Declarative Goal Mediation in Smart Environments

Giuseppe Bisicchia, Stefano Forti, Antonio Brogi
Department of Computer Science, University of Pisa, Pisa, Italy

***Abstract*—Smart environments enabled by the Internet of Things aim at improving our daily lives by automatically tuning ambient parameters 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 mediate between conflicting goals among different users and/or system administrators, nor across different IoT verticals. In this article, we propose a declarative approach (and its open-source Prolog prototype) to represent smart environments, user-set goals and customisable mediation policies to reconcile contrasting goals across multiple IoT systems.**

***Index Terms*—Goal-oriented systems, Smart Environments, Internet of Things, Logic Programming**

I. Introduction

The Internet of Things is continuously growing and becoming an integrated part of our daily lives with a plethora of new different applications (e.g. wearables, home appliances) [1], that show even capable of affecting our mood [2]. Among the new verticals the IoT is enabling, *smart environments* are getting increasing attention from the market and the research community [3]. Indeed, they empower self-managing ambients, based on data from IoT sensors and triggering reactions enabled by IoT actuators. Besides their high potential to improve people's routines, these applications also lead to a more sustainable energy and resource management [4]–[6].

Reconciling contrasting goals among different users in a smart environment represent a challenging problem [7]–[9]. Indeed, people sharing a room in a building can possibly express very different desiderata on the temperature and on the light intensity they prefer, also depending on the activity they are currently carrying on. To this end, many techniques have been proposed to reconcile such contrasting goals, e.g. via fuzzy logic [10], multi-agent systems [11], [12] or neural networks [13]. However, most commercial solutions, e.g. IFTTT or Amazon Alexa, only allow setting simple goals and do not mediate between contrasting objectives [14].

Factually, two different types of conflict can arise among local and global goals: different users can set different goals (e.g. on target temperature), and the System Administrators 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, a final configuration of the actuators involved must also be determined. Indeed, given a final target state, we need to determine the correct configuration for each actuator acting on that state and possibly mediate between any conflicting configurations that a single actuator possibly receives.

In this article, we propose a declarative methodology to specify customisable mediation policies for reconciling contrasting goals and actuator settings in smart environments. The novel contribution mainly consists of a declarative framework to specify mediation policies for reconciling contrasting (user and/or global) goals and actuator settings in smart environments, and a Prolog prototype implementation, Solomon. The prototype, provisioned as a service, tames the effects of the aforementioned types of conflicts 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 among users' goals, among users' and system administrator's goals, and on actuators configuration. Last, but not least, the declarative nature of Solomon makes it easy to write, maintain and extend arbitrary mediation policies across multiple IoT verticals.

II. Methodology and Prototype

In this section, we briefly illustrate Solomon, by describing its overall functioning and its open-source Prolog implementation[1]. More details on the methodology and prototype, with examples of usage, can be found in [15].

A. Overview

Fig. 1 gives a bird's-eye view of the architecture of Solomon, which can be deployed to feature the autonomic mediation of goals in smart environments. 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.

Both *users* and *system administrators* interact with Solomon, through the Logic-Programming-as-a-Service (*LPaaS*) API [16] 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 goals and solve possible user-admin conflicts, and to determine actuator configurations useful to reach a target state for the smart environment, after goal mediation. Being provisioned as a service, Solomon features (*i*) easy integration with other

Work partly supported by projects: *GIÒ* funded by the Department of Computer Science of the University of Pisa, Italy; *LiSCIo* (F4Fp-08-M30) funded by Fed4Fire+; *CONTWARE* funded by the Conference of Italian University Rectors; and by the *Orio Carlini Scholarship Programme 2020* funded by the GARR Consortium.

[1]Freely available at: `https://github.com/di-unipi-socc/Solomon`

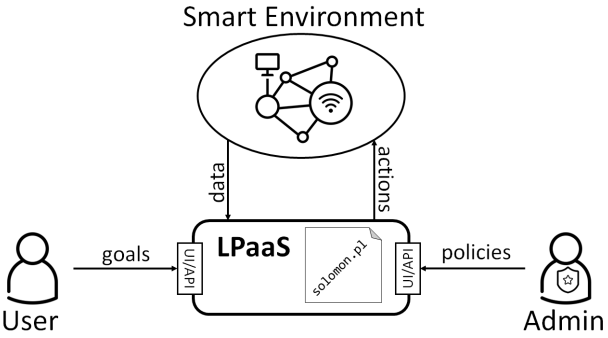

Fig. 1. Bird's-eye view of Solomon.

pieces of software such as user interfaces (UIs) or mobile applications and (*ii*) deployability to either Cloud or Edge servers, depending on its usage context.

B. 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, and declare them as in

```
propertyType(TypeId).
```

where `TypeId` is a literal value denoting an 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. 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. 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 ore 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.

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.

C. 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. 2 lists the kernel 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:

1) it *collects all user requests*[2] 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 2`) 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 3).

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 2, and `validActions/1`, line 3). This guides the system administrators in their task of specifying valid mediation policies.

Collecting Requests. First, Solomon collects all the requests through `getRequests/2` (line 2, lines 4–8), 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 5). The second one, `ValidRequests`, only contains valid requests (line 6–8), i.e. by default[3], requests for which the zone and the

[2] `findall(Template, Goal, Result)` finds all succesful 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.

[3] 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 7).

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

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

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

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

Fig. 2. Solomon code.

property instance exist, and the zone is among those the user associated with the request can set goals on.

Mediating Requests. Valid requests are then passed to the `mediateRequest/3` predicate (line 2) 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 2, 9–11) checks that the list contains no duplicates (line 10) and that all requests are still valid after mediation (line 11).

Determining Actions. After obtaining a target state for each property instance, Solomon generates a list of actions for available actuators to reach such target. An action is a pair `(AId, Value)` where `AId` is the identifier of an actuator and `Value` is the value it need to be set to. The `associatedActions/2` predicate (line 3) inputs a list of mediated requests and returns a list of actions, according to *Administrator* policies. The `validActions/1` predicate (lines 3, 12–14) checks whether there are no duplicate settings (line 13) and that obtained values are valid according to *Administrator* policies (line 14), which can check if the configuration for an actuator can factually be implemented, using `validValue/1`.

III. Concluding Remarks

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

In our future work, we intend to: (*i*) implement and test other mediation policies (e.g. based on fuzzy logic, learning or heuristics), (*ii*) embed a geo-localisation system for users to predict their movements so to reduce manual interactions, and (*iii*) make Solomon interoperable with *Web of Things*[4] to exploit it in actual smart environments.

[4] *Web of Things*, `https://www.w3.org/WoT/`

References

[1] Y. Perwej, M. A. AbouGhaly, B. Kerim, and H. A. M. Harb, "An extended review on internet of things (iot) and its promising applications," *CAE*, pp. 2394–4714, 2019.

[2] A. Gyrard and A. Sheth, "IAMHAPPY: Towards an IoT knowledge-based cross-domain well-being recommendation system for everyday happiness," *Smart Health*, vol. 15, p. 100083, 2020.

[3] A. K. Sikder, L. Babun, Z. B. Celik, A. Acar, H. Aksu, P. McDaniel, E. Kirda, and A. S. Uluagac, "Kratos: Multi-User Multi-Device-Aware Access Control System for the Smart Home," WiSec '20, p. 112, 2020.

[4] S. Merabti, B. Draoui, and F. Bounaama, "A review of control systems for energy and comfort management in buildings," in *ICMIC*, pp. 478–486, 2016.

[5] E. Torunski, R. Othman, M. Orozco, and A. E. Saddik, "A review of smart environments for energy savings," *ANT/MobiWIS*, vol. 10, pp. 205 – 214, 2012.

[6] S. Forti, A. Pagiaro, and A. Brogi, "Simulating fogdirector application management," *Simul. Model. Pract. Theory*, vol. 101, p. 102021, 2020.

[7] J. Palanca, E. Del Val, A. Garcia-Fornes, H. Billhardt, J. M. Corchado, and V. Julián, "Designing a goal-oriented smart-home environment," *Inf. Syst. Frontiers*, vol. 20, no. 1, pp. 125–142, 2018.

[8] A. Jantsch et al., "Hierarchical dynamic goal management for IoT systems," in *ISQED*, pp. 370–375, 2018.

[9] H. Sfar, B. Raddaoui, and A. Bouzeghoub, "Reasoning Under Conflicts in Smart Environment," in *ICONIP (3)*, pp. 924–934, 2017.

[10] A. Patel and T. A. Champaneria, "Fuzzy logic based algorithm for Context Awareness in IoT for Smart home environment," in *TENCON*, pp. 1057–1060, 2016.

[11] P. Davidsson and M. Boman, "Saving Energy and Providing Value Added Services in Intelligent Buildings: A MAS Approach," in *ASA/MA*, pp. 166–177, 2000.

[12] D. Booy, K. Liu, B. Qiao, and C. Guy, "A Semiotic Multi-Agent System for Intelligent Building Control," *AMBI-SYS*, 2008.

[13] R. Kumar, R. Aggarwal, and J. Sharma, "Energy analysis of a building using artificial neural network: A review," *Energy & Buildings*, pp. 352–358, 2013.

[14] C. Becker, C. Julien, P. Lalanda, and F. Zambonelli, "Pervasive computing middleware: current trends and emerging challenges," *CCF Trans. Pervasive Comput. Interact. 1*, vol. 1, pp. 10–23, 2019.

[15] G. Bisicchia, S. Forti, and A. Brogi, "A Declarative Goal-oriented Framework for Smart Environments with LPaaS," *arXiv preprint*, 2021. Available at: http://arxiv.org/abs/2106.13083.

[16] R. Calegari, E. Denti, S. Mariani, and A. Omicini, "Logic programming as a service," *TPLP*, vol. 18, no. 5-6, p. 846873, 2018.