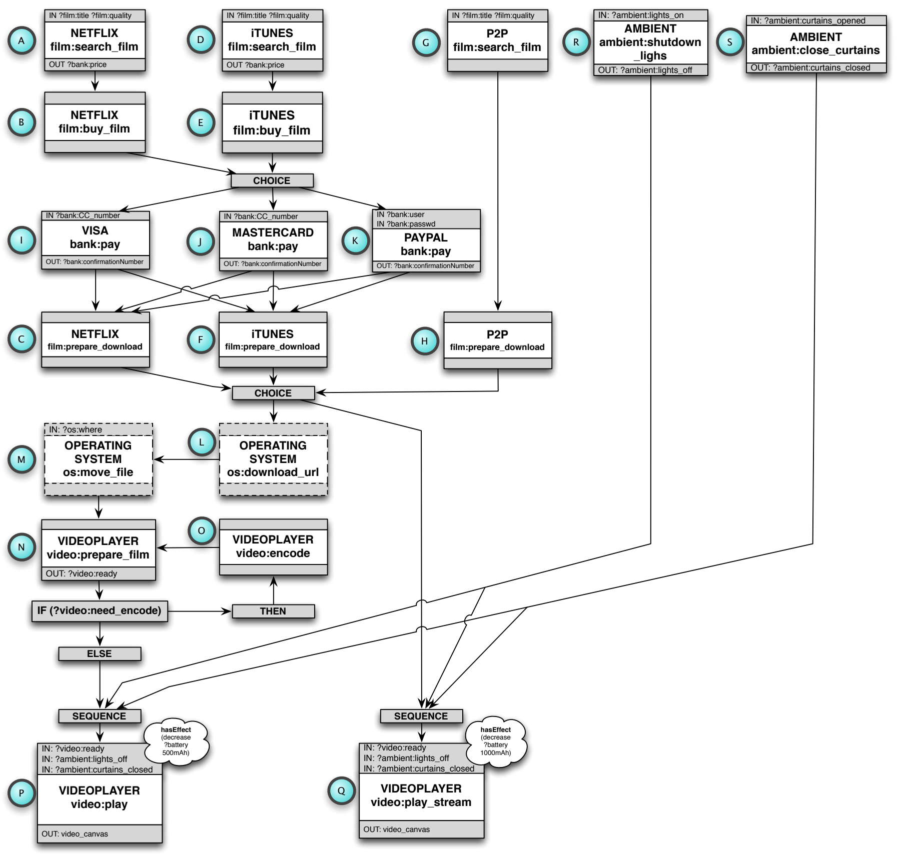

A
IN ?film:title ?film:quality
NETFLIX film:search_film
OUT ?bank:price
D
IN ?film:title ?film:quality
iTUNES film:search_film
OUT ?bank:price
G
IN ?film:title ?film:quality
P2P film:search_film
R
IN: ?ambient:lights_on
AMBIENT ambient:shutdown _lighs
OUT: ?ambient:lights_off
S
IN: ?ambient:curtains_opened
AMBIENT ambient:close_curtains
OUT: ?ambient:curtains_closed
B
NETFLIX film:buy_film
E
iTUNES film:buy_film
CHOICE
I
IN ?bank:CC_number
VISA bank:pay
OUT: ?bank:confirmationNumber
J
IN ?bank:CC_number
MASTERCARD bank:pay
OUT: ?bank:confirmationNumber
K
IN ?bank:user
IN ?bank:passwd
PAYPAL bank:pay
OUT: ?bank:confirmationNumber
C
NETFLIX film:prepare_download
F
iTUNES film:prepare_download
H
P2P film:prepare_download
CHOICE
M
IN: ?os:where
OPERATING SYSTEM os:move_file
L
OPERATING SYSTEM os:download_url
N
VIDEOPLAYER video:prepare_film
OUT: ?video:ready
O
VIDEOPLAYER video:encode
IF (?video:need_encode)
THEN
ELSE
SEQUENCE
hasEffect (decrease ?battery 500mAh)
P
IN: ?video:ready
IN: ?ambient:lights_off
IN: ?ambient:curtains_closed
VIDEOPLAYER video:play
OUT: video_canvas
SEQUENCE
hasEffect (decrease ?battery 1000mAh)
Q
IN: ?video:ready
IN: ?ambient:lights_off
IN: ?ambient:curtains_closed
VIDEOPLAYER video:play_stream
OUT: video_canvas

Figure 3: Plan graph of the proposed use case.

could require a replanning. Furthermore, since the goal includes not using *PayPal* service as the payment method as a *dropcondition*, the payment service *bank:pay* of the *PayPal* service provider will not be selected for execution.

Once the *On-line Planner* has calculated the set of possible paths that can be executed, it delivers the set of plans to the *Commitment Manager* in order to establish the preliminary agreements with the services that compose those plans. The *Commitment Manager* will choose which plans are the most appropriate to deal with the user's goal based on the user's constraints and service provider execution times.

The *Commitment Manager* calculates the probability of success of each path to make the final decision (i.e., which path of the plan will be executed). Every path that passes through the service K *(bank:pay)* of the *PayPal* service provider is pruned because the statement *bank:paymentmethod(PayPal)* fulfils one of the *dropconditions* of the goal. To estimate the execution time when an IF-THEN-ELSE sentence appears in the path, the *Deliberation Engine* takes the worst case and limits the cycles to at

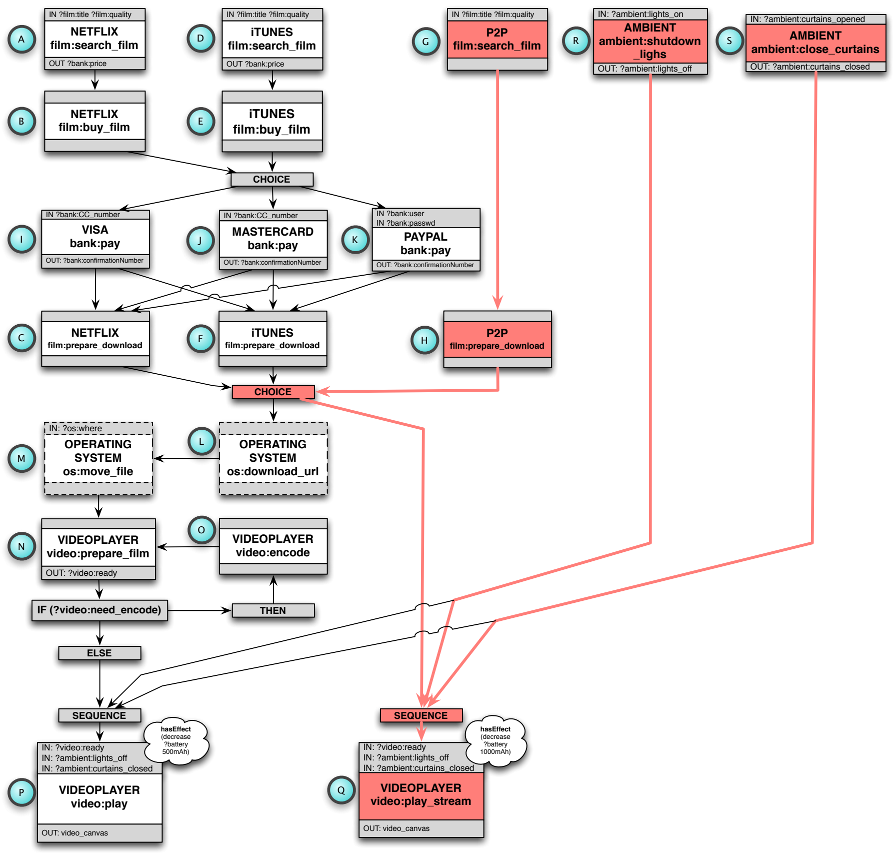

A
IN ?film:title ?film:quality
NETFLIX film:search_film
OUT ?bank:price
D
IN ?film:title ?film:quality
iTUNES film:search_film
OUT ?bank:price
G
IN ?film:title ?film:quality
P2P film:search_film
R
IN: ?ambient:lights_on
AMBIENT ambient:shutdown _lighs
OUT: ?ambient:lights_off
S
IN: ?ambient:curtains_opened
AMBIENT ambient:close_curtains
OUT: ?ambient:curtains_closed
B
NETFLIX film:buy_film
E
iTUNES film:buy_film
CHOICE
I
IN ?bank:CC_number
VISA bank:pay
OUT: ?bank:confirmationNumber
J
IN ?bank:CC_number
MASTERCARD bank:pay
OUT: ?bank:confirmationNumber
K
IN ?bank:user
IN ?bank:passwd
PAYPAL bank:pay
OUT: ?bank:confirmationNumber
C
NETFLIX film:prepare_download
F
iTUNES film:prepare_download
H
P2P film:prepare_download
CHOICE
M
IN: ?os:where
OPERATING SYSTEM os:move_file
L
OPERATING SYSTEM os:download_url
N
VIDEOPLAYER video:prepare_film
OUT: ?video:ready
O
VIDEOPLAYER video:encode
IF (?video:need_encode)
THEN
ELSE
SEQUENCE
hasEffect (decrease ?battery 500mAh)
IN: ?video:ready
IN: ?ambient:lights_off
IN: ?ambient:curtains_closed
P
VIDEOPLAYER video:play
OUT: video_canvas
SEQUENCE
hasEffect (decrease ?battery 1000mAh)
IN: ?video:ready
IN: ?ambient:lights_off
IN: ?ambient:curtains_closed
Q
VIDEOPLAYER video:play_stream
OUT: video_canvas

Figure 4: Shortest path of the plan.

most two iterations to avoid loops.

In order to estimate the suitability of each plan taking into account the user requirements, the *Commitment Manager* establishes a ranking, taking where each path has an associated value that represents the aggregation of the probability of success and the estimated execution time. Therefore, for each path x, the *Commitment Manager* calculates the probability of success PS_x (i.e., the probability of a successful execution of all the services that are part of the path) and its estimated execution time T_x (i.e., the time required for the complete execution of all the services that are part of the path). PS_x is calculated as follows: $PS_x = \prod_{i=0}^{N} PS_i * \omega_i$ and its estimated execution time is calculated by the equation $T_x = \sum_{i=0}^{N} T_i$. In order to simplify this trace, we consider that the base case does not yet have previous experiences, and, therefore, we set the value of $\omega_i = 1$. Finally, we normalize the value of T_x. To do this, we use the average value of all the T_x, which is called $\overline{T}$, and the standard deviation σ. The normalized value is calculated by the equation: $\widetilde{T_x} = 1 - \frac{T_x - \overline{T}}{\sigma}$. With this information, the paths extracted from the plan graph depicted in Figure 3 and their values PS_x, T_x and $\widetilde{T_x}$ are shown in Tables 11 and 12.

Once the probability PS_x and the estimated execution time $\widetilde{T_x}$ values for each path of the plan are calculated, in the next step, the *Commitment Manager* compares all the paths and makes a decision. The *Commitment Manager* determines that the best rated path is path #7 ($PS_7 + \widetilde{T_7} = 2.45$). After this, the *Commitment Manager* confirms the temporal commitments with service providers and sends this information to the *Runtime Engine* for their execution. Similarly, the rest of the previously established commitments for service providers that are not part of the selected path will be canceled, releasing the reservation made with the providers of those services.

Plan recovery would be executed if any service of the selected plan fails (i.e., there would be a replanning from the last successful execution point). The previously gathered information can be reused to establish new temporal commitments. If it is not possible to use any of the previously calculated options (for instance, because a provider is off-line), the *On-line Planner* will try to generate a new plan. Finally, in the worst case, if no new plan can be found, the goal would be marked as unreachable.

4.3. Evaluation

We ran several experiments to evaluate the proposed architecture applied to the goal-oriented smart-home environment scenario. We built an adhoc simulator that allowed us to validate different scenarios. The software developed for these experiments is an agent-based simulator which creates as many agents as each experiment has defined. The simulator allows us to configure synthetic scenarios where we can define a load, the behavior of each component and some events that will be triggered at defined instants of time. The simulator is also responsible of collecting all the intermediate information during the execution of the simulation in order to validate how changing some parameters improves the resulting metrics (number of completed goals, number of executed services, etc.). Then, it emulates a full network of connected agents within

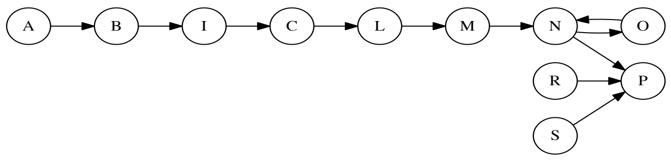
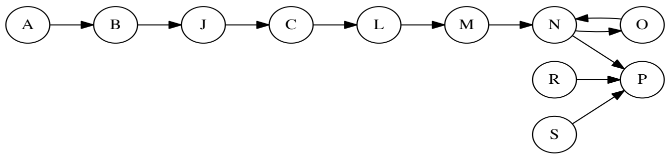
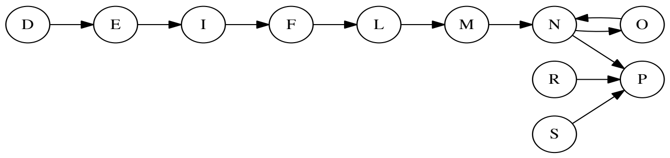
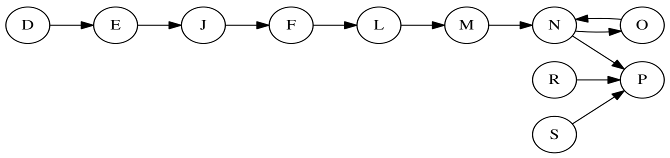
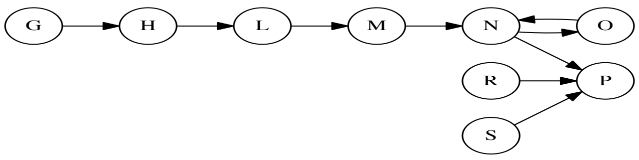
1		$PS_1 = 0.32$	$T_1 = 1080$	$\widetilde{T_1} = 0.0066$
2		$PS_2 = 0.21$	$T_2 = 1082$	$\widetilde{T_2} = 0.0001$
3		$PS_3 = 0.19$	$T_3 = 1078$	$\widetilde{T_3} = 0.0131$
4		$PS_4 = 0.13$	$T_4 = 1080$	$\widetilde{T_4} = 0.0066$
5		$PS_5 = 0.14$	$T_5 = 1090$	$\widetilde{T_5} = -0.0259$

Table 11: Set of possible execution paths (1 to 5).

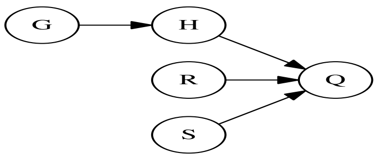
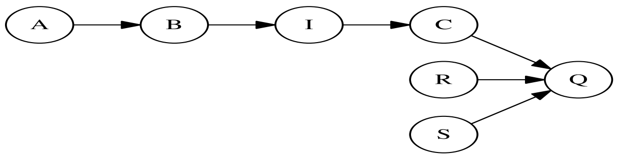
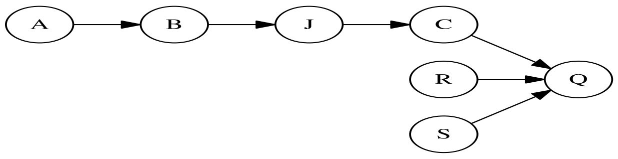
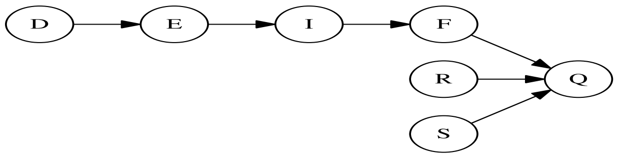
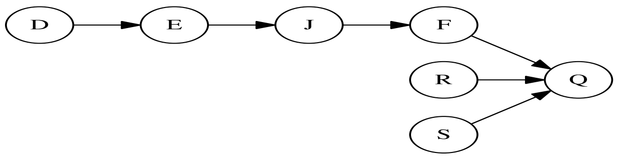
6		$PS_6 = 0.20$	$T_6 = 474$	$\widetilde{T_6} = 1.9739$
7		$PS_7 = 0.45$	$T_7 = 464$	$\widetilde{T_7} = 2.0064$
8		$PS_8 = 0.30$	$T_8 = 466$	$\widetilde{T_8} = 1.9999$
9		$PS_9 = 0.27$	$T_9 = 462$	$\widetilde{T_9} = 2.0129$
10		$PS_{10} = 0.18$	$T_{10} = 464$	$\widetilde{T_{10}} = 2.0064$

Table 12: Set of possible execution paths (6 to 10).

an environment where the execution of agent-provided services is emulated in a discretized implementation of the simulator. This emulated execution of services allows us to introduce some forced errors in the simulation that help the observer to validate the self-adaptation capabilities of the implementation. The execution of experiments is designed to accept some parameters to validate different behaviors of the prototype in different situations. It is possible to set the probability of success for the execution of each service to simulate the possibility of non-fulfilment of a commitment, the worst-case execution time of a service or some events that change the scenario at a defined instant of time (e.g. we can change the pre-defined probability of success of a service at the middle of the simulation). We have released as open source (LGPL license) the main components of our software in an agent platform called SPADE[2]. This is a platform developed and maintained since 2006 which includes all the significant components of this proposal. Specifically, the SPADE platform includes the main component of the On-line Planner, which is the Case-Based Planner, the Knowledge Base (KB) to allow agents to set goals and the protocol to offer and consume services remotely, make plans and run services that help to achieve their goals. This protocol is based on *jabber-rpc*[3] and takes as inputs the knowledge stored in the agent's KB and pushes the results of the service invocation also in the agent's KB. The SPADE platform is developed in Python and based on the XMPP[4] protocol (also known as Jabber) to perform communications. SPADE agents can publish their services to a Service Discovery agent. Other agents can search and find these services and invoke them, interacting with the Service Discovery agent. The Case-Base Planner is also prepared to search in this Service Discovery agent to look for the services that allow the agent to achieve its goals.

In the following experiments, in order to consider a more real scenario, we have increased the number of movie and payment providers that we have presented in the Execution Trace section . As a result, the number of possible service combinations to obtain the final plan to be executed increased. A more detailed description of the simulator can be found in [15]. In this case, we designed a client agent that represented the behavior of the user. Moreover, the different movie providers, payment methods, and auxiliary services were controlled by a dynamic set of agents playing the role of service providers. In each execution, a client agent activated its goal and then the *Deliberation Engine* selected a plan to perform this goal. After the execution of the plan, the client agent stored its results in the KB_i and re-activated the goal once more for the next experiment.

4.4. Evaluation of the degree of success

The first experiment evaluated the percentage of success in the fulfilment of the goals in three different scenarios (see Figure 5). In the first case (Own), each agent only employed its stored plans in order to fulfil the goal, and, consequently, its degree of success was below the minimum considered to be acceptable. In the second case

[2]http://github.com/javipalanca/spade
[3]https://xmpp.org/extensions/xep-0009.html
[4]https://xmpp.org

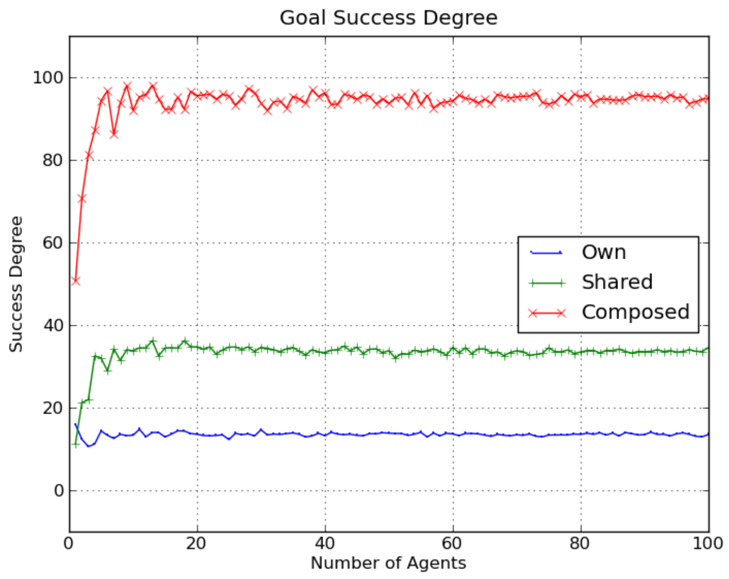


Figure 5: Study degree of success finding a plan.

(Shared), the agents could share their plans and, obviously, the degree of success increased considerably. In the third case, the agents included the on-line planner, which allowed them to compose new plans considering services offered by other agents. In this case, the degree of success obtained stood out from the rest. The results of this experiment show that collaboration among service providers and the inclusion of an *On-line Planner* produce an important increase in the success rate of goal fulfilment.

On the other hand, Figure 6 shows similar results from another perspective, taking into account the number of available pre-compiled plans. The figure shows that the success rate for goal fulfilment grows faster when the number of available agents increases than when the number of precompiled plans available increases. Therefore, the system achieves greater efficiency by increasing the number of agents that offer services in the system than by increasing the number of precompiled plans.

4.5. Self-adaptation evaluation

The next experiment (see Fig. 7 and Fig. 8) shows how the proposed system is able to adapt itself to changes in the environment. Self-adaptation is very important since it allows the system to have a dynamic behavior (i.e., the system reconfigures itself to take full advantage of current circumstances). This self-adaptation feature allows the system to be able to re-plan any running plan that is being executed by the Runtime Agent when any of the services included in the plan is no longer available (a network error, a service outage...). But self-adaptation not only allows plans to be recovered when some errors appear, it also allows agents to change their selections when composing new plans. This is done because agents take into account the reliability of the service provider

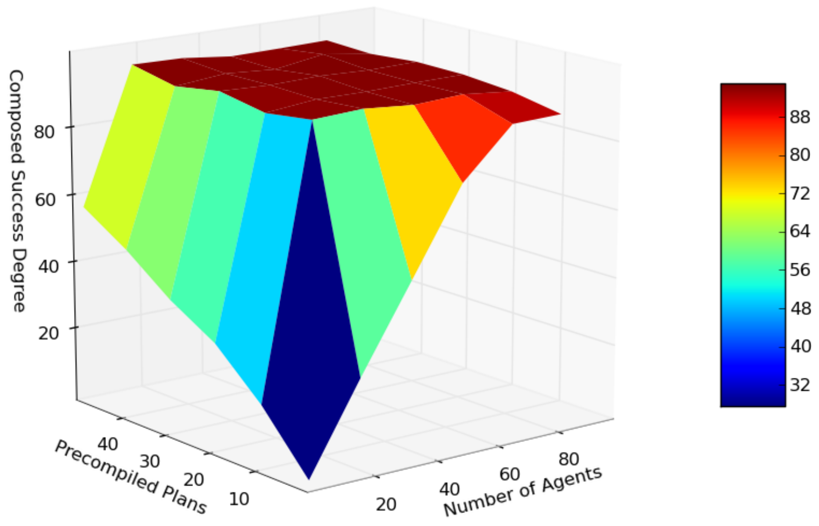


Figure 6: Study of success degree finding a plan taking into account the number of available precompiled plans.

agents based on a trust value, which takes into account the probability of success of a service execution. To calculate trust they use the execution history of its provider agent by means of the cases stored in the Case-Based Planner. This way, agents adapt their behavior by selecting at any time the most reliable service providers based on their own experience (stored as cases) and the shared experience of other agents by means of shared plans.

In this experiment, we modified the accuracy of the probability of success parameter PS of some nodes in order to analyze how the system adapts itself to changes in the environment. We activated three events that modified the environment. Specifically, the scheduled events were:

- (time 50000) Hulu node increases its accuracy up to 0.8
- (time 300000) P2P node increases its accuracy up to 0.9 and iTunes node decreases up to 0.1
- (time 600000) P2P node decreases its accuracy up to 0.5 and Hulu node decreases up to 0.01

Figure 7 shows how the trust that the client had in the provider nodes (y-axis) changed over time (x-axis) and how the environment executed scheduled changes (wrapped in the figure by vertical rectangles) in the reliability of some provider nodes. The self-adaptation took some time (see Figure 7), since the *Deliberation Engine* [15] took a while to realize that the reliability of some of the provider nodes had been reduced. The Figure 7 also shows at instant 50000 how trust in the *Hulu* node increased due to the first scheduled change. Note that this adaptation took time to consolidate. When

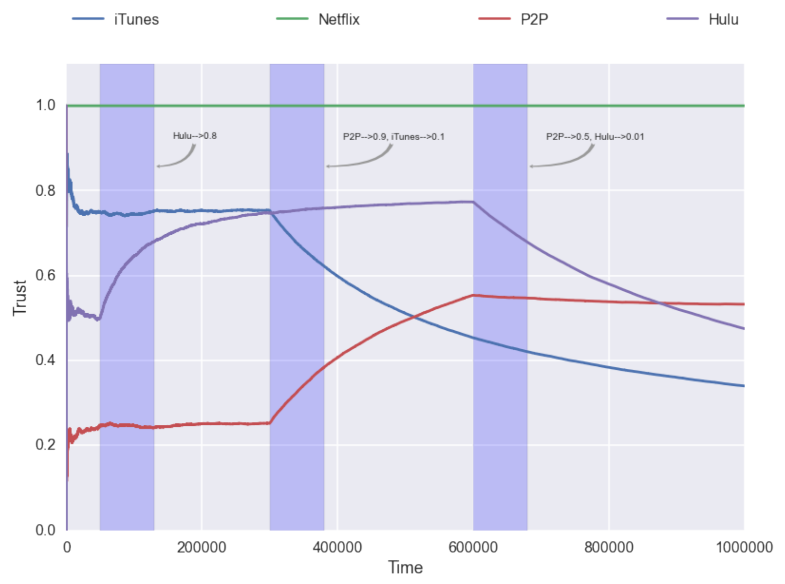


Figure 7: Evolution of the trust of provider nodes when changes in the environment occur.

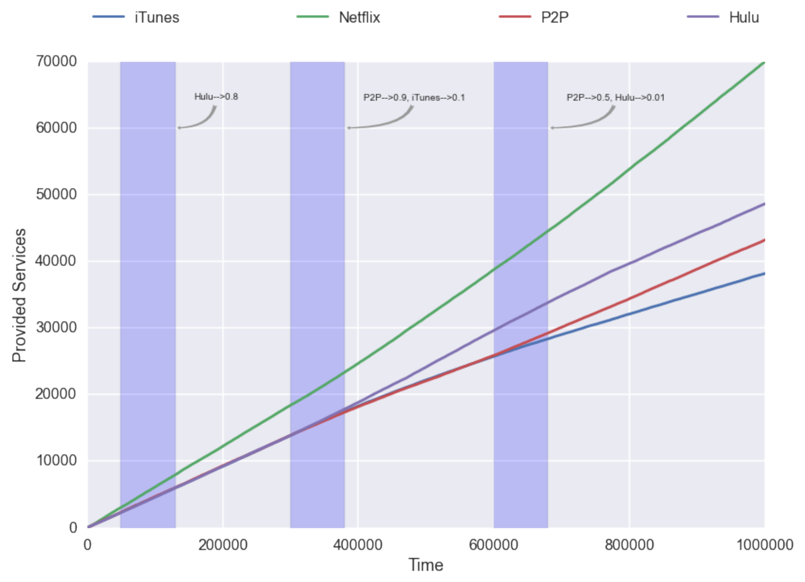


Figure 8: Study of the service provider's reorganization taking into account the # of services (aggregated).

the second event occurred at time 300000, trust in the *P2P* node began to increase (it increased close to 60%). Meanwhile, confidence in the *iTunes* node decreased. The rest of the nodes held their values. The third event changed the behavior of the system, giving less reliability to the *Hulu* node, which decreased to 10%. At the same time, the *Netflix* node maintained its confidence throughout the entire experiment. These results show that the proposed system is able to adapt itself when unexpected events occur and the environment changes. In this experiment, the client agent changed its trust values that were associated to the service provider nodes. Therefore, the number of requests that the client made to each node changed. Figure 8 shows an aggregated view of the number of services provided for each node when the three events occurred. It can be observed that the slopes of the lines that represent each node change over time according to the scheduled events. An interesting aspect to consider is the behavior of the *Netflix* node. The node was not directly affected by the events; however, indirectly, its number of service requests increased over time due to the loss of accuracy of the other providers.

Summarizing, the obtained results show that the proposed self-adaptive architecture is suitable for dynamic environments where there is not a predefined set of services and the users goals and restrictions (specifically temporal restrictions) change over time. Nevertheless, other aspects should be considered in real-life situations. For instance, one of the issues that can appear is related to services that are not semantically annotated. This would difficult their discovery and their inclusion in new plan compositions. Also, there could be services that are semantically annotated but with different ontologies. Therefore, it would be necessary to perform an ontology alignment in order to facilitate the composition of new plans. Another issue to consider in real-life scenarios is the complexity in the representation of complex user's goals. This complexity can be reduced using friendly user-interfaces.

5. Conclusions

This paper presents a goal-oriented smart environment that is based on the Distributed Goal-Oriented Computing architecture. We consider this approach to be appropriate for the development of smart environments where the immersion of users is a key factor. In the proposed architecture, users express their goals, and the architecture is in charge of achieving these goals by means of a service-oriented approach. The architecture facilitates the interaction from the perspective of users. This interaction could be performed through objects and actions that a person accustomed to using. Thus, the architecture allows users to reach a high level of immersion in the multi-agent system, minimizing the level of difficulty of the interaction. In other words, the satisfaction level of the user will be improved.

We have described how to define the properties of a goal and the parameters related to the goodness of a plan in a smart-home scenario. Moreover, a detailed execution trace of the whole process has been presented and several experiments have been done in order to evaluate the proposal. A prototype of the proposed architecture that covers all of the described functionalities was developed for the experiment. This scenario has allowed us to perform experiments under real environment conditions.

This proposal performs well with a low scalability threshold. Very high thresholds can lead to significant increases in terms of run-time and case-base size, while at the same time decreasing the agent's performance. However, since the number of cases also increases for more complex scenarios, a better handling of large scale case-bases is a problem that will have to be addressed in the future. When using the current approach in larger scenarios higher similarity thresholds might also be required to distinctively separate between cases. Optimizing the case-retrieval with a more sophisticated method can improve the performance of the CBP component of the proposed framework keeping run times at reasonable levels. The next steps should also include the ability to manage groups of people at service level. At the moment, the agents' behavior does not take into account the different preferences of a group of people (i.e., multi-occupancy), or conflict resolution among agents when they have competing goals. In the future, we would like to change this to enable more team-oriented strategies, which is a significant increase in complexity. We also plan to consider the inclusion of QoS (quality of service) of the available services in the negotiation process. This could be easily included to improve the current version.

Moreover, it would be necessary to move towards the integration and deployment of the architecture as a real OS. As future work, we have begun a study to analyze the feasibility of modifying an existing operating system. We are also planning the deployment of our proposal in the MEDERI living lab (http://mederi.ai2.upv.es/en/) in our university. This living lab is a multidisciplinary environment, mainly focused on health technology, that will give us the needed tools for a real involvement of users in order to improve the experience and robustness of the proposed techniques.

6. Acknowledgments

This work is partially supported by the Spanish Government through the MINECO/FEDER project TIN2015-65515-C4-1-R.

References

[1] M. Huhns, et al., Research directions for service-oriented multiagent systems, IEEE Internet Computing 9 (2005) 69–70.

[2] Y. Reddy, Pervasive Computing: Implications, Opportunities and Challenges for the Society, 1st International Symposium on Pervasive Computing and Applications (2006) 5.

[3] J. M. Molina, J. M. Corchado, J. Bajo, Ubiquitous Computing for Mobile Environments , in: Issues in Multi-Agent Systems, Birkhäuser Basel, 2008, pp. 33–57.

[4] M. R. Alam, M. B. I. Reaz, M. A. M. Ali, A review of smart homes: Past, present, and future, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 42 (6) (2012) 1190–1203.

[5] L. C. De Silva, C. Morikawa, I. M. Petra, State of the art of smart homes, Engineering Applications of Artificial Intelligence 25 (7) (2012) 1313–1321.

[6] C. Cetina, P. Giner, J. Fons, V. Pelechano, Autonomic computing through reuse of variability models at runtime: The case of smart homes, Computer 42 (10) (2009) 37–43.

[7] F. Dalpiaz, P. Giorgini, J. Mylopoulos, An architecture for requirements-driven self-reconfiguration, in: Advanced Information Systems Engineering, Springer, 2009, pp. 246–260.

[8] D. J. Cook, Multi-agent smart environments, Journal of Ambient Intelligence and Smart Environments 1 (1) (2009) 51–55.

[9] R. B. Matthews, N. G. Gilbert, A. Roach, J. G. Polhill, N. M. Gotts, Agent-based land-use models: a review of applications, Landscape Ecology 22 (10) (2007) 1447–1459.

[10] M. U. Iftikhar, D. Weyns, Activforms: active formal models for self-adaptation., in: SEAMS, 2014, pp. 125–134.

[11] A. Andrushevich, M. Staub, R. Kistler, A. Klapproth, Towards semantic buildings: Goal-driven approach for building automation service allocation and control, in: Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on, IEEE, 2010, pp. 1–6.

[12] I. Ayala, M. Amor, L. Fuentes, Self-configuring agents for ambient assisted living applications, Personal and ubiquitous computing 17 (6) (2013) 1159–1169.

[13] K. Kucher, D. Weyns, A self-adaptive software system to support elderly care, Modern Information Technology, MIT.

[14] G. Loseto, F. Scioscia, M. Ruta, E. Di Sciascio, Semantic-based smart homes: a multi-agent approach, in: 13th Workshop on Objects and Agents (WOA 2012), Vol. 892, 2012, pp. 49–55.

[15] J. Palanca, M. Navarro, V. Julian, A. García-Fornes, Distributed Goal-oriented Computing, Journal of Systems and Software (http://dx.doi.org/10.1016/j.jss.2012.01.045) 85 (7) (2012) 1540–1557.

[16] L. de Silva, L. Padgham, Planning as needed in BDI systems, International Conference on Automated Planning and Scheduling.

[17] A. Rao, M. Georgeff, BDI agents: From theory to practice, Proceedings of the first international conference on multi-agent systems (ICMAS95) (1995) 312–319.

[18] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, others, OWL-S: Semantic markup for web services, W3C member submission 22 (2004) 2007–2004.

[19] P. Singh, The public acquisition of commonsense knowledge, in: Proceedings of AAAI Spring Symposium: Acquiring (and Using) Linguistic (and World) Knowledge for Information Access, 2002.

[20] H. Liu, P. Singh, ConceptNet—a practical commonsense reasoning tool-kit, BT technology journal 22 (4) (2004) 211–226.

[21] H. Lieberman, J. Espinosa, A goal-oriented interface to consumer electronics using planning and commonsense reasoning, Proceedings of the 11th international conference on Intelligent user interfaces (2006) 226–233.