Semantic Reasoning to Support End User Development in Intelligent Environment

Narayan C. Debnath*, Shreya Banerjee*, Giau Ung Van*, Phat Tat Quang*, and Dai Nguyen Thanh*
Eastern International University, Binh Duong, VIETNAM-

Abstract

Intelligent Environment (IE) creates complex applications on top of an existing network of sensors and actuators. Proliferation of Internet of Thing (IoT) objects in Intelligent Environments exhibits the rapid growth of End User Development (EUD). Trigger Action programming is a popular approach for EUD in IE. However, the inability of end users to interpret and compose suitable Trigger Action rules often makes inconsistencies in behavior of IoT objects. To address this issue, a semantic based reasoning framework is proposed in this paper to support end users. The proposed framework is based on an upper-level ontology specification named as Trigger Action Ontology (TAO). This framework includes a rule-based reasoner implemented in Apache Jena. The framework will assist end users of IE applications on various domains to represent triggers, actions and their respective combinations. In addition, the proposed reasoner can aid end users in recognizing programming bugs and reason about how to fix them. Further, two case studies in two different domains home automation and smart factory have been specified to prove the efficiency of the proposed framework. Moreover, a detailed comparison study has been provided to demonstrate the usefulness of the proposed work over the existing approaches.

Key Words: Intelligent environment, semantic reasoning, end user development, inference rules, reasoner, trigger action programming, ontology, programming bugs.

1 Introduction

End-User Development (EUD) empowers non-professional developers to build or modify their own applications to address their various and frequently changing requirements. One of the approaches considered in this area is the use of rule-based systems [16]. Trigger Action Programming (TAP) is such kind of rule-based system in EUD. TAP is a programming model enabling users to connect services and devices by writing if-then rules in the form: *if condition then action* [2, 20]. These kinds of rules are simple to implement. However, nuances in their interpretation can lead to user errors with consequences such as incorrect and undesirable functionalities [20].

The term Intelligent Environments (IE) refers to a diverse range of scenarios and applications that include smart homes, smart factories, smart farming, autonomous vehicles, and so on. Several commercial approaches facilitate end user to specify the creation of trigger action rules in IE such as Smart Home. Some popular names are IFTTT [13], Microsoft Flow [14], Zapier [22], Mozilla's Things Gateway [12], Stringify [19], etc. Intelligent environments are full of diverse and complicated device automation scenarios that constantly arise with multiple devices. Unfortunately, in these kinds of environments, end users may write rules with bugs or struggle to understand why particular automations are running [24]. In general, TAP connects a single trigger towards a single action. However, these rules can become complicated based on underlying behaviors, which require precise and rigorous expressiveness [2]. To cope with these complexities, several commercial platforms such as Stringify [19] and SmartRules [21] support conjunctions in a single trigger. Nevertheless, behaviors can become more complicated and will demand creation of more complex rules.

In this context, existing TAP based approaches face several challenges. Crucial challenges among these are as follows. Firstly, these approaches have nil or very little support towards empowering end users in order to realize different types of triggers, actions and their in between various connections. Existing literature [2, 8, 11] represents that triggers can be of different kinds such as event and state. Likewise, actions can be also of different kinds such as immediate action, extended action and sustained actions. Event kind of triggers happen in a specific moment. "When I enter the room" is an example of an event type trigger. State kind of triggers persist for a long period. "As long as it's raining" is an example of a state type trigger. On the other hand, immediate actions can happen at a moment. "Sending an e-mail" is an example of an immediate action. Extended actions can persist for some time and then end. "Brewing the coffee" is an example of extended action. Sustained actions can persist until other behavior is defined on the same object [21]. "Turn on the light" is an example of sustained action. Identification of these different kinds of triggers and actions are essential to form a correct and consistent TAP rule. Besides these, to achieve complex behaviors in intelligent environment, various triggers can be connected with each other. Similarly, different actions also can be connected with each other. These intra trigger or intra action connections can be different kinds such as "and" connection, "or" connection. However, non-expert end users have no or very little interpreting capability of these kinds of semantics. Hence, a semantically empowered framework is needed that can assist end users to realize the correct semantics of TAP.

Secondly, existing approaches lack facilities that may help end users to discover their mistakes and wrong interpretation when they are composing the rules. A formal semantic based tool can help end users to identify bugs in their rules and help them to rectify their mistakes [4]. However, most of the

* Department of Software Engineering. Email: narayan.debnath@eiu.edu.vn, shreya.banerjee@eiu.edu.vn, giau.ung@eiu.edu.vn, phat.tat@eiu.edu.vn, dai.nguyen@eiu.edu.vn.

commercial approaches lack these features. Existing studies [2, 11, 21] have shown that several bugs can be present in TAP rules due to incorrect interpretation of users. Several crucial examples of those bugs are infinite loop, contradict action, and timing window fallacy. Infinite loop bugs occur when rules trigger one another, resulting in loops [2]. Contradict action bugs can happen if two rules include the same trigger and create two contradictory actions on the same device. Timing window fallacy bugs arise when the time window is mismatched if multiple triggers or multiple actions are combined. For example, two event kinds of triggers will generate an immediate action. One event and one state kind trigger will generate sustained or extended actions. In this case, event trigger should happen within the time window of the state kind trigger. In addition, if one event produces a sustained or extended action, then it is also included in the timing window fallacy bug since, an event can only create an immediate action [2]. Similarly, a state kind of trigger can only create sustained or extended action. Besides presence of these crucial bugs, synthesis of TAP rules also includes several issues such as redundant rules and unused rules. End users create rules, which have the same triggers but different actions happening at the same time. Similarly, end users can also create rules, which have the same actions but different triggers than those occurring at the same time. These kinds of redundant rules are created by the end users very frequently. Besides these, several TAP rules are created which are never getting a chance to be executed. These kinds of rules are unused rules [11]. Further, end users often forget to deactivate the sustained action. This kind of bug is known as lack of action reversal [20]. Sometimes end users have not specified how long the extended action can be executed. This kind of inconsistent behavior of TAP rules is known as extended action related bugs [2]. All of these above-mentioned issues are very crucial for creation of consistent and correct TAP rules [21]. Hence, a tool is required that can detect these bugs and other issues efficiently. Moreover, the specified issues are related with precise semantics of different building blocks of TAP rules. Therefore, a semantic based reasoning tool will be in great demand that may assist an end user to discover bugs and reasons to repair them.

This paper is aimed to address these aforementioned challenges. The proposed work deals with the following research questions related with these specified challenges. *Q1*. How a semantically empowered framework can assist end users in order to create TAP rules? *Q2*. How a semantic based reasoning tool can be developed that can empower end users to identify crucial bugs and rectify those? With the objective to answer these research questions, this paper has proposed a semantic based reasoning framework. The proposed framework is based on an ontology-based specification named as Trigger Action Ontology (TAO) described in [6]. This framework can assist end users to create TAP rules according to precise semantics. These precise semantics are provided by the upper ontology TAO [6]. Ontology is defined as an explicit specification of shared conceptualization. It specifies an abstract view of the world in terms of concepts and their in between relationships [9]. The literature recognizes the value of semantic enrichments, through ontologies, for facilitating the event driven programming of IoT devices also in other domains [5]. In addition, the proposed framework includes a rule-based reasoner that can assist end users to identify important bugs and reason about how to fix them.

The contribution of the proposed work are manifolds. Firstly, it assists end users to synthesize TAP rules based on precise semantics, since the framework is based on an upper-level based ontology specification. To represent precise semantics related to a single TAP rule, end users are asked a set of questions related with the 5W1H (Why, Who, When, What, Where and How) contextual information. Based on the answers provided by end users and with the help of the formal semantics of underlying ontology-based specification TAO, the proposed framework assists the end users to create efficient TAP rules. Secondly, the proposed framework provides a rule based generic reasoner that can help end users to identify crucial bugs. The proposed reasoner is based on a set of inference rules. These inference rules are proposed based on the formal semantics provided by axioms in TAO. Using this proposed reasoner, end users can identify indefinite loop, contradict actions, time-window fallacy, lack of action reversal, extended action related bugs, redundant rules and unused rules in the synthesized TAP rules. Thirdly, the reasoner also helps end users to reason how to fix them. The proposed semantic based reasoning framework is implemented using Java based Ontology API Apache Jena [1]. Fourthly, the proposed framework can be applied in various IE domains such as smart healthcare, smart factory, smart home etc. The framework is based on an upper-level ontology specification TAO which is domain independent. In addition, effective case studies in two different domains, smart home and smart factory are used to prove the effectiveness of the proposed framework. A detailed comparison study is provided between the proposed approach and the existing approaches to illustrate the improved performance of the proposed work.

The rest of the paper is organized in the following way. Related work is represented in Section 2. A brief description of TAO [6] is represented in Section 3. Proposed methodology is specified in Section 4. Further, the effectiveness of the framework is evaluated using suitable case studies in Section 5. A detailed comparison study is provided in Section 6. Finally, in Section 7, the paper is concluded with indication of crucial future works.

2 Related Work

Several state-of-the art approaches exist in the literatures that have created framework and bug identification and fixing tools to assist end users in IE. Very few approaches have applied ontology. The majority of those approaches have used other methodologies rather than ontology. Brief descriptions of these approaches are specified next.

In [4], authors have created both user interface and a tool that can identify bugs in TAP rules. Authors have developed a debugging tool that generates the possible problems by the rules synthesized by end users. The described debugging tool also displays systematic simulation. Authors have applied a hybrid approach named as Semantic Colored Petri Net Model (SCPN) in devising the tool. This hybrid approach is based on colored petri net models and an ontology specification. However, in this approach the authors mainly focus on three kinds of bugs indefinite loop, inconsistent rule and

redundancy. Authors have not focused on bugs such as lack of action reversal or extended action related bugs. The approach is applicable in a smart home domain. In [23], authors let end users represent desired properties for devices and services. Authors have transformed these properties into linear temporal logic (LTL) and then create property satisfying TAP rules from scratch and repairs existing TAP rules. However, proposed approach in [23] cannot consider the fact, that an end user can make mistakes in specifying the properties. The approach does not consider the identification of different kinds of triggers, actions, intra-trigger, intra-action combinations and related bugs. In [24], the authors automatically synthesize TAP rules from traces. Traces are time-stamped logs of sensor readings and manual actuations of devices. This approach applies to both symbolic reasoning and SAT to synthesize consistent TAP rules, although their application area is mainly smart homes. They have also not considered about bugs and wrong interpretation due to different kinds of triggers and actions. In [21], authors have verified Event-Condition-Action (ECA) in IE rules using symbolic verification. They have mainly focused on three criteria: unused rules, redundant rules and incorrect rules. However, they have also not considered different kinds of action related bugs such as extended action related bugs and sustained action related bugs. In [15], authors have described the use of visual analytics to support analysis of the interactions carried out by users with trigger action rule-based personalization tools. Authors have also presented the application of the described method to data generated by the use of the PersRobIoTE tool. However, the approach does not include any tools that can identify bugs. In [18], the authors have presented a technique that mainly identifies errors due to missing triggers and the consequent unexpected behavior and security vulnerabilities specifically in a smart home. The authors have focused on event kinds of triggers. They also have developed a tool based on the described methodology and they considered that actions are defined by end users correctly. Further, they did not mention other kinds of bugs such as indefinite loop and contradict actions. In [3, 17], authors have developed visual tools analyzing the users' behavior when interacting with a trigger action rule editor for personalizing their IoT context dependent applications. In [17], authors also recommended how to combine multiple triggers or actions. However, in both approaches, authors have not developed any tool to find out bugs. In [25], authors have introduced interfaces that help users compare and contrast TAP program variants. The described interfaces help users reason about syntax differences, differences in actions under identical scenarios and property differences. However, they have not considered differences among triggers. In [7], authors have developed a composition paradigm of events and actions in the domain of IoT. They also have considered about 5W1H. However, they did not consider about a bug identifying tool. In [16], authors have developed a tool that mainly focuses on semantic correctness of TAP rules. They have provided end users the information related to "why/why not". Yet, they have not considered various bugs such as lack of action reversal or indefinite loops.

The majority of the existing approaches have not considered bugs that arose due to a wrong interpretation of different kinds of action's semantics. Further, very few considered a generic tool that can be applied over various kinds of domain. The proposed framework and the reasoning tool in this paper have identified different kinds of bugs related with both triggers and actions. The proposed framework can also be applied in various domains, since it is supported by an upper-level ontology specification TAO. A detailed comparison study between the proposed work and selected existing approaches have been provided in section 6.

3 Brief Description of Trigger Action Ontology (TAO) [6]

Trigger Action Ontology (TAO) described in [6] is an upper-level ontology to represent meta-rules for TAP. TAO consists of three layers - *Rules*, *Context* and *IoT Resources*. The bottom most layer of proposed TAO is *IoT Resources*. It provides ontology-based descriptions for IoT devices, services and related attributes. *Context* is the middle layer that represents the contextual information (5W1H) related to triggers and actions. This 5W term presents the basic information related to trigger and actions as follows. "Who" represents who is responsible for triggering or performing an action? "Who" can be an IoT device, a service or an end user. "When" represents, the temporal aspects that when trigger or action can happen. "Where" provides the location information related to the trigger and action. "What" represents what the trigger and the action specified. "Why" describes the reason of the trigger and performing the action. This contextual information is classified as *primary context* and *auxiliary context*. The top most layer is *Rules* that provides the precise semantics towards different kinds of triggers, actions, multiple triggers and multiple actions. Formal semantics of TAO is expressed in first order mathematical logic. Figure 1 has demonstrated the TAO model. Table 1 has described the facades of TAO.

4 Proposed Methodology

A semantic based reasoning framework is proposed in this section in order to support end-users in IE to synthesize trigger action rules. In addition, using the proposed framework, end users can identify bugs in their TAP programming rules and reasons to fix them. Section 4.1 has represented the modules and workflow of the proposed semantic based reasoning framework. Further, Section 4.2 has represented the proposed inference rules and the generic reasoner to identify the bugs in synthesized TAP rules. Section 4.3 represents the implementation of the proposed framework**.**

4.1 Proposed Semantic Based Reasoning Framework

The proposed semantic model is based on an upper-level ontology specification TAO described in Section 3. The proposed framework includes interfaces for both end users and service providers. In addition, it includes a rule based generic reasoner. Figure 2 has illustrated the outline of the proposed framework. The framework consists of two modules. The first module is the *User Interface module*. Using this user interface module, both end users and service providers can connect with the framework. Second is *Reasoning Module*. This module consists of *a Generic rule-based reasoner*, *Questionnaire Module* and the *Web*

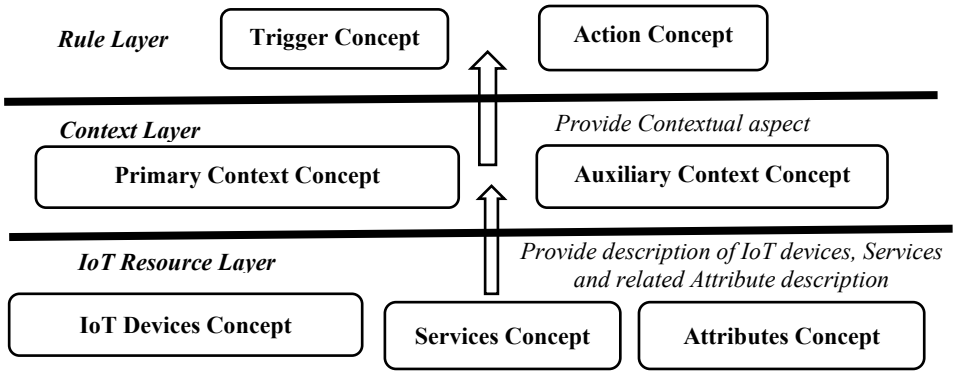

Figure 1: Trigger action ontology (TA) model specified in [6]

Table 1: Brief description of different constructs of TAO

Principle Constructs in TAO [6]		**Description**
IoT resource layer	IoT devices	Distinct domains based on IoT consists of different kinds of devices, such as sensors, actuators, and tag devices.
	Service	IoT devices can provide and consume several services
	Attributes	This concept represents properties of different IoT devices and services
	Different Kinds of relationships	Provide, Consume and Has Attribute. Provide and Consume relationship exist between device and service. Has Attribute relationship exist between either device and attribute, or between service and attribute
Context layer	Primary Context	Primary context represents 5Ws information – What, Why, Where, When and Who
	Auxiliary Context	Auxiliary context represents additional information related to primary context
Rule layer	Trigger	This concept represents causes for activation of actions. Triggers can be further categorized as two types – event and state
	Action	Actions are activated due to triggers. Actions can be further classified as two types – immediate, extended, and sustained
	Multiple Triggers and Multiple Actions	Triggers can be connected to each other using different kinds of connections such as “And”, “Or”. Likewise, actions are connected with each other using different kinds of connections
	Triggering Relationship	This relationship exists between triggers and actions.

Ontology Language (OWL) specification of TAO. At first (1), service providers are asked to enter the details about devices, services and attributes related with the domain. Then, end users are asked 5W1H questions to get the answers related with a TAP rule. End users are asked to choose trigger, action, and triggering relationships. They are also asked about the devices, location and time related to triggers and actions. Next in (2), based on these answers provided by end users, a specific TAP rule is generated. In the proposed framework, a rule file is created from TAP, which consists of inference rules specified in Apache Jena rule Language [1]. Following in (3), based on the inference rules, the generic reasoner evaluates TAP rules synthesized by end users in order to find out the bugs and inconsistencies within the rule. If any bug or inconsistencies are discovered, then end users will be notified through the user interface module.

Figure 3 has specified the workflow of the framework. According to Figure 3, the first step (1), service providers are asked to enter the details of triggers and actions. For example, if the service providers belong to a Smart Home domain, they will submit details related to triggers and actions included to that domain. Likewise, if the service providers belong to Smart Factory domain, they will submit detailed triggers and actions related to the smart factory domain. In the second step (2), the Web Ontology Language (OWL) specification of TAO is populated based on the information entered by service

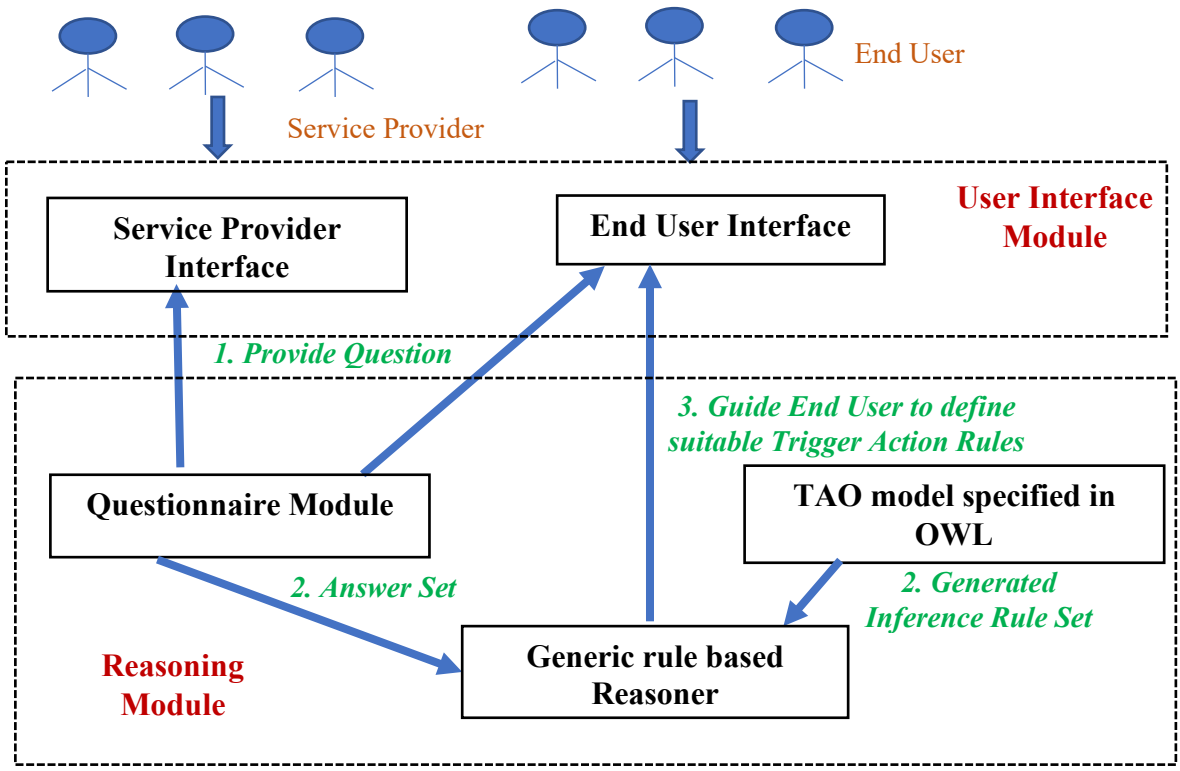

Figure 2: Proposed semantic reasoning framework

providers. In step (3), end users are asked to enter a behavior description. For example, in the case of smart factory domain, end users such as a manufacturing company can enter a behavior such as, "We want to be notified when the RPM count of our machine is unusual". Similarly, in the case of smart home domain, end users can enter a behavior such as, "When it is raining, the doors and windows need to be closed." In step (4), end users are asked to choose triggers and actions related with the entered behaviour from the list of triggers and actions entered by the service providers. They are also asked to enter detailed information related to triggers and actions based on a set of questions. These questions are based on 5W1H contextual information. The entered information from end users is used to populate the OWL specification of TAO. In step (5), the rule is synthesized by the framework based on the answers achieved in the previous step. Next, the rule is evaluated by the reasoner based on the proposed inference rules in Section 4.2 to check if the rule contains any bugs or inconsistencies. In step (6), the bugs and inconsistent rules are displayed toward end users along with the reason why the rule creates a problem. This helps end users find the solution to fix those bugs. Table 2 has specified an example set of questions related to 5W1H context asked to the end users. This set of questions specified in Table 2 is related with actions. Likewise, similar sets of questions are asked to end users to acquire the 5W1H context related information about triggers. These sets of questions are derived from the context layer of TAO.

4.2 Proposed Inference Rules and Reasoner

This section has represented a set of inference rules based on whether the proposed generic reasoner will check if there are any bugs present in the synthesized TAP rule or not. The proposed inference rules are created based on the formal axioms of TAO. These inference rules are based on IE and domain independent. Hence, the proposed reasoner can check the consistency or presence of bugs in TAP rules in various kinds of domains such as smart home, smart factory, health automation system etc. In this paper, the proposed inference rules are related with identification of several crucial bugs and consistency issues. Table 3 has represented those bugs with brief descriptions and examples. Further, Figure 4 has illustrated an example of an inference rule that can identify a timing window fallacy bug. This example represents, that there is a TAP rule *r1*. This rule has trigger *t1* and *t2*. The rule initiates an extended action *a1*. Trigger *t1* is an event and trigger *t2* is a state. Trigger *t1* happens at moment *Ti1*. Trigger *t2* is started at *TiS* time stamp and ended at *TiE* time stamp. Now, if value of *Ti1* is less than *TiS* or greater than *TiE*, this means the event *t1* is not occurring within the period of the state *t2*. However, *t1* needs to happen within the period of *t2* since, both have initiated an action *a1*. This rule is defined to identify the bug as specified in Table 3 time window fallacy bug description (iv).

4.3 Implementation of the Proposed Framework

In this section the proposed semantic based reasoning framework is implemented using Java programming language and Apache Jena [16]. Apache Jena is a java-based API used for building semantic web and linked data applications. This API supports different reasoners. Rule based generic reasoner is one of those. Using this kind of reasoner, an ontology specification can be reasoned based on inference rules in order to get additional information. In this paper, the inference

rules proposed in Section 4.2 are used to create a rule based generic reasoner. The proposed inference rules are implemented using Jena Rule Language [16]. The implemented generic reasoner in Apache Jena will take OWL specification of the populated TAO obtain after step 4 in Figure 3. Then reasoner checks for bugs and inconsistencies within the TAP rules using proposed inference rules. If bugs or inconsistencies are found then the reasoner will display the issues toward end users along with the solutions to fix them. The proposed framework has two text-based user interfaces. One interface is devised to interact with service providers. Another interface is devised to interact with end users using a 5W1H questionnaire. Using Java programming language and Apache Jena, both interfaces are implanted and information users acquired through these are populated into OWL specification of TAO. Figure 5 has specified the partial view

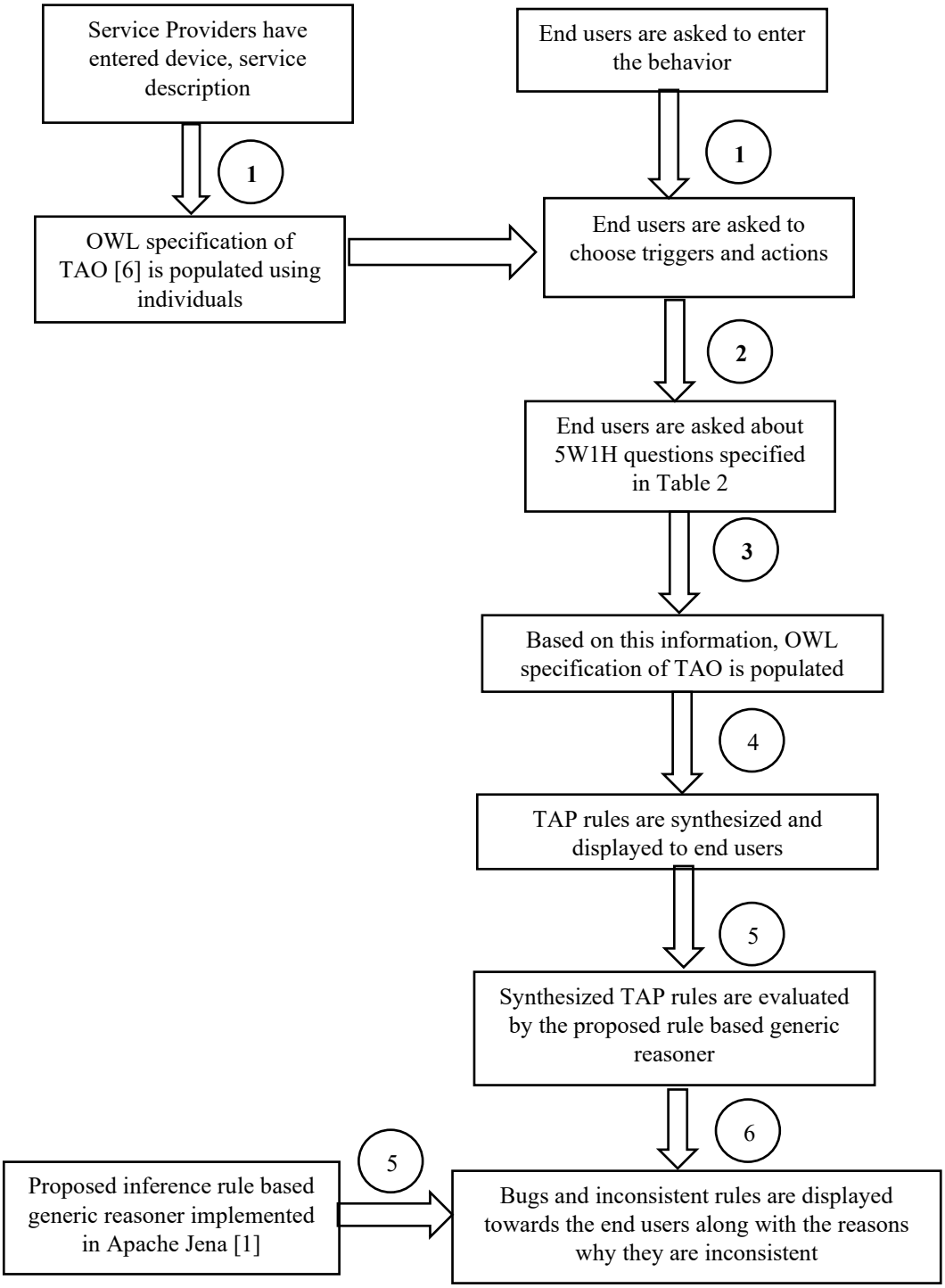

Figure 3: Workflow of proposed semantic reasoning framework

Table 2: Example of 5WlH context related question asked to end

Questions	Descriptions
Q1: What do you want the action to do?	What aspect of action
Q2: Do you want the action to do the job instantly or during some time? Specify time information.	When aspect of action. Thus, end users can choose between immediate action and other types of action
Q3: Do you want the action to be terminate itself or upon activation of another trigger?	When aspect of action. Thus, end users can choose between extended action and sustained action
Q4: Who is responsible to do the action?	Who aspect of action
Q5: Do you want to make "AND" combinations with other actions?	Add combination of triggers
Q6: Do you want to make "OR" combinations with other actions?	Or combination of triggers
Q7: Do you want to prevent other actions to this job, when this action is going on?	No combinations of trigger
Q8: Where the action will happen?	Where aspect of the action

Table 3: Descriptions of crucial bugs and related examples

Bug Name	Bug Description	Example
Indefinite Loop	If several rules trigger each other, then those rules have created indefinite loop bug.	Rule 1: If AC in the room is switched on, then windows in the room need to be closed. Rule 2: If windows in the room are closed, then the AC in the room will be switched on
Contradict Action	If one similar trigger can activate contradict actions in two different rules, then those rules can create contradict action bug.	Rule 1: Within 7 a.m. to 9 p.m. if room temperature is beyond 30° centigrade, then temperature of AC needs to be decreased. Rule 2: Within 7 a.m. to 9 p.m. if it is raining, then temperature of AC needs to be increased.
Redundant Rules	If two triggers activate the same action in different rules or one trigger activates two different actions in different rules, then those rules become redundant rules.	Rule 1: If it is raining, then the windows need to be closed. Rule 2: If AC is switched on, then the windows need to be closed.
Time-Window fallacy	(i) Semantics of event and state trigger can be changed based on the wrong time-window interpretation.	Rule 1: If music system is on and someone is present in the room, then lights in the room need to be dim.
	(ii) Semantics of immediate, sustained and extended action can be changed based on the wrong time interpretation.	In this rule, music system needs to be on within the time window when someone is present in the room. This rule is representing (iv) bug.

	(iii) When two states are combined, if their time windows are not matched, then this kind of bug is generated.	
	(iv) When one state and one event are combined, if the event does not happen within the time window of state, then this kind of bug is generated.	
	(v) When different kinds of actions are combined, then immediate action does not happen within the time frame of extended or sustained action, then this kind of bug is generated.	
	(vi) One event and one state should generate either extended or sustained action. Two states can generate either immediate or extended, sustained actions.	
Lack of Action Reversal	If sustained action is not deactivated by another rule, then it can be a continued. This situation creates lack of action reversal bug.	Rule 1: If requested item part is found inside the warehouse, then it should be delivered by drone toward the assembly line In this rule, the delivery action should be deactivated after it is finished. Otherwise, the drone will be in the assembly line.
Extended Action Bug	If end users forget to define the finishing time or finishing condition of extended actions, then the generated bug is known as Extended Action Bug.	When it is 7 pm, the coffee starts to brew. In this rule, there is no finishing time or condition about how long coffee will be brewed.
Unused Rules	Several times end users have defined some rule, which yet cannot be used in real time. This kind of bug is known as Unused Rules.	If it is raining heavy outside, then close the windows of the garage room. In this rule, if garage room has no window, then this rule will never be executed.

of the proposed framework implemented in Java. Figure 6 has represented a partial view of OWL specification of TAO in Protégé Tool [10].

5 Illustration of the Proposed Framework using Case Studies

In this section, proposed framework is evaluated based on two case studies from two different domains based on IE. Section 5.1 has demonstrated the applicability of the proposed framework using a case study from smart home domain. Section 5.2 has illustrated the effectiveness of the proposed framework using a case study from smart factory domain.

5.1 Illustration of Case Study 1

The case study specified in this section is based on smart home domain. Let's assume, an end user in smart home domain wants to implement the following behavior: Jenny wants to open the windows of the living room from morning to night. During this time, if it is raining, then windows will be closed. When the rain is over, then windows will be opened again. If the temperature of the living room exceeds 28° centigrade, then temperature in AC situated in that room will be decreased. If it is raining, then the temperature in AC situated in the living room will be increased. When the coffee maker starts to brew then the music system will be

```
IF r1 is an instance of concept Rule
t1 is an instance of concept Event
a1 is an instance of concept Extended Action
t2 is an instance of concept State
Ti1 is a value of date time stamp
TiS is a value of date time stamp
TiE is a value of date time stamp
AiS is a value of date time stamp
AiE is a value of date time stamp
has_Trigger is a relationship
has_Action is a relationship
has_Start_Time is a relationship
has_End_Time is a relationship
has_Time_Stamp is a relationship
And_Connection is a relationship
r1 has_Trigger t1
r1 has_Action  a1
r1 has_Trigger t2
t1 has_Time_Stamp Ti1
t2 has_Start_Time TiS
t2 has_End_Time  TiE
a1 has_Start_Time AiS
a1 has_End_Time  AiE
t1 And_Connection t2
greaterThan(TiS, Ti1)
lessThan(TiE,Ti1)
-----------------------------------------------------------------
Then, print (Rule r1 has a bug. Trigger t1 need to be occurred within
the time span if Trigger t2)
```

Figure 4: Example of a proposed inference rule for identifying time window fallacy bug

turned on in the living room. When someone turns on the music system in the living room, then the coffee is starts to brew.

Let assume based on this behaviour, the end user has created the following TAP rules using the proposed framework.

(1) If it is 7 a.m., the window in the living room will be opened.
(2) If it is 9 p.m., the window in the living room will be closed.
(3) If the time is between 7 a.m. and 9 p.m. and it is raining, the window in the living room will be closed.
(4) If the temperature goes beyond 28°centigrade in the living room, the AC temperature situated in the living room will be decreased.
(5) If coffee starts to brew, then the music system will be turned on in the living room.
(6) If music system is turned on, then coffee starts to brew.
(7) If it is raining, then the AC temperature in the living room will be increased.

Among these above-mentioned rules, rule number 5 and rule number 6 have created an indefinite loop, because both rules have triggered each other. Rule number 3 has produced lack of action reversal bug. In this case, the action is a sustained type of action. End users have not defined any rule to deactivate the rule. Rule number 4 and rule number 7 have created contradictory actions. There may be situations when raining and the temperature of the room exceeds 28° centigrade can occur at the same time. Rule number 1 and 2 will produce time window fallacy bugs. In both rules triggers are event kinds, but actions are not immediate types of actions. These rules also create bugs such as how long the extended action should be continued. Figure 7 has demonstrated that the proposed framework has helped the end users to create the above-mentioned rules. Figure 8 has illustrated, that illustrated, that the proposed framework has assist to find the bugs in the created TAP rules.

5.2 Illustration of Case Study 2

The case study specified in this section is based on smart factory domain. Let's assume an end user in a smart factory domain wants to implement the following behavior.

ABC is a manufacturing company. It wants to monitor the work performance of a water purification system if the water pressure is greater than 50 and water flow is greater than 15. It also wants to check the stock of parts of the water purification system. If the stock of parts is nil, then the supplier employee is notified through Short Message Service (SMS). Further, when the warehouse has received

```
 public class Main {
      public static void main(String[] args) {
            BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
            String choice=null, choiceRule=null,
choiceBehavioure=null;
            DomainExpert expert=new DomainExpert();
            //EndUser user=new EndUser();
            Interface inter=new Interface();
            System.out.println("What is your role? Please enter
either 'Service Provider' or 'End User'.");
-----------------------------------------------------------
                  //expert.readOntology();
                  expert.createDomainTrigger();
                  expert.createDomainAction();
                  expert.createContradictRelation();
                  expert.writeOntology();
            } catch (Exception e) {

                  e.printStackTrace();
            }
            System.out.println("..You as a Service Provider done
your Job....");
            }
            if(choice.equalsIgnoreCase("End User"))
            {

-------------------------------------------------------------------
                  System.out.println("..You as an End User done
your Job....");
                  System.out.println("Debugging is started");
                  inter.reasoningRule();
                  System.out.println("Debugging is stopped");

            }
      }
}
```

Figure 5: Partial view of proposed semantic reasoning framework implemented in Java and Apache Jena [1]

a request on a particular lightweight material and the material is found in the warehouse, then it will be delivered to the assembly line.

Let's assume based on this behavior, the end user has created the following TAP rules using the proposed framework:

(1) If water pressure is greater than 50, then working performance of the water purification needs to be monitored.
(2) If water pressure is greater than 25, then working performance of the water purification needs to be monitored.
(3) If the stock of parts is nil, then the supplier employee is notified through email.
(4) If the warehouse received a request on a particular lightweight material, the material will be delivered to the assembly line using a drone.
(5) If the requested material is found in the warehouse, the material will be delivered to the assembly line using a drone.

Among the previous specified rules, rule 1 and rule 2 are redundant, since they have created the same actions. Likewise, rule 4 and rule 5 are redundant. Hence, these rules can be combined. Rule 3 can be an example of an unused rule, since in the rule how suppliers will be notified is mentioned by SMS. However, in rule 3, the end user wants that supplier employee to be notified through email. Figure 9 has demonstrated that the proposed framework has helped the end users to create the above-mentioned rules. Figure 10 has illustrated that the proposed framework has assist to find the bugs in the created TAP rules.

6 Comparison Study

In this section, the proposed work is evaluated based on the comparison with several selected existing works. All of these

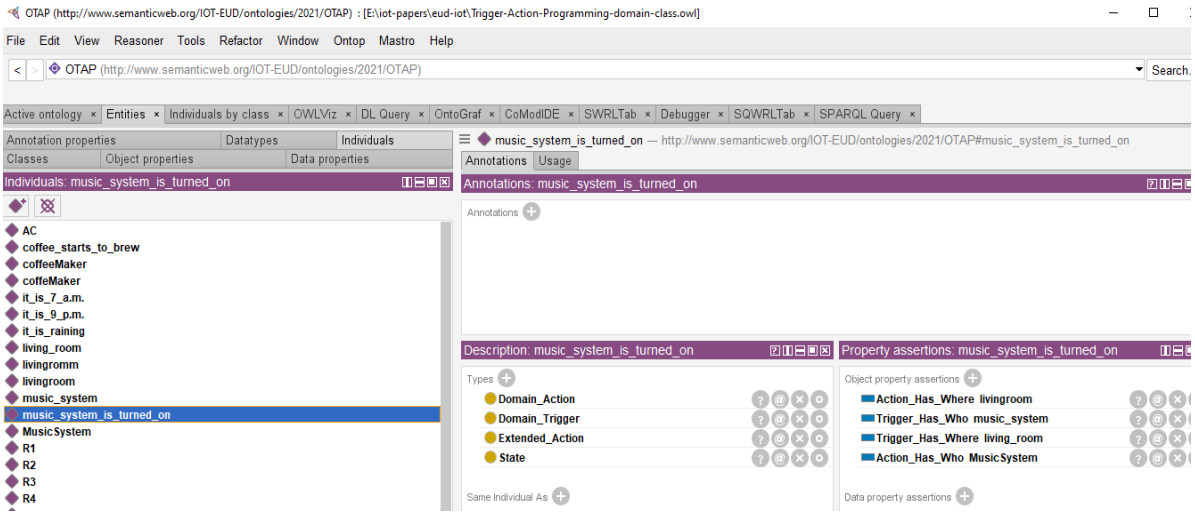

Figure 6: Partial view of OWL specification of TAO in Protégé tool [10]

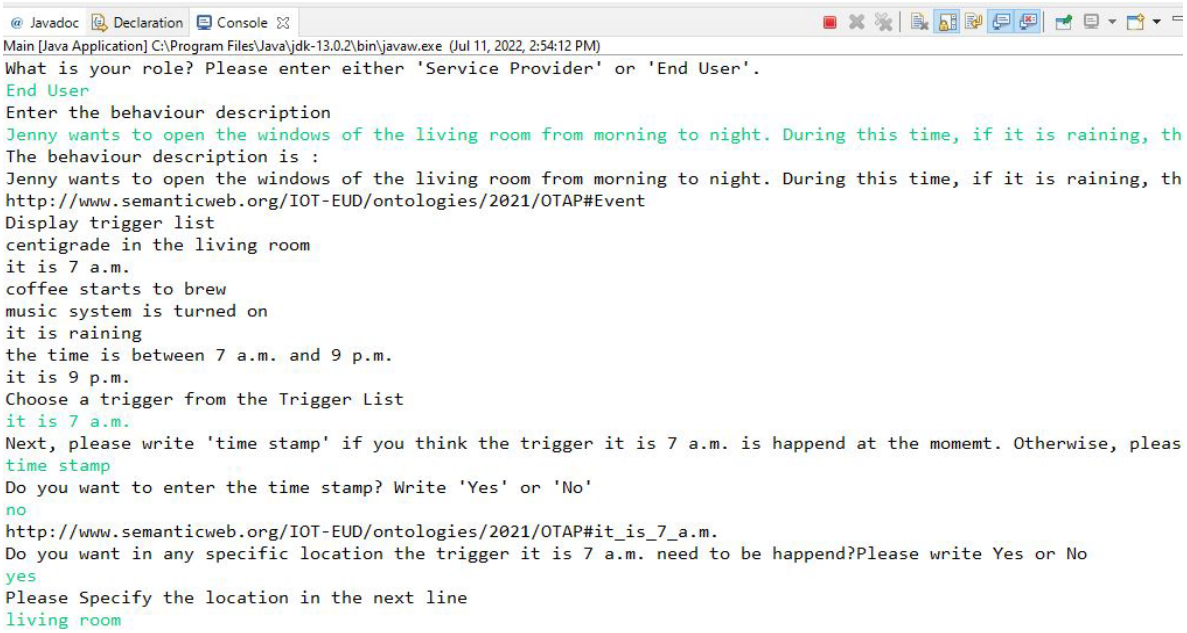

Figure 7: Illustration of Case study 1 - end users are supported to enter Answers of 5W1H questions and synthesize TAP rules

works have devised frameworks to support end users in synthesizing TAP rules. However, the proposed framework in this paper is distinct from these existing works due to possessing some useful features. The set of those useful features is specified next. Table 4 has summarized the comparison.

(a) *Trigger Related Features*: These kinds of features represent semantics related to different kinds of triggers – event and state.
(b) *Action Related Features*: These kinds of feature represent semantics related to different kinds of actions – immediate, extended and sustained.
(c) *Triggering Feature*: This feature represents semantics that will specify the kind of triggering, single triggering or multiple triggering. Single triggering specifies that, a single trigger can activate a single action. On the other hand, multiple triggering represented the presence of either multiple triggers, multiple actions or both in a triggering
(d) *Context Support*: This feature represents the support towards representation of 5W1H contextual information. 5W1H has represented Why, Who, What, Where, When and How related context for a particular trigger and action.
(e) *Domain Independency:* This feature represents the scope of the application towards different domains.
(f) *Reasoning Support:* This feature represents the support devise TAP rules.
(g) *Easy Interaction*: Interface between user and the framework needs to be simple and convenient for use. provided to the end users in making reasons when they

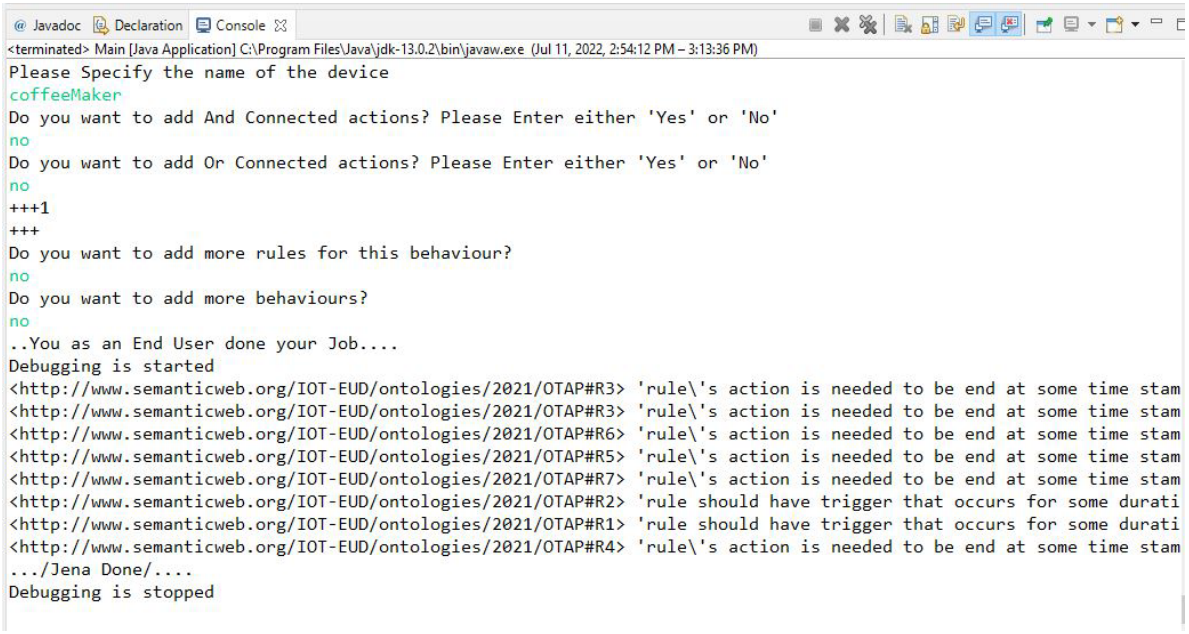

Figure 8: Illustration of Case study 1 – Identification of bugs in synthesized TAP rules and reasons to fix those bugs

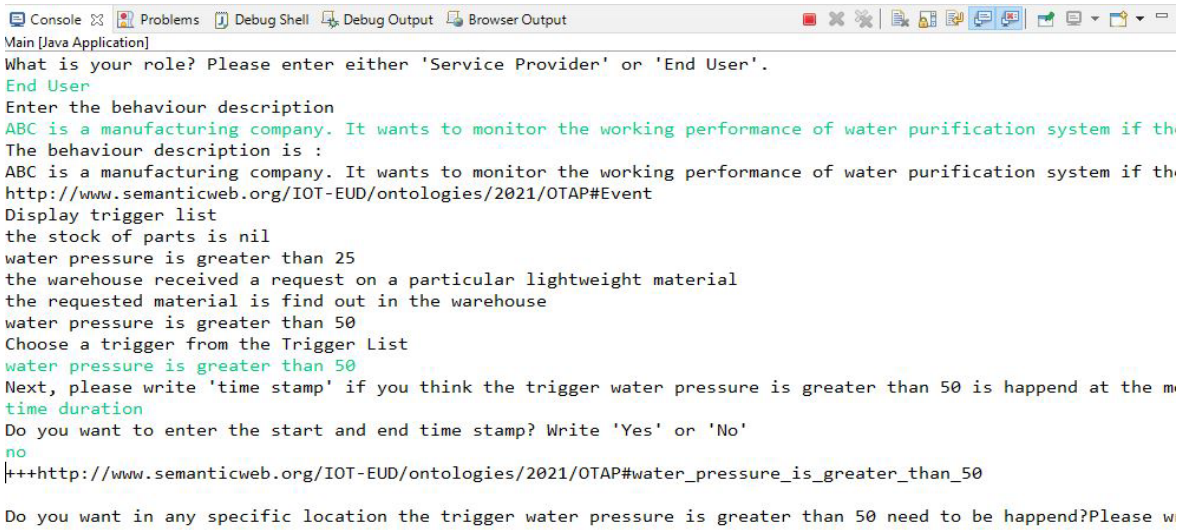

Figure 9: Illustration of Case study 2 – end users are supported to enter Answers of 5W1H questions and synthesize TAP rules

(h) *Debugging Support*: The feature indicates the support to identification of bugs' presence and reasons to fix them.

(i) *Identification of Crucial Bugs*: This feature represents support to important bugs – (i) Time Window Fallacy, (ii) Contradict Rules, (iii) Indefinite Rules, (iv) Redundant Rules, (v) Lack of Action Reversal Bugs, (vi) Extended Action Related Bug, and (vii) Unused Rules.

(j) *Interoperability:* This feature indicates the representation of interoperable TAP rules that can be applied on different domains at the same time.

From Table 4 it is proved that the proposed framework has supported different specified features such as trigger, action, triggering and contextual information related semantics. On the other hand, the majority of existing approaches have represented trigger related semantics. Very few have recognized the importance of action, triggering and contextual information related semantics. However, semantics related to all of these building blocks of TAP are essential in order to interpret the create a TAP TAP rule effectively. Proposed framework can be applied to various domain on IE since it is based on an upper-level ontology specification. In addition, the proposed framework is domain independent and also capable to represent domain interoperable rules. This feature is not exhibited in the majority of the approaches. This reasoning support has been provided by very approaches partially. The majority of the existing approaches have support to identifications of bugs applied to various domains such as smart home, smart factory, autonomous vehicle etc. Two separate interfaces have been such as contradict action,

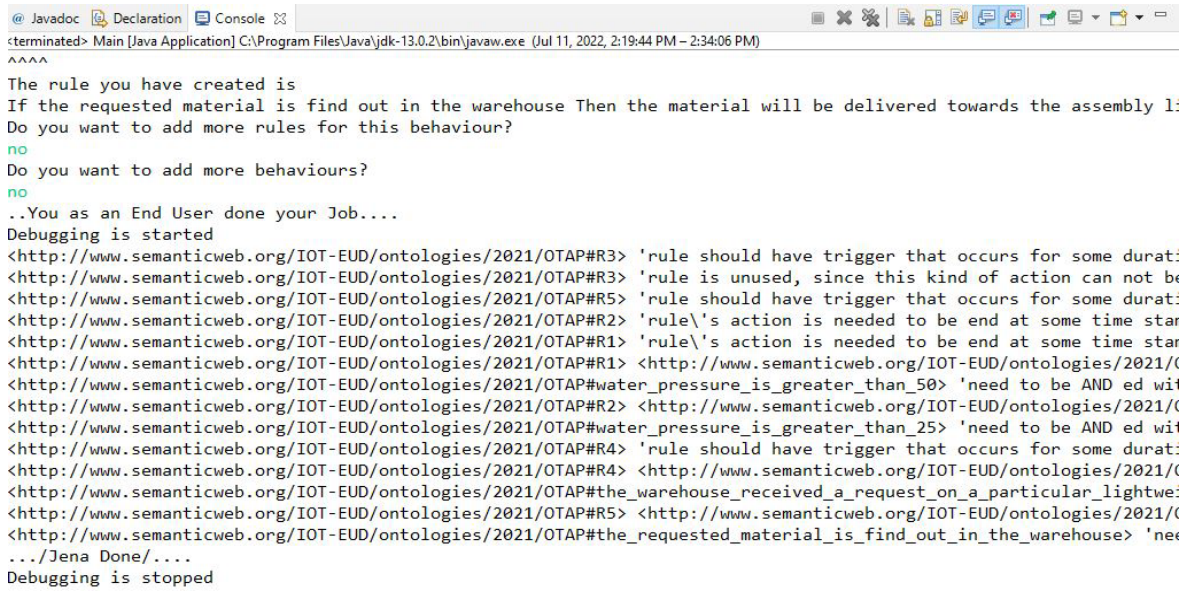

Figure 10: Illustration of Case study 2 – Identification of bugs in synthesised TAP rules and reasons to fix those bugs

Table 4: Comparison of proposed work with existing work based on several crucial features of TAP

Approaches	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)							(j)
									(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	
[4]	Y	-	Y	P	-	P	Y	Y	P	Y	Y	Y	-	-	-	-
[23]	-	-	-	-	-	-	Y	Y	-	P	P	P	-	-	-	-
[24]	-	-	-	-	-	Y	Y	-	-	P	P	P	-	-	-	-
[21]	Y	-	P	P	Y	Y	Y	Y	P	Y	Y	Y	-	-	Y	Y
[15]	Y	Y	Y	P	-	-	Y	-	-	-	-	-	-	-	-	-
[18]	P	-	P	-	-	-	Y	Y	P	-	-	P	-	-	-	-
[3, 17]	Y	Y	Y	Y	-	-	Y	-	-	-	-	-	-	-	-	-
[25]	-	Y	P	-	-	Y	Y	Y	-	-	-	-	-	-	-	-
[7]	Y	Y	Y	Y	-	Y	Y	-	-	-	-	-	-	-	-	-
[16]	Y	P	Y	Y	-	Y	Y	Y	P	P	-	P	-	P	P	-
Proposed Framework	Y	Y	Y	Y	Y	Y	P	Y	Y	Y	Y	Y	Y	Y	Y	Y
Notes: P: Partial support; Y: Full support; –: Not Mentioned.																

redundant rules, and indefinite rules. Very few have support to other mentioned bugs. These drawbacks of the existing approaches can be resolved by the proposed framework. Further, the user interface of proposed framework is textual based which is inconvenient for end users. Hence, there is a need to devise a graphical user interface for the proposed framework and this need is identified as a future work.

7 Conclusion

In Intelligent Environment (IE) context, end user development still exhibits a lack of tool supports that can assist end users to create Trigger Action programming (TAP) rules effectively. In order to address this issue, this paper has proposed a semantic based reasoning framework that may help end users to create TAP rules. In addition, the proposed framework also can help end users to check the semantic correctness of the TAP rules. The contributions of the proposed work are manifolds. Firstly, the proposed framework is based on an upper-level ontology TAO thus, it can support end users to devise TAP rules based on precise and rigorous semantics. Secondly, it has proposed a rule based generic reasoner that can check the semantic correctness of the synthesized TAP rules. The proposed reasoner also helps end users to reason about the mistakes and fixing those. Thirdly, a set of inference rules have been proposed based on the formal semantics provided by the underlying upper ontology TAO. The proposed reasoner can check the semantic correctness of TAP rules based on these inference rules. The proposed inference rules are aiming to find crucial bugs present in TAP. Examples of those bugs are, time

window fallacy, contradict rules, indefinite rules, redundant rules, lack of action reversal bugs, extended action, related bug, and unused rules. Fourthly, the proposed rules are domain independent based on an upper-level ontology. Hence, the proposed framework and the reasoner can be provided to support both service provider and end users. In addition, a comparative study has been conducted to demonstrate the usefulness of the proposed work. Future work includes developing a graphical user interface application based on the proposed framework that can make easy interaction with both end users and service providers. Further, an exploratory user test of the proposed framework in various domains will be important. In addition, a security vulnerability test of synthesized TAP rules will be a crucial future work. Moreover, integration of the proposed framework within the intelligent environment architecture will be a prime future work.

References

[1] Apache Jena. Retrieved June 10, 2022, from https://jena.apache.org.

[2] W. Brackenbury, A. Deora, J. Ritchey, J. Vallee, W. He, G. Wang, and B. Ur, "How Users Interpret Bugs in Trigger-Action Programming," *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1-12, May 2019.

[3] L. Corcella, M. Manca, F. Paternò, and A. Santoro, "A Visual Tool for Analysing IoT Trigger/Action Programming," International Conference on Human-Centred Software Engineering, Springer, Cham, pp. 189-206, September 2018.

[4] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering End Users in Debugging Trigger-Action Rules" *Proceedings of the 2019 CHI Conference on Human Factors in Computing System*s, pp. 1-13, May 2019.

[5] F. Corno, L. De Russis, and A. Monge Roffarello, "RecRules: Recommending IF-THEN Rules for End-User Development," ACM Transactions on Intelligent Systems and Technology (TIST), 10(5):1-27, 2019.

[6] N. C. Debnath, S. Banerjee, G. U. Van, and P. T. Quang, "An Ontology Based Approach towards End User Development of IoT," *Proceedings of 37th International Confer*, 82:1-10, 2022.

[7] G. Desolda, C. Ardito, and M. Matera, "End-User Development for the Internet of Things: EFESTO and the 5W Composition Paradigm," International Rapid Mashup Challenge, Springer, Cham, pp. 74-93, June 2016.

[8] G. Gallitto, B. Treccani, and M. Zancanaro, "If When is Better Than if (and While Might Help): On the Importance of Influencing Mental Models in EUD (A Pilot Study)," EMPATHY@ AVI, pp. 7-11, September 2020.

[9] N. Guarino, D. Oberle, and S. Staab, "What is an Ontology?" Handbook on Ontologies, Springer, Berlin, Heidelberg, pp. 1-17, 2009.

[10] M. Horridge, "A Practical Guide to Building OWL Ontologies using Protégé 4 and COODETools, Edition 1.3, The University of Manchester, Retrieved June 10, 2022, from https://mariaiulianadascalu.files.wordpress.com/2014/02/owl-cs-manchester-ac-uk_-eowltutorialp4_v1_3.pdf.

[11] J. Huang and M. Cakmak, "Supporting Mental Model Accuracy in Trigger-Action Programming," *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 215-225, September 2015.

[12] Matthew Hughes, "Mozilla's New Things Gateway is an Open Home for Your Smart Devices," TheNextWeb, 2018.

[13] IFTTT, Retrieved June 10, 2022, from https://ifttt.com/explore.

[14] Nat Levy, "Microsoft Updates IFTTT Competitor Flow and Custom App Building Tool PowerApps", GeekWire, 2017.

[15] M. Manca, F. Paternò, and C. Santoro, "Analyzing Trigger-Action Programming for Personalization of Robot Behaviour in IoT Environments," International Symposium on End User Development, Springer, Cham, pp. 100-114, July 2019.

[16] M. Manca, F. Paternò, C. Santoro, and L. Corcella, "Supporting End-User Debugging of Trigger-Action Rules for IoT Applications," *International Journal of Human-Computer Studies*, 123:56-69, 2019.

[17] A. Mattioli and F. Paternò, "A Visual Environment for End-User Creation of IoT Customization Rules with Recommendation Support," Proceedings of the International Conference on Advanced Visual Interfaces, pp. 1-5, September 2020.

[18] C. Nandi and M. D. Ernst, "Automatic Trigger Generation for Rule-Based Smart Homes," *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pp. 97-102, October 2016.

[19] Ed Oswald, "IFTTT Competitor Stringify gets a Major Update," TechHive, 2016.

[20] M. Palekar, F. Earlence and R. Franziska, "Analysis of the Susceptibility of Smart Home Programming Interfaces to End User Error," IEEE Security and Privacy Workshops (SPW), IEEE, pp. 138-143, 2019.

[21] C. Vannucchi, M. Diamanti, G. Mazzante, D. Cacciagrano, R. Culmone, N. Gorogiannis, and F. Raimondi, "Symbolic Verification of Event–Condition–Action Rules in Intelligent Environments," *Journal of Reliable Intelligent Environments*, 3(2):117-130, 2017.

[22] Zapier, Retrieved June 10, 2022, from https://zapier.com/.

[23] L. Zhang, W. He, J. Martinez, N. Brackenbury, S. Lu, and B. Ur, "AutoTap: Synthesizing and Repairing Trigger-Action Programs using LTL Properties," 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp. 281-291, May 2019.

[24] L. Zhang, W. He, O. Morkved, V. Zhao, M. L. Littman, S. Lu, and B. Ur, "Trace2tap: Synthesizing Trigger-Action Programs from Traces of Behavior," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(3):1-26 (2020).

[25] V. Zhao, L. Zhang, B. Wang, S. Lu, and B. Ur, "Visualizing Differences to Improve End-User Understanding of Trigger-Action Programs," Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1-10, April 2020.

Narayan C. Debnath is currently the Founding Dean of the School of Computing and Information Technology at Eastern International University (EIU), Vietnam. He is also serving as the Head of the Department of Software Engineering at EIU, Vietnam. Dr. Debnath has been the Director of the International Society for Computers and their Applications (ISCA), USA since 2014. Formerly, Dr. Debnath served as a Full Professor of Computer Science at Winona State University, Minnesota, USA for 28 years where he also served as the Chairperson of the Computer Science Department for 7 years. Dr. Debnath earned a Doctor of Science (D.Sc.) degree in Computer Science and also a Doctor of Philosophy (Ph.D.) degree in Physics. In the past, he served as the elected President and Vice President of ISCA, and has been a member of the ISCA Board of Directors since 2001. Professor Debnath has made significant contributions in teaching, research, and services across the academic and professional communities. Dr. Debnath is an author or co-author of over 500 publications in numerous refereed journals and conference proceedings in Computer Science, Information Science, Information Technology, System Sciences, Mathematics, and Electrical Engineering. Dr. Debnath is an editor of several books published by Springer, Elsevier, CRC Press, and Bentham Science Press on emerging fields of computing. He also served as a guest editor of the Journal of Computational Methods in Science and Engineering (JCMSE) published by the IOS Press, the Netherlands.

Shreya Banerjee is presently a Faculty Member in the Department of Software Engineering, Eastern International University, Vietnam. She received her PhD degree in Computer Science and Engineering from National Institute of Technology, Durgapur, India, in 2019. She received a Master degree in Software Engineering from National Institute of Technology, Durgapur, India, in 2013. She is an author and co-author of several publications in numerous refereed journals and conference proceedings. Her main areas of research interests include Software Engineering, Ontology Engineering, Semantic Reasoning, Database Management System and Service Oriented Computing.

Van-Giau Ung graduated MSc at HCMC University of Science in Dec 2015. After that, he continued working in software development companies for many years. He began teaching and research at University of Information Technology – Vietnam National University HCMC in 2017. In 2019, he moved to Software Engineering Department at Eastern International University. His research interests include Software Engineering, System and application Security, and Natural language processing. He loves to research new technologies, and share his knowledge and experience with students.

Phat Tat Quang received Master degree of Computer Science from VNU - University of Science, Ho Chi Minh, Vietnam in 2011, and Bachelor degree in Information Technology from Binh Duong University – Vietnam in 2007. Since 2011, he has been a full-time lecturer in School of Computing & Information Technology at Eastern International University, Binh Duong, Vietnam. His research areas include Computer Vision, Deep learning and their applications, and Software engineering.

Dai Thanh Nguyen joined The Eastern International University in 2015 as a lecturer in the School of Computing and Information Technology. He achieved a Bachelor's degree in Engineering in Computer Networks and Communication at Vietnam National University - Ho Chi Minh City - the University of Information Technology in 2012. He received his Master's degree in Computer Science at the University of Houston - Clear Lake, Houston, Texas, US in 2014. He is interested in software development and sharing knowledge to help students develop their careers