Extending Task Automation Systems with Event-State-Condition-Action Capabilities

Giuseppe Desolda[1], Francesco Greco[1], Massimo Zancanaro[2,3] and Maria F. Costabile[1]

[1] *Computer Science Department, University of Bari Aldo Moro, Italy*
[2] *Department of Psychology and Cognitive Science, University of Trento, Italy*
[3] *Fondazione Bruno Kessler*

Abstract

Trigger-Action programming recently emerged as a paradigm for supporting end-users in defining the behavior of Internet-of-Things devices. It is often implemented by allowing users to define Event-Condition-Action rules visually. The possibility of using either states or events in triggers has already been discussed in the literature and their distinction may be difficult to understand for users. In this paper, we discuss how the definition of Event-State-Condition-Action (ESCA) rules to customize the behavior of IoT devices could be implemented in a Task Automation System (TAS), by adding a lay-er for monitoring events/states.

Keywords

End-User Development, Internet of Things, Trigger Action Programming

1. Introduction

Trigger-Action programming (TAP) is emerging as a paradigm for supporting end-users, particularly those without programming skills, in defining the behavior of *Internet-of-Things* devices and digital web services. TAP is a simplified form of the *Event Condition Action* (ECA), a common approach for rule-based systems, originally employed to manage databases [1] and control industrial processes [2]. However, when applied in the form of *Trigger-Action* rules, the *Condition* part is usually left out for the sake of simplicity, and the rules take the simple form of "*IF <a trigger occurs> THEN <an action is executed>*".

A source of complexity in the TAP paradigm derives from the fact that triggers can indicate both instantaneous events or states [3], and users are not always able to understand the difference between the two [4, 5]. One of the most important causes is the temporal aspect of triggers and actions [4, 5] since users are often confused when they have to distinguish triggers based on events (i.e., that occur in a specific moment in time) and states (i.e., that are true over a time span).

2. A middleware to Monitor Events and States

Task Automation Systems (TAS) [6] are web-based tools that support users in defining smart objects' behavior through visual interfaces. Among the most popular ones there are IFTTT [7], Zapier [8], and EFESTO-5W [9-11]. They typically support users in the definition of the smart object's behavior through the visual creation of Event-Condition-Action (ECA) rules. From a technical point of view, one of the most critical aspects of TASs is trigger monitoring. Let us consider the following ECA rule:

EMPATHY: Empowering People in Dealing with Internet of Things Ecosystems. Workshop co-located with INTERACT 2021, August 30, 2021, Bari, Italy
EMAIL: giuseppe.desolda@uniba.it (G. Desolda); francesco.greco@uniba.it (F. Greco); massimo.zancanaro@unitn.it (M. Zancanaro); maria.costabile@uniba.it (M. F. Costabile)
ORCID: 0000-0001-9894-2116 (G. Desolda); 0000-0003-2730-7697 (F. Greco); 0000-0002-1554-5703 (M. Zancanaro); 0000-0001-8554-0273 (M. F. Costabile)

IF the alarm is turned on
THEN switch off all the lights

Two main solutions are currently adopted by TASs to monitor the triggering events, and this depends on the technology implemented by the smart devices. The simplest and less effective solution consists of periodically invoking the smart object's API related to the triggering event. Considering the previous rule, a TAS must invoke the API of the alarm to check if it is turned on. This is a low-effective solution because of the overloading of connection and the overusing of device energy. A more efficient and effective solution is the use of a Publish-Subscribe architecture, such as the MQTT protocol. In this architecture, there are three main actors, i.e., the publisher, the subscribers, and the broker. The publisher is an entity, a smart object in our case, that sends a message to other subscribed entities. The subscriber is an entity, a TAS in our scenario, that receives the messages from the publisher. The broker is a virtual component hosted on a web server that implements the publish/subscribe mechanism: the publisher notifies the broker only when an event occurs (e.g., the alarm is turned on) and the subscribers are notified by the broker only when the event is triggered. An advantage of this architecture is that several entities can subscribe to the same publisher, in our case several rules can subscribe to the same event without overloading the smart object.

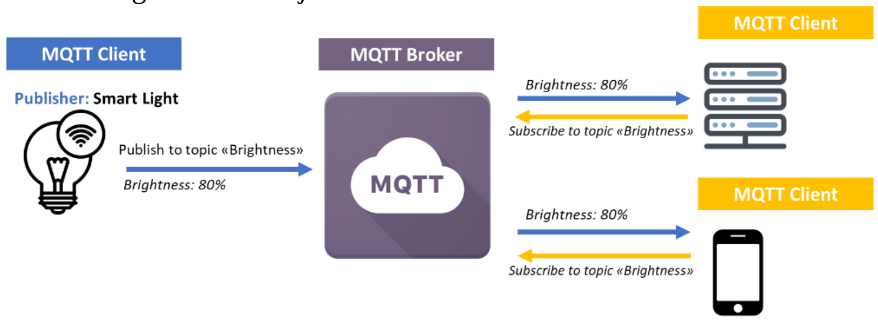

Figure 1. An example of MQTT Publish/Subscribe architecture

2.1. Generate states of a smart object starting from its events

The majority of the smart objects expose sensors data to third party-applications as (instantaneous) events. In order to allow TASs to monitor the states of those smart objects exposing only events, the proposed solution is to create a middleware in the TAS rule engine where, for each smart object, the internal representation of the device is extended to generate states starting from its events. As reported in the following code, the first time a smart object is connected with the TAS, there is an initialization phase: for each event *i*, its API is invoked and the variable $state_i$ is set according to a mapping between the event values and the state values, previously defined by the TAS administrator. For example, in the device Smart light, the API of the event "Is turned on" can return two values, i.e., true or false; these values are mapped in the values "Is on" and "Is off" that can be associated with the variable *state*.

Then, every n seconds, the event value is read through the API and the *state* variable is updated if differs from the previous value. The device states are made available to the TAS through MQTT APIs automatically generated so that, if the *state* value changes, all the subscribers are notified.

```
Initialization:
   Var $state_i$ = map($event_i$)
Loop:
   if (map($event_i$) != $state_i$) then:
     $state_i$ = map($event_i$);
   endif;
endLoop;
```

2.2. Generate the events of a smart object from its states

Although most of the smart objects expose their sensor data as events, in some cases, smart objects could provide API for monitoring only their states. Similar to the solution presented in the previous section, in the middleware the internal representation of the smart objects must be extended to provide its states as events. In this case, to generate the *event*$_i$ starting from the *state*$_i$, a specific component in the middleware has to continuously query the API of *state*$_i$ and, if the state value is changed with respect to the last query, the *event*$_i$ is triggered. For example, in the smart device Thermostat, the state "Temperature hot" would be associated with the event "Temperature goes above 25°C"; if the state changes from False to True (e.g., if the temperature goes from 24°C to 26°C), then the event is triggered. This behavior is summarized more formally in the following pseudocode:

```
Initialization:
    Var event$_i$ = map(state$_i$)
Loop:
  new_state$_i$ = API_state(i)
  if (state$_i$ != new_state$_i$) then:
    trigger(event$_i$);
       state$_i$ <- new_state$_i$;
  endif;
endLoop;
```

3. Conclusions

In this position paper, it has been proposed an approach to enrich a TAP architecture with coordinated states and events for a more expressive definition of ECA rules. The main goal is to inform designers and developers of TASs of the possibilities offered by the Event-State-Condition-Action paradigm and on the possible technical solutions to implement it in their TAS.

Acknowledgements

This work is partially supported by the Italian Ministry of University and Research (MIUR) under grant PRIN 2017 "EMPATHY: EMpowering People in deAling with internet of THings ecosYstems.".

References

[1] G. Ghiani, M. Manca, F. Paternò, and C. Santoro, "Personalization of Context-Dependent Applications Through Trigger-Action Rules," *ACM Transaction on Computer-Human Interaction*, vol. 24, no. 2, p. 33 pages, 2017, Art no. Article 14 (April 2017), doi: 10.1145/3057861.

[2] B. Joonsoo, B. Hyerim, K. Suk-Ho, and K. Yeongho, "Automatic control of workflow processes using ECA rules," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 8, pp. 1010-1023, 2004, doi: 10.1109/TKDE.2004.20.

[3] W. Brackenbury *et al.*, "How Users Interpret Bugs in Trigger-Action Programming," in *Human Factors in Computing Systems*, Glasgow, Scotland Uk, 2019: Association for Computing Machinery, p. Paper 552, doi: 10.1145/3290605.3300782. [Online]. Available: https://doi.org/10.1145/3290605.3300782

[4] B. Ur *et al.*, "Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes," in *SIGCHI Conference on Human Factors in Computing Systems*, San Jose, California, USA, 2016, New York, NY, USA: ACM, pp. 3227-3231, doi: 10.1145/2858036.2858556.

[5] J. Huang and M. Cakmak, "Supporting mental model accuracy in trigger-action programming," presented at the ACM International Joint Conference on Pervasive and Ubiquitous Computing, Osaka, Japan, 2015. [Online]. Available: https://doi.org/10.1145/2750858.2805830.

[6] M. Coronado and C. A. Iglesias, "Task Automation Services: Automation for the Masses," *IEEE Internet Computing,* vol. 20, no. 1, pp. 52-58, 2016, doi: 10.1109/MIC.2015.73.

[7] IFTTT Inc. "IFTTT." https://ifttt.com/ (accessed June 1, 2021).

[8] Zapier Inc. "Zapier." https://zapier.com/ (accessed May 9, 2021).

[9] G. Desolda, C. Ardito, and M. Matera, "Empowering end users to customize their smart environments: model, composition paradigms and domain-specific tools," *ACM Transactions on Computer-Human Interaction,* vol. 24, no. 2, p. 52 pages, 2017, Art no. Article 12 (April 2017), doi: 10.1145/3057859.

[10] C. Ardito, G. Desolda, R. Lanzilotti, A. Malizia, and M. Matera, "Analysing Trade-offs in Frameworks for the Design of Smart Environments," *Behaviour & Information Technology,* vol. 39, no. 1, pp. 47-71, 2019, doi: 10.1080/0144929X.2019.1634760.

[11] C. Ardito *et al.*, "User-defined semantics for the design of IoT systems enabling smart interactive experiences," *Personal and Ubiquitous Computing,* vol. 24, no. 6, pp. 781-796, 2020/12/01 2020, doi: 10.1007/s00779-020-01457-5.