An Ontology-Based Approach for Setting Security Policies in Smart Homes

An Ontology-based Approach For Setting Security Policies in Smart Homes

Alberto Monge Roffarello[1] and Luigi De Russis[1]

Politecnico di Torino, Corso Duca degli Abruzzi, 24 Torino, Italy 10129
{alberto.monge,luigi.derussis}@polito.it

Abstract. To preserve the security and the integrity of smart home environments, a smart home system should provide end users with mechanisms to define security-based policies on their devices and services without the need to know (and specify) details that strongly depend on the underlying technology. To this end, this paper presents an End-User Development tool that allows users to a) define high-level security policies like *"do not record any sound in the living room tonight, "* b) check and debug high-level security policies against inconsistencies and redundancies, and c) translate high-level security policies into device-specific policies that can be applied at run-time. The tool implements a trigger-action programming paradigm, and it exploits a hybrid formalism based on ontologies and Petri Networks.

Keywords: End-User Development · Internet of Things · Trigger-Action Programming · High-Level Policies.

1 Introduction

The Internet of Things (IoT) is the paradigm whereby everyday objects are no longer disconnected from the virtual world, but they can be controlled remotely and serve as an access point to the Internet [30]. The advent of the IoT already helps society in many ways through applications ranging in scope from the individual to the planet [17]. People, in particular, can nowadays interact with a multitude of IoT devices in their homes: with lamps, thermostats, and many other appliances, including fridges and ovens, that can be connected to the Internet, homes are becoming "smart." Besides physical devices, many different online services, ranging from social networks to news and messaging apps, are greatly used by almost everyone: the number of people using the Internet passed 4.5 billion marks in January 2020, with more than 3.8 billion people actively using social media [34]. As a result, users can easily access a complex network of connected entities, be they smart devices or online services, that can communicate with each other, humans, and the environment.

The complexity of the IoT poses several security challenges, especially in the smart home context. Errors in automated behaviors, for example, can lead to unpredictable and dangerous behaviors [14]. While posting content on a social

network twice could be considered a trivial issue, wrong automation could unexpectedly unlock the main door of a house, thus generating a security threat. To preserve the security and the integrity of smart home environments, a smart home system should provide end users with mechanisms to define security-based high-level policies on their devices and services, without the need to know (and specify) details that strongly depend on the underlying technology. Following this need, this work proposes the Policy Translation Point (PTP) system, an end-user development tool that aims to support users to express high-level policies like *"Do not record sound in the living room tonight."* To this end, PTP uses an ontological representation for end-user development and employs a trigger-action programming paradigm through which high-level security policies are expressed as abstract trigger-action rules. These policies ultimately ensure that the behavior of the devices and applications involved in a given smart home adheres to the latest underlying policy description. In particular, PTP can translate high-level policies into device-level policies, when possible. Stemming from a high-level policy, for instance, the system could limit the features of a smart home device or inhibit the operation of a non-reconfigurable device. In addition, it could verify whether a given home configuration is compatible with one or more active (or suggested) policies. Besides empowering users to define and translate rules, PTP is also able to detect potential conflicts between high-level policies, namely redundancies (i.e., policies that produce equal or overlapping results) and inconsistencies (i.e., policies with contradictory actions).

2 Related Work

Smart home is an emerging application paradigm that has been gaining popularity in the last few years. Most recently, the IoT has fostered a vision of smart home systems, where users can install smart devices and applications that cooperate to manage home services and functionalities automatically. This emerging market rapidly attracts software developers to produce novel applications and services to provide additional smart home functionalities. However, noticeable barriers and concerns are still present, mainly related to cyber-security and safety within smart home systems, as well as to the privacy and integrity of produced and consumed data, most of which are personal and sensitive. In our work, we aim to design and implement a solution allowing end users to specify, debug, and translate high-level security policies that can be applied in a given smart home. This section contextualizes our work by discussing state-of-the-art literature on End-User Development (EUD) and rules modeling and analysis.

2.1 End-User Development in the IoT

Lieberman et al. [29] define End-User Development (EUD) as *"a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact."* With the technological advances we are confronting today,

people are increasingly moving from passive consumers to active producers of information, data, and software [32], and EUD approaches and methodologies have been extensively explored in different contexts, e.g., mobile environments [33], smart homes [37, 13], and web mashups [35, 21]. The explosion of the IoT further increased the need to allow end users to customize the behavior of their smart devices and online services. One of the most popular paradigms to empower end users in directly programming their connected entities is trigger-action programming [37, 23]. Trigger-action programming offers a straightforward and easy-to-learn solution for creating end-user applications, according to Barricelli and Valtolina [11]. It is not surprising that, in the last years, several commercial trigger-action programming platforms were born to allow end-user personalization of connected entities. Examples include IFTTT [2], Zapier [6], Microsoft Flow [3], Mozzilla's Thing Gateway [5], SmartRules [4], and many others. In its basic form, trigger-action programming allows users to connect a single event to a single action: by defining trigger-action (IF-THEN) rules, users can connect a pair of devices or online services in such a way that, when an event (the *trigger*) is detected on one of them, an *action* is automatically executed on the latter. Although some behaviors would require greater expressiveness to be defined in a single rule, e.g., through multiple actions or additional trigger conditions, many of the most popular trigger-action programming platforms, e.g., IFTTT, Zapier, and Microsoft Flow, still continue to adopt the basic form of the trigger-action programming paradigm [12].

Given its advantages and widespread adoption in EUD solutions for IoT environments, including smart homes, we decided to adopt the trigger-action programming paradigm to empower end users to define high-level security policies. Our approach is inspired by the work described in [20],in which the authors proposed a method based on Semantic Web technologies to express abstract (high-lelvel) trigger-action rules that adapt to different contextual situations, e.g., *"increase the home temperature when I'm coming home."*

2.2 Rule Modeling and Analysis

Despite the trigger-action programming paradigm can express most of the behaviors desired by potential users [11, 37], and is adopted by the most common EUD platforms [22], the definition of trigger-action rules can be difficult for non-programmers. Multiple studies investigated different aspects of contemporary platforms like IFTTT, ranging from empirical characterization of the performance and usage of IFTTT [31] to human factors related to their adoption in the smart home [37]. Large-scale analysis of publicly shared rules on IFTTT [36], and changes to the underlying models are proposed as well [22, 18]. In these studies, in particular, conflicts and ambiguities among rules emerged as possible challenges [37]. As a result, users frequently misinterpret the behavior of trigger-action rules [14], often deviating from their actual semantics, and are prone to introduce errors [25].

All the described problems naturally apply to the context of high-level security policies expressed as trigger-action rules. Consequently, our work also aims

to allow users to check and debug their policies. Many prior works face the problem of formally or semi-formally verifying event-based rules with different approaches, especially in the area of databases [24, 28], expert systems [39], and smart environments [38, 9]. Rules, indeed, can interact with each other, and even a small set of dependencies between them makes it hard (and often undecidable) the problem of predicting their overall behavior [10]. Li et al. [28], for instance, propose a Conditional Colored Petri Net (CCPN) formalism to model and simulate Event-Condition-Action (ECA) rules for active databases. Petri nets are used by Yang et al. [39] to verify rules in expert systems, and by Jin et al. [27] to dynamically verify ECA properties such as termination and confluence. In the field of smart environments, Vannucchi et al. [38] adopt formal verification methods for ECA rules, while Augusto and Hornos [9] propose a methodological guide to use the Spin model checker to inform the development of more reliable, intelligent environments.

Most of the works described above aim to check the consistency of a set of fixed and *already* defined rules, not in real time, and employ predefined use cases to validate the algorithms. The goal of the PTP system is different. Instead of performing such an "off-line" verification of rules, PTP aims at assisting end users *during* the definition of their own security policies. For this purpose, we empower the PTP interface with a novel Petri net formalism, similar to CCPN but enhanced with new elements and with semantic information.

3 The Policy Translation Point System

The Policy Translation Point (PTP) is a system that has three main goals:

1. supporting users to express high-level security policies like *"Do not record sound in the living room tonight"*;
2. translating high-level security policies into device-level policies, when possible;
3. detecting potential conflicts between high-level security policies.

Figure 1 shows the client-server architecture of the PTP system. Through the web-based PTP User Interface, users can compose new high-level security policies. The PTP Server analyzes these policies taking into account the devices and applications installed in the smart home, and produces alarms in case of conflicts and/or translates the defined high-level policies into a set of device-level policies expressed in the XACML formalism [7].

In this Section, we present the models and formalisms adopted in the PTP system (Section 3.1), and we detail how users can compose and check high-level security policies through the PTP User Interface (Section 3.2). Finally, Section 3.3 presents the implementation details.

3.1 Adopted Models and Formalisms

Concept Modeling and Translation: the SIFIS-Home Ontology The PTP system uses the SIFIS-Home ontology to model high-level security policies,

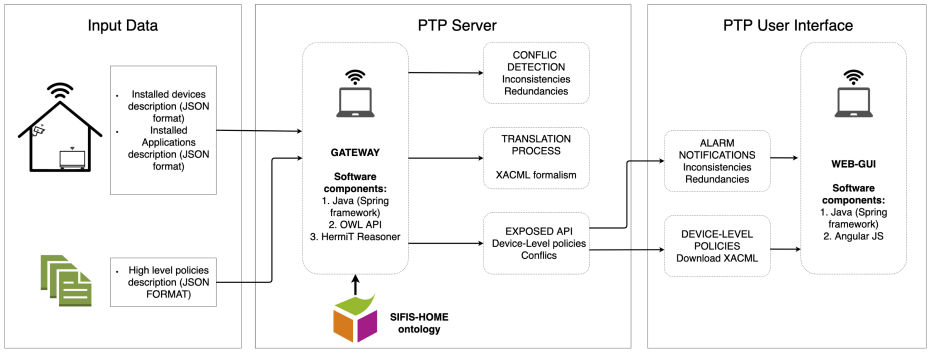


Fig. 1. The architecture of the PTP system.

smart home devices/applications, and users. Figure 2 shows the architecture of the SIFIS-Home ontology. We designed and implemented it by exploiting state-of-the-art vocabularies like foaf [1] and EUPont [20]. EUPont, in particular, is a high-level ontological representation of trigger-action programming that describes smart devices and online services based on their categories and capabilities, i.e., their offered services. In detail, for each trigger or action, the ontology provides information about the device or online service by which they are offered, and any relationship with other triggers or actions, e.g., the fact that an action implicitly activates a given trigger. Furthermore, triggers and actions are classified through a tree of classes that represents the final behavior they monitor, in case of triggers, or produce, in case of actions. Triggers or actions that are classified under the same EUPont classes, in particular, are similar in terms of final functionality, while triggers or actions that do not share any EUPont class are functionally contradictory. For example, the two actions *"set the Nest thermostat to Home mode"* and *"set 25 Celsius degree on the Nest thermostat"* share the same final functionality, because they are both classified under the same EUPont class, i.e., *IncreaseTemperatureAction*. Compared to these actions, the action *"set the Nest thermostat to Away mode"* is contradictory in terms of functionality, because it is classified under a different EUPont class, i.e., *DecreaseTemperatureAction*.

In our work, we specialized the EUPont classes to the context of high-level security policies in the smart home context. Each policy follows a simple trigger-action programming paradigm, and is defined through an abstract trigger-action rule composed of a single trigger and a single action. In the initial version of the SIFIS-Home ontology, we included the following triggers and actions:

- *Temporal triggers*: events that fire every morning, afternoon, evening, or night, respectively.
- *Video actions*: actions that allow or forbid video recording in a given location, e.g., the bedroom.
- *Audio actions*: actions that allow or forbid audio recording in a given location, e.g., the bedroom.

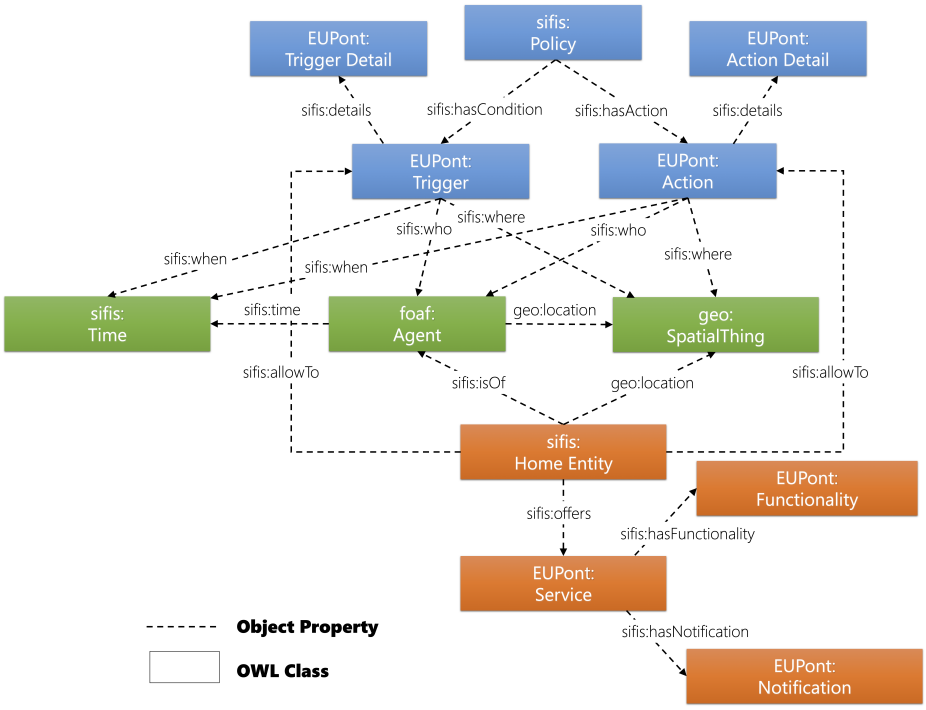


Fig. 2. The architecture of the SIFIS-Home ontology.

Figure 3a shows how a policy is modeled inside the SIFIS-Home ontology. The OWL class `POLICY` has two subclasses, i.e., `TRIGGER` and **ACTION**. A set of OWL restrictions have been added to specify that a policy must have a single trigger and a single action. Following the EUPont model, the `TRIGGER` and `ACTION` classes are in turn specialized in a hierarchy of OWL sub-classes representing events and actions of different categories. These hierarchies of sub-classes are expressed at different levels of abstraction: this potentially allows users to specify high-level policies in different ways, by choosing to be more or less specific. Figure 3b exemplifies some video-related actions included in the initial version of the SIFIS-Home ontology. For example, the `SIFIS dont-record-video` action is a RDFS instance of the `STOP VIDEO` class, while the `SIFIS record-video` action is a RDFS instance of the `START VIDEO` class.

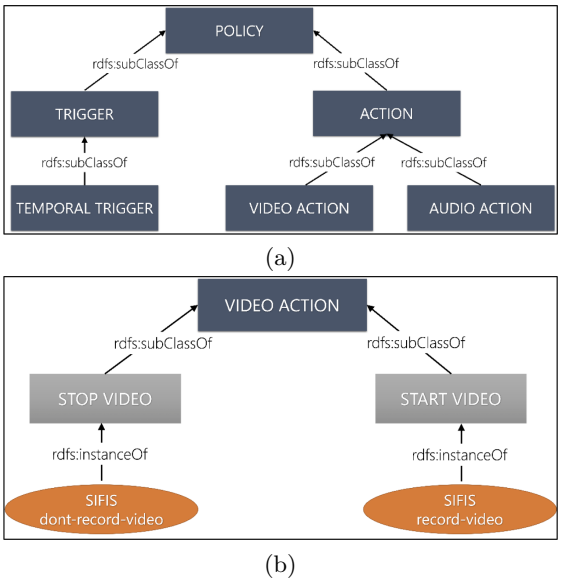


Fig. 3. Modeling of a policy inside the SIFIS-Home ontology.

As shown in Figure 2, each trigger and action is directly linked with contextual information, e.g., locations (`SpatialThing`) and users (`Agent`), and indirectly linked with devices and applications installed in the smart home (`Home Entity`). PTP uses this information to translate the defined high-level policies into the corresponding set of device-level policies in the XACML formalism.

Conflicts Detection: Semantic Colored Petri Nets To model and check the behavior of high-level security policies at run-time, we defined a formalism inspired by the Semantic Colored Petri Net (SCPN) approach defined in [19]. Petri nets are bipartite directed graphs, in which directed arcs connect places

and transitions. Places may hold tokens, which are used to study the dynamic behavior of the net. They can naturally describe policies expressed as trigger-action rules as well as their non-deterministic concurrent environment [27]. We chose such an approach to allow users to simulate step-by-step the execution of their policies: by firing a transition at a time, tokens move in the net by giving the idea of a possible execution flow. As a member of Petri nets family, Colored Petri Nets (CPNs) [26] combine the strengths of ordinary Petri nets with the strengths of a high-level programming language. In particular, SCPN is a Colored Petri Net similar to the Conditional Colored Petri Net (CCPN) formalism [28] proposed to model ECA rules in active databases. Differently from such a formalism, we do not consider conditions and use a semantic model to generate and analyze the net. Furthermore, each token assumes different semantic "colors" by moving in the net: places, in particular, are labeled with the corresponding OWL classes extracted from the SIFIS-Home ontology. Such semantic information allows the inference of more information from the simulation of the net, i.e., to discriminate between problematic and safe policies.

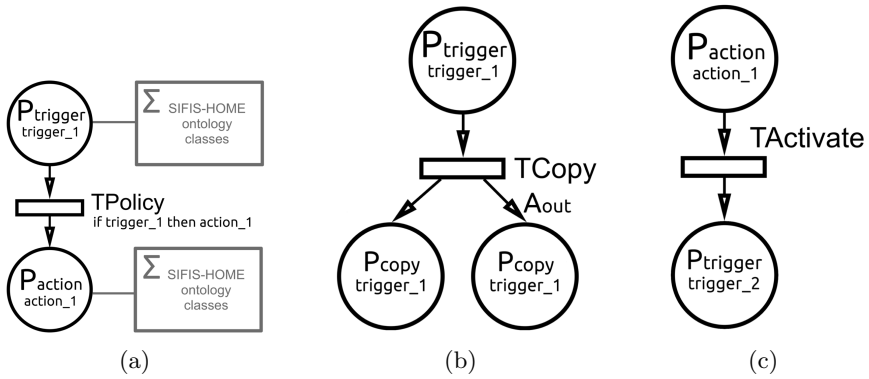


Fig. 4. The Semantic Colored Petri Net (SCPN) formalism adopted to model the runtime behavior of high-level security policies.

Figure 4 summarizes the adopted approach. Specifically:

- High-level policies' triggers and actions are modeled as places in the Petri Net. When a trigger is in common between more than one policy, the associated places are duplicated and connected through a dedicated copy transition (*TCopy*, Figure 4a). When a token is in the original place, the copy transition simply replicates the token in each copied place. Instead, action places can be directly reused by policies that have the same action.
- Places can be connected each other through a policy transition (*TPolicy*, Figure 4a), i.e., a connection between the trigger and the action of the same policy, or through an activate transition (*TActivate*, Figure 4c), i.e., a connec-

tion used when an action of a high-level policy triggers the event of another high-level policy.
– Places, i.e., high-level policies' triggers and actions, are labeled with the corresponding OWL classes extracted from the SIFIS-Home ontology (Figure 4a).

Using the described model, PTP is able to detect two possible conflicts among the currently available high-level policies: inconsistencies and redundancies.

Inconsistencies occur when policies that should be activated at (nearly) the same time[1] try to execute contradictory actions. In trigger-action rules, an inconsistency occurs when the execution order of rules may render different final states in the system [16]. In this work, we generalized this concept to consider the entire smart-home ecosystem, i.e., not only physical devices but also online services. For this reason, we analyze the *meaning* of the actions executed by the involved policies rather than their execution order. An example of a set of policies that produces an inconsistency is:

– *when* in the morning, from 9 to 12 AM, *then* record video in the bedroom;
– *when* in the morning, from 9 to 12 AM, *then* do not record any video in the bedroom;

Here, the two policies are executed simultaneously because they share the same trigger and produce two contradictory actions, i.e., allowing and prohibiting video recording in the bedroom.

Redundancies, instead, occur when two or more policies that are activated (nearly) at the same time have replicated functionality [16]. An example of a set of policies that produce a redundancy is:

– *when* in the evening, from 6 to 9 PM, *then* do not record any audio in the entire home;
– *when* in the evening, from 6 to 9 PM, *then* do not record any audio in the living room.

Also in this case, the two policies are executed simultaneously because they share the same trigger. Here, however, the action of the second policy is redundant with the action of the first policy, as the living room is part of the entire home.

3.2 User Interface

The PTP user interface can be logically split into three parts: a) *Policy Composition* (Figure 5), b) *Problem Checking* (Figure 6a), and c) *Step-by-Step Explanation* (Figure 6b). The Problem Checking and the Step-by-Step Explanation interfaces implement two well-known end-user debugging strategies: identification of rule conflicts and simulation of the run-time behavior.

[1] e.g., when policies share the same trigger or when some policies trigger other policies

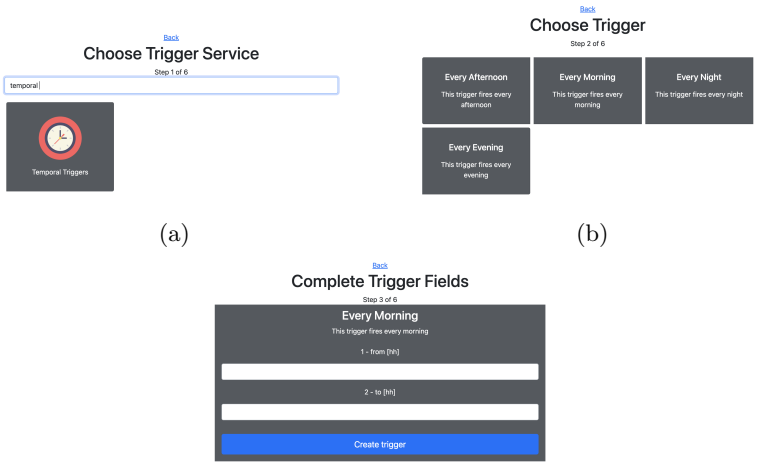
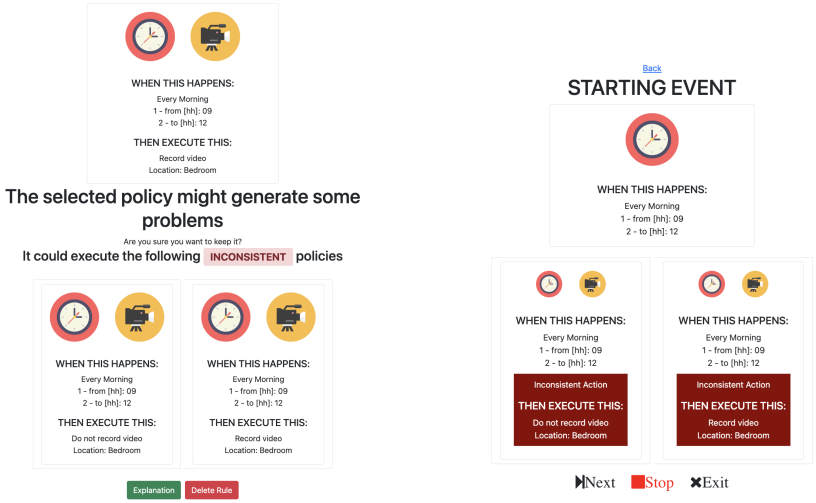


Fig. 5. The definition of a new high-level security policy in the PTP interface.

To allow the composition of high-level security policies, we designed a user interface based on the form-filling paradigm, an approach that has been found to be effective and easy to use in trigger-action programming platforms by several previous works, e.g., [15, 22]. In addition, the form-filling procedure it adopts helps users to avoid syntactical errors during the composition process. To compose a policy, a user must first select which service they want to use as a trigger, e.g., "Temporal Triggers" (Figure 5a). Once they select a service, they can choose the specific trigger to be used (e.g., "Every Morning," Figure 5b) and fill in any additional information required by the trigger (e.g., the specific time interval, Figure 5c). To define the action part of the rule, the user has to repeat the same steps.

When a rule has been composed, PTP uses the mechanisms described in Section 3.1 to find any possible conflicts with the policies that have been defined in the past, highlighting a problem to the user if necessary. The *Problem Checking* interface, in particular, shows the policy just defined by the user and any problems that the policy may generate. In Figure 6a, for instance, a possible inconsistency between two policies is highlighted. To better understand the problems and to foresee the run-time behavior of the involved policies, the user can click on the "Explanation" button to open the *Step-by-Step Explanation* interface (Figure 6b). In such an interface, the user can simulate step-by-step what happens within their policies, to try to understand why the highlighted problems arise.

At the end of the composition procedure, if no problems are detected, the user has the possibility to translate the defined high-level policy into a set of XACML policies.



(a) (b)

Fig. 6. The Problem Checking interface showing an inconsistency between an already existent policy and the defined one (a), and the corresponding Step-by-Step Explanation (b).

3.3 Implementation

The implementation of the PTP system consists of two main components:

PTP Server It is built in Java with the Spring framework[2]. It is composed of three modules: *Policy Service*, *SCPN Service*, and *Policy Controller*. The Policy Service offers the features needed to manage collections of policies i.e., to create, read, update, and delete policies through the interaction with a MySQL database. Once a user has completed a policy, the SCPN Service generates and analyzes the SCPN by retrieving the defined policies from the Policy Service, and by using the OWL API[3] library to extract the needed semantic information from the SIFIS-Home ontology. The same module is also responsible for the step-by-step simulation of the involved policies. Finally, the Policy Controller exposes a list of REST APIs to interact with the two services.

PTP User Interface It is the web-based interface built with the Angular framework[4]. It interacts with the PTP Server through the provided REST APIs.

[2] https://spring.io, last visited on November 07, 2022
[3] http://owlapi.sourceforge.net, last visited November 07, 2022
[4] https://angular.io, last visited on November 07, 2022

4 Conclusions

In this paper, we have presented the Policy Translation Point system, and End-User Development tool that empowers users to define, translate, and debug high-level security policies. The tool exploits a novel formalism based on Semantic Web technologies and Petri Nets, and it implements a trigger-action programming approach through which users can define policies as abstract trigger-action rules that do not depend on any specific technology. Furthermore, it is able to detect redundancies and inconsistencies between high-level security policies, and it can translate a high-level policy into a set of XACML policies that can be directly applied to the devices and applications installed in the smart home.

Acknowledgments

The work described in this paper is part of the SIFIS-Home Project that is supported by funding under the Horizon 2020 Framework Program of the European Commission SU-ICT-02-2020 GA 952652.

References

1. Foaf vocabulary specification (2004), http://xmlns.com/foaf/0.1/, accessed: 2022-11-07
2. Ifttt (2019), https://ifttt.com/, accessed: 2019-11-20
3. Microsoft flow (2019), https://flow.microsoft.com/en-us/, accessed: 2019-11-20
4. Smartrules (2019), http://smartrulesapp.com/, accessed: 2019-11-20
5. Webthings gateway (2019), https://iot.mozilla.org/gateway/, accessed: 2019-11-20
6. Zapier (2019), https://zapier.com/, accessed: 2019-11-20
7. Oasis extensible access control markup language (xacml) tc (2020), https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, accessed: 2022-11-07
8. Secure interoperable full-stack internet of things for smart home (2022), sifis-home.eu, accessed: 2022-11-07
9. Augusto, J.C., Hornos, M.J.: Software simulation and verification to increase the reliability of intelligent environments. Advances in Engineering Software **58**(Supplement C), 18 – 34 (2013). https://doi.org/10.1016/j.advengsoft.2012.12.004
10. Bailey, J., Dong, G., Ramamohanarao, K.: On the decidability of the termination problem of active database systems. Theoretical Computer Science **311**(1), 389 – 437 (2004). https://doi.org/10.1016/j.tcs.2003.09.003
11. Barricelli, B.R., Valtolina, S.: End-User Development: 5th International Symposium, IS-EUD 2015, Madrid, Spain, May 26-29, 2015. Proceedings, chap. Designing for End-User Development in the Internet of Things, pp. 9–24. Springer International Publishing, Cham, Germany (2015). https://doi.org/10.1007/978-3-319-18425-8_2
12. Brackenbury, W., Deora, A., Ritchey, J., Vallee, J., He, W., Wang, G., Littman, M.L., Ur, B.: How users interpret bugs in trigger-action programming. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. pp. 552:1–552:12. CHI '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3290605.3300782

13. Brich, J., Walch, M., Rietzler, M., Weber, M., Schaub, F.: Exploring end user programming needs in home automation. ACM Transaction on Computer-Human Interaction **24**(2), 11:1–11:35 (Apr 2017). https://doi.org/10.1145/3057858
14. Brush, A.B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., Dixon, C.: Home automation in the wild: Challenges and opportunities. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2115–2124. CHI '11, ACM, New York, NY, USA (2011). https://doi.org/10.1145/1978942.1979249
15. Caivano, D., Fogli, D., Lanzilotti, R., Piccinno, A., Cassano, F.: Supporting end users to control their smart home: design implications from a literature review and an empirical investigation. Journal of Systems and Software **144**, 295–313 (2018). https://doi.org/https://doi.org/10.1016/j.jss.2018.06.035
16. Cano, J., Delaval, G., Rutten, E.: Coordination of eca rules by verification and control. In: Kühn, E., Pugliese, R. (eds.) Coordination Models and Languages. pp. 33–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
17. Cerf, V., Senges, M.: Taking the Internet to the Next Physical Level. IEEE Computer **49**(2), 80–86 (Feb 2016). https://doi.org/10.1109/MC.2016.51
18. Corno, F., De Russis, L., Monge Roffarello, A.: A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT. Computer **50**(11), 18–24 (2017). https://doi.org/10.1109/MC.2017.4041355
19. Corno, F., De Russis, L., Monge Roffarello, A.: Empowering end users in debugging trigger-action rules. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. p. 1–13. CHI '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3290605.3300618
20. Corno, F., De Russis, L., Monge Roffarello, A.: A high-level semantic approach to end-user development in the internet of things. Int. J. Hum.-Comput. Stud. **125**(C), 41–54 (may 2019). https://doi.org/10.1016/j.ijhcs.2018.12.008
21. Daniel, F., Matera, M.: Mashups: Concepts, Models and Architectures. Springer Publishing Company, Incorporated (2014)
22. Desolda, G., Ardito, C., Matera, M.: Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools. ACM Transactions on Computer-Human Interaction **24**(2), 12:1–12:52 (2017). https://doi.org/10.1145/3057859
23. Dey, A.K., Sohn, T., Streng, S., Kodama, J.: icap: Interactive prototyping of context-aware applications. In: Proceedings of the 4th International Conference on Pervasive Computing. pp. 254–271. PERVASIVE'06, Springer-Verlag, Berlin, Heidelberg (2006). https://doi.org/10.1007/11748625_16
24. Gatziu, S., Dittrich, K.R.: Detecting composite events in active database systems using petri nets. In: Proceedings of IEEE International Workshop on Research Issues in Data Engineering: Active Databases Systems. pp. 2–9 (Feb 1994). https://doi.org/10.1109/RIDE.1994.282859
25. Huang, J., Cakmak, M.: Supporting mental model accuracy in trigger-action programming. In: Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing. pp. 215–225. UbiComp '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2750858.2805830
26. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol. 2. Springer-Verlag, London, UK, UK (1995)
27. Jin, X., Lembachar, Y., Ciardo, G.: Symbolic Termination and Confluence Checking for ECA Rules, pp. 99–123. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45730-6_6

28. Li, X., Medina, J.M., Chapa, S.V.: Applying petri nets in active database systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **37**(4), 482–493 (July 2007). https://doi.org/10.1109/TSMCC.2007.897329
29. Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End User Development, chap. End-User Development: An Emerging Paradigm, pp. 1–8. Springer Netherlands, Dordrecht, Netherlands (2006). https://doi.org/10.1007/1-4020-5386-X_1
30. Mattern, F., Floerkemeier, C.: From the Internet of Computers to the Internet of Things, pp. 242–259. Springer Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17226-7_15
31. Mi, X., Qian, F., Zhang, Y., Wang, X.: An empirical characterization of IFTTT: Ecosystem, usage, and performance. In: Proceedings of the 2017 Internet Measurement Conference. pp. 398–404. IMC '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3131365.3131369
32. Munjin, D.: User Empowerment in the Internet of Things. Ph.D. thesis, Université de Genève (May 2013), http://archive-ouverte.unige.ch/unige:28951
33. Namoun, A., Daskalopoulou, A., Mehandjiev, N., Xun, Z.: Exploring mobile end user development: Existing use and design factors. IEEE Transactions on Software Engineering **42**(10), 960–976 (Oct 2016). https://doi.org/10.1109/TSE.2016.2532873
34. are Social, W.: Digital in 2020 (2020), https://wearesocial.com/blog/2020/01/digital-2020-3-8-billion-people-use-social-media
35. Stolee, K.T., Elbaum, S.: Identification, impact, and refactoring of smells in pipe-like web mashups. IEEE Transactions on Software Engineering **39**(12), 1654–1679 (Dec 2013). https://doi.org/10.1109/TSE.2013.42
36. Ur, B., Pak Yong Ho, M., Brawner, S., Lee, J., Mennicken, S., Picard, N., Schulze, D., Littman, M.L.: Trigger-action programming in the wild: An analysis of 200,000 ifttt recipes. In: Proceedings of the 34rd Annual ACM Conference on Human Factors in Computing Systems. pp. 3227–3231. CHI '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2858036.2858556
37. Ur, B., McManus, E., Pak Yong Ho, M., Littman, M.L.: Practical trigger-action programming in the smart home. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 803–812. CHI '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2556288.2557420
38. Vannucchi, C., Diamanti, M., Mazzante, G., Cacciagrano, D., Culmone, R., Gorogiannis, N., Mostarda, L., Raimondi, F.: Symbolic verification of event–condition–action rules in intelligent environments. Journal of Reliable Intelligent Environments **3**(2), 117–130 (Aug 2017). https://doi.org/10.1007/s40860-017-0036-z
39. Yang, S.J.H., Lee, A.S., Chu, W.C., Yang, H.: Rule base verification using petri nets. In: Computer Software and Applications Conference, 1998. COMPSAC '98. Proceedings. The Twenty-Second Annual International. pp. 476–481 (Aug 1998). https://doi.org/10.1109/CMPSAC.1998.716699