Conflict Detection Scheme Based on Formal Rule Model for Smart Building Systems

Yan Sun, Xukai Wang, Hong Luo, and Xiangyang Li

***Abstract*—Smart building systems can provide flexible and configurational sensing and controlling operations according to users' requirements. As the number and the complexity of service rules customized by users have significantly increased, there is an increasing danger of conflict during the interaction process between users and the system. To address this issue, we propose a new rule conflict detection scheme tailored for the smart building system. First, we present a formal rule model UTEA based on User, Triggers, Environment entities, and Actuators. This model can handle not only controlled devices with discrete status but also real-valued environmental data such as temperature and humidity. In addition, this model takes multiple users with different authorities into account. Second, we define 11 rule relations and further classify conflicts into five categories. Third, we implement a rule storage system for detecting conflicts and design a conflict detection algorithm, which can detect the conflict between two rules as well as cycle conflict/multicross contradiction among multiple rules. We evaluated our scheme in a real smart building system with more than 30 000 service rules. The experiment results show that our scheme improves the performance in terms of error/missed-detection rates and running time.**

***Index Terms*—Conflict detection, rule model, service, smart building system.**

I. Introduction

SMART buildings, utilizing networks of electronic devices, embedded sensors, and actuators, provide responsive, effective, and supportive environments to fit occupants' lifestyles and to allow occupants to control lighting, Heating, Ventilation and Air Conditioning and humidity control, and other subsystems. Traditional building automation systems progress slowly due to high cost. Meanwhile, wireless sensor-actuator networks (WSANs) comprise groups of low-cost and self-organized wireless sensors and actuators that cooperate in monitoring and controlling the physical environment [1], [2] and, thus, can enhance smart building systems [3], [4]. As a proof of the feasibility of such a system, we have constructed a smart building system on our campus by deploying WSANs in each building.

Manuscript received February 15, 2014; revised August 23, 2014; accepted October 17, 2014. Date of publication November 20, 2014; date of current version March 12, 2015. This work was supported in part by the National Natural Science Foundation of China under Grant 61272520, Grant 61370196, and Grant 61272517; NSF ECCS-1247944 and NSF CMMI 1436786; and the Research Fund for the Doctoral Program of Higher Education under Grant 20110005110007. This paper was recommended by Associate Editor W. Shen.

Y. Sun, X. Wang, and H. Luo are with the Beijing Key Lab of Intelligent Telecommunication Software and Multimedia, Beijing University of Posts and Telecommunication, Beijing 100876, China (e-mail: sunyan@bupt.edu.cn; wangxk@bupt.edu.cn; luoh@bupt.edu.cn).

X. Li is with the Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: xli@cs.iit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/THMS.2014.2364613

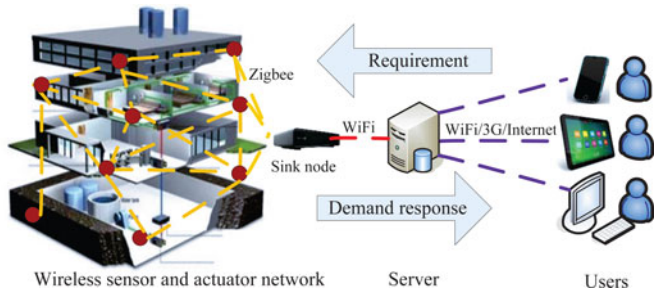


Fig. 1. Overview of the WSAN-based smart building system. Sensors in WSAN collect temperature, light, humidity, smoke, audio, and other environmental data. The actuators include switches, air conditioning controllers, curtain controllers, lighting controllers, and others.

As shown in Fig. 1, according to their requirements, users access a central server to customize services via smart devices, such as mobile phones, tablets, and PCs. For instance, a user can customize the room service: "Open the air-conditioner if room temperature is higher than 30 °C." The server converts the service into rules that can be verified and implemented. Sensors and actuators in WSAN execute the rules and provide the demand response. They also send the corresponding data to the sink node via multihop communication. Sink nodes, which perform multiprotocol conversion, further report this information to the server. Traditional work on smart building systems mainly focuses on issues of automatic control and energy efficiency, but omits the human–machine interaction problem. In [5] and [6], the authors address human activity recognition in the home environment, which can inform the smart building systems from the perspective of human–machine systems.

In this paper, we focus on another important human–machine interaction problem for smart building systems, which concerns conflict detection during the interaction process between multiple users and the system. The increase in services and their variety has brought up several challenges including conflicting services customized by the users. For example, suppose there are two users A and B in a room. The following cases could happen.

Case 1: User A subscribes to service 1 "if PM2.5[1] value is greater than 75 μg/m^3, then close window." In order to understand this service, we analyze the service composition to construct a rule model. In this service, the PM2.5 sensor and the actuator responsible for closing the window are directly

[1]PM2.5 (the full name is fine particulate matter) is microscopic solid or liquid matter suspended in the Earth's atmosphere with the diameter of 2.5 μm or less. As an important air quality index, the PM2.5 concentration can be measured by PM2.5 sensors.

involved. The trigger condition is "if PM2.5 value is greater than 75 μg/m^3," and the execution behavior is "close window." User B subscribes to service 2 "during 7:00 A.M.–7:30 A.M. every day, open window." According to service 2, at 7:00 A.M. every day, the window will open, while at 7:30 A.M. every day, the window will close. If the PM2.5 value meets the trigger condition of service 1 during 7:00 A.M.–7:30 A.M., then the action of closing the window will be executed. In this situation, a conflict between the two services happens. However, if A and B have different permissions, for example, A has the administrator permission and B has the normal user permission, no conflict exists between the two services.

Case 2: User A subscribes to service 1 "if door is closed, then open air-conditioner," while user B subscribes to service 2 "if air-conditioner is open, then close door." The trigger condition of service 2 is dependent on the execution of service 1, and the trigger condition of service 2 is also dependent on the execution of service 1. As a result, the behaviors of services 1 and 2 will be executed repeatedly.

If the service information, such as the trigger condition, executive action, and environmental attributes, is extracted for constructing a rule model with user's authorities, the conflicts can be detected. However, most available methods for conflict detection do not consider multiple users with different authorities in smart building systems. Moreover, the existing detection methods mainly focus on devices with binary states (ON/OFF) without considering the values for most kinds of environmental data.

Taking the above problems into account, this paper proposes a rule conflict detection scheme based on a formal rule model for smart building systems. First, we map sensors and controllers, which directly interact with environment, into physical environment entities with specific attributes, and then, we formalize these environment entities to describe the changes of environment status. We define a formal rule model, UTEA, according to User, Triggers, Environment entities, and Actuators that are involved in the process of rule execution. Based on the disjunctive normal form (DNF) transformation, we decompose a complex rule into multiple basic rules, and we further utilize this formal rule model to abstract and analyze the relation among basic rules. In particular, we achieve a fine-grained division of 11 rule relations including: similar trigger relation, similar action relation, similar prestate relation, contrary poststate relation, explicit dependence relation, implicit dependence relation, negative relation, trigger event relation, peer authority relation, leapfrog authority relation, and compatible authority relation. Based on the rule relations, we classify the rule conflicts into shadow conflict, execution conflict, environmental mutual conflict, direct dependence conflict, and indirect dependence conflict. Finally, we design a rule conflict detection algorithm based on the conflict classification.

The remainder of this paper is organized as follows. Section II presents related work. Section III describes the conceptual architecture of smart building systems. Section IV proposes a rule model and classifies rule conflicts. Section V analyzes the algorithm and the storage technology of rules. Experiments and simulation results are in Section VI. The paper concludes with Section VII.

II. Related Work

To provide smart service, the rule system is an indispensable component in smart buildings. Velik *et al.* [7] addressed both the perception and the decision-making process and present an alerting model based on the rule model of the rule markup language. Chen *et al.* [6] introduced an ontology-based hybrid approach to activity modeling that combines domain knowledge-based model specification and data-driven model learning. Rules can be extracted though daily activities learning and recognition. However, in a smart building system, there are many users and most of the rules are created by users. In fact, manual input may cause inexact logics or input errors and multiuser rule making may also result in content conflicts among multiple rules. Hence, rule conflict detection is essential to ensure the correctness of rule execution [8].

For rule conflict verification, some existing studies have focused on storage structure of services and their conflict detection algorithms [9]–[11]. Petri net-related technologies are widely used in rule verification and fault detection based on expert systems. Yang *et al.* [12] proposed a Petri net model for rule verification in an expert system. Ma *et al.* [13] presented a rule-map-based technique for data inconsistency detection, where rule map was used to describe the hierarchical structure of rules and estimate judgment standards for consistency dynamically. In the rule map, knowledge was illustrated as state sets of related objects, and logical inconsistency can be determined by the relations between those state sets. Xu *et al.* [14] proposed an ontology-based framework to facilitate the automatic composition of services. However, they did not address conflict among services and how to predict mutual conflict among rules.

Shehata *et al.* [15] proposed a semiformal method, called IRIS (Identifying Requirements Interactions using Semiformal methods), for detecting interactions between policies in the smart home domain. A major component within IRIS, which is an interaction taxonomy, was also presented. In addition, the method addressed feature interactions beyond telecommunication features and investigated interactions between policies. The authors also extended their research on IRIS and gave a more comprehensive taxonomy for interaction detection in software systems [16]. Based on the 29 interaction types in [16] and semantic web rules, Hu *et al.* [17] proposed a Semantic Web-based complementary methodology named SPIDER for automated detecting of interactions for user policies in smart homes. The work tackles the process of composition for smart home services and features, that is, user policy interaction detection. Considering that service conflict among smart buildings is produced by function interaction among various nodes, Nakamura *et al.* [18] proposed an object-oriented approach to detect node interaction and judge service conflict. Leelaprute [19] proposed a classification method and conflict solution in the context of smart homes and further introduced model detection techniques to analyze conflict interaction automatically [20]. Luo *et al.* [21] proposed a set of lightweight rule verification

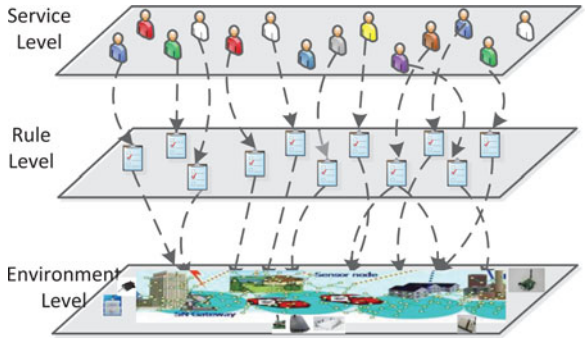


Fig. 2. Three layers of a smart building system.

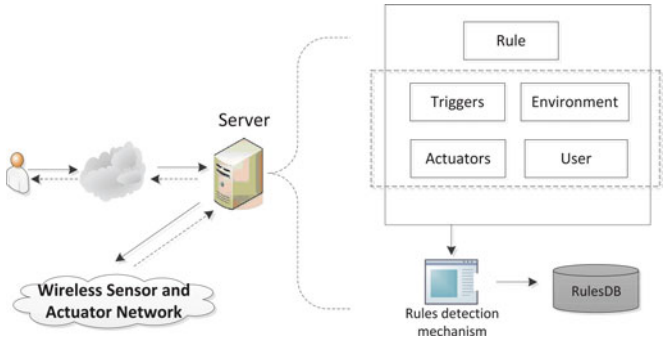


Fig. 3. Rule execution framework.

mechanisms for WSAN, and achieved abnormal rule detection and conflict detection using probability analysis and expression. Xu *et al.* [22] presented a method by embedding the service policy into the traditional WSDL2.0 schema to describe the input-to-output mapping relationships. Nakamura *et al.* [23] proposed an environment impact model and introduced an environment requirement to define the expected environment state achieved by each service. Maternaghan and Turner [24] addressed a technique for offline conflict analysis among policies (the analog of the feature interaction problem).

Most of these rule verification algorithms have focused on the rule model, but they only consider binary state devices and ignore multiuser and the user authority.

III. Conceptual Architecture of Smart Building Systems

Usually, there are many users in a smart building system, and they control the building's environment entities by subscribing services. An environment entity may be controlled by multiple services simultaneously, and the commands executed by the physical environment entity may be inclusive or completely opposite.

To better detect service conflicts among different users in the smart building system, we divide the smart building system into three layers: service layer, rule layer, and environment layer (see Fig. 2). The users only participate to subscribe services and do not need to know how the services perform. This is because the rule layer and the environment layer are transparent to the service layer. In the rule layer, the rule models are constructed for services, which contain the useful information of the corresponding services, including related nodes controlled by rules. The sensor nodes and actuator nodes of smart building are in the environment layer.

Users access the smart building system through customizing their services. The system converts these services into corresponding rules, which are described by a formal description model. As shown in Fig. 3, our model is composed of the user, triggers, environment entities, and actuators. Fig. 3 also illustrates the rule execution framework. The system begins to detect conflict when the model is constructed. If a rule conflicts with another existing rule in the database, this rule will not be executed, and this conflict will be reported to the user. Otherwise, this rule will be stored into the database and the corresponding commands will be sent to a specific node through the wireless networks. After executing this rule, the result will be reported to the user.

To design an algorithm for conflict detection, we abstract each basic component of the rules corresponding to services and further construct a formal description model. In addition, we decompose a complex rule into several basic rules as a basis for the confliction detection algorithm.

IV. Rule Model and Conflict Classification

A. Rule Model

Generally, the process of rule execution is to get specific information from some environment entities and control some other environment entities using this information. According to such a composition relation, we define a rule model based on User, Triggers, Environment entities, and Actuators. Hence, we name this mode UTEA.

Definition 1: A service rule can be represented by a Rule $=$ {User, Triggers, Envirs, Actuators}, where Triggers $=$ $\{\text{Trigger}_1, \text{Trigger}_2, \ldots, \text{Trigger}_n\}$ is a set of trigger conditions, Envirs $=$ $\{\text{Envir}_1, \text{Envir}_2, \ldots, \text{Envir}_n\}$ is a set of environment entities, and Actuators $=$ $\{\text{Actuator}_1, \text{Actuator}_2, \ldots, \text{Actuator}_n\}$ is a set of actuators. The number of triggers, environment entities, and actuators in a rule may be one or more, while the number of users must be one.

1) User: To show various relations of different users conveniently, we introduce the user authority into the service rule.

Definition 2: User $=$ {User_ID, User_Au}, where User_ID uniquely identifies a user and User_Au represents the user authority.

In our system, the value range of User_Au is a positive integer, and the smaller User_Au is, the higher the user authority will be. Different authorities correspond to different priorities. Higher user authority represents higher priority for operating environment entities. For example, a user, whose User_Au is "1," has higher operating priority than the user whose User_Au is "2."

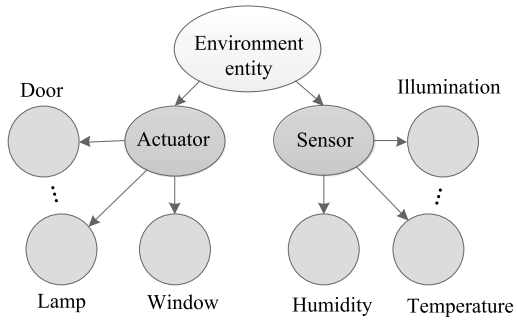


Fig. 4. Environment entity classification.

2) Triggers: No matter whether a rule is simple or complex, it requires some conditions to trigger rule execution. Thus, we abstract and represent the trigger condition as follows.

Definition 3: Trigger = {Trigger_ID, Event, Priority}, where Trigger_ID identities the environment entity involved in the trigger condition, Event is the trigger condition, and Priority represents its priority.

For example, the trigger of rule "if temperature is higher than 30 °C, then open air-conditioner" can be described as Trigger = {Temperature_ID, >30 °C, 1}, where "1" denotes the priority of trigger condition.

3) Environment Entities: As shown in Fig. 4, environment entities can be divided into two categories: sensors and controllers. A given sensor or controller can be defined as follows.

Definition 4: Envir = {E_ID, Pre_s, Next_s}, where E_ID identifies the environment entity, and Pre_s and Next_s represent the states before and after rule execution, respectively. The values of Pre_s and Next_s are both real.

For instance, window is a controller described as Controller = {Air-conditioner_ID, 0, 1}, where "0" indicates the closed state of air-conditioner and "1" indicates the open state. The temperature sensor can be also described as Sensor = {Temperature_ID, >30 °C, <30 °C}. This means the temperature is changed from >30 °C to <30 °C. Using such formal description of environment entities, we can describe the states of sensors/controllers with both Boolean and continuous values.

4) Actuators: Similar to the trigger, the actuator is an essential component of a rule. We abstract and represent the rule action as follows.

Definition 5: Actuator = {Actuator_ID, Action}, where Actuator_ID identifies the environment entity involved in the action, and Action denotes the state of actuator after executing the rule.

For example, the actuator of the rule "if temperature is higher than 30 °C, then open air-conditioner" can be represented as Actuator = {air-conditioner_ID, open}.

A rule can be expressed as $f : t_1, t_2, \ldots, t_n \rightarrow (a_1 \wedge a_2, \ldots, \wedge a_m)$, where $t_1, t_2, \ldots, t_n$ are n triggers, $a_1, a_2, \ldots, a_m$ are m actuators, and f denotes a propositional calculus consisted of the logical operators $\wedge$ and $\vee$.

For example, a rule can be expressed with the formula like $t_1 \wedge t_2 \wedge (t_3 \vee (t_4 \wedge t_5)) \rightarrow a_1 \wedge a_2$. We know that each propositional formula can be converted into an equivalent formula based on DNF. Therefore, the rule $t_1 \wedge t_2 \wedge (t_3 \vee (t_4 \wedge t_5)) \rightarrow a_1 \wedge a_2$ can be decomposed as follows:

$$t_1 \wedge t_2 \wedge (t_3 \vee (t_4 \wedge t_5)) \rightarrow a_1 \wedge a_2$$

$$\Downarrow$$

$$(t_1 \wedge t_2 \wedge t_3) \vee (t_1 \wedge t_2 \wedge t_4 \wedge t_5) \rightarrow a_1 \wedge a_2$$

$$\Downarrow$$

$$(t_1 \wedge t_2 \wedge t_3) \rightarrow a_1 \wedge a_2$$

$$(t_1 \wedge t_2 \wedge t_4 \wedge t_5) \rightarrow a_1 \wedge a_2.$$

The rule $t_1 \wedge t_2 \wedge (t_3 \vee (t_4 \wedge t_5)) \rightarrow a_1 \wedge a_2$ can be decomposed into several basic rules.

Based on the DNF transformation, we decompose a complex rule into multiple basic rules, which provides a foundation for the rule conflict detection. The basic rule can be executed only when the triggers are satisfied simultaneously.

B. Rule Conflict Classification

Based on the four basic components of the formal model rule, we classify the relation among rules into 11 categories: *similar trigger relation*, *similar action relation*, *similar prestate relation*, *contrary poststate relation*, *explicit dependence relation*, *implicit dependence relation*, *negative action relation*, *trigger event relation*, *peer authority relation*, *leapfrog authority relation*, and *compatible authority relation*, as listed in Table I.

Take the *implicit dependence relation* for example. We call a rule R_B implicitly dependent on another rule R_A if two rules satisfy: 1) the environment entities involved in the trigger condition of R_B belong to the environment entities of Rule R_A; and 2) the trigger condition of R_B is one of the states after R_A execution. We then write the conditional expression of implicit dependence relation as

$$ImpDepeAc(R_A, R_B) =$$
$$(E\_ID_A \supseteq Trigger\_ID_B) \wedge (Next\_s_A \supseteq Event_B)$$

where $E\_ID_A \triangleq \{Envir_i.E\_ID, \forall Envir_i \in Envir_A\}$, and $Envir_A$ denotes the set of environment entities for rule R_A. The similar symbols in Table II also follow this definition form.

In fact, two rules can satisfy multiple relations simultaneously. The composition of different relations possibly causes the rule conflict. According to combination of various relations, we classify the rule conflicts into five categories: *shadow conflict*, *execution conflict*, *environment mutual conflict*, *direct dependence conflict*, and *indirect dependence conflict*, as given in Table II. We can judge whether there exist conflicts between two rules or among multiple rules using the conditions in Table II for these five kinds of conflicts.

Conflict C1: Shadow Conflict (SC)

Definition:

For two rules R_A and R_B, if the actuators of R_A and R_B are the same and the trigger condition of R_A is contained in the trigger condition of R_B, then there exists a shadow conflict between R_A and R_B.

TABLE I
RULE RELATION CLASSIFICATION

Relation	Abbreviation	Condition
Similar trigger	SimlTr(R_A , R_B)	Trigger_ID$_{A=B}$ $\wedge$ Event$_{A\subseteq B}$ $\wedge$ Priority$_{A=B}$
Similar action	SimlAc(R_A , R_B)	Actuator_ID$_{A=B}$ $\wedge$ Action$_{A\subseteq B}$
Similar prestate	SimlPr(R_A , R_B)	E_ID$_{A=B}$ $\wedge$ Pre_s$_{A\subseteq B}$
Contrary poststate	ContPs(R_A , R_B)	E_ID$_{A=B}$ $\wedge$ Next_s$_{A\neq B}$
Explicitly dependence	ExplDepe(R_A , R_B)	(Actuator_ID$_A$ = Trigger_ID$_B$) $\wedge$ (Action$_A$ $\supseteq$ Event$_B$)
Implicitly dependence	ImplDepe(R_A , R_B)	(E_ID$_A$ $\supseteq$ Trigger_ID$_B$) $\wedge$ (Next_s$_A$ $\supseteq$ Event$_B$)
Negative action	NegiAc(R_A , R_B)	Actuator_ID$_{A=B}$ $\wedge$ Action$_{A\neq B}$
Trigger event	TrigEve(R_A , R_B)	E_ID$_{A=B}$ $\wedge$ (Pre_s$_A$ $\subseteq$ Next_s$_B$)
Peer authority	PeerAuth(R_A , R_B)	User_Au$_{A=B}$
Leapfrog authority	LeapAuth(R_A , R_B)	User_Au$_{A>B}$
Compatible authority	CompAuth(R_A , R_B)	User_Au$_{A<B}$

TABLE II
CONFLICT CLASSIFICATION

Conflict	Condition
Shadow Conflict	(SimlTr(R_A , R_B) $\wedge$ SimlAc(R_A , R_B)) $\wedge$ (PeerAuth(R_A , R_B)
Execution Conflict	(SimlTr(R_A , R_B) $\vee$ TrigEve(R_A , R_B)) $\wedge$ NegiAc(R_A , R_B) $\wedge$ (PeerAuth(R_A , R_B)
Environment Mutual Conflict	SimlTr(R_A , R_B) $\wedge$ ContPs(R_A , R_B)
Direct Dependence Conflict	(ExplDepe(R_A , R_B) $\vee$ ImplDepe(R_A , R_B)) $\wedge$ (ExplDepe(R_B , R_A) $\vee$ ImplDepe(R_B , R_A))
Indirect Dependence Conflict	(ExplDepe(R_A , R_B) $\vee$ ImplDepe(R_A , R_B)) $\wedge$ (ExplDepe(R_B , R_C) $\vee$ ImplDepe(R_B , R_C)) $\cdots$ $\wedge$ (ExplDepe(R_N , R_A) $\vee$ ImplDepe(R_N , R_A))

Condition:

(SimlTr(R_A, R_B) $\wedge$ SimlAc(R_A, R_B)) $\wedge$ (PeerAuth(R_A, R_B)) $\Longrightarrow$ Shadow Conflict.

Example:

R_1: light intensity $<$ 400Lux $\rightarrow$ open curtain

R_2: light intensity $<$ 500Lux $\rightarrow$ open curtain.

Conflict analysis:

R_1 and R_2 can execute separately, and there is no conflict. However, simultaneous existence of them will result in a redundancy conflict. That is, if R_1 exists and R_2 starts to execute, we can find that the trigger condition of R_1 is contained in R_2 and their trigger priorities are the same. If there is no authority priority between them, a shadow conflict will occur.

Conflict C2: Execution Conflict (EC)

Definition:

For two rules R_A and R_B, if the trigger condition of R_A is contained in the trigger condition of R_B and the actions of the same actuator in R_A and R_B are contrary, then there exists an execution conflict between R_A and R_B.

Condition:

(SimlTr(R_A, R_B) $\vee$ TrigEve(R_A, R_B))$\wedge$ NegiAc(R_A, R_B) $\wedge$ PeerAuth(R_A, R_B) $\Longrightarrow$ Execution Conflict.

Example:

R_3: humidity $<$ 45%RH $\rightarrow$ open humidifier

R_4: humidity $<$ 65%RH $\rightarrow$ close humidifier.

Conflict analysis:

From the descriptions of R_3 and R_4, the trigger condition of R_4 is contained in R_3, and the actions for humidifier are contrary. Hence, when humidity captured by humidifier is less than 45%RH, the behavior of the humidifier is unstable. In addition, if the user authority of R_4 is not less than that of R_3, then an execution conflict will happen.

Conflict C3: Environment Mutual Conflict (EMC)

Definition:

Two rules, R_A and R_B, execute simultaneously. For a given environment entity, if the state (Next_s) after R_A execution is inconsistent with the state after R_B execution, then there exists an environment mutual conflict between R_A and R_B.

Condition:

SimlTr(R_A, R_B) $\wedge$ ContPs(R_A, R_B) $\Longrightarrow$ Environment Mutual Conflict.

Example:

R_5: temperature $>$ 30 °C $\rightarrow$ open air-conditioner $\wedge$ keep temperature below 28 °C

R_6: temperature $<$ 25 °C $\rightarrow$ open heater $\wedge$ keep temperature above 28 °C.

Conflict analysis:

When the temperature exceeds 30 °C, air-conditioner will open for cooling which makes the temperature decrease below 28 °C. Once, the temperature decreases to 25 °C, the heater will open according to R_6. In this way, air-conditioner and heater work at the same time. Then, the environment entity of temperature is possibly in an unstable state, i.e., the environment mutual conflict happens.

Conflict C4: Direct Dependence Conflict (DDC)

Definition:

For two rules R_A and R_B, if the trigger condition of R_B is the state after R_A execution and vice versa, then there exists a direct dependence conflict between R_A and R_B.

Condition:

(ExplDepe(R_A, R_B) $\vee$ ImplDepe(R_A, R_B)) $\wedge$ (ExplDepe(R_B, R_A) $\vee$ ImplDepe(R_B, R_A)) $\Longrightarrow$ Direct Dependence Conflict.

TABLE III
COMPARISON AMONG UTEA, SPIDER, AND IRIS

conflict classification	UTEA	SPIDER	IRIS
Shadow Conflict	√	△	△
Execution Conflict	√	△	△
Environmental Mutual Conflict	√	√	√
Direct Dependence Conflict	√	√	
Indirect Dependence Conflict	√		

√: type of detected conflict.
△: type of conflict detected without considering authority.

Example:

R_7: air-conditioner opens → close window

R_8: window closes → open air-conditioner.

Conflict analysis:

There is interdependency between two rules. The trigger condition of R_7 depends on the execution result of R_8 while the trigger condition of R_8 depends on the execution result of R_7. Thus, both R_7 and R_8 execute in an endless loop and cause a direct dependence conflict.

Conflict C5: Indirect Dependence Conflict (IDC)

Definition:

For multiple rules R_A, R_B, R_C, ..., R_N, if the trigger condition of R_B is the state after R_A execution, ..., and the trigger condition of R_{N-1} is the state after R_N execution, and the trigger condition of R_A is the state after R_N execution, then there exists an indirect dependence conflict among R_A, R_B, R_C, ..., R_N.

Condition:

(ExplDepe(R_A, R_B) ∨ ImplDepe(R_A, R_B)) ⋯ ∧ (ExplDepe(R_{N-1}, R_N) ∨ ImplDepe(R_{N-1}, R_N)) ∧ (ExplDepe(R_N, R_A) ∨ ImplDepe(R_N, R_A)) ⟹ Indirect Dependence Conflict.

Example:

R_9: air purifier opens → close window

R_{10}: window closes → close curtain

R_{11}: curtain closes → open air purifier.

Conflict Analysis:

The trigger conditions of R_{10}, R_{11}, and R_9 depend on the execution results of R_9, R_{10}, and R_{11}, respectively. That is any rule-pair among the three rules is a dependent relation and the dependent relations form a loop.

C. Classification Completeness Analysis

To verify the completeness of conflict classification, we compare UTEA with SPIDER [17] and IRIS [15], as listed in Table III. Symbol "√" represents the type of conflict that can be detected, and symbol "△" represents the type of conflict that can be detected without considering user authority. In our solution, we take user authority into account. Therefore, false conflicts can be avoided and the efficiency of conflict detection can be improved.

The SPIDER method did not consider the continuous states of an environment entity. Therefore, a device with two states can be used in the comparison. Particularly, the SPIDER method cannot detect IDC mainly because that it only focused on two users. The IRIS method considered two kinds of conflicts: 1) negative impact conflict—the two attributes are still preserved, but one attribute will negatively affect the second one; and 2) override conflict—one attribute will override and cancel the other attribute. However, the IRIS method did not consider the dependence between two rules or among more than two rules in some cases. Therefore, it cannot detect DDC as well as IDC. Our method extends the number of rules involved in a conflict detection algorithm from 2 to n ($n \geq 3$), i.e., can detect not only conflicts between two rules but also conflicts among multiple rules (such as cycle conflict and multicross contradiction).

In addition, SPIDER and IRIS do not consider user authority; therefore, their detections on shadow and execution conflicts are incomplete. For example, when there is a conflict between two rules, if the user authority of one rule is higher than that of the other, the conflict does not exist. Therefore, SPIDER and IRIS

TABLE IV
EXAMPLE OF XML-FORMAT RECORD

```
< ?xml version="1.0" encoding="UTF-8"? >
<SubDescription>
    if temperature is more than 25 °C, open air-conditioner
</SubDescription>
<User_ID>7</User_ID>
<Area>
    <Building>Building 3</Building>
    <Floor>9</Floor>
    <Room> 902</Room>
</Area>
<Is_elec_control> true </Is_elec_control>
<IsOver> false </IsOver>
<IsNotify>true</IsNotify>
<Rule>
    <triggerGroup>
        <condition>
            <trigger>
                <sensorType>temperature</sensorType>
                <sensorID> 17</sensorID>
                <clusterWay> SINGLE</clusterWay>
                <operation>More than</operation>
                <threshold> 30 °C</threshold>
            </trigger>
        </condition>
    </triggerGroup>
    <actuatorGroup>
        <actuator>
            <actuator> Air-conditioner</actuator>
            <actuatorID> 33</actuatorID>
            <action>open</action>
        </actuator>
    </actuatorGroup>
</Rule>
```

TABLE V
RULE TABLE FOR DETECTION

Field	Values
ID	Integer
Location	String, identifies the location in location tree
User	XML, including subscriber ID and his authority
Triggers	XML, including Trigger_ID, event and its priority
Environment Entities	XML, including E_ID, Pre_s and Next_s
DependRules	Array, records ID on which this rule depends

methods will increase the amount of false negative detections and reduce the detection efficiency.

In this paper, we summarize five kinds of conflicts after considering conflict conditions in a smart building system. Shadow and execution conflicts contain device conflicts, as a shadow conflict defines redundant action while an execution conflict defines contrary action. Environment mutual conflict is used to detect environment conflict. Besides, considering that dependent relations will exist in rule, we define direct conflict and indirect conflict to detect dependent conflict. In particular, user authority is an essential factor for conflict detection, and it can be included in the process of judgment. Based on the classification mentioned above, we include all conflicts in a smart building system.

V. Rule Storage and Algorithm Design

A. Rule Storage

When a user subscribes to a rule "temperature > 30 °C $\rightarrow$ open air-conditioner," which is executed in Room 902, an XML-record of this rule will be generated, as shown in Table IV. In the XML record, "Rule" tag contains the execution information of triggers and actuators. "SubDescription" tag records the rule content, and "Area" tag records the location where the rule can be applied. When the rule is executed completely, the result contained "IsNotify" tag will be communicated back to the user according to "IsNotify" tag. Besides, "Is_ele_control" tag and "IsOver" tag, respectively, represent whether the actuator is electrical and whether the rule has already executed.

It is convenient to record all the information related to the implementation of a rule using this kind of XML format. However, this XML record contains too much useless information for detecting conflicts. Thus, we need to convert the XML record into a more simple format.

According to Definition 1, we extract relevant fields from the XML record and store them in rule detection base (RDB) including: rule table which stores the rule information used in detection; location tree which records rules control area hierarchically; and authority tree which records the priority of the user.

1) Rule Table: Different from the XML-format rule, the rule table only stores the detection-relevant fields, as listed in Table V. ID identifies the RDB rule. Location represents the place where the rule executes and can be extracted from Location Tree. By location information, we can decide which rule needs to be compared. User indicates the subscriber of this rule and his corresponding authority.

2) Location Tree: For reducing the comparison times during the detection process, we construct a tree to record the hierarchical building structure. Every rule is bound to one node in the location tree according to its control area.

As illustrated in Fig. 5, if a rule R_x is bound to R902, i.e., the control area of R_x is Room 902, we only need to compare R_x with the rules on R902, R_e,...,R_f, for conflict detection.

3) Authority Tree: In order to gain the user's authority of our smart building system, we also utilize a tree structure to store authority.

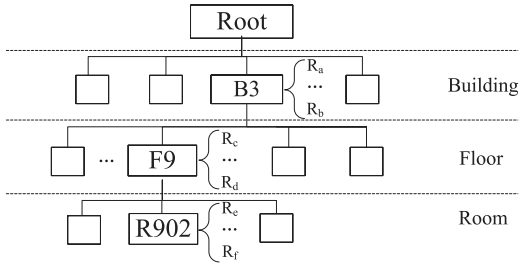


Fig. 5. Location tree.

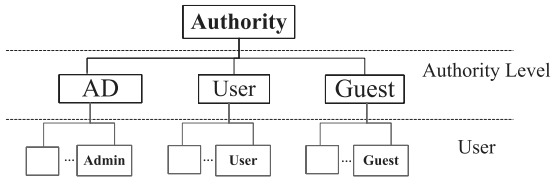


Fig. 6. Authority tree.

In our smart building system, we define three types of authority groups. As shown in Fig. 6, AD represents administrator authority, and User authority belongs to the ordinary users, while the Guest authority is for the guest. According to this storage structure, we can find the authority group to which the user belongs.

B. Conflict Detection Algorithm

After converting the XML-formal rule to the specific recorded rule in RDB, we can use definitions of rule relation (see Table I) and rule conflicts (see Table II) to detect conflicts among rules. We propose Algorithms 1 and 2 to judge the rule relations and detect conflicts, respectively.

Algorithm 1 returns relation, a field of 16 bits that corresponds to the relations in Table I. In Algorithm 1, the relation between R_A and R_B can be compared among user authority, triggers, environment entities, and actuators, and the 11 kinds of relations correspond to the low 11 bits of relation, which are represented with *relation.SimlTr*, *relation.SimlAc*, *relation.SimlPr*, *relation.ContPs*, *relation.ExplDepe*, *relation.ImplDepe*, *relation.NegiAc*, *relation.TrigEve*, *relation.PeerAuth*, *relation.LeapAuth*, and *relation.CompAuth* from the lowest bit to 11th bit orderly. To simplify the calculation in Algorithm 2, the 12th and 13th bits record the explicitly dependence and implicitly dependence relations, respectively, which can be represented with *relation.contrary_ExplDepe* and *relation.contrary_ImplDepe*. For each bit, "1" denotes the corresponding relation exists; otherwise, "0" denotes the corresponding relation does not exist.

Using the output of Algorithm 1, Algorithm 2 detects conflicts and determines whether this rule can be stored in RDB. In Algorithm 2, we traverse each rule in RDB to judge whether there is conflict and, then, record the relation between each rule and R_B using array rel. If there is no shadow conflict, execution conflict, environment contradiction and direct dependence

Algorithm 1 rule_relation(R_A, R_B)

Require: R_A and R_B
Ensure: *relation*
1: **for each** R_B.trigger_ID, R_B.actuator_ID and R_B.e_ID
2: **for each** R_A.trigger **do**
3: **if** R_A.trigger_ID = = R_B.trigger_ID **and** R_A.event contains R_B.event **then**
4: *relation.SimlTr* = 1
5: **if** R_A.trigger_ID = = R_B.E_ID **and** R_B.next_s contains R_A.event
6: *relation.contrary_ImplDepe* = 1
7: **if** R_A.trigger_ID = = R_B.actuator_ID **and** R_B.action contains R_A.event
8: *relation.contrary_ExplDepe* = 1
9: **for each** R_A.actuator **do**
10: **if** R_A.actuator_ID = = R_B.trigger_ID **and** R_A.action contains R_B.event **then**
11: *relation.ExplDepe* = 1
12: **if** R_A.actuator_ID = = R_B.actuator_ID **then**
13: **if** R_A.action contains R_B.action **then**
14: *relation.SimlAc* = 1
15: **if** R_A.action exclude R_B.action **then**
16: *relation.NegiAc* = 1
17: **for each** R_A.envir **do**
18: **if** R_A.E_ID = = R_B.E_ID
19: **if** R_A.pre_s contains R_B.pre_s **then**
20: *relation.SimlPr* = 1
21: **if** R_A.next_s exclude R_B.next_s **then**
22: *relation.ContPs* = 1
23: **if** R_A.pre_s contains R_B.next_s **then**
24: *relation.TrigEve* = 1
25: **if** R_A.E_ID = = R_B.trigger_ID **and** R_A.next_s contains R_B.event **then**
26: *relation.ImplDepe* = 1
27: **if** R_A.user_Au $<$ R_B.user_Au **then**
28: *relation.LeapAuth* = 1
29: **else if**R_A.user_Au = = R_B.user_Au **then**
30: *relation.PeerAuth* = 1
31: **else**
32: *relation.CompAuth* = 1

Algorithm 2 Conflict Detection Algorithm

Require: R_B
1: **for each** R_A in RDB **do**
2: *rel[R_A.ID]* = rule_relation(R_A, R_B)
3: **if** *rel[R_A.ID].SimlTr* = = 1 **then**
4: **if** *rel[R_A.ID].SimlTr* = = 1 **and** *rel[R_A.ID].ContPs* = = 1 **then**
5: **return** Environmental Mutual Conflict
6: **if** *rel[R_A.ID].SimlAc* = = 1 **and**(*rel[R_A.ID].PeerAuth* = = 1 **then**
7: **return** Shadow Conflict
8: **if** *rel[R_A.ID].TrigEve* = = 1 **or** *rel[R_A.ID].SimlTr* = = 1 **then**
9: **if** *rel[R_A.ID].NegiAc* = = 1 **and** (*rel[R_A.ID].PeerAuth* = = 1 **then**
10: **return** Execution Conflict
11: **if** *rel[R_A.ID].ExplDepe* = = 1 **or** *rel[R_A.ID].ExplDepe* = = 1 **then**
12: **if** *rel[R_A.ID].contrary_ExplDepe* = = 1 **or** *rel[R_A.ID].contrary_ImplDepe* = = 1 **then**
13: **return** Direct Dependence Conflict
14: **else**
15: put R_A in Queue **and** put R_A.ID in $R_B.DependedRules$
16: **while** Queue is not empty do
17: R_C = dequeue from Queue
18: **if** *rel[R_C.ID].contrary_ExplDepe* = = 1 **or** *rel[R_C.ID].contrary_ImplDepe* = = 1 **then**
19: **return** Indirect Dependence Conflict
20: **else**
21: **for each** i **do**
22: tmp = $R_C.DependedRules[i]$
23: **put** R_{tmp} in Queue
24: **for each** j **do**
25: **if** *rel[j].contrary_ExplDepe* = = 1 **or** *rel[j].contrary_ImplDepe* = = 1 **then**
26: **add** $R_B.ID$ **to** $R_j.DependRules$
27: **stored** R_B in RDB
28: **return** NoConflict

conflict between two rules, then we use Queue to determine whether there is an indirect dependent conflict. If not, all the rules that are dependent on it will be found and put into Queue using the dependenceRule field of each rule in the array. Repeat the process until the Queue is empty. In this case, it indicates that there exists no indirect dependence conflict between R_B and the rule in RDB, then R_B can be stored into RDB and the dependenceRule field of rule in RDB, which R_B depends on, can be modified.

C. Computational Complexity

Let M_t, M_a, and M_e be the number of triggers, actuators, and environment entities, respectively. M is the sum of M_t, M_a and M_e. In Algorithm 1, from line 2, the time complexity of traversing the triggers, actuators and environment entities of R_A is $O(M)$, where $M = M_t + M_a + M_e$. From lines 4, 11, and 19, we use three "for" loops to compare the relations of two rules which cost $O(M_t)$,$O(M_a)$, and $O(M_e)$, respectively. As a result, the running time of Algorithm 1 is $O(M^2)$.

Let N be the number of rules. In Algorithm 2, from line 1 to line 30, there is one "for" loop running N times. Because function rule_relation(R_A, R_B) in line 2 runs Algorithm 1, and thus, the time complexity of the loop is $O(NM^2)$. The "$while$" loop from line 17 to line 23 is used to detect the indirect dependence conflict. In the worst case, all N rules are dependent, and we need to traverse the N rules for detecting the indirect dependence conflict. This means the "$while$" loop runs up to N times. Generally, for a rule, the number of triggers, actuators, and environment entities are limited, i.e., M is a relatively small number. Therefore, Algorithm 2 runs in $O(N)$.

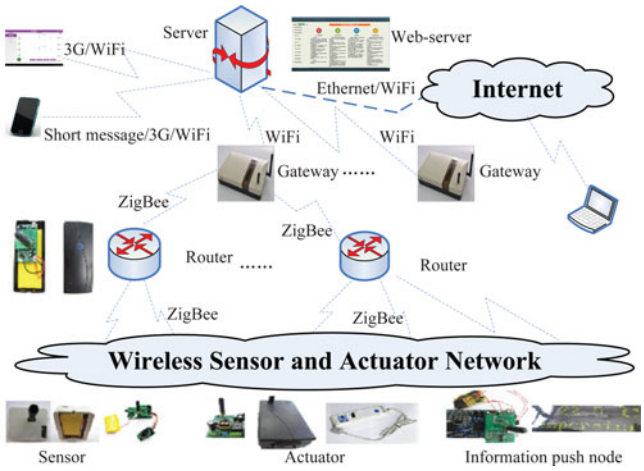


Fig. 7. Real deployment environment of smart building platform. The sensors, actuators, and router are designed with STM32F103 processing chip and CC2530 RF; the gateways are designed with the AT91SAM7X256 processing chip, CC2530 RF, and WiFi module; the server consists of Intel Core2 Duo P8400, 2 GB memory, WiFi module, and 100 M Ethernet card.

VI. Experiment

In order to verify our proposed model and algorithms, we perform a series of experiments, which are based on a practical smart building system. We also compare our scheme with the traditional semantic-based method [17] and semiformal method [15].

A. Platform Introduction

In the smart building platform, we deployed 200 sensor nodes (such as temperature, humidity, light, audio, and image sensor nodes) and 50 actuators (such as lighting and air-conditioner controllers), 50 routers, five gateways, and one server, as illustrated in Fig. 7. The teaching building has ten floors with 20 sensor nodes, five routers, and five actuators on each floor, and a gateway on every two floors. Each sensor node self-discovers its sensing capability, location, and action range and, then, reports the information to routers. The information is further sent to the gateway (sink node). The capabilities of the whole underlying network are given on the web pages. The router in path is responsible for fusing data and transferring the results to the router at the higher level via a multihop ZigBee network and, eventually, to the gateway. The gateway reports all data to the server via a WiFi network. The server analyzes data and makes decision and then forwards customer-ordered information and control commands to the appropriate users and actuators, respectively.

For the results in this paper, the platform stored more than 30 000 rules in the RDB. The following conflict case study is based on the platform.

B. Case Study

To analyze and verify conflicts defined in this paper, we assume that six basic rules have already been stored in the RDB, as listed in Table VI. User G is assigned to Room 902, which is on ninth floor of No. 3 Building, with guest authority (values "2").

TABLE VI
Basic Rules Listed in RDB

Rule ID	Rule Description	Authority
r_1	time = 7:00 A.M. → open window ∧ open lamp.	2
r_2	time = 7:30 A.M. → close window ∧ close lamp.	2
r_3	time ∈ [7:00 A.M., 7:30 A.M.] ∧ illumination value > 80Lux → close lamp.	1
r_4	temperature > 30 °C ∧ infrared = 1 → open air-conditioner ∧ keep temperature below 28 °C.	1
r_5	temperature > 25 °C ∧ infrared = 1 ∧ air-conditioner open → close the fan.	2
r_6	infrared = 1 ∧ CO_2 density > 1000 ppm → open air-conditioner.	2

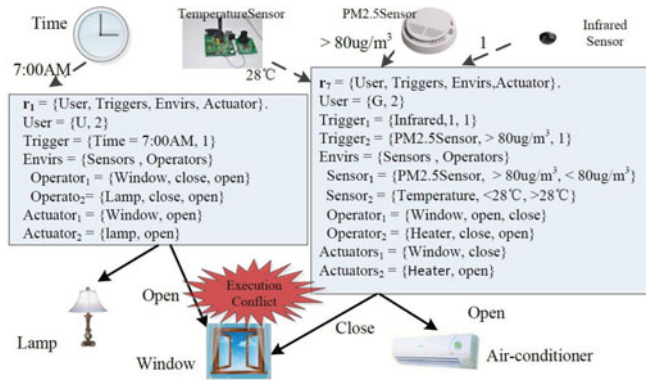


Fig. 8. Case of execution conflict.

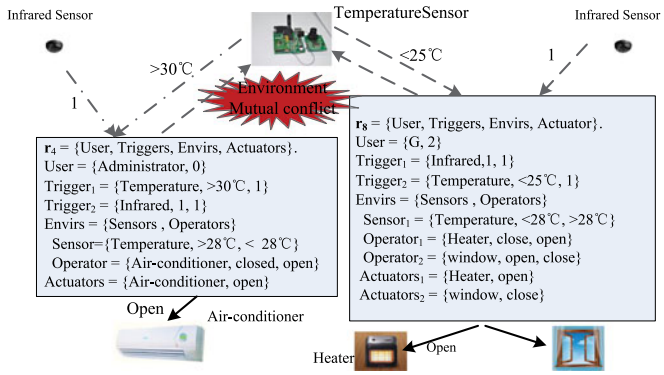


Fig. 9. Case of environment mutual conflict.

User G subscribes to a rule, R_A, which is expressed as "Infrared = 1 ∧ (PM2.5 > 80 μg/m^3 ∨ temperature < 25 °C) → close window ∧ open heater ∧ keep temperature above 28 °C." This rule means "when somebody is at room,[2] if the value of PM2.5 is greater than 80 μg/m^3 or temperature is less than 25 °C, then close the window and open the heater for keeping the temperature above 28 °C." For detecting conflicts efficiently, the rule should be decomposed into basic rules first. Based on DNF,

[2]In our system, the infrared sensors are used for determining whether someone is in a room. "Infrared = 1" means someone is in the room.

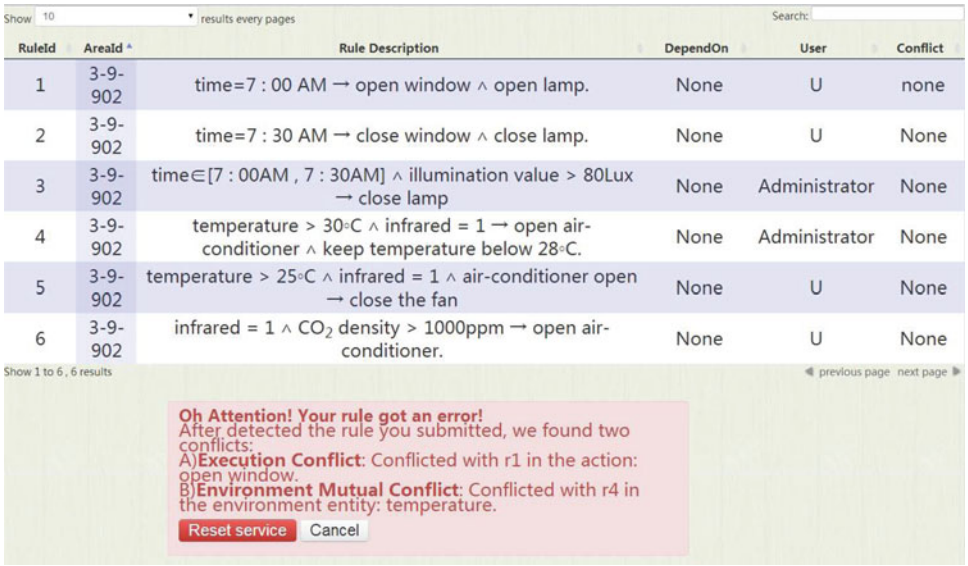


Fig. 10. System reports the detection results to the user.

R_A is decomposed into two basic rules: r_7 "Infrared $= 1 \wedge$ PM2.5 > 80 μg/m^3 $\rightarrow$ close window $\wedge$ open heater $\wedge$ keep temperature above 28 °C" and r_8 "Infrared $= 1 \wedge$ temperature < 25 °C $\rightarrow$ close window $\wedge$ open heater $\wedge$ keep temperature above 28 °C." Then, we detect conflicts among r_7, r_8 and the basic rules in RDB.

According to the UTEA models of r_1 and r_7 (see Fig. 8), the trigger conditions are different: "time $=$ 7:00 A.M." for r_1, and "Infrared $= 1 \wedge$ PM2.5 > 80 μg/m^3" for R_7. When the time is 7:00 A.M., the events of r_7 may be triggered at the same time; that is, the trigger condition of r_7 is contained in r_1, satisfies the trigger event relation. Moreover, the actuators(window) of the two basic rules are the same while the actions are opposite. Then, r_1 and r_7 also satisfy the negative action relation. Due to the same user authority, an execution conflict between r_1 and r_7 occurs.

From the UTEA models of r_4 and r_8 shown in Fig. 9, the action of r_4 is "open air-condition $\wedge$ keep temperature below 28 °C," while the action of r_8 is "close window $\wedge$ open heater $\wedge$ keep temperature above 28 °C"; then, r_4 and r_8 satisfy the contrary poststate relation. Meanwhile, the guest authority of G is lower than that of the administrator. Although their actuators are different, the environment mutual conflict will occur according to the above relations between r_4 and r_8. After conflict detection of r_7 and r_8, the report for conflict detection of R_A will be communicated to user G, as shown in Fig. 10.

Next, we consider the case that the user G submits another rule, R_B, which is expressed as "Infrared $= 1 \wedge$ (temperature >25 °C $\wedge$ fan closed $\vee$ CO_2 density $>$ 1500 ppm) $\rightarrow$ close window $\wedge$ open air-conditioner." R_B means "When somebody is at room, if the temperature is greater than 25 °C and the fan is closed, or CO_2 density $>$ 1500 ppm, then close window and open the air-conditioner." Similarly, two basic rules: r_9 and r_{10} will be decomposed as "Infrared $= 1 \wedge$ temperature $>$ 25 °C $\wedge$ fan closed $\rightarrow$ close window $\wedge$ open air-conditioner" and

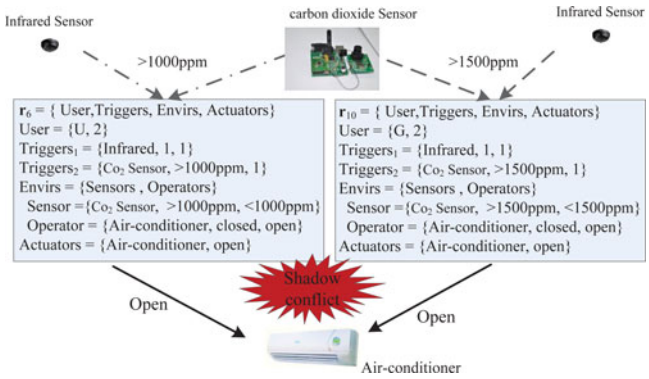


Fig. 11. Case of shadow conflict.

"Infrared $= 1 \wedge CO_2 > 1500$ ppm $\rightarrow$ close window $\wedge$ open air-conditioner." From r_6 and r_{10} illustrated in Fig. 11, the trigger conditions of r_6 and r_{10} are "infrared $= 1 \wedge CO_2 > 1000$ ppm" and "infrared $= 1 \wedge CO_2 > 1500$ ppm," respectively. Then, the trigger condition "$CO_2 > 1000$ ppm" of r_6 contains the "$CO_2 > 1500$ ppm" of r_{10}. In addition, the actuators, actions, and user authorities of the two rules are the same. Therefore, r_6 and r_{10} satisfy the similar trigger relation, and a shadow conflict occurs.

According to the UTEA models of r_5 and r_9 (see Fig. 12), the trigger$_3$ of r_9 and the actuator of r_5 are the same, i.e., the trigger$_3$ of r_9 depends on the execution result of r_5. Moreover, the trigger$_3$ of r_5 depends on the execution result of r_9. Therefore, r_5 and r_9 satisfy the explicit dependence relation, and the direct dependence conflict occurs, which will make the execution enter an endless loop.

C. Efficiency Comparison and Performance Analysis

1) Efficiency Comparison: In the smart home/building domain, IRIS [15] is well known and commonly used as a reference

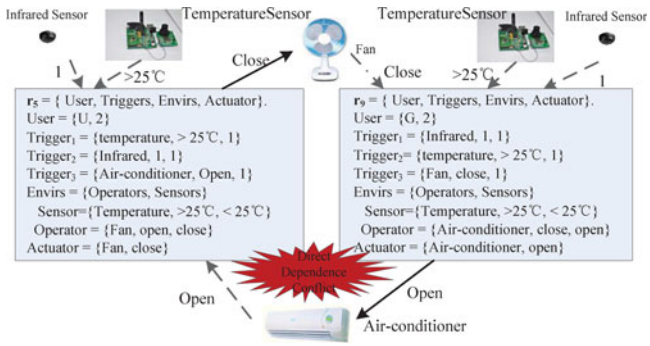


Fig. 12. Case of direct dependence conflict.

method. As mentioned in Section II SPIDER [17], based on the extension of IRIS [16], is also proposed for smart homes specially. Therefore, we compare our UTEA-based scheme with SPIDER and IRIS. We collected 152 rules in Room 902, which are subscribed to by admin users and guest users from Sep. 26, 2013 to Sep. 28, 2013 and utilize UTEA, SPIDER, and IRIS to detect conflicts among these rules, respectively.

From the results listed in Table VII, we have the following.

1) For environment mutual conflict, UTEA, SPIDER, and IRIS all detect 11, 5, and 4 in three days. This means that the three methods have a similar effect on environment mutual conflict detections.
2) Both SPIDER and IRIS detect the same number of shadow and execution conflicts, i.e., they have a similar effect on shadow and execution conflicts detections, but UTEA detects less shadow and execution conflicts. Take Sep. 26, 2013 for example, for shadow and execution conflicts, both SPIDER and IRIS detect 13 and 7, while UTEA detects 7 and 5, respectively. This is because the user authority considered in UTEA can avoid some "fake" shadow and execution conflicts. Thus, the error-detection rate is determined by the proportion of these fake conflicts, which is affected by rule authorities and the number of shadow and execution conflicts.
3) Because IRIS cannot detect both direct and indirect dependent conflicts, the missed-detection rate is the ratio of direct and indirect dependent conflicts number to the total conflicts number. Similarly, the missed-detection rate of SPIDER is the ratio of indirect dependent conflicts number to the total conflicts number. This means the proportions of direct and indirect dependent conflicts affect the difference of missed-detection rate among the three methods.

Therefore, compared with SPIDER and IRIS, UTEA decreases the error-detection rate and the missed-detection rate so as to improve the efficiency of the smart building system.

2) Performance Analysis: In the smart building system, the number of rules increases with the number of users. In Section V-C, we analyzed the complexity of our detection algorithm, which runs in $O(N)$, where N is the number of rules. The runtime of rule detection linearly increases as the size of RDB increases. In order to verify this result, we increase N from 50 to 1000 with the interval of 50. For a given value of N, we calculate the mean times of nonconflict detecting cases and conflict detecting cases, respectively. Fig. 13 illustrates the results. According to linear regression of the results in Fig. 13, we obtain the linear functions of N, Time $= 1.87N + 25.04$ and Time $= 1.096N + 19.66$, for nonconflict detecting cases and conflict detecting cases, respectively. Both of the coefficients of determination are larger than 0.99. This means that the

TABLE VII
COMPARISON AMONG UTEA(U), SPIDER(S), AND IRIS(I)

Conflict Type	Sep. 26, 2013			Sep. 27, 2013			Sep. 28, 2013		
	U	S	I	U	S	I	U	S	I
SC	7	13	13	6	8	8	5	11	11
EC	5	7	7	7	8	8	6	10	10
EMC	11	11	11	5	5	5	4	4	4
DDC	6	6	0	7	7	0	3	3	0
IDC	9	0	0	8	0	0	2	0	0
Total Conflict	38	37	31	33	28	21	20	28	25
Error-detection rate	0%	21.1%	21.1%	0%	9.1%	9.1%	0%	50%	50%
Missed-detection rate	0%	23.7%	39.5%	0%	24.2%	45.5%	0%	10%	25%

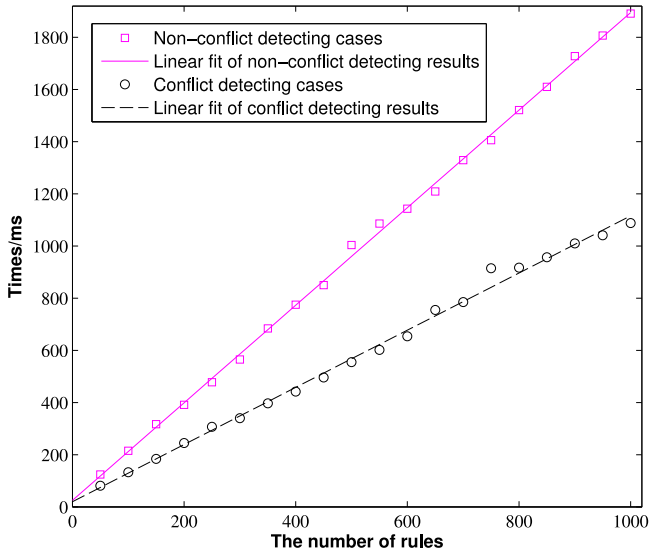


Fig. 13. Consuming time of detection conflict.

TABLE VIII
COMPARISON OF DETECTION AND TIME

Conflict	r_7	r_8	r_9	r_{10}
SC	false	false	false	true
EC	true	false	false	false
EMC	false	true	false	false
DDC	false	false	true	false
IDC	false	false	false	false
Time(s)	0.0011	0.0013	0.0015	0.0013

relation between N and detection time growth is approximately linear.

From the pseudocode of the conflict detection algorithm, Algorithm 2 completely executes if there is no conflict and partially executes if a conflict occurs. Then, the mean time of nonconflict detecting cases is larger than the conflict detecting cases. When N increases to 1000, the conflict detection time is only about 1 s and the detection time of nonconflict is about 1.4 s. This result indicates that our algorithm may be suitable for real-time services in the smart building system.

To analyze runtime performance, we compare our approach to [20]. Here, we use four basic rules from r_7 to r_{10} in the Section VI-B to detect conflicts and calculate time. From Table VIII, "true" means conflict occurs, for example, execution conflict (EC) in r_7 was detected compared with previous rules. These preliminary results are promising as each is faster than the 0.2 ms required in [20].

VII. Conclusion

To improve the efficiency of rule conflict detection in smart building systems, a formal rule model UTEA and a conflict detection scheme are proposed in this paper. The UTEA model contains user, triggers, environment entities, and actuators. The rule relations, defined by four parts of the UTEA model, are utilized to classify rule conflicts. In order to further improve detection efficiency, we design a rule database including rule table, location tree, and authority tree to filter out useless information in the XML format. We design a rule conflicts detection algorithm, which can detect conflicts between two rules as well as cycle conflict/multicross contradiction among multiple rules. Furthermore, we verify our algorithm in the smart building system deployed in our campus.

In this paper, to detect conflicts among complex policies, we decompose complex policies into several basic simple rules based on DNF and then detect conflicts among these basic simple rules. This parser approach is effective but with low efficiency sometimes. In the future work, we will further study the constitution of complex policies and then construct an efficient model for detecting the conflicts among complex policies directly.

References

[1] T. Wark, D. Swain, C. Crossman, P. Valencia, G. Bishop-Hurley, and R. Handcock, "Sensor and actuator networks: Protecting environmentally sensitive areas," *IEEE Pervasive Comput.*, vol. 8, no. 1, p. 30–36, Jan.–Mar. 2009.

[2] E. C.-H. Ngai, J. Liu, and M. R. Lyu, "An adaptive delay-minimized route design for wireless sensor-actuator networks," *IEEE Trans. Veh. Technol.*, vol. 58, no. 9, p. 5083–5094, 2009.

[3] E. Jafer, O. B. Flynn, O. C. Mathuna, and W. Wang, "Design of miniaturized wireless sensor mode and actuator for building monitoring and control," in *Proc. 17th Int. Conf. Telecommun.*, Doha, Qatar, 2010, pp. 887–892.

[4] C. Suh and Y.-B. Ko, "Design and implementation of intelligent home control systems based on active sensor networks," *IEEE Trans. Consumer Electron.*, vol. 54, no. 3, pp. 1177–1184, Nov. 2008.

[5] C.-H. Lu, Y.-C. Ho, Y.-H. Chen, and L.-C. Fu, "Hybrid user-assisted incremental model adaptation for activity recognition in a dynamic smart-home environment," *IEEE Trans. Human-Mach. Syst.*, vol. 43, no. 5, pp. 421–436, Sep. 2013.

[6] L. Chen, C. Nugent, and G. Okeyo, "An ontology-based hybrid approach to activity modeling for smart homes," *IEEE Trans. Human-Mach. Syst.*, vol. 44, no. 1, pp. 92–105, Feb. 2014.

[7] R. Velik and H. Boley, "Neurosymbolic alerting rules," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, pp. 3661–3668, Nov. 2010.

[8] Xu, Yuemei, Niu, Wenjia, and T. Hui, "A policy-based web service redundancy detection in wireless sensor networks," *J. Netw. Syst. Manage.*, vol. 21, no. 3, pp. 384–407, 2013.

[9] H. Lee, Y.-K. Jeong, and I.-W. Lee, "A mechanism of ontology-based rule management for smart building energy saving service," in *Proc. 2012 Int. Conf. Digital Object Identifier*, 2012, pp. 737–738.

[10] S. Chakraborty, T. Ito, R. Kanamori, and T. Senjyu, "Application of incentive based scoring rule deciding pricing for smart houses," in *Proc. Power Energy Soc. General Meeting*, 2013, pp. 1–5.

[11] M. Shehata, A. Eberlein, and A. Fapojuwo, "Managing policy interactions in KNX-based smart homes," in *Proc. 31st Annu. Int. Comput. Softw. Appl. Conf.*, 2007, vol. 2, pp. 367–378.

[12] S. J. H. Yang, A. S. Lee, W. C. Chu, and H. Yang, "Rule base verification using petri nets," in *Proc. 22nd Annu., IEEE Comput. Soc. Int. Comput. Softw. Appl. Conf.*, Vienna, Austria, 1998, pp. 476–481.

[13] J. Ma, J. Lu, and G. Zhang, "A rule-map based technique for information inconsistency verification," in *Proc. IEEE Comput. Soc. Inf., Decision Control*, Adelaide, Australia, 2007, pp. 296–301.

[14] J. Xu, Y. Lee, and W. Tsai, "Ontology-based smart home solution and service composition," in *Proc. Int. Conf. Embedded Softw. Syst.*, Hangzhou, China, May 2009, pp. 297–304.

[15] M. Shehata, A. Eberlein, and Fapojuwo,, "Using semi-formal methods for detecting interactions among smart homes policies," *Sci. Comput. Program.*, vol. 67, no. 2, pp. 125–161, 2007.

[16] M. Shehata, A. Eberlein, and Fapojuwo, "A taxonomy for identifying requirement interactions in software systems,"*Comput. Netw.*, vol. 51, pp. 398–425, 2007.

[17] H. Hu, D. Yang, L. Fu, H. Xiang, C. Fu, J. Sang, C. Ye, and R. Li, "Semantic web-based policy interaction detection method with rules in smart home for detecting interactions among user policies," *IET Commun.*, vol. 5, no. 17, pp. 2451–2460, 2011.

[18] M. Nakamura, H. Igaki, and K.-I. Matsumoto, "Feature interactions in integrated services of networked home appliances," in *Proc. Int. Conf. Feature Interactions Telecommun. Netw. Distrib. Syst.*, 2005, pp. 236–251.

[19] P. Leelaprute, "Resolution of feature interactions in integrated services of home network system," in *Proc. Asia-Pacific Conf. Commun.*, Bangkok, Thailand, Oct. 2007, pp. 363–366.

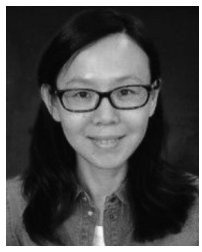
[20] P. Leelaprute, T. Matsuo, T. Tsuchiya, and T. Kikuno, "Detecting feature interactions in home appliance networks," in *Proc. ACIS Int. Conf. Softw. Eng., Artificial Intell., Netw. Parallel/Distrib. Comput.*, Phuket, Thailand, Aug. 2008, pp. 895–903.

[21] H. Luo, R. Wang, and X. Li, "A rule verification and resolution framework in smart building system," in *Proc. 19th IEEE Int. Conf. Parallel Distrib. Syst.*, 2013, pp. 438–439.

[22] Y. Xu, W. Niu, H. Tang, G. Li, Z. Zhao, and S. Ci, "A policy-based web service redundancy detection in wireless sensor networks,"*J. Netw. Syst. Manage.*, vol. 21, pp. 1–24, 2013.

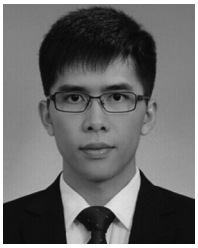
[23] M. Nakamura, K. Ikegami, and S. Matsumoto, "Considering impacts and requirements for better understanding of environment interactions in home network services,"*Comput. Netw.*, vol. 57, pp. 2442–2453, 2013.

[24] C. Maternaghan and K. J. Turner, "Policy conflicts in home automation,"*Comput. Netw.*, vol. 57, pp. 2429–2441, 2013.



Yan Sun received the B.S. degree from Beijing Jiaotong University, Beijing, China, in 1992, and the M.S. and Ph.D. degrees from the Beijing University of Posts and Telecommunications, Beijing, in 1999, and 2007, respectively.

She is currently an Associate Professor with the School of Computer Science, Beijing University of Posts and Telecommunications, Beijing. She is also a Research Member of the Beijing Key Lab of Intelligent Telecommunication Software and Multimedia. Her research interests include internet of things, sensor networks, smart environments, and embedded systems.



Xukai Wang received the B.S. degree in computer science from Beijing University of Chemical Technology, Beijing, China, in 2012. He is currently working toward the M.S. degree with Internet of Things Lab, Beijing University of Posts and Telecommunications, Beijing. His research interests include internet of things, sensor networks, and smart environments.

Hong Luo received the B.S., M.S., and Ph.D. degrees from the Beijing University of Posts and Telecommunications, Beijing, China, in 1990, 1993, and 2006, respectively.

She is currently a Professor with the School of Computer Science, Beijing University of Posts and Telecommunications. She is also a Research Member of the Beijing Key Lab of Intelligent Telecommunication Software and Multimedia. Her research interests include internet of things, wireless networking, sensor networks, smart environments, and communication software.

Xiangyang Li received the Bachelor's degree from the Department of Computer Science and the Bachelor's degree from the Department of Business Management, Tsinghua University, Beijing, China, both in 1995, and the M.S. and Ph.D. degrees from the Department of Computer Science, University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2000 and 2001, respectively.

He is a currently a Professor with the Illinois Institute of Technology, Chicago, IL, USA. He holds the EMC-Endowed Visiting Chair Professorship with Tsinghua University. He currently is a Distinguished Visiting Professor with Xi'an JiaoTong University and the University of Science and Technology of China. His research interests include wireless networking, mobile computing, security and privacy, cyber physical systems, smart grid, social networking, and algorithms.