A Declarative Goal-oriented Framework for Smart Environments with LPaaS*

Giuseppe Bisicchia, Stefano Forti, and Antonio Brogi

Department of Computer Science, University of Pisa, Pisa, Italy

Abstract. Smart environments powered by the Internet of Things aim at improving our daily lives by automatically tuning ambient parameters (e.g. temperature, interior light) and by achieving energy savings through self-managing cyber-physical systems. Commercial solutions, however, only permit setting simple target goals on those parameters and do not consider mediating conflicting goals among different users and/or system administrators, and feature limited compatibility across different IoT verticals. In this article, we propose a declarative framework to represent smart environments, user-set goals and customisable mediation policies to reconcile contrasting goals encompassing multiple IoT systems. An open-source Prolog prototype of the framework is showcased over two lifelike motivating examples.

Keywords: Goal-oriented systems · IoT · Logic Programming · LPaaS

1 Introduction

The Internet of Things (IoT) is continuously growing and becoming an integrated part of our daily lives with a plethora of new different applications (e.g. smart-environments, wearables, home appliances) [18, 26], that show even capable of affecting our mood [14]. Among the new verticals the IoT is enabling, *smart environments* are getting increasing attention from the market and the research community [21]. Indeed, they empower private and public ambients to self-manage cyber-physical systems (e.g. A/C, lights, plants watering) based on data from IoT sensors, triggering reactions enabled by IoT actuators. Besides their high potential to improve people's routines, these applications can also lead to a more sustainable energy and resource management.

Especially for those applications that include human goals in the self-management loop of smart environments, the problem of reconciling contrasting goals among different users emerges clearly [25, 31]. Colleagues sharing a room in a public building – even for a limited amount of time – can possibly express very different desiderata on the temperature and on the light intensity they prefer to experience while they work. To this end, many techniques have been proposed

---

* Copyright © 2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0). Work partly supported by projects: *GIÒ* funded by the Dept. of Computer Science of the Univ. of Pisa, Italy; *CONTWARE* funded by the Conference of Italian University Rectors; *O. Carlini Scholarships 2020* funded by the GARR Consortium.

to reconcile such contrasting goals set by users or *system administrators*, e.g. via fuzzy logic [28], multi-agent systems [7,12] or neural networks [17]. However, most commercial solutions, such as IFTTT [1] or Amazon Alexa [2], only allow setting simple goals to be met by the IoT systems they manage and do not consider the possibility of mediating among contrasting objectives [5].

Additionally, despite being deployable out-of-the-box by their final users, existing commercial solutions show inherent limitations, mainly due to their proprietary nature. These limitations prevent them to be extended and from work *across* IoT verticals enabled by different vendors. They also make it difficult to develop policies to mediate between users and administrator objectives, i.e. set local and global goals. Factually, two different types of conflict can arise:

User-user – Different users can set different goals on their desired state of the environment (e.g. on target temperature),

User-admin – The *System Administrator* can set global objectives that must be met (e.g. on maximum energy consumption, on law constraints), which may conflict with the user-set goals.

Even after reconciling the previous types of conflicts into one target state satisfying all set (user and/or global) goals, a final configuration of the actuators involved must also be determined. Indeed, given a final target state, we need to (a) determine the correct configuration for each actuator acting on that state, and (b) mediate between any conflicting configurations that a single actuator possibly receives.

In this article, we propose[1] a declarative methodology to specify customisable mediation policies for reconciling contrasting goals and actuator settings in smart environments. The methodology can solve contrasting goals by reasoning on the available IoT infrastructure and on (possibly contrasting) goals set by the users and by system administrators. The novel contribution mainly consists of:

(1) a declarative framework to specify mediation policies for reconciling contrasting (user and/or global) goals and actuator settings in smart environments,
(2) a Prolog prototype implementation of (1), Solomon, provisioned as a REST service by relying on Logic Programming-as-a-Service ($LPaaS$), recently proposed by Calegari et al. [9] and succesfully applied to complex IoT wireless networks [8].

Solomon tames the effects of the aforementioned types of conflict by allowing to flexibly specify *ad-hoc* mediation policies for distinct zones of a smart-environment and possible conflicting settings of target actuators. Such policies can resolve conflicts (i) among users' goals, (ii) among users' and system administrator's goals, and *(iii)* on actuators configuration. Last, but not least, the declarative nature of Solomon makes it easy to write, maintain and extend arbitrary mediation policies encompassing multiple IoT verticals.

The rest of this article is organised as follows. After illustrating two motivating examples (Sect. 2), we present our methodology for goal mediation and its prototype (Sect. 3), showcasing them over the first motivating example. The full

[1] A work-in-progress and preliminary version of this study was presented in [6].

prototype is subsequently assessed over the second motivating example (Sect. 4). Finally, we discuss some closely related work (Sect. 5) before concluding (Sect. 6).

2 Motivating Examples

In this section, we illustrate two scenarios from smart environments to better highlight the need for reasoning solutions capable of mediating among contrasting goals and encompassing different IoT verticals. Both scenarios consider two main stakeholders:
User – a human or digital agent that can set goals on the ambient around them, aiming at creating the most comfortable environment for them to live in,
System Administrator – a human or digital agent that can define conflict resolution policies, and set global goals on the environment (e.g. on energy savings).

Smart Home – Consider a shared room in a student apartment, equipped with three lights – the main light, a bed light and a corner light – and an A/C system. In this case, depending on the time of day and the activity that is taking place (e.g. studying, watching a film, reading a book), different lighting configurations could be required. Conflicts might arise as, for instance, Alice may want to watch a movie while Bob is still studying in the same room. Moreover, Alice might prefer to stay in a cool room (20°C) while Bob prefers a warmer ambient (26°C). Natural questions raised by the above scenario are:

- *Is it possible to find a configuration of the three lights which allows Alice and Bob to comfortably carry on their different activities?*
- *Is it possible to find a configuration of the A/C system which mediates among the preferences of Bob and Alice on the environment temperature?*

Smart Building – Consider now the smart building floor sketched in Fig. 1, consisting of a West and an East wings. The West wing is exposed to light most of the day while the East wing is less illuminated.

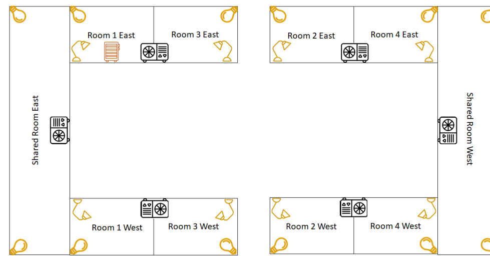

Fig. 1. An example of *Smart Building*

In each wing, there are 5 rooms (4 single and one shared), the single rooms in pairs share the air conditioning system and the relative temperature sensor. Also, each room has a large light and a desk light and a brightness sensor. The shared rooms have two large lights and their air conditioning system as well as a temperature sensor and a brightness sensor. Besides, the first single room in the East wing ha a small heater.

Each user has been assigned a single room and has full access to shared rooms. Furthermore, targeting sustainability, company policies require that the

temperature in the environment stays within 18°C and 22°C in autumn and winter, and between 24°C and 28°C in spring and summer. Also in these settings, some questions arise such as:

Q1 *How to describe the available Smart Building so that it is possible to apply ad-hoc policies for the West and East wings?*
Q2 *How to specify policies to manage temperature and brightness in the different rooms of the building, handling conflicts so as to ensure the comfort of its inhabitants and to meet sustainability policies?*
Q3 *After finding a target state for a specific room, how to determine suitable settings of the available (shared and non-shared) actuators to reach it?*

All questions raised above highlight the need for novel models and methodologies to flexibly manage smart environments, such as the one we propose in this article. In the next section, we will detail our proposal by relying on the *Smart Home* example. The *Smart Building* example will be used instead in Sect. 4 to assess the methodology over a larger scale scenario.

3 Methodology and Prototype

In this section, we illustrate Solomon, a declarative framework featuring autonomic goal mediation in smart environments, in presence of multiple users. Solomon is prototyped and open-sourced[2] in Prolog, using *LPaaS*. Hereinafter, we give an overview of the architecture we foresee for Solomon to be deployed (Sect. 3.1), and we detail the model (Sect. 3.2) and methodology (Sect. 3.3) underlying our framework.

3.1 Overview

Fig. 2 gives a bird's-eye view of the architecture of Solomon. Solomon interacts with a *smart environment*, consisting of IoT sensors and actuators (or the services they are wrapped in). Indeed, Solomon periodically receives updated *data* from the sensors deployed in the smart environment, depending on which it can trigger suitable *actions* for the available actuators.

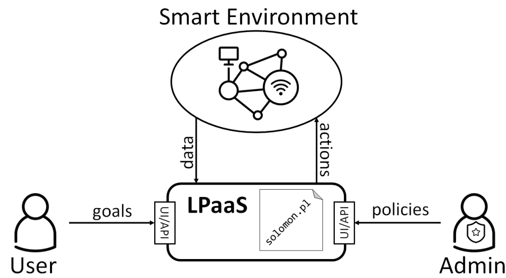

Fig. 2. Bird's-eye view of Solomon.

Both *users* and *system administrators* interact with Solomon, through the *LPaaS* API or through available UIs. On one hand, users can declare *goals* on the target state they wish to experience while being in the smart environment. On the other hand, system administrators can declare global mediation policies to solve user-user conflicts, to set global

[2] Open-sourced and freely available at: `https://github.com/di-unipi-socc/Solomon`

goals and solve user-admin conflicts, and to determine actuator configurations useful to reach a target state for the smart environment, after goal mediation. Note that Solomon is provisioned as a service, enabled by an *LPaaS* engine, which allows (*i*) to easily integrate it with other pieces of software such as user interfaces (UIs) or mobile applications and (*ii*) to deploy it either to Cloud or Edge servers, depending on the usage context.

3.2 Model

Smart Environment. To model smart environments, we first build up a dictionary of all types of environmental parameters we can monitor (via sensors) and/or act upon (via actuators). We call *property types* the elements in such a dictionary, assuming they are declared as in

```
propertyType(TypeId).
```

where `TypeId` is a literal value denoting the unique property type identifier. Given a `propertyType` we can then define actuators and sensors that sense or operate on that.
Actuators are declared as in

```
actuator(ActuatorId, TypeId).
```

where `ActuatorId` is the unique actuator identifier and `TypeId` the associated property type.
Analogously, sensors are declared as in:

```
sensor(SensorId, TypeId).
```

where `SensorId` is the unique sensor identifier and `TypeId` is the associated property type. Environmental values monitored by each sensor are denoted by

```
sensorValue(SensorId, Value).
```

where `SensorId` identifies the sensor and `Value` is the last value it read.
Example. Based on the above, the shared room of the *Smart Home* example of Sect. 2 can be declared as in

```
propertyType(light).          propertyType(temp).
sensor(brightness,  light).   sensorValue(brightness, 20).
sensor(temperature, temp).    sensorValue(temperature, 22).
actuator(smallLight, light).  actuator(mainLight, light).
actuator(cornerLight, light). actuator(ac, temp).
```

where two sensors measure two different property types (i.e. temperature and brightness), having three lamps that can act on brightness and the AC system capable of changing the temperature. Please note that the current temperature settles at 22°C and the brightness at 20 out of 255. ◇
System administrators can divide smart environments into different *zones*, which allow distinguishing which global policy to apply to specific sets of sensors and actuators:

```
zone(ZoneId, MediationPolicy).
```

where `ZoneId` is the unique zone identifier and `MediationPolicy` is the unique identifier of the global management policy the zone is subject to. A zone groups one or more *property instances*, defining a set of actuators and a set of sensors that operate on a specific property type. A property instance is declared as in

```
propertyInstance(ZId, PIId, TypeId, Actuators, Sensors).
```

where `ZId` identifies the zone to which the instance belongs, `PIId` is the property instance identifier, `TypeId` is the `propertyType` of `PIId`, `Actuators` is a list of actuators that operate on the property and `Sensors` is a list of sensors that monitor it within the zone. All actuators and sensors in a given property instance must have the same property type. The identifier of a property instance is unique only within the zone, allowing for distinct zones to have instances with the same identifier.

Example. The property instances of the *Smart Home* example can be described by declaring a single `livingroom` zone and, for instance, four property instances as in

```
zone(livingroom, _).
propertyInstance(livingroom, studyingLight, light, [cornerLight, mainLight], [brightness]).
propertyInstance(livingroom, movieLight, light, [cornerLight, smallLight], [brightness]).
propertyInstance(livingroom, readingLight, light, [smallLight], [brightness]).
propertyInstance(livingroom, roomTemp, temp, [ac], [temperature]).
```

It is worth noting that the first three property instances all refer to the `light` property, grouping the brightness sensor with the lamps needed to realise different settings on such property, e.g. for studying (`cornerLight` and `mainLight`), watching a movie (`cornerLight` and `smallLight`), or reading a book (`smallLight` only). The last property instance refers instead to the `temp` property, grouping to the `temperature` sensor and the A/C system (i.e. `ac`). $\diamond$

Users and Goals. A *user* is declared as in

```
user(UserId, AllowedZones).
```

where `UserId` is the unique user identifier and `AllowedZones` is the list of the zones on which the user can set goals. User *goals* are declared as in

```
set(UId, ZId, PIId, Value).
```

where `UId` is the user identifier, `ZId` identifies a zone, `PIId` is one of the property instances of the zone, and `Value` is the goal expressed by the user on the property instance.

Example. Still following the *Smart Home* scenario, Alice and Bob, and their goals on brightness and temperature are represented as per

```
user(alice, [livingroom]).                user(bob, [livingroom]).
set(alice, livingroom, movieLight, 20).   set(bob, livingroom, studyingLight, 80).
set(alice, livingroom, roomTemp, 20).     set(bob, livingroom, roomTemp, 26).
```

Alice aims at setting the `movieLight` property instance to 20 out of 255 and the `roomTemp` to 20°C. Bob, on the contrary, wants to set the `studyingLight` property instance to 80 out of 255, and the `roomTemp` to 26°C. $\diamond$

3.3 Reasoner

The model described up to now denotes the inputs that Solomon receives from the smart environment it manages as well as from its users. Fig. 3 lists the core code of Solomon, which works in three main steps that constitute the top-down methodology of the proposed framework to determine a target state for a smart environment. Those steps are as follows:

1. it *collects all user requests*[3] that are currently submitted to the system (`getRequests/2, line 2`) and extracts only those that are valid,
2. it *mediates requests* referring to the same property instance by applying the mediation policies specified by the system administrator, so to determine a target state for each property instance (`mediateRequests/2, line 3`) by solving all user-user and user-admin conflicts,
3. it finally *determines actions* (i.e. settings) for individual IoT actuators so to achieve the target state, by also resolving possible conflicting actions found for a single actuator (`associateActions/2`, line 4).

```
1  react(Requests, MediatedRequests, Actions) :-
2     getRequests(Requests, ValidRequests),
3     mediateRequests(ValidRequests, MediatedRequests), validMediation(MediatedRequests),
4     associateActions(MediatedRequests, Actions), validActions(Actions).

5  getRequests(Requests, ValidRequests) :-
6     findall((ZId, PIId, Value, UId), set(UId, ZId, PIId, Value), Requests),
7     findall( (ZId, PIId, Value, UId),
8              ( member((ZId, PIId, Value, UId), Requests), user(UId, Zones), member(ZId ,Zones),
9                validRequest(ZId, PIId, Value) ), ValidRequests).

10 validMediation(Reqs) :-
11    sort(Reqs, OrderedReqs),
12    \+( ( member((Z,PI,V1), OrderedReqs), member((Z,PI,V2), OrderedReqs), dif(V1,V2) ) ),
13    \+( ( member((Z,PI,V), OrderedReqs), \+( validRequest(Z,PI,V) ) ) ).

14 validActions(Actions) :-
15    sort(Actions, OrderedActions),
16    \+( ( member((A,V1), OrderedActions), member((A,V2), OrderedActions), dif(V1,V2) ) ),
17    \+( ( member((A,V), OrderedActions), \+( validValue(A,V) ) ) ).
```

Fig. 3. Solomon code.

Overall, the `react/3` predicate (line 1) returns three lists: the list of all `Requests`, the list of `MediatedRequests` containing the target states for each property instance and the list of `Actions` to perform to reach a final target state. It is worth noting that, while the framework leaves complete flexibility to the system administrators in defining their own mediation policies, it also checks that inputs and outputs of each phase are well-formed (through predicates `validMediation/1`, line 3, and `validActions/1`, line 4). This guides the system administrators in their task of writing (formally) valid mediation policies.

[3] `findall(Template, Goal, Result)` finds all successful solutions of `Goal` and collects the corresponding instantiations of `Template` in the list `Result`. If `Goal` has no solutions then `Result` is instantiated to the empty list.

Collecting Requests. First, Solomon collects all the requests through getRequests/2 (line 2, lines 5–9), which determines two lists of tuples (ZId, PIId, Value, UId), where each tuple corresponds to a set(UId, ZId, PIId, Value) with arguments rearranged for easier handling in later stages. The first list Requests contains all current requests from users (line 6). The second one, ValidRequests, only contains valid requests (line 7–9), i.e. by default[4], requests for which the zone and the property instance exist, and the zone is among those the user associated with the request can set goals on.

Example. In the *Smart Home* scenario, querying

```
?- getRequests(Requests, ValidRequests).
```

returns the following:

```
Requests = ValidRequests,
ValidRequests = [(livingroom, movieLight, 20, alice), (livingroom, studyingLight, 80, bob),
                 (livingroom, roomTemp, 20, alice), (livingroom, roomTemp, 26, bob)].
```

collecting all requests from Alice and Bob. ◇

Mediating Requests. Valid requests are then passed to the mediateRequest/3 predicate (line 3) which can be flexibly and freely specified by the system administrator. The objective of this phase is to mediate between the possible conflicting goals of the users by determining one target value for each property instance. The mediateRequests/2 predicate outputs a list, MediatedRequests, of such values for each property instance, in the form of triples (ZoneId, PropertyInstanceId, Value). Then, the validMediation/1 predicate (lines 3, 10–13) checks that the list contains no duplicates (line 12) and that all requests are still valid after mediation (line 13).

Example. In our *Smart Home* scenario, a possible mediateRequests/2 that simply averages user requests for a same property instance is as follows:

```
mediateRequests(Requests, Mediated) :-
    groupPerPI(Requests, NewRequests),
    mediateRequest(NewRequests, Mediated).

mediateRequest([],[]).
mediateRequest([(Z,PI,Rs)|Reqs], [Mediated|OtherMedReqs]) :-
    mediatePI(Z,PI,Rs,Mediated),
    mediateRequest(Reqs, OtherMedReqs).

mediatePI(Z, PI, Ls, (Z, PI, Avg)) :-
    findall(V, member((V,_),Ls), Values), avg(Values,Avg).
```

Input Requests are first grouped per property instance by groupPerPI/2, which returns a list of triples (Z,PI,Rs) where Z and PI identify a property instance and Rs is the list of requests that target it. By recursively scanning such list, mediateRequest/2 exploits mediatePI/4 to average all requests grouped for each property instance.

By querying mediateRequests/4, we obtain

[4] System administrators can easily extend the concept of *valid request* by including further checks based on domain-specific knowledge, e.g. on the range of allowed values for a given property. This can be done by extending the validRequest/3 predicate exploited by getRequests/2 (line 9).

```
Mediated = [(livingroom,movieLight,20), (livingroom,roomTemp,23), (livingroom,studyingLight,80)].
```

which represents a target state where `movieLight` and `studyingLight` are set to 20 and 80 respectively, and `roomTemp` to 23°C, i.e. the average of Bob and Alice's goals. ◇

Determining Actions. After obtaining a target state for each property instance, Solomon generates a list of actions for available actuators to reach such a target. An action is a pair `(AId, Value)` where `AId` is the identifier of an actuator and `Value` is the value it needs to be set to. The `associatedActions/2` predicate (line 4) inputs a list of mediated requests and returns a list of actions, according to *System Administrator* policies.

The `validActions/1` predicate (lines 4, 14–17) checks whether there are no duplicate settings (line 16) and that obtained values are valid according to *System Administrator* policies (line 17), which can check if the configuration for an actuator can factually be implemented, using `validValue/2`.

Example. A simple policy that computes the setting for each actuator by dividing the target value of a `propertyInstance` by the number of its actuators, is specified as in

```
associateActions(Requests, ExecutableActions) :-
    actionsFor(Requests, Actions),
    setActuators(Actions, ExecutableActions).

actionsFor([],[]).
actionsFor([(Z, PI, V)|Reqs], Actions) :-
    propertyInstance(Z, PI, _, Actuators, _),
    selectActionsForPI(Z, PI, V, Actuators, _, Actions1),
    actionsFor(Reqs, Actions2),
    append(Actions1, Actions2, Actions).

selectActionsForPI(_, _, V, Actuators, _, Actions) :-
    length(Actuators, L),triggerAll(V, L, Actuators, Actions).

triggerAll(_, _, [], []).
triggerAll(V, L, [A|Actuators], [(A,VNew)|Actions]) :-
    VNew is V/L, triggerAll(V, L, Actuators, Actions).

setActuators(Actions, ExecutableActions) :-
    setActuatorsWithMax(Actions, 0, 100, ExecutableActions).
```

First, for each input `requests`, a triple `(Zone, PropertyInstance, TargetValue)`, `actionsFor/2` gets the list of actuators of that specific `propertyInstance`. Then, it `selectActionsForPI/6` computes the list of actions to be performed by dividing the target value for each `propertyInstance` by the number of its actuators. Note that when an actuator belongs to more than one `propertyInstances`, `setActuators/2` selects the highest value available cutting that value with a lower bound of 0 and an upper bound of 100.

By querying `associateActions/2` in the *Smart Home* scenario, given the target state of the previous example, we obtain:

```
?- associateActions(Mediated, Actions).
Actions = [(ac, 23), (cornerLight, 40), (mainLight, 40), (smallLight, 10)]
```

Note that the `ac` actuator is set to 23°C, i.e. the value of the target state. As for `movieLight` and `studyingLight`, being composed of several actuators, a further mediation happens. The target value of 20 for `movieLight` is split across `cornerLight` and `smallLight`, setting each to 10. Analogously, the target value of 80 for the `studyingLight` is split across `mainLight` and `cornerLight`, setting each to 40. The conflict on `cornerLight`, being in both property instances, is solved by picking the maximum between 10 and 40, viz. 40. ◇

4 Smart Building Example Retaken

In this section, we exploit Solomon to answer the questions raised about the *Smart Building* scenario of Sect. 2. The answer to **Q1** is obtained by specifying different `zone(ZoneId, MediationPolicy)` facts for the rooms in the smart building (Fig. 1), as in

```
zone(room_E_1, east).    zone(room_W_1, west).    zone(room_E_2, east).    zone(room_W_2, west).
zone(room_E_3, east).    zone(room_W_3, west).    zone(room_E_4, east).    zone(room_W_4, west).
zone(commonRoom_E, east).    zone(commonRoom_W, west).
```

The `east` and `west` literals identify two different mediation policies, specified by *System Administrator*, to be applied to the property instances grouped under the zone. Such grouping can be obtained by specifying suitable `propertyInstance(ZId, PIId, TypeId, Actuators, Sensors)` facts as, for instance, in

```
propertyInstance(room_E_1,roomTemp,temp,[acOdd_E,heater],[tempOdd_E]).
propertyInstance(room_E_1,roomLight,light,[biglightRoom_E_1,smalllightRoom_E_1], [lightRoom_E_1]).
propertyInstance(room_E_3,roomTemp,temp,[acOdd_E],[tempOdd_E]).
propertyInstance(room_E_3,roomLight,light,[biglightRoom_E_3,smalllightRoom_E_3],[lightRoom_E_3]).
```

that describes the sensors and actuators available in the *Room 1* and *Room 3* of the East wing. Note that the two rooms share the `acOdd_E` actuator for the A/C system and that *Room 1* contains the `heater` actuator that is not available in *Room 3*.

Based on the knowledge representation above, we can now answer **Q2** by suitable implementations of `mediateRequests/2`. Indeed, the *System Administrator* can easily declare mediation policies to solve user-user and user-admin conflicts in a context-aware manner. Such behaviour can be obtained through predicate `mediatePI/4` (which is used by `mediateRequests/2` as illustrated in Sect. 3) :

```
mediatePI(Z, PI, Ls, (Z, PI, Avg)) :-
    findall(V, member((V,_),Ls), Values),
    avg(Values,AvgTmp),
    zone(Z, MediationPolicy),
    propertyInstance(Z, PI, Prop, _, [Sensor]),
    sensorValue(Sensor, SensedValue),
    findValue(Policy, Prop, SensedValue, AvgTmp, Avg).
```

First `mediatePI/4` averages all user requests for a specific `propertyInstance` so to mediate possible user-user conflicts. Then, it exploit `findValue/4` to mediate between the obtained average with the global policy enforce by the *System Administrator*. Such mediation is based on the `MediationPolicy` of the wing (i.e.

`east`, `west`), on the property type `Prop` (i.e. `light`, `temp`) on the value obtained by the sensor of that instance (i.e. `SensedValue`) and on the computed average value `AvgTmp`.

Fig. 4 lists the code of predicate `findValue/4`. The first clause of `findValue/4` (lines 1–4), manages the temperature in both wings in the same way. After determining the current season (line 2), it enforces that the target value is within the season-dependent ranges specified for sustainability purposes (line 3–4), viz. 18–22°C in Winter and Autumn and 28–24°C in Summer and Spring. The second and the third clauses of `findValue/4` (lines 5–6 and 7–9) manage instead the environmental brightness, depending on the wing of the room, on the current brightness and the weather. This process determines a mediated target state that reconciles all user-user and user-admin conflicts on each property instance, which fully answers **Q2**.

```
1 findValue(_, temp, _, SVal, Value) :-
2    season(S),
3    (((S = winter ; S = autumn), (SVal > 22, Value is 22; SVal < 18, Value is 18; Value is SVal));
4    ((S = summer ; S = spring), (SVal > 28, Value is 28; SVal < 24, Value is 24; Value is SVal))).

5 findValue(east, light, _, SVal, Value) :-
6    (SVal > 255, Value is 255; SVal < 100, Value is 100; Value is SVal).

7 findValue(west, light, Brightness, SVal, Value) :-
8    ((Brightness > 100, (SVal > 255, Value is 255; SVal < 100, Value is 100; Value is SVal));
9    (SVal > 255, Value is 255; SVal < 180, Value is 180; Value is SVal)).
```

Fig. 4. `findValue/4` implements global policies in the *Smart Building*.

Finally, the answer to **Q3** is achieved through the implementation of the `associateActions/2` predicate. In our *Smart Building*, the policy we chose to adopt consists of dividing the workload equally between the various actuators, with the only exception of the heater that only accepts two values, viz. 0 or 100. The code is similar to the one proposed in the previous section for the *Smart Home*, in which the `selectActionsForPI/6` is adapted to the new policy and in case of multiple requests to the same actuator, now the maximum value is chosen.

```
selectActionsForPI(_, _, V, Actuators, _, Actions):-
    length(Actuators, L),triggerAll(V, L, Actuators, Actions).

triggerAll(_, _, [], []).
triggerAll(V, L, [A|Actuators], [(A,VNew)|Actions]):-
    dif(A, heater),
    VNew is V/L, triggerAll(V, L, Actuators, Actions).
triggerAll(V, L, [heater|Actuators], [(heater,100)|Actions]):-
    V > 0, triggerAll(V, L, Actuators, Actions).
triggerAll(V, L, [heater|Actuators], [(heater,0)|Actions]):-
    V =< 0, triggerAll(V, L, Actuators, Actions).

setActuators(Actions, ExecutableActions) :-
    setActuatorsWithMin(Actions, -inf,inf, ExecutableActions).
```

First `selectActionsForPI/6` computes the number `L` of actuators of the `propertyInstance`, then `triggerAll/4` is called which distributes the workload to the actuators with the exception of the `heater`. Finally, `setActuators` choose the maximum

in case of multiple sets for a specific actuator (with no lower or upper bound). With this process we can determine the correct configuration for each actuator acting on that state, and mediate between any conflicting configurations that a single actuator possibly receives, answering to **Q3**.

We conclude this section by describing a use case for the scenario above exploiting the policies described. Suppose that it is a sunny day in winter, with the brightness value sensed in the West common room at 160 out of 255, and user u1 sets the brightness of her room to 0 and the temperature to 18°C. On the contrary, user 3 sets the temperature at 28°. Assume that the two rooms share the A/C system but not the lighting system. Meanwhile, user u4 sets the temperature and brightness of room2, which she is not authorised to handle. Finally, users u2 and u8 are both in the commonRoom_E with the same goal for the light but different goals for the temperature (respectively 23°C and 18°C).

```
season(winter).
user(u1, [room_E_1,commonRoom_E,commonRoom_W]).
user(u3, [room_E_3,commonRoom_E,commonRoom_W]).
user(u8, [room_W_4,commonRoom_E,commonRoom_W]).
set(u1, room_E_1, roomLight, 0).
set(u1, room_E_1, roomTemp, 18).
set(u3, room_E_3, roomTemp, 28).
set(u4, room_E_2, roomLight, 0).
set(u4, room_E_2, roomTemp, 18).
sensorValue(lightCommonRoom_W, 160).
user(u2, [room_E_2,commonRoom_E,commonRoom_W]).
user(u4, [room_E_4,commonRoom_E,commonRoom_W]).

set(u2, commonRoom_W, commonRoomLight, 255).
set(u2, commonRoom_W, commonRoomTemp, 23).
set(u8, commonRoom_W, commonRoomLight, 255).
set(u8, commonRoom_W, commonRoomTemp, 18).
```

For each room the result of the mediation phase with the application of global policies described before is:

```
[(room_E_1,roomLight,100), (room_E_1,roomTemp,18), (room_E_3,roomTemp,22),
(commonRoom_W,commonRoomLight,255), (commonRoom_W,commonRoomTemp,20.5)]
```

where roomLight of room_E_1 is bounded on 100 because it is the minimum bound for the light in the East wing, while the roomTemp is within the bounds. Meanwhile, roomTemp of room_E_3 is bounded to 22 the maximum temperature allowed in winter. Instead in commonRoom_W the temperature is the average of the two requests because it is within the boundaries and also the light because we are in the West wing and it is sunny (brightness > 100) so the maximum bound is 255, the goal of both users. Finally, the goals of user u4 are ignored because is not authorised to interact with room_E_2. Then, the actions to be carried out, given the states computed, will be:

```
[(acCommonRoom_W, 20.5), (biglightCommonRoom_W_1, 127.5),(biglightCommonRoom_W_2, 127.5),
(acOdd_E, 22), (heater, 100), (biglightRoom_E_1, 50), (smalllightRoom_E_1, 50)]
```

where the temperature of commonRoom_W is managed only by acCommonRoom_W and the light is implemented by two main light which equally divide the goal. Meanwhile, acOdd_E is the air conditioning system shared by room 1 and 3 and is setted to the maximum of the two goals (18 and 22) and also in room_E_1 the heater is working. Finally, the small and big light work together to implement the goal.

5 Related Work

In this section, we discuss some closely related work on the self-management of smart environments. Most of these works fall within three main categories, viz. *goal-oriented* [23], *hierarchical* [15], and *neural and fuzzy* [22].

First, [3, 29, 32, 33] and [30] propose goal-oriented approaches to conflict mediation. Targeting global goals like energy efficiency, users comfort, and system security, [3] presents a solution to manage smart buildings by adding a semantic layer on top of the stack of IoT devices for reaching the desired global goals, exploiting an ontology of goal types. With a more formal approach, [29] devise a methodology for autonomic device management describing the evolution of a smart environment as the set of evolutions of single device states, modelled as command sequences. Given a global goal, this solution determines the correct sequence of commands to reach it. Besides, [33] proposes an access control mechanism exploiting a priority-based policy negotiation technique to solve user-user conflicts in a smart home, made of multiple devices. Finally, Tartarus [30], is a Prolog platform designed to integrate cyber-physical systems and robots, supporting mobility, cloning, and payload carrying. More in general, [32] propose a solution for the problem of conflicts resolution in a multi-agent system, through argumentation-based reasoning. Similarly, [13] propose a multi-agent framework to contextually compose existing web services in Smart Envirnonments via reasoning and learning. As per its goal-oriented nature, Solomon enables system administrators to write customised policies that can accommodate sophisticated and expressive mediation policies exploiting for example the semantic ontology of [3] or the negotiation technique of [33].

Second, hierarchical solutions for goal mediation have been proposed by [15] and [20]. Dynamic hierarchical goal management for different IoT systems is discussed in [15], considering conflicting local and global goals, and the availability of limited resources that can vary at runtime. Regarding security in smart environments and in particular *Smart Offices*, [20] propose a hierarchical, agent-based solution that considers the high number of potential users, their security roles and the heterogeneity of devices and spaces. An interesting extension to Solomon is to include hierarchical approaches to solve goals and to consider security aspects as well.

Third, and last, fuzzy logic [10, 24, 28] and neural network [4, 17] approaches to goal mediation have been studied recently, along with their combination [16]. Fuzzy logic can be used for context-awareness in Smart Home as illustrated in [24], where raw data from the sensors are processed to manages actuators according to the computed context based on the user movement and activity. The works in [10, 28] propose expert systems to control the A/C of smart buildings, based on the current status of the sensors and the outside temperature. Similarly, [16] manages A/C systems through a neuro-fuzzy controller where an adaptive neural network is used to better tuning the fuzzy rules, making them more robust. Neural networks have also been successfully used to predict energy consumption more reliably than traditional techniques [17] and for indoor temperature forecasting [4]. As Prolog is well-suited to implement fuzzy logic [27], an

interesting extension of Solomon is to accommodate fuzzy controllers. Similarly, predictions based on neural networks can be made available in the knowledge base of Solomon from external services or by relying on recent implementations of Prolog that support neural networks (e.g. DeepProbLog [19]).

6 Concluding Remarks

This article proposed a declarative framework – and its Prolog prototype Solomon– to specify policies for mediating contrasting (user and/or global) goals and actuator settings in smart environments. The prototype is provisioned as a service through *LPaaS*, and it can resolve user-user and user-admin conflicts into a target state for the smart environment with actuator settings.

The wide variety of smart environments and the desiderata of their users and system administrators calls for new frameworks to easily develop and continuously adapt domain-specific mediation policies. This work moves some first steps towards this direction, aiming at contributing a novel declarative approach, enabled by *LPaaS*, to the field of goal-driven management of smart environments. As showcased in our example, thanks to its declarative nature, Solomon features a suitable level of *abstraction* and *flexibility* to accommodate different needs of smart environments, making it easy to express, maintain and update mediation policies as per the ever-changing needs of IoT scenarios.

In our future work, we intend to:

- *New Policies and Data.* Implement other mediation policies (e.g. based on fuzzy logic, learning or heuristics), by also proposing a set of *building blocks* that *System Administrators* can use to compose their own policies, and by enabling users' geolocalisation to predict movements and preferences.
- *Answer Set Programming.* Extend Solomon's API to support Answer Set Programming (ASP) as in [11] so to allow processing more expressive policies.
- *Web of Things.* Integrate Solomon with Web of Things to make it more interoperable and easier to exploit in existing smart environments.

References

1. IFTTT: If This Then That. https://ifttt.com/
2. Amazon: What Is Alexa? https://developer.amazon.com/en-US/alexa
3. Andrushevich, A., et al.: Towards semantic buildings: Goal-driven approach for building automation service allocation and control. In: ETFA 2010. pp. 1–6 (2010)
4. Attoue, N., et al.: Smart building: Use of the artificial neural network approach for indoor temperature forecasting. Energies **11**(2), 395 (2018)
5. Becker, C., et al.: Pervasive computing middleware: current trends and emerging challenges. CCF Trans. Pervasive Comput. Interact. 1 **1**, 10–23 (2019)
6. Bisicchia, G., Forti, S., Brogi, A.: Declarative goal mediation in smart environments. In: SMARTCOMP, *Work in Progress Track* (2021)
7. Booy, D., Liu, K., Qiao, B., Guy, C.: A Semiotic Multi-Agent System for Intelligent Building Control. AMBI-SYS (2008)
8. Calegari, R., Denti, E., Mariani, S., Omicini, A.: Logic Programming as a Service (LPaaS): Intelligence for the IoT. In: ICNSC. pp. 72–77 (2017)
9. Calegari, R., Denti, E., Mariani, S., Omicini, A.: Logic programming as a service. Theory and Pract. Log. Program. **18**(5-6), 846873 (2018)

10. Calvino, F., et al.: The control of indoor thermal comfort conditions: introducing a fuzzy adaptive controller. Energy & Buildings **36**(2), 97 – 102 (2004)
11. Catalano, G., et al.: A rest-based development framework for ASP: tools and application. In: PADL 2018. LNCS, vol. 10702, pp. 161–169. Springer (2018)
12. Davidsson, P., Boman, M.: Saving Energy and Providing Value Added Services in Intelligent Buildings: A MAS Approach. In: ASA/MA. pp. 166–177 (2000)
13. Ferilli, S., et al.: An agent architecture for adaptive supervision and control of smart environments. In: 2015 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS). pp. 1–8 (2015)
14. Gyrard, A., Sheth, A.: IAMHAPPY: Towards an IoT knowledge-based cross-domain well-being recommendation system for everyday happiness. Smart Health **15**, 100083 (2020)
15. Jantsch et al., A.: Hierarchical dynamic goal management for IoT systems. In: ISQED. pp. 370–375 (2018)
16. Jian, W., Wenjian, C.: Development of an adaptive neuro-fuzzy method for supply air pressure control in HVAC system. In: SMC. pp. 3806–3809 (2000)
17. Kumar, R., Aggarwal, R., Sharma, J.: Energy analysis of a building using artificial neural network: A review. Energy & Buildings pp. 352–358 (2013)
18. Lee, S.K., Bae, M., Kim, H.: Future of IoT Networks: A Survey. Applied Sciences **7** (2017)
19. Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., Raedt, L.D.: DeepProbLog: Neural Probabilistic Logic Programming. arXiv 1907.08194 (2019)
20. Marsá-Maestre, I., De La Hoz, E., Alarcos, B., Velasco, J.R.: A Hierarchical, Agent-based Approach to Security in Smart Offices. In: ICUC (2006)
21. Merabti, S., Draoui, B., Bounaama, F.: A review of control systems for energy and comfort management in buildings. In: ICMIC. pp. 478–486 (2016)
22. Naji, S., et al.: Application of adaptive neuro-fuzzy methodology for estimating building energy consumption. Renew. and Sustain. En. Reviews **53**, 1520–1528 (2016)
23. Palanca, J., et al.: Designing a goal-oriented smart-home environment. Inf. Syst. Frontiers **20**(1), 125–142 (2018)
24. Patel, A., Champaneria, T.A.: Fuzzy logic based algorithm for Context Awareness in IoT for Smart home environment. In: TENCON. pp. 1057–1060 (2016)
25. Perumal, T., et al.: Rule-based conflict resolution framework for Internet of Things device management in smart home environment. In: GCC. pp. 1–2 (2016)
26. Perwej, Y., AbouGhaly, M.A., Kerim, B., Harb, H.A.M.: An extended review on internet of things (iot) and its promising applications. CAE pp. 2394–4714 (2019)
27. Rubio-Manzano, C., Iranzo, P.J.: A Fuzzy Linguistic Prolog and its Applications. J. Intell. Fuzzy Syst. **26(3)**(3), 1503–1516 (2014)
28. Salih, A.: Fuzzy Expert Systems to Control the Heating, Ventilating and Air Conditioning (HVAC) Systems. IJERT **4** (2015)
29. Sanaullah, M., Corno, F., Razzak, F.: Autonomic goal-oriented device management for Smart Environments. J. Ambient Intell. Smart Environ. **7**(4), 425–448 (2015)
30. Semwal, T., et al.: Tartarus: a multi-agent platform for integrating cyber-physical systems and robots. In: AIR. pp. 20:1–20:6. ACM (2015)
31. Sfar, H., Raddaoui, B., Bouzeghoub, A.: Reasoning Under Conflicts in Smart Environment. In: ICONIP (3). pp. 924–934 (2017)
32. Shams, Z., et al.: Argumentation-Based Reasoning about Plans, Maintenance Goals, and Norms. ACM Trans. Auton. Adapt. Syst. **14**(3), 9:1–9:39 (2020)
33. Sikder, A.K., et al.: Kratos: Multi-User Multi-Device-Aware Access Control System for the Smart Home. p. 112. WiSec '20 (2020)