Complex & Intelligent Systems
https://doi.org/10.1007/s40747-024-01349-8

ORIGINAL ARTICLE

Intelligent conflict detection of IoT services using high-level Petri nets

Rong Yang[1] · Mou Wu[1,2] · Xueqin Gui[1] · Hongsheng Chen[1]

Received: 7 February 2023 / Accepted: 7 January 2024

Abstract
Internet of Things has become a common paradigm for various domains. To meet a user's complex requirement, we should compose multiple IoT devices (IoT services) to provide comprehensive services to the user. However, these services usually coexist, which is likely to lead to conflicts. Moreover, each user may have different kinds of needs. Suppose that in a smart home there are more than one person, a conflict may occur when they request the same service in this environment. Actually, even though they request different IoT services, among which if there exists function impact or QoS impact, a conflict could still occur. In this paper, we propose to employ high-level Petri nets to detect conflicts among IoT services. We first model the formal methods for conflict policies. Then, we present a Petri nets-based mechanism for modeling and detecting conflicts. Finally, we expand the previous model and use high-level Petri nets for handling fuzzy IoT conflict policies modeling and reasoning. The experimental results show that the proposed approach performs well in accuracy.

Keywords Internet of things · IoT service · Conflict · Petri net

Introduction

Nowadays, the Internet of Things (IoT) has become a hot research topic [1] and has a wide range of applications [2]. By means of data sharing, IoT applications and devices can provide unified services to the users. Smart home is an especially successful application of IoT, where a number of IoT devices are deployed. In this paper, we borrow the idea of the service paradigm. Specifically, we abstract each smart home device as an IoT service, which has both functional attributes and non-functional properties (i.e., Quality of Service, QoS). For example, an air conditioner in a smart home is represented as an air conditioner service. The functional property of the air conditioner service is to regulate the temperature of the room. Meanwhile, the air conditioner service has to achieve certain non-functional properties, such as refrigeration effect and low noise.

There are a large number of studies on context-aware applications in intelligent spaces. Generally speaking, by means of obtaining various contexts from users and their environments, the context-aware applications aim to provide customized and personalized services [3, 4]. Actually, in a context-aware application environment, the coexistence of multiple users will make the problem more complicated. For example, in a smart home, there may be three persons, i.e., the son, the father and the mother. However, each person may have his or her own preferences for the IoT service. The son, for instance, likes watching cartoons but the father prefers a Hollywood war movie. If there is only one TV in the room, this can result in a conflict. In other words, due to different preferences of people over IoT services, there may be conflicts in a smart space. So, in a smart home environment, it is a crucial factor to deal with the multiuser conflict over IoT applications. In recent years, researchers have made great effort on addressing the multiuser conflict problem [9, 10]. However, in real smart homes, multiple applications coexist and their services are frequently interrelated. Therefore, we should find a movable balance between the shared applications. Moreover, resolving conflicts, in turn, can greatly improve service security [9, 49].

In this paper, we are presenting a Petri nets-based mechanism for conflict detection of IoT services in a smart home. As we can see from Fig. 2, this mechanism consists of

✉ Mou Wu
mou.wu@163.com

1 College of Computer Science and Technology, Hubei University of Science and Technology, Xianning 437100, China

2 Laboratory of Optoelectronic Information and Intelligent Control, Hubei University of Science and Technology, Xianning 437100, China

Published online: 24 February 2024

three phases: conflict policies normalization, conflict policies transformation, and conflicts detection. The detailed process is shown in Sects. "Petri nets for IoT conflict policies modeling and verification"–"Conflict detection". By the way, in Sect. "Formal definitions of conflicts", we detail three types of conflict policies (i.e., functional conflict policies, non-functional conflict policies and controller safety policies). There are exactly five policies and they are described in Definitions 6–10, respectively.

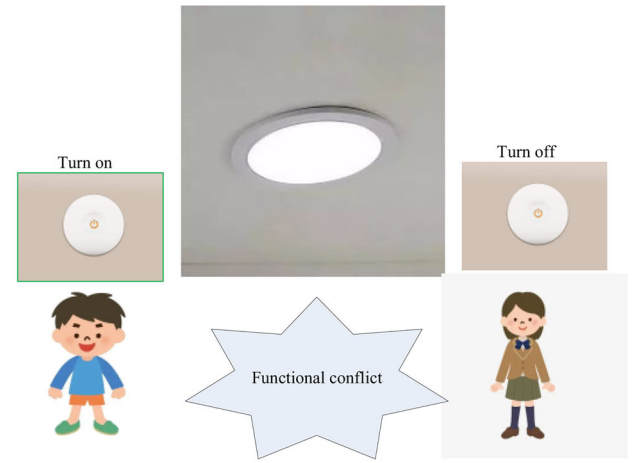
Why do we prefer Petri nets, not other formal modeling tool for conflict detection of IoT services? The advantages of applying Petri nets formalism to conflict detection can be summarized as follows: first, the graphical nature of Petri nets can visualize sequences of firing via token passing over the net that can be used to simulate the occurrence of a series of IoT service events. Second, Petri nets are perfect for modeling and property checking of concurrent and loop structures of IoT service events. Third, by means of Petri nets, we can well model truth trigger conditions, and can perfectly model and reason fuzzy IoT conflict policies.

Motivation

For the sake of clarity, we first detail the motivation of this work.

The first example is about functional conflict. As shown in Fig. 1a, suppose that there is only one light (the light service) in the bedroom, and the son, Tom, has a habit of playing games in the bedroom, turning on the light in the evening. The mother, Alice, however, has a habit of turning off the light early at night. Actually, the intelligent home systems had better understand the users' habits and may adjust the IoT services' settings based on the contexts. In a smart space, there may be a variety of contexts (i.e., several users' requirements). For instance, when Tom and Alice both are in the bedroom during the evening, there are two typical contexts, i.e., keeping the light turned off for Alice and turning on the light for Tom (since he is a game enthusiast, and he often goes to bed late). As a result, there are two different contexts at the same time and location. However, a service cannot implement different functions simultaneously. Specifically, for the light service in Fig. 1a, suppose its state is "On", it can realize its functionality. Conversely, when a resident turns off the light, it cannot implement its functionality. In a word, the multiple distinct states of a service may cause functional conflicts.

Then, in Fig. 1b, we detail an example of non-functional conflict on stereo service. In the smart home of Tom and Alice, there is a stereo service in the living room. Both of them enjoy listening to music, but they prefer different volume. Tom's preferred volume is between 60 and 80 dB. But Alice, has a preferred volume between 40 and 50 dB. Similar to the above example, if Tom and Alice are both in the living room, the stereo service cannot satisfy their volume requirements. This is because they have different volume preferences, and the stereo service does not work concurrently with two different volume settings. So, as a result, a non-functional conflict occurs.



(a) The example of functional conflict on light service

(b) The example of non-functional conflict on stereo service

Fig. 1 Example of conflicting scenarios based on users' requirements

Our approach

This research has contributed to developing a conflict detection framework of IoT Services using high-level Petri nets for future intelligent environments. Figure 2 indicates the high-level schema of our approach. For the rest of the paper, we use habits, preferences, and requirements interchangeably. By means of observing their application accessing history, we can understand different users' service usage habits. Actually, every individual has his or her own preference. Therefore, based on the analysis of the previous section, there is a need to design a conflict detection framework.

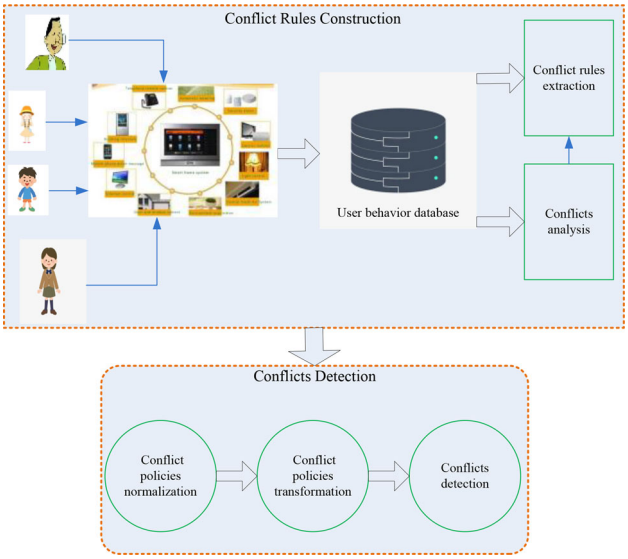


Fig. 2 The architecture of our system

As shown in Fig. 2, our conflict detection system consists of two major steps, i.e., conflict rules construction and conflicts detection. In the first step, the user behavior database recorded the residents' application accessing history. Then, through conflicts analysis of inhabitants' behavior and habits, we can get a large collection of conflict scenarios. The system further analyzes those conflict scenarios by means of classification, and a series of conflict rules can be returned. However, in the second step, the system mainly achieves conflicts detection, where three phases are included: conflict policies normalization, conflict policies transformation, and conflicts detection. We first normalize conflict policies with certainty factors, transform the normalized conflict policies into high-level Petri nets, then detect conflicts by verifying the normalized Petri net which we get from one specific scenario.

In summary, the main contributions of this paper are fourfold:

1) We present a formal method for conflict policies of IoT services. We give the formal definitions with respect to three types of conflict policies of a smart home system. Specifically, we generalize five conflict policies and give their formal descriptions.
2) We employ a Petri nets-based mechanism for modeling and detecting IoT service conflicts of a smart home. Based on our method, truth trigger conditions and IoT service events both can be transformed to Petri nets. And, we transform each conflict policy to a Petri net model.
3) We use high-level Petri nets (HLPN) to model and reason fuzzy IoT conflict policies. We expand the models in Sect. "Petri nets for IoT conflict policies modeling and verification" and transform them to Petri nets with certainty factors further. We can do conflicts analysis by means of HLPN with certainty factors.
4) A wide range of experiments are performed on a real dataset collected from the CASAS system [37] to validate the feasibility of our proposal. Experiment results indicate that the proposed approach performs well in accuracy.

The remainder of this paper is organized as follows: we first discuss related works in the following section. Then, in "Preliminaries", we present several relevant models of IoT service, IoT service event and Smart Home System. We address three types of conflict policies with formal method in Section "Formal definitions and conflicts". We detail a Petri nets-based mechanism for modeling and detecting IoT service conflicts in "Petri nets for IoT conflict policies modeling and verification", and conflicts detection is discussed in detail in "Conflict detection". In " Fuzzy Petri nets for fuzzy IoT conflict policies modeling and reasoning with certainty factors", we detail how to utilize high-level Petri nets to model and reason IoT conflicts with certainty factors. We evaluate our approach in "Evaluation" and conclude the work in "Conclusions and future work".

Related work

In this section, we discuss and analyze existing relevant approaches regarding conflicts detection of IoT services.

IoT service

Nowadays IoT technology is widely used in many fields, and scholars have carried out a lot of research work on this topic, such as IoT service selection, IoT service discovery, IoT service conflicts detection and so on. For example, Baranwal et al. presented a framework for IoT service selection, which considers users' requirements regarding QoS (quality of service) [5]. Since the number of IoT applications is growing day by day, it is really difficult to discover and locate appropriate IoT services. To overcome this problem, [6] specifically dealt with service discovery in the Internet of Things. Now IoT is also widely used in intelligent transport systems, where security and trustworthiness are two key issues that must be addressed. In [7], Dass et al. proposed a trustworthy service provisioning scheme for IoT-based intelligent transport systems. In addition, IoT technology contributes much to our intelligent home (i.e., Smart Home), and a great deal of relevant work has been done on the problem of learning and reasoning in this thesis. For instance, Kim et al. presented a method to learn user behaviors, and addressed how to provide robot-IoT services in a Smart Home [8].

Conflict detection techniques of IoT service

Despite a substantial number of successful cases of IoT in Smart Home, there are still several key issues to be worked out. Moreover, in a smart home, it is a crucial factor to deal with the multi user conflict over IoT applications.

In [9], Chaki et al. presented a framework to detect conflicts among IoT services, where the functional and non-functional properties of IoT services both are considered. However, this work pays more attention to "opposite conflicts" than other types of conflict. Similarly, Farooq et al. proposed a formal method (i.e., IoTC^2) for detecting conflicts in IoT systems, and they verify policies by means of built-in model verification blocks of MATLAB [10]. And, their work did not consider QoS properties of IoT services. In [11], every networked appliance of a smart home was regarded as an object, and a model was presented to detect feature interactions (FIs). However, this work cannot achieve our Non-functional Conflict Policies (detailed in Definitions 7–8). In a smart space, there may be multiple apps, which can operate sensors and actuators to bring better experience to users. However, conflicts may occur if two or more apps are running simultaneously, especially when they use a same actuator or when they utilize different actuators changing the same environment parameter (e.g., the room temperature). Therefore, to solve this problem, Yagita et al. addressed a Kripke structure to detect conflicts [12]. In addition, a similar work was leveraged in [13], where an ontological framework was proposed to detect conflicts based on knowledge-based analysis. However, the Function Impact Conflict (in Definition 9) was ignored in [12] and [13]. In [14], the authors proposed a method to detect and resolve conflicts by means of a weighted-priority scheduling algorithm, where priorities are predefined. However, in an intelligent environment, the priorities are usually dynamic and context-dependent, rather than static. Through multi-users information sharing, [15] presented a method to detect and resolve conflicts, which is based on a kind of determination tree. Actually, [15] uses an ontology to describe applications and their services, but the ontology construction is important and really complicated. Another similar work is detailed in [45], where the authors proposed a tool of the behavior monitoring of IoT apps, i.e., IoTGuard. By means of IoTGuard users can be protected from unsafe and insecure device states. However, the approach cannot resolve policies that represent complex interactions between smart devices and residents. Munir and Stankovic presented an infrastructure DepSys, which can provide comprehensive strategies to specify, detect, and resolve conflicts in a smart home [46]. However, [46] also deals with conflicts by means of assigning priority to different services, but that only depends on the domain or administrator's understanding of each IoT service, among others. In [24], Shah et al. presented incompleteness conflicts, which indicate that those incomplete rules do not cover all possible sensor values. However, [24] did not consider our Qualitative Non-functional Conflict. To represent the relationships between ECA rules defining IoT services and environmental properties, a generic knowledge graph model was proposed in [25] was only evaluated statically on predefined rules, and the impact of rules in a real-time setting was also overlooked. Ahmed and Elfaki presented a method to detect conflicts among multi-resident smart-home services, which considered the functional properties of devices to distinguish conflicts [26]. Likewise, [26] did not consider our Non-functional Conflict Policies. In [27], Chi et al. presented a system named HomeGuard to resolve the problem of Cross-App Interference (CAI) threats. They have to translate the application's source code for performing model checking, but the method cannot detect violations mediated through physical channels. Carreira et al. also proposed a framework to do conflicts detection, which is based on the proposed conflict taxonomy and a CSP (Constraint Satisfaction Problem) [50]. In [51], Trimananda et al. detailed conflicts between apps on Samsung SmartThings, and they found three categories of app conflicts. However, [50] and [51], both cannot achieve the Function Impact Conflict (seen in Definition 9). Meanwhile, [47] gave the overview of conflict detection and resolution in IoT systems, where the authors surveyed the work already performed, and discussed the future challenges.

Through the above analysis on the conflict detection and resolution of IoT services, we can reach the conclusion that all the works above can only implement few conflict policies defined in Sect. "Formal definitions of conflicts". Moreover, none of the above works targets conflict detection by means of Petri net. To the best of our knowledge, state of the art solutions lack capability in modeling and reasoning fuzzy IoT conflict policies that our method offers.

Fuzzy rule reasoning and verification with Petri nets

In this subsection, we detail several works on fuzzy rule reasoning and verification that are based on Petri nets.

Wang and Zhu proposed a method to enhance the consistency verification ability of data transmission in Petri net, which is based on fuzzy c-means and spectral feature extraction [16]. In a rule-based system, Wu and Lee utilized graphs to describe the dependency relationship of the rules, and they used topological structures to do knowledge verification [17]. To resolve the problem of clone node attack on the Internet of Things network, Hameed et al. presented a method, where formal modeling, analysis and verification were all detailed [18]. Similarly, Li et al. proposed a data-driven hybrid petri-net (DDHPN) to establish the energy behavior meta-model, which made the most efficient use of gaussian kernel extreme learning machine [19]. In addition, in another work of Wu and

Table 1 Notations

Symbol	Description
IS	An IoT service
m	The number of functional properties
n	The number of non-functional properties
L	The service location
U	The service user
st	The service start time
et	The service end time
ET	The service execution time
p_1 and p_2	The service properties
P_i	A place
T_j	A transition
P	A set of places
T	A set of transitions
α	The degree of truth of a place
β	The degree of truth of an IoT service event
$\tilde{n}$	The number of transitions
$\check{n}$	The number of places

Lee, they proposed an enhanced high-level Petri net model in which variables and negative information can be represented and processed properly [20]. In [21], Wang presented a Timed Petri Net (TPN)-based mental workload evaluation model, which can improve task design and performance based on the evaluation results of individuals. Xu and Lin used colored Petri net (CPN) to verify the basic properties and the safety of train-centric communication-based train control (TcCBTC) system, and the method can identify the potential unsafe control paths [22].

In this work, we propose to employ high-level Petri nets to detect conflicts among IoT services. We borrow some ideas of rule verification to model the formal methods for conflict policies. Similar to the works detailed in Sect. "Fuzzy rule reasoning and verification with petri nets", we also use high-level Petri nets for handling fuzzy IoT conflict policies modeling and reasoning.

Preliminaries

In this section, we first detail the models of IoT service and IoT service event. Then, we present the model of Smart Home System (SHS). A lot of notations are used in the next models and experiments of the paper. To facilitate unified access to these notations, we list them in Table 1.

IoT service model

Appliances in a smart home exhibit the same behavior as represented in the service paradigm. Further, we model each appliance as an IoT service. In other words, each appliance has functionality (functional attribute) that deliberates with non-functional attributes (QoS). Actually, in a smart home, most IoT services are shareable, and they can be requested by multiple users simultaneously. So, there may be conflicts among them. For example, in a smart home, all the people can simultaneously enjoy the service of television, loudspeaker, light or AC, respectively. However, intelligent closestool, toaster, microwave oven, and treadmill, are all non-shareable IoT services. Since only one resident can enjoy one service at a time, there are no conflicts among those services of a home. In this work, we discuss only shareable IoT services.

Definition 1 (IoT Service) .An IoT service IS is a 4-tuple $IS = <IS.id, IS.name, IS.FP, IS.QP>$, where $IS.id$ and $IS.name$ are the identifier and the name of IS, respectively. $IS.FP$ is the set of functional properties of service IS, and $IS.FP = \{fp^1_{IS}, fp^2_{IS}, \cdots\cdots, fp^m_{IS}\}$ (Suppose IS has m functional properties). $IS.QP$ is defined as the set of $\{qp^1_{IS}, qp^2_{IS}, \cdots\cdots, qp^n_{IS}\}$ where each qp^i_{IS} denotes a non-functional property of service IS.

For example, in the bedroom, there is an air conditioner. For this air-conditioning service, it can be represented as < ac001, AC, {purifying the air, dehumidifying, heating, cooling}, {EER (energy efficiency ratio): 5.2(APF), refrigerating capacity: 2600 W, heating capacity: 4400 W, 3 years} > . {purifying the air, dehumidifying, heating, cooling} denotes the set of functional properties. However, the non-functional properties of the air-conditioning service are EER with 5.2, refrigerating capacity with 2600 W, heating capacity with 4400 W, and a three-year guarantee.

IoT service event

In a smart space, when a user requests an appliance service, suppose this appliance successfully responds to the user's request. In such a scenario, we call that an IoT service event happens. Generally speaking, an event can record all of the relevant information such as service user and spatiotemporal information.

Definition 2 (IoT service event) The IoT service event (ISE) of a IoT service IS can be represented as a tuple of $< \{IS.id, IS.FP, IS.QP\}, ET, L, U >$, where $IS.id$ is the identifier of IS, L and U are the location and the user of IS, respectively. For IS, its functional and non-functional properties are $IS.FP$ and $IS.QP$, respectively. ET denotes IS's execution time, and it can be denoted as $\{ET^{st}_{IS}, ET^{et}_{IS}\}$ where ET^{st}_{IS} and ET^{et}_{IS} are the start and end time of IS, respectively.

To better understand, here's an example. An IoT service event of a digital sound service is < {*sm001*, {*playing music*}, {*rock music*, *40 dB*}}, {*21:00*, *22:00*}, *main bedroom*, *us005* > . In the above tuple, *sm001*is the id of the digital sound service. Its functionality is playing music, and {*rock music*, *40 dB*} denote QoS attributes such as music type and volume (*40 dB*). The resident with the unique identifier *us005* is in the main bedroom. He/she turns on the digital sound at 21:00, but turns it off at 22:00. Hence the duration of the service is 22:00–21:00 = 1 h.

In a smart home, as time goes on, residents may request multiple IoT services. Moreover, the interactions of each resident are stored in a database, and they form an IoT service event sequence (ISES).

Definition 3 (IoT service event sequence) The IoT service event sequence of a smart home can be represented as a set *ises*, $ises = \{ise_1, ise_2, \cdots\cdots, ise_k\}$, where each ise_i is an IoT service event.

As mentioned before, each IoT service owns functional and non-functional attributes (QoS). However, among the attributes of services, there may be an effect on each other. For example, when a resident opens the intelligent curtain, the sunlight floods into the room, and that may affect the brightness attribute of a luminometer. Definition 4 shows the relationship between the properties of two IoT services.

Definition 4 (Relationships between attributes of IoT services) Given two attributes p_1 and p_2, there are two relationships between p_1 and p_2, i.e., direct dependency and indirect dependency relation. Direct dependency relations mean that changes inp_1 affect p_2. If p_1 affects an attribute p_3, and p_3 affects p_2, we affirm that they (p_1 and p_2) have an indirect dependency relation. For simplicity, the relationships between p_1 and p_2 are modeled as:

$$p2 \xrightarrow{d} p1$$

Definition 5 (overlapping events) In a smart home, some IoT service events may be the same or similar by their properties and functionality. If they happen in a same time frame, we call them overlapping events.

For example, Tom turned on the TV at 9 P.M. (named IoT service event ise_1), and he intends to go to bed at 10 P.M., hence the duration is 1 h. However, Alice would like to watch TV (named IoT service event ise_2) after she returned home at 9:30 P.M., suppose the duration is also one hour. So, as a result, there is a common part between the times of ise_1 (from 9 P.M. to 10 P.M.) and ise_2 (from 9:30 P.M. to 10:30 P.M.), and that is from 9:30 P.M. to 10 P.M. According to Definition 5, ise_1 and ise_2 are overlapping events. Assume there is only one TV, and then a conflict appears. Further,

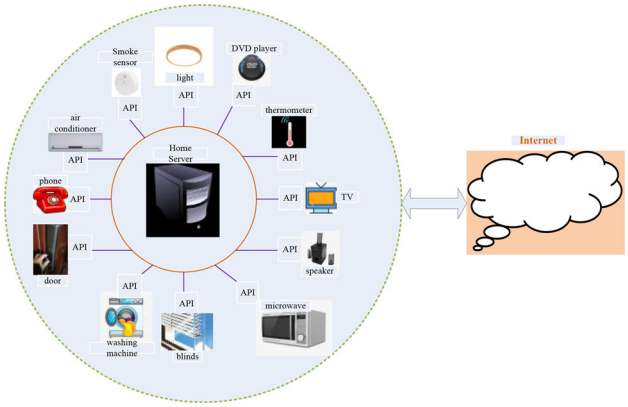


Fig. 3 A smart home system

the above conflict occurs since an individual service cannot satisfy the requirements of multiple users at the same time duration and location.

Smart home system (SHS)

In a smart home system (SHS), there are always multiple networked IoT devices, which are installed into SHS as software applications. By means of one or more control APIs, the residents can control the IoT services which are provided by those devices. For instance, we can utilize APIs of a light to control its brightness. We can also change the volume and channel of a TV with the help of its APIs.

As we can see from Fig. 3, it is a typical architecture of SHS. A SHS can access the external network by means of the home server. Moreover, the home server can also serve as an application server, which is a generalized server that exposes business logic to client applications through various protocols, possibly including HTTP [23]. Generally speaking, when a resident puts in a request for one IoT service of a SHS, the system uses the home server to invoke its appliance APIs.

Representing truth trigger conditions

In an IoT service event firing, response result of the event becomes truth and the trigger conditions may be true or false after the event has fired. For example, in a smart home, the smoke sets off the fire alarm. The smoke (trigger condition) remains true, unless we remove it by means of smoke exhausting skylight controlled by infrared detector. Different from the fire alarm, suppose a resident turns on the smart light, her or his trigger action disappears at the moment the light goes on. Based on the work of [31], containing a new token means that a trigger condition or a response result becomes true. When a transition is firing, it will consume

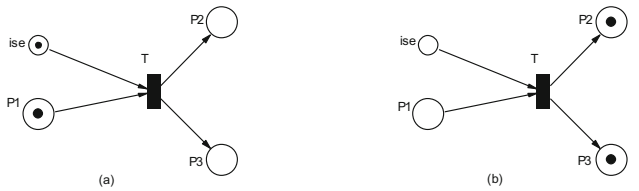


Fig. 4 The Petri nets mode (trigger condition P1 does not hold after T firing) of IoT service event *ISE*: P1 → P2∧P3 firing behavior **a** before the event firing and **b** after the event firing

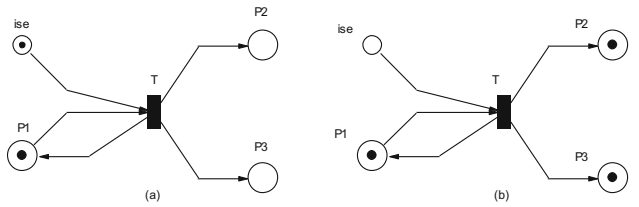


Fig. 5 The modified Petri nets mode (trigger condition P1 still holds after T firing) of IoT service event *ISE*: P1 → P2∧P3 firing behavior **a** before the event firing and **b** after the event firing

some tokens of each input place, and put tokens into each output place.

To let the truth conditions still hold after a transition firing, much work has been done. Among those works, there are two typical methods, i.e., adding extra arcs [32] and duplicating tokens [33, 34]. However, the former approach will make the net more complex due to the extra arcs. If the latter approach is used, the firing rules of Petri nets will be changed.

In a smart home, there may be several residents. So, there will be multiple IoT service events at the same time. In the work, we borrow the idea of [32] to model a transition firing in the design phase. Moreover, when we analyze a modeled Petri net, we do not take into account the extra arcs and places. So, some techniques of conventional Petri net analysis can still be used.

As shown in Figs. 4, 5, by means of the modified Petri nets model, we can simulate the behavior of the IoT service event *ISE*: P1 → P2∧P3 (P1 is a trigger condition, P2 and P3 both are response results, and → denotes the action of *ISE*) under the scenarios of trigger condition holding or not. The extra place (ise) is named the event place. Figure 4a indicates that condition P1 is true, then T of corresponding event *ISE* is enabled due to the event place with a token. Figure 4b expresses that response results P2 and P3 are true after T fires but the condition P1 does not hold the token again. In other words, the trigger condition becomes false. In Fig. 5, we add an extra arc from the transition T to the condition P1. As a result, after T fires, the trigger condition P1 still are true. The transition of each IoT service event fires once, and the other events can also use the condition P1 to fire after *ISE* used.

As we can see from Fig. 6, there are two IoT service events, i.e., ISE_1 and ISE_2, which own a common trigger condition P1. Suppose P1 is true, and after event ISE_1 fired (shown in Fig. 6b) then event ISE_2 still can fire (shown in Fig. 6c).

Formal definitions of conflicts

A lot of scholars have defined conflicts from different aspects. For example, Fisher declares that conflicts indicate incompatibility relationships among goals or values of several parties [28]. In most cases, conflicts are negative, and they usually make people's behavior less effective. However, when conflicts are well managed, they can breathe life and energy into our relationships, and can make us much more innovative and productive [29]. Similarly, in [30], the authors presented two types of conflict policies, i.e., no policy and a reward and punishment policy.

In this section, we detail three types of conflict policies of a smart home system that are functional conflict policies, non-functional conflict policies and controller safety policies. For the sake of understanding, the formal definitions of these types of conflicts are given, which are the foundation of the corresponding Petri Nets in Sect. "Petri nets for IoT conflict policies modeling and verification".

Functional conflict policies

In a smart home, there are usually several residents. Without loss of generality, suppose people have different preferences for each IoT service. For an example, one resident enjoys watching TV when he/she comes home from work, but another resident prefers it turned off since he/she is so tired. In this case, a functional conflict occurs. Moreover, most IoT services have two opposite functional states, e.g., startup and shutdown. Generally speaking, a functional conflict means that some residents expect an IoT service to work, but at the same time the others would rather the IoT service does not implement its functionality.

Definition 6 (functional conflict) Given two users U_1 and U_2, one IoT service *IS*, functional conflict (*fc*) is:

$$fc \mapsto (IS_{U_1}^{\pm} \wedge IS_{U_2}^{\mp}) \vee (IS_{U_1}^{\mp} \wedge IS_{U_2}^{\pm})$$

where $\pm$ and $\mp$ denote two opposite functional states of *IS*.

Non-functional conflict policies

Suppose there is only one appliance with a specific function in a separate location of a smart home. For instance, each room is usually equipped with only one air conditioner or

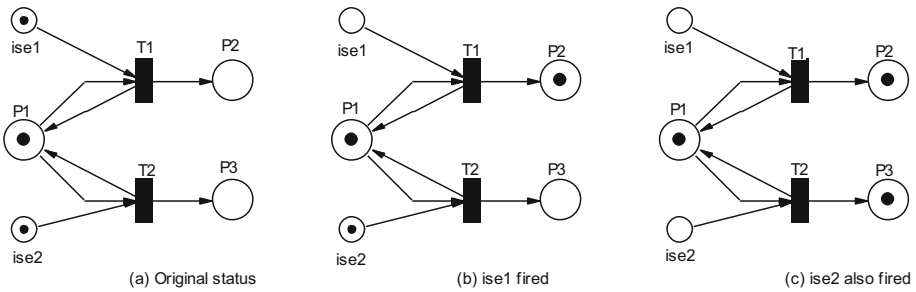


Fig. 6 A model of trigger conditions reused

one TV. Most obviously, two or more different QoS preferences cannot be satisfied concurrently. So, the non-functional conflict usually occurs among the residents. However, similar to QoS attributes of services computing, each IoT service may also have two distinct non-functional properties, i.e., the qualitative property and the quantitative property. So, in this work, we discuss two types of non-functional conflicts, i.e., the qualitative non-functional conflict and the quantitative non-functional conflict.

QoS is a frequently mentioned concept in services computing, where several typical QoS attributes include Reliability (the success probability of a service), Reputation degree (the recognition extent of users to service), Availability (a percentage of time, that indicates when the service is operable) and so on. In this work, we take the non-functional properties of an IoT service as QoS attributes. Similarly, examples of non-functional properties of an IoT service include size, power consumption rate, durability. However, except for several typical features, different IoT services may have a few different non-functional properties. For example, a light service owns luminosity level, but a stereo service has a property known as volume.

Definition 7 (qualitative non-functional conflict) Given two users U_1 and U_2 (if the number of people is more than 2, we get the same result), for the same IoT service IS, if there is at least one qualitative QoS attribute which is different between the preferences of U_1 and U_2, a qualitative non-functional conflict occurs and it (*lnfc*) is defined as follows:

$$lnfc = \exists qp^k_{IS} \in IS.QP,\ qp^k_{U1,\,IS} \neq qp^k_{U2,\,IS}$$

where the IoT service events are overlapping events, and the two users are at the same location.

Definition 8 (quantitative non-functional conflict) Then, similar to Definition 7, at the same place, given two users U_1 and U_2, a quantitative non-functional conflict (*nnfc*) is denoted as:

$$nnfc = \exists qp^i_{IS} \in IS.QP,\ qp^i_{U1,\,IS} \neq qp^i_{U2,\,IS}$$

where $qp^i_{U1,\,IS}$ and $qp^i_{U2,\,IS}$ are quantitative QoS properties of IS, and the IoT service events are also overlapping events.

Controller safety policies

In an IoT system, by means of data collection from sensors, the controller can generate actuation commands for the appropriate actuators. Actually, when multiple IoT services are accessed at the same time, a service impact conflict may occur, since there may be a function impact (function impact conflict) or a QoS impact (QoS impact conflict) between any two IoT services. In other words, if two controllers trigger actions of two different actuators that affect their function or QoS features at the same time, there will be a function impact conflict or a QoS impact conflict, respectively.

Definition 9 (function impact conflict) Given two users U_1 and U_2, and two IoT services IS_1 and IS_2, function impact conflict (*fic*) is:

$$fic \mapsto (IS^{\oplus}_{1,\,U_1} \wedge IS^{\otimes}_{2,\,U_2}) \vee (IS^{\otimes}_{1,\,U_1} \wedge IS^{\oplus}_{2,\,U_2})$$

where $\oplus$ and $\otimes$ denote two interdependent functional properties of IS_1 and IS_2. Suppose there is a function impact conflict between two IoT services, they do not work properly.

Definition 10 (QoS impact conflict) Given two users U_1 and U_2, and two IoT services IS_1 and IS_2, QoS impact conflict (*qic*) is:

$$qic \mapsto (IS^{f1}_{1,\,U_1} \wedge IS^{f2}_{2,\,U_2}) \wedge ((f1 = f2) \vee (f1 \leftrightarrows f2))$$

where f_1 and f_2 are two QoS features, and they are the same (denoted as $f1 = f2$) or dependent features (denoted as $f1 \leftrightarrows f2$).

For example, in a bedroom of a smart home, suppose there are a window opener and a thermostat, which are two different actuators and can be controlled by different controllers. If they are triggered at the same time, a QoS impact conflict occurs, since they can affect the same feature (temperature) of the room.

Petri nets for IoT conflict policies modeling and verification

In this section, we detail a Petri nets-based mechanism for modeling and detecting IoT service conflicts of a smart home. As shown in Fig. 2, our conflicts detection consists of three phases: conflict policies normalization, conflict policies transformation, and IoT service conflicts detection. We borrow the idea of [31] to model and detect conflicts, where transitions are used to represent IoT service events (ISEs), and the input and output places of a transition are used to represent conditions and service response result of the policy, respectively. If an input place has a token, the corresponding condition is true. However, if an output place owns a token, the corresponding response result becomes true. Suppose that an IoT service is activated, its trigger conditions must be satisfied and the corresponding transition is enabled. After a transition is fired, its output places will get a certain number of tokens. We detail our method of handling truth value of trigger condition/response result in next section. For conflicts detection of IoT services, we will present two analysis approaches, i.e., the static analysis and the dynamic simulation of Petri net models.

IoT service event normalization

As for each IoT service event, it may have one or more trigger conditions, and can also produce multiple response results. Without loss of generality, we borrow the idea of predicate logic and Horn clause logic [35, 36] to model IoT service events. For the sake of convenience, the normalization of an IoT service event is performed before we transform it to the corresponding Petri net.

In a smart home, any service event can be described as a logical implication as the following two forms or their combinations:

- $P1 \wedge P2 \wedge \cdots \wedge P_{i\text{-}1} \rightarrow P_i \wedge P_{i+1} \wedge \cdots \wedge P_j$.
- $P1 \vee P2 \vee \cdots \vee P_{i\text{-}1} \rightarrow P_i \wedge P_{i+1} \wedge \cdots \wedge P_j$.

where ' $\rightarrow$ ' denotes the action of one IoT service event, the places just to the left of ' $\rightarrow$ ' are trigger conditions, but the places on its right denote the corresponding response results.

Definition 11 (Normalization 1) IoT service events with the logical implication of $P1 \wedge P2 \wedge \cdots \wedge P_{i\text{-}1} \rightarrow P_i \wedge P_{i+1} \wedge \cdots \wedge P_j$ can be deconstructed to as a set of predicate logics with the form of $P1 \wedge P2 \wedge \cdots \wedge P_{i\text{-}1} \rightarrow P_k$, $1 < i \leq k \leq j$.

The derivation is as follows:

$$
\begin{aligned}
&P1 \wedge P2 \wedge \ldots \wedge P_{i-1} \rightarrow P_i \wedge P_{i+1} \wedge \ldots \wedge P_j \\
&\equiv \neg(P1 \wedge P2 \wedge \ldots \wedge P_{i-1}) \vee (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j) \\
&\equiv (\neg(P1 \wedge P2 \wedge \ldots \wedge P_{i-1}) \vee P_i) \wedge \\
&\quad (\neg(P1 \wedge P2 \wedge \ldots \wedge P_{i-1}) \vee P_{i+1}) \wedge \ldots \wedge \\
&\quad (\neg(P1 \wedge P2 \wedge \ldots \wedge P_{i-1}) \vee P_j) \\
&\equiv (P1 \wedge P2 \wedge \ldots \wedge P_{i-1} \rightarrow P_i) \wedge \\
&\quad (P1 \wedge P2 \wedge \ldots \wedge P_{i-1} \rightarrow P_{i+1}) \wedge \ldots \wedge \\
&(P1 \wedge P2 \wedge \ldots \wedge P_{i-1} \rightarrow P_j)
\end{aligned}
$$

In a smart home, an IoT service can be started successfully, if and only if all the prerequisites of the service are met. As shown in the Definition 11, the place P_l ($1 \leq l \leq i$-1) denotes a trigger condition, and the places from P_i to P_j are service results. The places on the either side of ' $\rightarrow$ 'are connected by '$\wedge$', which means a relationship of logic and. In other words, after a service is started, all the service results occur at the same time. So, the service results can be split into several independent individuals, which own the same trigger conditions. The above derivation verified the validity of the conclusion of the Definition 11.

Definition 12 (Normalization 2) IoT service events with the logical implication of $P1 \vee P2 \vee \cdots \vee P_{i\text{-}1} \rightarrow P_i \wedge P_{i+1} \wedge \cdots \wedge P_j$ can be deconstructed to as a set of predicate logics with the form of $P_k \rightarrow P_i \wedge P_{i+1} \wedge \cdots \wedge P_j$, $1 \leq k \leq i$-1.

The derivation is as follows:

$$
\begin{aligned}
&P1 \vee P2 \vee \ldots \vee P_{i-1} \rightarrow P_i \wedge P_{i+1} \wedge \ldots \wedge P_j \\
&\equiv \neg(P1 \vee P2 \vee \ldots \vee P_{i-1}) \vee (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j) \\
&\equiv (\neg P1 \wedge \neg P2 \wedge \ldots \wedge \neg P_{i-1}) \vee (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j) \\
&\equiv (\neg P1 \vee (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j)) \wedge \\
&(\neg P2 \vee (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j)) \wedge \ldots \wedge \\
&(\neg P_{i-1} \vee (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j)) \\
&\equiv (P1 \rightarrow (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j)) \wedge \\
&(P2 \rightarrow (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j)) \wedge \ldots \wedge \\
&(P_{i-1} \rightarrow (P_i \wedge P_{i+1} \wedge \ldots \wedge P_j))
\end{aligned}
$$

Different from the case that the Definition 11 indicates, in a smart home, an IoT service event may be triggered by multiple conditions. Furthermore, one or more of these conditions can trigger the service, which is described by the logical connective '$\vee$' in the Definition 12. Similar to the Definition 11, when an IoT service event occurs, multiple results may be produced simultaneously. As a result, '$P_k \rightarrow P_i \wedge P_{i+1} \wedge \cdots \wedge P_j$' can embody the above mapping between an individual trigger condition and service results. The derivation of the Definition 12 further verified the above conclusion.

We conclude the normalizations of IoT service events as the following four types:

- Type 1: $P1 \rightarrow P2 \wedge P3 \wedge \cdots \wedge P_i$.

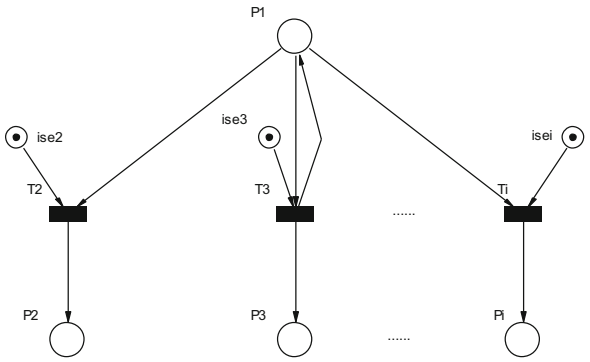


Fig. 7 The transformation model of Type1

- Type 2: P1 $\wedge$ P2 $\wedge\cdots\wedge$ $P_{i-1} \rightarrow P_i$.
- Type 3: P1 $\vee$ P2 $\vee\cdots\vee$ $P_{i-1} \rightarrow P_i$.
- Type 4: P1 $\wedge$ P2 $\wedge\cdots\wedge$ $P_{i-1} \rightarrow P_i \wedge P_{i+1} \wedge\cdots\wedge P_j$.

Transformation from service events to petri net models

Then in this section, we detail how to transform the above four types of IoT service events to Petri nets, which are used as references for Petri nets construction of IoT conflict policies in the next section.

The corresponding Petri net model of Type 1 is presented in Fig. 7, where P1 is the trigger condition, and the places from P2 to P_i are all response results. Based on the logically equivalence and Horn clause, Type 1 can be deconstructed to i–1 subevents (i.e., P1 $\rightarrow P_k$, $2 \leq k \leq i$), which have the same trigger condition P1. So, in Fig. 7, all the transitions have the same input place P1. As we can see from Fig. 7, P1 has an arc to T3, but T3 also has an arc to P1. In other words, T3 fires cannot change the value of the trigger condition.

The corresponding Petri net models of Type 2 and Type 4 are presented in Figs. 8 and 9, respectively. Each IoT service event of Type 2 and Type 4 has multiple trigger conditions. However, Type 2 has only one response result, and Type 4 has multiple response results. For example, suppose a user puts some food into the microwave oven (Condition 1), then sets heating time (Condition 2), he/she finally presses the start button (Condition 3). Only if the three conditions are met, the microwave oven service is triggered. After the service finished, the user can get hot food (the response result).

Finally, according to the logically equivalence and Horn clause, Type 3 can be normalized into i–1 subevents (i.e., $P_k \rightarrow P_i$, $1 \leq k \leq i$-1), which have the same response result P_i. As shown in Fig. 10, which is a transformation model of Type3, T1 and T_{i-1} both have an arc to their input places respectively, i.e., P1 and P_{i-1}. Generally speaking, after T1 and T_{i-1} fired, P1 and P_{i-1} still hold their tokens. The corresponding trigger

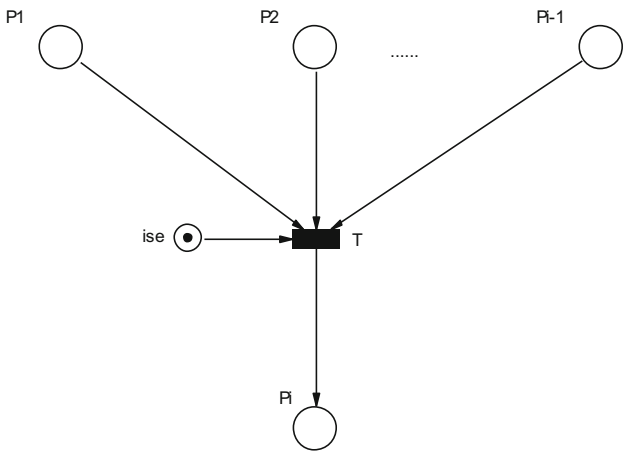


Fig. 8 The transformation model of Type2

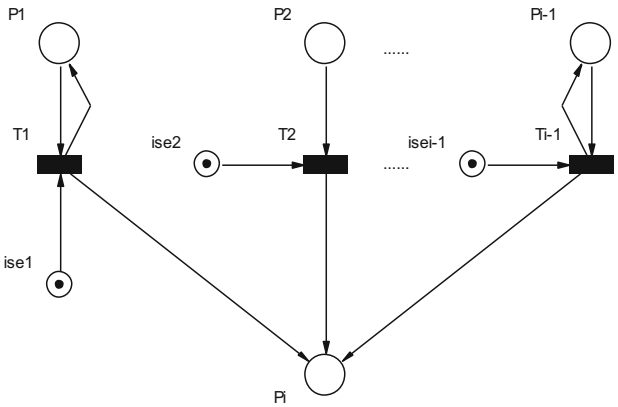


Fig. 9 The transformation model of Type3

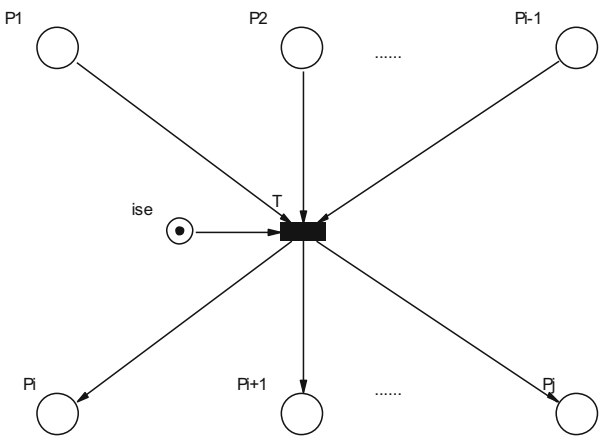


Fig. 10 The transformation model of Type4

conditions that P1 and P_{i-1} represent, in other words, are always true.

To help the reader understand, as shown in Fig. 11, we give the algorithm (IoT Service Events Transformation to

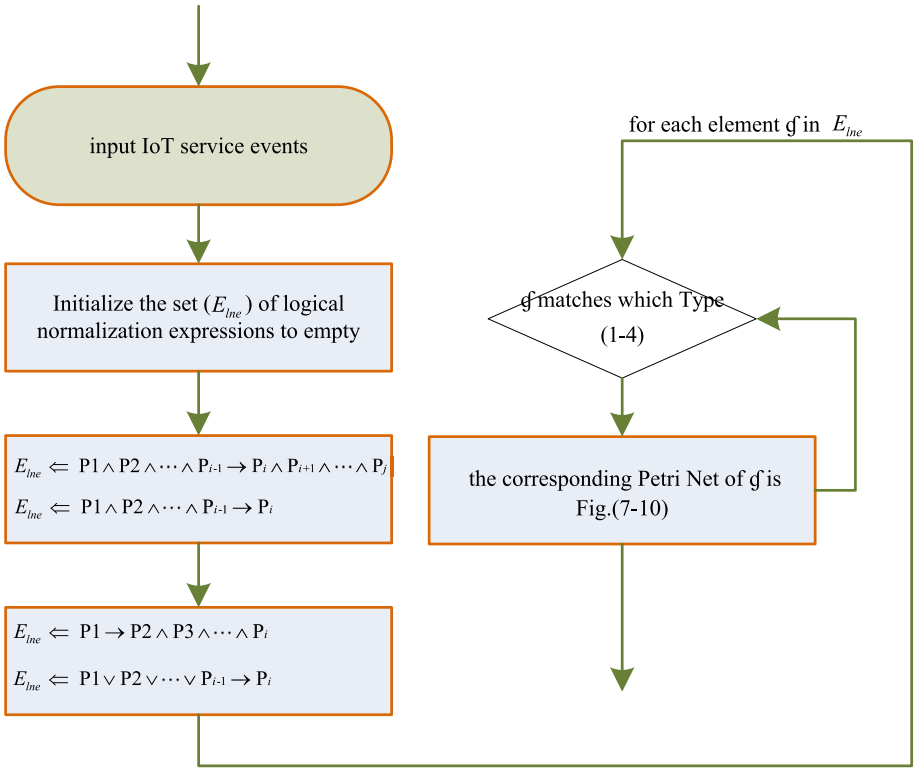


Fig. 11 The flowchart of algorithm *ETPN*: IoT service events transformation to petri nets

Petri Nets: *ETPN*) flowchart of how to transform an IoT service event to a Petri net, which consists of two steps, i.e., IoT Service Events Normalization and Logical Normalization Expressions Transformation.

Transformation from IoT conflict policies to petri net models

In Sect. "Formal definitions of conflicts", we have identified five conflict policies of IoT services, which are described in Definitions 6–10, respectively. In this subsection, we detail how to transform each conflict policy to a Petri net model. When we discuss conflict detection later in this article, we transform the actions of any two people to a Petri net, and search for a match from those confliction patterns detailed in this section. If the Petri Net matches one pattern, there is a corresponding conflict between the two people. Otherwise, there is no conflict between them.

In Definition 6 of Sect. "Formal definitions of conflicts", we gave a definition about functional conflict. Simply speaking, when two different residents would like to use a same IoT service at the same time, a functional conflict occurs. Figure 12 is a transformation model on functional conflict, where P1 and P2 correspond to the conditions of two different users, respectively. Moreover, P1 and P2 both have a token. That is to say, the two users both would like to use the IoT service, where an extra place *fc* is added. However,

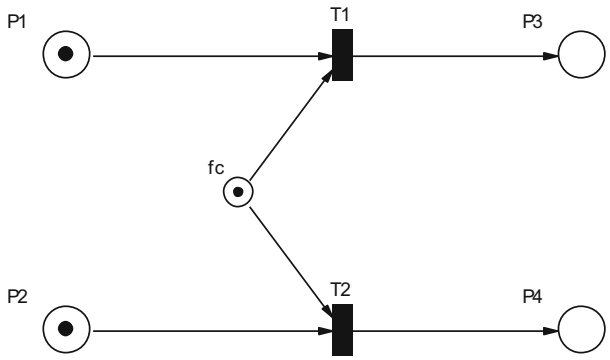


Fig. 12 The transformation model on functional conflict

the place *fc* has only one token. So, suppose T1 fires, T2 will fail, and vice versa. As a result, a functional conflict occurs.

We detailed the non-functional conflict policies in Sect. "Conflict detection techniques of IoT service", which is divided into two types, i.e., qualitative non-functional conflict and quantitative non-functional conflict. Figure 13 is a transformation model on qualitative non-functional conflict, where P1 and P2 correspond to two different users and their wishes. The extra place *lnfc* denotes one qualitative non-functional attribute of the IoT service, and it has only one token. If T3 fires (it does not consume P1's token), the place $lnfc_1$ will get a token, which means that the user P1 indicated will get access to the IoT service. Meanwhile, it is obvious that T2 has no chance to fire, since T3 consumed *lnfc*'s token,

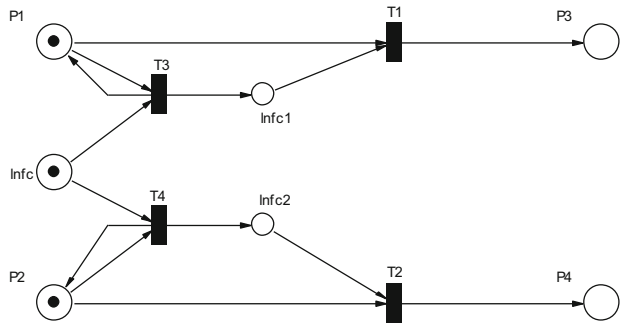


Fig. 13 The transformation model on qualitative non-functional conflict

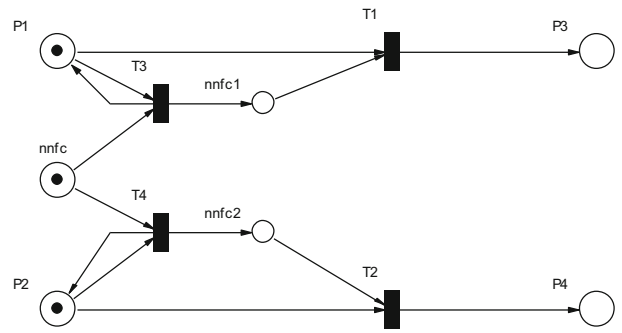


Fig. 14 The transformation model on quantitative non-functional conflict

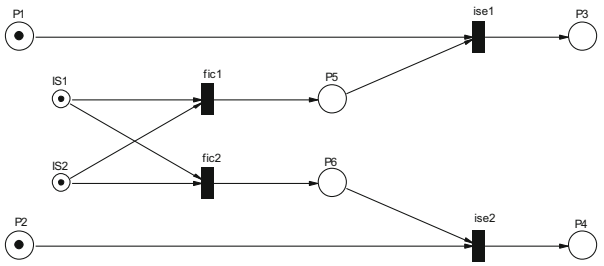


Fig. 15 The transformation model on function impact conflict

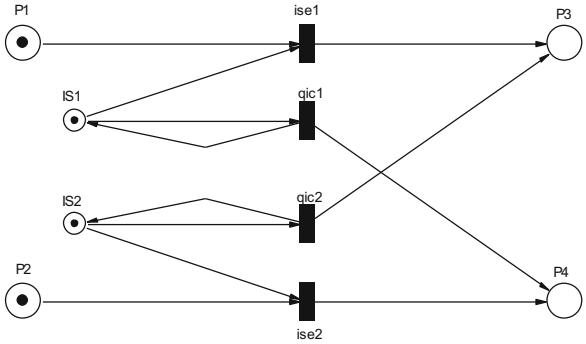


Fig. 16 The transformation model on QoS impact conflict

which will result in failure of T4. So, if T3 fires, the place $lnfc_2$ will never get a token. In this case, the firing condition of T2 can never be satisfied and a qualitative non-functional conflict occurs.

As we can see from Fig. 14, it denotes a transformation model on quantitative non-functional conflict. The analysis process is similar to the core idea of Fig. 13. Simply speaking, the places $nnfc_1$ and $nnfc_2$ decide that there is only one place of P1 and P2 that can fire at one point of time. In other words, at the same time, suppose two people have different preferences for one quantitative non-functional attribute of a same IoT service, only one person can use the IoT service. That is, a quantitative non-functional conflict occurs.

In a smart home, there may be a function impact or a QoS impact between two IoT services. The two impact relationships were detailed in Sect. "Fuzzy rule reasoning and verification with petri nets". When two different residents would like to access two different IoT services, a service impact conflict (i.e., Function Impact Conflict or QoS Impact Conflict) may occur. Figure 15 is the transformation model on function impact conflict. The places P1 and P2 denote two different users U_1 and U_2, who would like to access the IoT services that the places IS_1 and IS_2 denote, respectively. If the transition *ise1* fires, U_1 can use the service that IS_1 denotes successfully. In the same way, the transition *ise2* decides whether U_2 can use the service that IS_2 denotes successfully. However, *ise1* and *ise2* depend on the tokens of P5 and P6, respectively. And, P5 and P6 are the output places of the transitions *fic1* and *fic2*, respectively. As shown in Fig. 15, IS_1 and IS_2 both are the input places of *fic1* and *fic2*. Moreover, since IS_1 and IS_2 both have only one token, at any given moment only one of *fic1* and *fic2* can fire. Generally speaking, they form an interlocking structure, which consists of IS_1, IS_2, *fic1* and *fic2*. So, if *ise1* fires, *ise2* cannot fire, and vice versa. In other words, there is a Function Impact Conflict between the two IoT services above, and they cannot perform properly at the same time.

Finally, we present a transformation model on QoS impact conflict in Fig. 16. The places P1 and P2 still denote two different users U_1 and U_2, who would like to enjoy the services that the places IS_1 and IS_2 denote, respectively. P3 and P4 correspond to the QoS (Quality of Service) status of the two IoT services after they performed (i.e., after *ise1* and *ise2* fired), respectively. As we can see from Fig. 16, the transitions *qic1* has an arc to P4, and *qic2* has an arc to P3. Specifically, the implementations of the two IoT services directly affect the each other's QoS status. That is, a QoS impact conflict occurs.

In effect, most people have fixed habits of thought and action. For example, the first thing is to turn on the TV and set the TV Volume 30 when Tom gets home. But Alice after she comes back home, prefers firstly enjoying a stereo service with 45 dB, then she turns on the coffee machine. To improve

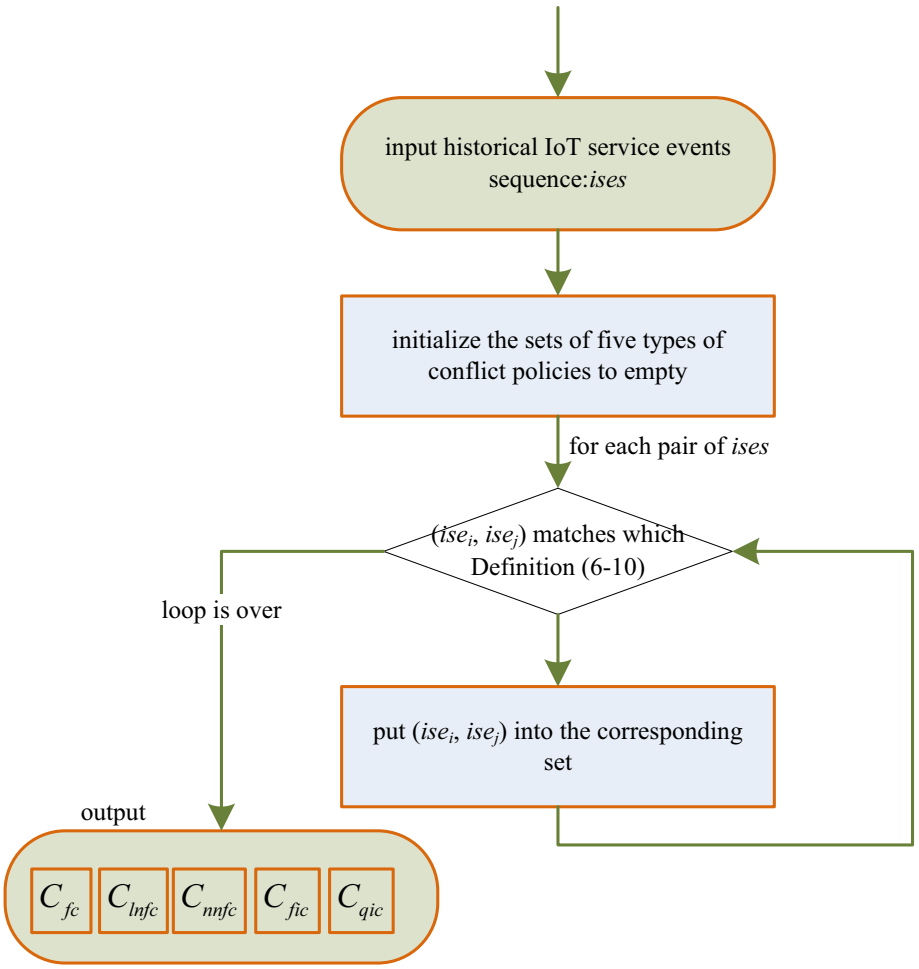


Fig. 17 The flowchart of algorithm *CRC*: conflict rules construction

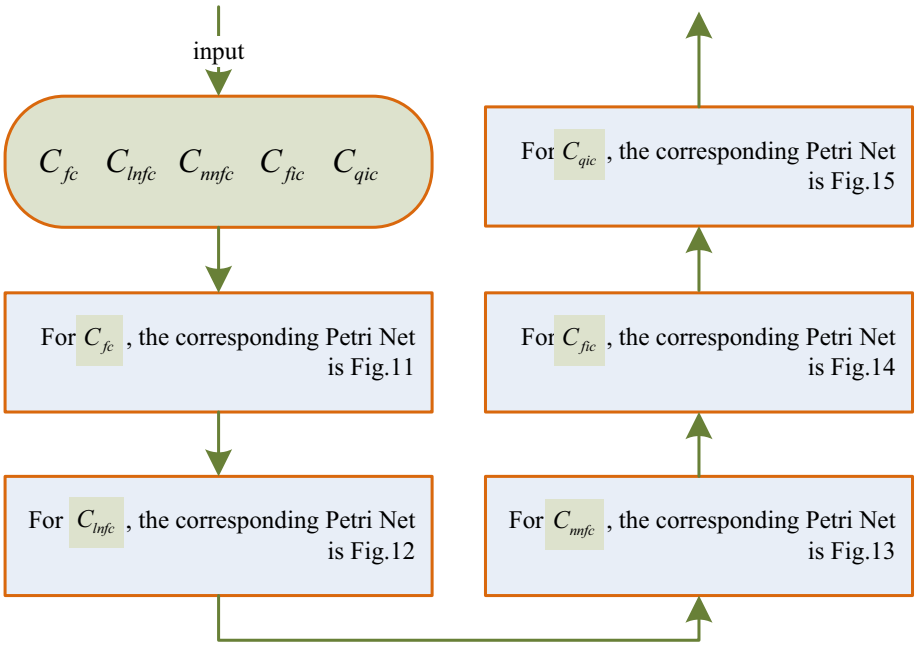


Fig. 18 The flowchart of algorithm *CPTPN*: conflict policies transformation to petri nets

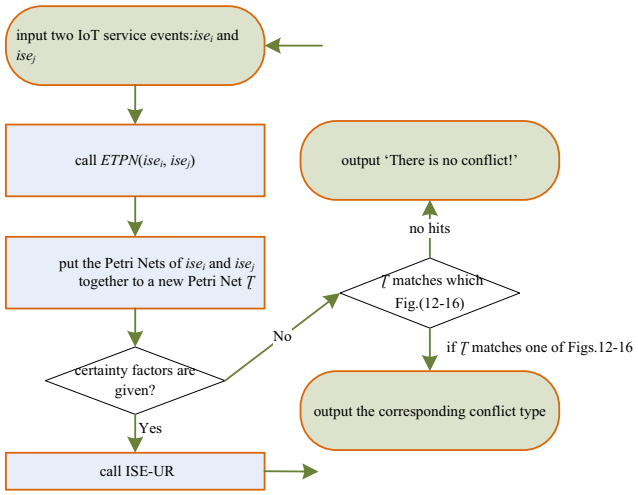


Fig. 19 The flowchart of algorithm *CD*: conflict detection

Table 2 Places and transitions description of the formalism example

Symbol	Description
P1	9 p.m
P2	Tom is watching TV
P3	Alice is sleeping
P4	The light is on
P5	The light is off
P6	Tom is at home
P7	Alice is at home
T1	Look at the time
T2	Tom clicks on the television
T3	Alice goes to bed
T4	Turn on the light
T5	Turn off the light

the search and match efficiency for the permanent residents in a smart home, we construct the rule base in advance. Specifically, we use Algorithm *CRC* (Conflict Rules Construction) to construct five typical types of conflicts, where the historical IoT service events are scanned and analyzed. Then, we use Algorithm *CPTPN* (Conflict Policies Transformation to Petri Nets) to detail how to transform conflict policies to Petri Nets. Figures 17 and 18 are the flowcharts of *CRC* and *CPTPN*, respectively. Significantly, given a historical data, the *CRC* algorithm and *CPTPN* algorithm need to be run only once, and their role is to provide the Petri net templates of the five types of conflicts.

Conflict detection

Simply put, we have two ways to do conflicts detection, i.e., solving linear equations and template matching. In the process of study of conflict rule models, we have found, accidentally, based on the incidence matrix and state equation, solution vectors can be used to judge if there is a conflict. It is just amazing, and this finding should be one of our contributions. However, solving linear equations is really complicated.

Figure 19 presents the flowchart of algorithm *CD* (Conflict Detection), which is the main program. Given two IoT service events, ise_i and ise_j, ISE-UR (the method of IoT service events uncertainty reasoning, detailed in Sect. "Uncertainty reasoning algorithm") is called if certainty factors are provided. Otherwise, after the Petri Nets of ise_i and ise_j are put together to a new Petri Net $\mathcal{T}$, we compare$\mathcal{T}$ with the Petri Nets of five typical types of conflicts, and determine whether there is a conflict between ise_i and ise_j.

The method based on solving linear equations

In this subsection, we will use an example to detail how to formalize high-level Petri nets of IoT Service Events. The formalism method is based on the incidence matrix and state equation discussed in [31]. We judge if there are conflicts among multiple IoT service events in an algebraic form. Generally speaking, our approach consists of three steps, i.e., PN transformation of service events, incidence matrix calculation and solving of state equation.

Let's have a look at an example using this formalism approach in the context of a smart home where there may be IoT service conflicts. Suppose there are two residents in a family, Tom and Alice. When it is 9 p.m., if Tom is at home, he would like to watch TV and prefers the light (let there be only one light service) on. But Alice will go to bed and prefers the light off at the same time. For simplicity, suppose Tom was the first to arrive home, and he began to watch TV at 9 p.m. Then we judge whether Alice could go to bed with the light off successfully. We borrow the idea of predicate logic and Horn clause logic to model the above scene. Suppose $\bar{\mathrm{U}} = \{\rightarrow \mathrm{P1}, \rightarrow \mathrm{P6}, \mathrm{P1}\wedge\mathrm{P6} \rightarrow \mathrm{P2}, \mathrm{P2} \rightarrow \mathrm{P4}, \mathrm{P3} \rightarrow \mathrm{P5}\}$ denotes the initial scenario, and $\mathring{\mathrm{U}} = \{\mathrm{P1}\wedge\mathrm{P7} \rightarrow \mathrm{P3}\}$ denotes the query. $\bar{\mathrm{U}}$ and $\mathring{\mathrm{U}}$ combine to form $\breve{\mathrm{U}} = \{\rightarrow \mathrm{P1}, \rightarrow \mathrm{P6}, \mathrm{P1}\wedge\mathrm{P6} \rightarrow \mathrm{P2}, \mathrm{P2} \rightarrow \mathrm{P4}, \mathrm{P3} \rightarrow \mathrm{P5}, \mathrm{P1}\wedge\mathrm{P7} \rightarrow \mathrm{P3}\}$ (P1, P2, ⋯, and P7 are all places). For the sake of clarity, all the places and transitions of this example are detailed in Table 2. We can derive answers to the query $\mathring{\mathrm{U}}$ following four steps [42].

Firstly, we combine the initial scenario $\bar{\mathrm{U}}$ and the query $\mathring{\mathrm{U}}$ into the set $\breve{\mathrm{U}}$. We then transform $\breve{\mathrm{U}}$ into a Petri net (as shown in Fig. 20) based on the transformation method described in Sect. "Transformation from service events to Petri net models". If $\breve{\mathrm{U}}$ can satisfy the query $\mathring{\mathrm{U}}$, the transition T3 must fire at

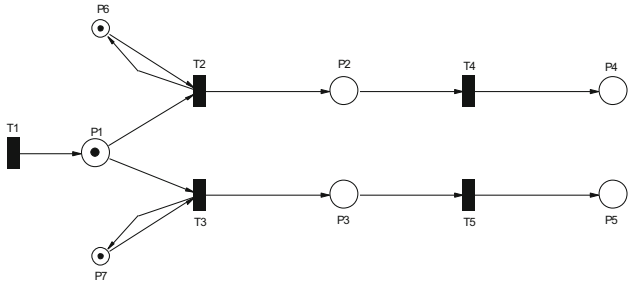


Fig. 20 The corresponding Petri net of Ŭ

least once. In other words, if we would like to judge whether Alice can go to bed with the light off successfully in the context that Tom is watching TV, we only need to know whether the transition T3 can fire at least once under the constraints of Petri net theory.

Then, we calculate the incidence matrix R of the Petri net shown in Fig. 20, which is a ñ × ň (ñ and ň are the numbers of transitions and places, respectively) matrix. $R = R^{\triangleright} - R^{\triangleleft}, R^{\triangleright} = [r^{\triangleright}_{i,j}]$, and $R^{\triangleleft} = [r^{\triangleleft}_{i,j}]$. The calculated values of $R^{\triangleright}$ and $R^{\triangleleft}$ are shown in Eqs. 1 and 2, respectively. In $R^{\triangleright}$, each element $r^{\triangleright}_{i,j}$ denotes the weight of arc from T_i to its output place P_j. However, each element $r^{\triangleleft}_{i,j}$ of $R^{\triangleleft}$, denotes the weight of arc to T_i from its input place P_j. For each element $r_{i,j}$ of R, it is defined as $r_{i,j} = r^{\triangleright}_{i,j} - r^{\triangleleft}_{i,j}$. If there is no arc connection between T_i and P_j, the value of $r_{i,j}$ is zero.

$$R^{\triangleright} = \begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \end{array} \begin{array}{c} \begin{array}{ccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \tag{1}$$

$$R^{\triangleleft} = \begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \end{array} \begin{array}{c} \begin{array}{ccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{2}$$

$$R = R^{\triangleright} - R^{\triangleleft} = \begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \end{array} \begin{array}{c} \begin{array}{ccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \tag{3}$$

Thirdly, solve X of the equation $R^T \ \mathrm{X} = \Delta M^T$. Suppose M_d is an intermediate or a final marking, from the initial marking M_0, we can get a firing sequence β. We have that $\Delta M = M_d — M_0$. $X = [x_1, x_2, \cdots, x_i, \cdots, x_{\tilde{n}}]$, where x_i represents the number of T_i firing in β. If X has positive integer solutions, M_d is a reachable marking from M_0. Given a query Ů, if it can be satisfied by the initial scenario Ū, the firing number of the corresponding transition (x_i) of Ů must be greater than one and can simultaneously satisfy the policies of Petri net theory. For the above example in Fig. 20, T3 is the transition of the corresponding query. Based on Table 2 and Ū, we get $M_0 = \{1, 0, 0, 0, 0, 1, 1\}$. and $M_d = \{0, 0, 0, 1, 1, 1, 1\}$. In M_d, the 5th element is 1, i.e., M(P5) = 1. That is to say, if the query Ů is successful, the light will be off (the corresponding place P5 has a token). So, $\Delta M = M_d — M_0 = \{-1, 0, 0, 1, 1, 0, 0\}$. $R^T \ \mathrm{X} = \Delta M^T$ is as follows:

$$\begin{bmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

By means of matrix–vector multiplication, we get the following:

$$\begin{cases} x1 - x2 - x3 = -1 \\ x2 - x4 = 0 \\ x3 - x5 = 0 \\ x4 = 1 \\ x5 = 1 \end{cases}$$

We further solve the above linear equations, and we can get $x2 = x4 = 1$. However, we can also get $x3 = x5 = 1$. So, it is completely obvious that $x2$ and $x3$ are in conflict with each other. Since $x3 = x2 = 1$, that indicates the transition T3 and the transition T2 can both fire at the same time. However, according to the theory of Petri net, only one of T3 and T2 can fire at some point because of a single token of the place P1. Moreover, as we can see from Fig. 20, based on the properties of Petri net, when, and only when, T3 fires, will T5 fire.

Since Tom was the first to arrive home, and he turned on TV at 9 p.m. In other words, T2 has already fired. Finally, through the above analysis, we can see, T3 fired unsuccessfully, and no more did T5. That is to say, the query Ů is unsuccessful. If we compare Fig. 20 with Fig. 12, we can get that there is a functional conflict in this formalism example, where the light service is the only IoT service, and it cannot be opened and closed at the same time.

Template matching method

In Sects. "Transformation from service events to Petri net models" and "Transformation from IoT conflict policies to Petri net models", we detailed how to transform IoT service events and conflict policies to Petri nets, respectively. We use the tool of Visual Object Net + + (version number, 1.44.2) to model Petri Nets. Visual Object Net + + is an innovative Petri-Net CAE Tool for PC that supports mixed continuous and discrete event Petri-Nets. It possesses many advanced characteristics, such as easy designing, quick simulation and uncomplicated documentation of hybrid Petri-Nets.

Given two residents in a smart home, we transform the actions of the two people to a Petri net, then we can do template matching to judge if there is a conflict between them. If the Petri net matches one conflict pattern detailed in Sect. "Transformation from IoT conflict policies to Petri net models", there is a corresponding conflict between the two people. Otherwise, there is no conflict between them. Concretely speaking, we can use two methods to do template matching. The first way is similar to isomorphism judgment between two graphs. Simply, if the Petri net has an arc from T_i to P_j (or from P_j to T_i), there will be an arc in the corresponding pattern Petri net. In the second case, we create two matrices with respect to all the transitions: input place matrix and output place matrix. In the Petri net denoting two residents' actions, suppose there is an arc from a place to a transition, the corresponding element of its input place matrix is assigned 1 (otherwise, the value is 0). Similarly, the value of an element in the output place matrix is 1 if an arc is from a transition to a place. As for the pattern Petri net, it also has an input place matrix and an output place matrix. We just need to compare their input place matrices and output place matrices, respectively. Due to space constraints, we're not going to give examples here.

The processing flow of algorithms

To clearly explain the arithmetic processing of IoT service conflict detection or fuzzy IoT service conflict modeling and reasoning, Fig. 21 shows a flow chart representing the five algorithms (i.e., *CRC*: Conflict Rules Construction, *ETPN*: IoT Service Events Transformation to Petri Nets, *CPTPN*: Conflict Policies Transformation to Petri Nets, *CD*: Conflict Detection, and *ISE-UR*: IoT Service Events Uncertainty Reasoning) for processing data. As we can see from Fig. 21, we construct conflict rules in advance by means of Algorithm *CRC*. Then based on Algorithm *CPTPN*, each class of conflict policy (the output of *CRC*) can be transformed to a Petri Net, and the Petri Nets of five typical types of conflicts form a set *C*. When two IoT service events (i.e., ise_i and ise_j) come, the pair (ise_i, ise_j) is used as input for Algorithm *CD*. And *CD* calls Algorithm *ETPN* to transform ise_i and ise_j to two Petri nets, which are put together to a new Petri Net ϒ. Now we judge whether certainty factors are given. If the answer is Yes, the system calls Algorithm *ISE-UR* to do uncertainty reasoning. Otherwise, we compare ϒ with the Petri Nets of five typical types of conflicts, and determine whether there is a conflict between ise_i and ise_j.

Fuzzy petri nets for fuzzy IoT conflict policies modeling and reasoning with certainty factors

In our experiments, we use a real dataset [37] to evaluate our approach. Actually, different people have different living habits. For example, in most cases, the first thing Tom does after work is to turn up the air conditioner. As for Alice, when she is back home, she would like to open the window, then put on her favorite songs. Generally speaking, in a smart home, each person uses a particular IoT service with a probability that can be learned from his/her daily life. And, in this work, we call those probabilities certainty factors, which can be learned from CASAS dataset [37].

In this section, we will detail conflict policies with certainty factors, which are renamed fuzzy policies. And, the corresponding Petri nets with certainty factors are called fuzzy Petri nets. We use high-level Petri nets (HLPN) to model and reason fuzzy IoT conflict policies. We expand the models in Sect. "Petri nets for IoT conflict policies modeling and verification". When we do conflicts analysis by means of HLPN in a Smart Home System (SHS) with certainty factors, transitions denote IoT service events. In an expanded HLPN, each transition is assigned with a certainty factor indicating the degree of truth of the corresponding IoT service event happening. Input and output places of a transition are used to represent triggering conditions and response results of an IoT service event, respectively. An enabled transition means the triggering conditions of the IoT service event have been satisfied and the IoT service event is activated. In this work, the degree of truth may be regarded as a weight or a probability value.

IoT service events with certainty factors

We borrow the idea of [38] to realize the fuzzy IoT service events and IoT conflict policies modeling in a Smart Home System, thus reason service conflicts with certainty factors. We classified IoT service events with certainty factors based on the four types as shown in Sect. "Petri nets for IoT conflict policies modeling and verification"A as follows:

- Type 1: ise_1^{β} : $\mathrm{P1}(\alpha 1) \rightarrow \mathrm{P2}(\alpha 2) \wedge \mathrm{P3}(\alpha 3) \wedge \ldots\ldots \wedge \mathrm{Pi}(\alpha \mathrm{i})$..

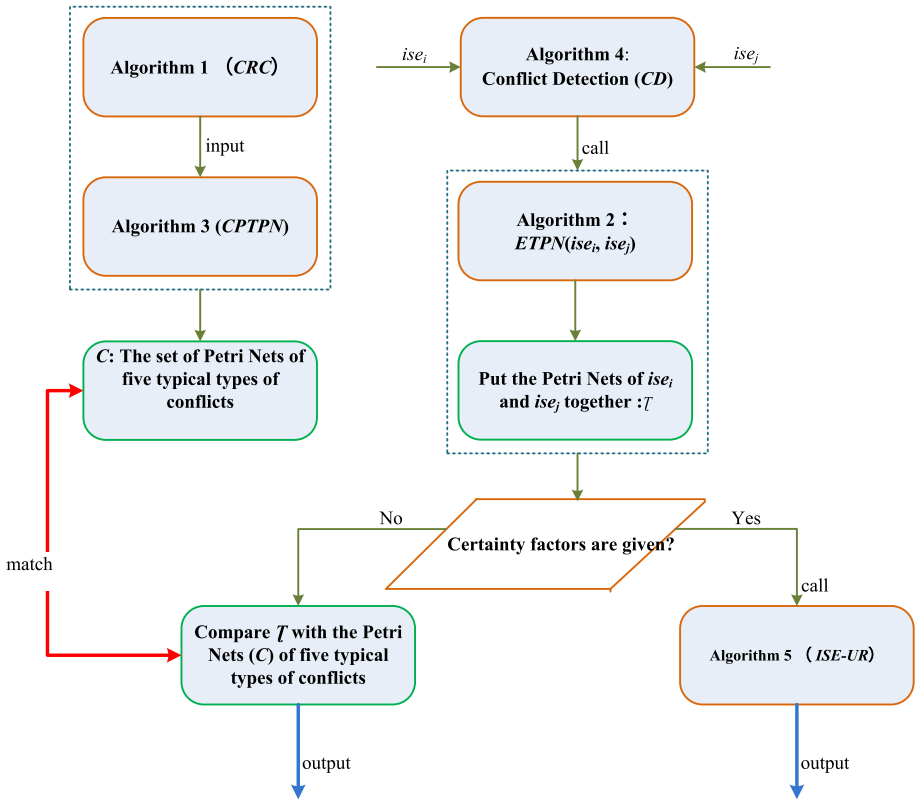


Fig. 21 Algorithms processing flow

- Type 2: ise_2^{β} : $\mathrm{P1}(\alpha 1) \wedge \mathrm{P2}(\alpha 2) \wedge \ldots\ldots\ldots \wedge \mathrm{Pi}\text{ - }1(\alpha \mathrm{i}\text{ - }1) \rightarrow \mathrm{Pi}(\alpha \mathrm{i})$..
- Type 3: ise_3^{β} : $\mathrm{P1}(\alpha 1) \vee \mathrm{P2}(\alpha 2) \vee \ldots\ldots\ldots \vee \mathrm{Pi}\text{ - }1(\alpha \mathrm{i}\text{ - }1) \rightarrow \mathrm{Pi}(\alpha \mathrm{i})$..
- Type 4: ise_4^{β} : $\mathrm{P1}(\alpha 1) \wedge \mathrm{P2}(\alpha 2) \wedge \ldots\ldots\ldots \wedge \mathrm{Pi}\text{ - }1(\alpha \mathrm{i}\text{ - }1) \rightarrow \mathrm{Pi}(\alpha \mathrm{i}) \wedge \mathrm{Pi}+1(\alpha \mathrm{i}+1) \wedge \ldots\ldots\ldots \wedge \mathrm{Pj}(\alpha \mathrm{j})$.

where $ise_i^{\beta}(i = 1, 2, 3, \text{or } 4)$ denotes the degree of truth of Type$_i$ event happening, and α_i denotes the degree of truth of triggering condition/response result P_i of a IoT service event *ise*.

The first type of IoT service event, i.e., Type 1, indicates that an independent condition can trigger one IoT service, and multiple results can appear simultaneously. Type 2 represents the type of IoT service events, which can only be triggered by multiple conditions together. In Type 3, one or more of a set of conditions can trigger this type of IoT service events, and a service result is returned. Finally, with respect to Type 4, only several conditions can together trigger an event and produce a set of results at the same time.

Then, we get the uncertainty reasoning results of the above four types of IoT service events:

- Type 1: ise_1^{β} : $\mathrm{P1}(\alpha_1) \rightarrow \mathrm{P2}(\alpha_2) \wedge \mathrm{P3}(\alpha_3) \wedge \ldots\ldots\ldots \wedge \mathrm{P_i}(\alpha_i)\alpha_2 = \alpha_3 = \ldots\ldots = \alpha_i = \beta \times \alpha_1$.
- Type 2: ise_2^{β} : $\mathrm{P1}(\alpha_1) \wedge \mathrm{P2}(\alpha_2) \wedge \ldots\ldots\ldots \wedge \mathrm{P_{i\text{ - }1}}(\alpha_{i\text{ - }1}) \rightarrow \mathrm{P_i}(\alpha_i)\alpha_i = \min\{\alpha_1, \alpha_2, \ldots\ldots, \alpha_{i\text{ - }1}\} \times \beta$.
- Type 3: ise_3^{β} : $\mathrm{P1}(\alpha_1) \vee \mathrm{P2}(\alpha_2) \vee \ldots\ldots\ldots \vee \mathrm{P_{i\text{ - }1}}(\alpha_{i\text{ - }1}) \rightarrow \mathrm{P_i}(\alpha_i)\alpha_i = \max\{\alpha_1, \alpha_2, \ldots\ldots, \alpha_{i\text{ - }1}\} \times \beta$.
- Type 4: ise_4^{β} : ise_4^{β} : $\mathrm{P1}(\alpha_1) \wedge \mathrm{P2}(\alpha_2) \wedge \ldots\ldots\ldots \wedge \mathrm{P_{i\text{ - }1}}(\alpha_{i\text{ - }1}) \rightarrow \mathrm{P_i}(\alpha_i) \wedge \mathrm{P_i}+1(\alpha_{i+1}) \wedge \ldots\ldots\ldots \wedge \mathrm{P_j}(\alpha_j)\alpha_i = \alpha_{i+1} = \ldots\ldots = \alpha_j = \min\{\alpha_1, \alpha_2, \ldots\ldots, \alpha_{i\text{ - }1}\} \times \beta$.

As for the above uncertainty reasoning result of Type 1, '$\alpha 2 = \alpha 3 = \ldots\ldots = \alpha i = \beta \times \alpha 1$' means that all service results have the same degree of truth, i.e., $\beta \times \alpha 1$. Since the service results of Type 1 appear equally at the same time, their degree of truth should obviously be equal. In Type 2, only all the trigger conditions are met, the corresponding IoT service event can occur and produces one result, whose degree of truth is the product of $\min\{\alpha 1, \alpha 2, \ldots\ldots, \alpha i\text{ - }1\}$(the minimum value of the trigger conditions) and β(the degree of truth of Type 2 event). In Type 3, any of its conditions can trigger the service, so the degree of truth of the service response result is the product of $\max\{\alpha 1, \alpha 2, \ldots\ldots, \alpha i\text{ - }1\}$(the maximum value of the trigger conditions) and β(the degree of truth of Type 3 event). In the end, Type 4 is the full version of Type 2. Therefore, all the response results of Type 4 have the same degree of truth.

The petri net model of uncertainty reasoning

In this work, we use the static and the dynamic Petri net models to analyze IoT service conflicts. For ease of understanding, in this section, we will give the definitions of the static analysis mode and dynamic simulation mode on uncertainty reasoning.

Definition 13 (The static analysis Petri net model) A static analysis Petri net Model can be represented as a tuple $SPN = < P, T, R, M, TC, TR, TS > ..$

- P is a set of places, i.e., $P = \{P1, P2, \cdots, P_p\}$;
- T is a set of transitions, i.e., $T = \{T1, T2, \cdots, T_t\}$;
- R is a $p \times t$ matrix in which $r^{\triangleright}_{i,j}$ is the weight of arc from T_i to its output place P_j, and $r^{\triangleleft}_{i,j}$ is the weight of arc to T_i from its input place P_j;
- M is a sequence representing the markings, $M = \{M(P1), M(P2), \cdots, M(P_p)\}$, where $M(P_k)$ denotes the number of tokens in placeP_k, and M_0 is the initial marking;
- TC is the link between the places and their corresponding degree of truth, denoted as $TC(P_i) = \alpha_i$;
- TR is the link between the transitions and their corresponding degree of truth, denoted as $TR(T_i) = \beta_i$;
- TS is a set of thresholds of transition firing, denoted as $TS(T_i) = \lambda_i$.

Similar to the definitions of traditional Petri nets, $\bullet T_i$ and $T_i\bullet$ are the input places and the output places of T_i, respectively. If and only if the degree of truth of all T_i's input places is greater than or equal toT_i's threshold, T_i can fire. If we set the thresholds of all transitions to 1, a HLPN becomes a traditional Petri net.

Definition 14 (The dynamic simulation Petri net model) A dynamic simulation Petri net model can be represented as a tuple $DPN = < \{P_{PC}, P_{IS}\}, T, R, M, TC, TR, TS >$.

- P_{PC} andP_{IS} both are place sets, i.e., $P_{PC} = \{P1, P2, \cdots, P_p\}$, $P_{IS} = \{P^1{}_{IS}, P^2{}_{IS}, \cdots, P^l{}_{IS}\}$;
- T is a set of transitions, i.e., $T = \{T1, T2, \cdots, T_t\}$;
- R is a $p \times t$ matrix in which $r^{\triangleright}_{i,j}$ is the weight of arc from T_i to its output place P_j, and $r^{\triangleleft}_{i,j}$ is the weight of arc to T_i from its input place P_j;
- M is a sequence representing the markings, $M = \{M(P1), M(P2), \cdots, M(P_p)\}$, where $M(P_k)$ denotes the number of tokens in placeP_k, and M_0 is the initial marking;
- TC is the link between the places and their corresponding degree of truth, denoted as $TC(P_i) = \alpha_i$;
- TR is the link between the transitions and their corresponding degree of truth, denoted as $TR(T_i) = \beta_i$;
- TS is a set of thresholds of transition firing, denoted as $TS(T_i) = \lambda_i$.

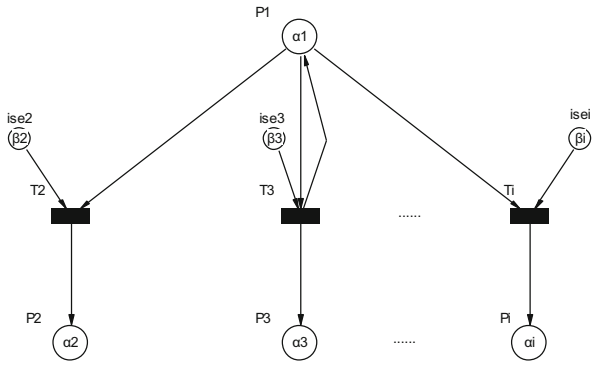


Fig. 22 The uncertainty transformation model of Type1

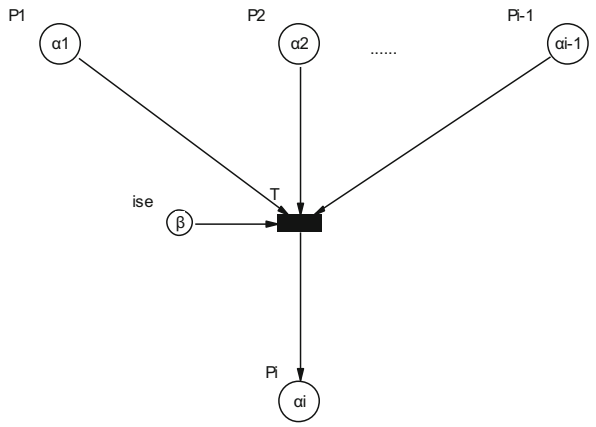


Fig. 23 The uncertainty transformation model of Type2

Transformation from service events to uncertainty petri net models

As we can see from Sect. "Fuzzy petri nets for fuzzy IoT conflict policies modeling and reasoning with certainty factors", we have defined four types of IoT service events with uncertainty factors. In this section, we follow the Sect. "Petri nets for IoT conflict policies modeling and verification" to transform the four types into the dynamic simulated Petri net model. Based on the logical equivalence, a fuzzy event in the form of P1(α1) $\rightarrow$ P2(α2) $\wedge$ P3(α3) $\wedge$ ..... $\wedge$ Pi(αi) is logically equivalent to the form of {P1(α1) $\rightarrow$ P2(α2),P1(α1) $\rightarrow$ P3(α3),$\cdots$,P1(α1) $\rightarrow$ Pi(αi)}; Similarly, the form of P1(α1) $\vee$ P2(α2) $\vee$ ..... $\vee$ Pi - 1(αi - 1) $\rightarrow$ Pi(αi) is logically equivalent to the form of {P1(α1) $\rightarrow$ Pi(αi),P2(α2) $\rightarrow$ Pi(αi), $\cdots$, Pi - 1(αi - 1) $\rightarrow$ Pi(αi)}. The corresponding transformation of the Type 1, 2, 3, and 4 fuzzy events are shown in Figs. 22, 23, 24, and 25, respectively. As seen in Figs. 22 and 24, the multiple subevents after decomposition correspond to the multiple transitions. Actually, this decomposition equivalence is consistent with a real-world IoT application. For example, a thermostat can control the operation of an air conditioner or a window. Dense smoke

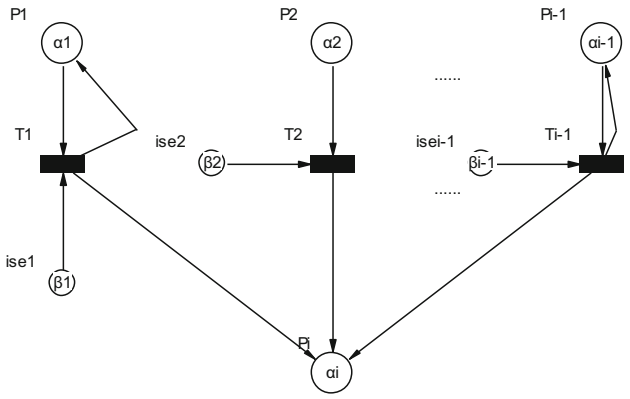


Fig. 24 The uncertainty transformation model of Type3

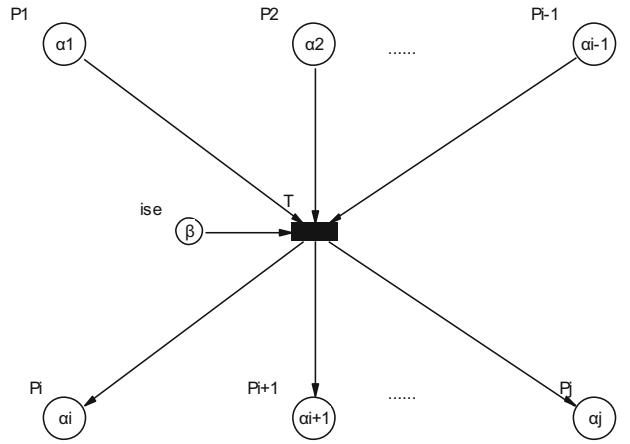


Fig. 25 The uncertainty transformation model of Type4

and someone walking around in the middle of the night both can trigger the security alarm.

Transformation from IoT conflict policies to petri nets with certainty factors

In this section, we discuss how to transform IoT conflict policies to Petri nets with certainty factors further. The corresponding transformation of Figs. 12, 13, 14, 15, 16 are presented in Figs. 26, 27, 28, 29, 30, respectively. After transformation, each place and each transition both have a certainty factor. In other words, all the triggering conditions and response results of an IoT service event have a certain probability to become true. For example, as shown in Fig. 26, when two different residents would like to use a same IoT service, the probabilities of their triggering conditions are α_1 and α_2, respectively. After the firing of transitions (T1 and T2) with a common probability β, they get response results with the probabilities α_3 and α_4, respectively. Due to space

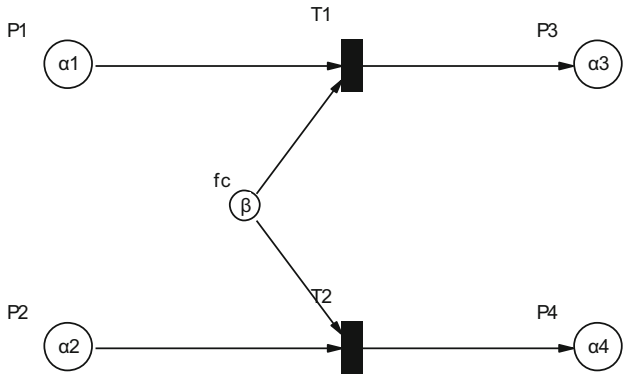


Fig. 26 The uncertainty transformation model of functional conflict

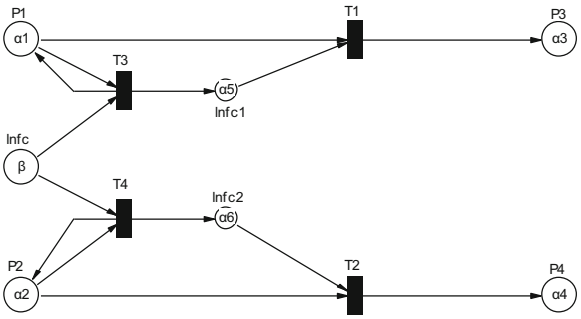


Fig. 27 The uncertainty transformation model of qualitative non-functional conflict

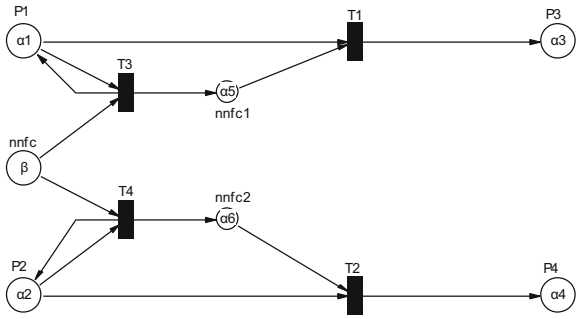


Fig. 28 The uncertainty transformation model of quantitative non-functional conflict

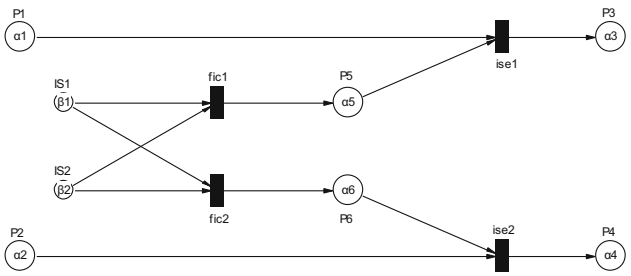


Fig. 29 The uncertainty transformation model of function impact conflict

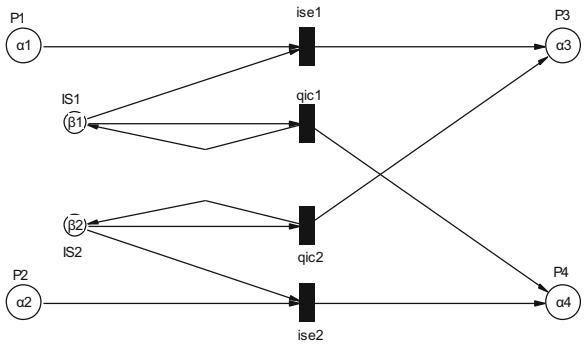


Fig. 30 The uncertainty transformation model of QoS impact conflict

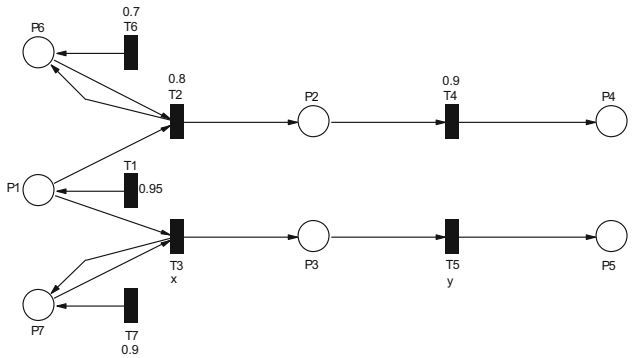


Fig. 31 The corresponding static analytic PN model of the formalism example with certainty factors

limitation, the detailed analysis methods of Figs. 27, 28, 29, 30 are similar to that of Fig. 26, which are omitted here.

Formalism of Petri nets with certainty factors

Based on the algorithm ISE-UR below, in this subsection, we will detail how to formalize a high-level Petri net with certainty factors. We still take the example from Sect. "Conflict detection" to introduce our approach, but we added certainty factors. Suppose the factors of the transitions T1 to T7 are $\beta = \{0.95, 0.8, x, 0.9, y, 0.7, 0.9\}$, respectively. The corresponding static and dynamic simulated PN models of the formalism example are presented in Figs. 31 and 32, respectively. Different from Sect. "Conflict detection", all the places of Figs. 31 and 32 are given no initial tokens. In other words, T2 and T3 have equal opportunities to fire. That is to say, Tom may watch TV with the light on, at the same time, Alice can also have the opportunity to go to bed with the light off successfully. Given the certainty factors set β, we calculate the values of T3 and T5, i.e., x and y, which indicate the probability of T3 and T5 firing.

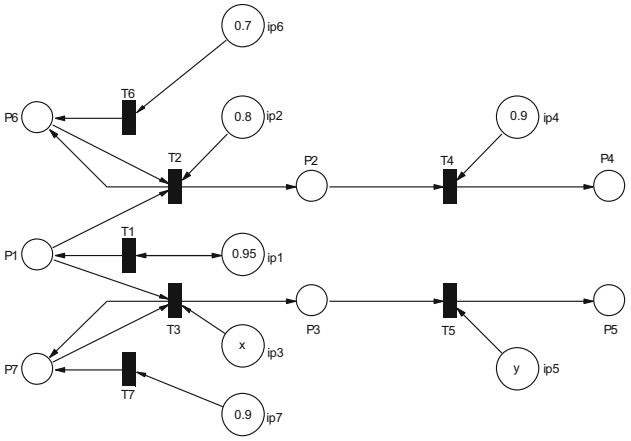


Fig. 32 The corresponding dynamic simulated PN model of the formalism example with certainty factors

$$R^{\triangleright} = \begin{array}{c} \\ \mathrm{T1} \\ \mathrm{T2} \\ \mathrm{T3} \\ \mathrm{T4} \\ \mathrm{T5} \\ \mathrm{T6} \\ \mathrm{T7} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathrm{P1} & \mathrm{P2} & \mathrm{P3} & \mathrm{P4} & \mathrm{P5} & \mathrm{P6} & \mathrm{P7} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{4}$$

$R^{\triangleright}$ and $R^{\triangleleft}$ are calculated in Eqs. 4 and 5, respectively. Then we get $R = R^{\triangleright} - R^{\triangleleft}$ in Eq. 6. We apply the algorithm ISE-UR for computing the degree of truth of T3 and T5. The fuzzy incidence matrix before the uncertainty reasoning, i.e., $R_{IoT}^{f\_b}$, is shown in Eq. 7. Finally, the result of the fuzzy uncertainty reasoning is detailed in Eq. 8. As we can see from the Eq. 8, x and y are both 0.95, which means that Alice has a probability 0.95 of going to bed with the light off successfully. Actually, as shown in Fig. 32, the input place ip6 of the transition T6 has a degree of truth 0.7. However, the degree of truth of T7's input place ip7 is greater than 0.7, and its value is 0.9. So, this determines that x and y both have a high probability to fire.

$$R^{\triangleleft} = \begin{array}{c} \\ \mathrm{T1} \\ \mathrm{T2} \\ \mathrm{T3} \\ \mathrm{T4} \\ \mathrm{T5} \\ \mathrm{T6} \\ \mathrm{T7} \end{array} \begin{array}{c} \begin{array}{ccccccc} \mathrm{P1} & \mathrm{P2} & \mathrm{P3} & \mathrm{P4} & \mathrm{P5} & \mathrm{P6} & \mathrm{P7} \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \tag{5}$$

$$R = R^{\triangleright} - R^{\triangleleft} = \begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \\ T6 \\ T7 \end{array} \begin{array}{c} \begin{array}{ccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \tag{6}$$

$$R_{IoT}^{f\_b} = \beta R^{\triangleright} - R^{\triangleleft} = \begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \\ T6 \\ T7 \end{array} \begin{array}{c} \begin{array}{ccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 \end{array} \\ \begin{pmatrix} 0.95 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0.8 & 0 & 0 & 0 & -0.2 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.9 \end{pmatrix} \end{array} \tag{7}$$

$$R_{IoT}^{f\_a} = \begin{array}{c} \\ T1 \\ T2 \\ T3 \\ T4 \\ T5 \\ T6 \\ T7 \end{array} \begin{array}{c} \begin{array}{ccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 \end{array} \\ \begin{pmatrix} 0.95 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.95 & 0.76 & 0 & 0 & 0 & -0.2 & 0 \\ -0.95 & 0 & 0.95 & 0 & 0 & 0 & 0 \\ 0 & -0.76 & 0 & 0.684 & 0 & 0 & 0 \\ 0 & 0 & -0.95 & 0 & 0.95 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.9 \end{pmatrix} \end{array} \tag{8}$$

Uncertainty reasoning algorithm

We present an algorithm, ISE-UR, to calculate the resulting degree of truth of solutions to queries. Given an IoT service event *ise*: P1 $\rightarrow$ P2, P1 is a trigger condition, P2 is a response result, and ' $\rightarrow$ ' denotes the action of *ise*. Suppose α_i indicates the degree of truth of a place P_i, and β denotes the degree of truth of the IoT service event. Based on the previously mentioned policies of IoT service conflicts transformation to Petri nets, we can transform a IoT application scenario into a HLPN with ñ transitions and ň places. Then, since we take advantage of the characteristics of HLPN, for a transition T_i, we can compute the degree of truth from its incoming arcs to its outgoing arcs ($\alpha(T_i\bullet) = \min\{\alpha(\bullet T_i)\} \times \beta_i$, $\bullet T_i$ and $T_i\bullet$ denote the input places and output places of T_i, respectively). Given a place P_j, we can also compute the degree of truth from its incoming arcs to its outgoing arcs ($\alpha(P_j\bullet) = \max\{\alpha(\bullet P_j)\} \times \beta_i$, $\bullet P_j$ and $P_j\bullet$ denote the input transitions and output transitions of P_j, respectively). Suppose $\beta = \{\beta_i\}$, $0 \le i \le$ ñ, a fuzzy incidence matrix can be solved by $R_{IoT}^{f} = \beta R^{\triangleright} - R^{\triangleleft}$.

Algorithm: ISE-UR

Input: $R^{\triangleright}$, $R^{\triangleleft}$, β, X

Output: the incidence matrix R_{IoT}^{f} after uncertainty reasoning

```
1.  for (i=1; i ≤ ñ; i++)
2.   { if (X [i]≠0)
3.       {
4.       for (j=1; j ≤ ň; j++)
5.            { if (r_r[i, j] > 0)
6.                  r_r[i, j]= r_r[i, j]×β[i];
7.                  r_f[i, j]= r_r[i, j]− r_l[i, j]; }
8.        }
9.   } // Compute the incidence matrix R_IoT^f before uncertainty reasoning

10. for (i=1; i ≤ ñ; i++)// Initialize the minimum value in rows
11.    row_min_abs[i]=1;
12. for (j=1; j ≤ ň; j++)// Initialize the maximal value in columns
13.    col_max[j]=0;

14. int Tn[];//Record the location of each row's positive value according to the
        given PN model
15. for(i=1; i ≤ ñ; i++)
16.    for(j=1; j ≤ ň; j++)
17.        if (r_f[i, j]> 0)
18.            {Tn.push_back(i);
19.            break;}

20. std::vector<std::vector<double>> tmp;   // a temporary variable
21. for(i:=0 to Tn.size()   //Assign values to variable tmp
22.    for(j := 0 to ñ)
23.        tmp[Tn[i] - 1].push_back((r_f[ñ *i+j]);
24. r_f.clear();   //clear the original values of r_f
25. for(i:=0 to tmp.size())
26.        r_f.push_back(tmp[]);   //Assign tmp's elements to r_f

27. for (i=1; i ≤ ñ; i++)// uncertainty reasoning
28. {
29. j=1;
30. while(r_f[i, j] ≤0 and j≤ň)
31.    {
32.        if(r_f[i, j] < 0)
33.            {
34.                r_f[i, j]= r_f[i, j]×col_max[j];
35.                if(abs(r_f[i, j])< row_min_abs[i])
36.                    row_min_abs[i]=abs(r_f[i, j]);
37.            }
38.        j++;
39.    }
40. if(r_f[i, j] > 0)
41.    {
42.        r_f[i, j]= r_f[i, j]  ×row_min_abs[i];
43.        if( r_f[i, j]> col_max[j])
44.                col_max[j]= r_f[i, j];
45.    }
46. }
```

r_r[i, j] denotes the element of $R^{\triangleright}$, which is the weight of arc from a transition T_i to its output place P_j. And r_l[i, j] is the element of $R^{\triangleleft}$, which indicates the weight of arc to a transition T_i from its input place P_j. We use a 2-dimensional array r_f[] to produce the values of the incidence matrix R^f_{IoT}, whose values before uncertainty reasoning are initialized in lines 1–9. The minimum value in rows and the maximal value in columns are initialized in lines 10–11 and lines 12–13, respectively. Actually, before uncertainty reasoning, there is only one positive value in a row of R^f_{IoT} (i.e., $R^{f\_b}_{IoT}$) with respect to the corresponding static analytic and dynamic simulated Petri net models. As we can see from Eq. 7, if all the subscripts ($T_i \rightarrow P_i$) satisfy a strict partial ordering, all the positive values of R^f_{IoT} are on the diagonal. Therefore, if we disrupt this relation and input the values by means of a random order of rows (for example, we swap two rows of values of $R^{f\_b}_{IoT}$ randomly), we must restore the relation by means of lines 14–26.

Finally, in lines 27–46, we present the concrete uncertainty reasoning process. If the value of r_f[i, j] is negative, we multiply it with the maximum value in the jth column of r_f[] (see line 34), and set the value of r_f[i, j] as the minimum value in the ith row of r_f[] if it is less than the current minimum value (lines 35–36). However, if the value of r_f[i, j] is positive, we multiply it with the minimum value in the ith row of r_f[] (see line 42), and set the value of r_f[i, j] as the maximum value in the jth column of r_f[] if it is greater than the current maximum value (lines 43–44).

Evaluation

In this section, we use experiments to evaluate our approach.

Experiment setup

Our dataset is a real dataset collected from the CASAS system [37]. The hardware environment is a machine with the Intel(R) Core(TM) i5 CPU 760, 2.80 GHz, and 4 GB RAM running Windows 7 (64-bit). Our programming language is java. As we can see from CASAS's website, we can download single-resident and multi-resident activity datasets. However, since there are no conflicting situations of IoT service usage among those multi-resident activity datasets, they hardly contribute to the evaluation of our approach. Actually, those multi-resident activity datasets are the results of compromises of residents interacting with smart-home devices, and that cause no conflicts. However, the records of single-residents are their actual interactions with IoT services, which can reflect their preferences for what conflicts might occur. So, in this work, we choose the datasets of twelve individual residents, which contain two months of usage records.

Table 3 Description of the attributes of the dataset

Attributes	Description
Date	The service execution date
Time	The service execution time
Sensor	Name of the sensors
Status	Service status, i.e., ON or OFF

As we can see from the "README.txt" file of each single-resident's dataset, there are several different types of sensors, which are considered as IoT services in our approach. The attributes of our dataset are detailed in Table 3. For example, the following are two records extracted from the dataset, i.e., "2012-07-18 12:54:45.126257 D001 CLOSE" and "2012-07-18 12:54:45.767498M009 ON". Identifiers starting with "D" indicate magnetic door sensors, and identifiers starting with "M" indicate infrared motion sensors.

As mentioned before, each sensor is regarded as an IoT service, so they are used interchangeably for the rest of the paper. Moreover, each record of the dataset only contains the on/off time of the corresponding sensor, which are denoted as two statuses of the service, i.e., "ON" and "OFF". Obviously, based on the raw data, if there exist overlapping events (see Definition 5), only functional conflicts can be detected. Since the dataset has no QoS attributes, in our work, we simulate some QoS properties to test our approach.

In this work, we mainly focus on four types of services when we simulate QoS properties, i.e., stereo service, light service, AC (air-conditioning) service, and TV (television) service. Firstly, we consider volume and channel two properties for TV service, and the ID of channel varies randomly from 1 to 40. However, for a senior resident, the volume of a TV is randomly assigned around 35 dB, otherwise a random value is generated between 45 and 75 dB. Then, we consider three run modes for the AC service, i.e., ventilation mode (a), heating mode (b) and cooling mode (c). Moreover, the velocity of each mode has four patterns and each pattern is assigned an integer value, i.e., strong (-1), medium (0), low (1) and auto (2). In a concrete scenario with respect to an AC service event, we randomly generate a run mode from the set of {a, b, c}, then randomly generate a velocity pattern from the set of $\{-1, 0, 1, 2\}$. If the run mode is heating mode, we randomly generate a temperature between the range of 17–31 °C. However, if the system gets a cooling mode, the value of temperature is only generated from the range of 16–27 °C. In addition, a light service may own multiple QoS attributes, such as luminosity level, durability and color of light. The range of luminosity level is 15 lx to 500 lx, and color of light is an element of the set of {white

(0), red (1), yellow (2), blue (3)}. Suppose a resident is reading, the system generates a random value of luminosity level around 300 lx, and the color of light is set to white. Beyond that, the value of luminosity level will be randomly created around 150 lx, and the color is randomly assigned a value from the set of {0, 1, 2, 3}. Finally, we consider two QoS properties for a stereo service, i.e., volume and subwoofer. If the resident is an elder people, we set the subwoofer of the stereo service below 30%, and randomly generate a value of volume below 45 dB (since most older people prefer quietness). Besides that, there will be no constraints on volume and subwoofer of a stereo service when we simulate the two properties for a resident.

In several sets of experiments, to better evaluate our method's performance, we will take on a double focus, i.e., non-conflict scenarios and conflict scenarios. And their definitions are presented as follows:

Definition 15 (conflict situation) In a smart home, given two users U_1 and U_2, a conflict situation is that they request a same IoT service or two different IoT services, but they cannot be satisfied at the same time.

Definition 16 (no-conflict situation) In a smart home, given two users U_1 and U_2, when they request a same IoT service or two different IoT services, a no-conflict situation is that they both can be satisfied at the same time.

In this work, we use the ground truth of the thermal preference which is detailed in [40] discussed in detail the behavior of PMV (predicted mean vote) [43] when used for predicting subjective thermal comfort votes of people in buildings. By the way, PMV has become an international standard, i.e., International Standard ISO 7730 [44]. Based on the work of [40], if greater than 3 °C, a person can feel the difference. In other words, in a same room, if the difference between their thermal preference is no more than 3 °C, they both are comfortable. So, the ground truth with respect to the difference of thermal preference is set to 3 °C in this work.

Experimental results

In the first set of experiments, we evaluate and compare the accuracy of our approach (named ISE, without thinking about certainty factors) with an ontology-based approach OBA, which is augmented and based on the ideas of [13] and [39]. Our augmented dataset contains 2360 rows of service events (involving primarily the sensors of TV, AC, light, and window). Among them, there are 837 events, which recorded the thermal preference of those residents in 2 months. By means of the statistical analysis, there are 518 overlapping service events of AC. In this work, if two residents cannot be satisfied at the same time, we think that a conflict will occur. Similarly, people have such different preferences for

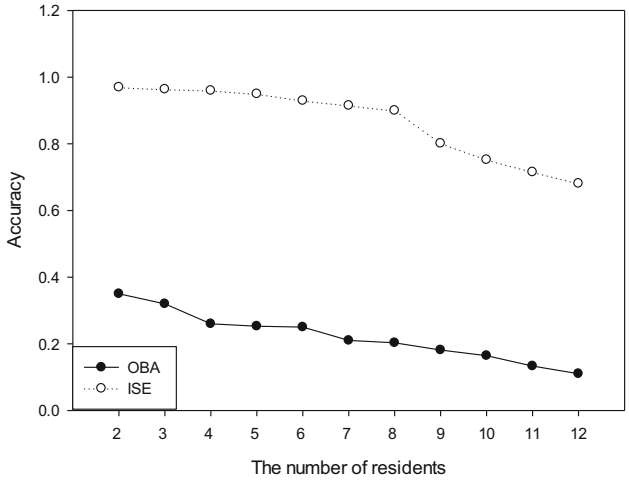


Fig. 33 Accuracy evaluation of non-conflict scenarios

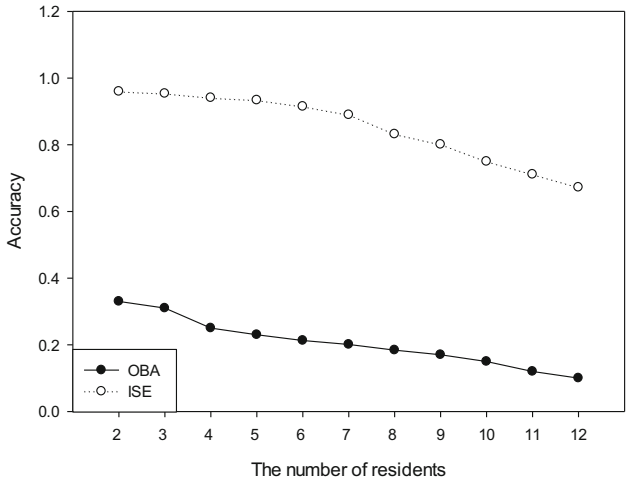


Fig. 34 Accuracy evaluation of conflict scenarios

luminance. In our experiments, we are of the opinion that there is a conflict, where the difference between individual residents' luminance preference is more than 5. Given a fixed number of residents with functional and non-functional preferences, we try to find conflicts among them from the service events records. Concretely, this set of experiments take on a double focus, i.e., non-conflict scenarios and conflict scenarios. The corresponding experimental results are shown in Figs. 33, 34, respectively. All experiments are conducted independently for 100 times and the average results are calculated. As we can see from Figs. 33, 34, when the number of residents varies from 2 to 12, the accuracy of ISE and OBA both have a slight decline under the two scenarios. This is because more people will produce more IoT service events. At the same time, conflict probability is greatly increased, and the match among the conflicting parties is more complicated. However, ISE invariably yields a much better accuracy

than OBA. On the other hand, ISE has a slightly better performance in the non-conflict scenarios, where the calculation is simple, moreover, we can easily handle it.

As mentioned earlier, based on the ground truth models [40], suppose two residents are in a same smart home, if the difference between their thermal preference for an AC service is more than 3 °C, one resident must be uncomfortable and he would like to adjust the air conditioner temperature. As a result, a conflict will occur. On the other hand, there is no conflict if the temperature preference difference is no more than 3 °C. As long as the temperature preference difference is more than 1 °C regardless of no more than 3 °C, the ontology-based approach OBA decides that there is a conflict. So, under different number of residents, a large number of instances are classified as conflicts by OBA, which are actually non-conflicts according to the ground truth models. For example, when the number of residents is 12, there are actually 1948 conflict instances and 2826 non-conflict instances. ISE detects 1932 instances as conflicts and 2706 instances as non-conflicts. However, OBA detects 2816 instances as conflicts, which are actually non-conflicts. Similar to AC service, if the difference between two residents' luminance preference for light service is no more than 5, we declare a non-conflict. For lack of space, we won't discuss any more details with respect to light service. Although ISE has a high accuracy, it is a little time-consuming. Since it needs more time to do policies transformation and verification.

In the second set of experiments, we focus on how the certainty factors affect our approach's performance. The method is named ISE-UR (IoT Service Events Uncertainty Reasoning), which considered certainty factors. Our dataset consists of two sets: (1) 80% is used for partial certainty factors calculation, and 20% is used for performance verification. Our method consists of three main steps, i.e., certainty factors calculation of a portion of service events (training data), uncertainty factors reasoning of test data, and conflicts prediction and validation. In the first step, by means of Eq. 9, we can solve the problem of partial certainty factors calculation. The certainty factor of a user U on an IoT service IS can be computed as follows:

$$\beta i = {}^{Num_U^{IS}}/_{Num_U^{ISs}} \tag{9}$$

where Num_U^{IS} is the number of times user U enjoyed the IoT service IS, but Num_U^{ISs} is the total number of times relating to all those services used by U. Then, based on the algorithm ISE-UR, we do uncertainty factors reasoning for those residents, who relate to the people involved in the first step. Finally, we take advantage of the certainty factors gotten from step 2 to do conflicts prediction and validation. ISE-UR can detect both conflict situations and non-conflict situations. Figures 35, 36, 37, 38 are the experimental results, which evaluate the performance of our approach from accuracy,

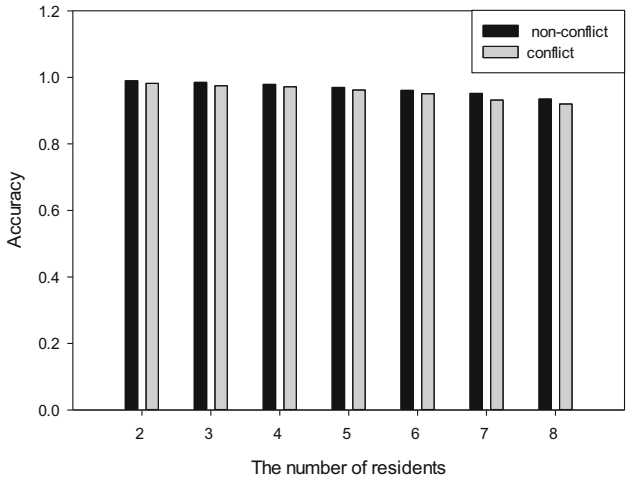


Fig. 35 Accuracy evaluation of conflict and non-conflict scenarios with certainty factors

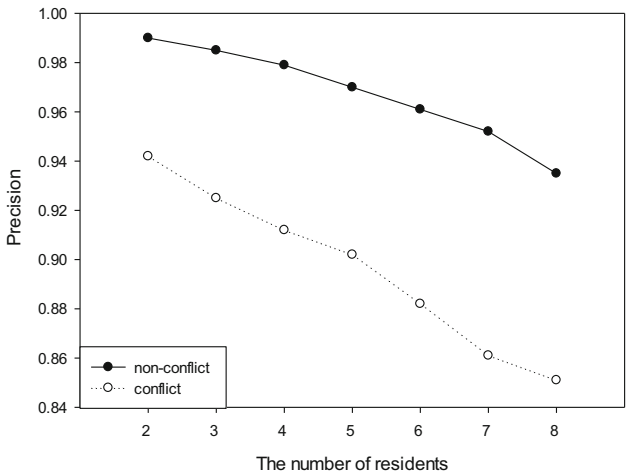


Fig. 36 Precision evaluation of conflict and non-conflict scenarios with certainty factors

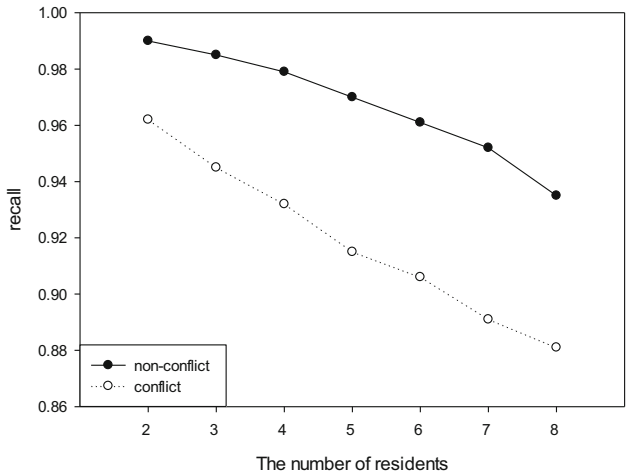


Fig. 37 Recall evaluation of conflict and non-conflict scenarios with certainty factors

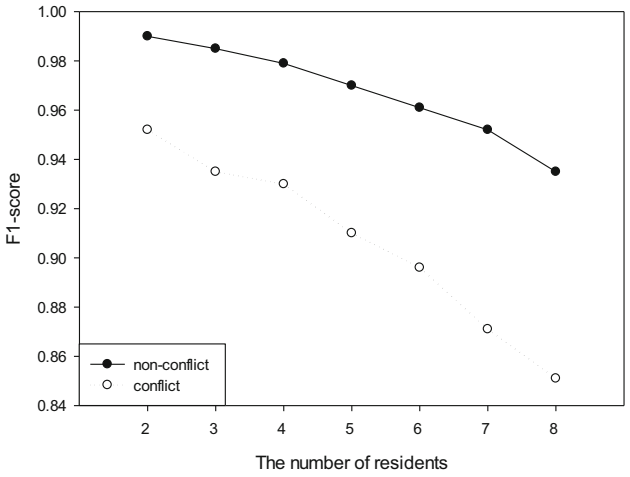


Fig. 38 F1-score evaluation of conflict and non-conflict scenarios with certainty factors

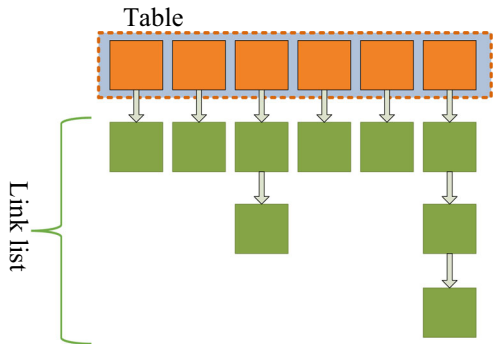


Fig. 39 Data structure of IoT services in DepSys

precision, recall and F1-score, respectively. For simplicity, in this set of experiments, we only consider non-functional quantitative conflict (e.g., temperature preference or luminance level). As we can see from Figs. 35, 36, 37, 38, since uncertainty reasoning is incorporated into our method, ISE-UR can efficiently detect both conflicts and non-conflicts more accurately based on precision, recall, and F1-score. As the number of residents increases, ISE-UR has a decline on precision, recall, and F1-score.

In the third set of experiments, we try to compare the accuracy of our approach with UTEA [41], IRIS [42], and the brute-force approach of [9] (BFA for short). Specifically, we compare five types of conflicts, i.e., functional conflicts, qualitative non-functional conflicts, quantitative non-functional conflicts, function impact conflicts and QoS impact conflicts (detailed in Sect. "Formal definitions of conflicts"). For comparison, we use the simulation dataset explained earlier, which contains some QoS attributes of IoT services. The number of residents is fixed at 5, 6, 7 and 8, the comparison results are demonstrated in Tables 4, 5, 6, 7, respectively.

As we can see from Tables 4, 5, 6, 7, the four approaches all can capture functional and qualitative non-functional conflicts. IRIS cannot detect quantitative non-functional, function impact and QoS impact conflicts. However, it has a high accuracy on functional conflicts and qualitative non-functional conflicts. Actually, IRIS mainly detects device conflicts, which has opposite functionality [9]. So, it can accurately detect functional and qualitative conflicts. As discussed in [9], though UTEA can capture quantitative non-functional and direct impact (function impact) conflicts, it has a high miss-detection rate for capturing quantitative conflicts. Moreover, UTEA cannot detect QoS impact conflicts. From the experimental results, we get that ISE-UR can detect quantitative non-functional and QoS impact conflicts more accurately. Specifically, on quantitative non-functional conflicts, the ground truth values of Tables 4, 5, 6, 7 are 73, 112, 176 and 306, respectively. Meanwhile, when we consider QoS impact conflicts, the ground truth values of Tables 4, 5, 6, 7 are 121, 196, 273 and 426. Actually, our fuzzy uncertainty reasoning can fully use historical data to model preferences. That, plus the excellent features of Petri nets [31], ISE-UR not only enhances reliability, but also has high precision. There are, of course, more conflicts with the increment in the number of residents (the five types of conflicts all have the same results).

In addition, to further evaluate the performance of ISE, we compare its effectiveness and accuracy with IoTGuard [45], DepSys [46], and CityResolver [48].

To evaluate the effectiveness of ISE, we firstly start with a comparison of IoTGuard and ISE. IoTGuard is a dynamic enforcement system for the usage of the most sensitive resource in an IoT environment, which can detect violated policies by means of an assertion of the code blocks. We developed 30 hand-crafted SmartThings apps which contain 8 representative policy violations, and 15 is the same as the apps detailed in [45]. The apps cover diverse real-life use cases, such as light, door, television, air conditioner, thermostat and so on. We designed 10 subsets of the apps (i.e., from Group 1 to Group 10), and the scale varies from 3 to 12. We implement SmartThings on a local server. We use the IoT policies detailed in APPENDIX of [45], and we added 10 new policies with respect to application-specific policies. We also use IoTGuard Policy Language (GPL) syntax in BNF, which was detailed in [45]. In addition, we use the SmartThings simulator [45] to execute the apps, which can simulate the behavior of physical devices with virtual devices. As we can see from the results in Table 8, IoTGuard and ISE both can find all violated policies.

Then, we conduct a set of experiments considering ISE and DepSys simultaneously. DepSys analyzes dependencies among various systems, which are all regarded as apps [46]. In order to implement DepSys in [46], we use the data structure of Fig. 39 to store IoT services in a smart home. The

Table 4 Accuracy comparison with 5-residents

Conflict type	UTEA	IRIS	BFA	ISE-UR
Functional	823	823	823	823
Qualitative non-functional	284	284	284	284
Quantitative non-functional	95	0	62	69
Function impact	115	0	115	115
QoS impact	0	0	98	108

Table 5 Accuracy comparison with 6-residents

Conflict type	UTEA	IRIS	BFA	ISE-UR
Functional	1108	1108	1108	1108
Qualitative non-functional	325	325	325	325
Quantitative non-functional	143	0	88	102
Function impact	214	0	214	214
QoS impact	0	0	147	177

Table 6 Accuracy comparison with 7-residents

Conflict type	UTEA	IRIS	BFA	ISE-UR
Functional	1316	1316	1316	1316
Qualitative non-functional	527	527	527	527
Quantitative non-functional	227	0	145	153
Function impact	358	0	358	358
QoS impact	0	0	195	235

top row in the table stores all the rooms, and their children (i.e., those IoT services in each room) is stored in underlying link lists respectively. In our programming setting, we use a Map interface to implement this data structure, and the Map is fulfilled by HashMap class. Actually, HashMap is a combination of table and link list, whose structure is similar to Fig. 39. In addition, the link list is a good option for storing dependency relationships of DepSys, such as actuator control dependency and name dependency. We just need to add some dependent properties to elements of link lists.

The dataset of DepSys is also the real dataset collected from the CASAS system [37]. In this set of experiments, we use 4 days of the datasets of eight individual residents, which are introduced before. Based on the datasets, we can provide enough information by means of sensor data and ground truth

Table 8 Effectiveness evaluation

Groups of apps (No. of apps)	IoTGuard	ISE
Group 1: (3)	√	√
Group 2: (4)	√	√
Group 3: (5)	√	√
Group 4: (6)	√	√
Group 5: (7)	√	√
Group 6: (8)	√	√
Group 7: (9)	√	√
Group 8: (10)	√	√
Group 9: (11)	√	√
Group 10: (12)	√	√

Table 7 Accuracy comparison with 8-residents

Conflict type	UTEA	IRIS	BFA	ISE-UR
Functional	1509	1509	1509	1509
Qualitative non-functional	802	802	802	802
Quantitative non-functional	378	0	256	269
Function impact	519	0	519	519
QoS impact	0	0	293	374

Table 9 Accuracy comparison between DepSys and ISE

No. of residents	DepSys	ISE
2	73	80
3	112	124
4	138	155
5	179	191
6	234	244
7	271	281
8	323	331

of activities of the residents to simulate the behavior of the apps that DepSys needed. Specifically, we use 11 apps (App# 1, 2, 3, 8, 14, 15, 16, 19, 22, 30 and 31), which are defined in Table 3 of [46]. And, we design these apps and define the relevant metadata. The selected apps cover the sensors of TV, stereo, light, and window. Meanwhile, when the number of residents varies from 2 to 8, the ground truth values (i.e., the exact number of conflicts) correspond to 80, 124, 155, 196, 248, 286, and 337, respectively. The experimental results are listed in Table 9.

CityResolver is a decision support system for conflicts resolution in smart cities [48], where an Integer Linear Programming (ILP) method are used to select a small set of candidate options. We also implement a smart city simulator using the Simulation of Urban MObility (SUMO) [48] in MATLAB. SUMO is an open source traffic simulation package, which includes net import and demand modeling components. We simulate traffic volume information on main roads and the number of vehicles passing on each road at specified intervals. In addition, we create an excel file for each intersection. All the excel files are imported into MATLAB, where we do simulation. We have launched a simple comparison between CityResolver and ISE in a simulated urban environment, and we still use the ten services detailed in Table I of [48]. By making different combinations of the ten services, we can get a large number of scenarios. But in the paper, we focus on five subsets of the whole services (from Case 1 to Case 5). We randomly simulate city requirements, and count the number of violated requirements (that we deem a conflict). Specifically, the ground truth values (conflicts) of the Cases 1–5 are 22, 30, 35, 50 and 65, respectively. The results are shown in Table 10. When the scale of the subset of services is small, CityResolver and ISE both can detect all the violated requirements. As the number of services of the Case increases, CityResolver and ISE-UR both have a decline on precision. In other words, if the scenario is more complex, the two methods both cannot find all the conflicts.

Finally, we focus on the efficiency of our approach (ISE). Suppose there are two smart home scenarios (i.e., 3-residents and 6-residents) and each smart home owns a certain number

Table 10 Accuracy comparison between CityResolver and ISE

Case	CityResolver	ISE
Case 1: [S1, S2]	22	22
Case 2: [S1, S2, S3]	30	30
Case 3: [S1, S8, S10]	35	35
Case 4: [S6, S7, S9, S10]	45	50
Case 5: [S2, S3, S4, S5, S9]	52	65

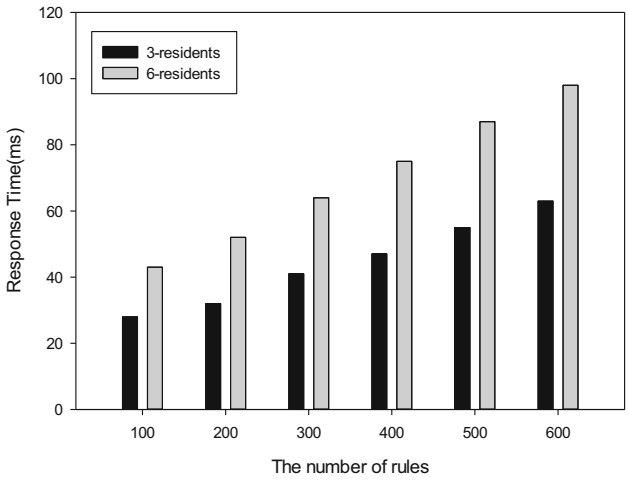


Fig. 40 Efficiency evaluation with 3-residents and 6-residents

of rules or policies, we calculate the time that ISE will take under each experimental scene. Specifically, the number of rules varies from 100 to 600. As we can see from Fig. 40, with the increasing number of rules, both response time of the two scenarios grows. When experimental condition becomes more and more complicated (i.e., more rules or policies), both of them need more query time. However, under each problem scale, ISE is obviously faster when there are fewer people in the smart home. Actually, if there are more residents in a smart home, the probability of conflict will be greater. That is to say, we'll take more time to handle those conflicts.

Discussion

In this paper, we present a synthetic system to detect conflicts in a smart home. Simply put, we have two ways to do conflicts detection, i.e., solving sets of linear equations and template matching. By comparing with the state of the art in conflicts detection of IoT services, we get that the system is effective and efficient. However, our system has three major drawbacks. Firstly, solving linear equations is really complicated (this way is more suitable for the scenarios with few people), especially when the number of variables is large. Secondly, given two IoT service events, how to properly design Petri

nets for them is very challenging. That needs not only the theoretical knowledge with respect to Petri net, but also needs relevant domain knowledge. Finally, we're to be focusing on integrating our approach into a real IoT system, and that is also very difficult. Since we must consider the design and simulation of Petri net models, and we should also consider solving sets of linear equations with quite a few variables.

Conclusions and future work

In a multi-resident smart home environment, various appliances (IoT services) are connected to a home network and provide residents with value-added services, such as appliance integration. Since people have different preferences, that may produce five broad categories of conflicts. In this paper, we propose a Petri nets formalism to detect conflicts among IoT services in multi-resident smart homes. We first model the formal methods for conflict policies. Then, we present a Petri nets-based mechanism for modeling and detecting the IoT service conflicts in a smart home. Finally, we expand the previous model and use high-level Petri nets (HLPN) for handling fuzzy IoT conflict policies modeling and reasoning. The experimental results show the effectiveness and efficiency of the proposed approach in accuracy. According to the experimental results, we show that our approach can detect the five types of conflicts (i.e., functional conflicts, qualitative non-functional conflicts, quantitative non-functional conflicts, function impact conflicts, and QoS impact conflicts) more accurately than UTEA, IRIS and BFA. In the next step, we plan to implement an IoT service recommender system for multi-resident smart homes.

Acknowledgements This work was supported by the Research on philosophy and Social Science in Hubei Province under Grants 21Y212 and 22Y156, the natural science foundation of Hubei province, grant number No. 2021CFB388, the outstanding young and middle-aged science and technology innovation team of universities in Hubei Province, grant number No. T2022032, the innovation team of Hubei University of Science and Technology, grant number No. 2022T04, the science and technology planning project of Xianning city, grant number No. 2022GXYF056.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Yavuz AA, Ozmen MO (2022) Ultra lightweight multiple-time digital signature for the internet of things devices[J]. IEEE Trans Serv Comput 1:15
2. Zhang J, Yang Y, Liu X et al (2022) An efficient blockchain-based hierarchical data sharing for healthcare internet of things[J]. IEEE Trans Industr Inf 18(10):7139–7150
3. Xu C (2022) Context aware mobility in internet of things enabling technologies, applications, and challenges[J]. Trans Emerging Telecommun Technol 8:33
4. Wu C, Liu S, Zeng Z et al (2022) Knowledge graph-based multi-context-aware recommendation algorithm[J]. Inf Sci 595:179–194. https://doi.org/10.1016/j.ins.2022.02.054
5. Baranwal G, Singh M, Vidyarthi DP (2020) A framework for IoT service selection[J]. J Supercomput 76:2777–2814
6. Pourghebleh B, Hayyolalam V, Anvigh AA (2020) Service discovery in the Internet of Things: review of current trends and research challenges[J]. Wireless Netw. https://doi.org/10.1007/s11276-020-02405-0
7. Dass P, Misra S, Roy C (2020) T-Safe: trustworthy service provisioning for IoT-based intelligent transport systems[J]. IEEE Trans Veh Technol 69(9):9509–9517
8. Kim UH, Kim JH (2019) A stabilized feedback episodic memory (SF-EM) and home service provision framework for robot and IoT collaboration[J]. IEEE Trans Cybern 99:1–14
9. Chaki D, Bouguettaya A, Mistry S (2020) A conflict detection framework for IoT services in multi-resident smart homes[J]. arXiv e-prints
10. Farooq AA, Al-Shaer E, Moyer T, et al. (2018) IoTC^2: a formal method approach for detecting conflicts in large scale IoT systems[J]
11. Igaki H, Nakamura M (2010) Modeling and detecting feature interactions among integrated services of home network systems[J]. IEICE Trans Inf Syst 93(4):822–833
12. Yagita M, Ishikawa F, Honiden S (2015) An application conflict detection and resolution system for smart homes[C]// 2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS). ACM
13. Rui C, Carreira P, Lyn Ce I et al (2014) An ontology-based approach to conflict resolution in Home and Building Automation Systems[J]. Expert Syst Appl 41(14):6161–6173
14. Perumal T, Sulaiman MN, Datta SK, et al. (2016) Rule-based conflict resolution framework for Internet of Things device management in smart home environment. In Proceedings of the 2016 IEEE 5th Global Conference on Consumer Electronics, Kyoto, Japan, 11–14 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–2
15. Shin C, Woo W (2009) Service conflict management framework for multi-user inhabited smart home[J]. J Univ Comput Sci 15(12):2330–2352
16. Wang Z, Zhu Z (2022) Construction of data transmission consistency verification model of Petri net based on fuzzy C-means[J]. Wireless Netw 5:28
17. Wu CH, Lee SJ (2000) A token-flow paradigm for verification of rule-based expert systems. IEEE Trans Knowl Data Eng 30(4):616–624
18. Hameed K, Garg S, Amin MB et al (2022) Towards a formal modelling, analysis and verification of a clone node attack detection

scheme in the internet of things[J]. Comput Netw. https://doi.org/10.1016/j.comnet.2021.108702
19. Li H, Yang D, Cao H et al (2022) Data-driven hybrid petri-net based energy consumption behaviour modelling for digital twin of energy-efficient manufacturing system[J]. Energy. https://doi.org/10.1016/j.energy.2021.122178
20. Wu CH, Lee SJ (1997) Enhanced high-level petri nets with multiple colors for knowledge verification/validation of rule- based expert systems. IEEE Trans Syst Man Cybern 27(5):760–773
21. Wang WGB (2021) Mental workload evaluation and its application in train driving multitasking scheduling: a Timed Petri Net-based model[J]. Cogn Technol Work. https://doi.org/10.1007/s10111-019-00608-w
22. Xu Q, Lin JT (2023) Safety requirement verification of train-centric CBTC by integrating STPA with coloured Petri net[J]. International journal of industrial and systems engineering: IJISE
23. Bourcier J, Chazalet A, Desertot M, et al. (2006) A dynamic-soa home control gateway. Proc. Int'l Conf. on Service Computing (SCC'06), pp.18–22, Sept
24. Shah T, Venkatesan S, Ngo T, Neelamegam K (2019) Conflict detection in rule based IoT systems. In Proceedings of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 17–19 October 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 0276–0284
25. Huang B, Dong H, Bouguettaya A (2021) Conflict Detection in IoT-based Smart Homes. arXiv 2021. arXiv:2107.13179
26. Ahmed MO, Elfaki SEE (2021) Adaptation conflicts of heterogeneous devices in IOT smart-home. Am Acad Sci Res J Eng Technol Sci 81:64–78
27. Chi H, Zeng Q, Du X, Yu J (2020) Cross-app interference threats in smart homes: Categorization, detection and handling. In Proceedings of the 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Valencia, Spain, 29 June-2 July 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 411–423
28. Fisher RJ (1997) Interactive conflict resolution. Syracuse University Press
29. Tjosvold D (1997) Conflict within interdependence: Its value for productivity and individuality. Using Conflict in Organizations, pp 23–37
30. Wang W, Ting S (2011) Development of a computational simulation model for conflict management in team building. Intl J Eng Bus Manag 3(2):9–15
31. Murata T (1989) Petri nets: properties, analysis and application. Proc IEEE 77(4):541–580
32. Nazareth DL (1993) Investigating the applicability of petri nets for rule-based system verification. IEEE Trans Knowl Data Eng 4(3):402–415
33. Looney CG, Alfize AR (1987) Logical control via Boolean rule matrix transformation. IEEE Trans Syst Man Cybern 17(6):1077–1082
34. Looney CG (1988) Fuzzy Petri nets for rule-based decision marking. IEEE Trans Syst Man Cybern 18(1):178–183
35. Jeffrey J, Lobo J, Murata T (1996) A high-level petri net for goal-directed semantics of horn clause logic. IEEE Trans Knowl Data Eng 8(2):241–259
36. Lin C, Chaudhury A, Whinston AB, Marinescu DC (1993) Logical inference of horn clauses in petri net models. IEEE Trans Knowl Data Eng 5(3):416–425
37. Cook DJ, Crandall AS, Thomas BL et al (2013) CASAS: a smart home in a box[J]. Computer. https://doi.org/10.1109/MC.2012.328
38. Mou X, Zhang Q, Liu H, Zhao J (2021) Knowledge representation and acquisition using R-numbers Petri nets considering conflict opinions. Expert Syst. https://doi.org/10.1111/exsy.12660
39. Xu J, Lee YH, Tsai WT, et al. (2009) Ontology-Based Smart Home Solution and Service Composition[M]. DBLP, 2009, 2009 International Conferences on Embedded Software and Systems
40. Humphreys MA, Nicol JF (2002) The validity of iso-pmv for predicting comfort votes in every-day thermal environments. Energy Build 34(6):667–684
41. Sun Y, Wang X, Luo H, Li X (2014) Conflict detection scheme based on formal rule model for smart building systems. IEEE Trans Hum-Mach Syst 45(2):215–227
42. Shehata M, Eberlein A, Fapojuwo A (2007) Using semi-formal methods for detecting interactions among smart homes policies. Sci Comput Program 67(2–3):125–161
43. Fanger PO (1970) Thermal comfort. Danish Technical Press, Copenhagen
44. ISO 7730 (1994) Moderate thermal environments-determination of the PMV and PPD indices and specification of the conditions for thermal comfort. ISO, Geneva
45. Celik Z B, Tan G, Mcdaniel P (2019) IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT[C]//Network and Distributed System Security Symposium
46. Munir S, Stankovic JA (2014) DepSys: Dependency Aware Integration of Cyber-Physical Systems for Smart Homes[C]// 5th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS '14). ACM, 2014, pp.127–138
47. Pradeep P, Krishna K (2022) Conflict detection and resolution in IoT systems: a survey. IoT 3(1):191–218
48. Ma M, Stankovic J A, Feng L (2018) Cityresolver: A decision support system for conflict resolution in smart cities. In Proceedings of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), Porto, Portugal, 11–13 April 2018; IEEE: Porto, Portugal, 2018; pp. 55–64
49. Pradeep P, Pal A, Kant K (2021) Automating Conflict Detection and Mitigation in Large-Scale IoT Systems. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021
50. Carreira P, Resendes S, Santos AC (2014) Towards automatic conflict detection in home and building automation systems. Pervasive Mob Comput 12:37–57
51. Trimananda R, Aqajari S A H, Chuang J, et al. (2020) Understanding and automatically detecting conflicting interactions between smart home IOT applications. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Online, 8–13 November 2020; pp. 1215–1227