Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Software Engineering and Technology Research Group

Configuration of Ambient Environments by speech

Master Thesis

Nanne Wielinga

Advisor:
Prof. Mark van den Brand

Tutor:
Ir. Anne van Rossum

Examination Committee:
Prof. Mark van den Brand
Dr. Tanir Özçelebi
Dr.Ir. Ion Barosan
Ir. Anne van Rossum

For committee

Eindhoven, May 2018

Abstract

We create a voice interface for a configurational language for ambient environment. Privacy issues are currently a barricade for reaching the ideal ambient intelligence, as people will have to be fully monitored in order to discover their ideally ambient environment. Our solution focuses on empowering users to manually configure their own living environment. By providing a voice interface, users can interactively discover the rules in their environment and change them into the wished behavior.

In a broader perspective, we attempt to create a speech interface for editing a program originating from a domain specific language. A speech interface consists of conversation between the user and the system, which results in an interactive editing experience. Our work is by far not complete enough for realistic usage and there are still several technical challenges such as, there too many variations of sentences that users can speak and many of those variations are currently not accepted. Furthermore, we do not provide a full editor and the expressiveness is limited. Despite these limitations, we do provide a working prototype where a home environment can be configured.

Acknowledgements

First of all, I would like to thank my supervisor Mark van den Brand, who luckily was patient enough to keep supporting me and guiding me with the topics I was interested in. His advice, encouragement, and wisdom especially helped me through this difficult process.

I would also express my thanks to my tutor Anne van Rossum, who leads Crownstone B.V. and has experience with Internet of Things and ambient environments. His perspective and knowledge helped me shaping my work. Thanks to the colleagues in Crownstone for a great working environment and much fun during the lunch time.

Furthermore, I want to express my gratitude to dr. Ana-Maria Sutii, who especially helped me with my seminar and the period leading to my graduation project. My dear friends, Leroy van Zeeland and Mathijs van der Worm who provided feedback and support. Besides, I would like to thank my examination committee members: Prof. Mark van den Brand, Dr. Tanir Özçelebi, Dr.Ir. Ion Barosan, and Ir. Anne van Rossum.

I would especially like to thank my family. My fiancée, Wei Zhang has been extremely supportive of me throughout my entire study and helped me to get to this point. My parents, who always supported and believed in me. My sister, Hester Wielinga who supported me by listening to me. Without their support, I would not have succeeded.

Contents

List of Figures

List of Abbreviations

AmI Ambient Intelligence. 4, 5

CASE Computer-Aided Software Engineering. 13

DSL Domain Specific Language. 12, 14, 21

EMF Eclipse Modeling Framework. 11

GPL General Purpose Language. 12, 14

HTML Hypertext Markup Language. 12

IFTTT If This Then That. 20

IoT Internet of Things. 2

MDA Model-Driven Architecture. 11

MDE Model-driven Engineering. 11

MDSE Model-driven Software Engineering. 11

MOF Meta-Object Facility. 11

MPS Meta Programming System. 13, 14

OMG Object Management Group. 11

SQL Structured Query Language. 12

UML Unified Modeling Language. 11

Chapter 1

Introduction

As the cost of integrated circuits is decreasing, homes and offices start adopting more connected electronics such as lightning, heating, speakers, cameras, fridges, and others. The vision of Ambient Intelligence imagines that these electronics adapt to the user, such that the electronic environment is sensitive and responsive to the user in order to improve people's lives. However, due to privacy issues [12], which we explain in Section 2.1.4, there is currently a barricade for reaching the ideal Ambient Intelligence as the environment cannot adapt to the user without adequate data.

In spite of privacy issues, we still want to create an environment that improves the lives of people. Consequently, we choose to focus on empowering users to manually configure their own living environment. With this method, users can decide themselves which personal information they link and they can choose in which manner the environment adapts to their behavior. In order for humans to use such a configuration system, it is essential that the system is accessible and therefore has a low learning curve.

To build an accessible solution, we focus on creating a configuration language for ambient environments that is configured the voice interfaces. With the recent advancements in speech recognition and text-to-speech, it becomes possible to create more advanced voice interfaces. Successful voice assistants such as Apple Siri show that speech interfaces can be user friendly, as users can perform various tasks such as playing music, setting timers, perform knowledge retrieval, do arithmetic, finding sports results, etcetera. Commercial voice assistants such as Google Home and Amazon Alexa provide developers the possibility to extend upon the skillset of their voice assistants.

The configurational language, that configures these ambient environments, consists of computer instructions and can be seen as a domain-specific language. Domain-specific languages together with model-driven engineering provide abstractions, protocols, standards, design patterns, and tools for creating languages and models in a particular application domain. For ambient environments, it is useful when standards, abstractions, protocols, and especially implementations are shared and commonly used. Environments can be more ambient when there is more compatibility between different electronic devices and control systems. A provider of such standards and implementations is Eclipse SmartHome, which is a open-source framework that together with the automation software of OpenHAB provide more than 200 different technologies [41].

In this thesis, we design a configurational language together with a voice interface that can configure ambient environments. Furthermore, we implement the designed language and voice interface as a prototype in order to try to proof the concept. Besides designing and implementing, we make an attempt to answer the research questions that arise from creating voice interfaces and domain specific language.

1.1 Stakeholder

The main stakeholder of this thesis is Crownstone B.V., which shares our research interest into configuration of ambient environments. Crownstone B.V. is the creator of the 'Crownstone', which is a bluetooth connected smart home device. The Crownstone provides a power switch and power dimmer for electronic appliances. Their smart phone application allows manual switching, scheduled switching, power consumption measuring, switching by presence, and indoor positioning [13]. Crownstone B.V. is in the process of implementing smart home functionality in several voice assistants [54][53] and has an interest in configuration by voice as well. We interviewed Anne van Rossum, who is leads Crownstone B.V. and has experience with Internet of Things (IoT) and ambient environments. With his insights we constructed requirements for a prototype.

1.2 Research questions

As we want users to configure their own environments with a voice interface, there are two research questions that arise.

RQ$_1$: *What would be a suitable domain specific language that controls ambient environments?*

Firstly, we want to investigate solutions for the configuration of ambient environments. The configurational language can be seen as a domain specific language, and which could benefit from the research being done on domain specific languages. To answer this question, we look to other solutions for configuring ambient environments. Furthermore, we provide background information on domain specific languages.

RQ$_2$: *How does a voice interface for a domain specific language looks like?*

Secondly, a hypothesis we have is that voice interfaces are user friendly for configuring ambient environments. We know that voice interfaces are easy for some tasks, but it is still unknown how well voice interfaces perform for more complicated tasks such as configuring an ambient environment. We think that it depends on the voice interface itself. So how does a voice interface that configures a configurational language looks like? And more general, how does a voice interface for a domain specific language looks like?

To answer this question, we design and implement a voice interface and a domain specific language for configuring ambient environments. Our findings from this design and implementation address this research question.

1.3 Contributions

The primary contribution of this thesis is the Domain Specific Language together with the voice interface, for configuration of Ambient Environments. Our proposed language is based on IFTTT [47] and Eclipse SmartHome [18], and we adapted the language to have a voice interface. The voice interface itself contains the ability to add, change, and remove rules from the configuration. As well as having the possibility for the user to inspect certain behavior.

1.4 Thesis organization

In Chapter 2, we provide a background information of Ambient Environments, Voice Assistants, Model-driven engineering, we present related work, and we provide a discussion of the topics. Next, in Chapter 3 we provide an requirement analysis where we provide an overall motivation, interview our stakeholder, provide use cases, present the requirements, provide some discussions,

and discuss our research questions. Afterwards, in Chapter 4 we present the architecture of our voice interface and the configuration language, which we follow up with the design of an ambient environment, the design of our voice interface, and the design and semantics of our configuration language. Furthermore, in Chapter 5 we present our implementation of the prototype, we evaluate our work, and discuss technical challenges that we have found. Finally, in Chapter 6 we conclude our research with a summary of our contributions, a broader discussion about speech interfaces, and suggestions of future work.

Chapter 2

Background and related work

In this chapter, we provide background information on Ambient Environments, Voice Assistants, Model-Driven Engineering, related work, and a discussion of how these topics lead to our design. In Section 2.1, we discuss Ambient intelligence as it has the same aim of improving human lives by making the environments more intelligent. In Section 2.2, we discuss voice assistants as we search for a platform to create a voice interface on. Furthermore, in Section 2.3 we introduce background information on model driven engineering as it is beneficial for understanding domain specific languages. Lastly in Section 2.4, we discuss related work and projects.

2.1 Ambient environments

We discuss Ambient Intelligence because this field has the same aim of improving human lives by creating a more intelligent environments. When the ideal of Ambient Intelligence is successfully implemented, it would make our approach for manual configurable environments unnecessary.

Ambient Intelligence (AmI) is a ideal vision of electronic environments that are sensitive and responsive to the presence of people [19]. The vision is used as a top down method of reasoning about technological science. Technological developments such as lightning, sound, vision, domestic appliances, and personal health care products collectively improve people's lives when integrated. The devices work in a distributed manner where information and intelligence is hidden. The AmI paradigm offers a base for technological innovation for a diverse set of use cases. The notion ambience in Ambient Intelligence refers to a non-obstructive integration of technology into people's lives. Where the notion Intelligence stands for understanding and learning the environments that people live in.

The idea of AmI originated in 1998 and was developed by a team from Palo Alto Ventures, a US management consultant company. The work was commissioned for the board of management of the Philips Company, which was responsible for consumer electronics equipment at the time. The first official publication that mentions Ambient Intelligence was by Aarts and Appelo in 1999 [20]. Later in 2001, the European Commission used the vision for their sixth framework (FP6) in Information, Society, and Technology, with a subsidiary budget of 3.7 billion euros.

One big application of AmI is the home environment, where AmI provides context-awareness and proactiveness to support daily life. Where devices in the home environment, such as lights, heating, curtains, television, smart phones, speakers adjust to support the daily life of a user.

As AmI is a broad domain of different research areas, the future of AmI is dependent on these areas. In the book of Emiele Aarts[19] topics as Smart Materials, Electronic Dust and e-Grains, Computing Platforms, Mobile Computing, Sensory Augmented Computing, and others are discussed. Each of these areas could bring new improvements to the people's lives. However according to the authors, there is a desire for truly spontaneous and smart cooperation in ubiquitous computing. Where the solution needs to be invisible and intuitive in order to become successful.

2.1.1 Examples

In figure 2.1, we present different examples that could be implemented in an Ambient Home Environment. Each example describes a certain activity that users might want. Example (a) is about creating a comfortable temperature in the rooms that have humans inside. Such a scenario needs to acquire knowledge about usage of the rooms, as well as which temperature per room is comfortable in order to create a desired environment. Example (b) is about closing and potentially opening the curtains when it is dark, or possibly against direct sun light. Example (c) is about changing the brightness and color of the light based on the night cycle. When it is becoming dark, a more yellow light is preferred over white light. And before bed time it is good to dim the brightness of the light. Example (d) is about playing some suitable background music based on the mood and time. When a user is working at home, more suitable music should be less distracting. While during the dinner more romantic music could be played.

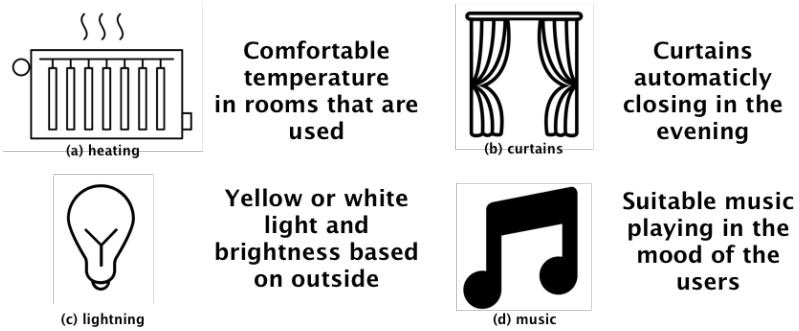


Figure 2.1: Ambient Intelligence Examples

2.1.2 Applying in practice

Current home automation solutions such as Eclipse Smart Home [18] and IFTTT(IF This Then That) [47] can be seen as a primitive form of AmI, as they both allow control of electronic environments and can be configured to be sensitive and responsive to the presence of people. However, the user manually constructs the configuration as the intelligence is missing. In IFTTT, users can create applets which combine two services, one service for triggering the applet such as a weather update and another service as reaction to the trigger such as turning off the lights. For IFTTT there are standard and popular recipes between services, where users can enable skills in a few clicks. However, IFTTT misses features such as copying applets, having programming constructs as if statements, while loops or functions. In Section 2.4 we present more details about IFTTT and Eclipse Smart Home, including their differences.

2.1.3 Interaction between home automation and user

In the ideal situation, interaction with the user is kept to a minimum as the electronic environment anticipates on the users needs. However, as explained in Section 2.1.4, systems do not know enough to fully anticipate what the users want. Therefore there is a need for an interface (or multiple) between the user and the underlying systems, in order for the user to configure the behavior of the system. There are many types of interfaces to choose, from keyboards, touchscreens, speech recognition etcetera [17].

Therefor, we need to choose an interface we use for our prototype. As the home environment can be divided over different rooms and users could be doing different tasks where they cannot use their hands, we believe that speech recognition and voice assistants are an excellent choice. Consequently, we describe voice assistants and speech recognition in Section 2.2.

2.1.4 Privacy issues

The privacy issues are one of the arguments why we propose using a domain specific language, as the user can manually configure an ambient home. In fully ambient environments, there are privacy issues as the user could be exploited by the vendor providing the services [12]. In order for an ambient intelligent environment to learn about the user, training data for the machine learning algorithms has to be collected. Therefore an ambient intelligent system requires data about the user in order to create a fully ambient environment. However, such data could provide private information about the user's life. The vendor collecting such data could potentially exploit and harm the user. As a consequence, society and potential users can be wary about the data being collected by the vendor and such issues can obstruct the creation of a fully ambient environment.

2.2 Voice Assistants

As we want to provide a voice interface for a domain specific language, it is beneficial to discuss voice assistants and speech recognition. The reason why we choose for a voice assistant and not to extend upon speech recognition is that a voice assistant is extendable by other parties as well. In that sense, a voice assistant could be seen as an operating system upon which we build one of the applications. In Section 2.2.1, we explain the definitions and vocabulary related to voice assistants and speech recognition as well as how these concepts relate to each-other. In Section 2.2.2, we give a brief historical overview and possible developments of voice assistants in order to provide more context. In Section 2.2.3, we discuss different forms of extendability of voice assistants. In Section 2.2.4, we briefly share some of the human aspects that need to be taken in consideration when designing voice interfaces. In Section 2.2.5, we compare several arbitrary voice assistants including both commercial and open-source assistants. Finally, we discuss the current state of ambient environments in combination with voice assistants in Section 2.2.6.

2.2.1 Definitions and vocabulary

A voice assistant is equivalent to virtual personal assistance, which communicates with the user via speech. The voice assistant is running on a computer with a microphone and a speaker. A voice assistant is a combination of speech recognition, text-to-speech, and artificial intelligence. Speech recognition is a sub-field of computational linguistics that focusses on recognition and translation of spoken language into text by computers. The fields together rely on a combination of linguistics, computer science, and electrical engineering.

For each part behind the voice assistant we describe the mechanisms in general:

Voice assistant The voice assistant is a virtual personal assistant that communicates with the user via speech. A voice assistant is build on top of three techniques, namely speech recognition, text-to-speech, and dialog systems. All of these three techniques are described below. A dialog system has mostly fixed sentences, which allows the speech recognition to be trained more specific to what the voice assistant supports.

Speech recognition Speech recognition is the translation of audio containing speech to computer parsable sentences. For recognizing speech, a frequently used method is machine learning [9]. Features are extracted from the speech signal, and these features are send to the speech decoder. Training speech data together with training text data, and a lexical model allow the decoder to produce a sentence. Notice that the training text data can be originating from the voice assistant.

Text-to-speech Text to speech is the digital process of translating the text into spoken words. There are two essential components [46], a text analysis system which decodes the text and formulates a structure, together with a speech synthesis system which encodes this structure into speech. For both components there are different techniques used which differ per system.

In general, text-to-speech is supported by the fields linguistic analysis, signal processing, and machine learning.

Dialog system A dialog system is a computer system designed for having conversations with humans. Such a dialog system uses different technologies, including natural language processing, natural language understanding, and natural language generation. Furthermore, techniques such database systems are used in order to quickly provide answers. Speech recognition is trained with sentences that the dialog system supports.

In the development of voice interfaces, a certain vocabulary is commonly used and we introduce the vocabulary below:

Vocabulary

Utterance An **utterance** is a continuous piece of speech beginning and ending with a clear pause.

Intent An **intent** is a collection of utterances that share one intention.

Slot A **slot** is variable that is to be filled in by the user in a speech.

Conversation A **conversation** is an exchange of utterances back and forth between user and assistant, a conversation can be realized as a state machine where intents are actions and in each state different intents are possible.

Skill A **skill** is an ability of the voice assistant that groups certain intents and actions.

Example

To provide some examples of these words, we present an example Smart Home Intent where the user can turn on the lights and possibly at a certain brightness. Example:

1. SmartHome turn the {room} lights on
   (Slot)
   (Utterance)
2. SmartHome lights on
3. SmartHome turn the lights on in {room} at {brightness} percent brightness

2.2.2 Development of voice assistants

The first successful voice assistant on the market was Apple Siri, which was a "very impressive piece of engineering" according to Boris Katz [6]. He describes that the challenge lies into bypassing errors that each of the subsystems make, as the errors of all subsystems are multiplied. A solution that Apple patented is restricting the queries to specific areas, which is similar to creating skills in Google Home and Amazon Alexa as is described in Section 2.2.3. Siri show that speech interfaces can be user friendly, as users can perform various tasks such as playing music, setting timers, knowledge retrieval, arithmetic, finding sports results. As commercial voice assistants collect data from their customers, they can use data analytics to find out which information the customers are missing and manually add this functionality. As a result, creating voice assistants is vast amount of manual labour.

As voice assistants currently rely mostly on speech recognition, text-to-speech, natural language processing, and integration of third party services, both scientific and engineering improvements of these components will improve voice assistants. Furthermore, from the side of Ambient Intelligence the assistants could become more sensitive and responsive to the users. Currently, the assistants seem to only provide simple intent matching based on third party services offered, but for true Ambient Intelligence there is a need for more personalization, contextualization, and cross integration between services. As illustration, when a sports match is coming up from a team that the user likes and the match is nearby the user, the voice assistant could ask if the user would like to book tickets from the match. Despite of such improvements, the privacy issues arriving from Ambient Intelligence [12] as discussed in Section 2.1.4 could also be an concern for voice assistants which could potentially halt or limit the development.

2.2.3 Extendability

We discuss the extendability of voice assistants because in this thesis we are interested in creating new voice interfaces. Note that we discuss different types of extendability in voice assistants. We look at both commercial and open-source assistants, where all open-source assistants are by definition extendable. More information about these existing voice assistants is introduced in Section 2.2.5.

Fixed intents Firstly, we discuss the first variant of extendability where the third-party provides a file to the voice assistant with all accepted intents, utterances, and slots structure as described above in Section 2.2.1. After the intent is recognized in the speech, a web request is send to the third-party which provides a reply. A benefit from this extendability approach is that the speech recognizer can be trained towards possible inputs, as these are defined in advance. The commercial assistants, Google Home, Amazon Alexa, and Apple Siri, support adding functionality via this variant [25][2][5]. Both Google Home and Amazon Alexa have a web-application where the third-party uploads the file containing intents. For Apple Siri, intents are received in an application build by the third-party running on the device of the user. Because Google Home and Amazon Alexa have a similar structure and both send requests to a web-service, it is possible to create cross-platform extensions that support both assistants.

In contrast, while Google Home, Amazon Alexa, and Apple Siri all support adding these intents, there is a difference in how the third-party extensions are invoked. Both Amazon Alexa [4] and Apple Siri [5] are more limited in this aspect as they do not support implicit invocation. Which means that the user first has to ask to talk to the third-party extension before being able to use the third-party intents. Google Home supports implicit invocation [27], which provides invocation of the app without users calling the app by name. There is an exception for Apple Siri, as they provide Intent Domains where the app name does not have to mentioned.

Text processing Secondly, an alternative method of extendability is when there is no intent matching but more use of natural language processing. A speech-to-text engine is used for translating the speech to text and afterwards a matching algorithm is used to find the correct action. A benefit of this approach is more flexibility in the implementation. A disadvantage compared to the first approach is that the quality of speech recognition might be lower, because the speech recognizer can have difficulties with similar sounding words as it cannot predict as the input is not fixed. The commercial assistant Homey and all the open-source assistants we looked at use this approach. Homey provides a more flexible matching with regular expressions and events [7]. Where Mycroft provides a more dictionary and regular expression based approach [38].

2.2.4 Human aspects of voice interfaces

Besides technology, there is a social aspect to conversations and therefore voice assistants. Conversations are not equivalent to mere exchange of information and tend to be illogical and non-mathematical. For instance, a question like “Do you have hobbies?” would logically be a yes or no question, however commonly the expected answer would be an enumeration of hobbies.

In speech often errors occur, whether it is between humans or from human to computer. The speaker might have mispronounced a word or the listener has misheard it. Furthermore, there might be hesitations or interruptions. Alternatively, corrections happen to recover from such errors. However, voice assistants do not yet support such subtleties and is therefore more error-prone. Therefore, it is important that the voice assistant instills user confidence and has strategies for handling errors. Prompts such as "What was that?", "Sorry, what time?" sound more natural then giving a warning or repeating the original question. Furthermore, as the user does not know if the assistant correctly received the command, it is important for the assistant to rephrase and repeat the command.

2.2.5 Comparison of voice assistants

We want to compare the voice assistants we considered for our implementation. In the previous Section 2.2.3, we dive deeper in the extendability differences between the voice assistants. As there are various different voice assistants, we arbitrary selected and compared a small group of voice assistants. In our selection criteria, we did require voice assistant to able to run on a custom hardware. For commercial voice assistants we require that the company offers such hardware.

Although all assistants differentiate widely in features, price, and quality of voice recognition, such a comparison would become out-of-date as many assistants are being improved regularly. Furthermore, comparing assistants would be subjective as the usefulness of functionality would differ per user and the quality of voice recognition differs per user accent. Therefore, we leave out comparisons on features, price, and quality of voice recognition.

We choose to compare the assistants on license, speech recognition engine, and extendability. Considering that voice assistants can always use new functionality and improvement, we mention the date of latest code change for open-source projects. As long gaps in development might indicate that the developers lost interest. We think the software license can be important when considering creating extensions as certain licenses allow or disallow certain applications. We highlight which speech recognition system is used because there could be a possible dependence on commercial solutions. Another reason for highlighting the speech recognition system is that future comparison studies might show which solution has a higher voice recognition quality. Lastly, we briefly compare the extendability of each voice assistant which we describe as well in Section 2.2.3.

In Table 2.1, we present a brief overview of differences in the selected voice assistants. As can be observed, the licensing differs per open-source project and many open-source voice assistants are based on external speech recognition for translating the voice to text.

Furthermore, we provide a small description for each voice assistant below:

Amazon Alexa is a **commercial** voice assistant provided by the company 'Amazon.com, Inc.', where the hardware solution is called Alexa Echo.

Google Home is a **commercial** voice assistant provided by the company 'Google LLC', where the hardware is called Google Home.

Apple Siri is a **commercial** voice assistant provided by the company 'Apple Inc.', where the hardware is called Homepod.

Homey is a **commercial** voice assistant provided by the company 'Athom B.V.', where the hardware is called Homey.

Mycroft is an **open-source** voice assistant made by the Mycroft AI Team. For obtaining funding, they sell a hardware solutions called Mycroft Mark 1. The team is still developing further the capabilities of the assistant, as there is daily activity in their Github repository.

Lucida is an **open-source** voice assistant which is the next generation of Sirus which was made by the Clarifty Lab at the University of Michigan with the last Github change on 7 July 2017.

Jasper is an **open-source** voice assistant made by a small community with the core developers originating from Princeton and Ruhr Uni Bochum. Where the latest Github change was on 27 Jan 2017.

Open Assistant is an **open-source** voice assistant which is maintained by a working group lead by Andrew Vavrek with the last Github change on 31 August 2017.

Adrian is an **open-source** voice assistant by Gergely Hajcsak and Jamie Deakin with the last Github change on 24 March 2017.

Table 2.1: Comparison between voice assistants

Name	License	Speech Recognition	Extendability
Apple Homepod	Commercial	Siri	Fixed intents, explicit invocation
Google Home	Commercial	Google Speech	Fixed intents, implicit invocation
Amazon Alexa	Commercial	Amazon Speech	Fixed intents, explicit invocation
Homey	Commercial	?	Text processing
Mycroft	Apache License	Google Speech	Text processing
Lucida	BSD License	Kaldi	Text processing
Jasper	The MIT License	Various	Text processing
Open Assistant	GNU v3.0	CMUSphinx	Text processing
Adrian	Apache License	Various	Text processing

Choosing a voice assistant to extend upon

As we implement our design of Chapter 4 upon an existing voice assistant, we pick an voice assistant from the above comparison. While text processing might be more powerful, we find fixed intents to be more important as recognizing such intents could be trained in machine learning algorithms. As Google Home and Amazon Alexa have a compatible method of extending, an extension from one could easily be adapted to the other. Google Home has a benefit over Amazon Alexa as it supports implicit invocation. However, as for us an Amazon Alexa Echo was locally available, we choose for Amazon Alexa.

2.2.6 Voice assistants and smart homes

In the combination of voice assistants and control of smart homes, there are several commercial solutions such as Google Home [26], Amazon Alexa [3], and Homey [8]. These voice assistants provide integration and control of smart home devices. The amount of integration with smart home devices differs per platform and these platforms provide developer interfaces for smart home device manufacturers. User commands are limited to basic control of the smart home devices, for example turning lights on and off, configuring temperature, etcetera.

JASPER is an open-source voice assistant that includes a module for home automation [43]. For voice recognition it uses CMUSphinx and supports the commercial voice recognition 'Google Speech' as well. As for speech accuracy, it is mostly dependent on the voice recognition software used. The accuracy CMUSphinx is dependent on the test database size, configuration of the decoder, and the transcribed audio [15][16].

2.3 Model-driven engineering

As the success of ambient intelligence relies on corporation of electronic devices, it would be useful to apply the software development methodologies that increases abstraction and compatibility

between electronic devices. Furthermore, as we design a domain-specific language for ambient environments, it is beneficial to discuss Model-Driven Software Engineering and in extension Domain Specific Languages.

Model-driven Software Engineering (MDSE) is a software development methodology with at the center domain models, which are conceptual models related to a specific topic. The methodology includes trying to encapsulate abstract representations of the knowledge and activities that belong to a certain application domain. One of the goals from MDSE is to increase development speed, this is achieved through automating the generation of runnable code from formal models[56]. Other goals include increasing abstraction, enhancing software quality, reusability, and making export knowledge widely available [56].

In this section, we discuss modeling languages, four-layered architecture of modeling, domain specific languages, workbenches, and editors. With modeling languages we introduce well-known languages and standards used within MDSE, as they are connected to modeling and domain specific languages. This connection we explain with the four-layered architecture of modeling. Afterwards, we introduce and explain in depth what domain specific languages are. Next, we discuss so-called language workbenches which is tooling for defining, reusing, and composing domain specific languages. Finally, we discuss different types of editors for manipulating domain specific languages. Here we also include the link to voice interfaces.

2.3.1 Modeling languages

We discuss modeling languages as they are important in MDSE. Furthermore, domain specific languages could be seen as a textual of modeling.

One of the better known Model-driven Engineering (MDE) initiatives are Object Management Group (OMG), which proposed the Unified Modeling Language (UML) among others which together form the Model-Driven Architecture (MDA) [40]. Where Meta-Object Facility (MOF) is the meta-metamodel of OMG.

UML aims to provide system architects, software engineers, and software developers with tools for analysis, design, and implementation of software-based systems. They provide structural, behavioral, and supplemental modeling where the structural and behavioral parts have semantics. Structural modeling includes class and organizational diagrams. Whereas behavioral modeling includes state machines, activity, and interaction diagrams. Supplemental modeling provides use case, deployment, and information flow diagrams. Together, the ecosystem around UML provides standards, design patterns, tooling with code generation, protocols.

Another well-known initiative is the Eclipse Foundation which is known for Eclipse platform which is a multi-language software development environment [28]. The Eclipse Foundation build the Eclipse Modeling Framework (EMF) which is a modeling framework that uses the tooling from Eclipse Platform [45]. It unifies the programming language Java, XML Schemas, and UML connected to EMF models. Behind EMF models there is the Ecore metamodel, which is a reference implementation of MOF.

2.3.2 Four-layered architecture of modeling

In our efforts to explain MDSE, we discuss the four-layered architecture of modeling as it presents a overall connection between model languages, domain specific languages, and overall abstraction.

In Figure 2.2, we present the four-layered architecture of modeling [39], where each layer conforms to the layer above and the top layer conforms to itself. A metamodel (M2) is the model of a model, and metamodeling is the process of creating these metamodels. If the model encapsulates knowledge about a certain domain, then a metamodel would describe abstract representation of the model. Furthermore, a meta-metamodel (M3) is a model of metamodel that usually conforms to itself. A meta-metamodel is useful for example tooling and verification purposes. As when tools integrate with a meta-metamodel, then all following metamodels will benefit from the integration. Another possibility is transforming from one kind of model to another kind of model.

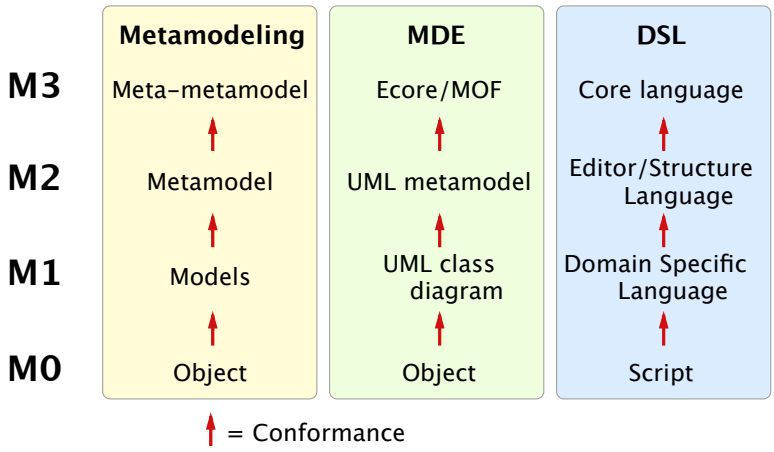


Figure 2.2: Four-layered architecture of modeling

The meta-metamodel is the connection between domain specific languages and modeling languages. Therefore, models created with modeling languages can be referenced to in domain specific languages. As discussed in Section 2.3.4, the Eclipse Modeling framework uses the Ecore as meta-metamodel (M3) and Xtext relies on Ecore for the meta-metamodel.

2.3.3 Domain Specific Languages

As the design of a configurational language for ambient environments can be described as a Domain Specific Language (DSL), we discuss what these languages are, what kind of tooling there is, and what kind of interfaces exist in order to manipulate these languages.

A DSL is a formal, processable language focussed for a specific application domain. There are a vast amount of DSLs [23] both widely used, such as Hypertext Markup Language (HTML) for webpages and Structured Query Language (SQL) for database queries. But as well as more proprietary languages that are for example only used within a certain company.

Using Domain Specific Language could lead to multiple benefits, such as more productivity, higher quality, validation and verification, productive tooling, more domain expert involvement, and more [55]. With many tasks in various domains being performed in General Purpose Language (GPL)s, a task could be replaced by a DSL with a few lines of code. This could lead to more productivity, higher quality, and more domain expert involvement. As there is tooling for domain specific languages, such as the tools we discuss in the next Section 2.3.4, these language workbenches can provide productive tooling together with validation and verification.

Despite of the benefits, there are however also challenges [55]. The effort of building domain specific languages can be a great deal. Furthermore, the languages have to be taught and maintained as well. Besides maintenance, language evolution can also be an issue as all programs become outdated when languages are changed [36]. Another issue is that languages might not serve all the needs of the users demand and the user might be vendor locked, as the user cannot move away to an alternative solution. There are more possible issues described in the book from Völter et al..

2.3.4 Language workbenches

A language workbench, which is a term coined by Fowler[22], is a software development tool in order to define, reuse, and compose DSLs. We discuss language workbenches as they are relevant to model driven engineering and especially domain specific languages. A language workbench provides integration and offers functionalities such as syntax coloring, code completion, static analysis, and more. Language workbenches offer editors to manipulate program of domain specific

languages, we discuss what editors are in Section 2.3.5. Below we introduce several language workbenches:

Xtext Eclipse Modeling Framework offers Xtext [21] to create domain specific languages on top of their platform, with the benefit of their tooling and integration with their modeling as well. Xtext contains a grammar language that is designed for describing textual languages. Xtext uses the EMF Ecore models for describing the structure and can infer Ecore models from a grammar, but can import existing Ecore models as well.

Jetbrains MPS An other workbench for designing domain-specific languages is Jetbrains Meta Programming System (MPS) [32], with the unique feature that it uses projectional editing. Using the open-source Jetbrains IntelliJ IDEA as a foundation to build upon, MPS uses functionality such as auto completion, navigation, error checking, and quick fixes.

ASF+SDF The ASF+SDF Meta-Environment [50] was one of the first tools to be described as a language workbench. Meta-Environment is an interactive development environment which offers syntax-directed editing of ASF+SDF specifications as well as user interface that can provide a graphical view of the specification. Other features include, program analysis, transformation, visualization of parse trees, and a pretty printer generation. Both Rascal [51] and Spoofax [33] are successors based on the ASF+SDF Meta-Environment.

Rascal Offered as an Eclipse-based IDE, Rascal is programming language for source code analysis and transformation [51]. Rascal follows the Extract-Analyse-SYnthesize (EASY) paradigm in order to provide analysis. For example, a programming language such as Java could be extracted, all the classes could be analyzed, and the classes could be synthesized as a diagram. It is also possible to create domain specific language as a custom extraction, where extraction could include the derivation from the abstract syntax tree to a concrete syntax tree.

Spoofax Spoofax is a language workbench build on top of Eclipse [33]. Spoofax provides the creation of domain specific languages on top of SDF, both syntactic as semantic editor services, and transformations including code generation.

MetaEdit+ is a platform-independent graphical language workbench for domain specific modeling [44]. Improving upon Computer-Aided Software Engineering (CASE) tooling, came the more flexible and MetaEdit. MetaEdit is based on the Object-Property-Role-Relationship (OPRR) data model, where MetaEdit+ is based on Graph-Object-Property-Port-Role-Relationship. Where the Graph for example could be a UML Class Diagram. An object is related to the Class or Object from the UML Class Diagram. Relationship relates to Inheritance and Association of the UML Class Diagram. Role describes the relationships of the objects. And ports relates to optional specification of an specific part of an object to which role can connect. A property is a describing characteristic associated with other types, such as a name, an identifier, or a description. Together, the data model can construct languages or diagrams.

2.3.5 Editors

A program from a domain specific language is manipulated by the user in an 'editor'. While the word editor has multiple uses, we refer to the manipulation of programs or models. Often editors are included in language workbenches, but not necessarily as editors could be included into different types of software. As of our second research question is "How does a voice interface for a domain specific language look like?". We first want to look at different style of interfaces for manipulating DSLs that currently exist. We start with textual editing, but continue to interfaces that are more graphical.

Textual There are different editors for changing programs from a DSL [55]. There is the parser-based approach, where the grammar specifies the sequence of tokens and words. The parser is generated from the grammar and recognizes valid programs in their textual form. Regular text editors are capable of editing the programs, however a language workbench such as ASF+DSF Meta Environment [50], Rascal [51], Spoofax [33] or Xtext [21] also provides an editor with autocompletion, syntax coloring, error checking, formatting, and much more.

Projectional Editing An alternative approach is a projectional editor, which works without grammar and parsers. A projectional editor allows the users to directly modify the abstract syntax tree. A workbench offering projectional editing is Jetbrains MPS [32]. MPS can mimic the behavior of a textual editor for textual notations, but also provides a diagram editor for diagrams and a tabular editor for editing tables. Furthermore, the Blockly library provides an editor based on interlocking draggable blocks [24]. Where the output can be code to a language of choice and custom blocks can be created for new applications.

Graphical In MetaEdit+ the graphical editing or rather modeling is at the front of the workbench. A graphical modeling tool allows edit models usually via graphical diagrams. A key functionality is that models created via the graphical editor can be executed, exported to a GPL, or used within other software. Often language workbenches such as, Xtext via Eclipse Modeling, or Jetbrains MPS, have the ability to incorporate graphical editors. The UML Class Diagram is a very familiar modeling language to be edited with graphical editors.

Speech Interface The approach we research in this thesis is editing of programs by speech interfaces. Our goal is to provide an accessible solution for editing programs. In contrast with the other editing solutions, a speech interface can at the moment not be generated based on only the grammar. The interface has to be adapted to the domain.

A speech interface includes conversations, which provide an interactive editing experience. Edit actions can be performed in multiple steps, where the interface remembers the context within the program. The interface also requires the possibility to query the program, in order to add, change or delete parts inside of the program.

There are several limitations to our approach, namely the error-proneness of speech as we described in Section 2.2.4. Furthermore, voice interfaces have a delay in confirmation, whereas in comparison graphical interfaces have a direct confirmation of user input. Nevertheless, in our design and implementation in Chapters 4 and 5 we attempt to create such an interface.

2.4 Related work

In this section, we explore projects and work related to DSL configuration by voice interfaces or ambient environments, or both voice interfaces and ambient environments. In Section 2.4.1, we discuss two studies that focus on voice interfaces for controlling environments. These studies are relevant as they attempt to translate speech into their domain specific language. Afterwards, in Section 2.4.2 we discuss various projects and studies related to configuring ambient environments. Such projects and studies provide us more domain knowledge about the configuration of ambient environments, which is our first research question. As two of these projects, namely IFTT and Eclipse Smart Home, are referenced more often in this thesis we provide a comparison in Section 2.4.3.

2.4.1 Controlling environments with natural language interfaces

For answering our second research question, about how a voice interface for a DSL looks like. Two studies [37][11] utilize natural language processing and machine learning in order to translate human language into a sequence of instructions for environment alteration. The human language can be both in written form or originating from speech and being converted to text.

The first study "Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions" from Misra et al., interprets natural language commands and transforms them to a sequence of mobile manipulation tasks which is executed by a personal household robot.

The second study "Reading Between the Lines: Learning to Map High-level Instructions to Commands" from Branavan et al., maps high-level instructions to low-level commands in an external environment. They apply this method of mapping to GUI actions in the Windows application, but applying this method to other domains is possible as well.

Both studies translate natural language to code unrelated to ambient environments, however the methods used could be adopted by using similar natural language processing and machine learning techniques. A problem is that their methods are currently limited to instructions, where the requirements of the ambient environment require more event based programs. Furthermore, the performance of these studies is limited by the quality of their machine learning algorithms.

2.4.2 Projects or work related to configuring ambient environments

Below we discuss several projects and studies for our first research question, what would be a DSL for controlling ambient environments.

Eclipse Smart Home The Eclipse IOT team created a open-source framework that is a development environment for smart home solutions [18]. The Eclipse Smart Home provides a pluggable architecture and a custom domain specific language together with a development environment to program the smart home. This environment is aimed for a more technical end user as you have to download and learn a domain specific language. However, besides the textual configuration they also provide different graphical interfaces that does allow the less technical end-user to use it. Eclipse Smart Home as a framework provides the standards, the development environment with a domain specific language, higher-level services, extension points, etcetera. Eclipse Smart Home is the underlying framework for OpenHAB and other solutions [18], such as Qivicon which is a commercial solution by Deutsche Telekom AG.

OpenHAB, which stands for Open Home Automation Bus, is a home automation platform. OpenHAB focuses on providing integrations with over 200 different technologies [41], such as lights, alarms, electricity meters, and more. Furthermore, OpenHAB provides custom graphical interfaces for administration and control, where users can configure their integrations, create new rules, and manage the state of their devices.

IFTTT IFTTT stands for IF This Then That [47], and is a free but closed-source web-based service with a vast number of integration which consists of both smart home devices and internet services. It is based on a trigger-action programming model, where users can connect two services. One service provides the trigger, which could be time-based, a weather based condition or an incoming e-mail. The other service is the action, which could be turning on a light, making a coffee, or playing a song. Besides that IFTTT is it not open-source and the business model remains undisclosed, the programming model is very limited as we point out below in the comparison. Despite the limited programming model, the usage numbers [35] and evaluation of model in online study [49] shows that the interface is easy for users.

Ontology based An ontology is a formal naming and definitions of types, properties, and interrelationships of entities. The ontology work we discuss have a focus on smart home solutions. We discuss such ontologies as they could be used for design of smart home solutions. To provide an example of an ontology, we present the class hierarchy diagram from [29] in figure 2.3. Academically there have been made multiple attempts to create an ontology or language for creating a common standard of smart home programming [1][30][52][57][58][34][42]. Most of these attempts are based on Web Ontology Language called OWL [30][52][57][34][42], which is a family of knowledge representation languages which allows for both modeling and deducing facts. Furthermore, many of these attempts also created working prototypes that use smart home devices and sensor [1][52][57][58][42]. While others promised to make prototypes [30][34].

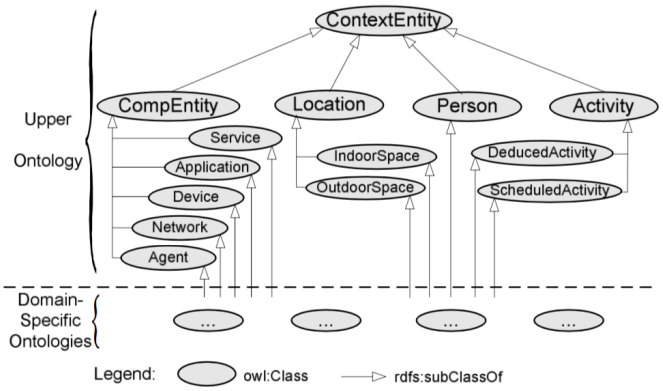


Figure 2.3: Example of an ontology diagrammed as the class hierarchy diagram from [29]

CAMP In an effort to create more natural programming environments, Truong et al. introduced a magnetic poetry interface for end-user programming of home applications [48]. Their interface is inspired by magnetic poetry, draggable magnetic words that stick on a fridge, and used it to extract programs from the combined sentences. With phrases such as "beginning at 3 P.M. for 2 hours", they could inspire users to create creative programs. While ideas are inspirational, the paper does not disclose details about converting the poetry to working programs.

2.4.3 Comparison between IFTTT and Eclipse Smart Home

As we reference IFTTT and Eclipse Smart Home more often in this thesis, we compare these two automation platforms.

License Eclipse Smart Home is open-source, while IFTTT is a freeware web-based service.

Integrations OpenHAB claims to have "over 200 integrations" with technologies, while IFTTT claims to have over "over 360 different partners". Other Eclipse Smart Home partners have their own integrations. However, IFTTT provides many integrations unrelated to ambient environments, for instance Skype which is a telecommunications application software.

Expressiveness The underlying concepts of IFTTT are more limited than Eclipse Smart Home as for instance IFTTT only provides one action after a trigger while Eclipse Smart Home provides multiple. Furthermore, Eclipse Smart Home provides more programming constructs such as if statements and conversions of data values. Where IFTTT only provides variables.

Ease of use With IFTTT, integrations and installation is relatively easy as many of their partners provide the possibility to integrate with one click. With OpenHAB, extra configuration is needed for each integration, for instance the IP address and possibly other settings. Other partners using Eclipse Smart Home can provide their own graphical interfaces and configuration methods. OpenHAB provides several different graphical interfaces as-well.

2.5 Discussion

We described the essentials of ambient environments, voice assistants, model-driven engineering, and related work in order to construct a basis for our research questions. We continue with a short discussion on ambient environments, voice assistants, and configuration.

Modeling ambient environments

Devices in AmI environments are controlled by microprocessors and in order to achieve success in ambient intelligence, these processors need to be driven by some kind of configuration. In addition, in practice systems often lack the knowledge to fully anticipate on the users wishes. Therefor, it would be valuable if the user could easily change the configuration by themselves. As low level languages would be to cumbersome to work for humans with when controlling many computers, a higher level language would be required. Both IFTTT and Eclipse Smart Home provide a sufficient model for configuring an ambient environment, as they both have integrations for many ambient environment.

Voice assistants and ambient environments

Current voice assistants, such as Google Home or Alexa, lack control over ambient environment configuration, but only provide changing the current state. For example turning on or off a light, but not configuring at which time the light should turn off. Furthermore, IFTTT and Eclipse Smart Home, are not easily manageable in the ambient environment as a computer or mobile device is required to inspect and alter the configuration. Therefore we can conclude that a solution where users can configure an ambient environment without voice is currently missing.

Configuration by speech

Configuration of ambient environments by voice interfaces provide their own challenges. In Section 2.2.4, we found that voice interfaces are error-prone and that conversations are more illogical and non-mathematical. Furthermore in Section 2.2.3, we discussed that adopting the more limited approach results in a higher quality voice recognition. All these challenges together sound severe when comparing speech interfaces to regular DSL editing on a computer.

Chapter 3

Requirement analysis

To answer our research questions, "What would be a suitable domain specific language that controls ambient environments?" and "How does a voice interface for a domain specific language looks like?", we designed and implemented a voice interface and a domain specific language for configuring ambient environments. In this chapter, we provide an analysis of all topics that lead to our requirements.

Our analysis starts with an overall motivation in Section 3.1. Afterwards in Section 3.2, we discuss the relation with our stakeholder and we present their insights that led to the requirements. As we need to focus on a specific ambient environment, in Section 3.3 we introduce and discuss the by us imagined use cases. Afterwards, we make some observations on other home automation platforms in Section 3.4. We provide the scope of our prototype in Section 3.5. Next, we introduce the requirements in Section 3.6. Besides requirements, we provide some design decisions in Section 3.7. Lastly, we provide a discussion where we return to our research questions in Section 3.8.

3.1 Rationale

Our research questions arise from the interest in providing users a voice interface that can configure their ambient environments. Users can decide in which manner the environment adapts to their behavior, in order to make the environment more ambient.

The reason why we suggest using a speech interface, is that these interfaces do not require a computer or mobile device and result in a more accessible environment. Letting users inspect and change configurations by speech could make ambient environments more approachable, compared to graphical-based configuration with existing tools. Furthermore, current voice assistants are not powerful enough to configure ambient environments, as discussed in Section 2.2.6 they can merely change the state of an environment. By allowing users to configure more abstract rules, we make voice assistants more powerful.

3.2 Stakeholder

Our main stakeholder is the company Crownstone B.V. which is a subsidiary company from research and development company Almende B.V. Crownstone B.V. is the creator of Crownstone, which is bluetooth connected smart home appliance. 'Crownstone' is the literal translation of the Dutch word 'kroonsteen' which in English would be understood as screw terminal. The main functionality consists of switching and dimming electrical circuits in ambient environments. The device has the following capabilities: switching devices of 16 Ampere or lower, dimming LEDs or incandescent light bulbs, measuring electricity usage, and tracking bluetooth devices [14]. Crownstone is sold both as a plug and a built-in variant, where the former can be plugged in a socket and the latter can be used within the wiring of a building.

In the context of our research, Crownstone B.V. is mainly interested into research effort related to ambient environments. Crownstone B.V. is in the process of implementing smart home functionality in several voice assistants [54][53] and has an interest in configuration by voice.

We interviewed Anne van Rossum, who leads Crownstone and has experience with Internet of Things and ambient environments. With the following insights from him we constructed the requirements for our prototype:

- The work should be accessible for newcomers, as a difficult solution would scare users away. This leads to requirement **REQ 3** and **REQ 9**.
- The users should not be bothered with details, they for instance do not have to know about technicalities from hardware. This leads to requirement **REQ 4**.
- That the solution is usable by only speech, as voice assistants are taking over in the home environments. This leads to requirement **REQ 8**.
- Ideally, Anne wants the home environment to have full ambient intelligence and suggested self-learning elements. We concluded that ambient intelligence still have privacy difficulties and we therefore focus on manual configuration and support of ambient environments which leads to requirement **REQ 2**.
- The ambient environment should be inspectable for users who do not know what is happening. This leads to requirement **REQ 7**.

3.3 Use cases

We describe imaginary common use cases that are desired to supported in an ambient environment. Such use cases help us analyzing which functionality we should have in our prototype and configuration language.

- After the sun is down, turn on the lights

The above scenario requests that after the sun goes down, the lights should turn on. **Issues:** There are several issues with above sentence, 1: it is undefined when the lights should turn off, 2: it is undefined which lights should turn on. **Complexity:** There is a condition and a trigger, similar to structures in IFTTT. The condition is based on the location of the user, and the astronomy of the sun. Furthermore, the trigger indicates changing the state of the lights.

- Everyday from 7 am to 9 pm, turn on the air conditioner.

In the above scenario, the air conditioner should turn on between 7 am and should turn off at 9 pm. **Issues:** Again unspecified is which air conditioner should turn on. **Complexity:** Again we meet a condition, trigger situation similar to IFTTT. The condition is based on time and as well on date, as the condition should trigger everyday of the week and not only once. Furthermore, the air conditioner should turn off as well.

- Once someone is in the kitchen, turn on the kitchen light

In the above scenario, the light of the kitchen should turn on once someone is in the kitchen and turn off once someone leaves the kitchen. **Issues:** It is unspecified what happens once someone leaves the kitchen. If the light turns off directly it might be too abrupt, while staying on too long would seem like the system is broken. A timeout would in general be expected. **Complexity:** Again we meet a condition, trigger situation similar to IFTTT. However, we should add a delay to once the light turns off. The condition is based on activity reported by a detection sensor. Furthermore, the light turns on when detection is seen and the light turns off later.

Earlier in Section 2.1.1, we presented examples of the Ambient Environments. These scenarios are more idealized as they also illustrate ambient intelligence. Namely the following examples:

- "Comfortable temperature in rooms that are used" is about creating a comfortable temperature in the rooms that have humans inside. **Issues:** A 'comfortable' temperature is subjective and besides heating the temperature depends on outside temperature, insulation, and other factors. **Complexity:** The use case contains the condition where heating is only activated in used rooms. Furthermore, there is a need of a feedback mechanism from the users in order to make the temperature comfortable. As well as a thermostat that can control the heating per individual room.
- "Curtains that automatically close in the evening" is about closing and potentially opening the curtains when it is dark, or possibly against direct sun light. **Issues:** It is unspecified if the curtains should open in the morning. **Complexity:** Based on whether it is evening or perhaps based on the sun set, the curtains should close and open in the morning.
- "Yellow or white light and brightness based on the outside conditions" is about changing the brightness and color of the light based on the night cycle. **Issues:** We assume that during the night the light becomes more yellow, while during the day the light is more white. Furthermore, the weather might be more or less light based on fog and clouds. **Complexity:** This use case can be realized with either weather information and astronomical sunset information, or with multiple light sensors. Afterwards, based on the input variables a decision is made that influences the light color and brightness.
- "Suitable music playing in the mood of the users" is about playing some suitable background music based on the mood and time. **Issues:** There are multiple open questions: which music is suitable? should the music also play when there are no users? What if there are multiple users in the room with different moods? How do we find out which moods users are in? **Complexity:** A simple implementation could be playing certain music playlists when users are present. It is possible to speculate in which mood people are based on weather and activity analysis by machine learning.

3.4 Observations from home automation platforms

We make several observations from existing home automation platforms that help our configuration language design. We examine both IFTTT and Eclipse Smart Home, as these platforms are both substantially used for configuring ambient environment.

IFTTT

As described in Section 2.4, If This Then That (IFTTT) provides a simple to use programming model for connecting services. IFTTT offers a trigger-action programming model together with so-called recipes. The programming model seems well suited for Ambient Environments, as actions often happen after reading input from sensors. Furthermore, the recipes in IFTTT bring preconfigured rules that can be quickly activated by the user. In the voice interface, this could be copied by providing intents that create preconfigured rules, such as "Wake to the colors of roses on Valentine's Day!" [31] while the rule behind it would be "If today is February 14, turn bedroom lights RGB(255, 0, 127)".

To see if IFTTT would be suited as a model, we check the available services and triggers and compare those to our use cases from Section 3.3. In our use case section we mention lights, air conditioning, kitchen light, heating, curtains, color of lights, and music. IFTTT provides integration with for instance Philips Hue, Nest, Spotify, Samsung Room Air Conditioner, although we could not find connected curtains at the time of writing. There are other services, such as garage doors, refrigerators, dishwashers, robot vacuum, and more. Despite of the connected services, as explained before in Section 2.4.3 there is no support for logical expressions or other programming constructs. If a user wants to combine two conditions, for example temperature and time then it would not be possible.

We can conclude the following points:

- The programming model of IFTTT seems well suited for Ambient Environments.
- The recipes from IFTTT are a great method for adding accessible but advanced functionality to the interface.

Eclipse SmartHome and OpenHAB

As described in Section 2.4, Eclipse SmartHome together with OpenHAB provide an open-source vendor and technology agnostic automation software, with over 200 integrations with domestic appliances and such. Eclipse SmartHome provides abstractions among the integrations, such as the following appliance categories: Blinds, Car, CleaningRobot, Door, FrontDoor, GarageDoor, LightBulb, Speaker, Window, and many more. These appliances also have properties, while they are abstracted the actual properties may differ of appliances within the same category. For instance, with two different lights, one light might support different colors while the other is only white. There are however certain assumptions that could be made in order to identify and use certain appliances, for example a light might contain certain capabilities like On/Off, Brightness, Color, or Temperature.

Because software from Eclipse SmartHome and OpenHAB is open-source and has a foundational architecture, it is possible to extract either individual integrations or the domain specific language, or both. Extracting all the light appliance integrations could make our prototype functional in practice, similarly if we would translate our domain specific language to the rule system of OpenHAB.

We can conclude the following points:

- It is possible to translate our domain specific language to the rule system of OpenHAB
- It is possible to extract possible appliance integrations and make our functional in practice

3.5 Scope

There are many aspects to configuring an ambient environment, in order to realize a functional prototype we restrict the scope. Below we present two limitations of our scope:

No integrations Integrating home appliances is a part of making the environment more ambient, this normally includes connecting the system to home appliances and naming them. For example, a Philips Hue light has a bridge with an IP address. In order to control a light, the light has to be connected to the bridge, the bridge has to be connected to the system, and the light needs to be categorized in a room. In our prototype we assume all the appliances are already connected.

Simplified Home appliances have many properties that could be configured, in our prototype we have simplified versions of lights, heating, and air conditioning. For example, heating and air conditioning can be configured on a certain temperature which we do not include.

3.6 Requirements

Below we list the requirements that are either necessary or desired in order to construct the DSL and voice interface. We split up the requirements into two sets of requirements, one set for the domain specific language and another set for the voice interface.

Domain Specific Language

We build a new domain specific language because a speech interface is very different from a graphical interface. In addition, the configurational language is created from scratch in order to identify which properties are needed to create an usable DSL that supports a speech interface.

We have the fundamental requirements **REQ 1** and **REQ 2** that are needed in order to have a voice interface for a configurational language of ambient environments. Requirements **REQ 4** and **REQ 5** are about the desired characteristics for the configuration language. Where requirement **REQ 3** is about making the language more accessible.

REQ 1 Support Speech Interface. The configurational language should be suitable for voice interfaces. **To verify:** 1: all voice utterances that we define in the design have a corresponding conceptual construct in the configuration language. 2: For all possible conceptual constructs from the configuration language, there exists an utterance where the user can create, add, or delete such constructs from the programs.

REQ 2 Support Ambient Environments. The language should be powerful enough for supporting ambient environments, especially the home environment. **To verify:** We require that all use cases described in Section 3.3 have corresponding constructs in the configurational language.

REQ 3 Human understandable. The language should be readable and understandable by a human. **To verify:** We require that people without programming experience could construct a program using a graphical editor.

REQ 4 Higher level. The language should be at a higher level of abstract with respect to hardware and domestic appliances. **To verify:** we require that the language does not contain any details about execution.

REQ 5 In-between language. The language should be between the voice interface and smart home environment, in the sense that the configuration language does not depend on either the voice interface or smart home environment. **To verify:** 1: Programs from the configuration language can be constructed in a graphical editor and be executed in a prototype environment. 2: The voice interface can construct programs that are presented in a graphical editor without being executed. Having such a construction allows separate implementations of voice interfaces or smart home environments.

Speech Interface

The voice interface has to be able to manipulate the configuration language. Our voice interface is build from scratch and using the requirements below. These requirements are constructed in order to have a functional and desired voice interface.

The requirements **REQ 6** and **REQ 8** are essential to fulfill the premise of configuring an ambient environment by a speech interface. While requirement **REQ 7** provides more control over the environment. Furthermore, requirement **REQ 8** is to keep the voice interface accessible.

REQ 6 Add, edit, and delete rules. It is possible for end-user to create new, edit, and delete rules. **To verify:** There exists a combination of one or more utterances such that the user can create new a rule, edit a rule, and delete a rule.

REQ 7 Examination. End users can question behavior in the environment and intervene the corresponding rules. **To verify:** For each possible state of all actuators, there is an utterance formed as a question. After speaking such a question, the system responds with a suiting explanation. Lastly, there exists a combination of one or more utterances such that the user can edit, and delete the corresponding rule.

REQ 8 Speech only. A phone or laptop should not be required in order to use the voice interface. **To verify:** we require that the voice interface can be used without laptop or phone.

REQ 9 No programmer experience. We want the user to be able to use the solution with having a background in developing software. **To verify:** in the user manual, there are no mentions made of programming concepts.

3.7 Design decisions

We make several design decisions that shape the design of our configurational language and voice interface. These decisions are the result of limitations we found during the design.

3.7.1 Configurational language

Inside the configuration language, we want to avoid ambiguity and it has to support a speech interface because of requirement **REQ 1**. As a result, we introduce the following decision on logical operations:

Simplification of conditions. Although we want to have logical operations, there are two limitations in speech. We disallow double negations because "not not" is confusing. Furthermore, there is an ambiguity with conjunctions or disjunctions together with negations as "not (A or B)" sounds the same as "(not A) or B". Therefore, we interpret the condition as having the negation on the inside of the conjunction or disjunction. This influences the design of the language as there is a restriction of how the language is build up.

3.7.2 Speech Interface

Within the speech interface, we made two decisions that shaped the design. The first decision decides how we design the voice interface and the second decision decides how we fill in details.

No intelligence in voice interface. As commercial voice assistants does not provide any natural language processing or other artificial intelligence in their extension methods, we therefore only work with intentions, utterances, and slots. The consequence is that our interface loses some expressiveness as the user cannot use complex sentences. This influences the design, as we do not have the flexibility of accepting certain sentences under certain conditions because we have to pre-upload the intentions. Furthermore, we cannot apply natural language processing in order to understand sentences that are not defined yet. Especially for longer sentences or sentences that contains words we did not specify it would become a problem.

Educated guess or asking. As users do not provide all the details in the voice interface, we make assumptions on actual intention or ask further. For example, the user asks for the lights to be switched off but does not provide details on which light that needs to be switched off. An educated guess could be made based on time, location of user, or history. If making an educated guess fails, then the details could be inquired. This influence the design of how the conversations work, we choose to make guesses in order to reduce the conversation length.

3.8 Discussion

In this chapter, we introduced our motivation, the stakeholder, use cases, observations from other platforms, requirements, and some design decisions.

To return back to our research questions:

- "What would be a suitable domain specific language that controls ambient environments?"

We conclude that both OpenHAB and IFTTT are capable of controlling an ambient environment. IFTTT provides a trigger-action model together with recipes. Most use cases we described could be integrated into IFTTT or Eclipse SmartHome. Where Eclipse SmartHome has the benefit of more expressive rules while IFTTT is easier to configure as they do not require coding.

- "How does a voice interface for a domain specific language look like?".

We introduced requirements for the speech interface that are about the functionality it should have and that it is easy to learn. More specifically, we require the ability to add, edit, and delete rules. Together with the ability to let the users examine the created programs.

Furthermore, we made two decisions based on the speech interface. Namely that we do not use natural language processing or other artificial intelligence for extending voice interfaces. While this decreases the flexibility we have, it potentially increases the performance of the machine learning.

Chapter 4

Design

In this chapter, we introduce the design of our proposed solution. In figure 4.1, we present an conceptual overview of our designed software. The diagram is split into two halves, namely the voice interface and the configuration our ambient environment. Below we describe each half in more detail and we explain the connections inside of our architecture.

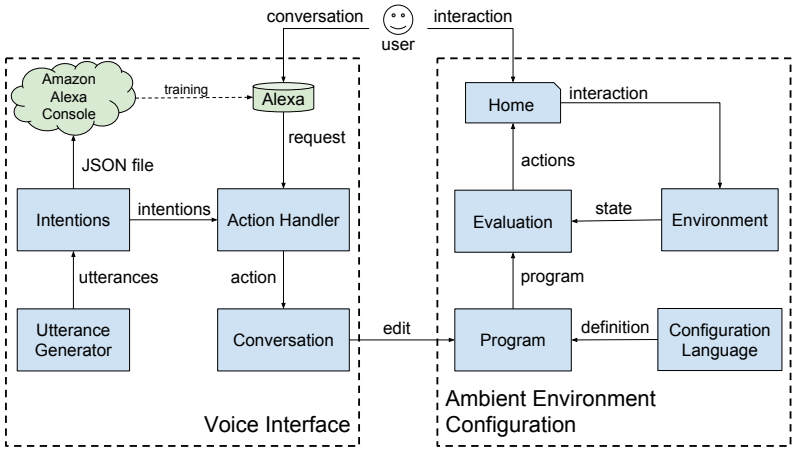


Figure 4.1: Conceptual architecture

Voice Interface In the left half of figure 4.1, we present our software design in order to realize the voice interface which is an extension on top of the voice assistant Amazon Alexa. For more background information about voice assistants, we introduced extending voice assistants in Section 2.2.3, and more about voice assistants in general in Section 2.2.

Extending interface of Amazon Alexa As can be observed, the user has a conversation with the voice assistant from Amazon called Alexa. In order to build a voice interface on top of Amazon Alexa, we need to provide a file with all accepted intentions, utterances, and slots. We generate utterances that together are grouped into intentions, this corresponds to the 'Utterance Generator' and the 'Intentions' boxes from the diagram. We describe the process of generating utterance more detailed in Section 4.2.

Handing requests from Alexa Once both Amazon Alexa received and trained on all the utterances and intentions, it is possible for the skill to be published and used by the user. When the user speaks the correct utterances towards Amazon Alexa, a request from Alexa is send to our prototype which is handled by the 'Action Handler'. The 'Action Handler' together with 'Conversation' is responsible for handling the requests

and providing responses. The 'Action Handler' focusses on filling empty slots and matching to corresponding concepts of the configurational language. For example, it could translate to an action of "Turn on the light in the sleeping room". In Section 4.2, we discuss the action handlers and conversations in more detail.

Ambient Environment Configuration In the right half of figure 4.1, we present our software design in order to configure our ambient environment.

Configuration Language and Program The Configuration Language is the definition for all the programs written in the prototype. A program is used for managing the specific ambient environment from the user. The configuration language is used for the graphical editor, as well as for the design of the object-orientated implementation. As we have a graphical editor together with a speech interface, we define our configuration language as class diagrams in Section 4.3.

Evaluation, environment, and home The created programs are evaluated as well, our design includes an evaluation component together with a virtual interactive home. As a consequence, the 'Evaluation'-component sends actions to the virtual home, while the virtual home sends back interaction information back to the 'Environment'-collector. An example of action is enabling a light and an example of interaction is an user moving within rooms. In Section 4.4, we introduce the evaluation together with the semantics of the configuration language.

4.1 Smart Home, the Ambient Environment

In this section, we introduce the ambient environment that both the speech interface and configuration language support. We based our ambient environment on the use-cased we provided in Section 3.3.

We follow the action-trigger model where based on a condition, we follow with an action in the environment. Furthermore, we adopted the recipes from IFTTT and provide some pre-configurations.

4.1.1 Home

In figure 4.2, we present a floor plan from the ambient environment of our prototype. It contains eight rooms and a balcony.

4.1.2 Actuators

To make our system alive, we require domestic appliances that can be controlled. Below we list the appliances that we support in our prototype.

Lights our ambient environment offers one or more light bulbs in each room. A light can be turned on or off and has a configurable color.

Heating our ambient environment offers radiators in a selection of the rooms. Per room, the heating can be turned on or off. The heater does not contain any temperature settings as of our scope limitations.

Air Conditioning our ambient environment offers air conditioning in a selection of the rooms. Likewise as the heating, the air conditioning can be turned on or off. The air conditioner also does not have any additional settings.

Music our ambient environment offers music in a selection of the rooms. The music can be turned on or off and a specific genre could be chosen.

Curtains our ambient environment offers curtains in the rooms with windows. The curtains can be closed or opened.

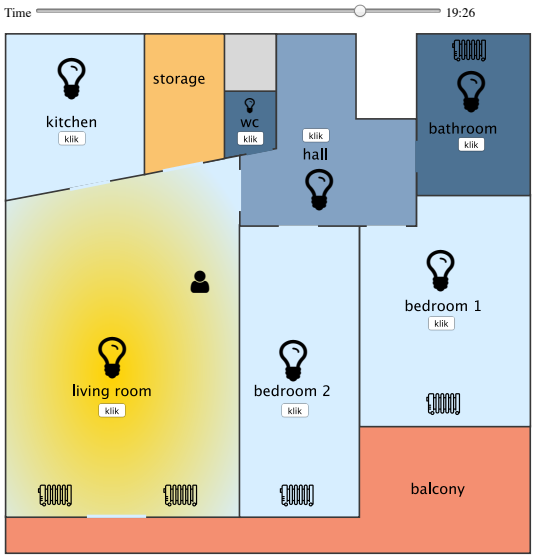


Figure 4.2: Floor plan ambient environment

4.1.3 Conditions

We categorize our conditions into three groups: logical condition, physical conditions, and temporal conditions. Firstly, the logical conditions allow the configuration language to be more expressive as multiple conditions can be linked. Secondly, the physical conditions are based on the ambient environment and grouped per room, which is required to make the environment ambient. Lastly, the temporal conditions allow actions happen relative to time, which is also required to make the environment more ambient.

These are the conditions we support:

Logical Logical conditions allow the configuration language to be more expressive as multiple conditions can be linked.

- **And** is a conjunction that contains two other conditions, whom both have to be true in order for the 'And' to be true. This is useful for example combining a physical condition with a temporal condition.
- **Or** is a disjunction that contains two other conditions, where either has to be true in order for the 'Or' to be true. This is useful for example combining conditions based on multiple rooms or conditions.
- **Not** is a negation that contains one other condition, where if this condition is true the 'Not' is false and vice versa. This makes the language more expressive by for example requiring that nobody is at home.

Physical Physical conditions are based on the ambient environment and grouped per room.

- **Activity** is a condition based on whether someone is inside a certain room. For example, if someone is in the living room.
- **Button** is a condition based on whether a button inside of a room is toggled. For example, a button could toggle a light.
- **Temperature** is a condition based on temperature, it could activate or deactivate based on whether it is a certain temperature.

Temporal Temporal condition are based on time and makes the environment more ambient.

Time contains a start and an end time. If the current time is within these two times, then true is returned. Otherwise false is returned.

Date contains a day and an month. If the current date has the same day and month, then true is returned. Otherwise false is returned.

DateRange contains two dates. If the current date is on or later date 1 and before or on date 2, then true is returned. Otherwise false is returned.

Sun is based on whether the sunset or sunrise, and will only trigger once per day.

Day contains a day of the week. If the current day of the week is that day, then true is returned. Otherwise false is returned.

From contains a time. If the current time is after the configured time then the condition returns true. Between 0 am and the configured time, the condition returns false.

Now condition will always return true.

4.1.4 Recipes

Below we describe recipes that are essentially preconfigured rules. Once the user speaks the following utterances, the corresponding rule is added. In future work more recipes could be added.

Romance "Wake me up to the colors of roses on Valentine's Day!", this recipe adds a rule with a date condition on 14 February and changes the lights to pink.

ComfyTemperature "Set all the rooms at a comfortable temperature", this recipe adds a rule that turns on the heating for all rooms.

CloseCurtains "Close all the curtains automatically in the evening", this recipe adds a rule that closes the curtains everyday at 18 pm.

4.2 Voice Interface

We provide a voice interface to manipulate and intervene the configuration of the ambient environment. Our voice interface is extended upon an existing voice assistant as our project complements a voice assistant that already contains other functionality such as setting timers, sending and reading messages, weather information, etcetera. As discussed in Section 2.2.5, we have chosen to extend upon Amazon Alexa as our voice assistant. Amazon Alexa accepts a JSON file with all intentions and the corresponding utterances. Once Amazon Alexa matches an intention of ours, it sends a HTTP-request to our web service which in return performs actions and provides a response. In Section 4.2.1, we discuss how we designed and generated our intentions and utterances. In Section 4.2.2, we discuss how we handle requests from Amazon Alexa.

4.2.1 Creating the voice interface

To create a voice interface around our configuration language, we need to comply with the requirements **REQ 6** and **REQ 7**. The former requirement wants that we allow the end-user to create new, edit, and delete rules. The latter requirement wants that we allow the end-user to question and intervene the rules. To edit or delete a rule, the user needs to refer a rule which can be done with the examination feature. In fact, intervening is equivalent with editing or deleting rules.

A rule consists of a condition and zero or multiple actions. Therefore we need utterances for conditions and actions. Furthermore, we need utterances for questioning, editing and deleting. With editing rules we re-use utterances from the actions and conditions and we rely on the context, we explain how our prototype relies on context in Section 4.2.2. Below we present some example utterances, how we generate utterances, and a limitation we found.

Examples

Below we provide examples of generated utterances. A comprehensive list of all our ungenerated utterances can be found in Appendix A.

Example Actions Turn on the lights in the living room. / Make the lights blue. / Turn off the radiator.

Example Conditions Once there is someone in the kitchen. / The time is between 8 pm and 4 am. / Someone clicked the button. / **And** once someone is in the kitchen / **Or** when the sun rises / When there is nobody home.

Example Why questions Why are the lights on in the living room? / Why is it so warm?

Example Recipes Wake me up to the colors of roses on Valentines Day / Set all the rooms at a comfortable temperature

Example Delete Remove that rule. / Stop doing that. / Delete last rule.

Utterance generation

Utterances are generated as there is no intelligent natural language processing and the more sentences we generate, the more sentences can be recognized. We use a form of regular expressions where we have concatenation, alternation, and optionality. A sentence like "Why (is|are) (the)? (lights|light|lamp) (on|not off) (in {room})?" will expand to 48 different sentences with a optional room slot. Clearly as can be seen the regular expressions allow us to be more concise.

Limitation

Originally we planned to combine actions and conditions in one sentence, like "If it is 8 pm, then open lights". But the amount of combinations became too excessive and therefore unfeasible to upload. We instruct the user to speak the rule separately as a condition and an action. Furthermore, combining multiple conditions in one utterance becomes unfeasible as well.

4.2.2 Handing requests from Alexa

We implement conversation in order to provide the user the ability to add, edit, and delete rules. Additionally, adding conversations allows the user to create new rules in multiple steps and to correct previous speech errors, together with the possibility to edit or delete a rule. Our implementation needs to remember where the conversation is, therefor we maintain a memory. The memory remembers zero or one **condition**, zero or one **rule**, and zero or one **room**.

Below we enumerate for each intention the action we perform and what we remember for the purpose of the conversation.

Action After the user speaks an action, either the following is true:

- **We have a rule in memory** We add the action to the existing rule.
- **No rule in memory** We directly create a new rule with a possible remembered condition or a new Now-condition. Afterwards, this new **rule** is remembered.

Condition After the user speaks a condition, either the following is true:

- **We have a rule in memory** We change the condition of the rule into the new received condition and we confirm to the user.
- **No rule in memory** We remember the **condition** in memory and suggest to the user to add a rule.

AndCondition After the user speaks an and condition, we retrieve a new condition in combination with the fact that the condition be a conjunction of the old and new condition. Therefore we require a rule is already in memory so that we can put the old condition on the left side of the new conjunction and the new condition in the right side.

OrCondition After the user speaks an or condition, we retrieve a new condition in combination with the fact that the condition be a disjunction of the old and new condition. Therefore we require a rule is already in memory so that we can put the old condition on the left side of the new disjunction and the new condition in the right side.

WhyQuestion After the user speaks a question, we potentially have a matching **rule** which we remember and confirm back to the user.

Recipe After the user speaks a recipe, we directly add the corresponding **rule** which we remember as well. Furthermore, we confirm back to the user that we created the rule.

Delete If there is a rule in memory, we will delete that rule and we confirm to the user.

4.3 Configurational Language

Based on our rationale, observations, requirements, use cases, and design decisions we construct a configuration language. The language is based on a trigger-action model that supports the use cases we described earlier in Section 3.3 and the examples from Section 2.1.1. We present our language below in the form of UML class diagrams, as it shows our overall architecture as well as the capabilities of the language and the possibility to extend. We use UML interfaces in order to allow extensions to be added in the future.

Architecture

In figure 4.3, we present our main classes that describes the core of our language as well as some details that are useful for execution. As can be observed, there is a program that can have zero or more rules. The rule can be run with the 'run'-operation, (de)serialized as XML with the 'parse' and 'toXML' operations, and can provide reason for the behavior with the 'reason'-operation. A rule has one condition and one or more actuators. Below in this section we describe more about conditions and actuators. There is also a state class that contains all the state that all the conditions need.

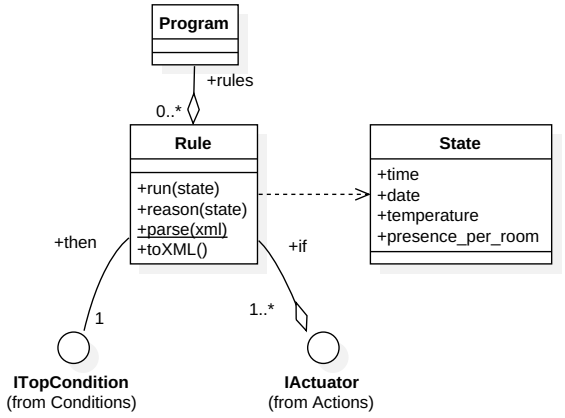


Figure 4.3: Top-level classes

Conditions Conditions are triggers that will activate a rule. We have conjunctions, disjunctions, negations, activity, button, temperature, times, dates. All of these conditions should allow for making an environment more ambient to the user. In figure 4.4, we present the class diagram of the conditions in our configurational language. We provide multiple interfaces to support future extensions.

The 'ITopCondition'-interface is the main interface that is used by the above rule. Classes that implement this interface, need to implement the 'holds'-method which based on the state will return a true or a false. The 'reason'-method is for explaining why a condition is true for a certain state. Furthermore, the 'should_update'-method is used for the semantics.

There are three logical conditions, namely the classes 'And', 'Or', and 'Not'. These are the logical part of the language namely the conjunction, disjunction, and negation. As we explained in the design decisions of Section 3.7, because of the confusion of speaking certain logical expressions we require the distinct interfaces 'ITopCondition' and 'ICondition'. As a result, the 'Not'-condition is unable to have other logical conditions.

The 'ITemporal' is an interface for all time and date related classes, such as 'Day', 'Now', 'From', 'Sun', 'DateRange', 'Date', and 'Time'. These classes return true or false based on the current time and date of the environment. This interface also implements 'ICondition', and therefore 'ITopCondition' as well.

Classes that implement the 'IPhysical' interface base their condition on the physical ambient environment. All these classes have a room as these conditions are based on information of the environment. Furthermore, these classes currently consist of 'Temperature', 'Button', and 'Activity'.

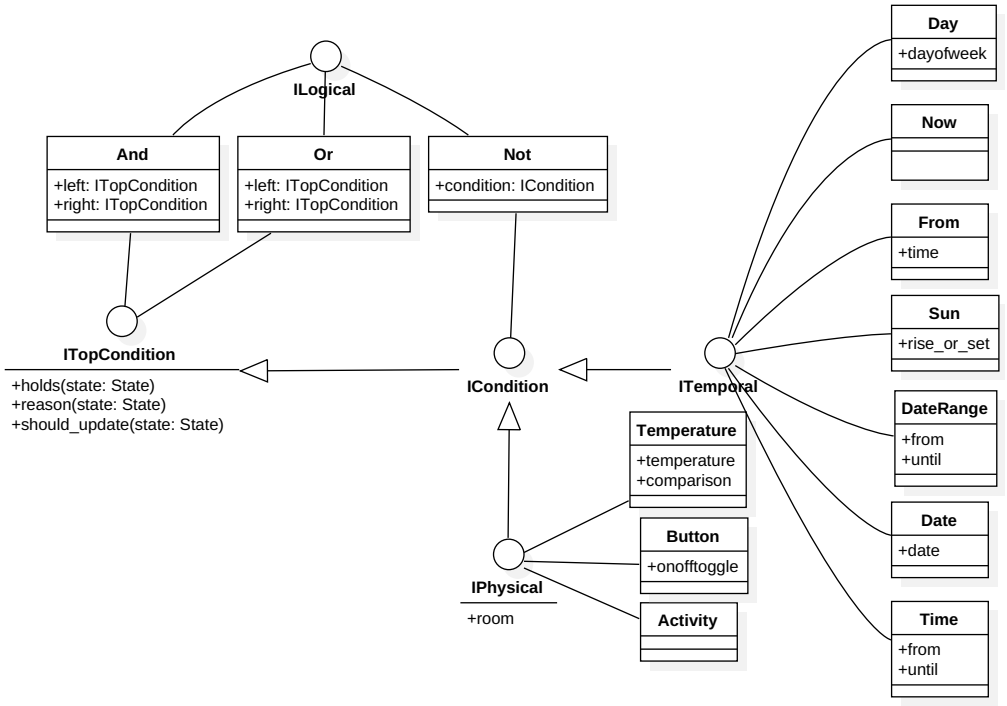


Figure 4.4: Condition classes

Actuator The actions make the configuration language alive. We have five different types of actions: lights, heating, air conditioning, curtains, and music. Lights have colors and music can be played with different genres. In figure 4.5 we present the class diagram of our actuators. As can be observed, there is one interface and five classes that implement this interface.

The 'IActuator'-interface requires the 'id'-method for recognizing a specific appliance. Notice that there is no room information, as this information can be derived from the id. There is a 'execute'-method, that when executed will perform the action on the specific appliance. Furthermore, there are the 'parse' and 'toXML' methods for XML (de)serialization.

Each class stands for a specific action on an appliance, and each action has certain actions. For example, a light can be turned on, turned off, or can be toggled. The Light and Music class also have the possibility to configure an extra parameter, the former allows a color and the latter allows a genre.

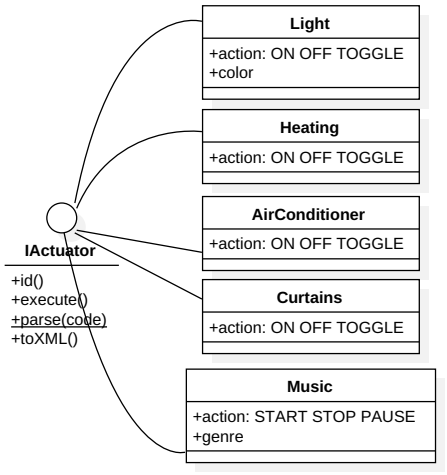


Figure 4.5: Action classes

4.4 Evaluation and semantics

In Section 4.1, we introduced an ambient environment including all conditions and actions. Additionally, in that section we already introduced when the conditions are satisfied. This section shares more details about the evaluation and semantics, as evaluating programs is currently still ambiguous.

4.4.1 Evaluation

The program originating from the voice editing process has to be evaluated on a smart home environment. We implemented our smart home environment virtually where the tester can see the results of the voice interface. However, the architecture can be connected to a real smart home with appliance integrations. The evaluator evaluates the program by the semantics and the state of the ambient environment. The new resulting actions are send to the smart home environment, which in our case is the virtual interactive prototype.

Our smart home environment provides interaction events such as users moving, time, and temperature. This information is used for the satisfaction of the conditions. In future work, the interactions could be saved into a database in order to use the historical data to make the conditions more adaptive to the users.

4.4.2 Semantics

Our DSL programs consists of zero or more rules, these rules have a condition and one or multiple actuators. There is an order of processing rules, as certain rules have priority over other rules. When an user create a new rule, the user likely has the expectation that it directly applies.

Especially when the rule contains a 'now' condition, or is bounded to time. As a result, we choose to process the rules from old to new. Where the newest rules overrule the older rules.

Once the condition is satisfied, the actuator should follow the given properties. In Section 4.1, we already described for each condition when it is satisfied. It is trivial that an action follows when a condition is satisfied, however our prototype also reverts the action once the condition is no longer satisfied. For example, if the condition is "The time is between 4 am and 8 pm" then at 4 am the action happens and at 8 pm the action reverts. Almost all physical or time conditions follow this behavior, except the 'From'-condition which only lets the action happen and does not revert the action. Furthermore, the logical conditions follow with the update behavior of their child conditions. Below we define when an update happens and when not.

When do we respond on a changing condition?

It is trivial that an action follows when a condition is satisfied, however our prototype also reverts the action once the condition is no longer satisfied. As we said before, for all conditions except 'From' a revert happens. For example, for a 'Time' condition, at 7am an event happens and at 8pm the reverse happens such as switching a light. While with a 'From' condition, the light would only turn on. As we have conjunctions, disjunctions, and negations, we specify for each condition when we respond to an update happens.

Below we have the function UPDATE, that accepts a condition together with the output of the condition as input, and returns a boolean. With this function we can determine once conditions changes their boolean values, if an update should happen. For 'From' a revert should not happen, while for others a revert should happen. So when a 'From' becomes true, an update should happen and when 'From' becomes false then an update should not happen. But for other conditions, such as 'Time' the update should happen when the condition becomes true or false. For conjunction, disjunction, and negation, an update should happen once one of the conditions requires a update.

We specify the above behavior more formal with the function UPDATE. Where the conditions C, C_1, and C_2 stand for all possible conditions we have. Furthermore, the booleans B_1 to B_3 stand for all possible values that the condition could provide. Lastly, we use the variable P for all possible parameters in the given condition.

To provide an example, And(From(8 am), Date(5 September)) should update once the date is 5 September and once the time is 8 am. Furthermore, it should update once it is 0 am at 6 September, as the date is not 5 September anymore. Therefore, UPDATE should be true once either of the conditions has a true update. Another example would be the From(8 am) alone, where the update should only true once per day at 8 am but not at 0 am.

And $\text{UPDATE}(\text{And}(C_1, C_2) = B_1) \coloneqq \text{UPDATE}(C_1 = B_2) \vee \text{UPDATE}(C_2 = B_3)$ for all booleans B_1, B_2, B_3 and all conditions C_1 and C_2 (note, the $\vee$ is no mistake if any of the booleans change a update should propegate)

Or $\text{UPDATE}(\text{Or}(C_1, C_2) = B_1) \coloneqq \text{UPDATE}(C_1 = B_2) \vee \text{UPDATE}(C_2 = B_3)$ for all booleans B_1, B_2, B_3 and all conditions C_1 and C_2

Not $\text{UPDATE}(\text{Not}(C_1) = B_1) \coloneqq \text{UPDATE}(C_1 = B_2)$ for all booleans B_1, B_2 and all conditions C_1

From $\text{UPDATE}(\text{From}(P) = \text{true}) \coloneqq \text{true}$ and $\text{UPDATE}(\text{From}(P) = \text{false}) \coloneqq \text{false}$

All others $\text{UPDATE}(C(P) = B) = \text{true}$, for C in [Activity, Button, Temperature, Time, DateRange, Day, Date, Now] and for all boolean B

Chapter 5

Results

In this chapter, we present our results which include the implemented prototype and a reflection of our findings. We describe our deliverables in Section 5.1. Afterwards in Section 5.2, we discuss the achievements, technical challenges, and conformity of our requirements.

Video demonstration

You can watch a video at `https://youtu.be/W-lS-h6uGSg` where we demonstrate our voice interface and home environment. We start by questioning why the heater is activated in the bath room and we change the condition of the rule. Afterwards we create two new rules, namely closing the curtains once the sun is down, and switching on the light once someone is standing in the kitchen.

5.1 Deliverables

Below we present our deliverables together with a brief description. Our deliverables consist of a user guide, a voice interface, a configurational language, and a web-application.

5.1.1 User Guide

To help new users, we provide a user guide of our prototype. The user guide provides examples of sentences that can be used within the prototype, and the user guide provides a context for the prototype. The user guide can be found in Appendix B.

5.1.2 Voice Interface

The voice interface consists of the generation of utterances and the handling of those utterances. We implemented our interface on top of Amazon Alexa, which receives all the utterances and sends voice requests to our web application.

Utterance generation and handling

As designed in Section 4.2, we implemented the utterance generation and the action handler in our web application. The application has two execution modes, namely the exporting of generated utterances to a JSON file and hosting the web application which includes receiving requests from Amazon Alexa and handling these.

Amazon Alexa

We created an Alexa Skill called 'Voice Code', which includes making a description, and upload our skill JSON, and uploading the SSL certificate of our web application. Afterwards, we added our Alexa appliance to the list of users that could test the skill. To be able to start using the utterances, the sentence "Open Voice Code" has to be used. Afterwards the conversation with our web application starts and the by us designed utterances can be used.

5.1.3 Configurational language

We developed our code in TypeScript [10], which allows us to fully implement the class diagrams from Section 4.3 as there are equivalent concepts for the UML classes and interfaces.

Block editor

For prototyping purposes, we implemented block editor called Blockly [24] into our web application. A screenshot of our implementation can be seen in Figure 5.1. A block editor lets users manipulate constructs defined by the programming language. Each element from the abstract syntax tree is a block. We created custom blocks for each element of our DSL, and build two methods in all DSL classes described in Section 4.3. The first method is for exporting the program into XML of Blockly and the second method is importing new rules from Blockly. Changes in the program via the voice interface are pushed to the web page, and changes from the web page are pushed back to the program.

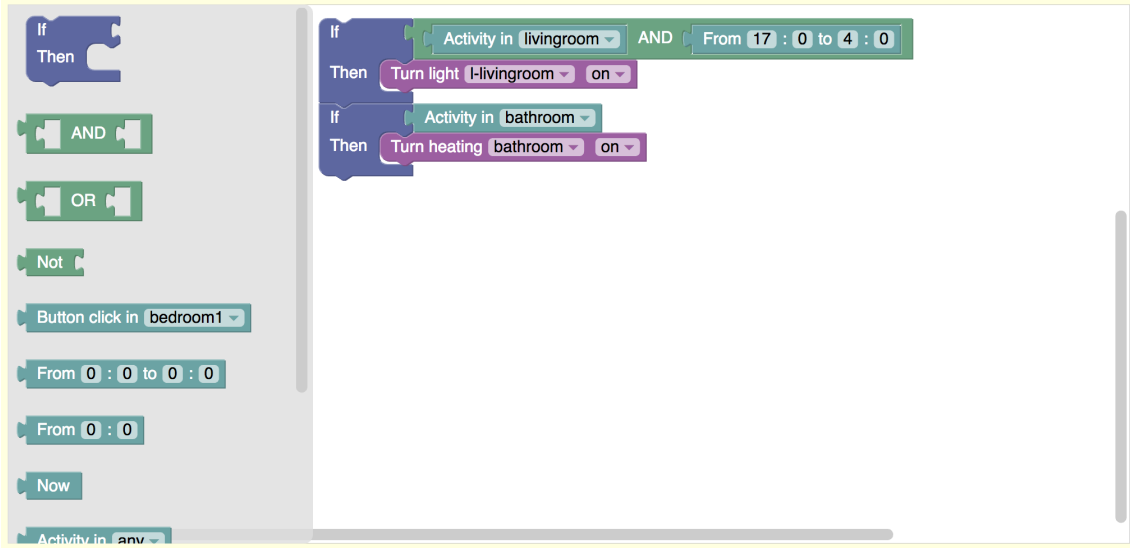


Figure 5.1: Screenshot of the Blockly editor

5.1.4 Web application

The web application provides both the virtual environment, the block editor we previously described, and the action handling of voice requests.

Virtual interactive home environment

Instead of using a real environment, we build a web page with a virtual environment projected. In Figure 5.2, a screenshot of this web page can be seen. On the top is a time bar where the passing of time can both be observed and altered. On the left side of the figure the floor plan of

the apartment can be seen, it provides several lights, buttons, heating. Furthermore there is an user icon which can be dragged and influences the environment. On the right side is the Blockly editor which we previously discussed.

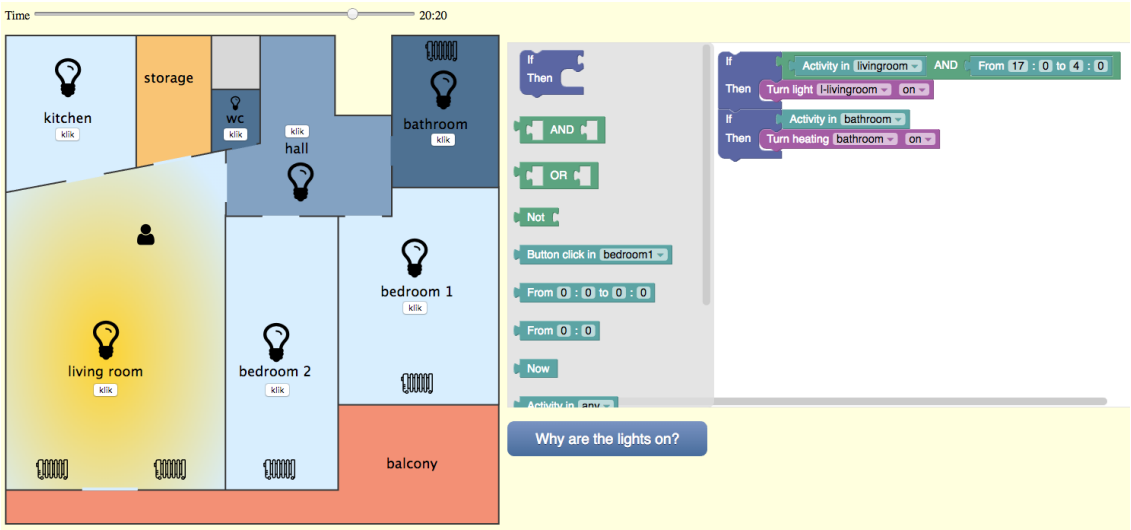


Figure 5.2: Screenshot of prototype

Crownstone Our prototype currently only implemented a virtual environment and does not include an integration with a Crownstone-plug. An integration with a Crownstone plug could realized within the current design. As a Crownstone plug can be connected between the power socket and the light, heating, air conditioning, or music appliances. Where the plug could switch the appliances on or off based on the configuration. For instance, 'Turn off lights in living room' would translate to sending off signals to the Crownstone plugs that controls a light.

Source code

The source code of our prototype can be found at `https://github.com/whazor/voicecode`. As mentioned before, the project is written in TypeScript and requires to be publicly hosted as a web service with a SSL certificate that Amazon Alexa can contact to.

5.2 Discussion

In this section, we discuss our prototype. Despite that the prototype is functional and follows the requirements together with our design, we initially wanted to implement more functionality but that was impossible both to time and technical constraints. Below we highlight several technical challenges that are observed from the prototype. Furthermore, we describe how the prototype followed the requirements.

5.2.1 Technical Challenges

Too many utterances We initially wanted to have a 'IF condition THEN action' utterance, but the combinations of all utterances became too many to upload. There are other situations as well where we will have too many utterances, when combining too many different words. The challenge will be to have a voice assistant that supports more abstract sentences.

Enormous effort To create a voice interface that responds as desired by the user, a vast amount of intents and utterances are required. As for the same intent, there are various utterances and especially with longer sentences there are more variations. Furthermore, for responding as desired, the interface is required to be more conversational which as well requires a vast amount of intents.

Not a full editor Furthermore, to create a voice interface that can fully edit the configuration, the voice interface will need to be streamlined for special requests. Such as deleting and changing configurations, currently it is only possible for a user to find out that a configuration exists when the user notices the results. For example, when a light is on then the user can ask a question. However, if the light only turns on at night when the user is sleeping, then the user might never discover this rule.

Expressiveness More advanced programs that for numeric calculations or perform other algorithms are currently not possible. It might be useful to for example link the brightness of the light based on the cloudiness outside.

Device integration While open-source solutions such as OpenHAB provide more integrations that we could connect to. Actually connecting all these integrations requires configuration, such as filling in IP addresses of devices or device bridges. Our design did not include these steps that are needed to configure ambient environments.

Ambient Intelligence The DSL does not provide the ease of use that the vision of Ambient Intelligence requires, as the user has to configure everything from start. In comparison, IFTTT provides recipes where with one click new behavior is added. While adding recipes is still a manual action, it is an improvement over creating configurations from scratch.

5.2.2 Verification of requirements

We implemented and designed our prototype based on the requirements originating from Section 3.6. Below for each requirement we repeat what we need to verify and we describe the result.

Domain Specific Language

REQ 1 **To verify:** "1: all voice utterances that we define in the design have a corresponding conceptual construct in the configuration language. 2: For all possible conceptual constructs from the configuration language, there exists an utterance where the user can create, add, or delete such constructs from the programs" **Result:** 1: All voice utterances have a corresponding construct in the language. 2: Can all rules be created, edited and deleted by the voice interface? Changing the program via the voice interface is not efficient as deleting a rule is required, further the rule has to be questioned first before it can be changed. Additionally, not every logical condition could be created as conjunctions and disjunctions can only have other logical expressions on the left side.

REQ 2 **To verify:** "We require that all use cases described in Section 3.3 have corresponding constructs in the configurational language" **Result:** All use-cases from Section 3.3 are incorporated into the design and implementation. However, the heating condition is oversimplified as we do not allow to configure a temperature.

REQ 3 **To verify:** "We require that people without programming experience could construct a program using a graphical editor" **Result:** We provide testers with a graphical programming editor called Blockly and they can change and observe the programs inside the browser.

REQ 4 **To verify:** "we require that the language does not contain any details about execution" **Result:** When looking at the actuators from the language in Section 4.3. It can be observed that there is an execution-method which has to be implemented in order to make the language functional. We can conclude that the design itself does not conclude any execution details.

REQ 5 **To verify:** "1: Programs from the configuration language can be constructed in a graphical editor and be executed in a prototype environment. 2: The voice interface can construct programs that are presented in a graphical editor without being executed. Having such a construction allows separate implementations of voice interfaces or smart home environments" **Result:** 1: All possible programs from the configuration language can be constructed in the graphical editor and be executed in the virtual environment. 2: Programs constructed in the voice interface are shown in the graphical editor.

Voice Interface

REQ 6 **To verify:** "There exists a combination of one or more utterances such that the user can create new a rule, edit a rule, and delete a rule" **Result:** Although we support creation, updating, and deletion of rules. There are not enough intents and utterances to make it intuitive. Furthermore, referencing a configuration is only possible when the user notices the rule as it cannot ask inspection questions otherwise.

REQ 7 **To verify:** "For each possible state of all actuators, there is an utterance formed as a question. After speaking such a question, the system responds with a suiting explanation. Lastly, there exists a combination of one or more utterances such that the user can edit, and delete the corresponding rule" **Result:** Every actuator has questions for every possible state of the actuator. Afterwards, the rule is put in memory and the user can edit or delete the rule.

REQ 8 **To verify:** "we require that the voice interface can be used without laptop or phone" **Result:** The prototype is usable via Amazon Echo and is therefor speech only.

REQ 9 **To verify:** "in the user manual, there are no mentions made of programming concepts" **Result:** The user guide can be found in Appendix B. We make no references to programming concepts.

Chapter 6

Conclusions

In this work, we designed and created a domain specific language together with a speech interface for the configuration of Ambient Environments. We initially studied the background of ambient environments, voice assistants, domain specific languages, and discussed the shortcomings of ambient environments and voice assistants. We discussed the benefits from domain specific languages and design a domain specific language for ambient environments. We designed a voice interface that is capable of configuring our language. To further research our initial questions, we have implemented both the language and voice interface which resulted in a functional prototype. However, while implementing we have found several technical challenges that restricted us from making a better interface.

6.1 Contributions

Our work resulted in several artifacts. We build a technically working DSL and voice interface that controls virtual ambient environment. The voice interface is working on top of an existing voice assistant and the DSL is connected to the voice interface together with a demonstration of a virtual ambient environment.

Besides artifacts, our thesis provides research in voice interfaces for DSLs. For voice assistants, we provide experience for creating more complicated voice assistants. Our implementation details, together with the challenges we found could help new implementation become more successful.

Furthermore, we provide an interesting method to troubleshoot with running programs as the user can question why certain behavior is happening. Often during the development of software mistakes or wrong assumptions are made, which lead to unwanted behavior when the software is in production. When the users of the program observe the unwanted behavior, they could ask why it is happening and given an advanced enough interface they could fix it.

6.2 Future work

Our work is by far not complete enough for realistic usage. By our limited testing we found that our solution is cumbersome by several issues. We describe these challenges in Section 5.2.1. Below we describe work that can be performed for either solving the challenges, but as well as further researching our research questions.

More intents and utterances The more utterances and intents a skill has, the more chance there is that what the user desires is accepted. This work takes enormous effort, as there are many variations of certain sentences. Analytics and user testing could be performed in order to discover which utterances and intents are required.

Validation To validate the concept, a more realistic testing environment is required where multiple users test the concept. Also, different environments and devices could be tested, such

as a office, factories, warehouses or hospitals.

Generalization Integrate with OpenHAB to support more categories and broaden the usage. The current prototype is limited because it only supports lights and heating, for real use it should support more devices. As OpenHAB and Eclipse SmartHome already provide generalizations of devices and is open-source with build on top of Java, it would be straightforward to broaden and integrate with our proposed domain-specific language.

Conversational In order to make the voice interface more useful, more conversations should be implemented such that the assistant can steer the conversation and explain the options. Currently it is not clear what the voice interface can do without a tutorial or explanation. It would be useful if the interface could introduce itself and provide options.

Expressivity More research towards a more expressive domain-specific language is required, while at the same time the interface has to be kept understandable. Making voice interfaces more expressive can be challenging, as expressiveness could lead to more intents and utterances that would have be defined.

Bibliography

[1] Abdaladhem Albreshne and Jacques Pasquier. A domain specific language for high-level process control programming in smart buildings. *Procedia Computer Science*, 63(Supplement C):65 – 73, 2015. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2015.08.313. URL http://www.sciencedirect.com/science/article/pii/S1877050915024412. The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015)/ The 5th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2015)/ Affiliated Workshops. 15

[2] Amazon. Define the interaction model in json and text. https://developer.amazon.com/docs/custom-skills/create-intents-utterances-and-slots.html, 2018 1. (Accessed on 01/14/2018). 8

[3] Amazon. Amazon.com help: Use your smart home device with alexa. https://www.amazon.com/gp/help/customer/display.html?nodeId=201749260, 4 2017. (Accessed on 01/05/2018). 10

[4] Amazon. Understanding how users invoke custom skills. https://developer.amazon.com/docs/custom-skills/understanding-how-users-invoke-custom-skills.html, 12 2017. (Accessed on 01/14/2018). 8

[5] Apple. Sirikit. https://developer.apple.com/documentation/sirikit, 12 2017. (Accessed on 01/14/2018). 8

[6] Jacob Aron. How innovative is apple's new voice assistant, siri? *New Scientist*, 212(2836): 24, 2011. 7

[7] Athom. Athom developer. https://developer.athom.com/docs/apps/tutorial-Speech.html, 12 2017. (Accessed on 01/14/2018). 8

[8] Athom. Discover smart home scenarios. https://www.athom.com/en/usecases/, 01 2018. (Accessed on 01/05/2018). 10

[9] Laurent Besacier, Etienne Barnard, Alexey Karpov, and Tanja Schultz. Automatic speech recognition for under-resourced languages: A survey. *Speech Communication*, 56:85 – 100, 2014. ISSN 0167-6393. doi: https://doi.org/10.1016/j.specom.2013.07.008. URL http://www.sciencedirect.com/science/article/pii/S0167639313000988. 6

[10] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In *European Conference on Object-Oriented Programming*, pages 257–281. Springer, 2014. 35

[11] S. R. K. Branavan, Luke S. Zettlemoyer, and Regina Barzilay. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1268–1277, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1858681.1858810. 14, 15

[12] Philip Brey. Freedom and privacy in ambient intelligence. *Ethics and Information Technology*, 7(3):157–166, 2005. 1, 6, 8

[13] Crownstone B.V. Readme of the crownstone app. https://github.com/crownstone/CrownstoneApp/blob/f31a72913929d7a5b11c35113eff3c3963cbf7cf/README.md, 9 2017. (Accessed on 04/09/2018). 2

[14] Crownstone B.V. Crownstone hardware specification. https://shop.crownstone.rocks/pages/specifications, 02 2018. (Accessed on 03/22/2018). 18

[15] CMUSphinx. Frequently asked questions (faq) of cmusphinx. https://cmusphinx.github.io/wiki/faq/, 10 2017. (Accessed on 01/07/2018). 10

[16] CMUSphinx. Tuning speech recognition accuracy - cmusphinx. https://cmusphinx.github.io/wiki/tutorialtuning/, 10 2017. (Accessed on 01/07/2018). 10

[17] Alan Dix. Human-computer interaction. In *Encyclopedia of database systems*, pages 1327–1331. Springer, 2009. 5

[18] Eclipse. Eclipse smarthome - a flexible framework for the smart home. http://www.eclipse.org/smarthome/documentation/concepts/index.html, 1 2018. (Accessed on 10/04/2017). 2, 5, 15

[19] José Encarnação Emiele Aarts. *True visions: the emergence of ambient intelligence*. Springer, 2006. 4

[20] Aarts Emile and Lisette Appelo. Ambient intelligence: thuisomgevingen van de toekomst (dutch), 1999. 4

[21] Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 307–309. ACM, 2010. 13, 14

[22] Martin Fowler. Language workbench. https://www.martinfowler.com/articles/languageWorkbench.html, 6 2005. (Accessed on 02/22/2018). 12

[23] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010. 12

[24] N Fraser et al. Blockly: A visual programming editor. *https://code.google.com/p/blockly*, 2013. 14, 35

[25] Google. Actions documentation. https://developers.google.com/actions/reference/rest/Shared.Types/Action, 5 2017. (Accessed on 01/14/2018). 8

[26] Google. Control smart home devices using google home. https://support.google.com/googlehome/answer/7073578, 11 2017. (Accessed on 01/05/2018). 10

[27] Google. Discovery and actions on google. https://developers.google.com/actions/discovery/, 11 2017. (Accessed on 01/14/2018). 8

[28] Richard C Gronback. *Eclipse modeling project: a domain-specific language (DSL) toolkit*. Pearson Education, 2009. 11

[29] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *Proceedings of communication networks and distributed systems modeling and simulation conference*, volume 2004, pages 270–275. San Diego, CA, USA., 2004. vi, 15, 16

[30] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, 28(1):1–18, 2005. 15

[31] IFTTT. Wake to the colors of roses on valentine's day! https://ifttt.com/applets/136364p-wake-to-the-colors-of-roses-on-valentine-s-day, 2018. (Accessed on 02/13/2018). 20

[32] MPS Jetbrains. Meta programming system. https://www.jetbrains.com/mps/, 2014. (Accessed on 02/19/2018). 13, 14

[33] Lennart CL Kats and Eelco Visser. The spoofax language workbench: rules for declarative specification of languages and ides. In *ACM sigplan notices*, volume 45, pages 444–463. ACM, 2010. 13, 14

[34] Michael Klein, Andreas Schmidt, and Rolf Lauer. Ontology-centred design of an ambient middleware for assisted living: The case of soprano. In *Towards Ambient Intelligence: Methods for Cooperating Ensembles in Ubiquitous Environments (AIM-CU), 30th Annual German Conference on Artificial Intelligence (KI 2007), Osnabrück*, 2007. 15

[35] Ingrid Lunden. Ifttt launches 3 'do' apps to automate photo sharing, tasks, notes; rebrands main app 'if'. https://techcrunch.com/2015/02/19/ifttt-launches-3-do-apps-to-automate-photo-sharing-tasks-notes-rebrands-main-app-if/, 2015 2. (Accessed on 01/20/2018). 15

[36] Josh Mengerink, Bram van der Sanden, Bram Cappers, Alexander Serebrenik, Ramon Schiffelers, and Mark van den Brand. Exploring dsl evolutionary patterns in practice: a study of dsl evolution in a large-scale industrial dsl repository. In *6th International Confenrence on Model-Driven Engineering and Software Development (MODELSWARD 2018), 22-24 January, 2018, Funchal, Madeira-Portugal*. SciTePress, 2018. 12

[37] Dipendra K Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300, 2016. 14, 15

[38] MycroftAI. Mycroftai/mycroft-skills: A repository for sharing and collaboration for third-party mycroft skills development. https://github.com/MycroftAI/mycroft-skills, 01 2018. (Accessed on 01/14/2018). 8

[39] OMG. Iso/iec 19502:2005 - information technology – meta object facility (mof). https://www.iso.org/standard/32621.html, 11 2005. (Accessed on 02/19/2018). 11

[40] OMG. Omg specifications and process. http://www.omg.org/gettingstarted/overview.htm, 05 2018. (Accessed on 02/21/2018). 11

[41] OpenHAB. openhab - supported technologies. http://www.openhab.org/technologies.html, 2018 01. (Accessed on 05/02/2018). 1, 15

[42] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koen De Bosschere. Towards an extensible context ontology for ambient intelligence. In *EUSAI*, volume 3295, pages 148–159. Springer, 2004. 15

[43] Jasper Project. Control everything with your voice. http://jasperproject.github.io/, 1 2015. (Accessed on 01/06/2018). 10

[44] Kari Smolander, Kalle Lyytinen, Veli-Pekka Tahvanainen, and Pentti Marttiin. MetaeditâĂŤa flexible graphical environment for methodology modelling. In *International Conference on Advanced Information Systems Engineering*, pages 168–193. Springer, 1991. 13

[45] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. 2 edition, 2008. ISBN 0321331885,9780321331885. 11

[46] Paul Taylor. *Text-to-speech synthesis*. Cambridge university press, 2009. 6

[47] Linden Tibbets. ifttt the beginning. https://ifttt.com/blog/2010/12/ifttt-the-beginning, 2010 12. (Accessed on 10/11/2017). 2, 5, 15

[48] Khai Truong, Elaine Huang, and Gregory Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. *UbiComp 2004: Ubiquitous Computing*, pages 143–160, 2004. 16

[49] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 803–812. ACM, 2014. 15

[50] Mark van den Brand, Arie van Deursen, Jan Heering, Hayco De Jong, Merijn de Jonge, Tobias Kuipers, Paul Klint, Leon Moonen, Pieter A Olivier, Jeroen Scheerder, et al. The asf+ sdf meta-environment: A component-based language development environment. In *International Conference on Compiler Construction*, pages 365–370. Springer, 2001. 13, 14

[51] Tijs van der Storm. *The Rascal language workbench*. CWI. Software Engineering [SEN], 2011. 13, 14

[52] Tam Van Nguyen, Wontaek Lim, Huy Nguyen, Deokjai Choi, and Chilwoo Lee. Context ontology implementation for smart home. *arXiv preprint arXiv:1007.1273*, 2010. 15

[53] Anne van Rossum. Integration with homey on crownstone transparent product roadmap. https://trello.com/c/bEitBerq/34-integration-with-homey, 6 2017. (Accessed on 03/22/2018). 2, 19

[54] Anne van Rossum. Integration with alexa on crownstone transparent product roadmap. https://trello.com/c/XHZi1Pey/11-integration-with-alexa, 6 2017. (Accessed on 03/22/2018). 2, 19

[55] Markus Völter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser, and Guido Wachsmuth. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*. dslbook.org, 2013. ISBN 978-1-4812-1858-0. URL http://www.dslbook.org. 12, 14

[56] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013. 11

[57] Xiao Hang Wang, D Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. Ieee, 2004. 15

[58] Jingjing Xu, Yann-Hang Lee, Wei-Tek Tsai, Wu Li, Young-Sung Son, Jun-Hee Park, and Kyung-Duk Moon. Ontology-based smart home solution and service composition. In *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, pages 297–304. IEEE, 2009. 15

Appendix A

Intents and Utterances

Actions

lights {action} (the)? (all lights|all the lights|light|lights|bulb|lamp|lamp bulb) (in {room})?

lightsColor (turn|make) the lights {color} (in {room})?

heating {action} (the)? (heating|heater|furnace|radiator) (in {room})?

airConditioner {action} (the)? air conditioner (in {room})?

playMusic (play|start) (suitable)? music (in {room})?

stopMusic (pause|suspend|halt|stop|end) music (in {room})?

musicGenre play {genre} music (in {room})?

openCurtains (slide)? open curtains (in {room})?

closeCurtains (close|shut) curtains (in {room})?

toggleCurtains toggle curtains (in {room})?

Conditions

sunUp "(when|once) the sun goes up"

sunDown "(when|once) the sun goes down"

temperatureAbove "temperature (in {room})? is above {temperature} (degrees|celsius)?"

temperatureBelow "(the)? temperature (in {room})? is below {temperature} (degrees|celsius)?"

temperatureAround "temperature (in {room})? is around {temperature} (degrees|celsius)?"

activity "(there is someone|someone is) in {room}"

button "(the)? (button|switch) (in {room})? is {click}"

time "(the time is between|everyday from|from|between)? {start} (to|and) {end}"

from "(from|at) {start}"

date "(on)? {date}"

now "(from)? now"

Why questions

whyLightsOn "Why (is|are) (the)? (lights|light|lamp) (on|not off) (in {room})?"

whyLightsOff "Why (is|are) (the)? (lights|light|lamp) (off|not on) (in {room})?"

whyLightsColor "Why (is|are) (the)? (lights|light|lamp) (not)? {color} (in {room})?"

whyHeatingOn "Why is it (so)? warm (in {room})?", "Why is the heating (on|not off) (in {room})?"

whyHeatingOff "why is it (so)? cold (in {room})?", "why is the heating (off|not on) (in {room})?"

whyMusicOn "Why is the music (playing)? (in {room})?"

whyMusicOff "Why is there no music (playing)? (in {room})?", "Where is the music (in {room})?"

whyMusicGenre "Why is there {genre} playing (in {room})?"

whyCurtainsOpen "Why are the curtains (still)? (open|not closed) (in {room})?"

whyCurtainsClosed "Why are the curtains (still)? (closed|not open) (in {room})?"

Editing

removeThatRule "(can you|could you)? (remote|delete|cut out|erase|abolish) that rule"

inRoom "(only|in|only in|change to) {room}"

A.1 Recipes

romance "Wake me up to the colors of roses on Valentines Day"

comfyTemperature "Set all the rooms at a comfortable temperature"

closeCurtains "Close all the curtains automatically in the evening"

Slot Types

room

- hall
- balcony,
- bed room bedroom1, sleeping room
- bathroom bathroom, shower room, spa, shower, washroom
- living room, the living room
- kitchen, cookery
- toilet, wc

action

- switch, toggled, toggle
- turn off, turn down, put out, turn out, shut down, put off, kill, halt, close, unplug, switch off, shut off
- turn on, fire up, put in gear, initiate, start up, begin, activate

click

clicked, pushed, switched

genre

African, Asian, East Asian, South and southeast Asian, Avant-garde, Blues, Caribbean and Caribbean-influenced, Comedy, Country, Easy listening, Electronic, Folk, Hip hop, Jazz, Latin, Pop, R&B and soul, Rock

partOfDay

Early morning, late morning, morning, early afternoon, afternoon, noon, early evening, evening, night, midnight

color

White, Silver, Gray, Black, Red, Yellow, Green, Blue, Purple, Pink

Others

temperature number

start time

end time

date date

temperature: number, start: time, end
{room}, {color}, {action}, {genre}

Appendix B

User guide

See next page.

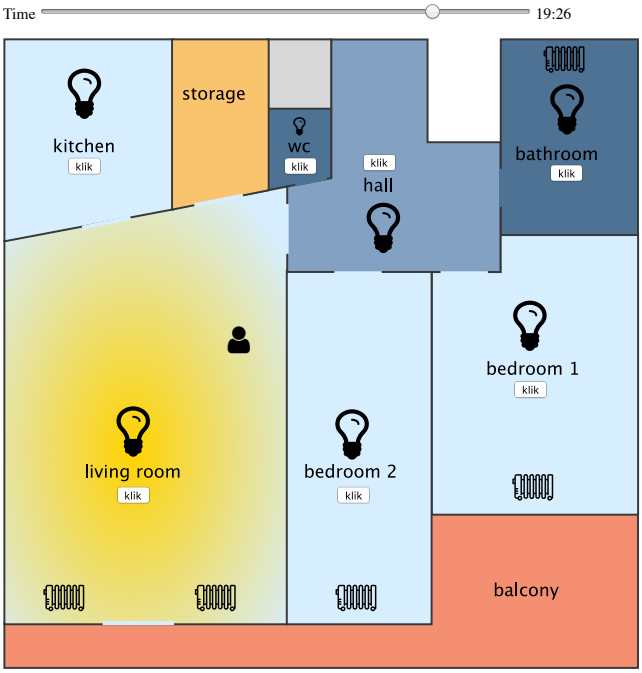
Introduction to VoiceCode

Welcome

Thank you for trying out Voice Code on Amazon Alexa. In this document we explain how to use Voice Code to configure a smart home.

Smart Home

On the right you see an illustration of the prototype smart home environment. Each room has a name and many of the rooms have lights, heating, and buttons. Furthermore, there is a time that can be changed for testing purposes.



Things you can say

All the categories of sentences that Voice Code supports.

- Actions
- Conditions
- Inspecting
- Changing or deleting
- Recipes

Actions

Lights

Turn on the lights in the living room. / Make the lights blue. / Turn off the lights.

Heating

Turn the heating on. / Turn off the radiator. / Open the heating in the living room.

Air conditioner

Turn the air conditioner on. / Start air conditioner in bed room.

Music

Turn the music on. / Play country music. / Start jazz music in the kitchen. / Pause the music. / Stop music.

Curtains

Open the curtains / Close the curtains / Open curtains in bed room

Conditions

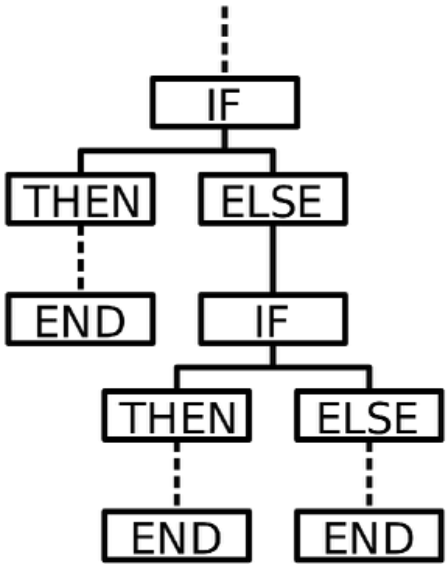
With conditions you can make actions happen when you want them to happen. For example it is possible to turn on the lights in the night or to close the curtain. If you speak these conditions to Alexa together with an action then you create a new rule.

Connecting conditions with actions

First speak a CONDITION, then speak an ACTION. Or visa versa. Afterwards the two will be connected and once the condition holds the action will happen. For example say “When the sun goes down” and say “open the lights”.

Combinations

You can combine conditions by mentioning a CONDITION and afterwards saying “And CONDITION” or “Or CONDITION“. After the conditions are created, only once either or both of them hold then the action will happen.



Time and date

Sun once the sun goes up / when the sun goes down
Time The time is between 7 am and 9 pm / the time is from 9 pm to 4 am
From from 8 pm / at 4 am
Date On 5 march
Now Now

Based on your home

Button Someone pressed the button in the living room / The button is clicked
Activity Someone in the living room / There is someone in the hall
Temperature Temperature in kitchen is above 25 Celcius / Temperature in bed room is below 15 Celcius / temperature is around 22 Degrees

Inspecting

After there are rules, you might forget about these rules and wonder why something is happening. Voice Code lets you inspect your current rules and change or delete them.

Ask for answer

Why lights

Why are the lights on in the living room? / Why are the lights not on? / Why are the lights red? / Why are the lights off in the kitchen?

Why heating or air conditioning?

Why is it so warm? / Why is it so cold? Why is the heating not on in the living room?

Why music

Why is the music playing? / Why is there no music playing? / Why is there jazz playing in the kitchen?

Why curtains

Why are the curtains still open? / Why are the curtains closed? / Why are the curtains closed in the living room?

Changing or deleting

After asking a question or creating a new rule, a rule is in memory that you can change or delete.

Changing a rule

Change or add actions with “Change to ACTION” or “Add ACTION”. Changing a condition is possible with “Only when CONDITION“. Or use the And/Or combinations to expand the condition.

Removing a rule

Say: “Delete that rule” or “Stop doing that”

Recipes

There are senteces that directly create a rule, we call them recipes.

Romance

“Wake me up to the colors of roses on Valentine’s Day!”, this recipe adds a rule with a date condition on 14 February and changes the lights to pink.

Comfortable Temperature

“Set all the rooms at a comfortable temperature”, this recipe adds a rule that turns on the heating for all rooms.

Close curtains

“Close all the curtains automatically in the evening”, this recipe adds a rule that closes the curtains everyday at 18 pm.