POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica



SUPPORTING MULTI-TENANT SCENARIOS IN SMART BUILDINGS: RULESET MINIMIZATION, CONFLICT RESOLUTION AND SINK STATE ANALYSIS.

Relatore: Prof. Marco D. SANTAMBROGIO
Correlatore: Alessandro A. NACCI
Vincenzo RANA

Tesi di Laurea di:
Andrea PISCITELLO
Matr. 819881

Anno Accademico 2014 / 2015

Andrea Piscitello: *Supporting multi-tenant scenarios in smart buildings: Ruleset Minimization, Conflict Resolution and Sink State Analysis.* | Tesi di Laurea Magistrale in Ingegneria Informatica, Politecnico di Milano.
© Copyright Aprile 2016.

Politecnico di Milano:
www.polimi.it

Scuola di Ingegneria Industriale e dell'Informazione:
www.ingindinf.polimi.it

Acknowledgments

This thesis is dedicated to entire my family. To my father Nicolo' and my mother Maria Grazia that supported my studies economically and, above all, emotionally since I was born. They taught me that the most important thing is how to live honestly in the world; they are the best parents that a person could desire. To my sister Claudia who represented the model of all my childhood and who taught me how to grow up. To my aunt Gabriella who has been like a second mother and who has (involuntarily) contributed to the development of my computer skills. To my grandmother Francesca who has been the first teacher of my life and that supported my off-site life with tons of *braciole*. To my uncle Luigi and aunt Sara who despite the distance in these years were very close to me with the heart.

If the family is the standard equipment and the friends are the optionals of the life, I chose the full optional package since mine are the best friends anyone could have ever had. I want to thank all of them, starting with my best friend Luca, who has developed an interest in engineering together with me and who has fought with me the adversities to our common project during High School. After all we succeeded and, despite the thousands of kilometers (miles) that are between us, our friendship is even stronger than before.

I want to thank my second family: Daniele, mate of night games and irrational fights in front of a computer and Laura my *ally* against the common enemy.

A special thanks goes to my travel companions: all the friends I made since my first day in Milan starting from Bruno, my very first friend in

Milan, who shared with me all the joys and pains of the Politecnico; Giada, who became much more than a simple university mate; Padu who has been an exceptional companion in Europe, America and soon(?) Asia (fingers crossed); Raffaele, my bungler friend who added (and then spilled to the floor) some pepper in our too serious group; Paolo (the kid) and Francesca (the senior), the best couple I have ever met and the most stressful camping mates I have ever had; Mauro (*Bussola*), my eternal uncertain friend from *Paleimmo* who has immediately became one of my best friends; Sara, my short-arm *Genovese* friend who once believed to have turned into a Milan tram; Checco, my brother of many adventures: "*izza mena me'!*" and *Angela*

I owe a big thanks to my friends from Messina: Antonella, who has became in the last years a very special friend to me; Fulvio, the incurable *minchione* whom I shared a lot of experiences with; Giorgio, companion of life and of *braciole suffocations*; the other Giorgio, the promoter affected by *balbizie*; Adriana (*Oliva*), historically my best friend and companion in Abergele; Fabrizio (*Pollozzen*) mate of professionals tennis matches and very serious injuries; Cecilia, the only female *zio* I know. I would like to thank all the other guys from the MHS crew.

I want to thank: Prof. Marco Santambrogio for having conducted me in this thesis work with his typical frankness and sincerity, and for having taught to me really important things about academia and team work; *the enterpreneur* Alessandro Nacci for his incredible support he gave to me in this thesis work; Vincenzo Rana for his precious contributions.

A. P.

Milano, Aprile 2016

To whom it may concern.

Contents

Acknowledgments iii

Abstract xvii

Sommario xix

Estratto in Lingua Italiana xxi

1 Introduction 1

1.1 Smart Buildings: Overview . . . 1

1.2 Smart Buildings: Related Works . . . 3

1.2.1 The current idea of a Smart Building . . . 3

1.2.2 Current challenges and limitations . . . 7

1.3 *BuildingRules* . . . 11

1.3.1 Overview . . . 11

1.3.2 System Components and Architecture . . . 12

1.3.3 Rules . . . 14

1.3.4 Conflicting Situations . . . 16

1.3.5 Users . . . 19

1.4 Multi-tenant programming . . . 20

1.5 Contribution and outline . . . 23

2 Ruleset Minimization 25

2.1 Introduction . . . 25

2.2 Background and Related Works . . . 26

2.2.1 Karnaugh Map . . . 27

2.2.2 Quine-McCluskey 28
2.2.3 Espresso 28
2.3 Proposed Method 31
2.3.1 Binarization 33
2.3.2 Karnaugh Map construction 33
2.3.3 Karnaugh Algorithm 34
2.3.4 Karnaugh with ON-Set only 35
2.3.5 Double Karnaugh on ON-Set and OFF-set 35
2.3.6 Exploitable Don't cares 36
2.3.7 Information Extraction/Abstraction 38
2.4 Implementation 38
2.5 Results 41
2.5.1 Experimental Set 42
2.5.2 Experimental Campaign 43
2.5.3 Results Discussion 44
2.6 Conclusions 45

3 Conflict Resolution **47**
3.1 Introduction 47
3.2 Related Works 48
3.3 Proposed Method 51
3.3.1 Priority System Reform 52
3.3.2 Satisfaction Model 53
3.4 Results 57
3.4.1 Experimental Set 58
3.4.2 Experimental Campaign 59
3.4.3 Results Discussion 60
3.5 Conclusions and Future Works 62

4 Building Behavioral Analysis **65**
4.1 Introduction 65
4.2 Related Works 66
4.3 Proposed Method 69

4.3.1 'Proper' Sink State Analysis 72
4.3.2 Semi Sink State Analysis 73
4.4 Implementation 73
4.4.1 Back-end 74
4.4.2 Frontend 78
4.5 Results 80
4.5.1 Experimental Set 81
4.5.2 Experimental Campaign 82
4.5.3 Results discussion 83
4.6 Conclusions and Future Works 84

List of Figures

1.1 Standard architecture of a smart building.[1] 4
1.2 Centralized and distributed BMS architectures[2]. 6
1.3 Smart Building stack. BuildingRules is inserted on top of BMS as an additional layer between it and user interaction one. 13
1.4 BuildingRules Architecture 14
1.5 Priority levels hierarchy. There are four main kind of *users* for BuildingRules: (Building) Manager, Applications, (Human) Users, Default (rules that are triggered when no other rules exist.) 19

2.1 Example of Karnaugh map construction starting from a Truth Table representation of a boolean logic function. Over the Karnaugh map, in different colors, the largest minterms composing the final expression are drawn. 28
2.2 Binarization process to generate binary rules starting from ordinary rules defined over intervals. The temperature attribute is split in all the possible sub-intervals generated by the defined rules. 32
2.4 Comparison between: (a) Karnaugh Algorithm *as-is*, (b) Karnaugh using ON-Set only (i.e. without taking into account *Don't Cares*); (c) Two different execution of Karnaugh algorithm for ON-Set and OFF-Set. 34
2.3 The translation from Truth table to Karnaugh Map happens as in the traditional algorithm. 34

2.5 Example of minimization that can exploit *Don't cares* due to impossible combination of the input. 37

2.6 Architecture of Back-end and Front-end with main data models and processing flow. 39

2.7 Screenshot of the Front-end webapp. As it can be noticed on the left the user (Building Manager) can choose the room to visualize the rules for. He can also Binarize and/or Minimize the given Truth Table. 41

2.8 Virtual Office Environment planimetry 42

2.9 Rule distribution with respect to triggers and actions. . . . 44

3.1 Satisfaction gaussian curve with $\mu = 22°C$ and $\sigma^2 = 3°C$. 55

3.2 Three possible intervals that can be defined with the extended formulation. (a) is an asymmetric interval that can be defined with two different (left and right) tolerance ranges. (b) is an asymmetric interval with *right tolerance* set to 0. (c) is a symmetric interval with both tolerance set to 0. 56

3.3 Two-users system with individual satisfaction curves (blue and green) and overall satisfaction curve (red). 57

3.4 NECST Lab planimetry. Red circles represent the locations of the temperature sensors and the green square is the Android device used as gateway. 59

3.5 Experimental set architecture. User preferences and feedbacks are directly stored into the Parse database. Sensor data is retrieved by an Android device which acts as a gateway. 60

3.6 User preferences models (all little gaussian curves) and overall thermal satisfaction model (big red gaussian curve). . . 61

3.7 User satisfaction models (blue curve) compared to actual user feedbacks (red dots) 63

4.1 Temperature trend estimation model suggested by Cesaraccio et al [16]. It is modeled as two sinus functions (from sunrise to sunset) and a square-root function from sunset to sunrise of the following day. 75
4.2 Occupancy model derived by [63, 72]. It shows how the probability of having people inside an office is distributed along the hours of a weekday. 77
4.3 Gauge indicator in the bottom part of the Tenant User Interface for adding a new rule. It can be noticed that values outside the '*good*' range prevent users from inserting the given rule. 79
4.4 Screenshot of the Building Manager console. Building Manager can exploit it to look for assertive, rules, useless, rules, sinks and semi-sinks inside the building rooms. 80

List of Tables

1.1 Currently supported rule triggers (T) and actions (A) categories. An example of trigger or action for each category is provided. 16
1.2 Translation to Z3 examples. 18

2.1 Results of the minimization algorithm for each room. . . . 45

4.1 Rules attributes and data sources employed for building the probabilistic method. 74
4.2 Rules attributes and dependencies used for building the probabilistic method. (✓implies a dependency between the corresponding attributes; **N.A.** shows a dependency that was not exploited due to missing data reasons; **-** implies no dependency) 78

Abstract

Nowadays, the incessant technology development in the field of sensors and actuators devices has reached such an advanced stage to have shaped many aspects of our everyday life. The interest of both academic and industry worlds in this field have even brought to the birth of new branch of the Information Technology called *Internet of Things* (IoT). The technologies and the products developed in this particular field have contributed to the birth of new human-computer interactions paradigms and to actually transform peoples' lives. In the last years, many research and industrial projects have focused on the design of complex softwares hat unlocked the concretization of the *Smart Space* vision. Smart Spaces are environments such as homes, schools, hospitals and outdoor areas that exploit the cooperation of different connected objects (e.g. sensors, actuators, devices and appliances) in order to provide services to customer and citizens to make their lives more comfortable and efficient. Even if many challenges for the creation of such environments have been already addressed, with the adoption of such new technologies, new challenges and questions arouse.

Within this thesis work we focus on the multi-tenant management inside *Smart Buildings*. Even if advanced Building Management Systems (BMS) have been proposed in the last years, enabling users to create rules to program the building behavior, a concrete support for properly handling the huge amount of users present inside a building is missing in all of them. In particular there are three main issues that have to be faced in order to make Smart Buildings satisfy modern users needs:

- A way for minimizing the ruleset keeping its size to the minimum

necessary to guarantee the correct behavior of the building is needed;

- An accurate conflict resolution strategy that can scale well on conflicts involving many users, trying to accommodate all the users needs is missing;
- A behavioral analysis regarding the correct control of the building has not been performed.

In order to address these problems novel techniques and approaches have been proposed. They have been used in three different experimental campaigns in order to show their effectiveness.

Sommario

Oggigiorno, l'incessante sviluppo tecnologico nel campo dei sensori e attuatori ha raggiunto uno stadio così avanzato da aver plasmato molti aspetti della nostra vita quotidiana. L'interesse di entrambi i mondi accademico e industriale in questo campo ha persino portato alla nascita di una nuova branca della Information Technology, chiamata *Internet of Things* (IoT). Le tecnologie e i prodotti sviluppati in questo particolare campo hanno contribuito alla nascita di nuovi paradigmi di interazione uomo-macchina e alla reale trasformazione della vita delle persone. Negli ultimi anni, molti progetti di ricerca ed industriali si sono concentrati sulla progettazione di complessi software che hanno permesso la concretizzazione della visione di *Smart Space*. Gli Smart Space, sono ambienti come case, scuole, ospedali e luoghi esterni che sfruttano la cooperazione tra oggetti interconnessi (e.g. sensori, attuatori, dispositivi ed elettrodomestici) al fine di fornire servizi a clienti e cittadini per rendere le loro vite più comfortevoli ed efficienti. Nonostante molte delle sfide legate alla creazione di tali ambienti siano già state affrontate, l'adozione di queste nuove tecnologie ne ha fatte nascere di nuove. In questo lavoro di tesi ci concentriamo sulla gestione della multi-utenza negli *Smart Buildings*. Anche se negli ultimi anni sono stati proposti Sistemi di Gestione degli Edifici (BMS) molto avanzati, che permettono agli utenti di programmare il comportamento dell'edificio attraverso delle regole, un supporto concreto per gestire adeguatamente il gran numero di utenti presenti all'interno di un edificio risulta essere mancante. In particolare ci sono tre problemi principali che devono essere affrontati affinchè gli Smart Buildings possano soddisfare efficacemente gli utenti moderni:

- Esiste un bisogno di minimizzare l'insieme di regole al fine di rendere la sua dimensione la minima necessaria a garantire la correttezza del comportamento dell'edificio;
- Manca una strategia di risoluzione dei conflitti che possa scalare bene su conflitti che coinvolgono molti utenti e che cerchi di soddisfare tutti gli utenti;
- Una analisi comportamentale dell'edificio che riguardi il corretto controllo dell'edificio stesso non è mai stata presa in considerazione.

Per la risoluzione di tali problemi sono state proposte tecniche ed approcci innovativi. Essi sono stati impiegati in tre differenti campagne sperimentali al fine di mostrarne la loro efficacia.

Estratto in Lingua Italiana

Gli *smart building* sono ambienti come appartamenti, uffici, ospedali, scuole che sono caratterizzati dalla co-operazione tra oggetti "intelligenti" (per esempio sensori, attuatori, elettrodomestici) e, dalla interazione con visitatori che possono essere sia sporadici che frequenti. Nonostante questa definizione sia molto simile a quella adoperata in questa tesi, ci sono molte altre definizioni che dipendono dai diversi contesti di utilizzo. Per evitare confusione, è quindi necessario fornire una descrizione precisa di smart building che verra utilizzata nel resto di questo documento. In questa tesi, uno smart building è definito come un edifcio che fornisce funzionalità di automazione, nella prospettiva di aumentare il comfort e la sicurezza degli abitanti, mantenendo allo stesso tempo l'efficienza energetica dell'edificio stesso.

Sotto un punto di vista teorico, uno smart building è un sistema di controllo distribuito, caratterizzato da una notevole quantità di sensori, attuatori ed elementi computazionali. Essi sono tutti integrati all'interno delle infrastrutture dell'edificio e sono strettamente connessi insieme in una rete continua ed invisibile. Tale rete di sensori e attuatori, permette di superare l'approccio convenzionale della gestione degli edifici, rendendo possibile creare politiche di gestione personalizzate che fanno uso di tali infrastrutture.

La tipica architettura di uno smart building è composta da:

- Layer Hardware
- Layer Software di Back-end
- Layer Software di Front-end.

Il **layer hardware** è composto da una rete di sensori che è adoperata per recuperare dati (per esempio: temperatura, umidità, occupazione...) in maniera distribuita e per trasmetterli al layer software. Una rete di attuatori è usata per modificare il comportamento dell'edificio agendo sulle stesse variabili di stato dei sensori (come temperatura, umidità, ecc.) o azionando elettrodestici, accendendo/spegnendo luci, monitor, finestre.
I dati dei sensori e i comandi degli attuatori sono gestiti da una porzione di software responsabile per il corretto funzionamento dell'edificio. Esso è denominato **Building Management System (BMS)** e rappresenta un ponte tra l'hardware e gli utenti.
L'interazione uomo-macchina è invece gestita da software di **front-end** che espone interfacce user-friendly sotto forma di monitor, console, applicazioni web e dispositivi mobili.

Fino ad oggi, le politiche di gestione degli edifici sono sempre state manualmente inserite da tecnici e specificate dai *building-manager*, i quali hanno sfruttato la loro esperienza e le loro conoscenze a tale scopo. Una infrastruttura come quella degli smart buildings, al contrario, permette a tutti gli abitanti dell'edificio, di interagire direttamente con essa per automattizare la procedura di definizione delle politiche, attraverso una sempice interfaccia di programmazione che, anche utenti privi di una preparazione tecnica, possono adoperare. In tal modo, gli edifici possono essere gestiti prendendo in considerazione realmente le esigenze e le preferenze degli utenti e rispettando, al contempo, i vincoli energetici.

Architetture di automazione orientate agli utenti sono state studiate in maniera estensiva nel vicino campo della *home automation*. Molti sistemi commerciali forniscono agli utenti delle interfacce intuitive e semplici da usare, che gli consentono di programmare il comportamento di diversi elettrodomestici. Per quanto riguarda la definizione delle politiche di gestione, la programmazione *trigger-action* si è rivelata una soluzione promettente per coinvolgere gli utenti nell'automazione domestica, in quanto fornisce un'interfaccia espressiva ed intuitiva. Essa consiste in un paradigma di programmazione basato sulle regole, che gli utenti possono sfruttare per inserire politiche personalizzate che coesistono insieme, cioè gli occupanti

possono esprimere le loro policy sfruttando il paradigma *IF This (happens) Then (do) That* (IFTTT).

Questa tesi sfrutta le funzionalità di un particolare sistema di gestione degli edifici, *BuildingRules* che estende il potere dimostrato dal paradigma IFTTT a scenari particolarmente complessi come gli smart building. BuildingRules fornisce infatti una piattaforma di programmazione degli smart building che fanno uso di diverse tecniche. Per prima cosa, l'intera rete di sensori ed attuatori viene nascosta all'utente, il quale può fare affidamento su interfacce di programmazione estremamente uniformi e standardizzate, che innalzano il livello di astrazione da una granularità di sensore/attuatore ad una di più alto livello. Come mostrato in letteratura, infatti, gli utenti preferiscono non interagire direttamente con sensori ed attuatori: per esempio, essi preferiscono riferirsi a "qualcuno è entrato in una stanza" piuttosto che "il sensore di movimento è stato attivato".

Nonostante lo scopo di BuildingRules sia quello di fornire interfacce utente semplici da usare, così come altri BMS, anche questo si adatta più facilmente alle *smart home* o a scenari altrettanto semplici. Quello delle smart home è infatti un contesto piuttosto elementare, in cui l'utenza è generalmente composta da un singolo o al più pochi utenti. Gli smart building, d'altro canto, rappresentano uno scenario significativamente più complesso a causa della loro dimensione e della loro natura condivisa, dove molti utenti, con esigenze differenti, vivono e lavorano negli stessi spazi producendo, in tal modo, molti conflitti. Per queste ragioni i metodi proposti per le smart home non sono direttamente impiegabili a causa della mancanza di supporto per problemi tipicamente inerenti allo scenario degli smart building multi utente.

Questa tesi mira a colmare le lacune che sono emerse in questo campo di ricerca, dedicandosi al particolare problema della *gestione della multi-utenza negli smart building.* In questa prospettiva, tre problemi principali sono stati identificati: il problema della minimizzazione delle regole, il problema della risoluzione dei conflitti ed il problema dell'analisi comportamentale dell'edificio.

Minimizzazione delle regole

Gli smart building, essendo basati su un paradigma di programmazione collaborativo, non garantiscono che l'insieme di regole prodotto sia corretto e sensato. Gli utenti sono liberi di inserire regole ridondanti o persino inutili, che non essendo necessarie per il corretto funzionamento dell'edificio, hanno come unico effetto quello di produrre enormi insiemi di regole. Le grandi dimensioni di questi, a loro volta, producono altri problemi. Innanzitutto, essi sono di difficile lettura per gli utenti che vogliono inserire nuove regole e vogliono capire come le nuove regole possano interagire con le altre. In secondo luogo, più regole vengono definite, più è probabile che esse possano interferire tra di loro, limitando e persino modificando le rispettive influenze. *La minimizzazione dell'insieme di regole* è quindi una parte fondamentale della gestione di uno smart building. Avere un insieme di regole ristretto, come detto, produrrebbe una notevole quantità di vantaggi sia per l'edificio che per i suoi occupanti. Per esempio, un insieme minimo sarebbe benefico per aumentare la percezione che gli utenti hanno riguardo le proprie regole. Esso potrebbe incentivarli ad inserire nuove regole ed a scoprire necessità e preferenze condivise con altri utenti, attivando così un circolo virtuoso di utilizzo.

Nel Capitolo 2, verrà presentato un metodo per minimizzare l'insieme di regole di uno smart building basato sullo stato dell'arte degli algoritmi di minimizzazione logica che furono introdotti a partire dal 1950. Sfruttando il parallelismo tra i circuiti logici combinatori e gli smart building, e prendendo in considerazione anche le differenze tra i due campi applicativi, è stato concepito un algoritmo *ad-hoc*. La sua efficacia è stata dunque mostrata applicandolo su dei dati provenienti da una campagna sperimentale.

Risoluzione dei conflitti

Un altro problema, strettamente legato ad insiemi di regole di grandi dimensioni, è quello della risoluzione dei conflitti. Vista la differenza tra gli occupanti dell'edificio, essi possono esprimere preferenze che sono in

contrasto tra di loro, e possono persino provare a sopraffare gli altri utenti sfruttando le infrastrutture degli smart building a loro disposizione. Per esempio, un utente potrebbe esprimere la regola di aprire le finestre durante il mattino, mentre un altro potrebbe desiderare di avere le finestre chiuse quando piove. Durante una mattinanata piovosa il sistema non saprebbe come comportarsi. Situazioni analoghe sono probabili in ambienti abitati da diverse persone come negli smart building. Avere una strategia che sia in grado di risolvere tali conflitti massimizzando la soddisfazione globale degli utenti è di primaria importanza.

I sistemi BMS odierni non sono sufficientemente preparati ad affrontare tali situazioni ed implementano quindi tecniche di risoluzione dei conflitti piuttosto banali. La strategia più adoperata consiste nell'assegnare diversi livelli di priorità agli utenti, che spesso riflettono l'ordine gerarchico dell'edificio. Quando un conflitto viene rilevato, l'utente con la maggiore priorità viene soddisfatto a dispetto degli altri. Questa semplice soluzione può portare a risultati accettabili quando gli utenti coinvolti nel coflitto sono pochi (massimo 2 o 3). Essa diventà però irragionevole quando il numero di utenti in conflitto cresce.

Nel Capitolo 3 verrà presentato, a tal scopo, un metodo per risolvere i conflitti tra regole negli smart building che si basa sulla modellizzazione della soddisfazione personale dei singoli utenti e, che mira a massimizzare la soddisfazione generale del bacino d'utenza dell'edificio. La correttezza e l'efficacia del metodo proposto sono stati validati attraverso una campagna sperimentale condotta internamente all'Università e che ha prodotto risultati incoraggianti.

Analisi comportamentale degli stati pozzo

Il paradigma di programmazione collaborativo può anche produrre scenari in cui alcune regole mancano del tutto. Un tipico esempio di tale situazione è rappresentato da una stanza in cui gli utenti hanno espresso solo regole per accendere le luci, senza aver definito alcuna regola per spegnerle. In tale situazione le luci potrebbero rimanere accese indefini-

tamente. Questo tipo di scenario è comunemente conosciuto nel campo della teoria del controllo come *stato pozzo*. Uno smart building, come detto precedentemente, è uno sistema di controllo automatico dotato di sensori ed attuatori. Esso dovrebbe sempre rimanere in controllo di tutti questi dispositivi e quindi, uno stato pozzo rappresenta un problema da risolvere.

Nel Capitolo 4 verrà presentato un approccio per superare le difficoltà tecniche che si manifestano durante la verifica formale del sistema di controllo di uno smart building. Differentemente dalle soluzioni appartenenti al campo della teoria del controllo, che presentano la necessità di personale altamente specializzato per la loro attuazione, è qui proposto un metodo stocastico che si relaziona con le probabilità che ogni regola ha di essere azionata. Questo metodo è stato sviluppato per trarre beneficio dallo specifico scenario rappresentato dagli smart building, al fine di semplificare le mansioni dei building manager nella vita reale. L'efficacia di tale metodo è stata validata su un dataset di regole collezionate durante una campagna sperimentale svolta all'interno dell'Università.

Chapter 1

Introduction

This chapter aims at giving an overview on Smart Buildings describing the state of the art and the theoretical background that regards them. At the beginning an operational definition of a smart building and its main features is given. Then the limits and the issues that are still unsolved are presented, highlighting how the contributions of this thesis work will go in the perspective of overcoming them.

1.1 Smart Buildings: Overview

Smart buildings refer to built environments such as apartments, offices, hospitals, schools, that are enabled for co-operation of smart objects (e.g. sensors, actuators, appliances) and systems, and for ubiquitous interaction with frequent and sporadic visitors [35]. Even if this definition is quite similar to the one here considered, there are a lot of other definition depending on the subjects involved in it [43]. Final users define as *smart*, buildings that are accessible remotely from other devices like smartphones, to turn on and off appliances or lights. Researchers define smart buildings as self-adaptive systems (i.e. systems that can adapt themselves dynamically to a change in their internal or external environment), which leverages complex strategies, like machine learning algorithms, to react and respond to inhabitants necessities. In industry, finally, *smart* is the marketing term

for programmable and reconfigurable technologies. To avoid confusion, it is necessary though to provide a precise description that will be used in the rest of this document for identifying smart buildings. In this thesis, a *smart building* is defined as a building that provides automation features in the perspective of increasing user *comfort* and *safety* while maintaining *energy efficiency* for the building itself.

As regards **user comfort**, smart buildings infrastructures can easily enhance occupants wellness by acting on environmental variables like temperature, humidity and air quality, adjusting them according to users needs. For instance, buildings are able to switch light on and off automatically when users occupy a room, or set the temperature according to external weather conditions and user preferences.

Fundamental in the management of a space occupied by human subjects is the **safety** aspect. It has to be treated in two complimentary ways. First users have to be guaranteed that smart building infrastructures and automation features do not compromise their safety. On the other hand, all those infrastructures (composed by sensors and actuators) can be exploited to improve the safety level of the building occupants. It is possible to imagine alarm systems and emergency equipments that make use of them to understand the context of an emergency and properly react to it.

Finally and possibly more consistent is the **energy efficiency** aspect. This topic has driven the research and the development of automated building management system unlocking the possibility of rationalizing the energy management. A very simple application would activate *Heating Ventilation and Air Conditioning* (HVAC) systems only when users are inside the building rooms, thus avoiding energy waste. In a prospective way, a smart building management system can be included in the so-called *smart grid*, the next generation of reconfigurable electric grid for the optimization of energy consumption.

The evolution of the research and development about sensor-based technologies has rapidly grown in the last years defining a branch of IT industry called *Internet of Things* (IoT). Day after day the pervasiveness of such technologies increases and embraces always more aspect of our

everyday life. This growth is supported by both industry and academia and this represents a solid basis for enabling the feasibility of the smart building vision.

In the following section a technical description of smart buildings will be provided together with a deep review of the current state of the art in this field. Finally some limitations and issues will be presented, introducing the premises for the solution that is proposed in this thesis.

1.2 Smart Buildings: Related Works

Throughout the years buildings have shaped to accommodate the latest needs of modern companies. Commercial Building Management Systems (BMS) have turned from simple HVAC management systems, to omni-comprehensive systems which take into account different tasks, handling, among the others, lighting, door openings, elevators, fire safety and emergency systems. While they drastically increase their complexity, modern BMS have just become a collection of sensors and actuators, controlled by very simple control logic [22, 65]. Most of them don't take into account latest trends and research topics about energy efficiency and the integration with emerging technologies such as Smart Grid [5], MicroGrid [38], Demand Response [59] and Building Automation [2, 39, 52].

In the following section the main architecture topologies will be described. Then in Section 1.2.2, the main challenges in Smart Buildings scenario will be presented.

1.2.1 The current idea of a Smart Building

From a theoretical point of view, a smart building is a distributed control system, endowed with a remarkable amount of sensors, actuators and computational elements. All of these are integrated in the building infrastructures and strictly connected together inside a continuous and invisible network. This connected network of sensors and actuators allows to overcome the conventional approach of building management, enabling

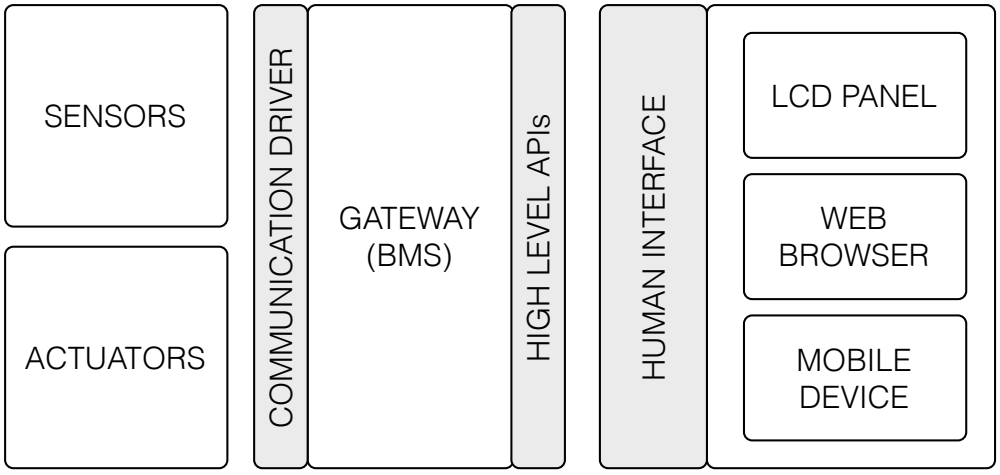


Figure 1.1: Standard architecture of a smart building.[1]

the possibility of creating custom policies which make use of them. Figure 1.1 represents the standard architecture of a smart building, mainly composed of three different layers:

- Hardware layer
- Backend Sofware Layer
- Frontend Software Layer.

The **hardware layer** is composed by a **sensor network** that is exploited to retrieve data (e.g. temperature, humidity, occupancy...) in a distributed fashion and to report it to the software layer. An **actuator network** is used to modify the building behavior acting on the same variables that are sensed (like temperature, humidity, etc.) or actuating different appliances such as lights, windows, monitors.
Sensor data and actuator commands are managed by a piece of software that is responsible for the building functioning. It is called **Building Management System (BMS)** and it represents a bridge between hardware and users.

[1] Credits: Alessandro Nacci: "BuildingRules: a trigger-action based system to manage complex commercial buildings"[44].

Human Interaction is handled by a **front-end software** that exposes user-friendly interface in the shape of monitors, consoles, web applications and mobile devices.

Different hardware devices can be exploited in smart buildings being already available on the market. Some of them are really simple sensors being able of "reading" quite basic values (temperature, humidity, light, window or door opening status...). Others are way more complex, sensing multiple variables or directly actuating on the controlled environment. Examples of commercial sensors and actuators are *Nest Learning Thermostat* [46], *Philips Hue Lamp* [51], *August Smart Lock* [7] or *Netatmo Weather Station* [47].

The sensor network management is a non-trivial task entailing the necessity of interfacing with different protocols and topologies. Apart from proprietary solutions that are exploited by many sensors and actuators, many different standard protocols have been defined for the intercommunication of devices belonging to wired or wireless sensor networks. Some examples are represented *Zigbee* [80], *Z-wave* [77] and *openHAB* [48]. Given the enormous quantity of different protocols and standards a hardware abstraction layer is generally introduced inside the BMSes. Specifically building management systems are responsible for gathering data from the sensors and uniforming it to a defined standard. This data is eventually made available for being accessed by external systems in the perspective of building complex application with it. Depending on the complexity of the building BMSes can be designed according two different paradigms: a *centralized* and a *distributed* one.

The *centralized* topology is depicted in Figure 1.2(a) and consists in a single central BMS module responsible of gathering data from all sensors and directly sending commands to all the actuators. This solution is very simple to be implemented but has an important scalability limitation. With a high number of nodes the central computational element has to process a considerably large number of I/O operations thus requiring large computational power. For this reason this first topology can be implemented in quite simple buildings like smart-homes for instance.

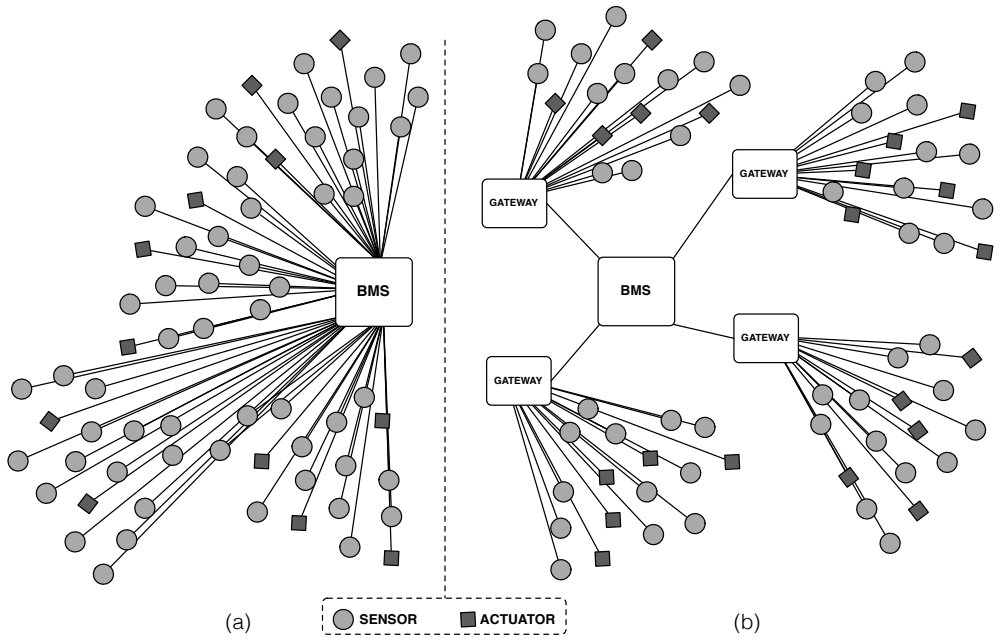


Figure 1.2: Centralized and distributed BMS architectures[2].

The *distributed* topology (Figure 1.2(b)), on the other hand, makes use of several gateways that are responsible of a limited amount of sensors in their nearby. Following this approach it is possible to keep the number of communication of the BMS under safe limits. Adding sensors require just the add of a new gateway or the exploitation of an already existing one. In this way it is possible to handle a remarkably larger number of sensors and actuators at the cost of building a more complex architecture. This topology hence represents the state of the art for large commercial buildings composed of many floors and rooms.

Modern BMSes [6, 74] moreover, have been implemented exploiting web services, in order to easing the interaction with them. Furthermore such web services have been developed in the perspective of scalability and maintainability also employing standard state of the art solutions for the task scheduling and programming or the human interaction. Furthermore this kind of BMSes improve comfort and productivity while optimizing energy efficiency.

[2]Credits: Alessandro Nacci: "BuildingRules: a trigger-action based system to man-

Representative of these features are the following web-based BMSes:

- BOSS [25], a distributed control system, developed at UC Berkeley, which aims at supporting the physical resources in large commercial buildings being scalable and fault-tolerant. It consists in two layers providing respectively an hardware abstraction layer for the underlying sensor network and a uniform interface for interacting with external applications.

- BuildingDepot [74] which is an extensible and distributed system for handling building-related data developed at UC San Diego. It provides mechanisms to access data from distributed sensor networks and store it in distributed data points. Moreover it exposes uniform interfaces to allow external application use the collected data for high-level purposes.

- Energy@Home Jemma [28], an Italian project developed at Telecom Italia, is a modular, resource-oriented middleware running in Home Gateways and in the Cloud able to support energy monitoring. management and awareness applications at home.

1.2.2 Current challenges and limitations

Smart Buildings topic has been extensively studied in the last years both in the academic and industrial world. The latest works have discussed about the spread of *Ubiquitous Computing* and about the impact that it will have on the concretizzation of *smart-environments* actuation [1, 8]. In the last years, several challenges have been addressed especially to make *smart-environment* vision possible from a technical point of view. After having reached this goal, the research have moved on higher level challenges to make it widely adoptable and available.

In the remaining part of this section we will focus on three main challenges that arouse in the last years: *building behavior description*, *safety and security* and *building programming* problem.

age complex commercial buildings"[44].

Building Behavior-Description

Smart Buildings, as already said, are distributed control systems. They are designed to operate in an autonomous way, taking into account users needs but without depending on their operate. For this reason it appears evident that the design of smart building control systems have to be sound and correct. Being able to verify the correct functioning of all the parts that compose it is paramount and, the capability of foresee future evolutions of building behavior is essential for both safety and economic reasons. In this perspective is hence needed a theoretical and operational framework to conduct specific behavioral analysis to guarantee, at each time, that the correct actuation strategies are performed.

Safety and Security

Strictly related to the previous challenge is the safety and security one. As already stated, Smart Buildings are autonomous in their decisions and actuations. This autonomy raises doubts in building inhabitants who are scared about the drawbacks that such technology might have on them, both in common and extraordinary situations. Users have to be assured about the fact that building automation systems will not become dangerous for their safety in particular scenarios.

Different from safety is security. Being an IT system, a building management system suffers from the same problems deriving from the IT world. Malicious users can perform attacks exploiting the vulnerabilities of its design in order to create threats to user safety, or just for economic reasons. Moreover, even experienced users could exploit system vulnerabilities at their own advantage, forcing the building to behave in a different way from the designed one. Security must be included in the design of such system and attack resiliency is a mandatory feature for a system that deals with human subjects.

Building Programming

As previously stated the enabling technology for Smart Buildings spread have been developed and is currently available on the market. Both hardware devices and software systems have been studied and implemented to make the Smart Building creation a standard and straightforward process. Nonetheless the effort that have been put in this context, there are still some issues that limit the diffusion of such environments.

BMSes today are a prerogative of only building managers and maintenance personnel. The user interface that is exposed to building managers requires a deep technical training thus preventing common user to leverage all the infrastructures that modern buildings make available. Tenants, which are the majority of the people that occupy the building, cannot express their preferences in order to automatically actuate simple policies for customizing, for instance, the temperature or the lighting settings. They can interact with them only by using manual controls thus wasting the potentiality offered by Smart Buildings infrastructures. Furthermore, while *programming* building behavior, building managers cannot count on an appropriate abstraction level. They are indeed able to only interact with sensors and actuators values rather than directly control the state variables they want to be in control of. On the contrary, in the perspective of a user-oriented building management, there is a concrete need of a simple and intuitive user interface which provides a right abstraction level to not overwhelm the user with unnecessary implementation details.

User oriented automation design have been extensively studied in the similar field of Home Automation. Many commercial systems [64] provide user with intuitive and simple to use interfaces they can use to program the behavior of many house appliances. 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 [66, 69, 70]. It consists in a rule-based programming paradigm which users can exploit to insert custom policies that coexist together, i.e., occupants can express policies using the *IF This*

(happens) Then (do) That (IFTTT) pattern". Dey et al. have demonstrated that with IFTTT paradigm it is possible to express 95% of all the applications conceived by users in smart homes, thus presenting a vast collection of context aware applications which emphasize its expressiveness power [27]. In another work, Ur et al. demonstrated that 63% of smart home application desired by users needed some sort of programming and all of them could be programmed through the IFTTT paradigm [70]. IFTTT.com is a widely-adopted web application that exploit this paradigm to create a connection among several web services and smart home appliances [32].

All the aforementioned works have been proposed for smart homes or similar simple scenarios. Smart Homes represent indeed a very basic scenario where the user-base is usually composed by a single or (at most) few users. Moreover, smart homes inhabitants are generally members of a family or are at least very close to each other. For this reason they share similar needs and preferences or are very inclined to find a common wellness. Furthermore, even if the home automation system is exploited by many users there is usually a single user which is elected as technological *guru* as shown by Brush et al. [12]. The *guru* is the user who has the technical expertise and who generally installed the system. He also usually trains other users to use the system and hence he is the one who usually inserts the majority of the rules. For all the reasons that have been discussed so far it is evident that in the home scenario the number of rules and the possible conflicts among them is kept under safe limits. Smart Buildings are instead a significantly more complex scenario 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. For these reasons, the methods proposed for the Smart Homes are not directly applicable, lacking the support for problems that are typical of multi-tenant smart buildings scenarios.

In Section 1.3, *BuildingRules* will be presented as a partial solution for some of the issues here described. This BMS, developed at *Politecnico di Milano*, have been chosen as building management platform since it

builds a programming abstraction layer in the perspective of easing the programming task for building occupants. In Section 1.4, multi-tenant implications and issues in the building management and programming tasks will be presented, laying the foundations for the work that have been developed in this thesis work.

1.3 *BuildingRules*

So far, buildings policies have been manually written by technicians and specified by building-managers, which have exploited their expertise and knowledge to accomplish this task. An infrastructure like the one of Smart Buildings, on the contrary, allows the occupants to directly interact with it, 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. In this way, buildings can be handled taking into consideration users needs and preferences while respecting energy constraints in the perspective of energy efficiency.

1.3.1 Overview

BuildingRules [44] fills this gap by combining the power demonstrated by the IFTTT paradigm, extending it to cover also particularly complex scenarios like Smart Buildings. BuildingRules, provides indeed, a framework for programming Smart Buildings leveraging different techniques. First of all, the whole underlying sensors and actuators network is hidden to the user, who can rely on uniform and standardized programming interfaces that increase the abstraction layer from a sensor/actuator granularity to a more user-friendly one. As shown in prior work [66, 69, 70], 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. As already stated, BuildingRules exploit the IFTTT paradigm by mapping the *antecedent* of the rule (*if*) to sensors values and the *consequent* (*then*) to command that are sent to actuators. All the

rules are then converted into first order logic and checked using z3 SMT (Satisfiability Modulo Theory) solver [26]. This logic module is responsible of the actuation process that is periodically executed by checking the rules whose *antecedents* are triggered and actuating their *consequent*. The SMT solver is also used to detect very basic *static* conflicts which are pointed out to the user when he is inserting a new rule. Some conflicts, instead, are not evident at the time of the rule specification and appear only in the actuation phase. For this kind of *dynamic* conflicts, a priority based system is exploited, actuating the *consequent* of the rule with the highest priority. Finally, as stated in BuildingRules 2.0 revision [33], BuildingRules also perform some behavioral analysis about how the building state evolves through the time. These analysis are conducted starting from the ruleset by analyzing how all the possible combination of the rules can affect the building behavior.

In the following sections the architecture of the system will be presented together with the main features and challenges that it has to face in real world applications.

1.3.2 System Components and Architecture

The main goal of BuildingRules is to provide building inhabitants with a simple, scalable and intuitive system to permit them to express their preferences. This task is accomplished letting them customize their spaces according their needs. In order to expose such functionality to tenants it is possible to exploit information provided by commercial BMSes. Hence BuildingRules has been developed as an additional layer between BMS and users, as it can be noticed in Figure 1.3. Such design allows to further decouple user interaction layer from hardware and building management details. BuildingRules provides users with a simple and user-friendly interface that allows them to easily program building behavior without taking care about implementation details. User commands are then processed to adhere to the format required by the BMS which manage the physical actuation (or sensing) phase. For example, if the rule *"if someone is in*

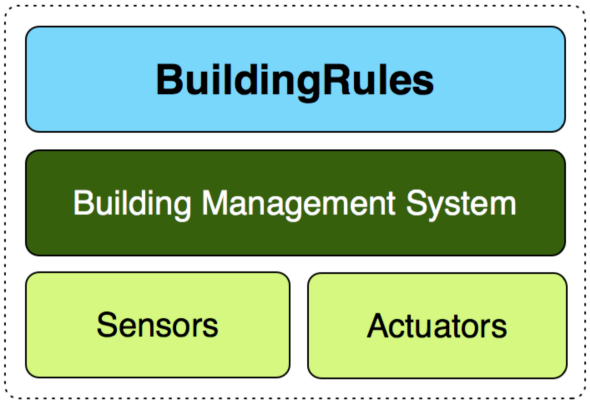


Figure 1.3: Smart Building stack. BuildingRules is inserted on top of BMS as an additional layer between it and user interaction one.

the room then turn on the lights" is inserted in BuildingRules, it will periodically query the BMS about the occupancy status of that room. If this condition is verified, BuildingRules will send a command, to the BMS for, actually, turning the lights on.

BuildingRules is addressed to both simple environments and complex commercial buildings. The latter environments are generally composed of many rooms and group of rooms (e.g. floors). There exists different kinds of rooms featuring different structures and appliances as for instance offices, conference rooms, lobby or kitchens. In BuildingRules occupants are assigned to these rooms by the building manager, who authorizes them to customize the behavior of those rooms through the insertion of rules into the system. To handle all these different components, BuildingRules architecture has been developed in a modular and extensible fashion as depicted in Figure 1.4.

Here below the main components of BuildingRules architecture are described:

- **Rule**: it is the enabling feature for the building programming through a trigger-action mechanism. There exists different categories of rules.
- **User**: representing a human inhabitants or other devices or applications, it can be assigned different privilege levels according to the

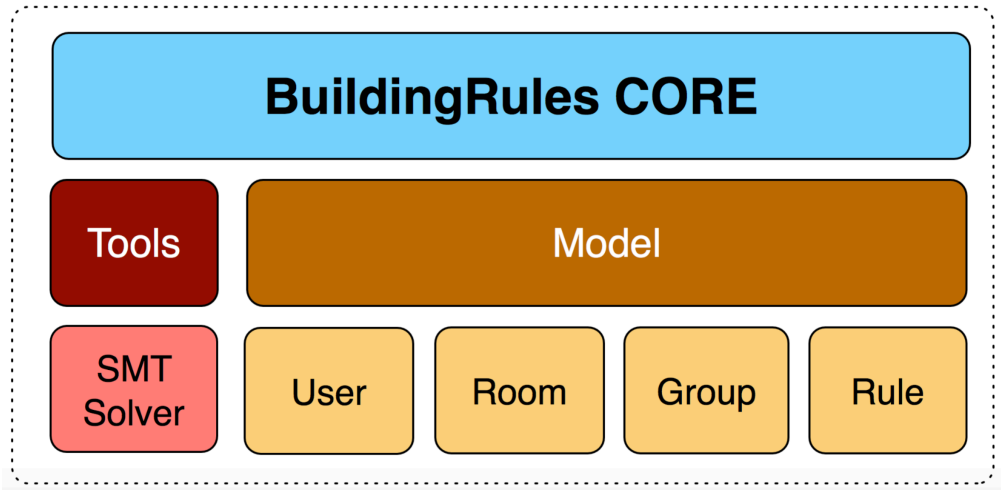


Figure 1.4: BuildingRules Architecture

type of the user

- **Room**: physical space which rule antecedents or cosequents are referred to.

- **Rooms Group**: aggregations of rooms used to simplify the insertion of rules that regards more than one rooms (e.g. floor-granularity rules)

A fundamental component of BuildingRules architecture is represented by **SMT Solver**. It handles conflicts between rules and assure that the entire ruleset (expressed in propositional logic) is formally consistent.

In the following sections the aforementioned components will be described in more details.

1.3.3 Rules

As previously mentioned, BuildingRules exploits the trigger-action paradigm to allow occupants express rules in the following format:

if (something happens) then (do something)

The *if* part of the rule is called *trigger* while the *then* is called *action*. Following this paradigm it is possible to specify an action to be performed when certain event conditions are met. For instance a simple rule for switching on the light according to occupancy can be defined as follows:

if (someone is in the room) then (turn on lights)

.

Apart from such very simple rules, more complex rules can be inserted. A *complex* rule is defined as a rule having more than one *trigger* connected by a logical AND. For example, *if it is raining then close the windows* is a simple rule while *if it is Sunday and it is after 10am then close the curtains* is a complex rule. In BuildingRules complex rules are restricted to just multiple triggers; in fact, rules with multiple actions are not supported to keep the user interaction paradigm as simple as possible. Anyway, this is not a limitation for the expressivity of the system since those rules can be easily decomposed into multiple rules with the same trigger.

Table 1.1 shows the list of triggers and actions currently supported by BuildingRules. There exist two different kind of attributes: *boolean* (used for lights, windows monitors...) and *integer*, defined over range intervals (e.g. time, temperature...). Triggers and actions are also divided in different categories related to different sensors or actuators kinds. For example, the rule *if it is rainy then turn on the light* has the antecedent that belongs to the *Weather* category and the consequent to the *Light* category. *NO_RULE* is a special trigger available for the building administrators (Rule 7 in Table 1.1). This trigger is always set to True and is used for setting the default conditions of the building that can be used as a fallback in case no rule is specified by the user to override them. For example, if the rule *if no rule specified then close the windows* has been inserted, in case the occupants forget to insert a rule that manage the windows during the night, the windows will be closed in any case.

BuildingRules also supports external applications through virtual triggers that are controlled via RESTful APIs (Rule 9 in Table 1.1).

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

Table 1.1: Currently supported rule triggers (T) and actions (A) categories. An example of trigger or action for each category is provided.

1.3.4 Conflicting Situations

Since users coexist together in shared spaces conflicts can arise among the rules they insert. In BuildingRules two rules conflict if they are defined on the same actuator and try to apply opposite effects on it. Two main kind of conflicts can occur: **static conflicts** and **run-time conflicts**.

Static Conflicts

Two rules conflict statically if the triggers are defined over non-disjoint interval belonging to the same category and the actions on the same actuator yet in incompatible ways. They conflict because they can be triggered together, thus bringing the system in an incoherent state. For the sake of simplicity two statically conflicting rules are here reported:

if (time is between 9am and 6pm) then (turn the HVAC on)
if (time is between 5pm and 8pm) then (turn the HVAC off)

In the previous example between 5pm and 6pm the system would turn the HVAC on and off at the same time. This incoherent state can provoke malfunctioning and even damages to the controlled HVAC system and should be though avoided.

SMT Solver

Static conflicts can be avoided using a Satisfiability Modulo Theroy (SMT) solver. In BuildingRules the SMT implementation provided by z3 [26] is expoited for this purpose.

As said before, a rule is composed of two parts: a trigger or a conjunction of triggers, and an action. Before adding a rule, it is verified against the set of rules already in the room. Each rule is represented as a propositional formula composed by an implication in which the trigger implies the action. In general, the implication is satisfied if the trigger is not satisfied, or if both the trigger and the action are satisfied. In this context, the action is considered as a proposition that is true if the action can be executed, 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, the rules are not in conflict with each other. If not, two or more rules are in conflict and it needs to be resolved.

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. Since z3 supports boolean and integer variables, translating the formalization of rules into the z3 language is straightforward.

When a user inserts a rule that is conflicting with existing rules, a notification is raised and the user is asked to modify the rule. The rules are translated from human readable syntax to the z3 syntax using a predefined look up table (see Table 1.2).

z3 has been chosen to detect conflicts as it is efficient and reusable. Although the satisfiability problem is computationally expensive, it is possible to ensure low latency as z3 solves this problem efficiently.

Run-Time Conflicts

Some conflicts cannot be detected using SMT solvers. They are conflicts on rules that are defined over actions which belong to the same category but over triggers that do not. For example the couple of rules

	TYPE	EXAMPLE HUMAN READABLE SYNTAX	EXAMPLE Z3 SMT TRANSLATION
1	T	someone is in the room	(inRoom)
2	T	external temperature is between @val and @val	(and (>= (extTempInRoom) @val) (<= (extTempInRoom) @val))
3	T	time is between @val and @val	(and (>= (time) @val) (<= (time) @val))
4	T	the date is between @val and @val	(and (>= (day) @val) (<= (day) @val))
5	T	it is sunny	(sunny)
6	T	room temperature is between @val and @val	(and (>= (tempInRoom) @val) (<= (tempInRoom) @val))
7	T	no rule specified	(noRule)
8	T	today is @val	(= (today) @val)
9	T	calendar meeting event	(meetingEvent)
10	A	turn on the room light	(light)
11	A	open the windows	(openWindows)
12	A	set temperature between @val and @val	(and (>= (tempSetpoint) @val) (<= (tempSetpoint) @val))
13	A	turn on the coffee machine	(coffee)
14	A	send complain to building manger	(sendComplain)
15	A	open the curtains	(openCurtains)

Table 1.2: Translation to Z3 examples.

below:

if (nobody is in the room) then (turn off the light)
if (time is between 5pm and 8pm) then (turn on the light)

do not statically conflict because SMT solver cannot predict if there will be nobody in the room between 5pm and 8pm. Analyzing such conflict would mean to taking into account all the possible combination of triggers which entails a high complexity with a rule-base with a high number of rules. Such kind of conflicts are yet likely to occur in shared contexts like Smart Buildings one. Given the complexity of the problem that has to be solved, though, it is not possible to solve these conflicts *a priori* but rather a run-time resolution is performed.

In order to provide a run-time resolution strategy BuildingRules exploit a **priority** based method. When inserting a rule users are asked to select a priority level for that rule. Such priority has to be uniquely-attributed for each rule belonging to the same actuator category (i.e. two rules defined over the same actuator category cannot have the same priority level). In this way when a conflict is detected at run-time, the rule with the highest priority is selected as "winner" and is then actuated.

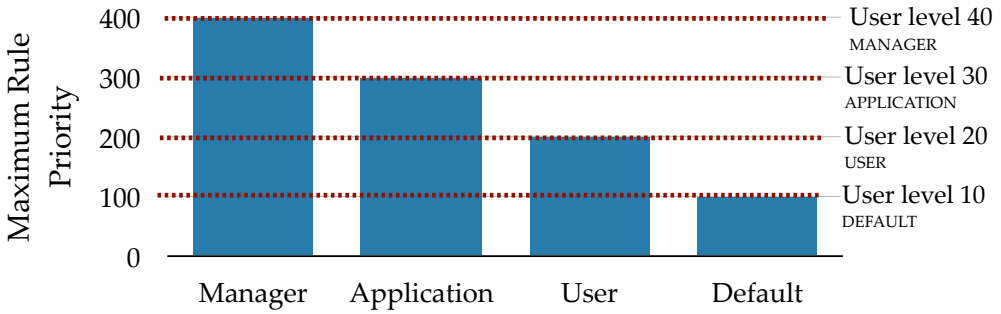


Figure 1.5: Priority levels hierarchy. There are four main kind of *users* for BuildingRules: (Building) Manager, Applications, (Human) Users, Default (rules that are triggered when no other rules exist.)

1.3.5 Users

As already expressed before, in commercial buildings, many different people coexist together, exploiting different tasks and duties. A Smart-Buildings that let users express their own policies has to take it into account when dealing with them. Different users can have different access permissions: an employee can access her office (and the meeting room for instance) while a building manager can have access to all the rooms of the building. Moreover external applications are also able to insert rules in the systems. Depending on the *authority* of the kinds of user different privilege levels has to be assigned to them.

BuildingRules support different user categories as it is depicted in Figure 1.5. The actual building occupants may be classified as building *managers* or standard *users*; while special users may be assigned to *Applications* and *Default* categories. Applications may be external software that interact with BuildingRules, while Default can represent agents controlling the default status of the building, i.e. when users do not specify any rule (represented by the DEFAULT_STATUS category at line 7 of Table 1.1).

User levels are used in two different ways. First, they ensures that low level users cannot edit or delete the rules specified by users with higher level. For example, consider a room R1 shared by two users (U1 and U2) (level of U1 $>$ level of U2), both users are allowed to enter rules

into R1; U1 can edit or delete rules specified by U2 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 more priority in case of run-time conflicts, i.e. overriding rules expressed by a lower-level user.

1.4 Multi-tenant programming

Previous sections presented many different solutions for the building automation problem. Different kind of architectures and implementations have been described for different kind of environments. Such solutions have been proposed to be suitable for those different scenarios but it is clear that different contexts require different management strategies. Even from just the technical point of view enormous differences exist when dealing with commercial or residential buildings [45]. Nonetheless both are featured with a widespread network of sensors, network and computational elements enabling the management of the entire building, they are organized in a completely different way. While **commercial buildings** are generally characterized by an extended and fine grained network infrastructure that interconnects all their rooms and floors, **residential buildings** are usually composed of different apartments that have independent communication networks and have a limited amount of shared resources with others. The management of such spaces should take into consideration those differences. In residential scenarios the automated management should reflect the independence between apartments, providing separated interfaces for each apartment. Commercial buildings, on the other hand, are inherently more complex due to their scale and their shared nature. In fact commercial buildings are usually occupied by several people that share different spaces. Each one has different needs and preferences that, if put together, can result in a huge set of policies really difficult to handle.

Ruleset Minimization

Exploiting a *collaborative* programming paradigm, there is no guarantee that the ruleset that is produced is sound and correct. Users may insert redundant or useless rules that are not strictly necessary for the proper functioning of the building, but that produce, on the other hand, a huge ruleset. Large rulesets produce in turn other problems. First of all, they are difficult to be read by users who want to insert new rules and want to understand how the new rules will interact with others. Secondly, the more policies are defined, the more probable it is that they can interfere each other, thus limiting and even modifying the influence each one of them has. **Ruleset minimization** is then a fundamental part of a smart building management. As already stated, having a restricted set of rules would produce a remarkable amount of advantages for the building and their occupants. For instance, providing them with a limited set of rules could be beneficial for increasing the perception they have about their rules. It can thus foster them to insert new rules and to discover needs and preferences that are shared with other users, enabling then, a positive usage loop.

It is clear hence that a minimum ruleset provides users remarkably benefits. The minimization task is not a simple task though. In fact, shrinking the size of the ruleset is not sufficient to minimize it. This task has to be accomplished while respecting the preferences that have been insserted by the users. The minimum ruleset should produce thus, the same exact building behaviors of the original one. In Smart Building context this is a pretty new concept that has never been taken into consideration before. In parallel fields, on the other hand, solutions to similar problems have been carried out leveraging propositional calculus and first order logic. Such solutions could be exploited also for smart buildings but are probably too specifically related to the original application field which they were originally designed for. Furthermore most of them have been designed to solve very specific little problems rather than providing a unique solution to all of them. For this reason, it would be necessary

to exploit many different optimization techniques that could corrupt the performance of the building management system.

Conflict resolution

Another problem, that is also related to large ruleset, is the **conflict resolution** one. Since tenants are different each other, they can express preferences which contrast each other or even try to overpower each other exploiting the Smart Buildings infrastructures they are provided with. For instance a user could express the rule of opening windows during the morning, while another one would want to have window closed when it is rainy. During a rainy morning the system would not know how to behave. Such situation are likely to occur in ambients inhabited by several people like the smart buildings one. Having a strategy that is able to resolve such conflicts while maximizing the overall users satisfaction is paramount.

Modern BMSes are not properly prepared to face such situations and implement quite basic conflict resolution techniques. The most exploited strategy consists in assigning different priority levels to users also according to some hierarchical structure. When a conflict is detected, the user with the highest priority level is satisfied against others. This pretty simple solution produces acceptable results when the users involved in the conflict are few (maximum 2 or 3). On the other hand it becomes unreasonable to select just one person to satisfy when the conflicting people grow.

Sink states analysis

Finally the collaborative paradigm could also produce scenarios in which rules are missing. An example of this situation would be a room in which users have expressed only rules to turn the lights on, without any rule for switching them back off. In such scenario the lights may stay indefinitely turned on. This kind of situation is commonly known in control theory as *sink state*. A smart building, as previously stated, is an automated control system that is endowed with both sensing and actuating

devices. It should always be in control of every one of these appliances, and hence a sink state represent a problem that has to be solved. **Sink state analysis** and resolution techniques from the control theory world involve the exploitation of advanced mathematical techniques like *model checking* or *automated theorem proving*. Even if these approaches can be automated, it is not always simple to do it and, in any case, it requires the exploitation of very expert people that are able to do it. In the perspective of being used by building managers such solution become practically unusable.

1.5 Contribution and outline

After having exposed the main concepts and ideas, the current state of the art, the challenges that have been addressed and the ones that are still unsolved, the next chapters will present the work done in the thesis highlighting the contribution given to the smart building field. As previously stated, this thesis tries to fill the gaps that are emerged in this research field addressing the particular problem of the *multi tenant management in smart buildings*.

Even if BuildingRules provides a significantly advanced framework for the automation in Smart Buildings, the multi-user managements is not totally supported in it. Multi-tenant implications have not been adequately investigated in the current state and some issues remain still unsolved:

- A way for minimizing the ruleset keeping its size to the minimum necessary to guarantee the correct behavior of the building is needed;
- An accurate conflict resolution strategy that can scale well on conflicts involving many users, trying to accommodate all the users needs is missing;
- A behavioral analysis regarding the correct control of the building has not been performed.

These and other concepts will be faced in this thesis work trying to provide solution to support the multi-tenant management in smart buildings.

In particular in Chapter 2, a method for minimizing the ruleset of a SmartBuilding will be presented. It has been developed starting from State of the Art boolean logic minimization algorithms that have been firstly introduced in 1950s. Exploiting the parallelisms between boolean logic circuits and Smart Buildings, and taking into account also the differences between these two fields, a custom algorithm has been conceived.

In Chapter 3, a method for solving conflicts among rules in Smart Buildings will be presented. It has been developed starting from State of the Art priority-based conflict resolution methods, by extending their power while solving some issues that was not taken into account. It is based on user satisfaction modelization and aims at maximizing the overall satisfaction of the building userbase.

Finally Chapter 4 will present an approach for overcoming the technical difficulties that can be faced while trying to formally verify a Smart Building control system. In contrast with existing solution derived from the control theory world, it is presented a stochastic method that copes with the probabilities that each rule, inside a ruleset, has to be triggered. This method will be presented in order to leverage the features and the capabilities of the specific scenario represented by Smart Buildings, in order to simplify Building managers duties in real world applications.

Chapter 2

Ruleset Minimization

In this Chapter, a method for minimizing the ruleset of a SmartBuilding will be presented. It has been developed starting from State of the Art boolean logic minimization algorithms that have been firstly introduced in 1950s. Exploiting the parallelisms between boolean logic circuits and Smart Buildings, and taking into account also the differences between these two fields, a custom algorithm has been conceived. It has been then integrated in BuildingRules building management system in order to evaluate its effectiveness. Its performances have been measured, observing coherent results to what was expected.

2.1 Introduction

In multi-tenant smart building contexts the management of the ruleset is a crucial aspect in the perspective of an efficient and correct administration of the spaces and environments.

Exploiting a *collaborative* programming paradigm, there is no guarantee that the ruleset that is produced is sound and correct. Users may insert redundant or useless rules that are not strictly necessary for the proper functioning of the building, but that produce, on the other hand, a huge ruleset.

Large rulesets create several issues in the building management. First of

all, they impede users from obtaining what they program the building for. Secondly, they are difficult to be read by users who want to insert new rules and want to understand how the new rules will interact with others [75]. Moreover, the more policies are defined, the more probable it is that they can interfere each other, thus limiting and even modifying the influence each one of them has. Finally they create also a technical issue, slowing down the analysis that are conducted on the ruleset [33, 67].

For all these reasons there is a concrete need of making the ruleset minimum while maintaining the desired behavior of the building. The conflicting nature of multi-tenant rulesets is thus here exploited to supply logic minimization techniques, borrowed from the EDA research field.

In the next Section, the State of the Art techniques of logic minimization in the EDA field are presented, together with some parallelisms and differences between EDA field and Smart Building rule-set minimization. Section 2.3 will present the proposed method to minimize the rule-set size, while maintaining the correct functioning of the building and without affecting the preferences expressed by user policies. Section 2.4 provides a description of how it has been implemented inside BuildingRules management platform. Finally Section 2.5 contains a description of how the proposed system has been validated and discuss the results coming from the validation phase.

2.2 Background and Related Works

Binary Logic Minimization is a research field of the Boolean Algebra that was extensively studied from early 1950s. It studies the problem of producing the smallest logic circuit that represent a given Boolean function or truth table. In real world applications, this problems correspond to minimizing the usage of constrained resources like logic gates and connection to implement a logic function. Even if this problem has been demonstrated to belong to the second level of the Polynomial-Time Hierarchy complexity class [13], in the latest century many heuristics have been developed to obtain good results in a brief time. The main ones will

be presented in the following paragraphs.

2.2.1 Karnaugh Map

Named after his inventor Maurice Karnaugh who introduced it in 1953 [34], this algorithm exploits the human ability of finding pattern in visual representation of data to simplify boolean algebra expressions. The function outputs, commonly represented by means of a truth table, are used to fill a grid (Karnaugh Map) in which the cell are ordered in Gray code, i.e. in such a way that adjacent cells have unitary Hamming distance. This order guarantees that only one variable changes between each pair of adjacent cells. Cells positions represent the combination of the input conditions, while the value inside the cell represent the output value.

After the construction of the map, it is analyzed to find one of the simplest possible canonical forms. The *minterms* that will form the final expression are found by encircling groups of 1s in the map. These groups must have rectangular shape and an area which is power of two (i.e. 1, 2, 4, 8...). The largest possible rectangles which does not contain any 0s will be chosen in the final expression even if they overlap together to reach greater dimensions. Minterms can also make use of don't cares (-), input configurations that are not specified in the boolean function because not relevant or impossible to happen. Don't cares allow to further extend the dimension of the rectangles but it is not mandatory to use them, since they could also lead to suboptimal solutions.

In figure 2.1 an example of Karnaugh map is represented, together with its corresponding truth table and the constructions of the minterms of the final expression. It can be noticed that the final expression is obtained by evaluating the input conditions that hold inside each minterm rectangle. For instance, the red (4x1) rectangle correspond to the input configuration for which C=1 and D=0.

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1

A	B	C	D	F
1	0	0	0	-
1	0	0	1	-
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	0	1
10	-	-	1	1

F = A'C + CD' + AB' + AC'D'

Figure 2.1: Example of Karnaugh map construction starting from a Truth Table representation of a boolean logic function. Over the Karnaugh map, in different colors, the largest minterms composing the final expression are drawn.

2.2.2 Quine-McCluskey

The Quine-McCluskey algorithm is a method developed by Willard V. Quine [57, 58] and extended by Edward J. McCluskey [41] in the first half of 1950s. It is functionally identical to Karnaugh Map method, but due to its tabular fashion it is more efficient to be implemented in computer programs. It also specifies a deterministic method to check that the minimal form has been found.

Even if it scales better than Karnaugh map with function of more than 4 input variables, its resolution time grows exponentially with the number of input variables as well, given the NP-hardness of the problem that has to be solved.

For this reason neither Karnaugh maps nor Quine-McCluskey algorithm find application in real world scenarios.

2.2.3 Espresso

Espresso logic minimizer, so called to stress the speed boost of its resolution time (compared to the time of making a cup of coffee), has

been developed at IBM by Ribert Brayton [11]. Instead of dealing with minterms, as previous methods do, manipulates *cubes*, representing the product terms in the ON-, DC- and OFF-covers iteratively. Even if it does not guarantee to reach the global minimum, it can be very closely approximate with it. In comparison with previous methods, it provides a significant reduction of the execution time and the memory usage that can be quantified in several order of magnitude, thus removing any restriction on the number of variables, output functions and product terms. In practical cases, it can handle easily optimization of tens of variables with tens of output functions.

Espresso, together with the numerous variants that have been developed after it (e.g. Espresso-SIGNATURE [42]), represent the de-facto standard for boolean logic circuit minimization and is currently implemented in the modern EDA optimization tools.

The aforementioned methods represent the State of the Art in terms of boolean logic minimization, having been used and tested for more than half a century by now. Therefore many minimization problems, also belonging to totally uncorrelated fields, are connected to the boolean logic minimization in order to exploit these methods. An example of this trend is represented by [62], where a great effort is put in the translation of a set of Fuzzy Logic rules in equivalent Boolean Logic functions to be minimized.

In the specific context of Smart Buildings, as the best of our knowledge, the interest in minimizing the ruleset has not show up yet. For this reason we looked for related works in similar contexts represented by different research fields. A very similar research field that had to deal with the ruleset minimization task in the past years is the Expert System one. Expert Systems are composed by set of rules responsible of producing outcomes for supporting (or independently taking) decisions, starting from some inputs. Such rule-sets are composed by rules expressed in the if-then format, hence are very similar to the rules that compose a Smart

Building management system. Suwa et al. [67] present an approach for verifying the completeness and consistency in a rule-based expert systems by checking all the possible combinations of inputs to find conflicts or redundancies. In [31], a solution for finding redundancy in fuzzy-logic-based expert system is presented leveraging the power of propositional calculus. A very interesting taxonomy of many possible anomalies is provided by Preece et al. [55, 56] together with a survey of automated verification tools. This work highlights the existence of many different anomalies that has to be faced in the design and verification phases of building an expert system. Moreover it describes many different and independent solutions that have been proposed along the time for solving all the highlighted issues. Expert Systems can reach very high level of complexity and all such solutions contribute to increase its complexity. Smart Buildings, on the other hand, represent a much simpler scenario to be handled, thus requiring less complex management tools. Furthermore, in the specific context of BuildingRules management system, many of these problems (like duplicate rules and opposite rules) have been already addressed by the SMT solver that has been included in the system.

Going on the same line of the previous works, we consider very useful the possibility of exploiting such a consolidated theoretical background, also considering the many contact points between boolean logic circuits and Smart Buildings. First of all the most immediate similarity is the rule-based architecture of the Smart Buildings trigger-actions policies. The cost represented in logic circuits by the area size of the circuit itself, is, in the Smart Building context, directly connected to the legibility of the ruleset and the execution time of the building behavioral analysis [75]. Moreover since many input configuration can result impossible to occur, Smart Buildings ruleset can take advantage of these configuration as *don't cares* in logic truth tables. On the other hand, while rules can fill a truth table, they are not defined over binary attributes as occur in boolean logic. Finally the intrinsic nature of Smart Buildings rules is of being sparse and not completely defined over the whole domain of the attributes. This fact cannot be exploited, as it happens in common logic minimization, without

altering the desired behavior of the building.

These similarities and differences will be further described and analyzed in the following section where the steps, necessary to construct the proposed method for solving the stated problem, will be presented.

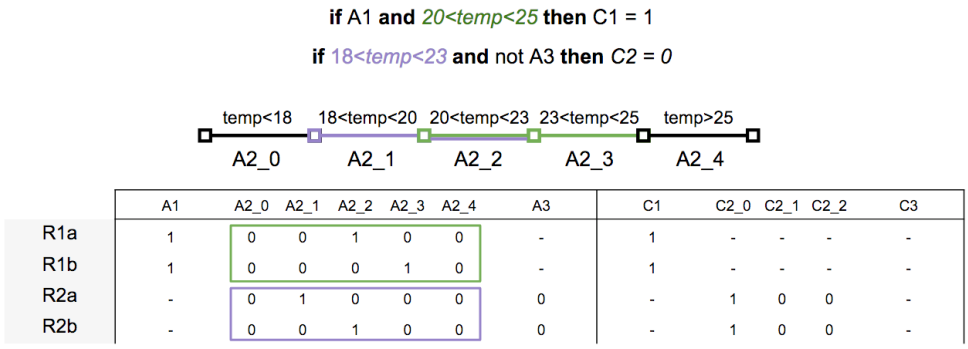
2.3 Proposed Method

As emerged from previous Section, Smart Buildings have many similarities with combinational logic circuits. Both application fields deal with function expressed in a rule-based fashion. Moreover, both logic circuits and Smart Buildings have limited resources whose usage has to be optimized. While in circuits these resources are typically represented by the number of gates and connections needed to implement a given boolean function, in Smart Buildings it is very important to have a minimum rule-set because the rules can interfere each other and, when they are too many, it can become overwhelming for users to properly understand the overall behavior of the building [75]. Furthermore, BuildingRules performs some analysis on the rule-set which take into account all the possible combination of these rules [33]. Thus a minimum rule-set could considerably speed up the performance of these analyses.

Nonetheless a parallelism between these two fields exists, there are anyway also some differences, which make the existing EDA algorithms unexploitable *as-are*. First of all, rules in Smart Buildings are usually defined over discrete attributes rather than binary ones as it happen in boolean logic circuits. This very first difference is really important since it requires a binarization phase of the rules attributes in order to take advantage of the existing algorithms.

Don't cares management represent another very important example of differences between the two fields. *Don't cares* can generally occur for two reasons:

- *Not-completely-specified* functions;
- The combination of input conditions are *impossible* to occur.



	A1	A2_0	A2_1	A2_2	A2_3	A2_4	A3	C1	C2_0	C2_1	C2_2	C3
R1a	1	0	0	1	0	0	-	1	-	-	-	-
R1b	1	0	0	0	1	0	-	1	-	-	-	-
R2a	-	0	1	0	0	0	0	-	1	0	0	-
R2b	-	0	0	1	0	0	0	-	1	0	0	-

Figure 2.2: Binarization process to generate binary rules starting from ordinary rules defined over intervals. The temperature attribute is split in all the possible sub-intervals generated by the defined rules.

While, in logic circuit synthesis, they are treated in the same way, as it will be described forward in this section, they worth two specific procedures in Smart Building context. In a parallel way, existing minimization algorithms produce *completely-specified* resulting functions as output. This changes the semantics of the original input function but, since these algorithm can take advantage of both kind of *don't cares* it results in an acceptable solution. In Smart Buildings context, on the other hand, this assumption does not hold as well. In fact, due to the intrinsic nature of Smart Buildings ruleset which are sparse and *not-completely-specified* on all input combinations, this change in the semantics would produce inaccurate results. For this reason, *Don't cares* due to *not-completely-defined* functions should not be exploited for minimization purposes in Smart Building context. While inapplicable for minimization purposes, such *don't cares* find other applications in different analysis about the building behavior as will be described in Section 2.3.7.

In the remaining part of the section the general ideas that have been presented above, will be explained by means of an example while presenting a possible solution for the problem.

2.3.1 Binarization

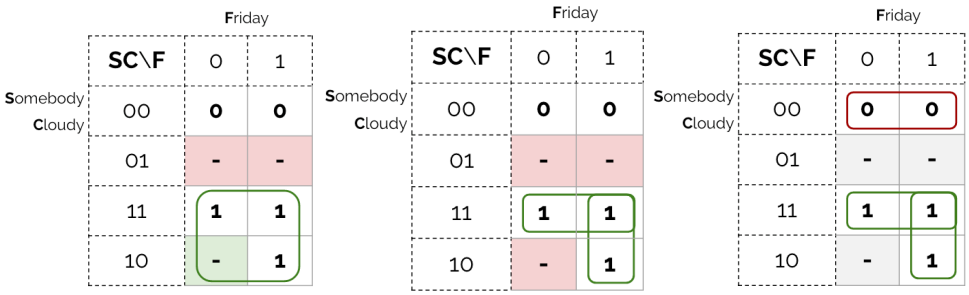
As previously stated, before actually starting the minimization phase, exploiting boolean logic algorithms, it is necessary to binarize the attributes that are not. Figure 2.2 shows how this process is conducted. It is shown how an *interval attribute* is split into many *binary attributes*. For each interval covered by an existing rule, a new rule is generated with the corresponding new synthesized binary attribute.

2.3.2 Karnaugh Map construction

The construction of the Karnaugh map is left as is in the original algorithm. As it can be seen in Figure 2.3, the Truth Table corresponding (in the *worst* case) to 5 rules:

- if $\overline{somebody}$ and $\overline{cloudy}$ and $\overline{friday}$ $\Longrightarrow$ LightOFF
- if $\overline{somebody}$ and $\overline{cloudy}$ and $friday$ $\Longrightarrow$ LightOFF
- if $somebody$ and $\overline{cloudy}$ and $friday$ $\Longrightarrow$ LightON
- if $somebody$ and $cloudy$ and $\overline{friday}$ $\Longrightarrow$ LightON
- if $somebody$ and $cloudy$ and $friday$ $\Longrightarrow$ LightON

is exploited to fill the Karnaugh Map.



(a) Karnaugh Algorithm **(b)** ON-Set only **(c)** ON-Set + OFF-Set

Figure 2.4: Comparison between: (a) Karnaugh Algorithm *as-is*, (b) Karnaugh using ON-Set only (i.e. without taking into account *Don't Cares*); (c) Two different execution of Karnaugh algorithm for ON-Set and OFF-Set.

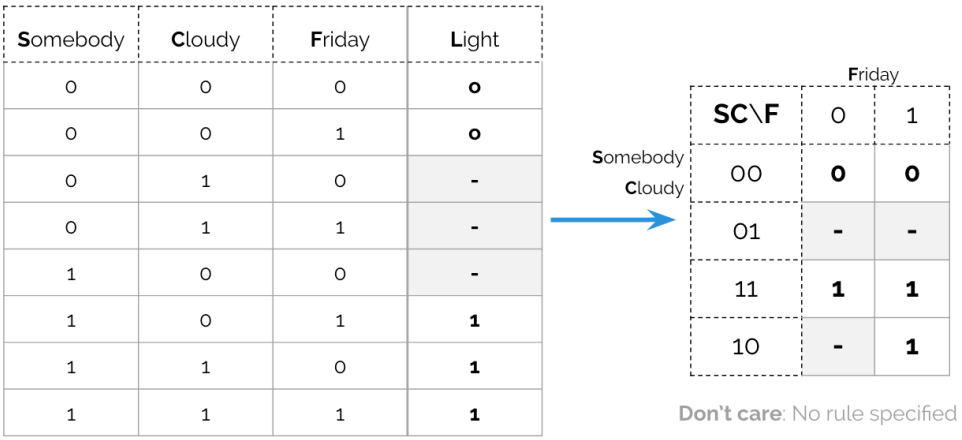


Figure 2.3: The translation from Truth table to Karnaugh Map happens as in the traditional algorithm.

2.3.3 Karnaugh Algorithm

If the Karnaugh mimization algorithm is applied as is to the previously described Karnaugh map, the input function will result in:

$$LightON \iff somebody$$

as depicted in Figure 2.4(a). This result drastically changes the semantic of the function originally described by the 5 rules.

It indeed adds the following rules to the given formulation:

- if $\overline{somebody}$ and $cloudy \implies$ LightOFF
- if $somebody$ and $\overline{cloudy}$ and $\overline{friday} \implies$ LightON

.

Karnaugh algorithm, as previously stated, makes use of *don't cares* for minimization purposes. The *don't cares* are responsible of the change in the semantics in Smart Building context and hence should not be taken into account.

2.3.4 Karnaugh with ON-Set only

If we minimize the function with a modified version of Karnaugh which does not exploit the *don't cares*, we will obtain as a result,

$$LightON$$
$$\Longleftrightarrow$$
$$(somebody \text{ and } friday) \text{ or } (somebody \text{ and } cloudy)$$

This result, shown in Figure 2.4(b), is also incorrect since it introduces some pieces of information that were not present in the original formulation. It indeed adds:

- if $\overline{somebody}$ and $cloudy \implies$ LightOFF
- if $somebody$ and $\overline{cloudy}$ and $\overline{friday} \implies$ LightOFF

.

2.3.5 Double Karnaugh on ON-Set and OFF-set

In order to actually obtain the desired behavior coming from the 5 rules formulation, we propose to execute the modified version (i.e. not

taking into account *don't cares*) of the Karnaugh algorithm twice. The first execution is done to obtain the *ON-Set* formula and the second for the *OFF-set*.

As shown in Figure 2.4(c), this method expands only the minterms that do not contain *illegal don't cares*, producing as a solution:

- if *somebody* and $friday \implies$ LightON
- if *somebody* and *cloudy* $\implies$ LightON
- if $\overline{somebody}$ and $\overline{cloudy} \implies$ LightOFF.

This solution is a minimized version of the initial formulation since it contains 3 rules rather than the initial 5. Moreover it does not change the semantic of the initial formulation, thus leaving the resulting building behavior unaffected.

2.3.6 Exploitable Don't cares

As previously stated, there exist two different kinds of *don't cares*. As we have seen in the previous example, *don't cares* coming from *not-completely-specified* functions are not exploitable for minimization purposes. On the other hand *don't cares* due to impossible input conditions do are.

In figure 2.5, an (incomplete) truth table is shown containing rules defined on mutually exclusive attributes (Cloudy, Sunny, Windy). In the table there are three rules:

- if *cloudy* $\implies$ LightON
- if *sunny* $\implies$ LightON
- if *windy* and $\overline{friday} \implies$ LightOFF

It is important to notice that the mutual exclusion is modeled by putting 0 in the excluded attributes (e.g. Sunny=1 $\implies$ Cloudy=0, Windy=0).

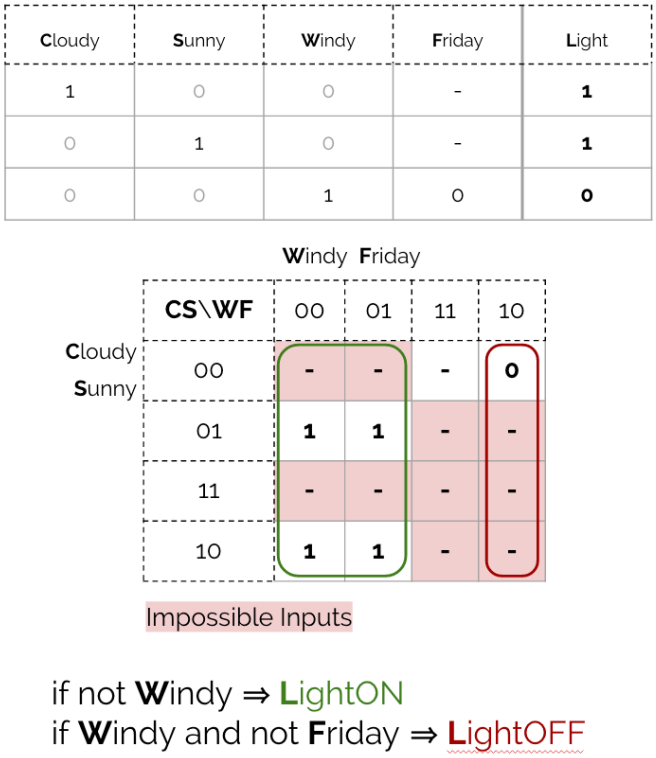


Figure 2.5: Example of minimization that can exploit *Don't cares* due to impossible combination of the input.

After that, we will execute our minimization algorithm, including *don't cares* which comes from impossible input combinations (red area in Figure 2.5.) The proposed algorithm will takes into account these *don't cares* thus producing the minimum solution:

- if $\overline{windy} \implies$ LightON
- if $windy$ and $\overline{friday} \implies$ LightOFF

It is important to notice that without this modification to include *legal don't cares*, it would not have been possible to minimize the initial given formulation composed of 3 rules.

2.3.7 Information Extraction/Abstraction

Having a further look at Figure 2.4(a), it is possible to notice that, as previously stated, applying Karnaugh *as-is* the semantic of the rule-set would change. In that case only one rule would be produced being:

$$LightON \iff somebody$$

Even if not semantically consistent, this rule could hide some information about tenants preferences tendency. The reasons why a don't care exists inside the ON-set minterm (green square), could be that nobody inserted that rule *yet*. Since Smart Buildings are daily programmed by their occupants it can be likely that someone, sooner or later, will insert that rule, thus completing the ruleset and making Karnaugh minimization finally legit. In such scenario hence, even if the result of Karnaugh minimization is still unsuitable, it can be highlighted to the building manager suggesting that a potential pattern has been discovered. Building manager could hence conduct further analysis or manually insert the suggested missing rules in order to generalize the ruleset.

2.4 Implementation

The proposed method has been implemented inside BuildingRules system by developing two main components:

- Back-end module
- Front-end minimization tool.

The module has been designed as a RESTful HTTP/JSON web service, with **frontend** for the user interface, and **backend** which communicates with BuildingRules, retrieves rules from it, runs minimization algorithm and provides RESTful APIs for native mobile applications or building management applications. We have implemented this module in Python 2.7 using the Flask framework [61].

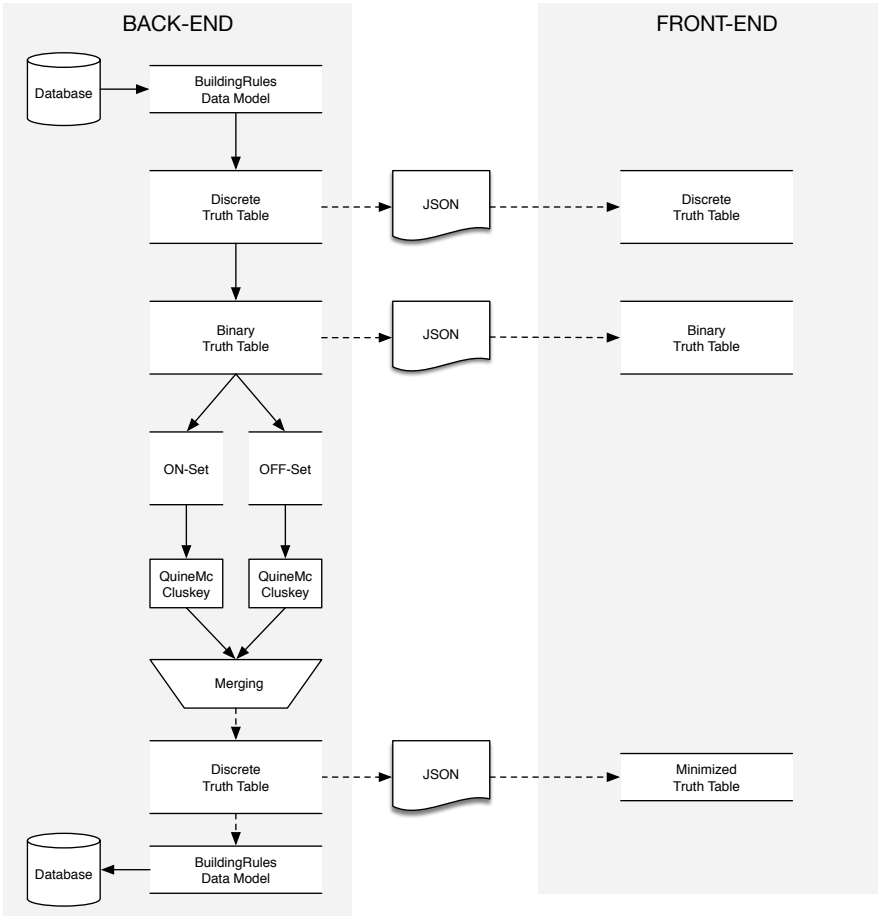


Figure 2.6: Architecture of Back-end and Front-end with main data models and processing flow.

The **backend** follows the Model-View-Controller (MVC) design pattern. The REST interface enables communication with the *frontend*, and the controller implements the application logic. The controller stores the data to manage buildings, rooms, and rules by using the model which works as a database abstraction layer. Data about rooms and rules is processed several times to feed different stage of the minimization algorithm.

In Figure 2.6 this processing flow is depicted. When the minimization routine is requested from a particular room, the back-end module requests the rules data of that room to BuildingRules.Data coming from the Database is retrieved leveraging BuildingRules abstraction layer. These

data models are converted into another data model which is suitable for being treated as a Truth Table. This data model is also exploited because it can be easily converted into a JSON format to be sent to external applications such as the *front-end* app. As expressed in Section 2.3, in order to leverage logic minimization techniques, it is necessary to convert the table into a binary truth table. The truth table is hence binarized as depicted in the figure. At this point the minterms that compose the ON-set and the OFF-set are isolated and sent to two different Quine-McCluskey minimization instances which produce an optimal solution each. The module that performs the minimization exploiting Quine-McCluskey algorithm has been developed on top of the Python optimization library *quine_mccluskey 0.2* [53]. The two solutions coming from such module are put together inside a new TruthTable data model that can be potentially sent to an external app via a JSON file. Finally all the new rules composing the minimized TruthTable are sent to BuildingRules which takes care of updating the database according to the optimization.

The **front-end** module has been developed as an external *web app*, exploiting the extensible, module-based programming paradigm of BuildingRules. As already stated it communicates with the *back-end* through a REST API, exchanging JSON messages. It consists in a web-application developed in *JavaScript* and *HTML*, also leveraging the power of *d3.js* visualization library [10]. Exploiting the power such visualization layer it is able to draw any kind of table according to the data it is fed with. In fact, *d3.js*, which stands for data driven document, permits to build interactive responsive applications that automatically react to changes in the data to fit them in the best way.

As depicted in Figure 2.7, the *front-end* module provides just a presentation layer that can be useful to Building Manager for analysis purposes. It is composed of two main parts. On the left side there is a list of buildings and rooms that can be selected to conduct analyses on. When a room is selected the truth table of that room is drawn on the main part of the screen. Some button can be exploited by the BuildingManager to launch the different stages of the minimization routine. It is indeed possible to

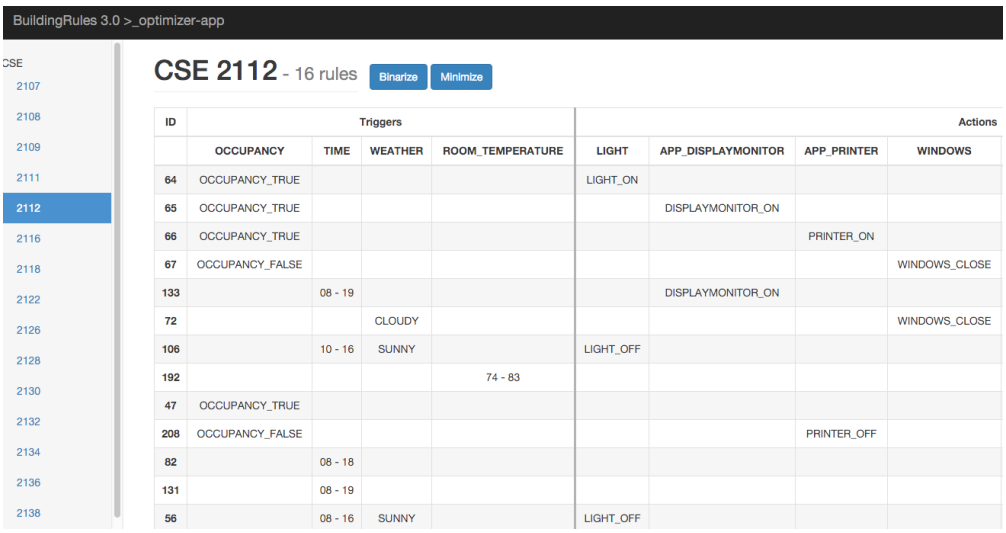


Figure 2.7: Screenshot of the Front-end webapp. As it can be noticed on the left the user (Building Manager) can choose the room to visualize the rules for. He can also Binarize and/or Minimize the given Truth Table.

Binarize the table, *Minimize* it and *Save* data in BuildingRules. The front-end can be hence used also as a console for conducting analyses and performing optimizations on specific rooms.

2.5 Results

To validate the proposed approach we ran a study involving 15 users who interacted with BuildingRules for a period of 17 days. We wanted to show that the proposed approach could actually improve the system usability by minimizing the set of rules to contain just the necessary rules that maintain the desired building behavior. During the study we collected data about users needs and preferences to populate the building ruleset. Such data have been fundamental in order to have a realistic set of rules for determining the effectiveness of the proposed approach. After having collected the data we applied the proposed method to evaluate its performances and discover how many rules it was possible to remove from each room. The results that we obtained, as it will be discussed in

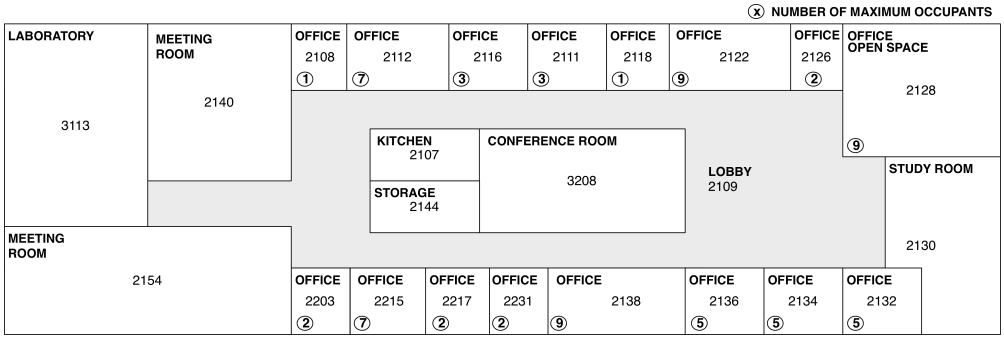


Figure 2.8: Virtual Office Environment planimetry

Section 2.5.3, showed that in almost every room it was possible to remove or aggregate rules in order to obtain a minimum ruleset.

2.5.1 Experimental Set

The experimental campaign that we conducted leveraged the building programming capabilities offered by BuildingRules. Installing BuildingRules in a real environment is not trivial though. Altough it is quite simple to create from scratch a testing environment able to manage a smart commercial building and installing an appropriate BMS, to make tests relevant, the dimension of the building, the number of occupants and the number of smart devices which are exploited need to be appropriate. Thus creating a realistic scenario for validation purposes is prohibitive in terms of costs. Installing BuildingRules in a real environment would have been also dangerous for safety reasons. It has never been tested to actuate in real environments, hence in case of dangers like fires or earthquakes it could risk occupants lives.

For these reasons, and since we did not have access to sensing and actuating devices in a real environment, in order to evaluate our work in a realistic setting, we decided to simulate such an environment through a virtual building designed to recreate the structure of a real commercial building. We chose a virtual setting because it is simpler to study and because it gave us the opportunity of focusing on the validation of our algorithms and methods rather than on the technologies used. Moreover

it was possible for us to include into the study different kind of *simulated* sensors that otherwise we would not have at our disposal.

Usually, a commercial building is composed of many rooms and each of them has a specific purpose. In addition to offices that are usually occupied by a limited amount of people such buildings are featured with shared spaces like meeting rooms, kitchen or conference rooms. As depicted in Figure 2.8, we designed the building planimetry to be as similar as possible as a real office environment. We included different kinds of rooms such as conference rooms, research laboratory, kitchen, storage rooms and offices of varying dimensions. A unique id has been assigned to each room and offices have been characterized by a number of maximum people that could house. These rooms have been endowed with components and applicances typical of real buildings. Computers, lights, printers, windows have been placed in offices and projectors and audio systems in meeting rooms.

2.5.2 Experimental Campaign

As said, the study involved 15 users among students, professors and researchers. Users have been chosen according their experience in working in shared office environments, in order to accurately simulate a real working environment similar to the target of smart buildings. In particular each user was required to have experienced at least one month of shared office life before joining the study. Not all the participants started at the same time, and we had a total of 15 users spread over 17 days. Each participant was assigned to a random set of rooms, for instance, an office, kitchen and conference room. This choice has been made to actually simulate a typical smart building environment where different users have different offices and areas which can exploit for their daily activities. Building inhabitants were given a short video tutorial and were also instructed to use BuildingRules for at least 10 minutes a day while pursuing common operations like inserting, updating and deleting rules (10 actions the first day, then decreasing each day, with an average of 5 actions per day). Also this choice reflects a typical scenario that can occur in smart

	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

Figure 2.9: Rule distribution with respect to triggers and actions.

environments. Users are in fact inclined to use more frequently the system in the very first days for configuration purposes. After that they mainly maintain or apply some slight modification to the existing rules.

2.5.3 Results Discussion

We obtained a total of 216 rules from this study, spread over 18 rooms and with an average of 12 rules for each room and 14 rules for user. Figure 2.9 depicts the rules distribution over the triggers and actions that it was possible to select. After having collected this data we launched the minimization algorithm that has been described in this Chapter. After its execution we observed good results in terms of number of rules that were removed. It is important to remind that the resulting ruleset coming out from our method leaves the building behavior unaltered.

Table 2.1 shows the number of rules produced by users for each room and the number of rules remaining after the minimization routine. From a total of 216 rules it has been possible to shrink the ruleset to contain only

Table 2.1: Results of the minimization algorithm for each room.

ROOM	BEFORE	AFTER
CSE 2107	30	22
CSE 2109	20	19
CSE 2111	4	2
CSE 2112	16	15
CSE 2116	9	9
CSE 2118	10	7
CSE 2122	5	5
CSE 2128	1	1
CSE 2130	22	19
CSE 2134	5	2
CSE 2136	2	2
CSE 2140	13	13
CSE 2144	19	14
CSE 2154	7	6
CSE 2203	4	4
CSE 2215	9	8
CSE 3113	22	15
CSE 3208	18	15
TOTAL	**216**	**178**
AVG	**12**	**9.88**

178 rules, obtaining a reduction of 21,3%. Moreover the average number of rules has been decreased from 12 to 9.88.

It is important to notice that our ruleset, even if collected while simulating a realistic work environment, is probably less complex and variegated with respect to a real one. Moreover we conducted an experimental campaign that lasted just one week. It is hence conceivable that the performance of our method will increase in real settings because of both the larger dimensions of the ruleset and the number of intersecting rules.

2.6 Conclusions

In this chapter, a method for minimizing the ruleset of a SmartBuilding has been presented. It has been developed starting from State of the

Art boolean logic minimization algorithms that have been firstly introduced in 1950s. Exploiting the parallelisms between boolean logic circuits and SmartBuildings, and taking into account also the differences between these two fields, a custom algorithm has been conceived. It has been then integrated in BuildingRules building management system in order to evaluate its effectiveness. Its performances have been shown during an experimental campaign.

Chapter 3

Conflict Resolution

In this Chapter, a method for solving conflicts among rules in Smart Buildings will be presented. It has been developed starting from State of the Art priority-based conflict resolution methods, by extending their power while solving some issues that have not been taken into account. Its soundness and effectiveness has been proved by integrating it into BuildingRules BMS and by conducting actual experiments in a real environment.

3.1 Introduction

Smart Buildings are inhabited by many different users. Each of them has different roles and habits inside the building. This diversity among individuals of the building can lead to conflicts that arise between users or groups of them. Conflicts occurrence is directly proportional to the number of users and Smart Buildings have an extended user-base which leads to a huge number of different needs that can be expressed by users. The conflicts that are generated prevent the building management system to properly satisfy the users thus producing an extended and generalized dissatisfaction.

Hence, finding a way to accommodate all users' needs and to solve conflicts in a *democratic* and *efficient* way is paramount. Current State of the Art conflict resolution methods (especially in building and home

automation context) are quite rudimentary and for this reason, a novel approach for overcoming the simplicity and effectiveness of those methods have been studied, as it will be described in Section 3.3.

The method that will be proposed, tries to overcome the effectiveness of rudimentary State of the Art solutions, in order to maximize the overall satisfaction of the users taking into account both their amount and their hierarchical positions inside the building.

In the next Section, the State of the Art techniques in conflict resolution are presented, especially in the context of Smart Buildings and multi-tenant automation scenarios. In Section 3.3 a complete description of the proposed approach is given with examples and considerations that has to be taken into account. Finally Section 3.4 contains a description of how the proposed system has been validated and discuss the results coming from the validation phase.

3.2 Related Works

Conflict Resolution is a problem that recurs in very different fields and with different implications. The thing that combines all those fields, though, is that conflicts usually arise in multi-user scenarios or more simply, when opposing parts can express contrasting policies. Although most BMSes properly handle single-user scenarios, it is quite natural that more than one user often coexists together with others in real life environments such as home, office etc. In these environments users often compete in an explicit or implicit way, particularly when they have access to shared resources.

Existing BMSes, already implement some kind of conflict resolutions functions. BOSS [25], for instance, defines every sensor write as a transaction thus providing a way to recover from conflicts that arise at sensor level. BACNet (Building Automation and Control Networks) protocol [14] is a widely adopted standard in industrial BMSes. It includes the support for a priority table that allow to solve basic conflicts for every writable sensor. Conflicts are resolved applying levels of priority to different appli-

cations according to their importance. Even more abstracted web-based BMSes like SensorAct [6] benefit from priority-based conflict resolution approaches which tackle conflicts by assigning a priority to each rule. BuildingDepot [74] offers priority tables like BACNet, but slightly extend this paradigm by requiring the admin to insert two default values for every actuation point. The first is a special low-priority default value in case no other actuation requests are triggered. The second is a conflict-default value that will be set in case two or more users with the same priority level conflict. This is usually the solution that is more conservative in the specific context (for instance, leaving the HVAC system on).

While the previously described solutions regard low-level BMSes and are usually conducted at sensor/actuator level, more advanced methods have been developed for a higher abstraction layers than the BMS one. Many conflict resolution strategies attempt to automatically solve conflicts without involving the user in the resolution process. Xu et al. [76] provide a framework for the automatical detection and resolution of *inconsistencies* for context-aware computing. GAIA's [60] main focus is to enable context-awareness in ubiquitous computing but conflict resolution is also treated exploiting priority tables. Even if it adopts a a widely used technique, some interesting future alternatives are proposed to overcome its simplicity. Its authors suggest to investigate for a way to find intersections between conflicting actions. They also propose to make use of Machine Learning to explore possible alternatives when two conflicting actions are mutually exclusive.

As it is possible to notice most of the solution so far presented takes advantage of priorities to solve conflicts. Even if priority-based approaches are quite effective and simple to implement, their limitations have been revealed in some works like [18]. Park et al. [49] propose an interesting alternative leveraging utility functions to get the policy between conflicting services. This function is computed as a weighted mean over values representing users preferences. They assume however that the value of all user preferences is given by users in an explicit way. Another interesting approach which leverages utility functions is defined by CARISMA [15].

It took inspiration from microeconomic sealed-bid auction mechanisms. It treats the rule-system as an *economy*, where subjects compete to obtain the control over the shared resources. Each one make its bids according to its belongings. After the bids phase one or more winner subjects are eventually declared computing a utility function over the intersection of the subjects outputs. It is important to notice that if the intersection is empty, there is no way to resolve the conflict in a proper way.

A different approach is exploited in the context of logic policy definition and conflict resolution. A policy is a rule that enables the execution of an action when an event takes place. It can be expressed as an event-condition-action (ECA) rule. There exists many different languages for policies definition such as Ponder [23] and PDL [17]. Policies description languages generally have built-in conflict detection and resolution mechanisms. Ponder is a declarative, object-oriented language defined for the specification of security and management policies. It allows to insert both primitive and composite ECA-rules. In a similar way, PDL allows the definition of ECA-rules that can also be translated in Datalog. Both languages are endowed with priority and grouping methods to specify which group the policies are addressed to and with which priority. Of particular interests are the conflict resolution mechanisms that are based on *meta-policies*, i.e. ad-hoc strategies that can be defined to solve conflicts between user-defined policies.

Another interesting method which makes an extensive use of Machine Learning is described by Zhao et al. [78, 79]. It consists in finding a way to converge to the optimal user thermal comfort by feeding machine learning algorithms with thermal compliant provided by the same users. It makes use of one-class classifiers to build thermal (and humidity) regions according to users complaints. Once a common thermal comfort zone is identified, the HVAC system is brought to the minimum-energy-consumption point inside of it.

BuildingRules [44], which is the Building Management System that has been chosen to develop our method upon, already implements a basic conflict resolution based on State of the Art priority tables. The current

implementation of BuildingRules forces the user, when she inserts a new rule, to choose a priority value to assign to the new rule. This value ranges in a scale from 0 to 100 and two rules with the same kind of consequent (e.g. HVAC, lights, windows...) and the same priority level cannot be defined for the same room. Furthermore, the priority value that the user chooses, lays inside the user priority range. As previously depicted in Figure 1.5, there are four different kind of *users* which BuildingRules interact with and they are related to four different priority levels. Building Manager is the subject who has the highest level of priority. This is due to the fact that he has to insert maintenance rules to properly handle the main building functions. Applications are external sofwares that can interact with BuildingRules via RESTful APIs. Human Users belong to the third level of the priority hierarchy. Finally the Default level is referred to rules which are triggered when no other rules have been defined.

In this section, the current State of the Art in conflict resolution has been presented, underlining the strong and weak points of each approach described. The method that we propose, makes use of this knowledge in order to provide an effective and simple conflict resolution mechanism. It extends the power provided by priority-based approaches and shares some features with the aforementioned methods that make use of utility functions for the definition of user satisfaction in the building. The method that has been developed, finally, tries to overcome the limits which have been pointed out in all the previously described approaches.

3.3 Proposed Method

In this section the method that we developed for solving conflicts among rules in Smart Buildings is presented. As emerged from the previous section, most of the exiting systems make use of State of the Art priority-based approaches for this purpose. Even if these methods are quite effective in simple scenarios, their limitations appear rather evident in grand scale environments. The main concept behind priority-based approaches is that when a conflict is detected, only the rule with the highest

priority is activated. When few rules are conflicting this approach produces an acceptable solution but, if these rules are many, this method does not seem to be sufficient anymore. It is indeed restrictive to totally satisfy a user while not taking into account all the other competitors.

We propose a solution that changes the conflict resolution perspective, having as goal to maximize the overall satisfaction of the users inside a building. This is done taking in consideration also the intersection between two or more user preferences (when this is possible). In order to accomplish this task it is though required a reform of the existing rules priority system in BuildingRules.

3.3.1 Priority System Reform

As previously stated in Section 3.2, the current implementation of BuildingRules asks the user to insert a priority value for the rule in a range from 0 to 100. The selected value is directly assigned to the rule in the ruleset. This kind of approach yet reveals some issues. Firstly, the range that is provided to the user (0-100) is too wide to be actually usable and understandable for the user. Secondly, and maybe more importantly, it is that the current approach does not take in consideration the different levels the users can inherently have inside a building hierarchy. It is though needed a more user-friendly priority index to present to the user which also let her express her status degree inside the building organization.

For these reasons, we present a new model for assigning priorities to the rules. The final priority, that will be assigned to each rule, will depend on two main components:

User Priority

it represents the *importance* that the user has inside the building and the organization she belongs to. This value is indeed higher for higher positions inside the organization rank. Even if this is initialized to a certain value (which reflects the position of the individuals), it is not fixed over time. It can, in fact, vary depending on how good a user is in

programming the building in a democratic and not assertive way. This variations can be done both by relying on automatic procedures which aim at detecting bad (or good) behaviors in building programming; or they can be manually made by the Building Manager if she detect some strange behaviors.

Rule Priority

it is the value that the user choose to associate to the rule she is inserting. The range that is used is considerably tighter than before, allowing only 5 values in a range that consist of: Lowest, Low, Medium, High, Highest.

The aforementioned priorities are then combined to assign the priority that the rule will maintain inside the ruleset, according to the following formula:

$$Priority = \sum_{user=1}^{N} UserPriority_{user} \cdot RulePriority_{user} \tag{3.1}$$

It is important to notice the summation meaning. While in the current implementation of BuildingRules more than one user is prevented to insert the same rule, we remove this limitation enabling the possibility for users to express common preferences. When two or more users insert the same rule, the overall priority of the rule is indeed computed by summing the individual priorities of the users. In this way we promote users to find common solutions in order to increase the probability of being satisfied.

3.3.2 Satisfaction Model

Once defined the new rules priority system all the requirements for building the conflict resolution method are in place. The approach that we follow in this work, as previously stated, is to compute the satisfaction of the users of a Smart Buildings, trying to maximize the sum of all of them, also considering possible intersection between users preferences.

In this perspective, it is useful to determine two different actuation attributes that users can define while composing their rules:

Discrete attributes

this kind of attributes are the ones for which the user can just choose between a limited set o values. They are generally represented by boolean decision the user can take (e.g. light on/off, window open/closed...). They can also include attributes defined on wider non-boolean domains like, for instance, telling the drink machine to brew coffee, tea or milk. To properly handle these attributes, values in between the ones defined in the given set are not considered (it just does not make sense to turn the light on and off at the same time).

For these kind of attributes, the conflict resolution algorithm consists in a Weighted Majority Voting process which all the triggered rules take part in. In order to determine which action has to be actuated, the action which has the maximum value of aggregated satisfaction, weighted on rules priority, is selected. In a more formal way this process can be expressed by the following expression:

$$\arg\max_{a=action} \sum_{a=1}^{N_1} Priority(a) = \\ = \sum_{a=1}^{N_1} \sum_{user=1}^{N_2} UserPriority_{user}(a) \cdot RulePriority_{user}(a)$$

This kind of approach allows to take into consideration also how many people are satisfied by each action and their hierarchical level. In this way the outcome that satisfies the largest number of users with the highest hierarchical level is chosen.

Continuous attributes

this kind of attributes is referred to those which can exploited by user to define values or intervals on a continuous domain. Examples of continuous attributes are, among the others, temperature, humidity and window

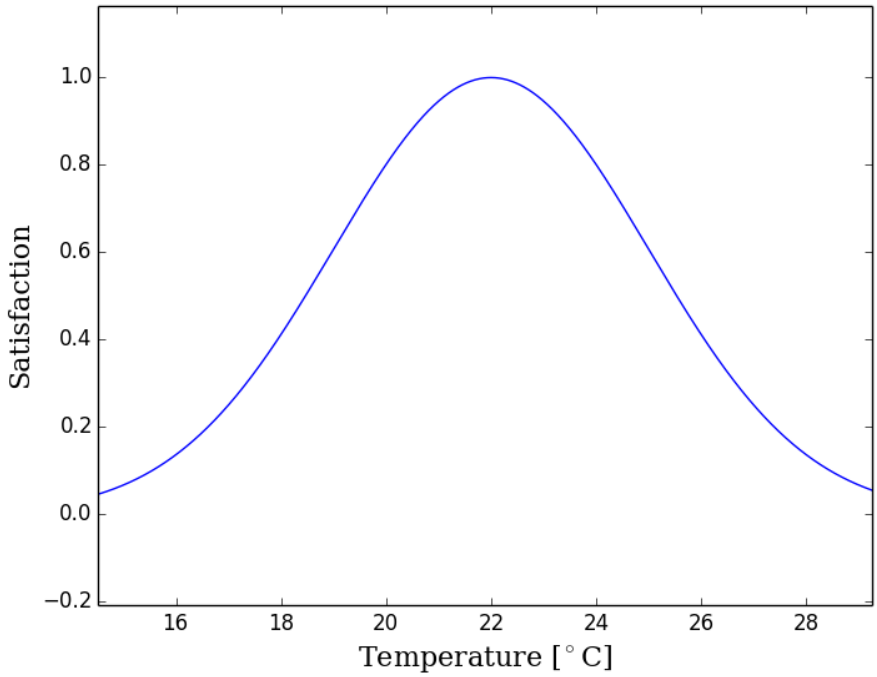
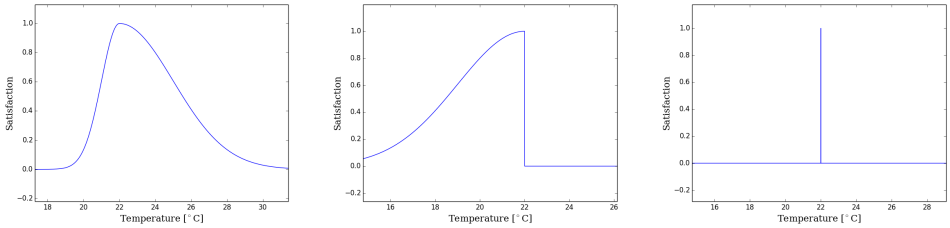


Figure 3.1: Satisfaction gaussian curve with $\mu = 22°\text{C}$ and $\sigma^2 = 3°\text{C}$

curtain opening level.

While handling this kind of attributes, it is possible to extend the Majority Voting method by considering in the conflict resolution computation, also values laying outside the ideal value defined by the user. In order to enable this kind of functioning, a way to model the user satisfaction over the considered domain is needed. For attributes like temperature, a gaussian-based model seems to be a realistic model. Moreover since current implementation of BuildingRules ask the user to insert an *ideal temperature* to set and a *tolerance range*, these two values can be used to build the gaussian satisfaction curve. The *bell* part of the gaussian is indeed built using the ideal value defined by the user as *mean* and the tolerance range as *variance* as it is depicted in Figure 3.1. This is done because the *bell* part of the gaussian ranges inside an acceptable range of values for the y-axis (satisfaction), while the tails of the gaussian are related to low values of satisfaction.



(a) Asymmetric interval **(b)** Asymmetric interval with zero tolerance **(c)** Symmetric zero interval

Figure 3.2: Three possible intervals that can be defined with the extended formulation. (a) is an asymmetric interval that can be defined with two different (left and right) tolerance ranges. (b) is an asymmetric interval with *right tolerance* set to 0. (c) is a symmetric interval with both tolerance set to 0.

This simple model can be easily extended to include more complex non-standard interval definitions. A user, for instance, could define an interval which is not symmetrical with respect to the ideal value and could also define an interval that starts (or ends) with the ideal value to set. This extensions can be obtained by simply attaching two different semi-gaussian curves together or working with the skewness parameter of the generalized skew normal distribution. Some examples of *non-standard* intervals are provided in Figure 3.2.

In an analogous way with respect to *discrete attributes* the optimal value of satisfaction is found looking for the value that maximize the overall utility function. This concept is expressed by the following formula:

$$\begin{aligned} &\arg\max_{a=action} \int_{a=1}^{N_1} Priority(a) = \\ &= \int_{a=1}^{N_1} \sum_{user=1}^{N_2} UserPriority_{user}(a) \cdot RulePriority_{user}(a) \end{aligned}$$

This approach allows to consider also values that are not optimal for a single user in order to compute the overall satisfaction of the user-base. As shown in Figure 3.3, the optimal value of temperature, for the depicted *two-users* system, lays in between the two users ideal values.

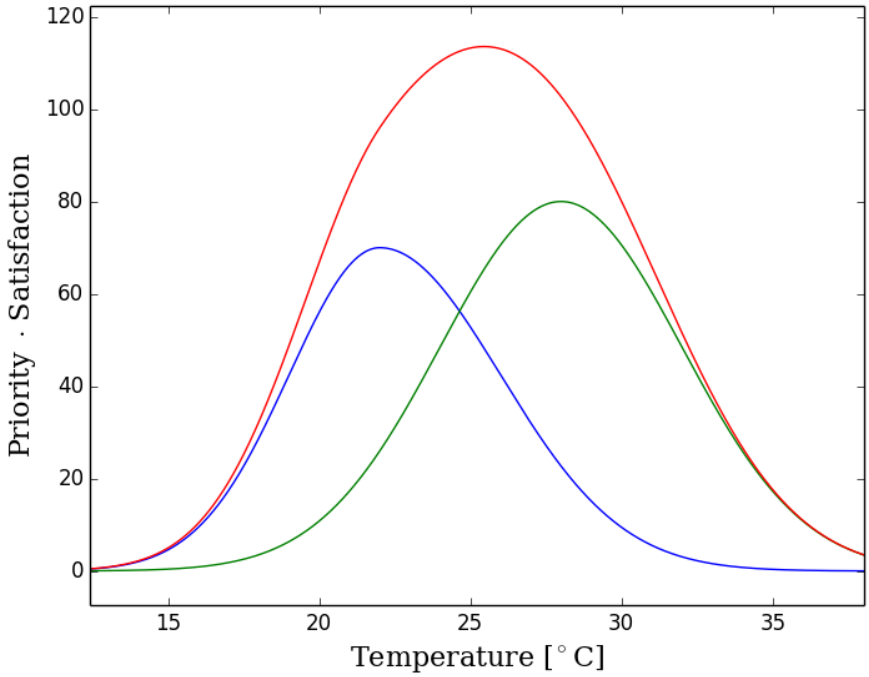


Figure 3.3: Two-users system with individual satisfaction curves (blue and green) and overall satisfaction curve (red).

Particularly interesting in Figure 3.3, is that the two users involved in the conflict resolution process have defined two different priority for the two rules. As it can be noticed, the green curve is slightly higher than the blue one. This is due to the priority value, that is a multiplying factor for the satisfaction curve, as shown in the above formula. The resulting overall optimal value, described by the red satisfaction curve, will be nearer to the higher priority user's optimal.

3.4 Results

During the previous chapters the proposed method and the corresponding implementation has been presented. We have discussed about the benefits of modeling the user satisfaction by means of a gaussian curve. It provides indeed a smooth and fine description of the user satisfaction

trend. Hence, while its effectiveness has been extensively previously described, it is really important to verify that users can actually exploit this kind of formalism to effectively describe their thermal preferences. As known thermal comfort is subject to several variables and people are not so able to determine their one in an effective way. For these reason we conducted a user study involving 30 users who interacted with BuidingRules for 10 days. During this period we collected data about user thermal preferences. Such data was then used to build our thermal satisfaction model and to actually set the environment temperature according to the preferences expressed by users. Feedbacks were eventually retrieved by users in order to refine the model and to perform further modification to the temperature set point. The results that we will discuss about in Section 3.4.3, showed that the overall satisfaction of the users was actually increased with respect to the standard actuation strategies.

3.4.1 Experimental Set

For an easy evaluation of the developed algorithm we decided to focus on the user thermal comfort asking users about temperature preferences and feedbacks about their comfort. The experimental campaign consisted in collecting information about temperature and user preferences inside the NECST Lab, located in the basement of the Department of Electronics, Information and Bioengineering building at Politecnico di Milano. The lab planimetry where we conducted these experiments is shown in Figure 3.4.

In order to conduct the aforementioned experiments we installed 6 temperature sensors to produce an accurate and precise map of the temperature distribution inside the laboratory. For the sake of simplicity, and in order to have a real-time monitoring capability of the temperature in the lab, we exploited *blukii SmartBeacons* [9]. They are Bluetooth Low Energy (BLE) enabled devices, that exploit the InfoBeacon proprietary protocol to advertise other devices in the nearby about their presence and provide them useful information like temperature. In order to capture such data we implemented a simple Android application that we installed

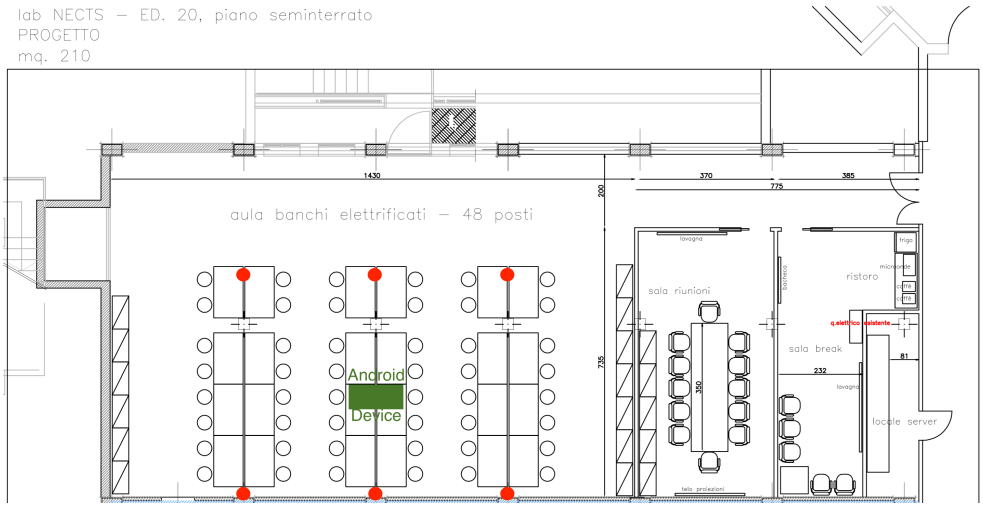


Figure 3.4: NECST Lab planimetry. Red circles represent the locations of the temperature sensors and the green square is the Android device used as gateway.

on a BLE enabled device, to turn it into a gateway between the sensor network and the database layer. The Android device, in fact, is responsible of constantly uploading the data coming from the sensors to a Parse web server [50] that we configured for this purpose. It also acts as a storage buffer since it is able to hold data coming from sensors even without data connection. Moreoever, a web application has been designed to provide a user-friendly interface which lets user insert preferences and provide thermal feedbacks to us. Figure 3.5 sums up the architecture of the system that has been built for conducting the experiments, providing a clarification of all the interconnections between the components.

3.4.2 Experimental Campaign

As previously stated the campaign lasted 10 days and involved 30 users. People have been asked to register into the system through the web application that we developed. After that, they have been required to insert a thermal profile, that consisted into defining an ideal temperature, and a tolerance range (in degrees) for lower temperatures as well as for higher temperatures. With these three pieces of data we built the thermal

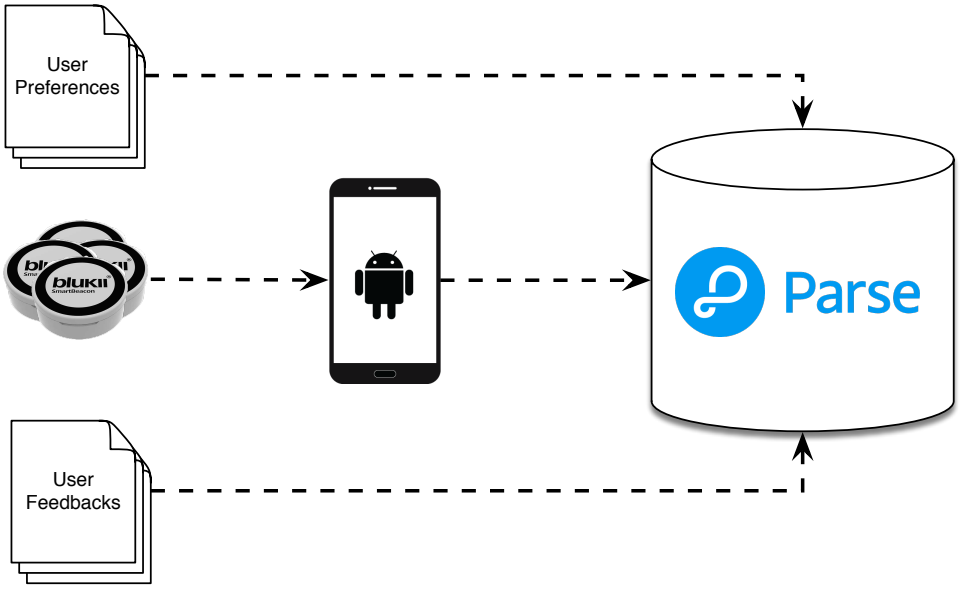


Figure 3.5: Experimental set architecture. User preferences and feedbacks are directly stored into the Parse database. Sensor data is retrieved by an Android device which acts as a gateway.

model for each of the users. At constant intervals the system defined the temperature set point in accordance with the proposed algorithm and with the preferences expressed by the users. In Figure 3.6 an instance of the conflict resolution algorithm is shown to having detected the temperature which maximizes the overall users satisfaction. At every change of the temperature set point, the temperature was changed in the thermostats of the laboratory accordingly.

Users were also encouraged to insert feedbacks about their thermal comfort, expressing a value in the following scale: [VERY COLD, COLD, COOL, GOOD, WARM, HOT, VERY HOT]. Through these feedbacks they had the opportunity of reactivating the conflict resolution procedure.

3.4.3 Results Discussion

During the aforementioned experiments we collected more than 36,000 temperature data and 129 users feedbacks. From this amount of data it is

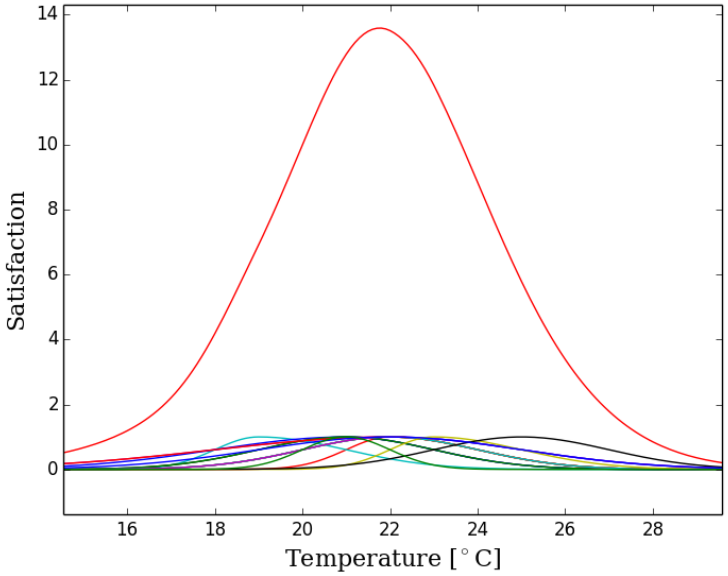


Figure 3.6: User preferences models (all little gaussian curves) and overall thermal satisfaction model (big red gaussian curve).

possible to draw some concluding results about the proposed method. During the experiments we applied two different actuation strategies in order to set the environment temperature: the standard policy that consisted in setting the temperature to the ideal value expressed by the user with the highest priority; and the proposed method that involves the modelization of user satisfaction and into determining the point of overall maximum satisfaction. Applying the standard strategy the feedbacks, that resided in the comfort area [COOL, GOOD, WARM], were just 58%. Looking at the feedbacks collected during the actuation of the proposed method instead, it is possible to notice that 87 feedbacks out of 105 feedbacks (83%) were in the comfort area. These results demonstrate the power of the proposed approach that allow to significantly increase the overall user satisfaction while maintaining the building energy efficiency.

Moreover, also the thermal satisfaction model has been validated comparing the data about the thermal model (built with the ideal temperature and the tolerance ranges) and the user feedbacks. In many cases the two

data models showed coherent results, demonstrating the accuracy of the proposed model, as depicted in Figure 3.7.

Nevertheless the good accuracy level and the effectiveness of the proposed method, some limitations have been discovered and has to be pointed out for future improvements. Firstly not all the users are comfortable to deal with temperature values. It is indeed not easy to determine the ideal temperature in absolute terms. The actual thermal comfort of a person also depends on other factors (external such humidity and airflow; or internal like metabolism and dressing) which can alter the perception of a given temperature. Another important consideration is that users, even if encouraged to provide feedbacks, are not naturally inclined to provide positive feedbacks because they are more likely to notice a discomfort rather than a comfort.

3.5 Conclusions and Future Works

In this chapter, a method for solving conflicts among rules in Smart Buildings has been presented. It has been developed starting from State of the Art priority-based conflict resolution methods, by extending their power while solving some issues that have not been taken into account. Its soundness and effectiveness has been proved by integrating it into BuildingRules BMS and by conducting actual experiments in a real environment.

In the immediate future we are planning to integrate the system also with other kind of user comfort models approaches. A particularly interesting one is represented by ThermoSense [**thermosense**], a work that aims at providing user thermal comfort estimations taking into account user complaints about temperature and humidity.

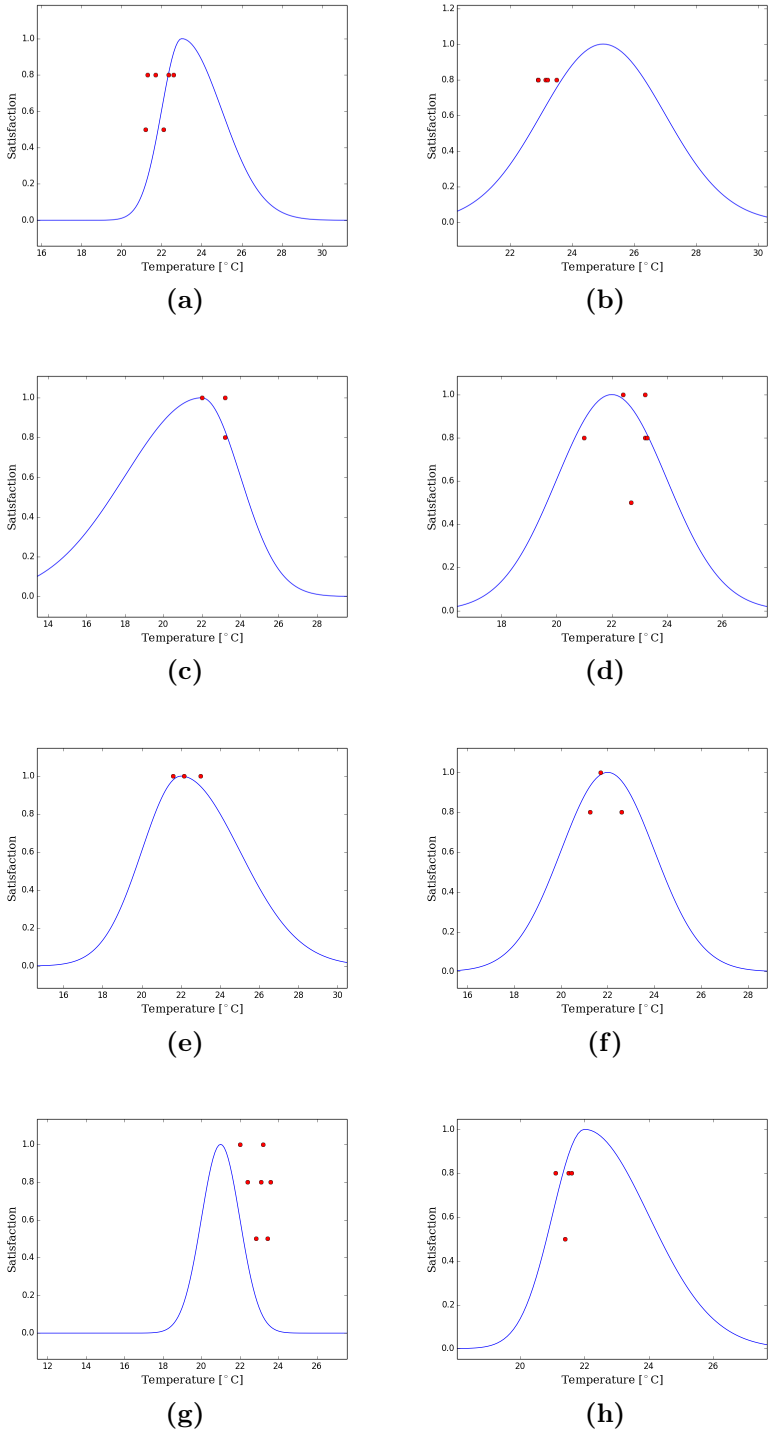


Figure 3.7: User satisfaction models (blue curve) compared to actual user feedbacks (red dots)

Chapter 4

Building Behavioral Analysis

In this Chapter we will present an approach for overcoming the technical difficulties that can be faced while trying to formally verify a Smart Building control system. In contrast with existing solution derived from the control theory world, we propose a stochastic method that copes with the probabilities that each rule, inside a ruleset, has to be triggered. This method will be presented in order to leverage the features and the capabilities of the specific scenario represented by Smart Buildings, in order to simplify Building managers duties in real world applications. The power of this method has been tested by applying it to previously collected ruleset. From the analyses it emerged that most of the building rooms contain sinks and assertive rules, symptom that a collaborative management approach is far to be applicable in real world applications with the current state of the research, but that can be facilitated with tools like those here presented.

4.1 Introduction

In this chapter the behavioral analysis of the building is analyzed. This topic is a direct consequence of the multi-tenant paradigm that is used in Smart Buildings and can be considered part of the work conducted to provide a way for solving conflicts between users. As stated in Chapter 3 conflicts occurrence is directly proportional to the number of users and

Smart Buildings have an extended user-base which leads to a huge number of different needs that can be expressed by users. Two different kinds of conflicts can be distinguished: *well-behaved conflicts* that are caused by the different needs that users can have and reflect a natural distribution of discordance among them; *pathological conflicts* that can be either caused by assertive users or derived by the collaborative programming paradigm that produces inconsistencies or ill-formed rulesets.

Hence, analyzing the building behavior to find undesired states is paramount to ensure the proper functioning of the control system thus producing, as final result, the satisfaction of the user-base that is involved in it. In Smart Buildings context there is not such kind of interest so far, but similar problems have been studied in other fields related to automatic control in the past years, as discussed in the following section. In this chapter however, we propose a novel technique that can be exploited in Smart Building management scenario that has, as its strength point, the creation simplicity as it will be described in Section 4.3.

In the next section, the State of the Art techniques in fields related to automatic control are presented. In Section 4.3 a complete description of the proposed approach is given with examples and considerations that has to be taken into account. Section 4.4 describes how it has been implemented inside the chosen building management platform. Finally Section 4.5 contains a description the experimental phase that has been conducted and discuss about the results the came out of it.

4.2 Related Works

In recent years, the interest in the analysis and design of systems which merge physical processes with information systems has grown. A meaningful example is represented by the control of physical systems performed by softwares inside cars, aircrafts, robots, etc. Dynamical systems, i.e. systems which show a direct time dependence defined through a function, have been studied by both computer scientists and control theorists, even if focusing on different models and properties. There is a class of models,

in the control world that is called "discrete-time linear systems" which provide a remarkably overlap between the two study fields. The two communities are interested in different topics which are stability and controllability for control theory and reachability, invariants and termination in computer science. In computer science this model is represented by unconditional *while* loops that are typical of automated control systems. While in conventional computing programs the termination problem is faced in the perspective of guaranteeing that a given program will terminate, in control world the opposite often occurs. Control software is designed to (ideally) never terminate. A control program, to perform its duties has to run *forever*, taking as inputs some controlled values and giving as output a resulting outcome for controlling the same values. Techniques to prove termination of programs have been extensively proposed in the latest years. It is pursued mainly by producing linear ranking functions [21, 54] also through effective heuristics [20, 24]. Tiwari [68] instead, investigates termination at a more basic theoretical level.

In recent years, computer scientist have started applying methods and techniques, which have been borrowed from system control theory, to analyze automated systems exploiting the mathematical knowledge that have been developed in that world. The control paradigm of physical processes supervised by digital programs has lead to the development of a theory of hybrid systems [30, 40, 71] combining discrete event systems with differential equations, in a way that is ideal for the modeling analysis and design of automated systems. The safety conditions, especially in real-time applications, are critically important and require the use of formal methods to guarantee that an unsafe region of the state space is not reachable from a set of initial conditions.

For this reason, reachability and safety are now being recognized as central problems in designing controllers. While their counter-parts in control theory, like stability and controllability are well studied, there is a general lack of results about computing the reachable set for hybrid systems. For purely discrete systems this problem has matured to the level of developing tools which perform either *model checking* or *automated theo-*

rem proving. Model checking [19] is a powerful technique for the automatic verification of systems: a model checking algorithm determines whether a system meets a requirement specification that is provided in the form of a temporal formula. For discrete finite-states systems model-checking has proven to be effective for validating communication protocols and hardware circuits, and even in purely-continuous state spaces model-checking algorithms have been extended to support their infinite cardinality. In hybrid systems, where also differential equations come into play, the crucial issue become *decidability* and proving that a model verifies a temporal formula is *undecidable* in general. Known classes of hybrid systems for which reachability problem has been proven to be decidable are timed automata [3], multirate automata [4] and some classes of linear hybrid systems [36, 37].

Another interesting research [29] regards the prevention, detection, recovery and avoidance of deadlocks exploiting three modeling formalisms which are digraphs, automata and Petri nets.

The landscape presented by computer science research field and supported by control theory background, involves a deep understanding of the mathematical models which are built on top of physical phenomena. They also require a wide technical expertise in the exploitation of formal methods. All of this requires, in turn, a huge effort that in Smart Building would lean on building manager shoulders. Even in previous works [33] on BuildingRules system, which is the BMS platform that we exploited to build our system upon, concepts from automata research field have been exploited. Behavioral graphs have been constructed in order to verify the building behavior by performing analysis on sink states and goal states reachability. Even if these analysis brought good results in terms of finding sink states, they have demonstrated to be onerous to perform and overwhelming for the final user.

For these reasons, we decided to face the sink state analysis and reachability problems from a different perspective. In the next section we provide, instead of formal automatic verification of the rule system, a stochastic method which deals with the state space exploration by means of proba-

bilities. This is done by exploiting Smart Buildings infrastructures and in order to relieve users from the onerous and too specific tasks represented by formal analyses.

4.3 Proposed Method

In this section, the method that we developed for performing behavioral analyses in Smart Buildings is presented. As emerged from the previous section, this problem has not been specifically tackled in Smart Buildings scenario but has been studied for a long time in automated control world. Existing solutions consider the use of formal verifications that imply the exploitation of formal methods and formalisms which, in turn, require time and expertise.

Smart Buildings control represent a particular scenario that is considerably different from conventional control systems. First of all Smart Buildings make use of a distributed collaborative (or adversarial in some cases) programming paradigm exploited by several users. Instead of designating a technician to program the overall building behavior, users are let programming the building thorugh the insertion of rules (in the IFTTT paradigm) that reflect their needs. This change in the paradigm makes it clear that this kind of environment is extremely more prone to programming errors and that the ruleset produced by users is more subject to contain inconsistencies. Moreover being inhabited by different people, conflicts can arise in the way people want the building to behave, and, malicious people can even perform attacks to the system in order to make their rules overcome others. For all these reason it appears evident the need of a method for tracking the building behavior, detecting unexpected events or even preventing them to happen. The expected behavior of a Smart Building should follow an unconditioned loop, as a normal control program, which makes the building explore different states in a quite fair way. If a building persists for a too extended amount of time inside a state, this could be a symptom of a ill-formed ruleset. Such a rule-set can be either generated by inexpert or bad-trained users, or by assertive ones

that try to overcome others. Whichever is the cause of a ill-formed ruleset, finding a solution for it, is paramount.

Exploiting automata jargon, states that endure for a long time are called *sink states*. In our application field (BuildingRules), rules antecedents truth values are checked every few minutes to make the building transitioning from a state to another one. Moreover, rules are grouped according to their consequents (i.e. the action associated to them). For this reason, conflicts can arise only among rules defined over the same consequent type. As an example, a rule for switching on the HVAC system can conflict only with rules for switching it off or for changing the temperature. Hence, it is quite unlikely that a building can be in a sink state which prevents it to change the state of all the controlled variables. Instead the focus has to be put in finding sink states that regard each controlled variables singularly, i.e. states in which a single actuator stops to change its status. From a deeper analysis on sink states in rule based systems two different kinds of sink states emerged to exist. The first type is represented by *proper sink states*: which are caused by an ill-formed rule set, e.g. a ruleset in which there is a rule for turning the light on but not for turning it off. In such a ruleset a light would remain on forever (if users cannot operate on it manually). Another situation that can be assimilated to sink state is represented by a case in which a building persists inside a state for a considerably long amount of time. This case will be called, from now on, *semi sink state* and can occur for example in a ruleset in which the only rule for switching off a light is triggered on Christmas day. In this scenario the light will remain on for the majority of the time.

With these premises, we developed a method for detecting *proper sink states* and to accurately estimate the probability that *semi sink states* occur in a given ruleset. While the problem regarding proper sink states is already solvable in an exact way through existing techniques described in Section 4.2, the latter analysis is performed applying a probabilistic method instead of conducting an exhaustive formal verification. This approach has been chosen because of its simplicity while compared to those based on formal methods and because information about probabilities can-

not be derived from existing methods. Existing approaches, indeed, can only infer the possibility that a given state could occur in the given model, without analyzing its actual feasibility. Moreover antecedents which rules are based on, are not uniformly distributed and hence it is not possible to draw definite predictions about their occurrence. Rules can be defined over random variables like weather conditions or temperature and hence a good estimation of their probability can be considered an acceptable approximation. Finally, given the intrinsic nature of Smart Buildings, all their sensors can give access to a huge amount of historical data to look for making accurate predictions on antecedents probability.

The main idea behind the proposed method is hence to estimate the probability that each rule has to be triggered in order to make predictions about the behavioral evolution of the building. This probability is computed starting from the probabilities of the antecedents that compose the rule, weighted according to the priority the rule is assigned. A rule can be defined over multiple triggers (antecedents), hence the probability of the rule is computed as the probability of the intersection of all those triggers. Since some of them can also depend each other, this probability is computed as conditional probability according to Bayes formula.

An example is here presented to make it clear. Given a rule that is defined as follows:

$$if\ A, B, C\ then\ D \tag{4.1}$$

the probability of this rule to be triggered is given by the following formula:

$$P_{rule} = P(A, B, C) = P(A \mid B, C)P(B \mid C)P(C) \tag{4.2}$$

recalling that if two events (for instance B and C) are not dependent each other:

$$P(B \mid C) = P(B) \tag{4.3}$$

Once computed this probability, it is multiplied for the priority which the rule is assigned by the user who inserted it, obtaining a value that represents the *influence* that the rule has inside the ruleset.

$$Influence = priority \cdot \prod_{i=1}^{N} P(A_i \mid A_{i+1,N}) \tag{4.4}$$

This metric gives an idea of how strong a rule is inside a ruleset and which is the frequency that it will be triggered in a given period of time. The importance of this metric resides in two benefits that it can produces:

- Highlighting (and eventually removing) rules with values of *influence* that are too high (assertive) or too low (useless).

- Providing a feedback to users, when inserting a new rule, about how much influential the new rule will be inside the system.

In the remaining part of the Section it is described how the influence metric is exploited to perform the two aforementioned analyses.

4.3.1 'Proper' Sink State Analysis

The research for *proper sink states* occur in a quite straightforward way. For each kind of consequent inside a ruleset a *reset rule* is searched. A *reset rule* is defined as a rule with an associated action that brings the related actuator state back to initial conditions. In most cases reset actions involve turning off the related appliances. If a reset action is found, it can be concluded that the given consequent cannot produce a sink state. This conclusion can be derived with certainty because the SMT solver (z3 [26]), which BuildingRules is built upon, already guarantees that there exists at least one model for which the rule holds true. In case no reset action is found instead, it can be derived that a sink state will be produced for the related consequent.

4.3.2 Semi Sink State Analysis

The search for *semi sink states* is pursued in a similar way. For every kind of consequent inside a ruleset, all the *reset rules* (if exist) are analyzed exploiting the influence metric that has been previously presented. If the influences of the reset actions are too low, it can be concluded that they will be rarely triggered, thus making the building persist in active states.

It is important to notice that this kind of analysis can be exploited also to signal rules (or group of rules) that are too assertive and that can prevent others to be triggered. Such rules can create situations very similar to sink states in which there is not a fair transition between rules and hence have to be limited. Finally this piece of information, can be also useful, for users who insert new rules, to have a better sense of the impact that the new rule will have on the final ruleset.

In this Section we have presented our proposed approach for overcoming the technical difficulties that can be faced while trying to formally verify a Smart Building control system. This method has been presented in order to leverage the features and the capabilities of the specific scenario represented by Smart Buildings, in order to simplify Building managers duties in real world applications. In the following Section, implementation details will be provided including the data models that have been exploited to design the probabilistic engine.

4.4 Implementation

In this section we provide a description of the proposed method implementation inside BuildingRules platform. As already mentioned, BuildingRules provides some interfaces to program the building at a high abstraction level and for this reason we focused on building a software module inside of it, capable of exploiting the features described in the previous section. As a side note, it has to be pointed out that in our experimental set, we did not have an actual building at our disposal and hence most of the data that we employed had to be obtained from different sources.

Table 4.1: Rules attributes and data sources employed for building the probabilistic method.

	Trigger	Domain Type	Source
1	External Temperature	Degree Range	Weather Underground [16, 73]
2	Weather Condition	[SUNNY, CLOUDY, RAINY]	N.A.
3	Occupancy	[TRUE, FALSE]	Probabilistic Model [63, 72]
4	Room Temperature	Degree Range	Data internally collected
5	Date	Timestamp Range	Not needed
6	Time	[0 ÷ 24]	Not needed
7	Weekday	[Sunday ÷ Saturday]	Not needed

In a Smart Building application instead, most of the data sources that we will go over in this section will be provided by the Smart Building infrastructures themselves.

The software that we developed in this work include two main components: a *backend* module which is responsible for the actuation of the aforementioned probabilistic method and a *frontend* application which connects to the backend module, retrieving the probabilistic values and providing a presentation layer to Building manager and users.

4.4.1 Back-end

This module has been built starting from the skeleton that was already present inside BuildingRules. We built our probabilistic model, exploiting the components and the attributes that were employed for previous applications of the system. Since, as already discussed, we did not have access to a real Smart Building we had to retrieve the data required for building the model from different sources. Table 4.1 provides a summary of the existing attributes which can be used to compose rules inside BuildingRules and the sources (if any) that we exploited to retrieve the data.

Here we analyze all the individual cases describing how the data models were built. The following models were built for providing estimations about the probability of the values inside each definition domain.

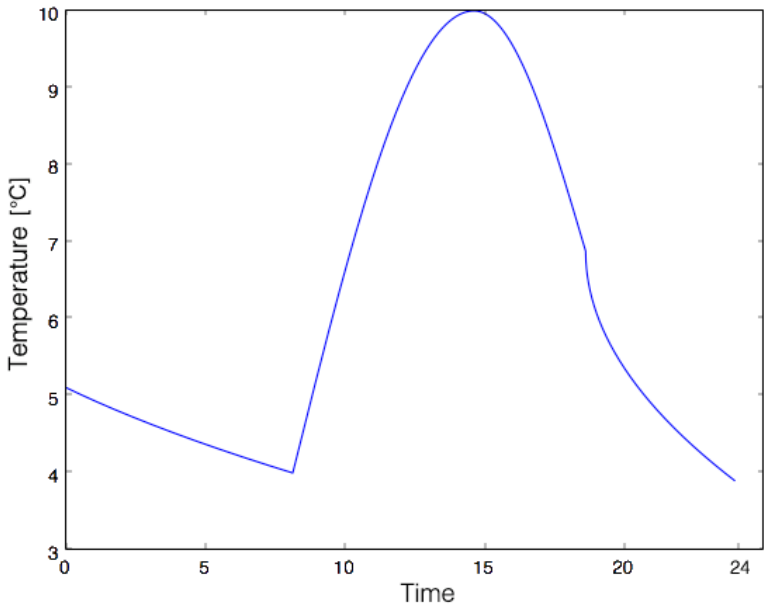


Figure 4.1: Temperature trend estimation model suggested by Cesaraccio et al [16]. It is modeled as two sinus functions (from sunrise to sunset) and a square-root function from sunset to sunrise of the following day.

External Temperature

To actually determine the probability of having a given temperature there should exist a model of the temperature variation of the place which the system is installed in. This kind of information is straightforward to be derived from historical data retrieved by Smart Building sensors. To face our lack of this data, and to cope with preliminary phases in which data is missing, we built a model based on data retrieved by WeatherUnderground [73]. This weather forecast service, provide historical data about maximum and minimum temperature for each day. We downloaded this data regarding the city of Milan (which the studies were conducted in) over a period of 10 years in the past. This dataset was further refined applying the studies from Cesaraccio et al. [16], which provide a model for the estimation of temperature trend in a day given the minimum and the maximum value of that day. This study suggests to model the temperature

trend as two sinus functions (from sunrise to sunset) and a square-root function from sunset to sunrise of the following day, as depicted in Figure 4.1.

This model is finally used to derive probability about temperature ranges with a good level of accuracy by looking in which days and for how long time the temperature has resided inside the given range. These probability can be also computed in a conditional way over time and dates simply filtering on the interesting periods of time.

Weather Condition

This attribute was not taken into consideration because no data was found to build model with. Also in this case it can be assumed that some sort of historical data can be stored by Smart Buildings in real world applications exploiting for instance paid features of weather forecast services.

Occupancy

The occupancy estimation model was directly taken by previous studies involving offices environments [63, 72]. These studies provide time dependent models for weekdays in commercial buildings. The model that has been derived from these studies is depicted in Figure 4.2. From this model it is also possible to derive conditional probabilities over time and weekday by means of filtering processes.

Room Temperature

This attribute is the only one that we had the opportunity to collect data about internally. We had collected, for a period of about 12 months, data about 4 different office rooms at two different heights (1m and 3.5m). All the data is referred to rooms that are inside the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) at Politecnico di Milano. Also in this case data can be filtered on time and dates in order to compute conditional probabilities.

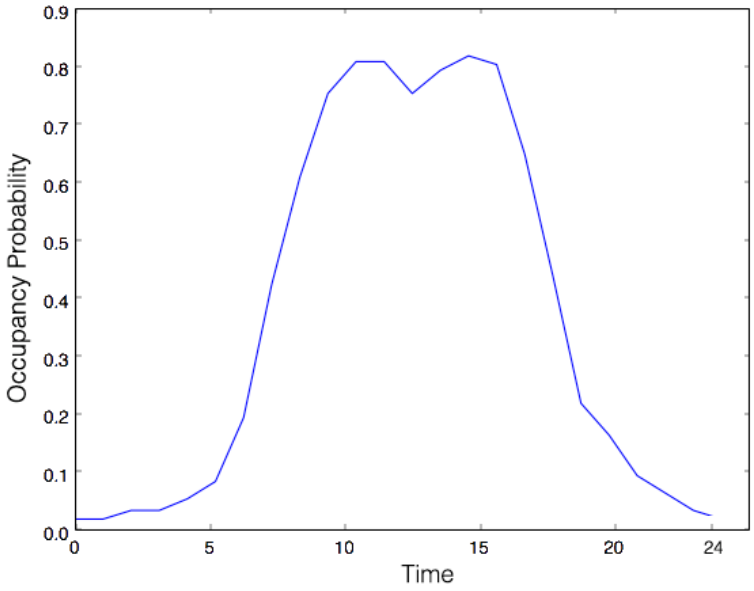


Figure 4.2: Occupancy model derived by [63, 72]. It shows how the probability of having people inside an office is distributed along the hours of a weekday.

Date, Time and Weekday

These three attributes did not need a data source since are uniformly distributed over time and are conditionally independent from all the other ones. Probability values were computed respectively as:

$$P(Date) = \frac{\#days}{365} \tag{4.5}$$

$$P(Time) = \frac{\#hours}{24} \tag{4.6}$$

$$P(date) = \frac{1}{24} \tag{4.7}$$

All the probabilistic dependencies are resumed in Table 4.2. As it is possible to notice, not all the dependencies have been exploited in this work. This is due to missing or uncorrelated data that could not be used for that purpose. Future works can head towards exploiting the missing dependencies in order to enrich the model increasing its accuracy.

Table 4.2: Rules attributes and dependencies used for building the probabilistic method. (✓implies a dependency between the corresponding attributes; **N.A.** shows a dependency that was not exploited due to missing data reasons; **-** implies no dependency)

	1. Ext_Temp	2. Weather	3. Occupancy	4. Room_Temp	5. Date	6. Time	7. Weekday
1. Ext_Temp	-	N.A.	N.A.	N.A.	✓	✓	✓
2. Weather		-	N.A.	-	N.A.	-	-
3. Occupancy			-	N.A.	N.A.	✓	✓
4. Room_Temp				-	✓	✓	N.A.
5. Date					-	-	-
6. Time						-	-
7. Weekday							-

4.4.2 Frontend

This module has been developed as an external app, exploiting the modular and extensible capabilities of BuildingRules. It consisted in two different applications developed in ordert to provide a presentation layer for two different reasons and users.

Tenant User Interface

This module already existed in BuildingRules platform before, and allowed users to see, insert and modify rules inside the Smart Building. We enriched the insertion and modification part by adding a gauge indicator that provides useful information to the user about the newly inserted rule. The gauge indicator, depicted in Figure 4.3, shows the influence value of the considered rule in a simple and user-friendly way, exploiting 5 different ranges and three colors. In this way users can have an immediate feedback about the strength of her rule without having to consider probabilities or overwhelming numbers.

Building Manager Console

This module has been developed from scratch and included inside a wider work for empowering BuildingRules features. This specific feature consists in a console that allows the Building Manager to perform some analysis on building rooms. For each room he can retrieve visualizations about assertive rules, useless rules, sink states and semi sink states in order

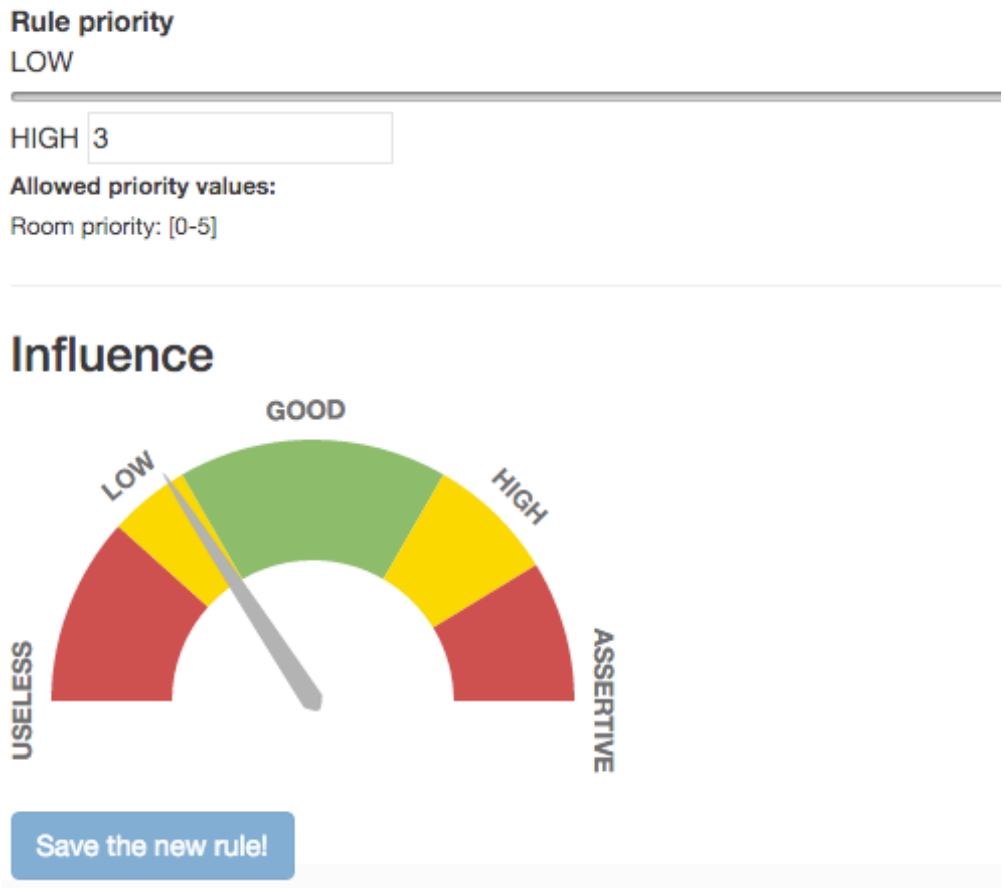


Figure 4.3: Gauge indicator in the bottom part of the Tenant User Interface for adding a new rule. It can be noticed that values outside the '*good*' range prevent users from inserting the given rule.

to properly understand the way each room is controlled by BuildingRules and tenants. This simple user interface is depicted in Figure 4.4.

In this section we presented our work and how we implemented the method proposed in the previous section inside BuildingRules BMS. In the following section a brief experimental setup will be presented with some results that have been collected.

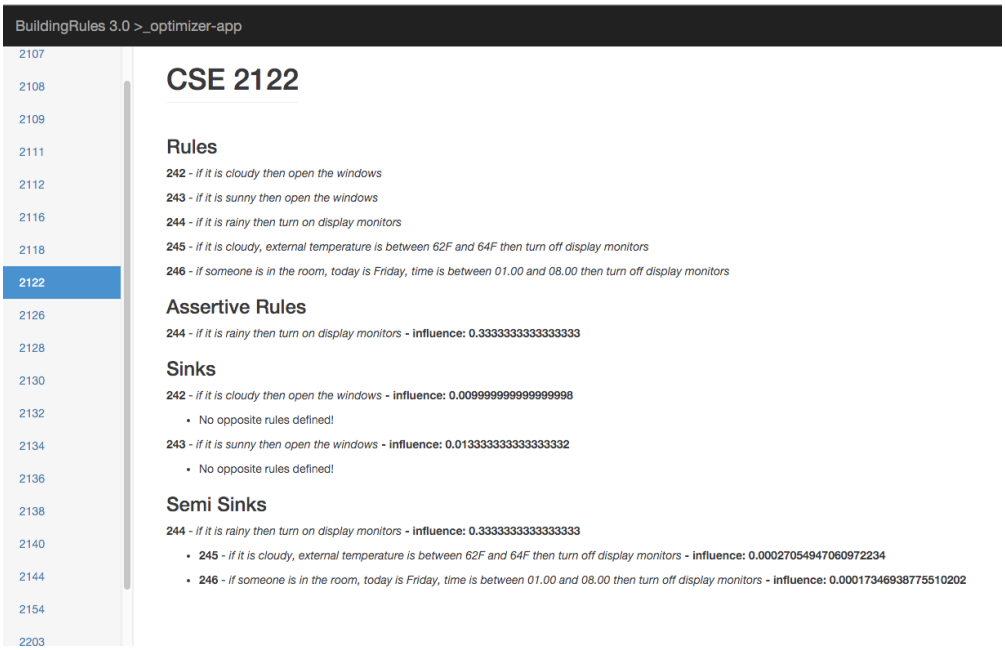


Figure 4.4: Screenshot of the Building Manager console. Building Manager can exploit it to look for assertive, rules, useless, rules, sinks and semi-sinks inside the building rooms.

4.5 Results

After having described both the proposed method and its actual implementation, we wanted to demonstrate their real effectiveness by measuring how probable it is to have sink states in collaborative programming scenarios like smart buildings.

The proposed method has been tested over some experimental data that was collected during an experimental campaign. These experiments lasted 17 days, during which users were asked to program the rooms that they were associated to, according to their preferences. They were authorized to program both office rooms (shared with at most two or three people) and shared rooms like kitchen or meeting room. During this campaign a total of 173 rules were produced.

As previously described, the proposed method consisted in two main parts: the probabilistic method that runs on the back-end of the system in order to evaluate the influence of each rule; and the front-end

application that prevents users from inserting assertive or ineffective rules. During these experiments we just applied the back-end feature in order to determine how, the effect of the prevention brought by the front-end component, could reduce the sink state occurrence in such a scenario. The results that will be described in Section 4.5.3, showed that the proposed approach could drastically reduce the number of problems derived by this collaborative programming paradigm, also providing a sort of educative effect on users.

4.5.1 Experimental Set

The experimental campaign is similar to the one described in Chapter 2. The description of the experimental set and campaign are reported below. The experimental campaign that we conducted leveraged the building programming capabilities offered by BuildingRules. Installing BuildingRules in a real environment is not trivial though. Altough it is quite simple to create from scratch a testing environment able to manage a smart commercial building and installing an appropriate BMS, to make tests relevant, the dimension of the building, the number of occupants and the number of smart devices which are exploited need to be appropriate. Thus creating a realistic scenario for validation purposes is prohibitive in terms of costs. Installing BuildingRules in a real environment would have been also dangerous for safety reasons. It has never been tested to actuate in real environments, hence in case of dangers like fires or earthquakes it could risk occupants lives.

For these reasons, and since we did not have access to sensing and actuating devices in a real environment, in order to evaluate our work in a realistic setting, we decided to simulate such an environment through a virtual building designed to recreate the structure of a real commercial building. We chose a virtual setting because it is simpler to study and because it gave us the opportunity of focusing on the validation of our algorithms and methods rather than on the technologies used. Moreover it was possible for us to include into the study different kind of *simulated*

sensors that otherwise we would not have at our disposal.

Usually, a commercial building is composed of many rooms and each of them has a specific purpose. In addition to offices that are usually occupied by a limited amount of people such buildings are featured with shared spaces like meeting rooms, kitchen or conference rooms. As depicted in Figure 2.8, we designed the building planimetry to be as similar as possible as a real office environment. We included different kinds of rooms such as conference rooms, research laboratory, kitchen, storage rooms and offices of varying dimensions. A unique id has been assigned to each room and offices have been characterized by a number of maximum people that could house. These rooms have been endowed with components and applicances typical of real buildings. Computers, lights, printers, windows have been placed in offices and projectors and audio systems in meeting rooms.

4.5.2 Experimental Campaign

As said, the study involved 15 users among students, professors and researchers. Users have been chosen according their experience in working in shared office environments, in order to accurately simulate a real working environment similar to the target of smart buildings. In particular each user was required to have experienced at least one month of shared office life before joining the study. Not all the participants started at the same time, and we had a total of 15 users spread over 17 days. Each participant was assigned to a random set of rooms, for instance, an office, kitchen and conference room. This choice has been made to actually simulate a typical smart building environment where different users have different offices and areas which can exploit for their daily activities. Building inhabitants were given a short video tutorial and were also instructed to use BuildingRules for at least 10 minutes a day while pursuing common operations like inserting, updating and deleting rules (10 actions the first day, then decreasing each day, with an average of 5 actions per day). Also this choice reflects a typical scenario that can occur in smart environments. Users are in fact inclined to use more frequently the system

in the very first days for configuration purposes. After that they mainly maintain or apply some slight modification to the existing rules.

4.5.3 Results discussion

As previously stated, during these experiments, users were free to insert whatever kind of rules they wanted. Hence they were able to insert both assertive and ineffective rules. Looking at the total of 173 rules more than half of them (54%) were declared as assertive. This is a signal that users tend to think at their needs without thinking in a "democratic" way. This can be attributed to the fact that users want their interests to be satisfied without considering others, or that they are not able to actually understanding the contribution of their rules. For such reasons the huge number of assertive rules could have been reduced by exploiting the front-end application that have been presented in the previous section, making the proposed method a very effective way to educate and lead users to act in a collaborative fashion. Collected data thus demonstrates the power of the proposed approach, that consists in preventing users to produce states in which all the actuators are not controlled anymore by the building control system.

It has to be stated though, that among the assertive rules, 29% of those rules were recognized as false positives, and hence they should not be prevented to be inserted by our method. This discovery has been very important because the vast majority of the false positives are represented by rules for the management of special cases (like night period or when nobody is occupying the room). This allows us to filter those special rules out in future updates of our system. Another interesting discovery has been that only 2 rules (about 1%) have been defined with a too low influence thus producing two semi sink states. This is perfectly in line with what has been previously asserted about people not thinking in a "democratic" way.

Finally 14 proper sink states have been detected in 7 rooms out of a total of 14. This is a very interesting fact that points out the limits of

IFTTT paradigm. For users, when inserting a new rule, it is not natural to think also to insert a rule that resets the status of the related actuator when the defined antecedents stop to hold. This represent a very important aspect that further development of this work, and Smart Building management applications, has to keep into consideration.

4.6 Conclusions and Future Works

In this chapter, a method for conducting behavioral analyses of a rule-based Smart Building has been proposed. It has been developed in contrast with existing solution derived from the control theory world. Instead of providing a formal verification tool we developed a stochastic method that copes with the probabilities that each rule, inside a ruleset, has to be triggered. This approach provide an approximation of the expected behavior of the building that represent an acceptable solution, also if considering that rules are often defined over random variables like weather conditions and temperatures. The biggest benefit that the proposed method produces is that it offers to building managers and users very simple and user-friendly interfaces to interact with, that does not imply formalisms and technical expertise.

The power of this method has been tested by applying it to a previously collected ruleset. From the analyses it emerged that most of the building rooms contain sinks and assertive rules, symptom that a collaborative management approach is far to be applicable in real world applications with the current state of the research, but that can be facilitated with tools like those here presented.

In the near future we are also planning to increase the accuracy of the model that has been built by replacing data estimated by existing models with actual data collected inside the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) at Politecnico di Milano. Finally actual test on a real Smart environment are forthcoming.

Bibliography

[1] Gregory D Abowd. "What next, ubicomp?: celebrating an intellectual disappearing act". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM. 2012, pp. 31–40 (cit. on p. 7).

[2] Yuvraj Agarwal et al. "Duty-cycling buildings aggressively: The next frontier in HVAC control". In: *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*. IEEE. 2011, pp. 246–257 (cit. on p. 3).

[3] Rajeev Alur and David L Dill. "A theory of timed automata". In: *Theoretical computer science* 126.2 (1994), pp. 183–235 (cit. on p. 68).

[4] Rajeev Alur et al. "The algorithmic analysis of hybrid systems". In: *Theoretical computer science* 138.1 (1995), pp. 3–34 (cit. on p. 68).

[5] S Massoud Amin and Bruce F Wollenberg. "Toward a smart grid: power delivery for the 21st century". In: *Power and Energy Magazine, IEEE* 3.5 (2005), pp. 34–41 (cit. on p. 3).

[6] Pandarasamy Arjunan et al. "SensorAct: a privacy and security aware federated middleware for building management". In: *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM. 2012, pp. 80–87 (cit. on pp. 6, 49).

[7] August. *August Smart Lock*. http://august.com. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).

[8] Christoffer Björkskog. "Human computer interaction in smart homes". In: *Helsinki, Finland* (2007), p. 1 (cit. on p. 7).

[9] *blukii Smart Beacons*. http://www.blukii.com/beacons_en.html. [Online; accessed 26/03/2016] (cit. on p. 58).

[10] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. "D3: Data-Driven Documents". In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011). URL: http://vis.stanford.edu/papers/d3 (cit. on p. 40).

[11] Robert King Brayton et al. *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984. ISBN: 0898381649 (cit. on p. 29).

[12] AJ Brush et al. "Home automation in the wild: challenges and opportunities". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 2115–2124 (cit. on p. 10).

[13] David Buchfuhrer and Christopher Umans. "The complexity of boolean formula minimization". In: *Journal of Computer and System Sciences* 77.1 (2011), pp. 142–153 (cit. on p. 26).

[14] Steven T Bushby. "BACnet TM: a standard communication infrastructure for intelligent buildings". In: *Automation in Construction* 6.5 (1997), pp. 529–540 (cit. on p. 48).

[15] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. "Carisma: Context-aware reflective middleware system for mobile applications". In: *Software Engineering, IEEE Transactions on* 29.10 (2003), pp. 929–945 (cit. on p. 49).

[16] Carla Cesaraccio et al. "An improved model for determining degree-day values from daily temperature data". In: *International Journal of Biometeorology* 45.4 (2001), pp. 161–169 (cit. on pp. 74, 75).

[17] Jan Chomicki, Jorge Lobo, and Shamim Naqvi. "A Logic Programming Approach to Con ict Resolution in Policy Management". In: () (cit. on p. 50).

[18] Jan Chomicki, Jorge Lobo, and Shamim Naqvi. "Conflict resolution using logic programming". In: *Knowledge and Data Engineering, IEEE Transactions on* 15.1 (2003), pp. 244–249 (cit. on p. 49).

[19] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999 (cit. on p. 68).

[20] Michael A Colón and Henny B Sipma. "Practical methods for proving program termination". In: *Computer Aided Verification*. Springer. 2002, pp. 442–454 (cit. on p. 67).

[21] Michael A Colóon and Henny B Sipma. "Synthesis of linear ranking functions". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2001, pp. 67–81 (cit. on p. 67).

[22] Johnson Controls. *Johnson Controls*. URL: http://www.johnsoncontrols.com/ (visited on 01/30/2016) (cit. on p. 3).

[23] Nicodemos Damianou et al. "The ponder policy specification language". In: *Policies for Distributed Systems and Networks*. Springer, 2001, pp. 18–38 (cit. on p. 50).

[24] Dennis Damsth, Rob GerthÜ, and Orna Grumbergß. "A heuristic for the automatic generation of ranking functions". In: (2000) (cit. on p. 67).

[25] Stephen Dawson-Haggerty et al. "BOSS: building operating system services". In: *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 2013, pp. 443–457 (cit. on pp. 7, 48).

[26] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340 (cit. on pp. 12, 17, 72).

[27] Anind K Dey et al. "iCAP: Interactive prototyping of context-aware applications". In: *Pervasive Computing*. Springer, 2006, pp. 254–271 (cit. on p. 10).

[28] *Energy at Home specifications*. http://ismb.github.io/jemma/resources.html (cit. on p. 7).

[29] Maria Pia Fanti and MengChu Zhou. "Deadlock control methods in automated manufacturing systems". In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 34.1 (2004), pp. 5–22 (cit. on p. 68).

[30] Thomas A Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. "HyTech: A model checker for hybrid systems". In: *Computer aided verification*. Springer. 1997, pp. 460–463 (cit. on p. 67).

[31] YL Huang and LT Fan. "A fuzzy-logic-based approach to building efficient fuzzy rule-based expert systems". In: *Computers & chemical engineering* 17.2 (1993), pp. 181–192 (cit. on p. 30).

[32] IFTTT. *IFTTT.com*. URL: https://www.ifttt.com/ (visited on 01/30/2016) (cit. on p. 10).

[33] Andrea Mariani Jacopo Fiorenza. "Improving Trigger-Action Programming in Smart Buildings through Suggestions Based on Behavioral Graphs Analysis". MA thesis. Politecnico di Milano, Sept. 2015 (cit. on pp. 12, 26, 31, 68).

[34] M Karnaugh. *The map method for synthesis of combinational logic circuits', Commun.* 1953 (cit. on p. 27).

[35] ICT Labs. *European Institute for Innovation and Technology ICT Labs*. http://www.eitictlabs.eu/innovation-areas/smart-spaces/. [Online; accessed 01/03/2016]. 2014 (cit. on p. 1).

[36] Gerardo Lafferriere, George J Pappas, and Sergio Yovine. "A new class of decidable hybrid systems". In: *Hybrid Systems: Computation and Control*. Springer, 1999, pp. 137–151 (cit. on p. 68).

[37] Gerardo Lafferriere, George J Pappas, and Sergio Yovine. "Symbolic reachability computation for families of linear vector fields". In: *Journal of Symbolic Computation* 32.3 (2001), pp. 231–253 (cit. on p. 68).

[38] Robert H Lasseter and Paolo Paigi. "Microgrid: a conceptual solution". In: *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*. Vol. 6. IEEE. 2004, pp. 4285–4290 (cit. on p. 3).

[39] Abhinandan Majumdar, David H Albonesi, and Pradip Bose. "Energy-aware meeting scheduling algorithms for smart buildings". In: *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM. 2012, pp. 161–168 (cit. on p. 3).

[40] Oded Maler. *Hybrid and Real-time Systems: International Workshop, HART'97, Grenoble, France, March 26-28, 1997, Proceedings*. Vol. 1201. Springer Science & Business Media, 1997 (cit. on p. 67).

[41] Edward J McCluskey. "Minimization of boolean functions*". In: *Bell system technical Journal* 35.6 (1956), pp. 1417–1444 (cit. on p. 28).

[42] Patrick C McGeer et al. "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions". In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* 1.4 (1993), pp. 432–440 (cit. on p. 29).

[43] Sarah Mennicken, Jo Vermeulen, and Elaine M Huang. "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. 2014, pp. 105–115 (cit. on p. 1).

[44] Alessandro A Nacci et al. "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 and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. ACM. 2015, pp. 381–384 (cit. on pp. 4, 7, 11, 50).

[45] Alessandro Antonio Nacci, Vincenzo Rana, and Donatella Sciuto. "A Perspective Vision on Complex Residential Building Management Systems". In: *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*. IEEE. 2014, pp. 209–214 (cit. on p. 20).

[46] Nest. *Nest Learning Thermostat*. https://nest.com/thermostat/meet-nest-thermostat/. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).

[47] Netatmo. *Netatmo Weather Station*. https://www.netatmo.com/product. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).

[48] openHAB. *openHAB, empowering the smart home*. www.openhab.org. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).

[49] Insuk Park, Dongman Lee, and Soon J Hyun. "A dynamic context-conflict management scheme for group-aware ubiquitous computing environments". In: *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*. Vol. 1. IEEE. 2005, pp. 359–364 (cit. on p. 49).

[50] *Parse*. https://parse.com/. [Online; accessed 26/03/2016] (cit. on p. 59).

[51] Philips. *Hue Lamp*. http://www2.meethue.com. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).

[52] MF Pichler et al. "Simulation-assisted building energy performance improvement using sensible control decisions". In: *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. ACM. 2011, pp. 61–66 (cit. on p. 3).

[53] Thomas Pircher. "An implementation of the Quine-McCluskey algorithm". In: *URL: https://pypi.python.org/pypi/quine_mccluskey(visited on 03/15/2016)* (2014) (cit. on p. 40).

[54] Andreas Podelski and Andrey Rybalchenko. "A complete method for the synthesis of linear ranking functions". In: *Verification, model checking, and abstract interpretation*. Springer. 2004, pp. 239–251 (cit. on p. 67).

[55] Alun D Preece. "A new approach to detecting missing knowledge in expert system rule bases". In: *International journal of man-machine studies* 38.4 (1993), pp. 661–688 (cit. on p. 30).

[56] Alun D Preece, Rajjan Shinghal, and Aida Batarekh. "Verifying expert systems: a logical framework and a practical tool". In: *Expert Systems with Applications* 5.3 (1992), pp. 421–436 (cit. on p. 30).

[57] Willard V Quine. "A way to simplify truth functions". In: *American mathematical monthly* (1955), pp. 627–631 (cit. on p. 28).

[58] Willard V Quine. "The problem of simplifying truth functions". In: *American mathematical monthly* (1952), pp. 521–531 (cit. on p. 28).

[59] Farrokh Rahimi and Ali Ipakchi. "Demand response as a market resource under the smart grid paradigm". In: *Smart Grid, IEEE Transactions on* 1.1 (2010), pp. 82–88 (cit. on p. 3).

[60] Anand Ranganathan and Roy H Campbell. "An infrastructure for context-awareness based on first order logic". In: *Personal and Ubiquitous Computing* 7.6 (2003), pp. 353–364 (cit. on p. 49).

[61] Armin Ronacher. "Welcome| Flask (A Python Microframework)". In: *URL: http://flask.pocoo.org/(visited on 02/02/2015)* (2010) (cit. on p. 38).

[62] Riccardo Rovatti, Roberto Guerrieri, and Giorgio Baccarani. "An enhanced two-level Boolean synthesis methodology for fuzzy rules minimization". In: *Fuzzy Systems, IEEE Transactions on* 3.3 (1995), pp. 288–299 (cit. on p. 29).

[63] Francis Rubinstein et al. "Analyzing occupancy profiles from a lighting controls field study". In: *Lawrence Berkeley National Laboratory* (2003) (cit. on pp. 74, 76, 77).

[64] Samsung. *Smart Things*. URL: https://www.smartthings.com/ (visited on 01/30/2016) (cit. on p. 9).

[65] Siemens. *Siemens Building Technology*. URL: http://www.buildingtechnologies.siemens.com (visited on 01/30/2016) (cit. on p. 3).

[66] Timothy Sohn and Anind Dey. "iCAP: an informal tool for interactive prototyping of context-aware applications". In: *CHI'03 extended abstracts on Human factors in computing systems*. ACM. 2003, pp. 974–975 (cit. on pp. 9, 11).

[67] Motoi Suwa, A Carlisle Scott, and Edward H Shortliffe. "An approach to verifying completeness and consistency in a rule-based expert system". In: *Ai Magazine* 3.4 (1982), p. 16 (cit. on pp. 26, 30).

[68] Ashish Tiwari. "Termination of linear programs". In: *Computer Aided Verification*. Springer. 2004, pp. 70–82 (cit. on p. 67).

[69] Khai N Truong, Elaine M Huang, and Gregory D Abowd. "CAMP: A magnetic poetry interface for end-user programming of capture applications for the home". In: *UbiComp 2004: Ubiquitous Computing*. Springer, 2004, pp. 143–160 (cit. on pp. 9, 11).

[70] Blase Ur et al. "Practical trigger-action programming in the smart home". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2014, pp. 803–812 (cit. on pp. 9–11).

[71] Frits W Vaandrager and Jan H Van Schuppen. *Hybrid Systems: Computation and Control: Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999 Proceedings*. Springer Science & Business Media, 1999 (cit. on p. 67).

[72] Danni Wang, Clifford C Federspiel, and Francis Rubinstein. "Modeling occupancy in single person offices". In: *Energy and buildings* 37.2 (2005), pp. 121–126 (cit. on pp. 74, 76, 77).

[73] WeatherUndergound. *wunderground.com*. URL: https://www.wunderground.com/ (visited on 02/17/2016) (cit. on pp. 74, 75).

[74] Thomas Weng, Anthony Nwokafor, and Yuvraj Agarwal. "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. 2013, pp. 1–8 (cit. on pp. 6, 7, 49).

[75] Jong-bum Woo and Youn-kyung Lim. "User experience in do-it-yourself-style smart homes". In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM. 2015, pp. 779–790 (cit. on pp. 26, 30, 31).

[76] Chang Xu and Shing-Chi Cheung. "Inconsistency detection and resolution for context-aware middleware support". In: *ACM SIGSOFT Software Engineering Notes*. Vol. 30. 5. ACM. 2005, pp. 336–345 (cit. on p. 49).

[77] Z-Wave. *Z-Wave*. http://www.z-wave.com. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).

[78] Qianchuan Zhao et al. "Experimental assessment of a satisfaction based thermal comfort control for a group of occupants". In: *Automation Science and Engineering (CASE), 2015 IEEE International Conference on*. IEEE. 2015, pp. 15–20 (cit. on p. 50).

[79] Qianchuan Zhao et al. "Preliminary study of learning individual thermal complaint behavior using one-class classifier for indoor environment control". In: *Building and Environment* 72 (2014), pp. 201–211 (cit. on p. 50).

[80] ZigBee. *ZigBee*. http://www.zigbee.org. [Online; accessed 01/03/2016]. 2016 (cit. on p. 5).