



Maastricht University

KNOWLEDGE IN ACTION

Faculteit Wetenschappen School voor Informatietechnologie

master in de informatica

Masterthesis

Mediating Conflicts in the Near future of IoT

Mathias Van Ginneken

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

dr. Davy VANACKEN

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.



UHASSELT

KNOWLEDGE IN ACTION

www.uhasselt.be

Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

2019
2020



Maastricht University

Faculteit Wetenschappen

School voor Informatietechnologie

master in de informatica

Masterthesis

Mediating Conflicts in the Near future of IoT

Mathias Van Ginneken

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

dr. Davy VANACKEN

Samenvatting

Introductie

Het Internet of Things groeit, en het doet dat niet alleen in populariteit. Elke dag worden er steeds meer en meer apparaten toegevoegd, en nog meer opportuniteiten verrijzen om omgevingen te creëren die de kwaliteit van ons leven verbeteren. Smart homes zijn een voorbeeld van hoe deze zogenaamde “dingen” een deel worden van ons dagelijkse leven. Hun doel is om onopvallend ons te helpen en te ondersteunen in alles wat we doen. Met andere woorden, ze horen ons leven eenvoudiger te maken. Door eindgebruikers in staat te stellen om hun eigen apparaten en hun gedrag te programmeren, wordt de controle van deze omgevingen volledig gegeven aan hen. Tools zoals IFTTT zijn dan ook speciaal ontworpen om dat programmeren voor deze gebruikers veel gemakkelijker te maken. Aan de hand van deze tools kan men regels maken die beschrijven welke acties uitgevoerd moeten worden wanneer een bepaalde gebeurtenis plaatsvindt of wanneer er aan een bepaalde conditie wordt voldaan. Dit brengt echter ook wel weer problemen met zich mee. Systemen die opgebouwd zijn uit zulke regels, die door eindgebruikers zijn samengesteld, kunnen al snel zeer complex worden wanneer deze hoeveelheid aan regels toeneemt. Dit zorgt er dan ook voor dat een overzicht van alle regels houden en het opvolgen van alles wat er in het systeem gebeurt een moeilijke taak wordt. Daarentegen introduceert het hebben van veel regels ook een grotere kans op het hebben van conflicten tussen regels. Om zulke conflicten te voorkomen in regel-gebaseerde systemen werden verscheidene tools ontwikkeld die de eindgebruikers zouden moeten helpen tijdens het creëren van nieuwe regels. Deze tools controleren, wanneer een nieuwe regel wordt gecreëerd, of deze mogelijks in een conflict zou kunnen komen met al reeds bestaande regels in het systeem. Als dit het geval zou zijn, dan wordt de gebruiker hierover gewaarschuwd en krijgt deze de mogelijkheid om de regels alsnog toe te voegen of onmiddellijk een oplossing te maken. Het oplossen van zo een conflict vereist echter wel dat eindgebruikers tijdens het aanpassen van regels rekening moeten houden dat het gewenste gedrag nog steeds bekomen kan worden in elke mogelijke situatie. Paterno zei hierover het volgende:

“It may be very difficult for developers to predict all possible context-dependant scenarios, because there may be (at design time) unforeseen requirements that need to be supported when the application is actually used.” [56]

Aangezien wij leven in heel dynamische en onvoorspelbare omgevingen, wordt het dus erg moeilijk om tijdens het creëren van regels rekening te houden met alle mo-

gelijke contextuele veranderingen die er zouden kunnen optreden. Daarnaast zijn conflicten ook erg complex. Ze kunnen voorkomen op apparaten of beleid, ze kunnen veroorzaakt worden door een enkele gebruiker of meerdere gebruikers, ze kunnen gedetecteerd worden tijdens de run-time of nog voor dat ze eigenlijk voorkomen en ze kunnen genegeerd of opgelost worden. Ondanks dat er vele tools bestaan, focussen de meeste van hen op het detecteren en oplossen van conflicten tijdens het creëren van nieuwe regels. Door gebruik te maken van verschillende visualisatie methoden kunnen gebruikers de conflicten beter begrijpen en zelfs onderzoeken waarom een regel al dan niet in een conflict is verwickeld. Wanneer conflict detectie gebeurt tijdens de run-time, worden eindgebruikers meestal uitgesloten uit het hele detectie- en oplossingsproces aangezien deze systemen automatisch een oplossing genereren wanneer een conflict zich voordoet. Wanneer conflicten automatisch opgelost worden, maakt het systeem meestal gebruik van prioriteiten om te bepalen welke regels al dan niet uitgevoerd mogen worden. Deze prioriteiten moeten echter wel op voorhand al gedefinieerd en in het systeem gebracht worden.

Hieruit volgt dus dat het grootste deel van conflicten oplossen al reeds moet gebeuren tijdens het opstellen van de regels, of het nu is door extra regels en condities toe te voegen of door prioriteiten te definiëren. Eenmaal de regels opgeslagen en toegepast worden in het systeem, hebben gebruikers geen enkele controle meer over conflicten wanneer deze zich voordoen. In dit werk onderzochten wij een alternatieve en aanvullende aanpak voor de reeds bestaande tools. In plaats van conflicten te detecteren tijdens het creëren van nieuwe regels en een gebruiker deze onmiddellijk te laten oplossen, stellen wij een oplossing voor waarbij in realtime eindgebruikers verwittigd worden wanneer conflicten zich binnenkort zullen voordoen en wordt de eindgebruiker de mogelijkheid aangeboden om deze in-situ op te lossen aan de hand van de huidige situatie. Hierdoor kan men eenmalige uitzonderingen maken op regels door tijdelijk acties te snoozen of custom acties te maken die het nabije conflict zouden oplossen. Om dit alles te verwezenlijken voorspellen we de statussen van de apparaten in de nabije toekomst, detecteren we of er daarin mogelijke conflicten voorkomen en presenteren we dit aan de gebruiker aan de hand van de uCoRe applicatie. Met gebruik van deze applicatie kan het conflict dan opgelost worden, wat er dan toe zal leiden dat de gewenste status van het apparaat op dat moment gezet zal worden.

Contributies

In dit werk worden volgende contributies gemaakt:

- Conflicten detecteren en oplossen at run-time in een IoT smart home omgeving.
- Oplossingen worden verkregen door acties tijdelijk te activeren of te snoozen of door custom acties te definiëren.
- Een applicatie geeft de controle over de conflict oplossingen in de handen van de eindgebruiker die in-situ beslissingen kan maken.
- Een studie die gebruiksvriendelijkheid en succes in het oplossen van conflicten evalueert voor de applicatie.

uCoRe App: conflicten detecteren en oplossen

Dit werk bestaat uit vier grote delen: het voorspellen van de nabije toekomst, conflict detectie, conflicten oplossen en een applicatie dat dit alles samenbrengt in de handen van de eindgebruiker. Veranderingen van apparaatstatus worden voorspeld via het FORTNIoT framework en binnen dit framework hebben wij de functionaliteit uitgebreid zodat het ook in staat is om conflicten te detecteren en op te lossen tijdens het voorspellen van de statusveranderingen van apparaten in de nabije toekomst.

Tijdens het voorspellen wordt er een voorspellingsboom gemaakt die de nabije toekomst voorstelt. Elke knoop in de boom representeert een statusverandering en elke boog representeert een gevalideerde en uitgevoerde regel. Bij het opstellen van deze boom wordt conflict detectie meermaals toegepast door, elke keer wanneer nieuwe knopen gevonden worden, de boom af te vlakken en de acties te markeren die dezelfde apparaten rechtstreeks beïnvloeden. Eenmaal conflicten gevonden worden zal er in de bestaande conflictoplossingen gezocht worden naar een oplossing wiens set van conflicterende acties een exacte match vormt met de huidig gevonden set van conflicterende acties. Wanneer zo een oplossing gevonden wordt, zal dit resulteren in custom acties die toegevoegd worden aan de boom alsook acties die gemarkeerd zullen worden als “snoozed”.

De voorspelling van statusveranderingen hoort correct te verlopen en een goede representatie te zijn van de mogelijke nabije toekomst. Dit betekent dat in de voorspelling enkel statusveranderingen mogen opgenomen worden waarvan het systeem zeker is dat deze statussen zullen worden toegepast gegeven de huidige informatie. Statussen die betrokken zijn in een conflict zouden bijgevolg er niet voor mogen zorgen dat andere regels getriggerd worden, wat zou resulteren in nieuwe statussen. Als dit wel het geval zou zijn, zou dit betekenen dat de representatie van de nabije toekomst totaal zou kunnen veranderen eenmaal het conflict wordt opgelost. Het is om deze reden dat de conflict detectie niet post-voorspelling kan gebeuren, maar tijdens de voorspellingsfase moet worden gedaan. Zodra conflicten worden gedetecteerd, worden deze gelogd en de betrokken acties worden gemarkeerd als conflicterend, waarna die voorspelling opnieuw zal moeten starten.

In plaats van regels aan te passen door bijvoorbeeld condities toe te voegen, hebben wij ervoor gekozen om te werken met de acties van de regels om zo oplossingen voor conflicten te bekomen. Dit stelt ons meer in staat om flexibel om te gaan met de regels en de uiteindelijke wensen van de eindgebruiker. Het is bijgevolg perfect mogelijk dat regels nog steeds getriggerd worden, maar in plaats van dat alle acties uitgevoerd worden zoals de regel definieert, kan er geselecteerd worden welke acties de gebruiker precies uitgevoerd wil hebben en welke niet. Dit is mogelijk doordat acties gemarkeerd kunnen worden als actief of snoozed, wat er voor zal zorgen dat tijdens de voorspellingsfase, de voorspellings-engine statussen zal kunnen activeren of negeren wanneer de actie die die status genereert gemarkeerd is. Oplossingen voor conflicten moeten ook toegepast worden tijdens de voorspellingsfase omdat statussen dat geactiveerd worden door toegepaste oplossingen, op hun beurt weer regels kunnen triggeren en dus nog meer nieuwe statussen genereert, wat uiteindelijk leidt tot een meer complete voorspelling van de nabije toekomst.

Wat conflictoplossingen zo complex maakt is de verbinding maken tussen de

oplossing en het conflict dat de gebruiker wenst op te lossen. Doordat het systeem de toekomst voorspelt, is het mogelijk dat deze toekomst kan veranderen bij de volgende voorspelling. Het is daarom ook plausibel dat conflicten in eerst kunnen voorkomen en daarna niet meer. Meer nog, een conflict zou zelfs kunnen schuiven in tijd met bijvoorbeeld een aantal minuten tot zelfs uren. In dit werk maken wij gebruik van de set van conflicterende acties als identificatie voor een bepaald conflict. Dit zorgt ervoor dat een oplossing niet gebonden is aan een bepaald tijdstip en het bijgevolg mogelijk maakt om doorheen de tijd mee te bewegen met conflicten die de dezelfde set van conflicterende acties bevatten. Deze aanpak heeft echter ook als bijwerking dat een oplossing meermaals kan matchen met conflicten doorheen de tijd, wat kan leiden tot ongewenst gedrag.

In vergelijking met andere methodes die conflicten detecteren en oplossen tijdens de run-time, worden eindegebruikers wel betrokken in onze aanpak door gebruik van de uCoRe applicatie. Eindegebruikers worden geïnformeerd over de statusveranderingen van hun apparaten in de nabije toekomst, en wanneer er zich conflicten voordoen, worden deze gemarkeerd door het systeem. Drie types van conflicten worden afgehandeld: inconsistenties, redundanties en loops (lussen). Hoewel het systeem redundanties en loops automatisch oplost tijdens de voorspellingsfase, krijgen eindgebruikers nog steeds de mogelijkheid om deze oplossingen aan te passen naar hun eigen wensen. Door gebruik te maken van iconen, afbeeldingen en tekst probeert de applicatie op een zo eenvoudig mogelijke manier relevante informatie, zoals conflict details en oplossing details, duidelijk te communiceren.

In onze online within-subject studie kregen deelnemers vijf scenario's, welke elk een type conflict bevatte, en werd hen gevraagd deze op te lossen aan de hand van twee methoden: regels aanpassen, wat mogelijk was via een gedeeld Google document, en het gebruik van de uCoRe applicatie, waar participanten bevelen moesten geven aan de facilitator om de applicatie te bedienen. De uCoRe applicatie bewees eenvoudiger en subjectief aangenamer in gebruik te zijn dan de methode van de regels aan te passen. Deelnemers moesten minder denken over het echte conflict, maar konden zich eerder focussen op het gewenste resultaat. Dit zorgde dan ook voor een hoger succes in het oplossen van conflicten wanneer de applicatie werd gebruikt. Om eenmalige conflicten op te lossen of om eenmalige statusveranderingen door te voeren wordt de uCoRe applicatie verkozen boven de regel-aanpassen methode.

Beperkingen van het huidige werk

Aan dit huidige werk zijn ook nog een aantal beperkingen gevonden. Conflict detectie vindt enkel plaats binnen een tijds tick. Dit houdt in dat tijdens het voorspellen, conflicten alleen gedetecteerd worden voor apparaat statussen die op hetzelfde moment veranderen. Wanneer de lamp bijvoorbeeld aan gaat en binnen een minuut weer uit, zou dit door een gebruiker beschouwt kunnen worden als een conflict. Echter op dit moment kan het systeem zulke conflicten, waar dus enige tijd tussen zit, niet detecteren.

Oplossing voor conflicten worden momenteel enkel gevormd door een combinatie van het activeren en snoozen van acties. Dit omwille van de verhoogde flexibiliteit dit het oplevert. Het zou echter completer zijn moesten operaties op regels ook

mogelijk zijn.

In dit werk waren we voornamelijk gefocust op drie types van conflicten, namelijk inconsistenties, redundanties en loops. Deze drie types hoeven echter niet de enige conflicten te zijn die het systeem zou moeten kunnen oplossen. Een interesse conflict zou bijvoorbeeld kunnen ontstaan wanneer het systeem de status van een apparaat verandert maar de gebruiker dit eigenlijk niet wenst. In het beste geval zou de eindgebruiker dus steeds in staat moeten zijn om eender welke status die het systeem voorspelt te overschrijven met de gewenste status van de gebruiker.

Een van de grootste beprekingen op dit moment is om op een zo correcte manier mogelijk ervoor te zorgen dat oplossing mee veranderen met conflicten wanneer deze schuiven in de tijd. De manier hoe we dit nu aanpakken heeft de bijwerking dat deze oplossing meermaals toegepast kan worden wanneer een de set van conflicterende acties van het conflict overeenkomt met de set van conflicterende acties waarvoor er een oplossing bestaat, resulterende in ongewenst gedrag. Het zou op een of andere manier mogelijk moeten zijn om oplossingen toch ergens enigszins in een tijdsrange af te baken en ervoor te zorgen dat deze slechts eenmalig worden uitgevoerd.

Wanneer een conflict wordt opgelost, wordt deze oplossing opgeslagen in het framework. Wanneer statusveranderingen verplaatsen door een nieuwe voorspelling, kan het mogelijk zijn dat het conflict niet meer bestaat. Dit zorgt ervoor dat de gekozen oplossing voor dat eerdere conflict niet toegepast kan worden en dus de status van dat apparaat kan verschillen met wat de gebruiker zou verwachten dat er zou gaan gebeuren. Op dit moment houdt het systeem niet bij welke conflicten er eerst voorspelt zijn geweest en of dat deze veranderd zijn door een nieuwe voorspelling. Hierdoor kunnen eindgebruikers niet geïnformeerd worden wanneer zoiets zou gebeuren.

Conclusie

Er is nog steeds veel vooruitgang te boeken in het Internet of Things en de smart home omgevingen. In het werk dat wij hebben geleverd is nog veel ruimte voor verbetering, vernieuwing en uitbreiding. Mogelijke uitbreidingen zouden dan bestaan uit het toevoegen van Artificiële Intelligentie, het gebruik van nieuwe visualisatie technieken zoals AR of VR en het gebruik maken van feedforward technieken om gebruikers nog meer te informeren over hun acties vooraleer ze deze bevestigen.

Om te besluiten draagt ons werk bij door aan conflict bemiddeling te doen in een IoT smart home omgeving *at run time*, gebruik makende van technieken om *acties te activeren of te snoozen* en de *eindgebruikers daarover de controle te geven*. Onze studie toonde aan dat onze applicatie verkozen wordt boven het aanpassen van regels wanneer het gaat over eenmalige conflicten en uitzonderingen maken op de status van een apparaat. Dit alles opent uiteindelijk een heleboel mogelijkheden voor toekomstig onderzoek.

Abstract

Within the Internet of Things and smart home environments, end users are given the tools to create and customize the behaviour of devices in their environment. This is done by creating trigger-action rules that describe what actions should be taken when an event occurs or a condition is met. Problems arise when multiple defined rules are triggered at the same time and influence the same device. These conflicts can leave the involved devices in an unknown state. Therefore tools are created to detect and resolve these conflicts. With existing tools focusing on creating a conflict free environment by detecting and resolving conflicts through changing rules at creation time, we propose an additional approach that detects conflicts at run time in the near future and facilitates the user into solving these one time specific conflict situations by temporarily disabling actions in rules without it having a long-term impact on the smart home system. These solutions would involve snoozing actions or creating custom actions. We build the uCoRe application that interfaces this conflict detection and resolution method. In a within-subject study we compared our approach with a rule editing method and evaluated its ease of use and success rate. The results not only show a higher success rate in favor of the application, the uCoRe application is also preferred by participants for solving one time specific conflicts and making one time state exceptions. Our work is the first step towards giving end-users the control over device state conflicts in their smart home environments at run time. This opens up a lot of opportunities for further research.

Acknowledgement

I would like to thank my promoter dr. Davy Vanacken en PhD student Sven Coppers for their guidance and support during the year. I am very thankful for the time they took to discuss our progress and to be involved in every step along the way. It was a really interesting and fun topic to work on and this team only made it even more engaging.

Contents

1	Introduction	13
2	Related work	15
2.1	Internet of Things	15
2.2	Smart Home	15
2.3	Trigger-Action	17
2.4	Conflicts	20
2.4.1	Conflict types	20
2.4.2	Existing tools	22
3	uCoRe: A Conflict Resolution Application	29
3.1	Introduction	29
3.2	Visualizing status, conflict and solution	30
3.2.1	Status	31
3.2.2	Conflict	32
3.2.3	Solution	33
3.3	Predicting status: Near Future Prediction	35
3.4	Conflict detection	39
3.4.1	Data structure	40
3.4.2	Conflict detection algorithm	41
3.5	Conflict resolution	44
3.5.1	Data structure	45
3.5.2	Apply a solution	47
3.6	Limitations	51
4	Executing a User Study	53
4.1	Study design	53
4.1.1	Experimental Design and Measurements	53
4.1.2	Participants	55
4.2	Analysis and results	55
4.2.1	Scoring understanding of the conflict	55
4.2.2	Solving the conflict	56
4.2.3	Subjective Usability Scale	59
4.3	Discussion	60
4.3.1	Summary of key findings	60
4.3.2	Limitations and future work	61

5 Conclusion	63
Appendices	67
A Executing a User Study	69
A.1 Training document	69
A.2 Scenarios	74
A.3 Questionnaires	76
A.3.1 Understanding the conflict questionnaire	76
A.3.2 Observation questionnaire	79
A.3.3 SUS and preference questionnaire	83

Chapter 1

Introduction

The Internet of Things is growing. More and more devices are added each day, and even more opportunities arise to create environments that would improve our quality of life. Smart homes are an example of how these so called “things” become part of our day to day living. Their goal is to not be intrusive but to aid and support us in everything we do. Thus, making our lives easier. End users are given control over their own environment by letting them “program” their devices on their own. Tools like IFTTT are created to make that programming a lot easier. However, using these tools to create rules that describe what actions should be taken when an event occurs or a condition is met, introduces issues. The more rules are part of the system, the more complex the system becomes. Therefore, keeping track of all the rules and everything that is going on in the system becomes difficult. Having a lot of rules also introduces more chances of rules conflicting with one another. In order to prevent conflicts from occurring in such rule-based systems, different tools were created to aid users in their rule creation at design time. These tools check, when new rules are added, whether they can possibly be in a conflict with some other rule(s) in the system. If so, users can ignore these warnings or solve them immediately by adding more conditions or adding more rules. This requires end users to think ahead of every possible situation that might occur, making sure that every rule that they create exactly does what they want them to do. As Paterno once said:

“It may be very difficult for developers to predict all possible context-dependant scenarios, because there may be (at design time) unforeseen requirements that need to be supported when the application is actually used.” [56]

Since we interact with our applications in such very dynamic and unpredictable environments, it is not possible to foresee at design time how an application should react to all the possible contextual changes that can occur during its use.

Not to mention that conflicts are complex. They can occur on resources or on policies, they can be caused by a single user or multiple users, they can be detected at run-time or even before they occur (e.g. at design time) and they can be avoided or resolved. Nonetheless, these conflicts can be classified into three main classes: loops, redundancies and inconsistencies.

Although many tools exist, most of them focus on detecting and resolving conflicts at design time. Using different ways of visualization, users can understand and even investigated why or why not a rule is in conflict. When conflict detection is done at run-time, however, users are most of the time left out, because these systems use automated resolution of conflicts. When resolving conflicts is done automatically, the system uses most of the time a form of prioritization of some sorts. Thus letting users define in advance which rules or people are more important than others.

A lot of conflict solving has to be done at design time, whether it is by creating or changing rules, or by assigning priorities. Once the rules are saved and applied, users don't have control anymore over the conflicts that might arise. In this work we considered an additional approach to the existing tools. Instead of detecting conflicts at design time and letting the user solve them immediately, we propose a solution whereby at real-time end users get notified of upcoming conflicts and in-situ can solve them according to the situation at hand. Thus letting them make exceptions on the rules by temporary snoozing actions or creating custom actions on their own. To accomplish all of this, we predict the near future of device states, detect conflicts and present them to the user using the uCoRe application. Through this application the conflict can be solved, resulting in the desired states at the right time.

In Chapter 2 we discuss the related work of conflict detection and resolving in IoT. In Chapter 3 we present our work and discuss the four main components: State prediction, Conflict detection, Conflict resolution and the uCoRe application. After that, a small study is conducted in Chapter 4. In the end we close with a conclusion and some perspectives on the future in Chapter 5.

Chapter 2

Related work

2.1 Internet of Things

The term Internet of Things (IoT) was first mentioned by Kevin Ashton in 1999 in the context of supply chain management [7]. Since then, the Internet of Things grew in interest and popularity, extending not only to supply chain management but also to other domains like health, transport, logistics, smart homes and smart cities [20]. The Internet of Things can be described as a network of interconnected everyday objects that not only harvests information from the environment (sensing) and interacts with the physical world (actuation/command/control), but also uses existing Internet standards to provide services for information transfer, analytics, applications, and communications [14].

Although many definitions have been formed throughout the years, all agree on the potential of this emerging paradigm. Not only can it make people live and work smarter than before, it offers us the possibilities to gain even more control over our lives and our environments [64]. For example, in a business context, IoT enables a company to monitor their business processes in real time, or even automate some of them. Saving them time and money. Even in our personal lives, our homes, more of these smart objects (things) can be found and improve our quality of life. Think of a smart thermostat, which can sense the temperature of the room and adjust it in an autonomous way, making sure that when people are not home, the heating is off. Saving energy costs. With all these new smart devices, it becomes important to understand how users operate with them, as well as the applications and services that are being used. In order to do that, analyzing the IoT data generated is crucial in helping us improve these systems [45, 25].

2.2 Smart Home

A smart home is one of the domains where IoT has been changing the way we think of living with technology. According to Investopedia, a smart home refers to a convenient home setup where appliances and devices can be automatically controlled remotely from anywhere with an internet connection using a mobile or other networked device [62]. The purpose of a smart home, and home automation in general, is to support inhabitants in their daily lives through technological means. Areas

of interest include security functions like access control and surveillance systems, resource management with regard to water and electricity, multimedia functions for ubiquitous content streaming, and comfort functions like light and heat management [21, 28].

It is Rogers that argues for more engaging technologies that “enables people to do what they want, need or never even considered before by acting in and upon the environment” [5]. However, an inhabitant can never have the feeling that they are not in control, because this can lead to an inhabitant not trusting his own home. Therefore it becomes a challenge of balancing home automation and user control. Besides, an inhabitant can vary in age and ability, resulting in a wide variety of possible users of such systems. This diverse user profile makes interaction design and development for smart homes incredibly challenging. On the one hand, solutions should be ubiquitous and refrain from impacting the user’s life; on the other hand, people expect a certain degree of control over what is going on in their own homes [11, 15, 18, 19, 21].

Different smart home systems and platforms have been developed to optimize user control and understanding of a smart home [29, 39, 44]. For example, Castelli et al. developed a smart home system featuring predefined visualizations and a visualization creation tool for even more customization [29]. On the other hand, Purmaissur et al. developed a platform for smart home control and energy monitoring, interfaced with augmented reality (AR) [44]. In Figure 2.1, this platform is used in combination with AR to show live energy usage of electrical components.



(a) width=7cm



(b) width=7cm

Figure 2.1: AR energy monitoring [44].

Even though these kind of tools have been developed, tailoring them for every user or possible smart home inhabitant is difficult. As people we have our own

habits, preferences, goals and values, and as mentioned before, a smart home should be able to support us in every need. It is therefore not interesting for a developer to create smart home applications that will need to be adapted for every user's specific needs. By giving users the opportunity to create their own applications, they gain full control over their home and can personalize it in any way they want. A more specific term for this is End-User Development.

End-user development (EUD) aims to put the applications development in the hands of the people who are most familiar with the actual needs to be met (e.g., domain experts). Its goal is to allow non-professional developers to create or modify their applications so that they can meet their diverse and frequently changing needs better than before [4]. As Paterno and Alawadi pointed out, there is an emerging need to support personalization of IoT environments [56]. And this can be done in many different ways. Barricelli et al. describe and map all the research that has already been conducted in this area of end-user development [48].

It is by understanding the user and its use cases, that rule-based and process-oriented paradigms were formed. Rule-based notations are ways for a user to express what they want the system to do given some event that occurred. For example, if Sarah is home then turn on the music. The system will then know that when Sarah is home, it should play some music. Many rules can be created in this way. Although rule-based notations are sufficient to express these simple automation tasks, it can be limiting for more complex use cases. Therefore process-oriented notations can be very useful. Instead of, for example, modelling a whole business process, which can be complex, into many different rules, process-oriented notations help to overcome these limitations. According to participants that used the process-oriented notation, expressiveness was greatly valued and they presented them with more complex use cases involving more devices in their homes [28]. The downside of these process-oriented notations, however, is that it is often coupled to more complex user-interfaces, making it thus harder for an end user to use.

Because of this, many research proposed an end-user development solution based on trigger-action rules [13, 55, 56]. Ur et al. even provided evidence that average users can successfully engage in trigger-action programming with multiple triggers and actions [23]. In the following section we will dive deeper into this.

2.3 Trigger-Action

Within the Internet of Things domain, the Trigger-Action paradigm has been widely adopted in the last few years, because it allows end users without programming experience to describe how their application should react to many events that can occur in such very dynamic contexts. This paradigm makes use of the “if . . . , then . . .” paradigm, where in the if-part an event or condition is described (trigger) and in the then-part an action is described to be executed. A trigger is thus an event, which is corresponding to a context change (e.g., if the TV is turned on), or condition, which is more of a prolonging state (e.g., while Sarah is home).

As mentioned in the previous section, giving users the means to describe to the system how they want it to interact with them and the environment, empowers them, letting them do the personalization themselves. By defining rules, an end user can

specify how they want certain things to be done. Many tools and editors have been created in order to help end users do this in a very simple way. IFTTT¹ (If This Then That) is such an (commercial) application tool that enables end users to do just that. By creating “if . . . , then . . .” rules, in this environment also called recipes, existing services can be coupled to one another in a one-on-one connection. A rule or recipe could look something like this: “IF it is raining tomorrow, THEN send me an email”. It is actually really easy to do. Ur et al. confirmed this by expressing the growth of trigger-action programming in the IFTTT environment, pointing out that most users create “recipes” not for immediately sharing it with the whole world, but to fill the gaps in their own lives, customizing their environments using end-user programming[27]. Due to the simplified way of IFTTT for customizing the environment, creating complex use cases can become very difficult because of the limited expressiveness it offers. Debugging such a system is therefore not possible in this environment.

In research many alternatives have been created in order to deal with some of the limitations these commercial tools have. In the work of Cabitza et al., different existing rule-based tools were compared with each other [26]. The work presented in [12] even describes a usability study with 16 participants, which compares three composition paradigms for smart environments:

- filtered lists, where condition-response compositions are obtained by selecting conditions and responses from respective lists
- wiring composition, which is based on the metaphor of coupling and wiring together separate user interface components
- jigsaw puzzle composition, where users specify compositions by combining puzzle-like user interface elements representing trigger or response components

From this study, it emerges that the approaches based on the wiring and the jigsaw puzzle present issues concerning composition readability and overview; whilst, filtered lists communicate more effectively the overall picture of compositions.

Many tools use the filtered lists composition paradigm. Some examples are: IFTTT, TARE [34], EUPONT [30], E-Free [31] and E-Wizard [31]. All have in common that they look similar and are based on IFTTT. Although TARE, EUPONT, E-Free and E-Wizard are all trying to improve the expressiveness by adding more semantics (specific information) about the triggers and the actions using the 5W-model (Who, What, When, Where, Why) [31], they also are a bit different from one another. TARE is a web-based authoring tool that can combine multiple triggers using boolean operators, and it can even differentiate between an event and a condition. Figure 2.2 shows the editor user interface that was created. EUPont, on the other hand, is a semantic web ontology that enables users to meet their needs with fewer, higher-level rules that can be adapted to different contextual situations and as-yet-unknown IoT devices and services. Even though it is similar to IFTTT, besides adding more semantics, it strives to be simpler and more expressive.

¹<https://ifttt.com/>

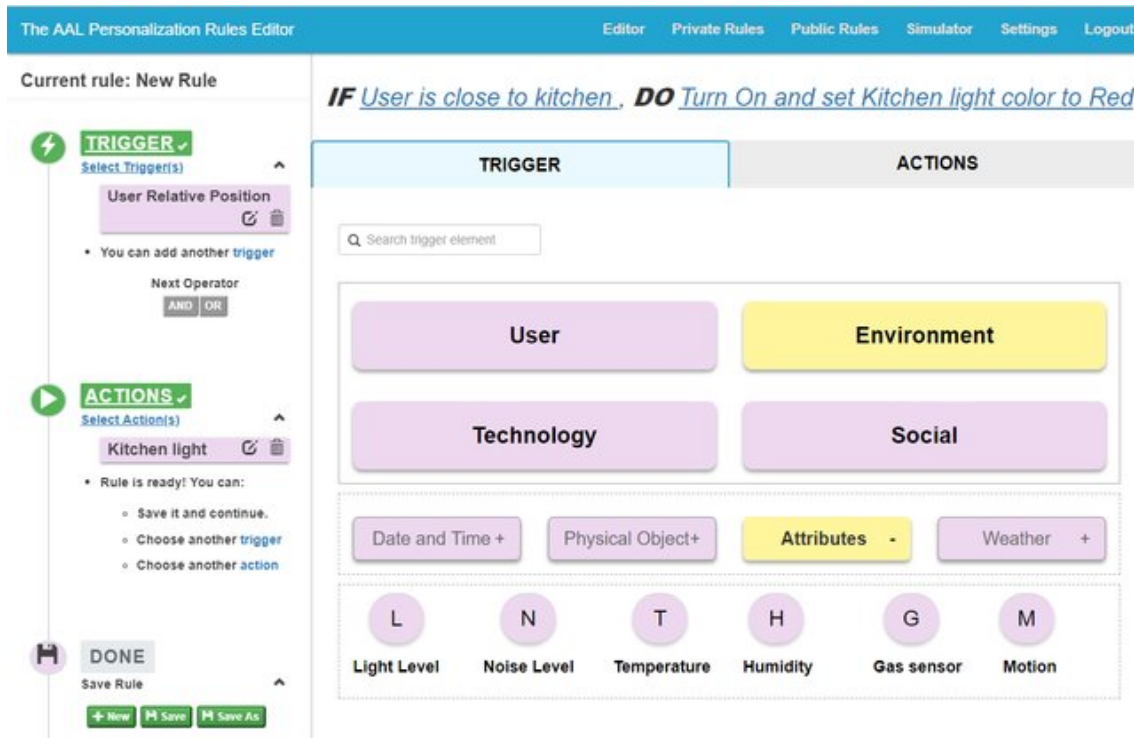


Figure 2.2: The TARE editor user interface [34].

A popular tool implementing the wired paradigm is Node-RED². Besides offering a set of predefined services, it allows users to register personal smart objects by invoking their RESTful interfaces. In addition, Node-RED supports the creation of complex automation rules characterized by:

- multiple services that trigger events and multiple services that react by performing actions
- special nodes, used for example to control the communication flow among services by means of custom JavaScript code
- debug functionality to simulate and check the rules under creation

However, such features often require technical skills and thus they are not adequate for non-technical people [6, 9, 10]. That is why E-Wired also used this graph-like notation, but simplified it, for creating a customized environment [31].

My IoT Puzzle is a tool to compose and debug IF-THEN rules based on the Jigsaw metaphor [52]. The tool interactively assists users in the composition process by representing triggers and actions as complementary puzzle pieces, and by providing real-time feedback to test on-the-fly the correctness of the rule under definition. In the next section we will look closer to the visualization and debugging features of this tool.

The goal of all these tools is to help the end user in personalizing their environments, and being able to do this in a simple way that is instructive and informative

²<https://nodered.org/>

when it needs to be. Although this sounds easy, it is hard to get it right. With a lot of different triggers and actions, many candidate rules exist for non-programmers to generate or chose from. Thus, making trigger-action programming often a complex task. A recommendation system could improve both the reuse and the definition of trigger-action rules, thus helping the users to easily customize their smart devices [56]. The goal of RecRules [53] is to recommend by functionality: it suggests rules based on their final purposes (e.g., light up a place), thus overcoming details like manufacturers and brands. Still, it only solves one thing of many. Another way of making trigger-action programming easier can be by using natural language instead of the IF-THEN paradigm. Huang et al. introduced InstructableCrowd [54], a crowd-powered system that allows users to program their devices via conversation. The user verbally expresses a problem to the system, in which a group of crowd workers collectively respond and program relevant multi-part IF-THEN rules to help the user. Yet, even with simplifying the way of trigger-action programming, or using a recommendation system, users can still make mistakes and create buggy rules. In the next section we will dive deeper into conflicts between rules and the tools that are created to help end users better understand these problems and fix them.

Lastly, when creating tools for end-user development it is important that the focus is on the user, their understanding of the system and the control they have over the system. Analyzing tools can help in order to visualise end users and their behaviour. Corcella et al., for example, created a visual tool for analysing IoT Trigger-Action Programming using the TARE editor [50]. The goal of this tool is to provide better understanding of what end users' personalization needs are, how they are expressed, how users actually specify rules, and whether users encounter any issues in interacting with the personalization features offered by the editors.

2.4 Conflicts

In a smart home environment, a conflict can be understood as the system doing something that the inhabitant does not want or expect. For example, in such an environment it is possible for the lights to turn on or off automatically. When the inhabitant expects that the lights should go on, and it doesn't happen, something probably went wrong. Within research, some studied the different ways of how these conflicts are formed, and tried to come up with solutions to detect, prevent and resolve them.

2.4.1 Conflict types

When addressing conflicts in the trigger-action paradigm, we specifically look at conflicts that arise between trigger-action rules. First of all, an understanding of the different types of conflicts is needed before even starting to fix them. Brackenbury et al. synthesised a taxonomy of ten bugs likely to impact Trigger-Action Programming [49]. Resendes et al. even created a taxonomy of conflict, which is organized according to their classification dimensions, possible types and respective meaning (see Figure 2.3) [17]. However, when a conflict occurs and is classified according

Classification dimension	Possible types	Meaning
Source	resource application policy role	concurrency over a resource concurrency over an application conflicting policies in a context conflicting profiles in a context
Intervenients	single user user vs. user user vs. space	conflicting user intentions concurrency over a resource or application user conflicts with space constraints
Time of Detection	<i>a priori</i> when it occurs <i>a posteriori</i>	detected before its occurrence (predicted) detected during occurrence detected after it has occurred
Solvability	conflict avoidance conflict resolution acknowledge inability acknowledge occurrence	the system resolves the conflict before its occurrence the system resolves the conflict during its occurrence the system recognizes its inability to resolve conflict the system acknowledges too late that a conflict occurred

Figure 2.3: A four-dimensional taxonomy of conflict, organized according to their classification dimensions, possible types and respective meaning [17].

to this taxonomy, a conflict still can be subdivided into three main classes that are defined as loops, inconsistencies and redundancies [40]. Loops are formed when two or more rules chain together, leading to the last rule in the chain triggering the first rule again. As an example, the following rules form a loop:

- R1. IF I post a photo on Facebook, THEN store it in my iOS library
- R2. IF I store a photo in my iOS library, THEN post it on Instagram
- R3. IF I post a photo on Instagram, THEN post it on Facebook

Once a photo gets stored or posted, it enters a cycle of storing and posting the same photo over and over again. An inconsistency conflict occurs when the actions of two or more rules operate on the same device, and result in an opposing state. The following rules, when executed at the same time, result in two different states of the lamp (on or off), leading to unpredictable behaviour.

- R4. IF I come home, THEN turn on the lights
- R5. IF the TV is turned on, THEN turn off the lights

Redundancy conflicts are almost the same as inconsistency conflicts, but instead of resulting in opposing states, the actions result in doing a similar thing. Again, the following rules give an example of how this would look.

- R6. IF my Android GPS detects that I exit the home area, THEN set the Nest thermostat to Away mode
- R7. IF the entrance door is locked, THEN set the Nest thermostat to Away mode

It is possible that both rules are executed at the same time, resulting in two actions that do the same thing. Results show that participants perceived redundancies as less dangerous than loops and inconsistencies [40, 51]. For example, by looking at a

redundancy that simultaneously posted on Twitter two tweets about the same topic, i.e., “I listened to the new song of Ed Sheeran on YouTube” and “I listened to the new song of Ed Sheeran on Spotify”, two users said “this is not a problem for me, the two tweets are different so I want to save the rule anyway”. Although depending on the domain, redundancies can be very dangerous if they occur. In healthcare, for example, a redundancy can result in giving a medicine dose twice, which could lead to an overdose with tragic consequences. Even though these three main classes are defined, depending on the domain, more or less classes can be created and used to define and specify the different conflicts that can occur in such a system[35].

2.4.2 Existing tools

To make sure that no conflicts can occur when the rules are executed, many developed tools to not only guide users in the creation of their trigger-action rules, but also to immediately notify them when a rule they just created is in conflict with another rule that already exists in the system. Here follows a brief overview of all the different research and tools we examined:

ICAP [3] considered two possible levels of conflicts: potential conflicts at design time and actual conflicts detected at run time. At design time, when rules are saved, it checks whether any of the saved rules could potentially conflict and, if any conflict occurs, the concerned rules are highlighted to the user to resolve the conflict or ignore it. If rules conflict at run time, iCAP, by default, executes the rule most recently updated rule.

ITAD (Interactive Trigger-Action Debugging) [43] is a tool that supports the Interrogative Debugging paradigm approach [2] for trigger-action rules. This approach lets users ask “why” and “why not” questions to know the reason why a rule is (respectively: is not) verified. Moreover, this tool integrates a conflict analysis functionality able to highlight potential conflicts between rules that are simultaneously verified (see Figures 2.4 2.5). ITAD also helps users better understand the difference between events and conditions. Indeed, on the one hand a rule is presented as verified only if the user specifies the right option between event and condition and provides the correct values to use in the simulation. On the other hand, when a rule is not verified, in the explanation associated with ‘why not’ the tool highlights in red the elements that do not match, to inform the user about the reason why the rule is not verified (see Figure 2.4).

EFESTO-5W [31, 32, 38] is a web-based platform that, by means of a visual composition paradigm, allows non-technical end users to synchronize the behavior of multiple smart devices; and it is a platform for the specification and execution of trigger-action rules, introduced in [34]. A solution for integrating end user debugging features in such an approach is presented in [43].

EUDebug [51] is a system for debugging trigger-action rules that allows the user to compose a new rule, view any problems that the rule may generate, further

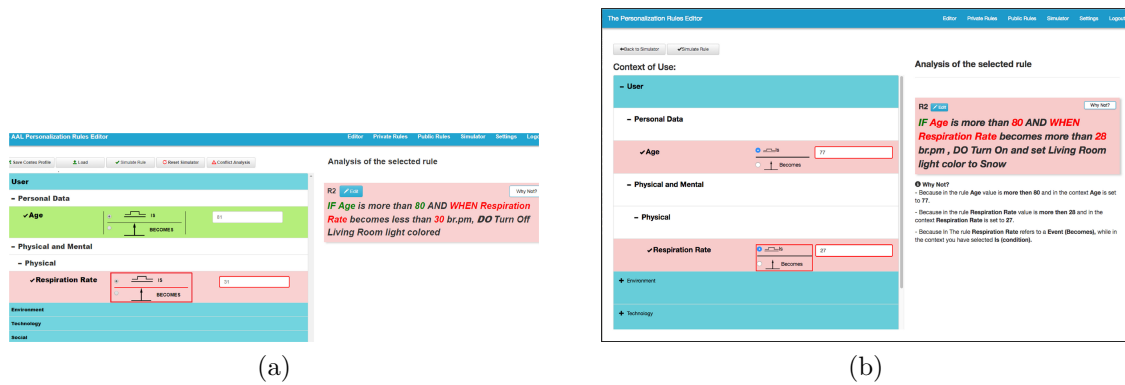


Figure 2.4: ITAD simulation of rules [43]. a) A detection of a conflict, which is pointed out by the system by marking the values and boxes that cause the problem; b) Revised interface where also textual feedback is given for better understanding.

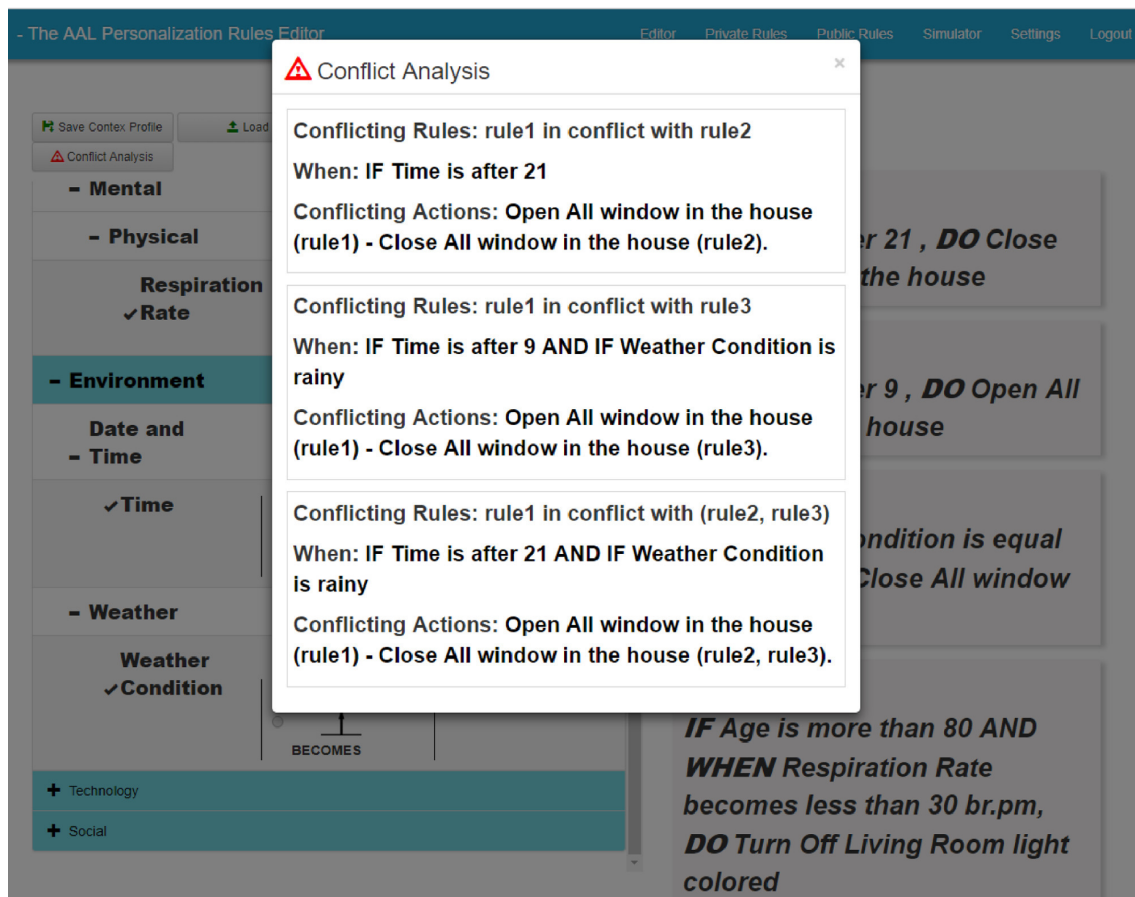


Figure 2.5: ITAD: conflict detection and analysis tool, giving an overview of what causes the conflicts [43].

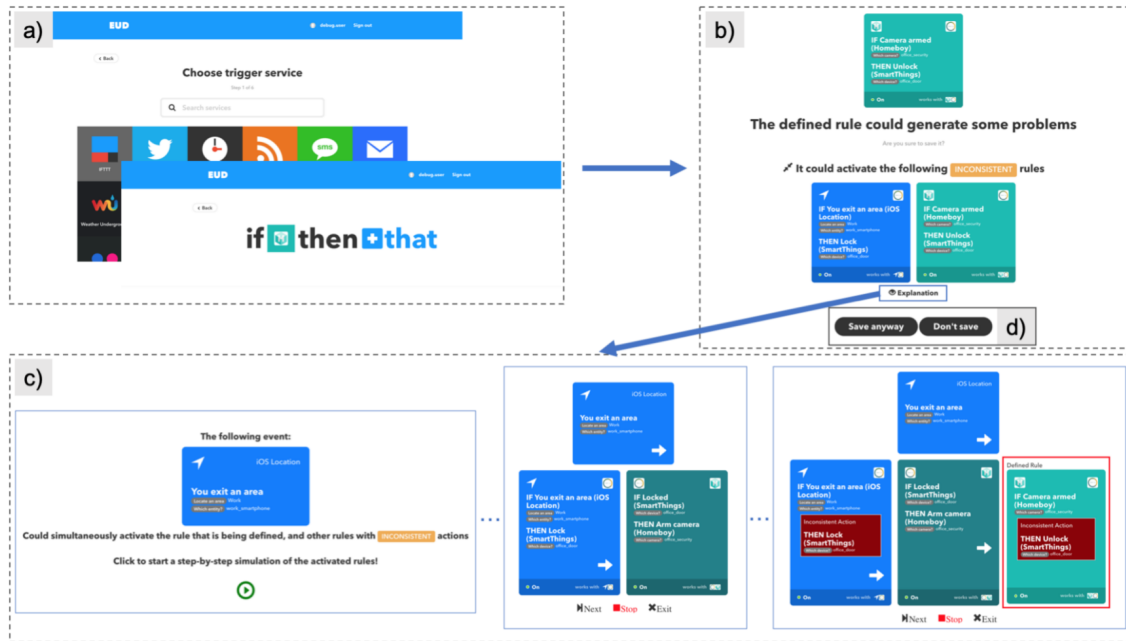


Figure 2.6: EUDebug is a system for debugging trigger-action rules that allows the user to: a) compose a new rule; b) view any problems that the rule may generate; c) further investigate each problem with a step-by-step simulation; and d) edit the rule to fix the problem or save it anyway [51].

investigate each problem with a step-by-step simulation and edit the rule to fix the problem or save it anyway (see Figure 2.6). During rule composition phase, EUDebug automatically detects potential problems with no user intervention. At the end of the composition process, EUDebug shows any conflicts that the composed rule may generate by interacting with the previously defined trigger-action rules, and allows users to further investigate why the problem happens. They use Semantic Colored Petri Net in order to detect problems in rules and characterizes problems in trigger-action rules as loops, inconsistencies and redundancies.

My IoT Puzzle [52] is a tool to compose and debug IF-THEN rules based on the Jigsaw metaphor. The tool interactively assists users in the composition process by representing triggers and actions as complementary puzzle pieces, and by providing real-time feedback to test on-the-fly the correctness of the rule under definition. Puzzle pieces, for example, deteriorate over time according to their usage, while the tool is able to warn users of conflicts, namely loops, inconsistencies and redundancies. Furthermore, the tool empowers end users in resolving problems through textual and graphical explanations. Following the Interrogative Debugging paradigm [2], for instance, the tool is able to answer questions such as “why it is not working?”, thus providing the user with a textual explanation of the detected problem.

Out of research they discussed three categories for End-User Development tools: form-filling (wizard-based languages such as IFTTT, EUDebug, . . .), block programming (such as Scratch [8] and the Jigsaw metaphor), and data-flow (process-oriented nature and makes them one of the best choices for complex use cases). My IoT Puz-

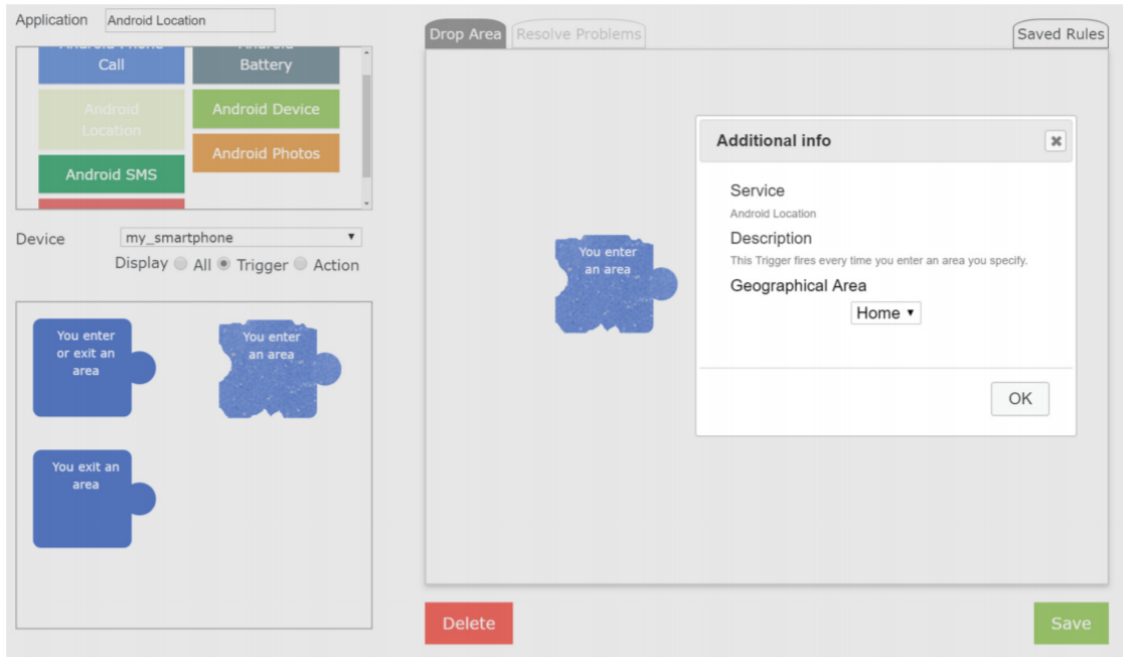


Figure 2.7: The user starts to compose a new rule by dragging a new trigger on the Drop Area. The tool provides the user with an initial feedback: the piece of puzzle is worn, since it has been already used in other rules [52].

zle uses a block programming approach for composition of IF-THEN rules, because they are less restrictive and stimulate the users creativity, in spite of form-filling approaches which are, although intuitively, more perceived as restrictive because of their closed form [36, 37]. They also use a data-flow visual language for representing the behaviour of multiple trigger-action rules, with the aim of helping users understand and identify unwanted run-time behaviours.

Wrong operations are prevented by the shape of the puzzle pieces, e.g., two trigger pieces cannot be connected. Furthermore, as shown in Figure 2.7, the dropped trigger piece is worn, since it is already be used in other rules. Puzzle pieces deteriorate over time according to their usage history. Using the same trigger in multiple rules, in fact, means that the involved rules will be executed at the same time, thus increasing the chances of introducing conflicts such as redundancies and inconsistencies.

If a rule is created that forms an inconsistency with previously saved rules, red feedback is given to the user (see Figure 2.8). In order to gain a better understanding of the conflict, a Resolve Problems area is provided with textual and graphical explanation of the inconsistency (see Figure 2.9). In order to resolve the problem, the rule has to be changed.

AutoTap [58] lets users specify desired properties for devices and services, instead of just adding trigger-action rules, which it translates to linear temporal logic (LTL), and both automatically synthesizes property-satisfying trigger-action rules from scratch and repairs existing trigger-action rules. Figure 2.10 demonstrates the problem AutoTap tries to solve with this method. When a user wants the windows

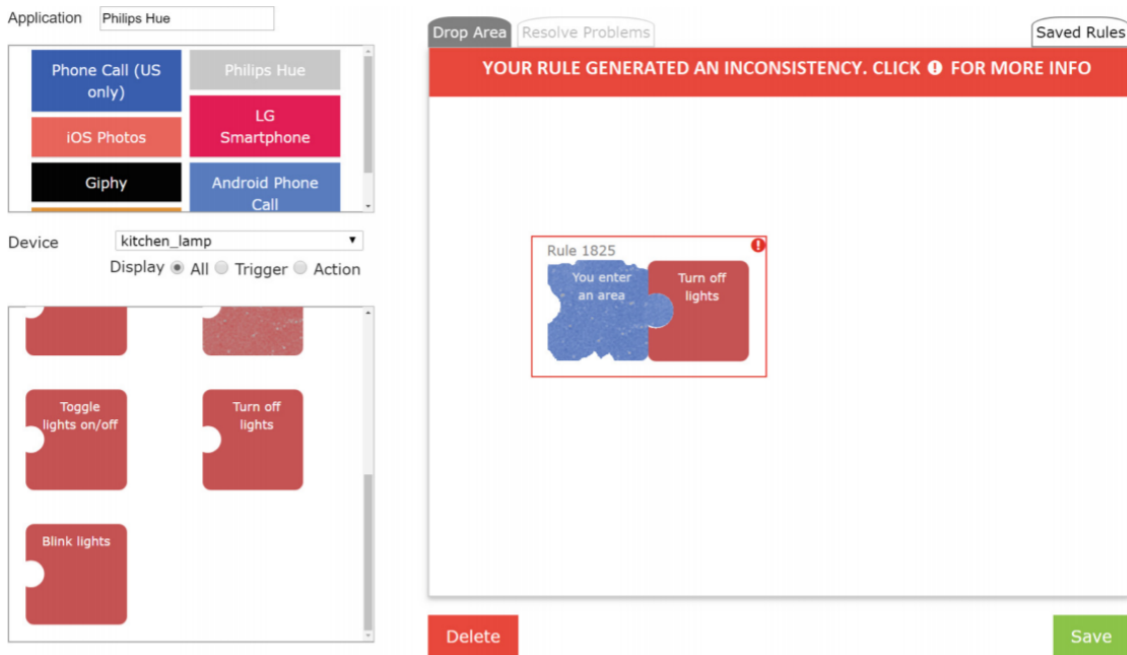


Figure 2.8: The user connects to the trigger an action that is inconsistent with some previously saved rules. The system gives a warning using a red feedback [52].

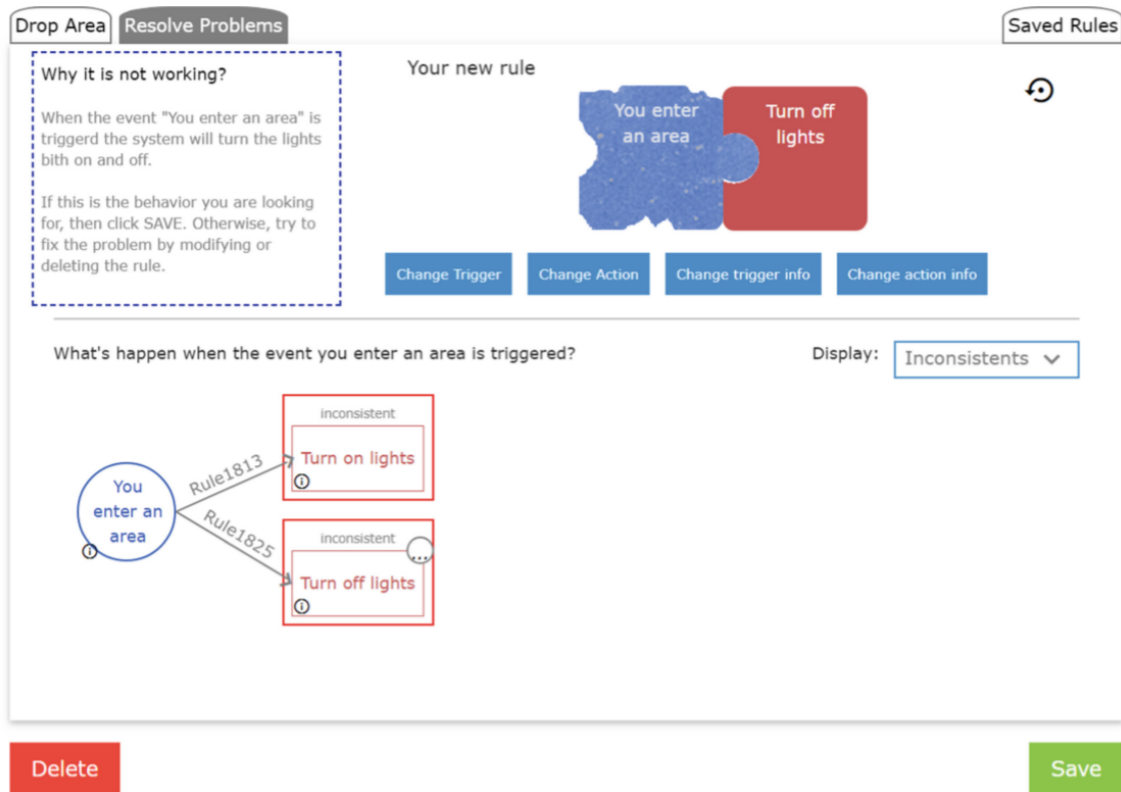


Figure 2.9: By opening the Resolve Problems area, the user can see textual and graphical explanation of the inconsistency, and can resolve the problem by changing the rule [52].



Figure 2.10: AutoTap: On the left: a (buggy) trigger-action rule.; On the right: a proposed trigger-action property. The trigger-action rule (left) cannot guarantee the property (right) [58].

to be closed when it starts raining, the rule on the left is probably what the user will create. But if another rule should exist that opens the windows when it is too warm in the house, the windows could be open when it is raining. Resulting in behaviour that the user does not want. Therefore, by specifying the desired properties as seen on the right side of the figure, the system creates the rules themselves to make sure everything happens the way the user wants.

CityGuard [35] detects and resolves conflicts in a city, even among actions of different services, considering both safety and performance requirements. They detect different types of conflicts by intercepting actions ahead of time, analyzing the details of the actions and then running simulations to predict potential conflicts with a temporal and a spatial range. They primarily focused on conflict detection for environmental conflicts. In addition, conflict resolution is based on priority and performance.

DepSys [22] specifies, detects and resolves specific conflicts in the smart home. These conflicts are a result of interdependency problems between a number of applications who share physical world entities. It uses priority of groups (health, security, entertainment, energy) to solve these conflicts.

IoTC2 [46] uses a formal method approach for detecting conflicts using Prolog. Finding conflicts is done by an iterative deepening strategy.

CLIPPER [24] is a framework for novel rule-conflict detection and resolution algorithms. It requires the syntactic structure of the rules and the rule clauses to be

presented in Event Calculus format [1]. CLIPPER uses a rule-based device priority system to handle conflict resolution, and as a result disables rules temporarily.

Shah et al. [57] detect and resolve conflicting and incomplete rules and can also detect conflict where actions nullify each other (For example, Turn on the heater and turn on the air-conditioner). A set of rules are considered incomplete if, for any actuator, a sensor value range exists for which the actuator's action is not defined. Meaning that when a rule is created that turns on the heating when the temperature is lower than 15 °C, another rule must exist that covers the temperatures above 15 °C. Rules are converted into its DNF (disjunctive normal form) form. After that, for each actuator, they find overlapping DNF terms from the rules and if those rules perform conflicting actions, those rules are marked. To detect incomplete rules, a variant of the covering polygon problem is used. Their resolution system is also more or less based on priority.

ABNA [47] utilizes agents and applies negotiation, enabling services with contrary features to work simultaneously, doing this at run-time. It avoids applying priority between services or house residents' preferences whenever a space for a compromise exists. The mechanism of ABNA is based on the use of a hierarchy of features based on their contribution to the function of the service or on the importance of these features to house residents. To achieve a compromise between conflicting services, ABNA models services and residents by using agents, and implements a negotiation algorithm that allows services with conflicting features to work simultaneously.

Although many tools exist, most of them focus on detecting and resolving conflicts at design time. It therefore isn't possible to make occasional exceptions that are caused by the context of a situation. Even doing conflict detection and letting users resolve them at run time is something that is not possible in the current tools.

Chapter 3

uCoRe: A Conflict Resolution Application

3.1 Introduction

With the growth of IoT, the growing amount of devices and the opportunities it brings into our homes, giving users the tools to program and control their environment becomes more and more important. With the focus on end-user development within IoT, many looked at ways to create relationships between events/conditions and actions, as well as the visualization and presentation of it all.

Previous work (Sections 2.3 and 2.4) proposed different ways of creating trigger-action rules, but only a few ways of debugging such a system. Most of them have in common that the debugging stage is part of the creation of new rules. Which results in resolving a conflict at design time. Although this might seem more than fair, it requests users to specify every situation as good as they can by creating even more rules or adding more conditions in order to avoid conflicts. Once the rules are saved, the system takes over and a user loses control over any conflict that will occur.

In this chapter we propose an additional approach in conflict detection and resolving. Instead of detecting and resolving conflicts at design time, we predict at run-time the near future of device states, we detect conflicts in the near future and let the user resolve them at run-time. This work contains four major components: the predicting of device states, the conflict detection, the conflict resolution and an application as mediator between the user and the Home Assistant. Figure 3.7 gives the overall structure of our work. By extending the FORTNIoT prediction engine and combining it with an application for visualizing and solving conflicts, a new approach is formed.

In this work, we focused on conflicts that occur on *resources* in a *single user* environment, where the detection is done *a priori* and thus a *conflict can be avoided* (conflict taxonomy as seen in Section 2.4). To aid us in discussion of this work, we use three scenarios.

Scenario 1. During the day, Sarah likes to enjoy as much sunlight as possible. Therefore she wants all the rolling shutters to be raised when she is home. To save some money, she does not want the lights to be on when there is enough sunlight

already in the house. To achieve this, she created the following rules in her home system.

- R1.1: IF the sun comes up, THEN turn off all the lights
- R1.2: IF the lights are off AND Sarah is home, THEN raise the rolling shutters
- R1.3: IF the rolling shutters are raised, THEN turn off all the lights

Scenario 2. Alex has only a few things he wants his house to do for him. When he is home, the TV should always be on because he likes the background noise. Every time he watches TV, his rolling shutters should be lowered and the lights should be turned off, otherwise light would reflect in the screen. And lastly, every evening when the sun goes down, he wants his rolling shutters to lower as well. The following rules were created to achieve this:

- R2.1: IF the sun goes down, THEN lower the rolling shutters
- R2.2: IF the TV is on, THEN lower the rolling shutters AND turn of the lights
- R2.3: IF Alex is home, THEN turn on the TV

Scenario 3. Michael automated his home some time ago. He started off with some simple things. For example, when he is home and the rolling shutters are lowered, he wants all the lights in the house to be turned on. When he is watching TV, light should not be reflected in the screen. In order to achieve that, the rolling shutters should be lowered and the lights should be off. Thus creating a nice environment to watch a movie.

Lately Michael started to work from home, but he does not like it when it is totally quiet. So every time Michael is home during the day, he likes to have some background noise. He achieves this by turning on the TV. The following rules are created in his home system:

- R3.1: IF the sun comes up AND Michael is home, THEN turn on the TV
- R3.2: IF the TV is on, THEN lower the rolling shutters AND turn off the lights
- R3.3: IF Michael is home AND the rolling shutters are lowered, THEN turn on the lights

3.2 Visualizing status, conflict and solution

It was in a study on smart home intelligibility by Jakobi et al., that the needs of IoT control and accountability were explored [41]. They found that participants initially looked for in-depth awareness information from their dedicated web-based dashboard. In the later phases of appropriation, however, their interaction and information needs shifted towards management by exception on mobile or ambient

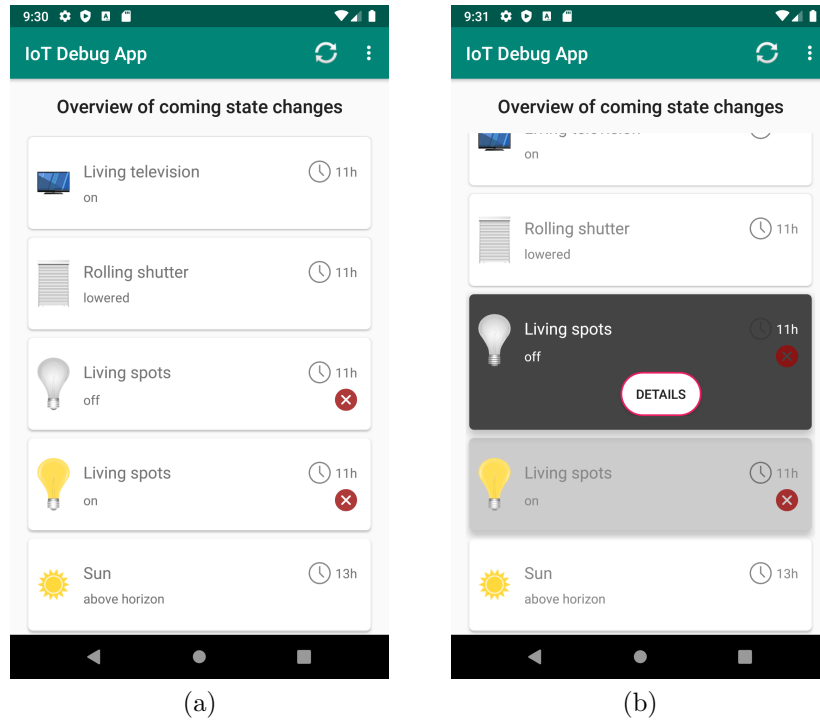


Figure 3.1: Overview of Michael’s state changes: a) a list of state changes, describing the entity that changes, the relative time when it changes, the new state and eventual markings when part of a solution or conflict; b) once a state is selected and part of a conflict or solution group, all states involved will light up.

displays – only focusing on the system when things were “going wrong”. The application we created has the purpose to aid users in this management by exception stage.

3.2.1 Status

In Scenario 2, mentioned in the previous section, three rules are created. Knowing what trigger-action rules are (see Section 2.3), when a trigger occurs (e.g., the sun comes up), an action will be executed (e.g., turn off all the lights). The result of an executed action is a device state (e.g., lamp 1 = *off*). This would mean that in Scenario 2, when the sun comes up, rule R2.1 will be executed, and all the lights will be turned off. So every light in the house will have the state *off*. These states can then be displayed in the application.

Instead of displaying the state of every devices at all times, with management by exception in mind, we are more interested in when the state of a device will change and into what new state it will enter. Therefore the application displays state changes of devices. To be more specific, as we mentioned earlier, we predict and display the near future and all the state changes that might happen in it. How the prediction is done, will be discussed in Section 3.4.

Let us now take a look at Michael (Scenario 3). Within the application, the first thing Michael will see is an overview of all the state changes in the near future. In Figure 3.1 the overview of Michael’s state changes are shown. Here you can see that

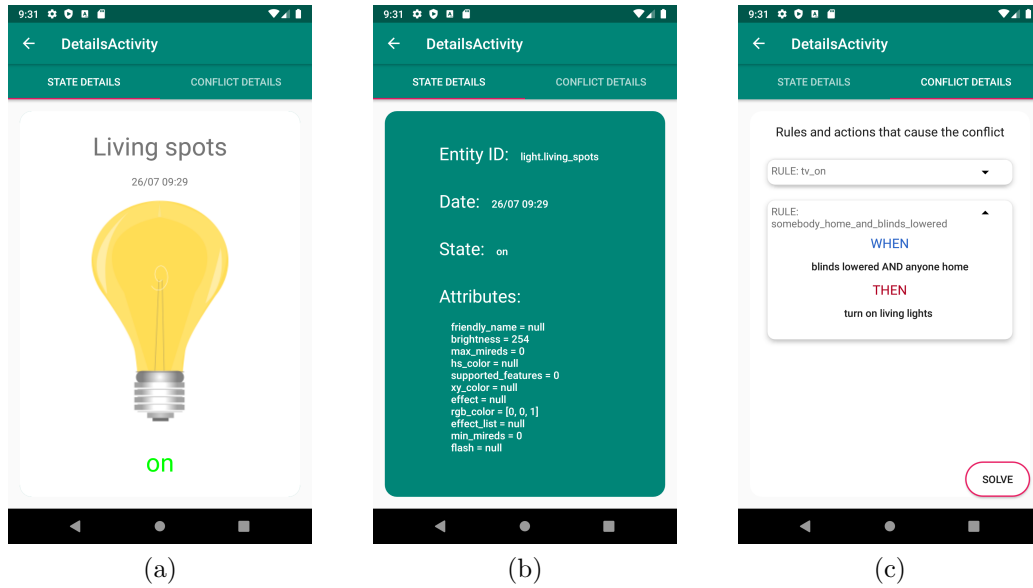


Figure 3.2: Overview of state information: Michael can see the state (a), more specific details about the state (b) and the details of the conflict the state is part of (c).

every state change is presented by the name of the device on which the state change occurs, the state itself and the relative time of when it will happen. Besides the obvious information of a state change, icons are also used to mark them in special occasions.

When Michael selects the state of the living spots being off (see Figure 3.1b), he can get more information about that state by pressing on the Details button. Once pressed, the application visualizes the status of the device after the state change and the conflict information it is involved with (see Figures 3.2b and 3.2c). To visualize the status of the device, it is important to identify all the information that is relevant to the user. As we can see in Figure 3.2a, an image and text are used to communicate the state of the device to the user.

3.2.2 Conflict

When a conflict between states occur in the system, the application communicates this to the user. As seen in Figure 3.1a, Michael can see that the living spots appear twice in the list and both are marked by a red cross icon.

In general, a conflict is typically marked by using this red cross icon. Every state change that is involved in a conflict will be marked. In Michael's case, both state changes occurring on the living spots are being marked. When multiple conflicts are found, many state changes can have the same marking, and thus it can become rather difficult to see which state changes are in conflict with one another. To aid the user on that regard, we use a secondary marking method to group state changes from the same conflict. As seen in Figure 3.1b, once a state change is selected and it is part of a conflict, every other state change that is part of the same conflict will be highlighted.

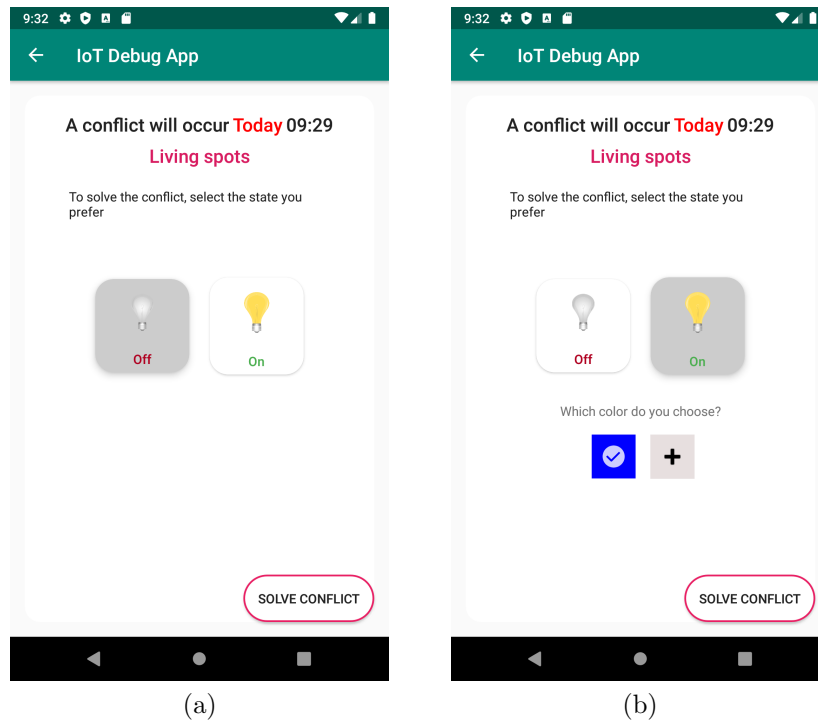


Figure 3.3: By selecting the desired state for the living spots, the system gets notice of the users preference for solving this conflict. (a) the lights can be turned off, or (b) the lights can be turned on and a desired color for the light can be selected.

As mentioned before (see Subsection 2.4.1) there are different types of conflicts. The visualization should make it clear to the user what kind of conflict the system is dealing with. In Figure 3.2c a conflict is displayed by pointing out which device is involved in the conflict, as well as the rules or actions that are causing this conflict to exist. It is at this screen the user gets the opportunity to first inform themselves about the conflict before solving it. Michael can see in Figure 3.2c that the conflict is caused by two rules, R3.2 and R3.3.

3.2.3 Solution

Using the application, a solution can be selected for the conflict at hand. In Figure 3.3, Michael (Scenario 3) can solve the conflict on his living spots by selecting the desired state for that moment. How this solution is exactly applied will be further discussed in Section 3.5. Once a solution is applied by the system, the application makes sure the user is aware of this. As seen in Figure 3.4a, Michael can see that the state of the living spots turning on is marked by a green check icon.

In general, a solution is marked by using the green check mark icon. Every state change that is involved in, and thus the result of, the same solution will be marked. Again we use the same method as with conflicts to group solutions when selecting them. As seen in Figure 3.5, Sarah (Scenario 1) can see that two state changes are part of the same solution. When a state change is part of a solution as well as part of a new conflict, they will appear to have both markings. On selection, conflict grouping gets priority over solution grouping.

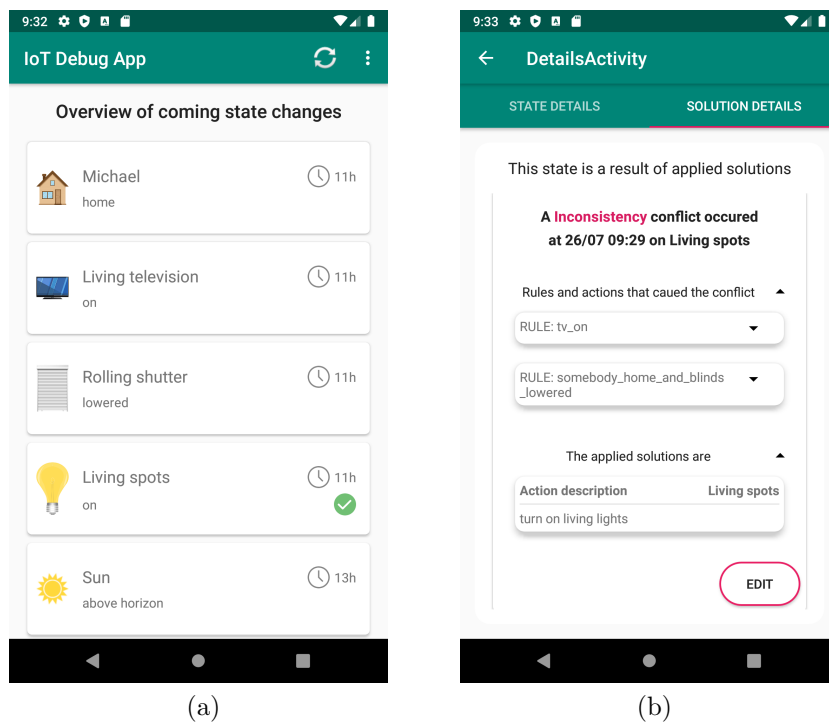


Figure 3.4: Once Michael applies a solution, this overview is displayed with the solution state change marked with a green check mark (a), and more details about the solution can be requested (b).

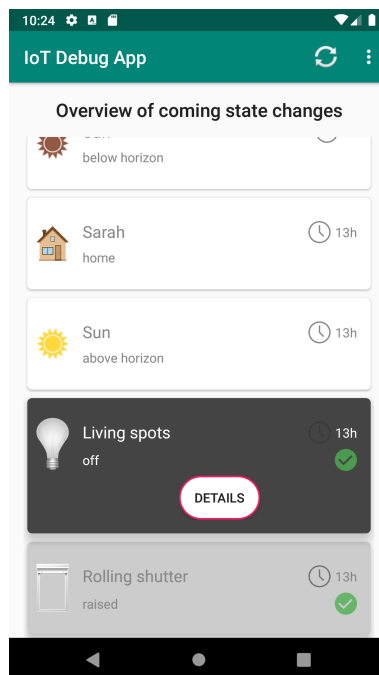


Figure 3.5: Sarah (Scenario 1) can see that both state changes belong to the same solution.

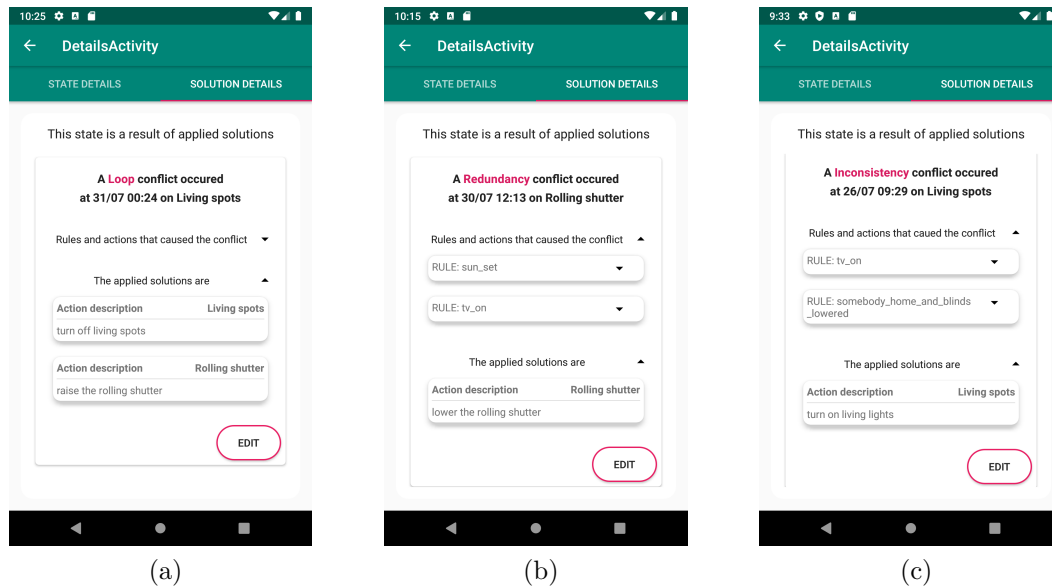


Figure 3.6: In the details of a solution, more information is presented about the conflict (conflict type and conflicting rules) it solved, as well as the chosen action to be executed.

When Michael selects the state of the living spots turning on and clicks on the Details button, he will see an extra tab that holds the information of the solution that has been applied (see Figure 3.4b). Here the user can find all the information to understand what kind of conflict has been solved and how. As we can see in Figure 3.6, this information contains the type of conflict, the date when this conflict will occur, the rules that are involved in the conflict and the actually actions that will be executed as a result of the applied solution. It is important to point out that it is definitely possible for multiple devices to be involved in a solution (as seen in Figures 3.5 and 3.6a). Therefore it is also important for a user to get an overview of all the involved devices and applied actions on them when they are part of the solution. Michael can see that the selected state was the result of solving an inconsistency conflict (see Figure 3.4b). If Michael does not like the applied solution, he is offered the chance to edit the solution, thus changing it to his preferred state.

3.3 Predicting status: Near Future Prediction

In order for us to detect conflicts at run time before they actually happen, a major component is needed to predict this potential future. Sven Coppers et al. introduce FORNTIoT [63], a framework that is built on top of Home Assistant [59], that provides an approach to predict future smart home behaviour. Figure 3.7 presents their proof-of concept implementation of FORNTIoT. This architecture consists of three main components: The Context Manager that keeps track of the current states of all the entities, a custom Rule Manager that allows the simulation of trigger-condition-action rules and the Prediction Engine which implements the prediction algorithm.

To predict the future, the framework uses all the data it receives from the different

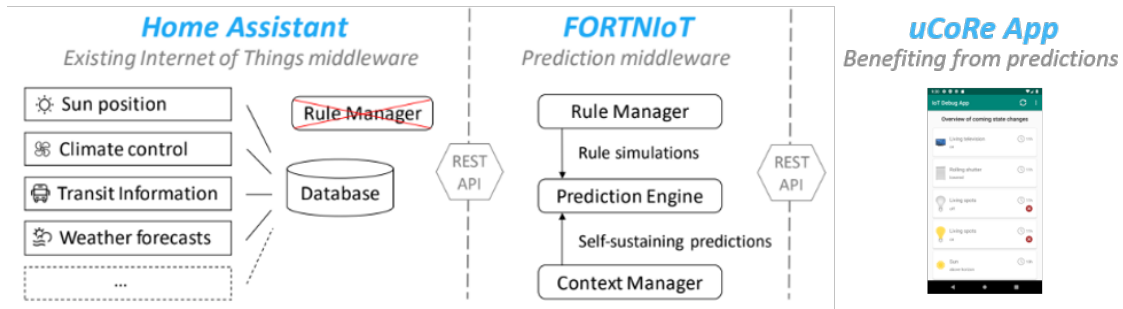


Figure 3.7: Proof-of-concept implementation of FORTNIoT, built on top of Home Assistant [59], an existing middleware platform to manage IoT entities in a smart home. A custom rule manager was implemented to allow the simulation of trigger-condition-action rules. The uCoRe App offers visualization and resolving means for conflicts.

sensors and information sources, and then validates which rules will get triggered and thus which actions are executed, resulting in new state changes of the devices. For example, the framework can exactly know whenever a sun rise or sun set will occur or even when something is planned in the calendar of the inhabitant. All these different events happen at a specific time, and some of them might even occur at the same time. Using time ticks, the prediction engine groups these events based on the time of occurrence. Once events are grouped in their specific time tick, the prediction of all the other states starts.

In Figure 3.9 a visual example of Scenario 2 is given of the algorithm predicting new states starting from events in a single time tick. In this specific example, the time tick starts with the following events: the sun goes down and Alex is home (Figure 3.9a). These are the starting nodes and form the first layer. All nodes in a single layer are visited and activated (Figure 3.9b), resulting in potential rules that could be triggered (Figure 3.9c). Once the condition of a rule is satisfied, for example Alex is really home, the rule is executed, resulting in new states, thus new nodes. From there, the following layer is next. In combination with all states earlier generated from all nodes above, the new nodes are visited and activated, the rules are triggered and validated and new nodes are acquired. This keeps on going until no new nodes are found. Then, the engine moves on to the next time tick until that runs out as well.

During the prediction stage, two things are very important to be aware of: that is race conditions and loops. Race conditions occur when two rules are executed at the same time, and thus race to be the first executed. This results in not always getting the same order of nodes and states in the tree as presented in Figures 3.8 and 3.9. Since states (nodes) can trigger rules, and if we would visit those nodes one by one, the prediction engine could be sensitive to the consequences of race conditions, leading to different rules or no rules at all to be executed due to the order of execution. By developing the tree layer by layer, treating all the nodes in the same layer at the same time, race conditions are prevented. Due to this, it does not matter in which order the nodes are within a layer. Once in that layer, all the states of the nodes within that layer are known and used, and together

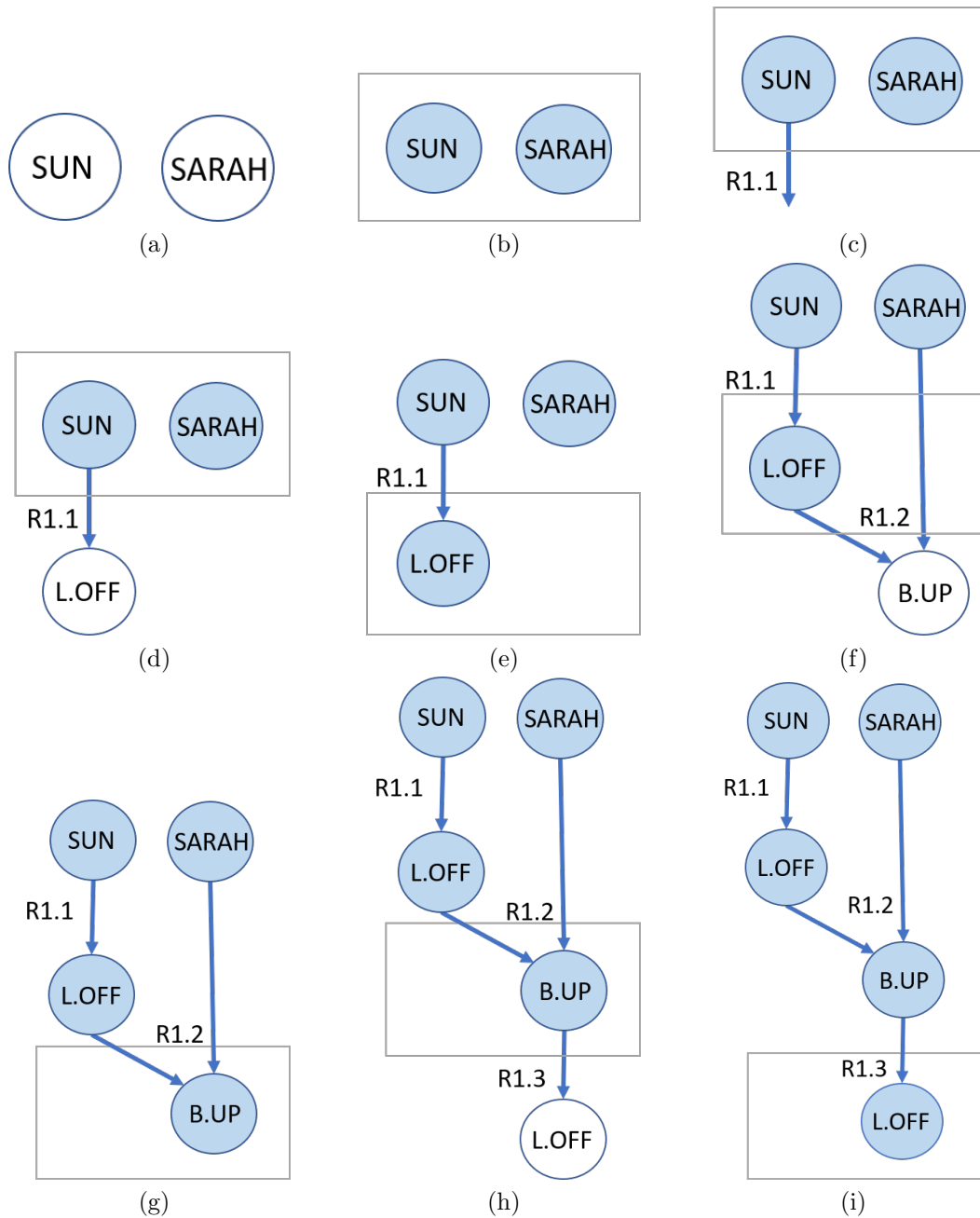


Figure 3.8: Scenario 1: near future prediction of a single tick. Beginning with the starting nodes (a), they are visited and activated (b), resulting in triggering rules (c) which leads to new nodes (d). This process is repeated for the next layers until no new nodes are found (e - i).

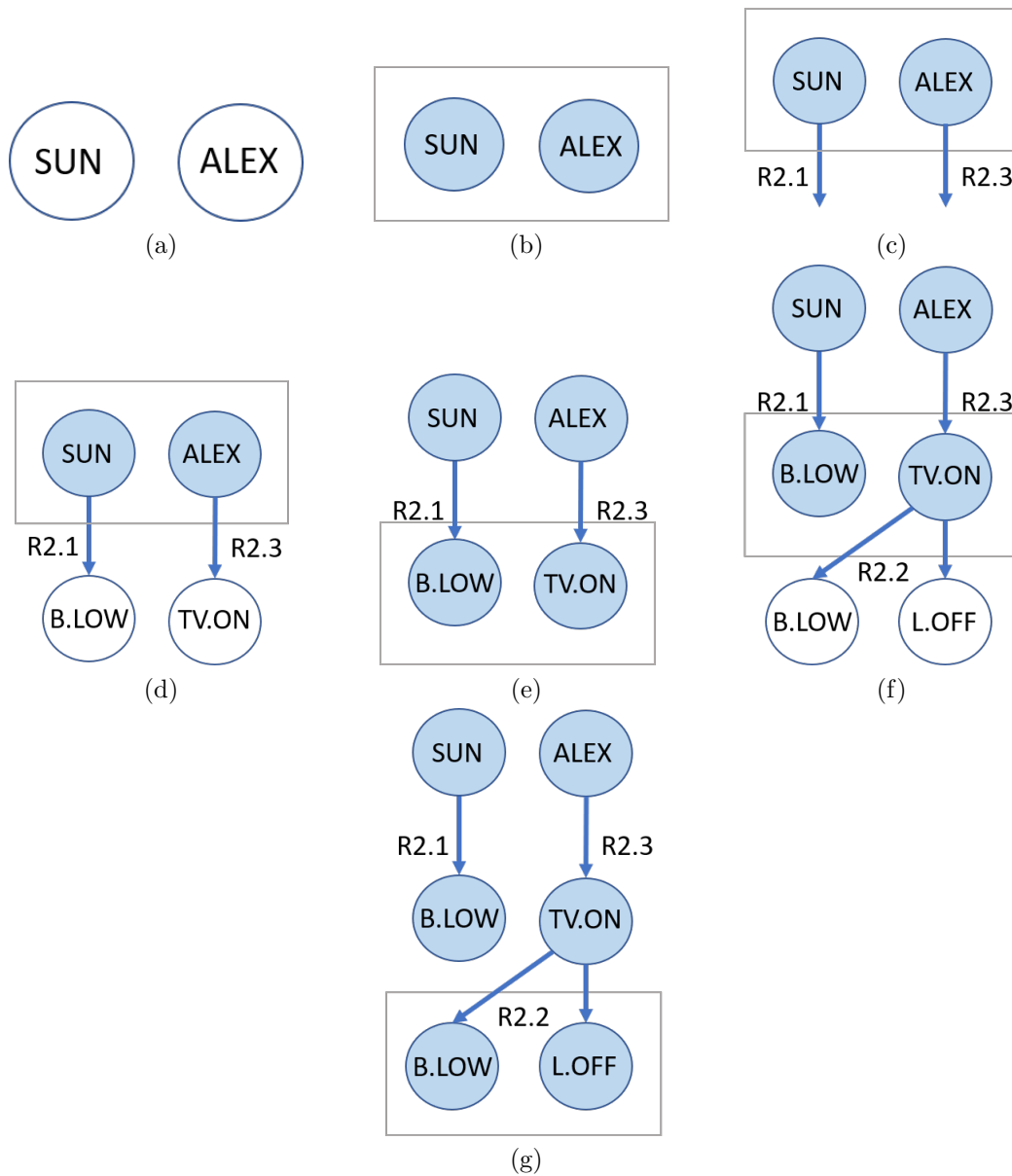


Figure 3.9: Scenario 2: near future prediction of a single tick. Beginning with the starting nodes (a), they are visited and activated (b), resulting in triggering rules (c) which leads to new nodes (d). This process is repeated for the next layers until no new nodes are found (e - h).

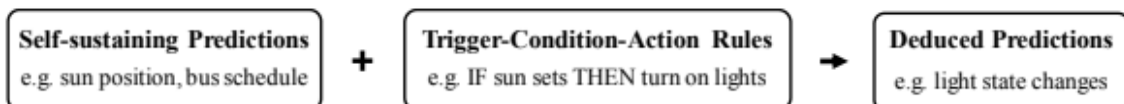


Figure 3.10: The core principal of FORTNIoT. Based on self-sustaining predictions and trigger-condition-action rules, additional predictions about the future can be deduced.

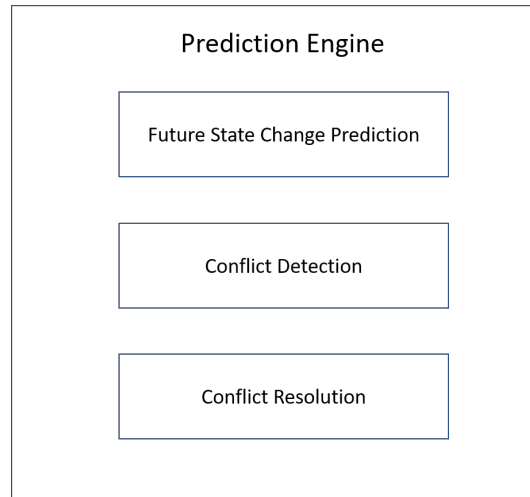


Figure 3.11: The three components of the Prediction Engine: Future state changes are predicted, then conflict detection is started and once conflicts are found existing solutions are applied.

they trigger the rules, resulting in new nodes (states). A loop would cause the prediction to get stuck, and thus always find a new node. Scenario 1 is an example of a system which contains a loop, and Figure 3.8 is the visual example of how the prediction algorithm would handle this. When looking closely, Figures 3.8e and 3.8i are practically similar. The same node is visited and activated, which in both cases will result in triggering the same rule (R1.2), leading back to Figure 3.8f. Loops are a type of conflict, and thus we will discuss this further in the next Section 3.4.

3.4 Conflict detection

Before being able to visualize a conflict and solve it, it needs to get detected by the system first. The conflict detection does not take place in the application itself, rather it is implemented in the FORTNIoT framework, written in Java. This framework uses near future prediction in order to come to a possible future, containing device states, rule executions, conflicts and applied solutions. We therefore extended the Prediction Engine (as seen in Figure 3.11), which now does three main things: predicting the future state changes, detecting conflicts during the prediction and resolving conflicts during the prediction. Figure 3.12 visualizes the flow of the prediction in a single time tick. We argue that there are multiple reasons for doing the conflict detection during the prediction stage instead of in the application, post prediction.

The main reason would be the conflicts themselves. Loops have the tendency to keep on going, so they have to be detected and stopped once they occur, otherwise the prediction will get stuck. Therefore a detection mechanism must be in place in order to achieve this during the prediction stage. Inconsistencies, however, do not contribute in any way during the prediction stage. As we already mentioned before (see Subsection 2.4.1), an inconsistency occurs when the actions of two or more rules operate on the same device, and result in an opposing state (e.g. lights

on vs lights off). If the state of a device is unknown during the prediction stage because of an inconsistency, it is impossible and illogical to predict any other states by executing the rules that have the conflicting entity as a condition. For example, when in Scenario 3 the TV would be turned on and turned off at the same time due to a conflict, the state of the TV becomes unknown because it can not be both. Therefore rule R3.2, which has the TV state as its condition, should not be executed, and thus prevented from generating new state changes (nodes). When it comes to redundancies during the prediction stage, they do not change the behaviour of the prediction, because both actions in this type of conflict do the exact same thing, resulting in the same rules being executed. In that case, no prevention is really needed.

Besides the problems conflicts bring during the prediction stage, they are also tied to that specific future and the predicted states. When a new prediction is made about the future, and thus future state changes are updated, potential conflicts should be updated as well. With all the information about the state changes, and the rules and triggers that caused them, inside the prediction engine, conflict detection could use it to give more precise feedback of all the responsible entities that caused the conflict in the first place.

As already mentioned, resolving a conflict also happens during the prediction stage, but in order to resolve conflicts during prediction the system should first be able to find them. More about resolving conflicts in the next Section 3.5.

Figures 3.8 and 3.9 show the structure that is created during the prediction of state changes in the same time tick. Conflicts will be found in this structure. A conflict is identified when more than one node contains the same device, meaning that more than one state change occurs on the same device. We go more into detail about the exact algorithm in Subsection 3.4.2.

3.4.1 Data structure

In order to store and communicate conflicts between the framework and the application, we used JSON and the Jackson library to do any (de)serialisation from and to Java objects. Conflicts are not permanently stored in the system, but are always recreated with every new prediction the system makes.

When observing the JSON object in listing 3.1, a conflict is defined by an entity id on which the conflict occurs, a timestamp that identifies the moment in time when this conflict will take place and the conflicting actions themselves. Each of these actions are represented by their action id and rule id. It is possible for conflicting actions to not have a rule id. The reason for that is because it can be possible to have actions in the system that aren't bound to any rule, but still can be executed in some way, and thus become part of a conflict.

A conflict is identified by its entity id and the set of actions that are in conflict with one another. Although for inconsistencies and redundancies this looks exactly the same, loops, however, contain not only the actions that affect the same entity, but also contain every other action that is part of the same loop. It is perfectly possible that the actions in a conflict not all affect the same device. When this happens, the conflict depicts a loop.

```

1 {
2     "entity_id": "light.living_spots",
3     "datetime": "2019-12-10T14:02:39.265+01:00",
4     "actions": [
5         {
6             "action_id": "actionId1",
7             "rule_id": "rule.sun_rise"
8         },
9     ],
10    "action_id": "actionId3",
11    "rule_id": "rule.person_at_home"
12    }
13 ]
14 }

```

Listing 3.1: JSON representation of a conflict. A conflict is represented by the id of the entity the conflict occurs on, the time the conflicts occurs and the actions that are in conflict with one another, identified by the action id and rule id to which the action belongs to.

3.4.2 Conflict detection algorithm

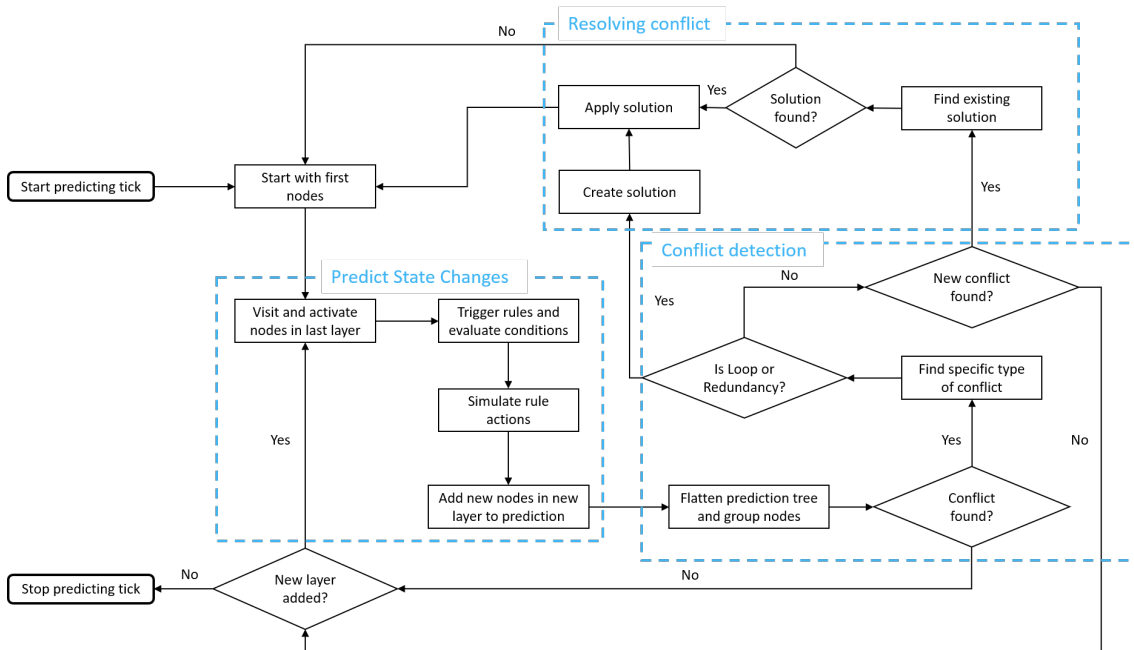


Figure 3.12: The flow of the prediction algorithm. First, given the starting node, new nodes are predicted. Then, conflict detection will be done over all the gathered nodes up until that moment. At last, solutions are created or found and applied if conflicts were found. Using the newly found nodes, this process repeats itself until no new nodes are found.

There are three different types of conflicts that can be found during the prediction: inconsistencies, redundancies and loops. All three types are found in the structure that is created as shown in Figures 3.8 and 3.9. Let us start off with Scenario 1 and Figure 3.8. This scenario contains a loop and its rules are the following:

- R1.1: IF the sun comes up, THEN turn off all the lights
- R1.2: IF the lights are off AND Sarah is home, THEN raise the rolling shutters
- R1.3: IF the rolling shutters are raised, THEN turn off all the lights

When the sun comes up and Sara is home at the same time, the time tick of this prediction would start with the nodes as seen in Figure 3.8a. These nodes are visited and activated, meaning that their state changes are applied and thus will trigger potential rules. In this instance, only rule R1.1 will be triggered. After validating the condition (e.g. the sun is indeed up), new nodes are added to the tree-like structure (Figure 3.8d). Before visiting and activating the new layer of nodes, it is important to look for conflicts among all the nodes gathered so far. This is done before activating a new layer, in order to make sure that no rules are triggered when there are conflicts present.

This detection is done by flattening the tree and grouping the nodes that affect the same device. When more than one node affects the same device, we consider it a conflict. In Scenario 1 this occurs for the first time in Figure 3.8h. This results in marking the nodes as a conflict and group them together (see Figure 3.13a). Once a conflict is found, it is important to determine what type of conflict it is. When the nodes in the conflict affect the same device and have opposite states (e.g. light on vs light off) an inconsistency is found. In this scenario, however, both nodes contain the same state change (e.g. turn the lights off). This could mean two things: the conflict is a redundancy or it is a loop. The algorithm checks if it is a loop by starting with the highest node of these conflicting nodes in the overall structure. From there it follows the arrows down until it finds the other conflicting node or no node at all. In Figure 3.13b we can see that in this scenario both nodes reach each other, meaning that a loop is found.

When the type of conflict is determined, the Prediction Engine logs the conflict and tries to find solutions for it if there are any (see Section 3.5). If no solutions are found, the Engine wants to prevent the tick from having potential nodes that were resulting out of conflicting nodes. To be sure that no unwanted nodes, resulting from conflicting states, are in the prediction, the prediction restarts again from the first nodes in that same time tick (Figure 3.8a). The difference now would be that once the same nodes that occurred in the conflict are found in the prediction, they are ignored, meaning that they are not activated and thus can not trigger other rules. This prevents the system from generating state changes that can be affected by an earlier found conflict. So in a second run for this specific scenario, the first node with the state change “lights off” (Figure 3.8d) will be found, but not activated as in Figure 3.8e. This results in the end of the prediction for this time tick.

In Scenario 2 and Figure 3.9 a redundancy conflict can be found. The rules of this scenario are the following:

- R2.1: IF the sun goes down, THEN lower the rolling shutters

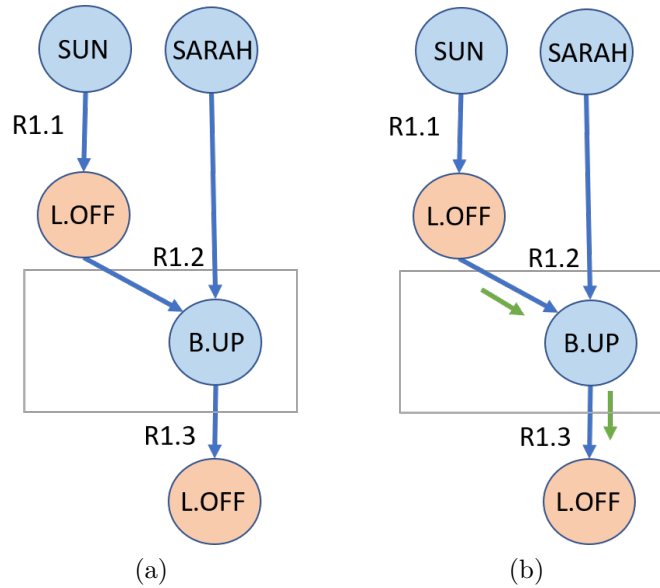


Figure 3.13: Scenario 1: During the prediction a conflict is detected because two nodes affect the same device (a), and starting from the highest node in the conflict a loop check is done by following the arrows down (b). If the other node of the conflict is found, a loop is detected.

- R2.2: IF the TV is on, THEN lower the rolling shutters AND turn of the lights
- R2.3: IF Alex is home, THEN turn on the TV

When the sun goes down and Alex is home at the same time, both will appear as starting nodes in the same time tick as seen in Figure 3.9a. Again, these nodes are visited and activated, resulting in potential rule executions. Once the rules are validated, new nodes are added to the tree-like structure. After adding the new nodes, but before visiting and activating them, conflict detection is done. It is only until Figure 3.9f that a conflict can be found on the rolling shutters (see Figure 3.14a). To determine the type of the conflict, the state changes of the involved nodes are inspected, leading to the findings that both nodes affect the same device with the same state change. Again, a loop check is done but this time no loop is found (see Figure 3.14b). Because a conflict is found, the prediction of this specific time tick starts all over again from the starting nodes, ignoring all nodes that are marked as a conflict along the way. Conflict detection for Scenario 3 would look similar to that of Scenario 2, with the difference that state changes now have opposite actions on the same device.

All these scenarios are simple examples of the different types of conflicts, but things can become more complex when more nodes are added. In all these scenarios only two nodes appear to be in a conflict with each other, but it is perfectly possible for more than two nodes to be in a conflict with one another. It is therefore important to group all the nodes affecting the same device, making the largest group possible, in order to best represent all the rules and actions that conflict with each other at that given time.

Although in the end all three types of conflicts are looked for during each predic-

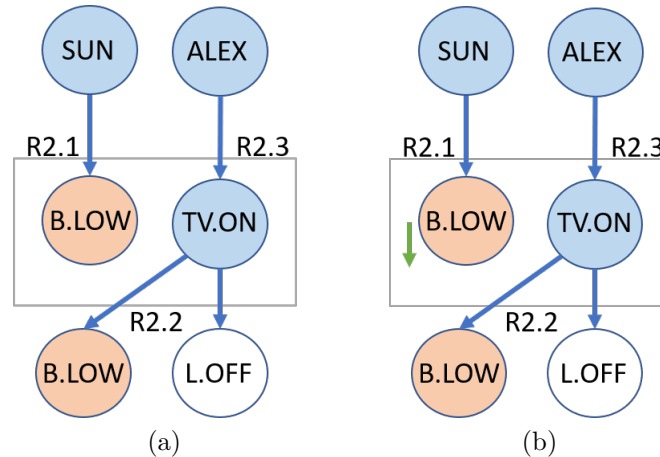


Figure 3.14: Scenario 2: During the prediction a conflict is detected because two nodes affect the same device (a) and starting from the highest node in the conflict, a loop check is done by following the arrows down (b). No other node of the conflict is found, thus a redundancy is detected.

tion time tick, not all of them are done at the same time. Loops and inconsistencies are detected for every layer, once the new nodes are added to the tree-like structure, as seen in the examples previously. This for the simple reason that, as mentioned at the beginning of this subsection, both affect the ongoing prediction of state changes in the time tick. Loops do not stop for themselves and inconsistencies bring uncertainty inside the prediction, and thus both need to be detected as soon as possible and prevented from doing any harm. Redundancies, however, are detected only once the full structure has been found. This means that in the end of predicting a time tick, redundancies are found as in Scenario 2 and logged. If such a conflict is found in the end, the Prediction Engine would start the prediction of that time tick one more time all over again to filter out all the nodes that contain the same state change as the ones in the redundancy conflict. The main reason why redundancies are detected at the end of the prediction tick is not only as being it a more efficient way of doing this type of detection, but also because of all the complications it could give during the detection and the applying of solutions as we will discuss in the next Section 3.5.

3.5 Conflict resolution

Resolving a conflict can be achieved in two different ways. A conflict can be automatically resolved in the prediction stage by the Engine itself, or a solution for the conflict has to be created using the application (see Subsection 3.2.3). The way the conflict resolution is achieved is dependant on the type of conflict. Loops and redundancies are both detected and solved immediately by the Prediction Engine, during the prediction of state changes. They both do not require any user feedback to solve. Loops just have to be executed once and then stopped, while redundancies are actually multiple state changes that achieve exactly the same thing (e.g. all turn the lights off). Inconsistencies, on the other hand, require a user's input in order

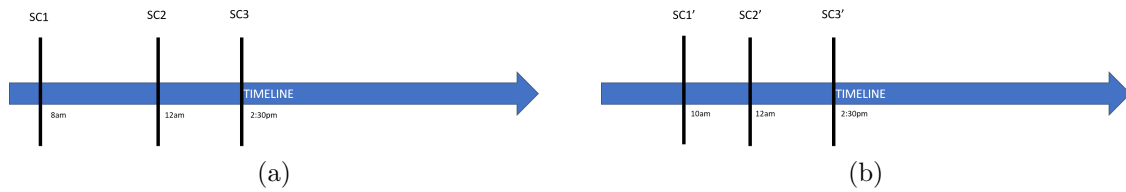


Figure 3.15: Predictions of state changes (SC) can shift in time. (a) shows the predictions of a few state changes, where in the next prediction (b) state change 1 (SC1) moves in time.

for the system to know what action should be taken (e.g. light on vs light off), thus our application is needed to provide an interface for it.

It speaks for itself that a solution solves a specific conflict. This would mean that a solution is tied to that one conflict. In Scenario 3, Michael has to deal with a conflict on the living spots, which is a result of two rules and actions that result in an inconsistency (see Figure 3.2c). By using the application’s solver, he decides to turn on the living spots (see Figure 3.3b). What this means is that for this specific conflict, that is the result of these two actions, the preferred state of the living spots is to turn them on. To be able to identify this specific conflict, we decided to use the entity id and the set of conflicting actions occurring together. For now, we do not use the time as an identifier for the conflict because of the following reason: we detect conflicts in state change predictions of the near future. These are not all just facts, thus it is not guaranteed that these state changes are really happening in the future at the predicted time. Figure 3.15 gives an example of some state changes that are predicted at a given time, but at a later prediction update some state changes moved in time. A good example of this might be the prediction of when someone comes home. At one point in time the system would predict that Michael is home around 5pm, while the next time this prediction can be moved up if Michael would get stuck in travel. Because state changes can move in time, conflicts can move with them as well. As mentioned before, a solution should be tied to a conflict, even when it moves in time. Therefore a solution uses only the entity id and the conflicting actions as identifiers for the conflict.

3.5.1 Data structure

In order to store and communicate solutions for conflicts between the framework and the application, we used JSON and the Jackson library to do any (de)serialisation from and to Java objects, similar to conflicts. In contrast to conflicts, a solution is permanently stored in the system, unless the user removes the solution from it.

When observing the JSON object in listing 3.2, a solution is defined by an entity id on which the conflict occurred, the actual actions which contributed to the conflict itself, the actions which are active or snoozed and custom actions. Each of these actions, except for the custom actions, are represented by their action id and rule id. Custom actions contain more information about the specific type and the attributes of the action that has to be recreated in the framework.


```
1 {
2     "solution_id": "light.living_spots_actionId10
3         _actionId12",
4     "entity_id": "light.living_spots",
5     "conflicting_actions": [
6         {
7             "action_id": "actionId10",
8             "rule_id": "rule.
9                 nobody_home_lights"
10        },
11        {
12            "action_id": "actionId12",
13            "rule_id": "rule.blinds_up"
14        }
15    ],
16    "snoozed_actions": [
17        {
18            "action_id": "actionId10",
19            "rule_id": "rule.
20                nobody_home_lights"
21        },
22        {
23            "action_id": "actionId12",
24            "rule_id": "rule.blinds_up"
25        }
26    ],
27    "active_actions": [],
28    "custom_actions": [
29        {
30            "deviceID": "light.living_spots",
31            "actionID": "actionId0",
32            "id": "actionId0",
33            "description": "Custom solution
34                action",
35            "enabled": true,
36            "action_name": "sven.phd.iot.
37                rules.actions.LightOnAction",
38            "start_time_disable": [ ],
39            "stop_time_disable": [ ]
40        }
41    ]
42 }
```

Listing 3.2: JSON representation of a solution. A solution is identified by its own id and contains more information about the specific conflict it solves, using the entity id of the entity that is in conflict and the conflicting actions upon it, as well as the actions that the solution will execute in order to solve the conflict. This latter is represented by the snoozed actions, active actions and custom actions.

3.5.2 Apply a solution

Before applying any solution to a conflict, it is first good to understand what a solution really does. As seen in the previous Subsection, a solution contains three different actions it can execute to solve a conflict; snoozed actions, active actions and custom actions.

Snoozed actions are actions that will not be activated and executed when they appear during the prediction stage. They do not trigger any rule and appear to be snoozed, thus they do not lead to generating other nodes during prediction. In Scenario 2, for example, rule R2.3 states that “IF Alex is home, THEN turn on the TV”. The action that turns on the TV can be marked as a snoozed action, which would mean that during the prediction of this time tick, instead of activating the node (like in Figure 3.9e), leading to the execution of rule R2.2 (see Figure 3.9f), this node is ignored and not added to the prediction.

Active actions are actions that will be activated and executed once they appear during the prediction stage. These actions actually do not alter the normal behaviour of the Prediction Engine, which always visits and activates all the nodes in a new layer, except for conflicting actions or snoozed actions.

A *custom action* is an action that can be created by a user using the application, and represents an action that was not originally in the conflict. In Scenario 3, Michael can solve a conflict on the living spots by selecting the preferred state of the living spots. He can chose to turn off the lights or turn them on. The latter gives him the option to customize the color of the light (see Figure 3.3b). The actual actions that caused the conflict turned the light off or turned it on and set the color to blue. When he decides to turn the lights on and set the color to green, a new action is needed to represent this wanted behaviour, because this was not an original conflicting action. This can be also the case in Scenario 2, where the redundancy conflict only contains actions that lower the rolling shutters. If Alex wants the rolling shutters to be raised, a custom action has to be created to represent that behaviour, because in the original conflicting actions no action exists that raises the rolling shutters. Therefore custom actions are created and used to let the Prediction Engine know that these states are not the result of generating nodes and executing rules, and thus needs to be handled differently. So instead of activating or snoozing nodes, a custom action results in a new node being created and added to the first layer. In Scenario 2, for example, a node that raises the rolling shutters would be added in Figure 3.9a, next to the nodes “SUN” and “ALEX”. Of coarse, a solution consists of a combination of snoozed actions and active- and/or custom actions.

We mentioned previously that solutions can be created automatically or by the user using the application. In Scenario 1 and 2, solutions are created automatically,

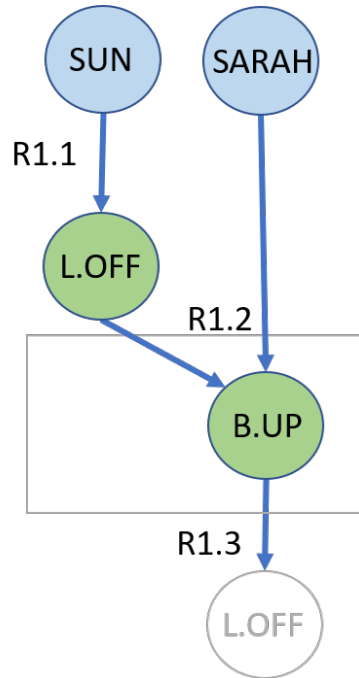


Figure 3.16: Solving a loop conflict during prediction: active actions are marked green, and keep being executed, while snoozed actions are marked grey, meaning that these nodes will be ignored and not be added to the prediction again after a restart of the prediction tick.

because the system solves a loop and a redundancy conflict. Let us take a closer look at both scenarios. Scenario 1 contains a loop. During the prediction stage a loop gets detected by the engine as previously discussed in Subsection 3.4.2. Once the loop is detected, a solution is created immediately (see Figure 3.12). We also mentioned earlier that a loop conflict is the only conflict that contains conflicting actions that affect more than one device. In this scenario, the conflicting actions are affecting the living spots and the rolling shutters. The solution tries to achieve that the loop is executed once, but stops after that. In order to achieve this, every action belonging to a node, starting from the highest node in the loop chain, is marked as an active action, except for the last one, which turns the lights off again. This last action is snoozed, thus preventing the loop from getting started again. This can be visualized as in Figure 3.16, where active actions are marked green and snoozed actions are marked in grey. Once a solution is applied, a restart of the prediction is required in order to adequately add the right nodes and actions to the prediction of the future.

Scenario 2 is an example of a situation that contains a redundancy. We mentioned in the previous section that redundancies are only detected at the end of a prediction tick, once every node is found, and every other conflict is detected and/or already solved. There are a few reasons for that. One of them is solving a redundancy results in the conflict being logged (with these specific actions as the cause) and one action being marked as active and the others as snoozed. If this would be done during prediction it might look something like in Figure 3.17: in the first run a redundancy is found on the living spots and solved by marking one action as active

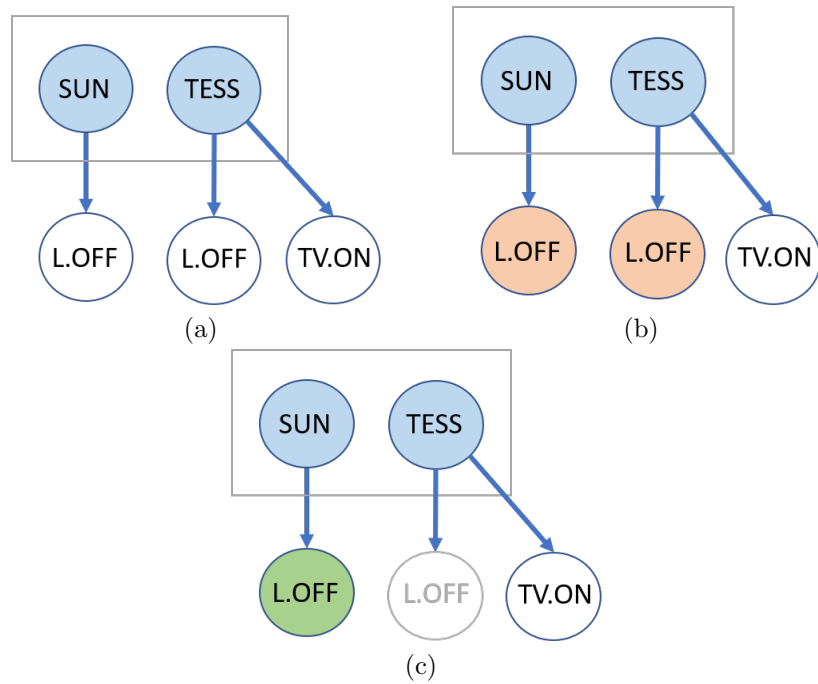


Figure 3.17: First run of fictive redundancy scenario where redundancy is found in the second layer (b) and automatically solved by marking the first action as active and the second action as snoozed (c).

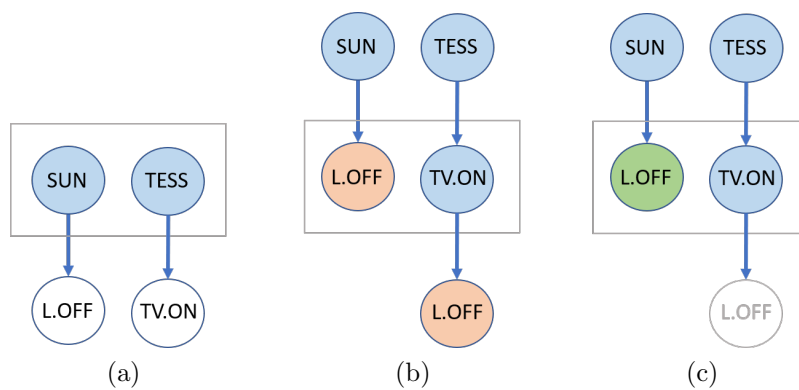


Figure 3.18: Second run of fictive redundancy scenario where new redundancy is found in the second and third layer (b) and automatically solved by marking the first action as active and the second action as snoozed (c).

and the others as snoozed. In the rerun (see Figure 3.18), the solution is applied but its active action will end up in another redundancy conflict. Again, this will automatically be solved by the Engine. In general what should happen is that there is only one redundancy conflict on the living spots, and it is solved only once. By detecting and solving redundancies for every layer instead of only once at the end, is not only inefficient, it also logs the existence of multiple redundancy conflict at the same time on the same device, thus solving all of these conflicts multiple times as well.

Solving redundancies during prediction also changes the identification of other conflicts. As mentioned previously, a conflict is identified by the entity on which the conflict occurs, as well as all the actions that are involved in the conflict or are causing the conflict. If an inconsistency conflict occurs because action1, action2 and action3 are all happening at the same time, we want to see this reflected in the identification of the conflict. But if action1 and action2 are also a redundancy, the identification of the inconsistency conflict can take many different forms. Possible identifications for this conflict are the following:

- action1 - action2 - action3: this occurs when the inconsistency is found before the redundancy
- action1 - action3: redundancy is found before inconsistency and solved by marking action1 as active
- action2 - action3: redundancy is found before inconsistency and solved by marking action2 as active

All three of these identifications are possible and sensitive to race conditions. Not only is it possible that the identification of a conflict is more generalised (simplified) and not very specific anymore for that situation, as the last two examples show, tying a solution to a conflict whose identification is dependant on the order of execution is not desired. Besides, having a redundancy conflict during prediction is not a problem because it does not alter the behaviour of the engine during its prediction. No other rules would or would not be executed if the redundancy is solved or not, because these actions and their state changes are doing the exact same thing, triggering the same rules, and thus generating the same nodes. Therefore we decided to do a redundancy conflict detection at the end of a prediction tick.

Inconsistency conflicts are solved a bit differently. These conflicts are not solved automatically but need the user's feedback on how to handle this conflict. Once the user solved the conflict using the application, the solution is stored in the FORT-NIoT framework. During prediction, when an inconsistency conflict is found, the system looks for a solution that exactly matches the identification of that conflict. It compares entity id's and the conflicting actions. When a perfect match is found, the custom actions in the solution are added to the first layer of the prediction and the actions that should be snoozed are marked. The solution is applied. After that, the prediction of that time tick starts all over again.

3.6 Limitations

Some limitation can also be found in this work. To start off, conflict detection across time ticks is not possible due to the fact that conflict detection is only done during the prediction stage inside and bound by a time tick. Thus, actions have to occur at exactly the same time for it to be a conflict. If a user would find it a conflict if the light is turned on and, for example, one minute later turned off again, the system will not be able to detect that and solve it.

Solutions only exist out of operations on actions: activating, snoozing and creating customs. We decided that this way of solving conflicts gave more room to customize behaviour for a specific moment in time. For example, rules with two actions can still be triggered, but instead of executing both actions, it only executes one of them, making this approach more flexible than doing operations on rules instead. However, combining both methods might be interesting and more complete for a user to have and use.

In this work we focused on three types of conflicts: loops, redundancies and inconsistencies. All three of these are detected during the prediction stage in a time tick. Users, then, have the opportunity to control the desired state of these devices that are part of a conflict, by solving the conflict, as well as devices that are part of a solution, by being able to edit them. When a state change is predicted but not part of a conflict or solution, it can not be changed to the desired state of the user, resulting in a conflict of interest. The system can not deal with that at the moment.

One of the biggest limitations at the moment is making sure that if a conflict shifts in time, the solution shifts with it. In this work we generalized the conflict, by only looking at the entity id and conflicting actions, leaving time out of the picture. However, the goal of the application is to solve one time specific situational conflicts, but by generalizing the binding between a conflict and solution in order for it to shift with the conflict in time, the solution can be applied multiple times. It thus not only solves the conflict it was designed for, but also potential other conflicts that look the same. This can result in unwanted behaviour.

Besides that, if a conflict occurred and the user created a solution for it, this solution is stored in the framework. When state changes shift in the next prediction update, that previous conflict could be not existing anymore. Because there is no conflict anymore, no solution will be applied, which might result in behaviour that the user is not expecting. At the moment, the system does not keep track of earlier existed conflicts, and cannot inform users if a previous conflict, that was solved before, disappeared because of an updated prediction.

Chapter 4

Executing a User Study

We performed a small study to evaluate our conflict resolution method. In this experiment we were interested in how users experience the application for conflict detection and resolution and compared these results to a rule editing method.

4.1 Study design

We performed a within-subject study to evaluate the subjective satisfiability and ease of use of the uCoRe application for detecting and solving conflicts in the near future. We postulated following hypotheses:

- **H1.** *The uCoRe application has a **higher success rate** for conflict solving than rule editing.*
- **H2.** *The uCoRe application is **easier to use** for conflict solving than rule editing.*
- **H3.** *The uCoRe application is **subjective more satisfiable** for conflict solving than rule editing.*

4.1.1 Experimental Design and Measurements

We compared two methods for resolving conflicts: rule editing and the uCoRe application. In the rule editing method, participants could add, remove or change rules. Changing rules was done by altering the conditions of the rules, or by adding actions to or removing actions from the rules. The rules were presented to the participants using a Google document, and the rule editing was done in the same document as well. In the application method, participants would see the current state of the scenario, and use the application to solve conflicts if there were any. Due to the COVID-19 pandemic [61], our study was shifted to one-on-one remote video calls between the participant and facilitator. A shared Google document was used between the participant and facilitator to present the different use cases, as well as solving the conflicts using the rule editing method. The facilitator shared the application on his screen and the participants were asked to command the facilitator to do preferred actions on their behalf in the application. The procedure

Use case	Conflict Type	# Conflicts	# Rules	# Entities	# Wishes
Scenario 1	Inconsistency	1	3	4	3
Scenario 2	Inconsistency	2	4	7	6
Scenario 3	Redundancy	1	4	5	4
Scenario 4	Redundancy	1	3	5	3
Scenario 5	Loop	1	3	4	2

Table 4.1: Overview of the scenarios used and their differences in complexity, measured by type of conflict, number of conflicts, number of rules, number of entities and number of wishes. Users wishes are present for participants to understand what the environment is supposed to do.

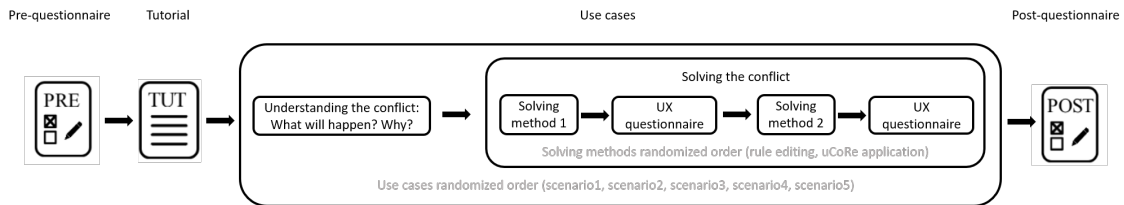


Figure 4.1: The procedure of the within-subject study.

of the study is outlined in Figure 4.1: after the informed consent, participants filled in a *pre-questionnaire* about their demographics and experience. During the *tutorial*, participants read a short manual about trigger-condition-action rules and the two methods they were going to use (Appendix A.1). To compare ease of use and subjective satisfiability between the two methods, participants answered questions related to smart home scenarios presented in five use cases (Appendix A.2). All of these use cases contained a set of three to five rules, a few user’s wishes (e.g. Michael wants the door to be unlocked when he comes home) and one or two conflicts of the same type (two inconsistencies, two redundancies, one loop) (Table 4.1). For each use case, participants were presented with the rules and user’s wishes, and were requested to think aloud about *what* would go wrong in the scenario and *why*. Participants were then asked to express their understanding of the conflict in the scenario using a 5-point Likert scale, and they were asked how they would tackle the problem (Appendix A.3.1). Afterwards, participants tried to solve the conflicts using both methods, each method followed by questions about their trust in the achieved solution and how easy it was to solve the conflict using that specific method (Appendix A.3.2). These questions were also scored on a 5-point Likert scale.

We decide against imposing any time limit for detecting and solving the conflicts. The facilitator continuously monitored the screen and took notes about participants’ think-aloud reasoning. In a final *post-questionnaire*, we queried participants about the two methods using the System Usability Scale [60], and we asked about their tool preferences for achieving certain goals (Appendix A.3.3).

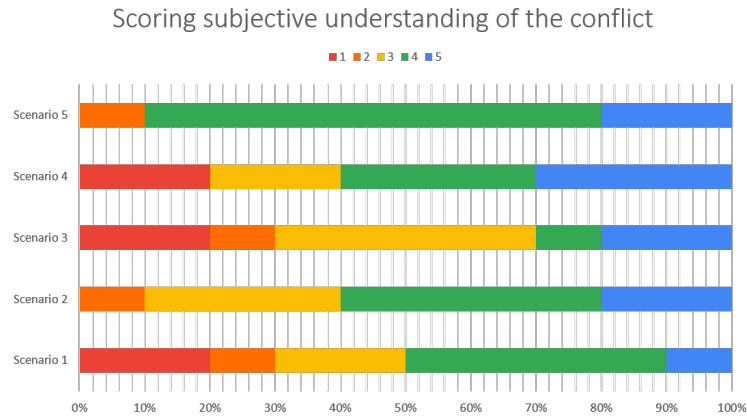


Figure 4.2: Before solving the conflicts, participants were asked to score themselves from 1 to 5, indicating how well they think they understand the conflict in the given scenario.

4.1.2 Participants

Via social media we recruited 10 participants with with varying backgrounds (6 students, 1 retail, 2 computer science and 1 unemployed). No prior knowledge about coding, trigger-condition-action programming or IoT was required. Although 4 participants were experienced to very experienced with programming, only 1 was experienced with trigger-condition-action programming and IoT. In total, 8 male and 2 female participants partake in this study.

4.2 Analysis and results

For binomial data (e.g. whether or not a participant solved a conflict), we used McNemar’s test to determine statistical significance. We used Wilcoxon Signed-rank tests to evaluate differences for Likert-scale ratings about the user experience. In order to measure Usability of both methods, we used the System Usability Scale (SUS) and combined it with a Paired T-test to determine statistical significance. Written qualitative remarks were grouped by the facilitator using thematic analysis.

4.2.1 Scoring understanding of the conflict

When asked participants to score themselves in their understanding of the conflict (Figure 4.2), in four out of five scenarios, 50% or more of the participants were confident in their understanding of the conflict (scoring 4 or 5). Only in Scenario 3, participants showed and expressed that they did not really understand or saw a conflict at all (four participants).

Across all scenarios, although confident, every participant mentioned at least once that they were still unsure if they understood everything, pointing out that it is always possible to miss something (Table 4.2). In scenarios that contained a redundancy (Scenario 3 and Scenario 4), four out of ten participants did not notice any problem or conflict at all, commenting “I don’t see a problem” (four

	Participants	
	Confident (% participants scoring 4 or 5)	Unsure (% out of confident participants)
Scenario 1	50%	20%
Scenario 2	60%	67%
Scenario 3	30%	0%
Scenario 4	60%	33%
Scenario 5	90%	44%

Table 4.2: An overview of every scenario and the percentage of participants that were confident about their understanding of the conflict. Still, these confident participants mentioned that they were still unsure, resulting in the percentage of confident participants that are still unsure about them really understanding the conflict.

participants) or “To me, this is not a problem” (P5). Although participants scored themselves fairly well in their understanding of the conflict, this not always reflected their real understanding. P3 and P8 scored themselves high (respectively 5 and 4), each in one scenario, while they actually did not understand the conflict.

When asked how they would tackle the conflict, for inconsistency scenarios, participants chose to edit rules and/or use priorities, while for redundancy scenarios four to five participants out of ten wanted to ignore the conflict. In the loop scenario, stopping the loop automatically (two participants) or adapting rules were the chosen solution. There was one participant (P2) that, after using the application once, from that point on, mentioned that he wanted to use it before trying to figure something out for himself.

4.2.2 Solving the conflict

All participants were able to solve the conflicts using the application (Table 4.3). When asked why they were able to solve it, participants responded by saying the application was “Straightforward to use” (four) or even mentioned that it was “easier than the rule editing method” (P4). Using the Google docs rule editing method, however, this was not always possible. Our McNemar’s test with continuity correction revealed that the number of the participants that solved the conflict significantly changed between the two methods ($\chi^2(1, N = 50) = 26.036, p < 0.01$). At first, we thought that this might be because in three of the five scenarios a solution was automatically reached by the application, solving the redundancies and the loop for them. When looking at the individual scenarios, we noticed that for Scenario 2, 3 and 4 a significance was found. Scenario 5 (loop) was solved by eight participants using the rule editing method. Possibly this loop was easy to solve because the last rule that created the loop could be simply removed without having unwanted side effects or a conflict with the user’s wishes. Participants had difficulties, using the rule editing method, to solve redundancies (five participant) while fulfilling all the wishes (three participants) or they chose to not solve it because they wanted to ignore the conflict (P7). Scenario 2 contained the most entities, the most wishes and the highest amount of rules, therefore it is possible that this made it harder for

		uCoRe		
		Solved	Not solved	
Rule Edit	Scenario 1	Solved	5	0
		Not solved	5	0
	Scenario 2	Solved	4	0
		Not solved	6	0
	Scenario 3	Solved	3	0
		Not solved	7	0
	Scenario 4	Solved	4	0
		Not solved	6	0
	Scenario 5	Solved	8	0
		Not solved	2	0
Overall	Solved	24	0	
	Not solved	26	0	

Table 4.3: For every scenario, participants were capable to solve the conflict or not. An overview is given of both methods and the ability for participants to solve the conflict.

participants to create a solution. Four participants even mentioned that there were a lot of conditions and a lot to look at, making it hard to solve.

In order to understand how sure participants were about their solution, they were asked how much they trusted their created solution if it would solve the current conflict only once in this specific situation, or this solution would be used to solve this conflict every time it reoccurs (Figure 4.3).

The medians of the Google docs rule editing method and the uCoRe application were 4 and 5, respectively. A Wilcoxon Signed-rank test shows that there is a significant effect of method for trusting the solution for one time use ($W = 1$, $Z = -4.3394$, $p < 0.05$, $r = 0.43394$), for trusting the solution for multiple time use ($W = 1$, $Z = -3.4428$, $p < 0.05$, $r = 0.34428$) and for ease of use for solving the conflict ($W = 1$, $Z = -5.7666$, $p < 0.05$, $r = 0.57666$). Although this is the case over all scenarios, this does not hold up when looking at each scenario individually.

Trust the solution once

In three out of five scenarios, +80% of the participants trusted their solution fairly well (neutral or up) using the rule editing method. Only in Scenario 3 and 4, both containing redundancy conflicts, participants mentioned having more difficulties. This was a result of not seeing any problem to be fixed (P7) or noticing that multiple factors were part of the situation, that made it difficult to come up with a solution (three participants). In the scenarios with an inconsistency, participants mentioned

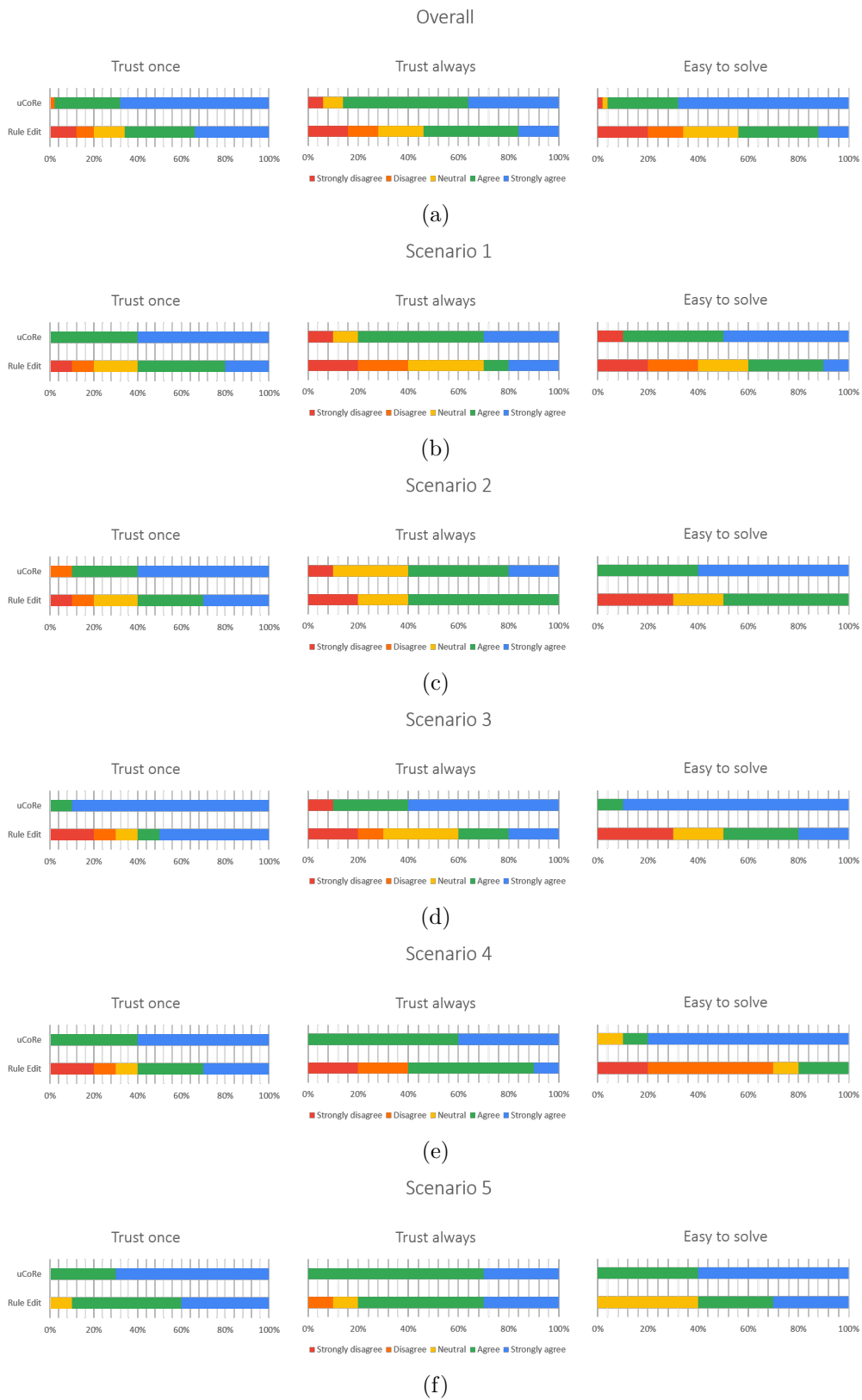


Figure 4.3: Overview of the results after solving the conflicts. Participants scored their trust in their own solution, which they created by using one of the two methods.

that it was difficult to know what the system will do and how it works (three participants). When participants used the uCoRe application, all of them trusted their solutions (agree or higher), because of the following reasons: “it only happens once” (P3), “redundancies do not need solving” (nine participants) or “the system told me that everything was fine” (five participants). Only in Scenario 4 a significant effect of method was found ($W = 1$, $Z = -2.563$, $p < 0.05$, $r = 0.5731042$), with both medians 4 and 5 respectively to rule editing and the uCoRe application.

Trust the solution always

For trusting the system in applying their created solution every time the conflict would occur, participants became a bit more hesitant. “Depending on the situation” or “Depending on the users need” were answers that were given across all the scenarios when asked why they scored their trust in the way they did. Again with redundancies, participants mentioned that they did not need any solving (six participants). In scenarios that contained an inconsistency (Scenario 1 and 2), only when using the uCoRe application, participants mentioned that their solution is dependant on the situation and therefore have difficulties to trust it for applying it more than once (four participants). Again only in Scenario 4 a significant effect of method was found ($W = 1$, $Z = -2.3968$, $p < 0.05$, $r = 0.5359408$), with both medians 4 respectively.

Easy to solve

Participants pointed out that Scenario 4 was the hardest to solve and required a lot of thinking (four participants) using the rule editing method (70% scored a disagree or less). Across all scenarios, using the rule editor, other mentions were that “understanding how the system would think is hard” (two participants), “too many conditions were presents” (two participants) or “it was difficult to visualize the problem” (P5). Using the application, however, Scenarios 3 to 5 were all solved automatically by the application, and thus did not require participants to solve the conflicts themselves. However, from time to time participants mentioned that they were confused that the system solved redundancy conflicts and marked them as solved, because they did not agree with the redundancy conflict being an actual conflict (three participants). On the other hand, solving the loop automatically resulted in the expected behaviour of the participants. When participants were asked how easy it was for them to solve the conflict using a certain method, participants had to give a score from 1 (very difficult) to 5 (very easy). In Scenario 2 ($W = 1$, $Z = -2.8226$, $p < 0.05$, $r = 0.6311525$), Scenario 3 ($W = 1$, $Z = -2.6925$, $p < 0.05$, $r = 0.6020613$), Scenario 4 ($W = 1$, $Z = -2.8308$, $p < 0.05$, $r = 0.6329861$) and Scenario 5 ($W = 1$, $Z = -2.4227$, $p < 0.05$, $r = 0.5417322$) a significant effect of method was found. Only for Scenario 1 this was not the case.

4.2.3 Subjective Usability Scale

At the end of the study, participants scored both methods using the Subjective Usability Scale (SUS). The SUS score mean of the rule editing method was 44.75,

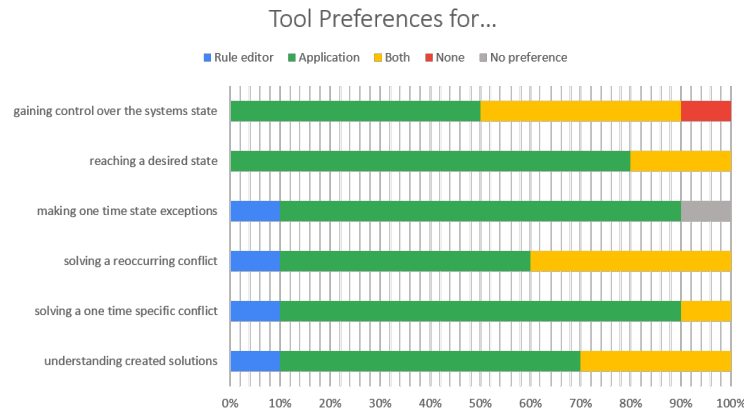


Figure 4.4: Overview of participants tool preferences for certain goals. Overall the application scores really well.

while the SUS score mean of the application was 85.75. A score of 68 is considered average, a score of 80.3 or higher is considered an A (top 10% of scores), while a score below 51 is considered an F (bottom 15%). With a Paired T-test, we found a significant difference in the means between the two methods ($t = 8.13$, $p < 0.05$, Cohen's $d = 2.57$) with the application outperforming the Google docs rule editing.

When asked to express their tool preferences for achieving certain goals, the application scored best for solving a one time specific conflict, or making one time state exceptions (Figure 4.4).

4.3 Discussion

4.3.1 Summary of key findings

Based on the results of this study, we revisit the three hypotheses from Section 4.1:

H1. *The uCoRe application has a higher success rate for conflict solving than rule editing.* Using the uCoRe application, all participants were able to solve all the conflicts presented in the scenarios. This was not the case for participants using the rule editing method. Using the application was “straightforward” and helped them making the conflict easier to visualize and understand.

H2. *The uCoRe application is easier to use for conflict solving than rule editing.* All participants were only able to solve all conflicts correctly using the application. Combined with the results from the Subjective Usability Scale, the application scores high for ease of use. Some participants even mentioned this method to be easier than the rule editing method.

H3. *The uCoRe application is subjective more satisfiable for conflict solving than rule editing.* The uCoRe application shows greatly improved scores on the

Subjective Usability Scale. For certain goals participants even prefer the application instead of the rule editing method.

To elaborate upon the latter, participants preferred the uCoRe application best for solving one time specific conflicts or making one time state exceptions. These results reinforce our motivation for the uCoRe application.

4.3.2 Limitations and future work

As with the implementation of the uCoRe application (Section 3.6), the evaluation also has a few limitations that present opportunities for future work.

Due to the COVID-19 pandemic [61], we shifted to an online study. As a result, it was hard to measure time accurately and draw any conclusions from it. Besides that, we noticed that during the study, in the first scenario presented, although participants had a training in advance, they were still overwhelmed by the rule editing method. Even when using the application for the first time, some participants (three participants) had difficulties knowing how to operate the system. This was due to the fact that they had to command the facilitator for making moves on their behalf. However, both could be avoided when the training would not only contain reading a manual but also a small scenario for the participants to solve in order to get the edge off. Another thing that could solve the problem for the application would be to have participants being able to control the application themselves instead of commanding the facilitator.

Participants also mentioned that it was hard to know what was possible to do in the system using the rule editing method (five participants). Because the rule editing was done in a Google document, there was not many help or guidance with entities or possible actions during the solving stage. This could be resolved when an actual tool or application can be used to do the rule editing with.

Loop and redundancy conflicts are solved automatically in the application. As a result, in three out of the five scenarios participants solved the conflict correctly without having to do anything. Although in most cases this automatic solution might be the desired outcome for these two types of conflicts, our study did not include scenarios where a different outcome was preferred. In a future study, scenarios should be incorporated that challenge the participant to make an exception on the normal behaviour for these two types of conflicts. These scenarios would then require a participant to edit a solution, which in the recent study, was not needed and thus not observed.

Lastly, instead of comparing the uCoRe application to other methods for conflict detection and resolution, a longitudinal study could be set up where participants would live in a smart home and use the application in their day to day lives. This would bring opportunities to see how useful the application would be in a real environment.

Chapter 5

Conclusion

In this work we presented an alternative approach for conflict detection and resolution in an IoT smart home environment. Previous work focused on preventing conflicts to enter the system, by doing conflict detection and resolution at the time of creation for new rules. Instead, we proposed an at run-time conflict detection method that uses near future prediction. This method grants users the ability to in-situ decide what they want their system to do in the near future, providing them with the opportunity to make one-time exceptions on rules by snoozing actions or creating custom ones of their own.

Our work contains four major components: the prediction of near future device states, conflict detection, conflict resolution and an application to bring it all together in the hands of the user. Device state changes are predicted by the FORT-NIoT framework [63] and within this framework we extended its functionality so that it can detect conflicts and resolve them during the prediction of the near future device states. The prediction is done by creating a prediction tree, where the nodes are state changes and the edges are the validated and executed rules. During the creation of this tree, conflict detection is done by flattening the tree and marking actions that are affecting the same devices. Once conflicts are detected, a solution is found by matching the set of conflicting actions and results in custom actions being applied and the others being snoozed.

In order to make sure that the prediction of state changes is as honest as it can be, meaning that only state changes are predicted when the system is sure that these states will be applied given the current information, states that are involved in a conflict should not trigger other rules and result in new states. If they would, a representation of the future would be given to the user that might be totally different once the conflict is solved. It is due to this reason that the detection of conflicts can not happen post prediction, but instead needs to be done during the prediction of new states. Once conflicts get detected, they are logged and marked and lead to a restart of that specific prediction tick.

For solving these conflicts, we chose to use operations on actions instead of altering, editing or disabling rules. Doing operations on actions instead of doing them on rules gives more flexibility in the sense that rules can still be triggered, but instead of all their actions being executed as a result, only the actions that the user prefers can and will be executed. By marking actions as active or snoozed, the

prediction engine can visit or ignore a found state that is caused by an action during the prediction stage. Applying solutions must also be done during the prediction stage because the states that these solutions activate can trigger rules that add even more states to the future, thus resulting in a more complete prediction of the future. What makes solutions so complex is tying them to the conflict that the user wants to solve. Because the system predicts the future, it is possible that this future can change with the next prediction. Therefore it is even possible that conflicts can occur once, can be gone the next time or even shift in time with a few minutes or even hours. In this work we used the set of conflicting actions as the identification of a conflict, which makes it possible for solutions to find these matching conflicts and shift with them through time. The side effect with our approach, however, is that a solution can be applied multiple times, leading to possible undesired behaviour.

In comparison with other at run-time methods, in our approach users are involved with this at run-time conflict detection and solving process by using the uCoRe application. Users get informed about the future state changes through this application, and conflicts get marked by the system. Three types of conflicts are dealt with: inconsistencies, redundancies and loops. Although the system solves redundancies and loops automatically during the prediction stage, users are still presented with the opportunity to edit the solution according to their own preferences. By the use of icons, images and text, the application tries to find the easiest way of clearly communicating relevant information such as conflict details and solution details.

In our online within-subject study, participants were given five scenarios where each scenario contained one of the three types of conflicts and were asked to solve them using two different methods: a rule editing method, which was done using a shared Google document, and the uCoRe application, for which participants had to command the facilitator to operate the application on their behalf. The uCoRe application proved to be easier to use and subjectively more satisfiable than the rule editing method. Participants had to think less about the actual conflict and could focus more on the desired outcome, resulting in a higher success rate of the application for conflict solving. For solving one time specific conflicts or making one time state exceptions, participants preferred the application over the rule editing method, confirming and reinforcing our motivation for the uCoRe application.

There is still a lot of progress to be made in the Internet of Things and the smart home environment. In Section 3.6, we discussed a few limitations of the application and the system at the moment. All these limitations need improving and can be considered as future work. However, we want to extend this future work even further. We will shortly discuss other visualization methods such as AR and VR, the role of AI and intelligibility.

In this work we decided to develop an Android application prototype to get an overview of the devices in the system and their state changes. However, if a lot is happening in the home and a lot of devices are present, it is important to keep a good overview of things when they go wrong. Filter techniques (e.g. show only conflicts, show only solutions, ...) can be used to help with that, but it might also be interesting to look at how AR or VR might aid with this. Promising work has already been done by using AR to inform users about the system [42] or to control

devices [16], and even in VR visual programming is used and a debug system is created [33]. Although VR proves to be useful in some case, in a smart home environment this might not be the first tool a user wants to use to debug their system. AR, however, is useful as a selection method to identify devices by pointing at them and getting more information about its state or even control its settings. It might be even more useful than using a list of devices when a lot of devices are present in the environment. Therefore this seems interesting to incorporate in the application as a device controlling mechanism in a smart home with a lot of devices. If this can also attribute to the conflict detection and conflict resolution aspects of the application still remains an open question.

One of the big things for understanding users and their patterns and habits is using Artificial Intelligence (AI). When a smart home environment can learn more about its inhabitants, it can tailor itself even more to their needs. At the moment our system uses calendar events, sensor information, information from the internet (e.g. sun set and sun rise), etc. . . .to predict the future. User patterns', however, such as certain routines that people have, are programmed into the system. But patterns can change, and therefore these routines should be able to change dynamically as well. When the home can learn its users' patterns on its own, it frees up users to update their routines in the system by hand. In our system, all this information can become incredibly valuable to predict the near future and might even attribute in learning how to solve certain conflicts in a more advanced way. Especially in multiple user smart home environments, were situations can become rather complex really fast, this approach might be really promising. Although the use of AI seems interesting, users still want and need to feel in control. Therefore a smart home system application to control the environment at any time is crucial and the combination with AI should be balanced.

At last, informing the users well about their home is also important. Now that conflicts get detected and solved, an effort has to be made so that users also are able to understand how their solutions work in order for them to trust it. Not only that, but by using feedforward techniques, user can be made aware of the consequences of their solutions. At the moment, when a conflict occurs, users can inform themselves about the conflict and can solve it. As mentioned multiple times before by now, a solution consists out of actions that are active, snoozed or even custom created and these actions can lead to rules being triggered and new states being acquired. However, during the solving stage in the application, a user is not aware of the rules that will be triggered when their solution would be applied. It is therefore possible that a situation can be created that is not totally desired by the user, because now states can be added to the future that were not present before due the the applied solution. Using a feedforward technique, users get informed about what their solution will cause in the future, before it even being applied. This could improve a users understanding and help them make better decisions when solving a conflict. Improving intelligibility is definitely work that still needs further improving.

For users to be able to make exceptions for every device and every rule at any time is very powerful and will definitely prove useful. Our work is the first step towards this kind of control. Ideas of extending the application with also the tools

for more advanced functions, such as rule editing, were out of the scope of this thesis, but mentioned by participants in the study.

To conclude, our work contributes by doing conflict mediation *at run time*, by actively *snoozing or creating custom actions* and *giving users the control* over all of it. Our study showed that for one time specific conflicts our method would be preferred over a rule editing method, and this opens up a lot of opportunities for future research.

Appendices

Appendix A

Executing a User Study

A.1 Training document

Before starting the actual study, participants receive the following training document and are instructed to read it carefully. The goal is to get participants familiar with Trigger-Action Rules and the two methods they will use during the study.

Training

In this study you will use two different methods to solve conflicts in trigger action rules. Here is a brief overview on trigger-action rules and the two methods.

Trigger-Action Rules

- Trigger-action **rules** can automate device behaviour
 - rules defined by inhabitants of the smart home
e.g. sun rises → turn off the living spots
 - rules built in to the device by the manufacturer
e.g. smoke detected → sound the alarm
- Rules are validated when devices that can trigger them change state, **not continuously!**
- **Triggers** are events or conditions that can occur and cause a rule to be validated
e.g. a user action, sensor output, ...
- **Conditions** are statements that need to be true in order to execute a rule
e.g. smoke has to be detected before validating the rule “smoke detected → sound the alarm”
- **Actions** are executed once a rule has been validated
e.g. sound the alarm once smoke is detected

Conflicts

- A **conflict** occurs when rules execute two or more actions that alter the state of the same device at the same time
e.g. sun rises → turn off lights; blinds lowered → turn on lights
- **Inconsistency**: the executed actions result in states of the device that are different from one another
e.g. turn on the lights vs turn off the lights
- **Redundancy**: the executed actions result in states of the device that are the same
e.g. raise the blinds vs raise the blinds

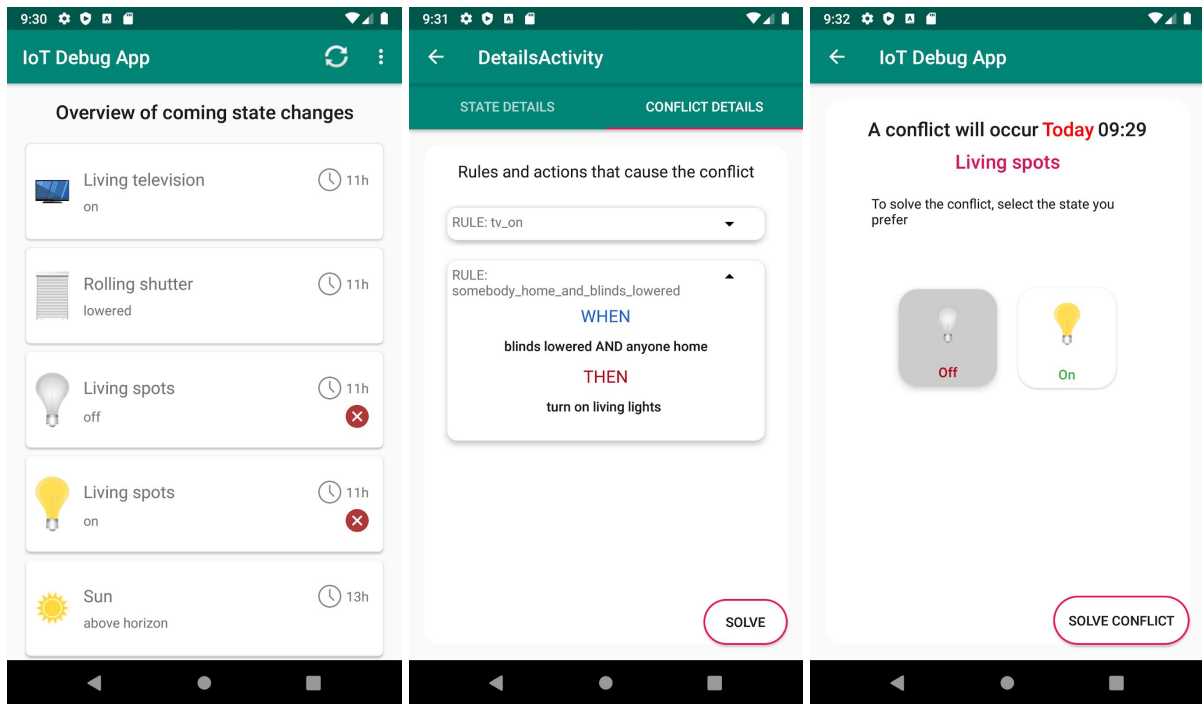
- **Loop:** the executed action of one rule triggers the next rule and so on, until it comes back to the first rule, where it starts all over again.

e.g. turn off lights → raise blinds → turn off lights → raise blinds →

Editing Rules

- new rules can be **added**
- rules can be **altered**:
 - add or remove more than one **condition**
e.g. sun rises and I am sleeping → sound alarm
 - add or remove more than one **action**
e.g. sun rises → raise the blinds and turn off lights
 - do a combination of **both**
e.g. TV is on and I am watching → lower the blinds and turn off the lights

The Application



1

2

3

1. Every row depicts a **predicted device state** in the near future

e.g. the living spots, the sun



A red cross shows that the state is in a **conflict**

e.g. conflict between two states that change the living spots



A green check shows that the state is the result of a **solution**

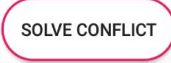
2. Inspecting a conflict

- a. By clicking the **arrow up** (▲) and **arrow down** (▼) buttons, more information can be hidden or revealed

- b. By clicking the **Solve button** (SOLVE) or **Edit button** (EDIT) a new screen will be displayed, giving the opportunity to solve or edit that specific conflict

3. Solving a conflict

- a. You can **select a desired state** using different visualisations
e.g. a card to select or a drop down menu

- b. Clicking the **Solve Conflict Button** () resolves the conflict and returns you back to the state change overview

A.2 Scenarios

The following scenarios were used during the study.

Scenario 1: This scenario contained an inconsistency conflict on the *roomba*. Michaels wishes:

- When he is home, the roomba can not be cleaning.
- It is okay for the roomba to be cleaning the house during the day, when Michael is not home.
- When he expects visitors, he wants to house to be cleaned.

The system's rules:

- Away & Sun shining → Roomba on
- Home → Roomba off
- Home & Party (calendar) → Roomba on

Scenario 2: This scenario contained an inconsistency conflict on the *door* and the *lights*.

Michaels wishes:

- When he comes home he want the door to be unlocked for him.
- The TV needs be on every time he comes home.
- The roomba needs to be docked when he is at home.
- Whenever someone comes home at dark, the lights should automatically turn on.
- When watching TV, no lights should reflect in the screen. Therefore lights should be turned off and the rolling shutters should be lowered.
- The door needs to be locked when it is dark.

The system's rules:

- Routine comes home → open Door, start TV, stop Roomba
- Sun set & Door open → Lights on
- TV on → Rolling shutters lowered, Light off
- Sun set → Door locked

Scenario 3: This scenario contained a redundancy conflict on the *lights*. Michaels wishes:

- The lights should always be off when the sun rises.
- When he is home during the day he wants the rolling shutters raised.
- As background noise, Michael prefers the TV to be on when he is home.
- Whenever he is watching TV, he does not want any lights reflecting the screen. Therefore the lights should be off.

The system's rules:

- Sun rises \rightarrow Light off
- Sun rises & home \rightarrow Rolling shutters raised
- Home \rightarrow TV on
- TV on \rightarrow Light off

Scenario 4: This scenario contained a redundancy conflict on the *rolling shutters*. Michaels wishes:

- Every evening, when the sun sets, the rolling shutters should be lowered as well.
- As background noise, Michael prefers the TV to be on when he is home.
- When watching TV, no lights should reflect in the screen. Therefore lights should be turned off and the rolling shutters should be lowered.

The system's rules:

- Sun set \rightarrow Rolling shutters lowered
- TV on \rightarrow Rolling shutters lowered, Light off
- Home \rightarrow TV on

Scenario 5: This scenario contained a loop conflict starting on the *lights*. Michaels wishes:

- When he is home during the day he wants the rolling shutters raised.
- The lights should always be off when there is already enough light inside the house.

The system's rules:

- Sun rises \rightarrow Light off
- Light off & home \rightarrow Rolling shutters raised
- Rolling shutters raised \rightarrow Light off

A.3 Questionnaires

A.3.1 Understanding the conflict questionnaire

In every scenario, participants first had to understand the conflict at hand by looking at the rules. The following questionnaire was then used to inform ourselves if the participant understood the conflict that was present and if they already would have some ideas on how to tackle it.

Survey What?

*Vereist

1. Participant ID *

2. Use case *

Understanding the conflict and Solution

3. Understanding of the conflict *

Markeer slechts één ovaal.

1 2 3 4 5

4. Why or why not?

5. How would you tackle and/or solve it? *

Deze content is niet gemaakt of goedgekeurd door Google.

Google Formulier

A.3.2 Observation questionnaire

Once a participant used a method (rule editing or the application) to solve a conflict, the facilitator used following questionnaire to learn about the participants trust in their own solution as well as to measure how easy it was for them to achieve a solution.

Observation

*Vereist

1. Participant ID *

2. Use case *

3. Method *

Questions

4. Solved conflict correctly *

Markeer slechts één ovaal.

Yes

No

5. Why or why not?

6. Trust in the solution (one time) *

Markeer slechts één ovaal.

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. Why?

8. Trust in the solution (always) *

Markeer slechts één ovaal.

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

9. Why?

10. Easy to solve *

Markeer slechts één ovaal.

1	2	3	4	5
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

11. Why?

12. Other solution possible? *

Markeer slechts één ovaal.

- Yes
- No
- Maybe
- I don't know

Deze content is niet gemaakt of goedgekeurd door Google.

Google Formulier

A.3.3 SUS and preference questionnaire

At the end of the study, a closing questionnaire was given to the participants, where we used the Subjective Usability Scale (SUS) to measure subjective satisfiability and a preference matrix to better understand when the application would be most desired or useful (preferred) for the participants.

Study questionnaire

*Vereist

1. Participant ID *

Conflict
solving: Rule
editing

In this section, you will be asked to answer a few questions about the rule editing method for solving conflicts in a smart home environment.

Rule editing

Regels:

- Weg & Zon schijnt → Stofzuiger aan
- Thuis → Stofzuiger uit
- Thuis & Feestje (calendar) → Stofzuiger aan

How much do you agree or disagree with the following statements?

2. I think I would like to use this method for conflict solving frequently *

Markeer slechts één ovaal.

1 2 3 4 5

Strongly disagree Strongly agree

3. I found this method for conflict solving unnecessarily complex *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

4. I thought this method for conflict solving was easy to use *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

5. I think that I would need the support of a technical person to be able to use this method for conflict solving *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

6. I found the various functions possible in this method for conflict solving were well integrated *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

7. I thought there was too much inconsistency in this method for conflict solving. *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

8. I would imagine that most people would learn to use this method for conflict solving very quickly *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

9. I found this method for conflict solving very cumbersome to use *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

10. I felt very confident using this method for conflict solving. *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

11. I needed to learn a lot of things before I could get going with this method for conflict solving. *

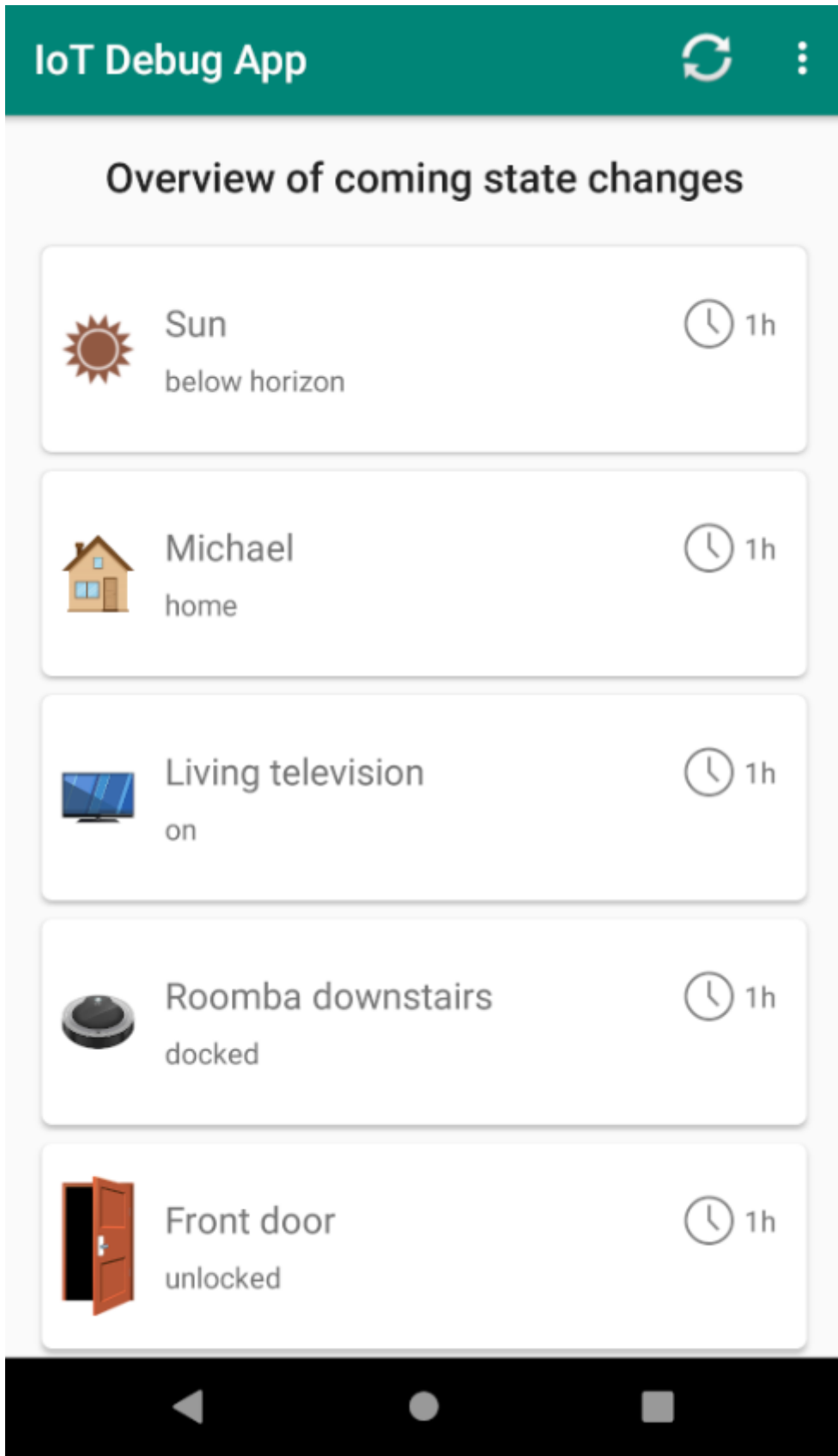
Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

Conflict
solving:
Application

In this section, you will be asked to answer a few questions about the application for solving conflicts in a smart home environment.

Application



How much do you agree or disagree with the following statements?

12. I think I would like to use this method for conflict solving frequently *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

13. I found this method for conflict solving unnecessarily complex *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

14. I thought this method for conflict solving was easy to use *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

15. I think that I would need the support of a technical person to be able to use this method for conflict solving *

Markeer slechts één ovaal.

	1	2	3	4	5	
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

16. I found the various functions possible in this method for conflict solving were well integrated *

Markeer slechts één ovaal.

1	2	3	4	5		
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

17. I thought there was too much inconsistency in this method for conflict solving. *

Markeer slechts één ovaal.

1	2	3	4	5		
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

18. I would imagine that most people would learn to use this method for conflict solving very quickly *

Markeer slechts één ovaal.

1	2	3	4	5		
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

19. I found this method for conflict solving very cumbersome to use *

Markeer slechts één ovaal.

1	2	3	4	5		
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

20. I felt very confident using this method for conflict solving. *

Markeer slechts één ovaal.

1	2	3	4	5		
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

21. I needed to learn a lot of things before I could get going with this method for conflict solving. *

Markeer slechts één ovaal.

1	2	3	4	5		
Strongly disagree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly agree

General

In this section we conclude with a few more general questions.

22. Which do you prefer for... *

Markeer slechts één ovaal per rij.

	Rule editor	Application	Both	None	No preference
understanding created solutions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
solving a one time specific conflict	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
solving a reoccurring conflict	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
making one time state exceptions	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
reaching a desired state	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
gaining control over the systems state	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

23. Do you have any remarks on the previous question?

24. Do you have any feedback on the application?

Deze content is niet gemaakt of goedgekeurd door Google.

Google Formulier

Bibliography

- [1] Vol Nr, Rob Miller, and Murray Shanahan. “The Event Calculus in Classical Logic - Alternative Axiomatisations”. In: *Journal of Electronic Transactions on Artificial Intelligence* 3 (June 2000).
- [2] Andrew J. Ko and Brad A. Myers. “Designing the Whyline: A Debugging Interface for Asking Questions about Program Behavior”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '04. Vienna, Austria: Association for Computing Machinery, 2004, pp. 151–158. ISBN: 1581137028. DOI: 10.1145/985692.985712. URL: <https://doi.org/10.1145/985692.985712>.
- [3] Anind K. Dey et al. “iCAP: Interactive Prototyping of Context-Aware Applications”. In: *Pervasive Computing*. Ed. by Kenneth P. Fishkin et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 254–271. ISBN: 978-3-540-33895-6.
- [4] Henry Lieberman et al. “End-User Development: An Emerging Paradigm”. In: *End User Development*. Ed. by Henry Lieberman, Fabio Paternò, and Volker Wulf. Dordrecht: Springer Netherlands, 2006, pp. 1–8. ISBN: 978-1-4020-5386-3. DOI: 10.1007/1-4020-5386-X_1. URL: https://doi.org/10.1007/1-4020-5386-X_1.
- [5] Yvonne Rogers. “Moving on from Weiser’s Vision of Calm Computing: Engaging UbiComp Experiences”. In: *UbiComp 2006: Ubiquitous Computing*. Ed. by Paul Dourish and Adrian Friday. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–421. ISBN: 978-3-540-39635-2.
- [6] Nan Zang and Mary Beth Rosson. “What’s in a mashup? And why? Studying the perceptions of web-active end users”. In: Oct. 2008, pp. 31–38. DOI: 10.1109/VLHCC.2008.4639055.
- [7] Kevin Ashton. “That ‘Internet of Things’ Thing”. In: *RFiD Journal* (2009).
- [8] Mitchel Resnick et al. “Scratch: Programming for All”. In: *Commun. ACM* 52.11 (Nov. 2009), pp. 60–67. ISSN: 0001-0782. DOI: 10.1145/1592761.1592779. URL: <https://doi.org/10.1145/1592761.1592779>.
- [9] Abdallah Namoun, Tobias Nestler, and Antonella De Angeli. “Conceptual and Usability Issues in the Composable Web of Software Services”. In: vol. 6385. July 2010, pp. 396–407. DOI: 10.1007/978-3-642-16985-4_35.

- [10] Abdallah Namoun, Tobias Nestler, and Antonella De Angeli. “Service Composition for Non-programmers: Prospects, Problems, and Design Recommendations”. In: Dec. 2010, pp. 123–130. DOI: 10.1109/ECOWS.2010.17.
- [11] A. Brush et al. “Home Automation in the Wild: Challenges and Opportunities”. In: May 2011, pp. 2115–2124. DOI: 10.1145/1978942.1979249.
- [12] Yngve Dahl and Reidar-Martin Svendsen. “End-User Composition Interfaces for Smart Environments: A Preliminary Study of Usability Factors”. In: *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*. Ed. by Aaron Marcus. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 118–127. ISBN: 978-3-642-21708-1.
- [13] Jussi Kiljander et al. “Enabling End-Users to Configure Smart Environments”. In: July 2011, pp. 303–308. DOI: 10.1109/SAINT.2011.58.
- [14] Jayavardhana Gubbi et al. “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions”. In: *Future Generation Computer Systems* 29 (July 2012). DOI: 10.1016/j.future.2013.01.010.
- [15] Sarah Mennicken and Elaine Huang. “Hacking the Natural Habitat: An In-the-Wild Study of Smart Homes, Their Development, and the People Who Live in Them”. In: vol. 7319. June 2012, pp. 143–160. DOI: 10.1007/978-3-642-31205-2_10.
- [16] Valentin Heun, James Hobin, and Pattie Maes. “Reality Editor: Programming Smarter Objects”. In: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp ’13 Adjunct. Zurich, Switzerland: Association for Computing Machinery, 2013, pp. 307–310. ISBN: 9781450322157. DOI: 10.1145/2494091.2494185. URL: <https://doi.org/10.1145/2494091.2494185>.
- [17] Sílvia Resendes, Andre Santos, and Paulo Carreira. “Conflict Detection and Resolution in Home and Building Automation Systems”. In: *Springer Journal of Ambient Intelligence and Humanized Computing* (Oct. 2013). DOI: 10.1007/s12652-013-0184-9.
- [18] Tom Rodden et al. “At Home with Agents: Exploring Attitudes Towards Future Smart Energy Infrastructures”. In: Apr. 2013, [In Preparation/Press]. DOI: 10.1145/2470654.2466152.
- [19] Enrico Costanza et al. “Doing the laundry with agents: A field trial of a future smart energy system in the home”. In: Apr. 2014. ISBN: 978-1-4503-2473-1. DOI: 10.1145/2556288.2557167.
- [20] Paolo Medagliani et al. “Internet of Things Applications - From Research and Innovation to Market Deployment”. In: (Jan. 2014).

- [21] 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*. UbiComp ’14. Seattle, Washington: Association for Computing Machinery, 2014, pp. 105–115. ISBN: 9781450329682. DOI: 10.1145/2632048.2636076. URL: <https://doi.org/10.1145/2632048.2636076>.
- [22] S. Munir and J. A. Stankovic. “DepSys: Dependency aware integration of cyber-physical systems for smart homes”. In: *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*. 2014, pp. 127–138.
- [23] Blase Ur et al. “Practical trigger-action programming in the smart home”. In: *Conference on Human Factors in Computing Systems - Proceedings* (Apr. 2014). DOI: 10.1145/2556288.2557420.
- [24] Evan Magill and Jesse Blum. “Exploring conflicts in rule-based sensor networks”. In: *Pervasive and Mobile Computing* 27 (Aug. 2015). DOI: 10.1016/j.pmcj.2015.08.005.
- [25] Mateusz Mikusz et al. “Repurposing Web Analytics to Support the IoT”. In: *Computer* 48 (Sept. 2015), pp. 42–49. DOI: 10.1109/MC.2015.260.
- [26] Federico Cabitza et al. “Rule-based tools for the configuration of ambient intelligence systems: a comparative user study”. In: *Multimedia Tools and Applications* 76 (Apr. 2016). DOI: 10.1007/s11042-016-3511-2.
- [27] Blase Ur et al. “Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes”. In: May 2016, pp. 3227–3231. DOI: 10.1145/2858036.2858556.
- [28] Julia Brich et al. “Exploring End User Programming Needs in Home Automation”. In: *ACM Trans. Comput.-Hum. Interact.* 24.2 (Apr. 2017). ISSN: 1073-0516. DOI: 10.1145/3057858. URL: <https://doi.org/10.1145/3057858>.
- [29] Nico Castelli et al. “What Happened in My Home? An End-User Development Approach for Smart Home Data Visualization”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 853–866. ISBN: 9781450346559. DOI: 10.1145/3025453.3025485. URL: <https://doi.org/10.1145/3025453.3025485>.
- [30] F. Corno, L. De Russis, and A. M. Roffarello. “A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT”. In: *Computer* 50.11 (2017), pp. 18–24.
- [31] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. “Empowering End Users to Customize Their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools”. In: *ACM Trans. Comput.-Hum. Interact.* 24.2 (Apr. 2017). ISSN: 1073-0516. DOI: 10.1145/3057859. URL: <https://doi.org/10.1145/3057859>.

- [32] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. “End-User Development for the Internet of Things: EFESTO and the 5W Composition Paradigm”. In: *Rapid Mashup Development Tools*. Ed. by Florian Daniel and Martin Gaedke. Cham: Springer International Publishing, 2017, pp. 74–93. ISBN: 978-3-319-53174-8.
- [33] Barrett Ens et al. “Ivy: Exploring Spatially Situated Visual Programming for Authoring and Understanding Intelligent Environments”. In: *Proceedings of the 43rd Graphics Interface Conference*. GI '17. Edmonton, Alberta, Canada: Canadian Human-Computer Communications Society, 2017, pp. 156–162. ISBN: 9780994786821.
- [34] Giuseppe Ghiani et al. “Personalization of Context-Dependent Applications Through Trigger-Action Rules”. In: *ACM Trans. Comput.-Hum. Interact.* 24.2 (Apr. 2017). ISSN: 1073-0516. DOI: 10.1145/3057861. URL: <https://doi.org/10.1145/3057861>.
- [35] Meiyi Ma, Sarah Preum, and John Stankovic. “CityGuard: A Watchdog for Safety-Aware Conflict Detection in Smart Cities”. In: Apr. 2017, pp. 259–270. DOI: 10.1145/3054977.3054989.
- [36] M. R. Reisinger, J. Schrammel, and P. Fröhlich. “Visual languages for smart spaces: End-user programming between data-flow and form-filling”. In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2017, pp. 165–169.
- [37] Michaela Reisinger, Johann Schrammel, and Peter Fröhlich. “Visual end-user programming in smart homes: Complexity and performance”. In: Oct. 2017, pp. 331–332. DOI: 10.1109/VLHCC.2017.8103495.
- [38] Carmelo Ardito, Giuseppe Desolda, and Maristella Matera. “Engineering Task-Automation Systems for Domain Specificity”. In: Feb. 2018, pp. 108–119. ISBN: 978-3-319-74432-2. DOI: 10.1007/978-3-319-74433-9_9.
- [39] Danilo Caivano et al. “Supporting end users to control their smart home: design implications from a literature review and an empirical investigation”. In: *Journal of Systems and Software* 144 (June 2018). DOI: 10.1016/j.jss.2018.06.035.
- [40] Luigi De Russis and Alberto Roffarello. “A Debugging Approach for Trigger-Action Programming”. In: Apr. 2018, pp. 1–6. DOI: 10.1145/3170427.3188641.
- [41] Timo Jakobi et al. “Evolving Needs in IoT Control and Accountability: A Longitudinal Study on Smart Home Intelligibility”. In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2.4 (Dec. 2018). DOI: 10.1145/3287049. URL: <https://doi.org/10.1145/3287049>.
- [42] Jisun Jang and Tomasz Bednarz. “HoloSensor for Smart Home, Health, Entertainment”. In: *ACM SIGGRAPH 2018 Appy Hour*. SIGGRAPH '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018. ISBN: 9781450358071. DOI: 10.1145/3213779.3213786. URL: <https://doi.org/10.1145/3213779.3213786>.

- [43] Marco Manca et al. “Supporting End-User Debugging of Trigger-Action Rules for IoT Applications”. In: *International Journal of Human-Computer Studies* 123 (Nov. 2018). DOI: 10.1016/j.ijhcs.2018.11.005.
- [44] Jannish Purmaissur et al. “Augmented-Reality Computer-Vision Assisted Disaggregated Energy Monitoring and IoT Control Platform”. In: Dec. 2018, pp. 1–6. DOI: 10.1109/ICONIC.2018.8601199.
- [45] Eugene Siow, Thanassis Tiropanis, and Wendy Hall. “Analytics for the Internet of Things: A Survey”. In: *ACM Computing Surveys* 51 (Apr. 2018). DOI: 10.1145/3204947.
- [46] A. Al Farooq et al. “IoTC2: A Formal Method Approach for Detecting Conflicts in Large Scale IoT Systems”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 2019, pp. 442–447.
- [47] Ahmed Alfakeeh et al. “Agent-based negotiation approach for feature interactions in smart home systems using calculus of the context-aware ambient”. In: *Transactions on Emerging Telecommunications Technologies* (Dec. 2019). DOI: 10.1002/ett.3808.
- [48] Barbara Barricelli et al. “End-User Development, End-User Programming and End-User Software Engineering: a Systematic Mapping Study”. In: *Journal of Systems and Software* 149 (Mar. 2019), pp. 101–137. DOI: 10.1016/j.jss.2018.11.041.
- [49] Will Brackenbury et al. “How Users Interpret Bugs in Trigger-Action Programming”. In: Apr. 2019, pp. 1–12. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300782.
- [50] Luca Corcella et al. “A Visual Tool for Analysing IoT Trigger/Action Programming: 7th IFIP WG 13.2 International Working Conference, HCSE 2018, Sophia Antipolis, France, September 3–5, 2018, Revised Selected Papers”. In: Jan. 2019, pp. 189–206. ISBN: 978-3-030-05908-8. DOI: 10.1007/978-3-030-05909-5_11.
- [51] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. “Empowering End Users in Debugging Trigger-Action Rules”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI ’19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13. ISBN: 9781450359702. DOI: 10.1145/3290605.3300618. URL: <https://doi.org/10.1145/3290605.3300618>.
- [52] Fulvio Corno, Luigi De Russis, and Alberto Roffarello. “My IoT Puzzle: Debugging IF-THEN Rules Through the Jigsaw Metaphor”. In: July 2019, pp. 18–33. ISBN: 978-3-030-24780-5. DOI: 10.1007/978-3-030-24781-2_2.
- [53] Fulvio Corno, Luigi De Russis, and Alberto Roffarello. “RecRules: Recommending IF-THEN Rules for End-User Development”. In: *ACM Transactions on Intelligent Systems and Technology* 10 (Sept. 2019), pp. 1–27. DOI: 10.1145/3344211.

- [54] Ting-Hao K. Huang et al. “InstructableCrowd: Creating IF-THEN Rules for Smartphones via Conversations with the Crowd”. In: *Human Computation* 6.1 (Sept. 2019). ISSN: 2330-8001. DOI: 10.15346/hc.v6i1.7. URL: <http://dx.doi.org/10.15346/hc.v6i1.7>.
- [55] Nicola Leonardi et al. “Trigger-Action Programming for Personalising Humanoid Robot Behaviour”. In: Apr. 2019, pp. 1–13. ISBN: 978-1-4503-5970-2. DOI: 10.1145/3290605.3300675.
- [56] Fabio Paternò and Sadi Alawadi. “Towards Intelligent Personalization of IoT Platforms End User Development; Internet of Things; Trigger-Action Programming ACM Reference format”. In: Mar. 2019.
- [57] T. Shah et al. “Conflict Detection in Rule Based IoT Systems”. In: *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. 2019, pp. 0276–0284.
- [58] Lefan Zhang et al. “AutoTap: Synthesizing and Repairing Trigger-Action Programs Using LTL Properties”. In: May 2019, pp. 281–291. DOI: 10.1109/ICSE.2019.00043.
- [59] Home Assistant 2020. *Home Assistant - Awaken your home*. URL: <https://www.home-assistant.io/>. (Accessed: 2020-05-06).
- [60] Usability.gov 2020. *usability.gov - Improving the User Experience*. URL: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>.
- [61] World Health Organization 2020. *Coronavirus disease (COVID-19) pandemic*. URL: <https://www.who.int/emergencies/diseases/novel-coronavirus-2019>.
- [62] James Chen. *Smart Home*. URL: <https://www.investopedia.com/terms/s/smart-home.asp>. (accessed: 01.03.2020).
- [63] Sven Coppers, Davy Vanacken, and Kris Luyten. “FORTNIoT: Intelligible Predictions to Improve User Understanding of Smart Home Behavior”. In: Under review.
- [64] Margaret Rouse et al. *Internet of Things (IoT)*. URL: <https://internetoftthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>. (accessed: 01.03.2020).