UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

HassTalk: An intelligible chatbot for IoT systems

Bram Kelchtermans

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

PROMOTOR :

dr. Davy VANACKEN

2019
2020

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

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

HassTalk: An intelligible chatbot for IoT systems

Bram Kelchtermans

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

PROMOTOR :

dr. Davy VANACKEN

Universiteit Hasselt

Master Thesis submitted to obtain the degree of
Master in Computer Science

HassTalk: Intelligibile IoT

By:
Bram Kelchtermans (1437654)

Promotor:
Dr. Davy Vanacken
Assessor:
Mr. Sven Coppers

Contents

1 Introduction **1**

2 Related work **3**
2.1 Chatbots & Natural Language Processing in the world of HCI 3
2.2 Evaluation of a Chatbot 6
2.3 Challenges with IoT-systems 8
2.4 Debugging IoT-systems 10
2.5 Code Assistance & Peer Debugging 12
2.6 Making applications intelligible 13

3 Exploratory study on information needs **17**
3.1 Participants 18
3.2 Procedure 19
3.3 Utilized use cases 19
3.3.1 Situation 1: the reason of a state 19
3.3.2 Situation 2: rule unexpectedly not executed 20
3.3.3 Situation 3: the future behavior of a device 22
3.3.4 Situation 4: the future behavior of a rule 22
3.3.5 Situation 5: verifying a rule 23
3.4 Analysis and results 24
3.5 Discussion 29
3.6 Conclusions 30

4 HassTalk architecture **33**
4.1 Home Assistant 34
4.2 FORTNIoT 36
4.2.1 Devices 36
4.2.2 State of a device 36
4.2.3 Rules 38
4.2.4 Simulation 40
4.2.5 Cause of device's state (why-route) 41
4.2.6 Cause of (non) execution of rules (why not-route) 44
4.3 Training DialogFlow 47
4.3.1 User's intent 47
4.3.2 Entity types 48
4.3.3 Answer intent 50

5 HassTalk **53**
5.1 Current state of a device 53
5.2 Future behavior of a device 55
5.2.1 State at given time 59
5.2.2 Complete future 60

5.2.3 Next occurrence of a device's state . . . 60
5.2.4 Reason of an occurrence of a device's state . . . 61
5.3 Reason of a device's state . . . 63
5.4 Reason why expectation did not happen . . . 68
5.5 Simulating behavior on a device's state change . . . 70

6 HassTalk user study **75**
6.1 Participants . . . 75
6.2 Procedure . . . 75
6.3 Measurement and analysis . . . 80
6.4 Results . . . 81
6.5 Discussion and conclusions . . . 86

7 Conclusion and future work **89**

A Explanation types **91**

B Response Templates **94**
B.1 Regarding the why (not) question . . . 94
B.2 Regarding the future behavior of a device . . . 94

C Input intents **95**
C.1 Current state of device: StateIntent . . . 95
C.2 Future behavior of a device . . . 96
C.2.1 StateAtTimeIntent . . . 96
C.2.2 CompleteFutureIntent . . . 97
C.2.3 WhenIntent . . . 97
C.2.4 WhyAtTimeIntent . . . 97
C.3 Reason of a device's state . . . 98
C.3.1 WhyIntent . . . 98
C.3.2 WhyNotIntent . . . 98
C.4 Reason why expectation did not happen: ExpectationIntent . . . 98
C.5 Simulating behavior on a device's state change: SimulationIntent . . . 99

D Answer intents **100**
D.1 Current state of device: StateAnswer . . . 100
D.2 Future behavior of a device . . . 100
D.2.1 StateAtTimeAnswer . . . 100
D.2.2 CompleteFutureAnswer . . . 100
D.2.3 WhenAnswer . . . 100
D.3 Reason of a device's state . . . 101
D.3.1 Reason_BecauseOfRule . . . 101
D.3.2 Reason_BecauseOfUser . . . 101
D.3.3 Reason_BecauseOfUnknown . . . 102
D.3.4 Reason_RuleDeniedByRule . . . 102

D.3.5 Reason_RuleDeniedByUser 102
D.4 Reason why expectation did not happen: ExpectationAnswer 103
D.5 Simulating behavior on a device's state change: SimulationAnswer . . . 103

E Entity Types **104**
E.1 DeviceName 104
E.2 State Attributes 105
E.3 AttributeValues 106

F Nederlandse samenvatting **107**
F.1 Introductie 107
F.2 Voorgaand onderzoek 107
F.3 Verkennende studie omtrent de informatieve noden van de gebruiker . . 108
F.4 HassTalk Architectuur 109
F.5 HassTalk: implementatie 112
F.6 Gebruikerstest 113
F.7 Conclusie 114

1 Introduction

We are experiencing a new era of Internet of Things (IoT), where many electronic devices surrounding us are interconnected by a network [1]. This paradigm enables copious amounts of data to be stored, processed and conferred in a proficiently interpretable form without human intervention. The emerging of IoT also sheds new light on the concept of a *smart home*. IoT-enabled house equipment allows for a smart home to be more intelligent, remote controllable and interconnected. These systems keep track of their connected devices (called entities) and their states, and moreover adds logic to these entities. It enables the creation of TCA-rules (Trigger-Condition-Action rules) to automate the behavior of the entities in the system. For example, "When I get home (Trigger) and the sun is below horizon (Condition), turn on the table lights (Action).

The global smart home market is expected to grow to USD 119.26 billion by 2022 [2]. Global companies, such as Google, Amazon, Apple... are entering this huge market, and they are providing innovative services and products to take advantage of the growing market. The smart home has been drawing attention recently due to the IoT, but it is not a new concept. In fact, the concept of a smart home has been discussed since 1980, and it has evolved from traditional home automation.

Despite the long history and growing interest, smart home service has not been widely accepted. There are many reason, like the high device prices, limited consumer demand... preventing smart home diffusion. Edwards and Grinter [3] researched the challenges to overcome with ubiquitous computing at home. The challenge for homeowners with their collection of (smart) devices will be to understand when their houses make the transition from dumb to smart and manage that transformation. The users, who do not necessarily have technical background knowledge, will be the administrator of their Smart Home system and its containing devices. These smart devices may be made by different manufacturers, which creates a challenge to ensure that there is no incompatibility and isolated islands of functionality. Moreover, these smart home systems need to be reliable and respect the privacy of their owner. More interesting for our research is the challenge of ensuring that users understand the pragmatics of sensors, interpretation, and machine action as well as they understand the pragmatics of devices in their home. Secondary, Edwards and Grinter argued that another reason is that the systems have to be designed for domestic use. Like any technology, smart technologies will be disruptive to the home environment and predicting these disruptions is difficult. The challenge for designers is to pay heed to the stable and compelling TCA-rules of the home, rather than external factors, including the abilities of the technology itself. These TCA-rules are subtle, complex and ill-articulated. For example, take the TCA-rule described above in which the lamp will turn on when you get home and the sun is below horizon. When the system fails in determining your position, this could result in the lamp not going on at all, while technically the conditions and triggers are met. Without the required knowledge, the user would not be able to identify the reason of that behavior.

This research aims to find a way to add intelligibility to IoT-systems, and thereby making these TCA-rules (and the behavior of the system in general) more understandable to the end-users. This research tries to exceed the threshold of understandability by utilizing natural language processing to enable users to question and validate the current and future behavior of their smart home systems. Therefore we proposed the chatbot HassTalk, which is built on top of Home Assistant[1] and utilizes natural language to interrogate the underlying IoT-system. For many users, natural language is already the default mode of interaction online, only there, the interaction is typically between users through a machine interface. At the same time, advances in AI mean that natural language interaction may be a feasible option for connecting machine agents and human users[4]. Natural language user interfaces are nothing new to the field of HCI. In fact, HCI researchers have studied these before, for example, in the context of multimodal systems, interactive voice-response systems, voice control in the context of accessibility and conversational systems[5]. HassTalk enables users to define what is happening and will happen in their systems, and why it is happening. Moreover, users are able to determine and validate the future behavior of their entities and implemented TCA-rules. These algorithms should allow users to detect possible errors and misbehavior in their systems, and more general lower the technical threshold which comes along with IoT- and Smart Home implementations.

This research includes a study to define the interesting pieces of information that are required to understand particular situations or phenomenons. This will be followed by a discussion about the implementation of the algorithms to provide the users with the right pieces of information, which will be tested by a set of users to validate the implementation.

[1] https://www.home-assistant.io/

2 Related work

As previously stated, smart home services are not widely accepted and used in the current era. This is due to many reasons, which are researched by Edwards and Grinter [3], in which they state that understanding the routines and behavior of a smart home system can be ill-articulated and thereby difficult to understand and interpret by a user that has no technical background. In the research of McManus et al. [6] is already concluded that the average user of an IoT system can successfully engage in trigger-action programming with multiple triggers and actions. But after the creation of such rules, there is no possibility for the user to determine if their created TCA-rules behave as expected. In our research we focussed on that challenge, of making an IoT-system more understandable and questionable for the end user and studied the opportunities of natural language processing in this field. Randall et al. [7] provided an early ethnographic account of using and living with smart home technology where they found that control in the smart home was not merely a technological, but also a social matter in multi-person households. Brush et al. [8] have identified poor manageability as a key barrier to the successful adaption of smart homes. They found that manageability and unreliable behavior were major concerns for smart home users. A particular problem has proven to be enabling non-programmers to successfully control and manage what amounts to a complex cyber-physical system.

Mennicken et al. [9] did a research covering the phases of growing a smart home. The resulting pipeline is shown in figure 1 and emphasizes the fact that, in order to create a smart home that fits the need of the end-user, a lot of iteration is needed. This mostly involves developers in order to create new behavior in the form of TCA-rules and debugging them in order to fit the needs of the end-user. HassTalk could be injected in the phase of "iterating until it fits", as it will help the end-user in understanding if the current behavior is according to their desires.

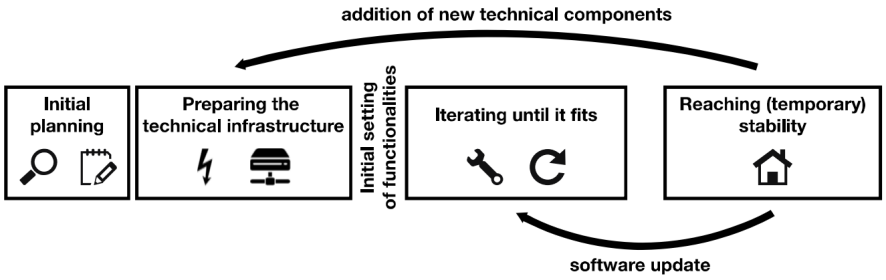


Figure 1: Mennicken et al. [9]: Different stages of creating a "smart home"

2.1 Chatbots & Natural Language Processing in the world of HCI

For decades, researchers and practitioners in human-computer interaction have been improving their skills in designing for graphical user interfaces. But more recently, there is more and more interest in the use of chatbots, and therefore natural language,

as the main interaction model. Interestingly, there are many definitions for chatbots in close relation with Software Agents (SA), Virtual Agents (VA) or Intelligent Personal Assistants (IPA) in literature and these have often been used in conjunction with each other. The term "Agents" itself has many definitions but among the earliest and most well-known uses of the term is[10]: *A self-contained, interactive and concurrently-executing object, possessing internal state and communication capability.* The scope of Software Agents can be most closely associated with chatbots and has been well documented in literature[11]. The following key properties have been associated with Software Agents[12]: (1) reactive, (2) pro-active and goal-oriented, (3) deliberative, (4) continual, (5) adaptive, (6) communicative and (7) mobile. The purpose of this research is not to explore the various types of Software Agents and agent based systems or its properties, but rather propose a solution for the end-user to use a chatbot in order to make smart home systems intelligible.

Følstad and Brandtzæg [13] researched the usability of chatbots in the world of human-computer interaction. In this research, the authors state that conversations can become the main object of design. This implies that there is a need to move from a user interface design to a service design, with a design for interaction in networks of human and intelligent machine actors. Introducing natural language conversations to the world of HCI, brings along new opportunities. According to Følstad and Brandtzæg the conversational model can combat the digital divide, because of the descriptive explanation of the desired task. When we understand conversational processes we can benefit from the massive volumes of user data and thereby make the solutions more applicable to the user.

Benotti et al. [14] used chatbots in their research, to engage high school students for getting interested in computer science. Therefore, they designed an educational chatbot tool that hides the complexity of programming. They applied a pedagogical strategy to increase the engagement of the students and combined this strategy with a classroom learning experience. Their research concluded that most indicators of engagement (task completion, participation, enthusiasm and self reported interest) where increased by using the chatbot. However, the researches state that only a few participants were self-motivated enough to complete the proposed task. This implies that in our study, we have to monitor this aspect and thereby determine if the user is enough motivated and thereby able to complete the proposed tasks.
Benotti et al. proved that using a chatbot can engage users in designing their own programs, without the required programming knowledge that would be necessary in the traditional method of programming.

These opportunities can bridge the difficulties in understanding and debugging IoT-systems. Nowadays, programming an IoT-system requires technical- and programming knowledge of the user. When using natural language and descriptive models, we can give every user the opportunity to understand and validate the behavior of their system.

A key issue for smart home systems is supporting non-expert users in their management. Jakobi et al. [15] did a research in which they equipped 12 households with DIY smart home systems for two years and studied participants' strategies for maintaining system awareness, from learning about its workings to monitoring its behavior. They found that participants initially looked for in-depth awareness information. In the later phases of appropriation, their interaction and information needs shifted towards management by exception. Jakobi et al. state that there is a pressing and significant need to make data intelligible in the context of smart homes, hence they seek to take a closer look at the potential of voice assistants to provide ambient system awareness in their future work.

Kar and Haldar [16] researched the opportunities for chatbots in the context of IoT-systems. They found that there are several challenges to counter while designing such a chatbot. Primarly, IoT systems operate in isolated technology or vendor specific silos which inhibits capability, value and interoperability and create a widely disparate area. This emphasizes the need of a uniform system which enables communication between smart devices of different manufactures. Secondly, Kar and Haldar found that a sheer number of connected things can create problems in application, data and device management in the context of IoT. It becomes increasingly difficult on users to keep track and access multiple applications, dashboards for every new *IoT object* in their ecosystem [17]. Hence unifying experiences across multiple connected things and providing them with a high degree of smartness for improved user experience is a key challenge. Kar and Haldar only researched about the opportunities in a theoretical aspect, they did not implement any of their findings. However, they provided an example of a conversation between a user and an intelligible IoT-system, which is depicted in figure 2.

Baby et al. [18] proposed a system that has a web application and a chatbot application to control the devices at home. The implemented chatbot application is capable of parsing a given sentence and uses this information to perform the relevant actions. Thereby, any device connected to the local network of the house can be controlled by a command given in natural language. However, this research is limited as it only allows controlling the appliances. HassTalk divides itself in the depth of usages of the chatbot. Where the research of Baby et al. only allowed the user to control their appliances, our research contributed in the understandability and the ability to interrogate the routines and automations that are built in the Smart Home systems.

Figure 2: Kar and Haldar [16]: Example conversation of a IoT chatbot-user conversation

Chatbots are a pull-based source of information, as the user first needs to ask something in order to get the required information. Out of the research of Cheverst et al [19] two key points were identified in such a pull-based system:

- The asynchronous nature of information pull meant that the information currently displayed ot a user could become inconsistent with his or her current context
  The information given by HassTalk needs to be consistent and may not be dependent of the user's context or situation.

- The stimulus used to cause the presentation of new information to occur comes directly from the user and therefore the user is already focussed on the system when the new content is presented.
  The information given by HassTalk needs to be direct and compact. Thererfore it is important to not give too much information, in order to not confuse the user.

These key points imply that the scope of the given information needs to be determined precisely. Giving too less information will result in the user having more questions, while giving too much information will confuse the user even more.

2.2 Evaluation of a Chatbot

From previous research, we got the objectives which are used in order to evaluate a chatbot. These measures are later used to evaluate HassTalk in the different perspectives. Folstad et al. [20] researched the elements that affect users trust in a chatbot, moreover they discuss the benefits and challenges regarding the design of a chatbot.

They invited 28 participants which had to use a chatbot, so that the researches could determine which factors are relevant to users' trust in chatbots. From their research, they concluded there are several benefits to working with chatbots. The participants state that the greatest advantage of a chatbot is the fast and accessible help and available information. They do not have to wait for a real person in order to receive an answer. This encouraged them in posing more questions, as there was no time pressure. However, their research emphasizes some challenges regarding the design of a chatbot. Their biggest concern is the occurrence of interpretational problems, especially when answering more complex questions. Another concern they had, were the privacy issues, as the system in this research does not use a public database, this concern is not applicable in our research. In table 1 are the results regarding the factors that affect users' trust in chatbots. As the research focussed on commercially available chatbots and HassTalk is not commercially available, the factors concerning the service environment are irrelevant.
Folstad et al. concluded their research with 5 key implications for future design of chatbots.

1. Prioritize efficient service provision: users should consistently experience the chatbot channel as superior on efficiency when choosing this option.

2. Be transparent on the chatbots features and limitations: clearly communicate both what it can do, and what it cannot do, to the user.

3. Strengthen the user experience through human-like conversation: a personal and human-like appearance may enhance user experience and trust in the chatbot.

4. Leverage users' trust in the brand: strategic use of branding and hosting of the chatbot may positively affect trust (not relevant in our research of HassTalk).

5. Demonstrate that security and privacy prioritize: design and dialogue of the chatbot should make it clear that security and privacy are top priorities.

Another research regarding the evaluation of chatbots is done by Cahn [21] which stated that in order to maximize the user satisfaction, surveys has to be taken. Surveys are perfectly suited to determine subjective measures regarding the user's satisfaction. Walker et al. [22] designed PARADISE (PARAdigm for DIalogue System Evaluation), a framework for evaluating spoken dialogue agents. This framework includes both subjective and objective measures, however, the objective measures are only applicable to commercially available chatbots and therefore irrelevant in this research. Regarding the subjective measures, they conduct the ease of usage, clarity, naturalness, friendliness, robustness regarding misunderstandings, and the willingness to use the system again. All these measures can be conducted by utilizing a survey, as stated in the research of Cahn [21]. We take the subjective measures of the PARADISE framework into account, by utilizing a SUS-survey.

High-level group	Factor name and description	Frequency
Factors concerning the chatbot	*Interpretation and advice.* Quality in interpretation of the user request and advise in response to request	9
	Human-likeness. The chatbot's appearance as human-like, personal, or polite	6
	Self-presentation. The chatbot's communication of what it can do and its limitations	3
	Professional appearance. The chatbot's appearance as being thoughtfully developed, with correct spelling and grammar	2
Factors concerning the service environment	*Brand.* The effect of the brand of the service provider hosting the chatbot	5
	Security and privacy. The importance of security and privacy aspects of the service	5
	Risk. The perceived risk associated with using the chatbot	2

Table 1: Factors that affect trust in chatbots

Kuligowska et al. [23] proposed some other evaluation methods regarding chatbots. The first one is the visual look of the chatbot, where they reward bots that resemble living people. Another criteria is the implementation of the chatbot in a webpage. Furthermore, the capabilities of a chatbot are determined by their ability to produce speech, their knowledge base and the ability to revise the conversation. Regarding the conversational abilities of the chatbot, they found that it is important that the chatbot is able to lead a coherent dialogue, understand the context and repair broken conversations. In order to determine the subjective satisfaction of the participants, we conducted a survey to determine the system usability scale [24]. This is a predefined survey which is widely used in the research field of Human-Computer Interaction. SUS measures the usability and learnability of an application and is a reliable and valid way to determine these factors considering an application.

2.3 Challenges with IoT-systems

The chatbot designed in this research needs to fulfill the needs for everyday smart home users, which do not necessarily have the needed technical background knowledge. Zhang et al [25] researched the effects of age, tasks and intelligence level on the design and evaluation of smart home user interfaces. They found that for different cognitive mode tasks, users would perform differently on interfaces with different intelligence levels. This means that when completing skill-based tasks, users would have best performance (less time and fewer errors) on low-intelligence level interface. When completing rule-based tasks, users would have best performance on high-intelligence user interface. Based on their findings, they propose the following guidelines for smart home interface designers:

- Both device-oriented and task-oriented interfaces are needed in the smart home. Designers should combine these two kinds of interface into one to provide users with multiple choices.
- For skill-based tasks, a low-intelligence level user interface with direct control style is highly recommended.
- For rule-based tasks, a task-oriented high intelligence user interface is recommended

In order to comply with these guidelines, the resulting chatbot of our research needs to take the intelligence level into account when answering questions of the user. When asking about the behavior of a device, the information needs to be basic and thereby have a low-intelligence level. In order to enable the user to understand a rule, the given information needs to be wide and precise and thereby have a high-intelligence level.

Leppänen and Jokinen [26] state in their research that most people seem to be quite traditional in thinking about home environment, and it seems to be hard to combine *technology* with *home*, even if people already have a lot of technology at home. Technology can be of help but a satisfactory compromise must be found, as it is a privilege for those who have the required knowledge about technology. In order to bridge this difficulty and thereby making smart home systems more accessible for everyday users, HassTalk tries to utilize natural language processing in order to provide the end-user with the information they require in order to understand their systems behavior.

Norman [27] defined the problem of automation as a matter of inappropriate feedback and interaction, rather than overautomation. His research found that there is a need of an appropriate way of feedback in automated systems. The task of presenting feedback is not easy to do and Norman found examples of how not to do it. In the research of HassTalk, we try to examine the possibilities of using a chatbot in order to provide the user of feedback. Moreover, Norman stated that a higher order of awareness is needed. The solutions to this challenge will require higher levels of automation, some forms of intelligence in the controls, an appreciation for the proper form of human communication that keeps people well informed. In order to communicate feedback to the user, a chatbot can bring great opportunities along, as it uses natural language to communicate.

In the research of Hwang and Hoey [28] they examined how to bridge the gap, created by the technical difficulty, between smart homes and their users. Therefore they explored the needs and requirements of the user in such systems. Smart homes need to be customizable and fulfil the desires of the users while not interfering on caregivers' values life roles. Users desire ongoing information on the status of their smart home and need a user interface that should be designed to work across a variety of form factors among which caregivers can select. Therefore they designed a user interface which uses the knowledge of the systems to inform the user about their smart homes. This user interface gave a 3D vision in which the user's home is explicitly modeled and that can

be customized by changing appliances and furniture and modifying the services these "smart" artifacts can provide. Giving this representation to the user, encouraged them to be more involved with their smart homes and thereby close the gab between the user and their system. Hwang and Hoey state that an addition to their framework could be the delivery of effective prompts that will both empower the user in understanding the behavior of their smart home systems. Despite adding customizability to the systems, the research did not implement a way for users to understand the existing behavior and the resulting behavior of their customizations.

Barkhuus and Dey [29] examined peoples' sense of control and preference in context-aware mobile computing. Their study shows that participants feel a lack of control when using the more autonomous interactivity approaches but that they still prefer active and passive context-aware features over personalization oriented applications. The researches conclude that users are willing to accept a large degree of autonomy from applications as long as the application's usefulness is greater than the cost of limited control. Giving more information about their IoT-system will bring along a rise of control for the user, as they will have more knowledge about the underlying behavior of their system. HassTalk enables the user to pose questions about the (reason of) the behavior of their system, and thereby are able to determine if the behavior is as wanted or not.

2.4 Debugging IoT-systems

Detecting conflicts in IoT-systems is an emerging field for research and development. The conflicts that are possible in the IoT-systems are often overlooked both in the design phase and during operations. Therefore, Al Farooq et al. [30] proposed a framework in which these conflicts are detected. By analyzing input from sensors and by using the implemented rules, the framework determines what action will be emitted. The framework then determines if the activities create conflicts, which they categorize under three policies: controller safety, multiple action trigger and multiple event handling. The research paper of Al Farooq et al. also proposed formulas to determine errors, which are used in HassTalk. Despite the fact that errors are detected in an automated manner, the results are not made understandable to the end-user. Therefore, the software still requires some knowledge about the implementation details in order to fully understand the occurring issue.

Manca et al. [31] introduced a system that notifies the user about problems in their IoT-system. While creating rules, the system searches for errors that will be caused by the new rule, which may not be visible while testing the system. This enables the user to create his own routines, without being concerned or bothered about demolishing the existing behavior. Occurring problems are not caused by errors in the rules per se. Sometimes an action is not as expected, while the rules are applied correctly without errors. In this case, the system will not find an error in the system and thereby can not help the user in defining the cause of the unexpected behavior. Manca et al. detect

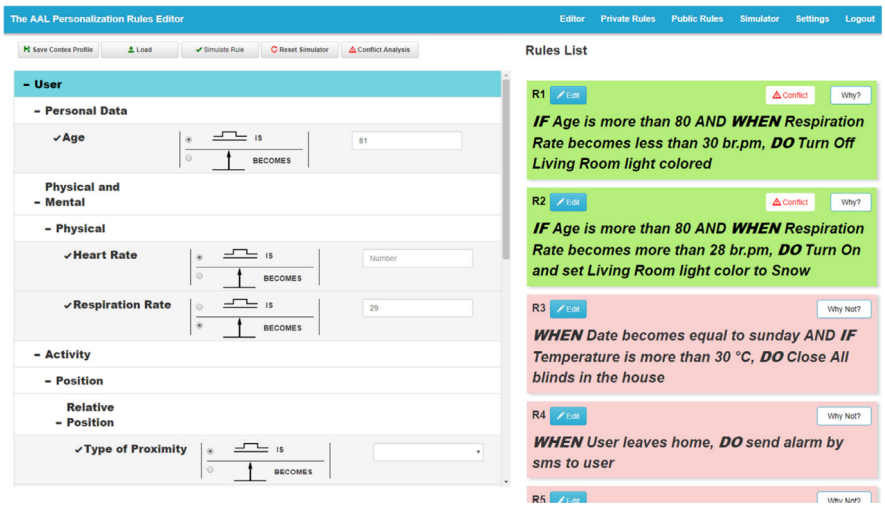


Figure 3: Manca et al. [31]: Environment for executing rules in a simulated context of use

errors by simulating the behavior of the system with the new rule included. Thereby it has a proper representation of the consequences that come along while implementing this rule. As the Internet of Things is emerging in many different domains (e.g. health care, transportation infrastructure, smart homes...), we can not assume that the user has technical- or programming knowledge or experience. Therefore, errors are not always that clear for the end-user. Introducing a descriptive method in combination of natural language as the communication model, will lower this threshold in terms of understandability. An example of their research is depicted in figure 3.

Corno et al. [32] designed EUDebug, which enables end-users to debug trigger-action rules. The occurring problems are characterized to be detected. Enabling this is done by implementing two complementary strategies. First of all, the software assists the user in identifying rule conflicts and secondly help them simulate and foresee the runtime behavior of their rules. With this software, end-users can deal with computer-related concepts, such as loops, inconsistencies and redundancies. Maybe more important, the software allows the user to understand why their rules may generate a specific problem. In this manner, the user knows how to avoid these sort of conflicts. While the software helps the end-users in detecting occurring issues, the study results show that the majority of the participants experienced difficulties in understanding the problems. This emphasizes that besides detecting errors, it is important to give a complete and understandable explanation to the end-user in order for them to understand.

2.5 Code Assistance & Peer Debugging

When developing software, programmers rely on each other. Chen et al. [33] designed CodeOn, an on-demand software development assistance tool, which allows developers to integrate external expert assistance in an asynchronous manner. This tool automatically passes the context of the problem to the other developers, thereby the developers get a wide description of the occurring problem. Chen et al. conclude that the developers using CodeOn are able to complete nearly twice as many tasks as they could using state-of-the-art synchronous video and code sharing tools. This conclusion emphasizes the importance of knowledge about the context of an occuring problem. As the effects were this significant on developers, we can conclude that this would be even bigger on non-developers. As the need of context and understandable explanation is even more important for them to fully comprehend the behavior of their IT-related problems.

Chen et al. [34] did two studies about remote experts, towards providing on-demand expert support for software developers. They concluded some interesting factors to take into account when providing assistance in a software context. First of all there is a need to consider the level of experience of the end user. Users with less technical- and programming experience require a more high level explanation of the situation. Secondly, they emphasize the need of knowledge about the context of an occurring issue. As IoT-systems are mostly context driven, this is an important aspect to take into account in our research. Furthermore, Chen et al. state that switching between applications (in their case an user interface and a chatbot) was found annoying by the participants. They stress that it is important to create an integrated system, so that there is one application for everything. As IoT-systems consist of multiple devices and their specific protocols, this factor is really important in our research. HassTalk has to form an integrated system, that understands these protocols in an universal manner.

Specifying IoT logic can be a complex task, especially for non-developers. Therefore, Akiki et al. [35] introduced ViSiT. By using a jigsaw-puzzle metaphor to specify transformation, they try to empower end-user to wire their IoT-objects. In this manner, non-programmers can define their own routines without any required technical knowledge. This study found that because of the introduced simplicity, the participants where more motivated to define their own logic and were less frustrated while doing it. This emphasizes the need for a high-level approach in communicating IoT-logic to the end-user. Where the study of Akiki et al. focussed on the creation of new routines, our research will be geared to debugging and understanding the behavior of a smart home environment.

Dey et al. [36] designed iCap, an interactive way of prototyping context-aware applications. iCap is a visual rule-based environment that supports end-users in prototyping context-aware applications without writing any code. Out of their user study, the conclude that every user could successfully use the software in order to create specific behavior using TCA-rules. This emphasizes the fact that it is perfectly possible to make TCA-rules understandable for the end-user, without requiring any programming knowledge.

2.6 Making applications intelligible

Adding intelligibility to applications makes them more understandable and improves the trust of the user. Ko and Myers [37] designed a software-environment that adds intelligibility to the Alice programming environment, the result of their research is depicted in figure 4. The user can ask multiple basic questions (e.g. *'Why is something happening'* or *'Where did a change occur'*) about the behavior of their code. These simple questions can help the end-user in understanding the software and its behavior. The researchers found that interrogative debugging can dramatically reduce the time that is needed to find and correct occurring errors, suggesting the potential as a highly effective approach to supporting debugging activity.

Lim and Dey [38] introduced intelligibility in Context-Aware Applications. They present a framework that provides automatic generation of eight explanation types: inputs, outputs, what, what if, why, why not, how to and certainty. From the results of this research, we can conclude that enabling these explanation types can raise the understandability and awareness of the end-user. Another example of adding intelligibility to the software context is given by the Enactor framework [39]. This framework can provide *What* explanations by exposing the state of input widgets, and *Why* explanations by reporting a relevant rule. The Crystal framework [40] only supports Why / Why Not explanations for desktop-based applications to explain themselves through Command Objects.

The research of Vermeulen et al. [41] explained how an existing pervasive computing framework was extended with support of *why* and *why not* questions, of which an example is shown in figure 5. As pervasive computing systems are generally context aware and therefore act without explicitly involving the user, users may be surprised as to why the system behaves in a certain way. Their framework enabled users to use the questions interface to find the causes of events which occurred in their system. Each participant of their study agreed strongly on the fact that these techniques are useful to allow users to understand what happens in their environment and offer them control over this behavior. This research however limits itself by making a predefined list of possible questions, so the user is not able to formulate their own questions. In their research, they found that the representation of the answers on the questions could be improved. HassTalk tries to bridge this remark by enabling the user in posing their own questions and receive their answers in a chat-based environment.

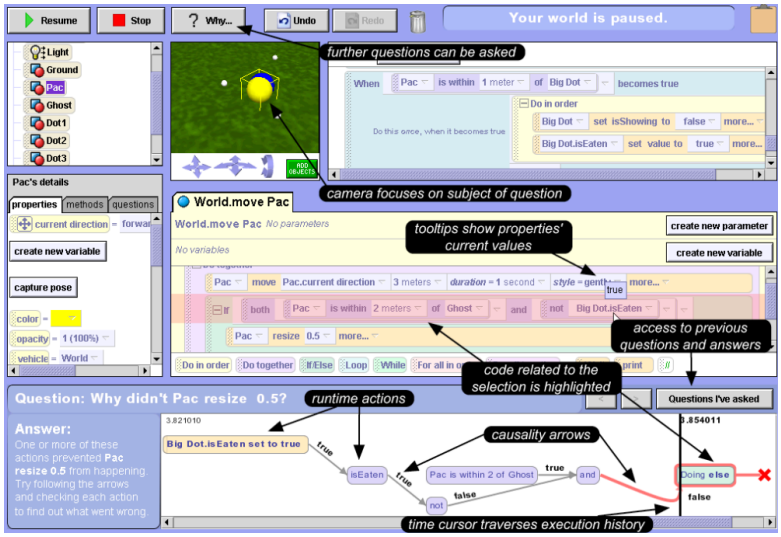


Figure 4: Ko and Myers [37]: adding the why question in a programming environment

Figure 5: Vermeulen et al. [41]: Answering the why question on the behavior of the lights in the system

Belotti and Edwards [42] present four design principles that support intelligibility of system behavior and accountability of human users that must be accounted in our chatbot:

- Inform the user of current contextual system capabilities and understandings
- Provide feedback including:
  - Feedforward: what will happen if I do this?
  - Confirmation: what am I doing and what have I done?
- Enforce identity and action disclosure
  Who is that, what are they doing and what have they done?
- Provide control to the user especially in cases of conflict and interest

These requirements are in conjunction with the information pieces proposed in the research of Lim and Dey [38]. Therefore implementing these types of questions in our chatbot will result in compliance with the requirements presented by Belotti and Edwards.
Mennicken et al [43] state that there are two important problems that stand in the way of attaining intelligibility in Smart Home systems:

1. The difficulty of understanding the complex reasoning of sensing technologies by users without a technical background
2. Users' lack of interest in and reluctance to invest time in learning how the underlying technology works

In the research of HassTalk we try to determine if language processing suffice in order to bridge these problems. Moreover, Mennicken et al. state that intelligibility in IoT-systems should be limited to the high-level rationale behind a certain automation action, with the potential to get more details if needed. This requires a proper understanding of the user's needs and an adequate answer which contains the needed information in order for the user to understand the underlying behavior of their system.

Lee et al [44] presented GALLAG Strip, a tool designed to enable end-user programming of context-aware applications for behavior change. They found that while their system is usable, people with a non-engineering background perceived it as less easy to use. This emphasizes the need of a proper way to communicate the required information to the end-user. As HassTalk is aimed at everyday home owners, we can not assume any technical background knowledge. With natural language, we try to make the knowledge of the IoT-system understandable to the end user.

Integrating intelligibility to the context of IoT-systems can help the end-user in understanding the behavior of their routines and smart home system. Therefore, where the previous researches were focussed on software developers, we focus on the end-user of the system. With adding intelligibility and questionability, we try to enable the users to debug and understand their system themselves, and thereby find the possible causes of the occurring misbehavior.

3 Exploratory study on information needs

As we are designing a chatbot, our medium of conversation is natural language. In this study, we want to determine how the user would formulate their question in a given situation, and what pieces of information the user wants to receive in order to fully understand the underlying behavior of their IoT-system. The formulation of the user's question is asked by a textbox and is thereby not restricted to a finite set of possibilities. The information pieces however are conducted from the paper *Toolkit to Support Intelligibility in Context-Aware Applications* [38]: what, what if, why, why not and when. These pieces of information are always explained in terms of inputs and outputs. The inputs are mostly received through input sensors and information sensors throughout the system, these inputs can result in different output options. The research of Lim et al included the explanation type of "How-To", in our research this would mean giving an explanation on how to reach a certain state. This type of explanation is merged with the "When" type. Take for example the how-to explanation of a lamp: *In order to turn on the lamp, the sun has to go above horzion*, in comparison with the when explanation: *The lamp would go on, when the sun goes above horizon.* Both of these explanations depict the same piece of information. In order to determine if this assumption is true, the fifth test in this study researched the similarity between these information types.
However, the user may require a combination of these pieces of information. Therefore a list was setup with all the different combinations, which can be found in Appendix A. In order to determine which pieces of information are required in a particular situation, the participants will have to give a likert scale on the different building blocks of information. This scale will then be utilized to determine the combination that is required to inform the user in a proper way. Below a list of the different types of information building blocks:

- [What]: defining the current state
  The lamp is currently on / The sun is below horizon...
- [What if]: defining the behavior on a particular change
  The lamp would turn on, if the sun goes below horizon
- [Why]: defining the reason of the current state
  Bram did it / Because the sun went below horizon
- [Why not]: defining why the current state is not a particular value
  Because the rule is not triggered
- [When]: defining when something will happen
  The lamp will turn on at 7:34pm

We want to know which building blocks of information are required in order for the user to understand the underlying behavior of the system. Therefore, in this study the following hypotheses are tested:

H1 When the users asks for the reason of an occurrence (state of a device / rule execution), they want to receive the why-piece of information

H2 When the users asks for the reason of a behavior, with another expectation in mind, the why-not formulation is used. (eg. If the lamp is off, but the user expected it to be on, this would result in a question in the form of "Why is my lamp not on?")

H3 When asking for the reason of the future behavior of a rule, the participant wants to know the timestamp on which the rule will be executed.

H4 When asking why a rule is NOT executed, the user wants to know if the rule is implemented correctly.

H5 When asking why a rule is NOT executed, the user wants to know the states of the entities that can cause the rule to be triggered.

H6 When asking for the next occurrence of an event (state of a device / rule execution), the user wants to know both the time of occurrence and the rule execution that results in the occurrence.

3.1 Participants

We recruited 21 participants via social media. The group of participants consists of 8 women and 13 men (1 aged -17, 2 aged 18-20, 13 aged 21-29, 2 aged 30-39, 1 aged 40-49 and 2 aged 50-59) with varying backgrounds (e.g. students, health, education, computer science and business). We asked the participants for their experience in the fields of programming, Internet of Things and the use of automated chatbots. The experience of the users over this field is very distributed, which was the aim of this evaluation.

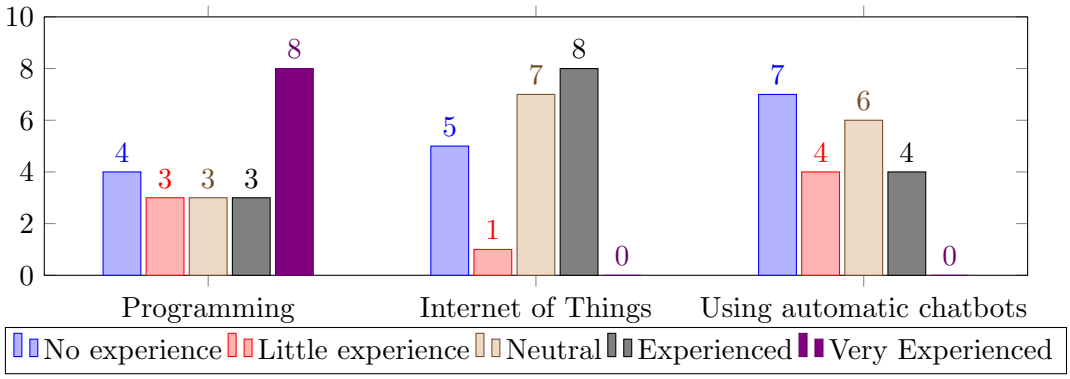


Figure 6: The distribution of the participant's experience in the different fields of the scope of this research.

3.2 Procedure

Because of the COVID-19 pandemic, we opted for an online survey in order to receive a large amount of responses in a short period of time and because it suits perfect for our goal. To alleviate the distraction effects of participating in an uncontrolled environment [45], we limited the number of tasks to keep the experiment short (estimated 15 minutes) and ensure a low drop-out rate[46].

The procedure of the survey commenced with a briefing about the study's purpose and our data policy, based on the GDPR legislation[2]. After giving their informed consent, participants completed a short demographic survey and then got a short introduction to the field of Intelligble Internet of Things. The participants were then introduced to 5 scenarios in which an IoT-system was explained. From previous research [37, 41, 38], we concluded that integrating the why-question brings along a great contribution in terms of understanding the behavior of a system. In this study, we want to determine what information the participant requires in the answer of the why question. In the field of IoT, answering the why question can be applied on the state of a device (why does a device have a given state / why does a device not have a given state) or on the execution of a TCA-rule (why is the rule X executed / why is the rule X not executed). Furthermore, the research of Manca et al [31] and Corno et al. [32], learned us that the ability of future simulations clarifies the occurring behavior of a system.

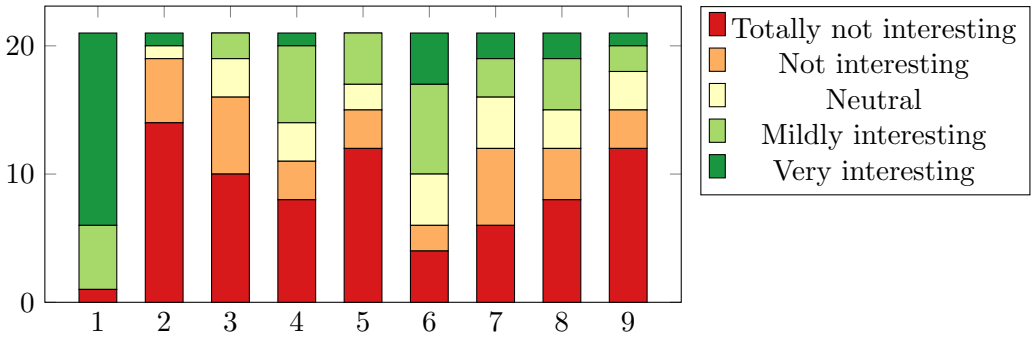
3.3 Utilized use cases

For each given scenario, the participant is given a list of information pieces. The participant has to rate these pieces of information by their interest. Secondly the participants are asked how they would formulate their question in order to receive the interesting information. After each scenario the participant is able to fill in some remarks or complaints.

3.3.1 Situation 1: the reason of a state

In the first situation, it is not clear why a device has its given state and therefore, the participants have to track down the reason of that particular state. The proposed system contains a rule that turns on the lamp when the sun goes below horizon. However, in the evening the lamp is actually off. In order to determine the cause of the state of the lamp, the participant had to give their interest on the given pieces of information. The proposed pieces of information and the distribution of the participant's interest is shown in figure 7.

[2] https://ec.europa.eu/info/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules/eu-data-protection-rules_en

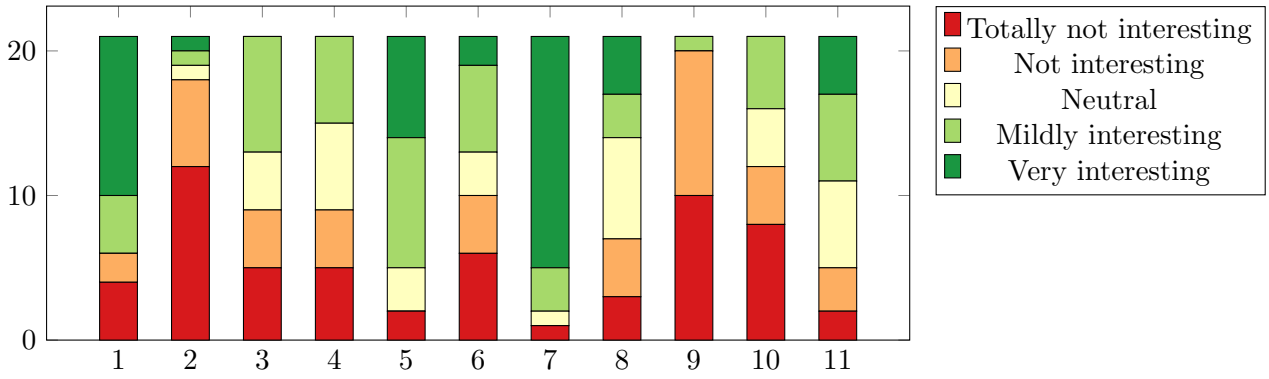


1. The lamp is off because Bram did it [**why**]
2. The lamp is off [**what**]
3. The sun is below horizon [**what**]
4. Bram is home [**what**]
5. You are home [**what**]
6. The lamp went off at 9:03pm [**when**]
7. The sun went below horizon at 7:54pm [**when**]
8. Bram is home since 5:54pm [**when**]
9. You're home since 4:31pm [**when**]

Figure 7: Information pieces that were given in the first situation. The graph shows what the participants considered to be interesting in order to determine the cause of a particular state.

3.3.2 Situation 2: rule unexpectedly not executed

When a rule is not executed, or the result is not visible, it is not always clear why the rule did not execute. The second situation describes a rule which should turn on the heater when the outside temperature is below 10 degrees Celsius, provided that anyone is at home. However, the user himself is home and it's freezing outside, but the heater did not turn on. The participants had to determine what caused the rule to not be executed. The list of given information pieces and the interest distribution is shown in figure 8.

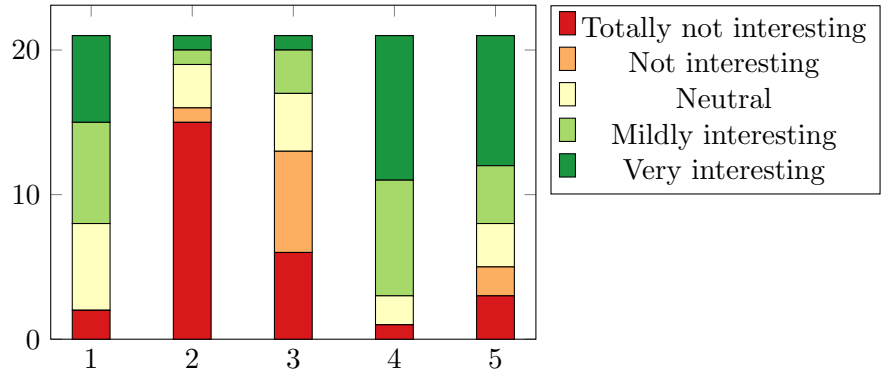


1. The heater is off because there is nobody home [**why**]
2. The heater is off [**what**]
3. It's -4 degrees Celsius outside [**what**]
4. It's colder then 10 degrees Celsius outside [**what**]
5. The location of the users are determined by their smartphone [additionally]
6. Henk is not at home [**what**]
7. The system could not detect your presence [**what**]
8. The heater is off since 6:34am [**when**]
9. It's -4 degrees Celsius since 5:23am [**when**]
10. It's colder then 10 degrees Celsius since 5:23am [**when**]
11. Henk left at 6:34am [**when**]

Figure 8: Interest distribution when determining why a rule is not executed

3.3.3 Situation 3: the future behavior of a device

As previously stated, determining the future behavior of a device can give the user insights about the underlying behavior of a system. The television in the third scenario turns on at *random* timestamps and the participant has to determine why the television turns on every day. In order to determine this, the participants is confronted with the pieces of information shown in figure 9 and gave their interest levels accordingly. In this scenario the why-piece is left out, because we want to determine the building blocks that are required in order to pose an answer that fulfils the needs of the user.

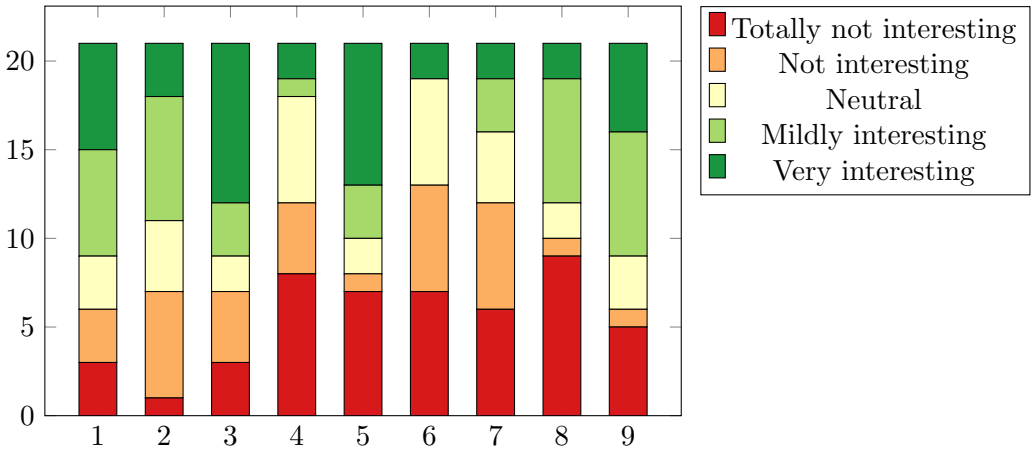


1. The news is on **[what]**
2. The television is on **[what]**
3. The news will end at 7:55pm **[when]**
4. The television will go on tomorrow at 7pm **[when]**
5. The news will start at 7pm **[when]**

Figure 9: Interest distribution when determining the behavior of a device

3.3.4 Situation 4: the future behavior of a rule

A rule is proposed to the participant, in which some expectations were built up. The participant has to discover if the rule behaves as expected and why it does (not). Without the participants having the knowledge, the fourth scenario is an expansion on the third one. However, the user created a new rule which should set an alarm when the movie starts, provided that the television is not already turned on. The participant has to determine why there is no alarm planned. The given pieces of information and their level of interest is shown in figure 10.

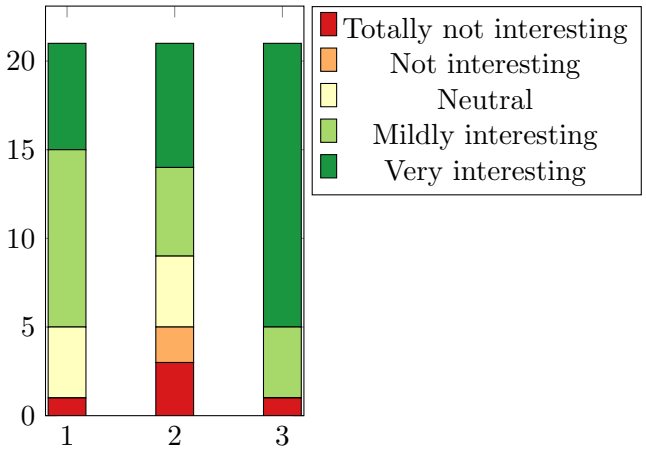


1. There is a rule which turns on the television when the news starts [what if]
2. The rule above will be executed today, at 7pm [when]
3. There is a rule which will set an alarm 5 minutes before the movie starts, if the television is off [what if]
4. The rule above will not be executed in the near future [when]
5. The television is currently off [what]
6. There is no alarm planned [what]
7. The news will start at 7pm [when]
8. The movie will start at 8pm [when]
9. The television will go on at 7pm [when]

Figure 10: Interest distribution when determining the behavior of a rule

3.3.5 Situation 5: verifying a rule

Sometimes a rule is not relevant in the given circumstances, but it is necessary to know if the rule behaves as expected. In this situation, a new rule is created which will power the coffeemaker, whenever the alarm of the user goes off. The participant has to track down if the rule behaves well in the circumstances that it should. In this situation we try to determine if the How-To information piece can be merged with the When information piece. Therefore, we want to determine if the user is more interested in a timestamp or an event on which the coffeemaker will be powered. The information pieces and their level of interest is shown in figure 11.



1. The coffeemaker will go on tomorrow, at 7am. [when]
2. Your alarm will go tomorrow, at 7am. [when]
3. The coffeemaker will go on, when your alarm goes. [when]

Figure 11: Interest distribution when validating a rule

3.4 Analysis and results

In the first situation the reason of the current state of the lamp is given in the first information piece, which is considered to be interesting (as shown in figure 7) as the participants thereby know why the lamp is currently off. The second situation is slightly different, as the current state was not expected by the user. The chart in figure 8 however shows that there is a certain level of interest in the reason of the current state of the heater. We conducted a Wilcoxon Signed Rank Test in order to determine if there is a significant difference in these situations. We will denote the results from the first information piece in the first situation as Group A and the results form the first information piece in the second situation as Group B. The medians of Group A and Group B were both equal to 5. A Wilcoxon-Signed-rank test shows there is a significant effect of Group ($W = 49$, $Z = 2.06$, $p < 0.05$, $r = 0.45$).

In the fourth situation is a rule which will not be executed in the near future, and is therefore relatable to both the first and second situation as these situations depict the (future) behavior of a rule. Therefore we will determine if the why-pieces of information are significantly different in these situations. A comparison between the first and second situation is done above, so we will continue with comparing the why-piece (first information building block) of the first situation with the why-piece of the fourth situation. The why piece that is relevant in the fourth situation, is the third information piece, as this states that the new rule is implemented correctly. The medians of Group A (first information piece of situation 1) and Group B (third information piece of situ-

ation 4) are both equal to 5. A Wilcoxon-Signed-rank test shows there is no significant effect of Group (W = 18, Z = 1.65, p > 0.05, r = 0.36). Next, we will repeat the steps for the why-piece in situation 2 and situation 4. The medians of Group A (first piece of information in situation 2) and Group B (third piece of information in situation 4) are both equal to 5. A Wilcoxon-Signed-rank test shows there is no significant effect of Group (W = 25, Z = -0.84, p > 0.05, r = 0.18). From these tests can be deducted that the why-piece of information is always required, despite if the expectations of the user are in line with the actual behavior of the system.

From figure 7 can be conducted that the when-piece is considered to be interesting by the participants. A Wilcoxon-Signed-rank test showed that the when-piece (sixth piece of information) is considered to be more interesting in comparison with the other information pieces (with the exception for the first information piece). The results that were calculated in these tests are shown in table 2. Just like before, we wanted to know if the when-piece is considered to be important in the other situations where the participants were asked to get the reason of a behavior (situation 2 and situation 4). Firstly, we compare the results from the first situation with the second situation (information piece 8). The medians of Group A (sixth piece of information in the first situation) and Group B (eight piece of information in the second situation) are 4 and 3, respectively. A Wilcoxon-Signed-rank test shows there is no significant effect of Group (W = 71, Z = 0.93, p > 0.05, r = 0.20). Next, a comparison is made between the when-pieces of the first situation and the fourth situation (information piece 9). The median of Group A (when piece of first situation) and Group B (when piece of fourth situation) are both equal to 4. A Wilcoxon-Signed-rank test shows that there is no significant effect of Group (W = 64.5, Z = -0.12, p > 0, r = 0.03). At last we repeat the steps for the when-piece of information in the second- and fourth situation. The median of Group A (when piece of second situation) and Group B (when piece of fourth situation) are 3 and 4, respectively. A Wilcoxon-Signed-rank test shows that there is no significant effect of Group (W = 50, Z = -0.89, p > 0, r = 0.19). These tests show that when the chatbot has to answer the why-question, the users require the timestamp of occurrence.

	1	2	3	4
P-value	0.999573	0.000275	0.001433	0.052098
Effect size	0.691610	0.734694	0.682540	0.539683
	5	7	8	9
P-value	0.009950	0.019680	0.006912	0.000636
Effect size	0.650794	0.562358	0.555556	0.657596

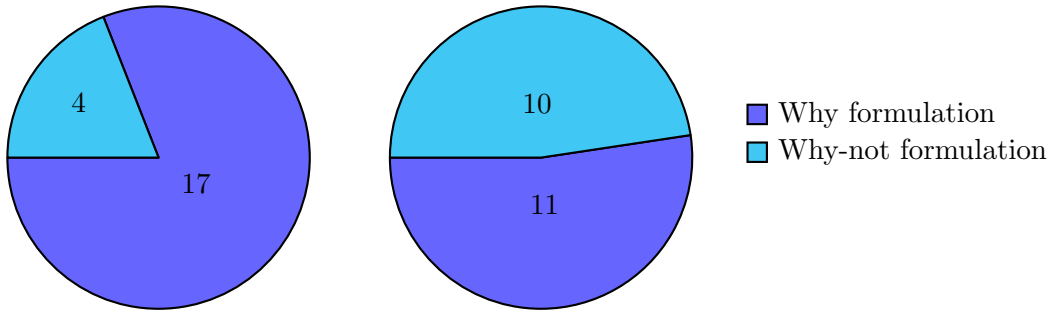
Table 2: Wilcoxon Signed Rank Test results: comparing the when piece of information of the first situation (information piece 6), with the other given pieces. The results show that the participants were significantly interested in when the behavior occurred.

The second situation describes a rule which was expected to be executed (as the triggers should be fulfilled). The rule states that whenever it's colder than 10 degrees Celsius and there is anyone home, the heater has to turn on. The triggering entities for this rule are the inhabitants of the house and the thermometer, from which the states are described in information piece 3, 4, 6 and 7. However, the third and fourth information piece both describe the state of the thermometer, but in another way. The third information piece gives the raw value that is read from the thermometer, while the fourth information piece describes that one of the triggers is fulfilled. A Wilcoxon-Signed-rank test is conducted in order to determine if there is a significant difference in terms of interest for these information pieces. The median of Group A (third information piece) and Group B (fourth information piece) are both equal to three. A Wilcoxon-Signed-rank test shows that there is no significant difference of Group ($W = 55.5$, $Z = 0.11$, $p > 0$, $r = 0.02$). This means that the participants do not prefer one of the formulations over the other one. However, the third and fourth information piece of the second situation describe the trigger that is fulfilled, while the sixth and seventh piece are the reason why the rule is not executed. We determined if there is a significant difference in terms of interest comparing the sixth and seventh information piece with the third one. The first test that is conducted consists of the comparison between the third (Group A) and sixth (Group B) information piece. The median of Group A and Group B are both equal to 3. A Wilcoxon-Signed-rank test shows that there is no significant effect of Group ($W = 79.5$, $Z = 0.18$, $p > 0.05$, $r = 0.04$). Secondly, we compare the third information piece (Group A) with the seventh information piece (Group B). The median of Group A and Group B are 3 and 5, respectively. A Wilcoxon-Signed-rank test shows that there is a significant effect of Group ($W = 3.5$, $Z = -3.71$, $p < 0.05$, $r = 0.81$). This means that the participants were significantly more interested in the state of the entity that does not fulfil the trigger of the rule, in comparison with the states that describe the entities that are fulfilled.

The third situation described the current and future behavior of the television and TV guide. Underlying is a rule which controls the television according to the program that is currently playing according to the TV-guide. From this test, we wanted to know if the participants were more interested in the current behavior of the system, in comparison with the future behavior. First, we compared the second information piece (Group A) with the fourth information piece (Group B), as they both describe the behavior of the television. The median of Group A and Group B are 1 and 4, respectively. A Wilcoxon-Signed-rank test shows that there is a significant effect of Group ($W = 2.5$, $Z = -3.85$, $p < 0.05$, $r = 0.84$). Next, a comparison is made between the first- (Group A) and fifth (Group B) information piece, as they both describe the behavior of the TV-guide entity. The median of Group A and Group B are both equal to 4. A Wilcoxon-Signed-rank test shows that there is no significant effect of Group ($W = 34$, $Z = -0.15$, $p > 0.05$, $r = 0.03$). We see that there is a significant difference when the information is about the rule's result, but not when the information is about the rule's trigger.

The test in situation 5 was conducted in order to determine the preference of time indication for the user. The coffeemaker will go on whenever the user's alarm goes off, which happens at 7am. Both the first and third information piece verify that the implemented rule behaves as expected. We want to determine if there is a significant difference in terms of interest of the participants in knowing the timestamp or the event on which the coffeemaker will turn on. Therefore a Wilcoxon-Signed-rank test is conducted, comparing these pieces of information with each other. First, a comparison is made between the first information piece (Group A) and the third information piece (Group B). The median of Group A and Group B are 4 and 5, respectively. A Wilcoxon-Signed-rank test shows that there is a significant effect of Group ($W = 11$, $Z = -2.67$, $p < 0.05$, $r = 0.58$). Next we repeat this procedure in order to compare the second information piece (Group A) with the third information piece (Group B). The median of Group A and Group B are are 4 and 5, respectively. A Wilcoxon-Signed-rank test shows that there is a significant effect of Group ($W = 10$, $Z = -2.57$, $p < 0.05$, $r = 0.56$). This means that the participants were more interested in the event on which the coffeemaker would turn on, rather than the timestamp.

In order to determine the utilized question formulation, we categorized the type of questions that the participants filled in. The insights of this test can then be used in the formulation of the training phrases of the chatbot. First we distinguish the why formulation and the why-not formulation. With the why formulation, the participant asks for the reason of the current state of a device (*e.g. Why is the lamp off?*). This while the why-not formulation asks for the reason why a device does not have another state (*e.g. Why is the lamp not off?*). In figure 12a we can see the distribution of these formulations in the first situation, while in figure 12b the distribution of situation 2 is given



(a) Distribution in situation 1 **(b)** Distribution in situation 2

Figure 12: Distribution of the formulation types when asking for the cause of a particular behavior. The results are grouped in a why- and why-not formulation.

Secondly we distinguish two types of formulations in case of defining the future behavior of a device. In this situation the current behavior is the same as the future behavior of the device. The distribution of the formulations is shown in figure 13.

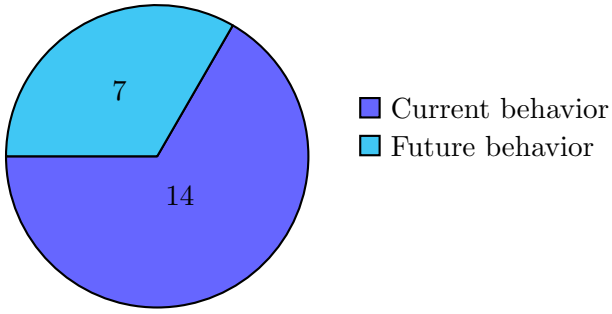
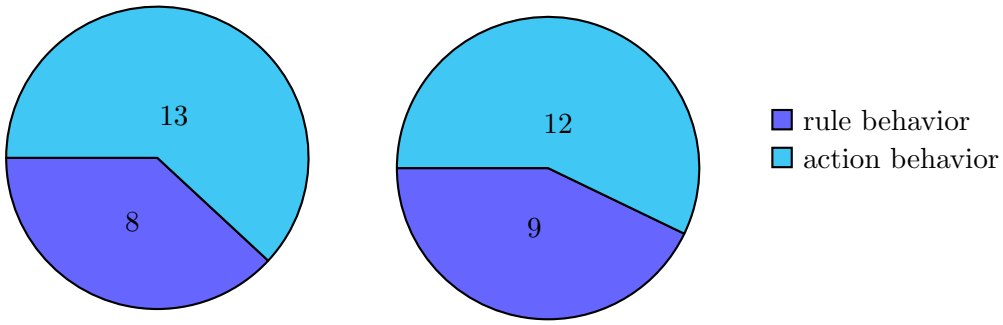


Figure 13: The questions asked in the fourth situation are categorized by the goal of the participant. We distinguish participants that asked for the current behavior of their device, in comparison to the participants that asked for the upcoming behavior.

When asking about a rule's behavior (and the actions that may be taken), we distinguish two types of formulations. The first type asks why the rule is not executed (*e.g. why is there no alarm planned for the movie at 8pm?*), while the second type asks why there is no action taken (*e.g. why did you not plan an alarm?*). The distribution is pictured in figure 14.



(a) Distribution in situation 4 **(b)** Distribution in situation 5

Figure 14: When asking about a rule's behavior, the participants were tend to ask about the outcome of the rule, rather than the rule execution itself.

3.5 Discussion

In the first-, second- and fourth situation, the reason of an occurrence had to be determined by the system. We proved that our hypothesis **H1** can be accepted, as the why-information piece in these situations were significantly considered to be interesting. There was no significant difference comparing the why-pieces between the different situations, from which we can conclude that the participant always want to know the reason of the current state, no matter what the expectations of the behavior were.
More interestingly, we found that the participants were interested in the time of occurrence for each of these situations. There was no significant difference in terms of interest, when comparing the when-pieces in the different situations. Therefore, we can conclude that when the user wants to know the reason of an occurrence, it is required to add the timestamp of occurrence in order for the user to fully understand the underlying behavior of the system.

At the start of the experiment, we assumed that the user would ask their question utilizing the why formulation (in this case: *why is the lamp off?*). Figure 12a however, shows us that 4 out of the 17 participants utilized the why-not formulation, which can be dedicated to the expectation that the lamp would be on in the given situation. However, this means that both the why-formulation and why-not formulation are utilized interchangeably and need to result in the same answer. This means that our hypothesis **H2** has to be rejected, and therefore we can not assume that the user expresses his or hers expectation by utilizing the why-not formulation. The second situation contains a rule which is unexpectedly not executed. It is evident to say that this situation is relatable to the first one, in terms of the question formulation. As with the first situation, the participants used the why- and why-not formulation interchangeably, as shown in figure 12b. An interesting conclusion that came out of the test, is that all the participants would ask why the result of the rule did not occur, instead of why the rule was not executed. This means that, when a user asks a why/why-not question about a particular state, the chatbot also needs to consider the rules in the system that are not executed.

From the results that came out of the tests on situation 3, we can conclude that the participants were interested in the time on which the rule is executed, rather than the time on which each trigger is fulfilled. Therefore, our hypothesis **H3** can be accepted. More interestingly are the results in figure 13, in which is depicted that the majority of the participants asked why the current behavior is occurring, instead of why the future behavior will occur. Figure 14a shows us that, just like in situation 2, the participants were tend to ask about the outcome of the rule, rather than about the execution of the rule itself.

The first-, second- and fourth situation all describe the behavior of a particular rule. We have proven that in each of this situation the participants were interested in the information that described that the implemented rule is correctly integrated in the system. This means that the **H4** hypothesis can be accepted, as the user wants to know if there is a problem in the implementation of the rule, and if not, why the current behavior is happening.

Furthermore, we utilized the results from the test of situation 3 in order to determine if the users were interested in the states of the entities that could trigger the rule. However, the results show us that the participants were more interested in the states of the entities that caused the rule to not be triggered, rather than the triggers that are fulfilled. Therefore, we can not accept our **H5** hypothesis, as the participants were not interested in all the states of the triggering entities.

Situation 5 was built in order to determine if the participants were more interested in the timestamp of an occurrence in comparison with the event which will cause the occurrence of a state. From the conducted Wilcoxon-Signed-rank tests, we can conclude that the event on which the state will occur is considered significantly more interesting as the timestamps. This means that when a user asks when a particular state will occur, the system needs to respond with the event on which the state will occur. This however means that hypothesis **H6** is rejected, as the participants were not interested in both the timestamp and the event. However, additionally, a timestamp can be given as the results in figure 11 shows us that the timestamps are not considered to be totally not interesting.

3.6 Conclusions

Given the distributions in section 3.4 we can make some interesting conclusions to take in consideration for the implementation of the chatbot. In figure 7 we can see that the participants where mostly interested in the why- and when-aspect of the information pieces, when determining the cause of a state. So the chatbot has to respond with the reason and the time of occurence. Figure 12 shows us that the why and why-not formulations are used interchangeably. This means that the chatbot has to respond in the same way on both these questions. This disempowers our hypotheses that the why-not formulation implicates that the current state was not expected, while the why formulation would only be to determine the reason alone. When determining the cause of a non-execution of a rule, the users are less interested in the fulfilled triggers. However, the participants were interested in the triggers that caused the rule not to be executed (as seen in figure 8 and figure 11).

Figure 9 shows us that, as with asking the reason of the current state, the why- and when-aspect are desired in the explanation about the future behavior of a device. The diagram in figure 13 shows us that the majority of the participant asked for the reason of the current state, while the hypotheses was that they would ask about the future

behavior of the device. However, figure 9 shows us that the participants where mostly interested in the future-aimed information pieces.

From figure 10 and figure 11 we can conclude that, when determining or validating a rule, the most interesting part is a confirmation about the rule. This confirmation is in the form of a textual representation of the rule's behavior. When asking questions about a rule, the participants tend to ask about the action that was taken, rather then asking about the rule itself. In figure 14 we can see the distribution of the two used formulations.

4 HassTalk architecture

Reasoning about and understanding a set of trigger-condition-action rules requires a substantial mental effort from users and depends on many external factors. Therefore we introduced HassTalk, a chatbot that adds intelligibility to a IoT-system. By enabling the user to ask questions like "Why did X happen", "When did X happen", "What will happen if X happens"... HassTalk tries to achieve a higher rate of understandability about the user's Smart Home system and its behavior.

The infrastructure of HassTalk consists of four major elements, as shown in figure 15. The root of the infrastructure, Home Assistant, is an open source home automation tool. It enables the end-user to have a smart home instead of just a collection of smart things. It supports connecting smart devices of different manufacturers and the creation of automated behavior using TCA-rules. Therefore, the user is able to automate relatively simple tasks, like waking up to fresh coffee. The user therefore can create a TCA-rule as follows: WHEN I wake up (Trigger) AND I am home (Condition) THEN power the coffeemaker (Action). Home Assistant delivers API endpoints to fetch the existing devices in the system, and their states. Despite the fact that Home Assistant enables the creation of TCA-rules, this feature is currently re-implemented in the FORTNIoT-Toolkit of Sven Coppers.

The FORTNIoT-Toolkit serves multiple API endpoints to get more insights in the behavior of the Home Assistant-system. Aside from providing the devices and their states, this API enables to fetch the future and historical behavior of these devices. FORTNIoT communicates with the Home Assistant in order to fetch information about the existing devices and their respective states. Because the TCA-rules are built in the FORTNIoT-Toolkit, it also capable of sending manipulation requests to the Home Assistant API. In this manner TCA-rules are actually executed. Implementing the TCA-rules locally is done in order for the FORTNIoT-Toolkit to enable reasoning and future predictions about the Home Assistant system. However, it is a simple addition to enable utilizing the rules in the Home Assistant system, as these utilize the same structure. As said earlier, TCA-rules consist of three components: a Trigger (eg. When I get home), a Condition (eg. It is cold outside) and an Action (eg. Turn on the heater). This is the case in both the Home Assistant as the FORTNIoT-Toolkit.

For the system we made in this research, I have added several API endpoints to enable alternative future simulations, fetch a description of a rule, get the cause of a device's state and get the cause why a particular rule is not executed or why a device has not a particular state. Figure 15 shows how the communication flow works in the system. To emphasize what was already implemented by Sven Coppers and what I have implemented, I have split up the types of information that are communicated between the FORTNIoT-Toolkit and the HassTalk API.

HassTalk utilizes the endpoints of the FORTNIoT-Toolkit to fetch the information that is required in order to formulate a proper answer to a user's question. For example, when the user asks "What is the state of my lamp?", the HassTalk API will fetch the state of the device from the FORTNIoT and pass this to the next node in the system.

The last node of the architecture, DialogFlow[3], is a framework created by Google. This framework enables it to create a chatbot with custom behavior, by matching the user's input with an intent. When we take the example from above, where the users asks for the state of the lamp, the DialogFlow Client will analyze the question and match it with the Why.State intent. This means that DialogFlow recognized that the user wants to know the reason of a particular state. In order to determine the state, the HassTalk API needs to know which device it has to take in consideration. Therefore, DialogFlow recognizes the parameters given by the user and will match it with an identifier that can be used in the HassTalk API.

4.1 Home Assistant

Home Assistant is an open source home automation that is powered by a worldwide community of tinkerers and DIY enthusiasts. This system enables combinations of smart devices of multiple manufacturers, so we are not limited to stay in a particular eco-system. It is very lightweight, so it suits perfect to run on a Raspberry Pi or a local server. HA will track the state of the devices in your home, so you have an overview of all your devices in one glimpse. Moreover, it allows controlling these devices from a single interface, and this solely on a local system, so no data is stored in any cloud. This means that the HassTalk system can also be built in a local environment, so it is independent of external factors and servers. HassTalk does not keep track of any personal information or input, so that we can respect the privacy of the user at a maximal level. Home Assistant also allows to automate behavior in a Smart Home by creating routines/rules. An example of such a rule can be *"Dim the lights when I start watching a movie on my Chromecast"*. However, we do not use this particular feature in our system, as this is handled by the FORTNIoT-Toolkit.

[3] https://dialogflow.com/

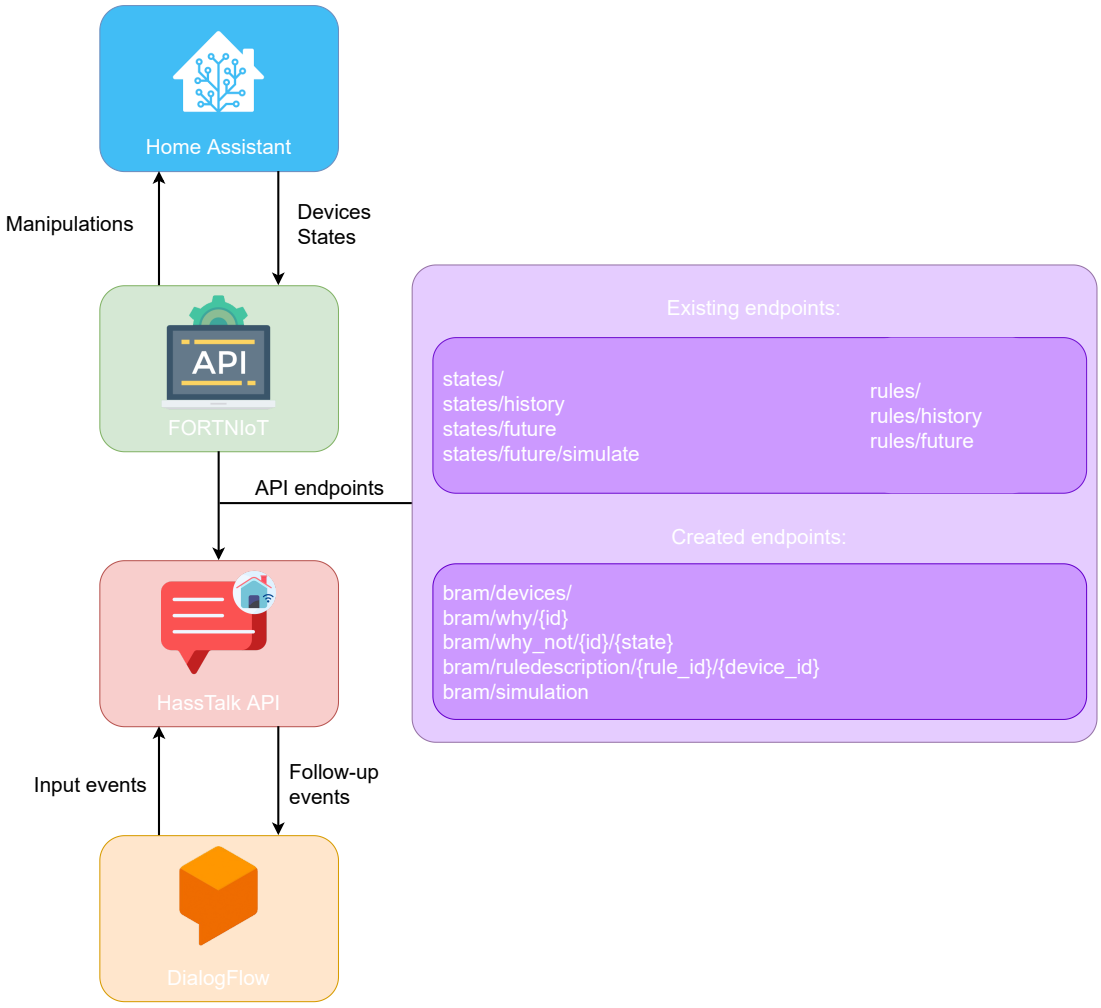


Figure 15: Architecture of the HassTalk system: Home Assistant (connecting and manipulating smart devices in a smart home), FORTNIoT-Toolkit (FORTNIoT-Toolkit to enable reasoning and future predictions about the system's behavior), HassTalk API (Translates the information in the FORTNIoT-Toolkit in natural language) and DialogFlow (Chatbot that enables communication between a human and machine)

4.2 FORTNIoT

The FORTNIoT-Toolkit is built by Sven Coppers and adds intelligibility to the Home Assistant system. This API serves multiple endpoints that are utilized by the HassTalk API to fetch the information that is needed to formulate answers to the users questions. On top of the existing endpoints, I built several new ones to help the chatbot in reasoning and fetching the right information in a given circumstance. We will discuss the used endpoints in this section to give more insight in the working principles of HassTalk.

4.2.1 Devices

In order for HassTalk to know which devices are implemented in the system a new API endpoint is implemented, that will respond with a JSON-array containing the different devices. Each entity in this array contains an identifier and a friendly name of the device. An example of such an JSON-object is shown in listing 1. This API-endpoint is used to map the device identifier to the friendly name, so it remains understandable for the end user. It is also used to check if an asked device really exist in the system, and if not to respond which devices are known by the system.

```
1 {
2     "friendly_name": "Smart TV",
3     "id": "media.smart_tv"
4 }
```

Listing 1: Resulting JSON-object received by the Devices-endpoint of FORTNIoT. This response contains a friendly name and identifier for the devices existing in the system.

4.2.2 State of a device

As stated previously, each device in the Smart Home system has its own state. Therefore, the FORTNIoT-Toolkit has an endpoint that enables fetching the states of the devices in the system. Despite the fact that devices can have a different type (eg. lamp vs. television) or manufacturer, the states are represented in a uniform way. In this manner, we are able to treat every device in a similar way, without having to be concerned about the manufacturer's implementation of the smart device. The states route of the API returns the current situation of the devices in the system. A GET-request results in a JSON-array of objects as in listing 2. Such a JSON-object contains the following parameters:

- *last_changed*
  The time of when the state has changed to it's current value.
- *attributes*
  Device specific parameters (with exception for the friendly name, which is given for every device). These describe more deeply the current state of the device.
- *entity_id*
  The unique identifier of the device.

- *context*
  Contains information about the cause of the state. The context identifies the situation in which the state results.

- *state*
  Represents the current state of the device. The values for this parameter are device specific.

```
"media.smart_tv": {
    "last_changed": "2020-05-12T14:23:29.386+02:00",
    "attributes": {
        "friendly_name": "Smart TV",
        ...
    },
    "entity_id": "media.smart_tv",
    "context": { ... },
    "state": "on"
}
```

Listing 2: JSON-object describing the state of a device in the system (in this case a Smart TV). A device's state contains a timestamp of occurrence, device-specific attributes, an identifier, a context and the current state of the device

History of states
The FORTNIoT-Toolkit also serves a route to fetch the history of the states. This endpoint returns a JSON-array containing objects like shown in listing 2 and is used by the HassTalk API to learn the possible values that a device's state can have.

Future of states
With the API, it is possible to get an estimation of the future behavior of the devices. This is done by validating the rules of the system, according to the expected behavior of the 'real-world'. For example, consider a rule where a light has to turn off, whenever the sun goes above horizon. The system will search for the next occurrence of the sun going above horizon and fetch the time of that occurrence, as it will be the same time the light will go off.
Fetching the expected behavior of the system is done by performing a GET-request to the future-route. This will result in an array containing JSON-objects as shown in listing 2. These objects resemble the expected states of the existing devices, at a given timestamp in the future. This enables HassTalk to estimate the future behavior of the devices and give an explanation to the user what will happen with their system.

4.2.3 Rules

As the TCA-rules are implemented in the FORTNIoT-Toolkit, HassTalk has to fetch them by executing a GET-request to the rules-route. This will result in a JSON-array containing a description of the implemented rules. Each rule has the following attributes:

- triggeringEntities
  An overview of the existing entities that can trigger the rule to execute.
- actions
  The description of the action that will be the result of the rule, including the device identifier of the device that will be affected by the rule.
- description
  A textual representation of the trigger that will cause an execution of the rule.

Originally the triggeringEntities-attribute was not included in the response. This part is added for this project, in order to get an overview of the entities that can cause the rule to be triggered. This is also the case for the device identifier in the actions-parameter, this time to get an overview of the entities that will be affected by the rule's execution. This route is used by HassTalk to form a textual representation of a rule, which can be done by combining the description of the trigger and actions.

```
"rule.start_news": {
    "triggeringEntities": [
        "person.Bram",
        "guide.tv_guide"
    ],
    "actions": [
        {
            "description": "Turn on the television",
            "deviceIdentifier": "media.smart_tv"
        }
    ],
    "description": "When anyone is home AND the news is on"
}
```

Listing 3: Resulting JSON-object of the rules-route of FORTNIoT. This object contains the entities that can cause the rule to be triggered, the actions taken when the rule is triggered and a description of the trigger of the rule.

Executions

In order to get information about the passed and upcoming rule-executions, the FORTNIoT-Toolkit serves two routes to fetch that information. The history of the rule executions is not used in HassTalk, as the application is currently mainly focused on the future. Performing a GET-request to the execution route will result in a JSON-array containing a description of the different rule-executions that will happen in the future. Such an execution (shown in listing 4) contains the following attributes:

- entity_id
  The identifier of the rule that will be executed.
- datetime
  The timestamp on which the rule will be executed.
- trigger_contexts
  The contexts that will cause the rule to be triggered.
- action_contexts
  The contexts resulting from the rule execution.

HassTalk uses this route to get an overview of the future rule-executions. In this manner, the system can determine if a rule is the cause of a state, and if so, which rule.

```
{
    "entity_id": "rule.start_news",
    "datetime": "2020-05-12T19:21:13.276+02:00",
    "trigger_contexts": [ ... ],
    "action_contexts": [ ... ]
}
```

Listing 4: JSON-object describing an execution of a specific rule (in this case rule.start_news). It contains an entity id of the executed rule, a timestamp on which the rule was executed, the contexts that resulted in the rule being triggered and the resulting context of the execution.

Rule description

Rules can effect multiple entities at once, for example a rule that turns the television and the lights off when somebody leaves the building. In order for HassTalk to give a more precise explanation of a rule's effects on a device, I created this route to fetch the consequences of a rule on a specific device.

The route takes a rule- and device identifier to define the effects of the rule on that device. The framework will then fetch the Rule-instance from its database and takes the trigger description to add to the result. Furthermore, a new method is implemented to fetch the actions taken on a device whenever the rule is executed. This is done by iterating through the devices which are effected and returning the Action-instance which is applicable on the device in question. Performing a GET-request to this route will result in a simplified representation of the rule's description and the effects that a rule has on the given device (shown in listing 5)

```
{
    "trigger": "When nobody is home",
    "action": "Turn off the lights",
    "rule_id": "rule.nobody_home"
}
```

Listing 5: JSON-object containing a textual description of a rule. The object contains a phrase representing the trigger on which the rule will be executed, a phrase representing the taken action when the rule is executed and an identifier for the rule in question.

4.2.4 Simulation

Before explaining the working principle of this route, an explanation is needed in order to understand what we mean by a "simulation". For example, imagine if you want to know what will happen when the sun goes below horizon, but it is daytime. Fetching the upcoming behavior from the future-states endpoint (described in section 4.2.2), would not give the required information as in the predicition the sun will behave as expected in reality. In order to know the consequences of a particular change of state, we have to create an adjusted state where the state-attribute of the sun will be adjusted to *"below_horizon"* and the value of *last_changed* is changed to the current timestamp. Thereby, we can explain to the FORNTIoT-Toolkit which state is alternated and how it is alternated. The newly created state will be injected in the future prediction algorithm of the FORTNIoT-Toolkit. Instead of taking the expected behavior of the sun in reality, the FORTNIoT-Toolkit now takes the adjusted state in consideration to validate the rules and behavior of the system.

Despite the fact that there was already a simulations-route implemented, I added a slightly adjusted version to the framework in order to simplify the process. The original route takes a list of adjusted states and a list of rules which have to be enabled. This would mean a lot more of communication between the HassTalk API and the FORTNIoT-Toolkit , as HassTalk needs to have notion of the existing rules in the system. To circumvent this I created a new route that only accepts one alternative state. Therefore it is not needed to attach a list of enabled rules.

The algorithm that performs the simulation however, is the same as the original one. It takes a list of adjusted states (in our case a list with one element) and a list of enabled rules. Complying to these requirements is done by fetching the existing rules in the system and simply passing them to the algorithm. Performing a GET-request to this route will result in a list of state as depicted in listing 2, followed by a list of rule executions like in listing 4.

4.2.5 Cause of device's state (why-route)

In order to identify the cause of the current state of a device, I implemented the why-route in the FORTNIoT-Toolkit. The workflow of the algorithm is shown in figure 16 and takes a device identifier as input. It uses the identifier in order to fetch the current state of that device from the internal Context Manager. As previously explained in section 4.2.2, a device's state contains a context element, which contains information about the cause of that state. The context of a state contains 3 attributes: an identifier, a parent identifier and a user identifier. In order to determine the cause of the state, the algorithm starts by checking if the user identifier has a valid value.

In the case it has, the state of the device is caused by a user and the value points to that particular user. So the algorithm takes that identifier and uses it to get the User-instance of the actor that has caused the change. Once found, it will return the user identifier and name of the actor, along with the information about the current state of the device (the device identifier, friendly name, current state and the time of change). An example of such a response is given in listing 6, in which a user named "Bram" has manually turned the lamp on.
When the device's context does not contain a user id, the algorithm assumes that the state is caused by a rule execution. In order to get the rule which resulted in the current state, I implemented an algorithm to get the latest rule execution which affected the device in question. As the FORTNIoT-Toolkit keeps track of rule executions, finding the right rule is a matter of iterating through the past rule events and taking the last event which affected the device. Once found, it will return the information about that rule (rule identifier, trigger- and action description), combined with the information about the current state of the device (the device identifier, friendly name, current state and the time of change). An example of such a response is given in listing 7, in which there is a rule that turns on the lamp whenever the sun goes below horizon.

If any of the above methods fail or the user or rule can not be found, the algorithm will return an unknown-response. This is only the case if the current state of the device was like that, when the FORTNIoT-Toolkit was initiated. An example of a response, in case of an unknown reason, is given in listing 8.

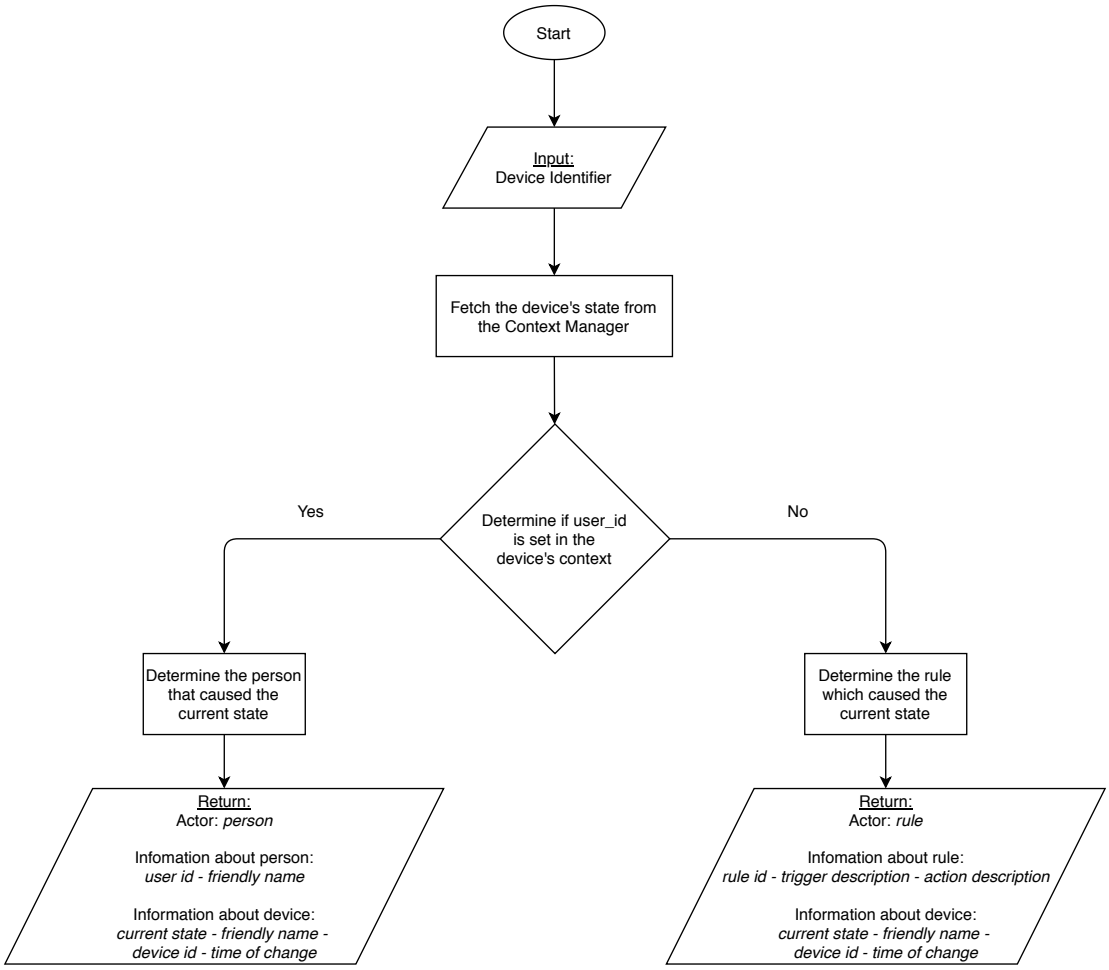


Figure 16: Workflow diagram of the why-route. Starting with the identifier of a device, the algorithm checks the reason of the occurrence of the device's current state.

```
{
    "actor": "user",
    "state": "on",
    "friendly_name": "Lamp",
    "device_id": "switch.lamp",
    "user_id": "928fd9bc47ed4a4eacea420fe13de68c",
    "user": {
        "id": "928fd9bc47ed4a4eacea420fe13de68c",
        "name": "Bram"
    }
}
```

Listing 6: State of the device is caused by a manual user's action. In this case, there is a user 'Bram' which has turned the lamp on.

```
{
    "actor": "rule",
    "state": "on",
    "friendly_name": "Lamp",
    "device_id": "switch.lamp",
    "rule_id": "rule.sun_set",
    "rule": {
        "trigger": "When the sun goes below horizon",
        "action": "turn on the lamp",
        "rule_id": "rule.sun_set"
    }
}
```

Listing 7: State established by rule execution. The rule with identifier rule.sun_set is executed (because the sun went below horizon) and resulted in the lamp being on.

```
{
    "actor": "unknown",
    "state": "off",
    "friendly_name": "Lamp",
    "device_id": "switch.lamp"
}
```

Listing 8: State established by unknown reason. The system was not able to identify the cause of the current state of the lamp. This behavior occurs whenever the state of the device was like that, before the Home Assistant/FORTNIoT started.

4.2.6 Cause of (non) execution of rules (why not-route)

HassTalk may have to determine why a given device does not have a specific state, for example, when a rule is not executed unexpectedly. As input parameters the route takes the device identifier and the desired state, on whether it has to check if it can be achieved by rule executions. The first step of the algorithm is determining the cause of the current state, which utilizes the algorithm described in section 4.2.5. This is done because of the fact that this is also a reason that the queried state is not the current state of the device. Secondly, the algorithm will fetch a list of the rules that will affect the given device once executed. Once we got the list, we iterate through the items to determine if the rule will result in the asked state. If so, the algorithm will add it to the result-list. The result will contain all the rules that affect the device in question and more specifically, will result in the queried state.

Now that the algorithm got the Rule-instances that will result in the asked state, the results will be transformed to make it understandable and usable for the HassTalk API. For each rule we save the description of the trigger and action of the rule. Next, the algorithm determines if the rule would be executed in the given circumstances, if so, the rule would be classified as a *denied rule*. This indicates that the reason of the current state denied the execution of the rule which would result in the asked state. Take for example a rule which turns on the lamp whenever the sun goes below horizon, but there is a user who manually turned the lamp back off. In that case, the system determines that the described rule was executed, but denied by a manual user action.

Next, the algorithm fetches the entities of the system that are part of the rule's trigger (in the example described earlier, this would be the sun, as it will trigger the rule whenever it goes below horizon). For each entity in that list, the algorithm determines the current state and friendly name of the entity. Moreover, it will be determined if the given entity causes a trigger. In this manner, HassTalk can eliminate fulfilled parts of the trigger and determine which parts caused the non-execution of the rule. If the algorithm fails in determining the current state of the entity, it will fill in the *unknown* indicator, as it indicates an issue that goes beyond the knowledge of the system (eg. entity was not reachable).

The result that will be returned after the request will include the reason of the current state and a list of rules that will result in the asked state. For every rule in that list, it is given why this rule is (not) executed. An example of such a response is given in listing 9. In the example the algorithm determined why the device with identifier *cleaner.Cleaner* is not turned off. The why-part of the result will contain an object as shown in listing 6, listing 7 or listing 8. Secondly, the algorithm determined that there is a rule which will turn off the cleaner whenever there is anyone at home. But this rule was not triggered, as the current state of *person.Bram* is unknown, and therefore considered *not home*.

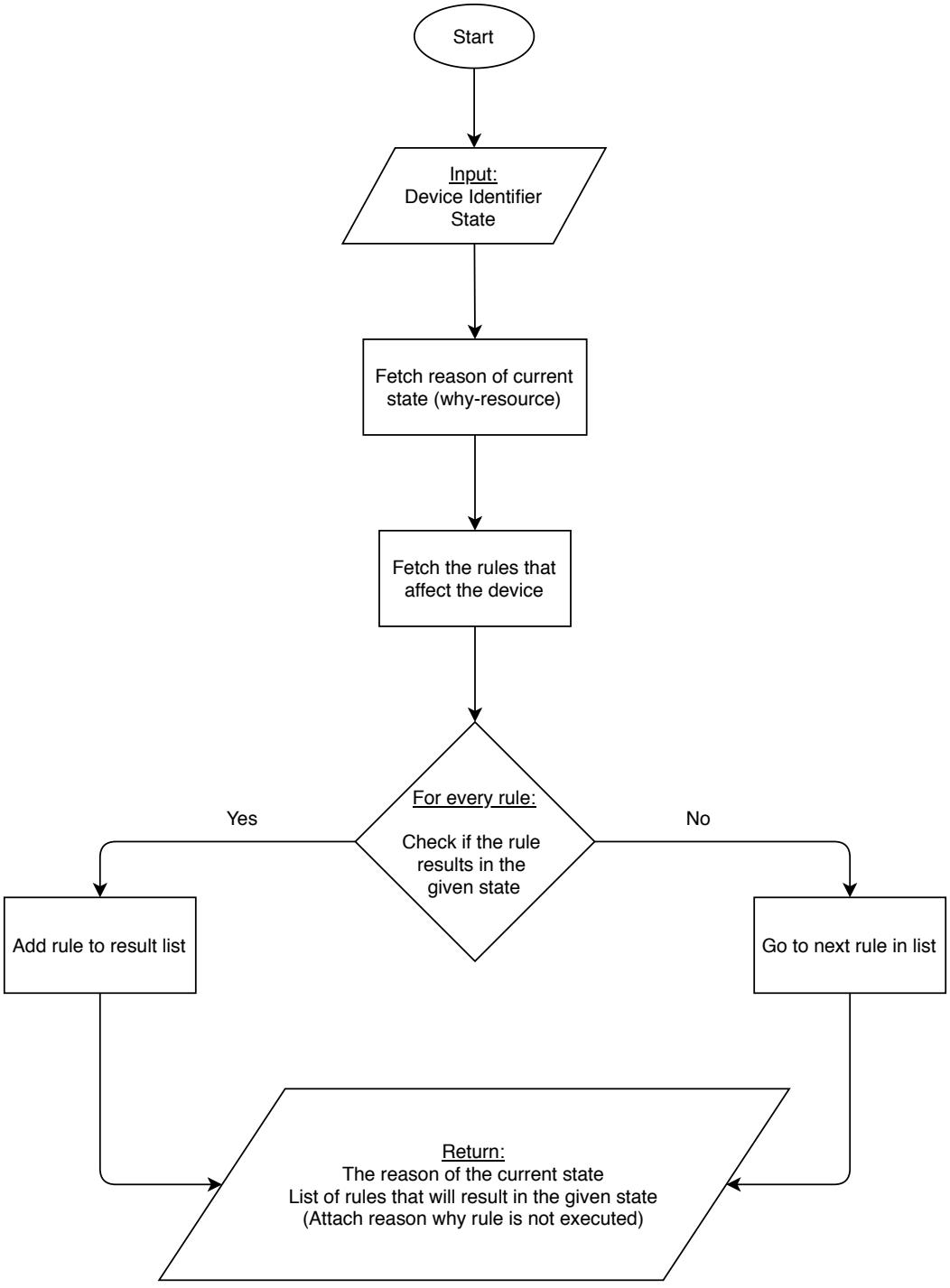


Figure 17: Workflow of the why not-route. The algorithm starts with a device identifier and a queried state and checks why the current state occurred. Moreover, it is determined which rules in the system will result in the device having the queried state.

```
{
    "why": {
        ...
    },
    "rules": [
        {
            "rule_id": "rule.cleaner_off",
            "trigger": "When anyone is home",
            "current_states": [
                {
                    "value": "unknown",
                    "device_id": "person.Bram",
                    "device_name": "Bram",
                    "is_triggering": false
                }
            ],
            "is_triggered": false
        }
    ]
}
```

Listing 9: Resulting JSON-object of the created why-not endpoint of the FORTNIoT-Toolkit. It contains the reason of the current state, along with a list of rules which will result in the queried state. In this case, there is one rule that would turn off the cleaner when there is anyone home. While currently, the cleaner is on for some reason.

4.3 Training DialogFlow

DialogFlow enables creating a chatbot, powered by AI. A DialogFlow agent is a virtual agent that handles conversations with your end-users. It is a natural language understanding module that understands the nuances of human language. DialogFlow translates end-user text or audio during a conversation to structured data that is understandable for the HassTalk API. In this manner, a developer can train the chatbot with training phrases including specific types of information. These training phrases will then be coupled to an action and will perform a POST-request to the HassTalk API on fulfillment, in order to fetch the answer.

4.3.1 User's intent

In order to determine the intent of the user that uses the chatbot, DialogFlow enables to create intents and train them with questions. An example of such an intent is given in figure 18, where an intent is created in order to determine the current state of an entity. The intent has to be fed with training phrases in order to enable DialogFlow to execute their AI-algorithm to determine more questions in the same manner. In this case there is one parameter required in the question of the user, namely the name of the device that is queried. Once DialogFlow has detected the intent of the user, it will perform a POST-request to the root of the API. This request will contain the initial phrase the user has given, the parameters and the action that has to be taken in order to get the answer. A full list of the implemented input intents can be found in Appendix C.

Training phrases
For each intent, a list of training phrases is required in order for DialogFlow to detect the intent of the user. When an end-user expression resembles one of these phrases, DialogFlow matches the intent and parses the input. The input is then added to the request-body that will be sent to the HassTalk API.

Action intents
Action intents represent something users want to get done by posing their expression. The *action* field is a simple convenience field that will assist the HassTalk API in executing logic. When an intent is matched at runtime, DialogFlow provides the action value to the fulfillment webhook of the HassTalk API, which will trigger specific logic in the service.

Parameters
When an intent is matched at runtime, DialogFlow provides the extracted values from the end-user expression as *parameters*. Each parameter has a type, called the *entity type*, which dictates exactly how the data is extracted. Unlike raw end-user input, parameters are structured data that can easily be used to perform some logic or generate responses. When building the agent, parts of the training phrases are annotated and configured to be detected as associated parameters.

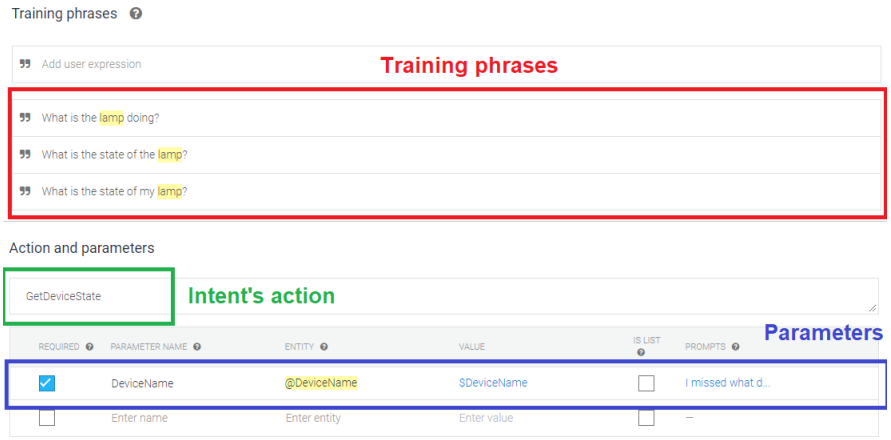


Figure 18: Creating a user intent in DialogFlow. The agent uses the training phrases to determine the intent's action and extracts the parameters from the user's formulation. After an intent is matched, a structured message will be sent to the HassTalk API.

4.3.2 Entity types

As previously explained, training phrases can contain parameters that will be sent to the HassTalk API once an intent is matched. In order to enable HassTalk to understand the parameters, these have to be parsed in a uniformed way. To achieve this, a script is created that will learn DialogFlow the different types of entities and their possible values. These values will be stored in the DialogFlow instance as shown in figure 19. Whenever DialogFlow detects one of the synonyms in the user's expression, it will deduct the reference value and pass this to the HassTalk API. In this manner, DialogFlow has notion of which entities exist and will result in a understandable value for the HassTalk API. This enables error detection when a user asks for an unknown device, state or other value.

In the current implementation there are three types of entities: DeviceNames (Used to identify a particular device in an expression. It extracts the name of the device and deducts the device identifier from that name.), StateAttributes (Indicates which attributes a device can have, for example the brightness for a lamp) and AttributeValues (Indicates the possible values an attribute can have, for example on/off for the state attribute of a switch).

Figure 19: Example of an entity type. A user's intent can contain parameters which have a specific type (eg. timestamp, string, numerical value...). In order to create custom types for these parameters, an entity type is created to make the possibilities of the parameters bounded.

Learning the device names

In order for DialogFlow to detect legitimate devices, I created a script which will scan the FORTNIoT-Toolkit for existing devices. This is done by performing a GET-request to the newly implemented devices-route (section 4.2.1). For each device in the result, there will be an entry created which contains the device's identifier as the reference value and the friendly name as a synonym. Thereby DialogFlow knows which devices exist in the system. If wanted, more synonyms can be given manually afterwards. An example list of the DeviceName entity type is shown in Appendix E.1.

Learning the attributes of a device

In section 4.2.2 is explained that each device has its specific attributes. An example of such an attribute is the color of a smart light bulb. In order to make these attributes questionable, DialogFlow has to know which attributes exist. This is done by performing a GET-request to the states-route of the FORTNIoT-Toolkit. The result will contain the states of all the devices present in the system. The algorithm then scans the attributes of each device and adds them to a list. Afterwards a filter is applied for the attributes that do not change over time and are therefore not elementary to add to the system, an example of such an ignorable attribute is the friendly name. Asking why or when the device got its friendly name does not make any sense, and is therefore skipped in the learning process.

The title of each attribute is used as reference value, as well as synonym. Such as with the device names, more synonyms can be added manually afterwards. An example list of the StateAttributes entity type is shown in Appendix E.2

Learning the possible values of a device's attributes
Now that DialogFlow knows which attributes exist for each device, it only has to learn which values are possible for these attributes. This is done by fetching the list of passed- and future states of each device, in order to have a more complete knowledge about the possible values. For each state-object in the result, the algorithm will scan which attributes are learned previously by DialogFlow and will add these to the system. For the reference value, a prefix is added to be able to determine the attribute on which the value applies. Figure 19 shows the possible values for the state of an entity in the system. As a synonym the raw value of the attribute is filled in, in order for DialogFlow to be able to match these in the user's expression. Like with the other entity types, synonyms are added manually afterwards.

The test described in section 3 showed us that the use of synonyms is very important in this aspect of the chatbot. Therefore, multiple synonyms are added to the list of attribute values. For example, the value *Home*, which is a possible state value for a person, has the synonym *arrives*. This enables the user to form their question like "What happens when Bram gets home" or "What happens when Bram arrives". Both of these expressions will be parsed to the value *state_home* in order for the HassTalk API to understand the intention of the end-user. An example list of the AttributeValues entity type is shown in Appendix E.3

4.3.3 Answer intent

Creating an answer intent in DialogFlow is analog to creating a user intent. But in this case, no training phrases or action are given, but rather response phrases and a follow-up event. Once the HassTalk API fires such an event, DialogFlow knows which intent to use in order to form the answer to the end-user. In figure 20 the answer intent is shown for the user's intent shown in figure 18. This intent will be used whenever the *GotStateAnswer* event is fired. This event must contain the parameters DeviceID, DeviceName and State which are given by the HassTalk API. These parameters are then used to form the answer for the end-user. A full list of the implemented answer intents can be found in Appendix D

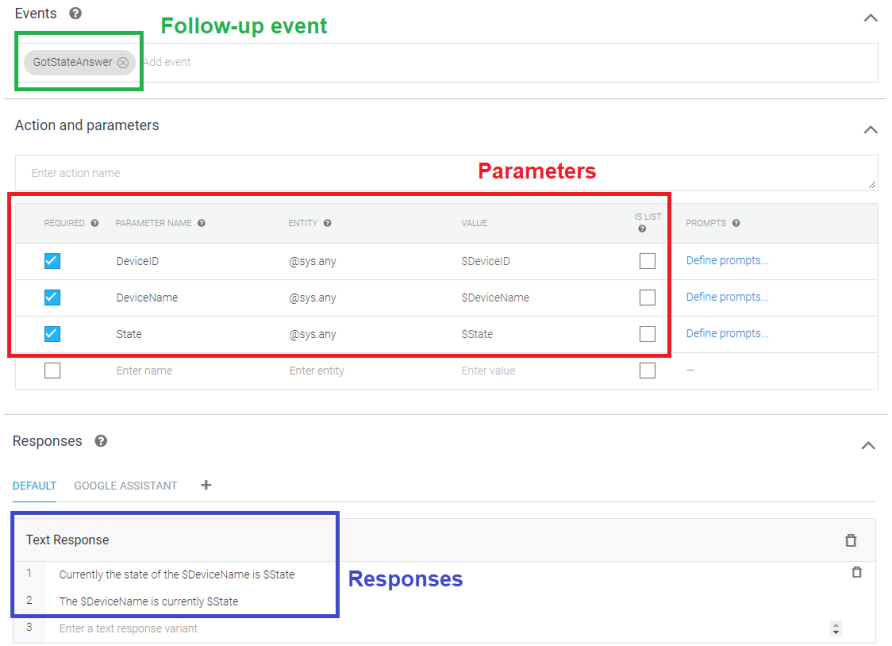


Figure 20: Example of an answer intent. Unlike the user's intent, this intent does not contain training phrases but rather response phrases. The HassTalk API responds to a user's intent with a structured message containing the parameters that has to be filled in in the answer intent. In order for DialogFlow to know which answer intent has to be used, the structured message also contains a follow-up event on which the action's intent is chosen.

5 HassTalk

The HassTalk API is responsible for taking the right action on a user's intent and responding with the corresponding information, which will be fetched by utilizing the FORTNIoT-Toolkit. As previously explained, DialogFlow will match the intent of the user and pass the intent's action and parameters to the HassTalk API. Whenever such a request is received by the HassTalk API, the first check is whether the API understands the given action and if not, respond with an empty answer. Whenever this happens, DialogFlow will automatically answer the user that it was not capable to determine their intent. Afterwards, the framework performs a check if the given device is a valid one. This can be applied to all the existing intents, as all of them contain the DeviceName parameter. Whenever the system did detect an unknown device, it will respond with an UnknownDevice event. This event contains the DeviceName that the user has given, along with a list of known devices. This is mostly done because of the fact that the tested systems contained a rather small amount of entities. Whenever there is a large list of known entities, this step has to be adjusted to react accordingly.
After these basic checks are performed, the chatbot will match the intent's action and will perform the matching algorithm in order to answer the chatbot with the right information. In this section, these algorithms will be further explained in detail.

5.1 Current state of a device

The response of this algorithm will notify the user about the state of a specific device. The user asks a question in the form of "What is the state of my lamp?", on which the DialogFlow Agent will match the given name of the device (in this case "lamp") with the respective device identifier (for example light.lamp). This is realized by adding one parameter to the training phrases with an entity type of *DeviceName* (Appendix E.1). An example of such an answer that results out of this algorithm can be: "The lamp is currently on". This algorithm responds whenever the intent's action is equal to *GetDeviceState* and takes as only required parameter a *DeviceName*.

The algorithm then uses the states route of the FORTNIoT-Toolkit (section 4.2.2) in order to determine the current state of the device. By adding the device identifier to the states route, HassTalk is able to fetch the state of that device specifically. Therefore, it is not necessary to iterate through a list of states.

After HassTalk successfully fetched the state of the device in question, it will respond with a *GotStateAnswer* follow-up event, which includes three parameters: the device identifier, device name and the current state. An example conversation regarding fetching the state of a device is given in figure 21.

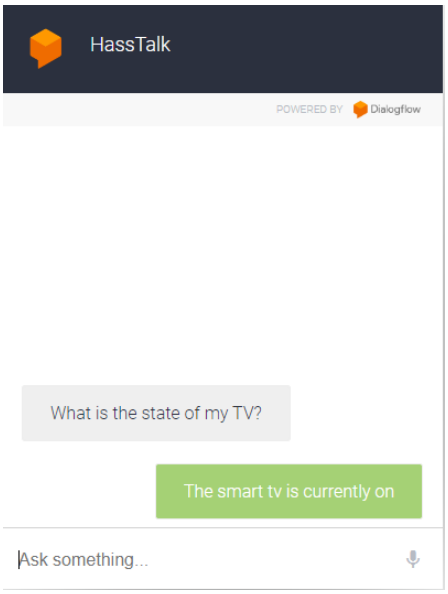


Figure 21: Example conversation in which the users asks what the state of a device currently is. The state is fetched from the FORTNIoT-Toolkit by utilizing the device identifier of the queried device.

5.2 Future behavior of a device

In order to enable users to fetch and understand the future behavior of a device, the software first has to create a future-object of a given device in order to query this future and thereby making it understandable for the end-user. This future-object makes it possible for HassTalk to query the future behavior of a specific device. More precisely, the future-class enables utilizing the following methods for a device's upcoming behavior:

- *getChangeAtTime(timestamp)*
  Fetch the change-object that is active at a certain point in the future. This method requires a timestamp as parameter, in order to determine the active change. A change object contains the state of the device at the given timestamp and the information of what changed in comparison with the previous state.

- *getAttributeValueAtTime(attribute_title, time)*
  Fetch the value of an attribute, at a given timestamp in the future. This method requires an attribute-title and a timestamp, in order to determine the attribute's value at that given time. For example, we could ask what the state (attribute_title) of a device would be at 8pm (time).

- *getNextOccurrence(attribute_title, attribute_value, time)*
  Get the next occurrence of a given value of an attribute, after a given timestamp in the future. This method requires a timestamp, attribute-title and -value in order to determine the next occurrence after the given timestamp, of that value. For example, we could ask for the time on which the state (attribute_title) of a device will be on (attribute_value) after 8pm (time).

- *getChangePhrases()*
  In order to make the changes of a device understandable to the end-user, the future-object contains this method. This method utilizes one of the Response Templates (appendix B) to form a textual representation of a change.

In order to create such a future-object for a device, the algorithm in figure 22 is used to fetch and combine the information from the FORTNIoT-Toolkit. The algorithm starts with the device identifier of which a future-object is required, and uses it to fetch the current- and upcoming states from the FORTNIoT-Toolkit. This is done by utilizing the states- and future-states route of the FORTNIoT-Toolkit (section 4.2.2). The resulting collection of states is then stored in a list and sorted chronologically by the *last_changed* attribute. Therefore, the algorithm now has a list of states that describe the upcoming behavior of the device and thereby knows what state a device will have at a given timestamp in the future. In order to know why these states occur, the algorithm fetches the upcoming rule executions from the FORTNIoT-Toolkit, utilizing the future-rule endpoint (section 4.2.3). The algorithm now has a raw representation of the upcoming states of a given device and a collection of upcoming rule executions.

The future-object is built upon a list of *Change-objects*, which includes the attributes of the device that changed at a given time, and the reason of that change. Listing 10 is a representation in pseudo-code of how a future-object is made out of a collection of upcoming states and a collection of upcoming rule executions. The algorithm iterates through the list of states and starts with determining the next state in the list (rule no. 6) in order to make a comparison (rule no. 7). The comparison between these two states is made by iterating through the list of device-specific attributes and comparing the values for these attributes, as said earlier each device as a state-attribute which needs to be considered in this procedure. Each attribute's value that is different for these two states, is stored in a list containing the attribute's title and the resulting value of that attribute in the next_state object. Now that the algorithm knows which attributes change between the two states, it has to determine the reason of that change.

In order to determine the reason of the change, the algorithm takes the identifier from the context-attribute of the next_state object (which identifies the situation which resulted in that state) and compares it with the identifiers of the action_contexts of the upcoming rule-executions. As mentioned in section 4.2.3, a rule-object contains trigger- and action context-attributes. Whenever the identifier of the rule-execution's action context is the same as the device's context identifier, this rule will be considered as the reason of the change. After this step, we can utilize the rule identifier to fetch the rule's description from the FORTNIoT-Toolkit, which is done by performing a GET-request to the RuleDescription-endpoint (see section 4.2.3). Thereby we have a textual representation of the rule's trigger and the action that will be taken on the device in question, which will be stored in a *RuleDescription*-object (rule no. 8).

Now that the algorithm has the required information regarding the changes between two states, it can create a *Change-object* (rule no. 11) and add it to the list of upcoming changes (rule no. 15). This list of changes is then used to create the Future-object that contains all the information regarding the upcoming behavior of a device. Using the result of this algorithm, the software is capable of determining the device's state at a given time, explaining the complete upcoming behavior of the device, get the next occurrence of a particular state for the device and determine the reason of a particular upcoming state.

```
Input: upcoming_states, rule_executions
Output: Future-object

upcoming_changes = []
foreach state in upcoming_states:
    next_state = next_upcoming_state(state)
    changed_props = determine_changed_properties(state, next_state)
    reason = determine_reason_of_state(new_state, rule_executions)

    // Make a change object from the information
    change_obj = build_change(state.entity_id,
                              changed_props,
                              next_state,
                              reason)
    upcoming_changes.push(change_obj)

return new Future(entity_id, friendly_name, upcoming_changes)
```

Listing 10: Pseudo-code of the algorithm that creates a Future-object out of a list of upcoming states for a device and a list of upcoming rule executions. The resulting future-object can be used to query the upcoming behavior of that device.

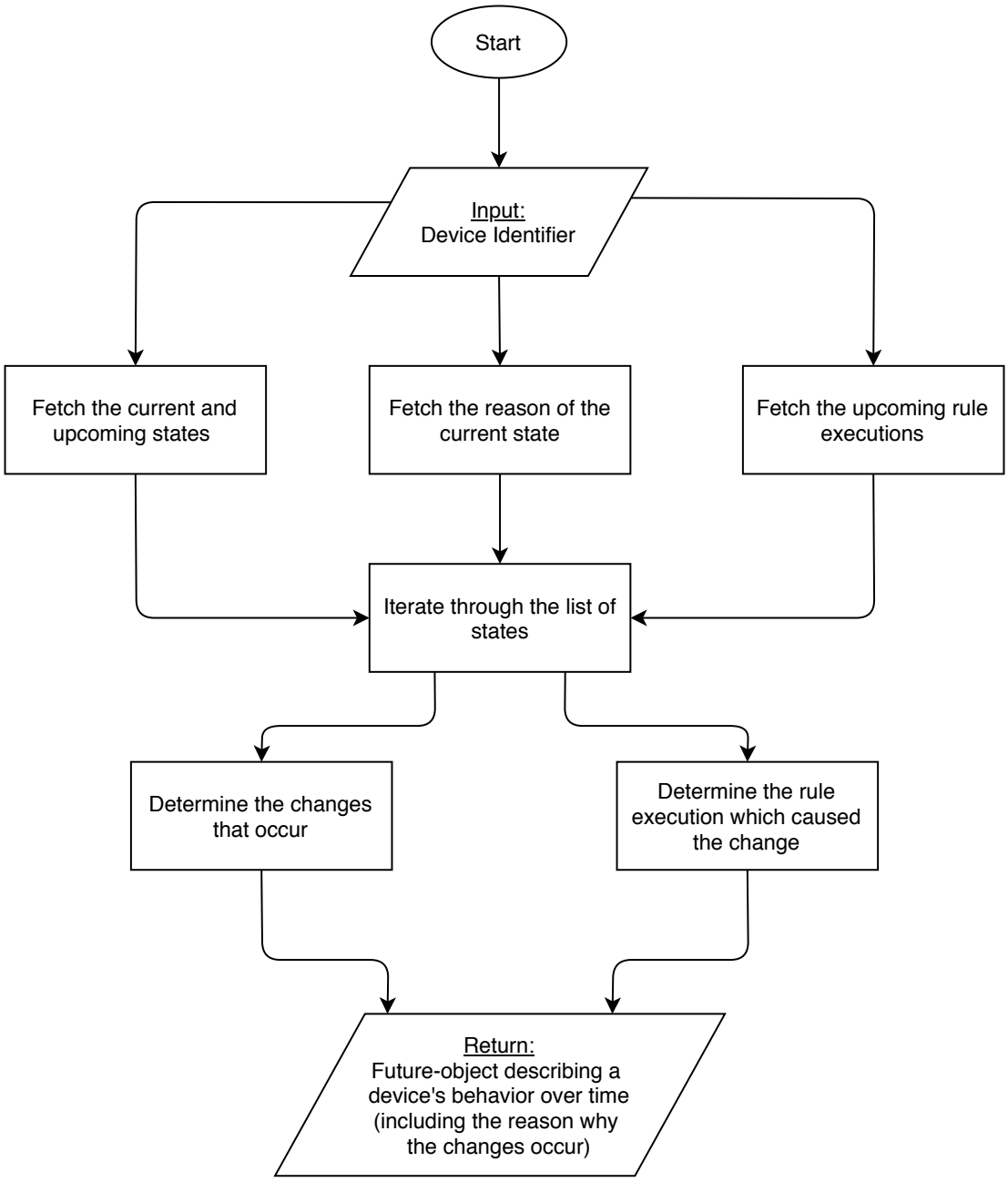


Figure 22: Algorithm to fetch a device's future behavior. Starting with a device identifier, the algorithm determines which changes will occur in the future and the reason why these changes will occur.

5.2.1 State at given time

HassTalk enables the user to ask about a device's state at a given time in the future. This is done by responding on the *What.Time* action intent, which requires the device identifier and a timestamp as parameters (eg. "What will be the state of the lamp at 8pm?"). First, the algorithm fetches the raw future as described in section 5.2. Once the raw future is fetched, the algorithm is able to call the *getAttributeValueAtTime* method using "state" as the attribute's title and the time given by the user. This method will return the value of the state-attribute at the given time. After performing these steps, the algorithm fires a WhatAtTimeAnswer follow-up event. This event contains a device identifier, friendly name of the device, a textual representation of the timestamp and the asked value of the attribute at the given time. The example used in this description, and its answer, is shown in figure 23.

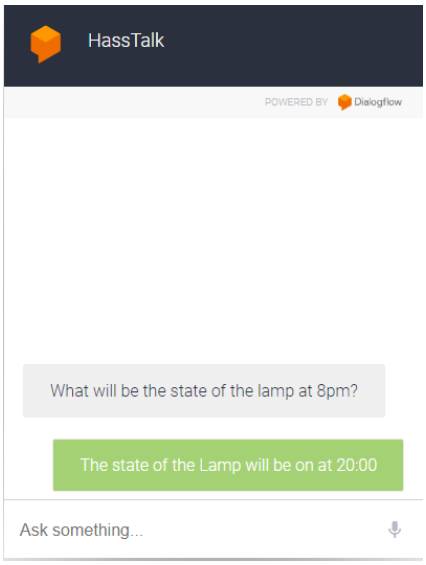


Figure 23: Asking for the state of a device at a specific time. In order to get the requested answer, the algorithm in figure 22 is utilized. The resulting future-object enables the software to fetch an attribute's value at a given time by utilizing the *getAttributeValueAtTime*-method

5.2.2 Complete future

The system gives the end-user the opportunity to get a description of the future behavior of a device in their system. Thereby, they can declare if the device behaves as expected in the upcoming time. Fetching the future behavior of a device is done by firing the *What.Future* action, which requires a device identifier as parameter. An example of such a question could be *"What will my lamp do in the future?"*, an example of the answer on this question is shown in figure 24. Upon receiving the *What.Future* action, the system will fetch the raw future-object, using the algorithm described above. Once the *future*-object is fetched, the algorithm utilizes the *getChangePhrases* method in order to form a textual representation of the upcoming changes of a device. Once the textual representations are created, the HassTalk API will fire the WhatAnswer follow-up event, which contains the friendly name of the device and the list of change-phrases.

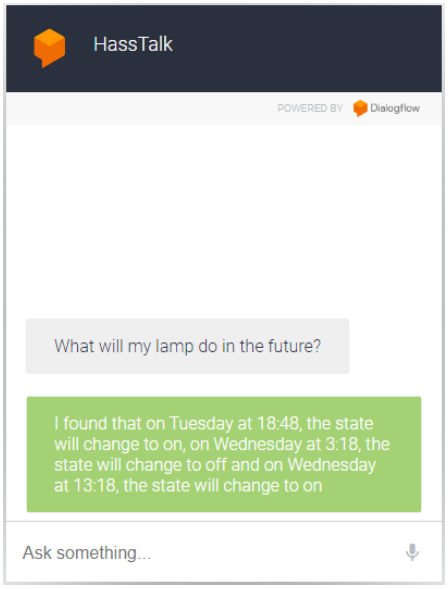


Figure 24: Asking for the future behavior of a device. The resulting object of the algorithm described in figure 22 is utilized in order to create an answer phrase representing the future behavior of the queried device.

5.2.3 Next occurrence of a device's state

In order to get the next occurrence of a particular state for a device, DialogFlow fires the *When.State* action event. This event contains a device identifier along with the AttributeValue which is queried (eg. when will the lamp turn off?). The algorithm starts with fetching the raw future of the device, using the algorithm described above. Then, it is able to utilize the *getNextOccurrence* method of the resulting *future*-object. Additionally, the algorithm fetches the *change*-object which will be active at the given timestamp, by using the *getChangeAtTime* method. From this change-object, we can fetch the textual representation of the reason why this will be the particular state. Once

this information is fetched, the HassTalk API will respond with a WhenAnswer follow-up event, which includes the device identifier, friendly name of the device, a textual representation of the given time and the asked state. When the algorithm was able to identify the rule which will cause the particular state, a description of the rule's trigger is attached. This is done because, when a user asks when something occurs, he may be interested in the timestamp of occurrence or the event on which the change will occur. The given example and its answer is shown in figure 25.

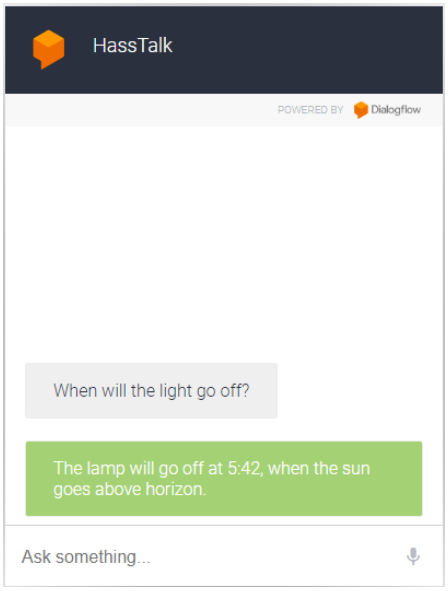


Figure 25: Asking for the next occurrence for the state a device. The object that resulted from the algorithm in figure 22 is utilized in order to call the *getNextOccurrence*-method. The resulting timestamp will be used in order to create an answer sentence.

5.2.4 Reason of an occurrence of a device's state

When the user used one of the algorithms above and thereby determined a future state of a particular device, it can be desired to know why that given state is active at that time. Therefore, HassTalk implements an algorithm in which the reason of a particular upcoming state is determined. When a user asks for the reason of an upcoming state of a device, DialogFlow will fire the *Why.Time* action event. This event includes a device identifier and a timestamp on which the state occurs (eg. Why is the lamp on at 8pm?). Note that this algorithm does not require a state, this is because the algorithm will determine the state at that given time itself. An additional step could be that the algorithm checks whether the queried state is the actual state at the given time, but this is not implemented in the current implementation.

Just as the algorithms described above, this algorithm will start with fetching the raw future-object of the given device. It will then utilize the *getAttributeValueAtTime* method, in order to determine the active state of the device at the given timestamp. Next, the algorithm will determine the cause of that state by calling the *getChangeAtTime* method. HassTalk will assume that upcoming states will always be caused by a rule, as a user's behavior can not be predicted without some sort of Artificial Intelligence. Once this information is fetched, the API will respond to DialogFlow with a WhyAnswerRule follow-up event. This event will include the friendly name of the device, the state that will be active and a textual representation of the rule's trigger and action on the device. There are two examples shown in figure 26, which utilizes the algorithm described above.

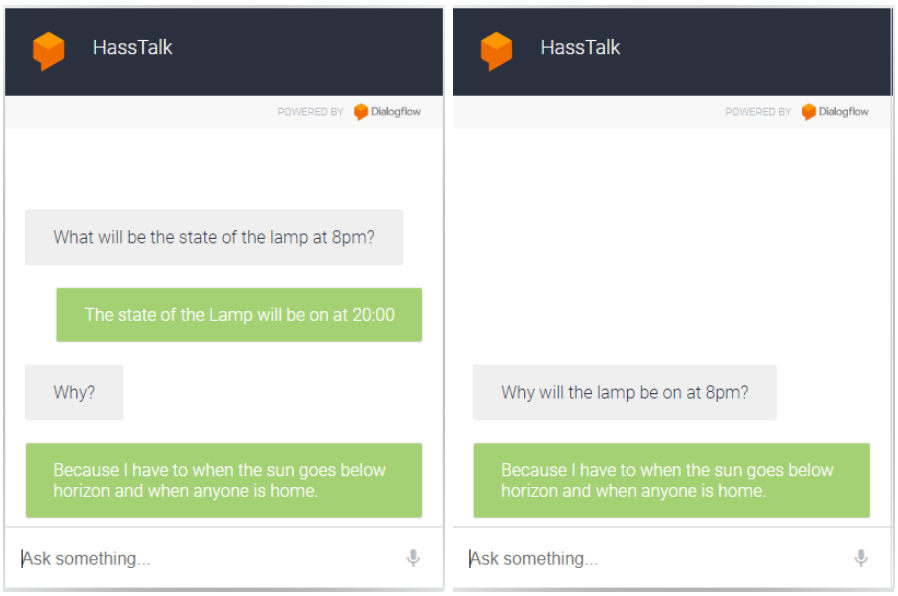


Figure 26: Asking for the reason of an upcoming state of a device. This can be either done by a follow-up question on the algorithm described in section 5.2.1 or by a standalone question.

5.3 Reason of a device's state

The test we conducted in section 3 pointed out that there are two types of questions utilized to fetch the reason of a device's state. We distinguish the why- and why not-formulation, which will fire another event (*why*- and *why.not*-event, respectively). This is done in order to determine if the queried state is actually valuable, otherwise this can indicate a malfunction in the device itself. Imagine, for example, that a user asks *"why is my lamp not on?"* or *"why is my lamp off?"*, and the system detects that the lamp should actually be on. This indicates that there may be something wrong with the device itself and therefore, HassTalk is not able to give an explanation about the current behavior of the device. The algorithm that determines the reason of a device's state is depicted in figure 28.

The first step is determining the current state of the queried device and determine if the question indicates a malfunction in the device itself. If the user has utilized the why-expression to pose the question, determining if the device is faulty is done by comparing the queried state with the actual state. If these are not the same, this could indicate a malfunction in the device itself (eg. User asks "Why is the lamp off?" but the system detects that the lamp is actually on). However, when the user utilized the why not-expression type and the queried state is the same as the actual state, it would indicate a malfunction in the device (eg. User asks "Why is the lamp not on" but the system detects that the lamp actually is on). If this test returns true, it will respond to DialogFlow with a WhyStateWrong follow-up event. This event includes the friendly name and identifier of the device, the queried state and the actual state. DialogFlow will then notify the user that the reason of the device's current behavior is because of a possible error occurring in the device itself. An example of such a situation is depicted in figure 27a, where both the why- and why-not formulation are utilized.

In order to formulate an anser to the user's question, the system first has to determine the cause of the current state. This is done by performing a GET-request to the why-endpoint of the FORTNIoT-Toolkit (section 4.2.5). As described earlier a device's state can be caused by a user, rule execution or unknown reason. Therefore, the software has three classes in which the particular information is stored: ReasonUser (contains the information shown in listing 6), ReasonRule (contains the information shown in listing 7) and ReasonUnknown (contains the information shown in listing 8). Now that the algorithm knows the reason of the current state, it has to check why it does not have another state (eg. we know why the lamp is on, but why is there no rule executed that would result in the lamp being off?).

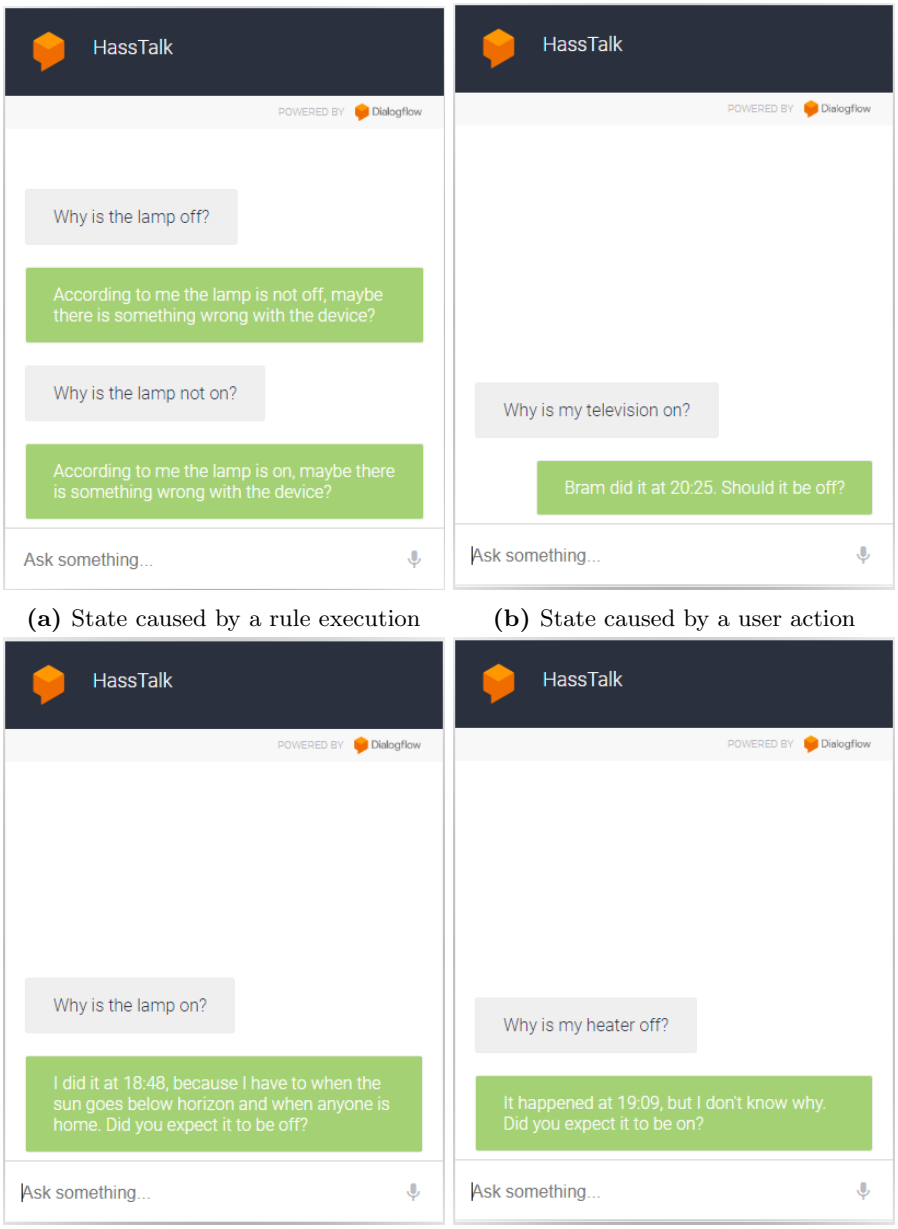
In order to know why the device does not have another state, the algorithm has to know which states are possible for that given device. This is done by querying the passed- and upcoming states of the device, utilizing the respective routes explained in section 4.2.2. By querying these routes, HassTalk is able to determine which states happened before and will happen in the future. The algorithm assumes that this will

result in a (almost) complete list of possible states. For each state in the fetched list (except for the current state), will be determined why that state is not currently active for the device. Therefore, the HassTalk API queries the why not-route (section 4.2.6) for every possible state that was found. This step results in a list of rules that will result in the device having another state as the current one. Along with these rules the information is attached, why the particular rule is not executed.

The algorithm now has the required information in order to determine which answer it has to give. Therefore it checks whether there is a rule that would result in another state as the current one, which would be executed in the current circumstances. When this is the case, the reason of the current state "denied" that rule. Take for example that there is a rule which turns on the light when the sun goes below horizon, but the user sees that the lamp is actually off at nighttime. The algorithm can then respond with the information that the rule was in fact executed, but is undone by another reason (eg. a user who has done it manually). When the system detects that there is no rule denied by another reason, it will determine which actor caused the current state: a user, a rule or an unknown reason. In each type of response there are several mutual parameters which are passed to DialogFlow: the current state and its inverse state, friendly name and identifier of the device and a textual representation of the time of occurrence. The inverse state is determined by iterating through the list of possible states for the device and take the first state which is not equal to the current state. The time of occurrence is added because this was considered as an interesting piece of information by the participants which participated in the test of section 3.
When the state is caused by a user's action, the username of the actor is attached and the algorithm will fire the *Reason_BecauseOfUser* follow-up event (example conversation shown in figure 27b). In the case that the current state is caused by a rule execution, a textual representation of the rule's trigger and identifier are attached and the HassTalk API will fire the *Reason_BecauseOfRule* follow-up event (example conversation shown in figure 27c). If the system was not able to determine the cause of the state, there is no additional information that could be added and the *Reason_BecauseOfUnknown* follow-up event will be fired (example conversation shown in figure 27d).

When there is a rule that would be executed in the current circumstances, which would result in another state, the response is slightly different as described above. On top of the information described above, a list of denied rules will be attached to the parameters of the event. In order for DialogFlow to add these rules to its explanation, other follow-up events are used. When a user caused the current state, the *Reason_RulesDeniedByUser* is fired. In case of a rule execution, the API fires the *Reason_RulesDeniedByRule* follow-up event. And at last, when the reason of the current state is unknown, the *Reason_RulesDeniedByUnknown* follow-up event is fired by the HassTalk API. An example of an answer would be in the form: "Normally the lamp would be on when the sun goes below horizon, but Bram has turned it on".

However, the algorithm above does not consider rules that are unexpectedly not executed. This may also be the reason why a device has a particular state. Take for example a situation in which a Roomba has to stop cleaning whenever there is somebody home, but the system was not able to detect the presence of the inhabitants. Therefore, the user would ask why the Roomba is still cleaning, on which the system would respond with the reason why it started cleaning (eg. "Because I have to when there is nobody home."). As this does not explain the reason why the Roomba did not stop, the response is incomplete. Because of this, the chatbot will ask the user if the current behavior is as expected. This is done by utilizing the inverse state, which is given in the follow-up event, and asking if this state was the expected state. In case of the Roomba, the question would be in the form of *"Did you expect it to be idle?"*. When the user answers this question in a positive manner (eg. "yes" or "I did"), the algorithm described in section 5.4 will be utilized in order to inform the user in a more complete manner.



(a) State caused by a rule execution **(b)** State caused by a user action

(c) State caused by a rule execution **(d)** State caused by an unknown reason

Figure 27: Example conversations in which the user asks for the reason of a particular state. In the examples shown there are no rules executed that would result in another state as the current one.

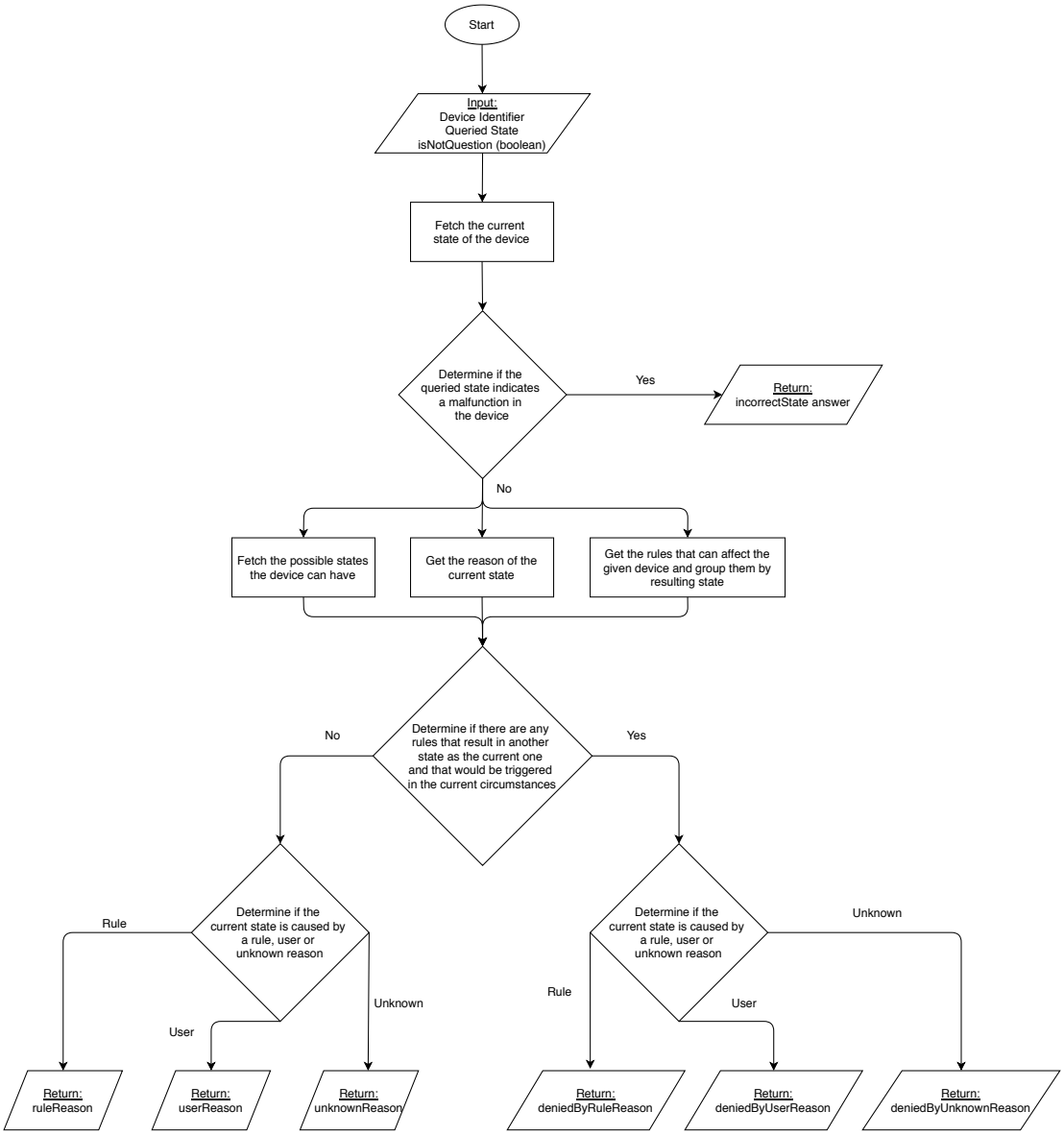


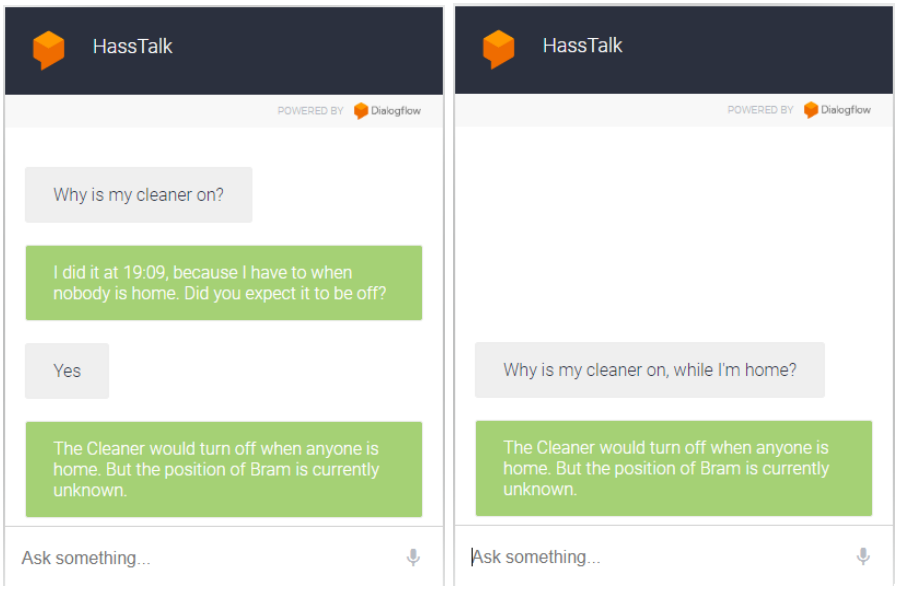
Figure 28: Algorithm to fetch the reason of a device's current state. The algorithm fetches the reason of the current state and additionally the reason why another state is currently not active. This is done by querying the existing rules that would result in the device having another state as the current one, and determining why these rules were (not) executed.

5.4 Reason why expectation did not happen

As described in section 5.3, the state of a device can not be as expected because of a rule which is not executed (eg. The user expects a lamp to be off, as there is a rule which would do that in the given circumstances. But the rule is never executed, so the lamp is actually on). This algorithm will determine the reason of the non-execution and is utilized whenever the *expectation* event is fired. This can be as a follow-up question of the answer given by the algorithm of section 5.3 or by a standalone question from DialogFlow itself. The event requires a device identifier and an expected state as the input parameters. The workflow of the algorithm is shown in figure 30.

The first step of the algorithm is fetching the rules which apply on the given device and would result in the expected state. This is done by executing a GET-request to the why not-route (section 4.2.6). For each of these rules, the algorithm makes a textual representation of their triggers. This will inform the user about the circumstances that are needed in order for the rule to be executed. Next, the algorithm takes the actual states of the triggering entities and checks whether they are fulfilled. When they are not, a textual representation of the actual state is made. These textual representations are formed according to the corresponding Response Template (appendix B).

Thereby we can inform the user about the circumstances that would trigger the rule and the actual circumstances, which caused the rule to not be executed. HassTalk will create an *Expectation_Answer* follow-up event in order to inform the user. This event includes the identifier, friendly name and state of the queried device, the textual representation of the rule's triggers and the textual representation of the states that caused the rule to not be executed. In figure 29 an example is given in which a user asks why their expectation did not happen. Figure 29a shows the initial implementation, in which a follow-up question was required on the answer of the why-question. After conducting the user test (section 6), a new intent was implemented which enables to express an expectation in a single question, which is depicted in figure 29b.



(a) Using the follow-up question on the why-question

(b) Using the standalone question (implemented after the user test (section 6))

Figure 29: Asking for the reason why an expectation did not happen. The current state of a device may not be as desired, because a particular rule is not executed in the given circumstances. From the test conducted in section 6, we concluded that the user formulates an expectation by mentioning the triggering device's states.

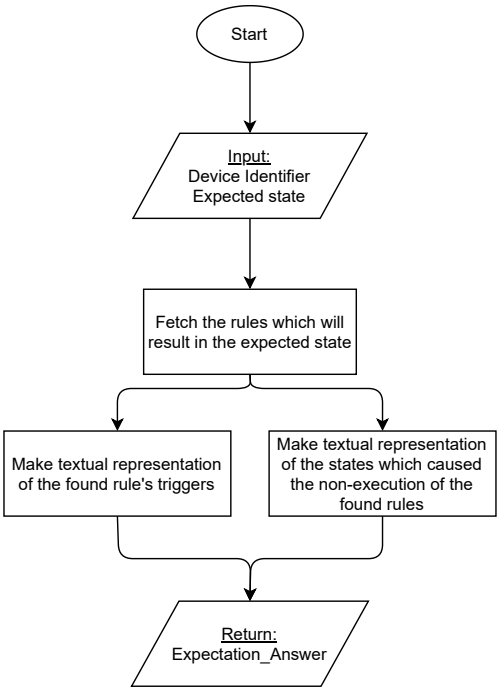


Figure 30: Algorithm to check the expectation of a user

5.5 Simulating behavior on a device's state change

Determining what would happen on a particular change can be done by simulating the systems behavior. For example, a user could ask *"What happens if the sun goes down?"* and would like to know which actions would be taken on the given change of state. Let's say there is a rule in the system which turns on the lamp whenever the sun goes below horizon, the system would then respond *"The lamp would turn on"*. Additionally, HassTalk enables it to compare it to the expected reality, in order to determine the differences. This is done by firing the *Simulation.FullDescription* action, which includes a device identifier, alternative state and optionally a time of change.

The workflow of the algorithm is shown in figure 32. The first step will be creating the alternative state for the given device. This is done by fetching the current state of that device from the state route of the FORTNIoT-Toolkit (section 4.2.2) and alternating the state attribute with the alternative value. If the user has given a particular time, it will be used to alternate the *last_changed* attribute of the state. If there is no time given, the current time is used. This alternative state will be used to upload it to the FORTNIoT-Toolkit, so that it can simulate the alternative behavior of the system.

As stated in section 4.2.4, an endpoint was created in the FORTNIoT-Toolkit in order for HassTalk to upload the created state and receive an alternative future. A POST-request to the simulations-endpoint will result in a list of states and rule executions. The algorithm will first group the received states by their device, in order to be able to create a future object for each one of those entities. For each of the devices, a future-object will be created with the algorithm described in section 5.2. This is done by passing the received list of states and rule-executions to the algorithm. This step will result in a list of future-objects containing the simulated behavior of the devices, which enables the algorithm to query the alternative future of the device in the same manner as described earlier. In order to avoid confusion we will call these the simulated futures.

Now the algorithm knows how the different devices will behave in the given circumstances. For each device in the list of simulated futures, the expected future in reality will be fetched in order to be able to compare it to the simulated future of the devices. This is done by iterating through the list of simulated futures, and for each device repeat the process described in section 5.2, this time using the upcoming states fetched from the states-future endpoint (section 4.2.2) and the upcoming rule executions fetched from the endpoint described in section 4.2.3. After this step, a list is retrieved that contains future-objects of the expected reality for each one of the devices. To ensure a clear explanation, these objects will be called real futures.

The algorithm now has a list containing the simulated- and real futures of each one of the devices and can now compare them with each other. As stated in section 5.2, a future-object contains a list of upcoming changes. For each change in the simulated future of a device, the timestamp is taken in order to determine which attributes differ in the real future. The *getAttributeValueAtTime*-method is used on the real future object for each attribute of the device, with the timestamp of the change in the simulated future as parameter. Thereby, the algorithm knows if the simulated future differs from reality at the given timestamp. If the simulated value is different from the real value, a difference is detected and the algorithm will continue by creating a FutureDifference-instance. This class requires the following parameters:

- Device name: the friendly name of the device
- Property: the title of the property that is different in the simulation
- Time: the time on which the change occurs in the simulation
- Simulated value: the value of the property in the simulation at the given time
- Real value: the expected real value of the property at the given time
- Next occurrence: the next occurrence of the simulated value in the real future
- Reason_simulated: the reason why the simulated value occurs
- Reason_real: the reason why the real value occurs

The algorithm already knows the device name, the title of the property that is different in the simulation and reality, the timestamp on which the change occurs in the simulation, the simulated value of that property and the value of that property in reality. In order to be able to create a FutureDifference-instance, the algorithm calls the *getNextOccurrence* method of the real future. Thereby, the algorithm knows when the simulated change will occur in reality (if it does). Let's take the example described above where the user asked what would happen when the sun sets. HassTalk is able to reformulate the answer like: *"The lamp would turn on, which would normally happen at 21:46"* (as the sun normally would go below horizon at that given time). To fill in the reason_real attribute of the FutureDifference-instance, the *getChangeAtTime* method is called on the real future. Thereby the algorithm knows which change-object will be active at the given time and the reason of that change is passed to the constructor of the FutureDifference-class.

These steps are repeated for each change occurring in a device's simulated future and that for each device in the result of the call to the simulations-route of the FORTNIoT-Toolkit. After this, the algorithm has a list of differences, in the form of FutureDifference-instances, between the simulated future and the real future. It then utilizes these instances to form a textual representation of the detected differences. In order to formulate this, a Response Template is utilized (appendix B). The algorithm fires the *SimulationAnswer* follow-up event, containing the textual representations of the differences. In figure 32 is an example shown in which the user simulates the actions taken when a person leaves the house. Note that in the example shown in the figure, there is no comparison made with the expected behavior of the device in reality. Despite the fact that this information is available to represent, this would result in a rather large block of text and is therefore not considered in the current implementation.

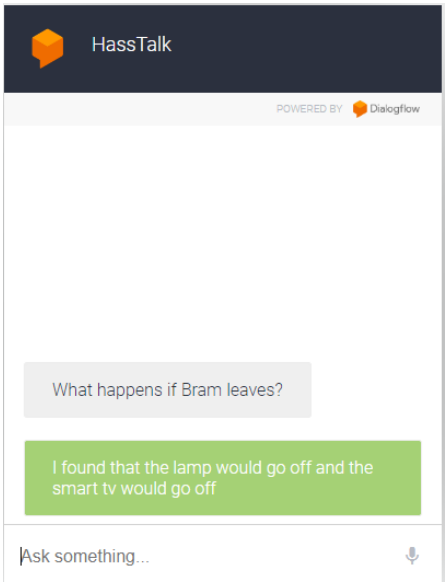


Figure 31: Simulating the behavior on a state change

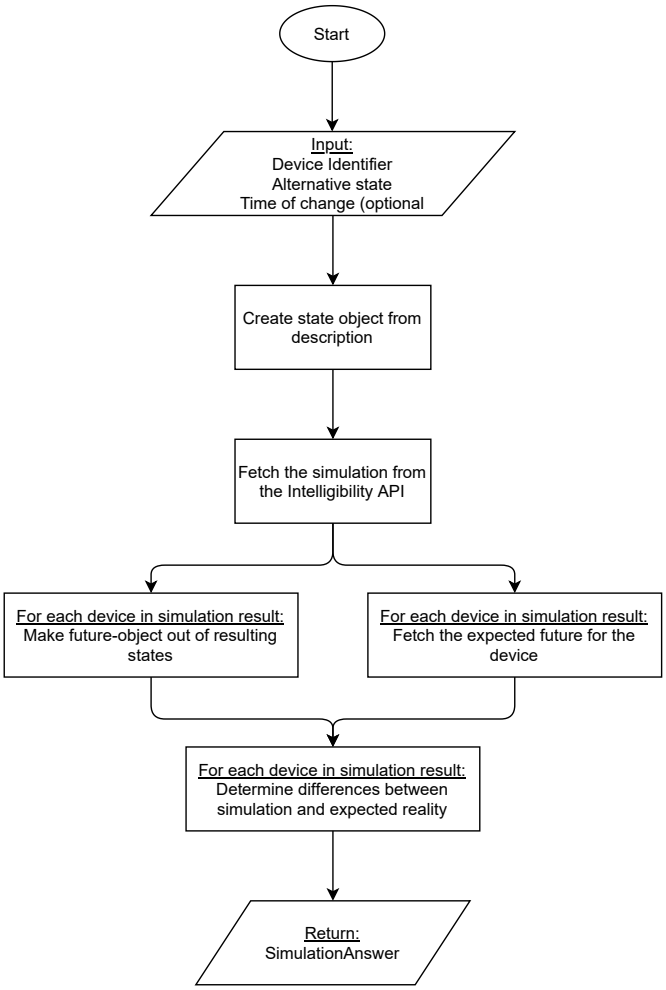


Figure 32: Algorithm to determine behavior on a particular change

6 HassTalk user study

The test consisted of two simulated systems which depicted two different household situations. A visual representation of the current situation was given (eg. given in figure 33 and figure 34) to the participants, along with the chatbot. In total 9 questions were asked on which the participant had to answer, using the chatbot. These questions were distributed along two different scenarios, in order to create a variety of occurring issues and keeping the test more interesting for the participants.
In this test, we will determine the completeness of the implemented algorithms, and if they are suited to solve the given issues. Moreover, we will identify the types of questions that are used and if the list of implemented questions is complete in order for the users to be able to formulate their desires, without further explanation given beforehand.
Next, it is tested whether the given explanations and pieces of information suffice for the user in order for them to understand the system's current and future behavior. At last, we will evaluate the over- or undertrust a participant has.

6.1 Participants

Eight participants were recruited to take part of the user test. The group of participants consists of 2 women and 6 men (4 aged 21-29, 2 aged 30-39 and 2 aged 50-59) with varying backgrounds (e.g. students, education, manufacturing, business and marketing). None of the participants had background knowledge in the field of Internet of Things or Smart Home systems.

6.2 Procedure

We opted for a supervised test in order to ensure a fluent and smooth experience for the participants. Due to the COVID-19 pandemic [4], we opted for remote video calls. As the test took 30 minutes on average, we tried to eliminate that the user would be frustrated or bored, which would result in a lower grade of motivation and participation.
The procedure of the test commenced with a briefing about the study's purpose and our data policy, based on the GDPR legislation[5]. After this step the participants received a short introduction about how the test will be executed and what the participants were expected to achieve. As stated above, there are two simulated Smart Home systems where there are some occurring issues or problems which the users had to solve using the chatbot. We gave the participants only a visual representation of the current situations, in order to avoid giving to much information or tips. This is done because it represents the intended end use-case where the chatbot is utilized in a household, where the only information available is the information that is visible for the user.

[4] https://www.who.int/emergencies/diseases/novel-coronavirus-2019
[5] https://ec.europa.eu/info/law/law-topic/data-protection/eu-data-protection-rules_en

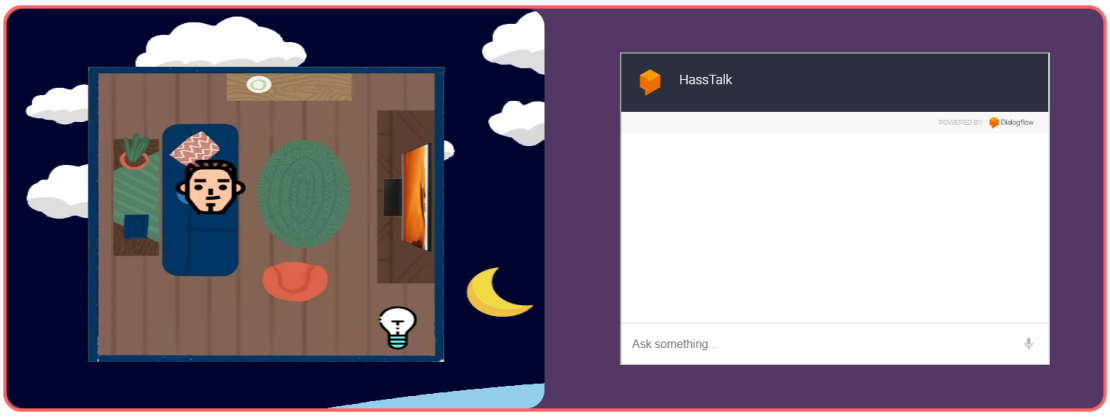


Figure 33: First simulated situation of the user test. This situation contains five entities: a person, a lamp, television, TV-guide and the sun. The lamp in this situation is broken, as the system thinks it's on. The television is currently playing "Friends" as the person manually turned it on.

The simulated situations

Figure 33 depicts the first situation with the chatbot aside of it. In this situation we see that there are five entities to take in consideration. The first one is the person which is at home and is named Bram. Secondly, there is a lamp in the system which is not burning and a television which is currently on and playing. Additionally, the system keeps track of the TV-guide, which states that currently the TV-show "Friends" is playing on the television. The last entity that is taken into consideration is the sun, which is currently below horizon. The second situation is visualized in figure 34 and includes four entities in the system. The first one is the thermometer, which is below 10 degrees celsius (considered cold). Like in the first situation, there is a person who is at home and is named Bram. The second situation also includes an heater which is currently off and a Roomba-cleaner which is currently cleaning. These images were the only pieces of information delivered to the participants in order to avoid giving them to much information and therefore making the solution to evident.
Each of those systems contained a list of rules which were not communicated to the participant. This is done because the participant had to determine the system's behavior by themselves, without any knowledge of the process that resulted in the current state. The first situation contained the following four rules in order to automate the system's behavior:

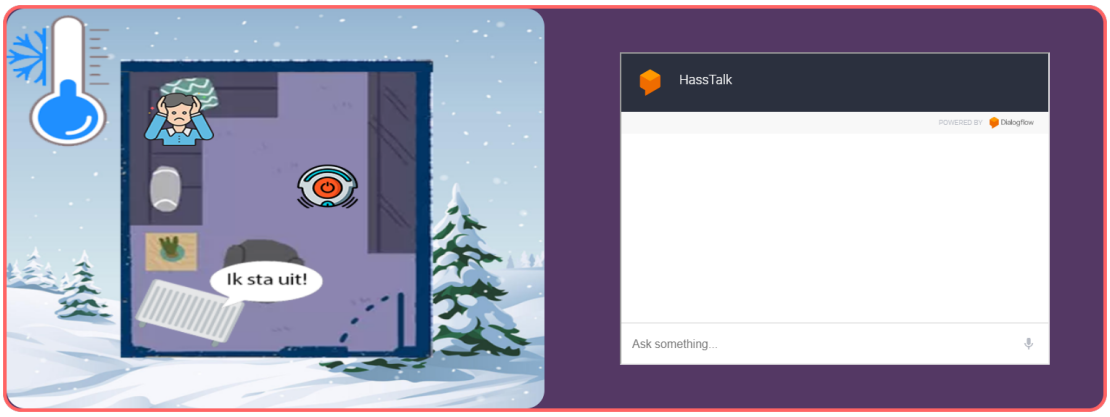


Figure 34: Second simulated situation of the user test. This situation contains four entities: a thermometer (currently reporting 4 degrees celsius), a person (currently reported as not home, but is actually at home), a heater (which is off) and a Roomba cleaner (which is currently cleaning). The system failed to determine that the person is actually home, therefore the heater is not turned on and the Roomba is still cleaning.

- **[rule.sun_set]:** WHEN the sun goes below horizon AND there is anyone home
  → *Turn on the lamp*
- **[rule.nobody_home]:** WHEN there is nobody home
  → *Turn off the lamp AND turn off the television*
- **[rule.sun_rise]:** WHEN the sun goes above horizon
  → *Turn off the lamp*
- **[rule.news_starts]:** WHEN the news starts AND anyone is home
  → *Turn on the television*

Considering the given states of the included entities, the only rule that should be triggered is the **[rule.sun_set]** rule. The second rule, **[rule.nobody_home]** is currently not triggered, as Bram is at home. The rule **[rule.sun_rise]** is not triggered because the sun is currently not above, but below horizon. And the last rule, **[rule.news_starts]** is not triggered, because currently not the news, but "Friends" is scheduled in the TV-guide.

The second situation included the following rules in order to automate the system's behavior:

- **[rule.heater_on]:** WHEN there is anyone home AND the outside temperature goes below 16° Celsius
  → *Turn on the heater*

- **[rule.nobody_home]:** WHEN there is nobody home
  → *Turn on the cleaner AND turn off the heater*

- **[rule.anyone_home]:** WHEN there is anyone home
  → *Turn off the cleaner*

Taken the current situation into account, the rule **[rule.heater_on]** should be triggered, as Bram is home and it is below 10° Celsius outside. The second rule, **[rule.nobody_home]** should not be triggered, because Bram is currently home. And therefore, the rule **[rule.anyone_home]** should be triggered.
In order to simulate upcoming events, there are several upcoming states implemented in order to trigger some of the rules above in the future. For the first situation, the TV-guide will schedule the news at 5 minutes from the start-up of the test situation. The sun in the first situation will go above horizon, 6 hours from the start of the simulation. The second situation has only one upcoming state, which states that Bram comes home 2 hours after the start of the simulation. This may be confusing, as the person is currently already home. Planning the future state is done because currently the system was not able to detect the presence of any person at home, but in order to trigger the rule **[rule.anyone_home]** the future state of the person was planned.

The information the participant has to determine
In order to test the functionalities of the system, several issues were created in the system. We tried to cover all the aspects and information types given in the research of Lim et al. [38]. Considering the first situation, the participant had to solve the following issues using the chatbot:

1. Which rule caused the current state?
   There is a lamp in the system, which is considered to be "on" by the system. The participant has to determine the description of the rule which caused that state.

2. What caused the faulty state report?
   As stated before, the lamp is not burning at all, but the system considers it to be on. The participant has to determine what caused the faulty report of the state. In this case, the error lays within the device itself, as the light bulb itself is broken.

3. What caused the unexpected behavior?
   The television in the system is on, which is not expected by the user. The participant has to determine what caused the television to be on. In this case, this is caused by a manual action by one of the inhabitants.

4. What happens on the given change?
   The participant has to determine the actions that are taken in the system on a state change. In this case, it has to be determined what will happen if the inhabitant (Bram) leaves the house.

5. When will something happen?
   The participant has to determine when the sun will go below horizon. This will happen 6 hours after the simulation has started.

6. Does the rule behave as expected?
   *The participant gets a description of an existing rule in the system (in this case **[rule.sun_rise]**). It is asked to determine if the rule will behave as expected. As the rule is implemented in a correct manner, the participant has to declare that it behaves as expected.*

After these issues are solved, the second situation is initialized and the participant has to solve the following occurring issues:

7. Why is a rule not executed?
   The cleaner in the system is currently running, but the rules state that this is not expected when someone is home. Therefore, the participant has to determine what caused the cleaner to be running. In this case, the system was not able to detect the presence of the inhabitant and therefore considered that there is nobody home.

8. Why will something happen in the future?
   *The participant is notified that the heater will be on later that day. He has to determine why this state is occurring at the given time. In this case it is because of the execution of the rule **[rule.heater_on]** (hence the upcoming state described earlier)*

9. Why is something not happening?
   *According to the given situation, the heater was expected to be on (by rule **[rule.heater_on]**), but this is not the case. The user has to determine why this did not happen. As with the first question in this scenario, this is caused because the system was not able to detect the presence of the inhabitants.*

6.3 Measurement and analysis

During the test the following measurements were made:

- Correctness
  Fraction of how many of the participants were able to get the right answer on the question
- Amount of posed questions to the chatbot
  Amount of questions to the chatbot that were posed, in order for the participant to solve the issue
- Amount of unrecognized questions used
  How many questions were not recognized by the chatbot in the process of solving the issue and which ones.
- Certainty of the answers
  How certain were the participants about their given answer (using a likert scale)
- Easiness of finding the answers
  How easy was it for the participants to solve the given issue? (using a likert scale)
- Completion time
  What was the time needed in order to solve a particular issue? We will refer to the Average Completion Time as ACT.
- Interpretation time
  How long did it take the participants to interpret the given answers? This is measured by taking the difference in time of the first question and the start of giving an answer. We will refer to the Average Interpretation Time as AIT.

With this data, we can determine if the implemented algorithms are complete in a way that the users are able to solve basic problems of IoT-systems. The total amount of questions and the amount of unrecognized questions will give us an indication wether the users are able to formulate their desires, without a given explanation. The correctness of the answers is measured in order to ascertain that the participants were able to understand the current and upcoming behavior of their system. To determine the over- or undertrust of the users in the chatbot, we measured the certainty and easiness they experienced in solving the given issues and compare them to whether the correct answer was given or not.

6.4 Results

The results of the measurements about the correctness and amount of questions that were posed to the chatbot is depicted in table 3. These results show that for almost every question, a success rate of 100% is achieved. For the fourth, seventh and ninth question, one of the participants was not able to find the correct answer. Each time, this was because the chatbot responded on one of their questions with the information that the device they had given, was not recognized. This answer was given because the chatbot misinterpreted the posed question, and returned the default response: a message that the given device could not be detected, followed by a list of known devices. This however, is not considered as an unknown input (the chatbot would then return an answer in the form of *"I could not understand that"*), as the question was matched to an action. However, the wrong action was matched and therefore, the participants thought the system was broken and thereby, not usable to solve the issue in question.
In theory, the first 6 issues and the 8th one, could have been solved by posing one single question (as shown in table 3). However, the results show us that in each one of those scenarios, there are more questions posed on average. When we take a look at the average amount of unknown questions (questions that are not recognized by the chatbot), we can determine a correlation between the high averages of the ACT- and AIT-values. For the sixth case specifically, the users wanted to know for sure that the rule behaves as expected. Which resulted in them asking for more confirmation by posing more questions, this can indicate an undertrust in this scenario. In case of the eight scenario, we cannot conclude that there is a correlation between the high average of posed questions and the average of unknown questions. Looking at the raw given answers, we saw that this was because the participants tend to ask when the heater will turn on. The answer given by the chatbot did not meet their needs, and therefore they were forced to pose another question. However, this did not bring any problems in determining the correct answer. In contrast to the other issues, the seventh and ninth issues were solvable by posing a minimum of 2 questions (1 why-question and 1 follow-up question). In the ninth scenario, we do not see any unexpected results, but in the seventh one, we again see a high average of amount of posed questions. When observing the participants solving the seventh issue, I remarked that the great majority of them were uncertain of the answer, and therefore asked another question as a confirmation for their answer. As stated before, the rule [**rule.anyone_home**] was not triggered because the system was not able to determine the position of the inhabitant Bram. When the chatbot notified the participants about this, they tend to ask if Bram's phone was connected or why the system was not able to detect the presence.

Task	Correctness	Avg. # of questions	Avg. # of unknown questions	Min. amount of questions
1	100%	1.25	0	1
2	100%	1.375	0.4	1
3	100%	1.25	0.2	1
4	87.5%	1.875	0.6	1
5	100%	2.375	1.2	1
6	100%	3	1.0	1
7	87.5%	3	1.2	2
8	100%	2.25	0.4	1
9	87.5%	2.375	0.8	2

Table 3: Objective measurements conducted from the user test. It includes the percentage of the participants that was able to solve the issue, the average amount of questions that was needed in order to solve the issue, the average amount of unrecognized formulations and the minimum amount of questions needed to solve the issue.

In table 4 are the time measures shown which are taken during the test. In the first scenario we noticed that the average completion time (ACT) is rather high (in comparison with the third scenario, which both require 1 question in order to solve the issue), however, the average interpretation time (AIT) has a normal value. The high ACT value can thereby be considered as a learning time in order to get familiar with the test environment.
In the second case, however we see that both the ACT and AIT are rather high, in comparison with the third situation. Looking at the remarks this is due to confusion, as this situation included a broken device. Thereby, HassTalk was not able to determine the cause of the actual state of the lamp. One of the participants commented that it would be useful to give an estimation of the problem, even if it is not necessarily correct. In the case of the second scenario, this could be in the form of *"This can indicate that the lamp itself is broken, please check it!"*. Table 5 shows us that both the average certainty and average easiness for the second scenario are rather low. This can be dedicated to the confusion of the broken device and the incapability of the chatbot to serve a proper response. The fourth scenario demanded both a high ACT and AIT, in comparison with the third situation, this was due to the fact that the participants experienced difficulties in formulating their question to the chatbot. As shown in table 3, one of the participants was not able to solve this issue due to this difficulty in formulation. Therefore, the average easiness score for the fourth scenario, shown in table 5, is rather low. As stated before, the participants were not sure about their answer in the sixth scenario and therefore asked for more confirmation. This accounts the rather high values for the ACT and AIT, and the lower easiness experienced by solving this issue. The results in table 4 and table 5 show us that solving the issue in scenario 7, was considered rather difficult. In this scenario, the system was not able to determine the location of the inhabitant. The participants stated that it would be helpful to give further explanation about this error. An example that was given, is informing the user

that the system was not able to communicate with the person's smartphone. This is also visible in the questions that are posed by the participants, as a majority of them asked if the phone of Bram was on and reachable. The system however, can not respond to this question, which accounts the lower value for the easiness score. As stated previously, issue 8 was a more difficult scenario and therefore the values shown in the measurements are considered to be normal. This is also the case for the ninth scenario.

Task	**Avg. completion time (min)**	**Avg. interpretation time (min)**
1	2 min. 45 s.	1 min.
2	2 min. 15 s.	1 min. 30 s.
3	1 min. 22 s.	0 min. 15 s.
4	1 min. 38 s.	1 min. 30 s.
5	2 min. 45 s.	0 min. 53 s.
6	4 min. 30 s.	2 min. 45 s.
7	4 min.	2 min.
8	3 min. 8 s.	1 min. 15 s.
9	1 min. 45 s.	1 min. 45 s.

Table 4: Time measurements of the user test. The completion time depicts the total time needed to interpret and solve the issue. The interpretation time describes the time needed to interpret the answer that was given by the chatbot.

Task	**Avg. certainty**	**Avg. easiness (min)**
1	4.875	4.75
2	3.75	3.75
3	4.875	4.75
4	4.375	3.75
5	4.875	4.0
6	4.25	3.875
7	4.375	3.25
8	4.75	3.875
9	4.625	3.875

Table 5: Subjective measurements conducted from the user test. After each step, the user had to determine how certain they were about their answer and how easy they found it to solve it. This is done by using a likert scale.

In order to determine the subjective satisfaction of the participants, we asked them to fill in a survey built around SUS [24]. The results of this questionnaire are depicted in figure 35. In order to calculate the SUS-score, the following steps were taken:

- For odd items: subtract one from the user response
- For even-numbered items: subtract the user responses from 5
- This scales all values from 0 to 4
- Add up the converted responses for each user and multiply that total by 2.5. This converts the range of possible values from 0 to 100 instead of from 0 to 40.

Applying these steps gave us a SUS-score of 72.5, which is considered above average (in comparison with 500 studies) [47] and gives us a percentile of 70%. When considering the graph shown in figure 36, we can conclude that our chatbot gets a B-rating on the scale, which is considered as a positive result.

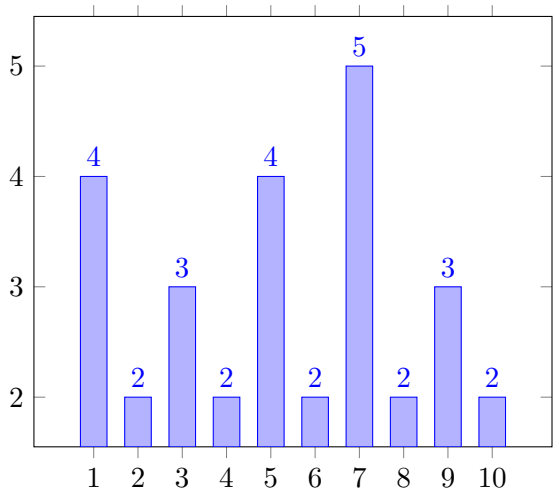


Figure 35: The results that came out of the System Usability Scale form. These results are used in order to determine the SUS-score of our application, which is 72,5 and gave a B-rating (good). The questions posed are listed below, along with the given scores of the participants individually.

1. I would use the chatbot frequently	4, 5, 5, 4, 5, 4, 4, 1
2. I thought the system was complex	3, 1, 1, 2, 4, 1, 3, 1
3. I thought the system was easy to use	3, 4, 2, 4, 4, 4, 3, 3
4. I would need technical help in order to use the chatbot	2, 1, 2, 2, 1, 2, 2, 2
5. I thought the functionalities are well implemented	4, 4, 4, 4, 4, 5, 3, 1
6. I thought that the chatbot was inconsistent	4, 2, 1, 3, 1, 1, 4, 2
7. I thought it was easy to learn to use the chatbot	5, 5, 5, 4, 5, 5, 5, 4
8. I thought the system was cumbersome to use	2, 1, 2, 2, 1, 1, 3, 4
9. I felt confident using the system	2, 4, 2, 3, 2, 5, 4, 3
10. I needed to learn a lot, before I could use the chatbot	2, 1, 2, 2, 1, 1, 2, 4

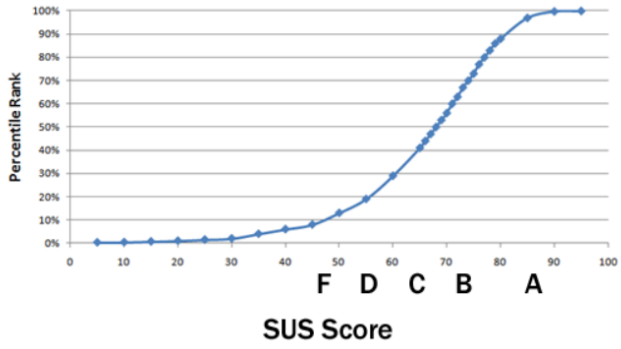


Figure 36: Percentile ranks associate with SUS scores and letter grades

6.5 Discussion and conclusions

The overall impression of the results that came out of the user test, is rather positive. As seen in table 3 the majority of the participants were able to solve all the given issues. In order to determine how HassTalk scores on the subjective measures of PARADISE [22], a System Usability Scale is calculated using the subjective responses on the survey after the test. This calculation gave us a SUS-score of 72,5 and a B-rating, which we can consider to be a positive result. Therefore, we can say that the chatbot could be a great addition to the life of smart home users. The majority of the participants stated that they would use the chatbot frequently and thought that it was an easy to use concept. One of the participants even thought he was doing the test incorrectly, as he thought it was going far to easy. Out of the information evaluation (section 3), we concluded that giving the time along with the reason of a state is required when answering the why-question. This conclusion once more got confirmed during this test, as 3 of the participants literally stated that the time was their indication that their answer was correct (in the first, seventh and ninth scenario).

However there were some remarks regarding the general feeling the participants experienced during the test. When the user expected something to happen, which did not (like in scenario 2 and 7), they stated that the explanation was rather confusing. One of the participants stated quite literally that an indication of what could be wrong, would be helpful. Initially, we aimed for a short and compact formulation of the chatbot's answer. But it is clear that the users rather have too much information, than less. In order to check why the user's expectation did not happen, they had to ask first why it happened. The chatbot would then give the reason of the occurrence and ask if the user expected to be otherwise. This means that it required to pose 2 questions to verify why an expectation did not happen. When taking a look at the posed questions, a majority of the participants posed their question in a *"Why ... while ..."* formulation. An example of such a formulation is given by one of the participants: *"Why is the cleaner on, while I'm home"*. With this information, we can conclude that this formulation indicates that the user want to check their expectation. Thereby, we can reduce the amount of questions that have to be asked in order to fulfill the user's need.

In general, the participants experienced a feeling of insecurity when answering the posed questions. This can be accounted to the errors that exist in the system and chatbot. As stated before, the chatbot has 2 ways of error handling. The first one is applied when the user's intent is not matched to an existing one, in this case the chatbot replies with an answer that asks the user to formulate their question in another way. This gave the participants the impression that the questions have to match a certain pattern, therefore we can conclude that the set of training phrases has to be expanded. In order to define the formulations of the end-user, a larger test have to be conducted in order to get representative results. The second way of error handling is applied whenever the intent is matched, but the DeviceName parameter was not recognized. This will result in the chatbot responding with the notification that the given device was not recognized,

followed by a list of known devices. This gave the participants the impression that the given issue could not be solved by the chatbot, which indicates undertrust. A better error handling would therefore be required, so that the user knows better what went wrong.

When we analyzed the posed formulations that were not matched to an intent, two groups of questions could be identified: Yes/No questions and large/complex questions. The chatbot does not implement yes/no questions at the time. One of the posed questions by the participants was *"Will my lamp go off, when the sun goes above horizon"*, as the chatbot was not able to recognize the used formulation, a feeling of uncertainty was experienced by the participant. Implementing these questions would not require any new algorithms, but does require some refactoring in order to match the user's intent to the right algorithm. More complex and large questions were not recognized by the chatbot. Therefore a hint was given to the participants to pose their question in a more simple and general way. Once the hint was given, the users were able to formulate their question and therefore receive the required answer.

In general, a positive result was conducted during this test. Basic problems in Smart Home systems were solvable by using the chatbot and the majority of the participants were able to do that. Subjectively speaking, the participants were satisfied by the chatbot and were tended to use it in the future, when it would be commercially available. However, there are some changes required in order to further apply the chatbot in Smart Homes. The participants had a general feeling of insecurity because of the undertrust in the system. While the participants were able to find the correct answer, they did not trust the chatbot enough to consider their answer to be completely correct. In order to make the chatbot more complete a larger test has to be conducted in order to identify more formulations regarding the questions posed to the chatbot. Improving the certainty of the user could be done by doing more research about the pieces of information that are required to solve a given problem. Were we assumed that a short and compact form of answering would be preferred, the participants would prefer a more wider and complete explanation.

7 Conclusion and future work

In this research we determined the possibilities of chatbots in order to make Smart Home systems intelligible. The user can use natural language in order to formulate their desires or needs. In order to be as complete as possible the pieces of information found in the research of Lim et al. [38] are implemented in the chatbot. Thereby, it is possible to ask questions about what is happening, why it is (not) happening, when it is happening and what will happen when a particular change occurs. In order to define the interesting pieces of information, a test was conducted which gave an indication of the question formulations that are used and the pieces of information that are required to deliver to the end user. From this test we concluded that when the user asks for the reason for a particular occurrence, not only the why-explanation is required but also a time indication on which the change occurs. Moreover, we concluded that a timestamp is desired in every explanation type as this gives the user a more complete description of the occurring behavior.

Taking the results of the first study, HassTalk is created in order to add intelligibility to the world of Smart Homes. HassTalk is built upon the FORTNIoT-Toolkit of Sven Coppers, which itself is built on a Home Assistant appliance. Posing questions to HassTalk will trigger querying the FORTNIoT-Toolkit in order to fetch the required pieces of information that are required to answer in a correct manner to the user's questions. In order to achieve the correct information, some additional endpoints had to be created in the FORTNIoT-Toolkit. Once the required API points were implemented, the DialogFlow framework was trained by giving the output of the first test as the training phrases of the input question. DialogFlow will thereby be able to match a user's question to a specific intent. Each intent will have an action, which will be used by the HassTalk API in order to determine the right algorithm to formulate the required answer. Each of those actions represent one of the information pieces proposed by Lim et al. [38]. HassTalk hereby makes it possible to ask for the current state of a specific device in the system and the reason of that state. The reason of a state can be because of a rule execution, a manual user action or an unknown reason. Moreover, the chatbot enables user to determine the future behavior of their system and the included devices of it, and additionally give information about the reason of those occurrences. More importantly, when a user has an expected behavior that did not occur in reality, HassTalk gives them the opportunity to ask why their expectation did not happen. At last, the chatbot will give users the opportunity to simulate the system's behavior on a given state.

These features are implemented in the chatbot in order to give more insight and understandability about the behavior of a Smart Home system. Users are able to ask for specific pieces of information and thereby understand the inner working of their systems. With the processing of natural language, the threshold of technical knowledge is lowered as it is not required to have any technical knowledge in order to understand the system's behavior.

After the chatbot was implemented, a user test was conducted in order to determine the usability of HassTalk. The results that came out of this test were rather positive, as the general impression of the participants was quite promising. The majority of the participants were able to solve all the given issues by utilizing the proposed chatbot. Performing a System Usability Scale research on the participants' findings, resulted in a B-score on the SUS-scale, which is considered as a good result.

Before further deploying the chatbot, more research has to be done about the different types of question formulations. The study in our research emphasized the need of a proper training set regarding the questions that are matched to an action. Before the chatbot could be useful in an household situation, other features may be implemented on which we like to refer to the research of Kar and Haldar [16]. An example of such a feature could be enabling the users to create routines in their system by using natural language. This can perfectly be acquired with the given framework as all the elements to implement this are present in the current system. Regarding the scope of our research, an interesting feature would be an algorithm to determine why something does not happen in the future. Considering the existing algorithms and services, this would not require a lot of extra work, as this would be a combination of the future algorithm (section 5.2) and the expectation algorithm (section 5.4).

A Explanation types

This list depicts the combinations of the different possible explanation types used in the exploratory study on information needs:

Single piece of information

- What
  The lamp is currently on / the sun is below horizon / Bram is at home
- What if
  (If the sun goes below horizon right now,) the lamp would turn on.
- Why
  Because the rule rule.sun_set is triggered / Bram did it / The sun is below horizon
- Why not
  Because the rule rule.sun_set is not triggered
- How to (not implemented)
  To turn on the lamp, the sun has to go below horizon.
- When
  The lamp will turn on at 19:34.

Two pieces of information

- Why + what
  The lamp is on, because the sun is below horizon
  The rule rule.sun_set is triggered, because the sun is below horizon
- Why + what if
  The lamp would turn on, because the rule rule.sun_set will be triggered.
  The lamp would turn on, because the sun went below horizon.
- Why + why not
  The lamp was off because the rule rule.sun_rise was triggered, but Bram has put the lamp on.
  The lamp was off because the sun was above horizon, but Bram has put the lamp on.
- Why + when
  The rule rule.sun_set is triggered at 19:56.
  The lamp will turn on at 19:56, because the sun will go below horizon
- What + what if
  When the sun goes below horizon, the lamp would turn on

- What + why not
  The lamp is not on, because the sun is currently above horizon.
  The rule rule.sun_set is not triggered, because the sun is above horizon.

- What + when
  The lamp is on since 12 o'clock

- What if + why not
  (if the sun goes below horizon) The lamp would turn on, currently it won't because the sun is above horizon.

- What if + when
  (if the sun goes below horizon, at 12 o'clock) The lamp will go on at 12 o'clock

- Why not + when
  The rule rule.sun_set is not triggered, it will be at 19:56.
  The lamp is not on because the sun is not below horizon, it will be at 19:56.

Three pieces of information

- Why + what + what if
  When the sun goes below horizon, the lamp would turn on because of the rule rule.sun_set

- Why + what + why not
  The lamp is not on, because the sun is currently above horizon. It needs to be below horizon

- Why + what + when
  The lamp is on, because the sun went below horizon at 19:56.
  The rule rule.sun_set is triggered because the sun went below horizon at 19:56

- Why + what if + why not
  (if the sun goes below horizon) The lamp would turn on, because the rule.sun_set will be triggered. Currently, the rule is not triggered because the sun is above horizon.

- Why + what if + when
  (if the sun goes below horizon, at 12 o'clock) The lamp will go on at 12 o'clock, because the rule rule.sun_set would be triggered.

- Why + why not + when
  The rule rule.sun_set was triggered at 19:56, but Bram has put the lamp off.
  The lamp was on because the sun is below horizon, but Bram has put the lamp off.

- What + what if + why not
  When the sun goes above horizon, the rule rule.sun_rise will be triggered and will turn off the lamp. Currently the rule is not triggered because the sun is below horizon.

- What + what if + when
  When the sun goes below horizon, the lamp would turn on. Which will happen at 19:56

- What + why not + when
  The lamp is not on, because the sun is not above horizon. This will happen at 19:56.
  The rule rule.sun_set is not triggered, because the sun is above horizon. It will go below horizon at 19:56

- What if + why not + when
  (when the sun goes below horizon) The lamp would turn on, currently the lamp is off because the sun is above horizon. It will be below horizon at 19:56.

Four pieces of information

- Why + what + what if + why not
  The lamp is currently on, because Bram did it. If the sun goes above horizon, the lamp would turn off.

- Why + what + what if + when
  When the sun goes below horizon, the lamp would turn on because of the rule rule.sun_set. Which will happen at 19:56.

- Why + what + why not + when
  The lamp is not on, because the sun is currently above horizon. It needs to be below horizon, which will happen at 19:56

- Why + what if + why not + when
  The lamp is currently on, because Bram did it. If the sun goes above horizon, the lamp would turn off. This will happen at 6:42

Five pieces of information

- Why + what + what if + why not + when
  The lamp is currently on, because Bram did it. If the sun goes above horizon, the lamp would turn off. This will happen at 6:42

B Response Templates

Sometimes the DialogFlow framework did not provide the necessary features in order to formulate an answer to the user. Therefore, manual response templates were implemented in order to create the answers.

B.1 Regarding the why (not) question

Name:	RULE_DENIED
Usage:	Describes the reason why a rule is not executed
Parameters:	@RULE_ID: the identifier of the rule which is not executed @TRIGGER_DESCR: Description of the rule's trigger @STATE_DESCR: Placeholder for other Response Template @DEVICE_NAME: The name of the device that is queried by the user @STATE: The state that the user expected
Current template:	the @DEVICE_NAME would be @STATE @TRIGGER_DESCR

Name:	STATE_DESCRIPTION
Usage:	Used in the RULE_DENIED template in order to describe the actual states of a rule's triggering entities
Parameters:	@DEVICE_ID: the identifier of the triggering entity @DEVICE_NAME: the name of the triggering entity @STATE: the current state of the triggering entity
Current template:	@DEVICE_NAME is currently @STATE

B.2 Regarding the future behavior of a device

Name:	CHANGE_DESCR
Usage:	Description of a change that is going to happen in the future
Parameters:	@DAY: The weekday of the change @HOURS: The hour on which the change occurs @MIN: The minute on which the change occurs @PROPERTY_CHANGES: Placeholder for other template
Current template:	on @DAY at @HOURS:@MIN, @PROPERTY_CHANGES

Name:	PROPERTY_CHANGE
Usage:	Description of which properties are changed in the previous template
Parameters:	@PROPERTY_TITLE: the title of the property that is changed @PROPERTY_VALUE: the new value of the property
Current template:	the @PROPERTY_TITLE will change to @PROPERTY_VALUE

Name:	FUTURE_DIFFERENCE
Usage:	Description of a difference between two simulations of futures
Parameters:	@DEVICE_NAME: The friendly name of the device @PROPERTY_TITLE: The title of the property which is different @PROPERTY_VALUE_1: The property's value in the first simulation @PROPERTY_VALUE_2: The property's value in the second simulation @TIME_DAYS: The weekday on which a difference is recorded @TIME_HOURS: The hour on which a difference is recorded @TIME_MINS: The minute on which a difference is recorded @NEXT_DAYS: The weekday of the next occurrence of @PROPERTY_VALUE_2 in the first future @NEXT_HOURS: The hour of the next occurrence of @PROPERTY_VALUE_2 in the first future @NEXT_MINS: The minute of the next occurrence of @PROPERTY_VALUE_2 in the first future @REASON_2: The reason of the next occurrence @REASON_1: The reason of the difference
Current template:	the @DEVICE_NAME would go @PROPERTY_VALUE_2

C Input intents

C.1 Current state of device: StateIntent

Action: GetDeviceState

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No

Training Phrases:

- What is the $DeviceName doing?
- What is the state of the $DeviceName
- What is the state of my $DeviceName

C.2 Future behavior of a device

C.2.1 StateAtTimeIntent

Action: What.Time

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	Time	@sys.time	$time	No
No	StateAttribute	@StateAttributes	$StateAttribute	No

Training Phrases:

- What is the $StateAttribute of the $DeviceName at $time?
- What is the state of the $DeviceName at $time
- What will my $DeviceName do at $time
- What will be the state of my $DeviceName at $time

C.2.2 CompleteFutureIntent

Action: What.Future

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No

Training Phrases:

- What is my $DeviceName going to do?
- What will my $DeviceName do?
- What will happen to my $DeviceName in the future?
- What will my $DeviceName do in the future?
- What will happen to my $DeviceName?

C.2.3 WhenIntent

Action: When.State

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	State	@AttributeValues	$State	No

Training Phrases:

- When will the $DeviceName $State?
- When does the $DeviceName turn $State?
- When will the $DeviceName go $State?

C.2.4 WhyAtTimeIntent

Action: Why.Time

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	Time	@sys.time	$time	No
Yes	State	@AttributeValues	$State	No

Training Phrases:

- Why does the $DeviceName turn $State at $time?
- Why is the $DeviceName $State at $time?
- Why will the $DeviceName go $State at $time?
- Why will the $DeviceName be $State at $time?

C.3 Reason of a device's state

C.3.1 WhyIntent

Action: Why.Device

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	State	@AttributeValues	$State	No

Training Phrases:

- The $DeviceName is $State, why?
- Why is the $DeviceName turned $State?
- Why are my $DeviceName $State?
- Why is my $DeviceName $State?

C.3.2 WhyNotIntent

Action: WhyNot.State

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	State	@AttributeValues	$State	No

Training Phrases:

- Why are the $DeviceName not $State?
- Why are my $DeviceName not $State?
- Why is the state of the $DeviceName not $State?
- Why is my $DeviceName not $State?

C.4 Reason why expectation did not happen: ExpectationIntent

Action: Expectation.Raw

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	State	@AttributeValues	$State	No

Training Phrases:

- Why is my $DeviceName $State, while I'm home?
- I thought the $DeviceName would be $State right now, why isn't it?

- Why is the $DeviceName not $State, while the sun is below horizon?
- The $DeviceName should be $State, why isn't it?
- I expected that the $DeviceName would be $State, why did this not happen?

C.5 Simulating behavior on a device's state change: SimulationIntent

Action: Simulation.FullDescription

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@DeviceName	$DeviceName	No
Yes	AttributeValue	@AttributeValues	$AttributeValue	No
No	SelectedAttribute	@StateAttributes	$SelectedAttribute	No
No	Time	@sys.time	$Time	No

Training Phrases:

- What happens if $DeviceName $AttributeValue $Time?
- What if $DeviceName was $AttributeValue $Time?
- What will happen when my $DeviceName goes $AttributeValue at $Time?
- What will happen, when the $SelectedAttribute of my $DeviceName is $AttributeValue at $Time?

D Answer intents

D.1 Current state of device: StateAnswer

Input event: GotStateAnswer

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
Yes	DeviceName	@sys.any	$DeviceName	No
Yes	State	@sys.any	$State	No

Response Phrases:

- Currently the state of the $DeviceName is $State
- The $DeviceName is currently $State

D.2 Future behavior of a device

D.2.1 StateAtTimeAnswer

Input event: WhatAtTimeAnswer

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
Yes	DeviceName	@sys.any	$DeviceName	No
Yes	Value	@sys.any	$Value	No
Yes	Time	@sys.any	$Time	No

Response Phrases:

- The state of the $DeviceName will be $value at $time

D.2.2 CompleteFutureAnswer

Input event: FutureAnswer

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	DeviceName	@sys.any	$DeviceName	No
Yes	Future	@sys.any	$Future	Yes

Response Phrases:

- I found that $Future.

D.2.3 WhenAnswer

Input event: WhenAnswer

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
Yes	DeviceName	@sys.any	$DeviceName	No
Yes	State	@sys.any	$State	Yes
Yes	Time	@sys.any	$Time	No
Yes	Trigger	@sys.any	$Trigger	No

Response Phrases:

- The $DeviceName will go $State at $Time, $Trigger.
- The $DeviceName will go $State at $Time.

D.3 Reason of a device's state

D.3.1 Reason_BecauseOfRule

Input event: Reason_BecauseOfRule

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
No	DeviceName	@sys.any	$DeviceName	No
Yes	InverseState	@sys.any	$InverseState	Yes
Yes	Time	@sys.any	$Time	No
Yes	Trigger	@sys.any	$Trigger	No
Yes	CurrState	@sys.any	$CurrState	No
No	RuleID	@sys.any	$RuleID	No

Response Phrases:

- I did it at $Time, because I have to $Trigger. Did you expect it to be $InverseState?

D.3.2 Reason_BecauseOfUser

Input event: Reason_BecauseOfUser

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
No	DeviceName	@sys.any	$DeviceName	No
Yes	InverseState	@sys.any	$InverseState	Yes
Yes	Time	@sys.any	$Time	No
Yes	Actor	@sys.any	$Actor	No
No	CurrState	@sys.any	$CurrState	No

Response Phrases:

- $Actor did it at $Time. Should it be $InverseState?

D.3.3 Reason_BecauseOfUnknown

Input event: Reason_BecauseOfUnknown

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
No	DeviceName	@sys.any	$DeviceName	No
Yes	InverseState	@sys.any	$InverseState	Yes
Yes	Time	@sys.any	$Time	No
No	CurrState	@sys.any	$CurrState	No

Response Phrases:

- It happened at $Time, but I don't know why. Did you expect it to be $InverseState?

D.3.4 Reason_RuleDeniedByRule

Input event: Reason_RuleDeniedByRule

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
No	DeviceName	@sys.any	$DeviceName	No
No	CurrState	@sys.any	$CurrState	No
Yes	CurrReason	@sys.any	$CurrReason	No
Yes	DeniedRules	@sys.any	$DeniedRules	Yes
Yes	Time	@sys.any	$Time	No

Response Phrases:

- Normally, $DeniedRules. But I did it at $Time, because I have to $CurrReason

D.3.5 Reason_RuleDeniedByUser

Input event: Reason_RuleDeniedByUser

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
No	DeviceName	@sys.any	$DeviceName	No
No	CurrState	@sys.any	$CurrState	No
Yes	Actor	@sys.any	$Actor	No
Yes	DeniedRules	@sys.any	$DeniedRules	Yes
Yes	Time	@sys.any	$Time	No

Response Phrases:

- Normally, $DeniedRules. But $Actor has turned it $CurrState at $Time.

D.4 Reason why expectation did not happen: ExpectationAnswer

Input event: ExpectationAnswer

Parameters:

Required	Parameter Name	Entity	Value	Is list?
No	DeviceID	@sys.any	$DeviceID	No
Yes	DeviceName	@sys.any	$DeviceName	No
Yes	State	@sys.any	$State	No
Yes	Triggers	@sys.any	$Triggers	No
Yes	FaultingStates	@sys.any	$FaultingStates	Yes

Response Phrases:

- The $DeviceName would turn $State $Triggers. But $FaultingStates.

D.5 Simulating behavior on a device's state change: SimulationAnswer

Input event: ExpectationAnswer

Parameters:

Required	Parameter Name	Entity	Value	Is list?
Yes	Differences	@sys.any	$Differences	Yes

Response Phrases:

- I found that $Differences

E Entity Types

As stated in section 4.3.2, the DialogFlow is trained with the possible entity types. This appendix contains the entity types used in order to conduct the study in section 6.

E.1 DeviceName

List of known devices containing their identifiers and possible friendly names.

Reference value	**Synonym(s)**
media.smart_tv	smart tv, tv, television, telly
sun.sun	sun
light.lamp	lamp, lights
person.Bram	bram
guide.tv_guide	tv guide, guide
heater.Heater	heater, heating
sensor.roomba_battery	roomba battery, cleaner battery
sensor.temperature	temperature sensor, thermometer
cleaner.Cleaner	cleaner, roomba
thermostat.Thermostat	thermostat

E.2 State Attributes

List of known attributes for the devices in the system.

Reference value	Synonym(s)
state	state
rising	rising
next_dawn	next dawn
next_dusk	next dusk
next_midnight	next midnight
next_noon	next noon
next_rising	next rising
next_setting	next setting
effect_list	effect list
max_mireds	max mireds
min_mireds	min mireds
supported_features	supported features
brightness	brightness
hs_color	hs color
rgb_color	rgb color
xy_color	xy color
effect	effect
flash	flash
editable	editable
gps_accuracy	gps accuracy
id	id, identifier
latitude	latitude
longitude	longitude
source	source
user_id	user id, user identifier
content_type	content type
heating_rate	heating rate
time_left	time left
entity_picture	entity picture, picture
restored	restored

E.3 AttributeValues

List of possible values for a given attribute (currently only the state), with the attribute's title used as prefix in order to match the value with the right attribute:

Reference value	**Synonym(s)**
state_The_News	the news
state_on	on, running, active, shining powered
state_off	off, out
state_above_horizon	above horizon, rise, rises, up
state_below_horizon	below horizon, set, sets, down
state_unknown	unknown
state_home	home, arrives
state_Friends	Friends
state_away	away, leaves, gone, not home

F Nederlandse samenvatting

F.1 Introductie

De wereld van Internet of Things kent een immense groei en zal in de komende tijd alleen nog maar verder groeien [2]. Maar ondanks deze groei, zijn smart homes nog niet prominent aanwezig in de leefwereld van vele mensen. Voorgaand onderzoek [3] toonde aan dat er nog verschillende uitdagingen zijn omtrent deze smart homes, die overwonnen zullen moeten worden vooraleer deze meer geaccepteerd zullen worden in ons dagelijks bestaan. In dit onderzoek trachten wij één van deze challenges te overwinnen: het begrijpbaar maken van een IoT-systeem.
HassTalk is een chatbot die de eigenaar van een smart home helpt met het begrijpen van het onderliggend gedrag van zijn of haar systeem. De software laat de gebruiker toe vragen te stellen over het gedrag van het systeem, zoals "Waarom is X gebeurd?", "Wanneer zal X gebeuren?", "Wat gebeurt er als X gebeurt?", etc.
Met deze implementatie trachten wij ondervraagbaarheid naar de wereld van IoT te brengen en zo de gebruiker te helpen in het verstaan en begrijpen van zijn of haar Smart Home.

F.2 Voorgaand onderzoek

Het gebruik van natuurlijke taal in termen van IoT-systemen is niks nieuws, er is tal van onderzoek gedaan naar de mogelijkheden van een chatbot in een Smart Home omgeving [10, 11, 13, 15, 16]. Dit onderzoek heeft aangetoond dat het gebruik van een chatbot een goede benadering kan zijn om de moeilijkheidsgraad van Smart Home en IoT-systemen te verlagen. Door het gebruik van natuurlijke taal is het immers gemakkelijker om een expressie uit te drukken of begrijpen.
Bijkomend is er onderzocht hoe een chatbot dient geëvalueerd te worden. Het onderzoek van Walker et al. [22] gaf ons een overzicht van de subjectieve metingen die we gebruiken in ons onderzoek. Om deze maatstaven te kunnen bepalen maken we gebruik van een System Usability Scale [24].
Zoals reeds aangehaald zijn er verschillende uitdagingen die dienen overwonnen te worden in verband met IoT-systemen. Er is onderzoek gedaan naar de invloed van leeftijd, taken en intelligentie van de gebruiker bij het gebruiken van IoT-systemen [25]. Ander onderzoek toonde dan weer aan dat het belang van de juiste feedback in zulke systemen niet onderschat mag worden [27]. De conclusie die we uit deze onderzoeken konden halen is dat er een gat bestaat tussen een IoT-systeem en zijn gebruiker, en dat deze overwonnen dient te worden [28].
Op vlak van het bevraagbaar maken van software-omgevingen toont voorgaand onderzoek aan dat het een grote additie kan zijn voor de gebruiker om vragen te kunnen stellen over het gedrag van een bepaald systeem. De waarom-vraag is in veel van deze onderzoeken naar boven gekomen [41, 38, 43], maar dit is slechts één voorbeeld van een mogelijke vraag. Meerbepaald het onderzoek van Lim en Dey [38] gaf ons een mooi overzicht van de verschillende soorten informatie die gevraagd kunnen worden.

F.3 Verkennende studie omtrent de informatieve noden van de gebruiker

De software die in dit onderzoek ontworpen wordt, maakt gebruik van natuurlijke taal als middel van communicatie. Daardoor is het belangrijk om te bepalen welke stukjes informatie een gebruiker noodzakelijk acht, om het achterliggend gedrag van het syteem volledig te kunnen begrijpen. Daarom hebben we via een Google Form een test uitgevoerd, die achterhaalt welke stukjes informatie een gebruiker interessant vindt in een bepaalde situatie. Meer bepaald werden de volgende hypotheses onderzocht:

- **H1** Wanneer een gebruiker achter de reden van een bepaalde gedraging vraagt, zal deze de waarom-informatie bouwblok verwachten als antwoord.
- **H2** Wanneer de gebruiker zijn of haar verwachting uitdrukt, zal deze gebruik maken van de waarom-niet-notatie (bv. Gebruiker verwacht dat de lamp brandt, maar dit is niet zo → "Waarom is mijn lamp niet aan?").
- **H3** Wanneer er achter de reden van een toekomstige gedraging gevraagd wordt, zal er tevens een tijdstip toegevoegd moeten worden aan de uitleg.
- **H4** Wanneer de gebruiker vraagt waarom een regel NIET is uitgevoerd, wil deze weten of de regel uberhaupt correct is geïmplementeerd.
- **H5** Wanneer de gebruiker vraagt waarom een regel NIET is uitgevoerd, wil deze de staat weten van de entitieten die de regel kunnen doen uitvoeren.
- **H6** Wanneer de gebruiker vraagt voor een toekomstige gedraging, zal deze het tijdstip van de gedraging willen weten, alsook het event waaruit de gedraging resulteert.

Het onderzoek toonde aan dat de hypothese **H1** geaccepteerd kan worden, wat een logische conclusie is. Wanneer de gebruiker vraagt waarom iets gebeurd is, zal deze het meest geïnteresseerd zijn in de reden van de gedraging. Intressanter was echter dat de deelnemers aantoonden dat het tijdstip van de gedraging tevens interessant is. Wanneer de gebruiker vroeg waarom een bepaalde gedraging wel of niet gebeurde, werd er afwisselend gebruik gemaakt van de waarom- en waarom-niet-formulering. Daardoor moeten we de hypothese **H2** weerleggen, aangezien we er niet van uit kunnen gaan dat de gebruiker zijn of haar verwachtingen uitdrukt aan de hand van deze formulering. De test gaf tevens duidelijk aan dat bij het vragen naar een toekomstige gedraging, de tijd altijd een interessant deeltje informatie is voor de gebruiker. Dit wilt zeggen dat we de hypothese **H3** kunnen accepteren. Ook hypothese **H4** kan geaccepteerd worden, aangezien de deelnemers duidelijk wilden weten of de regel in kwestie correct geïmplementeerd was. Dit zorgt er namelijk voor dat ze weten dat het probleem niet in de implementatie van de regel ligt. Wanneer een regel (onverwacht) niet uitgevoerd werd, gaven de deelnemers aan enkel geïnteresseerd te zijn in de staat van de entitieten die er voor zorgden dat de regel niet uitgevoerd werd. Daardoor kunnen we hypothese **H5** niet accepteren, daar de gebruikers niet geïnteresseerd waren in ALLE entitieten

die de regel omvat. Tenslotte gaf de test aan dat de gebruiker meer geïnteresseerd is in het event dat een bepaalde gedraging veroorzaakt, dan het tijdstip. Hierdoor kunnen we hypothese **H6** niet accepteren, toch bleek uit de andere testen dat de tijdsindicatie van een bepaalde gebeurtenis vaak als belangrijk aanschouwd werd. We willen dan ook benadrukken dat het meedelen van een tijdstip nooit als overbodige informatie gezien kan worden.

F.4 HassTalk Architectuur

De infrastructuur van HassTalk is opgedeeld in 4 onderdelen, zoals afgebeeld in figuur 37. Het eerste onderdeel in deze infrastructuur is Home Assistant, een open source Home Automation tool. Dit systeem laat gebruikers toe om combinaties van smart devices te interconnecteren en manipuleren. Home Assistant houdt bij welke apparaten zich in een Smart Home bevinden en houdt daarbovenop bij wat de staten zijn van deze verschillende apparaten.

De FORTNIoT-Toolkit van Sven Coppers interaggeert met Home Assistant om zo notie te krijgen van de huidige staat van het Smart Home systeem. Dit framework houdt tevens de verschillende devices en hun respectivelijke staten bij en laat het daarbij toe om logica te creëren tussen deze apparaten. Doordat de logica gespecifiëerd is in dit framework, kan de FORTNIoT-Toolkit voorspellingen doen over de gedragingen van de verschillende apparaten. Om de nodige informatie op te vragen werden er verschillende bestaande endpoints gebruikt, alsook toegevoegd voor HassTalk specifiek (zoals te zien in figuur 37). De bestaande endpoints van FORTNIoT lieten toe om de staten van de verschillende apparaten op te vragen. Ook is er een mogelijkheid om een historisch overzicht en een toekomstvoorspelling op te vragen voor deze staten. Daardoor is HassTalk capabel om een inschatting te doen over het afgelopen, huidig en opkomend gedrag van het systeem. Zoals reeds vermeld kunnen er ook routines opgebouwd worden in de FORTNIoT-Toolkit, die we vanaf nu TCA-regels (Trigger-Condition-Action regels) zullen noemen. Om een overzicht te krijgen van de bestaande regels in het systeem werd er een endpoint voorzien die het mogelijk maakt de nodige informatie over een bepaalde regel op te vragen. Dit houdt een beschrijving over de trigger van de regel (bv. Als de zon onder gaat en er is iemand thuis.) en de actie die ondernomen wordt (bv. Zet de lamp aan.) in. Hierdoor kan HassTalk bepalen of een bepaalde regel al dan niet uitgevoerd is in de huidige omstandigheden en waarom wel of niet. Bijkomend laat FORTNIoT toe om een overzicht op te vragen van de voorbije uitvoeringen van bepaalde regels, alsook de opkomende uitvoeringen. Hierdoor kan HassTalk bepalen welke regels zullen uitgevoerd worden in de nabije toekomst. Wat al eerder aangetoond heeft dat dit een duidelijk overzicht biedt omtrent de gedragingen van een systeem.
De endpoints die reeds geïmplementeerd waren, lieten HassTalk niet toe om volledige beschrijvingen te geven van bepaalde voorvallen. Daarom zijn er een aantal endpoints bijkomend geïmplementeerd om zo de missende informatie toch op te kunnen vragen. Een eerste endpoint is gemaakt om het toe te laten een overzicht te krijgen van de verschillende apparaten die zich in het systeem bevinden. Hierdoor is HassTalk capabel

om een bepaald ID te koppelen aan een meer bruikbare naam van het apparaat. Om de reden van een bepaalde staat te bepalen is de why-route geïmplementeerd, welke een device ID als parameter neemt. Dit algoritme zal vervolgens bepalen waarom dat specifiek apparaat zijn staat heeft bekomen. Dit kan het resultaat zijn van een gebruikersactie, een uitvoering van een regel of een onbekende reden. Deze endpoint laat het vervolgens toe om de reden van een bepaalde staat te achterhalen. Bijkomend is een endpoint gecreëerd die het toe laat te achterhalen waarom een apparaat NIET een bepaalde state heeft (bv. Waarom is de lamp NIET aan?). Dit algoritme zal een lijst teruggeven van regels die in die bepaalde staat zullen resulteren. Bijkomend zal hierbij vermeld worden of deze regel uitgevoerd werd in de gegeven omstandigheden, en waarom dit al dan niet het geval is. Dit wordt gedaan door per trigger die een regel bevat, aan te geven of deze trigger al dan niet voldaan is. Op deze manier kunnen we achterhalen of een regel correct geïmplementeerd en uitgevoerd is. Om HassTalk toe te laten een beschrijving van een bepaalde regel te geven is er een nieuwe endpoint gemaakt die een regel-identifier en een apparaat-identifier neemt. Dit algoritme gaat vervolgens na wat de gevolgen zijn van deze regel op dat specifieke apparaat (bv. ALS de zon onder gaat DAN zet ik de lamp aan!). Dit maakt het mogelijk een tekstuele representatie te maken van een regel, de omstandigheden waarin de regel uitgevoerd zou worden en de gevolgen die deze regel zou hebben mits deze uitgevoerd zou worden. De laatste route die geïmplementeerd diende te worden was de simulatie-route. Hier bestond reeds een versie van in de originele versie van FORTNIoT, maar deze voldeed niet aan de vereisten van HassTalk. Daardoor is een nieuwe route gemaakt die als input een aangepaste staat neemt van een apparaat. Vervolgens gaat het algoritme na wat de gedragingen van het systeem zullen zijn, moest deze staat actief zijn in het systeem. Een voorbeeld zou kunnen zijn dat er een staat gegeven wordt van de zon waarin beschreven staat dat deze op het huidig tijdstip onder zal gaan, het systeem gaat dan na welke regels uitgevoerd zullen worden in deze omstandigheden (bv. De lamp zal aan gaan.). Dit laat de gebruiker toe bepaalde gedragingen te simuleren en daardoor te valideren.

De HassTalk software zal er voor zorgen dat de informatie die verkregen werd van de FORTNIoT-Toolkit, verstaanbaar gemaakt wordt voor de eindgebruiker. Dit wordt gedaan door de juiste parameters door te geven naar de DialogFlow Client. DialogFlow laat het toe om de intentie van een gebruiker te bepalen aan de hand van de gebruikte zinsstructuur. Hierdoor kan iemand met natuurlijke taal een bepaalde vraag stellen, waarin bepaalde essentiële parameters vermeld zijn (bv. Waarom is mijn lamp aan?). DialogFlow herkent hierbij wat de intentie is van de gebruiker (bv. Reden bepalen waarom de lamp aan staat) en extraheert de parameters uit de zin (bv. Apparaat = lamp, staat = aan, intentie = waarom). Dit laat HassTalk toe om de juiste stappen te ondernemen om een antwoord te vormen naar de gebruiker toe. De parameters die meegegeven werden zullen gebruikt worden in een bepaald achterliggend algoritme, waarbij het resultaat teruggegeven wordt aan de DialogFlow Client. Dit resultaat zal vervolgens gebruikt worden om een zin te construeren die een geschikt antwoord vormt voor de gebruiker.

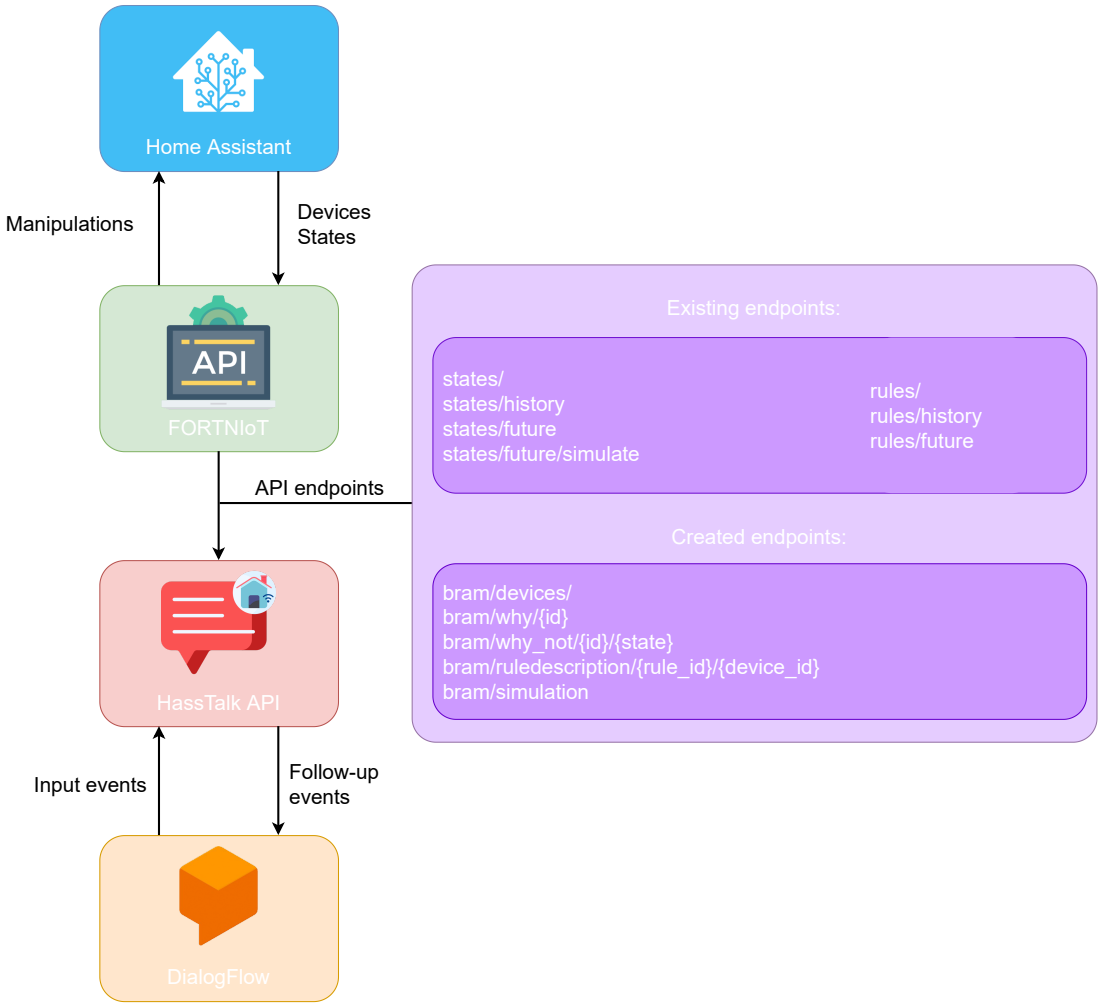


Figure 37: De architectuur van HassTalk: Home Assistant (verbinden en manipuleren van de smart devices in een huis), FORTNIoT-toolkit (laat toe om voorspellingen te doen over het gedrag van het systeem), HassTalk API (Vertalen van informatie tussen FORTNIoT-toolkit en de gebruiker) en DialogFlow (framework van Google dat toelaat zelf een chatbot te creëren)

F.5 HassTalk: implementatie

HassTalk is verantwoordelijk om de juiste stappen te ondernemen bij een bepaalde gebruikersintentie. Hierbij wordt de intentie bepaald door de DialogFlow Client, die tevens de nodige parameters voorziet voor HassTalk. HassTalk zal deze intentie en parameters vervolgens gebruiken om de FORTNIoT-Toolkit aan te spreken en de vereiste informatie op te vragen.

Opvragen van de huidige staat van een apparaat
HassTalk laat de gebruiker toe de huidige staat van een bepaald apparaat op te vragen. Wanneer de gebruiker vraagt naar de staat van een bepaald apparaat, zal HassTalk de states-route van de FORTNIoT-Toolkit aanspreken om de huidige staat van het apparaat op te vragen. Deze staat wordt vervolgens teruggegeven aan de DialogFlow Client die er voor zorgt dat er een adequaat antwoord gevormd wordt naar de gebruiker toe.

Opvragen van toekomstig gedrag van een apparaat
Om de gebruiker notie te geven over het opkomend gedrag van zijn of haar systeem, voorziet HassTalk verschillende toekomstgerichte vragen. Om deze vragen te kunnen beantwoorden zal HassTalk het opkomend gedrag van het device opvragen van de FORTNIoT-Toolkit, alsook de opkomende regelexecuties. Hierdoor is de software capabel om het toekomstig gedrag van het apparaat ondervraagbaar te maken, alsook de reden van dat gedrag. Hierdoor is het mogelijk om de staat van een apparaat op te vragen op een bepaald tijdstip (bv. Wat zal de staat van mijn lamp zijn om 8u?), het tijdstip bepalen wanneer een apparaat een bepaalde staat zal hebben (bv. Wanneer zal mijn lamp uit gaan?), de complete toekomst van een apparaat (bv. Wat zal mijn lamp doen de komende tijd?) en tenslotte de reden waarom een bepaalde gedraging zich voordoet (bv. Waarom zal mijn lamp uit zijn om 8u?).

De reden van een bepaalde staat
Soms kan het voor de gebruiker niet duidelijk zijn waarom een apparaat een bepaalde staat heeft. Daarom laat HassTalk het toe te bevragen waarom een apparaat een bepaalde staat wel of niet heeft. Hierbij worden de why en why-not endpoints van de FORTNIoT-Toolkit gebruikt om de reden te achterhalen. Zoals reeds vermeld kan de reden van een huidige staat drie vormen aannemen: een manuele gebruikersactie, een regelexecutie of een onbekende reden. Het kan echter ook dat een apparaat een bepaalde staat heeft doordat er een bepaalde regel niet is uitgevoerd (bv. een lamp die aan staat, doordat de regel die hem uitzet niet uitgevoerd is). Daarom laat HassTalk het tevens toe hierover bevragingen te doen, zodat de gebruiker weet waarom een bepaalde regel eventueel niet uitgevoerd is in de gegeven omstandigheden. Stel bijvoorbeeld dat de gebruiker vraagt “Waarom is mijn Roomba aan het poetsen, terwijl ik thuis ben?”, uit deze formulering kunnen we opmaken dat de gebruiker ervan uit gaat dat de Roomba niet bezig zou moeten zijn. Het systeem zal hier op antwoorden als volgt: “De Roomba zou uitgaan als er iemand thuis is, maar ik kon jouw locatie niet achterhalen”.

Simuleren van alternatief gedrag
Om een het gedrag van het systeem beter te kunnen begrijpen, kan een gebruiker vragen hoe het systeem zich zal gedragen onder bepaalde omstandigheden. Daarvoor voorziet HassTalk een optie door de gebruiker toe te laten een beschrijving van deze alternatieve staat te geven (bv. Wat gebeurt er als de zon nu opkomt?). De software gaat vervolgens na welke veranderingen er zullen optreden in het systeem. Bijkomend zal de software een vergelijking maken tussen de verwachte toekomst in realiteit en de gesimuleerde toekomst. Dit laat het toe om een antwoord te vormen als volgt: "De lamp zal uit gaan, wat normaal pas zal gebeuren om 7u43.". Dit maakt het mogelijk voor de gebruiker om het gedrag van het systeem in te schatten onder bepaalde omstandigheden en daar bovenop een vergelijking te maken met het huidige gedrag van het systeem.

F.6 Gebruikerstest

Om te bepalen of het resultaat uit dit onderzoek het gewenste resultaat behaald heeft, hebben we een gebruikerstest uitgevoerd. In deze test werden de deelnemers geconfronteerd met verschillende situaties waar een onverwachte gedraging is opgetreden. De deelnemers moesten vervolgens vragen beantwoorden over deze gedragingen, met behulp van de bijgeleverde chatbot. Een voorbeeld hiervan is een situatie waar de lamp onverwacht aan staat, de deelnemer moet vervolgens achterhalen hoe het komt dat deze lamp aan staat. Uit deze test kwamen vooral positieve resultaten, daar nagenoeg alle deelnemers in staat waren om de vragen correct te beantwoorden. Over het algemeen werd er wel een gevoel van twijfel uitgedrukt, wat benadrukt dat de gebruikers niet altijd even veel vertrouwen hadden in het systeem.
Vervolgens hebben we een enquête afgenomen bij de gebruikers om de subjectieve satisfactie te bepalen aan de hand van de System Usability Scale. De berekening gaf ons een SUS-score van 72,5 wat ons een B-rating geeft op de schaal. Dit wordt algemeen aanschouwd als een goed resultaat.
Uit de eerste test die we ondernamen werd duidelijk dat de gebruiker zijn verwachtingen niet uitdrukte aan de hand van een waarom-niet-expressie. Uit deze test echter bleek dat de gebruiker zijn verwachtingen uitdrukt door een beschrijving te geven van de huidige situatie (bv. Waarom is de lamp aan, terwijl er niemand thuis is?). We hebben dit resultaat dan ook verwerkt in de chatbot, waardoor het nu mogelijk is voor de gebruiker om zijn of haar verwachtingen op die manier uit te drukken.

F.7 Conclusie

HassTalk is een systeem dat aan de hand van natuurlijke taal toe laat voor gebruikers om het gedrag van een IoT-systeem beter te begrijpen. Hierbij laat het systeem toe om de huidige staat van een systeem op te vragen alsook bevragingen over de toekomstige staten van het systeem. Bijkomend is er de mogelijkheid om de reden achter deze gedragingen te achterhalen met de chatbot. Om een zo volledig mogelijke beschrijving te geven aan de gebruiker werd er een voorgaande studie uitgevoerd waarbij achterhaald werd welke stukjes informatie een gebruiker vereist in bepaalde situaties. De resultaten die uit deze test kwamen zijn meegenomen in de implementatie van de chatbot, waarna we de uiteindelijke uitwerking hebben getest. Uit die laatste test kwamen vooral positieve resultaten, aangezien de gebruikers weinig moeite ervaarden bij het oplossen van de voorgestelde problemen.

Dit onderzoek eindigt dan ook op een positieve noot, aangezien we bewezen hebben dat het gebruik van natuurlijke taal een goede benadering is om systemen ondervraagbaar en begrijpbaar te maken. Zoals reeds vermeld zijn er nog andere uitdagingen aan te gaan om het gat tussen de gebruiker en zijn systeem te dichten. Het ondervraagbaar en begrijpbaar maken van het systeem kan hier een aanzienlijk deel van uitmaken, maar om een complete chatbot te verkrijgen zijn er nog bijkomende features nodig. Denk zo aan de mogelijkheid om regels te creëren op basis van natuurlijke taal.

References

[1] Y. Nakamura, Y. Arakawa, T. Kanehira, M. Fujiwara, and K. Yasumoto, "SenStick: Comprehensive Sensing Platform with an Ultra Tiny All-In-One Sensor Board for IoT Research," *Journal of Sensors*, vol. 2017, 2017.

[2] MordorIntelligence, "Smart Homes Market — Growth, Trends, Forecast (2020 - 2025)."

[3] W. Keith Edwards and R. E. Grinter, "At home with ubiquitous computing: Seven challenges," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2201, pp. 256–272, 2001.

[4] J. Hill, W. Randolph Ford, and I. G. Farreras, "Real conversations with artificial intelligence: A comparison between human-human online conversations and human-chatbot conversations," *Computers in Human Behavior*, vol. 49, pp. 245–250, 2015.

[5] J. F. Allen, D. K. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent, "Toward conversational human-computer interaction," *AI Magazine*, vol. 22, no. 4, pp. 27–37, 2001.

[6] B. Ury, E. McManus, M. P. Y. Ho, and M. L. Littman, "Practical trigger-action programming in the smart home," *Conference on Human Factors in Computing Systems - Proceedings*, pp. 803–812, 2014.

[7] D. Randall, "Living Inside a Smart Home: A Case Study," in *Inside the Smart Home*, pp. 227–246, Springer-Verlag, apr 2006.

[8] A. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home automation in the wild: Challenges and opportunities," pp. 2115–2124, 05 2011.

[9] S. Mennicken and E. M. Huang, "Hacking the natural habitat: An in-the-wild study of smart homes, their development, and the people who live in them," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7319 LNCS, pp. 143–160, 2012.

[10] C. Hewitt, "Viewing control structures as patterns of passing messages," *Artificial Intelligence*, vol. 8, no. MA 02139, pp. 323–364, 1977.

[11] H. S. Nwana, "Software agents: an overview," *The Knowledge Engineering Review*, vol. 11, no. 3, p. 205–244, 1996.

[12] B. W. Schermer *et al.*, *Software agents, surveillance, and the right to privacy: a legislative framework for agent-enabled surveillance*. Leiden University Press, 2007.

[13] A. Følstad and P. B. Brandtzaeg, "Chatbots and the New World of HCI," *Interactions*, vol. 24, no. 4, pp. 38–42, 2017.

[14] L. Benotti, M. C. Martínez, and F. Schapachnik, "Engaging high school students using Chatbots," *ITICSE 2014 - Proceedings of the 2014 Innovation and Technology in Computer Science Education Conference*, pp. 63–68, 2014.

[15] T. Jakobi, G. Stevens, N. Castelli, C. Ogonowski, F. Schaub, N. Vindice, D. Randall, P. Tolmie, and V. Wulf, "Evolving Needs in IoT Control and Accountability," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 4, pp. 1–28, 2018.

[16] R. Kar and R. Haldar, "Applying Chatbots to the Internet of Things: Opportunities and Architectural Elements," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, 2016.

[17] M. LIttman and S. Kortchmar, "The Path To A Programmable World – Footnote," jun 2014.

[18] C. J. Baby, F. A. Khan, and J. N. Swathi, "Home automation using IoT and a chatbot using natural language processing," *2017 Innovations in Power and Advanced Computing Technologies, i-PACT 2017*, vol. 2017-Janua, pp. 1–6, 2018.

[19] K. Cheverst, K. Mitchell, and N. Davies, "Investigating Context-aware Information Push vs. Information Pull to Tourists," *in Proceedings of Mobile Hci 01*, vol. 1, p. 2001, 2001.

[20] A. Følstad, C. B. Nordheim, and C. Bjørkli, "What Makes Users Trust a Chatbot for Customer Service ? An Exploratory Interview Study . What Makes Users Trust a Chatbot for Customer Service ? An Exploratory Interview Study Chatbots are software agents that interact with users through natural language," vol. 11193, no. December, pp. 194–208, 2018.

[21] B. J. Cahn, "[Thesis] Chatbot Literature Review," 2017.

[22] M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella, "PARADISE: A Framework for Evaluating Spoken Dialogue Agents," pp. 271–280, 1997.

[23] K. Kuligowska, "Commercial chatbot: Performance evaluation, usability metrics and quality standards of embodied conversational agents," *Professionals Center for Business Research*, vol. 2, pp. 1–16, 01 2015.

[24] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *International Journal of Human-Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

[25] B. Zhang, P. L. P. Rau, and G. Salvendy, "Design and evaluation of smart home user interface: Effects of age, tasks and intelligence level," *Behaviour and Information Technology*, vol. 28, no. 3, pp. 239–249, 2009.

[26] S. Leppënen and M. Jokinen, "Daily Routines and Means of Communication in a Smart Home," *Inside the Smart Home*, pp. 207–225, 2006.

[27] A. Norman, "The " Problem " Inappropriate of Automation : and Interaction , Feedback Not " Overautomation " Donald July 1989 ICS Report A / c , c '. e- S '_/ Institute for Cognitive Science University of California , San Diego La Jolla , California 92093 Paper prepar," 1989.

[28] A. Hwang and J. Hoey, "Smart home, the next generation: Closing the gap between users and technology," *AAAI Fall Symposium - Technical Report*, vol. FS-12-01, pp. 14–21, 2012.

[29] L. Barkhuus and A. Dey, "Is context-aware computing taking control away from the user? Three levels of interactivity examined," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2864, pp. 149–156, 2003.

[30] A. Al Farooq, E. Al-Shaer, T. Moyer, and K. Kant, "IoTC2: A formal method approach for detecting conflicts in large scale IoT systems," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, pp. 442–447, 2019.

[31] M. Manca, Fabio, Paternò, C. Santoro, and L. Corcella, "Supporting end-user debugging of trigger-action rules for IoT applications," *International Journal of Human Computer Studies*, vol. 123, no. May 2018, pp. 56–69, 2019.

[32] F. Corno, L. De Russis, and A. Monge Roffarello, "Empowering end users in debugging trigger-action rules," *Conference on Human Factors in Computing Systems - Proceedings*, pp. 1–13, 2019.

[33] Y. Chen, S. W. Lee, Y. Xie, Y. W. Yang, W. S. Lasecki, and S. Oney, "Codeon: On-demand software development assistance," *Conference on Human Factors in Computing Systems - Proceedings*, vol. 2017-May, pp. 6220–6231, 2017.

[34] Y. Chen, S. Oney, and W. S. Lasecki, "Towards providing on-demand expert support for software developers," *Conference on Human Factors in Computing Systems - Proceedings*, pp. 3192–3203, 2016.

[35] P. A. Akiki, A. K. Bandara, and Y. Yu, "Visual simple transformations: Empowering end-users to wire internet of things objects," *ACM Transactions on Computer-Human Interaction*, vol. 24, no. 2, 2017.

[36] A. K. Dey, T. Sohn, S. Streng, and J. Kodama, "iCAP: Interactive Prototyping of Context-Aware Applications," *PERVASIVE*, pp. 254–271, 2006.

[37] A. J. Ko and B. A. Myers, "Designing the Whyline: A debugging interface for asking questions about program behavior," *Conference on Human Factors in Computing Systems - Proceedings*, vol. 6, no. 1, pp. 151–158, 2004.

[38] B. Y. Lim and A. K. Dey, "Toolkit to support intelligibility in context-aware applications," *UbiComp'10 - Proceedings of the 2010 ACM Conference on Ubiquitous Computing*, pp. 13–22, 2010.

[39] A. K. Dey and A. Newberger, "Support for context-aware intelligibility and control," in *Conference on Human Factors in Computing Systems - Proceedings*, (New York, New York, USA), pp. 859–868, ACM Press, 2009.

[40] B. Myers, D. A. Weitzman, A. J. Ko, and D. H. Chau, "Answering Why and Why Not Questions in User Interfaces," tech. rep., 2006.

[41] J. Vermeulen, G. Vanderhulst, K. Luyten, and K. Coninx, "PervasiveCrystal: Asking and answering why and why not questions about pervasive computing applications," *Proceedings - 2010 6th International Conference on Intelligent Environments, IE 2010*, pp. 271–276, 2010.

[42] V. Bellotti and K. Edwards, "Intelligibility and accountability: Human considerations in context-aware systems," *Human-Computer Interaction*, vol. 16, no. 2-4, pp. 193–212, 2001.

[43] S. Mennicken, J. Vermeulen, and E. M. Huang, "From today's augmented houses to tomorrow's smart homes: New directions for home automation research," *UbiComp 2014 - Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 105–115, 2014.

[44] J. Lee, L. Garduño, E. Walker, and W. Burleson, "A tangible programming tool for creation of context-aware applications," *UbiComp 2013 - Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 391–400, 2013.

[45] S. Clifford and J. Jerit, "Is there a cost to convenience? an experimental comparison of data quality in laboratory and online studies," *Journal of Experimental Political Science*, vol. 1, no. 2, p. 120–131, 2014.

[46] F. Dandurand, T. R. Shultz, and K. H. Onishi, "Comparing online and lab methods in a problem-solving experiment," *Behavior Research Methods*, vol. 40, pp. 428–434, May 2008.

[47] Sauro Jeff, "MeasuringU: Measuring Usability with the System Usability Scale (SUS)," 2013.