BuildingRules: A Trigger-Action–Based System to Manage Complex Commercial Buildings

ALESSANDRO A. NACCI and VINCENZO RANA, Politecnico di Milano
BHARATHAN BALAJI, University of California San Diego
PAOLA SPOLETINI, Kennesaw State University
RAJESH GUPTA, University of California San Diego
DONATELLA SCIUTO, Politecnico di Milano
YUVRAJ AGARWAL, Carnegie Mellon University

Modern Building Management Systems (BMSs) have been designed to automate the behavior of complex buildings, but unfortunately they do not allow occupants to customize it according to their preferences, and only the facility manager is in charge of setting the building policies. To overcome this limitation, we present BuildingRules, a trigger-action programming-based system that aims to provide occupants of commercial buildings with the possibility of specifying the characteristics of their office environment through an intuitive interface. Trigger-action programming is intuitive to use and has been shown to be effective in meeting user requirements in home environments. To extend this intuitive interface to commercial buildings, an essential step is to manage the system scalability as large number of users will express their policies. BuildingRules has been designed to scale well for large commercial buildings as it automatically detects conflicts that occur among user specified policies and it supports intelligent grouping of rules to simplify the policies across large numbers of rooms. We ensure the conflict resolution is fast for a fluid user experience by using the Z3 SMT solver. BuildingRules backend is based on RESTful web services so it can connect to various BMSs and scale well with large number of buildings. We have tested our system with 23 users across 17 days in a virtual office building, and the results we have collected prove the effectiveness and the scalability of BuildingRules.

CCS Concepts: • **Information systems** → **Triggers and rules**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Human-centered computing** → *User interface management systems*; *Ubiquitous and mobile computing*;

Additional Key Words and Phrases: Cyber-physical systems, mobile and ubiquitous systems, smart environment

ACM Reference format:
Alessandro A. Nacci, Vincenzo Rana, Bharathan Balaji, Paola Spoletini, Rajesh Gupta, Donatella Sciuto, and Yuvraj Agarwal. 2018. BuildingRules: A Trigger-Action–Based System to Manage Complex Commercial Buildings. *ACM Trans. Cyber-Phys. Syst.* 2, 2, Article 13 (May 2018), 22 pages.
https://doi.org/10.1145/3185500

This work was partially funded by Telecom Italia S.p.A., Strategy and Innovation/Open Innovation Research, Joint Open Lab S-Cube.

Authors' addresses: A. A. Nacci, V. Rana, B. Balaji, P. Spoletini, R. Gupta, D. Sciuto, and Y. Agarwal; emails: {alessandro.nacci, vincenzo.rana}@polimi.it, bbalaji@cs.ucsd.edu, pspoleti@kennesaw.edu, rgupta@ucsd.edu, donatella.sciuto@polimi.it, yuvraj@cs.cmu.edu.

https://doi.org/10.1145/3185500

1 INTRODUCTION

In the past few years, many research projects have focused on making long-standing smart building visions technically feasible, up to the point that augmented buildings are starting to become reality. However, living in and interacting with such spaces typically introduces a huge complexity while offering quite limited benefits. As recently discussed in two of the top conferences in the smart buildings field (Ubicomp[1] 2014 and 2015 and CHI[2] 2015) (De Russis and Corno 2015; Mennicken et al. 2014; Woo and Lim 2015; Huang and Cakmak 2015), one of the main challenges that still have to be solved is the automatic coordination of all the components of a smart building. This issue is crucial and requires a careful management of the policies that define the behavior of the complex distributed systems realizing generic smart buildings.

Currently, smart building policies are written manually by technicians and specified by building managers, which exploits their expertise and knowledge to accomplish this task. However, to enable widespread diffusion of these smart buildings and to allow the occupants to directly interact with them, it is necessary to automate the policies definition procedure with an easy-to-use programming interface, so that also users with a non-technical background can take advantage of this technology.

Within this context, Mennicken et al. (2014) underlines how it is now fundamental to focus on the interaction between humans and the smart-things network, i.e., sensors, actuators, and smart appliances. To this end, many technological aspects have to be faced: The right level of abstraction has to be defined, as well as the right model for the system, to let the users express their requirements (e.g., their comfort preferences) in a clear and simple way, without taking into account the low-level implementation details, such as the state of the sensors. As demonstrated in De Russis and Corno (2015), in fact, users find it non-intuitive to directly interact with sensors and actuators.

Starting from the preliminary work we presented at Ubicomp'15 (Nacci et al. 2015), in this article, we present an in-depth design of our trigger-action programming paradigm–based BuildingRules system for commercial buildings. Our approach is based on the idea of separating the language provided to the users to express the rules and the backend formal language used to represent the rules. With BuildingRules, users will be able to easily describe the rules of interest, and the internal formal representation will be instead used to automatically and unambiguously detect conflicts, i.e., unfeasible situations. We focus on quickly resolving conflicts that occur between user roles using the Z3 SMT solver. Our grouping and access control mechanism allows for building managers to specify high-level policies as well as external applications such as Automated Demand Response to interact with user policies. BuildingRules has been designed to solve most of the issues that currently exist to enable personalized and complex user policies, as highlighted by the smart buildings community (De Russis and Corno 2015; Mennicken et al. 2014).

2 CONTEXT DEFINITION AND GENERAL OVERVIEW

Over the years, commercial buildings have evolved to satisfy the different requirements of present day enterprises. Typically, modern buildings have centralized Heating, Ventilation, and Air Conditioning (HVAC) systems in addition to lighting, fire safety, elevators, and security systems. While there are numerous Building Management Systems (BMSs) (Johnson Controls nd; Siemens Building Technologies nd; Niagara AX nd) to manage and control buildings, these have evolved from traditional HVAC controls and do not support emerging smart building applications such as integration with the Smart Grid (Amin and Wollenberg 2005), Microgrid (Lasseter and Paigi 2004), Demand Response (Rahimi and Ipakchi 2010), and Building Automation (Agarwal et al. 2011;

[1] ACM International Joint Conference on Pervasive and Ubiquitous Computing.
[2] ACM Conference on Human Factors in Computing Systems.

Pichler et al. 2011; Majumdar et al. 2012). Recently, web service–based BMSs have been proposed (Agarwal et al. 2012; Arjunan et al. 2012; Dawson-Haggerty et al. 2013) to better address the challenges and requirements of scalability, maintainability, and easier application development.

BMSs deployed today (Johnson Controls nd; Siemens Building Technologies nd; Niagara AX nd) are mainly designed for building managers and maintenance personnel. Occupants interact with buildings in a limited way—e.g., by using thermostats for HVAC control, switches for lights, keys cards for locks, and outlets for plug loads. With existing BMSs, it is not possible for the occupants to automate and personalize the environment such as setting the temperature according to outside weather or automatically brewing coffee at 8am, and so on. Modern web service–based BMSs, and advanced sensor technology, would provide the flexibility to express and implement such policies that can improve occupant comfort and productivity (Haynes 2008) as well as building energy efficiency (Erickson and Cerpa 2012; Krioukov and Culler 2012; Balaji et al. 2013a).

However, while giving occupants the ability to personalize their living environment is promising, there are several challenges that must be addressed. First, building occupants do not understand the details of the building infrastructure and are not necessarily programmers. As shown in prior work (Sohn and Dey 2003; Truong et al. 2004; Ur et al. 2014), occupants prefer not to interact with sensors and actuators directly; for instance, they relate better to "someone walked into a room" than "motion sensor was activated." Therefore, it is critical that the right level of abstraction and an intuitive user interface is provided by a building automation system to enable occupants with varying levels of programming expertise to express their preferences (Ur et al. 2014). In addition, it is also crucial to provide the appropriate access control mechanisms when the number of users—i.e., both occupants and building managers—increases to ensure proper system operation. Finally, when multiple users customize the same spaces, a scalable mechanism to detect and resolve conflicts becomes necessary. Existing BMSs, in addition to being mainly designed for the facility managers, also have limited or no support for such type of access control or conflict resolution mechanisms.

To address the above challenges, we present the design and the implementation of BuildingRules, a system that allows building occupants to express their comfort preferences and can detect and resolve potential conflicts among them. BuildingRules is based on the *trigger-action programming* paradigm, i.e., occupants can express policies using the "IF something happens THEN do something" (IFTTT) pattern. Prior work has shown that the trigger-action programming is an expressive and intuitive interface to implement building automation policies for people without programming experience (Dey et al. 2006; Ur et al. 2014). BuildingRules extends the IFTTT abstraction to commercial buildings and addresses the challenges in integrating the system with our web service BMS (Agarwal et al. 2012). While similar systems have been proposed for *smart homes*, commercial buildings are significantly more complex due to their scale and their shared nature, where multiple occupants with different needs live and work in the same space, thus leading to more conflicts. To study the extent of conflicts, we conducted a survey with 72 users asking for their preferred rules in shared office spaces of varying capacity. The survey revealed conflicts in 99% of the cases (more details can be found in Section 6). To resolve these conflicts, in BuildingRules we leverage techniques from context-aware frameworks to check for conflicts at the moment of the policy definition (Zhang and Brügge 2004; Park et al. 2005) and assign rule priority to resolve conflicts that arise during actuation (Ranganathan and Campbell 2003). Furthermore, we show that BuildingRules is able to keep the latency of conflict detection low enough to ensure a good user experience.

Typically, the facility managers of commercial building set up automation policies by using the existing BMS, such as the minimum allowable temperature or air flow. It is critical that occupants customizations do not violate these policies. Furthermore, occupants should not be able to control rooms to which they do not have access to. As automated applications such as Demand

Response (Alliance nd) become prevalent, BuildingRules also needs to incorporate their policies. BuildingRules supports hierarchical levels of policy expression to address these challenges, and extends an open source web service–based BMS (Agarwal et al. 2012; Weng et al. 2013). Since BuildingRules is targeting commercial buildings, it has to scale to many hundreds of rooms. Notice that the number of occupants in the building can be very high, but the space is generally divided and each room (e.g., an office) is shared among a few people. For this reason, the system is designed to scale well with the number of spaces: We achieve this by implementing different design choices. First, BuildingRules supports the definition of rules for the entire building, or a subset of rooms, using a grouping mechanism. We expect only a few spaces shared across many users such as restrooms and kitchens. For such spaces, if the conflict resolution does not provide a satisfying solution, then the building manager can assign the rights of managing the room to a restricted number of users, who are considered to be the administrators of that space. Our conflict resolution mechanism can work in parallel for each room, and thus, the system latency latency does not increase with the number of rooms.

We evaluate BuildingRules by creating a virtual office environment with 30 rooms, and testing it on 23 users spread across 17 days. We show that the conflict detection latency is around 250ms in the worst case, and around 100ms in the average case; 636 rules were specified during the experiment, and we detected up to 50 conflicts in a day.

3 BACKGROUND AND RELATED WORK

It has been shown that automation reduces time spent by occupants to manage their office environment, such as adjusting temparature and light levels, which in turn improves their productivity (Haynes 2008). In addition, current buildings provide a limited control to the occupants, and they either ask the facility managers to override default settings or implement ad hoc solutions such as space heaters (Erickson and Cerpa 2012), leading to energy wastage and deviation from designed operation (Mills 2011). Providing control to the occupants by design, i.e., within the boundaries specified by the building manager, would help reducing energy wastage while improving the comfort and satisfaction to the occupants (Haynes 2008; Erickson and Cerpa 2012; Krioukov and Culler 2012).

Moreover, current sensing technology—occupancy sensors (Balaji et al. 2013b; Beltran et al. 2013), light sensors (DeBruin et al. 2013; Roisin et al. 2008), and plug meters (Jiang et al. 2009; Weng et al. 2011)—enable automation applications for office buildings. Several context-aware frameworks and web service–based BMSs have been developed in anticipation of such sensors (Agarwal et al. 2012; Dawson-Haggerty et al. 2013; Dey et al. 2001). For instance, Greenberg (2001) and Bellotti and Edwards (2001) observe that users need to be an integral part of such systems, as it is not possible to automatically infer their specific wishes with the sensing technology available today.

For what concerns the definition of custom policies, trigger-action programming has emerged as a promising solution to involve users in home automation, as it provides an expressive and an intuitive interface (Sohn and Dey 2003; Truong et al. 2004; Ur et al. 2014). Dey et al. show that the IFTTT paradigm expresses 95% of all the applications envisioned by users in smart homes, demonstrating its expressiveness across a wide set of context-aware applications (Dey et al. 2006). More recently, Ur et al. showed that 63% of smart-home applications requested by occupants required programming, and *all* of these applications could be expressed by the IFTTT paradigm (Ur et al. 2014). IFTTT.com is a popular web application that already uses this methodology for connecting various web services and different smart-home appliances (e.g., Belkin Wemo) (IFTTT nd). With BuildingRules, we focus on extending trigger-action programming to provide personalized automation in complex commercial buildings and address the challenges

that emerge when deploying such a system on a large scale, such as the resolution of conflicts that arise due to incompatible user requirements.

Existing BMSs already support conflict resolution to some extent. The Building Automation and Control Networks (BACnet) protocol is a widely adopted standard by industrial BMSs (Bushby 1997), with support for a priority table for every writable sensor. Conflicts are resolved by assigning levels of priority to the different applications, depending on their importance. Web service BMSs extend this methodology to include access control and provide more metrics for conflict resolution. SensorAct (Arjunan et al. 2012) manages permissions through a combination of priority and guard-rules, a script that specifies validation conditions based on time, date, duration, location, and the frequency of operation. BOSS (Dawson-Haggerty et al. 2013) defines every sensor write as a transaction to resolve conflicts. BuildingDepot (Weng et al. 2013) proposes a combination of priority and lease times at the sensor level. However, conflict resolution in these systems is at the sensor level and does not involve the users by design. Conflict resolution has been studied extensively also in context-aware systems (Resendes et al. 2013). A number of conflict resolution strategies focus on automatically resolving application inconsistencies without involving the user (Xu and Cheung 2005; Ranganathan and Campbell 2003). Systems that involve humans need to abstract information such that users can understand the nature of these conflicts and can resolve them (Dey et al. 2006). CARISMA (Capra et al. 2003) resolves conflicts among multiple users for a single application with pre-recorded preferences. Park et al. (2005) extend the conflict resolution to multiple applications, incorporating user preference and user intent in the application metadata. To avoid conflicts, BuildingRules checks if a proposed rule conflicts with the existing rules and shows the conflicting rules back to the user so that he or she can modify them appropriately. The rules are converted to first-order logic and checked using an Satisfiability Modulo Theories (SMT) solver (De Moura and Bjørner 2008), similarly to the strategy followed by Zhang and Brügge (2004). Some of the conflicts cannot be detected at the time of rule specification as the conflict is evident only during actuation. Users specify a priority of rules to resolve these *runtime* conflicts, similarly to the solution proposed by Gaia (Ranganathan and Campbell 2003).

In this work, we employ Z3 as the SMT solver for resolving conflicts (De Moura and Bjørner 2008). This choice provide us with advantages with respect to the other state-of-the-art solutions. For instance, Park et al. (2005) use JESS to build and execute their context-aware sets of rules and also manage conflicts among rules. Unlike Z3, Java Expert System Shell (JESS) (Friedman-Hill 2003) is not an SMT solver but a rule engine for the Java platform, which supports the development of rule-based systems that can be tightly coupled to code written entirely in Java. In their system, a context variable can change only in two directions (increase and decrease) and when rules cause a change in opposite directions to the same variable simulataneously, a conflict is detected. Thus, the conflict is detected only when actuation by a new activated rule is in contrast with the already active rules. Instead, we analyze conflict as soon as a rule is added, even if it is not active yet. This allows us to solve potential conflicts before they actually arise. Moreover, since JESS is not an SMT solver, it has to be enriched with inference rules to define what is considered a conflict. Thus, every time a rule type or structure is added, new customized inference rules need to be added as well.

Besides the work proposed for smart buildings, the literature on policy definition and policy conflict management is also related to our work. Since a policy is a rule that enables the execution of an action when an event takes place and if a pre-defined condition holds, it can be expressed as an event-condition-action and it is called ECA-rule. There are many different policy languages, such as Ponder (Damianou et al. 2001) and PDL (Chomicki et al. 2000). Policy languages have in general a built-in conflict detection and resolution approach and are defined for a specific context. Ponder is a declarative, object-oriented language defined to specify security and management policies. The language allows the specification of both primitive and composite ECA-rules. Analogously, PDL

Table 1. Currently Supported Rule Triggers (T) and Actions (A) Categories

	TYPE	DATA	CATEGORY	EXAMPLE NAME	EXAMPLE HUMAN READABLE SYNTAX	EXAMPLE Z3 SMT TRANSLATION
1	T	BOOLEAN	OCCUPANCY	OCCUPANCY_TRUE	someone is in the room	(inRoom)
2	T	INTEGER	EXT_TEMPERATURE	EXT_TEMPERATURE_RANGE	external temperature is between @val and @val	(and (>= (extTempInRoom) @val) (<= (extTempInRoom) @val))
3	T	INTEGER	TIME	TIME_RANGE	time is between @val and @val	(and (>= (time) @val) (<= (time) @val))
4	T	BOOLEAN	DATE	DATE_RANGE	the date is between @val and @val	(and (>= (day) @val) (<= (day) @val))
5	T	BOOLEAN	WEATHER	SUNNY	it is sunny	(sunny)
6	T	INTEGER	ROOM_TEMPERATURE	ROOM_TEMPERATURE_RANGE	room temperature is between @val and @val	(and (>= (tempInRoom) @val) (<= (tempInRoom) @val))
7	T	BOOLEAN	DEFAULT_STATUS	NO_RULE	no rule specified	(noRule)
8	T	INTEGER	DAY	TODAY	today is @val	(= (today) @val)
9	T	BOOLEAN	EXTERNAL_APP	CALENDAR_MEETING	calendar meeting event	(meetingEvent)
10	A	BOOLEAN	LIGHT	LIGHT_ON	turn on the room light	(light)
11	A	BOOLEAN	WINDOWS	WINDOWS_OPEN	open the windows	(openWindows)
12	A	INTEGER	HVAC	SET_TEMPERATURE	set temperature between @val and @val	(and (>= (tempSetpoint) @val) (<= (tempSetpoint) @val))
13	A	BOOLEAN	APPLIANCES	COFFEE_ON	turn on the coffee machine	(coffee)
14	A	BOOLEAN	MESSAGES	SEND_COMPLAIN	send complain to building manger	(sendComplain)
15	A	BOOLEAN	CURTAINS	CURTAINS_OPEN	open the curtains	(openCurtains)

Note: An example of trigger or action for each category is provided.

allows the description of ECA-rules that can be translated to Datalog. Both languages are equipped with a priority and grouping mechanism to specify to which group the rules apply and with which priority and with conflict resolution mechanism based on *meta-policies* that describes what to do when a conflict happens. The meta-rules define the conflicts and the events that characterized the conflict are monitored. When these events are detected, the resolution actions are performed. Differently from our approach, Ponder and PDL require the ad hoc definition of the resolution policies. Notice that while we automatically detect conflict as an unsatisfiable sets of rules, we can also define additional conflicts by adding rules to the specification that is given as input to the SMT-solver.

4 BUILDINGRULES DESIGN

The main goal of BuildingRules is to provide the building occupants with a simple, scalable, and intuitive framework to allow them to express their preferences regarding the behavior of the building itself, thus making it possible for them to customize their working environment according to their needs. The policies defined by the BuildingRules framework are then exploited to actually control the building subsystems, such as HVAC, more effectively. In this section, we describe the different aspects of the BuildingRules framework.

4.1 Rules

As mentioned earlier, similarly to prior work (Dey et al. 2006; IFTTT nd; Ur et al. 2014), we use the trigger-action paradigm to allow users to specify rules in the following format:

$$if\ (something\ happens)\ then\ (do\ something).$$

The "if" part of the rule is called *trigger* while the "then" part is called *action*. According to this paradigm, a user can specify an action to be performed when certain event conditions are met, and the combination is a *rule*. For example, "if it is cloudy then turn on lights," where "cloudy weather" is the trigger and "turn on lights" is the action. This paradigm allows the users to express rules using a constrained version of natural language. The constraints are given by limiting both the structure of the rules as explained above and the set of pre-defined triggers and actions (the list of currently available triggers and actions is represented in Table 1).

BuildingRules is designed to manage two entities of buildings: rooms and groups of rooms. Each room is owned by one or more occupants, and the rules are specified at the room level. A room represents a physical space—an office, a conference room, a lobby, or a kitchen. Occupants are assigned to these rooms by the building manager, and occupants can customize the behavior of their room by adding new rules. Rooms with similar characteristics can be grouped and controlled

using the same set of rules. In our implementation, we chose the room as the smallest controllable element, even though the representation can be easily extended to employ a more fine-grained solution, thus considering elements such as cubicles or desk spaces.

Each rule is assigned to one of several predefined *categories* (third column in Table 1), based on what they control. For example, the two rules "if it is rainy then turn on the light" and "if it is a holiday then turn off the light" are in the same "*Light*" category, while the rule "if it is rainy then close the windows" is in the "*Window*" category. *NO_RULE* is a special trigger available for the building administrators (Rule 7 in Table 1). This trigger is always set to *True* and is used for setting the default conditions of the building. Occupants can override these default rules with more specific rules. BuildingRules also supports external applications through virtual triggers that is controlled via RESTful APIs (Rule 9 in Table 1).

Even though there are many policy languages, such as PDL (Chomicki et al. 2000) and Ponder (Damianou et al. 2001), that can be used to formalize the same information in a similar way, we believe that the chosen format is more simple. Moreover, the fact that it is close to natural language considerably helps users in the overall understanding and usage of the system. This language represents only the frontend of our approach, and it corresponds to a logic language that is used as a backend representation, as explained in the next section. This separation allows us to have, on one hand, a user-friendly language for the users and, on the other, a formal representation that can be used for conflict resolution. Notice that the proposed conflict resolution approach described in the next section could be applied also on different input formats, with the only constraint of employing a language that can be encoded in the Z3 input language.

4.2 Conflict Resolution

Since users can express their own rules for rooms, some of which are shared by multiple users, conflicts can arise. We define two rules as conflicting when they are simultaneously active and specify actions that cannot be satisfied at the same time. If these conflicts are not resolved properly, then damage of equipment can occur or user comfort can be compromised. As an example, let us consider two users that independently specify the rules: "if time is between 9am and 6pm then turn the HVAC on" and "if time is between 5pm and 8am then turn the HVAC off." Between 5pm and 6pm, the system would be in an inconsistent state. This may cause discomfort to the occupants and could damage HVAC damper if not actuated properly (e.g., ON-OFF loops).

To identify the conflicts among rules, we formalize them as propositional formulae and analyze them by using the SMT Solver Z3 (De Moura and Bjørner 2008), which is a very mature tool that generally allows to obtain better results than ad hoc solutions.

A rule is composed of two parts: a (conjunction of) trigger(s) and an action. Before adding a rule, it is verified against the set of rules already active in the room. If triggers that can be true at the same time require actions that cannot be executed at the same time because the conjunction of the two is unsatisfiable, then a conflict is detected. Formally, we represent each rule as a propositional formula composed by an implication (the trigger implies the action) that is satisfied if the trigger is not satisfied or if both the condition and the action are satisfied. In this context, the action is considered as a proposition that is true if the action can be executed and false otherwise. The new rule together with the existing ones are seen as a specification and automatically verified to check their satisfiability. If the specification is satisfiable, then the rules are not in conflict with each other. Otherwise, two or more rules are in conflict. If a user tries to insert a conflicting rule, then the list of the conflicting rules is displayed to the user so that the conflict can be solved through priorities. The fact that our conflict detection system is run every time a new rule is added guarantees that the proposed approach prevents building to be in unsafe situation. Indeed, the system always detects conflict situations and requires a solution from the user before a new rule is added to the system.

We formalize the rules as propositional formulae compliant with the following grammar:

```
   rule    ::=   trigger ⇒ action
trigger    ::=   sTrig | sTrig ∧ trigger
 action    ::=   bAct | ¬bAct | iAct∈[n,m]
  sTrig    ::=   bTrig | ¬bTrig | iTrig∈[n,m]
```

A `rule` is an implication, where the `action` and `trigger` have a fixed structure. The `trigger` is a conjunction of conditions `sTrig` that are built on the triggers represented in Table 1 (Rows 1–9). When the `trigger` is Boolean (Rows 1, 4, 5, 7, and 9), the condition is satisfied when the data are true (`bTrig`) or false (`¬bTrig`). When the `trigger` is an integer (Rows 2, 3, 6, and 8), the condition is satisfied when value is in the specified interval `[n, m]`, where `n≤m` and they are both specified according to the data domain. A similar method is used for both Boolean (Rows 10 and 11 and 13–15) and integer (Rows 12) `action` values. We do not allow the use of disjunction in action or trigger or use of conjunction in the actions. The conjunction in the action (and the disjunction in triggers) is equivalent to specifying multiple rules with the same trigger (action) and different actions (triggers).

Note that the disjunction in actions introduces non-deterministic rules, meaning that there is a choice in how the actions can be performed. Currently, we require the user to completely specify the action for a rule using priority.

Since the rules correspond to a subset of propositional logic, they can be analyzed by encoding them in the language of the Z3 SMT Solver (De Moura and Bjørner 2008). In our implementation, the translation of the rules into the Z3 language is straightforward, since we have chosen to adopt variable types (integer and boolean) that are natively supported by Z3. Note that formulae must be expressed in the prefix form adopted by Z3. For example, the rule "if someone is in the room then turn on light" is represented as

```
(assert (=> inRoom lightOn))
```

and the rule "if someone is in the room then set temperature between 68F and 72F" is represented as

```
(assert (=> inRoom (and (<= 68 temp)
                        (<= temp 72)))).
```

We complete the Z3 model with a set of assertions that specify the characteristics of the integer data (e.g., `time` is between 0 and 24) and the relationship among data (e.g., if it is sunny, then it cannot be rainy). We then verify the model to check for possible conflicts. The model is verified multiple times by asserting the trigger of each rule in the same category. Thus, we can identify the conflicts related to the same trigger or related triggers. In BuildingRules, the rule verification is performed as soon as a new rule is inserted. When a user inserts a rule that conflicts with the existing ones, a notification is raised, and the user is asked to modify the rule. The user can also modify the existing rules if he/she has the right access levels or find a compromise by identifying a tradeoff with the other users. We translate the rules from human readable syntax to the Z3 syntax using a pre-defined look up table (see Table 1).

We chose Z3 to detect conflicts as it is efficient and reusable. Although the satisfiability problem is computationally expensive, we ensure low latency as Z3 solves this problem efficiently. The SMT solver, in fact, only requires the addition of transformation rules to create a new model but does not require re-evaluation of the algorithm, since it is computed efficiently by Z3. Further, the input language of our SMT solver is generic, and it would be easy to switch to different SMT solvers, such as Yices (Dutertre and De Moura 2006).

4.2.1 Runtime Conflicts. Some conflicts cannot be detected at design time using the SMT solver, because the involved rules work on different triggers. Consider the following example: (1) "if nobody is in the room then turn off the light" and (2) "if it is between 6pm and 8pm then turn on the light."

Using Z3, we would run the verification twice: once by asserting nobody is in the room and again with the time interval 6pm to 8pm. Both the runs are satisfiable, as it cannot be known *a priori* if the triggers will conflict in time (see Table 1). We cannot identify the conflict that arises when the room is empty between 6pm and 8pm. In this case, the light will be both on and off at the same time. Note that supporting rules that operate on the same variable (and, thus, can generate conflict) is totally natural, as the user may want to express a complex policy on a variable by combining rules that depend on different triggers. For example, the user may want a rule that generally turns on the light between 6pm and 8pm but not when the room is unoccupied.

To resolve these conflicts, we let the user assign a priority to each rule. If the desire of the user is to set a policy like "generally I want this behavior but not when this event happens," then the user sets a lower priority to the general rule and a higher priority to specific rule. The priority number is used to order the rules by importance and dynamically resolve conflicts during the actuation phase in an efficient way. Moreover, it is simple to explain this concept to the occupants as they only need to know the following: "give the more important rules a higher priority."

At the moment, BuildingRules does not supper any other mechanism to deal with conflicts and any other form of enriched semantics. The constrained natural language provided to the users is interpreted as crisp logic, and conflicts correspond to an unsatisfiable set of rules. These situations are solved only using priorities assigned by the users. However, the proposed framework is easily extensible. For example, using the same structure with natural language as frontend and logic as backend, the proposed framework can be extended to support fuzzy semantics, as done in Pasquale et al. (2016) in the context of requirements engineering.

```
triggeredRules  = []
for room in getBuildingRooms(building):
    for rule in getAllRoomRules(room):
        if isTriggered(rule):
            triggeredRules.append(rule)
triggeredRules = orderByPriority(triggeredRules)

alreadyAppliedCategories = []
for rule in triggeredRules:
    ruleCategory = getRuleCategory(rule)
    if ruleCategory not in alreadyAppliedCategories:
        actuate(rule)
        alreadyAppliedCategories.append(ruleCategory)
```

Listing 1. Rules selection pseudo-code.

Once a valid ruleset is saved and the desired priority is assigned to each rule, it is possible to apply those rules to the building. Obviously, in a specific instant, only some rules will be considered (the ones that are triggered and with the highest priority). Listing 1 shows the pseudo-code of the algorithm we designed for rule selection. For each rule in a room, we check if the rule is currently triggered. If the rule is triggered, then it is copied into a temporary list (line 8). For every rule category, the rules with the highest priority for that category that do not generate conflicts are chosen for actuation (line 10).

4.2.2 User Feedback. The output of the Z3 verification phase only specifies that a new rule is conflicting with one of the existing ones. Then, we need to show to the users the rules that are

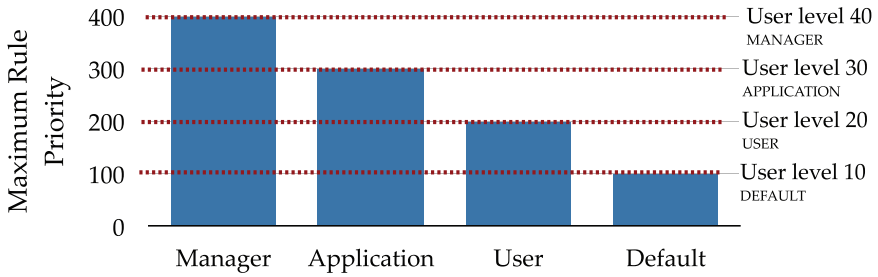

Fig. 1. User level and rule priority relation.

conflicting, so that they can either edit the existing rules or revise the new one. We cannot easily provide the list of conflicting rules, as Z3 only detects whether there is a conflict or not. A possible solution is to create a new SMT problem for all possible pairs of rules. However, this approach does not scale well with the number of rules: If n is the cardinality of the ruleset, then we would have in fact to solve n^2 SMT problems. We opted for a less precise but more efficient and feasible solution: When a conflicting rule is detected, we inform the user that the possible conflicting rules are the rules that may be logically conflicting, i.e., all the rules with the same trigger category and same action category. Suppose we have a conflicting rule, "if it is cloudy then set temperature between 70F and 75F"; then we report that rules in the weather category and the HVAC category can be statically conflicting. This strategy works in practice as there are generally few rules that have both the same action and rule category, and the user can easily pick the specific one that is conflicting.

4.3 Users

In a typical commercial building, there are different types of users who may express automation policies. A department chair and the building manager can have overriding control, a lab manager can control his or her specific lab, while individual users can control their own spaces. In BuildingRules, we need to express this hierarchy, since it affects how we resolve conflicts, by assigning *privilege levels* to each user.

Figure 1 provides an example with four categories of users—building managers and standard users, who are suited to represent the actual occupants of the building, and some special users—*applications* and *default* status. *Applications* are external softwares that interact with BuildingRules using RESTful APIs, while *default* represents agents controlling the default status of the building (when users do not specify any rule).

User levels are used in two ways. First, it ensures that low level users cannot edit or delete the rules specified by users with higher level. For example, for a room R_1 shared by two users (U_1 and U_2) (level of U_1 > level of U_2), both users are allowed to enter rules into R_1; U_1 can edit or delete rules specified by U_2 but not vice versa. Thus, a higher-level user, such as the building manager, can enforce entire building policies by creating rules with a priority higher than the standard user. The levels restrict the maximum priority a user can specify for a rule. A higher level user can assign a higher priority to a rule, giving it preference in case of runtime conflicts, thereby overriding rules expressed by a lower-level user.

4.4 Groups and Group Conflicts

In BuildingRules, we provide the facility manager with the possibility of creating groups of rooms to reuse rules across them. Let us suppose an administrator wants to turn off all the building lights

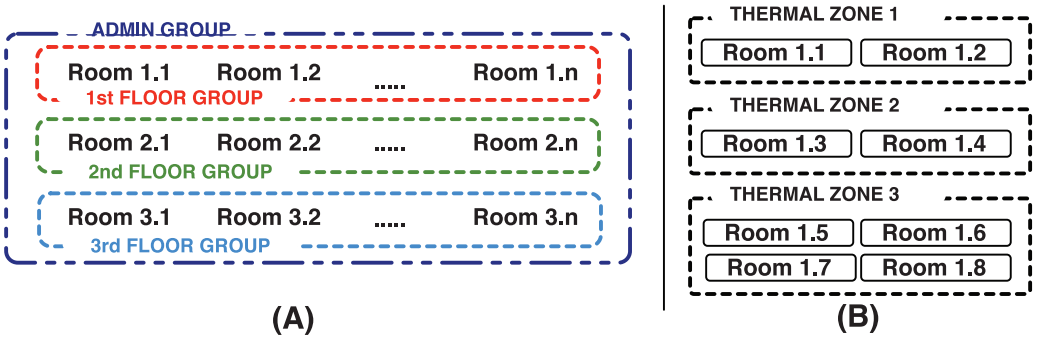

Fig. 2. (A) Example building groups. (B) Example building thermal zones distribution.

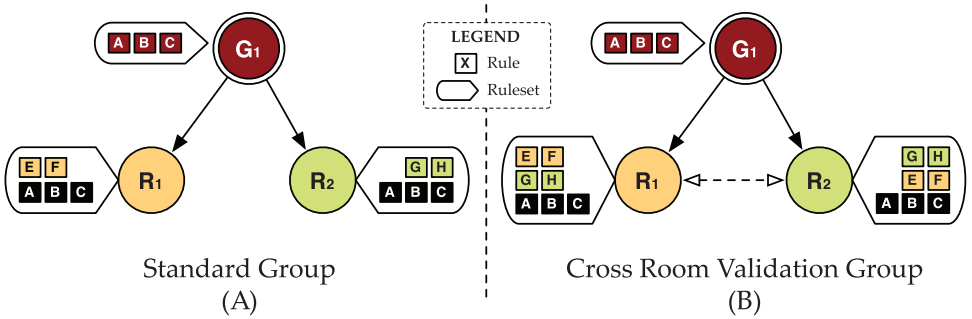

Fig. 3. Representation of the two different kinds of supported groups.

at night. Without the group feature, she would have to insert the rule "if time is between 10pm and 7am then turn off the lights" in every room. Using groups, he or she needs to create the rule just once by specifying it for a group with all the rooms in the building. Figure 2(A) shows an example grouping of rooms on a per-floor basis. In this example, if R_i is a generic room belonging to a group G, then the rules specified at the group level are inherited by all the R_i rooms.

We support another class of groups in BuildingRules to incorporate the physical characteristics of commercial buildings. HVAC systems and lighting systems often divide the building into zones of operation and can only be controlled at zone-level granularity (Balaji et al. 2013b; Krioukov and Culler 2012). Figure 2(B) shows an example of HVAC thermal zone in a building. As a result, if two rooms (R_1 and R_2) belong to the same thermal zone, then the HVAC rules specified for R_1 must also be propagated to R_2. We call such groups a *Cross Room Validation Group* (CRVG), and they are specified for action categories that need to follow this property. The behavior of a CRVG is depicted in Figure 3. In a CRVG, all the rules expressed (for specified actions) in one room within the group are propagated to the other rooms.

The conflict checking algorithm needs to take care of which groups a room belongs to. For a room not belonging to any group, the rule set is composed of the rules saved for the room. If the room belongs to one or more standard groups, then the rule set to be checked is composed of the rules saved in the considered room plus the union of all the rules saved in these groups. If the room belongs to a *Cross Room Validation Group*, then the rule set of the room is the union of all the rule set (for specified actions in CRVG) of the rooms belonging to that group. Listing 2 presents the

```
def getAllGroupRules(g):
    groupRuleSet = getGroupRules(g)
    if isCrossRoomValidation(g):
        for r in getGroupRooms(g):
            groupRuleSet.extend(getRoomRules(r))
    return groupRuleSet

def getAllRoomRules(r):
    ruleSet = getRoomRules(r)
    groups = getRoomGroups(g)
    for g in groups:
        ruleSet.extend(getAllGroupRules(g))
    return ruleSet
```

Listing 2. Ruleset generation pseudo-code.

pseudo-code that illustrates all the possible cases to generate rule sets for performing static rule verification.

5 IMPLEMENTATION

We have designed BuildingRules as a RESTful HTTP/JSON web service, with a *frontend* for the user interface, and a *backend* that communicates with the BMS, stores information about rules, runs the conflict resolution algorithm, and provides RESTful APIs for native mobile applications or building management applications. Figure 4 shows the software architecture of the system. We have implemented BuildingRules in Python 2.7 using the Flask framework (Flask, Web Development One Drop at a Time nda).

The *backend* design follows the Model-View-Controller (MVC) architecture. The REST interface enables communication with the *frontend*, and the *controller* implements the application logic. The controller stores the data to manage buildings, rooms, groups, users, and rules by using the *model* that works as a *database abstraction layer*. The controller also validates building rules using the Z3 SMT solver. Finally, the controller gathers the needed data about weather, building systems, and date-time status through a standardized *driver* interface. The drivers allow the controller to read building sensors values and send actuation signals to the building to apply the triggered rules. The driver interface allows BuildingRules to support a variety of web service BMSs like BuildingDepot (Agarwal et al. 2012) and conform to standards such as oBIX (Ehrlich and Considine 2006).

Using REST APIs, the frontend facilitates tasks such as registering users, adding triggers and actions, and specifying rules for individuals rooms or groups of rooms. On top of this API, applications can be implemented that can automatically insert rules. For example, a *Demand Response* application (Amin and Wollenberg 2005) can inject rules to reduce energy use across all rooms when a pre-registered trigger condition is met. Or a *Calendar Manager* that can insert rules like turn ON/OFF the projector or modify temperature set points based on room schedules.

Drivers in BuildingRules provide necessary abstractions between the core system and low-level sensors. These drivers act as a gateway to different sensor protocols, providing a standard naming convention across devices from different vendors and allowing us to deploy BuildingRules in different buildings with minimum effort. Drivers are of two types: *TriggerDrivers* and *ActionDrivers*. A *TriggerDriver* takes as input a sensor source (e.g., a temperature sensor in a room) and a condition to verify (e.g., the temperature is set between 70F and 75F). When the above condition is met, it provides a notification through the *eventTriggered* method. An *ActionDriver* takes a target actuator (e.g., an HVAC control system) and the actual value (e.g., set temperature to 70F) as input

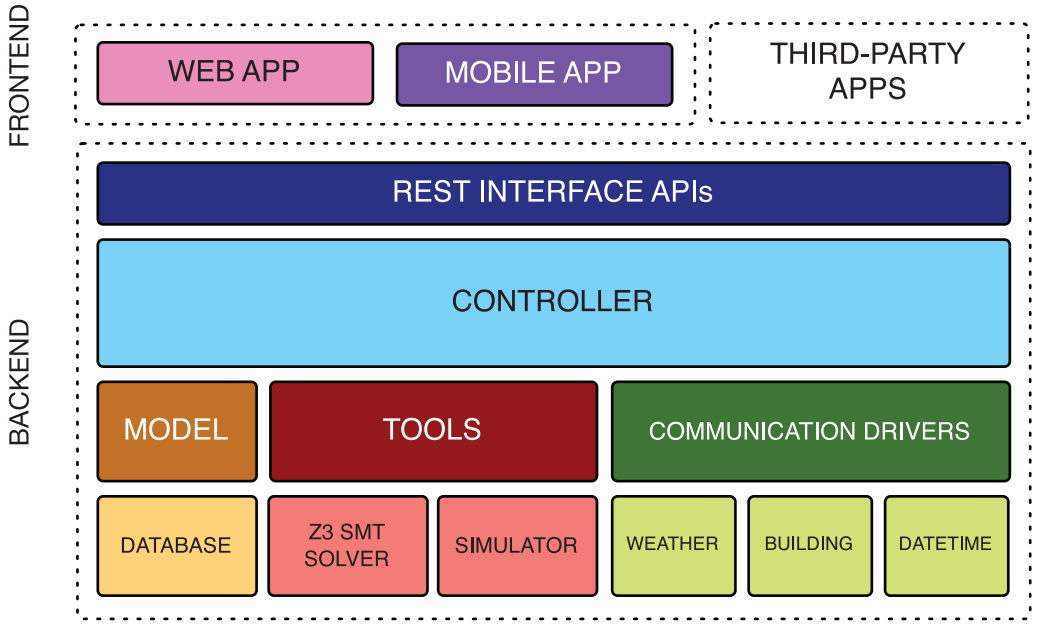

Fig. 4. BuildingRules System architecture.

and uses an *actuate* method to execute a rule action. In our current implementation, four *TriggerDrivers* and one *ActionDriver* are available. The four *TriggerDrivers* are for weather, date/time, external applications, and room-level sensors. We have implemented a single generic room level *ActionDriver* that can be used for actuation such as changing the temperature in the room.

The user interface (UI) is composed of a main page where the users can navigate among the rules. For each room three different visualization tools are available: a *rule editor* tab, a *room behavior* tab, and a *rule navigator* table The *rule editor* tab (Figure 5) presents the list of the inserted rules ordered by priority; through this interface, the user can add, delete, modify, enable, and disable the rules. As the number of rules in a room increases, it becomes harder to understand the impact of a list of rules. To solve this problem, we implemented a filter by rule category. The *room behavior* tab shows the expected behavior of the room with the expressed rules. Finally, the *rule navigator* tab visualizes rules in a matrix where each row is an action category.

A *simulator* is also available to predict the behavior of the rooms with the specified rules. It is a time-driven framework where, for each time step, the simulator reads the specified the environment conditions (the room temperature, the weather condition, the occupancy status, etc.), checks for rules that are triggered, and decides the actions based on priority. The actions are not executed but are written to a log file. The log file is converted to a timeline that represents the behavior of the room.

6 RESULTS

To evaluate BuildingRules, we first examined the rules expressed by users in a preliminary survey, looking for conflicts and measuring the latency of our conflict detection. Next we ran a larger user study with 23 users interacting with BuildingRules for a week. Using this dataset, we analyzed the rules expressed, the conflicts detected, and, finally, how our system can affect the office environment.

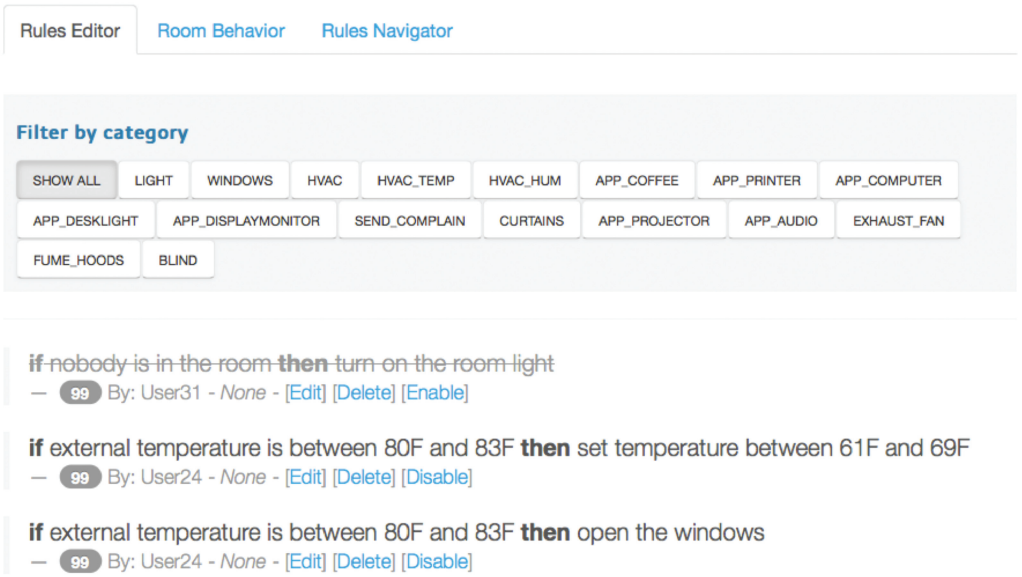

Fig. 5. Rule Editor graphical user interface.

6.1 Preliminary Survey

We conducted a preliminary survey to get an understanding of the type of rules that would be generated in an office setting and how these rules could conflict with each other. The participants were asked to create trigger-action rules on a web interface using specified triggers and actions. The trigger set included {*occupancy, temperature, time, weather*}, and the action set included {*lights, heating, cooling, window, curtains, coffee machine, microwave*}.

We received a total of 72 valid (and 2 invalid) replies with a minimum of 2 rules per participant and a total of 284 rules. The most popular trigger was *occupancy*, with 141 rules, and the most popular action was *lights*, with 100 rules. To analyze the potential conflicts between the rules, we assigned the participants randomly to shared offices (for a total of 3,069 offices combinations) and ran our conflict detection algorithm. The participants were assigned to offices with capacity of 1 to 30 occupants. We detected conflicts 99% of the time, and there were duplicate rules in 96% of these virtual offices. The conflicts are expected to be higher than what would be observed in a BuildingRules installation as the occupants cannot view the rules expressed by others in the same room. However, a significant number of rules would still conflict both statically (at rule specification time) and at runtime due to the varying preferences of the occupants as we will show in the next section. Figure 6 shows the trends of increase in static and runtime conflicts that occur as more occupants share a single office space.

6.2 Conflict Resolution Latency

Our conflict resolution module checks for conflicts on a per-room basis. Each of these checks can be run in parallel, thus making BuildingRules scalable to a large number of rooms in a building. A transaction mechanism is required for handling race conditions when users simultaneously insert rules in the same room or in rooms that belong to the same *cross room validation group*.

As BuildingRules is an interactive web application, the conflict resolution latency needs to be tolerable to the users. We collected the rules obtained from the preliminary survey and measured

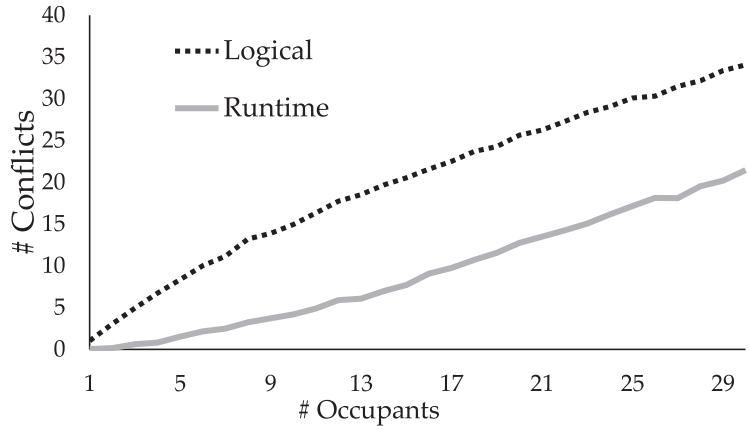

Fig. 6. Preliminary survey: Number of logical and runtime conflicts detected as participants share office spaces.

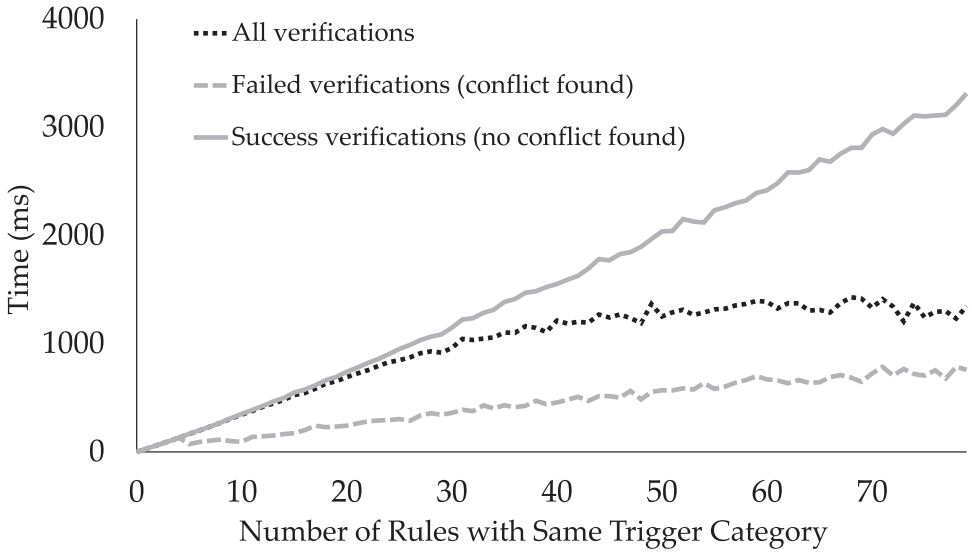

Fig. 7. Conflict checking time when the inserted or modified rule. Successful inserts are slower as the rule has been verified against all the rules with the same trigger category. Average detection time is 102ms for a room with 100 rules.

the latency for resolving conflicts as the number of rules in a room increases. Figure 7 shows the latency of conflict resolution when the inserted rules does not conflict with any of the existing rules. Note that if the rule were conflicting, then the latency would decrease as Z3 would return as soon as a conflict is found. Further, the conflict checking only occurs for rules that are in the same rule category, i.e., related actions and triggers. The maximum ratio of the number of rules in the same category to the total number of rules in a room is 6% from the rules collected in our virtual building user study (see the next section). Thus, we estimate that for a room with 100 rules in place, the average time for conflict checking is around 100ms, and the worst case time is around 250ms.

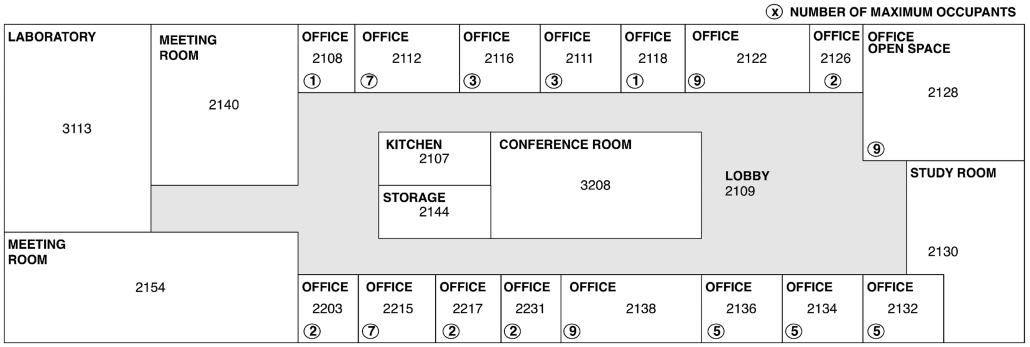

Fig. 8. Virtual office plan.

6.3 Virtual Building Study

To evaluate BuildingRules in a more realistic setting, we created a virtual office environment as depicted in Figure 8. We chose a virtual setting, because it is easier to study the rules made using different kinds of sensors that we cannot deploy in a real building.

We have designed the building plan to be representative of a typical office and incorporated different types of rooms, such as conference rooms, a research laboratory, a kitchen, storage, and offices with varying capacity. Each participant was assigned to a random set of rooms, for example, an office space, kitchen, and meeting room. The participants were told to use BuildingRules for at least 10 minutes and complete a set of actions, i.e., add, remove, edit rules, each day (10 actions the first day, then decreasing each day, with an average of 5 actions per day). A final survey was taken at the end of the week to understand the usability of the system. Each user was required to have at least one month of office experience for participation. We had a total of 23 users spread over 17 days. Notice that there is no bias in the choice of the population; the requirements in the selection were imposed to test the system with a population chosen according to the target of BuildingRules, i.e., offices.

We obtained a total of 636 rules from this study, with an average of 15 rules per room and 16 rules per user. Figure 9 shows the distribution of these rules across the various triggers and actions. The default status rules were inserted for scheduling of lights and HVAC system when no other rules were specified for a room. The most popular trigger was the day of week, followed by occupancy. The most popular action was lights, followed by plug loads (computer, desk light, monitor and printer). These results match with the ones obtained in the study presented in the previous section.

Figure 10 shows the addition of rules across the period of the study. Understandably, the number of logical conflicts have decreased compared to the preliminary survey as the users consider which rules to add after looking at the current rules in effect.

We show the effect of the combination of rules on a room using BuildingRules simulator in the UI. The simulator results are based on real weather data and temperature readings we obtain from the BMS, and we simulate occupancy using data collected in prior work (Balaji et al. 2013b). The study was designed in such a way that participants were forced to create or modify rules, so that we can analyze effect of the combination of these rules in the virtual office. As a result, some of the rules inserted in the rooms were very similar but not in conflict. For example, in one of the rooms there were three rules to open windows with three different temperature ranges. In a real deployment, users would probably use one rule to cover all the three temperature ranges.

	Date	Day	Default Statatus	External Temperature	Occupancy	Room Temperature	Time	Weather
Audio	0	9	0	0	6	0	3	1
Coffee Macchine	2	2	0	0	2	0	3	0
Computer	1	43	0	0	5	0	7	1
Desk Light	1	41	0	0	7	0	5	3
Display Monitor	2	39	0	0	9	0	1	1
Printer	1	40	0	0	5	0	5	0
Projector	0	10	0	0	10	0	6	0
Blind	3	8	0	0	6	0	7	24
Exhaust Fan	1	2	0	0	1	1	2	0
Fume Hoods	0	0	0	0	2	1	0	0
HVAC	12	18	0	4	23	13	3	6
Room Humidity	0	1	24	2	7	3	2	4
Room Temp.	3	4	24	6	9	6	1	1
LIGHT	5	47	5	0	38	0	18	24
Send Complain	0	1	0	2	8	19	5	0
Windows	4	12	0	26	22	9	4	29

Fig. 9. Rule usage frequency.

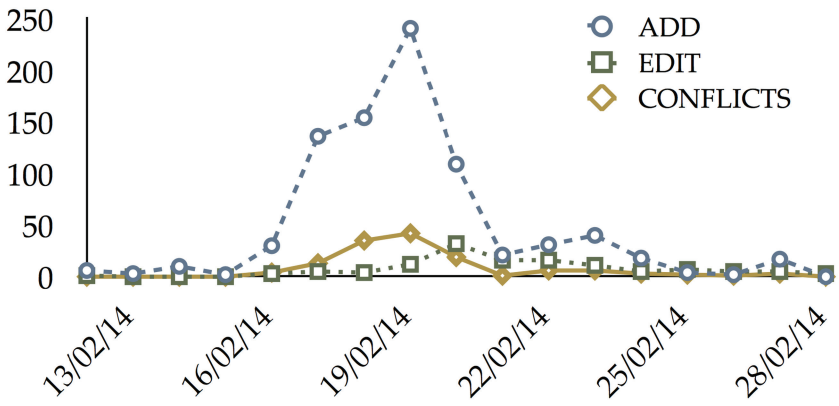

Fig. 10. User requests.

Based on the feedback from the participants of this user study, we created a second version of the study to improve the user experience and to help the users in the creation of realistic rules. We improved our building simulator to incorporate the effects of actuation. For example, room temperature changes linearly with time if HVAC is ON until it reaches its set-point, and temperature would change linearly as a function of difference between indoor and outdoor temperature when the HVAC is off. We also added power values to different appliances, with fixed values when they are ON. These effects were a representation of what the user might expect in a real setting . The user was assigned a happiness index based on comfort, i.e., an occupied room is within temperature/humidity bounds, and lights are ON when it was dark outdoors. The building manager was assigned happiness index based on the power consumption of the building.

	Date	External Temperature	Occupancy	Room Temperature	Time	Weather
Audio	0	0	2	0	2	0
Coffee Macchine	0	0	3	0	4	0
Computer	1	0	3	0	6	0
Desk Light	0	0	5	0	2	1
Display Monitor	0	0	5	0	2	0
Printer	0	0	4	0	2	0
Projector	0	0	4	0	2	0
Blind	0	0	1	0	0	1
Exhaust Fan	0	0	0	0	0	0
Fume Hoods	0	0	1	0	2	1
HVAC	0	0	2	1	1	0
Room Humidity	0	1	10	0	4	8
Room Temp.	1	3	4	19	2	3
LIGHT	0	0	23	0	14	24
Send Complain	0	0	1	1	0	0
Windows	0	9	4	4	2	13

Fig. 11. Rule usage frequency for the second experiment.

The experiment was conducted over three days, with 13 users. The users were given a short video tutorial and restricted to create 6 rules on the first day, 4 rules on the second, and 2 rules on the final day. Each rule could be edited only twice a day. The goal of the participants was to boost the happiness index of both the occupant and building manager, and the user with the highest score was awarded $20 Amazon Gift Card. We assigned two building managers for support to users, as well as monitoring rules being created; 179 rules were created in this experiment. Their composition is shown in Figure 11. The general behavior of the users were similar in both experiments: The majority of the rules were about occupancy, room temperature, time, and weather with respect to room temperature and humidity, lights, and windows.[3]

6.4 Usability Study

We asked our participants of virtual building study to fill a usability survey to better understand their needs and to evaluate BuildingRules from an occupant's view point. Table 2 shows the results of our survey. The participants liked the idea behind BuildingRules (7.0), thought that it would be useful to have such a system in their office (7.6), and liked the system overall (7.0). However, as the number of rules in a room increased, our UI was not adequate to give an overview of the rules in place. Users found it difficult to create (5.4), edit (5.8), and understand (5.0) the rules.

In the second version of BuildingRules, users assigned higher scores than the baseline case study. Thus, improving the user interface, a short preliminary tutorial, and giving a runtime support to the users improved our system usability.

[3]More details about the experiments we have conducted are available on our technical report (Flask, Web Development One Drop at a Time ndb).

Table 2. Results of Usability Survey Note: Scores are of 10.

Survey Question	**User Study 1**	**User Study 2**
Overall impression score	7.0	7.8
System usability	6.0	7.6
Would BuildingRules be useful in your office?	7.6	8.2
How difficult was it to insert new rules?	5.4	8.3
How difficult was it to edit existing rules?	5.8	5.9
Do you like the philosophy behind the system?	7.0	8.8
How easy was it to resolve conflicts within the rules?	5.8	5
Was it easy to understand how the combination of rules will affect the office?	5.0	7.2
How useful was the *rule editor* to identify individual rules?	x	7.2
Was it possible to grasp all rules using *rule navigator*?	x	7.1

Notice that the current implementation of BuildingRules is still an experimental tool, and further work on usability is required. However, even with a simple interface, the users liked the overall system and this demonstrates that BuildingRules is very promising and is considered an effective tool to help users in managing their offices.

7 CONCLUSION

We have presented the design and the implementation of BuildingRules, a system that enables expression of personalized automation rules in commercial buildings using trigger-action programming paradigm, which can then be integrated with existing BMSs to actuate buildings. We show that when multiple users express different policies for the same physical space, conflicts can occur. To resolve these conflicts, we have implemented two mechanisms in BuildingRules. First, we avoid logical conficts by detecting them as rules that are inserted using the Z3 SMT solver. Second, BuildingRules resolves runtime conflicts using a priority assigned to individual rules. We show that our conflict detection algorithm is parallelizable and scales to large commercial buildings, such that the latency is low enough to support the interactive web application UI of BuildingRules. To simplify rules expression and expose the physical constraints imposed by building systems, BuildingRules provides a grouping mechanism. To incorporate the hierarchy commonly seen in commercial buildings, BuildingRules provide mechanisms for access control and different levels of privileges for rule expression. The final component of BuildingRules is an intuitive web interface for building occupants to express their rules. Using this UI, we evaluated the use of BuildingRules in a virtual office building with 23 users across 17 days and found out that BuildingRules allows the definition of a wide and interesting set of rules and that is able to automatically resolve conflicts among them with a very limited timing overhead.

REFERENCES

Yuvraj Agarwal, Bharathan Balaji, Seemanta Dutta, Rajesh K. Gupta, and Thomas Weng. 2011. Duty-cycling buildings aggressively: The next frontier in HVAC control. In *Proceedings of the 2011 10th International Conference on Information Processing in Sensor Networks (IPSN'11)*. IEEE, 246–257.

Yuvraj Agarwal, Rajesh Gupta, Daisuke Komaki, and Thomas Weng. 2012. Buildingdepot: An extensible and distributed architecture for building data storage, access and sharing. In *Proceedings of the 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 64–71.

OpenADR Alliance®. OpenADR 2.0 Profile Specification - A Profile. Document Number: 20110712-1. http://savannah.gnu.org/task/download.php?file_id=27590.

S. Massoud Amin and Bruce F. Wollenberg. 2005. Toward a smart grid: Power delivery for the 21st century. *IEEE Power Energ. Mag.* 3, 5 (2005), 34–41.

Pandarasamy Arjunan, Nipun Batra, Haksoo Choi, Amarjeet Singh, Pushpendra Singh, and Mani B. Srivastava. 2012. SensorAct: A privacy and security aware federated middleware for building management. In *Proceedings of the 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 80–87.

Bharathan Balaji, Hidetoshi Teraoka, Rajesh Gupta, and Yuvraj Agarwal. 2013a. ZonePAC: Zonal power estimation and control via HVAC metering and occupant feedback. In *Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings*. ACM, 1–8.

Bharathan Balaji, Jian Xu, Anthony Nwokafor, Rajesh Gupta, and Yuvraj Agarwal. 2013b. Sentinel: Occupancy based HVAC actuation using existing WiFi infrastructure within commercial buildings. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 17.

Victoria Bellotti and Keith Edwards. 2001. Intelligibility and accountability: Human considerations in context-aware systems. *Hum.–Comput. Interact.* 16, 2–4 (2001), 193–212.

Alex Beltran, Varick L. Erickson, and Alberto E. Cerpa. 2013. ThermoSense: Occupancy thermal based sensing for HVAC control. In *Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings*. ACM, 1–8.

Steven T. Bushby. 1997. BACnet^{TM}: A standard communication infrastructure for intelligent buildings. *Autom. Construct.* 6, 5 (1997), 529–540.

Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. 2003. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Trans. Softw. Eng.* 29, 10 (2003), 929–945.

Jan Chomicki, Jorge Lobo, and Shamim A. Naqvi. 2000. A logic programming approach to conflict resolution in policy management. In *KR*, Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman (Eds.). Morgan Kaufmann, 121–132.

Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. 2001. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY'01)*. 18–38.

Stephen Dawson-Haggerty, Andrew Krioukov, Jay Taneja, Sagar Karandikar, Gabe Fierro, Nikita Kitaev, and David Culler. 2013. BOSS: Building operating system services. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

Luigi De Russis and Fulvio Corno. 2015. HomeRules: A tangible end-user programming interface for smart homes. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2109–2114.

Samuel DeBruin, Bradford Campbell, and Prabal Dutta. 2013. Monjolo: An energy-harvesting energy meter architecture. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 18.

Anind K. Dey, Gregory D. Abowd, and Daniel Salber. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.* 16, 2 (2001), 97–166.

Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive prototyping of context-aware applications. In *Pervasive Computing*. Springer, 254–271.

Bruno Dutertre and Leonardo De Moura. 2006. The Yices SMT solver. http://yices.csl.sri.com/papers/tool-paper.pdf.

Paul Ehrlich and Toby Considine. 2006. Open building information exchange (oBIX) version 1.0. OASIS Committee specification, December 2006.

Varick L. Erickson and Alberto E. Cerpa. 2012. Thermovote: Participatory sensing for efficient building hvac conditioning. In *Proceedings of the 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 9–16.

Flask, Web Development One Drop at a Time. ndb. BuildingRules Technical Report. Retrieved from https://csetechrep.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2014-1008.

Flask, Web Development One Drop at a Time. nda. Flask Web Microframework. Retrieved from http://flask.pocoo.org/.

Ernest Friedman-Hill. 2003. *JESS in Action*. Manning, Greenwich, CT.

Saul Greenberg. 2001. Context as a dynamic construct. *Hum.-Comput. Interact.* 16, 2 (2001), 257–268.

Barry P. Haynes. 2008. The impact of office comfort on productivity. *J. Facil. Manage.* 6, 1 (2008), 37–51.

Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 215–225.

IFTTT. nd. Home Page. Retrieved from https://ifttt.com/.

Xiaofan Jiang, Stephen Dawson-Haggerty, Prabal Dutta, and David Culler. 2009. Design and implementation of a high-fidelity ac metering network. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN'09)*. IEEE, 253–264.

Johnson Controls. Building Managament. http://www.johnsoncontrols.com/buildings/building-management.

Andrew Krioukov and David Culler. 2012. Personal building controls. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*. ACM, 157–158.

Robert H. Lasseter and Paolo Paigi. 2004. Microgrid: A conceptual solution. In *Proceedings of the IEEE 35th Annual Power Electronics Specialists Conference (PESC'04)*, Vol. 6. IEEE, 4285–4290.

Abhinandan Majumdar, David H. Albonesi, and Pradip Bose. 2012. Energy-aware meeting scheduling algorithms for smart buildings. In *Proceedings of the 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 161–168.

Sarah Mennicken, Jo Vermeulen, and Elaine M. Huang. 2014. From today's augmented houses to tomorrow's smart homes: new directions for home automation research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 105–115.

Evan Mills. 2011. Building commissioning: A golden opportunity for reducing energy costs and greenhouse gas emissions in the United States. *Energ. Efficiency* 4, 2 (2011), 145–173.

Alessandro Antonio Nacci, Bharathan Balaji, Paola Spoletini, Rajesh Gupta, Yuvraj Agarwal, and Donatella Sciuto. 2015. BuildingRules: A trigger-action based system to manage complex commercial buildings. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM.

Niagara AX. nd. Home Page. Retrieved from http://www.niagaraax.com.

Insuk Park, Dongman Lee, and Soon J. Hyun. 2005. A dynamic context-conflict management scheme for group-aware ubiquitous computing environments. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, Vol. 1. IEEE, 359–364.

Liliana Pasquale, Paola Spoletini, Mazeiar Salehie, Luca Cavallaro, and Bashar Nuseibeh. 2016. Automating trade-off analysis of security requirements. *Requir. Eng.* 21, 4 (2016), 481–504.

M. F. Pichler, A. Dröscher, H. Schranzhofer, G. D. Kontes, G. I. Giannakis, E. B. Kosmatopoulos, and D. V. Rovas. 2011. Simulation-assisted building energy performance improvement using sensible control decisions. In *Proceedings of the 3rd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 61–66.

Farrokh Rahimi and Ali Ipakchi. 2010. Demand response as a market resource under the smart grid paradigm. *IEEE Trans. Smart Grid* 1, 1 (2010), 82–88.

Anand Ranganathan and Roy H. Campbell. 2003. An infrastructure for context-awareness based on first order logic. *Pers. Ubiq. Comput.* 7, 6 (2003), 353–364.

Sılvia Resendes, Paulo Carreira, and André C. Santos. 2013. Conflict detection and resolution in home and building automation systems: A literature review. *J/Amb/Intell/Hum/Comput/* 5, 5, 699–715.

Benoit Roisin, Magali Bodart, A. Deneyer, and P. Dherdt. 2008. Lighting energy savings in offices using different control systems and their real consumption. *Energ. Build.* 40, 4 (2008), 514–523.

Siemens Building Technologies. nd. Home Page. Retrieved from http://www.buildingtechnologies.siemens.com.

Timothy Sohn and Anind Dey. 2003. iCAP: An informal tool for interactive prototyping of context-aware applications. In *Proceedings of the Extended Abstracts on Human Factors in Computing Systems (CHI'03)*. ACM, 974–975.

Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd. 2004. CAMP: A magnetic poetry interface for end-user programming of capture applications for the home. In *Proceedings of the Conference on Ubiquitous Computing (UbiComp'04)*. 143–160.

Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical trigger-action programming in the smart home. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*.

Thomas Weng, Bharathan Balaji, Seemanta Dutta, Rajesh Gupta, and Yuvraj Agarwal. 2011. Managing plug-loads for demand response within buildings. In *Proceedings of the 3rd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM, 13–18.

Thomas Weng, Anthony Nwokafor, and Yuvraj Agarwal. 2013. BuildingDepot 2.0: An integrated management system for building analysis and control. In *Proceedings of the 5th ACM Workshop on Embedded Systems for Energy-Efficient Buildings*. ACM, 1–8.

Jong-bum Woo and Youn-kyung Lim. 2015. User experience in do-it-yourself-style smart homes. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 779–790.

Chang Xu and Shing-Chi Cheung. 2005. Inconsistency detection and resolution for context-aware middleware support. *ACM SIGSOFT Softw. Eng. Not.* 30, 5 (2005), 336–345.

Tao Zhang and Bernd Brügge. 2004. Empowering the user to build smart home applications. In *Proceedings of the International Conference on Smart Home and Health Telematics (ICOST'04)*. 170–176.

Received July 2016; revised July 2017; accepted January 2018