



Hasso Plattner Institute for Digital Engineering

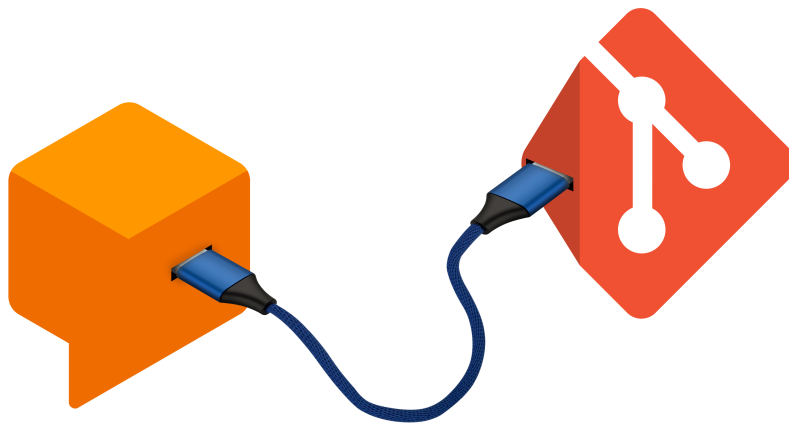
Bachelor's Thesis

Supporting Iterative Development of Voice Interfaces using a Domain Specific Language

Unterstützen iterativer Entwicklung von Sprachassistenten durch eine
domänenspezifische Sprache

Arne Zerndt

arne.zerndt@student.hpi.de



Supervised by Prof. Dr. Holger Giese
System Analysis and Modeling Group

Potsdam, June 27, 2019

Abstract (English)

This thesis seeks to address a current lack of version control systems for voice interface configurations. Instead of designing a new version control system for voice interface configurations, here a different solution is chosen. The voice interface configuration is adapted to fit into existing version control systems. This is done by designing a domain specific language to describe the configuration of a voice interface in a text based way. The voice interface technology chosen to demonstrate this solution is Google Dialogflow. The version control system chosen to focus on is Git, using Github. For evaluation, an experiment with five test cases was conducted, comparing the domain specific language with the existing technology. The results show a clear benefit in three of four criteria outlined, while maintaining parity in the fourth.

Abstract (Deutsch)

Diese Bachelorarbeit widmet sich einem derzeit bestehenden Mangel an Versionskontrollsystemen für Sprach-Interfaces. Anstatt aber ein neues Versionskontrollsystem für Sprach-Interfaces zu bauen, wird hier eine andere Lösung gewählt. Ein Sprach-Interface wird so adaptiert, dass es mit bestehenden Versionskontrollsystemen kompatibel ist. Zu diesem Zweck wird eine domänenspezifische Sprache entwickelt, die ein Sprach-Interface auf eine textbasierte Art spezifiziert. Um diese Lösung zu demonstrieren, wurde die Sprach-Interface-Technologie Dialogflow gewählt, in Verbindung mit dem Versionskontrollsystem Git. Für die Evaluation wurde ein Experiment mit fünf Testszenarien durchgeführt, in dem die domänenspezifische Sprache mit der existierenden Technologie verglichen wurde. Die Ergebnisse zeigen einen klaren Vorteil durch die Nutzung der DSL bei drei von vier zuvor spezifizierten Kriterien, ohne Verlust beim vierten Kriterium.

Contents

Table of Contents	V
1. Introduction	1
2. Status Quo	2
2.1. Voice Interfaces	2
2.2. Version Control	2
3. Problem Statement	4
4. Approach	6
4.1. Choosing a Direction	6
4.1.1. Version Control System	6
4.1.2. Git with exported JSON files	6
4.1.3. Domain Specific Language	6
4.2. DSL Engineering	7
4.2.1. What is a DSL?	7
4.2.2. Requirements for the DSL	7
4.2.3. Design Decisions	8
4.2.4. Usage	11
4.2.5. Dynamic Entity Control	11
4.3. Target Group	11
5. Evaluation	12
5.1. Length	12
5.2. Diff Size	14
5.3. Readability	16
5.4. Automatic compilation and update	19
5.5. Concessions and Drawbacks	19
5.5.1. Design Choices differing from the Dialogflow Website	20
5.5.2. The Difficulty of Deciding on Defaults.	20
5.5.3. Unsupported Dialogflow Features	21
6. Summary and further Work	22
Bibliography	23
A. Declaration of Authorship	24

B. DSL Code of Test Agents	25
B.1. RoomTemperature Agent	25
B.2. MusicPlayer Agent	26
B.3. Calendar Agent	27
B.4. JokeBot Agent	28
B.5. QuoteBot Agent	29
C. Examples of Diffs created from Test Agents	30
C.1. RoomTemperature Agent	30
C.2. Test Agents 2-5	30
D. Test Case Data and Averages	31
D.1. Line Length of Test Agents	31
D.2. Diff Size of Changes to each Test Agent	31
E. DSL Grammar	32
F. DSL Code Generator	36
G. JSON Versions of Test Agents	43
G.1. RoomTemperature Agent	43
G.1.1. Agent	43
G.1.2. Entities	43
G.1.3. Intents	44
G.2. Test Agents 2-5	47
H. Examples of Diffs created from Test Agent JSON files	48
H.1. RoomTemperature Agent	48
H.2. Test Agents 2-5	51

1 | Introduction

The following Bachelor thesis is part of the "Ask your Repository!" Bachelor project.¹

In this thesis, I will demonstrate a possible solution for the issue of synchronizing development on voice interfaces with that of application code by checking voice interface configuration into version control alongside the code it corresponds to. The focus will be on allowing the use of existing version control systems for voice interface configurations.

To approach this goal, I have designed a Domain Specific Language (DSL) to describe the configuration of a voice interface in a text-based but still abstract and intuitive way.

I will focus on development using Google Dialogflow as it is the most widely used tool [Sta]; it is also what my Bachelor team is using in our project.²

I propose that by designing a DSL that can be used to configure a Dialogflow agent in a way that is text based and can be managed by version control systems like Git - but that is intuitive to use for a developer familiar with the Dialogflow web interface -, development on voice interfaces can be simplified. Also, development teams in the future will be able to work on voice assistants in the same way that they are used to when working with code. This opens up the entirety of code based tools which exist for making development more streamlined for voice interfaces, while keeping the robustness of code which can be saved in version control.

In chapter 2, I will explain the current situation in regard to voice interface technology and version control. Afterwards, in chapter 3, I will specify the problem I am trying to solve in the thesis. Chapter 4 will focus on finding a solution to the above mentioned problem, as well as on the specific solution I built a prototype for. Going into more detail, chapter 4.2 will show how I used Xtext to create a grammar and code generator for configuring Google Dialogflow agents. This grammar was used to provide syntax highlighting and validation in Eclipse DSL; the output of the code generator is a valid voice interface configuration in its JSON representation, ready to be imported into Google Dialogflow. Lastly, in chapter 5 I will evaluate the success of the prototype by comparing it to an alternative way of checking voice interfaces into source control.

¹<https://hpi.de/giese/lehre/bachelorprojekte/ask-your-repository.html>

²The code for my DSL can be found here: <https://github.com/arne-z/BachelorThesis> and an implementation for the voice assistant in our bachelor project using the DSL can be found here: <https://github.com/hpi-sam/ask-your-repository-dialogflow-adapter/tree/agent-config-with-dsl>

2 | Status Quo

2.1. Voice Interfaces

Voice interfaces, voice assistants, and chat bots are increasingly popular technologies [OK19, page 8] that are experimented with and used by every major player in the technology market [Cha18b]. To keep up with this trend, developers either need to be able to build their own voice interfaces or integrate with an existing system of which there are many [Alt19] and of which Google Assistant and Amazon Alexa are the most relevant; this can be seen in a survey Microsoft conducted this year on the popularity of voice assistants [OK19, page 9]. Currently, depending on whether you are developing for Google Assistant or Alexa, designing a voice interface for one of these systems usually entails working with either Google Dialogflow or Amazon Lex. These are powerful tools which enable developers or domain specialists to quickly and easily design a voice interface. These tools are interacted with through a website which provides a graphical editor for the configuration of the voice interface.

2.2. Version Control

While the web interface makes initial setup of the voice interface easier for a single domain expert or developer, new difficulties arise when a team of developers is working on a voice interface in an iterative fashion. It becomes crucial for the team to manage versions and track changes to the interface along with the changes to the application the interface is supporting.

As Martin Charles states in the text accompanying his Dialogflow CLI community tool [Cha18a, page 1]:

DialogFlow stores intents and entities outside of source control. This makes rollbacks and keeping track of history difficult.

The state of the art solution for managing iterative work in development teams is Git. On GitHub alone there are more than 190 million Git repositories at present, as can be seen by looking at [\[Gitb\]](#). Git provides functionalities for saving snapshots of specific iteration in your project and handling the problems that come up in iterative work.

These problems are:

- merging work done by multiple developers
- ensuring that a stable version of a project is saved while developers are working on more experimental changes
- giving the team the ability to easily track and revert changes that have been made.

The issue that arises is that the above mentioned technologies are not compatible and the configuration of a voice interface can easily get out of sync with the changes made to the application and managed in Git.

3 | Problem Statement

As stated before, currently there is no version control system for Dialogflow agents. This is problematic because when designing a voice interface, it is necessary to make iterative changes, so the voice interface can evolve alongside the application it supports. This can lead to issues in a number of different situation, and causes voice interface developers to miss out on the advantages modern software development gains from using Git.

- When starting work on a new feature, a new branch is created. This is done so that changes, which might not work right away, are contained to this branch and can be merged into the main application at a later time. If this feature requires changes to the voice interface, a problem may arise, because the voice interface has no mechanism for branching.
- Once experimental changes on a feature branch advance to a point where they are meant to be included in the master branch, code can simply be merged from one branch to another. The changes to a voice interface configuration, however, cannot be included in a pull request.
- When working on a product, it is generally considered to be “best practice” to have code that should be merged into the master branch from a feature branch reviewed by at least one or two other developers to make sure that it is working as expected and does not have any obvious flaws. This cannot be done for changes to a voice interface.
- When working in a team, it is not always possible to remain aware of all the changes team members have made. Source code that is managed in Git automatically creates a history of all changes, which is highly valuable to the developers. This type of history does not exist for the voice interface configuration.
- Open source development is an important part of today’s development landscape. Since voice interfaces are not usually checked into source control alongside application code, open source development of voice interfaces or applications that use voice interfaces is stifled. In addition, open source projects can be forked by other developers and can be improved by many developers in an iterative fashion. This is another advantage that voice interfaces are currently lacking.

When working on a Dialogflow agent, one can save a version of the configuration and then continue as a draft, but Dialogflow assumes that you will only ever have one draft at a time. Versions are also designed in a linear fashion with no way to merge changes from multiple versions. This is obviously nowhere near the depth of features that are necessary in version control and all of these features are already provided for normal code by using the current state of the art version control system, Git.

In summary, the problem is that there is no sufficient version control system, that is compatible with Dialogflow.

4 | Approach

4.1. Choosing a Direction

4.1.1. Version Control System

Since there is no version control system that is compatible with Dialogflow, the obvious solution might be to build a new version control system that is compatible with Dialogflow. This has been done before for other technologies. An example of this would be the Open-source Version Control System for Machine Learning Projects that evolved from the necessity of specialized version control for machine learning models and data sets.[\[Pet\]](#)

I decided against this approach for multiple reasons.

Firstly, I believe that it will be hard to get developers to move from an established tool like Git; I assume that developers would not choose to give up feature rich support that exists for Git (Github, Gitlab, Bitbucket to only name a few). Secondly, using a specific version control system that is developed to allow compatibility with Dialogflow, it would be hard to also maintain compatibility with other tools.

Therefore, instead of trying and failing to develop a competing standard to Git, I decided to make use of a workaround that is possible with Dialogflow.

4.1.2. Git with exported JSON files

Another possibility would be to export a folder with a JSON representation of an agent and save this in Git. This solves some of the issues mentioned in chapter 3, but an agent's JSON representation is not intended for readability by humans, and neither is it meant for direct editing. This makes certain aspects of the workflows I described above much harder, e.g. conducting a code review, since it is difficult to read the changes to the JSON files describing the agent's configuration. This is the option currently used by many developers [\[Oos19\]](#) but that I decided against.

This is also the option that I will compare my prototype to in the evaluation chapter.

4.1.3. Domain Specific Language

While trying to solve the problems mentioned so far, it became obvious that a solution to these problems would require configuring an agent in a format that is compatible with text based tools like Git, but that also maintains or even enhances upon the maintainability and readability of Google Dialogflow configuration in the web interface. For this purpose, I designed

a domain specific language (DSL) in order to create a text-based notation (DSL code) for the configuration of an agent.

A solution like this has not been built for Dialogflow, most likely because voice interface tools like Amazon Lex or Google Dialogflow are a newer development, with Dialogflow having only existed in its current fashion since its acquisition by Google at the end of 2016 [Huf16] and the Dialogflow V2 API that I am using, and that was necessary for the success of this project currently being in Beta stage. The V2 API is generally available since April 2018 [Imr18] and “V1 of Dialogflow’s API will be deprecated on October 23, 2019” [Dia, page 1]. Dialogflow is currently still transitioning to the use of the new V2 API. It is the V2 API that makes this project possible because it includes the agent management APIs used for exporting, importing, and updating of Dialogflow agents ¹ using the JSON format.

4.2. DSL Engineering

4.2.1. What is a DSL?

In contrast to a GPL (general purpose language), a DSL (domain specific language) is a smaller and more narrow programming language, and oftentimes is not Turing complete. The advantage of a DSL comes in the form of concise syntax, that is streamlined for one specific purpose. While a GPL has to allow the developer to be able to build quite literally anything within the constraints of the language, the DSL makes no such claims but can instead provide shortcuts for the few things that it is able to do. (Compare chapter 2 in [Voe13].)

I built a DSL prototype in Xtext using the Eclipse DSL language workbench; it consists of two main parts.

Firstly, there is a grammar which defines all possible combinations of words that can be used to write in this language. Specifying this grammar allows the developer to have live IDE support, including syntax validation, while writing DSL code.

The second part I built is a code generator.

This generator compiles DSL code into a collection of JSON files in a format accepted by the Google API. It allows the developer to automatically generate a working Dialogflow agent from their DSL code.

4.2.2. Requirements for the DSL

In order to provide a tangible benefit, the DSL must fulfill a number of requirements:

- It needs to be significantly shorter than the JSON representation of an agent.
- It needs to create smaller diffs² than the JSON representation when making changes.
- It needs to be more readable than the JSON representation.
- A developer needs to be able to automatically compile and update an agent on the web from DSL code.

¹Agent is what Google calls a specific voice interface.

²A diff refers to the sum of changed lines in a change.

During the process of my work it became clear that solving the above mentioned problems made it necessary to find solutions for every one of these requirements.

4.2.3. Design Decisions

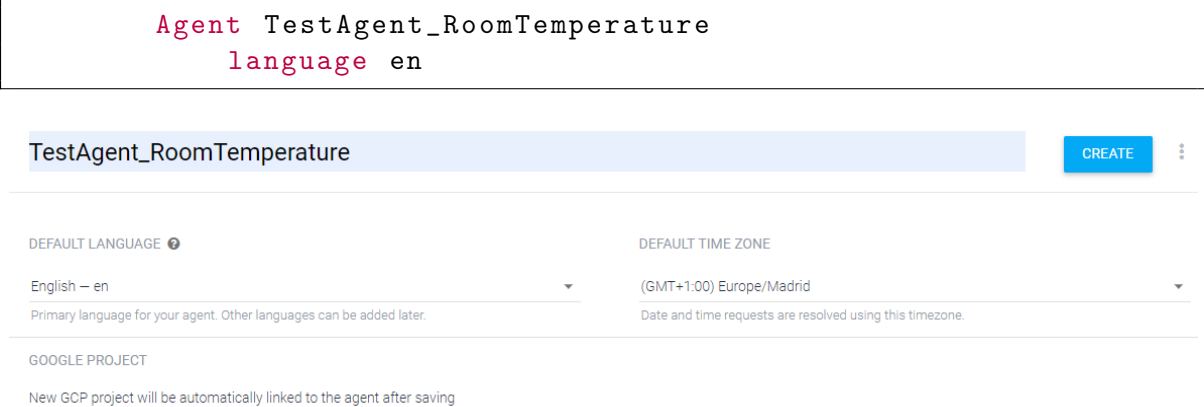
I wanted the DSL code to read naturally for a developer used to the Dialogflow interface. In order to achieve this, I tried to closely mimic the structure of the web interface in my DSL code.

In order to show this, I will walk you through an example agent I created for testing purposes. The agent is a very simple controller for regulating the air conditioning in a room. It listens to the user and sends their requests to a server. Throughout the four steps of this example, I will follow a pattern of first showing how a developer would set up an agent on the Dialogflow website, followed by showing how to do the same in DSL code.

Step 1: Creating an Agent

First a developer needs to create a new agent. To do so, they click on "create agent" and enter the agent's name and language as seen in Figure 4.1. To achieve the same effect in DSL code, a developer would enter the following:

```
1 Agent TestAgent_RoomTemperature
2 language en
```



The screenshot shows the Dialogflow console interface for creating a new agent. At the top, there is a text input field containing 'TestAgent_RoomTemperature' and a blue 'CREATE' button. Below this, there are two dropdown menus: 'DEFAULT LANGUAGE' set to 'English - en' and 'DEFAULT TIME ZONE' set to '(GMT+1:00) Europe/Madrid'. Underneath these are two lines of explanatory text. At the bottom, there is a 'GOOGLE PROJECT' section with a note: 'New GCP project will be automatically linked to the agent after saving'.

Figure 4.1. – Creating an agent on Dialogflow.

Step 2: Setting an Entity-Type

In the next step, an entity-type needs to be created to allow to intuitively turn on and off the air conditioning. In Dialogflow, this is achieved by filling in the form seen in Figure 4.2. In DSL code, the same can be done by writing the following:

```
1 Agent TestAgent_RoomTemperature
2 language en
3
4 Type ACState
5 values
6     "On" ("Active" "Enabled" "On"),
7     "Off" ("Inactive" "Disabled" "Off")
8 auto_expand
```

ACState SAVE

☒ Define synonyms ☒ Allow automated expansion

Separate synonyms by pressing the enter, tab or ; key.

Enter reference value	Enter synonym
On	On, Active, Enabled
Off	Off, Inactive, Disabled
Enter reference value	Enter synonym

[Click here to edit entry](#)

[Click here to edit entry](#)

Figure 4.2. – Creating an entity-type on Dialogflow.

Step 3: Setting a Webhook

For the agent to actually affect an air conditioner in the real world, it needs to send the user's request to a webserver. To do so, a webhook is enabled in Dialogflow by filling in the form as seen in Figure 4.3. In DSL code, the same can be achieved:

```

1  Agent TestAgent_RoomTemperature
2      language en
3
4      Webhook
5          active
6          url "https://fake.ac_controller.com/vi_webhook"
7
8      Type ACState
9          values
10             "Off" ("Inactive" "Disabled" "Off"),
11             "On" ("Active" "Enabled" "On")
12         auto_expand

```

Webhook ENABLED

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*

BASIC AUTH

HEADERS

[+ Add header](#)

SMALL TALK ☐ Disable webhook for Smalltalk

Figure 4.3. – Setting a webhook on Dialogflow.

Step 4: Writing an Intent

The last and most vital step is to create an intent that the agent can understand. This is done on the website by filling in the form seen in Figure 4.4 and can alternatively be achieved in DSL code as follows:

SwitchAC SAVE

Contexts ?

Events ?

Training phrases ? Search training phrases

Add user expression

Turn **state** the AC

Set the AC to **state**

Set the air conditioner to **state**

I want the AC turned **state**

Make sure the AC is **state**

Action and parameters

Enter action name

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	state	@ACState	\$state	<input type="checkbox"/>	Did you want th...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

+ New parameter

Figure 4.4. – Setting up an intent on Dialogflow.

```

1  Agent TestAgent_RoomTemperature
2      language en
3
4      Webhook
5          active
6          url "https://fake.ac_controller.com/vi_webhook"
7
8      Type ACState
9          values
10             "Off" ("Inactive" "Disabled" "Off"),
11             "On" ("Active" "Enabled" "On")
12          auto_expand
13
14      Intent SwitchAC
15          parameters

```



```

16         state ACState (required prompts
17             "Did you want the AC turned On or Off?")
18         trained with phrase
19             "Turn" state "the AC",
20             "Set the AC to" state,
21             "Set the air conditioner to" state,
22             "I want the AC turned" state,
23             "Make sure the AC is" state
24         webhook_fullfillment

```

4.2.4. Usage

When developing with the DSL I designed, in order to update the Dialogflow agent on the web, you run the compiler for the DSL using the DSL code as input; the output will be a full JSON representation of the Dialogflow agent which can be sent to the Dialogflow V2API. The V2API allows you to update the online version of your agent using the JSON files. For this you can either use the Dialogflow-Cli tool [Cha18a] the Dialogflow community has built, or build your own script like I did.

4.2.5. Dynamic Entity Control

In our Bachelor project, we have some entity types that are dynamically updated via the Dialogflow API, in order to stay up to date with our production database. For example, we have the team entity type, that is populated with the names of teams that are signed up to our service. This allows a user to select his team via the voice assistant. Since teams are created and deleted by the users, we need to update the entity type while the service is deployed. This makes it impossible to have a list of all teams written down in the DSL code to send to the server. To make this possible I added the keyword “dynamic” for an entity type in the DSL I designed. This keyword allows the developer to have the type exist in the DSL and be valid for syntax validation - but not have any JSON files generated for it, so as not to overwrite an entity type that is dynamically set on Dialogflow via the API.

4.3. Target Group

If you are a small group under time pressure, and you do not need to maintain the agent throughout it’s co-evolution with your application, continuing to use the Dialogflow web interface will be simpler than adopting the DSL I wrote.

However, this DSL is directed at teams of developers who will be working on an agent for an extended time, and for whom using version control is a necessity. I expect that the group of developers who will be most interested in this DSL will be those, who are currently using the approach of saving the JSON version of their agent in source control. These developers will find advantages in multiple points that I will discuss in the next chapter.

5 | Evaluation

In the following chapter I will evaluate the performance of my DSL-prototype. Testing and evaluation have been done by me, referring back to the requirements I stated earlier in chapter four. A future larger scale survey would be welcome but was not part of this evaluation. The categories I used for evaluation were:

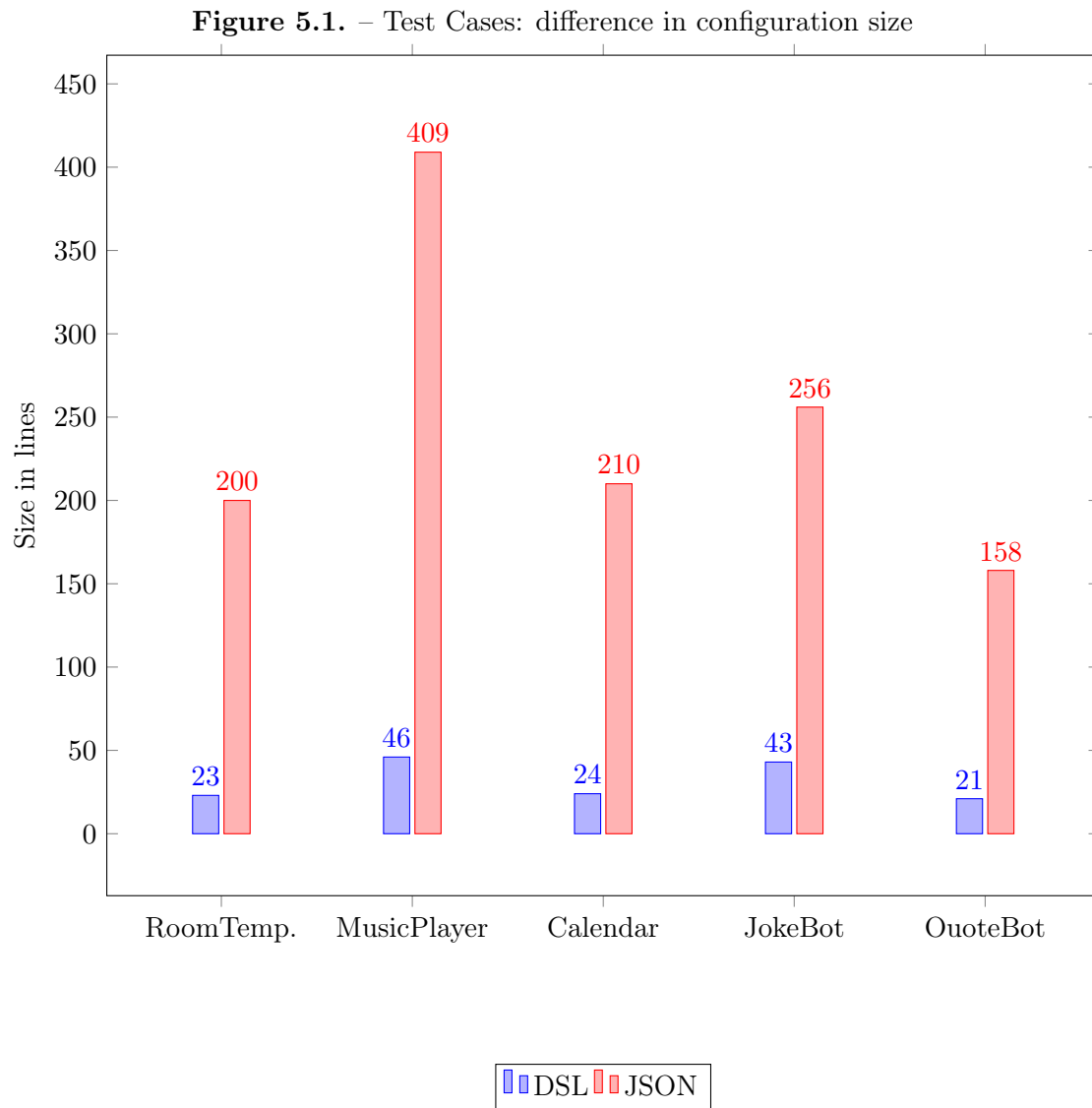
- length
- diff size
- readability
- automatic compilation and update

In the following sections I will introduce each of these categories and how the DSL prototype compares to the existing technology referring to each of them.

5.1. Length

In order to provide a tangible benefit, the DSL code must be significantly shorter than the JSON representation. Coming back to the test case agent used in chapter four, I can show that the DSL code for that agent is exactly 23 lines long, three of which are left blank for better readability (see section B.1). The JSON version for this same agent, however, comes in at exactly 200 lines (see section G.1) meaning it is almost ten times as long. This is a trend that continues accross all my five test cases, as can be seen in Figure 5.1.

This demonstrates that the DSL code for an agent is significantly shorter than the JSON version which saves the developer valuable time, e.g. when conducting a code review. This is a large benefit in and of itself, but it is amplified by the following point.



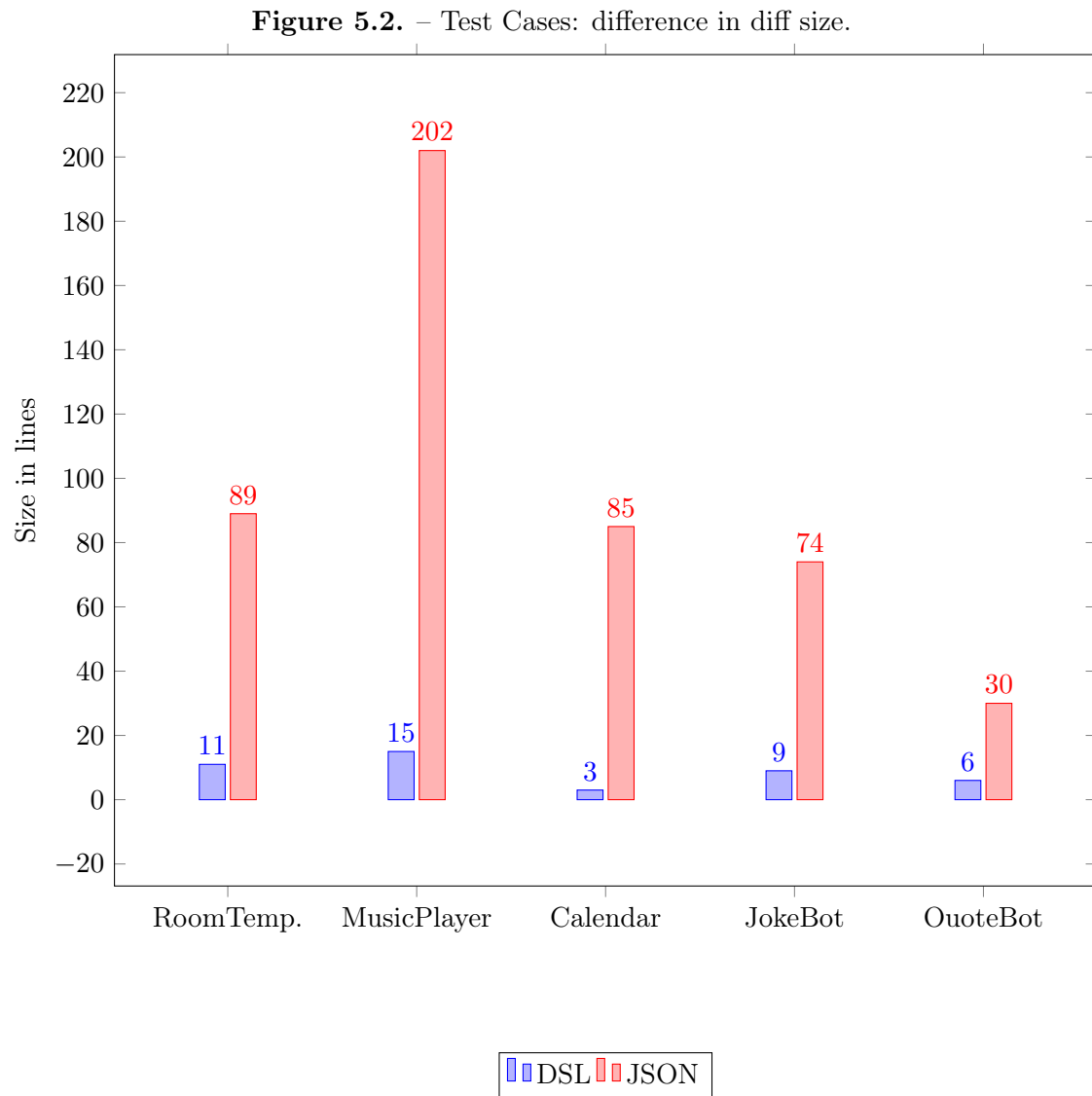
5.2. Diff Size

It is non length alone, but the DSL code must also create smaller diffs¹ than the JSON representation when making changes.

Using the air conditioning example again, when making a change - in this case adding more training phrases to an agent to allow additional functionality - the diff size shows a large difference between DSL code and JSON version. In this specific example, the DSL code has a diff of 7 lines (as seen in section C.1) and the JSON version one of 124 lines (see section H.1). An overview of all test cases is given in Figure 5.2.

It is also noteworthy that the DSL code causes less unintended diffs, as some elements of the JSON - like generated Unique Identifiers - can cause diffs that are unrelated to any intentional changes by the developer (as seen in section H.1).

¹A diff refers to the sum of changed lines in a change.



5.3. Readability

While length is an objective criterion, readability is innately more subjective, as it relates to how well a developer will be able to understand a piece of code.

For this comparison I will focus on five criteria; the first three are reworded from developer Egon Elbre's article "The psychology of code readability" [Elb18]; the last two are based on my own observations.

- Cohesive pieces of code conveying one idea at a time.
- Descriptive - but not overly long - names and keywords.
- Using idioms from natural language.
- Indentation and whitespace.
- Leaving out unnecessary values.

Cohesive pieces of code conveying one idea at a time

The following piece of JSON configuration defines a single training phrase for the SwitchAC intent, from the same example used above. The intent itself is not defined in the same file, since training phrases are kept in a separate file in the JSON configuration.

```
1      {
2          "id": "ed813a56-e6e9-4bce-b0ed-dca488102333",
3          "data": [
4              {
5                  "text": "Turn ",
6                  "userDefined": false
7              },
8              {
9                  "text": "state ",
10                 "alias": "state",
11                 "meta": "@ACState",
12                 "userDefined": true
13             },
14             {
15                 "text": "the AC ",
16                 "userDefined": false
17             }
18         ],
19         "isTemplate": false,
20         "count": 0,
21         "updated": 1560083189
22     },
```

This is equivalent to the following lines found in my SwitchAC example:

```
1      trained with phrase
2          "Turn" state "the AC"
```

As can clearly be seen, the JSON configuration is intended for automatic parsing, rather than for human readability. It is also limited by the key value and stringly typed nature of the configuration in JSON format. In my DSL, the training phrase can easily be traced back to it's intent, which is defined just three lines above in the room temperature example.

Descriptive - but not overly long - names and keywords

I chose all the keywords in my DSL to read like they would on the Dialogflow website. This means that setting a new intent uses the keyword "Intent", defining a parameter for the intent uses the "parameters" keyword, and contexts are managed be using the "contexts" keyword - followed by either "input" or "output", depending on whether input or output contexts are set.

In contrast, the JSON example I showed above contains keys with names like "data" for a training phrase, "meta" for the type a parameter corresponds to, and the "isTemplate" key which is always set to false, because it is depracated.

Using idioms from natural language

When designing the syntax for my DSL, I tried - whenever possible - to use syntax that reads like a normal sentence in natural language. This is why a developer writes:

```
1   trained with file
2   "filename"
```

in order to include a file with pre-generated training phrases. A reader can immediately understand what the otherwise not always recognizable filename refers to.

In contrast, the JSON representation is modeled after the structure of a JavaScript object and is meant to accurately describe objects and their attributes in computer progams. It does not use idioms from natural language to attempt intuitive understanding of an agent's behaviour. A specific advantage of the DSL version is that it was designed from the beginning to intuitively convey agent behaviour.

An example worth mentioning here is the definition of a fallback intent, which is an intent only used if no other intent can match the user's utterance. In the JSON representation a fallback intent is only different from a regular intent by a boolean value at the very bottom of the intent description:

```
1   "fallbackIntent": true,
```

This works perfectly fine if the intention is for the file to be parsed by a computer, but for human readability, the way the Dialogflow website handles this is much more beneficial. On the website a fallback intent is visually distinguished from normal intents and is created with a separate button. For the DSL version, I decided that a fallback intent should be defined as follows:

```
1   fallback Intent DefaultFallbackIntent
2   response
3       "I didn\u0027t get that. Can you say it again?"
4       "I missed what you said. What was that?"
```

```
5         "Sorry, could you say that again?"
6         "Sorry, can you say that again?"
7         "Can you say that again?"
8         "Sorry, I didn\u0027t get that. Can you rephrase?"
9         "Sorry, what was that?"
10        "One more time?"
11        "What was that?"
12        "Say that one more time?"
13        "I didn\u0027t get that. Can you repeat?"
14        "I missed that, say that again?"
15    action 'input.unknown'
```

This example is the default fallback intent that every agent on the Dialogflow website is created with. Note that the definition begins with *fallback Intent* instead of *Intent*. This immediately makes it clear to the reader that this intent should be read as a fallback and not as a regular intent.

Indentation and whitespace

An additional benefit for readability is the ability to separate a program into meaningful paragraphs.

This can be seen in section B.1, where between each instruction I left a blank line. This helps visually separate the code into meaningful units.

As a further project, but not possible in the scope of this thesis, an automatic linting and formatting tool for the DSL could be considered. Similar tools have been developed for general programming languages numerous times with a very popular example being Black for Python [Pyt] and numerous other examples existing [Gita].

Leaving out unnecessary values

One of the most important ways of reducing clutter in the agent configuration - and of thereby reducing the configuration's size - is to leave out unnecessary values like unchanged default values and empty values. If an agent does not have a description, the JSON representation will carry the following line.

```
1    "description": "",
```

Instead, the DSL version simply will not have a line concerning the description, since it is not changed from its default value of being an empty string. The same is done for a large number of settings that a developer is able to change in DSL code but that have a default value in Dialogflow. They need not be displayed in DSL code if the default value is not changed.

5.4. Automatic compilation and update

A developer needs to be able to automatically compile and update an agent on the web from DSL code. The compiler created alongside this prototype² is able to automatically generate the JSON representation of an agent from DSL code. Alongside this, I have written a script that automatically sends the resulting JSON files to the Dialogflow website. This can be used in continuous integration setups like CircleCI [Cir] to automatically keep the online version of a Dialogflow agent up to date with the most current version of the DSL code in source control.

Below, you can see an example of the CircleCI job configuration that I set up for the prototype in our Bachelor project.³

```

1      deploy_agent:
2          docker:
3              - image: circleci/openjdk:latest-node
4          steps:
5              - checkout
6              - run: >
7                  wget -O ./dfc_compiler.jar
8                      https://github.com/arne-z/BachelorThesis/...
9              - run: yarn install
10             - run: > java -jar ./dfc_compiler.jar
11                 ./Agent/Tobito.dfc
12             - run: > echo $GOOGLE_CERT_FILE_64
13                 | base64 --decode > ./googleKey.json
14             - run: >
15                 node ./utility/importAgent.js
16                 --dir ./src-gen
17                 --key ./googleKey.json
18                 --pid projects/newagent-bdb60

```

The above configuration starts a docker image with Java and Node installed - Java is required for the DSL compiler, Node for the import script - and checks out the most recent version of our project from source control. It then downloads the DSL compiler from my github repository and runs the compiler targeting the .dfc file containing the DSL code for our project's agent. Afterwards, it runs the importAgent script that will send the JSONs generated by the compiler to the Dialogflow API.

This setup allows a "hands free" approach to developing for Dialogflow, where all the developer has to do is change the DSL code; once that is pushed to Git, the Dialogflow agent is updated automatically.

5.5. Concessions and Drawbacks

Working with the DSL instead of Dialogflow directly comes with some drawbacks, as I had to make certain concessions during development. These concessions can be divided into four

²<https://github.com/arne-z/BachelorThesis/releases>

³The full configuration set up by our Bachelor project team can be found here:

<https://github.com/hpi-sam/ask-your-repository-dialogflow-adapter/blob/agent-config-with-dsl/.circleci/config.yml>

categories.

5.5.1. Design Choices differing from the Dialogflow Website

As I stated before, I attempted to stay as close to the Dialogflow website as possible, but there are some elements of the website that do not lend themselves to being translated into a text based configuration.

The way parameters are entered in Dialogflow is an interesting example of this, as is shown in Figure 4.4. Translating to a text based interface with the requirement of live syntax validation, it became necessary to enter the parameters allowed in training phrases before defining the phrases. Similarly, Dialogflow switched from what is called *template mode*, where a training phrase would be described as, e.g.: "Turn state the AC", to what is called *example mode*, which means the phrase would be: "Turn on the AC" - with the "on" being annotated to show that it is meant to represent a number of possible values. In order to translate this to a text based interface, I considered a solution that would have looked as follows:

```
1      Intent SampleIntent
2          trained with phrase
3          "Turn {on : state} the AC"
```

I eventually decided against this, as I perceived this solution to make both writing and reading the training phrases unnecessarily difficult. Instead I decided to continue using the template format for the DSL as seen below:

```
1      Intent SampleIntent
2          trained with phrase
3          "Turn" state "the AC"
```

This works and is fairly easy to read but is still a compromise and doesn't quite allow the same user experience as the Dialogflow website.

5.5.2. The Difficulty of Deciding on Defaults.

A large part of what makes both the web interface and the DSL easier to work with than the JSON files, is that here a developer can assume all the defaults to be reasonable. When first creating a new Dialogflow agent, all the default settings are already set for you to start working with. I found that for my tool I was able to use the same defaults that the website uses, so that the same workflow of creating an agent arrives at the same results on the website and in the DSL.

This led to a bit of a problem when it came to the two default intents that a Dialogflow agent is created with, the "Default Fallback Intent" and the "Default Welcome Intent".

I considered adding these default intents during compilation, unless the developer specifically disabled this, but I found that to be too unintuitive. On the Dialogflow website, a developer can always see the list of intents in the agent, including these two default intents on the intents panel, but in a text based interface the developer cannot easily be informed of the default intents existing in the background. Instead, I decided that only the intents described in the developer's code should be in the compiled agent.

My plan is to add a shortcut for enabling the default intents via a `use __default__ intents` flag,

so that the developer can easily use the default intents, and anyone reading the code will still know these intents are enabled.

5.5.3. Unsupported Dialogflow Features

Dialogflow is a large project, much bigger in scope than this thesis, so it was immediately apparent that my prototype would not be able to support all of the features and settings that are available in Dialogflow. Instead, I selected the features that were necessary to support development in my Bachelor project. This means that some projects might be unable to use this prototype, as they rely on features that are not supported.

6 | Summary and further Work

It was my self-set task for this Bachelor thesis to find a solution for shortcomings in available version control for Dialogflow agents. In order to accomplish this, I built a prototype of a DSL for describing Dialogflow agents. When starting this project my assumption was, that there should and could be a solution for using version control with a Dialogflow agent that is simpler, shorter, and more readable than saving the JSON configuration of the agent in source control. The solution I eventually built is aimed at teams of developers working longer term on maintaining an agent throughout its coevolution with their application. I arrived at this solution because, from the very beginning, I wanted to find a way for developers to work with dialogflow without leaving their familiar way of working.

In order to build this DSL, I had to design a grammar to describe the syntax possible in the language and build a code generator using this grammar to generate JSON files.

In order to test and evaluate this prototype, I designed five test cases in each of which I compared the performance of the prototype with that of the established way of saving the agent configuration using JSON files.

In result, I have found that the DSL provides a measurable improvement in both the size of agent configurations and the size of diffs created by changes to the configuration - to be more precise: a 87.15% reduction in size and a 88.9% reduction in diff size on average respectively (see Appendix D) - while also improving readability and almost entirely maintaining the ease of updating the agent's online version.

Teams working with this tool will find that it offers benefits in regard to teamwork, learning, consistency, and keeping track of their version history. The code for my prototype is available online, free and open source.

There are a number of questions still unanswered, and as I mentioned above, some improvements to the DSL are still possible. Some further projects one could consider are:

- A larger survey for better evaluation of long term benefits to teams working with this tool.
- Finding out whether it is possible to use a DSL like this to create agents in a uniform language for both Dialogflow and Amazon Lex.
- Build a production-ready tool from the prototype that achieves full feature parity with the Dialogflow web interface.
- Build a generator for DSL code from Dialogflow JSON exports, to help teams reduce the cost of switching to the DSL solution.

For the nearest future, my tool will help my Bachelor project team with working on our own Dialogflow agent.

Bibliography

- [Alt19] Alternativeto. *Dialogflow Alternatives and Similar Websites and Apps*. 2019. URL: <https://alternativeto.net/software/api-ai/> (visited on 06/23/2019).
- [Cha18a] Martin Charles. *dialogflow-cli*. 2018. URL: <https://github.com/0xcaff/dialogflow-cli>.
- [Cha18b] BRAIN [BRN.AI] White Label Chatbots. *Chatbot Report 2018: Global Trends and Analysis*. 2018. URL: <https://chatbotsmagazine.com/chatbot-report-2018-global-trends-and-analysis-4d8bbe4d924b> (visited on 06/21/2019).
- [Cir] CircleCI. *CircleCI*. URL: <https://circleci.com/> (visited on 06/23/2019).
- [Dia] Dialogflow. *Migrate from API V1 to API V2*. URL: <https://dialogflow.com/docs/reference/v1-v2-migration-guide> (visited on 06/21/2019).
- [Elb18] Egon Elbre. *Psychology of Code Readability*. 2018. URL: <https://medium.com/@egonelbre/psychology-of-code-readability-d23b1ff1258a> (visited on 06/14/2019).
- [Gita] Github. *#codeformatter github*. URL: <https://github.com/topics/codeformatter> (visited on 06/21/2019).
- [Gitb] Github. *Github Repository ID:191000000*. URL: <https://api.github.com/repositories?since=191000000> (visited on 06/23/2019).
- [Huf16] Scott Huffman. *Making Conversational Interfaces Easier to Build*. 2016. URL: <https://developers.googleblog.com/2016/09/making-conversational-interfaces-easier-to-build.html> (visited on 06/21/2019).
- [Imr18] Daniel Imrie-Situnayake. *Dialogflow API V2 and Enterprise Edition are now generally available*. 2018. URL: <https://blog.dialogflow.com/post/v2-and-enterprise-edition-generally-available/> (visited on 06/21/2019).
- [OK19] Christi Olson and Kelli Kemery. *Voice Report*. Tech. rep. 2019, p. 44.
- [Oos19] Ruben Oostinga. *Developing for Google Assistant with Dialogflow*. 2019. URL: <https://xebia.com/blog/developing-for-google-assistant-with-dialogflow/> (visited on 06/21/2019).
- [Pet] Dmitry Petrov. *Open-source Version Control System for Machine Learning Projects*. URL: <https://dvc.org/> (visited on 06/23/2019).
- [Pyt] Python. *Black The Uncompromising Code Formatter*. URL: <https://github.com/python/black>.
- [Sta] Stackshare. *Amazon Lex vs Dialogflow*. URL: <https://stackshare.io/stackups/amazon-lex-vs-dialogflow> (visited on 06/14/2019).
- [Voe13] Markus Voelter. *Introduction to DSLs*. 2013. ISBN: 978-1481218580. URL: <http://voelter.de/dslbook/markusvoelter-dslengineering-1.0.pdf>.

A | Declaration of Authorship

Declaration

I hereby declare that the thesis submitted is my own unaided work and that I did not use any other sources and aids than those referenced.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dafür keine anderen als die genannten Quellen und Hilfsmittel verwendet habe.

Potsdam, den 27. Juni 2019

Arne Zerndt

B | DSL Code of Test Agents

The following examples are in some cases formatted differently from the way they were during my tests, leading to slightly differing line numbers. They were reformatted to fit into the printed version. Original files can be found on my GitHub.¹

B.1. RoomTemperature Agent

```
1 Agent TestAgent_RoomTemperature
2   language en
3
4   Webhook
5     active
6     url "https://fake.ac_controller.com/vi_webhook"
7
8   Type ACState
9     values
10      "Off" ("Inactive" "Disabled" "Off"),
11      "On" ("Active" "Enabled" "On")
12   auto_expand
13
14   Intent SwitchAC
15     parameters
16       state ACState (required prompts
17         "Did you want the AC turned On or Off?")
18   trained with phrase
19     "Turn" state "the AC",
20     "Set the AC to" state,
21     "Set the air conditioner to" state,
22     "I want the AC turned" state,
23     "Make sure the AC is" state
24   webhook_fullfillment
```

¹https://github.com/arne-z/BachelorThesis/tree/master/dsl_code

B.2. MusicPlayer Agent

```
1 Agent MusicPlayer
2   language en
3
4   Webhook
5     active
6     url "https://fake.music_controller.com/vi_webhook"
7
8   Intent PlaySong
9     parameters
10      Artist music_artist
11      Genre music_genre
12      Title any
13    trained with phrase
14      "Play a " Genre "song",
15      "Play some" Genre,
16      "Play a song by "Artist,
17      "Play" Title "by" Artist,
18      "Play" Title,
19      "Play a random song"
20    webhook_fullfillment
21
22   Intent StopMusic
23     action "StopMusic"
24     trained with phrase
25       "Pause",
26       "Stop",
27       "Quiet",
28       "Stop the Music"
29     webhook_fullfillment
30
31   Intent VolUp
32     action "VolUp"
33     trained with phrase
34       "Louder",
35       "More Volume",
36       "Too quiet",
37       "Volume Up"
38     webhook_fullfillment
39
40   Intent VolDown
41     action "VolDown"
42     trained with phrase
43       "Lower Volume",
44       "Too loud",
45       "Volume Down"
46     webhook_fullfillment
```


B.3. Calendar Agent

```
1 Agent Calendar
2   language en
3
4   Webhook
5     active
6     url "https://fake.calendar_controller.com/vi_webhook"
7
8   Type Entry
9     values
10      "Appointment",
11      "Reminder",
12      "Event"
13     auto_expand
14
15   Intent AddEntry
16     parameters
17       Entry Entry
18       Time date_time (required prompts "For when should I
19         enter that entry?")
20       Title any (required prompts "What should I call the entry?")
21     trained with phrase
22       "Add a new" Entry "for" Time,
23       "Add a new" Entry,
24       "Add a" Entry "at" Time Title
25     webhook_fullfillment
```

B.4. JokeBot Agent

```
1 Agent JokeBot
2   language en
3
4   Webhook
5     active
6     url "https://fake.rate_my_jokes.com/vi_webhook"
7
8   Intent DadJoke
9     contexts
10      output joke
11      trained with phrase
12        "Tell me a Dad joke",
13        "Do you know any stupid jokes?",
14        "Do you know any dad jokes?"
15      response
16        "What is brown and sticky? ... A stick."
17        "My wife is really mad at the fact that I have no
18          sense of direction. So I packed up my stuff and right."
19        "I bought some shoes from a drug dealer. I don't know what
20          he laced them with, but I was tripping all day!"
21        "Why can't you hear a pterodactyl go to the bathroom?
22          Because the pee is silent."
23
24   Intent RateJokePositive
25     contexts
26       input joke
27       action "ratePositive"
28       trained with phrase
29         "Wow that was funny",
30         "That was awesome",
31         "I liked that joke"
32       response
33         "Glad you liked it."
34       webhook_fullfillment
35
36   Intent RateJokeNegative
37     contexts
38       input joke
39       action "rateNegative"
40       trained with phrase
41         "Bad joke",
42         "Don't tell that joke again",
43         "Thanks, I hate it."
44       response
45         "Well that's your loss, I like the Joke."
46       webhook_fullfillment
```

B.5. QuoteBot Agent

```
1 Agent QuoteBot
2   language en
3
4   Intent OscarWildeQuote
5     trained with phrase
6       "Tell me a quote by Oscar Wilde",
7       "What would Oscar Wilde say"
8     response
9       "To live is the rarest thing in the world.
10        Most people exist, that is all."
11       "Only dull people are brilliant at breakfast."
12       "I think God, in creating man,
13        somewhat overestimated his ability."
14       "Democracy means simply the bludgeoning of the people
15        by the people for the people."
16
17   Intent EinsteinQuote
18     trained with phrase
19       "Tell me a quote by Albert Einstein",
20       "What would Einstein say"
21     response
22       "Two things are infinite: the universe and
23        human stupidity; and I'm not sure about the universe."
24       "If you can't explain it to a six year old,
25        you don't understand it yourself."
26       "Never memorize something that you can look up."
```

C | Examples of Diffs created from Test Agents

C.1. RoomTemperature Agent

```
1  @@ -26,9 +26,14 @@ Agent TestAgent_RoomTemperature
2      Intent ChangeTemperature
3      parameters
4          temp temperature (required prompts "What temperature
5              would you like the AC set to?")
6  +      point time
7      trained with phrase
8          "Set the AC to" temp,
9          "Set the air conditioner to" temp,
10         "I want the AC turned to" temp,
11  -     "Make sure the AC is at" temp
12  +     "Make sure the AC is at" temp,
13  +     "Set the AC to" temp "at" point,
14  +     "Set the air conditioner to" temp "at" point,
15  +     "I want the AC turned to" temp "at" point,
16  +     "Make sure the AC is at" temp "at" point
17      webhook_fullfillment
```

C.2. Test Agents 2-5

The diffs created from the other four test agents are not printed here to prevent the appendix from becoming too long. They can be found on my GitHub instead.¹

¹See <https://github.com/arne-z/BachelorThesis/commits/master> all commits that are prefixed with "Test Case".

D | Test Case Data and Averages

D.1. Line Length of Test Agents

TestCase	DSL	JSON	Factor	Reduction in %
1	23	200	8,70	88,50%
2	46	409	8,89	88,75%
3	24	210	8,75	88,57%
4	43	256	5,95	83,20%
5	21	158	7,52	86,71%
Average	-	-	7,96	87,15%

D.2. Diff Size of Changes to each Test Agent

TestCase	DSL	JSON	Factor	Reduction in %
1	11	89	8,09	87,64%
2	15	202	13,47	92,57%
3	3	85	28,33	96,47%
4	9	74	8,22	87,84%
5	6	30	5,00	80,00%
Average	-	-	12,62	88,90%

E | DSL Grammar

```
1 grammar org.xtext.DialogflowConfig
2     with org.eclipse.xtext.common.Terminals
3
4     generate dialogflowConfig "http://www.xtext.org/DialogflowConfig"
5
6     Agent :
7         'Agent' name=ID
8         'language' language=Language
9         (('description' description=STRING)?
10        & ('version' version=VERSION)?
11        & ('ml_classification_threshold' mlMinConfidence=DOUBLE)?
12        & webhook=Webhook?
13        & interactionLogs?='disable_logs'?
14        & stackdriverLogs?='log_to_google_cloud'?
15        & isPublic?='public'?
16        & noHybridMatchMode?='no_hybrid_match_mode'?)?
17        elements+=AbstractElement*;
18
19     terminal DOUBLE:
20         INT '.' INT;
21
22     terminal VERSION:
23         INT '.' INT '.' INT;
24
25     AbstractElement :
26         Intent | EntityType;
27
28     Intent :
29         fallback?='fallback'?
30         'Intent' name=ID
31         ('contexts'
32         ('input' inputContexts+=InputContext+)?
33         ('output' affectedContexts+=OutputContext+)?)?
34         & ('parameters' parameters+=Parameter+)?
35         & (('trained' 'with' 'phrase' trainingPhrases+=TrainingPhrase
36         (',' trainingPhrases+=TrainingPhrase)*)
37         | ('trained' 'with' 'file' file=STRING))?
38         & ('response' responses+=STRING+)?
39         & ('action' action=STRING)?
```

```

40     & ( 'events ' events+=STRING+)?
41     & webHookFulfillment?='webhook_fullfillment'?
42     & webHookForSlotFilling?='webhook_slot_filling'?
43     & disable_ml?='disable_ml'?
44     & end?='end_of_conversation'?;
45
46 InputContext:
47     name=ID;
48
49 OutputContext:
50     name=ID ( 'lifespan ' lifespan=INT)?;
51
52 TrainingPhrase:
53     data+=AbstractWord+;
54
55 AbstractWord:
56     Text | customToken;
57
58 Text:
59     text=STRING;
60
61 customToken:
62     param=[Parameter];
63
64 Parameter:
65     name=ID
66     ( type=[EntityType] | builtInType=BuiltInType)
67     ( '('
68     ( required?='required'?
69     & ( 'prompts ' prompts+=STRING+)?
70     & list?='list'? )
71     ')')?;
72
73 EntityType:
74     'Type' name=ID
75     ( dynamic?='dynamic' |
76     'values ' values+=Entity ( ',' values+=Entity)*)
77     & isOverridable?='overridable'?
78     & isEnum?='enum'?
79     & automatedExpansion?='auto_expand'?
80     & allowFuzzyExtraction?='fuzzy_extract'?;
81
82 Entity:
83     name=STRING
84     ( '(' synonyms+=STRING* ')')?;
85
86 Webhook:
87     'Webhook'
88     available?='active'?
89     ( 'url ' url=STRING)

```

```
90      ( 'headers' headers+=Header+ )?;
91
92  Header :
93      key=STRING ':' value=STRING;
94
95  enum Language :
96      en |
97      de |
98      fr |
99      es |
100     da |
101     hi |
102     id |
103     it |
104     ja |
105     ko |
106     ni |
107     pl |
108     pt |
109     ru |
110     sv |
111     th |
112     tr |
113     uk;
114
115  enum BuiltInType :
116     date_time
117     | date
118     | date_period
119     | time
120     | time_period
121     | number
122     | cardinal
123     | ordinal
124     | number_integer
125     | number_sequence
126     | flight_number
127     | unit_area
128     | unit_currency
129     | unit_length
130     | unit_speed
131     | unit_volume
132     | unit_weight
133     | unit_information
134     | percentage
135     | temperature
136     | duration
137     | age
138     | currency_name
139     | unit_area_name
```


140		unit_length_name
141		unit_speed_name
142		unit_volume_name
143		unit_weight_name
144		unit_information_name
145		address
146		zip_code
147		geo_capital
148		geo_country
149		geo_country_code
150		geo_city
151		geo_state
152		place_attraction
153		airport
154		location
155		email
156		phone_number
157		given_name
158		last_name
159		person
160		music_artist
161		music_genre
162		color
163		language
164		any
165		url;

F | DSL Code Generator

This file is written in Xtend, a Java based language for implementing code generators. It uses some characters that are not supported outside of Xtend, which denote the beginning or end of a code block inside a multi-line string. I have replaced each of these characters with a `#` below.

```
1  /*
2  * generated by Xtext 2.16.0
3  */
4  package org.xtext.generator
5
6  import java.io.FileNotFoundException
7  import java.util.Date
8  import java.util.UUID
9  import org.eclipse.emf.ecore.resource.Resource
10 import org.eclipse.xtext.generator.AbstractGenerator
11 import org.eclipse.xtext.generator.IFileSystemAccess2
12 import org.eclipse.xtext.generator.IGeneratorContext
13 import org.xtext.dialogflowConfig.Parameter
14 import org.xtext.dialogflowConfig.impl.AgentImpl
15 import org.xtext.dialogflowConfig.impl.EntityTypeImpl
16 import org.xtext.dialogflowConfig.impl.IntentImpl
17 import org.xtext.dialogflowConfig.impl.TextImpl
18 import org.xtext.dialogflowConfig.impl.customTokenImpl
19
20 /**
21 * Generates code from your model files on save.
22 *
23 * See https://www.eclipse.org/Xtext/documentation/303\_runtime\_concepts.html#code-generation
24 */
25 */
26 class DialogflowConfigGenerator extends AbstractGenerator {
27
28     override void doGenerate(Resource resource ,
29         IFileSystemAccess2 fsa , IGeneratorContext context) {
30         // .dfc file can only contain one Agent.
31         val agent = resource.contents.filter(AgentImpl).get(0)
32
33         val intents = agent.elements.filter(IntentImpl)
34         val entityTypes = agent.elements.filter(EntityTypeImpl)
```

```

35
36     generateAgentFile(fsa , agent)
37
38     generatePackageFile(fsa , agent)
39
40     for (intent : intents) {
41         generateIntentFile(fsa , agent , intent)
42         generateIntentUsersaysFile(fsa , agent , intent)
43     }
44
45     for (entityType : entityTypees) {
46         generateEntityFile(fsa , agent , entityType)
47         generateEntityUsersaysFile(fsa , agent , entityType)
48     }
49 }
50
51 protected def void generatePackageFile(IFileSystemAccess2 fsa ,
52     AgentImpl agent) {
53     fsa.generateFile(''#agent.name#/package.json'', '')
54     {
55         "version": #IF agent.version != null ##
56             agent.version##ELSE#"1.0.0"#ENDIF#
57     }
58     '')
59 }
60
61 protected def void generateAgentFile(IFileSystemAccess2 fsa ,
62     AgentImpl agent) {
63     fsa.generateFile(
64         '''#agent.name#/agent.json'',
65         '',
66         {
67             "description": "#IF agent.description
68                 != null##agent.description##ENDIF#",
69             "language": "#agent.language#",
70             "disableInteractionLogs": #agent.interactionLogs#,
71             "disableStackdriverLogs": #!agent.stackdriverLogs#,
72             #IF agent.webhook != null#
73             "webhook": {
74                 #IF agent.webhook.url != null#
75                 "url": "#agent.webhook.url#",
76                 #ENDIF#
77                 #IF !agent.webhook.headers.empty#
78                 "headers": {
79                     #FOR header: agent.webhook.headers#
80                     #IF header
81                     != agent.webhook.headers.get(0)#,#ENDIF#
82                     "#header.key#": "#header.value#"
83                     #ENDFOR#
84                 },

```

```
85         #ENDIF#
86         "available": #agent.webhook.available#,
87         "useForDomains": false ,
88         "cloudFunctionsEnabled": false ,
89         "cloudFunctionsInitialized": false
90     },
91 #ENDIF#
92     "isPublic": #agent.isPublic#,
93     "customClassifierMode": #IF
94         agent.noHybridMatchMode#"use.instead"#
95         ELSE#"use.after"#ENDIF#,
96     "mlMinConfidence": #IF agent.mlMinConfidence
97         != null# #agent.mlMinConfidence# #ELSE# 0.3 #ENDIF#,
98     "onePlatformApiVersion": "v2"
99     }
100     ,,,
101 )
102 }
103
104 protected def void generateEntityUsersaysFile(IFFileSystemAccess2 fsa ,
105     AgentImpl agent, EntityTypeImpl entityType) {
106     if(entityType.dynamic) return;
107     fsa.generateFile(
108         ""#agent.name#/entities/#entityType.name
109         #_entries_#agent.language#.json "" ,
110         ,,,
111         [
112             #FOR entity : entityType.values#
113             #IF entity != entityType.values.get(0)#,#ENDIF#
114             {
115                 "value": "#entity.name#",
116                 #IF entity.synonyms.isEmpty#
117                 "synonyms": [
118                     "#entity.name#"
119                 ]
120                 #ELSE#
121                 "synonyms": [
122                     #FOR synonym : entity.synonyms#
123                     #IF synonym != entity.synonyms.get(0)#,#ENDIF#
124                     "#synonym#"
125                     #ENDFOR#
126                 ]
127                 #ENDIF#
128             }
129             #ENDFOR#
130         ]
131         ,,,
132     )
133 }
134
```

```

135 protected def void generateEntityFile(IFileSystemAccess2 fsa ,
136   AgentImpl agent, EntityTypeImpl entityType) {
137   if(entityType.dynamic) return;
138   fsa.generateFile(
139     '''#agent.name#/entities/#entityType.name#.json''' ,
140     '''
141     {
142       "id": "#UUID.randomUUID()#",
143       "name": "#entityType.name#",
144       "isOverridable": #entityType.isOverridable#,
145       "isEnum": #entityType.isEnum#,
146       "automatedExpansion": #entityType.automatedExpansion#,
147       "allowFuzzyExtraction": #entityType.allowFuzzyExtraction#
148     }
149     '''
150   )
151 }
152
153 protected def void generateIntentUsersaysFile(
154   IFileSystemAccess2 fsa, AgentImpl agent, IntentImpl intent) {
155   if (!intent.trainingPhrases.empty) {
156     fsa.generateFile(
157       '''#agent.name#/intents/#intent.name
158       #_usersays_#agent.language#.json''' ,
159       '''
160       [
161       #FOR phrase : intent.trainingPhrases#
162       #IF phrase != intent.trainingPhrases.get(0)#,#ENDIF#
163       {
164         "id": "#UUID.randomUUID()#",
165         "data": [
166         #FOR datum: phrase.data#
167         #IF datum != phrase.data.get(0)#,#ENDIF#
168         #IF datum instanceof customTokenImpl#
169         #IF datum.param.type != null#
170         {
171           "text": "#datum.param.name# ",
172           "alias": "#datum.param.name#",
173           "meta": "@#datum.param.type.name#",
174           "userDefined": true
175         }
176         #ELSE#
177         {
178           "text": "#datum.param.name# ",
179           "alias": "#datum.param.name#",
180           "meta": "@sys.#
181             datum.param.builtInType.toString().replace('_', '-')#",
182           "userDefined": true
183         }
184         #ENDIF#

```

```
185         #ELSEIF datum instanceof TextImpl#
186         {
187             "text": "#datum.text# ",
188             "userDefined": false
189         }
190         #ENDIF#
191     #ENDFOR#
192     ],
193     "isTemplate": false ,
194     "count": 0,
195     "updated": #new Date().time/1000#
196     }
197 #ENDFOR#
198     ]
199     ,,,
200 )
201 } else if (intent.file != null) {
202     try {
203         fsa.generateFile(
204             '''#agent.name#/intents/#intent.name#_usersays_#
205             agent.language#.json''',
206             fsa.readTextFile('''.../#intent.file#''')
207         )
208     } catch (FileNotFoundException e) {
209         return
210     }
211 }
212 }
213
214 private static final class Param {
215     String datatype;
216     String name;
217     String value;
218 }
219
220 protected def Param getParamTypeName(Parameter param) {
221     var obj = new Param();
222     if (param.type != null) {
223         obj.datatype = "@" + param.type.name
224     } else {
225         obj.datatype = '@sys.' +
226             param.builtInType.toString().replace('_', '-');
227     }
228     obj.name = param.name;
229     obj.value = '$' + param.name;
230     return obj;
231 }
232
233 protected def void generateIntentFile(IFileSystemAccess2 fsa ,
234     AgentImpl agent, IntentImpl intent) {
```

```

235 fsa.generateFile(
236     '''#agent.name#/intents/#intent.name#.json''' ,
237     '''
238     {
239         "id": "#UUID.randomUUID()#",
240         "name": "#intent.name#",
241         "auto": #!intent.disable_ml#,
242         "contexts": [
243             #FOR context : intent.inputContexts#
244                 #IF context != intent.inputContexts.get(0)#,#ENDIF#
245                 "#context.name#"
246             #ENDFOR#
247         ],
248         "responses": [
249             {
250                 "resetContexts": false ,
251                 #IF intent.action != null#
252                 "action": "#intent.action#",
253                 #ENDIF#
254                 "affectedContexts": [
255                     #FOR context : intent.affectedContexts#
256                     #IF context != intent.affectedContexts.get(0)
257                     #,#ENDIF#
258                     {
259                         "name": "#context.name#",
260                         "parameters": {},
261                         "lifespan": #if(context.lifespan > 0)
262                             {context.lifespan} else {5}#
263                     }
264                     #ENDFOR#
265                 ],
266                 "parameters": [
267                     #FOR param : intent.parameters#
268                     #IF param != intent.parameters.get(0)#,#ENDIF#
269                     {
270                         "id": "#UUID.randomUUID()#",
271                         "required": #param.required#,
272                         "dataType": "#getParamTypeName(param).datatype#",
273                         "name": "#getParamTypeName(param).name#",
274                         "value": "#getParamTypeName(param).value#",
275                         #FOR prompt: param.prompts#
276                         "prompts": [
277                             {
278                                 "lang": "#agent.language#",
279                                 "value": "#prompt#"
280                             }
281                         ],
282                         #ENDFOR#
283                         "isList": #param.list#
284                     }

```

```
285         #ENDFOR#
286     ],
287     #IF !intent.responses.empty#
288     "messages": [
289         {
290             "type": 0,
291             "lang": "#agent.language#",
292             "speech":
293             [
294                 #FOR response : intent.responses#
295                 #IF response != intent.responses.get(0)#,#ENDIF#
296                 "#response#"
297                 #ENDFOR#
298             ]
299         }
300     ],
301     #ENDIF#
302     "defaultResponsePlatforms": {},
303     "speech": []
304 }
305 ],
306 "priority": 500000,
307 "webhookUsed": #intent.webHookFulfillment#,
308 "webhookForSlotFilling": #intent.webHookForSlotFilling#,
309 "lastUpdate": #new Date().time/1000#,
310 "fallbackIntent": false,
311 "events": [
312     #FOR event : intent.events#
313     #IF event != intent.events.get(0)#,#ENDIF#
314     {
315         "name": "#event#"
316     }
317     #ENDFOR#
318 ]
319 },
320 },
321 )
322 }
323 }
```


G | JSON Versions of Test Agents

G.1. RoomTemperature Agent

G.1.1. Agent

agent.json

```
1 {
2   "description": "",
3   "language": "en",
4   "disableInteractionLogs": false,
5   "disableStackdriverLogs": true,
6   "webhook": {
7     "url": "https://fake.ac_controller.com/vi_webhook",
8     "available": true,
9     "useForDomains": false,
10    "cloudFunctionsEnabled": false,
11    "cloudFunctionsInitialized": false
12  },
13  "isPublic": false,
14  "customClassifierMode": "use.after",
15  "mlMinConfidence": 0.3,
16  "onePlatformApiVersion": "v2"
17 }
```

package.json

```
1 {
2   "version": "1.0.0"
3 }
```

G.1.2. Entities

ACState.json

```
1 {
2   "id": "31b65f06-3c68-4335-bedb-7eed2c98404d",
3   "name": "ACState",
4   "isOverridable": false,
5   "isEnum": false,
6   "automatedExpansion": true,
7   "allowFuzzyExtraction": false
8 }
```

ACState_entries_en.json

```
1 [
2   {
3     "value": "Off",
4     "synonyms": [
5       "Inactive",
6       "Disabled",
7       "Off"
8     ]
9   },
10  {
11    "value": "On",
12    "synonyms": [
13      "Active",
14      "Enabled",
15      "On"
16    ]
17  }
18 ]
```

G.1.3. Intents

SwitchAC.json

```
1 {
2   "id": "1b3f96d9-3e41-4ccc-95be-601237d1d20d",
3   "name": "SwitchAC",
4   "auto": true,
5   "contexts": [],
6   "responses": [
7     {
8       "resetContexts": false,
9       "affectedContexts": [],
10      "parameters": [
11        {
12          "id": "9a814870-de5f-4547-ab9a-42ef2fd73040",
```

```
13         "required": true,
14         "dataType": "@ACState",
15         "name": "state",
16         "value": "$state",
17         "prompts": [
18             {
19                 "lang": "en",
20                 "value": "Did you want the AC turned On or Off?"
21             }
22         ],
23         "isList": false
24     }
25 ],
26 "defaultResponsePlatforms": {},
27 "speech": []
28 }
29 ],
30 "priority": 500000,
31 "webhookUsed": true,
32 "webhookForSlotFilling": false,
33 "lastUpdate": 1560426984,
34 "fallbackIntent": false,
35 "events": []
36 }
```

SwitchAC_usersays_en.json

```
1 [
2   {
3     "id": "3269ba06-10da-4003-ba29-dd2348642a86",
4     "data": [
5       {
6         "text": "Turn ",
7         "userDefined": false
8       },
9       {
10        "text": "state ",
11        "alias": "state",
12        "meta": "@ACState",
13        "userDefined": true
14      },
15      {
16        "text": "the AC ",
17        "userDefined": false
18      }
19    ],
20    "isTemplate": false,
21    "count": 0,
```

```
22   "updated": 1560426984
23 },
24 {
25   "id": "dd84e278-63a9-4e84-afc6-b183463ee671",
26   "data": [
27     {
28       "text": "Set the AC to ",
29       "userDefined": false
30     },
31     {
32       "text": "state ",
33       "alias": "state",
34       "meta": "@ACState",
35       "userDefined": true
36     }
37   ],
38   "isTemplate": false,
39   "count": 0,
40   "updated": 1560426984
41 },
42 {
43   "id": "d013f09a-738d-4898-931d-bebcc61e67df",
44   "data": [
45     {
46       "text": "Set the air conditioner to ",
47       "userDefined": false
48     },
49     {
50       "text": "state ",
51       "alias": "state",
52       "meta": "@ACState",
53       "userDefined": true
54     }
55   ],
56   "isTemplate": false,
57   "count": 0,
58   "updated": 1560426984
59 },
60 {
61   "id": "6363fec0-0da7-406f-a000-b6c46ab72a35",
62   "data": [
63     {
64       "text": "I want the AC turned ",
65       "userDefined": false
66     },
67     {
68       "text": "state ",
69       "alias": "state",
70       "meta": "@ACState",
71       "userDefined": true
```

```
72     }
73   ],
74   "isTemplate": false,
75   "count": 0,
76   "updated": 1560426984
77 },
78 {
79   "id": "71975954-724f-4217-a333-0ba53ad5394e",
80   "data": [
81     {
82       "text": "Make sure the AC is ",
83       "userDefined": false
84     },
85     {
86       "text": "state ",
87       "alias": "state",
88       "meta": "@ACState",
89       "userDefined": true
90     }
91   ],
92   "isTemplate": false,
93   "count": 0,
94   "updated": 1560426984
95 }
96 ]
```

G.2. Test Agents 2-5

The other four test agents are not printed here to prevent the appendix from becoming too long. They can be found on my GitHub instead. ¹

¹<https://github.com/arne-z/BachelorThesis/tree/5100ed5f712fb93a079af095e1dcd6c159cfe68f/TestAgents>

H | Examples of Diffs created from Test Agent JSON files

H.1. RoomTemperature Agent

```
1  --- a/TestCase01_RoomTemperature/DialogflowExport/intents/ChangeTemperatur
2  +++ b/TestCase01_RoomTemperature/DialogflowExport/intents/ChangeTemperatur
3  @@ -21,6 +21,14 @@
4      }
5      ],
6      "isList": false
7  +    },
8  +    {
9  +      "id": "3f53e160-18ac-4318-99a8-5f84581bb045",
10 +      "required": false,
11 +      "dataType": "@sys.time",
12 +      "name": "point",
13 +      "value": "$point",
14 +      "isList": false
15 +    }
16   ],
17   "messages": [],
18
19   --- a/TestCase01_RoomTemperature/DialogflowExport/intents/ChangeTemperatur
20   +++ b/TestCase01_RoomTemperature/DialogflowExport/intents/ChangeTemperatur
21   @@ -1,9 +1,9 @@
22   [
23   {
24   -   "id": "e7dbd110-0604-476f-a997-5d90932ef9df",
25   +   "id": "46e61317-0882-4cb8-999b-2b5042cc27cf",
26   +   "data": [
27   +     {
28   -       "text": "Make sure the AC is at ",
29   +       "text": "Set the air conditioner to ",
30   +       "userDefined": false
31   +     },
32   +     {
33   @@ -17,10 +17,10 @@
34     "count": 0
```

```

35     },
36     {
37 -     "id": "1f80dd19-9fd9-4276-bbac-024bbf74b2f3",
38 +     "id": "62839ba2-4045-490d-8bc8-c74ff6a761cb",
39     "data": [
40         {
41 -         "text": "I want the AC turned to ",
42 +         "text": "Set the AC to ",
43         "userDefined": false
44     },
45     {
46 @@ -28,6 +28,16 @@
47         "alias": "temp",
48         "meta": "@sys.temperature",
49         "userDefined": true
50     },
51     {
52 +     "text": " at ",
53 +     "userDefined": false
54     },
55     {
56 +     "text": "point",
57 +     "alias": "point",
58 +     "meta": "@sys.time",
59 +     "userDefined": true
60     }
61 ],
62     "isTemplate": false,
63 @@ -45,11 +55,109 @@
64     "alias": "temp",
65     "meta": "@sys.temperature",
66     "userDefined": true
67 },
68 {
69 +     "text": " at ",
70 +     "userDefined": false
71 },
72 {
73 +     "text": "point",
74 +     "alias": "point",
75 +     "meta": "@sys.time",
76 +     "userDefined": true
77 }
78 ],
79 +     "isTemplate": false,
80 +     "count": 0
81 },
82 {
83 +     "id": "cf3e56d7-7e28-4d61-bea5-20bb052b0aa6",
84 +     "data": [

```

```
85 +     {
86 +         "text": "I want the AC turned to ",
87 +         "userDefined": false
88 +     },
89 +     {
90 +         "text": "temp",
91 +         "alias": "temp",
92 +         "meta": "@sys.temperature",
93 +         "userDefined": true
94 +     },
95 +     {
96 +         "text": " at ",
97 +         "userDefined": false
98 +     },
99 +     {
100 +         "text": "point",
101 +         "alias": "point",
102 +         "meta": "@sys.time",
103 +         "userDefined": true
104 +     }
105 ],
106 "isTemplate": false,
107 "count": 0
108 },
109 + {
110 +     "id": "0a434408-2e28-4719-b822-855b13df6dee",
111 +     "data": [
112 +         {
113 +             "text": "Make sure the AC is at ",
114 +             "userDefined": false
115 +         },
116 +         {
117 +             "text": "temp",
118 +             "alias": "temp",
119 +             "meta": "@sys.temperature",
120 +             "userDefined": true
121 +         },
122 +         {
123 +             "text": " at ",
124 +             "userDefined": false
125 +         },
126 +         {
127 +             "text": "point",
128 +             "alias": "point",
129 +             "meta": "@sys.time",
130 +             "userDefined": true
131 +         }
132 +     ],
133 +     "isTemplate": false,
134 +     "count": 0
```



```
135 + },
136 + {
137 +   "id": "e7dbd110-0604-476f-a997-5d90932ef9df",
138 +   "data": [
139 +     {
140 +       "text": "Make sure the AC is at ",
141 +       "userDefined": false
142 +     },
143 +     {
144 +       "text": "temp",
145 +       "alias": "temp",
146 +       "meta": "@sys.temperature",
147 +       "userDefined": true
148 +     }
149 +   ],
150 +   "isTemplate": false,
151 +   "count": 0
152 + },
153 + {
154 +   "id": "1f80dd19-9fd9-4276-bbac-024bbf74b2f3",
155 +   "data": [
156 +     {
157 +       "text": "I want the AC turned to ",
158 +       "userDefined": false
159 +     },
160 +     {
161 +       "text": "temp",
162 +       "alias": "temp",
163 +       "meta": "@sys.temperature",
164 +       "userDefined": true
165 +     }
166 +   ],
167 +   "isTemplate": false,
168 +   "count": 1
169 + },
170 + {
171 +   "id": "5940a192-1fe4-4849-bea3-aeff5824c916",
172 +   "data": [
```

H.2. Test Agents 2-5

The diffs created from the other four test agents are not printed here to prevent the appendix from becoming too long. They can be found on my GitHub instead.¹

¹See <https://github.com/arne-z/BachelorThesis/commits/master> all commits that are prefixed with "Test Case".