PERSONAL ASSISTANT FOR HEALTHY LIFESTYLE

DR 4.3: Multimodal Interaction leveraging non-expert coverage extensions

Bernd Kiefer, Ivana Kruijff-Korbayová, Anna Welker, Christophe Biwer, Bart Schreuder Goedheijt[†]

DFKI GmbH, TNO[†]
⟨{kiefer,ivana.kruijff,anna.welker,christophe.biwer}@dfki.de,
bart.schreudergoedheijt@tno.nl⟩

Project, project Id:	EU H2020 PAL / PHC-643783
Project start date:	March 1 2015 (48 months)
Due date of deliverable:	M36
Actual submission date:	26 April 2018
Lead partner:	DFKI
Revision:	final
Dissemination level:	PU

The present report describes the work carried out in the third project year regarding *Natural Multimodal Interaction*. It summarises the Deliverable D4.3: "Natural Multimodal Interaction leveraging non-expert coverage extensions".
In year 3, many additional functionalities have been added to the overall system, which are also supported by appropriate natural language interactions, which needed a significant amount of resources from this work package. A first open-source release of the dialogue management framework VOnDA has been deployed, which is complemented by a visual debugging and editing tool.

An episodic memory module was implemented that analyses the data base and refers back to past episodes (e.g., activities by the child) during the use of the MyPal app. This is a first step towards using memory-based interactions that show remembrance by the virtual agent.

As a preparation for the next phase, a prototype for off-activity task based on a symbolic memory and a prototype for robust interpretation using machine learning components has been implemented. To prepare for the higher load on the database, on one hand because there are now more modules actively monitoring the database, and on the other hand because of the increasing amount of users, we did research in the area of *stream reasoning*, that can provide efficient updates and results for our components even in a highly dynamic system with a bigger data load.

Executive Summary The overall objective of WP4 is to support the goals set for a patient using the PAL system by developing the means to conduct verbal communication, and to analyse textual data and extract relevant information. The components implemented in this work package must support this communication in a way to foster sustainable long-term interactions between a robot (or its avatar) and a human. This requires user-adaptive communication, coupling of verbal and non-verbal communication and grounding communication in long-term memory.

During Year 3, WP4 developed new interaction functionality and provided language and dialogue support for numerous extensions of the PAL agent that were requested for the upcoming experiments. The extensions concerning primarily WP4 were in the area of instructions, adaptivity and better user experience exploiting the interaction history.

This comprises verbal feedback for new activities like the *memory* and *break & sort* games, but also for targeted task suggestions based on the selected user goals, as well as introductory instructions and off-activity talk for the first time usage of the system. Moreover an *episodic memory* analysing the usage and interaction history and commenting on past events was planned and implemented.

Off-Activity-Talk is an important feature to increase the bond between the user and the artificial agent. Following our general strategy, we studied the possibility of using the symbolic information contained in our data store and the newly developed dialogue management platform to conduct such dialogues.

Having the PAL system as playground, we gained a lot of experience using the new dialogue management framework (now named VOnDA). This experience resulted in massive improvements and extensions, and in the (already planned) development of a debugger and editor GUI for the framework.

To explore the possibilities of machine learning approaches for natural language processing in PAL, we looked at available frameworks (rasa-nlu and spaCy) and applied them to robust natural language understanding

Finally, since several components in the PAL system are constantly monitoring the changes in the database, e.g., the *goal calculator* is computing goal progress based on incoming information, such as correctly answered quiz questions, the episodic memory tries to find past events that could be discussed with the user, and the dialogue manager gives feedback to recent activities while system is in use. All these components perform a kind of *streaming reasoning* on the database, which led us to look out for a way of efficiently doing this kind of computations.

The results of Year 3 are presented in a conference paper still under review, and two bachelor and two master theses.

1 The role of *Multimodal Natural Interaction* in PAL

WP4 focuses on the multimodal interaction around mHealth-Apps and additional conversational functionality in support of the high-level targets set in WP2 and actions selected in WP3. The challenge is to produce natural, flexible, personalised interactions that are sustainable in the long term as well as allow to extract data about the user. To achieve this, we are incorporating findings from the literature about what aspects are important for long term engagement.

The processing challenges for this work package are the robustness of input interpretation, flexibility *and* personal adaptation of the generated output, handling different situational contexts for both the physical and graphical embodiment of PAL, and allowing for interactions with one child alone or in the presence of an audience of multiple children. Additional challenges are posed by the need for extendable thematic and linguistic coverage.

The core functional component developed in WP4 is a multimodal dialogue system with a repertoire of multimodal dialogue acts (combining verbal and non-verbal means) modulated by affect. Generation as well as interpretation will use parameterised dialogue acts as an interface schema to other modules to abstract away from specific aspects of, e.g., natural language or emotion expression. Based on the high-level targets from WP2, action selection from WP3, the dialogue state (including the latest child's input and interaction history) as well as a long-term memory, the multimodal output generation module decides which act to activate ("what to express") and how to realise it multimodally in the given context ("how to express it").

In order to avoid repetitiveness in the long term, it is important to have flexible dialogue strategies and a rich repertoire of verbal and non-verbal expressions to allow for variation. The multimodal input processing module interprets verbal and non-verbal input. Interpretation is guided by information from the user model and the strategic planning (WP2 & 3) and provides information back to them. First, verbal input is needed for the dialog interaction itself as the dialogue's flow takes input from the child to progress. Second, an interpretation of the child's affective state is needed for engagement analysis used in WP2 to adapt the high-level goal self-management goals. Third, feedback is needed for WP3 as basis for adapting a child's preference model, and the long-term memory.

2 Tasks, objectives, results

The overall objective of WP4 is to develop the technologies for personalised multimodal natural interaction serving to actively foster long-term engagement with the robot and its avatar. Voluntary long-term use is required as a prerequisite for other system objectives. This encompasses natural language interpretation and multimodal generation, as well as dialogue management.

2.1 Planned work

Given the requests of the experimenters, the objective of WP4 in year 3 was to support the extended functionality of the overall system, and to improve its interactivity with respect to multimodal interaction. To achieve this, the following steps were planned:

- Develop and improve language and dialogue support for new and existing activities, such as games, but also instructions
- Develop an off-activity talk module based on symbolic information from the interaction history
- Apply and improve the new dialogue management framework VOnDA, given the experience gained in applying it to the system at hand
- Explore the potential of machine learning approaches for natural language processing in the context of the PAL project
- Improve and generalise the usage of the RDF database as a data source by implementing a streaming reasoner for HFC

All planned activities were successfully pursued and either resulted in an implementation or a research prototype for the topic in question. The following section will describe the work on these items and the results that have been achieved.

3 Actual work performed

3.1 Extension of dialogue functionality

Many new activities have been integrated into the PAL system, such as the *Memory* and *Break & Sort* Game, and an initial section for activity suggestions. For these activities, dialogue and language support was developed that shows the continuous presence of the virtual agent, which should increase the motivation of the child and guide it towards the most important goals.

In the last experiments, a lot of the detailed information of how to use the system at the first encounter in the hospital was given by humans. There was also almost no introductory conversation with the robot. For this system version, we integrated an extensive introductory part, where the robot asks the child for its name, age, and some preferences, e.g. hobby, sports or favourite colour, into the robot dialogue. After that, the robot gives detailed instructions, so that the first interaction really becomes a meeting of robot and child, and the role of the robot as a buddy is strengthened.

The number of basic semantic structures (dialogue acts) that are used for natural language generation have almost doubled from around 240 to 460 in the current prototype, not counting all variations in the arguments. This illustrates that improving and adapting the linguistic resources for this version was a major effort, as well as enhancing the dialogue logic.

3.2 Episodic Memory

To exploit the rich history of interactions and sensor data for dialogue, we developed an *episodic memory* module (EMM) that aggregates user data in search for past events that are worth mentioning in upcoming conversations with the user. For the virtual agent, the capability of showing that it is able to remember things is an important piece of personalised interaction, which makes it appear much more human and shows its connection to the specific user.

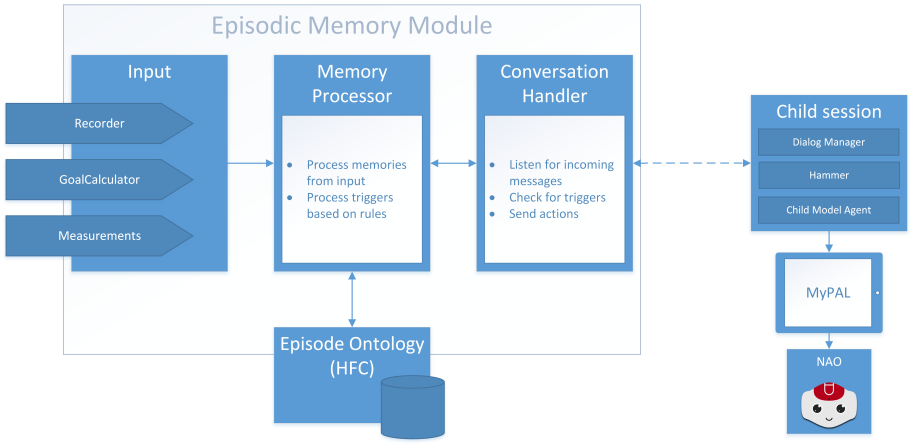


Figure 1: Overview of the Episodic Memory Module

In the end, the goal to be achieved by this kind of behaviours is that the affection of the user towards the agent and the user's intrinsic motivation increases. The research for this module is based on the Situative Cognitive Engineering Approach [3], which is also used in the overall design of the PAL system and ontology. We have leveraged the results of several studies

([10, 8, 1, 9]) to define meaningful episodes in the context of PAL.

This resulted in the specification of three scenarios: *Feedback on Goal Progress*, *Measurement of Blood Sugar*, *Past User Activities*. First, the relevant data to implement these episodes was identified in the PAL ontology. Using these specifications, a module was developed that monitors changes in the user data and eventually creates new episodes and adds them to the database. The dialogue handler then picks up new episodes and talks about them during pre-specified situations in the overall interaction with the MyPal app.

With the new module, the PAL actor will for example praise the child for sports activities that have been entered regularly into the timeline, or show empathy in cases where past events were marked with a low mood by asking for the reason and if the mood has improved since then.

More details to the Episodic Memory can be found in annex A.3.1.

3.3 Prototype for Off-Activity-Talk

The episodic memory described before is targeting mainly the goals directly relevant for the child's self-management. Previous work has shown that so-called off-activity talk (or social talk) also has a positive effect on bonding and adherence [7]. We aim to create a dialogue module for an autonomous agent that uses the symbolic knowledge stored in its information state to conduct an informed conversation, showing remembrance and relatedness to the user by using bits from the history, and making meaningful remarks on the child's reactions.

As a first step towards a more powerful module, small conversations about movie or travel preferences can be held, a special feature being that the system can relate back to previous user recommendations, e.g. to watch the movie "Iron Man", such as *U: Did you see Iron Man? – R: Not yet, my battery was low.*, or in a proactive way: *R: Hi, how are you, I watched Iron Man yesterday!*.

The prototype uses WordNet [4] as a static ontology to relate the concepts (frames/verbs and entities/nouns) in the semantic representation of the natural language to existing structures in the database, exploiting type similarities. If the existing ontology does not contain the identified concept, or no related concept could be found, the agent issues clarifications question and tries to find suitable existing information. If this fails, a new generic information structure is added to the database.

In the next year, we will implement a more powerful flexible natural language interpretation, which can handle more than the current, fairly small set of utterances, and will allow more elaborate conversations. For more detail, see A.2.1.

3.4 Robust Interpretation Prototype

We implemented a first prototype for machine-learning based natural language understanding for Dutch and Italian. It is based on the open-source libraries spaCy[1] and rasa-nlu [2]. We used our generation grammars to produce input data by generating all possible sentences from the test files which contain the possible semantic input structures and swapping input and output. From this, we conducted a first experiment which produced promising results, but the amount of data is too little to give authoritative figures. Also, intent recognition is done currently only on sentence level, which is bound to be wrong or too unspecific in many cases. We are currently working on possibilities to include features from the interaction history into this process, or, alternatively, have a second classifier that takes the history and the sentence analysis and eventually modifies it.

Computational Resources Required for Deep Learning Approaches (R2.10) Deep Learning Approaches need a lot of computational resources, especially for training. During run-time the resources needed strongly depend on the model in use (recurrent vs. non-recurrent, network layout and complexity). Many approaches are surprisingly fast and use little resources although producing very good results, like, e.g., the spaCy framework that we employ for the robust interpretation prototype. spaCy already has support for many European languages: English, German, Spanish, Portuguese, French, Italian and Dutch. Necessary adaptations could be achieved through adaptive re-training, starting from the existing models, which also needs significantly less computational resources than learning a language from scratch. For low-resource languages, transfer learning, using existing models for other languages has been employed successfully.

Concerning this, even application of such models locally on the tablet is feasible, especially since typically only one language has to be installed, and there is only one user at a time.

3.5 VOnDA framework

The dialogue management framework, now named VOnDA, was strongly improved in the last year, primarily based on the experiences with the PAL system. One of the main design goals was that the dialogue rule specifications should be as concise as possible. This is supported by more type inference on variables, but also on the accesses to objects in the data storage. Additional support for non-functional predicates returning sets of objects, and functional expressions for examining or filtering the contents of these sets have been implemented.

[1] `https://spacy.io/`

[2] `https://rasa.com/products/rasa-nlu/`

These improvements lead to even more compact code for the dialogue specifications, because for example many type declarations or casts can be omitted, and code working the values on non-functional predicates is much more compact due to the functional programming syntax.

Our goal concerning the system maturity was to arrive in a state that would allow us to put the system into the open domain. The effort we took is illustrated by the number of commits in this period, which is about the half of the total number of commits for the whole module (Total: 1233, May 1st 2017 – now: 569). Test coverage for the compiler has been massively increased to over 80%. VOnDA is now an open-source project on GitHub [3] under the Creative Commons Non-Commercial License. A demo paper has been submitted to the ACL 2018 conference [4], which is included in the non-public version of this deliverable (Annex A.1.1).

The off-activity prototype described earlier has also been implemented using VOnDA, and an improved version will be integrated in the last period into the PAL system.

Visual Debugger An important improvement of the system is the visual debugger which allows to monitor the workings of a running agent in real time. One problem with rule-based systems for dialogue management is to keep track of the rules that are triggered (or not) when the number of rules increases, and what the current information state looks like when the rules are evaluated.

To allow rule monitoring, special code is added during the compilation of the source code. The boolean expressions forming the preconditions are analysed down to their base terms, and the structure of the expression is stored in a way that allows the debugger to present it in a way that can be easily understood by the developer. The truth value of the base terms and the full expression is displayed using colours: green for true, red for false, and grey if a sub-term has not been evaluated because of an operator with short cut logic.

A tree view of all rules allows a fine grained selection of the rules that will be presented to avoid being swamped with information when debugging a specific situation. The debugger and the VOnDA run-time core talk to each other using a socket connection, which allows remote debugging, and attaching and detaching the debugger to the core at any time.

Figure 2 shows a typical screenshot of the debugger, that also provides the usual GUI functionality, like opening the files of a project in an editor, compiling the source code and monitoring if the source is newer than the compiled code. Detailed information about the functionality and the implementation of the debugger can be found in Annex A.2.2.

[3] `https://github.com/bkiefer/vonda`

[4] `http://acl2018.org/`

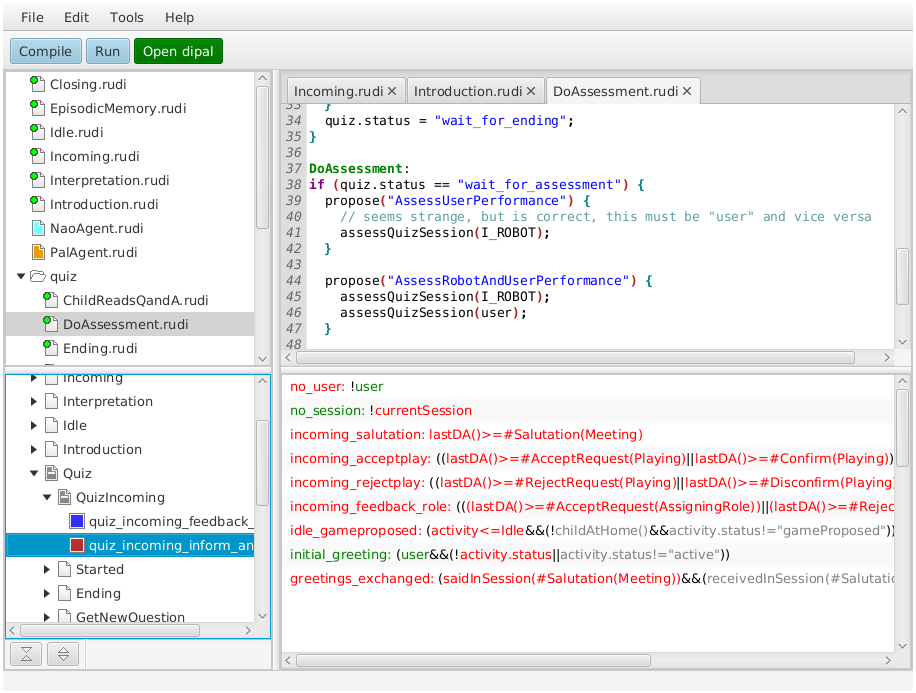


Figure 2: Screenshot VOnDA's visual debugger

Currently, the debugger lives in its own project on GitHub[5], available under the same license as VOnDA.

In addition, we have a small but currently detached GUI to do real-time queries on the database. Previous queries are stored in a history, and recently used queries are pushed to the top of this history, so that those that are most relevant can be accessed fast. However, we found that there is need for a better solution to this, more in the spirit of "watch expressions" that can be found in traditional debuggers for programming languages, which dynamically show the changing values of some given expressions (database queries in our case). Figure 3 shows a screenshot of the current GUI.

The visual debugger has become an indispensable tool for maintaining and debugging the dialogue functionality of the PAL system. It has strongly improved the productivity for writing new parts, also because navigating the source code during implementation and debugging has become so much easier.

3.6 Streaming Reasoning

When dealing with dynamic RDF databases, the topic of generating results from the constantly updated data stream, either to produce results for some

[5] `https://github.com/yoshegg/rudibugger`

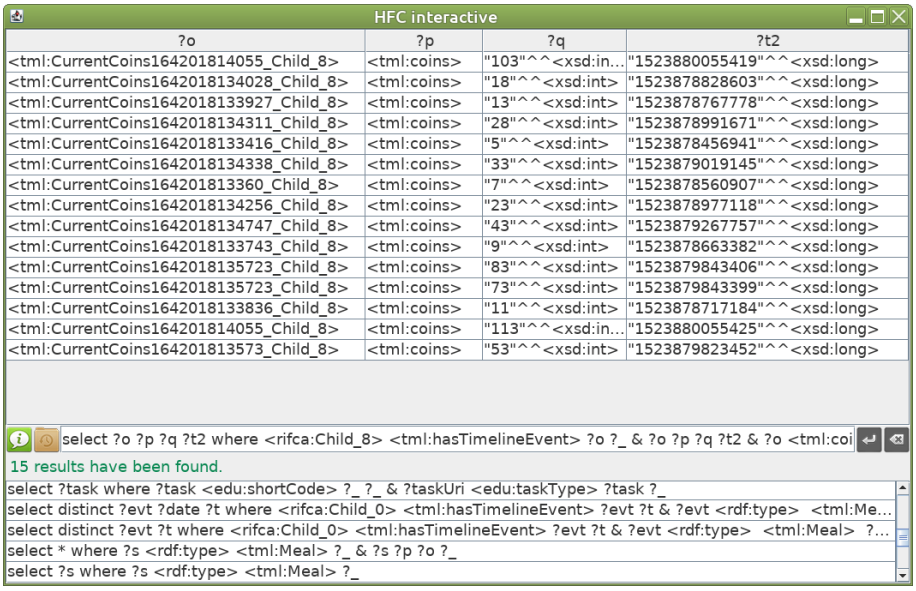


Figure 3: Screenshot of the database access GUI

external process or to add it to the database itself increases in importance with the amount of changing data and the number of consumers. In PAL, we now have several modules which are such consumers, namely the *Goal Calculator* (WP 2), the *Episodic Memory* (WP 2 & 4), the *Timeline Feedback* (WP 2 & 4), the *MyPal App* (WP 5, for synchronisation), and last, but not least, the Dialogue Management Framework. With an increasing number of users that will increase the load on the data base, this also becomes a topic for the central database of PAL.

For that reason, we did research on techniques for the integration of so-called *streaming reasoning* into HFC. Three existing approaches were compared and classified, namely C-SPARQL, CQUELS and ETALIS. For the implementation in HFC (the RDF store and reasoner developed at DFKI), streaming extensions for the query language were developed, with a special focus on temporal relations. These relations play a large role for stream reasoning in general, but especially for applications in dialogue systems. The actual streaming reasoning was implemented on the basis of this extension. To guarantee run-time efficiency, an indexing framework was added to the existing system, and several caching and indexing data structures were implemented and compared. A simplified schema of the new framework is shown in Figure 4.

Finally, the Streaming HFC implementation was compared to the existing frameworks, where possible. We plan to use the improved version in the next phase of PAL to guarantee scalability to an even more complex system and to an increased number of parallel users. Stream reasoning will allow

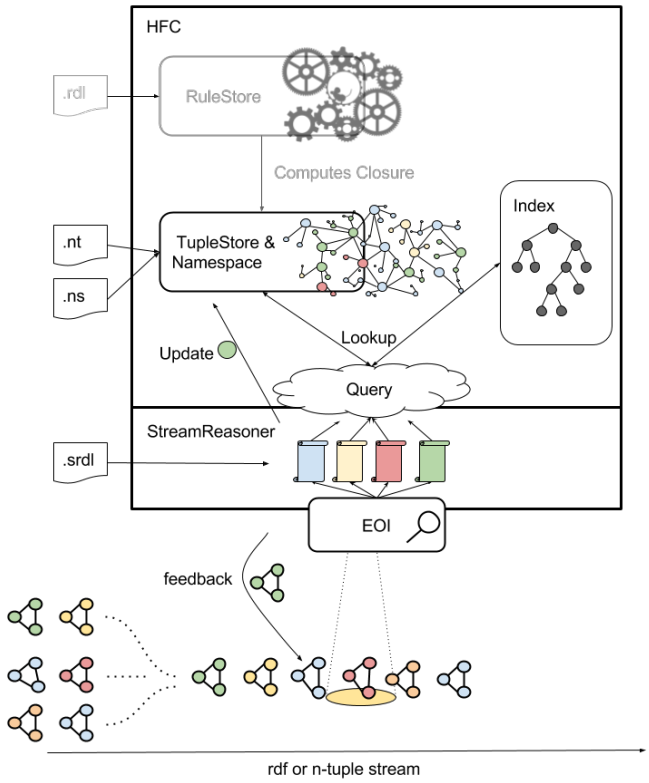


Figure 4: Structure of the stream reasoner framework

the different modules to reduce the amount of computing power needed for updates, improving the overall run-time efficiency. More details about this topic can be found in Annex A.3.2.

3.7 Use of ASR in the Project (R2.9)

We studied the new EU General Data Protection Regulation and the information provided by the Automatic Speech Recognition (ASR) cloud service providers Nuance and Google concerning data protection. From the available information, we could not deduce that we can safely use them as backends for ASR. This led us to the conclusion of dropping our goal to use ASR for the upcoming prototype, since the work package does not have the resources of setting up a local ASR infrastructure. ASR would be interesting for open-domain talk since for short, foreseeable reactions, other means of input are more efficient. For open-domain application, however, the local services that could be set up are not of sufficient quality.

Given that the companies providing Cloud ASR services will have to implement the EU General Data Protection Regulation in the future, and

that commercial applicants of the PAL technology will have more resources to handle the privacy and ethical issues, the use of this services may be feasible in the future, and in fact very desirable for a more natural communication.

If time permits, we will also investigate the possibility of using state-of-the-art free systems like Kaldi[6], but to set them up is not trivial and may exceed our means.

3.8 Commercial vs. Free Text-To-Speech (TTS) solutions (R2.8)

Currently, free voices for Mary TTS[7] are available for German, English, French, Italian, Luxemburgish, Turkish and Tibetan. For Festival, there is a Spanish voice[8]. Building new (free) voices, e.g., from freely available audio books is possible, but requires some expertise and work for data preparation and cleaning, which could possibly be financed through crowd funding. Voice building tools are included in Mary TTS. The requirements for the voice of a PAL agent or its successor, e.g., concerning naturalness, are not as high as for other applications since a slightly distorted or imperfect voice matches the artificial nature of the agent. This means that a building an open-source voice with a fair amount of data is an acceptable solution, if the system should be continued on a no-fare or low-fare business model.

Pricing information for commercial system is not easy to achieve, since it massively depends on the usage pattern. PAL invested around 5000 € for the commercial Dutch systems for 2.5 years of the project, which will certainly be higher in the case of commercial use.

A special requirement of PAL was that the physical robot had to speak with the same voice as the actor on the tablet. This made it necessary to use an external TTS. If the system was only used with the app, a TTS that is installed on the tablet might be a convenient and low-price solution. In the end, it depends on the business model of a PAL successor which kind of TTS is the most appropriate.

[6] `http://kaldi-asr.org/`

[7] `https://github.com/marytts/marytts/tree/v5.2`

[8] `http://homepages.inf.ed.ac.uk/jyamagis/release/SpanishHTSVoice-ver0.8.tar.gz`

4 Conclusions

The main work in year 3 was dedicated to support the system extensions requested by the experimenters, and to improve the overall interactivity. This was achieved by adding dialogue policies and linguistic resources for new activities, posing general questions at the first encounter, and for giving detailed instructions for how to use the system.

To further improve that in the last phase, a prototypical module of off-activity talk has been implemented, based on symbolic information from the database and the dialogue framework implemented in this project, and a prototype for robust natural language understanding build on machine-learning based modules.

Overall, the WP-tasks provided the following major outcomes:

- Extended dialogue and language support for new activities in the PAL system
- An open-source release of the dialogue management framework, and a visual debugger and editor to support development
- A prototype for Off-Activity Talk
- A prototype for robust natural language interpretation
- A stream reasoner for the HFC database and reasoner

With these outcomes, work package 4 achieved milestone 4.3, *Improved verbal and non-verbal adaptivity, more flexible dialogue, basic grammar and knowledge management support, improved integration of knowledge base and memory.*

References

[1] Lawrence J Becker. Joint effect of feedback and goal setting on performance: A field study of residential energy conservation. *Journal of applied psychology*, 63(4):428, 1978.

[2] Christophe Biwer. rudibugger - Graphisches Debugging der Dialogmanagementtechnolgie VOnDA. Bachelor's Thesis, Saarland University, 2017.

[3] TU Delft and TNO. Applying the situated cognitive engineering method - a comprehensive guide. Technical report, 2012.

[4] Christiane Fellbaum. *WordNet*. Wiley Online Library, 1998.

[5] Bart Schreuder Goedheijt. Personalized robot support for children with diabetes in the pal project. Master's thesis, KTH Information and Communication Technology, 2017.

[6] Bernd Kiefer, Anna Welker, and Christophe Biwer. Vonda: A framework for ontology-based dialogue management. Submitted as Demo Paper to the 56th Annual Meeting of the Association for Computational Linguistics (ACL2018).

[7] Ivana Kruijff-Korbayová, Elettra Oleari, Ilaria Baroni, Bernd Kiefer, Mattia Coti Zelati, Clara Pozzi, and Alberto Sanna. Effects of off-activity talk in human-robot interaction with diabetic children. In *Ro-Man 2014: The 23rd IEEE International Symposium on Robot and Human Interactive Communication. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), August 25-29, Edinburgh*, pages 649–654. IEEE, 8 2014.

[8] Edwin A Locke, Karyll N Shaw, Lise M Saari, and Gary P Latham. Goal setting and task performance: 1969–1980. *Psychological bulletin*, 90(1):125, 1981.

[9] Andrew M Nuxoll and John E Laird. Enhancing intelligent agents with episodic memory. *Cognitive Systems Research*, 17:34–48, 2012.

[10] Richard M Ryan and Edward L Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist*, 55(1):68, 2000.

[11] Anna Welker. Wissensbasiertes Dialogmanagement für Social Talk. Bachelor's Thesis, Saarland University, 11 2017.

[12] Christian Willms. Ontology-supported stream reasoning based on hfc. Master's thesis, Saarland University, 2017.

A Annexes

A.1 Submitted Papers

A.1.1 B. Kiefer, A. Welker, C. Biwer (2018), "VOnDA: A Framework for Ontology-Based Dialogue Management"

Abstract We present VOnDA, a framework to implement the dialogue management functionality in dialogue systems. Although domain-independent, VOnDA is tailored towards dialogue systems with a focus on social communication, which implies the need of a long-term memory and high user adaptivity. For these systems, which are used in health environments or eldery care, control over the dialogue process is of top-most importance, which excludes a purely machine-learning approach. The same holds for commercial applications, where customer trust is at risk. VOnDA's specification and memory layer relies upon (ex- tended) RDF/OWL, which provides a universal and uniform representation, and facilitates interoperability with external data sources.

Relation to WP Making the results of the PAL project open source was one of the first decisions taken at the beginning of the project. We are taking one step towards this goal by providing a first open-source release of VOnDA and make it known in the computational linguistics community.

Availability Submitted to ACL 2018. Included only in the non-public version of this deliverable (Annex B.1).

A.2 Bachelor Theses

A.2.1 Anna Welker (2017), "Wissensbasiertes Dialogmanagement für Social Talk"

Abstract Robots and virtual agents are becoming increasingly popular as buddies or coaches in everyday life. Studies show that the success of such an agent often depends upon its ability to establish a bond to the human user. Being able to conduct so-called off-activity talk (small talk) can be an important puzzle piece in this direction. To successfully do this in a situation where the goal is a long-term relation with many interactions, it is important to have a long term memory of interactions and events that can be exploited for this task, since not many things in communication are more convincing than showing signs of remembrance. This work presents a rule-based dialogue module that can be added to that of a virtual agent. It is not only domain independent, but can remember past conversations with the user for use in the ongoing interaction.

Relation to WP Including off-activity talk into the PAL system will help to increase the bond between the virtual agent and the user. We expect, based on former research, that this may also help to increase adherence, at least a prolonged usage period of the system.

Availability Unrestricted. Included in the public version of this deliverable (annex C.2).

A.2.2 Christophe Biwer (2017), "RudiBugger: Graphisches Debugging der Dialogmanagementtechnologie VOnDA"

Abstract Debugging rule-based systems is a challenge, especially when the number of rules increases above a certain threshold. In this thesis, a debugger for the VOnDA dialogue management framework is described that tries to facilitate the task of monitoring a large running rule-based system. It does so by providing a targeted view of the current rule set of interest, and quick access to the relevant part of the source code from the different views.

Relation to WP Since VOnDA is the central engine for dialogue management, which also provides an important part of the system's logic, a visual debugger greatly improves the development time of this part of the system.

Availability Unrestricted. Included in the public version of this deliverable (annex C.1).

A.3 Master Theses

A.3.1 Bart Schreuder Goedheijt (2017), "Recalling shared memories in an embodied conversational agent"

Abstract The PAL project aims to help children with type 1 diabetes to improve their self-management skills using a social robot and its virtual avatar. It has been challenging to gain a long-term relationship with a robot or virtual character. After the novelty effect wears off, the interest of the user decreases over time. The aim of this project was to explore and develop personalized interactions with the children, using episodic memory to improve the engagement and diabetes self-management. A module was built that could capture and refer to shared experiences between the PAL actor and the child. During an experiment with children, the usage decreased over time after the novelty effect wore off. No increases in affection, motivation and diabetes self-management were found after the implementation of the episodic memory module. The full potential of episodic memory was

however untested, as the novelty effect already wore off before the implementation. Further research is recommended in order to assess the benefits of an improved version of the episodic memory update during an A/B test.

Relation to WP

Availability Unrestricted. Included in the public version of this deliverable (annex D.1).

A.3.2 Christian Willms (2017), "Ontology-supported Stream Reasoning based on HFC"

Abstract In the past decades, the usage of ontologies in systems got more and more popular and they were integrated in a wide variety of applications. However, many systems often have handle highly dynamical contexts, where run-time data is continuously generated by multiple sensor networks, various online services, and so forth. This leads to the challenge of processing heterogeneous event streams, combining them with background knowledge, i.e., an ontology, as well as continuously performing reasoning operations on them. Another complicating factor is that only a small fraction of these events might actually be relevant, whereas the remaining data is negligible. This thesis will address this issue and present a possible solution. It presents an ontology-based stream reasoner which utilizes domain-specific rules to detect important events in a stream of events, and which infers new facts based on these rules, the events, and the current state of the ontology providing background knowledge. These facts are then used to update the ontology itself and to feed back inferred facts (events) into the stream. In contrast to existing approaches, the background ontology will not be restricted by the classical W3C standards, but will support the more general and more powerful n-tuple representation, which allows to add tempo-spatial information to the contextual facts in a lightweight manner. The proposed thesis will evaluate whether this n-tuple based approach leads to performance improvements compared to the few existing triple based stream reasoners.

Relation to WP At least four modules in the integrated PAL system use a form of streaming reasoning that are relevant to WP4: The *Goal Calculator*, *Episodic Memory*, the *Timeline Feedback* and the dialogue management itself

Availability Unrestricted. Included in the public version of this deliverable (annex D.2).

B Submitted papers

VOnDA: A Framework for Ontology-Based Dialogue Management

Bernd Kiefer **Anna Welker** **Christophe Biwer**
German Research Center for Artificial Intelligence (DFKI)
Saarbrücken, Germany
{kiefer|anna.welker|christophe.biwer}@dfki.de

Abstract

We present VOnDA, a framework to implement the dialogue management functionality in dialogue systems. Although domain-independent, VOnDA is tailored towards dialogue systems with a focus on social communication, which implies the need of a long-term memory and high user adaptivity. For these systems, which are used in health environments or eldery care, control over the dialogue process is of topmost importance, which excludes a purely machine-learning approach. The same holds for commercial applications, where customer trust is at risk. VOnDA's specification and memory layer relies upon (extended) RDF/OWL, which provides a universal and uniform representation, and facilitates interoperability with external data sources.

1 Introduction

Natural language dialogue systems are becoming more and more popular, be it as virtual assistants such as Siri or Cortana, as Chat Bots on websites providing customer support, or as interface in human-robot interactions in areas ranging from Industry 4.0 (Schwartz et al., 2016) over social human-robot-interaction (ALIZ-E) to disaster response (Kruijff-Korbayová et al., 2015).

A central component of most systems is the *dialogue manager*, which controls the (possibly multi-modal) reactions based on sensory input and the current system state. The existing frameworks to implement dialogue management components roughly fall into two big groups, those that use symbolic information or automata to specify the dialogue flow (IrisTK (Skantze and Al Moubayed, 2012), RavenClaw (Bohus and Rudnicky, 2009), Visual SceneMaker (Gebhard et al., 2012)), and those that mostly use statistical methods (PyDial (Ultes et al., 2017), Alex (Jurčíček et al., 2014)). Somewhat in between these is OpenDial (Lison and Kennington, 2015), which builds on probabilistic rules and a Bayesian Network.

When building dialogue components for robotic systems or in-car assistants, the system needs to take into account *various* system inputs, first and foremost the user utterances, but also other sensoric input that may influence the dialogue, such as information from computer vision, gaze detection, or even body and environment sensors for cognitive load estimation.

The integration and handling of the different sources such that all data is easily accessible to the dialogue management is by no means trivial. Most frameworks use plug-ins that directly interface to the dialogue core. The multi-modal dialogue platform SiAM-dp (Neßelrath and Feld, 2014) addresses this in a more fundamental way using a modeling approach that allows to share variables or objects between different modules.

In the application domain of social robotic assistants, it is vital to be able to maintain a relationship with the user over a longer time period. This requires a long-term memory which can be used in the dialogue system to exhibit familiarity with the user in various aspects, like personal preferences, but also common knowledge about past conversations or events, ranging over multiple sessions.

In the following, we will describe VOnDA, an open-source framework to implement dialogue strategies. It follows the information state/update tradition (Traum and Larsson, 2003) combining a rule-based approach with statistical selection, although in a different way than OpenDial. VOnDA specifically targets the following design goals to support the system requirements described before:

- Flexible and uniform specification of dialogue semantics, knowledge and data structures

- Scalable, efficient, and easily accessible storage of interaction history and other data, resulting in a large information state
- Readable and compact rule specifications, facilitating access to the underlying RDF database, with the full power of a programming language
- Transparent access to Java classes for simple integration with the host system

2 Architecture

VOnDA follows the information state/update paradigm. The information state is realized by an RDF store and reasoner with special capabilities (HFC (Krieger, 2013)), namely the possibility to directly use n-tuples instead of triples. This allows to attach temporal information to every data chunk (Krieger, 2012, 2014). In this way, the RDF store can represent *dynamic objects*, using either *transaction time* or *valid time* attachments, and as a side effect obtain a complete history of all changes. HFC is very efficient in terms of processing speed and memory footprint, and has recently be extended with stream reasoning facilities. VOnDA can use HFC either directly as a library, or as a remote server, also allowing for more than one instance, if needed.

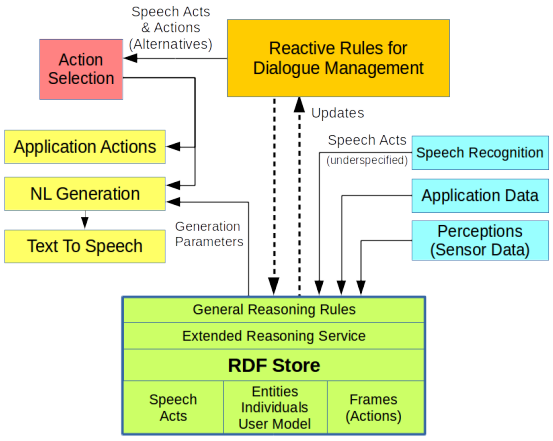


Figure 1: VOnDA Architecture

The RDF store contains the dynamic and the terminological knowledge: specifications for the data objects and their properties, as well as a hierarchy of dialogue acts, semantic frames and their arguments. These specifications are also used by the compiler to infer the types for property values (see section 3.1), and form a declarative API to connect new components, e.g., for sensor or application data.

We are currently using the DIT++ dialogue act hierarchy (Bunt et al., 2012) and shallow frame semantics along the lines of FrameNet (Ruppenhofer et al., 2016) to interface with the natural language understanding and generation units. While shallow semantics is enough for most applications, we already experience its shortcomings when trying to handle, for example, social talk.

A set of reactive rules is executed whenever there is a change in the information state (IS). These changes are caused by incoming sensor or application data, intents from the speech recognition, or expired timers. Rules are labeled if-then-else statements, with complex conditions and shortcut logic, as in Java or C. The compiler analyses the base terms and stores their values during processing for dynamic logging. A rule can have direct effects, like changes in the IS, or system calls. Furthermore, it can generate so-called *proposals*, which are (labeled) blocks of code in a frozen state that will not be immediately executed, similar to closures.

All rules are repeatedly applied until a fix point is reached: No new proposals are generated and there is no IS change in the last iteration. Then, the set of proposals is evaluated by a statistical component, which will select the best alternative. This component can be exchanged to make it as simple or elaborate as necessary, taking into account arbitrary features from the data storage.

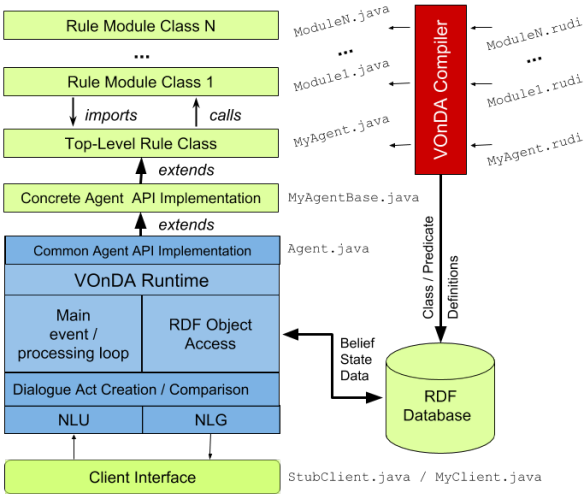


Figure 2: A schematic VOnDA agent

A VOnDA project consists of an ontology, a custom extension of the abstract `Agent` class (the

so-called *wrapper class*), a client interface to connect the communication channels of the application to the agent, and a set of rule files that are arranged in a tree, using `import` statements. The blue core in Figure 2 is the run-time system that is part of VOnDA, while all green elements are the application specific parts of the agent. A `Yaml` project file contains all necessary information for compilation: the ontology, the wrapper class, the top-level rule file and other parameters, like custom compile commands.

The ontology contains the definitions of dialogue acts, semantic frames, class and property specifications for the data objects of the application, and other assertional knowledge, such as specifications for "forgetting", which could be modeled in an orthogonal class hierarchy, and supported by custom deletion rules in the reasoner.

Every rule file can define variables and functions in VOnDA syntax which are then available to all imported files. The methods from the wrapper class are available to all rule files.

The current structure assumes that most of the Java functionality that is used inside the rule files will be provided by `Agent` superclass. There are, however, alternative ways to use other Java classes directly.

3 Implementation

3.1 Language features

VOnDA's rule language looks very similar to Java/C++. There are a number of specific features which make it much more convenient for the specification of dialogue strageties. One of the most important features is the way objects in the RDF store can be used throughout the code: RDF objects and classes can be treated similarly to those of object oriented programming languages, including the type inference and inheritance that comes with type hierarchies.

```
user = new Animate;
user.name = "Joe";
set_age:
if (user.age <= 0) {
  user.age = 15;
}
```

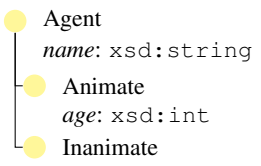


Figure 3: Ontology and VOnDA code

Figure 3 shows an example of VOnDA code, and how it relates to RDF type and property specifications, schematically shown on the right. The domain and range definitions of properties are picked up by the compiler and used in various places, e.g., to infer types, do automatic code or data conversions, or create "intelligent" boolean tests, like in line 4, which will expand into two tests, one testing for the existence of the property for the object, and in case that succeeeds, a test if the value is greater than zero. If there is a chain of more than one field/property access, every part is tested for existence in the target code, keeping the source code as concise as possible. Also for reasons of brevity, the type of a new variable needs not be given if it can be inferred from the value assigned to it.

New RDF objects can be created with `new`, similar to Java objects; they are immediately reflected in the database, as are all changes to already existing objects.

Many operators are overloaded, especially boolean operators such as **<=**, which compares numeric values, but can also be used to test if an object is of a specific class, for subclass tests between two classes, and for subsumption of dialogue acts.

```
if (!saidInSession(#Greeting(Meeting)) {
  // Wait 7 secs before taking initiative
  timeout("wait_for_greeting", 7000) {
    if (! receivedInSession(
              #Greeting(Meeting))
      propose("greet") {
        da = #InitialGreeting(Meeting);
        if (user.name)
          da.name = user.name;
        emitDA(da);
      }
  }

  if (receivedInSession(#Greeting(Meeting))
    // We assume we know the name by now
    propose("greet_back") {
      emitDA(#ReturnGreeting(Meeting,
                  name=^user.name);
    }
  }
}
```

Figure 4: VOnDA code example

There are two statements with a special syntax and semantics: `propose` and `timeout`. `propose` is VOnDA's current way of implementing probabilistic selection. All (unique) propose blocks that are in active rule actions are collected, frozen in the execution state in which they were encountered, like closures known from functional programming languages. When all possible proposals have been selected, a statistical component decides which one will be taken and the closure is

executed.

`timeouts` also generate closures, but with a different purpose. They can be used to trigger proactive behaviour, or to check the state of the system after some time period, or in regular intervals. A timeout will only be created if there is no active timeout with that name.

Figure 4 also contains an example of the shorthand notation for shallow semantic structures (starting with **#**). Since they predominantly contain constant (string) literals, this is the default when specifying such structures. The special "hat" syntax in `user=^user.name` allows to insert the value of expressions into the literal, similar to an *eval*.

This section only described the most important features of VOnDA's syntax. For a detailed description, the reader is referred to the user documentation.

3.2 Compiler

The compiler turns the VOnDA source code into Java source code using the information in the ontology. Every source file becomes a Java class. The generated code will not serve as an example of good programming practice, but a lot of care has been taken in making it still readable and debuggable. The compile process is separated into three stages: parsing and abstract syntax tree building, type checking and inference, and code generation.

The VOnDA compiler's internal knowledge about the program structure and the RDF hierarchy takes care of transforming the RDF field accesses to reads from and writes to the database. Beyond that, the type system, resolving the exact Java, RDF or RDF collection type of arbitrary long field accesses automatically performs the necessary casts for the ontology accesses.

3.3 Run-Time Library

The run-time library contains the basic functionality for handling the rule processing, including the proposals and timeouts, and for the on-line inspection of the rule evaluation. There is, however, no blueprint for the main event loop, since that depends heavily on the host application. It also contains methods for the creation and modification of shallow semantic structures, and especially for searching the interaction history for specific utterances. Most of this functionality is available through the abstract `Agent` class, which has to be extended to a concrete class for each application.

There is functionality to talk directly to the HFC database using queries, in case the object view is not sufficient or to awkward. The natural language understanding and generation components can be exchanged by implementing existing interfaces, and the statistical component is connected by a message exchange protocol. A simple generation engine based on a graph rewriting module is already integrated, and is used in our current system as a template based generator. The example application also contains a VoiceXML based interpretation module.

3.4 Debugger/GUI

VOnDA comes with a GUI (Biwer, 2017) that helps navigating, compiling and editing the source files belonging to a project. It uses the project file to collect all the necessary information.

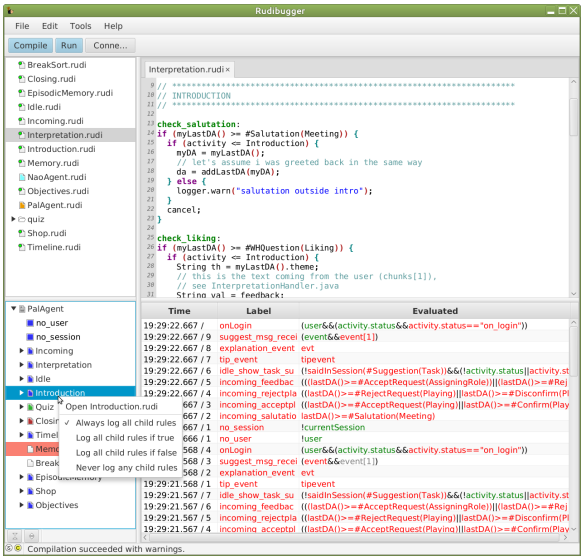


Figure 5: The VOnDA GUI window

Upon opening a project, the GUI displays the project directory (in a *file view*). The user can edit rule files from within the GUI or with an external editor like Emacs, Vim, etc. and can start the compilation process. After successful compilation, the project view shows what files are currently used, and marks the top-level and the wrapper class files. A second tree view (*rule view*) shows the rule structure in addition to the module structure. Modules where errors or warnings were reported during compilation are highlighted, and the user can quickly navigate to them using context menus.

Additionally, the GUI can be used to track what is happening in a running system. The connection

is established using a socket to allow remote debugging. In the rule view, multi-state check boxes are used to define which rules should be observed under which conditions. A rule can be set to be logged under any circumstances, not at all or if its condition evaluated to true or to false. Since the rules are represented in a tree-like structure, the logging condition can also be set for an entire subgroup of rules, or for a whole module. The current rule loggging configuration can be saved for later use.

The *logging view* displays incoming logging information as a sortable table. A table entry contains a time stamp, the rule's label and its condition. The rule's label is colored according to the final result of the whole boolean expression, and each base term is colored accordingly, or greyed out if short-cut logic led to premature failure or success of the expression.

4 Applications

VOnDA is used in the integrated system of the EU project PAL (PAL, 2018), which uses human-robot interaction to support children with diabetes type 1 in coping with their disease. Children interact with a real NAO robot[1], or with an Android app that connects to the core system and exhibits a virtual character that is as similar to the robot as possible, also in its behaviour.

The dialogue component, which is largely responsible for the agent's behaviour, is implemented using the VOnDA framework. In addition, HFC, the RDF store that VOnDA builds upon, is the main database of the system, storing all relevant information and being the central data exchange hub. The system runs as a cloud based robotic solution, spawning a new system instance for every user. It has been successfully tested with more than 40 users at a time on a medium sized virtual machine with only moderate load factors, giving a positive indication of the scalability of HFC and the VOnDA approach.

VOnDA has also been used in a recent bachelor thesis project aiming to implement a generalized, ontology-based approach to open-domain talk (Welker, 2017). The Smoto system uses an additional HFC server running WordNet (Miller, 1995; Fellbaum, 1998) as semantic database, thereby gaining knowledge about semantic concepts that can be used in the dialogue and to find appropriate reactions on arbitrary user input.

[1] Softbank Robotics
https://www.ald.softbankrobotics.com

5 Discussion and Further Work

Implementing social robotic assistants, especially for children, dictates two requirements that guided the development of the VOnDA framework. Firstly, it requires a lot of control over the decision process, since mistakes by the system are only tolerable in very specific situations. Secondly, efficient and easy access to the interaction and event history is paramount. This, together with the fact that data collection for our user group is even more challenging than usually, for ethical reasons, guided our decision to go for a rule-based system with RDF/OWL underpinning.

Using the well established RDF/OWL standard as specification layer makes it very easy to add or change application specific data structures, especially because of the existing tool support. We already use the reasoning facilities for type and partially for temporal inference, but given the possibility of attaching also confidence or credibility information to the RDF data, a more integrated probabilistic approach with soft preconditions could be implemented, e.g., on the basis of Dempster-Shafer theory. Moreover, additional meta knowledge, such as trustworthiness or validity period could be declared using multiple inheritance, which opens many interesting research directions.

VOnDA is still under development. We designed it such that it can be integrated in most applications and opens many ways for improvements and additions. Our next steps will be the improvement of the GUI, including features such as a watch window and/or a timeline to track changes of specific values in the database, and a tool that analyses the dependencies between rules on the basis of the conditions' base terms.

Given the popularity of automata-based methods, we are also thinking about a way to translate hierarchical graphs into VOnDA modules, to facilitate the implementation of new applications.

Source Code and Documentation

The VOnDA core system, core can be downloaded at `git@github.com:bkiefer/vonda.git`. The main page has detailed instructions for the installation of external dependencies. The debugger currently lives in a separate project: `git@`

github.com:yoshegg/rudibugger.git. Both projects are licensed under the Creative Commons Attribution-NonCommercial 4.0 International License[2], and are free for all non-commercial use. A screencast showing the GUI functionality and the running PAL system is available at https://youtu.be/nSotEVZUEyw.

Acknowledgments

The research described in this paper has been funded by the Horizon 2020 Framework Programme of the European Union within the project PAL (Personal Assistant for healthy Lifestyle) under Grant agreement no. 643783.

References

ALIZ-E. 2010. ALIZ-E project. http://www.aliz-e.org/.

Christophe Biwer. 2017. rudibugger - Graphisches Debugging der Dialogmanagementtechnolgie VOnDA. Bachelor's Thesis, Saarland University.

Dan Bohus and Alexander I Rudnicky. 2009. The RavenClaw dialog management framework: Architecture and systems. *Computer Speech & Language*, 23(3):332–361.

Harry Bunt, Jan Alexandersson, Jae-Woong Choe, Alex Chengyu Fang, Koiti Hasida, Volha Petukhova, Andrei Popescu-Belis, and David R Traum. 2012. ISO 24617-2: A semantically-based standard for dialogue annotation. In *LREC*, pages 430–437. Citeseer.

Christiane Fellbaum. 1998. *WordNet*. Wiley Online Library.

Patrick Gebhard, Gregor Mehlmann, and Michael Kipp. 2012. Visual SceneMaker—a tool for authoring interactive virtual characters. *Journal on Multimodal User Interfaces*, 6(1-2):3–11.

Filip Jurčíček, Ondřej Dušek, Ondřej Plátek, and Lukáš Žilka. 2014. Alex: A statistical dialogue systems framework. In *International Conference on Text, Speech, and Dialogue*, pages 587–594. Springer.

Hans-Ulrich Krieger. 2012. A temporal extension of the Hayes/ter Horst entailment rules and an alternative to W3C's n-ary relations. In *Proceedings of the 7th International Conference on Formal Ontology in Information Systems (FOIS)*, pages 323–336.

Hans-Ulrich Krieger. 2013. An efficient implementation of equivalence relations in OWL via rule and query rewriting. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 260–263. IEEE.

[2] http://creativecommons.org/licenses/by-nc/4.0/

Hans-Ulrich Krieger. 2014. A detailed comparison of seven approaches for the annotation of time-dependent factual knowledge in rdf and owl. In *Proceedings 10th Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation*.

Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim de Greeff, Koen Hindriks, Mark Neerincx, Petter Ögren, Tomáš Svoboda, and Rainer Worst. 2015. TRADR project: Long-term human-robot teaming for robot assisted disaster response. *KI-Künstliche Intelligenz*, 29(2):193–201.

Pierre Lison and Casey Kennington. 2015. Developing spoken dialogue systems with the OpenDial toolkit. *SEMDIAL 2015 goDIAL*, page 194.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Robert Neßelrath and Michael Feld. 2014. SiAM-dp: A platform for the model-based development of context-aware multimodal dialogue applications. In *Intelligent Environments (IE), 2014 International Conference on*, pages 162–169. IEEE.

PAL. 2018. The PAL Project Website. http://www.pal4u.eu/. Last access: 11.10.2017.

Josef Ruppenhofer, Michael Ellsworth, Miriam RL Petruck, Christopher R Johnson, and Jan Scheffczyk. 2016. *FrameNet II: Extended theory and practice*. Institut für Deutsche Sprache, Bibliothek.

Tim Schwartz, Ingo Zinnikus, Hans-Ulrich Krieger, Christian Bürckert, Joachim Folz, Bernd Kiefer, Peter Hevesi, Christoph Lüth, Gerald Pirkl, Torsten Spieldenner, et al. 2016. Hybrid teams: flexible collaboration between humans, robots and virtual agents. In *German Conference on Multiagent System Technologies*, pages 131–146. Springer.

Gabriel Skantze and Samer Al Moubayed. 2012. IrisTK: a statechart-based toolkit for multi-party face-to-face interaction. In *Proceedings of the 14th ACM international conference on Multimodal interaction*, pages 69–76. ACM.

David R Traum and Staffan Larsson. 2003. The information state approach to dialogue management. In *Current and new directions in discourse and dialogue*, pages 325–353. Springer.

Stefan Ultes, Lina M Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Inigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gasic, et al. 2017. Pydial: A multi-domain statistical dialogue system toolkit. *Proceedings of ACL 2017, System Demonstrations*, pages 73–78.

Anna Welker. 2017. Wissensbasiertes Dialogmanagement für Social Talk. Bachelor's Thesis, Saarland University.

C Bachelor Theses

Universität des Saarlandes

Fakultät 4 - Philosophische Fakultät II
Sprach-, Literatur- und Kulturwissenschaften

Bachelorarbeit zur Erlangung des akademischen Grades

Bachelor of Science

im Fach

Computerlinguistik

RUDIBUGGER

Graphisches Debugging der Dialogmanagementtechnologie VOnDA

Name	Christophe Jules Ralph Biwer
Matrik.	2554932
Kontakt	cbiwer@coli.uni-saarland.de

Gutachter	Dipl.-Inf. Bernd Kiefer
	Prof. Dr. Josef van Genabith

19. Dezember 2017

Ich möchte mich bei all den Menschen bedanken, die mich während der Anfertigung dieser Bachelorarbeit beraten, motiviert und unterstützt haben.

Mein ganz besonderer Dank gilt Sophie Henning, für ihren moralischen und emotionalen Rückhalt sowie ihre Freundschaft und Hilfsbereitschaft während meines Studiums.

Ebenfalls möchte ich Bernd Kiefer, der diese Arbeit betreut und begutachtet hat, danken. Seine fachliche Kompetenz, hilfreichen Anregungen und konstruktive Kritik haben maßgeblich dazu beigetragen, dass diese Bachelorarbeit in dieser Form vorliegt.

Außerdem möchte ich Anna Welker danken, die mir ebenfalls durch ihre fachliche Kompetenz, aber auch durch ihren freundschaftlichen Rat zur Seite stand.

Abschließend gilt mein Dank natürlich auch meinen Eltern Arlette Fischer und Marc Biwer, sowie meiner Schwester Isabelle Biwer, die mich während meines Studiums unterstützt haben.

Aus Gründen der leichteren Lesbarkeit wird in der vorliegenden Bachelorarbeit meist darauf verzichtet, in der Programmierwelt übliche Ausdrücke vom Englischen ins Deutsche zu übersetzen. Außerdem wird an vielen Stellen die CamelCase-Notation verwendet, um die Assoziation zu den verwendeten Java- sowie JavaFX-Klassen zu verdeutlichen.

Selbstständigkeitserklärung

Saarbrücken, den 20. Dezember 2017

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Ferner habe ich die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, unter Angabe der jeweiligen Quelle als Entlehnung kenntlich gemacht. Dies gilt ebenso für Zeichnungen, Skizzen und Pläne sowie bildliche und grafische Darstellungen, die nicht eigenhändig von mir angefertigt wurden.

Christophe Biwer

Abstract

In dieser Bachelorarbeit wird ein Debugger namens RUDIBUGGER für die Dialogmanagementtechnologie VOnDA vorgestellt. Hierbei handelt es sich um ein regelbasiertes Dialogmanagement-Framework mit einer probabilistischen Komponente, um Dialogsysteme zu implementieren. RUDIBUGGER verfügt einerseits über Funktionalität, die das Schreiben von Regeln in der Regelsprache von VOnDA unterstützt, andererseits erlaubt RUDIBUGGER einen dynamischen Zugriff auf den Laufzeitkern von VOnDA und ermöglicht so das Visualisieren des Ablaufs sowie die Identifizierung von Problemen in der Regelspezifikation des Dialogsystems.

Inhaltsverzeichnis

Einleitung

Regelbasiertes Dialogmanagement erreicht ab einer bestimmten Anzahl an Regeln eine zu hohe Komplexität. Aus diesem Grund besteht Bedarf an einer Möglichkeit, das zugrundeliegende System in einer Weise betrachten zu können, die Rückschlüsse auf den Ablauf eben dessen erlaubt. Man benötigt deshalb einen sogenannten *Debugger*.

Diese Bachelorarbeit beschreibt die Entwicklung und Funktionsweise eines Debuggers namens RUDIBUGGER [rudiˈbʌgɐ] für die Dialogmanagementtechnologie VONDA [ˈwʌnda].

Im Rahmen des Projekts PAL[1] (***P**ersonal **A**ssistant for a healthy **L**ifestyle*) wird ein System entwickelt, das an Diabetes Typ 1 erkrankte Kinder dabei unterstützt, das nötige Wissen über den Umgang mit ihrer Krankheit zu erlernen. Dazu gehören beispielsweise die regelmäßige Kontrolle des Blutzuckerspiegels oder das Führen eines Ernährungstagebuchs. Da die Nutzer des Systems zwischen sieben und vierzehn Jahren alt sind, wird ihnen ein Roboter zur Verfügung gestellt, der sie bei diesen Aufgaben unterstützt und dazu motivieren soll. So ist es zum Beispiel möglich, mit dem Roboter ein Quiz oder ein Sortierspiel zu spielen, bei denen das Kind spielerisch an das nötige Alltagswissen herangeführt wird. Während der Benutzung des Systems unterhält sich der Roboter mit dem Kind; dazu werden handgeschriebene Dialogregeln genutzt.

VONDA wurde am DFKI im Fachbereich MLT (***M**ultimodal **L**anguage **T**echnologies*) entwickelt. Dieses System soll in Zukunft in das Projekt PAL integriert werden und das Dialogmanagement übernehmen.

In Kapitel 1 beschäftige ich mich zuerst mit dem theoretischen Hintergrund dieser Technologie, um mich dann in Kapitel 2 mit VONDA selbst zu befassen. Kapitel 3 beschreibt die Implementierung von RUDIBUGGER wie zum Beispiel dessen Funktionalität und die Verbindung zu VONDA. Zuletzt blicke ich in Kapitel 4 auf andere Dialogsysteme und vergleichen diese mit der Funktionsweise von VONDA beziehungsweise RUDIBUGGER.

[1] `http://www.pal4u.eu/`

Persönliche Motivation

Persönlich ziehe ich es vor, praktisch orientiert zu arbeiten. Ich bin mir selbstverständlich darüber im Klaren, dass Praxis nur schwer ohne zugrundeliegende Theorie umsetzbar ist; allerdings habe ich mir zum Ziel gesetzt, in meiner Bachelorarbeit etwas zu erarbeiten, was praktischen Nutzen hat und auch nach dem Fertigstellen der Arbeit benutzt und weiterentwickelt wird.

Ein Problem, das mir während des Bachelorstudiums öfter aufgefallen ist, ist das Nichtvorhandensein von (guten) Benutzeroberflächen, die das Nutzen von computerlinguistischer Software vereinfachen würden. Ein persönliches Interesse daran, zu wissen, wie man seine eigene GUI entwickelt und diese mit Software aus dem computerlinguistischen Bereich nutzen kann, ist also meinen eigenen Erfahrungen geschuldet. Darüber hinaus habe ich im Rahmen meiner Arbeit als Hilfswissenschaftler mehrmals Benutzeroberflächen erweitern bzw. umbauen müssen, ohne jedoch eine eigene Anwendung von Grund auf zu entwickeln.

Eine vorhandene GUI ist außerdem nicht zwangsläufig von Vorteil, wenn sie schlecht dokumentiert oder ganz allgemein nur schwer zu verstehen und zu nutzen ist. Aus diesem Grund habe ich auch sehr darauf geachtet, eine gute Usability zu erzielen, damit meine Arbeit, wie eingangs erwähnt, auch in Zukunft noch Verwendung findet.

1 Theoretischer Hintergrund

Dialogmanagement (*Dialog Management*) ist unverzichtbar für ein Dialogsystem. Leider gibt es keinen allgemeinen Konsens darüber, was Dialogmanagement genau ist. In diesem Kapitel befasse ich mich deshalb mit den Definitionen von Dialog Management sowie der Information State Approach von Traum und Larsson.

1.1 Dialog Management

Traum und Larsson definieren Dialog Management folgendermaßen:

- updating the dialogue context on the basis of interpreted communication (both that produced by the system and by other communicating agents, be they human 'user" or other software agent)
- providing context-dependent expectations for interpretation of observed signals as communicative behavior
- interfacing with task/domain processing (e.g., database, planner, execution module, other back-end system), to coordinate dialogue and non-dialogue behavior and reasoning
- deciding what content to express next and when to express it

– Traum und Larsson (2003, S. 2)

Die Autoren merken allerdings an, dass sich einige Funktionen überschneiden, so dass man als Entwickler auch dazu tendieren könnte, eine Komponente für mehrere Funktionen zu definieren. Die Entscheidung dafür wird oft aufgrund externer Faktoren wie dem Vorhandensein bestimmter Software getroffen.

1.2 Information State Approach

Information State eines Dialogs wird von Traum und Larsson wie folgt definiert:

> The term information state of a dialogue represents the information necessary to distinguish it from other dialogues, representing the cumulative additions from previous actions in the dialogue, and motivating future action.
>
> – Traum und Larsson (2003, S. 3)

Als Beispiel werden Fragen genannt, die dann andere Teilnehmer des Dialogs zu eigenen Äußerungen motivieren.

Traum und Larsson teilen ihren Ansatz in fünf Bestandteile auf:

- ***Informational components***: Hierbei handelt es sich um allgemeinen Kontext (sogenanntes Weltwissen) sowie Eigenschaften der einzelnen Komponenten. Als Beispiel seien hier Agenten sowie deren Wünsche und Ziele genannt.
- ***Formal representations***: Eine formale Repräsentation der Informational Components, zum Beispiel als Listen, Sets oder mithilfe modaler Operatoren.
- ***Dialogue moves***: Aktionen, die ein Updaten des Information States zur Folge haben. Meist sind dies externe Aktionen, wie natürlichsprachliche Äußerungen.
- ***Update rules***: Diese Regeln verwalten das Updaten des Information States, je nachdem, welche Bedingungen erfüllt sind. Dabei werden sowohl der aktuelle Information State als auch die getätigten Dialogue Moves betrachtet.
- ***Update strategy***: Hiermit wird entschieden, welche Regeln zu welchem Zeitpunkt angewendet werden. Dazu wird aus dem Set der momentan anwendbaren Regeln gewählt, wobei diese Auswahl von verschiedenen Faktoren wie beispielsweise einer statistischen Komponente abhängt.

Der Information State enthält somit Informationen über den aktuellen Dialogstatus, aber auch über Wünsche und Ziele der einzelnen Agenten (Traum & Larsson, 2003, S. 6).

2 VOnDA

RUDIBUGGER ist ein Debugger für VOnDA. Um zu verstehen, wozu RUDIBUGGER benötigt wird, wird deshalb an dieser Stelle zuerst erklärt, worum es sich beim Projekt VOnDA handelt.

VOnDA (**V**ersatile **On**tology-based **D**ialogue and **A**gent Platform) ist ein regelbasiertes Dialogmanagement-Framework mit einer probabilistischen Komponente, um Dialogsysteme zu implementieren. VOnDA basiert auf dem *Information State-Update Approach* (vgl. Kapitel 1.2) und enthält Ideen aus BDI-Agentenarchitekturen (Bratman, 1987), ein Ansatz, der viele Parallelen zum Information State-Update aufweist.

Die Idee hinter BDI (*Belief Desire Intent*), einer Architektur für Software-Agenten, ist es, künstliche Agenten mit mentalen Fähigkeiten auszustatten. Dazu gehören

- Weltwissen (*beliefs*), mit Informationen über den aktuellen Zustand der Umwelt, in der der Agent agiert,
- Ziele (*desires*), die das Verhalten des Agenten bestimmen,
- sowie Absichten (*intentions*), eine hierarchische organisierte Plandatenbank, aus der der Agent einen Plan auswählt, der ihn seinem Ziel näher bringt.

Angewandt auf das PAL-System heißt das, dass das Weltwissen in der Ontologie gespeichert wird und die Absichten den formulierten Regeln entsprechen. Die Ziele aus der BDI-Architektur könnte man etwa gleichsetzen mit den Aufgaben, den Blutzuckerspiegel zu kontrollieren oder die impliziten Wünsche des Kindes zu respektieren.

Ein wichtiger Aspekt von VOnDA ist die Implementierung dieses Ansatzes als spezialisierte RDF/OWL-Datenbank (vgl. Kapitel 2.2), in der alle Spezifikationen und Daten in einer uniformen Repräsentation vorliegen. Die Spezifikation der Dialogakte und der Domäne im RDF-Format ermöglicht außerdem das Etablieren von Relationen unter den verschiedenen Elementen. Zudem werden alle Fakten mit einem Zeitstempel annotiert. Dies ermöglicht eine dynamische Sicht auf das „Gedächtnis" des Systems.

```
Beispielregel:                    // Label
if (child.forename) {             // Existenztest
  s = child.forename;             // Variable binden
  propose("activity") {           // Vorschlag
    emitDA(#Greet(Initial));      // Dialogakt senden
  }
}
```

Abbildung 1: Beispielregel in `.rudi`-Code

VOnDA besteht aus zwei Teilen: einem Compiler zum Übersetzen der Regelsprache in ausführbaren Code (vgl. Kapitel 2.3) und einem Laufzeitkern, auf dem ein konkretes Dialogsystem aufgesetzt wird (vgl. Kapitel 2.4). RUDIBUGGER ist in Kombination mit dem Laufzeitsystem ein nützliches Werkzeug zur Visualisierung des Ablaufs sowie zur Identifizierung von Problemen in der Regelspezifikation.

2.1 Regeln schreiben

Condition-Action-Regeln sind das Herzstück der Dialogmanagmenttechnologie von VOnDA. Die Regeln können in mehrere Dateien mit der Dateiendung `.rudi` aufgeteilt werden, die in etwa Modulen entsprechen. Unverzichtbar ist das Vorhandensein einer obersten Regeldatei, die von einer eigenen Implementierung des *Agents* (vgl. Kapitel 2.4) abgeleitet ist. Zusätzlich dazu können andere Regeldateien rekursiv importiert werden.

Innerhalb der einzelnen Regeldateien können beliebig viele Regeln definiert werden, die auch rekursiv weitere Regeln enthalten. Eine Regel ist eine bedingte Anweisung (*Conditional*), die ein Label trägt, mit dem der Debugger (und der Entwickler) die Regel identifiziert. In Abbildung 1 wird eine (sinnfreie) Regel gezeigt.

Die Regelsprache von VOnDA erleichtert durch spezielle syntaktische Konstrukte die Benutzung der Datenbank. Die Sprache ist an die Syntax von Java angelehnt, in die die Regeln letztendlich übersetzt werden. So sehen RDF-Objekte im Regelsystem wie Java-Objekte aus, und das Auslesen der Properties eines Objektes aus der Ontologie (vgl. Kapitel 2.2) gleicht einem Java-Feldzugriff. VOnDA macht Gebrauch von der OWL-Klassenhierarchie, um intelligente Typinferenz durchzuführen

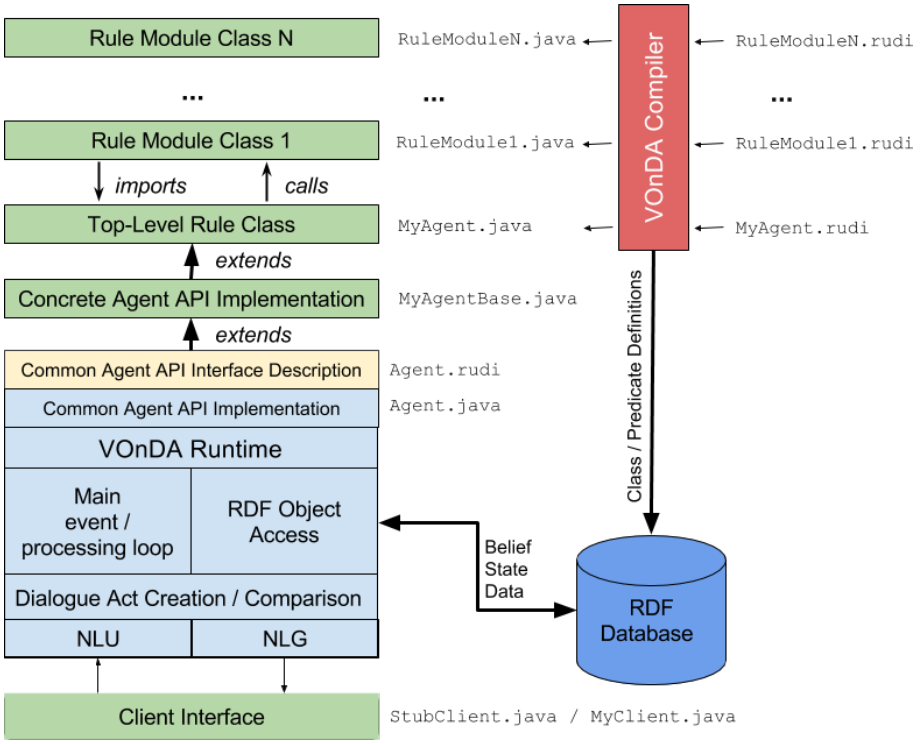


Abbildung 2: Schematische Darstellung des Aufbaus von VOnDA

und dadurch die Regeln so kompakt wie möglich schreiben zu können.

Zusätzlich erlaubt es der transparente Zugriff auf Java-Objekte und Methoden, komplexere Funktionalität zur Verfügung zu stellen, die ansonsten in der Regelsprache schlecht zu implementieren wäre.

Der schematische Aufbau von VOnDA wird in Abbildung 2 gezeigt. Diese Grafik wurde aus der Dokumentation (vgl. Anhang A) entnommen.

2.2 HFC

Der Information State von VOnDA wird als Ontologie und RDF-Repository implementiert. Dieser wird durch HFC implementiert, einen am MLT entwickelten Ontologieserver mit speziellen Reasoningfähigkeiten. Um den Rahmen dieser Arbeit nicht zu sprengen, sei an dieser Stelle auf das README von HFC verwiesen:

> HFC is a bottom-up forward chainer and semantic repository imple-

> mented in Java [...]. HFC supports RDFS and OWL reasoning [...], but at the same time provides a powerful language for defining custom rules, involving functional and relational variables, complex tests and actions, and the replacement of triples in favor of tuples of arbitrary length [...].
>
> – Hans-Ulrich Krieger

Die Verbindung von HFC und VOnDA wird durch ein drittes Projekt namens HFC-DATABASE hergestellt. In VOnDA werden einige der Funktionen dieses Projektes verwendet.

2.3 Compiler

Der Compiler von VOnDA ist dafür zuständig, vorgegebene `.rudi`-Dateien in Java-Code umzuwandeln. Dazu wird eine eigens für diesen Zweck geschriebene ANTLR-Grammatik (***AN****other* ***T****ool for* ***L****anguage* ***R****ecognition*) beim Parsen des `.rudi`-Codes verwendet. Abbildung 3 zeigt diesen erzeugten Baum basierend auf der Regel aus Abbildung 1. Nun wird der erzeugte `AntlrParseTree` vom sogenannten `ParseTreeVisitor` durchlaufen und ein abstrakter Syntaxbaum gebaut (*AST*, *abstract syntax tree*). Dieser AST wird dann von zwei weiteren *Visitors* traversiert, zunächst von `VisitorType`, der Typtests und -inferenzen durchführt, danach von `VisitorGeneration`, der die Generierung des `.java`-Codes übernimmt.

Der genaue Ablauf dieses Prozesses wird in Abbildung 4 schematisch dargestellt.

Während der Kompilierung greift VOnDA auf die Ontologie zu, um die Klassen von Objekten zu identifizieren, sowie Feldzugriffe richtig aufzulösen und sie gegebenenfalls als RDF-Zugriffe zu erkennen.

Jede generierte Java-Klasse enthält eine eigene `process()`-Methode, die die Regeln sequentiell abarbeitet. Von der `process()`-Methode der obersten Regeldatei wird zur Laufzeit die Auswertung aller Regeln angestoßen.

2.3.1 VisitorType

Während der geparste Baum vom `VisitorType` durchlaufen wird, werden mehrere Aspekte überprüft:

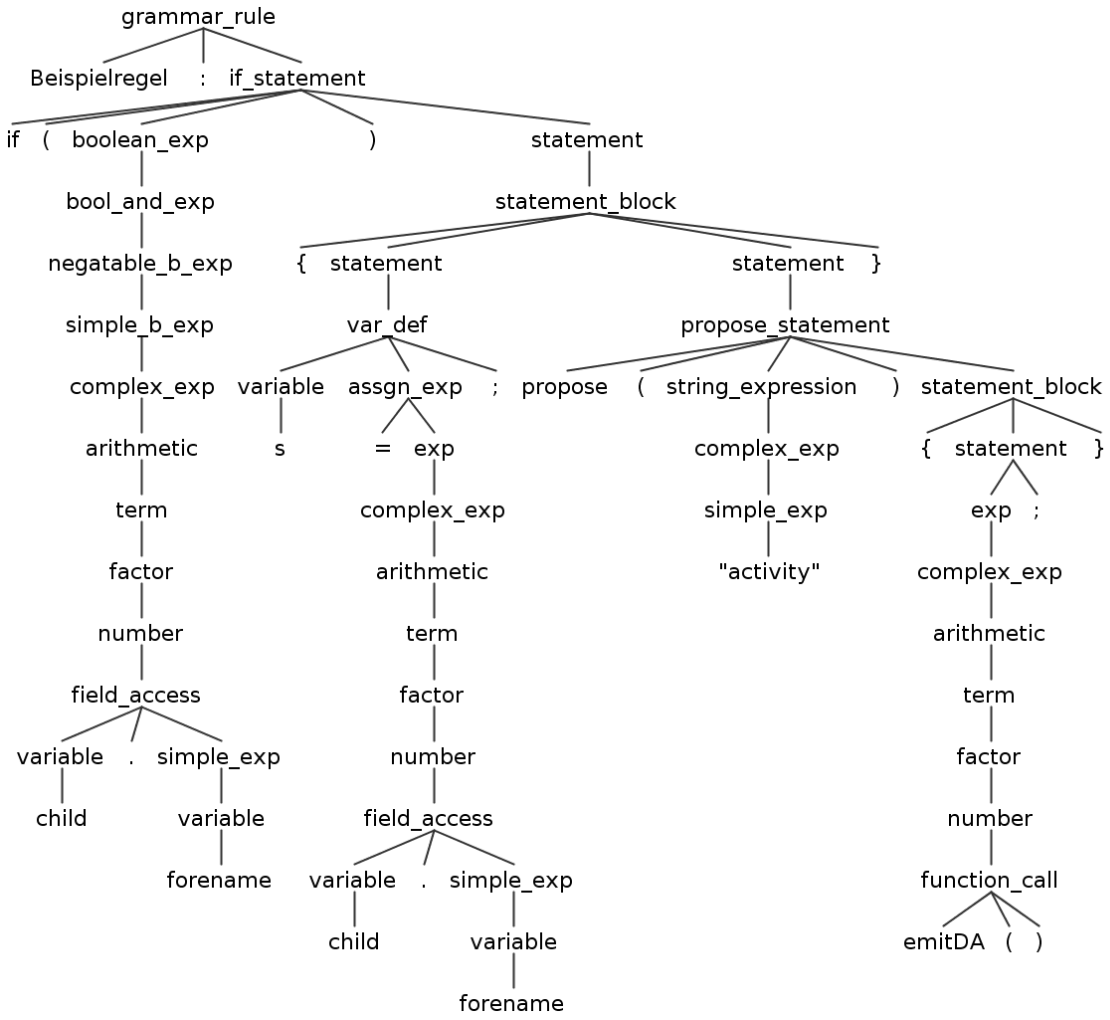


Abbildung 3: `AntlrParseTree` der Beispielregel aus Abbildung 1

- **Typentest**: Hierbei wird, wie bei einem normalen Compiler, überprüft, dass keine Operationen wie `"foo" + 1` im Code vorhanden sind, oder dass Werte automatisch ineinander umgeformt werden, falls es sinnvoll möglich ist.

- **Zugriffscheck**: Es wird überprüft, ob ein Prädikat, das auf ein Objekt angewandt wird (z.B. `child.forename`), überhaupt in der Ontologie für das Objekt definiert ist. Ist dies der Fall, wird der korrekte *Namespace* gewählt, so dass ein Zugriff mit dem vollständigen Prädikat möglich ist. Aus `child.forename` im `.rudi`-Code wird im Java-Code `child.getSingleValue("<dom:forename>")`.

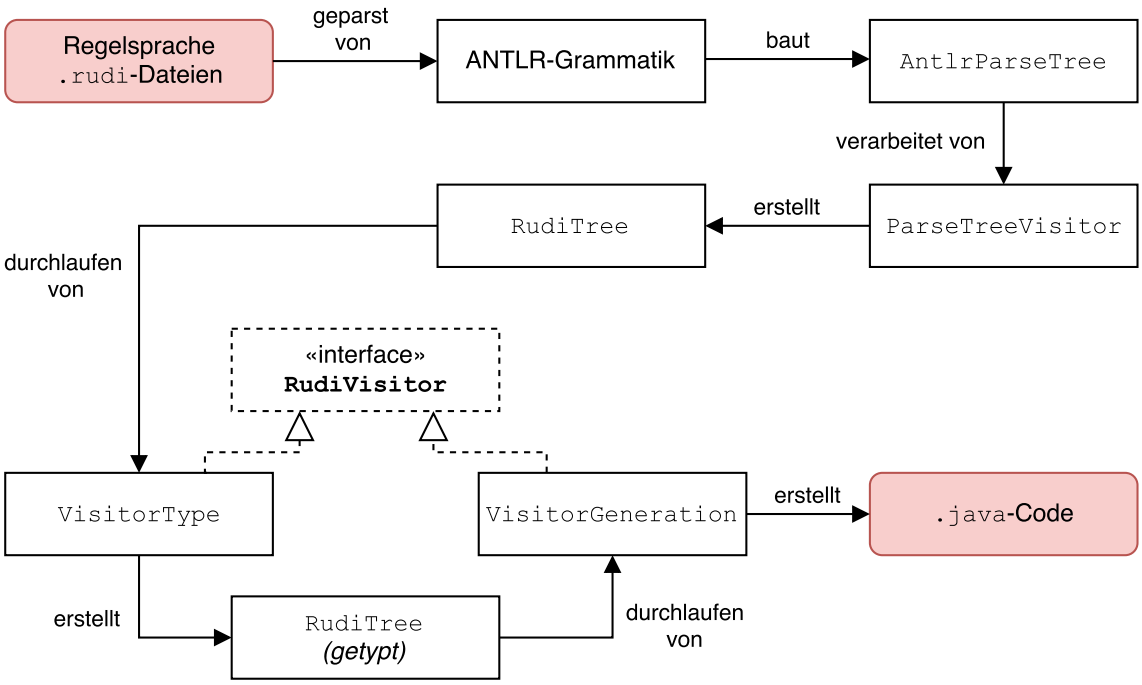


Abbildung 4: Schematische Darstellung des Kompilierens von `.rudi`-Code zu `.java`-Code

- **Typinferenz**: Ähnlich wie in *Python* ist es möglich, den Typ einer Variablen automatisch inferieren zu lassen: `x = 3` wird also zu `int x = 3`.

- **Existenztest**: In `.rudi`-Code ist es wie in C oder Python möglich, dass nicht-boolesche Werte als solche interpretiert werden, zum Beispiel eine leere Liste (`liste = []`) als `False`. Da Java dies nicht beherrscht, wird ein solcher Test automatisch expandiert, im vorliegenden Beispiel zu `liste.isEmpty()` .

2.3.2 VisitorGeneration

Im letzten Schritt wird aus dem AST Java-Code generiert. Dabei werden überladene Operatoren wie `<=` in das richtige Java-Äquivalent umgewandelt, so wird zum Beispiel ein Subklassentest ausgeführt, wenn der Operator auf RDF-Klassen angewandt wird.

2.4 Laufzeitsystem

Auf das Laufzeitsystem von VOnDA greift man innerhalb eines konkreten Projektes zu, indem man von der abstrakten Klasse `Agent` (im Folgenden nur *Agent* genannt) ableitet und die abstrakten Methoden implementiert.

Der Agent stellt ein Framework zur Verfügung, das Methoden und Felder bereitstellt, die als universell für Dialogmanagement angesehen werden und für dessen Handling unabdinglich oder nützlich sind. So gibt es zum Beispiel die Methode `lastDA()`, die die letzte Äußerung des Benutzers im momentanen Dialog zurückgibt.

In einem eigenen Projekt wird eine Klasse definiert, die vom Agent ableitet. Hier werden neben den in `Agent` abstrakten Funktionen auch Zusatzfunktionen deklariert, die für das eigene System benötigt werden. Diese Klasse wird als Wrapperklasse bezeichnet.

Das System ist reaktiv, das heißt, eintreffende Events aus der Außenwelt oder Änderungen der Datenbank triggern die Ausführung der Regeln. Durch die Regeln werden entweder direkt Systemaktionen ausgeführt, wie zum Beispiel eine Änderung des *Information-States*, oder es werden sogenannte *Proposals* erzeugt, die als benannte *closures* implementiert sind. Das wiederum bedeutet, dass die im Proposal definierte Aktion gekapselt wird und einer Queue hinzugefügt wird.

Die Regeln werden in einer Art Fixpunktberechnung so lange ausgeführt, bis keine Änderungen der Datenbank oder neue Proposals erzeugt werden. Dann wird mittels einer statistischen Komponente aus den aktuellen Proposals das in der jeweiligen Situation am besten passende Proposal ausgewählt und ausgeführt. Die nicht ausgeführten Proposals werden verworfen.

2.5 Notwendigkeit eines Debuggers

Das Problem bei regelbasierten Systemen ist, dass durch eine sehr große Menge an Regeln der Überblick über die Regeln selbst sowie über deren Ausführung verloren geht.

Standardmäßig wird jede einzelne deklarierte Regel geloggt. So werden beispielsweise beim Start des PAL-Projekts 10 Regeln ausgewertet, die in der Folge auch alle unübersichtlich in der Konsole, zusammen mit anderen Systemmeldungen, angezeigt werden.

Aus diesem Grunde möchten Entwickler eines Dialogsystems die Möglichkeit haben, auf effiziente Art und Weise nachvollziehen zu können, wie die Regeln durchlaufen und ausgewertet werden. Man braucht Tools zur Strukturierung und Darstellung des Laufzeitsystems und die Möglichkeit, dynamisch in eben dieses einzugreifen. Die Notwendigkeit solcher Funktionalität begründet den Bedarf an einem Debugger.

3 Implementierung

In diesem Kapitel beschäftige ich mich mit den aus programmiertechnischer Sicht interessanten Aspekten, die in dieser Arbeit Verwendung finden. Ich biete einen Überblick über die verwendeten Entwicklungswerkzeuge, den Aufbau von RUDIBUGGER sowie einige genutzte Patterns und sonstige spezielle Java-Konstrukte.

In Anhang B befindet sich außerdem ein Schema, das die Beziehungen der wichtigsten Klassen und Funktionen von RUDIBUGGER ausführlich darstellt.

Abbildung 5 zeigt eine RUDIBUGGER-Instanz, die gerade auf das laufende PAL-System zugreift.

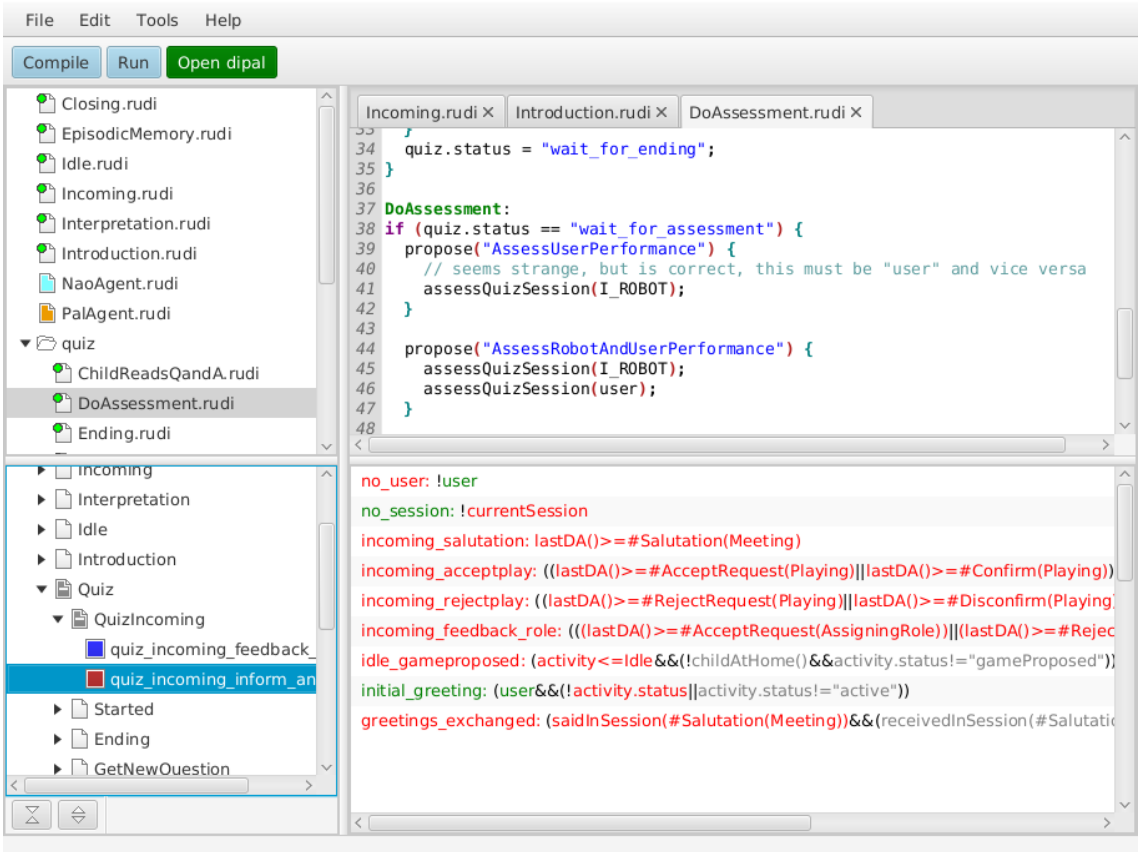


Abbildung 5: Screenshot von RUDIBUGGER

3.1 Entwicklungsumgebung

Entwickelt wird RUDIBUGGER mithilfe der *Netbeans*-IDE. Als Framework für den grafischen Teil kommt *JavaFX* zum Einsatz. Zur Konfiguration der Oberfläche wird das Programm *Scene Builder* der Firma *Gluon* verwendet. Zur Versionsverwaltung wird *git* genutzt. Um die Abhängigkeiten zu anderen Projekten zu verwalten, wird *Apache Maven* eingesetzt.

Netbeans bietet gute Möglichkeiten, mit JavaFX zu arbeiten. So ist es zum Beispiel möglich, ein neues JavaFX-Projekt mit Maven-Support zu erstellen.

Prinzipiell ist die Idee hinter Maven überzeugend, allerdings ist die Umsetzung sehr fehleranfällig, wenn zum Beispiel Dependenzen nicht kompilierbar sind.

Versionskontrolle, in diesem Fall mit git, hat sich als ein sehr nützliches Werkzeug erwiesen, da es mir leicht möglich war, andere Ansätze betreffend der Realisierung mancher Features auszuprobieren und bei Bedarf leicht wieder mehrere Schritte rückgängig zu machen. Hilfreich ist außerdem die Unterscheidung zwischen Remote und lokalem Repository, was die Kollaboration mit anderen Entwicklern vereinfacht.

3.2 JavaFX

JavaFX ist ein Framework zum Erstellen grafischer Java-Anwendungen, das 2007 von *Oracle* veröffentlicht wurde. Es gilt als Nachfolger der in die Jahre gekommenen Frameworks *AWT* und *Swing*. Laut Oracle[2] können bestehende Swing-Anwendungen auch mit JavaFX-Funktionalität erweitert werden.

Wir haben JavaFX unter anderem aufgrund folgender Kriterien gewählt:

- SceneBuilder, um die benötigten `.fxml`-Dateien erstellen und auch bequem modifizieren zu können,
- das Nutzen von *CSS* (*Cascading Style Sheets*), um das Design des Programms flexibel zu ändern,
- eine moderne Benutzeroberfläche als Standard und native Widgets, die Gefallen beim Nutzer finden sollten, so dass er das Programm auch gerne nutzt,

[2] `https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm`

- das Vorhandensein von *Properties*, die die Implementierung von Observer-Patterns sehr leicht ermöglichen (vgl. Kapitel 3.7),
- JavaFX wird aktiv von Oracle entwickelt. SceneBuilder wurde von Oracle aufgegeben, wird aber von einer anderen Firma weiterentwickelt.
- Hi-DPI-Support[3],
- eine gute Dokumentation, oft mit Beispielcode,
- eine aktive Nutzergemeinde, die bei Fragen und Problemen behilflich ist.

3.3 Aufbau des Programms

RUDIBUGGER befolgt das ***MVC*-Pattern**, wobei MVC für *Model-View-Controller* steht. Erstmals verwendet wurde es 1979, um User-Interfaces in *Smalltalk-80*, eine objektorientierte Programmiersprache, zu schreiben (Gamma, Helm, Johnson & Vlissides, 1994, S. 4). Abbildung 6 zeigt eine schematische Darstellung der einzelnen Komponenten.

Das **Model** (Modell) beinhaltet die Daten des Programms. In einigen Varianten des MVC-Patterns beinhaltet das Model auch die Anwendungslogik des Programms. Die Anwendungslogik beschreibt, wie die jeweiligen Daten verarbeitet und verändert werden. Dies ist auch in RUDIBUGGER der Fall: Die Klasse, die die ganze Anwendungslogik verwaltet und koordiniert, heißt `DataModel`.

Die **View** (Präsentationsschicht) ist für die Darstellung der Daten im Programm zuständig. Die View in RUDIBUGGER besteht aus vier Teilen. Die Darstellung der verschiedenen Views wird in sogenannten *FXML*-Dateien festgelegt. Es gibt davon insgesamt vier: `editor.fxml`, `menuBar.fxml`, `sideBar.fxml` und `statusBar.fxml`. Erstellt und bearbeitet wurden diese Dateien mit SceneBuilder.

Die **Controller** (Steuerung) definieren, wie GUI-Elemente in den einzelnen Views auf Aktionen des Nutzers reagieren, und leitet die entsprechenden Befehle und Anfragen an das Model weiter.

Das MVC-Pattern ist nützlich für das Entwickeln von GUIs, da es die drei Hauptkomponenten voneinander entkoppelt und so auch automatisierte Tests der Anwen-

[3] Unterstützung sehr hoher Bildschirmauflösungen

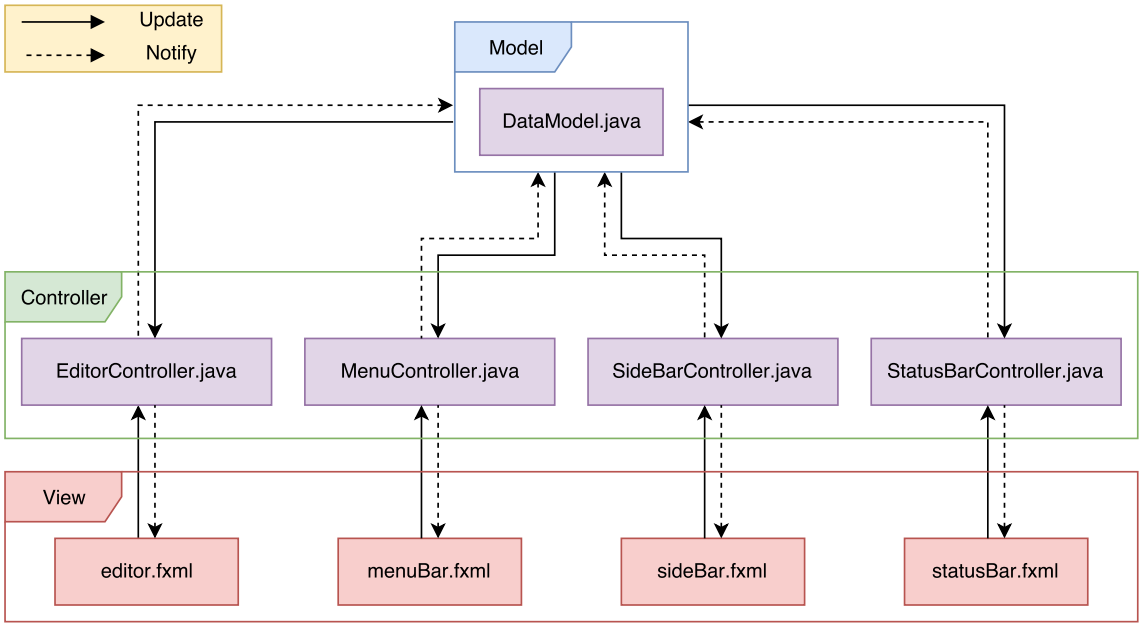


Abbildung 6: Aufbau von RUDIBUGGER

dungslogik ermöglicht. Diese sogenannte *Seperation of Concept* erhöht zudem die Kohärenz der einzelnen Bestandteile.

Beim Start des Programms werden die jeweiligen Controller dann ihrer View zugewiesen und anschließend mit dem Model initialisiert (siehe Abbildung 7).

3.4 Menubar, Toolbar und Statusleiste

Die Menüleiste (`menuBar.fxml`) ermöglicht den Zugriff auf die gängigen Funktionen des Programms: Man kann ein Projekt öffnen, neue `.rudi`-Dateien erstellen, Änderungen speichern und so weiter. Für das Erstellen einer solchen Leiste bietet JavaFX bereits eigene Klassen wie zum Beispiel `MenuBar`, `Menu` und `MenuItem` an.

Unterhalb der Menüleiste befindet sich außerdem eine *Toolbar*, die das Initiieren der Kompilierfunktion von `.rudi`-Dateien durch VONDA ermöglicht. So kann der Nutzer, ohne sein eigentliches Programm zu starten, die Regeln kompilieren lassen und mögliche Syntaxfehler noch innerhalb von RUDIBUGGER beheben.

Die verschiedenen Menüs und Buttons wurden im Sinne des *Command*-Patterns implementiert (Gamma et al., 1994, S. 203). Bei diesem Pattern geht es darum, eine Anfrage in einem Objekt zu kapseln und dann weiterzureichen. Das Anklicken

```
/* initialize View and Controller: menuBar */
FXMLLoader menuLoader = new FXMLLoader(getClass()
    .getResource("/fxml/menuBar.fxml"));
MenuController menuController
    = menuLoader.getController();

/* initialize DataModel */
/* ... */

/* bind Model to Controller */
menuController.initModel(model);
```

Abbildung 7: Beispiel anhand der Menüleiste: Binden von Controller und View sowie Initialisierung mit Model

der Bedienelemente in RUDIBUGGER ist jeweils mit einer Aktion verknüpft. Beim Klicken wird dann das Ausführen dieser Aktion initiiert, indem beim Model eine Anfrage gestellt wird, diese auszuführen. Umgesetzt wird die Aktion dann, wenn der JavaFX-Thread zur Verfügung steht.

Die Statusleiste (`statusBar.fxml`) soll dem Nutzer Feedback bei bestimmten Aktionen bieten. Zum Beispiel zeigt sie an, dass Änderungen an einer Datei gespeichert wurden.

3.5 Sidebar

Die Seitenleiste (`sideBar.fxml`) ist in zwei Teile getrennt: eine Baumansicht (`TreeView`) des Dateiverzeichnisses sowie eine Baumansicht, welche die jeweiligen momentan im Projekt genutzten Dateien und Regeln anzeigt.

Aufgrund ihrer Struktur ist eine `TreeView` ein komplexes Konstrukt, das Elemente verschiedener Klassen sowie eine weit verzweigte Hierarchie haben kann. Es gibt dementsprechend hier keine Funktion `setItems()`, die den Inhalt des Baumes festlegt, wie dies zum Beispiel bei einer `ListView` der Fall ist. Hier gibt man der `setItems()`-Funktion eine Liste als Argument mit und legt so den Inhalt fest. Bei einer `TreeView` hingegen legt man ein Wurzelelement der Klasse `TreeItem` fest und definiert dann rekursiv dessen Kinder, die von der gleichen Klasse sind. Als Argu-

ment übergibt man dem Konstruktor von `TreeItem` das Element, was die Daten des `TreeItem`s darstellt.

In Kapitel 3.8 gehe ich näher auf das Einlesen der Daten des Regelbaumes ein. In meinen ersten Versuchen habe ich allerdings das MVC-Pattern eklatant missachtet, indem ich im Konstruktor meiner eigens erstellten `TreeItem`-Klassen auch festgelegt habe, wie das Objekt schlussendlich in der `TreeView` aussehen soll. Ich habe also das Model mit der View vermischt. Mehrere Versuche, diesen Teil des Programms zu überarbeiten, sind an Zeitmangel sowie der bereits fortgeschrittenen Komplexität gescheitert. Unter anderem wird in diesen Klassen auch das Kontextmenü definiert, welches für das Setzen der Status verantwortlich ist.

Wie man eine `TreeView` richtig benutzt, zeigt allerdings die Darstellung des Dateisystems. Hier wird eine eigene Klasse (`RudiFolderHierarchy`) verwendet, welche die einzelnen `TreeItem`s erstellt und verbindet, sich allerdings auch merkt, in welcher Beziehung die jeweiligen Elemente zueinander stehen. So ist es auch möglich, gezielt ein Element zu entfernen oder an der richtigen Stelle einzufügen. Die Darstellung der Einträge wird über eine eigene Klasse namens `RudiTreeCell`, welche `TreeCell` erweitert, verwaltet. In dieser Klasse wird definiert, was genau aus dem mit dem `TreeItem` verbundenen Model extrahiert, eventuell angepasst und dann angezeigt wird. Standardmäßig wird die `toString()`-Methode des Wertes des TreeItems genutzt, die daher geeignet überschrieben werden muss. Schlussendlich muss man die `TreeView` so konfigurieren, dass sie als `CellFactory` diese Klasse nutzt:

```
rudiTreeView.setCellFactory(value -> new RudiTreeCell());
```

Das Anwenden dieser `CellFactory` entspricht dem Pattern der *Factory Method* (Gamma et al., 1994, S. 107). Bei diesem Pattern definiert man ein Interface zum Erstellen eines Objektes, überlässt jedoch den Unterklassen der View die Auswahl dessen, was denn genau instantiiert wird. Das Objekt wird dementsprechend nicht direkt über einen Konstruktor erzeugt.

In RUDIBUGGER wird `updateItem()` der Oberklasse `TreeCell` überschrieben. In dieser Funktion wird definiert, wie die Zelle aussehen soll, gegeben die darunter liegenden Daten.

3.5.1 Dateisystem

Die obere Baumansicht (`rudiTreeView`) zeigt das Dateiverzeichnis, in dem die `.rudi`-Dateien gespeichert sind. Diese `TreeView` wird automatisch aktualisiert, wenn eine Datei hinzugefügt oder gelöscht wird. Das Überwachen des Verzeichnisses geschieht mithilfe eines sogenannten `WatchService`s (cf. Kapitel 3.9).

Zur besseren Übersichtlichkeit für den Benutzer wird die Hauptdatei des Projektes mit einem orangefarbenen Symbol, die Wrapperklasse mit einem blauen Symbol und alle genutzten Dateien (Importe) mit einem grünen Symbol gekennzeichnet. Momentan nicht genutzte Dateien werden mit einem ausgegrauten Symbol angezeigt (vgl. Abbildung 5).

3.5.2 Regelansicht

Die untere Baumansicht (`ruleTreeView`) ist eine der zwei essentiellen Komponenten von RUDIBUGGER. An dieser Stelle kann der Benutzer festlegen, welche Regeln wie geloggt werden. Durch das Anklicken der jeweiligen `CheckBox` kann man die verschiedenen Status durchwechseln. Mit einem Rechtsklick kann man den Status des angeklickten Elements sowie seiner Kinder auf einmal festlegen. Auch ist es möglich, die dahinterliegende Datei im Editor zu öffnen und direkt zur gewählten Regel zu springen.

Da die benötigten Informationen zur Darstellung der Regeln in einer durch VONDA erzeugten `.yml`-Datei (vgl. Kapitel 3.8) stehen, können sich die Regeln bei jedem neuen Kompilieren des Projektes ändern. Aus diesem Grund gibt es auch hier einen `WatchService` (vgl. Kapitel 3.9), der dafür sorgt, dass die `TreeView` automatisch aktualisiert wird. Außerdem werden die Status der Regeln beibehalten, indem sie vor dem Aktualisieren des Baums gespeichert und anschließend wieder eingelesen werden. RUDIBUGGER merkt sich auch die Auswahl bei Regeln, die zeitweilig deaktiviert und dann reaktiviert wurden.

Das Zwischenspeichern dieser Auswahl entspricht ungefähr dem **Memento**-Pattern (Gamma et al., 1994, S. 283). Zwar kann man momentan noch nicht die Auswahl externalisieren, so dass man von außerhalb des Programms darauf zugreifen kann, aber zumindest das Erfassen innerhalb des Programms funktioniert. Das Zwischenspeichern ist allerdings für eine spätere Version des Programms geplant (die nötigen

Vorbereitungen wurden bereits getroffen), allerdings ist es nicht möglich, die Daten mit YAML zu serialisieren, ohne innerhalb der jeweiligen Klassen `public getter` und `setter` zu implementieren. Dies widerspricht allerdings dem Pattern, da so die Datenkapselung (das heißt das Verhindern des Zugriffs auf die innere Datenstruktur eines Objektes von außerhalb) missachtet wird. Darauf wird auch in Kapitel 3.8 eingegangen.

3.6 Editor

RUDIBUGGER ist keine IDE, sondern ein Debugger. Dennoch verfügt RUDIBUGGER über eine gewisse Funktionalität, die mit einer IDE vergleichbar ist. So kann man zum Beispiel `.rudi`-Dateien bearbeiten, mithilfe der `TreeViews` direkt zu bestimmten Stellen springen und innerhalb der `CodeArea` (dem Teil des Editors, in dem der `.rudi`-Code angezeigt wird) wird man durch Syntax-Highlighting visuell unterstützt (vgl. Kapitel 3.10). Zudem können mehrere Tabs geöffnet werden um ein gewisses Multitasking zu ermöglichen. Diese eingeschränkte IDE-Funktionalität soll spontane Änderungen während des Debuggens ermöglichen.

Zusätzlich zum Editor gibt es in dieser View (`editor.fxml`), bei bestehender Verbindung zu VONDA, eine `ListView`, welche den Logging-Output von VONDA anzeigt. Diese Ansicht ist die zweite essentielle Komponente von RUDIBUGGER.

3.7 Properties

Ein vor allem bei GUIs gängiges Entwurfsmuster (*Pattern*) ist das sogenannte **Observer**-Pattern (Gamma et al., 1994, S. 293). Hierbei geht es darum, verschiedene Komponenten (zum Beispiel Views) eines Programms zu benachrichtigen, wenn sich darunter liegende Daten (zum Beispiel das Model) ändern. In die andere Richtung teilt die View dem Model mit, wenn sich etwas an ihr geändert hat. Die Alternative zu diesem Pattern wäre, die View so zu implementieren, dass sie regelmäßig den Status des Models abfragt, was jedoch zu Lasten der Effizienz gehen würde. Die Komponenten sind außerdem entkoppelt: Die Verbindung ist flexibel, so dass die beobachtete Komponente nur das Nötigste über den Beobachter weiß.

Ein Beispiel aus RUDIBUGGER ist das Aktivieren des *Compile*-Buttons in der Toolbar. Dieser wird automatisch aktiviert, wenn sich der Zustand des Models verändert

```
/* this listener checks for a compile file */
_model.compileFileProperty()
        .addListener((o, oldVal, newVal) -> {
    if (newVal != null) {
        compileButton.setDisable(false);
    } else {
        compileButton.setDisable(true);
    }
});
```

Abbildung 8: Auszug aus `MenuController`: Binden eines `Change Listener`s an eine Property aus dem Model

hat, welches die Basis bildet. Im `DataModel` gibt es deshalb folgendes Feld:

```
_compileFile = new SimpleObjectProperty<>(null);
```

Die Klasse `SimpleObjectProperty` ist eine sogenannte *Property*. Properties werden in JavaFX (unter anderem) dazu genutzt, um Variablen aneinander zu binden. So werden Änderungen an einer Variable automatisch auf eine andere gebundene Variable reflektiert. In unserem konkreten Fall hängen wir im Controller (`MenuController`) einen `Change Listener` (Observer) an unser Model. Wenn sich das Model nun ändert, das heißt, wenn ein Projekt mit einem vorhandenen Compile-Skript geladen wurde, wird diese Datei der Property als Wert zugewiesen:

```
_compileFile.setValue(compilePath);
```

In der Folge löst der Listener aus und aktiviert den Button. Die dafür zuständige Funktion hat Zugriff auf den alten und den neuen Wert der Property sowie auf die Property selbst. Abbildung 8 zeigt diese Stelle im Code.

3.8 Beans

Die Hierarchie der Regeln ist sehr komplex. Dateien haben Regeln und können wiederum andere Dateien importieren. Regeln hingegen können Unterregeln haben, die dann wiederum auch Unterregeln haben können.

Zur Serialisierung dieser Informationen nutze ich *YAML*. Der Name ist ein rekursives Akronym und steht für *YAML Ain't Markup Language*. Dennoch handelt

es sich bei YAML um eine sogenannte Auszeichnungssprache (*markup language*), welche auch zur Datenserialisierung genutzt werden kann.

Um die Hierarchie der Regeln nach RUDIBUGGER zu übertragen, habe ich den Teil von VONDA modifziert, der die Importe und Regeln traversiert. Die gesammelten Daten habe ich dann in geschachtelten `Maps` gespeichert, welche dann mithilfe von SnakeYAML[4] serialisiert wurden. Beim Wiedereinlesen in RUDIBUGGER muss dann darauf geachtet werden, die verschiedenen Elemente richtig zu casten, was sehr aufwendig und fehleranfällig ist.

Glücklicherweise respektiert YAML die JavaBeans-Konvention. Ziel davon ist es, die Informationen in einem Objekt (dem *Bean*) zu speichern. Dazu müssen die verschiedenen Informationen (zumeist Felder einer Klasse) über `public getter`- und `setter`-Methoden verfügen, serialisierbar sein und einen leeren Konstruktor haben. Wir mussten deshalb nur eigene Klassen für Regeln und Importe erstellen, welche diese Bedingungen erfüllen. So ist es dann möglich, den Wurzelimport zu speichern (`RuleLoc.yml`) und über eine einfache Funktion in RUDIBUGGER wieder zu importieren.

Das Implementieren von `public getter` und `settern` verletzt allerdings die Datenkapselung und somit das Memento-Pattern (vgl. Kapitel 3.5.2). In Java gibt es keine einfache Möglichkeit, eine generelle Serialisierung zu implementieren, die die Kapselung vollständig sicherstellt.

3.9 WatchService

Ein `WatchService` ermöglicht das Überwachen eines Verzeichnisses. In RUDIBUGGER werden die Datei `RuleLoc.yml`, aber auch das Verzeichnis mit den `.rudi`-Dateien in Bezug auf Änderungen überwacht. Das `.rudi`-Verzeichnis kann weiter verzweigt sein, so dass auch die Unterordner überwacht werden müssen. Wegen der saubereren Trennung der Aufgaben wurde die Überwachung auf zwei `WatchService`s aufgeteilt.

Hierzu wird ein eigener Thread erstellt, welchem automatisch signalisiert wird, wenn eine Änderung (*Event*) im Dateisystem registriert wurde. Dafür gibt es verschiedene Arten von Events, je nachdem ob eine Datei erstellt, verändert oder

[4] `https://bitbucket.org/asomov/snakeyaml`

gelöscht wurde. Außerdem kann es zu einem *Overflow* kommen, wenn Änderungen möglicherweise übersehen wurden. Aufgrund unterschiedlicher Dateisysteme bei verschiedenen Betriebssystemen sowie den unterschiedlichen Speichermethoden von verschiedenen Programmen (wird eine Datei beim Speichern erweitert oder komplett überschrieben?) stellte das Implementieren eine besondere Herausforderung dar, die jedoch gelöst wurde: Änderungen werden RUDIBUGGER zuverlässig mitgeteilt.

Der Vorteil eines `WatchService`s besteht darin, wie beim Observer-Pattern üblich, Änderungen der Daten (hier das Dateiverzeichnis) an andere, davon abhängige Komponenten (hier RUDIBUGGER) weiterzugeben. So kann auf umständliches, manuelles Nachladen zum Beispiel der Regelstruktur durch *Polling* verzichtet werden.

3.10 Codeeditor

Eine sehr nützliche Erweiterung für JavaFX ist *RichTextFX*[5]. Diese Erweiterung ermöglicht das Verwenden von speichereffizienten `TextArea`s, um Programmcode darzustellen und zu bearbeiten. Eine `TextArea` ist ein Feld, welches mehrzeiligen Text darstellen kann. Zusätzlich kann man den Text bearbeiten, markieren und kopieren.

Meine eigene Klasse `RudiCodeArea` leitet von RichTextFXs `CodeArea` ab und war ursprünglich ein Beispiel für Java-Code. Da `.rudi`-Syntax vereinfacht ausgedrückt reduzierte Java-Syntax ist, genügte es, Syntax-Highlighting für Regelnamen und Importe über Pattern-Matching zu definieren. Die Farben werden wiederum über CSS (`rudi-keywords.css`) definiert.

Weitere Features sind das Hervorheben der momentanen Zeile, das Festlegen der momentanen Zeile von außen (zum Beispiel, indem man zu einer bestimmten Regel springt) sowie Zeilennummerierung. Außerdem sind manche Tastenkombinationen schon standardmäßig aktiviert, wie zum Beispiel ctrl+Z, um eine Änderung rückgängig zu machen.

RichTextFX respektiert allerdings nicht das MVC-Pattern, da dies laut den Entwicklern den Zugriff auf manche Komponenten der View verhindern würde.

[5] `https://github.com/FXMisc/RichTextFX`

3.11 Verbindung zu VOnDA

Der Hauptgrund für die Entwicklung von RUDIBUGGER war, wie bereits in Kapitel 2.5 erwähnt, die Möglichkeit, dynamisch auf das Laufzeitsystem von VONDA zuzugreifen und das Loggen der Regeln aktiv steuern und filtern zu können. Da beide Programme jedoch unabhängig voneinander starten, muss eine bidirektionale Verbindung hergestellt werden. Dabei muss RUDIBUGGER VONDA mitteilen, welche Regeln genau geloggt werden sollen, und VONDA muss RUDIBUGGER diese Daten für das gefilterte Log zur Verfügung stellen.

In einem ersten Anlauf haben wir versucht, VONDA und den Debugger in der gleichen JVM zu starten. Leider war dies nicht möglich, da das PAL-System mit *root*-Rechten läuft, somit keinen Zugriff auf das *Display* hat und deshalb auch kein grafisches Fenster anzeigen kann.

Nachdem wir diesen Ansatz verwerfen mussten, haben wir uns für eine Server-Client-basierte Lösung entschieden. Nach ersten Überlegungen, Protokolle wie XML-RPC (*Extensible Markup Language Remote Procedure Call*) einzusetzen, haben wir festgestellt, dass der Aufwand, diese Frameworks zu nutzen, für die einfachen Anforderungen in unserem Fall nicht gerechtfertigt ist.

Deshalb haben wir in der Folge einen eigenen Ansatz entwickelt, bei dem ein Server jeweils einen `Socket` öffnet und dann von einem sich verbindenden Client gesendete `String`-Arrays verarbeitet (*parst*). In diesen Arrays befinden sich an erster Stelle die aufzurufende Funktion und im Rest des Arrays die jeweiligen Parameter. Die Clients versuchen allerdings erst dann, die Verbindung herzustellen, wenn sie wirklich benötigt wird. So ist es möglich, zuerst RUDIBUGGER mit dem benötigten Projekt zu laden und dann das Livesystem zu starten, ohne dass es zu Verbindungsproblemen kommt.

Beim Festlegen des Loggingstatus einer oder mehrerer Regeln wird dies direkt an VONDA mitgeteilt und diese Einstellung wird dann nahtlos übernommen. VONDA überträgt gleichzeitig an RUDIBUGGER die Live-Daten für das Logging der Regeln. Innerhalb von RUDIBUGGER wird dies dann im rechten unteren Teil (vgl. Abbildung 5) angezeigt. An erster Stelle steht der Regelname, danach kommt das, was in der Regel definiert wurde. Diese einzelnen Regelteile werden ausgewertet und farblich in grün oder rot dargestellt. Sollte der Gesamtwert einer Regel feststehen, bevor

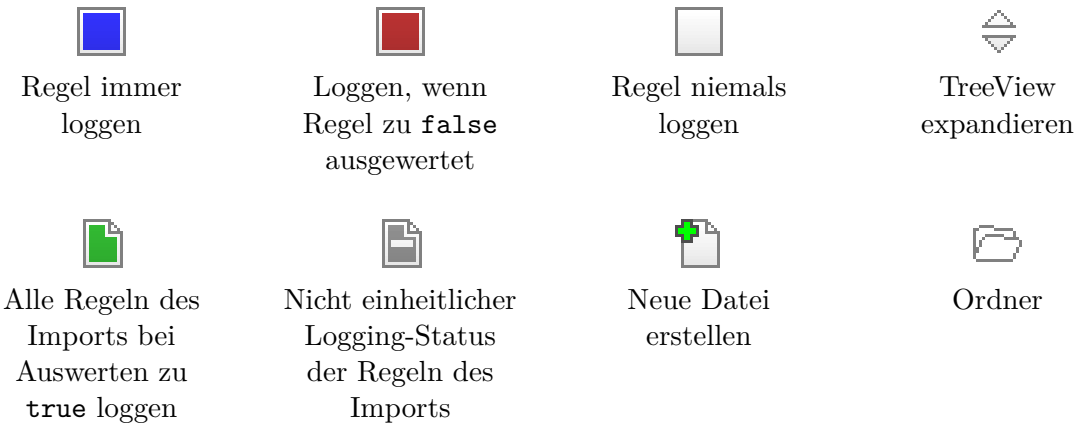


Abbildung 9: Einige in RUDIBUGGER genutzten Symbole

die ganze Regel überprüft wurde (zum Beispiel, weil es sich um eine Disjunktion handelt und das erste Disjunkt zu `true` auswertet), wird der Rest der Regel nicht mehr ausgewertet und deshalb grau dargestellt. Logische Konnektoren (`!`, `&&` und `||`) sowie Klammern werden schwarz dargestellt.

3.12 Sonstiges

Der Name RUDIBUGGER ist ein Portemanteau-Wort (Kofferwort). Hierbei wurde der Entwicklungsname von VONDA, RUDIMANT, mit dem Wort *Debugger* verschmolzen. Aufgrund der Endung „*-bugger*", ein Homophon zum deutschen Wort „Bagger", ist das Logo von RUDIBUGGER eine stilisierte Baggerschaufel.

Die verwendeten Bilder (Icons und Symbole) wurden entweder komplett selbst gestaltet oder basieren auf Symbolen von Alexey Egorov[6], welche übernommen oder angepasst wurden. Egorovs Symbole dürfen kostenfrei für kommerzielle Zwecke genutzt werden. In allen Fällen kam das Bildbearbeitungsprogramm GIMP zum Einsatz. Einige Symbole sind in Abbildung 9 zu sehen.

[6] `https://www.iconfinder.com/iconsets/uidesignicons`

4 Verwandte Dialogsysteme

An dieser Stelle werfen wir einen Blick auf andere Dialogsysteme, die Ähnlichkeiten zu VOnDA und seinem Debugger RUDIBUGGER aufweisen. Wir befassen uns unter anderem mit den eventuell vorhandenen Möglichkeiten, diese Systeme zu debuggen, um so Rückschlüsse auf den Ablauf eines gegebenen Dialogs zu ziehen.

4.1 NADIA

NADIA[7] (***Na****tural* ***Dia****logue System*) wurde von Markus M. Berg im Rahmen seiner Doktorarbeit geschrieben. Die Software soll das Designen von Dialogen ohne das Schreiben von Parsern, Dialog-Strategien oder komplexen Abläufen ermöglichen. Die Basis dafür ist ein XML-basiertes Dialogmodell (Berg, 2015), das jeweils einen Dialog mit verschiedenen Aufgaben definiert (Berg, 2014, S. 218).

Zum Erstellen der `.xml`-Dateien bietet der Autor eine GUI an, mit der man in einer Java-ähnlichen Syntax ein Modell erstellen kann. Beim Abspeichern wird dieses dann in eine äquivalente `.xml`-Datei umgewandelt, die dann von NADIA verwendet werden kann. Alternativ kann man das ganze System direkt als Java-Projekt aufsetzen (Berg, 2014, S. 217f.).

Das Umwandeln dieses Modells ist ähnlich zur Kompilierung einer `.rudi`-Datei in eine äquivalente Java-Datei durch VOnDA. Leider habe ich keinen Debugger für diese Software gefunden.

4.2 Dialogflow

DIALOGFLOW[8] der Firma Dialogflow, Tochterfirma von Google, ist vor allem dafür bekannt, Entwicklern die Möglichkeit zu geben, *Google Assistant* neue *Actions* zur Verfügung zu stellen. Hierzu baut man einen eigenen Agenten, der als Container für mögliche Anfragen (*Intent*), Weltwissen (*Entities*) sowie Antworten an den Nutzer dient.[9]

Beim Erstellen eigener Intents wird man von DIALOGFLOW unterstützt, indem

[7] `http://mmberg.net/nadia/index.php`
[8] `https://dialogflow.com/`
[9] `https://dialogflow.com/docs/getting-started/building-your-first-agent`

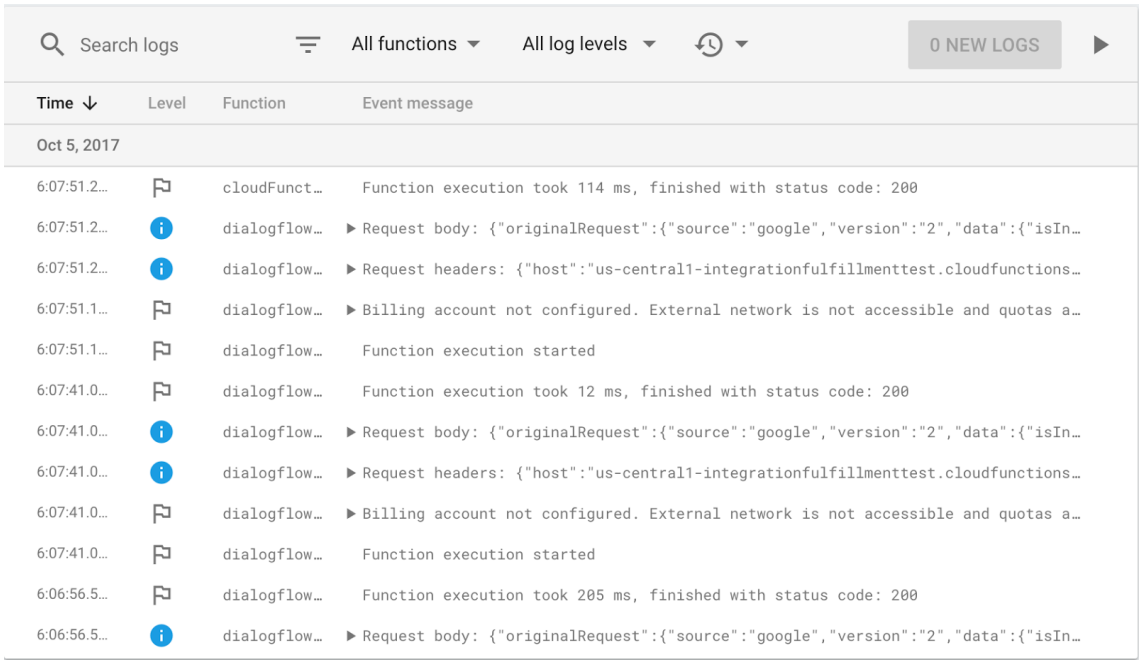


Abbildung 10: Logging Konsole von DIALOGFLOW, entnommen aus der Onlinedokumentation

es manche Elemente von möglichem Input direkt als *System Entity* erkennt. Basierend auf den angegeben Entities kann der Entwickler Antworten vorgeben oder diese als Parameter an eine API weiterreichen. Der Zugriff auf diese API wird in einer Javascript-Datei festgelegt, die direkt in der Entwicklungsoberfläche von DIALOGFLOW im Browser editiert werden kann.

Da Google Assistant darauf ausgerichtet ist, dem Nutzer für einen kurzen Zeitraum bei einer präzisen Anfrage zu helfen, sind die Dialoge nicht besonders lang. Zwar muss man zwischen

- linearen Dialogen, in denen eine Information nach der anderen gegeben wird, ohne Einfluss auf den restlichen Verlauf des Dialogs zu haben, und
- nicht-linearen Dialogen, in denen Antworten des Nutzers den restlichen Verlauf des Dialogs beeinflussen,

unterscheiden, aber die Anzahl möglicher auftretender Schwierigkeiten ist begrenzt.

DIALOGFLOW bietet außerdem ein Analyse-Tool, mit dem der Entwickler die Anzahl an Nutzungen (Sessions) sowie der verschiedenen Intents anzeigen kann. Debuggen kann man nur, indem man sich das Log anschaut, für dessen Betrachtung es ein integriertes Tool gibt (vgl. Abbildung 10). Diese Vorgehensweise weist also Ähnlichkeiten zu VONDA auf, wenn man sich das Log mit RUDIBUGGER anschaut, ohne das Logging innerhalb des Livesystems umzuschalten. Allerdings lässt sich das Log bei DIALOGFLOW nachträglich filtern; RUDIBUGGER verfügt (noch) nicht über eine solche Möglichkeit.

4.3 OpenDial

OPENDIAL wurde von der *Language Technology Group* der Universität Oslo unter der Federführung von Pierre Lison entwickelt (Lison & Kennington, 2015). Es handelt sich um ein quelloffenes Java-Toolkit zum Bauen und Evaluieren von Dialogsystemen. Der Dialogstatus wird, basierend auf der Information-State-Architektur, als bayessches Netz, das heißt als gerichteter azyklischer Graph, repräsentiert. Die verschiedenen Module nutzen diesen Status auch als geteiltes Gedächtnis. Durch die Möglichkeit der Anbindung von *Plugins* kann das System um Spracherkennung und -synthese erweitert werden.

Die internen Modelle der jeweiligen Domänen werden mit probabilistischen Regeln spezifiziert. Dafür wird das *XML*-Format genutzt, das den Autoren zufolge kompakt und gleichzeitig menschenlesbar (*human-readable*) ist. Die Regeln werden mit „if...then...else"-Konstruktionen als logische Formeln definiert, welche dann mit dem jeweiligen Status abgeglichen werden. Die Konsequenz beim Erfüllen des Antezedens ist dabei gewichtet (Lison & Kennington, 2016, S. 68f.). Da das Gewichten in den meisten Domänen im Vorfeld schwierig festzulegen ist, kann das Toolkit unbekannte Parameter mit bayesschem Lernen ermitteln.

Das Schreiben der internen Modelle weist Ähnlichkeiten zum Schreiben von Regeln für VONDA auf. Allerdings werden die Wahrscheinlichkeiten nicht beim Schreiben festgelegt, sondern später von der probabilistischen Komponente ermittelt.

Zum Testen, beziehungsweise Debuggen, stellen die Entwickler eine GUI zur Verfügung. Diese besteht aus drei *Views*:

- einem Chat-Fenster, das dem Nutzer die Eingabe neuer Äußerungen sowie

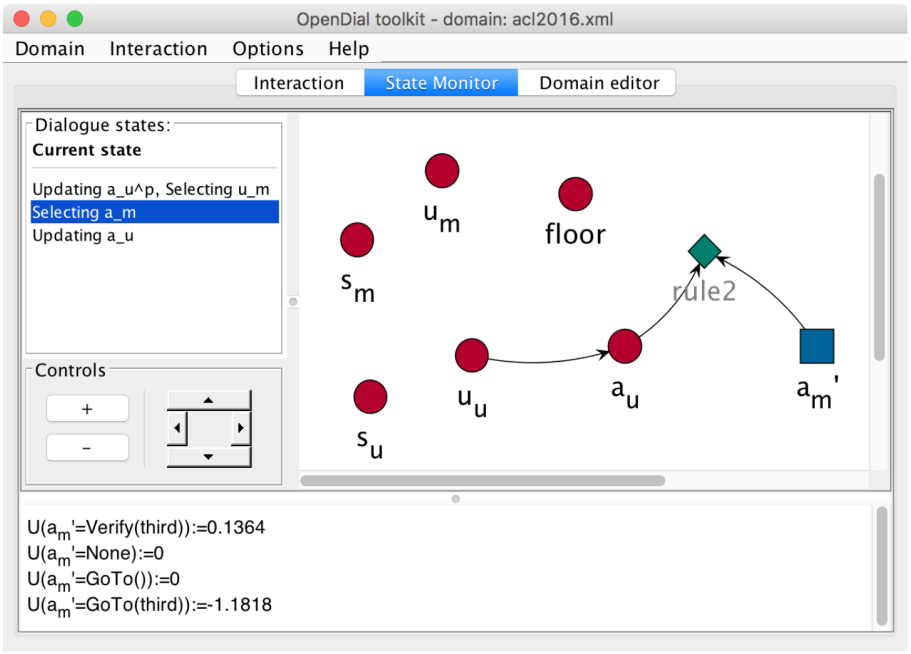


Abbildung 11: Dialogue state monitor (Lison & Kennington, 2016, S. 71)

einen Blick auf den bisherigen Verlauf bietet,

- einem *Dialogue Monitor*, in der der Nutzer das bayessche Netzwerk inspizieren, sich diverse Inferenzen berechnen lassen und vorherigen Zustand des Dialog Status ansehen kann, und
- einem Domänen-Editor, der sich zum Bearbeiten der `.xml`-Datei eignet.

Vor allem die zweite View (vgl. Abbildung 11) eignet sich besonders gut zum Debuggen (Lison & Kennington, 2016, S. 70). RUDIBUGGER weist Ähnlichkeiten zu dieser GUI auf, da es auch über eine gewisse Editor-Funktionalität verfügt, allerdings kann man sich im Moment noch nicht den vorherigen Status anzeigen lassen (siehe auch 4.4).

4.4 Visual SceneMaker

VISUAL SCENEMAKER[10] ist ein Tool zum Erstellen interaktiver Präsentationen und wurde am DFKI entwickelt. Beim Erstellen eigener Dialoge wird unterschieden

[10] `http://scenemaker.dfki.de/index.html`

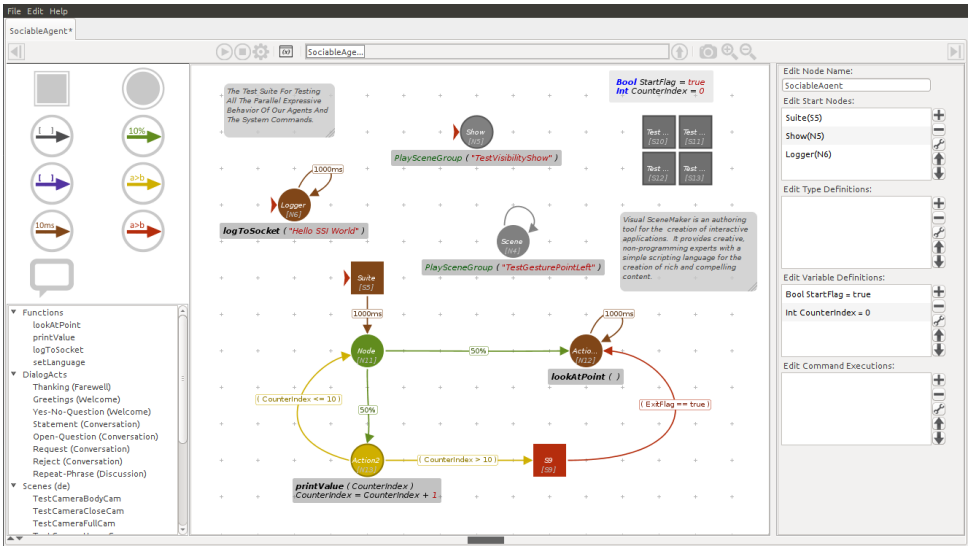


Abbildung 12: Visual SceneMaker Workspace, entnommen von der offiziellen Homepage

zwischen Inhalt (*Content*) und Logik (*Logic*). Zum Inhalt zählt, was das virtuelle Gegenüber äußert und wie es sich dabei ausdrückt. Organisiert wird der Inhalt in Form von *Scenes*, welche einem Filmskript mit Äußerungen und Bewegungen ähneln. Die Logik einer interaktiven Präsentation wird durch einen sogenannten *SceneFlow* organisiert. Dabei handelt es sich um ein Zustandsübergangsdiagramm (*Harel Statechart*).

Die Software verfügt über eine eigene grafische Oberfläche (vgl. Abbildung 12), die einem Entwickler ohne erweiterte Programmierkenntnisse das Erstellen eines eigenen SceneFlows ermöglichen soll.

Im Gegensatz zur textbasierten Vorgehensweise beim Definieren von `.rudi`-Regeln wird der Ablauf beim Visual SceneMaker also über ein grafisches Interface festgelegt. Die IDE bietet allerdings einen schnellen Zugriff auf definierte Konzepte (Elemente, Skript und SceneFlow) und ermöglicht so einen besseren Überblick auf das eigene Projekt.

Fazit

Persönliches Fazit

In dieser Arbeit habe ich dargelegt, wie RUDIBUGGER funktioniert und welchen Nutzen es erfüllt. Dazu habe ich die Dialogmanagementtechnologie VOnDA sowie die zugrundeliegende Theorie des Information-State-Approaches sowie der Belief-Desire-Intent-Agentenarchitekturen erklärt. Zum Schluss habe ich VOnDA beziehungsweise RUDIBUGGER mit anderen Dialogsystemen verglichen und einige Parallelen aufzeigen können.

Durch die Arbeit an RUDIBUGGER habe ich mir viel Wissen über das Entwickeln eigener GUIs sowie das Anwenden bestimmter Entwurfsmuster angeeignet. Außerdem habe ich, unter anderem, gelernt, wie man mit YAML Objekte serialisiert und wie man zwei Programme miteinander kommunizieren lässt. Der Vergleich zu anderen Dialogsystemen hat mir zudem weitere Ideen aufgezeigt, die ich in Zukunft in mein Programm integrieren möchte.

Am Anfang der Entwicklung habe ich eine bestimmte Funktionalität oft spontan und ohne vorherige Planung implementiert, so dass sie zwar funktioniert hat, aber der darunterliegende Code sehr komplex und somit auch schwer zu warten war. Einerseits lernt man nur, indem man Sachen ausprobiert. Andererseits hätte es sich auch oft gelohnt, vorher einen Plan aufzustellen und sich über eventuelle Patterns zu informieren.

Ausblick

RUDIBUGGER verfügt über die ursprünglich geplante Funktionalität. Es gibt allerdings einiges, was man noch hinzufügen kann oder überarbeiten sollte:

- Es wäre praktisch, wenn man auch loggen könnte, wie sich bestimmte Instanzen innerhalb der **Datenbank im Programmablauf** ändern. Die Funktionalität dafür ist zum Teil schon in einem anderen Projekt enthalten, nämlich dem Ontologieserver HFC; sie müsste also nur angepasst und in RUDIBUGGER integriert werden. Ich stelle mir hier eine ähnliche Funktion wie die einer *Watch* bei Netbeans vor, mit der man den Zustand eines Elements während des Programmablaufs ständig im Blick halten kann.

- Als Zusatzfunktionalität zur erwähnten Watch wäre eine Grafik nützlich, die einen **Zeitstrahl** repräsentiert, auf welchem die verschiedenen Änderungen angezeigt werden.
- Ähnlich wie im VISUAL SCENEMAKER wäre die Darstellung der Regeln in Form von **Graphen** auch eine Möglichkeit, für mehr Überblick zur sorgen.
- Wie in einer IDE wird man auch hier mit Sicherheit in der Situation sein, in der man mehrere `.rudi`-Dateien gleichzeitig anzeigen möchte. Es sollte also ermöglicht werden, **mehrere Tabs nebeneinander** anzuzeigen. Die nötige Basis für solch eine Funktionalität ist bereits in Teilen im Code vorhanden.
- Zwar unterstützt JavaFX **Hi-DPI**, dennoch bedarf es einiger Anpassungen. So reicht es nicht, die Schriftgröße pauschal zu erhöhen, da die Bedienelemente nicht automatisch „mitwachsen". Zudem müssen die jeweiligen Icons auch angepasst werden.
- Momentan kann das Kompilieren der `.rudi`-Dateien zwar über RUDIBUGGER angestoßen werden, allerdings öffnet sich dann ein Konsolenfenster, in welchem man den Output (d.h. **Fehlermeldungen und Warnungen**) von VONDA nur während der Kompilierung sieht. Es gilt also eine Möglichkeit zu implementieren, welche Fehler geordnet anzeigt, das Springen zu den relevanten Stellen im Code ermöglicht und dies grafisch auch im Editor hervorhebt. In den Klassen, welche die Importe und Regeln repräsentieren, befindet sich bereits ein Feld, in dem die Meldungen sowie deren Zeile gespeichert werden. Diese Informationen müssen also nur noch sinnvoll verarbeitet werden.
- Da sich durch diese zusätzliche Funktionalität mehr Daten ansammeln werden, ist es notwendig, über effiziente Möglichkeiten zu verfügen, um diese zu filtern. Hier wäre das **Filtern** nach Moment des Auftretens oder nach Typ (Regel, Watch oder Ähnliches) sinnvoll.
- Prinzipiell ist der bisherige Logging-Output etwas unübersichtlich. Zumindest der Regelname sollte visuell vom Regelinhalt selbst abgetrennt sein.
- Eine **Suchfunktion**, die Volltextsuche und das Suchen nach bestimmten Regeln ermöglicht, wäre nützlich.

- Eine weitere Funktionalität, die schon zum Teil implementiert ist, ist das Speichern des Logging-Status, das heißt, wie welche Regeln geloggt werden sollen. Beim Kompilieren wird dieser Status derzeit intern zwischengespeichert und dann auf die neue Regelstruktur angewandt. Nützlich wäre es, den Status speichern zu können, um so **mehrere Konfigurationen** zu haben, damit man bestimmte Funktionen testen kann. Hier müsste ich also noch dafür sorgen, dass der Zustand der Regelstruktur mit YAML serialisierbar ist.

- Man sollte das Rad nicht neu erfinden. Momentan ist die **Verbindung** zwischen VOnDA und RUDIBUGGER sehr fragil und fehleranfällig, da wir diese Verbindung von Grund auf neu implementiert haben. Man könnte so vorgehen, dass man alle möglichen auftretenden Verbindungsprobleme abdeckt, oder man sucht sich eine weitere Bibliothek, welche das Verbinden von zwei Anwendungen bereits perfektioniert hat.

RUDIBUGGER wird aktiv genutzt. So wird es zwangsläufig auch zu Situationen kommen, in denen sich der Nutzer ein bestimmtes Feature wünscht, an das ich selbst noch nicht gedacht habe. Das System muss also auch in Zukunft weiterentwickelt werden.

Anhang

A. Repository

RUDIBUGGER ist unter folgendem Link zu finden:

`https://github.com/yoshegg/rudibugger`

VONDA ist allerdings vonnöten, um RUDIBUGGER zu kompilieren. VONDA kann hier heruntergeladen werden:

`https://github.com/bkiefer/vonda`

B. Aufbau von rudibugger

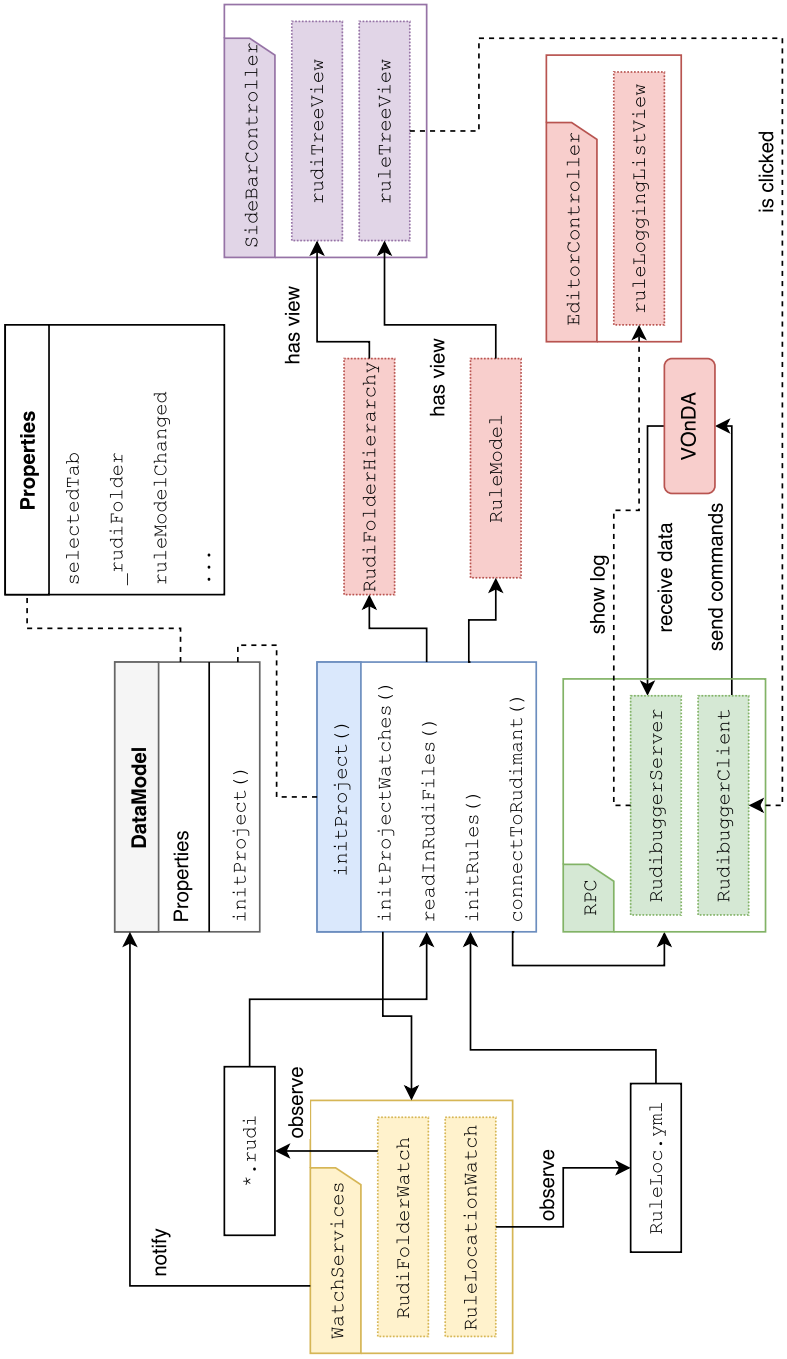


Abbildung 13: Schematische Darstellung der Beziehungen der wichtigsten Klassen und Funktionen von RUDIBUGGER

Quellenangaben

Literaturverzeichnis

Berg, M. M. (2014). *Modelling of Natural Dialogues in the Context of Speech-based Information and Control Systems* (Dissertation, University of Kiel). Zugriff auf `http://nbn-resolving.de/urn:nbn:de:gbv:8-diss-167394`

Berg, M. M. (2015). NADIA: A Simplified Approach Towards the Development of Natural Dialogue Systems. In C. Biemann, S. Handschuh, A. Freitas, F. Meziane & E. Métais (Hrsg.), *Natural language processing and information systems* (Bd. 9103, S. 144-150). Springer International Publishing. Zugriff auf `http://dx.doi.org/10.1007/978-3-319-19581-0_12`

Bratman, M. (1987). *Intention, plans, and practical reason.* Center for the Study of Language and Information.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. M. (1994). *Design patterns: Elements of reusable object-oriented software* (1. Aufl.). Addison-Wesley Professional.

Lison, P. & Kennington, C. (2015). Developing spoken dialogue systems with the opendial toolkit. In *Proceedings of the 19th workshop on the semantics and pragmatics of dialogue.*

Lison, P. & Kennington, C. (2016). OpenDial: A toolkit for developing spoken dialogue systems with probabilistic rules. In *Proceedings of the 54th annual meeting of the association for computational linguistics (demonstrations)* (S. 67–72). Berlin, Germany: Association for Computational Linguistics.

Traum, D. R. & Larsson, S. (2003). The information state approach to dialogue management. In J. van Kuppevelt & R. W. Smith (Hrsg.), *Current and new directions in discourse and dialogue* (S. 325–353). Dordrecht: Springer Netherlands. doi: 10.1007/978-94-010-0019-2_15

Sonstige Quellenangaben

Aus Gründen der besseren Übersichtlichkeit wurden alle Internetquellen direkt als Fußnote in der Arbeit erwähnt. Der Zugriff auf diese Quellen wurde am 15. Dezember 2017 geprüft.

Bachelorarbeit zur Erlangung des akademischen Grades

Bachelor of Science

im Fach Computerlinguistik

der Philosophischen Fakultät
der Universität des Saarlandes

Wissensbasiertes Dialogmanagement für Social Talk

Erstgutachter: Prof. Dr. Josef van Genabith
Zweitgutachter: Dipl.-Inf. Bernd Kiefer

Anna Welker
Robert-Koch-Straße 9
66287 Quierschied
s9aawelk@stud.uni-saarland.de
2547401

Selbständigkeitserklärung:

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Ferner habe ich die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, unter Angabe der jeweiligen Quelle als Entlehnung kenntlich gemacht. Dies gilt ebenso für Zeichnungen, Skizzen und Pläne sowie bildliche und grafische Darstellungen, die nicht eigenhändig von mir angefertigt wurden.

Saarbrücken, den _____ Unterschrift: _____

Zusammenfassung

Roboter und virtuelle Agenten werden immer häufiger als Gefährten oder Trainer im alltäglichen Leben von Menschen erprobt. Studien zeigen, dass der Erfolg solcher Anwendungen oft davon abhängt, eine persönliche Beziehung zwischen Agent und Mensch aufzubauen. Hierfür ist Small-Talk oder Off-Activity-Talk ein wichtiges Element. Eine wesentliche Rolle für den erfolgreichen Aufbau einer Beziehung sind Rückbezüge auf vorherige Interaktionen. Um dies zu implementieren, benötigt der Agent ein Langzeitgedächtnis, das neben dem semantischen Gedächtnis auch ein episodisches besitzt. Diese Arbeit stellt ein regelbasiertes Dialogsystem vor, das als Small-Talk-Modul zu solchen Agenten hinzugefügt werden kann. Es kann nicht nur Dialog betreiben, der von der Domäne unabhängig ist, sondern sich auch an vergangene Gespräche mit dem Benutzer erinnern und diese Information nutzen, um auf lange Zeit kohärente Unterhaltungen zu führen. Das implementierte Projekt zeigt, dass auch regelbasiertes Dialogmanagement dieser Aufgabe gerecht werden kann, sofern eine Möglichkeit existiert, Reaktionen für nicht abgedeckten Input zu improvisieren.

Inhaltsverzeichnis

1. Einführung

Das PAL-Projekt (**P**ersonal **A**ssistant for a healthy **L**ifestyle) hat zum Ziel, Kinder mit Diabetes Typ 1 darin zu unterstützen, Selbstmanagement und Gewissenhaftigkeit im Umgang mit ihrer Krankheit zu erlernen (PAL, 2017). Diabetes erfordert von den Kindern im Alter zwischen sieben und 14 Jahren regelmäßige Kontrollen ihres Blutzuckers und strenge Planung jeder einzelnen Mahlzeit. Diese gerade für Kinder nicht einfache intensive Kontrolle im Alltag unterstützt PAL mit einem System, das versucht, spielerisch zu regelmäßigen Einträgen von Blutzuckerspiegel und Mahlzeiten zu motivieren und auf Gefahren aufmerksam zu machen. Kern ist dabei eine Tablet-Anwendung, die als Tagebuch fungiert und in der die Kinder virtuell dem Roboter begegnen können, den sie im Krankenhaus oder in Trainingscamps kennengelernt haben.
Ein großes Problem von Versuchen, Roboter und virtuelle Agenten zu nutzen, um Kinder zum Lernen zu animieren, ist, dass das Interesse für die Anwendung abnimmt sobald der „Reiz des Neuen“ verflogen ist.
Diesem Phänomen begegneten schon Kanda et al. (2004) in ihrer Studie. Sie konnten zeigen, dass ein Roboter japanische Schulkinder beim Lernen von Englisch unterstützen konnte, sofern es ihm gelang, eine persönliche Beziehung mit ihnen aufzubauen und sie so auch noch in der zweiten Woche des Experiments zu Gesprächen zu animieren. Während die Kinder in der ersten Woche des Experiments durchschnittlich etwa zwölf Minuten mit dem Roboter verbrachten, waren es in der zweiten Woche nur noch unter drei Minuten pro Kind. Die Vermutung liegt nahe, dass die beschränkten Interaktionsmöglichkeiten des Systems, das laut den Autoren „mehr als 300 Sätze äußern und etwa 50 Wörter verstehen“ konnte, das Interesse zu schnell erlahmen ließen. Solche Ergebnisse ließen bereits darauf schließen, dass für einen Lernroboter, der über lange Zeit regelmäßig mit Kindern interagieren soll, der Aufbau einer persönlichen Beziehung zur Erweckung langfristigen Interesses von großer Bedeutung ist.
Dies bestätigten Kruijff-Korbayová et al. (2014a) in einem Wizard-Of-Oz-Experiment zu ALIZ-E, dem Vorgängerprojekt, das von PAL weitergeführt wird (PAL, 2017). Hier wurde bei einer Testgruppe von Kindern eine Off-Activity-Talk-Funktion zum System hinzugefügt, mit dem die Kinder die Möglichkeit hatten, Aktivitäten wie ein Quiz-Spiel kurzzeitig zu unterbrechen und sich frei mit dem System zu unterhalten. Während bei der anschließenden Evaluation alle Kinder angaben, wieder mit dem Roboter spielen zu wollen, reservierte nur die Hälfte von ihnen einen weiteren Slot während der Zeit im Trainingscamp. Von diesen waren neun von elf Interessenten zuvor in der Off-Activity-Talk-Gruppe. Auch wenn die Evaluation des Experiments keine großen Verbesserungen der Wahrnehmung zeigte - sowohl mit als auch ohne Off-Activity-Talk wurde die Inter-

aktion sehr positiv bewertet - lassen die Antworten auf die auf emotionale Informationen abzielenden Fragen darauf schließen, dass die freien Gespräche einen positiven Einfluss auf die Wahrnehmung des Roboters hatten. Dass die Möglichkeit, abseits der mit dem Roboter durchführbaren Aktivitäten mit ihm über beliebige Themen zu reden, von den Testpersonen sehr gut aufgenommen wurde, zeigt eine genauere Analyse der Sitzungen, in denen der Off-Activity-Talk zur Verfügung stand (Kruijff-Korbayová et al., 2014b). Die Kinder sprachen sehr gut auf etwa das Quiz unterbrechende Fragen zu ihrer eigenen Meinung an, aus denen sich kleine Dialoge entwickeln konnten. Etwa ein Fünftel der Subdialoge wurde zudem von Kindern selbst initiiert, was zeigt, dass sie bereit waren sich auf den Roboter als Gesprächspartner einzulassen und die Subdialoge nicht nur als eine nervige Zwischenerscheinung betrachteten.

Diese Bachelorarbeit beschreibt Smoto, ein System, das solchen Off-Activity-Talk betreibt und als Modul in das PAL-System aufgenommen werden kann. Es beschränkt sich dabei nicht wie moderne Chatbots auf die bloße Generierung von Sätzen, die möglichst gut zur letzten Äußerung des Dialogpartners passen. Zusätzlich zu dieser selbstverständlich nicht unwichtigen Fähigkeit verlässt sich Smoto auf ein episodisches Gedächtnis, in dem Informationen zu früheren Dialogen mit dem Gesprächspartner gespeichert sind. So erinnert es sich etwa an Aussagen des Kindes über Präferenzen oder Aktivitäten. Es kann hiermit einerseits verhindern, repetitiv zu sein und etwa Fragen zu stellen, deren Antworten bereits in früheren Treffen gegeben wurden. Gleichzeitig können diese Informationen jedoch auch genutzt werden, um selbst den aktuellen Dialog voranzubringen. Smoto ist daher kein System, das lediglich auf Äußerungen des Gegenübers reagiert und hin und wieder einen einem festgelegten Repertoire entstammenden Satz von sich gibt, der den Eindruck eines lebendigen Dialoges vermitteln soll. Vielmehr kann es die Informationen aus dem episodischen Gedächtnis aktiv nutzen, um sinnvollen Dialog zu generieren. Hat ein Kind etwa von einem Vorhaben erzählt, das es mittlerweile durchgeführt haben sollte, kann das System nach dessen Ausgang fragen. Mithilfe von WordNet als semantischem Gedächtnis können außerdem die geäußerten Präferenzen genau analysiert werden und dem System die Möglichkeit eröffnen, nach der Meinung des Kindes zu semantisch ähnlichen Themen zu fragen. Eine zum Zeitpunkt dieser Arbeit noch nicht realisierte, aber denkbare Erweiterung wäre, die Präferenzen und Einstellung des Agenten selbst an die des Kindes anzupassen, sodass sich das Kind noch mehr mit dem System identifizieren kann.

2. Related Work

2.1. Chatbots und Dialogsysteme

Aktuelle Ansätze, Small Talk zu betreiben, also einen nicht auf eine bestimmte Domäne beschränkten Dialog zu führen, sind oft End-To-End-Systeme mit Neuralen Netzen oder Deep Learning. Wohl am bekanntesten sind die Chatbots, die auf Twitter, Facebook und anderen Internetseiten zu finden sind und in vielen Fällen Posts generieren können, denen zuzutrauen wäre, von Menschen geschrieben worden zu sein.
Chatbots oder Chatterbots sind Computerprogramme, die dazu entworfen sind, mittels Text- bzw. Audio-Ein- und Ausgabe mit Menschen zu kommunizieren und dabei zu reagieren, als seien sie selbst Menschen (Wikipedia, 2017a). Die Unterschiede zwischen Chatbots und Dialogsystemen sind oft nur vage definiert und in Randfällen schwer zu benennen. McTear (2004) nennt etwa als Unterschiede, dass Dialogsysteme oft mehr theoretisch motivierte Techniken benutzen und dabei aber nur für eine bestimmte Domäne entwickelt werden. Klüwer (2011) macht dagegen darauf aufmerksam, dass der zweite genannte Punkt hauptsächlich durch den ersten bedingt ist und beschreibt verschiedene Dialogsysteme und Chatbots, um diesen zu unterstützen. Die theoretisch motivierten Techniken sollen auch in dieser Arbeit als Hauptcharakteristik von Dialogsystemen betrachtet werden.
Viele Chatbots haben eine Schwachstelle darin, dass sie nicht versuchen, eine Äußerung des Benutzers semantisch zu verstehen. Sie generieren lediglich eine Reaktion, deren Wahrscheinlichkeit, angemessen zu sein, möglichst hoch ist. Dies bietet vor allem dort Anfälligkeiten, wo die Systeme versuchen, sich mittels der erfahrenen Unterhaltungen selbst zu verbessern.
Aufregung verurachte im März 2016 der von Microsoft auf Twitter gestartete Chatbot Tay, der zur Unterhaltung von Teenagern gedacht war. Die künstliche Intelligenz Tay war darauf programmiert, an der Interaktion mit anderen Twitter-Nutzern zu lernen und sich konstant selbst zu verbessern. Sie musste nach kürzester Zeit wieder vom Netz genommen werden, weil es einer Untergruppe von Twitter-Nutzern gelungen war, eine Schwachstelle darin zu finden und Tay durch gezielte Tweets zu lehren, höchst rassistische und verletzende Tweets und Bilder zu posten (Lee, 2016).
Ein Beispiel für auf Machine Learning basierende Chatbots ist MILABOT (Serban et al., 2017), ein für die Amazon Alexa Prize Competition (Amazon Alexa, 2017) programmiertes System. In ihm sind 22 voneinander unabhängige Module - meist wiederum Chatbotsysteme - verbaut, die ausgehend vom eingehenden Dialogakt und der Dialoghistorie zahlreiche Antwortkandidaten generieren. Hierzu zählen unter anderem ein *Alicebot* (ALICE A.I. Foundation, 2016), der mithilfe von AIML Templates eine Äußerung basie-

rend auf Dialoghistorie und aktuellem Dialogakt generiert, ein *Elizabot*, der wie ELIZA (Weizenbaum, 1966) mit Templates und dem letzten Dialogakt arbeitet, und ein *Initiatorbot*, der eine aus einem vordefinierten Set zufällig gewählte Frage zu einem neuen Thema stellt. Daneben finden sich Bots, die versuchen, eine relevante Antwort mithilfe von Amazons Question-Answering-Webservices zu generieren oder die auf eine bestimmte Domäne, etwa Filme, spezialisiert sind. Hinzu kommen Retrieval-, Suchmaschinen- und Generierungs-basierte Neuronale Netze. Aus dem Pool möglicher Antworten, den diese Modelle erzeugen, wird am Ende nach einem auf 1458 Features basierenden Bewertungsmodell eine gewählt, die an das TTS geschickt wird. Das Live-System von MILABOT wird mit mehr als 32 Tesla K80 GPUs betrieben.
Ansätze wie der beschriebene können bereits beachtliche Leistungen vollbringen. Moderne Systeme kombinieren oft verschiedene Modelle, um aus den von ihnen gelieferten Vorschlägen die optimale Antwort zu wählen. Dabei sind die Antworten und die Reaktionen der Systeme, die auf annotierten Dialogen trainiert werden - handele es sich dabei nun um neuronale Netze oder Machine Learning - letztendlich auf aus dem Trainingskorpus extrahierten Wahrscheinlichkeiten basiert. Ihre Qualität hängt damit ab von der Qualität der unterliegenden Daten und deren Menge. Zudem hat jeder Korpus, der als Basis genommen wird, naturgemäß eine oder mehrere Domänen, während andere Themen vielleicht vernachlässigt werden - das System wird also für diese Themenbereiche naturgemäß schlechter funktionieren als für die vorkommenden, weil es keine Verhaltensweisen dafür lernen konnte. Zwar können diese Systeme unter Aufbringung von mehr Daten oft bessere Ergebnisse erzeugen, benötigen dafür jedoch auch immer mehr Speicherplatz, RAM und CPU bzw. GPU. Sie sind somit für kleine Anwendungen, die mit geringen Ressourcen arbeiten und sich zudem möglicherweise nicht immer auf eine Internetverbindung verlassen können, schwer nutzbar.
Verglichen hiermit sind Dialogsysteme, die theoretisch fundiert sind, oft ressourcenschonender. So stellte Klüwer (2015) ein Software-Toolkit vor, das in andere Systeme integriert werden kann, um sie zu befähigen, Small Talk zu betreiben. Genutzt werden hierbei eigens entwickelte *soziale Dialogakte*, basierend auf der Face-Theorie von Erving Goffman (Goffman, 1967). Die Dialogakte unterteilen sich in zwei Gruppen, *request-face acts* und *support-face acts*. Erstere beziehen sich auf die Aufforderung, dass das Gegenüber das eigene *face* unterstützt, während letztere das *face* des Gegenübers stützen. Hierbei bezeichnet *face* das Bild, mit dem eine Person von den Gesprächspartnern wahrgenommen wird. Basierend auf diesen Dialogakten entscheidet das System durch einen endlichen Automaten, welche Antwort oder Reaktion den Äußerungen des Benutzers angemessen ist. Es besitzt hierzu auch ein Dialoggedächtnis, in dem alle während des Gesprächs ausgetauschten Dialogakte gespeichert werden, und vermeidet so Redundanz in der Un-

terhaltung. Diese Informationen werden jedoch nicht über die Dauer einer Interaktion hinaus gespeichert. Das System wurde in zwei Anwendungen in Twinity eingesetzt, wo es einen Barkeeper und einen Möbelverkäufer simulierte.

2.2. Episodic-Memory-Dialogsysteme

> „Episodic memory is a neurocognitive (brain/mind) system, uniquely different from other memory systems, that enables human beings to remember past experiences. [...] I also suggest that episodic memory is a true, even if as yet generally unappreciated, marvel of nature.“ (Tulving, 2002)

Das Langzeitgedächtnis des Menschen unterscheidet sich in das - für Dialogsysteme wenig relevante - prozedurale Gedächtnis und das deklarative Gedächtnis. Letzteres besitzt wiederum zwei Teile, nämlich das semantische Gedächtnis, das das Weltwissen enthält, und das episodische Gedächtnis (engl. Episodic Memory), das vergangene Ereignisse erinnert (Wikipedia, 2017b).
Die existierenden Ansätze, Dialogsystemen ein episodisches Gedächtnis für Interaktionen mit einem Benutzer zu verleihen, sind sehr unterschiedlich. Chatbots besitzen für gewöhnlich kein solches Gedächtnis, da sie selten in Bereichen eingesetzt werden, wo es erforderlich oder auch nur sinnvoll ist, sich über ein Gespräch hinaus an einen Benutzer anzupassen. Auch bei Dialogsystemen kann dies der Fall sein. In Fällen wie den in Klüwer (2015) beschriebenen Anwendungen ist leicht vorstellbar, dass eine solche Speicherung von Dialoginformationen sehr schnell zu einer gewaltigen Datenmenge - und damit zu Speicher- oder Laufzeitproblemen - führen kann, obwohl viele Benutzer möglicherweise nie zurückkehren und die zu ihnen gespeicherte Information somit nie wieder genutzt werden wird. Vor dieser Gefahr warnen auch Castellano et al. (2008), weisen jedoch auf die Wichtigkeit eines dynamischen Gedächtnisses und der Fähigkeit zu lernen für Systeme hin, die eine Langzeitbeziehung mit dem Benutzer aufbauen sollen.
Ein Beispiel für ein System, welches solche Langzeitinteraktionen mit Menschen anstrebt, sich dazu jedoch nur an sehr wenige Informationen bezüglich des letzten Treffens mit dem Benutzer erinnern muss, ist der Lernroboter Eva (Kasap and Magnenat-Thalmann, 2010). Eva fragt den Benutzer über Konzepte von Computernetzwerken ab; sie besitzt einen Fragenkatalog, der in beliebig vielen Sitzungen abgearbeitet werden kann. Für die Laufzeit jeder dieser Sitzungen existiert ein Kurzzeitgedächtnis, in dem alle durch Sprach- oder Gesichtserkennung gefundenen Ereignisse der aktuellen Interaktion gespeichert werden. Im Langzeitgedächtnis, das Erinnerungen über eine Interaktion hinaus speichert, werden jedoch am Ende der Sitzung nur wenige dieser Informationen abgelegt - nämlich die Eckdaten solcher Ereignisse, die den emotionalen Zustand des Systems

ändern oder zur Erreichung oder Verfehlung eines Ziels beitragen. Zu diesen Eckdaten gehören die Zeit der Interaktion, der Name des Benutzers, verschiedene Informationen über den Zustand des zielbasierten Planners vor und nach dem Ereignis und Status und Intensität der Emotionen des Systems. Diese Informationen werden bei der nächsten Interaktion abgerufen, um zum Beispiel den Lernstand sowie die Erfolgsrate eines Benutzers zu aggregieren, oder um die aktuelle emotionale Beziehung zwischen Nutzer und System zu berechnen und die Kommunikation darauf anzupassen.
Wichtig ist auch die Frage, wie ein episodisches Gedächtnis genutzt werden kann, d.h., auf welche Weise Informationen daraus abgerufen werden. Lim et al. (2011) schlagen für einen Social Companion ein Retrieval-System vor, das die Theorien von Spreading Activation und Compound Cue vereint. Spreading Activation macht es dabei möglich, auf mit der aktuellen Situation bzw. mit dem aktuellen Ereignis zusammenhängende Situationen oder Ereignisse zuzugreifen, während Compound Cue aus dem Episodic Memory solche heraussucht, die dem aktuellen Ereignis bzw. der aktuellen Situation ähneln. Das System nutzt somit eine Methode, die dem menschlichen Erinnern ähnelt - einmal in der Form der Assoziation mit möglichen vorangegangenen oder folgenden Ereignissen, und einmal in der Form der Assoziation mit in ihrer Struktur ähnlichen Ereignissen, die bereits erlebt wurden. Grundlage für das Retrieval ist dabei ein in Langzeit- und Kurzzeitgedächtnis aufgeteiltes episodisches Gedächtnis, wobei Ereignisse wiederum nur dann ins Langzeitgedächtnis übergehen, wenn sie einen emotionalen Einfluss hatten oder mit dem Erreichen oder Verfehlen eines Ziels zusammenhingen. Ereignisse sind im Gedächtnis als Knoten dargestellt, die über ihre Attribute miteinander verbunden sind. Die Attribute eines Knotens selbst sind ebenfalls miteinander verbunden. Mithilfe der beiden Retrieval-Modelle gelingt es dem System so, zu „improvisieren“: Es kann mit neuen, unbekannten Ereignissen umgehen, indem es sein Wissen über vergangene, ähnliche Ereignisse nutzt, oder etwa durch andere Interaktion mit dem Benutzer eine vorangegangene negative Folge eines Ereignisses zu vermeiden versuchen.
Ein Beispiel für die Anwendung von episodischem Gedächtnis in PAL zeigte Goedheijt (2017) als Teil eines Projektes mit dem Ziel, personalisierte Interaktionen mit den Kindern zu entwickeln. Hierzu werden Episoden erstellt, die den Fortschritt oder Schwierigkeiten des Kindes bezüglich der Lernziele, die Eintragungen des Blutzuckers oder aus Mitteilungen des Kindes extrahierte Ereignisse repräsentieren können. Diese Episoden sind in der Ontologie mit auf dem 5W1H-Prinzip (When, Where, Who, Why, What und How) basierenden Relationen verlinkt. So sind sie z.B. über „Who“ mit dem Kind verbunden und über „How“ mit einem zugehörigen Ereignis wie *cooking* oder *playing*. Das „What“ wiederum, das das „Why“ impliziert, wird durch *EpisodeTags* dargestellt. Durch diese sind die Episoden in Klassen eingeteilt, die wiederum getriggert werden können,

um ein anzusprechendes Thema zu finden.
Das entsprechend des Projektes modifizierte PAL-System nutzt dies, um Kommentare und Nachfragen zu schwierigen oder noch ausstehenden Lernzielen sowie den Eintragungen bezüglich des Blutzuckers zu stellen. Des Weiteren werden Ereignisse aus Mitteilungen des Kindes extrahiert und als Episoden gespeichert. Ist das Ereignis vorbei, kann sich der Agent darauf beziehen. Außerdem kann er spezifische, vorherbestimmte Fragen zu bestimmten Wortgruppen, wie etwa Orten, stellen.

3. Das Dialogsystem Smoto

Im Gegensatz zu den angeführten Beispielen von Chatbots sind der Kern des in dieser Arbeit beschriebenen Projektes Smoto handgeschriebene Dialogregeln, die aus einer im Fachbereich MLT (Multimodal Language Technologies) des DFKI (Deutsches Forschungszentrum für künstliche Intelligenz) entwickelten Regelsprache nach Java übersetzt werden und eng mit dem im selben Fachbereich entwickelten Ontologieserver HFC verbunden sind.

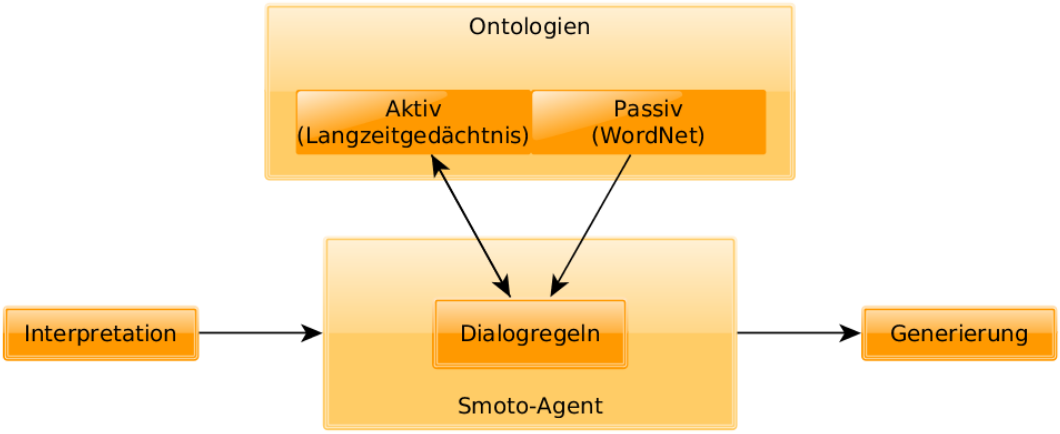


Abbildung 1: Systemkomponenten von Smoto

Die bewusste Wahl eines regelbasierten Systems hat mehrere Ursachen. Zum einen ist dies ein Versuch zu sehen, ob der angesprochene Datenbedarf für gelernte Systeme mit Dialogregeln und einer WordNet-Ontologie als Weltwissen umgangen werden kann, ohne dass die Antworten des Systems allzu sehr wie Patterns wirken. Zum anderen ermöglicht es die absolute Kontrolle der Regeln, nicht nur jederzeit Informationen in das episodische Gedächtnis einzutragen, sondern diese auch zu den bereits benannten Gelegenheiten zu nutzen. Der Rückgriff auf frühere Unterhaltungen soll dem Gesprächspartner das Gefühl geben, dass dem Agenten diese Unterhaltungen wichtig waren und den Wunsch wecken, immer wieder neue Gespräche mit ihm zu führen. Menschen bevorzugen Unterhaltungspartner, die sich daran erinnern, wenn ihnen wichtige persönliche Informationen mitgeteilt wurden. Auf diese einzugehen bzw. Rücksicht auf sie zu nehmen gehört zu den wichtigsten Eigenschaften eines angenehmen Gesprächspartners.
Dem aufmerksamen Leser wird auffallen, dass Smoto auf Äußerungen des Benutzers sehr oft mit Fragen reagiert. Dies hat mehrere Gründe. Zum einen möchte das System das Gegenüber „kennenlernen“, es animieren, für weitere Gespräche nutzbare Informationen über sich preiszugeben. Zum anderen versucht es, möglichst viel der Redezeit im Ge-

spräch an den Benutzer zu übergeben. Je mehr des Gesprächs der Benutzer übernimmt, desto geringer ist die Wahrscheinlichkeit, dass das System Fehler begeht oder sich wiederholt. Zuletzt muss angemerkt werden, dass auch die Speicherung der in der Äußerung des Benutzers gegebenen Informationen leichter ist, wenn es sich um eine Antwort auf eine Frage handelt, die dem System bekannt ist.

3.1. Interpretation und Generierung

Interpretation und Generierung sind zwei Teilmodule, die zwar für ein Dialogsystem unverzichtbar sind, mit dem Dialogmanagement selbst aber wenig zu tun haben. Da in dieser Arbeit der Aufbau der Dialogregeln und die Verwendung der Ontologie im Vordergrund stehen, wird es dem Smoto verwendenden Projekt überlassen, diesem eine elaborierte Interpretation und Generierung zur Verfügung zu stellen.
In der vorliegenden Implementierung von Smoto ist der Input von einer beschränkten Anzahl natürlichsprachlicher Sätze möglich, für die der Output der Interpretation hardgecoded wurde. Dem Benutzer wird die Antwort des Systems ebenfalls in natürlicher Sprache angezeigt, wofür eine rudimentäre CPlan-Grammatik sorgt (Kiefer, 2017).
Der Input von Smoto entspricht mit wenigen, aber systematisch anpassbaren Unterschieden in den Benennungen der Slots dem Output eines einfachen Dependenzparsers, der auf dem WordNet-annotierten Brown-Korpus (Mihalcea, 1998) trainiert wurde, gefolgt von einer Klassifikation des Dialogaktes, den der eingehende Satz darstellt. In Abbildung 4 ist dargestellt, auf welche Dialogakte Smoto aktuell reagieren kann. Die verwendete Typhierarchie der Dialogakte entspricht der des Projektes PAL, d.h., die Einordnung von Äußerungen orientiert sich stark am Schema der DIT++ Taxonomie (Bunt, 2009). Die Bezeichnungen mancher Dialogakte wurden dabei leicht angepasst; so entspricht etwa eine *WHQuestion* von PAL - bzw. Smoto - der *SetQuestion* von DIT++ und die *YNQuestion* der *PropositionalQuestion*.
Das Subjekt eines Satzes ist in dieser Betrachtung der Agent eines Dialogaktes, das Objekt der Patient, und das Verb die Proposition. Befindet sich ein Hilfsverb im Satz, wie etwa *want* oder *like*, wird dieses zur Proposition und das Vollverb in dem Slot *activity* abgelegt. Ergänzungen werden in den Slots *modTemp*, *modLoc* und *modMod* gespeichert. Die modale Ergänzung wird in der vorliegenden Implementierung des Systems immer auf das Objekt des Dialogaktes bezogen, falls dieses vorhanden ist.
Da es das Ziel des Systems ist, Vertrauen und persönliche Beziehungen aufzubauen, scheint es auch wichtig, Agent oder Patient begleitende Possessivpronomen korrekt zu erkennen und der Verarbeitung zu überführen. So kann Smoto mittels der Slots *ag_poss* und *pat_poss* registrieren, wenn der Benutzer über sich selbst oder das System spricht. Stellt der Benutzer eine Frage, wird das Fragewort im Slot *kind* festgehalten, sodass

Smoto später die richtige Art zu antworten wählen kann. Zusätzlich zu den in Abbildung 4 gezeigten Slots enthält jeder Dialogakt den Slot *tense*, in dem die Zeitform des konjugierten Verbs gespeichert ist.

Die von Smoto generierten Dialogakte sind aktuell noch genauso einfach gehaltene Bedeutungsrepräsentationen wie die eingehenden Dialogakte, können jedoch durch einen leichten Eingriff in die Methode, die sie an die Generierung sendet, an eine - etwa zu Zwecken der Morphologie - anspruchsvollere Generierung angepasst werden. Die CPlan-Grammatik kombiniert die in den Dialogakten übergebenen Slots so, dass der Output für den Menschen leicht verständlich ist.

3.2. Ontologien - Das Episodische Gedächtnis

Die Verwaltung der Ontologie erfolgt über HFC, einem Forward Chainer ähnlich Jena oder OWLIM (Krieger, 2013). Im Unterschied zum normalen RDF-OWL, das mit Tripeln arbeitet, kann HFC Quadrupel verarbeiten. Das vierte Element eines Tupels ist hierbei die Information, wann dieses eingetragen wurde. So verfügt man jederzeit über die Transaktionszeit, d.h., über die Information darüber, wann ein bestimmtes Wissen erlangt wurde. Es entsteht hierdurch eine Historie, die verwendet werden kann, um Veränderungen in den Daten zu dokumentieren. Ändert sich etwa eine Präferenz eines Benutzers, so wird diese nicht einfach überschrieben, sondern lediglich durch die Eintragung eines neuen Objektes mit denselben Slotwerten, aber unterschiedlichem Präferenzwert und neuerem Eintragsdatum ergänzt. Dabei bleibt sie für entsprechende Anfragen an die Ontologie abrufbar, sodass sie weiterhin in die Berechnungen mit einbezogen werden kann.
Beim Start von Smoto wird ein HFC-Server gestartet, auf den die von PAL verwendete Dialogakthierarchie geladen wird, sowie auch alle von Smoto aktiv verwendeten RDF-Klassen. Letztere wurden mitsamt ihren Prädikaten in Protégé (Stanford Research, 2017) erstellt, im OWL-Format abgespeichert und mit dem Tool Raptor (Beckett, 2017) in das von HFC erwartete N-Tripel-Format umgewandelt.
Neben diesen unveränderbaren Ontologieteilen wird als dritte Quelle die Datei mit den von Smoto selbst erstellten Ontologieeinträgen eingelesen. Es handelt sich dabei um alle in vergangenen Gesprächen gespeicherten Informationen über den Benutzer - wie etwa Gesprächsthemen, Präferenzen und Termine - sowie um die Erklärungen zu unbekannten Wörtern, die dem System gegeben wurden. Alle Ontologieeinträge von Smoto entsprechen Einträgen ins Langzeitgedächtnis, da zur Zeit kein Vergessen bzw. selektives Erinnern implementiert ist.

3.2.1. Finden und Nutzen von Events und Benutzerpräferenzen

Informationen aus eingehenden Dialogakten des Benutzers, die vom Dialogakttyp *Inform* subsumiert werden, werden im episodischen Gedächtnis gespeichert. Diese Ontologieeinträge bilden nicht nur die Dialoghistorie ab, sondern kodieren auch einen Großteil der über den Benutzer erlangten Informationen wie etwa Vorlieben, Abneigungen und Termine. Im Regelblock `AnalyzeLastDA` werden sie hierzu in drei Gruppen unterschieden: Aussagen über Präferenzen, Aussagen über zeitlich bestimmte Ereignisse und Aussagen, die in keine der beiden Gruppen passen.

RDF-Klasse	Slots	Beispiel
Preference	agent, patient, activity?, pref_score	*I like movies.*
Event (1)	agent, patient, activity?	*I watched a movie.*
Event (2)	agent, patient, activity?, time, expired	*Tomorrow I'll watch a movie.*

Abbildung 2: Aussehen von Einträgen des Episodengedächtnisses von Smoto

Enthält die Äußerung des Benutzers eine temporale Ergänzung, so kann sie klar als ein zeitlich bestimmtes Ereignis erkannt werden. Smoto berechnet dann mit der Zeitangabe aus dieser Ergänzung, die so ungenau sein kann wie *tomorrow*, einen Zeitpunkt in Long. Diese Zeitangabe wird, ebenso wie *agent*, *patient* und das Vollverb des Dialogaktes, in einem neuen RDF-Objekt der Klasse Event gespeichert. Liegt der betreffende Zeitpunkt in der Vergangenheit, so wird dies mit dem Marker *expired* für das Aggregatmodul markiert.
Handelt es sich beim eingehenden Dialogakt nicht um ein zeitlich bestimmtes Ereignis, so wird abgeprüft, ob eine Präferenz beschrieben wurde. Hierzu wird zunächst untersucht, ob die Proposition des Dialogaktes - d.h., das Prädikat des ursprünglichen Satzes - eine Präferenz ausdrückt. Ein Abgleich mit der WordNet-Datenbank ergibt, ob sie mit *like*, *love*, *dislike* oder *hate* verwandt ist und weist ihr darauf basierend einen Präferenzwert zwischen -2 und 2 zu. Wird hierbei kein Präferenzwert gefunden, so wird ein ähnliches Verfahren mit der modalen Ergänzung des Satzes durchgeführt, falls vorhanden. Smoto legt sodann eine neue Präferenz an, in die ähnlich wie bei einem Event die semantisch wichtigen Slots des Dialogaktes übernommen werden.
Äußerungen des Benutzers, die weder ein zeitlich bestimmtes Ereignis beinhalten noch als Präferenz erkannt werden, werden als Ereignis ohne Terminangabe angelegt. Sie wurden zwar nicht erkannt, enthalten aber auch ohne Verarbeitung relevante Informationen etwa über den Gesprächsverlauf und die Gesprächsthemen, die zur Aggregation wichtig sind. Nachdem die Auswertung des eingehenden Dialogaktes beendet wurde, wird das resultierende RDF-Objekt in der Dialoghistorie des Benutzers gespeichert.

3.2.2. Aggregieren

Das Aggregatmodul berechnet, wann zeitlich bestimmte Ereignisse ablaufen und welche Themen sich auf Grundlage der Dialoghistorie anzusprechen lohnen. Für jedes Thema wird eine Instanz der Java-Klasse Topic erzeugt, in der das das Thema repräsentierende Konzept ebenso gespeichert wird wie Verben, die im Zusammenhang mit diesem Konzept verwendet wurden. Sowohl für die abgelaufenen Ereignisse als auch für mögliche Konversationsthemen wird eine Liste angelegt.
Das Modul enthält unter anderem die folgenden Methoden:

`getMatchingRobotPrefs:` Methode, die zu gegebenen Argumenten in der Präferenzliste des Systems nach passenden Einträgen sucht und diese in einer Liste zurückgibt.

`addTopic:` Fügt zur Topic-Liste ein neues Element hinzu, das das als Argument mitgegebene Topic repräsentiert, und speichert die ebenfalls mitgegebene Verbliste darin.

`updateTopicsList:` Analysiert den als Argument gegebenen Eintrag der Dialoghistorie und erstellt ein passendes Topic, das zur Topic-Liste hinzugefügt wird.

`startTopic:` Generiert für ein gegebenes Topic eine Anzahl an Proposals, mit denen es angesprochen werden kann

`startNewTopic:` Wählt zufällig ein Topic aus der Liste aus und ruft `startTopic` mit diesem auf.

`updateExpEvents:` Analysiert alle Einträge der Dialoghistorie des Benutzers daraufhin, ob es sich um ein zeitlich bestimmtes Event handelt, das mittlerweile abgelaufen ist. Ist dies der Fall, wird es zur Liste der abgelaufenen Ereignisse hinzugefügt.

`loadRobotPrefs & loadUserPrefs:` Ruft für jeden Eintrag der entsprechenden Dialoghistorie die Methode `updateTopicsList` auf, um dessen Thema zur Liste hinzuzufügen.

`computeAggregates:` Wird zu Beginn jedes Regeldurchlaufs ausgeführt und ruft `updateExpEvents` auf. Lädt beim Systemstart mittels der obigen Methoden Themen aus der Dialoghistorie von Benutzer und System und fügt bei laufendem System das Thema des letzten berechneten Eintrags in der Dialoghistorie des Benutzers zu den Topics hinzu.

3.2.3. Präferenzen des Systems

Selbstverständlich können auch für das System Präferenzen eingetragen werden. Dies ermöglicht es nicht nur, während dem Gespräch erfundene Antworten auf Fragen wie „Do you like horses?“ zu speichern, um im nächsten Dialog dieselbe Meinung vertreten zu können. Es eröffnet auch die Chance, Smoto bereits vor dem ersten Gespräch an den Dialogpartner, bzw. den Anwendungsbereich anzupassen. So kann ihm beispielsweise eine Grundbegeisterung für Sport und gesundes Essen mitgegeben werden.
Außerdem kann zu einer angelegten Präferenz eine Menge an Dialogakten gespeichert werden, die sie begründen und die das System im Gespräch zur Rechtfertigung nutzen kann. So wird es möglich, nicht nur wie bei den für den Benutzer angelegten Präferenzen einen Eintrag als Grund für einen anderen anzugeben, sondern auch komplexere Begründungen zu äußern, wie etwa „Water makes my joints rust“ für die Abneigung gegen Wasser.

3.2.4. Auflösung von Referenzen und Reasoning über Possessivpronomen

Unterhaltungen zwischen Menschen sind voller Ausdrücke, die Konzepte und Personen nicht direkt benennen, sondern nur auf sie referieren. Ein Dialogsystem, das diese natürliche Form des Gesprächs beherrschen soll, benötigt die Fähigkeit, solche Referenzen bei der Interpretation eingehender Dialogakte aufzulösen. Smoto besitzt keine elaborierte Interpretation, implementiert jedoch ein Referenzauflösungsmodul, das die im Small Talk erwarteten Fälle von Referenzen korrekt zuordnen kann.
So werden die Personalpronomen *you* und *I* zu den URIs der den Benutzer bzw. den Agenten darstellenden Rdf-Objekte umgewandelt und die als *ref_backwards* geparsten Elemente wie *this* und *that* aufgelöst. Letztere werden dabei immer auf das Objekt bzw., falls es kein Objekt gab, auf das Subjekt des letzten vorangehenden von Smoto geäußerten Dialogaktes bezogen. In diesen Fällen wird in dem Slot, der den referenziellen Ausdruck enthielt, die URI des Konzeptes bzw. der Instanz gespeichert, auf die referiert wurde.
Handelt es sich beim eingehenden Dialogakt um eine Bejahung oder Verneinung, so ist er initial leer. Um das Zuschlagen auch von nicht diese Spezialfälle behandelnden Regeln zu ermöglichen, wird er mit den Slots der vorangegangenen Systemfrage - wie etwa Subjekt, Prädikat und Objekt - angereichert und mit der dem Dialogakttyp entsprechenden Polarität versehen.
Zudem werden alle Slots daraufhin überprüft, ob sie Zeichenfolgen enthalten, die keinen URIs entsprechen. Ein Vergleich mit den Labels der dem System bekannten Instanzen von WordNet-Konzepten stellt sicher, dass die der Interpretation unbekannten Zeichen-

folgen durch die URI einer Instanz ersetzt werden, falls sie eine solche bezeichnen.
Enthält der eingehende Dialogakt in den dafür vorgesehenen Slots Possessivpronomen wie *your* oder *my*, so werden diese ebenfalls durch die URIs von Agent bzw. Benutzer ersetzt. Zusätzlich muss in diesem Fall das Nomen im Satz, das von einem Possessivpronomen begleitet wird, näher betrachtet werden.
Solche besitzanzeigenden Pronomen deuten an, dass das zugehörige Konzept auf besondere Weise behandelt werden muss. Bei einer Äußerung wie „My name is Sonny“ liegt einer von zwei Fällen vor: entweder zu dem *name* repräsentierenden Konzept ist bekannt, dass es in der Ontologie als Prädikat repräsentiert ist, oder nicht. In der Minimalimplementierung von Smoto sind die existierenden Prädikate so gewählt, dass sie das Label des zugehörigen WordNet-Konzeptes sind und dementsprechend leicht identifiziert werden können. Nach Einbau des Moduls in ein größeres Projekt sollte jedoch eine Liste angelegt werden, die spezifiziert, welche Prädikate welchen Konzepten entsprechen.
Wurde ein Prädikat zugeordnet, muss der betroffene Slot im Dialogakt so ersetzt werden, dass alle späteren Regeln über dieses Prädikat auf den für die entsprechende Person gespeicherten Wert verwiesen werden. Dies passiert, indem der Slot auf das gefundene Prädikat gesetzt wird - die von der Vorverarbeitung im Slot des zugehörigen Possessivpronomens gespeicherte URI gibt an, bei welcher Person das Prädikat angewandt werden muss.
Bei einem System, das einen auf keine bestimmte Domäne beschränkten Dialog betreiben können soll, ist jedoch die Wahrscheinlichkeit groß, dass es viel öfter auf Fälle treffen wird, in denen es sich bei dem betreffenden Slotinhalt nicht um ein Prädikat handelt. Auch in solchen Fällen möchte Smoto sich jedoch an die Bedeutung der Äußerung erinnern können. Eine wichtige Information, die ein Satz wie „I visited my grandmother“ enthält, ist, dass der Benutzer eine spezifische Großmutter besitzt. Um diese implizit kommunizierte Existenz von konkreten Realisierungen im Langzeitgedächtnis ablegen zu können, sind mehrere Reasoning-Schritte vonnöten. Existiert bereits eine konkrete Instanz des im Falle des Beispiels im Slot *patient* gespeicherten WordNet-Konzeptes *<wn20instances:synset-grandma-noun-1>*, die dem Benutzer zugeordnet ist, so wird die URI des Konzeptes durch die URI der Instanz ersetzt. Falls noch keine solche Instanz existiert, wird eine entsprechende Entität im Langzeitgedächtnis angelegt. Darüber hinaus fügt Smoto ein Event ein, das das „Besitzverhältnis“ zwischen dem Benutzer und dieser spezifischen Entität ausdrückt. Dieses Event entspricht in seiner Darstellung der einer expliziten Mitteilung wie „I have a grandmother“. Sodann wird die URI des Konzeptes im betreffenden Slot durch die URI der neu erstellten Entität ersetzt.
Zukünftiger Arbeit bleibt hier die Anlegung und Behandlung von Fällen überlassen, in denen dem Benutzer mehr als eine solche konkrete Realisation zugeordnet ist. Es

empfiehlt sich in solchen Fällen, eine Nachfrage zu stellen, um zu disambiguieren. Stehen genug Gesprächsdaten zur Verfügung, könnte jedoch auch von der Häufigkeit, mit der der Benutzer von der einen oder anderen Instanz spricht, darauf geschlossen werden, welche gemeint ist.
Die beschriebene Verarbeitung von nicht als Prädikat angelegten, mit Possessivpronomen versehenen Konzepten wird in der vorliegenden Implementierung von Smoto nur für den Benutzer umgesetzt, nicht für das System selbst. Der Agent kann nichts Neues über sich selbst erfahren, weshalb keine Notwendigkeit besteht, neue Entitäten anzulegen, falls keine existierenden gefunden werden können.

3.2.5. Sich erinnern - die Initiative ergreifen

Neben der korrekten Reaktion auf eine Äußerung des Benutzers ist für das Dialogmanagementsystem Smoto auch wichtig, selbst agieren zu können, wenn das Gegenüber stumm bleibt. Dies ermöglicht es, die Initiative zu ergreifen und ein neues Thema zu beginnen, falls Lücken im Gespräch entstehen sollten. In Situationen wie etwa dem Gesprächsanfang, wenn der Benutzer schweigt weil ihm keine Antwort einfällt oder wenn Smoto keine passende Reaktion generieren konnte, wird deshalb ein Timeout aktiviert, das die Methode `takeInitiative` aufruf und so das Gespräch wiederbelebt.
Einfachste Basis eines Initiativenmoduls ist selbstverständlich, eine Reihe von Themen vorzudefinieren, über die man reden kann - etwa „What kind of sweets do you like?“ oder der Klassiker „What are your hobbies?“. Smoto bemüht sich, nicht immer solche nicht im Geringsten auf den spezifischen Benutzer zugeschnittenen Fragen zu stellen und nutzt, falls existent, frühere Unterhaltungen zur Inferierung von Themen. Hierzu werden die Erkenntnisse aus der Aggregation herangezogen, um ein neues Thema zu starten oder zu fragen, ob es Neuigkeiten zu einem Ereignis gibt, dessen „Termin“ seit seiner letzten Ansprache abgelaufen ist.
Als weitere Menge von Themen stehen die Präferenzen des Systems selbst bereit, die in diesem Fall auch den Zweck der vordefinierten Themenmenge erfüllen, da sie a priori nur solche Elemente enthalten, die beim Bau des Systems bedacht und für interessant befunden wurden. Auf diese Weise eingespeiste Themen können auch angesprochen werden, indem das System einfach eine Aussage über seine Präferenz bezüglich eines bestimmten Konzeptes macht, um daran die zugehörige vordefinierte Begründung oder aber eine Frage anzuhängen, wie es sich beim Benutzer verhält. Es kann natürlich auch immer auf den zweiten Teil verzichten und eine Reaktion des Benutzers abwarten.

3.2.6. Reaktionen auf unbekannte Wörter

Smoto besitzt lediglich das Weltwissen, das WordNet zur Verfügung stellen kann. Mit dieser Datenbank kann zwar auf einen sehr großen Knowledge-Graph zurückgegriffen werden, der jedoch aufgrund des Konzeptes von WordNet eine Art von Wissen vermissen lässt, das für Small Talk hochrelevant ist: das Wissen um konkrete Instanzen. So sind dem System etwa die Wörter *movie* oder *politician* bekannt. Jedoch wird es keinen einzigen Film oder Politiker erkennen, falls dessen Name genannt wird.
In dem Moment, in dem einem Dialogsystem ein eingehender Satz gegenübersteht, der an der für die Generierung einer sinnvollen Reaktion wichtigen Stelle ein unbekanntes Wort hat, gibt es mindestens drei verschiedene Möglichkeiten, mit dem Unwissen umzugehen:

- Den Satz ignorieren und die nächste eigene Äußerung z.B. auf einem neuen Thema basieren
- Versuchen, mit dem Restinhalt des Satzes alleine eine zufriedenstellende Reaktion zu generieren
- Den Benutzer darum bitten, das Wort zu erklären.

Der erste beschriebene Fall ist für ein Small-Talk-System offensichtlich ungeeignet, da er stets dazu führen würde, dass es versucht, abzulenken. Im zweiten Fall wäre der Versuch möglich, die Bedeutung des betreffenden Wortes aus dem Dialogzusammenhang der weiteren Äußerungen des Benutzers zu erschließen, was jedoch den Rahmen dieser Bachelorarbeit sprengt. Am ertragreichsten - und sichersten - für die Datenbank des Dialogsystems ist zweifellos Möglichkeit Nummer drei. Hier besteht selbstredend erst einmal die Gefahr, dass der Benutzer gelangweilt ist, weil das System anfangs sehr viele Fragen stellen wird. Ist die erste „Lernphase“ jedoch überstanden, so hat Smoto eine große Menge von Wissen über Entitäten angesammelt und ist ein angenehmerer Gesprächspartner, da Raten oder Themenwechsel überflüssig geworden ist. Wird das System von vielen Personen ohne Löschen des personenübergreifenden Speichers verwendet, ist daher diese Option eindeutig zu bevorzugen.
Aus dieser Überlegung heraus implementiert Smoto den dritten Ansatz. Verwendet der Benutzer ein unbekanntes Wort, so fragt das System nach, was dieses bedeutet. Auf die Antwort des Benutzers hin wird ein neues RDF-Objekt der Klasse *WordNetEntity* erzeugt, die als Label das unbekannte Wort erhält. Verweist der Benutzer direkt auf ein bekanntes WordNet-Konzept, so wird im neu angelegten Objekt gespeichert, dass es eine Instanz desselben ist. Handelt es sich bei der Antwort des Benutzers jedoch um einen Vergleich mit einer anderen konkreten Instanz, so schließt Smoto darauf, dass deren zugeordnetes WordNet-Konzept auch das des Wortes ist, dessen Bedeutung zur Frage steht. Ein Beispiel für einen solchen Fall stellt etwa die Antwort „It is similar

to Superman“ dar, wenn die Frage lautete, was *Superwoman* ist. Das System nimmt dann, da es *Superman* als Film kennt, an, dass *Superwoman* ebenfalls ein Film ist. Um die Korrektheit dieses Reasonings abzusichern, versichert sich das System mit einer Nachfrage, das richtige Wort gefunden zu haben. Ist die Instanz, mit dem der Benutzer vergleicht, allerdings wiederum ein unbekanntes Wort, so fragt Smoto danach, was dieses Wort bedeutet. Erhält das System hierauf eine Antwort, die in der Zuordnung zu einem WordNet-Konzept resultiert, wird es dieses auch dem ersten unbekannten Wort zuordnen (ein Beispiel hierzu zeigt Abbildung 7).

Auf diese Weise eingetragene Instanzen von Konzepten werden in einem der Resolutionsschritte zu Beginn der Auswertung eines eingehenden Dialogaktes mithilfe ihrer Labels in die entsprechenden Slots eingesetzt.

Das Wissen über Entitäten, d.h. Realisierungen, von WordNet-Konzepten ist nicht an Personen, nicht einmal an die Instanz des Systems selbst gekoppelt, sondern wird frei zugänglich im Langzeitgedächtnis abgelegt. Smoto kann dementsprechend den Eindruck erzeugen, ein breites Wissen über die Welt zu besitzen, da es semantische Konzepte kennt, muss jedoch Namen von konkreten Repräsentationen derselben erst noch lernen. Eine Lösung hierfür wäre zweifellos, ein passendes Lexikon einzulesen und so die Wissenslücken zu schließen. Dies würde jedoch nicht nur eine zusätzliche Speicherbelastung sowie verlängerte Laufzeit der Interpretation, sondern auch Annotationsarbeit bedeuten, da die Abbildung auf die richtigen WordNet-Konzepte entweder von Hand oder aber automatisch vorgenommen werden müsste - wobei im zweiten Fall zumindest eine manuelle Überprüfung Not täte.

3.3. Ontologien - WordNet

Neben dem episodischen Gedächtnis als „aktivem“ Teil der Ontologie wird auf einen weiteren HFC-Server WordNet (Fellbaum, 1998) als „passive“ Ontologie, also lediglich zum Lesezugriff, geladen. Hierzu wurde die von van Assem et al. (2006) erstellte RDF/OWL-Version von WordNet 2.0 in N-Tupel-Format umgewandelt. WordNet fungiert hiermit als Weltwissen von Smoto.
Wörter werden in WordNet durch Wordsenses repräsentiert, die durch ein Tag namens *label* mit ihrer natürlichsprachlichen Repräsentation verbunden sind. Die Instanzen, auf denen die für das angestrebte Verständnis natürlicher Sprache wichtigen Relationen definiert sind, sind jedoch die Synsets, abstrakte semantische Repräsentationen, die mehrere Wordsenses mit derselben Bedeutung beinhalten können. Jedes Synset bzw. Konzept besitzt wiederum ein *label*, das dem am häufigsten für dieses Synset genutzten natürlichsprachlichen Wort entspricht. Die notierten Konzepte umfassen die semantische Repräsentation von Nomen, Verben, Adjektiven, Adverben und Adjektivsatelliten. Diese sind durch Relationen miteinander verbunden, die sie etwa als Synonyme, Antonyme, oder eines als das Hyponym des anderen ausweisen.
In Smoto genutzte Relationen von WordNet sind:

`antonymOf:` Verlinkt ein Nomen, Verb, Adjektiv oder Adverb mit seinem Gegenteil

`causes:` Verbindet ein Verb mit einem anderen, das es verursacht

`entails:` Verbindet ein Verb mit den Verben, die seine Folgen beschreiben

`gloss:` Verlinkt ein Konzept mit einem String, der eine Kurzdefinition beinhaltet

`hyponymOf:` Transitive Relation, die ein Nomen oder Verb mit seinem Hyponym verlinkt

`memberMeronymOf:` Verlinkt ein Nomen mit seinen (Mitglieds-) Meronymen

`partMeronymOf:` Verlinkt ein Nomen mit seinen (Bestandteils-) Meronymen

`sameVerbGroupAs:` Verbindet ein Verb mit ähnlichen Verben, die in derselben Verbgruppe sind

`label:` Ordnet jedem Synset eine natürlichsprachliche Repräsentation zu

In der Klasse SmotoUtils.java sind alle Methoden zusammengefasst, die das Dialogmanagement durch komplexere Anfragen an die Ontologie unterstützen. Diese Funktionalitäten beinhalten auf der mit WordNet kommunizierenden Seite sowohl Methoden, die für ein Konzept eine der oben genannten Relationen abfragen, als auch solche, die auf Grundlage dieser Methoden Berechnungen anstellen.

3.3.1. Berechnung semantischer Ähnlichkeit von Konzepten

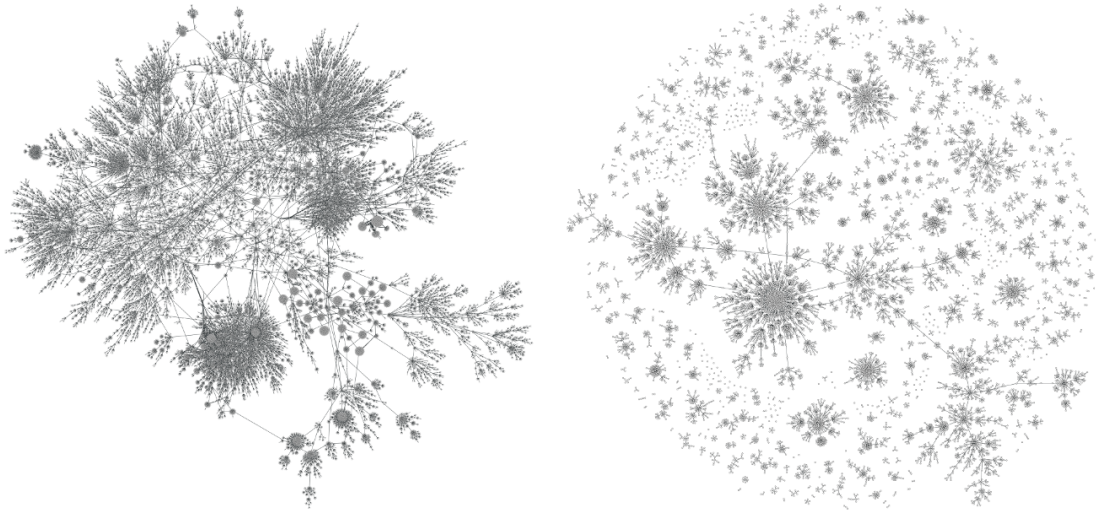


Abbildung 3: Der Graph von WordNet für Nomen (links) und Verben (rechts) (Nomen: kleem (2015a), Verben: kleem (2015b))

Da das Dialogsystem Smoto eine möglichst große Anzahl an Aussagen verstehen können soll, ohne für jedes denkbare Wort - oder auch nur Konzept - ein Reaktionsmuster vorgeschrieben zu haben, ist es von Bedeutung, eine Wissensbasis zu haben, über die Konzepte gut miteinander verglichen und in Gruppen eingeteilt werden können. Hierzu ist es notwendig, eine Methode zu finden, um die in WordNet definierten Relationen zur Berechnung solcher semantischer Ähnlichkeit zu nutzen. Ein Vergleich verschiedener Methoden, dies auf WordNet basiert für Nomen zu berechnen, findet sich in Budanitsky and Hirst (2006). Für die Anwendung in Smoto ist hierbei nicht wichtig, dass das Ähnlichkeitsmaß insgesamt möglichst genau ist, sondern vielmehr, dass die kleine Gruppe der einander sehr ähnlichen Wörter eine korrekte Beurteilung erfährt. Aus diesem Grund implementiert das vorliegende System in der Methode `getNounSimilarity` die Berechnung mit der Formel von Leacock and Chodorow (1998), wie sie im Vergleich von Budanitsky and Hirst (2006) verwendet wurde. Hierzu wurde den neun *unique beginners* von WordNet ein gemeinsames Hypernym als globale Wurzel gegeben.
Diese Methode kann deshalb nur die Ähnlichkeit von Nomen berechnen, weil Verben oder Adjektive nicht wie diese in einer vollständigen Hierarchie stehen. Während sich alle Nomen stets auf einen der *unique beginners* zurückführen lassen, sind Verben oder Adjektive weniger zusammenhängend. Abbildung 3 veranschaulicht dies mit Graphen für die Hyponym-Relationen der Synsets von Nomen und Verben in WordNet 3.0.

Da jedoch auch Verben und Adjektive eine nicht unerhebliche Bedeutung der Aussage mittragen, war es notwendig, auch für diese ein Ähnlichkeitsmaß zu finden. Hierzu wird zum einen bei Verben die Relation `sameVerbGroupAs` genutzt. Zum anderen nutzt Smoto für das Feststellen von Verwandschaft dieser Wortarten die für die Ähnlichkeitsformel der Nomen implementierte Pfadlänge über Hyponymbeziehungen, ohne diese mit einer „Höhe“ der Konzeptes in der Hierarchie zu gewichten. Verben werden außerdem als entfernt ähnlich betrachtet, falls eines das andere verursacht oder bedingt. Mit dieser Definition der Ähnlichkeiten wird der Präferenzwert von Verben oder Adjektiven berechnet.

3.3.2. Beziehungen zwischen Konzepten und Einbringen der aktiven Ontologie

Neben den Methoden, die die Ähnlichkeit von Konzepten berechnen oder auf deren Grundlage Einordnungen durchführen, enthält die Klasse SmotoUtils auch einige nützliche Funktionalitäten für Konzepte von Nomen. Über die Hyponym-Beziehung und die Pfadlänge kann etwa der direkte Hypernymknoten eines Konzeptes berechnet werden, sodass die Regeln mit Oberbegriffen arbeiten können. Ähnlich wird auch der in der Hierarchie niedrigste gemeinsame Vorfahre zweier Konzepte berechnet, oder überprüft, ob ein Konzept transitiv ein Hyponym eines anderen ist. In der umgekehrten Richtung steht auch eine Methode zur Verfügung, die die direkten Kinder eines Konzeptknotens zurückgeben kann.
Neben diesen Dingen sind auch einige Hybrid-Methoden vorhanden, die einen nahtlosen Übergang zwischen der Benutzung von WordNet und der Benutzung von im Langzeitgedächtnis abgelegten konkreten Realisierungen von WordNet-Konzepten garantieren. Viele der in diesem und im vorherigen Abschnitt angesprochenen Methoden unterscheiden implizit zwischen Konzepten und Instanzen. Ohne die Regeln mit der Unterscheidung zu belasten wird das korrekte Ergebnis zurückgeliefert.
Die Hybrid-Methoden sind zuständig, wenn zwischen beiden Ontologien vermittelt wird. So kann etwa für eine URI abgeprüft werden, ob es sich bei ihr um eine konkrete Realisierung handelt oder um ein Konzept; außerdem kann für sie das zugehörige Konzept geliefert werden, oder zu einem Konzept die Liste der bekannten Instanzen, die es realisieren.

3.4. Dialogregeln - Die Basis des Systems

Der Kern des Projektes, also das Dialogmanagement selbst, wird mithilfe des Dialogmanagement-Formalismus VOnDA (Versatile Ontology-based Dialogue and Agent Framework) implementiert. In einer Regelsprache, die viele Gemeinsamkeiten mit Java hat, gleichzeitig aber einige für die Spezifikation von Dialogakten wichtige Features bereitstellt, können hier Dialogregeln beschrieben werden. Diese werden zu ausführbarem Java-Code kompiliert. Das in VOnDA beinhaltete Framework stellt auch ein Laufzeitsystem zur Verfügung, das den Durchlauf durch die Regeln ermöglicht und Methoden zur Ausführung der für VOnDA speziellen bzw. über Java hinausgehenden Features bereithält. Smoto kann durch seine Implementierung in der Regelsprache von VOnDA als Modul betrachtet werden, das wiederum aus Submodulen besteht. Die Struktur des Formalismus unterstützt es explizit, diese Modularität zur Kombination oder zum Austausch von Teilmodulen zu nutzen. So kann die vorgesehene Integration von Smoto in PAL mit wenigen Handgriffen bewerkstelligt werden.
VOnDA ermöglicht es, aus Regeln nicht direkt Reaktionen zu folgern, sondern diese als Vorschläge, sog. Proposals, zu sammeln. In den meisten Fällen können so mehrere adäquate Reaktionen auf eine Äußerung des Benutzers angesammelt werden, aus deren Pool mittels einer statistischen Komponente die beste ausgewählt wird. Betrachtet man das System als Ganzes, von Input zu Output, ist es also auch bei gleichem Zustand der Ontologie keinesfalls streng deterministisch. Smoto nutzt dies, um eine größere Variation der Reaktionen zu erreichen. Daher sollte auch die folgende Beschreibung der Reaktionen, die Smoto möglich sind, nicht als deterministisch angesehen werden, sondern immer nur als Aufzählung einer Menge von Ergebnissen für gegebene Vorbedingungen, aus der letztendlich nur eines ausgewählt und ausgeführt wird.

3.4.1. Ordnung der Dialogregeln

Da bei einem Small-Talk-System, das theoretisch unendlich viele Themen beherrschen kann, eine Einteilung der Regeln in Dialogtopics nicht sinnvoll scheint, wurden die Regeln für Smoto dem eingehenden Dialogakt entsprechend geordnet. So gibt es Unterblöcke von Regeln, die auf Begrüßung und Verabschiedung reagieren können, welche, die Fragen behandeln, und welche, die einfache Aussagesätze behandeln. Als Aussagesätze werden dabei konform zu DIT++ etwa auch Antworten auf Fragen betrachtet, wie Bejahung oder Verneinung. Eine Grafik der Strukturierung der Dialogregeln im Detail ist im Anhang zu sehen (vgl. Abbildung 5).

3.4.2. Allgemeines - Vorbearbeitung und Aggregation

Die oberste der Regeldateien hat vor allem die Funktion, alle untergeordneten Regelblöcke aufzurufen. Daneben sind hier auch einige Funktionalitäten als Methoden definiert, die in disjunkten Gruppen von Subregeln zur Anwendung kommen. Die wichtigsten davon sind `selfDisclose` und `takeInitiative`, deren Funktion und Implementierung in den Abschnitten 3.2.3 und 3.2.5 behandelt werden.
Über diese Methoden hinaus stellt die oberste Regeldatei eine Instanz des ebenfalls in Regelsprache geschriebenen, jedoch keine Regeln enthaltenden Aggregatmoduls zur Verfügung. Dieses enthält Methoden, die aus dem episodischen Gedächtnis Informationen laden und für das Gespräch interessante Ereignisse und Themen berechnen, sowie solche Methoden, die nach Einträgen mit übergebenen Charakteristika suchen. Die Funktion des Aggregatmoduls sowie der dort definierten Methoden sind in Abschnitt 3.2.2 dargestellt.
Die an dieser Stelle implementierten Regeln sind neben der Entscheidung, ob ein neuer Dialogakt des Benutzers vorliegt, für den die Evaluation der Regeln gestartet werden soll, für die Vorverarbeitung des betreffenden Dialogaktes zuständig und damit Teile der Interpretation. Dort, wo Slots WordNet-URIs enthalten, müssen diese an die Gegebenheiten der aktuell benutzen Interpretation angepasst werden. Des Weiteren erfolgt die in Abschnitt 3.2.4 beschriebene Auflösung von Referenzen.
Die Regel `set_initiative_timeout`, die nach der Auswertung aller Subregelblöcke durchlaufen wird, sorgt dafür, dass das System in dem Fall, dass die Unterhaltung länger stockt, die Initiative ergreift und versucht, das Gespräch wieder aufleben zu lassen (vgl. 3.2.5).

3.4.3. Reaktionen auf Fragen

Small Talk ist wie jede andere Art von Gespräch auch ein Vorgang des Informationsaustauschs. Um dem Benutzer das Gefühl zu geben, einem normalen Gesprächsparter gegenüberzustehen, ist es daher wichtig, dass dieser Informationsaustausch nicht einseitig bleibt. Smoto stellt dem Benutzer viele Fragen und ist in der Lage, die Antworten zu verstehen, muss andererseits jedoch auch versuchen, die Fragen des Benutzers zu beantworten.
Aus einer beliebigen Domäne stammende Fragen korrekt oder auch nur sinnvoll zu beantworten gehört jedoch in den Bereich des Question Answering. An Systemen, die wie etwa IBMs Watson (Ferrucci, 2012) diese Aufgabe bewältigen können, wird im Augenblick viel geforscht. Sicher ist, dass sie hierzu eine komplexe, tiefgehende Analyse der gestellten Frage, riesige mit Weltwissen gefüllte Datenbanken und ausgefeilte Retrieval-

Methoden benötigen - Ressourcen, die Smoto nicht zur Verfügung stehen. Die Menge der Fragen, auf die Smoto eine Antwort generieren kann, ist dementsprechend beschränkt. Zukünftiger Arbeit bleibt es überlassen, diese durch Einbindung von Methoden aus der Question-Answering-Forschung zu erweitern.
Handelt es sich beim eingehenden Dialogakt um eine Frage, so prüft Smoto als erstes, ob das angesprochene Thema sich in seiner Todo-Liste befindet, zum Beispiel in der Form, dass der Benutzer dem System empfohlen hat, einen Film zu schauen. Da im Stand-alone-System kein Update der Datenbank passiert, kann das System über den spezifischen Film nicht mehr wissen als beim letzten Mal; es wird daher eine Ausrede dafür generieren, dass es den Film nicht anschauen konnte, und die Auswertung abbrechen. Durch den Abbruch wird verhindert, dass eine der folgenden Regeln zuschlägt und damit Antwortkandidaten generiert, die diese ausdrückliche Empfehlung des Benutzers nicht beachten.
Als nächstes wird geprüft, ob es sich bei dem, wonach gefragt wurde, um ein Prädikat handelt. Ist dies der Fall, wird der in der Ontologie gespeicherte Wert dieses Prädikats für die entsprechende Person als Antwort auf die Frage verwendet. Dies funktioniert selbstverständlich nur für die einfachen Input-Sätze, die diese Implementierung erwartet; sollen komplexere Fragen beantwortet werden können, so muss das Smoto verwendende System zumindest eine Interpretation zur Verfügung stellen, die wie bei einem Question-Answering-System bereits beim Parsen bestimmen kann, wonach gefragt ist.
Dreht sich die Frage um eine konkrete, dem Agenten zugeordnete Entität, wird zu dieser im Rahmen der Informationen, die das Langzeitgedächtnis über sie enthält, Auskunft gegeben.
Zuletzt wird versucht, Fragen des Benutzers nach Erklärungen zu beantworten. Sollte der Benutzer um eine Erklärung eines aus WordNet bekannten Konzeptes gebeten haben, so antwortet Smoto mit dem zugehörigen Lexikoneintrag. Wurde eine Frage nach einer konkreten Entität gestellt, so lautet die Erklärung, dass diese eine Instanz des Konzeptes ist, das sie realisiert. Bei Fragen nach unbekannten, d.h., nach von der Interpretation weder in WordNet-Konzepte noch konkrete Realisationen derselben umgewandelten Wörtern gibt das System zu, dass es die Antwort nicht kennt. Sollte dies der Fall sein, generiert es eine Bitte um Erklärung des Wortes.
Die einzige Art von Entscheidungsfragen, auf die Smoto zur Zeit eine Antwort geben kann, ist die nach einer eigenen Präferenz. Wird eine solche gestellt, gleicht das System ab, ob es eine Präferenz für das betreffende Konzept gespeichert hat. Ist dies nicht der Fall, vergleicht es mit Präferenzen zu Superklassen des Konzeptes, nach dem gefragt wurde. Wenn etwa der Benutzer fragt, ob Smoto Buchen mag, und das System keine Information zu Buchen selbst gespeichert hat, kann es immer noch antworten, dass es gerne Bäume mag. Falls es sich um eine hardgecodede Präferenz handelt, zu der auch

ein Grund angegeben ist, informiert Smoto den Benutzer über diesen (zu hardgecodeten Präferenzen siehe Abschnitt 3.2.3).

3.4.4. Reaktionen auf Aussagesätze

Selbstverständlich reicht es in einem normalen Gespräch nicht aus, lediglich auf die Fragen des Gegenübers zu reagieren. Auch bei Aussagesätzen erwarten wir, dass der Gesprächspartner aufmerksam zuhört und dies durch Kommentare oder Nachfragen signalisiert. Ein Dialogsystem für Small Talk muss daher auch dies beherrschen. Das gilt nicht nur für Antworten auf von ihm selbst gestellte Fragen, sondern auch für alle denkbaren anderen Aussagen. Um eine flüssige Unterhaltung zu ermöglichen, versucht Smoto deshalb, auf jede denkbare Antwort oder generelle Aussage des Benutzers eine passende Reaktion zu generieren.
Zu den Aussagesätzen, d.h., den Dialogakten, die DIT++ folgend *Inform* untergeordnet sind, gehören auch Bejahung und Verneinung. Dialogakte, die als *Confirm* oder *Disconfim* gelabelt sind, implizieren eine vorangegangene, vom System gestellte Entscheidungsfrage. Sie sind damit relativ leicht zu analysieren, da das Thema vom System selbst vorgegeben wurde. Wie sie verarbeitet werden, ist somit vom Inhalt bzw. der Intention der Frage abhängig, auf die sie antworten. Smoto nimmt hierbei einen kooperativen Benutzer an, der auf die Frage sinnvoll antwortet. Bedingt durch die Menge der Entscheidungsfragen, die Smoto stellen kann, existieren in diesem Block Regeln für die Bejahung bzw. Verneinung einer Nachfrage zu inferierten Informationen, das Anlegen oder die Korrektur von Präferenzen, das Anlegen eines Todos oder den Start eines neuen Gesprächsthemas.
Hat Smoto keine Entscheidungsfrage, sondern eine offene Frage geäußert, muss er die Antwort richtig zuordnen. Auch hier gilt, dass, solange der Benutzer kooperativ ist, die Antwort zu einem großen Teil durch die Frage bestimmt wird und damit für das System leicht interpretierbar ist. In der vorliegenden Implementierung des Systems sind aus diesem Grunde auch nicht alle möglichen Arten von Antworten berücksichtigt, sondern lediglich diejenigen, die von den Fragen des Systems getriggert werden können. So wird zum einen analysiert, ob es sich beim eingehenden Dialogakt um die Erklärung eines Smoto unbekannten Wortes handelt (zum Eintrag dieser Information in die Ontologie siehe Abschnitt 3.2.6). Smoto wird sich in diesem Fall entweder für die Erklärung bedanken oder sich, falls die Information nicht direkt gegeben wird, sondern inferiert werden muss, mit einer *CheckQuestion* versichern, die richtige Interpretation vorgenommen zu haben.
Eine weitere Regel versucht, den Dialogakt als eine Antwort auf eine die Fragewörter *why* oder *how* enthaltende *WHQuestion* zu deuten. Funktioniert dies nicht, wird der Subregelblock verlassen.

Nach der Bearbeitung des Spezialfalles von Antworten wird der Dialogakt durch eine Analyse geschleust, die seine Aussage ins episodische Gedächtnis übernimmt. Welche Operationen hierbei genau durchgeführt werden, wird in Abschnitt 3.2.1 beschrieben.
Nachdem die Analyse den Dialogakt in die Datenbank eingetragen hat, wird zunächst überprüft, ob Smoto über alle Informationen verfügt, die dazu nötig sind, ihn zu verstehen. Ist dies nicht der Fall, d.h., findet sich in den Slots des Dialogaktes eine Eintragung, die keiner URI entspricht, unterbricht Smoto die Regelauswertung an dieser Stelle und fragt den Benutzer erst nach der Bedeutung des Wortes (zur Motivation dieser Reaktion siehe Abschnitt 3.2.6).
Ging aus der Analyse des Dialogaktes hervor, dass er eine Präferenz enthält - was aufgrund der Fragen, die das System an den Benutzer stellen kann, recht wahrscheinlich ist - so reagiert Smoto auf diese. Um Abwechslung zu garantieren, generiert die Regel `react_to_preference` eine Vielzahl an unterschiedlichen Proposals, die für die Äußerung des Benutzers angemessen sind. Die Subregeln, die diese produzieren, sind:

`check_with_mine:` Falls das System selbst eine Präferenz zu diesem Thema hat, kann es sich zur Übereinstimmung der Meinungen äußern. Hinzu kommt, falls vorhanden, die Möglichkeit, eine Begründung dieser Präferenz anzugeben.

`ask_for_pref_reason:` Hat Smoto selbst keine Meinung dazu, besteht die Möglichkeit, den Benutzer nach dem Grund der Präferenz zu fragen.

`ask_for_opposite_mod:` War das Konzept, über das die Präferenzaussage gemacht wurde, mit einer modalen Ergänzung versehen, so kann danach gefragt werden, ob der Benutzer die Kombination der gegenteiligen Ergänzung und des Konzeptes nicht mag (etwa „Don't you like cold weather?“ als Reaktion auf die Aussage „I like warm weather.“).

`ask_for_opposite_patient:` Existiert ein Antonym zu dem Konzept, über das der Benutzer sich geäußert hat, so fragt Smoto, ob das Gegenüber für dieses gegenteilige Empfindungen hat.

`ask_for_related:` Hat sich der Benutzer schon mehrmals zu semantisch nahe beieinanderliegenden Konzepten - etwa Handball und Fußball - geäußert, so liegt es nahe, nach dem gemeinsamen Superkonzept - also Ballsport - zu fragen. Ist die Präferenz zu diesem bereits bekannt, bleibt die Möglichkeit, nach der Präferenz eines Schwesternkonzeptes - z.B. Volleyball - zu fragen.

`ask_why_pref_changed:` Falls bereits eine ältere, von der jetzt geäußerten Präferenz verschiedene Meinung des Benutzers zum Thema gespeichert ist, fragt Smoto nach dem Grund der Änderung.

`ask_for_parts`: Zuletzt besteht die Möglichkeit, in WordNet nach Meronymen des Konzeptes zu suchen und zu fragen, wie diese aussehen müssen, um den Benutzer zufriedenzustellen (zu einem Problem dieses Ansatzes siehe Abschnitt 4.3).

Handelt es sich bei dem Dialogakt weder um eine Äußerung zu einer Präferenz noch um eine Antwort auf eine vom System gestellte Frage, so ist es schwer, eine angemessene Reaktion zu finden. Die Regel `react_to_random_info` versucht hier mit zwei Subregeln, einen Teil der möglichen Äußerungen abzufangen.
`ask_for_specific` überprüft, ob das angesprochene Konzept laut WordNet eine *entity* oder ein *event* ist. Hat der Benutzer dabei keinen konkreten Namen angegeben, sondern sich nur zum Konzept im allgemeinen geäußert - etwa „I watched a movie." - so bietet es sich an, danach zu fragen, um welche spezifische Realisierung dieses Konzeptes es sich handelte. Sind überdies bereits solche Realisierungen bekannt, wählt Smoto zufällig eine davon aus und fragt, ob es sich um diese handelte.
Die zweite Subregel, `find_old_pref_score`, schlägt zu, wenn der Benutzer schon einmal eine Aussage über seine Meinung zum jetzigen Konzept gemacht hat, aktuell aber keine Bewertung angibt. Smoto fragt dann, ob die Erfahrung dieses Mal genauso gut bzw. schlecht war wie beim letzten Mal.

3.4.5. Begrüßung und Verabschiedung

Smoto wurde als alleinstehendes Small-Talk-System entwickelt, ist aber dazu bestimmt, innerhalb des PAL-Systems verwendet zu werden. Es enthält daher nur eine rudimentäre Implementierung für Begrüßung und Verabschiedung. Ob das Kind die App nutzt oder mit dem physisch präsenten Roboter interagiert - wenn Smoto aktiviert wird, um Off-Activity-Talk zu betreiben, hat die Begrüßung bereits stattgefunden. Aus diesem Grund besitzt Smoto nur ein sehr rudimentäres Begrüßungsverhalten; das System kann lediglich reagieren, sollte der Benutzer eine Begrüßung äußern und wird selbst Hallo sagen, falls es von diesem nach dem Start nicht selbst angesprochen wird.
Aus ähnlichen Gründen wurde die Verabschiedung sehr einfach gehalten; ohnehin sollte diese nie vom System selbst initialisiert werden, da das System auf eine möglichst lange Interaktion abzielt. Sollte der Benutzer sich jedoch verabschieden oder andeuten, dass er keine Zeit mehr hat, wird dies erkannt. Das System zeigt dann, dass verstanden wurde, indem es sich ebenfalls verabschiedet und sich beendet. Da Begrüßung und Verabschiedung eigenständige Regelmodule sind, können sie beim Einbau ins Gesamtsystem von PAL weggelassen werden.

4. Evaluation und Fazit

Die experimentelle Auswertung dessen, was das System Smoto leisten kann, überlässt diese Arbeit zukünftiger Forschung. Daher kann die Effektivität der angewandten Methoden zum jetzigen Zeitpunkt nicht objektiv beurteilt werden. Die folgende Evaluation beschränkt sich aus diesem Grunde auf die bei der Implementierung von Smoto mit den gegebenen Tools angetroffenen Vor- und Nachteile der Wahl des Setups. Zusätzlich präsentiert Anhang A.2 einige Beispieldialoge mit dem System, die illustrieren, wie die beschriebenen Funktionalitäten genutzt werden.

4.1. Der regelbasierte Ansatz von Smoto

Ein Vorteil des regelbasierten Ansatzes ist zweifellos die Kontrolle über die generierten Äußerungen, die er verleiht. Diese erleichtert das Debugging, wenn bei inkorrektem oder ausbleibendem Output die Ursache des Fehlschlages gesucht wird. Zudem ermöglichte er die Einbindung der Ontologie an genau den Stellen, die der Programmierer für sinnvoll hält. Auch hier war sehr gut zu kontrollieren, welche Eintragungen ins Langzeitgedächtnis wann und wie vorgenommen werden; die WordNet-Ontologie als Weltwissen ist bei der Auswertung und Ausführung von Regeln eng eingebunden. Die Proposal-Architektur von VOnDA ermöglichte es, trotz dieser strikten Kontrolle Diversität zu erzeugen.
Die Stärke des Ansatzes bleibt selbstverständlich auch seine Schwäche, nämlich, dass die Reaktionen des Systems immer auf das begrenzt sein werden, was bei der Implementierung vorgesehen wurde. Ein regelbasiertes System, das über alles reden können und auf alles reagieren können soll, benötigt für diese Art von vorgetäuschter Freiheit logischerweise sehr viele oder sehr generische Regeln, um alles abzudecken. Derart generische Regeln zu erschaffen, die differenzierter wirkende Ausgaben erzeugen als es etwa ELIZA tun würde, scheint jedoch mit viel Forschungsbedarf verbunden. Dimensionen, die hierzu betrachtet werden müssten, sind Sentimentanalyse, semantische Gruppierung verschiedenster Wortarten und wie diese zu gleich zu behandelnden „Klassen“ von Eingaben zusammenzufassen sind.
Der allgemeine Ansatz von Smoto, der versucht, ohne Festlegung auf Bereiche möglichst viel abzudecken, birgt außerdem die Gefahr, dass in bei der Implementierung nicht berücksichtigten Fällen syntaktisch oder semantisch fragwürdige Äußerungen generiert werden. Dies kann nur mit sehr vielen Tests abgesichert werden.

4.2. Verwendung von VOnDA zum Dialogmanagement

Die Verwendung von VOnDA hat sich sowohl durch die Einfachheit, Regeln und Ontologiezugriffe niederzuschreiben, als auch durch das bereitgestellte Framework, als für diese

Arbeit sehr sinnvoll herausgestellt. Vor allem die unkomplizierte und hierdurch wenig fehleranfällige Kommunikation mit der Ontologie erleichterte die Programmierarbeit. Hierzu kommt, dass einige Fehler - wie etwa Tippfehler in Prädikaten - von VOnDA bereits erkannt und dem Benutzer mitgeteilt werden.
Vorteil auch für die Nutzung des Systems Smoto in PAL ist zudem die abstrakte Bedeutungsrepräsentation der generierten Dialogakte. Sie sind sprachenunabhängig, da die Slots keine Wörter, sondern WordNet-Konzepte enthalten. Um Smoto zu ermöglichen, sich in einer anderen Sprache als dem bei der Implementierung verwendeten Englisch zu unterhalten, ist lediglich ein Austausch der Module für Interpretation und Generierung notwendig. Sofern für diese eine Interpretation vorliegt, die Wörter auf die korrekten WordNet-Konzepte abbilden kann, funktioniert Dialogmanagement von Smoto mit jeder beliebigen Sprache. Für das TTS gilt die umgekehrte Regel.
Ein Problem und Todo für VOnDA liegt jedoch in der internen Repräsentation der Dialogakte. Um anspruchsvolles Dialogmanagement betreiben zu können, braucht man besonders bei komplexeren Sätzen als den für Smoto angenommenen mehr Informationen über den eingehenden Satz. Die in VOnDA implementierten Dialogakte beherrschen jedoch nur eine flache Semantik, in der keine Verschachtelungen möglich sind. Dies führt zu der Notwendigkeit von Benennungen wie *pat_poss* für das zum *patient* gehörige Possessivpronomen. Während dies bei so einfachen Sätzen wie denen, die das System Smoto erwartet, noch akzeptabel ist, führt es jedoch zu Schwierigkeiten, falls etwa im Falle von Nebensätzen Teile des Dialogaktes rekursiv verarbeitet werden sollen. Für das Dialogmanagement sind im Moment keine Beziehungen zwischen Slots sichtbar, sondern nur die Slots selbst.

4.3. WordNet als Ontologie

WordNet als Weltwissen zeigte in dieser Arbeit dort einen Wert, wo die Bestandteile jeder Äußerung des Benutzers einem semantischen Konzept zugeordnet werden konnten. Durch die große Anzahl an Konzepten, die in WordNet vertreten sind, ist so sichergestellt, dass auch seltene Wörter eine semantische Repräsentation erhalten. Diese ermöglicht es durch die in WordNet enthaltenen Relationen, semantische Ähnlichkeiten zu berechnen, sich über Themen zu unterhalten, die den bekannten Lieblingsthemen des Nutzers ähneln und durch Ähnlichkeit zu positiven oder negativen Konzepten in der Äußerung enthaltene Wertungen zu inferieren. Des Weiteren fördern Informationen wie Synonyme oder Antonyme die Variation des Outputs.
Keine Datenbank kann jedoch alles menschliche Wissen vereinen, und so fiel auch bei der Arbeit mit WordNet auf, dass manche für ein Dialogsystem benötigten Informationen fehlten. Ein gutes Beispiel dafür stellt eine Funktionalität dar, die in Smoto eingebaut

wurde, um auf vom Benutzer geäußerte Präferenzen eingehen zu können. Spricht der Benutzer etwa in einer Unterhaltung davon, dass es ein schöner Tag sei, so kann die Regel `ask_for_parts` entscheiden, ein zufällig ausgewähltes Teilmeronym anzusprechen. Sie generiert dann zum Beispiel, wenn das Konzept *day* lautet, eine Frage wie *What is your favourite hour?*. Dieselbe Regel kann jedoch bei der zufälligen Wahl eines anderen Meronyms die semantisch falsche Frage *What is your favourite noon?* produzieren. WordNet bietet dabei keine Möglichkeit, die Konzepte *hour* und *noon* voneinander zu unterscheiden.
Eine Verbesserung an WordNet, die nicht nur im beschriebenen Beispiel, sondern auch in anderen Anwendungsfällen von Vorteil wäre, wäre es, jedem Konzept eine Information darüber mitzugeben, ob Instanzen davon existieren können. Denn nur bei solchen Konzepten, die konkrete, voneinander unterscheidbare Realisierungen haben können, erscheint es sinnvoll, nach solchen zu fragen.

4.4. Die übrigen Tools - HFC und CPlan

HFC zahlte sich für das Dialogmanagement und den Aufbau der Dialoghistorie schon allein durch die Möglichkeit aus, Quadrupel zu verwenden und so alte Einträge desselben Inhaltes nicht überschreiben zu müssen. Trotz der großen Menge an Tupeln startet auch der HFC-Server, auf den WordNet geladen wird, in einem angemessenen Zeitrahmen. Bei Berechnungen semantischer Ähnlichkeit mithilfe von WordNet wurden nur selten kurze Verzögerungen aufgrund von Ontologiezugriffen beobachtet.
Eine Beschreibung der Funktionalitäten des frei verfügbaren Graph-Transformators CPlan findet sich in der Dokumentation des GitHub-Repositories[1]. Um mit CPlan eine Grammatik zu erstellen, die für solch zufällige Eingaben wie die von Smoto produzierten eine Generierung erzeugt, die menschenähnlich wirkt, ist mehr Arbeit nötig, als für dieses Projekt in die Generierung investiert wurde. Da das Hauptaugenmerk hier nicht auf der Generierung lag, diese jedoch zur Überprüfung und Präsentation des Systems wünschenswert ist, ist die beinhaltete CPlan-Grammatik für Smotos Ausgaben sehr beschränkt. Sie wird jedoch bei zukünftigen Verbesserungen am System leicht zu erweitern sein, ein Umstand, der sich auch schon während seiner Implementierung bezahlt machte. In der Struktur der Grammatik war es sehr einfach, immer dann zeitsparend eine neue Generierungsregel an bestehende Strukturen anzuschließen, wenn in Smoto eine neue Art von Dialogakt, bzw. Kombination von Dialogaktslots, hinzugefügt wurde.

[1] siehe `gui/doc/` auf https://github.com/bkiefer/cplan

5. Weiterführende Arbeit

Da Smoto dem Arbeitsaufwand einer Bachelorarbeit entsprechend eher als Proof-Of-Concept zu verstehen ist gibt es naturgemäß viele Ansatzstellen, an denen das System durch Erweiterungen verbessert werden kann. Diese Erweiterungen lassen sich grob in solche aufteilen, die die Ontologie betreffen, und in solche, die Optionen zum Dialogmanagement an sich hinzufügen.

5.1. Ontologie

Zunächst einmal ist anzumerken, dass WordNet noch deutlich mehr potentiell nützliche Relationen enthält als die hier verwendeten. Zukünftiger Arbeit, die darauf abzielt, komplexere Sätze zu verstehen oder mehr Variation in der Formulierung der von Smoto generierten Reaktionen zu erzeugen, steht hier noch ein großer Pool an Optionen zur Verfügung.
Zur genaueren Unterscheidung der Intentionen des Gesprächspartners bietet es sich an, bei der Analyse der eingehenden Dialogakte nicht nur *like*, *love* und ihre Gegenteile zu berücksichtigen, sondern auch Modalverben wie *want* einzubeziehen und zu dokumentieren. Zu diesem Zweck bietet sich an, den Schritten von Interpretation und Vorverarbeitung ein auf Machine-Learning basierendes Sentiment-Analyse-Modul hinzuzufügen. Das so gewonnene, differenziertere Bild des Gegenübers kann dabei helfen, diesen besser zu verstehen und natürlicheren Dialog zu betreiben.
Eine sehr interessante, linguistische Aufgabe ist es, die in Abschnitt 3.2.6 angesprochene Behandlung unbekannter Wörter zu verfeinern. Nach diesen nicht fragen zu müssen, sondern auf Grundlage des Weltwissens sowie des Kontextes von Dialog und Äußerung das zugehörige Konzept erschließen zu können, würde die Glaubwürdigkeit des Dialogsystems als „natürlicher“ Gesprächspartner zweifellos verstärken.
Bei der langzeitigen Nutzung der Small-Talk-Fähigkeit von Smoto sollte außerdem in Erwägung gezogen werden, ein „Vergessen“ zu implementieren. Wird das System intensiv benutzt, besteht die Möglichkeit, dass die von Castellano et al. (2008) angesprochene Problematik des Speicherplatzes zutage tritt. Unter der gegebenen sparsamen Speicherung der Informationen in Tupeln ist dies recht unwahrscheinlich. Auch die Laufzeiteffizienz der Ontologieoperationen wird durch die Konstruktion von HFC durch große Datenmengen wenig beeinflusst, wie am WordNet-Server leicht zu sehen ist. Für Menschen ist es jedoch natürlich, hin und wieder Dinge zu vergessen. Es kommt uns sogar seltsam vor, wenn sich jemand Wochen nach einer Unterhaltung noch an völlig unwichtige Details erinnert. Eine Art von Vergessen, die vorstellbar ist, ist daher, eine Funktion zu implementieren, die Informationen nach ihrer emotionalen bzw. faktischen Bedeutung

und ihrer Eintragungszeit gewichten kann und somit berechnet, ob sie weiterhin genutzt werden sollen.

5.2. Gesamtsystem

Der nächste Schritt in der Entwicklung von Smoto sollte sein, schrittweise die Komplexität der eingehenden Dialogakte zu vergrößern und das System so anzupassen, dass diese weiterhin sinnvoll klassifiziert und ins Dialoggedächtnis eingetragen sowie dass eine passende Reaktion darauf erzeugt werden kann. Hierzu sollte zunächst darauf gewartet werden, dass VOnDA komplexere Dialogakte mit tieferer Semantik formulieren kann oder aber eine Möglichkeit bietet, anderweitig rekursiv Informationen über die getätigte Äußerung zu erhalten. Sodann können die bestehenden Regeln von Smoto an diese neue Syntax angepasst und für komplexere Inputsätze erweitert werden.
Der Ehrgeiz, in allen Bereichen besser zu werden, erfordert jedoch auch, für konkrete Bereiche Module hinzuzufügen, die deren spezielle Erforderlichkeiten abdecken. So sollte etwa, um den Regelblock auszubauen, der Fragen beantwortet, ein angemessenes Question-Answering-Modul angeschlossen werden. Dieses muss nicht die Qualität und Komplexität eines IBM Watson besitzen, könnte aber wohl nach Abstimmung auf die geringe Menge an Smoto zur Verfügung stehendem Weltwissen sowie dem Wissen über Dialoghistorie und Benutzer eine zufriedenstellende Menge an denkbaren Fragen beantworten.
Nutzbringend wäre sicherlich auch das Experiment, die Regeln von Smoto mit einem einfachen Chatbot zu kombinieren. Finden die Regeln einmal keine angemessene Reaktion auf eine Äußerung des Benutzers, könnte diese einem Chatbot-Modul eingespeist werden, das sodann eine Antwort ausspuckt. Bei einem System wie Smoto, das versucht, sich mit dem Benutzer zu unterhalten, ist es zum Beispiel vorstellbar, für den Fall eines Nichtverstehens einige ELIZA-Patterns bereitzuhalten, die auf ihre an Psychiater erinnernde Art eine weitere Äußerung provozieren, welche dann wiederum von den Dialogregeln verarbeitet werden kann.
Im Rahmen von PAL erscheint es außerdem wichtig, dem System eine Persönlichkeit zu geben, sodass es als vollwertiger Gesprächspartner anerkannt werden kann. Dies muss zum einen durch eine differenziertere Repräsentation des Agenten in der Ontologie geschehen, indem mehr persönliche Informationen für den Agenten erschaffen und in der Datenbank gespeichert werden. Zum anderen muss das System auch die Fähigkeit erhalten, sich ausführlich zu diesen Informationen zu äußern. Hierzu bietet es sich an, das Self-Disclosure-Modul, das von Burger (2016) ebenfalls für PAL konzipiert wurde, in das Konzept von Smoto aufzunehmen und es bei passenden Gelegenheiten, etwa als Teil des Initiativenmoduls, zu starten.

A. Anhang

A.1. Darstellung der berücksichtigten Dialogaktmuster und der Regelhierarchie des Projektes

Beispiele	Dialogakt & Argumente
I watched Titanic *That is a movie*	Inform(V, agent=S, patient=O)
It's such a nice day *That is a good meal*	Inform(V, agent=S, patient=O, modMod=Am)
France is nice	Inform(V, agent=S, modMod=Am)
It's such a nice day *That is a good meal*	Inform(V, agent=S, patient=O, modMod=Am)
Let us talk about that *I like planting flowers*	Inform/Suggestion(HV, agent=S, patient=O, activity=VV)
Because the sun is warm *Because the sun is shining nicely*	Inform(V, agent=S, (modMod=Am), kind=*reason*)
Tomorrow I'll go to France	Inform(V, agent=S, modTemp=At, modLoc=Al)
Do you like movies? *Do you like Titanic?* *Did you watch Titanic?*	YNQuestion(V, agent=S, patient=O)
What is hypoglycemia? *What is that?*	WHQuestion(V, agent=S, patient=O, kind=WH)
What is my name? *When is your birthday?*	WHQuestion(V, agent=S, patient=O, pat_poss=Poss, kind=WH)
Yes; I do; ...	Confirm
No; I don't; ...	Disconfirm
Hello; Hi; ...	Salutation(Greeting)
Bye; I have to go; ...	Salutation(Bye)

S	Subjekt	HV	Hilfsverb	Am	modale Ergänzung
O	Objekt	WH	Fragewort	At	temporale Ergänzung
V	Vollverb	Poss	Possessivpronomen	Al	lokale Ergänzung

Abbildung 4: Oben: Muster von Eingaben, auf die Smoto reagieren kann
Unten: Legende zu den verwendeten Abkürzungen

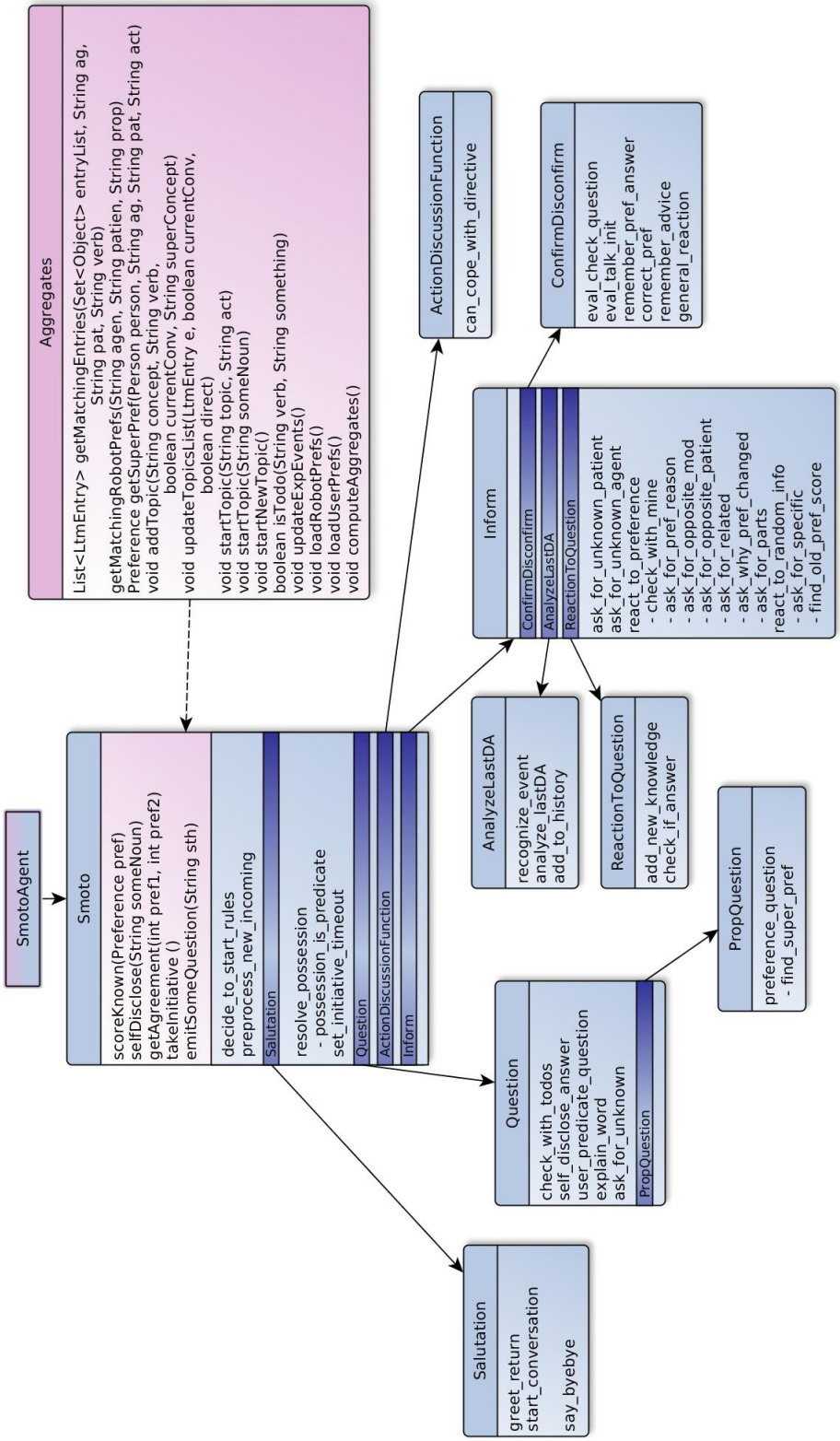


Abbildung 5: Die Regelstruktur von Smoto

A.2. Unterhaltungen mit Smoto

```
user@example: I watched Titanic.
>>> Smoto:    Can User explain Smoto what a Titanic is?
user@example: That is a movie.
>>> Smoto:    I see. Smoto thank User for explain. Do User like
              Titanic?
user@example: Yes
>>> Smoto:    Do User recommend Smoto to watch Titanic?
user@example: Yes
...
user@example: Did you watch Titanic?
>>> Smoto:    Wanted to do that, but my power was low.
```

Abbildung 6: Smoto unterhält sich über eine Aktivität des Benutzers, fragt nach einem unbekannten Wort, und merkt sich eine Empfehlung

```
user@example: I watched Iron Man.
>>> Smoto:    Can User explain Smoto what a Iron Man is?
user@example: It is similar to Titanic
>>> Smoto:    Do Iron Man be movie?
user@example: Yes
>>> Smoto:    Ah. Smoto thank User for explain.
```

Abbildung 7: Smoto erkennt einen Vergleich und schließt aus dem Konzept des genannten Beispiels auf das des unbekannten Wortes

```
user@example: My name is Anna
>>> Smoto:    Do Anna want to talk about weather?
user@example: Yes
>>> Smoto:    Do Anna like weather?
user@example: No
>>> Smoto:    Why don't Anna like weather?
user@example: Because it is warm
>>> Smoto:    I see.
```

Abbildung 8: Smoto erfährt den Namen des Benutzers und verwendet ihn fortan

```
>>> Smoto:    Do User want to talk about hypoglycemia?
user@example: What is that?
>>> Smoto:    Hypoglycemia do be abnormally low blood sugar
              usually resulting from excessive insulin or a poor
              diet.
```

Abbildung 9: Smoto schlägt ein Thema vor und erklärt auf Nachfrage hin den Begriff

```
user@example: Tomorrow I'll go to France
>>> Smoto:    Smoto wish User fun.
  (one day later)
>>> Smoto:    How be France?
user@example: France was nice
>>> Smoto:    Ah.
```

Abbildung 10: Smoto erkennt ein Event, wünscht viel Spaß und fragt nach Ablauf der dem Event zugeordneten Zeit, wie es war

```
user@example: It's such a nice day
>>> Smoto:    Why do User like day?
user@example: Because the sun is warm
>>> Smoto:    Ah. That sounds good.
>>> Smoto:    Do User want to talk about weather? Smoto do like
              dry weather. Smotos joints do not rust then.
user@example: Let us talk about that.
```

Abbildung 11: Smoto erkennt eine Präferenz und fragt nach deren Grund. Dann versucht das System, ein neues Thema anzusprechen, indem es über eigene Präferenzen redet

Literatur

ALICE A.I. Foundation (2016). http://www.alicebot.org/. Zuletzt überprüft: 11.10.2017.

Amazon Alexa (2017). https://developer.amazon.com/alexaprize. Zuletzt überprüft: 11.10.2017.

Beckett, D. (2017). Raptor RDF Syntax Library. http://librdf.org/raptor/. Zuletzt überprüft: 11.10.2017.

Budanitsky, A. and Hirst, G. (2006). Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47.

Bunt, H. (2009). The DIT++ taxonomy for functional dialogue markup. In *AAMAS 2009 Workshop, Towards a Standard Markup Language for Embodied Dialogue Acts*, pages 13–24.

Burger, F. (2016). Agents Sharing Secrets-Self-disclosure in Long-Term Child-Avatar Interaction.

Castellano, G., Aylett, R., Dautenhahn, K., Paiva, A., McOwan, P. W., and Ho, S. (2008). Long-term affect sensitive and socially interactive companions. In *Proceedings of the 4th International Workshop on Human-Computer Conversation*.

Fellbaum, C. (1998). *WordNet*. Wiley Online Library.

Ferrucci, D. A. (2012). Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1–1:15.

Goedheijt, B. S. (2017). Recalling shared memories in an embodied conversational agent.

Goffman, E. (1967). *Interaction Ritual: Essays in Face to Face Behavior*. AldineTransaction.

Kanda, T., Hirano, T., Eaton, D., and Ishiguro, H. (2004). Interactive robots as social partners and peer tutors for children: A field trial. *Human-computer interaction*, 19(1):61–84.

Kasap, Z. and Magnenat-Thalmann, N. (2010). Towards episodic memory-based long-term affective interaction with a human-like robot. In *RO-MAN, 2010 IEEE*, pages 452–457. IEEE.

Kiefer, B. (2017). cplan. a Rule-based Graph Rewriting Tool. https://github.com/bkiefer/cplan. Zuletzt überprüft: 23.10.2017.

kleem (2015a). WordNet noun graph. https://gist.github.com/kleem/6ab92f48ef961da271ab. Zuletzt überprüft: 18.10.2017.

kleem (2015b). WordNet verb graph. https://gist.github.com/kleem/6516ac84c6c186d8bc13. Zuletzt überprüft: 18.10.2017.

Klüwer, T. (2011). From chatbots to dialog systems. *Conversational agents and natural language interaction: Techniques and Effective Practices*, pages 1–22.

Klüwer, T. (2015). Social talk capabilities for dialogue systems.

Krieger, H.-U. (2013). An efficient implementation of equivalence relations in OWL via rule and query rewriting. In *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, pages 260–263. IEEE.

Kruijff-Korbayová, I., Oleari, E., Baroni, I., Kiefer, B., Zelati, M. C., Pozzi, C., and Sanna, A. (2014a). Effects of Off-Activity Talk in Human-Robot Interaction with Diabetic Children. In *Ro-Man 2014: The 23rd IEEE International Symposium on Robot and Human Interactive Communication. IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), August 25-29, Edinburgh*, pages 649–654. IEEE.

Kruijff-Korbayová, I., Oleari, E., Pozzi, C., Racioppa, S., and Kiefer, B. (2014b). Analysis of the Responses to System-Initiated Off-Activity Talk in Human-Robot Interaction with Diabetic Children. In *Proceedings of the 18th Workshop on the Semantics and Pragmatics of Dialogue. Workshop Series on the Semantics and Pragmatics of Dialogue (SEMDIAL-2014), Edinburgh, United Kingdom*, pages 90–97. Heriot Watt University.

Leacock, C. and Chodorow, M. (1998). Combining local context and WordNet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283.

Lee, P. (2016). Learning from Tay's introduction. https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/. Zuletzt überprüft: 11.10.2017.

Lim, M. Y., Aylett, R., Vargas, P. A., Ho, W. C., and Dias, J. (2011). Human-like memory retrieval mechanisms for social companions. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1117–1118. International Foundation for Autonomous Agents and Multiagent Systems.

McTear, M. F. (2004). *Spoken dialogue technology: toward the conversational user interface*. Springer Science & Business Media.

Mihalcea, R. (1998). Semcor Semantically tagged corpus.

PAL (2017). The PAL Project Website. http://www.pal4u.eu/. Zuletzt überprüft: 11.10.2017.

Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., Kim, T., Pieper, M., Chandar, S., Ke, N. R., et al. (2017). A Deep Reinforcement Learning Chatbot. *arXiv preprint arXiv:1709.02349*.

Stanford Research (2017). Protege.stanford.eu. http://protege.stanford.edu. Zuletzt überprüft: 11.10.2017.

Tulving, E. (2002). Episodic Memory: From Mind to Brain. *Annual Review of Psychology*, 53(1):1–25. PMID: 11752477.

van Assem, M., Gangemi, A., and Schreiber, G. (2006). RDF/OWL representation of WordNet. *W3C working draft*, 19.

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Wikipedia (2017a). Chatbot. https://en.wikipedia.org/wiki/Chatbot. Zuletzt überprüft: 28.10.2017.

Wikipedia (2017b). Gedächtnis. https://de.wikipedia.org/wiki/Ged Zuletzt überprüft: 1.11.2017.

D Master Theses

DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2017*

Recalling shared memories in an embodied conversational agent

Personalized robot support for children with diabetes in the PAL project

BART SCHREUDER GOEDHEIJT

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

KTH Information and
Communication Technology

Recalling shared memories in an embodied conversational agent

Personalized robot support for children with diabetes in the PAL project

BART SCHREUDER GOEDHEIJT

Master's Thesis at KTH Information and Communication Technology

Supervisors:
Drs. R. Looije (TNO)
Prof. dr. M.A. Neerincx (TNO)
Dr. P. Lungaro (KTH)

Examiner:
Prof. dr. K. Tollmar (KTH)

TRITA-ICT-EX-2017:187

Abstract

The PAL project aims to help children with type 1 diabetes to improve their self-management skills using a social robot and its virtual avatar. It has been challenging to gain a long-term relationship with a robot or virtual character. After the novelty effect wears off, the interest of the user decreases over time. The aim of this project was to explore and develop personalized interactions with the children, using episodic memory to improve the engagement and diabetes self-management. A module was built that could capture and refer to shared experiences between the PAL actor and the child. During an experiment with children, the usage decreased over time after the novelty effect wore off. No increases in affection, motivation and diabetes self-management were found after the implementation of the episodic memory module. The full potential of episodic memory was however untested, as the novelty effect already wore off before the implementation. Further research is recommended in order to assess the benefits of an improved version of the episodic memory update during an A/B test.

Sammanfattning

Att framkalla delade minnen i en förkroppsligad samtalsagent

Pal-projektet syftar till att hjälpa barn med diabetes typ 1 att förbättra färdigheter för att klara av diabetes-relaterade rutiner med hjälp av en social robot och dess virtuella avatar. Det har visat sig vara svårt att behålla relationen till en robot eller virtuell karaktär under längre tid. Efter att nyhetseffekten försvinner så sjunker användarens intresse relativt snabbt. Syfte med mitt examensarbete har varit att utforska och ta fram personanpassade interaktioner med barn genom att använda episodiskt minne för att öka engagemanget och på så sätt förbättra diabetes-relaterade rutiner. Första steget var att utveckla en modul som fångar och hänvisa till gemensamma erfarenheter mellan roboten och användaren. Detta testades därefter med hjälp av ett experiment som ingick i en större försök där denna modul användes. Resultat visar att barnen minskade användningen över tid även i detta fall, och vi har inte kunnat visa någon ökning i känslomässig spänning, motivation och förbättrade diabetes-relaterade rutiner. Det episodiska minnets fulla potential har dock inte kunnat testmts i detta experiment, eftersom nyhetseffekten delvis redan avtagit i det aktuella försöket. Framtida forskning rekommenderas att utvärdera fördelarna av en förbättrad version av det episodiska minnets uppdatering under en kontrollerad a/b-test.

This thesis is dedicated to the memory of my grandmother, Helene Tollenaar-Groen (1932 - 2017).

Acknowledgements

The final outcome of this thesis was not possible without the assistance from many people. I would hereby like to thank everyone that helped me to deliver this work.

I would first like to thank my supervisors Rosemarijn Looije and Mark Neerincx from TNO. They gave me the opportunity and the freedom to explore my own interests within the context of the PAL project. They supported me with continuous and in-depth feedback, which greatly improved the work in this thesis.

I also want to thank my examiner Konrad Tollmar for his patience and support while I was working on my thesis abroad. Even with the physical distance, he was able to provide me with good feedback and guidance during several Skype sessions.

A special thanks goes out to Bert Bierman (Produxi) for guiding me during the prototype development and for being so supportive and understanding in difficult times.

Finally, I would like to thank my parents, my girlfriend and my roommates for providing the encouragement and emotional support. I could not finish this thesis without their support. Thank you!

Bart Schreuder Goedheijt

Contents

List of Acronyms and Abbreviations

PAL	Personal Assistant for a healthy Lifestyle
T1DM	Type 1 Diabetes Mellitus
OWL	Web Ontology Language
RDF	Resource Description Framework
AEO	Affection-based Episode Ontology
SDT	Self-Determination Theory
CET	Cognitive Evaluation Theory
sCE	Situated Cognitive Engineering
5W1H	Who, What, Where, When, Why, How
EMA	Ecological Momentary Assessment

Chapter 1

Introduction

"Hey! It's nice to see you again! Did you enjoy the party yesterday?" It's such a small gesture, but the role of remembering is highly important in daily life. The feeling that someone cares and thinks about you is one of the cornerstones of a friendship. This is probably even more meaningful for children with a chronic disease.

1.1 Background

1.1.1 Type 1 Diabetes Mellitus

Type 1 Diabetes Mellitus (T1DM) is a disease that prevents the pancreas from creating insulin. The body breaks down food into glucose and sends it into the blood. The insulin is needed for moving the glucose from the blood into the cells. This is either directly used as fuel for energy or stored for later use. People with T1DM have to keep their blood glucose level at a normal level to prevent hypoglycemia (blood glucose level too low) or hyperglycemia (blood glucose level too high). This is done by checking blood glucose levels, planning meals (choosing what, how much and when to eat), taking insulin and by performing physical activity. This is required in order to feel well during the day and to prevent or delay long-term health problems. The disease therefore requires a high level of self-management. T1DM is usually diagnosed in children and young adults. [1]

1.1.2 PAL Project

The PAL (Personal Assistant for a healthy Lifestyle) project aims to help children (between the age of 7 and 14) with type 1 diabetes to improve their self-management skills using a social robot and its virtual avatar combined with a set of mobile health apps (figure 1.1). The robot and avatar act as a pal for these children and help them to achieve diabetes-oriented goals. Alongside the robot and the virtual avatar, the PAL system includes a module for use by health professionals to instruct and supervise the system, as well as a module to monitor progress and inform

the parents. All of these modules are connected to the PAL cloud environment. The project is funded by the European Union in the Horizon2020 program (ref. H2020-PHC-643783). The project builds on work that has been done in earlier projects, including the ALIZ-e project [2]. The PAL project is coordinated by TNO (Netherlands Organisation for Applied Scientific Research) and includes partners from The Netherlands, Italy, the United Kingdom and Germany [3].

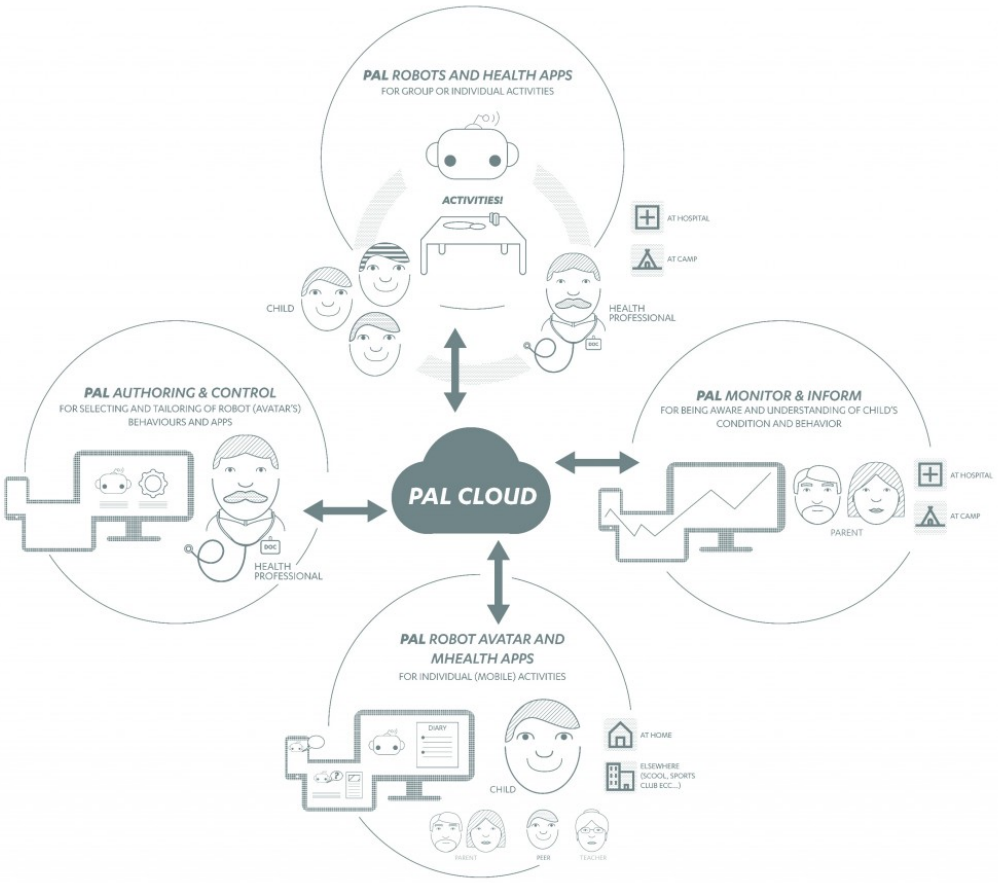


Figure 1.1. Overview of the PAL system [4]

Diabetes self-management

The self-management skills are measured using learning objectives. These learning objectives are set to acquire knowledge and skills or to do certain tasks. These objectives are based on the Know&Do goals from the Expertise group Paediatric Diabetes Nurses (EPDN) (see Appendix B). The healthcare professionals from the hospitals in the PAL project validated these goals and categorized them in to age

groups based on their difficulty. The learning objectives are integrated in the mobile health apps. The current system has the following mobile health apps:

- **Timeline:** a diary for the children which also contains blood-sugar measurements.
- **Quiz:** a quiz that helps the children to achieve knowledge goals related to their diabetes.
- **Break and sort game:** a game that helps children to identify and categorize different types of food.

These apps can be used with the physical robot in the hospital or at diabetes camps, but also with the virtual avatar on a tablet at home.

Long-term relationships with virtual and robotic characters

It has been challenging to gain a long-term relationship with a robot or virtual character. After the novelty effect wears off, the interest of the user decreases over time. In previous work of the PAL project, a way was found to decrease this novelty effect wear-off using self-disclosure (see § 3.9). Besides self-disclosure, it has been demonstrated that a long-term memory in (virtual) robots further decreases this novelty effect. This includes having and recalling shared experiences. Kasap et al. developed a system with the sole purpose of keeping the attention of users after the first interaction [5]. They found that the users' interest in the system did not decrease with time by remembering and referring to the engagement level of users.

1.2 Problem statement

As mentioned before, T1DM requires essential self-management by the young patients. One of the objectives within the PAL project is to develop an agent-based reasoning mechanism to set personalized learning and behavioural goals, and engagement strategies for children. This objective would support and increase the self-management of the children [4]. During several meetings in the hospital, the children will perform different activities with the social robot. In the Netherlands and in Italy, the robot also makes its appearance during diabetes related summer camps. These camps provide valuable input for the PAL project, as they provide greater insight into the practical usage of the PAL system using observations and by conducting interviews. Besides these hospital and camp sessions, the children will also use a tablet at home with the virtual avatar of the robot to perform these activities. In order to support and increase the children's diabetes self-management, the child has to gain a long-term relationship with the (virtual) robot. After the novelty effect wears off, the children still have to use the PAL system in order for it to be effective. To prevent the decrease in usage over time, it could be useful to memorize and refer to earlier meetings. For example, the robot should be able to

keep track of goals and give feedback on the actual progress compared with earlier meetings. It should give feedback on the stored measurements of their blood sugar level (e.g. in case that the child forgot to fill it in). Also, if the child tells the robot that (s)he will go to a birthday party, the robot could ask if they enjoyed it in the next meeting. The robot is currently unable to memorize and refer to earlier meetings. In addition, the improvement of diabetes self-management due to making reference to earlier meetings still has to be proven empirically.

1.3 Purpose and goal

The purpose is to research, explore and develop personalized interactions with the children using episodic memory to increase the overall engagement and usage of the health apps, which would ultimately lead to increased diabetes self-management. The goal of the project is to develop a memorization module that could be integrated with the current robot and the virtual avatar. This module should be able to capture two types of memory: the augmented memory of the child (activity reports and observed behaviour) and the episodic knowledge of the PAL actor (see § 3.4). It should therefore be able to reason over the intersection of these captured memories: the shared experiences between the PAL actor and the child. Finally, it should fit within the current system and architecture. The PAL system uses Ontologies for defining its knowledge and data storage (see § 3.3). Therefore, it should support and build upon the current PAL Ontology (see § 3.3.1).
The module is built in different iterations. The goal for the first iteration is to implement the ability to memorize goals and progress in order that they can be used to give summarized feedback. This would increase the effectiveness as mentioned in the Goal-Setting Theory (see § 3.2). Besides the Goal-Setting Theory, the Nudge Theory could be used in order to push the behaviour change in the right direction (see § 3.8). The second iteration would include the memorization and reasoning over blood sugar measurements in order to improve the intrinsic motivation (see § 3.1). The third iteration would contain the ability to memorize and reason over the activities that the children entered in the diary, which could be solved by using an Affection-based Episode Ontology (see § 3.6). The final experiment with real users should at least contain the first iteration.

1.4 Research question

The main research question in this project is:

> **How should the PAL actor capture and refer to past experiences that were shared with the child?**

The PAL actor is the social robot or the virtual avatar. The past experience in this case is a memory that has been shared between the PAL actor and the child. The

PAL actor should therefore be able to capture these shared memories and refer to them at a later moment.
To answer this research question, five subquestions have been defined. The breakdown of the main research question into these subquestions is shown in figure 1.2

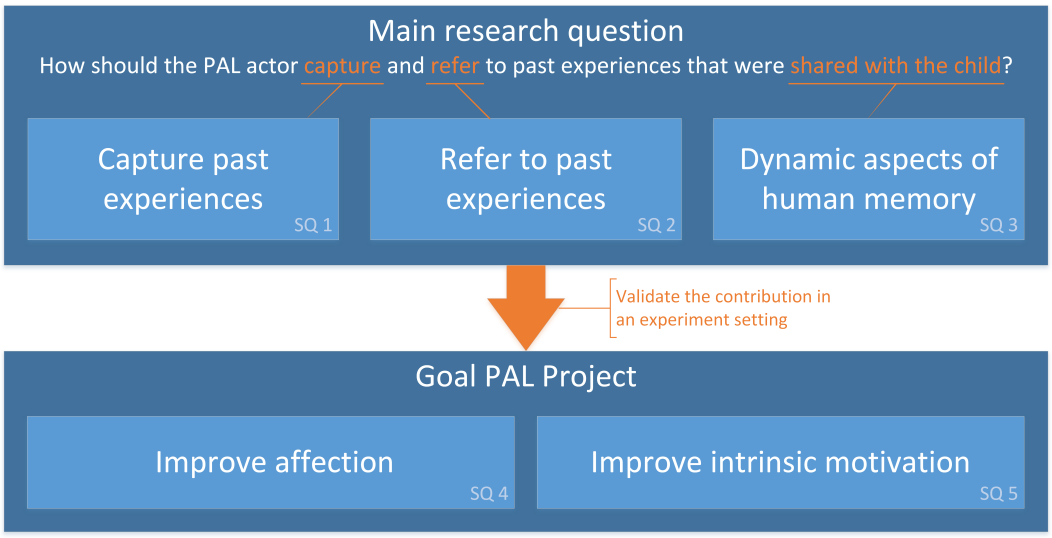


Figure 1.2. Research question breakdown

The first subquestion regarding the capturing of past experiences is:

SQ1: Which ontological model can capture the main concepts with their relations of the child-PAL shared episodic memory?

This question is intended to research the memory capture process. We need to define a memory and how to store it. This should be the shared experience between the PAL actor and the child. The second question is:

SQ2: Which reasoning mechanism can the PAL actor apply for constructive memory reference expressions?

A reasoning mechanism is essential in order to refer to these memories at a later moment. It should be able to reason about the memories stored in the ontological model from the previous subquestion.

SQ3: How could dynamic aspects from the human memory (e.g. forgetfulness, context association, memory combining and generalization) be translated to the PAL actor episodic memory?

The third subquestion is defined to investigate the possibilities of translating dynamic aspects of the human memory to the PAL actor. This question is added to make the reasoning mechanism stay closer to the actual shared experience with the child.

SQ4: Does the addition of episodic memory alter the affection with the embodied conversational agent?

SQ5: Does the addition of episodic memory improve the intrinsic motivation for achieving goals?

The last two subquestions are defined to evaluate how the system affects and contributes to the PAL project goals. This will be done in an experiment using a system that will be developed using the answers from subquestion 1, 2 and 3. Subquestions 4 and 5 are defined to answer the hypothesis.

1.5 Hypothesis

The hypothesis is that referring to past memories will delay the novelty effect wearing off. This will prevent the usage decrease of the PAL system. In other words, children will still like to use the system after the experiment ends. This effect will result in the improvement of the intrinsic motivation of the children and the affection with the PAL actor.

Chapter 2

Situated Cognitive Engineering

This research is using the Situated Cognitive Engineering method (sCE) [6]. This method was designed by the TU Delft and TNO for complex, intelligent and interactive technology. The sCE method allows quick, incremental and iterative cycles, which requires researchers to continuously specify, refine and integrate different parts of the system. It supports the collaborative process of multi-disciplinary teams as it addresses human factors, technical and operational issues. An active involvement of stakeholders is also included to understand users' support needs and to enhance user acceptance. As the overall PAL project also uses sCE, this research involves the usage of sCE for the creation of a module in several iterations.

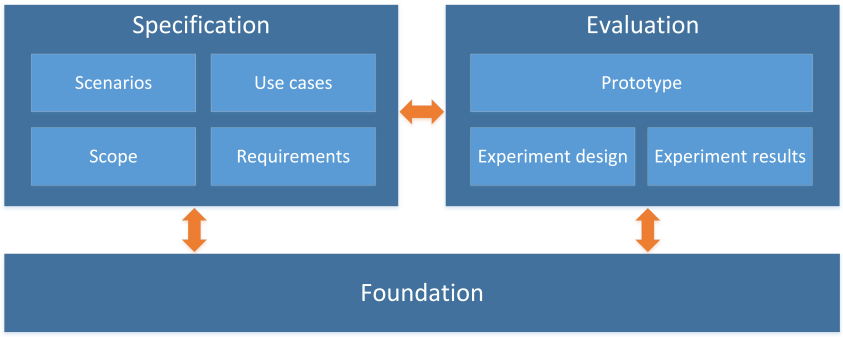


Figure 2.1. Situated Cognitive Engineering method

The upcoming chapters are formed according to the segments from the sCE method. As seen in figure 2.1, sCE distinguishes three main segments: Foundation (chapter 3), Specification (chapter 4) and Evaluation (chapter 5).

2.1 Foundation

The theoretical framework or background is called foundation in the sCE method. The foundation segment describes the relevant existing knowledge on ways to solve the problem. It also contains envisioned technologies or available options to develop

the new solution. These can use existing technologies or involve the creation of new technologies.

2.2 Specification

The specification describes the system design of the solution. It is based on the identified relevant human factors knowledge and the envisioned technology from the foundation layer. The specification consists of the design scenarios, the use cases, the artifact scope and the requirements.

2.3 Evaluation

The evaluation segment aims to test and validate the system's design, or to compare multiple design options in order to improve the current design in an iterative way. The three relevant parts of the system evaluation are: the artifact (an implementation or prototype), the evaluation method (experiment design) and the evaluation results in which the claims are tested. The experiment was designed and conducted by the PAL consortium and therefore out of the scope of this thesis. The creation of the prototype that was tested in this experiment as well as the analysis of the results are part of this thesis work. The experiment design is, however, still described in § 5.2 for completeness.

Chapter 3

Foundation

The following theories are part of the foundation:

Chapter	Theory	Relevant SQ
§ 3.1	Self-Determination Theory and Intrinsic Motivation	SQ2, SQ5
§ 3.2	Goal-Setting Theory	SQ2, SQ5
§ 3.3	Ontology	SQ1 - SQ5
§ 3.3.1	PAL Ontology and handling time	SQ1 - SQ5
§ 3.4	Episodic memory	SQ1 - SQ5
§ 3.5	Retroactive learning	SQ1
§ 3.6	Affection-based Episode Ontology	SQ1
§ 3.6.1	5W1H Extraction	SQ1
§ 3.7	Motivational interviewing	SQ2, SQ4, SQ5
§ 3.8	Nudge	SQ2, SQ5
§ 3.9	Self-disclosure and long term interaction	SQ2, SQ5

Table 3.1. Foundation overview

3.1 Self-Determination Theory and Intrinsic Motivation

The Self-Determination Theory (SDT) is a fusion of two different theories about psychological growth and personality development. The first view assumes that the will to learn new capabilities is congenital. In other words, the motivation to facilitate psychological growth is natural behaviour. In the second view it was argued that external factors are necessary for the learning process. The social context of a human influences the self-development and thus facilitates psychological growth. Ryan and Deci combined these two views into the SDT. They assume that people have the natural tendency for self-development, but the environment can either facilitate or undermine this. In order to facilitate this growth, three main needs are required from the environment: competence, relatedness and autonomy [7].

Deci and Ryan also presented the Cognitive Evaluation Theory (CET) as a sub-theory within SDT. This theory involves the specification of factors that influence intrinsic motivation. Intrinsic motivation is the inherent tendency to seek, learn and explore new capabilities. CET focuses on the effects of social-contextual events (e.g. feedback or rewards) on intrinsic motivation. CET also specifies that only the combination of competence and autonomy will enhance intrinsic motivation. Some studies also show that the factor relatedness can influence intrinsic motivation, however, Deci and Ryan suggest that there is a stronger link between the intrinsic motivation and the combination of competence and autonomy [7]. Intrinsic motivation is essential when it comes to achieving goals.

3.2 Goal-Setting Theory

Alongside intrinsic motivation, the goal setting theory is helpful in achieving goals. It concerns the relation between conscious goals, intentions and task performance. Based on different studies, Locke concluded that specific and challenging goals lead to higher performance than easy goals, 'do your best' goals, or no goals at all. Goal setting is most likely to improve the task performance when specific and sufficiently challenging goals are used [8]. Giving summary feedback during the achievement of a goal results in an even higher performance. Becker describes this as one of the best-established findings in psychology [9].

3.3 Ontology

Hoekstra describes the term Ontology as a method of knowledge representation. This is a field of Artificial Intelligence that deals with problems regarding the design and use of formal languages that are suitable for capturing human knowledge. It is a way to translate human knowledge to a computer model. The ultimate goal is to perform intelligent automated reasoning on these models. This knowledge-based reasoning can be used to perform tasks that are normally carried out by humans, but can, for example, involve large amounts of data which is difficult for humans to process. [10] Ontologies are often designed in the Web Ontology Language (OWL); a Semantic Web language designed to represent rich and complex knowledge about things, groups of things and relations between things [11]. It builds on top of the Resource Description Framework (RDF) which is a specification of how to write triples (e.g. Mary - eats - a pancake). Using OWL, the aspect of semantics is included to write a valid ontology.

3.3.1 PAL Ontology and handling time

Krieger et al. described the ongoing work carried out in the PAL project which described the unified ontology that was constructed [12]. It consists of seven independently developed sub-ontologies which were combined in an eighth sub-ontology

using a set of hand-written interface axioms. They extended the triple model of RDF by adding two extra arguments in order to represent time-varying data. The information within the PAL system was categorized as TBox (terminological knowledge), RBox (general information about properties), and ABox (assertional knowledge). The TBox and RBox of the PAL domain stay constant and will not change over time. The relation instances in the ABox might change over time (e.g. the weight of a child can change, but the birthdate will not). The PAL project used HFC as a semantic repository and bottom-up forward chainer in order to store the TBox and RBox information, but also to query the ABox time-varying data. A use case for this would be the processing of dialogs, which uses if-then-like rules in order to match those against dialog situations. Both the matching information as well as the derived new information is grounded in time, thus the time sequence should be recorded. Another use case is the progression of goals to inform the child, its parents and the healthcare professionals on the current status of self-management. The progression is also important for the PAL system to provide suitable content and activities. The children can achieve goals over time by answering quiz questions or by doing other activities. As soon as the child masters all the related knowledge or skills, the goal is achieved.
The next step for the PAL Ontology will be the creation of a sub-ontology that is able to capture episodic memory.

3.4 Episodic memory

Endel Tulving was the first to describe episodic memory 45 years ago [13]. The concept of episodic memory has changed a lot since then. Nowadays, episodic memory is seen as one of the major neurocognitive memory systems that allows humans to re-experience their own previous experiences. Turing describes it as “a possibility for mental time travel through subjective time, from the present to the past” [14]. The application of episodic memories in intelligent agents, as described by Nuxoll, makes task-independent episodic memories possible that support a wide range of cognitive capabilities within a cognitive architecture [15]. They made an artificially intelligent agent that was able to demonstrate five cognitive capabilities: virtual sensing, action modeling, decision-making based on past experiences, retroactive learning and boosting other learning mechanisms.

3.5 Retroactive learning

Nuxoll also described retroactive learning in the same paper. This is the theory of understanding an experience or review of learning experiences when resources become available [15]. It is, for example, useful to schedule the intense use of resources later, when the system is not in use. As a memory is, by definition, in the past, it does not matter if it is captured after the conversation with a child. This

concept can therefore be used for the resource intensive algorithms that the capture process of an episodic memory can bring.

3.6 Affection-based Episode Ontology

Lim et al. describe an affection-based episode ontology [16]. In this model, they combined Ontology-based unified robot knowledge (OUR-K) [17] with temporal episode ontology in order to model event episode knowledge. They state that knowledge about an event episode can be closely related to an emotion. Therefore, they added emotion ontology to episode ontology. As a result, the following affection-based episode ontology was defined:

- Definition 1: Affection-based Episode Ontology (AEO)
  AEO = [TES, Emotion]
  TES is Temporal Episodic Structure,
  Emotion is Emotion ontology,

- Definition 2: Temporal Episodic Structure (TES)
  TES = [When, Where, Who, Why, What, How, Inst]
  When is Time ontology,
  Where is Space ontology of OUR-K,
  What is Object ontology of OUR-K,
  Who is Human ontology of OUR-K,
  How is Action ontology of OUR-K,
  Inst is additionally recognized knowledge instance of OUR-K

3.6.1 5W1H Extraction

The temporal episode ontology from the previous section is built around the 5W1H (When, Where, Who, Why, What and How) principle. In the paper of Han et al., a method to extract the 5W1H from natural language is described [18]. They designed a Natural Language Understanding module that used a conditional random field (CRF) model [19], combined with lexical word features (lexical trigrams) to train their model. This was used to create a counseling dialog system. The system was able to interact with users by recognizing what the users said, by predicting the context, and by following the users' feelings. The counselling was done using three basic counselling techniques in order to attend and show empathy with clients: open questions, paraphrasing and reflection of feelings. This could become interesting as a method to capture the shared experiences between the child and PAL actor.

3.7 Motivational interviewing

In addition to the capturing of episodic memory, an episode also needs to be referred to in a proper way. Motivational Interviewing (MI) [20] is a method of interact-

ing with patients in order to enhance behaviour change. It tends to improve the patients own motivation and commitment to change. The method is well-tested and established with over 160 randomized clinical trials. It defines three ways of interaction with the patient: directing, following and guiding. In the directing style, the practitioner provides information, instruction and advice. The following style is doing the opposite by listening and understanding to the patient but respectfully refraining from inserting his/her own material. Motivational Interviewing is mostly built around the guiding style. Guiding lies between and incorporates elements of the directing and following styles. The overall goal is to guide the patient through the proper thought processes that will inevitably result in the salient reason for behaviour change. MI is built around the principles of partnership, acceptance, compassion and evocation. The partnership implies that it has to be done 'for' or 'with' the patient and not 'on' or 'to' them. Related to this partnership is acceptance of what the client brings. Compassion is a deliberate commitment to pursue the welfare and best interests of the other. Lastly, evocation is to detect deficits to be corrected and to install those. The implicit message is "I have what you need, and I'm going to give it to you". MI uses five different communication styles that are shared with many forms of counseling: asking open questions, affirming, reflective listening, summarizing, and providing information and advice. In order to improve the intrinsic motivation and with that the diabetes self-management of the children, the PAL actor could use the communication styles from MI.

3.8 Nudge

Alongside the techniques from motivational interviewing, a nudge can be useful to motivate the users to achieve certain goals. Thaler and Sunstein developed the nudge theory [21]. A nudge can be described as a push in the right direction. It is used to design policies that push individuals toward better choices without limitations on their liberty, also known as 'libertarian paternalism'. It is allowed to influencea the behaviour of others to improve their quality of life (paternalism), but it should be unobtrusive and not an obligation (libertarian). The famous example is to have fruits on the counter and place the unhealthy food a bit further away at high schools. This nudge is used in order to make children eat more healthy food. A nudge in the context of the PAL project could be that by achieving a goal, children will also receive a reward. It is still a choice to achieve goals, but the children will become extra motivated in order to earn the reward. For example, during the diabetes camps where the PAL robot also appears, the reward could be to do a fun activity with the robot, like dancing. In the home situation, children could be allowed access time to use games or apps out of the PAL context. This is, however, out of the scope for this research.

3.9 Self-disclosure and long term interaction

It is considered difficult to keep children interested in a companion agent after the novelty effects wear off. Burgers presented a way to include self-disclosure in long-term child-avatar interaction as a way to cope with this problem using different theories. [22] The Self-Determination Theory argues that motivation is sustained if the human feels related to the agent. According to the Social Penetration Theory, this relatedness can be forged through the reciprocal disclosure of information about the self. As a result, a disclosure dialog module was developed to study the self-disclosing behaviour of children in response to that of a virtual agent. The module was implemented in the PAL system. Tests showed that the relative number of disclosures was an indicator for the relatedness children felt towards the agent at the end of the study. Also, girls disclosed significantly more than boys and children preferred to respond to avatar disclosure of lower intimacy.

Chapter 4

Specification

This chapter contains all the specifications that have been used in order to build a prototype for the PAL actor. It contains the scenarios, the use cases, the scope of the artifact and the requirements.

4.1 Scenarios

4.1.1 Scenario 1: Feedback on goals

Esther had her first encounter with robot Charlie in the hospital a little over a week ago. During the meeting, the nurse discussed the first set of goals with Esther and her parents. In the days after the meeting, Esther played with the PAL app once or twice per day. The PAL avatar gave summarized feedback on her goals on a daily basis. It was quite clear that she had mastered one of the goals. After a phone conversation with the nurse, the goals were adjusted in order to challenge Esther a bit more. The previous time, the PAL robot was part of the meeting in the hospital, but could of course not attend the phone conversation. Esther wanted to see if the new goal was updated in the app. The avatar knew that the goals were changed, but it did not attend a meeting together with the nurse and the child. The avatar asked: "Hey! Did you have a conversation with the nurse over the phone today?". Esther confirmed this and explained the new goals. The avatar replied: "Oh, great to see that you understand the goals! Shall we start?". Afterwards, Esther and the avatar played the quiz with newly added questions.

4.1.2 Scenario 2: Measurements of blood sugar

Fred loves playing soccer with his friends. He does not like to worry about his diabetes all day and relies quite a bit on his mother. She makes sure that the blood sugar levels are measured and that the carbs are counted well. He does like to interact with the PAL app, but he often skips it in order to play soccer after school. When he launched the app one day, the avatar greeted him: "Hey, I did not see you in a while! How are you doing?". Fred wrote to the avatar: "Hey! I was playing

soccer with my friends again yesterday and today!". The avatar replied: "Oh, that's really nice! I can imagine that you did not have time to fill in your blood sugar levels yesterday. Do you still have them for me? I'm curious about how you did.". He asked his mother what his values were yesterday and added them to the diary.

4.1.3 Scenario 3: Activity

Maaike returned home from school. She had a hypo during gym class and therefore she had to quit. Maaike was quite angry at herself because she knew that she in fact did not eat enough before she started. As soon as she arrived, she took the tablet out and started the PAL app. She wrote about what happened in the diary. The avatar replied: "That is very unfortunate! I really do appreciate the fact that you shared this with me.". One week later, Maaike returned home from the gym again. Luckily, she was well prepared this time. She was very happy and excited. After she told her mother about the day at school, she grabbed the tablet and greeted the avatar. The avatar asked Maaike if she had gym lessons again and if it went better than previous week. She told the robot enthusiastically that she indeed had gym class which went very well as she paid special attention this time. The avatar gave her a compliment in return: "Well done Maaike!".

4.2 Use cases

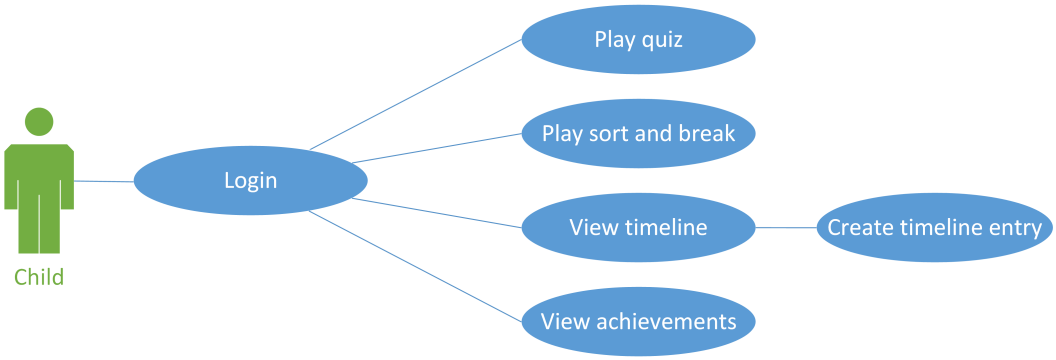


Figure 4.1. Use case diagram

In figure 4.1, the identified use cases from the MyPAL app are shown. In this research project, only alternative flows will be added to the existing main flows. As an example, the login use case including the added alternative flow is shown in Table 4.1 and Table 4.2.

Name	**Login**
ID	1.0
Summary	A child can login to the PAL system
Primary actors	Child
Secondary actors	
Preconditions	An account has been made for the child
Main Flows	1. Actor launches the app 2. Actor enters name and password 3. Actor selects 'Login' 4. System checks the combination of name and password [1] 5. System opens the main menu with the virtual avatar [2] 6. Virtual avatar welcomes the user with a question 7. Actor enters answer and selects 'OK' [3]
Postconditions	Main menu is visible. Actor may proceed with an activity.
Alternative	[1] No internet connection [2] Username / password wrong [3] Stored episode available with OnLogin trigger

Table 4.1. Login use case description: Main flow

Name	**Stored episode available with OnLogin trigger**
ID	1.3
Summary	If an episode is available with the OnLogin trigger, mention the episode.
Primary actors	Child
Secondary actors	
Preconditions	Unmentioned episode with OnLogin trigger is captured in an earlier session
Main Flows	1. System refers to episode
Postconditions	Continue with 1.0 postconditions
Alternative	

Table 4.2. Login use case description: Alternative flow

4.3 Scope

The following scope is defined for the artifact:

Iteration	**Subject**
Iteration 1	Goals and progress *Example: Shall we practice goal X again? Previous time it was quite challenging, remember?*
Iteration 2	Blood sugar measurements *Example: Oh, you forgot to fill in your measurements yesterday. Do you still have them for me?*
Iteration 3	Activities *Example: How was the birthday party? Did the organizer take your diabetes into account?*

Table 4.3. Artifact scope

4.4 Requirements

The functional requirements in this section are defined per iteration. Each iteration has a claim that corresponds with the hypothesis in § 1.4.

Iteration 1: Goals

ID	Requirement	Prio*
G01	**The PAL actor shall motivate the child to play the quiz again**	**S**
G01A	in case that the child is almost done with achieving a specific goal.	S
G01B	in case that a specific goal was too difficult for the child during the previous session	S
G02	**The PAL actor shall mention the overall goal progress of the child**	**S**
G02A	when the child launches the app (e.g. previous time we achieved goal X, let's go for goal Y today!)	S
G02B	when the child finishes a quiz game (e.g. we are almost done with goal X, let's finish it next time!)	C
G03	**The PAL actor shall notice changed goals that were set by the nurse**	**S**
G03A	at home: the virtual avatar shall ask the child if (s)he had a phone call with the nurse (not a shared experience)	S
G03B	in the hospital: the physical robot shall show awareness of the new goals (as it is a shared experience)	C
G04	**The PAL actor shall be aware of situational differences (hospital, camp, app)**	**C**
G04A	in this way, the PAL actor is able to mention the location it was previously at (e.g. we finished goal X in the hospital together)	C

Table 4.4. Requirements iteration 1

Claim: Children will stay motivated to play the quiz.

Iteration 2: Measurements

ID	Requirement	Prio*
M01	**The PAL actor shall remind the child to enter a measurement from the past**	**S**
M01A	in order to get a complete overview of the measurements (e.g. hey, you forgot to enter a measurement for yesterday's dinner. Do you still have that one perhaps?)	S
M02	**The PAL actor shall give summarized feedback over measurements**	**C**
M02A	for example to tell the child that the values were all in range for the past week	C
M02B	for example to ask the child if (s)he knows why the values were mostly out of range during the day	C

Table 4.5. Requirements iteration 2

Claim: Children will stay motivated to add diary entries.

Iteration 3: Activities

ID	Requirement	Prio*
A01	**The PAL actor shall extract an event from the child's input**	**S**
A01A	using the 5W1H model (Who, what, where, when, why, how)	S
A02	**The PAL actor shall store a trigger for referring to an event in the future**	**S**
A02A	in this way, when the event is over, the ECA can refer to this (e.g. did you enjoy event X?)	S
A03	**The PAL actor shall ask the child specific questions based on knowledge databases**	**C**
A03A	when the child, for example, mentions a place, the PAL actor could ask if (s)he saw highlight X of that place)	C

Table 4.6. Requirements iteration 3

Claim: Children have a higher affection with the PAL actor.

** MoSCow prioritization (M=must have, S=should have, C=could have, W=won't have)*

Chapter 5

Evaluation

This chapter starts with the development of the prototype. Afterwards, the experiment in which the prototype was tested will be discussed.

5.1 Prototype

5.1.1 Context viewpoint

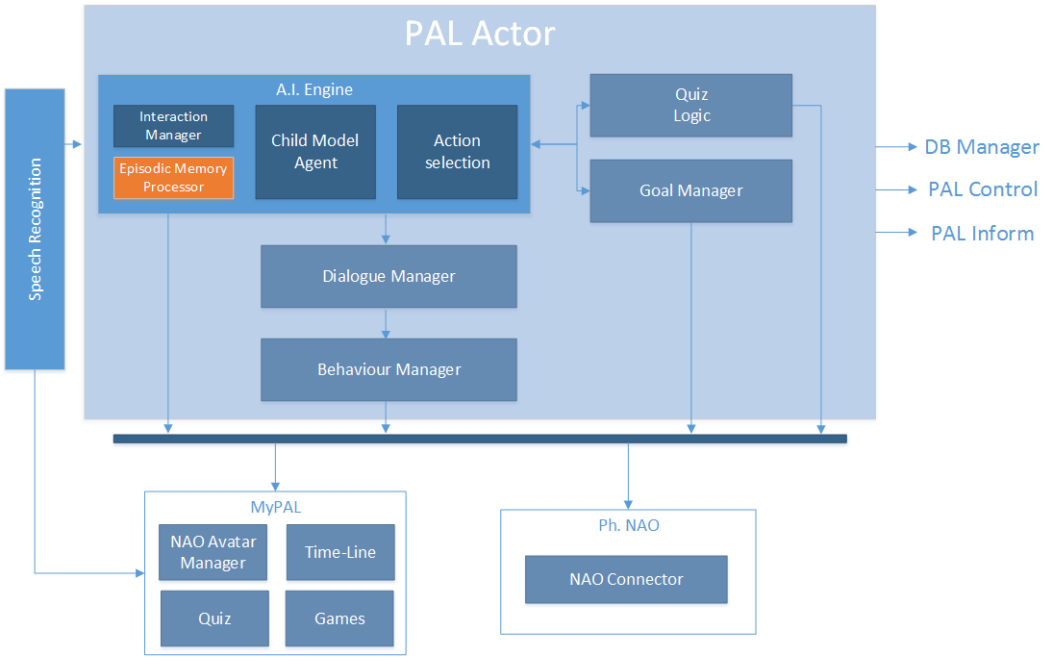


Figure 5.1. PAL overview

In figure 5.1, the architecture of the PAL system is shown. The Episodic Memory Module is added to this and marked orange. It is part of the A.I. Engine.

5.1.2 Functional viewpoint

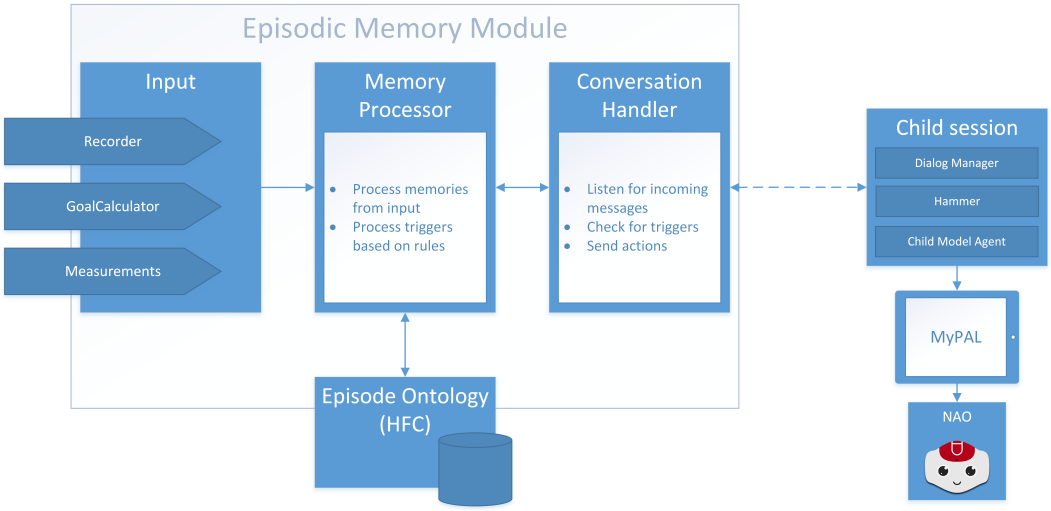


Figure 5.2. Overview of the Episodic Memory Module

The Episodic Memory Module consists of four parts: the input connectors, the memory processor, the episode ontology and the conversation handler. The input connectors provide the source for the memories. The memory processor gathers memories from these inputs and saves these into episodes (Episode ontology in the HFC database). Based on rules, each episode is tagged with possible triggers. As soon as a child session is live, the conversation handler listens to the session. It checks in the memory processor if there are memories available for specific triggers. When this is the case, the conversation handler forms an action and passes this on to the child session.

5.1.3 Ontology design

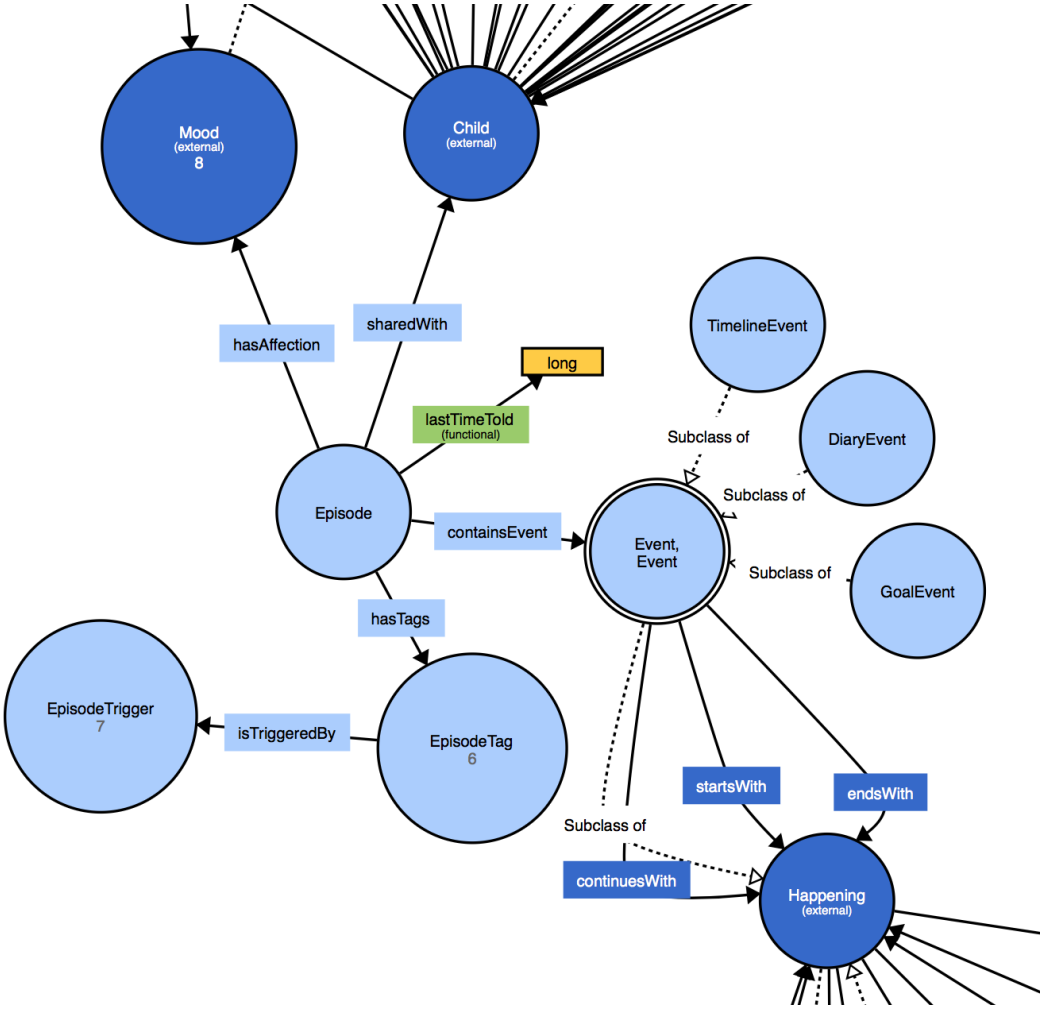


Figure 5.3. Visualization of episodic memory ontology, generated by WebVOWL

In figure 5.3, the relations in the episodic memory ontology are visualized. The Episode class is the main element that defines the memory. The concept of 5W1H (When, Where, Who, Why, What and How) is achieved using the relations. 'Who' is the child that has been mapped to the episode. 'When' is mapped in an Event, which is a subclass of the already existing ontology of a Happening. The 'What' is an Entity, that is also stored in the superclass Happening. The 'How' is mapped to the Happening in the form of an already existing class Activity, which involves a certain action like cooking, playing, or performing the quiz. The 'Why' was the most difficult part to map into an ontology. For now, it's part of the EpisodeTag. The EpisodeTag has several instances which defines the 'What' and implies the

'Why'. For example, the EpisodeTag contains an instance 'GoalWasTooDifficult'. The 'What' in the tag is a goal, but the specific instance of this goal is stored in the Event. In addition to the GoalWasTooDifficult, there are other instances like 'GoalAchieved' and 'GoalAlmostAchieved'. These EpisodeTags are important, as they are used to define what triggers them. This is done using the relation to EpisodeTrigger. A trigger is, for example, 'OnChildLogin' or 'QuizCompleted'. The relation between the Trigger and Tag is made to define when to mention a specific episode.
The current PAL ontology contains three different mappings of an emotion. They all have their pros and cons. To cope with this problem, they will soon be replaced by a new ontology of an emotional state, which would become the only mapping of an emotion in the PAL domain. The emotion is currently mapped as the already existing class 'Mood', but it is in fact a placeholder.

5.1.4 Reasoning mechanism

Dialog generation

The dialogs of the robot are being generated in the dialog manager project within the PAL project. This subproject is managed by the PAL project partner DFKI (Deutsche Forschungszentrum für Künstliche Intelligenz GmbH). In order to refer to episodic memories, a dialog for the robot should be written. DFKI developed a language for defining these dialogs, called Content Planner (CPlan) [23]. It uses a set of rules which all have a match part and an action part. The rule applies if the first part matches the input (also called SpeechAct). The action part is then executed which forms the actual dialog.

Rule

```
//SuggestPlayQuiz
:dvp ^ <SpeechAct>suggestion ^ <Content><About>Activity ^ <what>quiz
->
# ^ :canned ^ <stringOutput>concatenate("Shall we ", random("play", "do", "check out", "start")," the
quiz ", random("today", "soon", "now", ""), random(" in order to practice a bit extra ", " ", " so we
can continue our work on the goals "))
        ^ <SpeechModus>interrogative.
```

Action

Figure 5.4. CPlan example

As seen in the example dialog in figure 5.4, the first part defines the rule and the second part the action. The SpeechAct is a suggestion, and the content is about an activity. The activity in question is a quiz. This implies that the robot wants to suggest that the user does the activity 'quiz'. The action part forms a string which will become the spoken dialog from the robot. The syntax makes it possible to generate a variety of dialogs from one action. Two possible dialogs in the example are: 'Shall we play the quiz soon?' or 'Shall we check out the quiz today so we can continue our work on the goals?'.
The rule part can also contain variables which are usable in the action part. This is,

for example, used in the mentioning of specific goals. It could, for example, create a sentence: 'We finished the goal counting carbs yesterday.' in which 'counting carbs' is a variable associated with a specific goal.
These variables could also help to implement the dynamic aspects of the human memory. Forgetfulness could be simulated by creating very specific dialogs and gradually less specific dialogs. It would be possible to have a dialog as specific as 'The goal correct a hypo was a bit difficult yesterday in the quiz.', but also as general as 'Previous time, the quiz was quite difficult.'. Also, memory combining could be possible by having multiple subjects in the SpeechAct. It could, for example, generate a dialog that mentions that the goal was difficult but, after lots of practice, it is almost achieved. All possible sentences that are implemented from the scope of iteration 1 are defined in figure 5.1.4.

What	**Variations**	**Example sentence**
Goals updated	5	It looks like you have three new goals that we can work on! I see that you got some new goals!
Goal almost achieved	3	The previous time, we almost achieved the goal counting carbs! The goal counting carbs is almost done! Just a tiny bit of work left and we are able to finish the goal counting carbs!
Goal completed	2	We finished the goal counting carbs the previous time. Previous time you achieved the goal counting carbs!
Goal was too difficult	2	The goal counting carbs was a bit difficult the previous time. Previous time that we practiced the goal counting carbs, it was a bit challenging.
Suggest play quiz	48	Shall we start with the quiz? Shall we do the quiz soon in order to practice a bit extra? Shall we check out the quiz now so we can continue our work on the goals?

Table 5.1. Sentences currently possible in CPlan

Reasoning flow

Two different flows for reasoning about episodes have been identified:

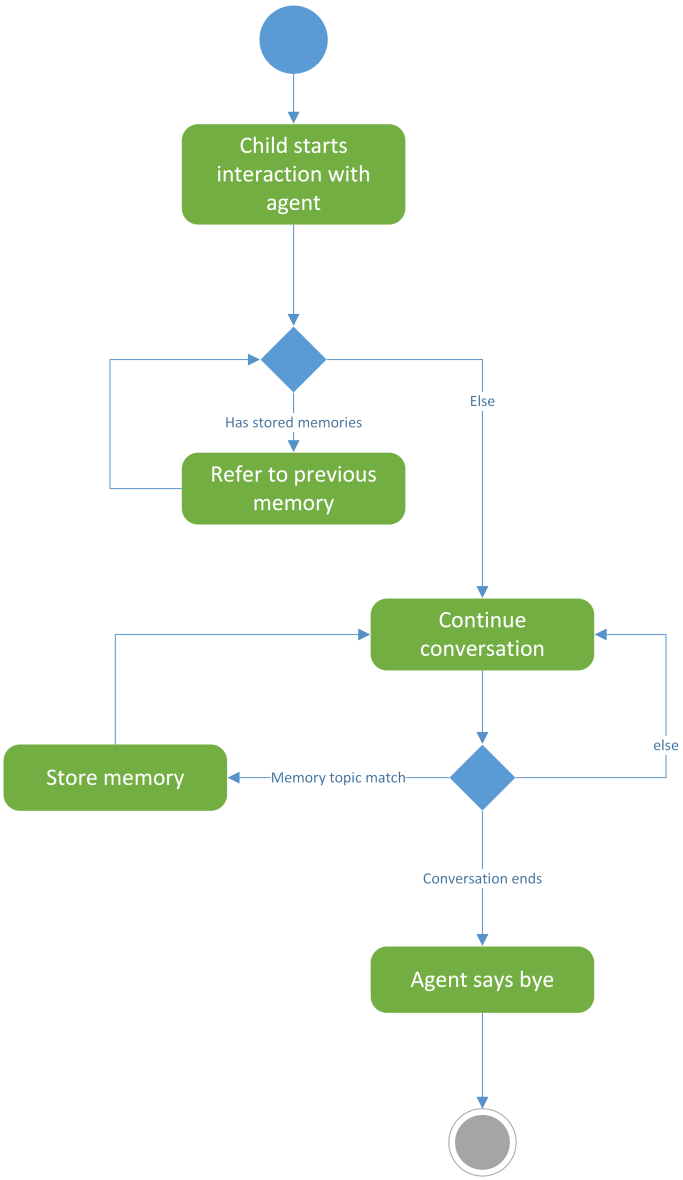


Figure 5.5. Reasoning flow 1: refer at the beginning

The first flow involves one fixed moment at the beginning of a conversation to refer to a previous memory. The second one is more dynamic and also involves the retrieval of episodic memory during the conversation.

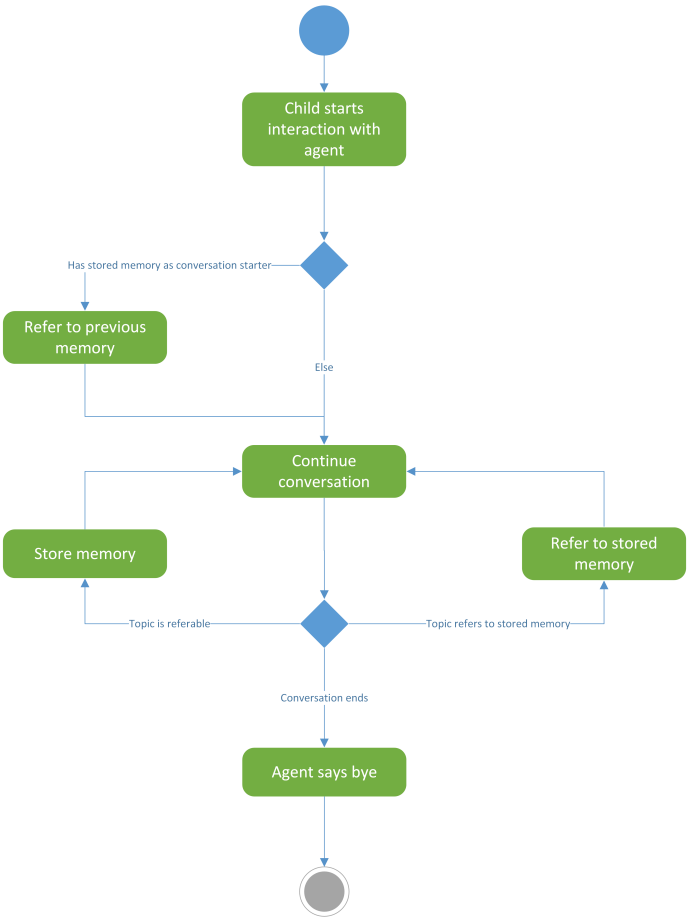


Figure 5.6. Reasoning flow 2: refer at the beginning and during conversation

The second flow would be preferable for iteration 2 and 3 as these contain feedback that is given during a conversation as seen in the scenarios (§ 4.1). The first flow would be sufficient for iteration 1, as referring to a goal or achievement is only done at the beginning of a session. Therefore, the first flow will be implemented in iteration 1 and can be extended to the second flow in iteration 2.

The moment to mention the episode is defined in the ontology using the EpisodeTrigger. In iteration 1, only the trigger 'OnChildLogin' is used which complies to reasoning flow 1. Each trigger has to be implemented in the right part of the dialog flow within the dialog manager. The part that handles the child login should therefore check for possible episodes to refer to which can be triggered using the 'OnChildLogin' trigger. This will generate a dialog which mentions the episode. Each mention of an episode updates the lastTimeTold value in the ontology. This prevents the robot from mentioning the same episode twice.

5.1.5 Artifact

The final artifact that was created for the experiment contained the scope of iteration 1. It was able to capture memories and refer to them according to reasoning flow 1. All requirements from iteration 1, except for requirement G02B and G04 (both with lower priority), were successfully implemented (see Table 4.4). In figure 5.7 below, the robot refers to an almost achieved goal and gives the suggestion to play the quiz afterwards.

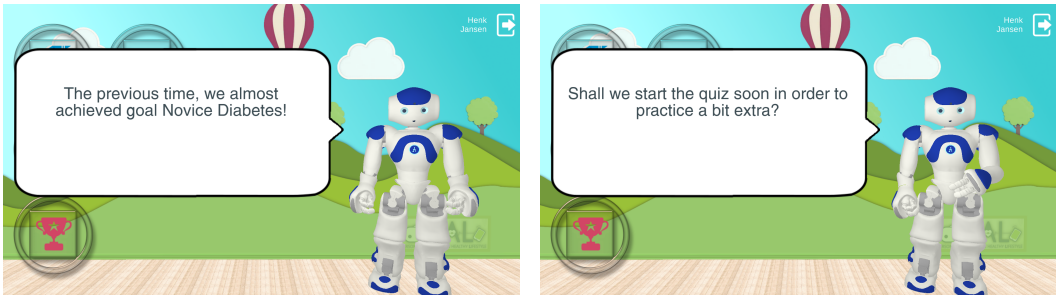


Figure 5.7. Charlie refers to an almost achieved goal

5.2 Experiment design

As this research is a small part of the bigger PAL project, the execution of the experiment will be out of the scope of this thesis. This is organized by other project members. The processing of the results that are relevant for this thesis will of course be within the scope. For completeness, the experiment is described briefly in this section. The full experiment protocol is added as an appendix (see Appendix C).

5.2.1 Participants

The participants will be children with T1DM (age 8 - 12), recruited from the hospitals or associations of the PAL consortium (Meander MC and Stichting ziekenhuis Gelderse Vallei -NL- and Ospedale San Raffaele -IT-) with a minimum number of 10 to a maximum number of 15 subjects per hospital. Thus, a total pool of 30 to 45 subjects will be ensured. The parents are also involved in the research.

5.2.2 Ethics

As this experiment involves the use of a humanoid robot in a medical context with children, the ethical considerations are important. This section will only highlight some aspects of the considerations as the full protocol is added in Appendix C. The PAL actor is made to be strictly educational and it will not manage the diabetes in any way. Therefore, it will not give suggestions on, for example, the insulin dosages or the estimated carbohydrates. The consent form has to be signed by both parents

or a legal guardian in the case of parental separation. Data will be in a totally anonymous form within publications or scientific conferences. The data from the Dutch participants is stored at TNO in the Netherlands in a secure environment. Data can be deleted any time at the request of the parents of the participating child. The protocol was approved by the ethical committee of the European Commission.

5.2.3 Materials

Every child will receive a tablet with the MyPAL app, containing the virtual robot. Every hospital will have a physical NAO robot. As well as the NAO robot, every hospital will have access to a tablet on a rotating stand for the quiz and a touchscreen table for the games.

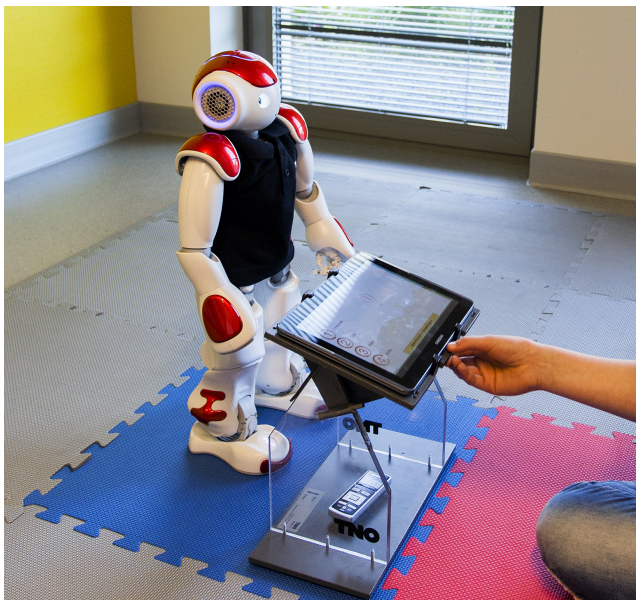


Figure 5.8. PAL robot and rotating tablet in Meander MC. Photographed by Rifca Peters in 2016

5.2.4 Procedure

The experiment will take place in three hospitals. The two hospitals in the Netherlands will organize the experiment from the end of May 2017 untill the end of August 2017. The experiment at the hospital in Italy will take place from the end of June untill the end of September.

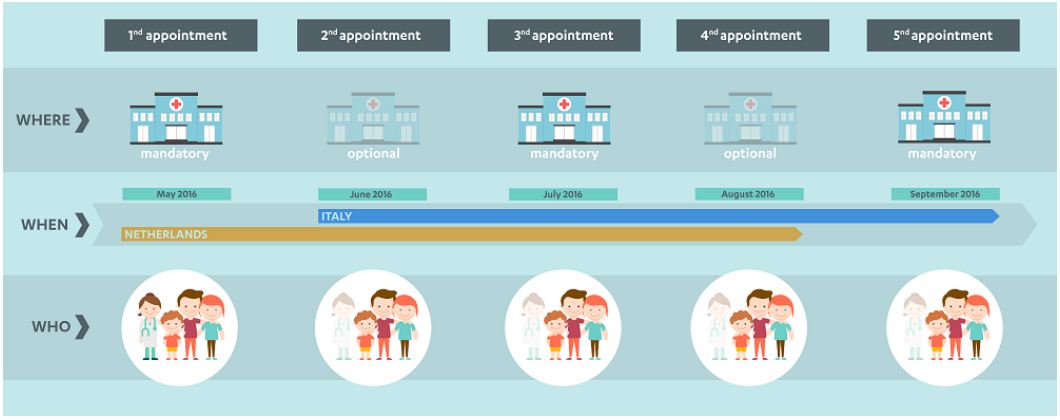


Figure 5.9. Experiment timeline (see Appendix C)

Every child will use the PAL system for approximately four months. In these four months, four appointments at the hospital are scheduled. During these appointments, the children can play with the physical NAO robot. Only the first and third appointment have a mandatory attendance in the hospital. The second and third appointment contain changes to the PAL system.

Appointment	**PAL improvements**
T1	Initial version
T2	Major update for diary in the MyPAL app
T3	Episodic memory update
T4	No changes

Table 5.2. PAL improvements during appointments

The episodic memory update will contain at least the first iteration of the episodic memory module. That includes memorizing as well as referring to achieved goals and related progress. The second and third are considered optional and can be implemented after this experiment.

The children as well as the parents are asked to fill in an EMA (Ecological Momentary Assessment) survey on a weekly basis. All questions have an answer option with a continuous Likert scale, which is stored as a score between 1 and 9. The survey questions for the children are shown in Table 5.3.

Claim	EMA question
Well-being	1. How do you feel right now?
Affection	2. Do you see the robot as a good friend?
Similarity	3. Is the robot in the app the same as the robot you have met in the hospital?
Motivation	4. Are you looking forward to play with the robot right now?
Diabetes Self-Management behaviour	5. Does the robot help you well in diabetes activities?
Diabetes knowledge	6. Does the robot teach you a lot about diabetes?

Table 5.3. Survey questions for children

The questions for the parents are stated in Table 5.4. The survey for parents also has an additional comment field in contrast to the survey for children.

Claim	EMA question
Stress	1. Do you often worry about your child?
Disclosure	2. How often does your child talk about diabetes?
Diabetes Knowledge	3. How much does your child know about diabetes?
Diabetes Self-Management Behaviour	4. How many diabetes activities can your child do him/her-self right now? (for example count carbs or prick him/her-self)
Motivation	5. Is your child motivated to play with the robot app?

Table 5.4. Survey questions for parents

Not all survey questions are relevant for the research in this thesis, but are meant for other researchers within the PAL consortium. In order to answer the research questions and confirm the claims, the questions regarding affection, motivation and diabetes self-management behaviour are included in the evaluation.

5.2.5 Data collection

The data will be collected from three sources. Usage data will be extracted from the HFC database in the PAL system. A method to retroactively capture episodes was

developed by checking the episode conditions on each database insert from the past. Therefore it was possible to measure the number of episodes that could have been made before its implementation. In addition to the PAL system, a weekly survey from both the parents and the children will be used. In Table 5.5, the relevant topics and sources are defined.

Topics	**Source**
Usage (sessions per child)	HFC database
Goals (progress and achievements per child)	HFC database
Episodes (stored and that have been referred to)	HFC database
Motivation and affection over time	Survey (parents and children)
Self-management behaviour	Survey (parents and children)

Table 5.5. Data collection

The topics and sources were used to form two different data files. The first one was ordered per participant per week. The second one was ordered per participant per session. In this way, it was possible to extract more detailed information regarding the usage of the system, but maintain one dataset with all sources combined. The variables are defined in Table 5.6.

Data	When	Description	Value
ParticipantId	T1	General identifier used to combine data from different sources	Text
Gender	T1	Gender of participant	Text
Age	T1	Age of participant	Number
Hospital	T1	The hospital that provides diabetes care for the participant	Text
Active	On change	1 = Active, 0 = Participant stopped	Number (0/1)
Child_q1	Weekly	Question regarding well-being	Number (1-9)
Child_q2	Weekly	Question regarding affection	Number (1-9)
Child_q3	Weekly	Question regarding similarity	Number (1-9)
Child_q4	Weekly	Question regarding motivation	Number (1-9)
Child_q5	Weekly	Question regarding diabetes self-management behaviour	Number (1-9)
Child_q6	Weekly	Question regarding diabetes knowledge	Number (1-9)
Parent_q1	Weekly	Question regarding stress	Number (1-9)
Parent_q2	Weekly	Question regarding disclosure	Number (1-9)
Parent_q3	Weekly	Question regarding diabetes knowledge	Number (1-9)
Parent_q4	Weekly	Question regarding diabetes self-management behaviour	Number (1-9)
Parent_q5	Weekly	Question regarding motivation	Number (1-9)
Parent comment	Weekly	Additional comments from parents	Text
TotalGoals	Weekly	Total number of goals set for the participant	Number
TotalAchieved	Weekly	Total number of goals that the participant achieved	Number
Sessions	Weekly	Number of sessions that a participant had	Number
QuizPlayed	Weekly	Number of sessions with at least one played quiz	Number
EpisodeMentioned	Weekly	Number of sessions with a mentioned episode	Number
EpisodeTag	Per session	Episode subject that was mentioned	Text
StartTime	Per session	Login time of a session	Timestamp
EndTime	Per session	Logout time of a session	Timestamp

Table 5.6. Variables

5.3 Experiment results

The gathered results in this chapter are only based on the data from the Dutch hospitals. Unfortunately, the Italian hospital data was delayed and therefore could not be included in this thesis research.

5.3.1 Participants

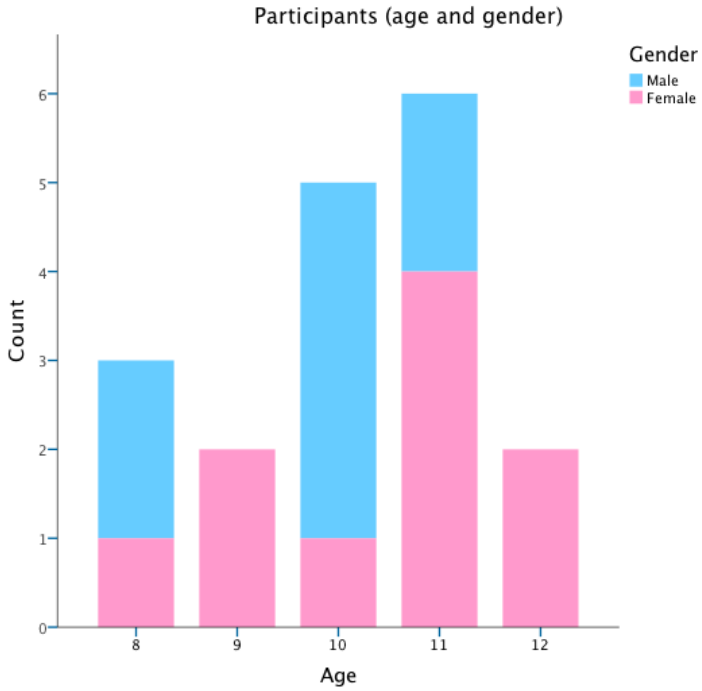


Figure 5.10. Participants: age and gender

As seen in figure 5.10, there were 18 participants in total in the Netherlands. Eight of them were boys and ten were girls.

5.3.2 Usage

The usage of the system over time is shown in figure 5.11. The usage was calculated using the cumulative child sessions that were sorted by date. Every successful child login attempt was considered to be a session. The first blue reference line marks the timeline update (T2) and the second reference line marks the implementation of the episodic memory (T3).

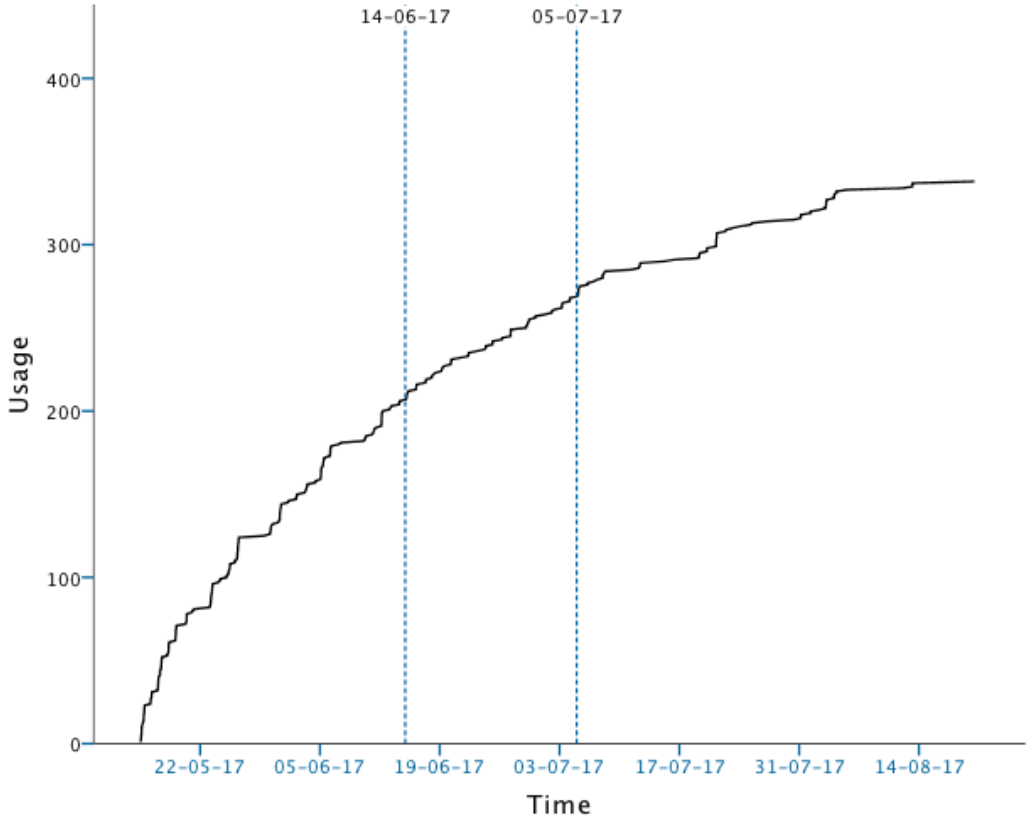


Figure 5.11. Cumulative sessions over time

In total, the children had 338 sessions with the PAL robot. The children played the quiz during 42.9% of the sessions before T3. After T3, 34.9% played a quiz during a session. The PAL actor referred to an episode in 23.8% of the cases after T3. All of these episodes contained the suggestion to play the quiz. The suggestion to play the quiz was accepted in 66.7% of the cases. This corresponds to 10 out of 15 cases.

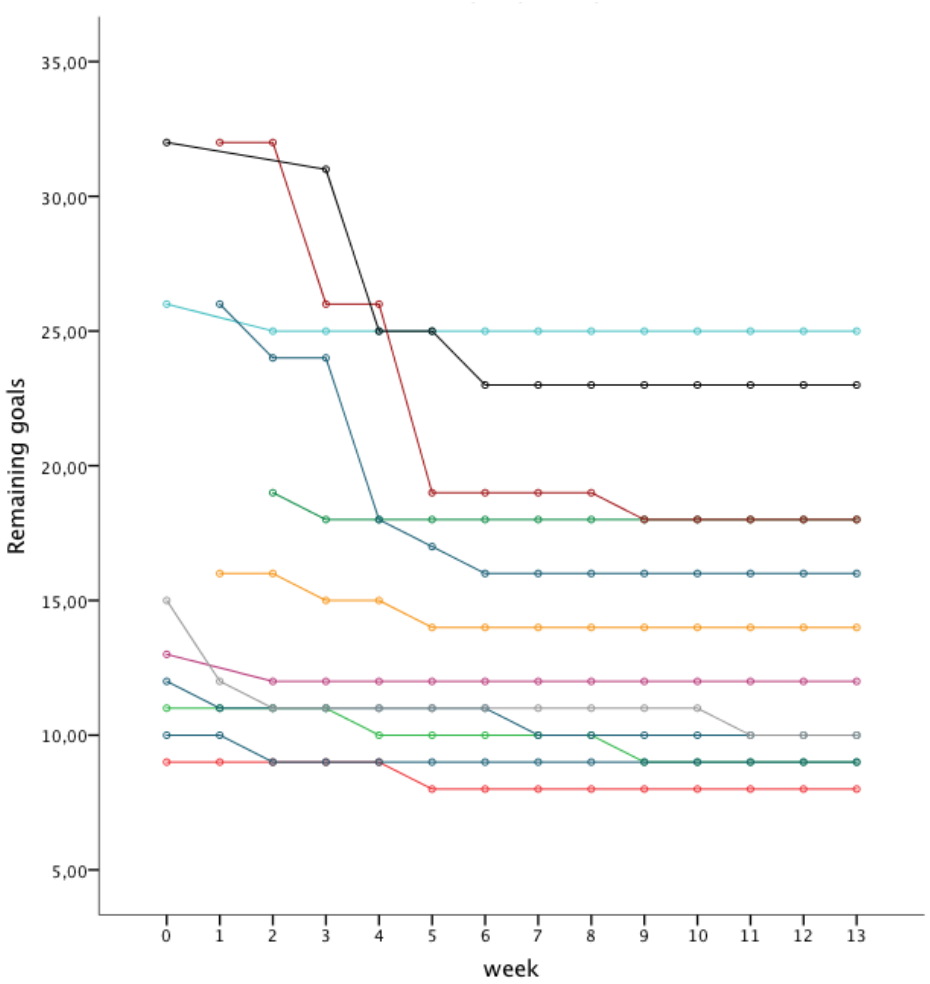


Figure 5.12. Remaining goals per participant

At the beginning of the experiment, the nurse in the hospital determined specific goals per child. The child was able to achieve these goals by performing certain tasks in the MyPal app (e.g. playing the quiz). In figure 5.12, the number of goals per child over time are shown. 6 of the 18 participants were filtered out of this chart, as they did not achieve any goal.

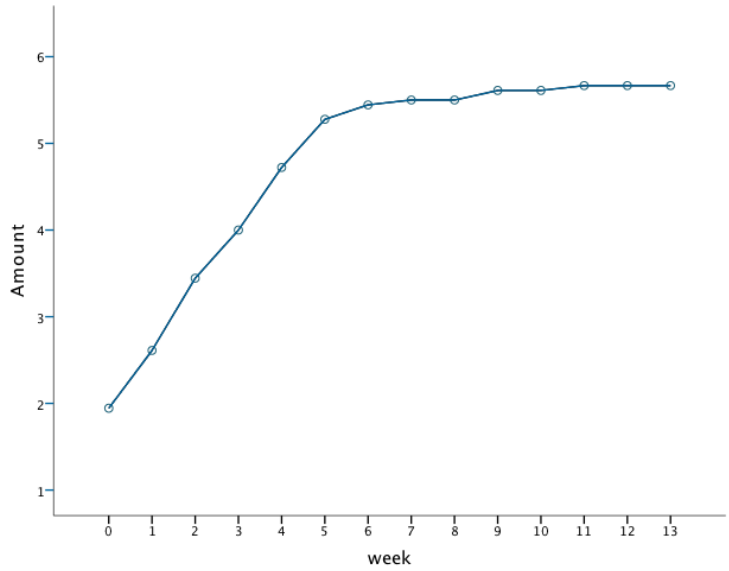


Figure 5.13. Mean achieved goals per participant

The mean achieved goals per child in figure 5.13 gives a better view of the decrease over time. A major decrease in the achieved goals per week was seen after the fifth week.

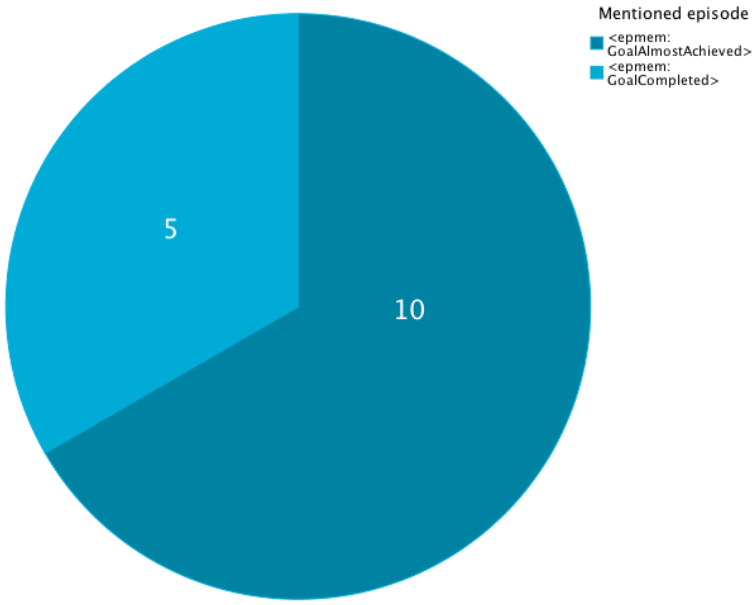


Figure 5.14. Mentioned episode types

Interestingly, as seen in figure 5.14, in the 15 cases of a mentioned episode, only two of the implemented types were captured. The goals were not updated in the

meantime which explains the fact that the corresponding episode type was never captured. Also, the quiz was too difficult type was never captured.

Correlations

		usage_episodeMentioned	usage_quizPlayed
usage_episodeMentioned	Pearson Correlation	1	,245**
	Sig. (2-tailed)		,000
	N	252	252
usage_quizPlayed	Pearson Correlation	,245**	1
	Sig. (2-tailed)	,000	
	N	252	252

**. Correlation is significant at the 0.01 level (2-tailed).

Table 5.7. Pearson correlation between mentioned episodes and quiz played

The mentioning of an episode and therefore the suggestion to play the quiz did seem to have a positive impact on the motivation to play the quiz according to Table 5.7. The Pearson correlation was, however, not very strong.

5.3.3 Affection

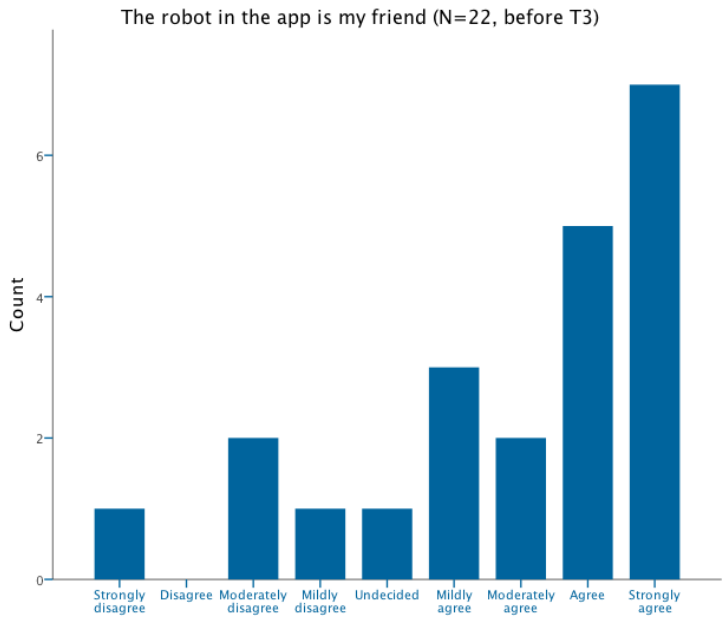


Figure 5.15. Self-assessed affection with robot before T3

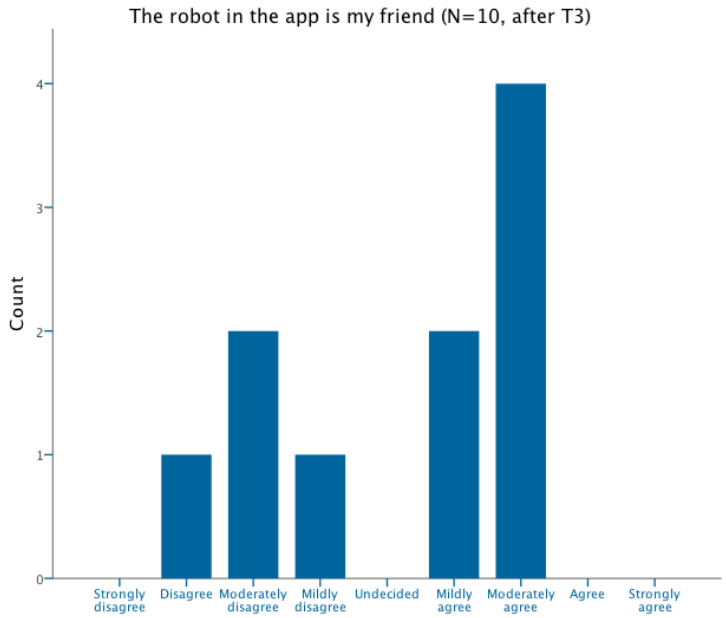


Figure 5.16. Self-assessed affection with robot after T3

The children filled in the survey 32 times. The affection level assessed by children is shown in figure 5.15 and figure 5.16. The median affection level of the children dropped from agree before T3 to mildly agree after T3. Unfortunately, the survey was only filled in once by a boy, which makes it impossible to explore the correlation between survey results and gender.

Correlations

		usage_episodeMentioned	The robot in the app is my friend
usage_episodeMentioned	Pearson Correlation	1	,275
	Sig. (2-tailed)		,270
	N	252	18
The robot in the app is my friend	Pearson Correlation	,275	1
	Sig. (2-tailed)	,270	
	N	18	18

Table 5.8. Pearson correlation between mentioned episodes and affection (children)

The survey of the parents did not contain a question regarding the affection with the PAL actor. Therefore, the affection was only measured by asking the children themselves. The Pearson correlation test (Table 5.8) did not show significance between the mentioning of episodes and the affection with the avatar.

5.3.4 Motivation

Not only the self-assessed affection, but also the self-assessed motivation dropped. The motivation level according to the children is visible in figure 5.17 and figure 5.18.

The median motivation level of the children dropped from moderately agree before T3 to mildly agree after T3.

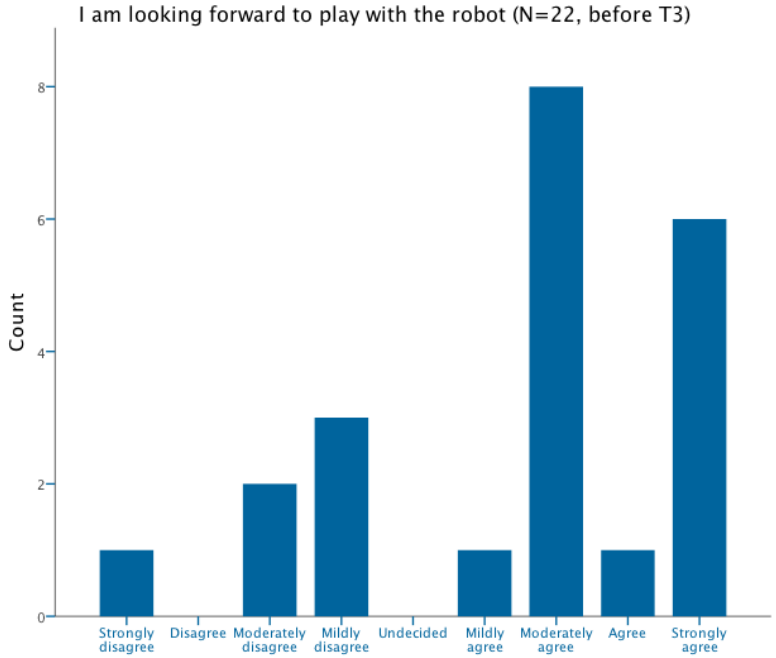


Figure 5.17. Self-assessed motivation of children before T3

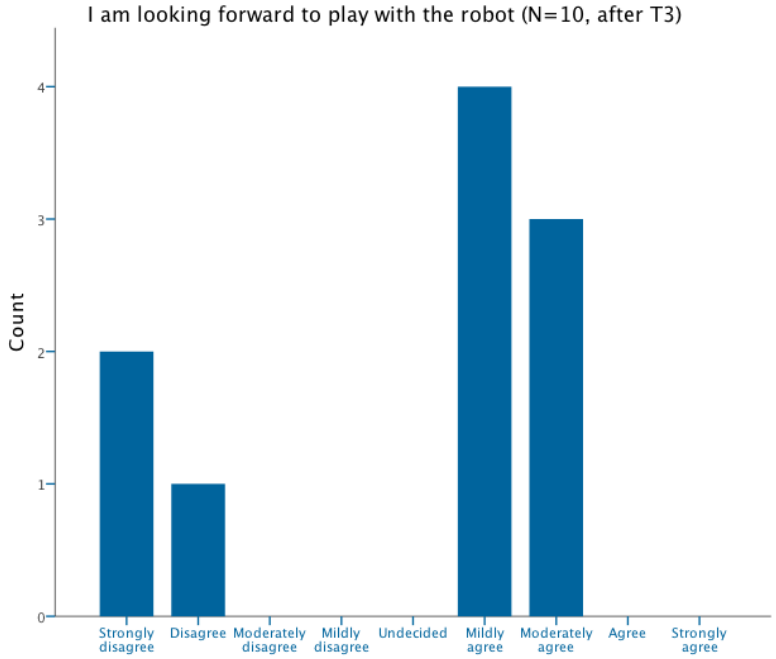


Figure 5.18. Self-assessed motivation of children after T3

The parents filled in the survey 69 times. The survey question regarding motivation for the parents is shown in figure 5.19 and figure 5.20. The median motivation level that was assessed by the parents dropped from undecided before T3 to moderately disagree after T3.

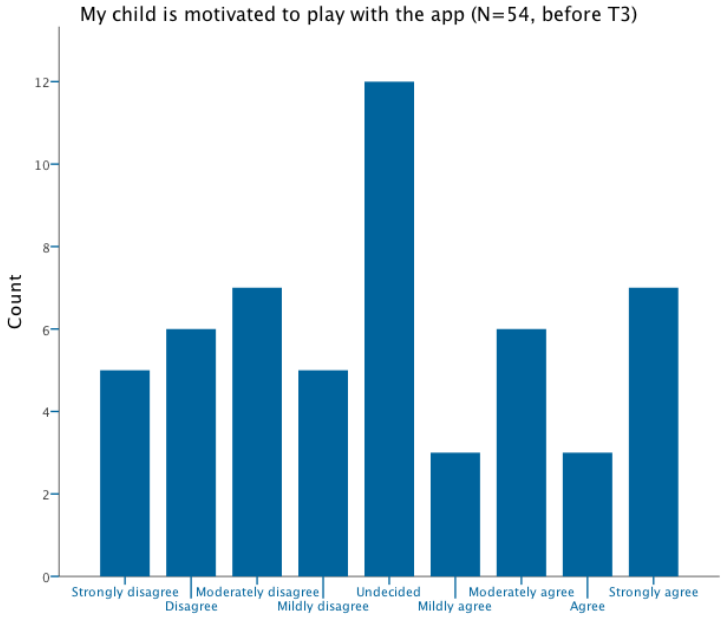


Figure 5.19. Motivation assessed by parents before T3

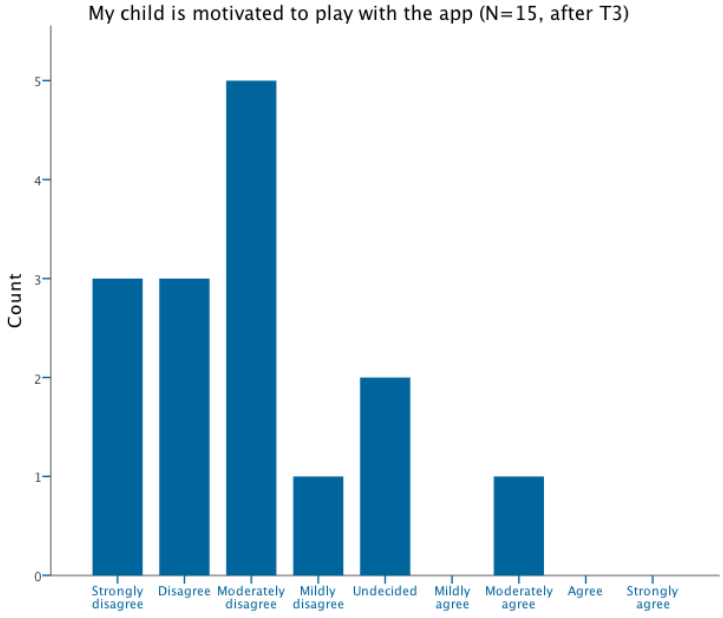


Figure 5.20. Motivation assessed by parents after T3

Correlations

		usage_episodeMentioned	I'm looking forward to play with the robot
usage_episodeMentioned	Pearson Correlation	1	,250
	Sig. (2-tailed)		,316
	N	252	18
I'm looking forward to play with the robot	Pearson Correlation	,250	1
	Sig. (2-tailed)	,316	
	N	18	18

Correlations

		usage_episodeMentioned	My child is motivated to play with the app
usage_episodeMentioned	Pearson Correlation	1	-,026
	Sig. (2-tailed)		,857
	N	252	51
My child is motivated to play with the app	Pearson Correlation	-,026	1
	Sig. (2-tailed)	,857	
	N	51	51

Table 5.9. Pearson correlation between mentioned episodes and motivation (left: children, right: parents)

The Pearson correlation test (Table 5.9) also did not show significant results. The mentioning of episodes did not lead to a significant change in motivation.

5.3.5 Diabetes self-management behaviour

The median diabetes self-management behaviour that was measured by the children (figure 5.21 and figure 5.22) stayed unchanged. The median before and after T3 was moderately agree.

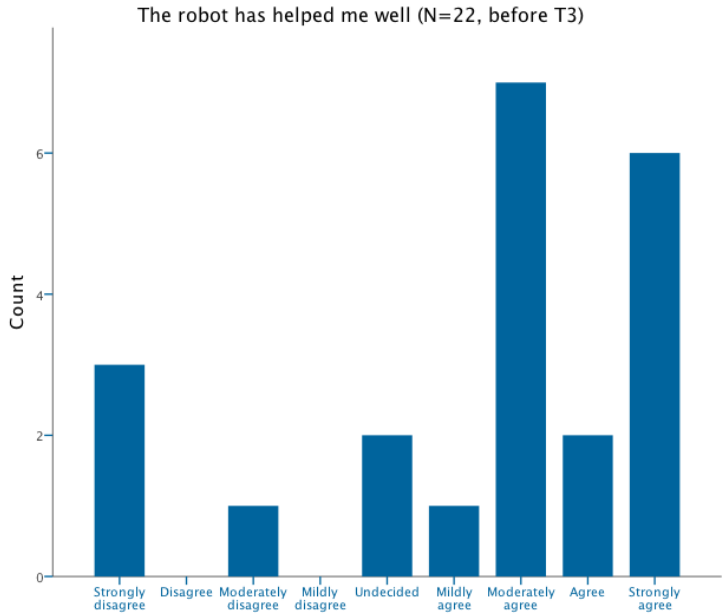


Figure 5.21. Self-assessed diabetes self-management behaviour of children before T3

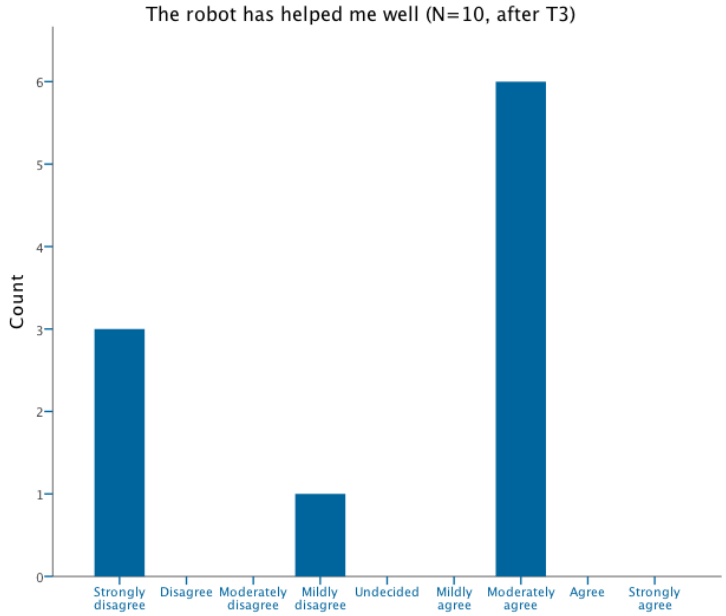


Figure 5.22. Self-assessed diabetes self-management behaviour of children after T3

The parents were also asked about the self-management behaviour of their children (figure 5.23 and figure 5.24). The median diabetes self-management behaviour level that was assessed by the parents dropped from undecided before T3 to mildly disagree after T3.

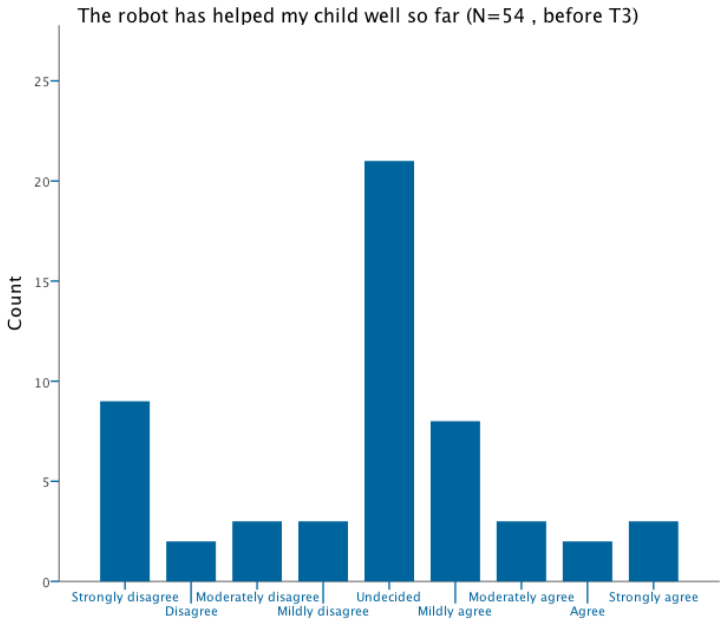


Figure 5.23. Diabetes self-management behaviour assessed by parents before T3

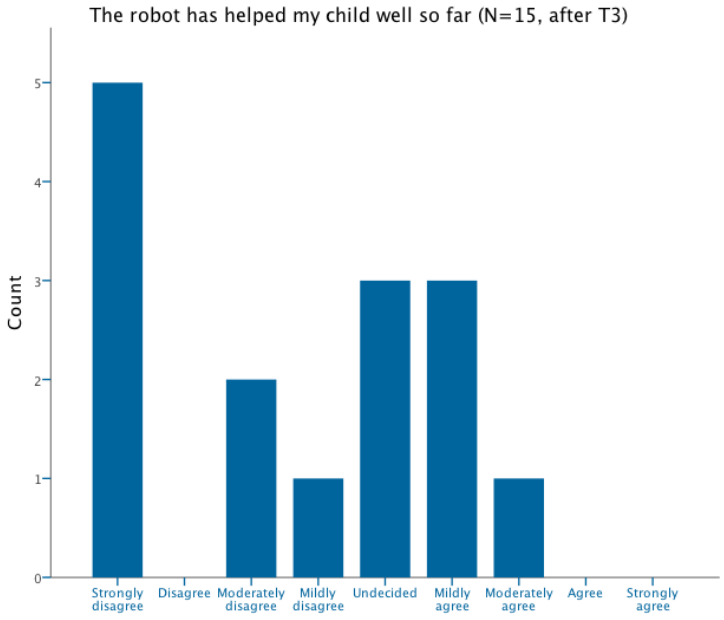


Figure 5.24. Diabetes self-management behaviour assessed by parents after T3

Correlations

		usage_episodeMentioned	The robot has helped me well
usage_episodeMentioned	Pearson Correlation	1	,268
	Sig. (2-tailed)		,282
	N	252	18
The robot has helped me well	Pearson Correlation	,268	1
	Sig. (2-tailed)	,282	
	N	18	18

Correlations

		usage_episodeMentioned	The robot has helped my child well so far
usage_episodeMentioned	Pearson Correlation	1	-,112
	Sig. (2-tailed)		,434
	N	252	51
The robot has helped my child well so far	Pearson Correlation	-,112	1
	Sig. (2-tailed)	,434	
	N	51	51

Table 5.10. Pearson correlation between mentioned episodes and self-management (left: children, right: parents)

According to the Pearson correlation test in Table 5.10, the mentioning of episodes did not result in any significant change in self-management behaviour, according to the children and parents.

Chapter 6

Discussion and conclusion

This chapter discusses the findings and contributions from this thesis, points out the limitations of the current work, and also provides directions for future research. It consists of the sections Discussion (§ 6.1), Limitations (§ 6.2), Future work (§ 6.3) and Conclusion (§ 6.4).

6.1 Discussion

In this section, each of the subquestions from § 1.4 will be revisited and discussed using the theoretical framework, the system design and the gathered results. Each subquestion adds up to the answer of the main research question as seen in figure 1.2.

SQ1: Which ontological model can capture the main concepts with their relations of the child-PAL shared episodic memory?
The ontology of an episodic memory as defined in this project (figure 5.3) was found suitable for storing memories about goals as defined in the first iteration. As both goals and measurements are part of the PAL ontology, the implementation of the second iteration using this ontology will not be a problem. The third iteration is, however, a bit tricky as it involves gathering episodes from an event that the child is talking about. Right now, the 'Why' in combination with a 'What' is stored in an EpisodeTag. It would be impossible to map all the possible cases that a child can talk about. The same applies to the 'How' which would require mapping all the actions in the ontology. This is however less of a problem than the EpisodeTag as an action is still defined atomically instead of as a combination of entities. The EpisodeTag should therefore be replaced by a separate 'What' and 'Why'. The reasoning using the EpisodeTrigger should check the precondition based on multiple relations instead of the single EpisodeTag. This would make it possible to have a goal that was too difficult, but also, for example, a test at school that was too difficult. In both cases the 'Why' is the same, as they were both too difficult. The moment to mention each episode could still be different, so just the 'What' or 'Why' would not be sufficient.

The current mapped Emotion is just a placeholder as a new ontology of the emotional state of a child is being created. As soon as this improved ontology is done, it should replace the mood in the episode ontology.

SQ2: Which reasoning mechanism can the PAL actor apply for constructive memory reference expressions?
The solution using the current dialog manager using CPlan was found suitable to reason about episodic memories. Two reason flows were identified from which the simple version (figure 5.5) was implemented. In order to proceed to the third iteration, the extended flow (figure 5.6) should be implemented which would require the implementation of other EpisodeTriggers in the dialog manager.
The recording of episodes is still a work in progress. The assumption that the stored episode is actually a shared experience with the child should be validated. Also, the rules when a goal is seen as almost finished or too difficult for the child would require evaluation.

SQ3: How could dynamic aspects from the human memory (e.g. forgetfulness, context association, memory combining and generalization) be translated to the PAL actor episodic memory?
The dialog manager is able to generate specific to gradually more general dialogs. This could simulate forgetfulness over time. The dialog manager is also able to combine memories using multiple episodes as inputs. The rules to perform these dynamic aspects are yet to be created. Context association was scheduled for iteration three, which would connect the episode to large knowledge databases. In this way, the robot could actually reply to memories that the child shared. The robot could, for example, connect and ask about the highlights of a certain city when the child mentions a day trip to that place.
Generalization was not taken into account as it is mostly used in a negative context. Generalization could also be dangerous when it is applied to diabetes factors like measurement values.

SQ4: Does the addition of episodic memory alter the affection with the embodied conversational agent?
The children rated their affection with the PAL actor considerably lower after the implementation of the episodic memory (figure 5.15 and figure 5.16). The usage of the system also gradually decreased over time (figure 5.11, figure 5.12 and figure 5.13). However, a significant correlation between mentioned episodes and affection was not found (Table 5.8). One explanation could be the fact that the experiment took place during the summer holidays. Some parents mentioned in the survey that they did not bring the tablet with them. Another explanation is the fact that only iteration 1 was implemented. The claim for iteration 3 is the increased affection with the avatar. The claim could still be proven after the implementation of this iteration.

SQ5: Does the addition of episodic memory improve the intrinsic motivation for achieving goals?
Unfortunately, the addition of episodic memory did not result in a decrease of the novelty effect according to the results. The parents as well as the children themselves rated their motivation lower (figure 5.15, figure 5.16, figure 5.19 and figure 5.20). The diabetes self-management behaviour stayed unchanged according to the children, but the parents rated it slightly lower (figure 5.21, figure 5.22, figure 5.23 and figure 5.24). However, a significant correlation between the mentioned episodes and motivation was not found (Table 5.9). This was also the case for the correlation between mentioned episodes and self-management behaviour (Table 5.10). The possible explanation here is the same as in SQ4. The usage gradually decreased over time, which could be the effect of the summer holidays.
The suggestion to play the quiz after a mentioned episode did seem to work. 66.7% of the children decided to play the quiz after the robot suggested to do so. However, the number of mentioned episodes was too low to prove this. Further research is needed in order to confirm this. Due to the limited number of cases, not all episodic memory types were covered and therefore the episodic memory update did not reach its full potential. It would be interesting to schedule a new experiment which is not in a holiday period. Also, the test results would be easier to validate using an A/B test in which one group of participants use the version with episodic memory and the other group uses the system without episodic memory. In this case, the novelty effect that was caused by the version without episodic memory in the weeks before T3 is prevented.

6.2 Limitations

The full potential of the episodic memory was not tested in this research due to different factors. First of all, the prototype was limited to feedback on goals and achievements. The claim regarding the increased affection with the PAL avatar would have been tested in iteration three.
The experiment design was mainly flawed due to the later introduction of the prototype while the experiment was ongoing. The experiment was designed this way by the PAL consortium with other researches in mind. The episodic memory was also not finished at the start of the experiment. It was not feasible to set up a completely separate experiment at a later stage for solely the episodic memory part as the medical focus of the project, combined with the fact that it involved children, requires approval from the ethical committee, which would take months.
The participants that stopped taking part before the introduction of episodic memory were not filtered out of my analysis. As the number of usages after the introduction of episodic memory were already limited, the complete analysis would be based a very small number of participants. Unfortunately, the results from the hospital in Italy were delayed, which would have been helpful in this case.

6.3 Future work

The new ontology that was built for capturing an episodic memory requires two changes in order to have it ready for the third iteration. The first important change is splitting up the EpisodeTag in a separate 'What' and 'Why'. The second change is the replacement of the mood ontology by the new emotional state ontology. This new sub-ontology is expected to be finished after the end of this thesis project.
The capturing rules that have been introduced in this prototype were not validated. The mentioned episodes by the PAL avatar could therefore not have been actual shared experiences between the PAL actor and the child. These rules should be validated before they can capture true shared experiences.
A new experiment should be designed as an A/B test. The prototype should also have the second and third iteration ready to be tested in this context. This would hopefully show the true potential of episodic memory.

6.4 Conclusion

The purpose of the current study was to design and evaluate personalized interactions with the children using episodic memory. The main aim of the developed episodic memory module for the PAL robot and its virtual avatar was to increase the overall engagement and usage of the health apps, which would ultimately lead to increased diabetes self-management.
This study has identified a suitable method for storing shared experiences between the child and the PAL actor, using the designed episodic memory ontology. Secondly, a reasoning mechanism was found suitable for referring to past episodes. Additionally, dynamic aspects from the human memory were identified and suggestions to translate those to the PAL actor were given. The results of the study did not prove that an increase in motivation, affection or diabetes self-management was due to the addition of episodic memory. The most important limitation lies in the delayed introduction of episodic memory while the novelty effect wear-off was already in progress. The study was also limited by the absence of the feedback about measurements and activities (iteration 2 and 3) in the prototype for the experiment. The delay of the results of the Italian hospital was unfortunate, which resulted in a small sample size.
Further studies need to be carried out in order to validate the capturing rules of episodic memory. This would make the capture of true shared experiences between the child and the PAL actor possible. It would be interesting to assess the effects of the second and third iteration in a working prototype. For this, some small changes are mandatory for the episodic memory ontology. Finally, a key policy priority should be that the experiment is carried out as an A/B test in order to mitigate the effects of the novelty effect wear-off before the introduction of the episodic memory module.

Bibliography

[1] American Diabetes Association, "Type 1 Diabetes," pp. 1–2, 2010. [Online]. Available: http://professional.diabetes.org/sites/professional.diabetes.org/files/media/type_1_1.pdf

[2] S. E. Wölfl, "The ALIZ-E Project : Adaptive Strategies for Sustainable Long-Term Social Interaction," *German Conference on Artificial Intelligence (KI-2012)*, pp. 39–41, 2012. [Online]. Available: https://www.dfki.de/KI2012/PosterDemoTrack/ki2012pd09.pdf

[3] R. Looije, "Pal for diabetic children | TNO," 2015. [Online]. Available: https://www.tno.nl/en/focus-areas/healthy-living/predictive-health-technologies/pal-for-diabetic-children/

[4] PAL4U, "About - PAL." [Online]. Available: http://www.pal4u.eu/index.php/project/about/

[5] Z. Kasap and N. Magnenat-Thalmann, "Building long-term relationships with virtual and robotic characters: The role of remembering," *Visual Computer*, vol. 28, no. 1, pp. 87–97, 2012. doi: 10.1007/s00371-011-0630-7

[6] TU Delft and TNO, "Applying the Situated Cognitive Engineering Method - A Comprehensive Guide," Tech. Rep. 1, 2012. [Online]. Available: http://scetool.nl/sce_manual_v1.0.pdf

[7] R. M. Ryan and E. L. Deci, "Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being." *The American psychologist*, vol. 55, no. 1, pp. 68–78, 2000. doi: 10.1037/0003-066X.55.1.68

[8] E. Locke, M. Saari, N. Shaw, and G. Latham, "Goal Setting and task performance," *Psychological Bulletin*, vol. 30, no. I, p. 1992, 1981. doi: 10.1037//0033-2909.90.1.125

[9] L. J. Becker, "Joint Effect of Feedback and Goal Setting on Performance: A Field Study of Residential Energy Conservation," *Journal of Applied Psychology*, vol. 63, no. 4, pp. 428–433, 1978. doi: http://dx.doi.org/10.1037/0021-9010.63.4.428

[10] R. Hoekstra and IOS Press., *Ontology representation : design patterns and ontologies that make sense*. IOS Press, 2009. ISBN 9781607500131

[11] World Wide Web Consortium (W3C), "OWL - Semantic Web Standards," 212. [Online]. Available: https://www.w3.org/OWL/

[12] H.-U. Krieger, R. Peters, B. Kiefer, M. A. van Bekkum, F. Kaptein, and M. A. Neerincx, "The Federated Ontology of the PAL Project. Interfacing Ontologies and Integrating Time-Dependent Data," in *8th International Joint Conference on Knowledge Engineering and Ontology Development*, INSTICC. SCITEPRESS, 2016. doi: https://doi.org/10.5281/zenodo.166692. [Online]. Available: http://www.dfki.de/web/forschung/iwi/publikationen/renameFileForDownload?filename=palonto.pdf&file_id=uploads_2973

[13] E. Tulving and W. Donaldson, "Organization of memory." pp. 381–403, 1972. [Online]. Available: http://psycnet.apa.org/psycinfo/1973-08477-000

[14] E. Tulving, "Episodic Memory: From Mind to Brain," *Annual Review of Psychology*, vol. 53, no. 1, pp. 1–25, feb 2002. doi: 10.1146/annurev.psych.53.100901.135114. [Online]. Available: http://www.annualreviews.org/doi/10.1146/annurev.psych.53.100901.135114

[15] A. Nuxoll and J. Laird, "Enhancing intelligent agents with episodic memory Action editor : Vasant Honavar," *Cognitive Systems Research*, vol. 17-18, pp. 34–48, 2012. doi: 10.1016/j.cogsys.2011.10.002. [Online]. Available: http://dx.doi.org/10.1016/j.cogsys.2011.10.002

[16] G. H. Lim, S. W. Hong, I. Lee, I. H. Suh, and M. Beetz, "Robot recommender system using affection-based episode ontology for personalization," *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, pp. 155–160, 2013. doi: 10.1109/ROMAN.2013.6628437

[17] G. Lim, I. Suh, and H. Suh, "Ontology-based unified robot knowledge for service robots in indoor environments," *Systems, Man, and Cybernetics-Part A ...*, 2011. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/5605259/

[18] S. Han, K. Lee, D. Lee, and G. G. Lee, "Counseling Dialog System with 5W1H Extraction," *Proceedings of the SIGDIAL2013 Conference*, no. August, pp. 349–353, 2013.

[19] J. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, vol. 8, no. June, pp. 282–289, 2001. doi: 10.1038/nprot.2006.61. [Online]. Available: http://repository.upenn.edu/cis_papers/159/%5Cnhttp://dl.acm.org/citation.cfm?id=655813

[20] W. R. Miller and S. Rollnick, *Motivational interviewing: Helping people change*. The Guilford Press, 2012. ISBN 1609182278. [Online]. Available: https://books.google.nl/books?hl=nl&lr=&id=o1-ZpM7QqVQC&oi=fnd&pg=PP1&dq=motivational+interviewing+rollnick+miller&ots=c0AhaQkhI_&sig=sacdDHfltb1uP___5jMOtlN-kSPs

[21] D. M. Hausman and B. Welch, "Debate: To nudge or not to nudge," *Journal of Political Philosophy*, vol. 18, no. 1, pp. 123–136, 2010. doi: 10.1111/j.1467-9760.2009.00351.x

[22] F. V. Burger, "Agents Sharing Secrets: Self-Disclosure in Long-Term Child-Avatar Interaction," Ph.D. dissertation, Radboud University Nijmegen, 2016.

[23] B. Kiefer, "The Talking Robots Toolkit: Documentation for the content planning module," 2012. [Online]. Available: http://talkingrobots.dfki.de/cms/wp-content/uploads/2012/01/tarot-doc.cplan_.pdf

Appendix A

Minor Thesis EIT: Commercializing E-Health apps

KTH Information and
Communication Technology

Commercializing eHealth apps without ethical consequences

Development of a business case without compromising the users data

BART SCHREUDER GOEDHEIJT

Minor Thesis for the EIT Innovation and Entrepreneurship module at KTH
Supervisors: Drs. R. Looije (TNO) & S. Temiz (KTH)

Abstract

Preventive apps in the eHealth domain are often struggeling with the switch from temporary finances to permanent finances. This research was intended to help eHealth apps to be launched commercially without ethical consequences for the users. The PAL project was used as a case to investigate possible ways to enter the Dutch healthcare sector. As a result, two different strategies were formed after a stakeholder analysis.

Contents

List of Acronyms and Abbreviations

PAL	Personal Assistant for a healthy Lifestyle
T1DM	Type 1 Diabetes Mellitus
UDID	Unique Device Identifier

Chapter 1

Introduction

1.1 Background

The PAL (Personal Assistant for a healthy Lifestyle) project aims to help children with diabetes type 1 (between the age of 7 and 14) to improve their self-management skills using a social robot and a virtual avatar combined with a set of mobile health apps. The robot and avatar act as a pal for these children and help them to achieve diabetes-oriented goals. Next to the robot and the virtual avatar, the PAL system consists out of a module for the health professionals used to instruct and supervise the system, as well as a module to monitor progress and inform the parents. The project is funded by the European Union in the Horizon2020 program (ref. H2020-PHC-643783). It continues work that has been done in earlier projects including the ALIZ-e project [Wölfl, 2012]. The PAL project is coordinated by TNO and includes partners from The Netherlands, Italy, the United Kingdom and Germany [Looije, 2015].

Figure 1.1. PAL robot and mobile health apps [PAL4U, 2015]

1.2 Problem statement

Preventive apps in the eHealth domain are often struggeling with the switch from temporary finances (e.g. subsidy) to permanent finances [Politiek and Hoogendijk, 2014, p. 285].

The permanent finances are often offered by the healthcare industry (pharmaceutical companies, insurance companies, etc.). One of the business developers at TNO mentioned that in order to get funding, it is nowadays quite normal to hand over the data of the users to these companies. He even mentioned that there is no proper business case for eHealth apps without doing so. This also applies to the PAL project.

1.3 Purpose & Goal

The PAL project contains an app for children with diabetes. This app can be seen as a diary which will contain their emotional state, the activities that they did during the day and the measurements of their blood sugar level. In order to make the app accessible, the end customer should be able to download the app for free. The purpose of this project is to find a fitting business model in order to launch the platform commercially that won't compromise the data of the users.

1.4 Research questions and hypotheses

The main research question is:

Can eHealth apps be launched commercially 'for free' without ethical consequences?

In order to answer the main research question, the following subquestions have been defined:

ID	Question
SQ1	Which business models are nowadays often used in eHealth apps?
SQ2	Which data gathered in eHealth apps is valuable without privacy issues?
SQ3	What would be a valid way of funding the solution that won't include selling data of the users?

Table 1.1. Research subquestions

Chapter 2

Literature analysis

2.1 Market analysis

According to Research2Guidance, nearly 100.000 eHealth apps have been added since the beginning of last year. That makes the total number of eHealth apps currently available in major apps stores 259.000 [Research2Guidance, 2015].
As seen in figure 2.1, most eHealth publishers don't use traditional app store revenue sources, such as paid apps, in-app advertisements or in-app purchases. Just 24% of the eHealth apps use these sources. Licensing, app development fees and service sales are the top three most common revenue sources.

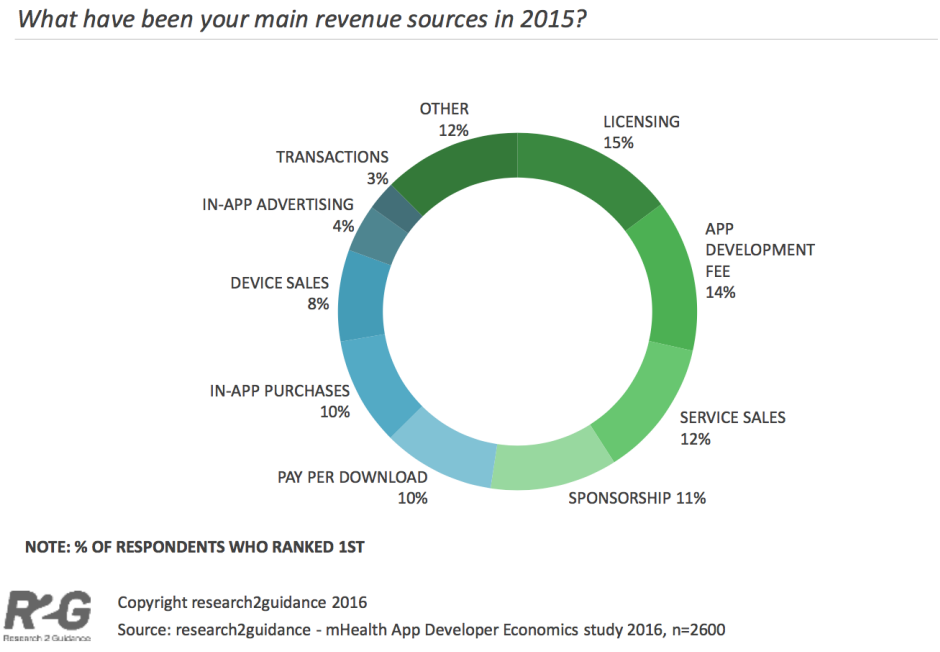


Figure 2.1. Common revenue streams for eHealth apps [Research2Guidance, 2015]

According to 82% of the eHealth practitioners, there could be a key role to play for Health Insurance companies in eHealth app publishing. The industry even expects that health insurance members would be willing to use the apps provided to them by Insurance companies, as well as share their data. "In return, members expect cheaper insurance plans (53%), receive health recommendations (18%) or support research (14%). Only 15% of eHealth market players do not foresee that members would share their data with Health Insurance companies via apps at all."

Research2Guidance further expects that in 5 years from now, eHealth can play a very imporant role in reducing hospital length of stay and reducing readmission costs. They also expect that diabetes will remain the number one chronic disease that offers the best business potential for eHealth app publishers.

2.2 Privacy and laws

Privacy is very important in the eHealth industry and it is a well-discussed topic in ethics. An insurance company could for example use certain information in someone's disadvantage. If the eHealth data shows that the person has an unhealthy lifestyle, it could increase the monthly fee. Big data can help pharmaceutic companies to develop new types of medication, but they can also use it to create a monopoly position using the knowledge that was gained from the data. This could lead to higher medication costs. These examples stress that privacy should have a high priority for eHealth products. In this case, the health related data is from children, which makes it even more crucial to preserve their privacy.
According to a study from Privacy Rights Clearinghouse, many health and fitness applications collect a great deal of personal information [Privacy Rights Clearinghouse, 2014]. Also, mobile applications, especially the free apps, depend on advertising to make money. "They may share personally identifiable information with advertisers, or allow ad networks to track you. Almost all applications send non-personal data about how you use an application to data analytics services. If an application collects your universal device ID (UDID) or embeds a unique ID in the application you download, analytics data can be tracked back to you personally." They found that 55% of paid and 60% of free apps that were investigated use third-party analytics services.

Chapter 3

Case analysis

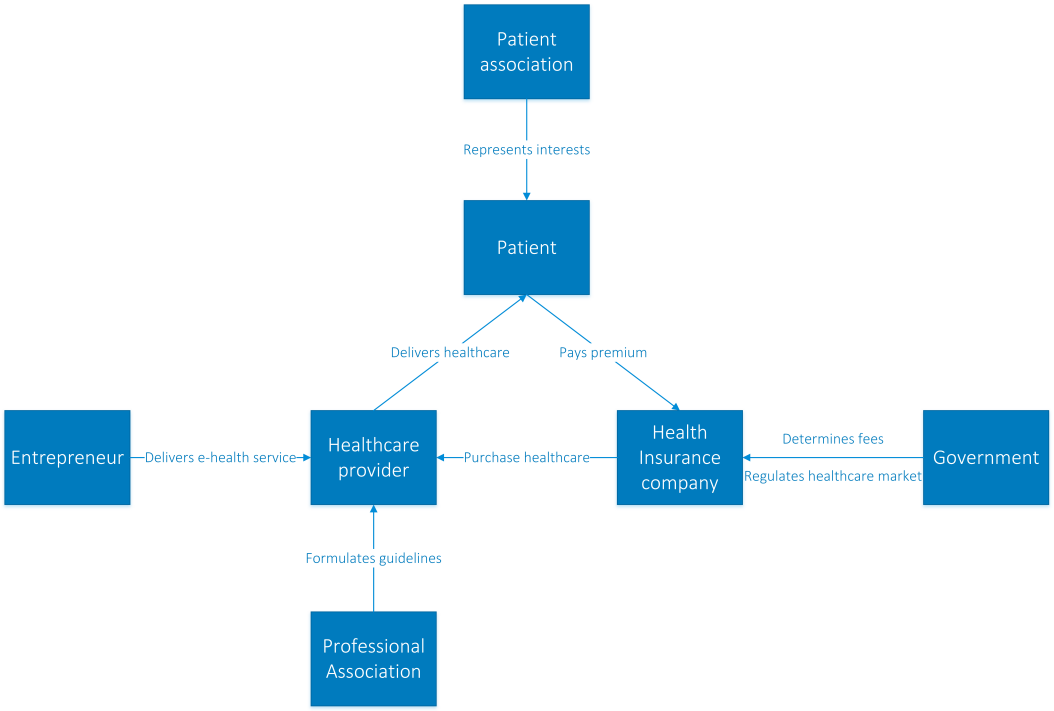


Figure 3.1. Innovation routes in the Dutch healthcare system [Hettinga, 2013]

The case will be analyzed using the innovation routes method from Windesheim [Hettinga, 2013]. This method is applied to the Dutch healthcare system.

3.1 Stakeholders

3.1.1 Patient

The patient is playing the central role in this case. The patients are children with diabetes type 1 (between the age of 7 and 14). Important in this case is that the parents or caregivers are involved as they can influence the usage and effectiveness of the solution. The patients main benefit for using the solution is the stimulation of their self-management skills in diabetes. The child is not able to choose the solution by themselves due to their age. The parents or caregivers need to be convinced in order to gain an user base.

3.1.2 Insurance company

The insurance company is responsible for covering the costs of the care that is given to patients. This party therefore has a big influence, as they want to provide good care for a low price. Insurance companies have different departments with interests in the eHealth market. The innovation department selects and ranks eHealth innovations, while the investment fund supports these innovations. The purchasing department negotiates with companies providing healthcare and purchashes this in big quantities in a preferably efficient way. The role of eHealth services in these negotiations is therefore limited. The commercial department on the other hand assembles the insurance packages for customers and organizations and sees the eHealth as a distinctive feature. The departments therefore don't share the same enthusiasm in eHealth.

3.1.3 Healthcare provider

The healthcare provider is the company that is using the eHealth solution to provide care to patients or customers. This party represents the medical professionals and nurses. Within the PAL project, several hospitals including their stakeholders like medical professionals are helping to co-create the solution. This would benefit the acceptance and integration of the solution in a later stage.

3.1.4 Patient association

The patient association (in our case the Dutch diabetes fund) serves the interests of the patients by influencing the insurance companies, professional associations, healthcare providers and the government. They are well up-to-date in the specifics and ongoing research of the disease and the needs of their target group. A patient association can become a strong partner to support and promote a solution in an early stage when they see a clear added value of an innovation. They can support the development with funds or evaluation with real users and promote the solution for patients, healthcare providers, professional associations, insurance companies, as well as the government.

3.1.5 Professional association

The professional association is the scientific association of professionals in a certain expertise. They tend to improve the education and accreditation of the professionals in order to improve the quality of the provided care. They define the standards and guidelines for doing so. They could be an interesting partner if the solution has a clear improvement in the safety or quality of the provided care.

3.1.6 Government

The Dutch Healthcare Authority is supervising for the healthcare market. They do this by defining performance of the healthcare and by determining the rates. The eHealth solution first has to become a performance before it can be provided and charged for. The National Health Care Institute is the governing body responsible for determining the primary healthcare package that is mandatory for all Dutch citizens. They follow the principles of 'evidence based medicine' for rating which care has to be allowed in the primary package and which not. Next to that, it is responible for stimulating the quality of the healthcare and to advice the government in new healthcare innovations.

3.2 Innovation routes

Innovation routes are different ways to publish an eHealth innovation, each involving different stakeholders. In this section, four different innovation routes are described.

3.2.1 Consumer route

In this route, the eHealth innovation is directly offered to the public (patients and customers). The solution can still be advertised by an insurance company, but the patient/consumer has to pay for it. The main focus point of this route is that the solution has to be fairly priced and that it solves a recognizable problem. It works the best for solutions focusing on wellness and comfort. The user could get the app for free by selling ads or by sharing their data with 3rd parties. This could lead to specific ads about medication according to their preferences, but this conflicts with the privacy of the user. As this solution has a different focus and the fact that TNO doesn't want to charge the end user, this route is not suitable for the solution.

3.2.2 Healthcare provider route

The healthcare provider route seems more relevant. This route is typically used for solutions that give direct benefits to the healthcare provider. These benefits are for example a more efficient healthcare, a competitive advantage, or an image improvement. Healthcare providers generally have own funds available for these kind of investments. Co-creation helps to improve the recognition and acceptation

of the solution for this target audience. The current PAL solution is made in close collaboration with two hospitals in the Netherlands and one hospital in Italy.

3.2.3 Insurance company route

In this route, the insurance company adds the eHealth solution to their insurance packages. Therefore, the patient is compensated for the costs. The contents of the healthcare stays unchanged, but it does change the way it is offered. It for example makes the healthcare more accessible or more efficient. The key to be successful in this route is that healthcare providers, patients and the patient association need to be excited about the solution and support it. They can influence the professional association which plays an important role in the creation of guidelines for provided care. Insurance companies have to follow these guidelines.
If the solution leads to cost or labor reductions without changing the quality of the healthcare, it is recommended to stay on the healthcare provider route without involving the insurance companies. If the solution on the other hand leads to a more sustainable healthcare which is significantly better, the insurance company route would be a good option. It is also recommended to leave the negotiations to an enthusiast healthcare provider instead of doing this directly.
It is furthermore crucial for an insurance company that the solution leads to replacement of the current healthcare, for example due to substitution or self management which in the end leads to less claims and thus reduced costs. As the main focus of this project is to increase the self-management of children with diabetes, this could definitely be interesting for insurance companies.

3.2.4 Government route

The government route is only possible when a medial-technological solution leads to new ways of healthcare, which are not offered or which are not compensated for. The route is used when the solution fundamentally changes the healthcare, the way it works or simply because it was not possible to deliver the healthcare before. The government route is therefore not applicable for most eHealth solutions. The PAL system tends to improve the self-management of the children with diabetes, but the healthcare itself stays unchanged. The children still need to measure and control their blood sugar levels. The government route is therefore not the optimal route for this project.

3.3 Route comparison

The previous section already highlighted certain pros and cons of the different routes. In this section, we defined the criteria and weights in order to properly rank them. The criteria and priorities were defined in collaboration with project members of the PAL project. The scores are based upon interviews with partners from the project.

Criteria	Weight	Consumer	Healthcare provider	Insurance company	Government
The service can be offered without costs to the end-user	10	○	●	●	●
The privacy of the end-user is maintained	10	●	●	◕	●
Valuable data of the end-user won't be sold to 3rd parties	10	●	●	◕	●
Easy to expand service to other countries	7	◕	◑	◑	◕
PAL project goals are important and will be maintained	8	◔	●	◕	◕
Time to market is relatively short	6	◕	◕	◑	○
Own funds that are mandatory for market launch are relatively low	6	◔	◑	◑	○
	57	**58%**	**86%**	**71%**	**72%**

Figure 3.2. Criteria and scores per innovation route

As seen in the scores, the consumer route is generally not preferred. The healthcare provider route seems to have the best support. The government route had support from the partners, but due to the extended requirements of this route given in § 3.2.4, this route is unsuitable for this project. To conclude, in the following subsections, the two most preferable outcomes are matched with an appropriate strategy.

3.3.1 Healthcare provider strategy

In this situation, the solution will be sold to hospitals providing diabetes care. As the solution is co-created with hospitals, it would be relatively easy to integrate the solution with their workflow. The nurses and specialists are already familiar with the PAL ecosystem, so there is no learning curve for the employees of the hospital. The PAL project brings a more efficient healthcare and an image improvement for the hospitals, thus there is added value for them. This could lead to competitive advantages. The disadvantage is a possible lock-in as specific hospitals can choose to integrate it. The solution is not widely available, but just for patients of the specific hospital(s). An advantage for the PAL team is that this solution is very suitable for retaining the privacy of the users. Hospitals need access to the personal data in order to provide healthcare. The data stays with the professionals and is therefore not available for 3rd parties.
To overcome the hospital lock-in, a freemium business model could be proposed. Hospitals can provide the full extended version to their patients which helps them to achieve personalized goals in self-management. The professionals and nurses can closely monitor their progress and provide an efficient healthcare. In the hospital, the PAL robot can be used, while the patients use the virtual avatar on a tablet at home. The general audience still has access to a simplified version of the virtual avatar in the tablet app. However, to maintain the competitive advantage, the hospital(s) can choose to leave out the personalization of self-management goals, but just make the basic games and diary function available. A premium version could still be made available using in-app purchases. Parents can choose to activate the full functionality including the monitoring functions. Children with diabetes from other hospitals are then able to fully use the app without consequences for the privacy.

3.3.2 Insurance company strategy

In this strategy, the insurance company covers the costs of the PAL system. They can be convinced by underlining the cost reduction due to a more efficient healthcare by improving the self-management skills of the children. This leads to replacement of existing care, and thus less claims. A convinced insurance company is a great first step in nation wide coverage, as insurance companies determine together with professional associations which care is covered by all the insurances. The writers from the innovation route method further recommend to have at least one enthusiast healthcare provider to convince the insurance company instead of trying to do this alone. In this case, the hospitals from the PAL consortium could initiate this process.
In exchange, the insurance company might want to have proof that the solution indeed improves the self-management of the children and thus results in a cost reduction. This could be resolved by making anonymous usage statistics available to them, which they could compare to the overall number of hospital visits from their patients. This number should decrease over time to proof the cost reduction. The insurance company that helps with the funding could have the exclusive right to use the system for their patients at the start. After this period, the system should be generally available to patients from other insurance companies too. This could be arranged in cooperation with the professional associations.

3.4 Conclusion and discussion

We saw that the most-used business models for eHealth apps didn't use traditional app store revenue sources, but rather licensing, app development fees and service sales. To protect the privacy of the end-users, the app should not give information to third parties that could expose the identity of the user in any way. Due to this, advertisements are not an option. For the PAL case in particular, two possible strategies were discussed.
One of the major shortcomings of this research is the fact that it's applied to the specific case of the PAL project, which is not comparable to simple eHealth apps from small publishers or start-ups. The PAL project is a research project with different partners all over Europe.
This leads to the second shortcoming of the research. We discussed the entry of the Dutch healthcare system, which is one of its kind. It would require a more specific research for each country, as they all have their own healthcare system, politics and laws. Currently, only the Dutch and Italian hospitals are involved in testing, so it would be a logical step to launch the solution at one of these countries. In order to continue, it would be helpful to investigate the Italian health care system. Besides partners, there are also potential users in both countries.

Bibliography

[Hettinga, 2013] Hettinga, M. (2013). Innovatieroutes in de zorg. (November). Available from: www.innovatieroutesindezorg.nl.

[Looije, 2015] Looije, R. (2015). Pal for diabetic children | TNO. Available from: https://www.tno.nl/en/focus-areas/healthy-living/predictive-health-technologies/pal-for-diabetic-children/.

[PAL4U, 2015] PAL4U (2015). About - PAL. Available from: http://www.pal4u.eu/index.php/project/about/.

[Politiek and Hoogendijk, 2014] Politiek, C. and Hoogendijk, R. (2014). Co-creatie ehealthboek. (december):337. Available from: http://www.zorginnovatieboek.nl/wp-content/uploads/Cocreatie_eHealthboek.pdf.

[Privacy Rights Clearinghouse, 2014] Privacy Rights Clearinghouse (2014). Mobile Health and Fitness Apps : What Are the Privacy Risks ? *Privacy Rights Clearinghouse*, pages 1–7. Available from: https://www.privacyrights.org/printpdf/67502.

[Research2Guidance, 2015] Research2Guidance (2015). mHealth App Market Sizing 2015-2020. (November):1–20. Available from: http://research2guidance.com/product/mhealth-app-market-sizing-2015-2020/.

[Wölfl, 2012] Wölfl, S. E. (2012). The ALIZ-E Project : Adaptive Strategies for Sustainable Long-Term Social Interaction. *German Conference on Artificial Intelligence (KI-2012)*, pages 39–41. Available from: https://www.dfki.de/KI2012/PosterDemoTrack/ki2012pd09.pdf.

Appendix B

Diabetes Know&Do goals

Know&Do self management goals for Children with Diabetes		AGE					
#	**DIABETES IN GENERAL**	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Can explain in own words that he/she has diabetes						
2	Can explain in own words that his/her body needs insulin						
3	Can explain in own words that insuline is being given with a insulin pen/pump						
4	Can explain in own words that if he/she uses insulin and eats regular, he/she can feel good and grow up as other children						
5	Knows that diabetes is a chronic disease and will not go away						
6	Knows that diabetes is not contagious						
7	Knows that nobody knows why some get diabetes and others do not						
8	Knows that not he/she, nor anybody else, is to blame that he/she has diabetes						
9	Can explain in own words that the insuline reaches his/her body through a insuline pen or pump in his/her belly or butt						
10	Knows why blood glucose needs to be pricked						
11	Can explain in own words that diabetes is a chronic disease and will not go away						
12	Can explain in own words that diabetes is not contagious						
13	Can explain in own words that nobody knows why some get diabetes and others not						
14	Can explain in own words that not he/she, nor anybody else, is to blame that he/she has diabetes						
15	Can explain in own words why blood glucose needs to be pricked						
16	Knows which organs are involved in digestion and the function of the pancreas						
17	Can explain in own words what diabetes is and it's impact on daily life						
18	Knows the fuction of the pancreas and the effect of insuline on the body						
19	Knows how the most important bodily organs and systems work and what diabetes means						
	BLOOD GLUCOSE	**6-7 engl**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Can assist parents with measuring glucose by getting the meter ready or even prick under supervision						
2	Can explain what the number on the glucose meter means						
3	Can prick glucose by him/herself, but glucose value is being interpreted by parent						
4	Can prick and interpret glucose value by him/herself and how to act accordingly						
5	Knows at which times glucose needs to be measured						
6	Know when to measure glucose in between						
7	Can explain the influence of insulin, carbohydrates and exercise on their blood glucose						
8	Know the causes and symptoms or high or low blood glucose						
9	Know what to do when they have a high or low blood glucose						
10	Know how to prevent a high or low blood glucose						
11	Is responsible for measuring and interpreting own blood glucose						
12	Can interpret the consequences of values that fall out of normal range						
13	Can explain the causes of high or low blood glucose						
14	Can recognize the symptoms and knows how to prevent and treat high or low blood glucose						
15	Know it is important to talk to others about diabetes						
16	Will tell others what to do during a serious hypo						
17	Know it is important to explain about Glucagon when they stay somewhere else with others						
18	Know that long-term high blood glucose values are a risk for ketoacidosis						
	LOW BLOOD & HIGH GLUCOSE	**6-7 yrs.**	**8-9 yrs.**	**10-11 yrs.**	**12-13 yrs.**	**14-15 yrs.**	**16-17 yrs.**
1	Children who already experienced a low blood sugar can describe the symptoms of a hypo and know that they should alert an adult immediately. Children who haven't experience a low blood sugar yet, know they should a alert an adult when they do not feel well.						
2	Knows to take glucose pills or lemonade when he/she has low blood sugar						
3	Knows the symptoms of low blood sugar						
4	Knows how many glucose pills of lemonade they need to get their blood sugar at a normal level						
5	Knows at which value a blood sugar is too low						
6	Can explain what the possible causes of a low blood sugar can be						
7	Can correct for a low blood sugar by taking dextrose pills or lemonade						
8	Knows that he/she can't always depend on the symptoms of a hypo and therefore should always, if possible, measure his/her blood sugar.						

	INSULIN INJECTIONS & INSULIN PUMP	6-7 yrs	8-9 yrs	10-11 yrs	12-13 yrs	14-15 yrs	16-17 yrs
1	It is the responsibility of parents to prepare and inject the insulin pen. The child can assist by counting the number of units, by pointing the right place of injection and counting aloud after injection.						
2	Children with a insulin pump can take a bolus under supervision if the child has knowledge of numbers.						
3	Knows that parents will indicate and control the amount of bolus.						
4	Knows that the dosage of insulin is the responsibility of the parent.						
5	Can self-administer a insulin injection						
6	Knows the difference between different kinds of insulin.						
7	Knows when they need which kind of insulin.						
8	Knows how to preserve insulin.						
9	Can self-administer a bolus.						
10	Can help prepare and insert the new system in stages to learn this their selves.						
11	Learn how to adapt the insulin dosage to the blood glucose values and amount of carbohydrates.						
12	Can independently operate the insulin pump.						
13	Can compute the size of the bolus with the blood glucose values and amount of carbohydrates.						
14	Can independently replace a reservoir and system.						
15	Are learning how they independently can adapt the insulin to special occasions through a custom bolus or temporary basal.						
16	Can independently adapt the insulin to special occasions through a custom bolus or temporary basal.						
17	Knows how to set up the basal and bolus and can independently replace an ampule and system.						
	NUTRITION	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Knows how much to eat at school or somewhere else.						
2	Knows when and what to eat, possible with a reminder of an adult.						
3	Knows to mention their diabetes if someone offers candy.						
4	Knows to consult an adult if candy is offered, to discuss whether to eat it immediately or take it home.						
5	Knows what products contain a lot or a little carbohydrates.						
6	Knows when and how much candy they can have.						
7	Knows how to vary with different foods and carbohydrates.						
8	Knows what food products are best to use.						
9	Understand information on nutrition labels.						
10	Knows different kinds of artificial sweeteners.						
11	Have general knowledge about nutrition.						
12	Knows of most food how much carbohydrates it contains or knows how to find out.						
13	Knows what food contains good an bad lipids.						
14	Knows how the human body uses food.						
15	Knows how to apply their knowledge to daily situations, sports, school, party's and eating fast-food.						
	PHYSICAL ACTIVITY	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Can give a simple explanation of the relationship between nutrition, insulin and physical activity.						
2	Knows the relationship between insulin, nutrition and physical activity, take precautions with adults when exercising.						
3	Learn how to independently adapt nutrition and insulin to the amount of physical activity.						
4	Knows what problems can occur while and after physical activity and how to solve this.						
5	Knows trough experience and testing blood glucose how exercising and physical activity can influence blood glucose values.						
6	Knows what precautions should be taken if they participate in sports						
	SOCIAL ENVIRONMENT	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	As preparation for child and parent, parents are encouraged to let the child play with others. Child can be involved in explanation of own parents to other adults.						
2	Is involved in the transfer of care of the parents to other adults when he/she has a sleepover.						
3	Can independently have a sleepover with others, whereas an adult is responsible for care taking.						
4	Can manage their diabetes mostly themselves, but need parental/adult support.						

5	Can be more independent than others, but sometimes care should be taken over by adults again without a reason.						
6	Can have a normal life between friends and peers.						
7	Knows how to independently cope with diabetes, but it is important that parents support child.						
8	Should always inform an adult in their environment (school/sports)						
9	Knows there are compensation rules for diabetics.						
10	Knows where to get diabetic materials.						
11	Knows there are several occupations that may not be carried out.						
12	Knows there are custom rules for driving license and a medical declaration is required.						
	SMOKING & DRUGS & ALCOHOL	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Knows smoking and diabetes is a bad combination.						
2	Knows smoking influences the insulin functioning, through the contraction of the blood vessels because of nicotine there is a decrease of insulin functioning.						
3	Knows long term smoking leads to a decreased insulin sensitivity						
4	Knows smoking can cause eye problems.						
5	Knows drugs influence the brain, which makes it difficult to control diabetes.						
6	Knows drugs are addictive.						
7	Knows alcohol use influences blood glucose.						
8	Knows there is no contra-regulation because the liver is breaking down alcohol.						
9	Knows that he/she should measure blood glucose before going to bed after drinking alcohol.						
10	Knows their blood glucose should be higher with than without alcohol and therefore they should eat long-working carbohydrates like bread, potato chips or French fries.						
11	Knows that when you have alcohol hypo symptoms are harder to notice						
12	Knows that he/she should inform his/her friends that hypo symptoms look like inebriation.						
13	Knows the advice to avoid drinking until 24.						
	ILLNESS	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Is being told the influence of being ill on blood glucose						
2	Knows the influence of being ill on blood glucose, they should determine with parents what is the best to do.						
3	Knows to regularly check blood glucose, potential ketones and body temperature if they have a fever.						
4	Knows which precautions should be taken if being ill.						
5	Knows how to adjust their insulin dosage when having a fever.						
6	Knows how to handle nausea and vomiting.						
7	Knows the possibility to call for the help of the child diabetes team						
	LONG TERM COMPLICATIONS	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Is being told the connection between blood glucose and the influences on the long term.						
2	Knows the importance of good regulation of the blood glucose because of the long term complications.						
3	Knows the regular checks to detect long term complications in an early stage. HbA1c determination, blood pressure, examination of urine microalbuminuria, eye- and foot control.						
4	Knows why every examination is done, how the results are assessed and what should be done if the results are aberrant.						
	TRAVELING	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Can travel independently.						
2	Knows blood glucose should be well regulated before travels.						
3	Knows how to transport and preserve insulin.						
	SEXUALITY AND PREGNANCY	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Knows the effect of sex on blood glucose.						
2	Knows the risks of diabetes and pregnancy.						
3	Knows the importance of contraception/birth control.						
4	(girls) Knows blood glucose should be well regulated before and while pregnancy.						
	EDUCATION	**6-7 yrs**	**8-9 yrs**	**10-11 yrs**	**12-13 yrs**	**14-15 yrs**	**16-17 yrs**
1	Has acces to education materials for their age what can be studied with parents.						

Appendix C

Experiment protocol

Protocol PAL – 643783-4

"Personal Assistant for healthy Lifestyle"

Funded by the Horizon 2020 Framework Program of the European Union under grant agreement no **643783**

PROTOCOL PAL - 643783-4

Descriptive Title: PAL – Personal Assistant for healthy Lifestyle, INTERMEDIATE PHASE – REFINEMENT OF USER REQUIREMENTS ANALYSIS.

Sponsor code: Grant Agreement nº 643783-4

Protocol Date and Version: XX/XX/2017, V. 01

Promoter: Dr. Rosemarijn Looije, TNO Netherlands Organization for Applied Scientific Research, coordinator of a Consortium of 11 European partners, financed by the European Commission (EC) under the research programme HORIZON 2020, call PHC-26-2014 "Self-management of health and disease: citizen engagement and mHealth (ii) mHealth applications for disease management".

Coordinator Center of the entire Project: TNO - Netherlands Organization for Applied Scientific Research

Financing Sponsor: European Commission (EC) under the research programme HORIZON 2020, call PHC-26-2014 "Self-management of health and disease: citizen engagement and mHealth (ii) mHealth applications for disease management".

TABLE OF CONTENTS

1. GENERAL INTRODUCTION ON THE PAL PROJECT

Type I Diabetes Mellitus (T1DM) is a complex disease, with a worsening prevalence and a therapeutic regimen that has to be adapted to the living conditions and activities of the patient. If not correctly handled it can lead to serious complications up to a reduction in life expectancy. All these factors represent a major challenge for younger patients, who need to familiarize with their condition and acquire knowledge and autonomy in managing their own therapy.

The PAL project "Personal Assistant for healthy Lifestyles", is a four-year (March 2015- February 2019) project funded by the European Commission, whose purpose is the development of a multi-system technology that can support young patients with T1DM (aged between 7 and 14 years), their families and the medical staff, in order to accompany them in a shared educational path towards a proper management of the disease. PAL will also provide these young patients with an educational tool able to strongly motivate children/ teens, eventually helping them in adopting healthy lifestyles. PAL refers to an interactive platform consisting of: (*i*) web applications – e.g.: a virtual timeline composed by therapeutic, nutritional, activity, emotional diaries and quizzes - , (*ii*) applications for the mobile technology - mHealth app and games - (*iii*) a humanoid robot (NAO[1]) which is able to interact playfully with children (Figure 1). All these components use a common knowledge-base and reasoning mechanism. PAL is a multicentric project, shared by a Consortium of 11 partners that ensures the right balance of the skills required for this kind of research:

1. Nederlandse organisatie voor toegepast natuurwetenschappelijk Onderzoek - TNO (the Netherlands), the coordinating Center
2. Fondazione Centro San Raffaele (Italy), in collaboration with Ospedale San Raffaele
3. Imperial College of science, technology and medicine (United Kingdom)
4. Mixel Scarl (Italy)
5. Deutsches Forschungszentrum fur Kunstliche Intelligenz GmbH - DFKI (Germany)
6. Technische universiteit Delft (the Netherlands)
7. Bierman egbertus petrus bartholomeus – PRODUXI (the Netherlands)
8. Stichting ziekenhuis gelderse vallei – (the Netherlands)
9. Meander MC – (the Netherlands)
10. Diabetesvereniging Nederland (the Netherlands)
11. SOStegno70 insieme ai ragazzi diabetici ONLUS (Italy)

[1] www.aldebaran.com

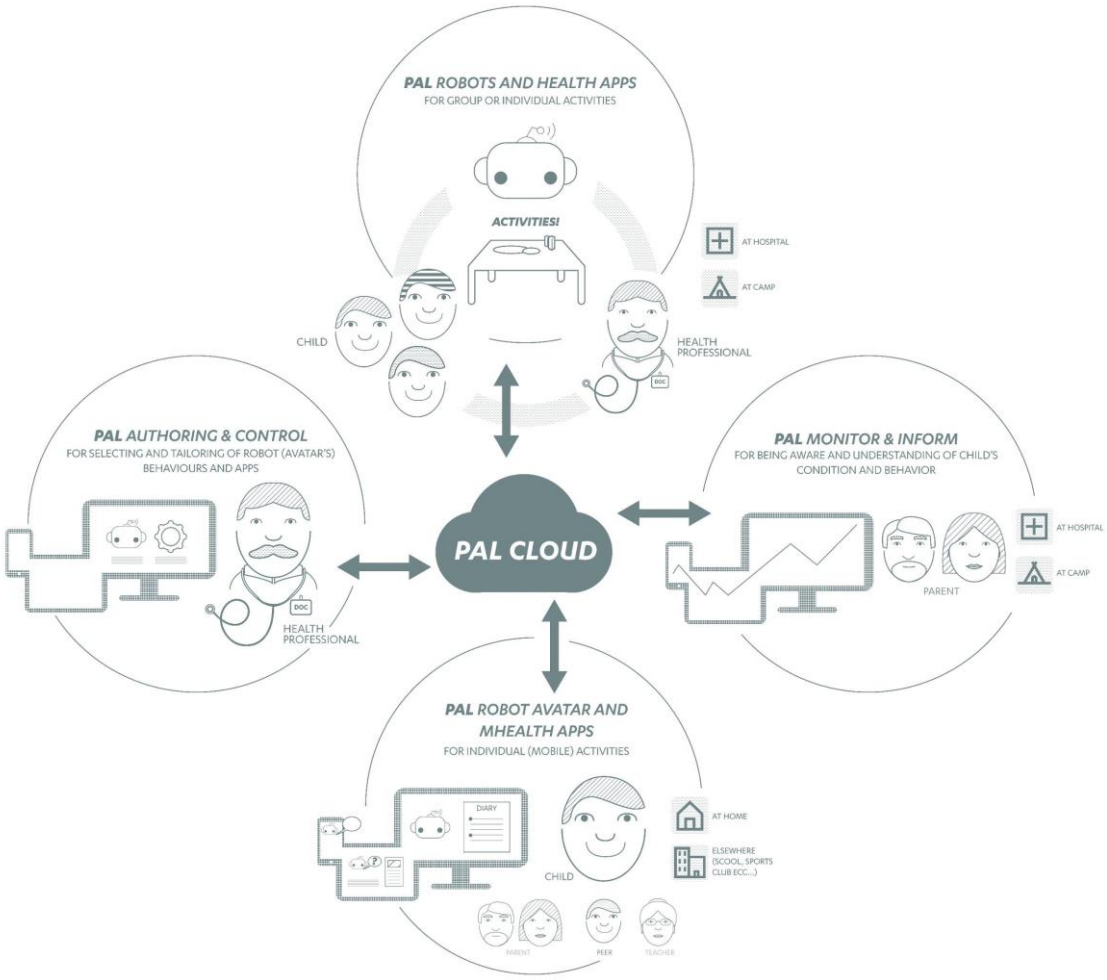


Figura 1PAL system architecture

Specifically, the entire project presents the following objectives, which can be summarized in the following five sub-points:

1. Determine the needs of the users of the system (children /teenagers, their families and the (in) formal caregivers) to support a conscious and sustained management of their own condition during the transition from childhood to preadolescence. This means, for example, individuating:
    a. the determining factors for a child/teenager (e.g.: knowledge, habits, attitudes, needs, etc ...) that may impact on disease management (e.g.: adherence) and then be reflected on his state of health (e.g.: metabolic and glycaemic control):
    b. the determining factors for parents (e.g.: knowledge, understanding, trust) in order to offer their children a support and a shared responsibility in achieving autonomous management of the disease;

c. the benefits and possible obstacles of using the PAL system for the healthcare professionals in order to offer a personalized support in the process of education and care of young patients.

The PAL system, through the interaction of single activities (quizzes, online diaries, mHealth apps for mobile technology, ...) aims to leverage on the determinants identified, to promote a correct approach to the management of the disease.

1. Develop an ontology towards a multi-user support, based on correct medical and health knowledge, able to evolve with different engagement strategies and customize them according to the learning objectives of each child.
2. Develop a support system for caregivers that articles in:
    a. A control module for the medical staff that allows you to customize the educational development of each child and to monitor
    b. A form of information for parents that allows to monitor the location of their children, in different ways depending on the needs (age, etc ...)
3. Develop a user-model to tune the educational support and motivational hints given to children/ young people depending of their behavioural and evolutionary change, their preferences, knowledge, skills and experience.
4. Develop a series of applications for mobile technology focused on the promotion of healthy lifestyles and strengthening of knowledge (e.g. Health rules, food composition, ...), design customized multimodal interaction with the robot NAO in order to promote the involvement and commitment over the long term with the PAL system by all stakeholders.

The project activities will be articulated in the following phases:

- ➢ INITIAL PHASE (executed in 2015-2016)
    - ○ USER REQUIREMENTS ANALYSIS: This first phase of the study is primarily exploratory and aims to define the technical and functional requirements of the system, based on expectations and real needs of both end users (*patients and families*) that experienced users (*medical staff and healthcare professionals*);
    - ○ FIRST DEVELOPMENT AND VALIDATION: a first group of the results obtained by the User Requirements Analysis will be implemented within the first prototype of the PAL technology. It will be validated through a test campaign dedicated to end-users of the system.
- ➢ INTERMEDIATE PHASE (planned for 2017 as described in this protocol)
    - ○ **REFINEMENT THE USER REQUIREMENTS ANALYSIS: following the validation of the initial phase, the technology components and the contents of the PAL system will be revised and improved according to the feedback and suggestions obtained from the users involved (iterative approach of co-creation methodology [1]);**

- DEVELOPMENT AND VALIDATION: the results of the first revision of the User Requirements Analysis will be implemented within the integrated prototype of the PAL system. Such a system upgrade will be validated as a result of successive test campaigns dedicated to end-users of the system.

➢ FINAL PHASE

- RELEASE: the focus will be on the implementation and release of the final PAL integrated system, as a result of the findings obtained in the intermediate steps of validation;
- VALIDATION: users will benefit in the long term the system developed, final impressions on the solution developed and tested will be collected to verify the achievement of the project objectives.

This specific protocol (PAL-643783-4), to be assessed by the Ethical Committee, is intended to describe the objectives and research activities that are planned for the REFINEMENT THE USER REQUIREMENT ANALYSIS of the INTERMEDIATE PHASE (2017). It is a continuation of the work as described in the previous protocols (PAL-643783-1_2).

2. RATIONALE

Type 1 Diabetes Mellitus (T1DM) is the most pervasive endocrine-metabolic condition in childhood. Diabetes self-management plays a crucial role, in order for young patients to maintain health-related quality of life and to avoid long-term complications (e.g.: heart and blood vessel diseases, nerve/ kidneys/ eyes damages, etc.). A proper management of the disease, and a healthy lifestyle, can help to prevent these complications [2]. One of the key objectives for the care of young patients is the acquisition of sufficient autonomy in the therapy management. This goal is not easy to achieve, as it requires long-term motivation and perseverance to eventually become a 'lifestyle'. The autonomous management of therapy in children and adolescents is strongly influenced by a variety of personal and environmental factors, such as the development of the child and the support/ care provided by the reference adults [3] [4].

A number of interventions exist to promote the acquisition of self-management skills, but it remains unclear which particular types of intervention are most beneficial and for which type of patient. It is in this context that technological innovations can help to improve care, especially for paediatric populations. This, for example, has been illustrated in the European project ALIZ-E (grant agreement n° 248116), that examined how a social robot can provide support, diabetes knowledge and skills gain, to children with T1DM (10-14 years old), through personalized, adaptive and long-term interaction.

With a vision to work further and advance the knowledge-base and support models of ALIZ-E, the PAL project, Personal Assistant for Healthy Lifestyle (PAL) is currently being developed[2]. Thanks to its multivariate nature, the PAL system offers the possibility for children to make use of it in various settings: in the hospital and diabetes camps children can interact with the social robot, at home and/or at school the interaction can be continued, through the virtual avatar of the robot, helping them in reinforcing key diabetes-related notions and procedures.

[2] http://www.pal4u.eu/

3. RESEARCH QUESTIONS AND AIM OF THE STUDY

Overall research question: Which are the opinion/feedback of the end-users (children with Type 1 Diabetes Mellitus and their parents) about the MyPAL app on the basis of the implementations developed as consequences of the preliminary results gained during the Cycle 2 Experiments (See protocol PAL643783-2, and PAL643783-3)?

How can a second release of the System provide social, educational and pleasurable support to children with T1DM, at the hospital and home, to achieve personalized diabetes management goals (decided together with the Healthcare Professionals involved)?.

The primary aim of the experiments in the *Intermediate Phase* is to refine, further develop and evaluate the the second release of the PAL System in iterative steps during a longer period of use (e.g. 4 months). This is done by evaluating:

(1) User experience, which affects the use of the PAL System, in children, parents and health care professionals;

(2) Effect of the use of the PAL Systems related to diabetes self-management of children. These effects are measured at three levels (divided over the three phases/cycles of the project, respectively): a) determinants of behavior (e.g. knowledge and skills), b) the actual self-management behavior (i.e., adhering to diabetes management regime), and c) the treatment outcomes (e.g., glycemic control).

In regard to User experience, the international standard on ergonomics of human system interaction, ISO 9241-210 [9], defines User Experience as "a person's perceptions and responses that result from the use or anticipated use of a product, system or service". According to the ISO definition, User Experience includes all the users' emotions, beliefs, preferences, perceptions, physical and psychological responses, behaviours and accomplishments that occur before, during and after use. The user experience data will be used to formulate user requirements. These user requirement provide input for the developers to refine and further develop the PAL System. Then we identify the user experience for three kinds of end-user with their attributes, which are quality components that assess how easy user interfaces are to use (Nielsen, 1994). These are the user experience with the PAL System of **children** (understanding, visual appealingness, pleasure and engagement, bonding, motivation and actual usage), **parents** (user experience and attitude with regards to PAL System), and **healthcare professionals** (HCP) (user experience and attitude).

In regard to diabetes self-management, in last year's study (**Cycle 1**), we studied the impact of the PAL System on determinants of diabetes self-management, including knowledge, skills, attitude and self-efficacy. The main focus for the current protocol of the intermediate phase (**Cycle 2**) is to get further insights in the children's treatment adherence and glucose monitoring, their parent's trust

and skills and the health care professionals' awareness of the PAL System. Determinants from the initial phase will also be partly taken into account, considering that the forthcoming experiments will last a more long period and it will be possible to compare them with the previous version of the System. In addition parental stress is observed to be a possible determinant that might be influenced by usage of the PAL System by the child. Determinants such as gender, age, socioeconomic status, etc. will also be explored in order to see if they have an influence on User Experience. These results will be useful inputs to understand which factors are most important for a proper personalization of the System, to be implemented in the next phases of the project

Tabella 1PROJECT ROADMAP - THREE SPECIFICATION & EVALUATION CYCLES WITH AN INCREASING SCOPE.

	Initial Phase (Y1)	**Intermediate Phase (Y2)**	**Final Phase (Y3)**
Determinants	**Knowledge & Awareness**	**Diabetes Regimen Adherence**	**Shared Child-Caregiver Responsibility**
Children - age dependent claims	7-10 yr: + knowledge + awareness + attitude + self-efficacy + skills	7-12 yr: + treatment adherence + glucose monitoring	7-14 yr: + shared responsibility + coping with anomalies - hypos/ hypers + glycemic control
Usage of the system	1 month	4 months	9 months
Caregivers - Claims	*Professionals:* + trust + acceptance *Parents:* + attitude + knowledge	*Professionals:* + awareness *Parents:* + trust + skills	*Professionals:* + tailoring *Parents:* + shared responsibility
Settings	hospital, home	hospital, home, camp	hospital, home, camp, elsewhere
mHealth apps	Timeline, Quiz game	Timeline, Quiz game, Sort&Break	Timeline, Quiz game, Sort&Break, other miniApps

In Cycle 2, we will deploy a PAL System, which is enhanced in comparison to Cycle 1. See for more detail section 4.4. In short, the PAL System at Cycle 1 covered a robot buddy for children with diabetes at the hospital. Children can have a short conversation with the robot and they can play a diabetes quiz together. At home, children can use the MyPAL app, which contains an avatar

(i.e., digital version) of the robot. The avatar looks, speaks and behaves the same as the robot. Moreover, children can play the diabetes Quiz with the avatar and keep track of diabetes related information (including self-management activities, measures, emotions) on a 'timeline', somewhat similar to a diary. Finally, children can achieve self-management goals (i.e., objectives), which are set with the HCP at the beginning of the study with PAL Control. Children can achieve these goals through the use of MyPAL. For example they can have a goal to improve their knowledge about counting carbs and they can get knowledge points by playing the diabetes Quiz with the avatar.

The enchantments for Cycle 2 consist of the following **additional features**. We added a new educative game to the System, called 'Sort&Break. Also, we have improved the visualization of MyPAL, including more user-friendly timeline and more natural dynamic behaviour of the avatar. Finally, we have improved the interaction, that is to say, the robot and avatar's dialogue is enriched and more natural, and the they provide more feedback on the entries in the timeline and the on the objectives (self-management learning goals).

The **research questions** we aim to answer in Cycle 2 of our study, over a course of four months, are the following:

1. Do the enhancements of the PAL System contribute to increased user experience of children, parents and HCP and to the usage of the PAL System by children?
2. Does increased user experience in children contribute to a stronger bond with the PAL robot and avatar?
3. Does the improved user experience in children, parents and HCP and a stronger bond between children and PAL robot and avatar contribute the increased use of the PAL System?
4. Does the use of the PAL System contribute to improved self-management determinants in children, including knowledge, motivation and awareness of their self-management objectives?
5. Does the use of the PAL System contribute treatment adherence and glucose monitoring in the participating children?

As we aim to study the effect of each feature, we will introduce them step by step during the study. We will start with a PAL System with improved visualization (T1, onset of study), then we will introduced feedback on entries in the timeline (T2, after 3-4 week), then we will introduce feedback on the self-management learning objectives (T4, after another 6-8 weeks). We also measure the impact of the System at T3 (3-4 week after T2) and T5 (end of study, 3-4 weeks after T4). For details see section 4.3.

Based on Table 1 and the discussed enhancements, we come to the following **hypotheses**. Where T1 and T4 point to the introduction times in the experiment.

T1. Children will use the PAL System (input timeline, number of times using PAL System, time spent with PAL System) more and find it more usable (SUS questionnaire), in comparison

with previous year, due to the redesign of the timeline and quiz. This is indicative for the adherence, glucose monitoring and motivation (usage and number of times measuring).

T2. Children will be more aware (awareness questions in EMA) of what their objectives are when the visualization of the objective section is changed. This is indicative for the adherence, glucose monitoring and motivation (usage and number of times measuring).

T3. Children will feel a stronger bond with the avatar (bonding questions) when the avatar provides feedback on "glucose" "meal" and "activities" in the timeline. This is indicative for adherence and motivation (usage).

T4. Children will feel a stronger bond with the avatar (bonding questions), are more aware of their objectives (awareness questions in EMA) and know better how to reach their objectives (question, knowledge gain). This is indicative for the adherence, glucose monitoring and motivation (usage and number of times measuring).

Other hypotheses are:

- Children will see the avatar and robot as more similar (questions on similarity and relatedness) than in previous experiment in 2016, due to the improved appearance (T1) and the improved interactivity (T3 and T4);
- Children enjoy the choice in activities they have (questions on the different activities)
- Professional Awareness: Professionals will have awareness of what it means for the children if they set objectives, due to PAL control and knowledge of quiz questions (interview)
- Parent trust: Parents have more trust in the self-management of their children when the child uses the PAL System (before/after measurement in trust) - CVS, child initiated diabetes tasks
- Parent skills: Parents provide more self-control to the child when the child uses the PAL System. (before/during/after measurement in task support by parent -- child and parent perspective.) EMA, CVS, and question in MyPAL

4. METHODS

4.1 PARTICIPANTS

Study participants enrolled in this study belonged to following categories:

1. **Children with T1DM (age range 8 to and including 12)** recruited from the hospitals or the associations of the PAL Consortium (Meander MC and Stichting ziekenhuis gelderse vallei -NL- and Ospedale San Raffaele -IT-) with minimum number of 10 to a maximum of 15 subjects per hospital, in this way a total pool of 30 to 45 subjects will be ensured.

Inclusion criteria will be:

(i) Informed consent signed by both parents or legal guardian in case of parental separation;
(ii) T1DM onset with at least of 6 months before the enrolment;
(iii) Children treated by Healthcare Professionals of the hospitals of the PAL Consortium;
(iv) Children native speaking

Exclusion criteria will be:

(i) Neurodevelopmental disorders (e.g. intellectual disability, autism spectrum disorders, cerebral palsy, genetic disorders)
(ii) Informedconsent not signed by both parentsor legal guardian in case of parental separation;
(iii) T1DM onset less than 6 months
(v) Children not treated by Healthcare Professionals of the hospitals of the PAL Consortium;
(iv) No children native speaking

Among the participants, children who already took part of the previous experimental campaign will be allowed to participate to the forthcoming experiments. In the analysis, we will explicitly look at moderating effect of having previous experiences with the PAL System. Amongst others on usability and use of PAL, and self-management behavior.

2. **Parents of the participating children** will be active and integral part of the protocol. At least one parent per child will always be involved in the investigations (minimum of 30/45 parents to a maximum of 60/90).

3. **Healthcare Professionals** will be involved in the implementation of the PAL System. They will support the recruitment of participants, introduction of PAL with children and their parents, and set and review elf-management objectives (see section on the mHealth PAL app). Also, they will provide a short evaluation of their trust and acceptance of the PAL System and awareness of the children's self-management (objectives).

According to what is previously described, in Ospedale San Raffaele are going to be involved:

- from 10 to 15 children with T1DM aged 8 to 12, with an onset not less than 6 months;
- at least one parent per child participating (from 10/15 to 20/30 parents);
- at least 1 Healthcare Professional belonging to the OSR paediatric diabetological team.

According to what is previously described, in the 2 hospitals in the Netherlands (Meander MC and Gelderse Vallei) are going to be involved, per hospital:

- ➢ from 10 to 15 children with T1DM aged 8 to 12, with an onset not less than 6 months;
- ➢ at least one parent per child participating (from 10/15 to 20/30 parents);
- ➢ 3 healthcare professional working for the MMC and 2 Healthcare Professional working for the GV paediatric diabetological team.

Being this current an exploratory phase, the sample size involved in the "INTERMEDIATE PHASE - REFINEMENT THE USER REQUIREMENT ANALYSIS" protocol cannot be defined with certainty in advance, but the minimum thresholds chosen guarantee a numericity sufficient to conduct an effective analysis of the issues. Obviously the greater the participation is, the greater the evidences, the lessons learned and features which can be improved or developed in the future releases of the PAL system.

4.2 MATERIALS

Technological components of the MyPAL app

➢ NAO robot - the NAO robot is manufactured by a French company (Aldebaran Robotics) as a safe and tested product (see the document "Declaration of conformity NAO robot.pdf"). It is important to underline the NAO robot is not a medical device.

➢ Tablets (one per child) - the Timeline and Quiz functionalities, that are going to be described in the next paragraph, are implemented as a PAL app (called MyPAL) running on commercial tablets (owned by the participants or supplied by the researchers if needed). The tablets are bought in common electronics stores for commercial purposes and therefore automatically CE certified.
In addition to that, all the Child-Robot interactions and the interviews with the end-users are going to be audio and video recorded. The tools used for the collection of audiovisual data will consist of digital video cameras and microphones, also in this case found in electronics stores for commercial purposes and therefore CE certified.

Functionalities implemented in the 2nd PAL system

The 2nd release of the PAL system is composed by the following parts:

- ➢ Quiz game with the robot (physically present at the hospital premises and also a game on the table) - children and the robot ask each other multi-choice questions from various domains, both general knowledge and diabetes-related. The database of the Quiz questions, especially for those regarding to diabetes, has been validated by the diabetological pediatric units cooperating to the project and are modulated (in terms of language and difficulty) depending on the age of the children. For the second release of the MyPAL System, it have been added new questions related to the management of the T1DM, which are tailored to different range of age of the children.

Figura 2 Quiz with the physical NAO robot and with the robot avatar on the tablet

➢ Sort and break game with the robot (physically present at the hospital premises and also a game on the table) - during the Summer Camp held in Italy in 2016, children were allowed to free use this game, during the lessons held by the nutritionist of the camp. Feedbacks from them were very good and the Consortium decided to implement that game in the MyPAL System. The insertion of another game specific to nutrition and T1DM was a requirement evaluated during the Cycle 1 of the Experiments conducted in May 2016, both in Netherland and in Italy. Both children and their parents requested the possibility to have other games to make them learn more about nutrition related to T1DM in a play manner. In the game the robot plays with the children for which they need to break open boxes and then sort the items inside in order of number of carbohydrates etc.

Figura 3 Sort and break game with the robot avatar on the tablet and with the physical

NAO robot:

➢ the mHealth PAL app (named MyPAL) for tablets, comprehensive of the Timeline, Sort & Break and Quiz game functionalities with the virtual NAO robot avatar. The MyPAL-Timeline is intended to be a diary feature, in which children have the possibility to fill in a personalized report of their daily activities, as shown in Figure 4-8. Through the timeline children, according both to their specific diabetes management objectives and engagement in the system, can compile day by day: (i) the therapy diary - with details of glycaemia

checks and insulin doses; (ii) a nutritional diary - with the details of the meals; (iii) an activity and emotion diary - in which they can freely describe what they've done during the day, also by uploading pictures, (e.g.: sports, party with friends, excursions, etc.) and which feelings they have experienced in this occasions. The virtual avatar of the NAO is going to interact with the children during the MyPAL-Timeline use, for example with greetings and personalized/ motivational feedbacks tuned on the activities done (e.g.: "Hi Sam, it's nice to see you today", or "Good work, you're accomplishing your objective very well today").

➢ Diabetes self-management learning objectives (i.e., goals). To manage diabetes children need to acquire certain knowledge, skills and attitudes. These are formulated as learning goals. Each learning goal should be a small step towards self-management that can be learned in a reasonable time. Not all goals are relevant to all children. Therefore, goals are selected by a professional in collaboration with the child and their parent(s). At the onset of the study HCP set self-management learning goals for the children to pursuit during the study. The HCP have an interface to the PAL System, called PAL Control. It enables them to add users (children), enter details (demographics) and set diabetes self-management goals. PAL Control also has been used in the previous year's experiments. Goals are intended to motivate the child to learn about diabetes. After setting goals, for the children, MyPAL displays the (active) goals relevant to the child, and provide progress feedback (i.e., display attained goals and progress of active goals). Moreover, they have tasks to attain a goal. MyPAL offers activities that are related to goals for this purpose (e.g., a child can play the sorting game to learn about carbohydrates in food products).

Figura 4 View of a functionality of the MyPAL app – Begin screen (Design 2017)

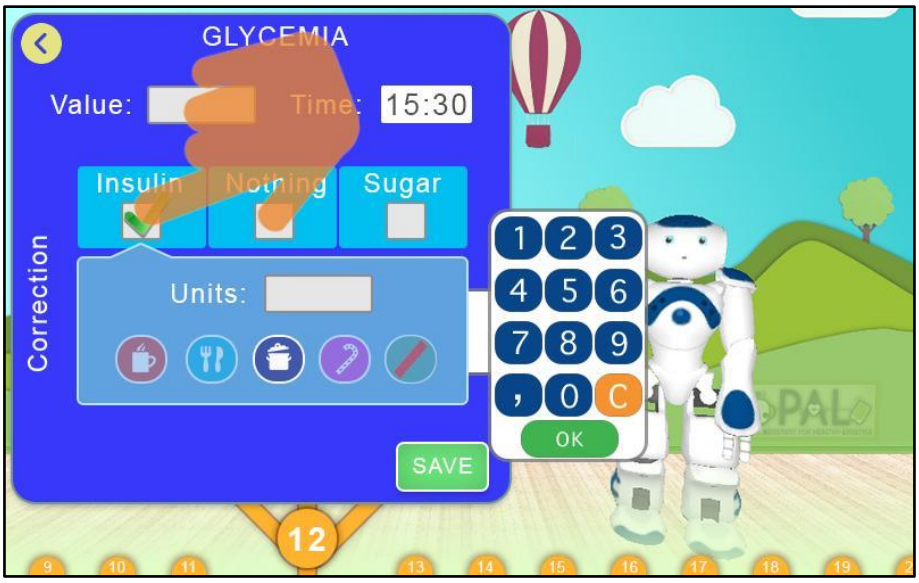


Figura 5 View of a functionality of the MyPAL app – Glycaemia section (Design 2017)

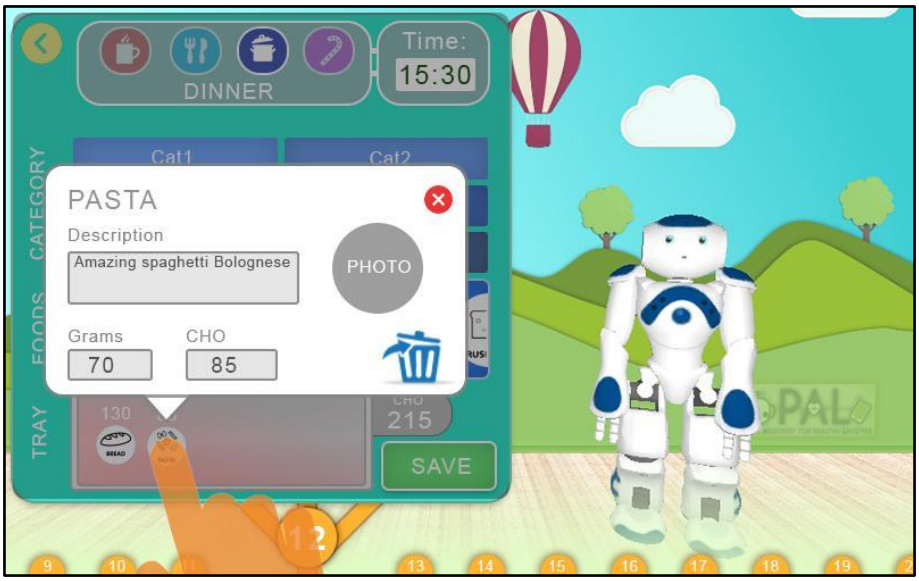


Figura 6 View of a functionality of the MyPAL app – Meals section (Design 2017)

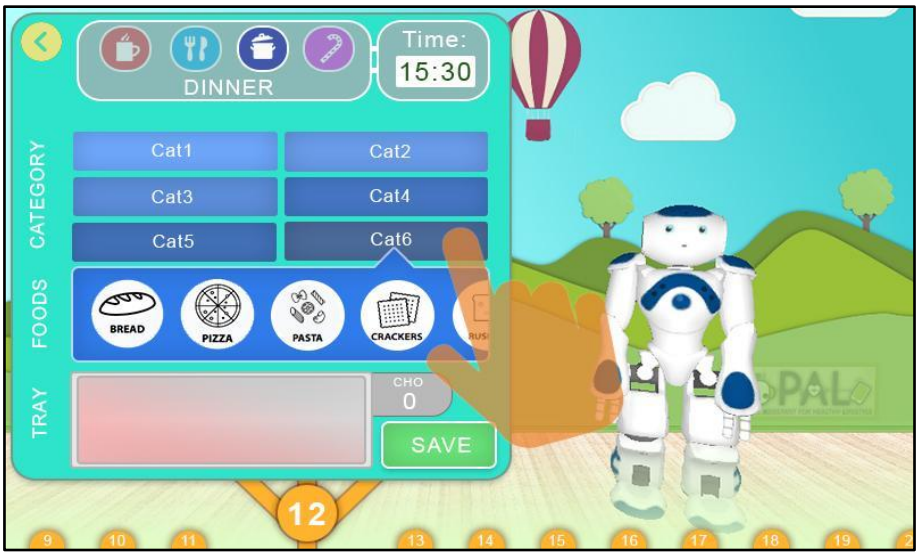


Figura 7 View of a functionality of the MyPAL app – Categories within meal section (Design 2017)

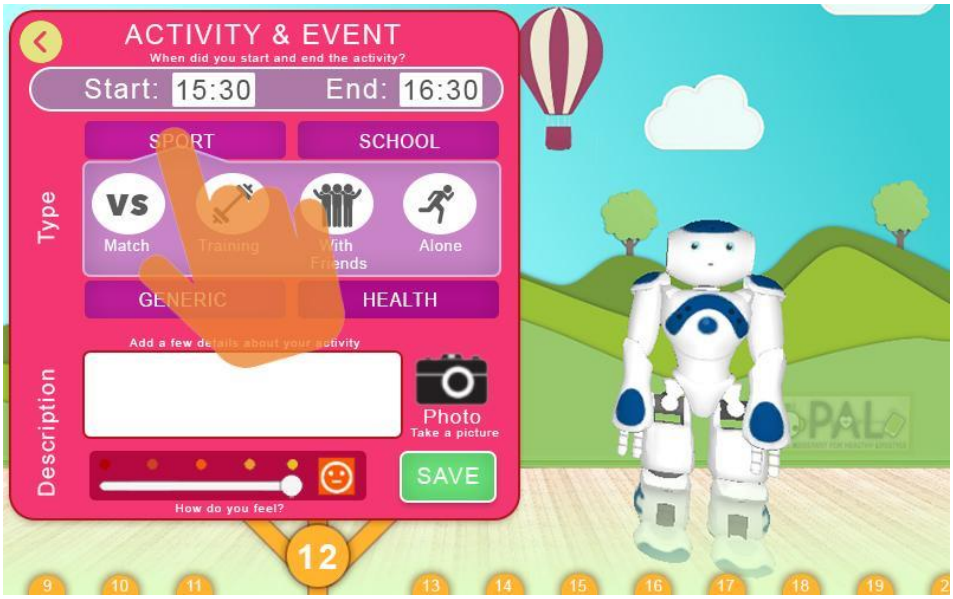


Figura 8 View of a functionality of the MyPAL app –Activity section (Design 2017)

4.3 PROCEDURE

Timeline of the experiments will be slightly different in Netherland and in Italy to match local requests, such as organizing visits around holidays. In Netherland the experimental campaign will be held from May 2017 to August 2017 (included), while in Italy it will be held from June 2017 to September 2017 (included). Each participant will use the second release of the currently available features of PAL System for approximately four months.

The activities related to the present Protocol will take place both in the premises of the hospitals

involved and in the houses of the participating families. Children and their families are going to be scheduled three mandatory visits to the hospitals (at the beginning, halfway and at the end of the study period), where each child will have the possibility to interact with the NAO robot and the families will meet the research team and the healthcare professionals in order to be instructed about the activities proposed. In between visits 1 and 3, and visit 3 and 5, children and their families will interact with the PAL System at home. In addition, two optional events are organized at the hospitals, between visit 1 and 3 and between visit 3 and 5.

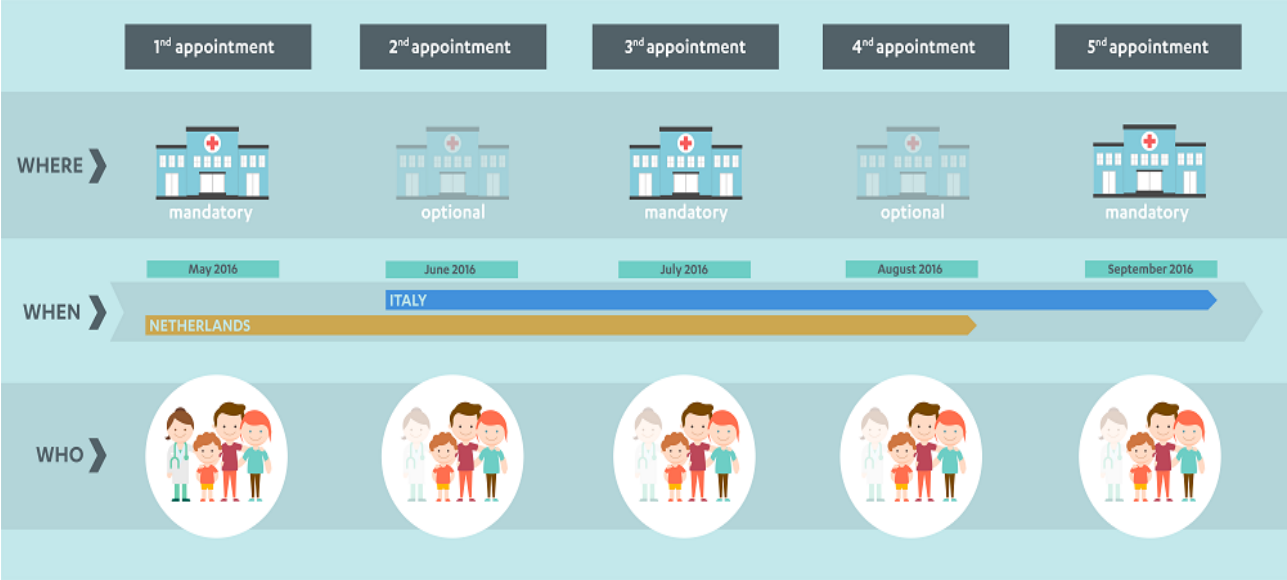


Figura 9 Timeline of the experiments

In the following section we report the details of the experiments' conduction of the Cycle 2 Experiments.

Before to the intervention period:

Children responding to the proper inclusion criteria, and their parents, are going to be contacted and briefly informed about the current study details and aims, and then invited to participate by the health care professionals (HCPs) involved and/or by the technical research team members. If they show interest in participating, will receive an information letter to take home. After approximately 1 week the parents and their children are asked whether they are willing to participate (or not). If so, further information will be given and there will be for them the opportunity to ask for any question or doubt. Then, five appointments are scheduled during the intervention period.

First appointment at the hospital (T1):

The participating children and their parents arrive at the hospital on the basis of their appointment schedule and are received by the multidisciplinary PAL team, composed by: the technical

researcher(s) and the healthcare professional figures among the researcher team - the psychologist and the caregiver, a medical professional profile among the paediatric diabetological team (e.g.: paediatrician/diabetologist, a nurse or a nutritionist - depending on their availability). Before starting the encounter, they have the opportunity to ask further questions and sign the informed consent forms - specifically designed for this protocol - before the activities take place (see documents "*infconsChildren/infconsParents/infconsHCPs.pdf*").

Firstly, both children and parents have to fill in some questionnaires, that are aimed to collect data to characterize the involved population. All the questionnaires to be used during this appointments and the others, are specified in section 4.4 Materials and measurements.

As long as both parents and children finish to complete the given questionnaires, they meet with the Healthcare Professionals and start to complete together the children's profile on the *PAL Control* panel of the Caregiver. After this introductory moment, the whole family is introduced to the *MyPAL* app by the team, who explains its functionalities and how to interact with it on a tablet. Then the Caregiver and the family collectively decide the diabetes self-management goals that each child has to accomplish in the study period. These goals are set in the *PAL Control* panel and directly transmitted to the *MyPAL* app. The personalized goals are strictly depending on children's age and doctor's judgement, T1DM-related knowledge, skills and degree of autonomy and combine objectives strictly related to system that have an impact on the self-management of the disease from an educational point of view (e.g.: fill in the MyPAL therapy diary for at least three times a week in autonomy, try to insert in the therapy diary regularly the glycaemic checks, remember that's important to check glycemic level before and after doing sport, etc).

It has to be underlined that the activities performed through the MyPAL app have not a direct impact on the therapeutic management of type I diabetes (e.g.: they does not affect the insulin dosages of the children or provide an estimation of the carbohydrates counting), but are focused on the educational importance of fostering a correct management of the disease following the proper medical guidelines (e.g.: fill in the glycaemic and nutritional diaries, follow the correct sanitary norms before checking the glycaemia, etc…).

To conclude, before going away, children are brought by the technical researcher(s) to the robot-interaction room. Here they are introduced to the NAO robot and start chatting, exchanging a few questions to break the ice and get familiar to each other, starting to create a bond. Then they start to play the quiz together. The Child-Robot Interactions are going to last about 20 minutes each and are video and audio recorded.

The appointment lasts a maximum of two hours in total; all the questionnaires are given to the participants in their native language (Italian or Dutch).

Event at the hospital and measurements (T2, 3 and 4):

In between visit 1 and the final appointment there are three events. During the events, children and

parents receive a short presentation about the PAL project and relevant updates of the System (i.e., new features added, new feedback when using the timeline). Also, they can ask questions about the PAL System (e.g., working of MyPAL or technical issues they experienced). They complete a number of surveys, as listed in section 4.4. Then children can interact with the PAL System (e.g., Quiz and Sort&Break with Robot) in groups and join social and creative sessions to share their experiences with the PAL System in an informal and pleasurable way (e.g., image theater, user Journey Map and photo collage) (Blanson Henkemans et al., 2016 HRI2016). We specify that the event at T3 is mandatory for both children and parents, while the events at T2 and T4 are optional.

Final appointment at the hospital (T5):

The children receive a notification by MyPAL that today they, together with their parents, are going go visit at the hospital the PAL research team and will interact again with the NAO robot. As soon as they arrive they are received by the technical researcher(s) (also the Healthcare Professionals might be present, depending on their availability). Children are brought to the robot-interaction room and have the chance to play with it another time. The Child-Robot Interactions are going to be video and audio recorded according to the consent of the parents.

In the meanwhile, the parents are invited to discuss together with the researchers their experience, focusing on their acceptance of the tool, the perceived usefulness, its usability and the performance of their sons in reaching the assigned goals. This discussion will be handled through a first questionnaire ("*Parents_final_questionnaire.pdf"*).

As soon as children finish to play with the robot, they undergo a last brief discussion with the researchers, focusing on:

- ➢ Their own diabetes self-management goals
- ➢ Self-efficacy
- ➢ Their perception of the robotic character (both embodied and virtual)

Before going away, children receive a little gift for participating in the current protocol (e.g.: a photo with the robot).

Observations at home

At home the children and parents will receive weekly a pop-up request by an installed EMA-app to complete a short questionnaire with some short questions.

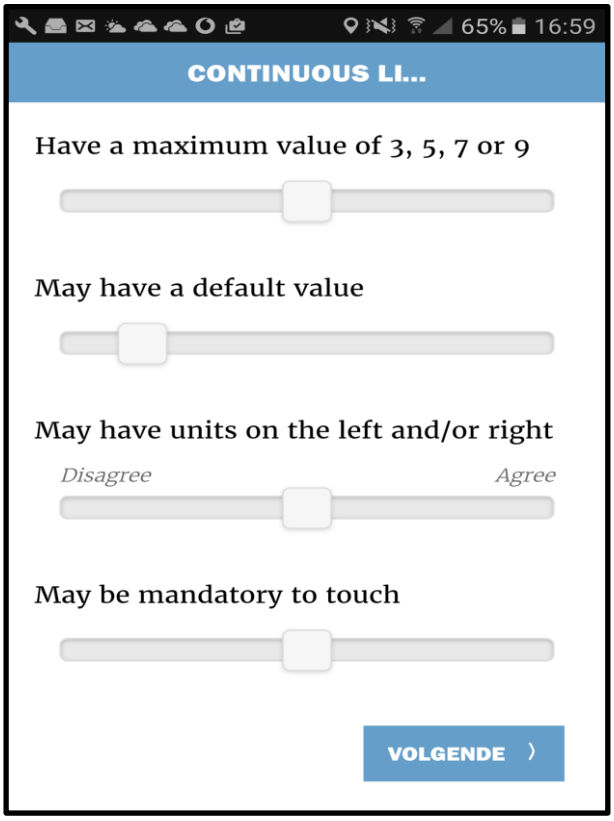


Figura 10 The EMA-app for short questions, eg. once a week

Tabella 2 Questions in EMA app (weekly pop-ups of short questions):

Child/ Parent	Claim	EMA question	Answer
Child	Well-being	1. How do you feel right now?	Very bad :-(<> Very good :-)
Child	Bonding	2. Do you see the robot as a good friend?	A very good friend <> not a very good friend
Child	Similarity	3. Is the robot in the app the same as the robot you have met in the hospital?	Very much the same <> Totally different
Child	Fun	4. Do you have a lot of fun with the	Lots of fun <> Not much fun

		robot right now?	
Child	Motivation	5. Do you look forward to playing with the robot right now? (Why yes/no?)	I do not look forward to playing with the robot <> I do look forward to playing with the robot.
Child	Diabetes knowledge	6. How much do you know about diabetes right now?	Not much <> A lot
Child	Diabetes Self-Management Behavior	7. How much diabetes activities can you do yourself right now? (for example count carbs or prick yourself)	Not much <> A lot
Parent	Stress	1. Do you often worry about your child?	Not very often <> Very often
Parent	Disclosure	2. How often does your child talk about diabetes?	Not very often <> Very often
Parent	Diabetes Knowledge	3. How much does your child know about diabetes?	Not much <> A lot
Parent	Diabetes Self-Management Behavior	4. How much diabetes activities can your child do him/her-self right now? (for example count carbs or prick him/her-self)	Not much <> A lot
Parent	Motivation	5. Is your child motivated to play with the robot app?	Very motivated <> Not very motivated

4.4 MEASURES AND INSTRUMENTS

Tabella 3 Overview of the measurements to be used during the Experiments

Measure	Instrument (participant)	Time	Country
Family Demographic questionnaire (used in PAL year 1 experiment)	Parents	T1	NL+IT
Health-Related Quality of Life (used in PAL year 1 experiment)	Parent: PedsQoL-Diabetes Module, 25 items Child: PedsQoL-Diabetes Module, 28 items - subset PedsQoL treatment 1 & 2: 11 items	T1 - T3, T5	NL+IT
SUS questionnaire	Children - 10 items based on methodology plus other 5 questions from researchers	T2, T3, T4, T5	
Bonding	Child: BOND robot+avatar-child 20 items as used in NL camp (also compare Avatar vs Robot again?) (also in EMA*)	T1, T3, T5	NL+IT
Attitude child robot and avatar	Child: part of SUS & BOND	T1-T5	NL+IT
Expectations of PAL	Child & parent: Open question	T1	NL+IT
Trust child and parent of PAL	Child & parent: BOND	T1, T3, T5	NL+IT
Acceptance child and parent of PAL	To what extent do children and parents feel that PAL could/would be a part of their daily life in the future (additional questions)	T1, T3, T5	NL+IT
Similarity PAL Robot and Avatar	Child: 3 items as used in PAL year 1 experiment/camp (also in EMA*)	T1, T3, T5	NL+IT

Usability child parent and HCP	Child: SUS 10 items Parent: SUS 10 items HCP: SUS 10 items + interview	T3, T5	NL+IT
Motivation to use PAL	3 items as used in PAL year 1 experiment/camp (also in EMA*)	T1, T3, T5	NL+IT
T1DM Self-management	Child & Parent: 3 items as used in PAL year 1 experiment/camp (also in EMA*)	T1, T5	NL +IT
T1DM Knowledge	Child & Parent: derive from quiz performance (objective calculator)	T1, T5	NL +IT
Child initiated diabetes talk	Parent: 3 items how often child initiates talk about Diabetes? (also in EMA*)	T1, T5	NL+IT
Perceived child vulnerability by parents	Parents: child vulnerability scale	T1, T5	NL
Parent's stress	Parents: time investment for child self-management, free time, influence on work, parental stress (also in EMA*)	T1, T3, T5	NL+IT
Log files system use	System usage, self-disclosures and information inserted in timeline, knowledge during quiz, goal attainment, Sort&Break performance.	T1->T5	
*=EMA (Ecological Momentary Assessment)	EMA app: randomized pop-ups of short questions: parents & children (see table of questions at EMA description)	T1->T5	weekly in NL+ monthly in IT
T1DM values	Glucometer values	T1, T3, T5	NL + IT
Wishlist PAL Y3	Creative co-design session	T2 and T4	NL+IT

*T1=first visit; T2=event 1 (optional); T3=event (mandatory) ;T4=event (optional); T5= final visit

The activities concerning the data collection phase for the present protocol will be conducted through the use of the following methodological tools:

- *Questionnaires* - the following questionnaires are going to be exerted:
  - the Family demographic questionnaire is aimed to the participating parents. It's

objective is to derive a preliminary database of descriptors for the participating pool, to see if there can found any differences among them (e.g.: age related, cultural related, etc…). It is composed by two sections, the first one aiming at collecting general data about the family (age of the parents, level of instruction, age of the child, gender, when s/he was diagnosed with T1DM, etc.) and the second one more focused on exploring the diabetes management dynamics in the family (perceived level of knowledge, communication, relation with the school, etc.). Please refer to *"Family demographic questionnaire.pdf"* for more details.

- Children's quality of life questionnaires (*Pedsql™* – diabetes modules for 8-12 y.o., see documents "*PedsqlTM_8 12_T1DM*.pdf"). The PedsQL™ Measurement Model is a modular approach to measuring health-related quality of life (HRQOL) in children and adolescents with T1DM). Among all the available QoL questionnaires, in the current context is used the PedsQL™ as it is available both in Dutch and in Italian. This feature ensures the comparability of the results obtained.

- Parents' perception of child's quality of life (*Pedsql™* – diabetes modules, see documents and "*PedsqlTM_parents_T1DM*.pdf"). It is the same questionnaire but aimed at the parents of children participating [14].

- Parents' final questionnaire, is aimed at the participating parents and is focused on deriving their opinion on the usability of the PAL system and their impressions on the experience (see "*Parents_final_questionnaire.pdf*").

5. DATA ANALYSIS

The before and after information of the usability test activities with the children were compared with explorative methods. The user experiences of the children, were collected and explored, and user requirements were derived from the data for the further development of the PAL system.

More specifically, the data collected were analysed using the most appropriate analytical methodologies. In the following points they can be found the details of the most common techniques of qualitative and quantitative analysis that were used in the present protocol.

- *Qualitative Analysis* uses analytical categories to describe and explain specific phenomena and reads the data to identify and index themes and categories centring on particular phrases, incidents, or types of behaviour. We can distinguish between general analysis techniques and specific ones (i.e.: related to the methods described in section 4.6).
  - **Specific analysis - Questionnaires.** In this phase of the study a frequency analysis will be used. Regardless of whether manual or automated methods are used to prepare a frequency distribution, it is usually necessary to encode the data numerically to facilitate their subsequent analysis. The frequency distributions summarize and compress data by

grouping it into classes and record how many data points fall within each class, by converting these raw numbers into percentages. The frequency distribution is the foundation of descriptive statistics. It is a prerequisite both for the various graphics used to display data for both basic statistics used to describe a set of data - the mean, median, mode, variance, standard deviation, and so on. The frequency analysis allows to condense and summarize large amounts of data in a useful format and describe all sorts of variables in terms of user needs.

- **General analysis - Video/audio recording and content analysis**. It consists of a simple visualization of the registration of the activity held by the children participating, in order to check all the steps done by them. The qualitative content analysis is defined, in this context, as an approach to a controlled empirical analysis of texts in their context of communication, following the rules of content analysis [12] and pre-defined templates.
- **General analysis – collection of anonimous data –** data collected about the impressions and the suggestions on MyPAL app during an activity such as the brainstorming with the caregivers, will be analyzed in form of a report, in order to collect a list of information.

- *Quantitative analysis:* the use of only qualitative results cannot determine how effectively a system is usable or if there are significant differences in the participating pools; hence the need to apply quantitative studies to test the research hypotheses formulated. The present study will apply, wherever it is possible, different analyses that can range from simple descriptive analysis of data reduction to more elaborate techniques of multivariate associations. Below some examples of possibly used techniques:
  - **Descriptive statistics** shows the central tendency (i.e. the mean, median, mode) and will be used to describe the basic features of the study data. It provides a summary about the sample of the study and its measures of interest, such as the average value of a given parameter, its variability and dispersion and the relative frequencies. Together with the analysis simple graphical forms the basis of almost all quantitative data analysis of study.

6. DATA MANAGEMENT

During the “INTERMEDIATE PHASE - REFINEMENT THE USER REQUIREMENTS ANALYSIS”, through the activities described in Paragraph 4, are going to be collected the following categories of data:

- Common data (personal data, information on lifestyle, hobbies, sporting activities carried out, etc ...) of the population involved in the present protocol;
- Children data that the privacy legislation defines as "sensitive" (information on the health status

and management of the diabetes therapy, such as glycaemic values and insulin doses);

- Feedbacks and insights from the methodologies (described in paragraph 5) used to collect the data, with information on their validation";
- Video and audio recordings of the children interacting with the MyPAL app.

The collection of such data has been carried out after the sign of an appropriate informed consent (see corresponding documents "*5_modulo CI_partecipante_SC_2016*").

Data collected in this Protocol has been managed and stored, either in paper or electronic formats, depending on their different nature. In Italy, they were and will be exclusively used for research purposes related to the PAL project, in accordance with the principles of necessity and not excess. In Netherlands, the modalities and possibilities to sharing data and video will depend on the written informed consent given by caretakers and children and by the ethical committees of the partners involved.

The personal and sensitive data, audio-video recordings of each participant, together with the other data collected during the activities has been identified by a code. With exception of the name, these data has been recorded, processed and stored through such code. Data collected were and will necessarily be shared, in their coded form, to the other partners of the PAL Consortium for the required research.

As we use online servers to store the data needed for the experiment, we have taken security precautions so only partners within the project can access the servers.

7. POTENTIAL THREATS AND COUNTERMEASURES

- Children will not be present at all three appointments because of:
  - External factors, e.g.: child needs to be hospitalized or unforeseen family affairs
  - Internal factors, e.g.: child does not want to use the PAL System
- Children and their families miss one of the three appointments:
  - due to External factors, e.g.: the child is sick
  - due to Internal factors, e.g.: the child/parents is/are bored of using the System and does not want to proceed in the activities proposed
- The Healthcare professionals are not able to participate:
  - due to External factors, e.g.: they are sick; they have emergencies in the ward
  - due to Internal factors, e.g.: the PAL Control Panel is not functioning
- Technological problems:
  - the MyPAL app is unstable
  - the MyPAL app is not properly working
  - the MyPAL app is not easy to understand and usable
  - the PAL robot is not properly functioning
  - the PAL backend system is not stable

Why are these inevitable?

- External factors can happen anytime and the participating end users must be really interested in the activities proposed.
- Internal factors are more difficult to be handled, as the participating pool were free to quit the study anytime, and these unforeseen are due to embedded problems in the System probably difficult to repair in time of the study.

If drops out from the study happen, the defecting end-users will be voluntarily interviewed, in order to discover the reasons of their choice and their impressions on the experience in general. All the information gathered in this way will constitute a valuable soil on which grounding the improvements to the system to be implemented in the next PAL System release

What measures will be taken to alleviate obstacles?

- A detailed description of what was the path of the activities will provided to all the participants before their beginning;
- The researchers, in particular, will explain to the participants that the present was an explorative study, in order to implement the System and, because of that, the System performances were limited. Nevertheless, it was pointed out that for the researchers it was of key importance to have this version tested in order to be able to improve the PAL System in the next project steps;
- The researchers will be available anytime for questions and clarifications by the participants and tried to be as flexible as possible in order to face possible unforeseen (e.g.: need to reschedule one of the appointments)

8. ETHICAL ASPECTS INVOLVED

In addition to some more traditional issues regarding the potential risks and benefits of this research protocol with respect to study participants, some more innovative considerations related to the specificity of this study might be pointed out. We will present the latter, by dividing them in two main categories:

i) *Ethical issues related to roboethics* (in particular, ethical issues related to the implications of the introduction of humanoid robots);

ii) *Ethical issues related to the use of humanoid robots in the biomedical context*.

1) Ethical issues related to roboethics

The characterizing feature of this project (which appears interesting also from an ethical standpoint) is the interaction of a humanoid robot (the so-called NAO) with the research subjects involved in the experiment: children, parents and healthcare professionals. However, since the two latter kinds of subjects will have only indirect contacts with NAO, most ethical concerns that we will highlight

below, regard only the relationship between NAO and the paediatric population. Most of these concerns belong to the interdisciplinary field known under the label of "roboethics", defined as the discipline which analyses and define the complex set of relationships between humans and their intelligent artefacts [19]. Since these latter highly differentiate amongst themselves, the identification of the ethical issues depends upon the definition of the specific type of artefact we are interested in. Within the *Roboethics Roadmap* [18] – the tool aimed at providing a "systematic assessment of the ethical issues involved in the Robotics R&D; to increase the understanding of the problems at stake, and to promote further studying and transdisciplinary research" [18] – a specific section is devoted to presentation of the ethical pros and cons to the use of humanoid robots. In what follows we will present and discuss those issues of the Roadmap that seem to be applicable also in our context and, therefore, that might have some interest for us.

Three benefits have been identified.

First of all, humanoids robots are intelligent machines that can assist humans to perform very difficult tasks, and behave like true and reliable companions in many ways. Translating this first issue in our context, we might claim that NAO appears as a fundamental element in the process of helping the paediatric patient to become more and more as an independent and autonomous agent in his/her therapeutic path.

Secondly, humanoids are robots so adaptable and flexible that will be rapidly used in many situations and circumstances. This is true also in our case. Although humanoids have shown particularly promising in the field of biomedicine (not only in the case of patients affected by diabetes, but also in the one of patients affected by autism), they will be probably be used in several different domains.

Finally, thanks to their shape, they seem able to stimulate those emotional channels which might otherwise be unreachable [17]. Regarding this last aspect, we might observe that it is precisely the physical resemblance of NAO with a young child which seems able to allow a natural and symmetrical relationship between NAO and paediatric patients.

Amongst the disadvantages of the use of robots that might have some relevance also in our context, we might point out the followings: i) reliability of the internal evaluation systems of the robots; ii) unpredictability of robots' behaviour. However, our study seems to be able to exceed both these criticalities. Indeed, on the one hand, this protocol configures as the further development of a preliminary study, presented and approved by the Ethics Committee of this Institution, which preliminary evaluated the protocol in which the first version of NAO was involved. This fact seems able to ensure, at least preliminary, the validity of NAO. On the other hand, we might say that NAO's behaviour is highly predictable, since it has been specifically devised in order to fulfil some very simple tasks; moreover, some trained researchers will monitor the entire course of the experiment so as to be ready to intervene in case of system malfunctioning.

2) *Ethical issues related to the use of humanoids robots in the biomedical context.*

In conjunction to what already said, some additional considerations might be added regarding the specific use of humanoid robots within the biomedical field, in particular when they interact with a paediatric population. In this specific field of inquiry, one of the main ethical issues regards to what extent is it ethically legitimate to substitute the role of humans, in this case healthcare professionals, with the one of machines in a domain, like biomedicine, where the emotional involvement appears to play a central role in the restoration of patient's health.

Moreover, when the patient enrolled appears to be the child, some additional considerations might be pointed out. On the one hand it might be suggested that, by interacting with the robot, the child might develop an emotional attachment with the robot, which, however, will not receive anything in exchange. In other words, there cannot be reciprocity between the emotional investment sustained by the child and the robot. On the other hand, it can be observed that the interaction with the robot might provoke, in the child, the reconfiguration of the relationship with his/her peers, making the same child ever more devoid of those social and human capacities he/she will need in the course of his/her life.

Against this background, it can be argued that precisely the brevity of the child-robot interaction seems to prevent the development of such a scenario.

9. DATA PRIVACY

During scientific conferences or through scientific publications, such data can be disseminated in a totally anonymous form, so without reference to the person either in the form of code. In such circumstances, in case there would be the need to projected the collected video recordings or to show images reproducing the children, the faces will be blanked out to prevent recognition.
For what concerns the data collected during the intervention, they're going to be managed and stored by the staff of Fondazione Centro San Raffaele and TNO (respectively for the Italian and Dutch participants) adopting appropriate security measures to prevent unwanted communication to third parties, their purloining or destruction.
The Ethical Committee, the European Commission, the Italian/Dutch and foreign Health Authorities can ask to look at the audio-video data concerning children participating, in order to assess the correctness and accuracy of the data collected by adopting, in any case, all the precautions so as to ensure the necessary confidentiality.

10. DATA HOLDER

The owner of the data collected during the years of the project are Fondazione Centro San Raffaele and TNO research center. The data collected will be accessible to the partners of the PAL Consortium limited to their competence and research area within the project. At the express request of the parents of the children participating, the collected data must be deleted. All data will be saved

for 15 years.

11. RESPONSIBLE INVESTIGATORS

➢ Netherlands: Olivier Blanson Henkemans (olivier.blansonhenkemans@tno.nl), Sylvia van der Pal (sylvia.vanderpal@tno.nl), Rosemarijn Looije (rosemarijn.looije@tno.nl).

➢ Italy: Francesca Sacchitelli (sacchitelli.francesca@hsr.it), Clara Pozzi (pozzi.clara@hsr.it), Francesco Sardu (sardu.francesco@hsr.it).

REFERENCES

[1] B. S. S. A. Vicini S., «User-Driven Service Innovation in a Smarter City Living Lab,» in *ICSS, International Conference on Service Sciences*, 2013.

[2] H. A. F. C. D. D. T. D. a. H. D. H. Hoey, «Good metabolic control is associated with better quality of life in 2,101 adolescents with type 1 diabetes,» *Diabetes Care*, vol. 24, n. 11, pp. 1923-8, 2001.

[3] L. Scott, «Developmental Mastery of Diabetes-Related Tasks in Children,» *Nurs. Clin. North Am*, vol. 48, n. 2, pp. 329-342, 2013.

[4] J. R. J. M. R. K. K. H. J. S. P. A. D. a. D. D. Y. P. Wu, «Autonomy Support and Responsibility-Sharing Predict Blood Glucose Monitoring Frequency Among Youth With Diabetes,» *Health Psychol. Off. J. Div. Health Psychol. A*, 2014.

[5] J. a. L. T. K. Nielsen, «A mathematical model of the finding of usability problems,» in *Proceedings of ACM INTERCHI'93 Conference*, Amsterdam, 1993.

[6] J. Nielsen, «Usability Engineering,» *Academic Press Inc*, p. 165, 1994 .

[7] J. Nielsen, «Human Computer Interaction, User Testing,» 2012.

[8] [Online]. Available: http://journalofinteractionscience.springeropen.com/articles/10.1186/2194-0827-1-1#CR5.

[9] [Online]. Available: http://journalofinteractionscience.springeropen.com/articles/10.1186/2194-0827-1-1#CR6.

[10] I. I. o. B. Analysis, Babok - A guide to the business analysis body of knowledge, 2015.

[11] J. Brooke, «SUS - A quick and dirty usability scale,» 1996 .

[12] J. &. L. H.-J. Becker, «Inhaltsanalyse - Kritik einer sozialwissenschaftlichen Methode,» Arbeitspapiere zur politischen Soziologie , vol. 5, 1973.

[13] O Blanson Henkemans, MA Neerincx, S Pal, RA van Dam, J Shin Hong, E Oleari, C Pozzi, F Sardu, F Sacchitelli (2016). Co-design of the PAL robot and avatar that perform joint activities with children for improved diabetes self-management. The 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN).

Milano, li ……..…..

Firma P.I. ……..…..

Firma Promotore …….

TRITA TRITA-ICT-EX-2017:187

Ontology-supported Stream Reasoning based on HFC

Christian Willms

Master Thesis

Faculty of Mathematics and Computer Science VI
Department of Computer Science
Saarland University

Supervisor:
Prof. Dr. van Genabith

Advisor:
Bernd Kiefer

Reviewers:
Prof. Dr. van Genabith
Prof. Weikum

Submitted: December 2017

UNIVERSITÄT
DES
SAARLANDES

German
Research Center
for Artificial
Intelligence

Declaration of Authorship

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Datum/Date: _____

Unterschrift/Signature: _____

SAARLAND UNIVERSITY
Department of Computer Science

Abstract

Ontology-supported Stream Reasoning based on HFC
by Christian Willms

In the past decades, the usage of ontologies in systems got more and more popular and they were integrated in a wide variety of applications.
However, many systems often have handle highly dynamical contexts, where runtime data is continuously generated by multiple sensor networks, various online services, and so forth. This leads to the challenge of processing heterogeneous event streams, combining them with background knowledge, i.e., an ontology, as well as continuously performing reasoning operations on them. Another complicating factor is that only a small fraction of these events might actually be relevant, whereas the remaining data is negligible. This thesis will address this issue and present a possible solution.
It presents an ontology-based stream reasoner which utilizes domain-specific rules to detect important events in a stream of events, and which infers new facts based on these rules, the events, and the current state of the ontology providing background knowledge. These facts are then used to update the ontology itself and to feed back inferred facts (events) into the stream.
In contrast to existing approaches, the background ontology will not be restricted by the classical w3c standards, but will support the more general and more powerful n-tuple representation, which allows to add tempo-spatial information to the contextual facts in a lightweight manner. The proposed thesis will evaluate whether this n-tuple based approach leads to performance improvements compared to the few existing triple based stream reasoners.

Acknowledgements

First and foremost, I would like to thank Hans-Ulrich Krieger for all the support and encouragement he gave me throughout the past years. Rest in peace, Uli.
Thanks also to Bernd Kiefer for advising me and making this work possible after all.
Also thanks to everyone who proofread this work. Thank you for finding all those mistakes, misspelled words and weird sentences.
Finally, I would like to thank my family and firends for their support, especially Valerie.
Thank you all!

Content

Chapter 1
Introduction

In the past years, two trends started to become visible. Firstly, the amount of streaming heterogeneous data increased and so did the need for methodologies to manage it, which resulted in the new research field called stream reasoning. Secondly, development and usage of ontologies has been moving from the realm of researchers to practical applications.
However, systems that rely on ontologies, and at the same time process streams of heterogeneous data have to overcome serious technical issues. The research field dedicated to these issues is called RSP, which stands for RDF-Stream-Procesing.
For example, an advanced driver assistance system (ADAS), that is based on an ontology will most likely be dealing with dynamically and especially temporally changing information. At the same time, reasoning support for this kind of emerging information is also not provided in most existing reasoners. Moreover, the rapidly changing context of such a system would lead to data streams containing lots of potentially important information, e.g. sensor data, to be processed in very short time, especially if related to safety-related applications. This makes it necessary to add a filtering layer, i.e. a stream reasoner, extracting only the important information with respect to the background knowledge and adding those new facts to the background ontology.
The presented thesis aims to contribute to the research field of RSP by combining the sophisticated abilities of HFC, an n-tuple based ontology reasoner and repository, with a preliminary stream reasoner.
This stream reasoner, *SHFC*, utilizes a rule-based approach to detect important events in a stream of events, e.g. sensor data in a car system, and infers new facts based on these events and the current state of the background ontology. These facts are then likewise used to update the ontology itself. The rules used in this approach may contain conditions such as combinations of different events, or distinct background knowledge. The collection of information from the stream into binding tables, as known from query engines, is also supported to some extend.
Hopefully, the usage of HFC and its ability to directly "annotate" a relation with additional information and to make the original entailment rules for RDFS [1] and OWL [2] sensitive to it will result in higher versatility and a performance increase, compared to stream reasoning frameworks that rely on traditional RDF/OWL based ontologies.
This work is based on the consideration that a stream reasoner will most likely benefit from an n-tuple based background ontology, as theoretically, this should result in faster and simpler lookups of background knowledge, which is essential for real-time stream reasoning.
To design and develop a system like `SHFC`, several preliminaries have to be discussed and defined. *Firstly*, it is absolutely necessary to ensure that the queries sent from the reasoner to the ontology can be processed within a very short period

of time. It would be optimal, if several queries would be processed in parallel, to ensure an even faster lookup. *Secondly*, it is important to formulate and process queries featuring temporal information in an appropriate manner.
This leads to a number of questions that shall be answered during the course of this thesis:

1. How can we extend the query language of HFC such that temporal facts and relations are respected properly?

2. How can we ensure that those queries are processed in time, i.e. before the events become irrelevant? Which indexing structures or combination of indexing structures should be used?

3. How exactly should the rule language utilized by the stream reasoner look like?

The main goal of this thesis is to construct a practicable stream reasoner. To do so, the following goals, which build on one another, had to be achieved:

1. Integration of an indexing framework including different kinds of basic structures and strategies to HFC.

2. Extending the query framework of HFC towards temporal facts, utilizing the former established indexing framework to enable fast lookups.

3. Implementation of the stream reasoner highly relying on the query framework of HFC.

In the following, the thesis' foundations and implementation will be described and the resulting reasoner will be evaluated. In Chapter 2, general ontologies, their extension towards n-tuples, as well as the forward-chainer HFC will be presented. Additionally, a comprehensive introduction to stream reasoning will be provided. Chapter 3 will describe the implementation of *SHFC* and its subcomponents, which will then be evaluated in Chapter 4. Finally, in Chapter 5, a conclusion and an anticipation of future work will be provided.

Chapter 2
Background

In this Chapter, I would like to cover some basics necessary to understand this work. Amongst others, this chapter will give a basic introduction to ontologies and semantic networks, and the forward-chainer HFC. Finally, stream reasoning and RSP will be covered, with focus on state-of-the-art systems, namely C-SPARQL, CQELS and ETALIS.

2.1 Ontology Standards

There is no general definition of what an ontology in the field of computer science is, nevertheless there are some common aspects among the different interpretations. An ontology is said to be an
"*agreement about a shared, formal, explicit and partial account of a conceptualization*" [3],[4].
Here, "*explicit*" refers to the fact that all elements of an ontology are explicitly defined, whereas "*formal*" means that the ontology specification is given in a language that comes with a formal syntax and semantics, thus resulting in a machine-executable and machine-interpretable ontology representation. Finally, "*shared*" captures the aspect that an ontology is representing consensual knowledge that has been agreed on by a group of people.
Informally, one can conclude that an ontology is a formal description of the knowledge for a specific domain. It provides a shared and common vocabulary, including important concepts, properties and their definitions, and constraints regarding the intended meaning of the vocabulary. Information encoded in a domain specific ontology can be communicated between heterogeneous and distributed applications due to the common and shared vocabulary.
Ontologies have a major advantage over the more common traditional data models, as ontologies consist of relatively generic knowledge that can be reused by different kinds of applications or tasks.
Another asset of ontologies is their ability to perform reasoning tasks. This means that ontologies are able to derive facts that are not explicitly expressed in the ontology by applying logical reasoning, i.e. tableaux for description logic.

Not only their reasoning ability, but also their relative independence of particular applications makes ontologies very popular nowadays. Thus, it does not surprise that the development of ontologies moved from the realm of researcher to the desktops of domain experts, for instance in economics or the automotive industry. This trend led to the development of languages, methods and tools dedicated to ontology design and development. The following sections will cover (1)

standard languages for ontologies and (2) some more elaborated languages and tools adding new features to ontologies.

2.1.1 The standards - RDF, RDFS & OWL

Over the past decades, the World Wide Web Consortium (W3C) introduced several standard languages and frameworks for ontologies, namely RDF[5], RDFS[1] and OWL[6].

RDF (Resource Description Framework) was introduced by the W3C in 1991 as a meta data model and became a standard for conceptual description or modelling of information since then.
An RDF statement is a triple of the form:

```
subject property object.
```

A set of these RDF statements is called an RDF graph. Each statement annotates or describes an entity in terms of named properties. Here, the term entity denotes the most general being, and, thus, subsumes subjects, objects, processes, ideas, etc. In RDF these entities are represented using uniform resource identifiers (URI). An URI consists, in general, of a so called namespace and an unique identifier separated by an #. The values of these properties can be URI-references or literals, i.e., representations of data values (such as integers and strings).
To clarify matters, the following simple example of an ontology representing relations between people, will be introduced. This ontology should model the following facts:

1. Peter is married to Liz.
2. Liz has age 42.
3. Peter has age 44.
4. Peter is a man, whereas Liz is a women.

The corresponding ontology in RDF notion would look as follows:

```
1 <Peter> <bio:marriedTo> <Liz>
2 <Liz> <bio:hasAge> "42"^^<xsd:int>
3 <Peter> <bio:hasAge> "44"^^<xsd:int>
4 <Peter> <rdf:type> <bio:Man>
5 <Liz> <rdf:type> <bio:Woman>
```

A short notion of the URIs namespace is used in the above representation. Full length URIs could look as follows:

```
http://www.example.org/biology#hasAge.
```

The first triple represents the fact that `Peter`(Subject) is `marriedTo`(Property) `Liz`(Object). The second and third triple depict the age of Peter and Liz. However, in contrast to the first triple the age is represented by literals and not by an URI. In the last two triples the `rdf:type` properties are used to represent that `Peter`

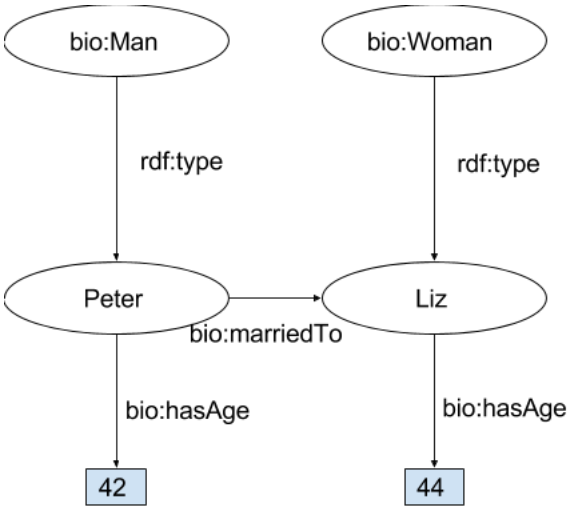


FIGURE 2.1: Graph representation of the example RDF ontology.

is an instance of `Man` and `Liz` is one of `Woman`, respectively. The same ontology could also be represented by the graph depicted in Figure 2.1.

As by itself RDF is only a data model, it does not make any assumptions about what a distinct URI represents, i.e., RDF does not have any semantic information. Thus, RDF is typically used in combination with vocabularies that provide semantic information about these entities. In the following, the vocabularies RDFS and OWL will be presented.

RDFS (Resource Description Framework Schema) defines a data-modelling vocabulary for RDF data and is thus de-facto a semantic extension of the basic RDF vocabulary allowing to define classes (concepts), resources and properties (roles) using predefined resources `rdfs:Class`, `rdfs:Resource` and `rdf:Property`, respectively. RDFS introduces some meta-properties which can be used to represent background assumptions, e.g. taxonomies, in ontologies.
The main modelling constructs provided by the RDF Schema are summarized in the table below:

Construct	Syntactic form	Description
Class (class)	C rdf:type rdfs:Class	C (entity) is an RDF class
Property (class)	P rdf:type rdf:Property	P (entity) is an RDF property
type (property)	I rdf:type C	I (entity) is an instance of C (a class)
subClassOf (property)	C1 rdfs:subClassOf C2	C1 (class) is a subclass of C2 (class)
subPropertyOf (property)	P1 rdfs:subPropertyOf P2	P1 (property) is a sub-property of P2 (property)
domain (property)	P rdfs:domain C	domain of P (property) is C (class)
range (property)	P rdfs:range C	range of P (property) is C (class)

TABLE 2.1: RDFS constructs as shown in the RDF 1.1 Primer [7]

RDFS uses an approach similar to object-oriented programming to specify categories to classify entities. Relations between an instance and its class are stated through the `rdf:type` property. RDFS also introduces the possibility to restrict

the subjects and objects of particular triples to instances of distinct classes by defining domain and range restrictions.
Again, the relation between Peter and Liz is used to show a simple informal example of an RDFS-based ontology:

```
<Peter> <bio:marriedTo> <Liz>
<Liz> <bio:hasAge> "42"^^<xsd:int>
<Peter> <bio:hasAge> "44"^^<xsd:int>
<bio:Man> <rdfs:subClassOf> <bio:Person>
<bio:Woman> <rdfs:subClassOf> <bio:Person>
<Peter> <rdf:type> <bio:Man>
<Liz> <rdf:type> <bio:Woman>
<bio:marriedTo> <rdfs:domain> <bio:Person>
<bio:marriedTo> <rdfs:range> <bio:Person>
```

Note that, while `marriedTo` is a property typically used as the predicate of a triple (as it was in the first examples), properties like this are themselves resources that can be described by triples or provide values in the descriptions of other resources. In the presented example, `marriedTo` is the subject of triples that assign domain, and range values to it.

OWL (Web Ontology Language) is another vocabulary adding semantics to RDF, and is the standard language used for ontology representation. OWL was introduced by the Web Ontology Working Group of W3C [6] because a number of characteristic use-cases for ontologies require a language that offers high expressive power and efficient reasoning support.
However, for such a language there is always a trade-off between those two features, as the richer the language is, the more inefficient the reasoning support becomes, often crossing the border of non-computability. Unfortunately, *efficient reasoning support* is very important, as it allows consistency checks for the ontology and the knowledge, checks for unintended relationships between classes, and allows the automatic classification of instances. Such checks are especially valuable for designing large ontologies, where multiple authors are involved, and ontologies from various sources are integrated and shared.
Thus, a compromise is needed, a language that can be supported by reasonably efficient reasoners, while being sufficiently expressive to define large classes of ontologies and knowledge. Ideally, OWL should also be designed as an extension of RDFS, in the sense that OWL would use the RDF meaning of classes and properties (rdfs:Class, rdfs:subClassOf, etc.), and would add language primitives to support the richer expressiveness identified above.
In short, one can conclude that OWL is intended to be a language with efficient reasoning support and convenience of expression for a language as powerful as a combination of RDF Schema with a full logic, which is not possible without compromises.
Due to this issue, OWL is defined as three different sublanguages, each of which aims towards fulfilling different aspects of these incompatible requirements:

1. **OWL Full** uses all the OWL languages primitives. It also allows to combine these primitives in arbitrary ways with RDF and RDFS. This includes the possibility to change the meaning of pre-defined (RDF or OWL) primitives, by applying the language primitives to each other.

The advantage of OWL Full is that it is fully upward compatible with RDF, both syntactically and semantically, meaning that any legal RDF document is also a legal OWL Full document, and any valid RDF or RDFS conclusion is also a valid OWL Full conclusion.
However, this comes not without any disadvantages, as the language has become undecidable, dashing guarantees on complete and efficient reasoning.

2. **OWL DL** (short for: Description Logic) is a sublanguage of OWL Full which restricts the way in which the constructors from OWL and RDF can be used, i.e., it forbids the application of OWLs constructors to each other, and thus ensures that the language corresponds to a description logic.
This permits efficient reasoning support, but with the drawback that full compatibility with RDF is lost. This means that an RDF structure will in general have to be extended in some ways and restricted in other ways before it is a legal OWL DL document. However, every legal OWL DL document is still a legal RDF document.

3. **OWL Lite** limits OWL DL to a subset of the language constructors. For example, OWL Lite excludes enumerated classes, disjointness statements and arbitrary cardinality. The advantage of this is a language that is both easier to grasp for users and easier to implement for developers. But it comes with the obvious disadvantage of restricted expressibility.

All of these sublanguages use RDF and RDFS to a large extend, i.e, they use RDF for their syntax, instances are declared as in RDF and OWL constructors like `owl:Class`, `owl:DatatypeProperty` and `owl:ObjectProperty` are all specialisations of their RDF counterparts (compare Figure 2.2).

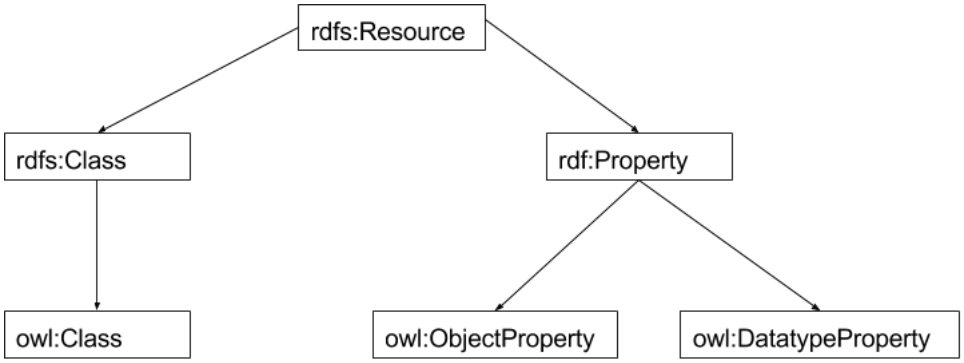


FIGURE 2.2: Subclass relationships between OWL and RDF/RDFS, as seen in [8].

Additionally, OWL comes with a set of entailment rules first proposed by ter Horst[2], allowing native inferencing on OWL ontologies. For instance, the entailment rule for transitive properties, allows to infer from the facts

```
1 <bio:relatedTo> <rdf:type> <owl:TransitiveProperty>
2 <Peter> <bio:relatedTo> <Sam>
3 <Sam> <bio:relatedTo> <Kate>
```

that

Construct	Syntactic form	Description
Class (class)	C rdf:type owl:Class	C (e) is an OWL class
Thing (class)	owl:Thing	most general class
Nothing (class)	owl:Nothing	empty class
ObjectProperty (class)	P rdf:type owl:ObjectProperty	P (e) is an OWL ObjectProperty
DataTypeProperty (class)	P rdf:type owl:DataTypeProperty	P (e) is an OWL DataTypeProperty
disjointWith (property)	C1 owl:disjointWith C2	C (c) is disjoint with C2 (c)
inverseOf (property)	P1 owl:inverseOf P2	P1(p) is inverse of P2 (p)
equivalentProperty (property)	P1 owl:equivalentProperty P2	P1 (p) is equivalent to P2 (p)
TransitiveProperty (class)	P1 rdf:type owl:TransitiveProperty	P1 is a transitive property
SymmetricProperty (class)	P1 rdf:type owl:SymmetricProperty	P1 is a symmetric property
FunctionalProperty (class)	P1 rdf:type owl:FunctionalProperty	P1 is a functional property
InverseFunctionalProperty (class)	P1 rdf:type owl:InverseFunctionalProperty	P1 is an inverse functional property

TABLE 2.2: A selection of OWL constructs.

```
1  <Peter> <bio:relatedTo> <Kate>
```

also holds. For a detailed explanation of the entailment rules, the interested reader is referred to [2]. Table 2.2 describes the most important language primitives of OWL, for an exhaustive introduction please have a look at [6].

```
1  <bio:Person> <rdfs:subClassOf> <owl:Thing>
2  <bio:Man> <rdfs:subClassOf> <bio:Person>
3  <bio:Woman> <rdfs:subClassOf> <bio:Person>
4  <bio:hasAge> <rdfs:domain> <bio:Person>
5  <bio:hasAge> <rdfs:range> <xsd:int>
6  <bio:hasAge> <rdf:type> <owl:DataTypeProperty>
7  <bio:marriedTo> <rdfs:domain> <bio:Person>
8  <bio:marriedTo> <rdfs:range> <bio:Person>
9  <bio:marriedTo> <rdf:type> <owl:ObjectProperty>
10 <bio:marriedTo> <rdf:type> <owl:SymmetricProperty>
11 <bio:Man> <owl:disjointWith> <bio:Woman>
12 <Peter> <rdf:type> <bio:Man>
13 <Liz> <rdf:type> <bio:Woman>
14 <Liz> <bio:hasAge> "42"^^<xsd:int>
15 <Peter> <bio:hasAge> "44"^^<xsd:int>
16 <Peter> <bio:marriedTo> <Liz>
```

The example above uses several key features of OWL. Triple (1) to (3) define a subclass hierarchy from `owl:Thing`, i.e., the most general class, down to `bio:Woman` and `bio:Man`, which are the most specific classes in our example. Next, the triples (4) to (9) are used to define the properties `bio:hasAge` and `bio:marriedTo`. The domain of both properties is restricted to `bio:Person` (lines (4) and (7)), which

means that only instances of this type or one of its subclasses can be used as a subject for this relation. This follows from the entailment rules proposed by ter Horst. The assignment of range values for a property works analogously. The `bio:hasAge` property is of type DatatypeProperty, whereas `bio:marriedTo` is of type `owl:ObjectProperty`. This means that the range of bio:hasAge is restricted to literals, and the one of `bio:marriedTo` is restricted to entities. The fact that the marriedTo relation is declared to be of type symmetric relation (line 10) means that instances of this relation hold implicitly for both direction, i.e. Peter marriedTo Liz also implicates Liz marriedTo Peter. This also follows from the OWL entailment rules.

This was a fairly rough introduction into ontologies and the "standards" introduced by the W3C. Interested readers might have a look at [8], as well as the descriptions provided by the W3C, for a more elaborated explanation.

2.2 Alternative Representations - n-tuples and the HFC reasoner

One can conclude that the decision of the semantic web community to favour RDF, RDFS and OWL has proven to be useful, as they have several advantages. Firstly, general OWL ontologies have been made available that can be easily interfaced with domain-specific ontologies. Secondly, one can use (tableaux-based) description logic reasoners to check the consistency of an ontology or to query its information.

But nevertheless, there are also some issues regarding the descriptive power of OWL if it comes to the representation of dynamic, e.g. temporally changing, information. Similarly, reasoning support for this kind of emerging information is not provided in existing description logic reasoners. These issues led to the development of more elaborated approaches which will be covered in the next section.

2.2.1 Extended OWL Ontologies

In 2015 Krieger and Willms [9] presented an extension to OWL ontologies that tries to overcome the issues described above.
This extension is based on the fact that additional arguments representing, e.g. valid time, transaction time, grading trust etc., are often better modelled in relation and information extraction systems as direct arguments of the relation instances, instead of being hidden in deep structures[1].

Their extension introduces cartesian types, composed from standard OWL classes and XSD datatypes. With cartesian types it is now possible to define m

[1]For example, the so-called *N-ary relation encoding* by Hayes and Welty [10], a W3C best practise recommendation, sticks to triples and uses container objects to encode additional arguments.

+ n-ary relations of the following form, with the domain D and the range R of a binary property p,

$$p \subseteq D_1 \times \cdots \times D_m \times R_1 \times \cdots \times R_n.$$

The extension also provides a further distinct annotation-like notion, called *extra arguments*, which allow to encode additional information, e.g. valid time statements. These extra arguments have a special status, as they are separated from the domain and range of the relation. As an example, the already well known `marriedTo` relation would now have the following signature:

$$\frac{\text{person}}{\text{domain}} \times \frac{\text{person}}{\text{range}} \times \frac{\text{xsd:gYear}}{\text{extra argument}} \times \frac{\text{xsd:gYear}}{\text{extra argument}}$$

Even with both extensions it is still possible to generalize property characteristics introduced by OWL, e.g., symmetry or transitivity. This can be achieved by replacing atomic classes with cartesian types in the corresponding entailment rules (ter Host [2] & [11]) and by adding the extra arguments to them. For a more elaborated explanation please take a look at [9].

A simple extension to the example ontology should clarify the benefits of this approach, compared to classical OWL ontologies. Assuming one would like to represent that Peter and Liz got divorced at some point in time, the corresponding n-tuple would look as follows:

```
1 <Peter> <bio:marriedTo> <Liz> "1999"^^<xsd:gYear> "2010"^^<xsd:gYear>
```

Here the extra argument is used to encode the information in a more convenient way than suggested by the W3C:

```
1 <Peter> <bio:marriedTo> <ppt>
2 <ppt> <rdf:type> <nary:PersonPlusTime>
3 <ppt> <nary:value> <Liz>
4 <ppt> <nary:starts> "1999"^^<xsd:gYear>
5 <ppt> <nary:ends> "2010"^^<xsd:gYear>
```

Here, the relation name is retained, however, the range of the relation changes to a container object.
Cartesian types can also be used in subject or object position. For example one could represent the fact that Bob is a child of Liz and Peter using the following two inverse relations:

```
1 <Bob> <bio:hasParents> <Peter> <Liz>
2 <Peter> <Liz> <bio:haveChild> <Bob>
```

where the relations are defined as:

```
1 <bio:hasParents> <rdfs:domain> <bio:Person>
2 <bio:hasParents> <rdfs:range> <bio:Person> <bio:Person>
3 <bio:haveChild> <rdfs:domain> <bio:Person>
4 <bio:haveChild> <rdfs:range> <bio:Person> <bio:Person>
```

They have a cartesian type (`<bio:Person>` $\times$ `<bio:Person>`) in subject or object position, respectively. Other non-temporal examples of extra arguments might involve space (or spacetime in general), using further XSD custom types, such as point2D or point3D, in order to encode the position of a moving object over time (Keshavdas and Kruijff, 2014 [12]).

2.2.2 HFC

To benefit from the higher expressiveness of the n-tuples model, compared to the classical triple notion, Krieger [13] introduced HFC. It is a bottom-up forward-chainer and semantic repository for n-tuple based ontologies and is comparable to popular RDFS/OWL systems such as Jena[14] and OWLIM[15]. It is capable of RDFS and OWL reasoning, but additionally provides an expressive language RDL (Rule Description Language), allowing the definition of custom rules.
Finally, HFC comes with the query language QDL, which is a SPARQL dialect that, whilst implementing only a subset of the original SPARQL, provides powerful custom aggregates, which are not available in SPARQL.
A structural overview of HFC is given in Figure 2.3. The remaining parts of this section will introduce the key aspects of HFC, namely *buttom-up forward chaining*, *TupleStore and .nt Files*, *RuleStore and RDL* and *Queries and QDL*.

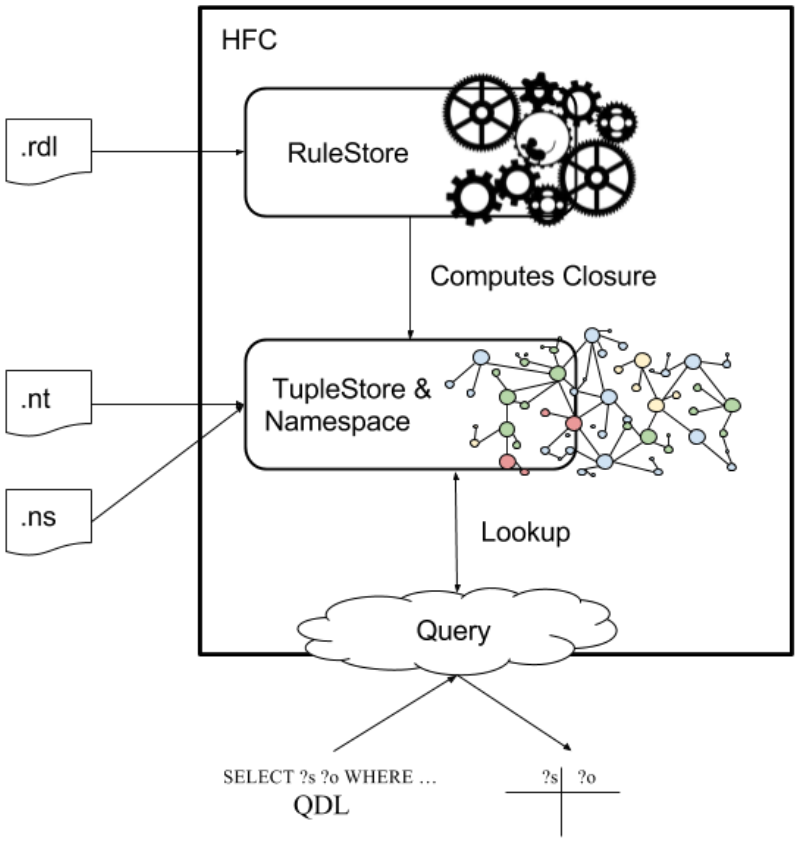


FIGURE 2.3: General structure of HFC.

Bottom-up forward-chaining generally carries out all possible inferences at compile time, such that querying is reduced to an indexing problem at runtime. This process, i.e., the process of making implicit information explicit, is achieved by computing the deductive closure of a set of ground atoms w.r.t. a set R of if-then rules. Bottom-up means in this context that the computation of inferences starts from the ground atoms to which the rules are applied.
Krieger used several optimizations to make forward-chaining scalable, but they will

not be discussed in depth here, as they are not relevant for the understanding of this thesis. For a detailed description of the closure computing and optimizations, interested readers might take a look at [16].

TupleStore and .nt Files The TupleStore is used to store the information represented by an ontology, which is read in from so called .nt files, where the facts are stored in an extended N-Triples syntax, see [17]. This means that all tuples must be finished in a single line, constrained by the following conditions:

- a tuple starts with an URI or a blank node
- tuples must have at least one argument
- elements of a tuple (URIs, blank nodes, XSD atoms) must be separated by the space character ' '
- URIs start with "<" and end with ">" (both for short and long prefixes)
- blank nodes start with "_:"
- XSD atoms start with '"' and end with '"', potentially followed by type or language information
- comments only start with the '#' character at the very first position of a line

An example for a valid .nt file is shown in Listing 2.1.

```
1  # = 1/min
2  <logic:true> <xsd:min-1> <rdf:type> <rdfs:Datatype> "0"^^<xsd:long> .
3  # = calorie
4  <logic:true> <xsd:cal> <rdf:type> <rdfs:Datatype> "0"^^<xsd:long> .
5  # = centimeter
6  <logic:true> <xsd:cm> <rdf:type> <rdfs:Datatype> "0"^^<xsd:long> .
7  # = gram
8  <logic:true> <xsd:gm> <rdf:type> <rdfs:Datatype> "0"^^<xsd:long> .
9  # = international unit
10 <logic:true> <xsd:iu> <rdf:type> <rdfs:Datatype> "0"^^<xsd:long> .
11 # = kilogram
12 <logic:true> <xsd:kg> <rdf:type> <rdfs:Datatype> "0"^^<xsd:long> .
```

Listing 2.1: A valid .nt file

This information is stored using efficient data structures, such as open-address hash tables, integer arrays for tuples, specialized sets with strategy objects to support binding table projection, etc.. The key part of this representation is the fact that each entity in the ontology is mapped to an unique integer, as working with integers in java is much more effective then handling strings. Due to this mapping each tuple read from the .nt File is internally represented as an array of integer values.
To enable quick lookups of the stored information a data structure is needed which is capable of finding all those tuples which contain an object represented as an integer value at a specific position. The structure of choice is a array of mappings, where each index of the array corresponds to the same index in the tuple. The mappings then use the ids of the entities that appear at this position of the tuple

to create a mapping between entities and all tuples in which the entity appears at the position associated with the index, compare Figure 2.4

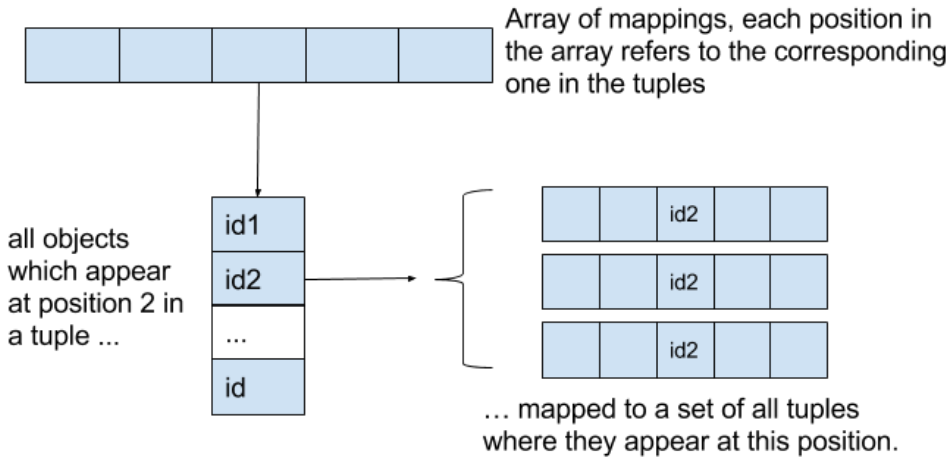


FIGURE 2.4: Indexing structure used in the TupleStore of HFC.

This structure enables good lookup performance, however adapting the structure is not that cheap. In addition, this structure is not suitable for ranged lookups, such as intervals etc.
A detailed explanation on how lookups on this structure are performed will be discussed in connection with the QDL processing.

RuleStore and Rule Description Language (RDL) The RuleStore is responsible for the closure computation at compile time. This closure is computed according to a set of rules defined in the Rule Description Language RDL. The RuleStore reads these rules from so called .rdl files and performs the closure computation according to them. The computation it self is performed in several iteration to insure that rules are also applied to facts which were inferred in former iterations. Usually, the closure computation stops as soon as no more new facts were derived, however to prevent infinite iterations, for example triggered by successor rules, an upper bound for the number of performed iteration is defined.
There are rules that implement OWL entailment as introduced by Hayes[1] and ter Horst[2], but they can also be used to make the entailment rules for RDF and OWL sensitive to extra arguments, such as transaction time. They can even be used to define custom behaviors, such as moving from a point-based encoding to an interval-based one. To achieve this, RDL features extended RDF n-tuple notation, accommodated by functional and relational variables, complex tests and actions.
A rule consists of an *antecedent* or left-hand side(LHS) and a *consequence* or right-hand side(RHS). The transition between antecedent and the consequence is defined by either `->`(if-then or composition rule) or `=>`(aggregation rule). Both LHS and RHS are represented as a sequence of tuples. Each tuple element is either a URI,

```
<rule>          ::= <name> [<pri>] <ante> <imply> <cons> [<test>] [<action>] <cr>
<name>          ::= "$" <nwchar>^+ <cr>
<pri>           ::= "&priority" <+int0> <cr>
<ante>          ::= { <tuple> <cr>}^+
<imply>         ::= { "->" | "=>"} <cr>
<cons>          ::= { <tuple> <cr>}^+
<test>          ::= "@test" <cr> {<pred> <cr>}^*
<action>        ::= "@action" <cr> {<func> <cr>}^*
<pred>          ::= <var> "!=" {<var> | <uri> | <atom>} |
                    <nwchar>^+ {<var> | <uri> | <atom>}^*
<func>          ::= <var> = <nwchar>^+ {<var>| <uri> | <atom>}^*
<var>           ::= "?"<nwchar>^+
<tuple>         ::= {<var>| <uri> | <atom>}^+
<uri>           ::= "<" <nwchar>^+ ">"
<atom>          ::= "\"" <char>^* "\"" ["@" <langtag> | "^^" <xsdtype>]
<char>          ::= any character incl. whitespaces, numbers, even '\"'
<nwchar>        ::= any non-whitespace character
<+int0>         ::= any positive intege, including 0
<cr>            ::= the carriage return/newline character
```

FIGURE 2.5: EBNF defining syntactically valid RDL rules.

an XSD atom, or a variable which is interpreted locally within the scope of the rule.

```
?p rdf:type owl:SymmetricProperty
->
?p owl:inversOf ?p
```

RDL also features options to restrict the applicability of a rule through `@test` clauses. These support inequality constraints, but also arbitrary user definable predicates. Additionally, so called `@action` clauses can be used to define function performing small computations such as shown in the example below.

```
?p <rdf:type> <owl:TransitiveProperty> "0"^^<xsd:long>
?x ?p ?y ?time1
?y ?p ?z ?time2
->
?x ?p ?z ?time
@test
?x != ?y
?y != ?z
@action
?time = LMax2 ?time1 ?time2
```

A formal definition of the *RDL* is given by the EBNF shown in Figure 2.5.

Query Description Language (QDL) is a SPARQL [18] dialect. SPARQL is the de facto standard query language for RDF/OWL based ontologies. In its current state QDL only implements SELECT queries and leaves ASK, CONSTRUCT and DESCRIBE queries aside. But, at the same time QDL supports custom filter predicates, as well as aggregates. An example of a query returning all persons with an age less then 42 will look as follows:

```
SELECT DISTINCT ?s
        WHERE ?s <bio:hasAge> ?o
```

```
1  <query>          ::= <select> <where> [<filter>] [<aggregates>]
2  <select>         ::= "SELECT" ["DISTINCT"] {"*" | <var>^+}
3  <var>            ::= "?"<nwchar>^+
4  <nwchar>         ::= any non-whitespace character
5  <where>          ::= "WHERE" <tuple> {"&" <tuple>}
6  <tuple>          ::= <token>^+
7  <token>          ::= <var> | <uri> | <atom>
8  <uri>            ::= "<" <nwchar>^+ ">"
9  <atom>           ::= "\"" <char>^* "\"" ["@" <langtag> | "^^" <xsdtype>]
10 <char>           ::= any character incl. whitespaces, numbers
11 <langtag>        ::= "de" | "en" | ...
12 <xsdtype>        ::= "<xsd:int>" | "<xsd:long>" |"<xsd:float>" |
13                      "<xsd:double>" | "<xsd:dateTime>" |"<xsd:string>" |
14                      "<xsd:boolean>"
15 <filter>         ::= "FILTER" <constr> {"&" <constr>}^*
16 <constr>         ::= <ineq> | <predcall>
17 <ineq>           ::= <var> "!=" <token>
18 <predcall>       ::= <predicate> <token>^*
19 <predicate>      ::= <nwchar>^+ (e.g., isUri, NoValue)
20 <aggregate>      ::= "AGGREGATE" <funcall> {"&" <funcall>}^*
21 <funcall>        ::= <var>^+ "=" <function> <token>^*
22 <function>       ::= <nwchar>^+ (e.g., Count, Identit, ...)
```

FIGURE 2.6: EBNF defining syntactically legal QDL queries.

```
3          FILTER ILess ?o "42"^^<xsd:int>
```

The SELECT DISTINCT returns only distinct values for `?s`. This is especially useful as within the result table of a query, a column can often contain many duplicates and sometimes one only needs to list the different values. The FILTER statement in this example uses the custom predicate `ILess`, which returns true for all ?o that are smaller than `"42"^^<xsd:int>`.
However, if the query slightly modified by adding an AGGREGATE clause, one can query the absolute number of people having an age under 42.

```
1  SELECT DISTINCT ?s
2  WHERE ?s <bio:hasAge> ?o
3  FILTER ILess ?o "42"^^<xsd:int>
4  AGGREGATE ?number = CountDistinct ?s
```

LISTING 2.2: A valid QDL query.

The custom aggregate `CountDistinct` is used to count all distinct instantiations of `?s` returned by the WHERE and FILTER clause. Now, the resulting table would contain a column with the header `?number` and a single row containing result.
A complete EBNF of the QDL can be found in Figure 2.6.
Internally, these queries are processed according to the work-flow shown in Figure 2.7. We will use the example shown in listing 2.6 to clarify matters. In the first step, the String representation of the query is read in and then separated into the single statements, i.e. SELECT, WHERE, FILTER and AGGREGATE clauses. In the next step, the SELECT statement is internalized, meaning that the target variables for this query are extracted and converted into negative integer values. The `?s` variable used in the example will be assigned to the value -1.
In the following steps the WHERE clauses, and possibly the FILTER as well as the AGGREGATE clauses are internalized. The internalization of the WHERE

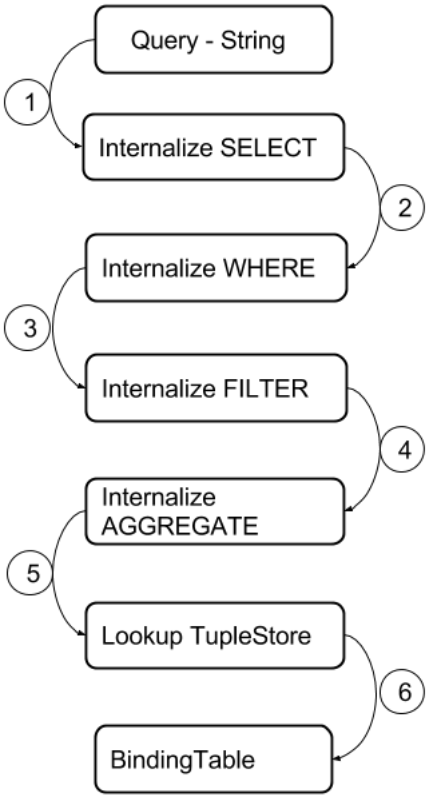


FIGURE 2.7: The QDL processing workflow.

clause results in an integer array, where constants are represented by their integer representation as described in section 2.2.2. Variables are represented using negative integers. These internalized WHERE clauses are also called *patterns*. Thus, the WHERE statements from the example are transformed into $[-1, 2, -2]$ assuming that `<bio:hasAge>` is assigned to the id 2.
Internalizing the FILTER clauses results in sets of variable-variable inequalities, variable-constant inequalities and predicate statements. The FILTER clause used in the example will introduce a predicate statement with the predicate `ILESS` and arguments $[-1, 4]$, assuming `"42"^^<xsd:int>` is assigned to the id 4.
Whereas the internalization of AGGREGATES results in the introduction of new variables whose values are bound to functions applied to the selected values. The internalization of the aggregate statement of the example would for example result in the new variable `?number` associated to id -3 and assigned to the function `CountDistinct` with argument -1. In step 5 each pattern derived from the WHERE clauses is used to lookup all matching tuples. These lookups work according to the algorithm described in Listing 1. For each internalized WHERE clause of a query (pattern) first check whether the pattern exclusively consists of constants (positive IDs). If this is the case, one simply needs to check whether the specified tuple exists in the ontology and return it if so. Otherwise an empty set is returned. If the pattern contains variables, the algorithm iterates over all positions in the tuples and computes the intersection between the tuples that have the specified constant at the given position. Finally, the matches for all patterns are collected and the actual binding table is build by extracting only the selected

variables from the tuples.
The results of all these lookups are then joined to derive one binding table containing the combined results for all WHERE clauses. Finally, FILTER and AGGREGATE statements are applied if available.

Algorithm 1 Querying the tupleStore.

```
1: procedure queryIndex(pattern,vars)
2:     varsOnly = true
3:     length = pattern.length
4:     if vars.size() == 0 then
5:         if tupleStore.contains(pattern) then
6:             return set including pattern
7:         else
8:             return empty set
9:         end if
10:    end if
11:    result = empty set of tuple
12:    query = empty set of tuple
13:    for i in range length do
14:        if pattern[i] > 0 then
15:            varsOnly = false
16:            query = all tuple that have constant pattern[i] at position i
17:            if result is empty then
18:                result = query
19:            else
20:                result = intersection between result and query
21:                if result empty then // no match possible
22:                    return empty set;
23:                end if
24:            end if
25:        end if
26:    end for
27: end procedure
```

Chapter 3
Stream Reasoning

Now, as the basics regarding ontologies are covered, one other important foundation of this thesis, namely stream reasoning and especially RDF-Stream-Processing, will be discussed. The research trend of stream reasoning was introduced almost a decade ago by the Database Group of the Politecnico di Milano [19], which argued that with the growing amount of streaming heterogeneous data the need of new methodologies to manage these data will grow. Nowadays, the field has grown to a multidisciplinary research field, studied by groups with different research focuses, e.g., Semantic Web, Data Management or Artificial Intelligence.
For the sake of this work, this research field will be examined from the Semantic Web perspective by first taking a closer look at what stream reasoning exactly and RSP are and secondly by discussing three RSP engines, namely C-SPARQL, CQELS and ETALIS.

3.1 Stream Reasoning - Terminology

A data stream can be defined as an unbounded sequence of time-varying highly dynamic information, also called events, with the recent information being more relevant than older ones.

Nowadays, such data streams occur in a variety of modern applications, such as the web, or even advanced driver assistance systems (ADAS) in cars. The raw processing of these streams has already been investigated by the database community, and there are specialized DBMS available for nearly all major database products, e.g. Oracle. However, these applications do not allow to make inferences on the data extracted from the streams. Thus Barbieri et al. [19] defined stream reasoning as

the continuous processing of data stream with respect to a rich background knowledge using specialized reasoners in almost real-time.

This kind of reasoning is quite contrary to the traditional work on semantic networks, which assumed a static or only slowly evolving data base. Thus, classical reasoners are based on the assumption that all available information should be taken into account when solving a problem. However, for stream reasoning this assumption has to be adapted and the reasoner has to be restricted to the most recent information. Older information may be ignored or treated as less important.

Currently, there are only very few frameworks and reasoners that can handle highly dynamical data such as those that appear with data streams. Some of these are C-SPARQL[20], C-QELS[21] and ETALIS[22]. Both will be discussed in the next sections.

3.2 C-SPARQL

C-SPARQL [20], short for Continuous-SPARQL, is among the first contributions to the area of RSP, and thus is often cited as a reference work in this field. It is a language for continuous queries over streams of RDF data. This means that queries are registered in the C-SPARQL engine and the queries are then performed continuously, according to the configuration of the engine. The queries consider windows, i.e., the most recent triples of a streams, observed while data is continuously flowing.
Figure 3.1 shows the engine of C-SPARQL, which is capable of registering queries and running them continuously.
C-SPARQL uses a black-box approach to delegate the execution of queries to an underlying data stream management system (DSMS) and an RDF engine. A DSMS is similar to a database management system (DBMS), however it executes a continuous query on volatile data streams. Each query is separated by the C-SPARQL Engine into a structure mapping static and streaming parts of the query to relational bindings. The C-SPARQL engine coordinates the execution of queries, while employing a DSMS and an RDF engine (e.g. Sesame or Jena) against these relations. Semantic-wise C-SPARQL is a pull-based system, where the matches are produced periodically. This means that at every periodically execution new triples enter into the window and old triples exit from the window. Triples arriving in the stream between these points in time are queued until the next execution and do not contribute to the result until then. When executing

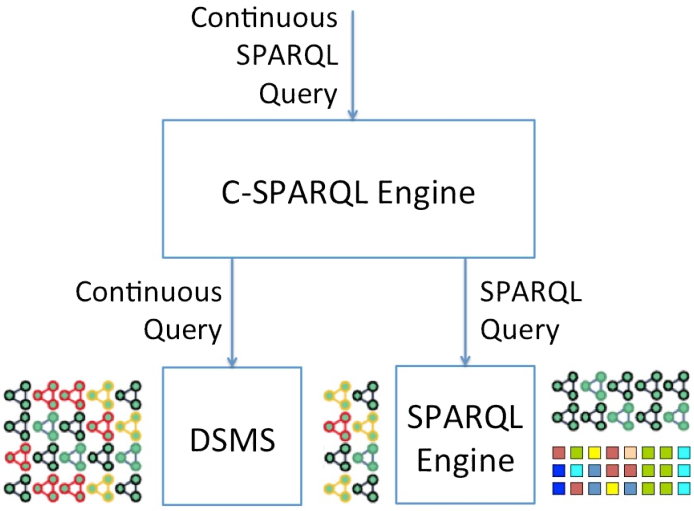


FIGURE 3.1: The C-SPARQL engine as seen in [23]

a query, static and streaming knowledge are joined as follows: First, all triples matching the dynamic relational bindings are extracted by the DSMS from the current window over the stream. Then the remaining static relational bindings of the query are matched against the static knowledge base.

```
1 SELECT ?temp ?id
2 FROM STREAM <http://sensorStream.com/source1 > [ RANGE 3 s STEP 1s]
3 FROM STREAM <http://sensorStream.com/source2 > [ RANGE 3 s STEP 1s]
4 WHERE {
5         ?source :sensor_id ?id.
6         ?source :temperature ?temp.
7 }
```

Listing 3.1: C-SPARQL query.

Listing 3.1 shows a simple C-SPARQL query to obtain the temperature and IDs of sensors from two different input streams. The `FROM STREAM` operator is used to register the RDF streams. Whereas the operators `RANGE` and `STEP` define the sliding window, i.e., defining the size (3s) and the granularity it slides with (1s). An additional static background knowledge-base could be registered using the `FROM` clause. However, there is no option to directly update the used background knowledge with newly derived information.
`CONSTRUCT` and `DESCRIBE` clauses can be used to create new RDF-stream of data derived from the observed stream(s). C-SPARQL also supports other main features of SPARQL such as `OPTIONAL` statements, aggregates and negations, e.g. `MINUS`, for filter clauses.

3.3 CQELS

CQELS [21] is another query-based stream reasoning frameworks providing continuous RSP and utilizes static background data to enrich the accessible knowledge. It is quite similar to C-SPARQL, as both systems are query-based and use a SPARQL like syntax.
However, contrary to the latter one, CQELS uses a push-based evaluation semantics, i.e., the evaluation of the queries is triggered with the arrival of new events within streams, and only newly produced matches are added to the output stream. It uses a white-box approach porting the DSMS concepts, which are realized as blackbox call in C-SPARQL, directly into the SPARQL engine. Due to this structure and additional optimizations, such as adaptive reordering of query operators, it is a much more performance competitive as compared C-SPARQL.
CQELS' query evaluation is realized through so called data flows. A data flow constitutes a directed tree of operators, where the root of the tree is either a relational or a streaming operator, while leaves and intermediate nodes are window and relational operators respectively. Figure 3.2 depicts the data flow for the query shown in Listing 3.2. Here, P_1 and P_2 are defined to be sets of triple patterns for the stream S_1 and S_2, respectively. The matches of each operator are joined together and sent to the output streams S_{out}.

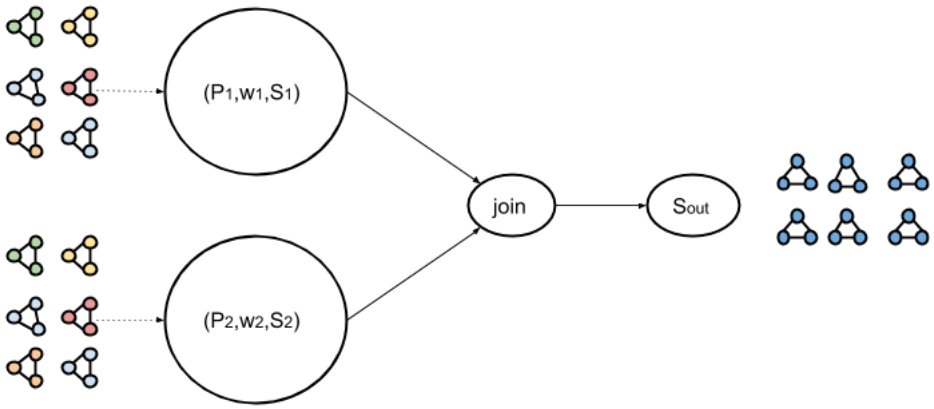


FIGURE 3.2: CQELS data flow for the Listing 3.2.

```
1  SELECT ?temp ?id ?temp2 ?id2
2  WHERE {
3  STREAM <http://streamsensor.com/source1 > [ RANGE 3 s STEP 1s]
4  {
5          ?source :sensor_id ?id.
6          ?source :temperature ?temp.
7  }
8  STREAM <http://streamsensor.com/source2 > [ RANGE 3 s STEP 1s]
9  {
10         ?source2 :sensor_id ?id2.
11         ?source2 :temperature ?temp2.
12 } }
```

LISTING 3.2: CQELS

Listing 3.2 describes the same query as the C-SPARQL example discussed in Listing 3.1 above. However, here the window content is accessed within each `STREAM` clause, allowing the definition of different types of windows for each stream.
Similar to C-SPARQL, new streams can be created by CQELS using `CONSTRUCT` statements. Unlike C-SPARQL, it lacks support for other key features of SPARQL, such as `OPTIONAL` statements and negations for `FILTER` clauses. Directly updating the used background ontologies is also not possible.

CQELS currently is under development, however newer versions are not yet publicly accessible[1].

3.4 ETALIS

ETALIS, a system for Complex Event Processing[24] was developed by Anicic et al.[22]. Complex Event processing is a specific part of stream processing and not necessarily performed on RDF streams. It is a method of tracking and processing streams of data describing events and deriving a conclusion from them. The goal of complex event processing is to identify meaningful events, e.g. opportunities or

[1]https://github.com/cqels/CQELS4J

threats, and respond to them as quickly as possible. Thus, this tool offers more features than the ones presented before.

Conceptual Architecture

The general framework of ETALIS can be described as follows: Within the setting or domain, atomic events from multiple sources take place. Here, an event, which is represented by an RDF-triple, represents something that occurs, or something that changes the current state of affairs. Those events enter ETALIS together with timestamps and possibly further annotations in the order of their occurrence.
Further on, ETALIS combines these atomic events to complex ones depending on their temporal, causal and semantic relations. These relations are encoded in so-called *complex event descriptions* or *event patterns*, which are defined using either a rule-based language called ELE (*ETALIS Language for Events*)[24] or EP-SPARQL (Event Processing-SPARQL). Detected complex events are fed back into the system, either to produce more complex events, or to trigger external actions in a timely fashion. Finally, when detecting complex events, ETALIS may consult the background knowledge for additional information, in case the event pattern requests this.

ELE

The rule description language ELE[24] is one of the languages used to define complex event in ETALIS. However, considering the scope of this work, only the key aspects, such as general syntax and functionality of ELE rules will be discussed here.
ELE supports two different kinds of rules:

1. **static rules**, accounting for static background information about the considered domain.
   Example:

```
1        lessThenFourty(X):-X < 40.
```

2. **event rules**, which are used to capture the dynamic information by defining patterns of complex events.
   Example:

```
1        cep(id) <-
2        (e1(id)SEQ e2(id)).10sec.
```

Both kinds of rules may be intertwined through the use of common variables.

The patterns used to define events are based on Allens interval algebra [25]. Thus, these patterns can also be used for rich temporal reasoning.

Again a small example is used to clarify how the engines syntax looks like. A `temperatureWarning` event is created whenever two events, denoting a overheating for the same sensor, subsequently occur within 10 seconds. The rule below detects such a situation.

```
temperatureWarning(id) <-
(sensor(id,temp)SEQ
 sensor(id,temp‘)).10sec
 WHERE{
 overheating(id,temp),
 overheating(id,temp‘)
 }.
```

A sensor event carries information about the id (id) and the current temperature (temp) of the source sensor. Apart from the temporal condition, which are denoted with the SEQ operator and the 10-second time window, the sensor events need to satisfy other conditions, too. First, they need to be considered for the same sensor (i.e., the two sensor events are joined on the same attribute, id). Second, they need to denote an overheating for this sensor (see WHERE clause in the rule above). To enable an evaluation of these conditions, static rules are used. Further on, a set of temperature thresholds for particular sensors need to be defined, e.g., temperature over 40°C is considered to high for the sensor with `id1` (compare facts below).

```
threshold(id1, 40)
threshold(id2, 25)
threshold(id3, 100)
...
```

The rule below gets information about temp from the sensor events, and evaluates to true if the temp is higher then the threshold for a sensor.

```
overheating(id,temp):-threshold(id,refTemp), temp > refTemp.
```

Now, when a sensor event occurs, followed by another occurrence of the same event, ETALIS checks the time window constraint. If the constraint is satisfied, ETALIS will check whether the temperature is too high (by the overheating rule), in which case a termperatureWarning event is triggered.

EP-SPARQL

Event Processing-SPARQL (EP-SPARQL) is the second language that can be used with ETALIS. It is in fact a wrapper around ELE rules, simplifying the process of manually modeling queries as shown above in ELE. EP-SPARQL is an SPARQL dialect that adds time-dependent event processing capabilities by adding several binary relations, namely `SEQ`, `OPTIONALSEQ`, `EQUALS`, and `EQUALSOPTIONAL`. These are used to combine graph patterns in the same way as `UNION` and `OPTIONAL` are in pure SPARQL. All those operators behave like a selective (left, right or full) join depending on how the constituents are temporally interrelated. Meaning that a statement `P1 SEQ P2` joins output of `P1` and `P2` only if `P2` occurs strictly after `P1`, whereas `P1 EQUALS P2` performs the join only if `P1` and `P2` are exactly simultaneous. `OPTIONALSEQ` and `EQUALOPTIONAL` are temporal-sensitive variants of `OPTIONAL`.
EP-SPARQL also introduces three helper-functions to be used in filter statements.

- `getDURATION()`: returns a literal of type xsd:duration giving the length of the time interval associated to the graph pattern the filter condition is placed in.

- `getSTARTTIME()`: returns a literal of xsd:dateTime giving the start of the currently described interval.
- `getENDTIME()`: returns a literal of xsd:dateTime giving the end of the currently described interval.

The following query is supposed to search for sensors which have an overheated event subsequently in a time frame of five seconds.

```
1 SELECT ?sensor WHERE
2 (?sensor <tr:hasStatus> <overheated>)
3 SEQ(?sensor <tr:hasStatus> <overheated>)
4 FILTER (getDuration() < "P5S"^^<xsd:duration>)
```

ETALIS Workflow

In this paragraph, it will be roughly explained how event detection in ETALIS works, i.e. how events defined in ELE or EP-SPARQL can be detected at run time.
To fulfill this task, ETALIS uses a event-driven backward chaining approach. It starts with a list of goals (Complex Events) and works backwards from the consequent to the antecedent, whenever an event occurs that supports any of these consequences. An inference engine using event-driven backward chaining evaluates a rule whenever an event matching the rules head occurs. Then, the rule will insert a new goal to the memory, denoting that a certain event happened, and the engine waits for another appropriate event. The basic workflow is depicted in Figure 3.3:

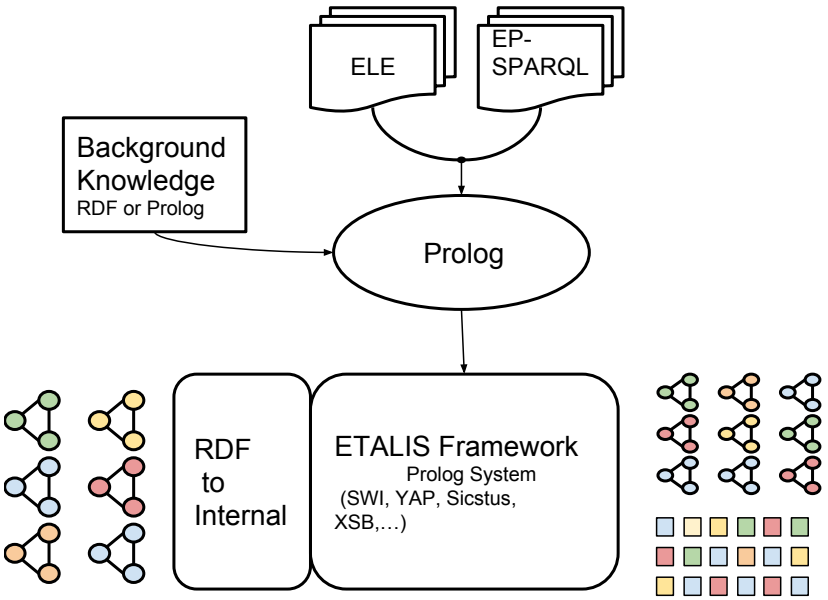


FIGURE 3.3: System Diagram: ETALIS

The system starts with user-written rules specifying complex event patterns according to ELE or EP-SPARQL. ETALIS validates these rules with respect to the language grammar, and parses them. As a result, rules are compiled into executable Prolog rules. These rules are a basic mechanism enabling an inference system to derive a complex event at the moment it occurs.
Complex event patterns may be accompanied with background knowledge to describe the domain of interest expected to be expressed either in Prolog, or in the form of an RDFS ontology.
Compiled rules, together with the domain knowledge, are then executed by a standard Prolog system (e.g., SWI, YAP, XSB etc.). The rules are triggered by events from the streams and continuously derive complex events as soon as they appear. Finally, detected complex events are fed back into the system, either to produce more complex events, or to trigger external events.

3.5 Summary

We have seen three rather different RSP systems, each with its own structure, semantics and functionality.
To provide a better overview and to allow a simpler classification of the new stream reasoner the presented tools were classified in Table 3.1 according to the following criteria.

- Structure: either "black-box" (delegating the execution of queries to underlying DSMS and an RDF engines) or "white-box" (fully integrating DSMS and RDF in the process of query processing).
- Strategy: either "periodically" (query performed as soon as a time window closes, incoming events are not considered until then) or "reactive" (query performed as soon new information arrive).
- Input: The supported input format, mainly RDF.
- Output: The supported output formats.
- Task description: The way the RSP tasks can be defined, e.g. query.
- Background: The way the background knowledge of the RSP can be represented.
- Time aware: Indicating whether the system supports time related reasoning, e.g. Allen's Interval Relations[25].
- Others: Features of the tool not matching the other categories.

Table 3.1: Classification of presented RSP Systems.

	C-SPARQL	CQELS	ETALIS
structure	black-box	white-box	white-box
strategy	periodically	reactive	reactive
input	rdf	rdf	rdf + timestamp
task description	SPARQL-like query	SPARQL-like query	SPARQL-like query or rule
background	RDF/OWL	RDF/OWL	RDF/OWL Prolog
time aware	no	no	yes
output	binding tables, instantaneous RDF graphs, RDF stream	stream of RDF or SPARQL Result format	data, RDF stream
others	supports timed windows	supports timed windows	/

Chapter 4
Implementation

This chapter will give an explanation on how the tasks defined in Chapter 1 were fulfilled. A short reminder, these tasks were:

- The introduction of an indexing framework to HFC,
- Extending the QDL of HFC with Allen's Interval Relations, and
- Developing the actual Stream Reasoning

For each of the mentioned task we will shortly discuss where to place the implementation in the overall framework. In addition necessary background knowledge will be provided before the implementation will be discussed.

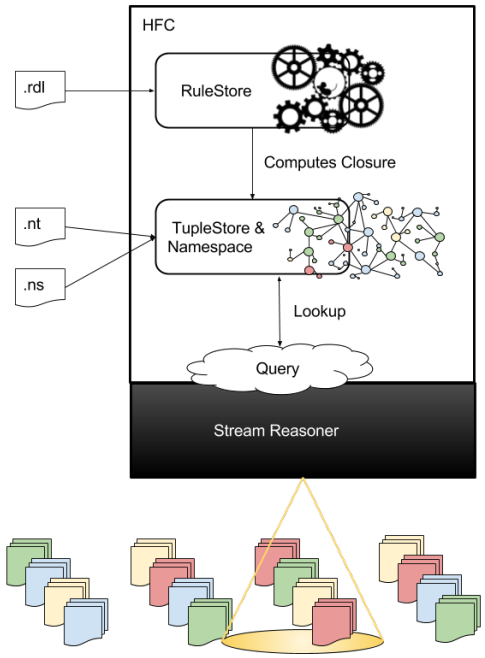


Figure 4.1: General structure of the Stream Reasoner Framework.

4.1 Indexing

The indexing structures are a direct extension of HFC and are one of the pillars of the stream reasoner, as they enable fast and reliable lookups of the background knowledge.
Several structures were integrated, which can handle single instance, but also range lookups. Thus these structures can be used for e.g. Transaction & Valid Time.

The implemented structures are Red-Black Trees, B- and B+Trees, as well as Interval Trees.
Red-Black and B-/+Trees are good all-round structures suited for indexing single keys, such as points in time or spatial information[1]. They provide good performance when looking up single matches. The Interval Tree on the other side, is designed to perform range lookups for which it should perform very well. However, for simple lookups it might not perform as good as the basic structures.
During the course of this section a short introduction to all of these structures and their implementation will be given.

4.1.1 Red-Black Tree

Red-Black Trees were first introduced by Rudolf Bayer in 1972 [26] who called them symmetric binary search trees. Their final name was shaped by Leonidas J. Guibus and Robert Sedgewick [27], which introduced the red-black color convention in 1978. A Red-Black Tree is a self-balancing binary search tree (BST) with the following properties:

- Every node has a color either RED or BLACK.
- The root of the tree is always BLACK.
- A RED node cannot have a red parent or a red child.
- Every path from root to a NULL node has same number of BLACK nodes.

An example of a valid Red-Black Tree, respecting the described properties is shown in Figure 4.2.

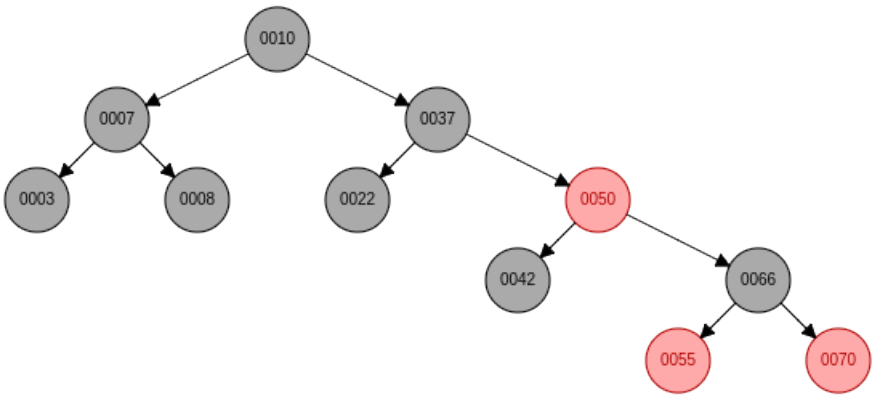


FIGURE 4.2: A valid Red-Black Tree.

[1]Only if linearised.

Search

The search algorithm for Red-Black trees is the same as for regular binary search trees. Reminder: Here, one examines the root and then recursively searches the left subtree if the searched key is less than that of the root, or the right subtree if the key is greater or equal than the root's key.

However insertions and deletions are not that simple, as they usually violate the aforementioned Red-Black Tree properties, which makes it necessary to restore them afterwards.

Insertion

In a Red-Black tree every new node must be annotated with the color RED upon insertion. After every insertion operation, one needs to recheck whether all properties are still fulfilled. If this is not the case the following operations need to be performed to restore its properties again.

1. *Recolor* nodes:
   S, S', and R are nodes, where S is a sibling of S' that is colored RED and R is the parent node for S and P. Then the color of S and S' is changed to BLACK and the color of R is changed to RED. However, if R is the root, in which case R is leaved BLACK to preserve the root property.

2. *Rotate and Recolor* nodes:
   A rotation is a local operation in a Red-Black Tree that preserves in-order traversal key ordering, compare Figure 4.3.

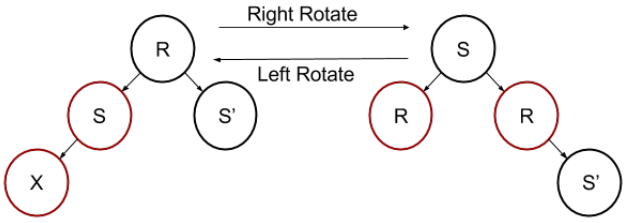


FIGURE 4.3: Rotation followed by recoloring.

More formally the insertion operation is performed using the following steps:

1. If the tree is empty then insert the new node as root node with color BLACK.

2. Else, perform standard BST insertion[2], where the new node is inserted as a leaf node with color RED.

   (a) If the parent node of the new node is BLACK there is nothing to do.

[2]Reminder: Here, one examines the root and recursively insert the new node to the left subtree if its key is less than that of the root, or the right subtree if its key is greater or equal than the root's key.

(b) If the parent node of the new node is RED then check the color of the parent node's sibling.

i. If it is BLACK or NULL, then perform a suitable *Rotation and Recolor* operation.

ii. If it is colored RED, then perform a *Recolor* operation and recheck the trees properties. Repeat cases 2a and 2b until the tree becomes a valid red-black tree.

To clarify this algorithm, an example where we add $3, 7, 10$ and 8 subsequently to an empty Red-Black Tree, as shown in Figure 4.4, will be discussed.
While adding 3 the algorithm simply adds a new root node with value 3, according to case 1. As 7 is added to the tree, it simply adds a new leaf node with color RED (case 2). However, while adding 10 the standard BST insertion results in a tree that clearly violates the third Red-Black Tree Property (A red node cannot have a red parent or a red child.). Thus the algorithm has to perform a suitable *Rotation and Recolor* operation as defined in case 2b, resulting in the tree shown in the lower right part of Figure 4.4. Adding 8 according to the standard BST insertion results again in a structure that violates the red black properties. However, this time a simple recoloring following from case 2a can be used to restore them.

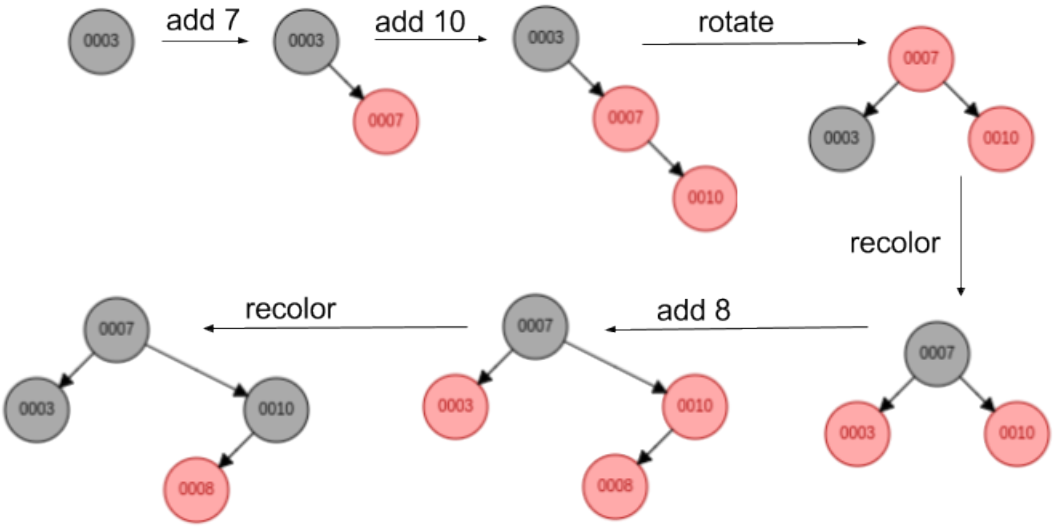


FIGURE 4.4: Adding values 3, 7, 10 and 8 to an empty Red-Black Tree.**Create new picture, this one is too blurred.**

Deletion

In a Red-Black Tree, the deletion is similar to a deletion operation of Binary Search Tree. Simply search the node to be deleted and if its a leaf, remove it. If the node to be deleted has one child, simply replace the deleted node by its child, else find the inorder successor of the node and replace the deleted node with the successor. However, after every deletion operation the properties have to be rechecked. If any of these properties is violated then *Recolor* or *Rotation and Recolor* operation

must be applied.

Assume for example, one wants to remove the keys 7 and 3 from the tree that was populated in the example before. Then, the root note will be removed and replaced by its left child note. Now, a rotate and recolor action must be performed to restore the forth Red-Black Tree property (Every path from root to a NULL node has same number of BLACK nodes.). This results in the structure shown in Figure 4.5.

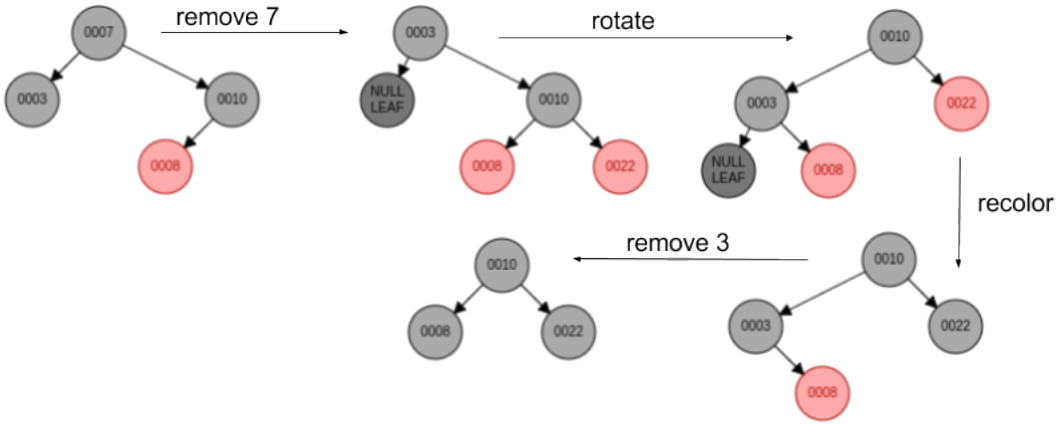


FIGURE 4.5: Removing values 7 and 3 from tree populated in Fig. 4.4. **Create new picture, this one is too small and blurred.**

Implementation

Here, the `java.util.TreeMap<K,V>` is used, which is an implementation of Red-Black Tree to sort keys according to their natural ordering or a comparator. This implementation of the tree was chosen, as the `java.util` implementations usually perform well and were tested extensively.
As every Red-Black Tree with n nodes has height `<= 2 log(n+1)`, the Red-Black Tree is a basic indexing structure suitable for single key lookups. However, due to its internal structure it is not able to handle interval lookups by itself. Nevertheless, a simple wrapper function was integrated, allowing simple range lookups, by using the API of the TreeMap to iterate over the key value pairs in the tree and collecting the values between two given keys.

4.1.2 B-Tree

A B-Tree is a sorted self-balancing tree data structure that can be seen as a generalization of a binary search tree in a way that a node can have more than two children. The number of a node's children is defined by the B-Tree's *balancing factor* `m`. Most of the tree operations (search, insert, delete ...) are performed in time O(log n), where n is the size of the collection. To achieve this, a B-Tree must satisfy the following properties:

- Every path from the root to a leaf has the same length, meaning that all leaves are at the same level.

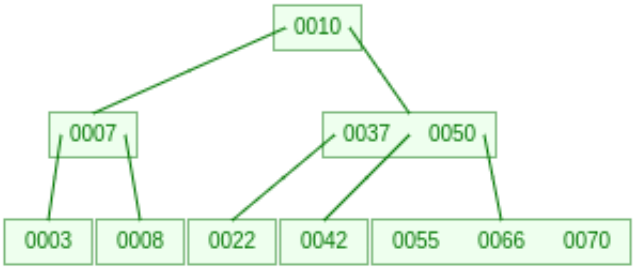


FIGURE 4.6: A valid B-Tree. **Create new picture, this one is too blurred.**

- Each node has at most m children.
- Each internal node has at least $\lceil \frac{m}{2} \rceil$ entries.
- A non-leaf node with m nodes has m-1 keys.
- All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in range from k1 and k2.

There are two possible ways to store data in a B-Tree, depending on its implementation. One could store the values only in the leaf nodes using the keys in the inner nodes as guides, or one can store the values in the inner nodes too, using key value pairs. Here, the latter approach was chosen, as it is more intuitive and the the first one does not offer any benefits compared to it. As an example, Figure 4.6 shows a valid B-Tree.

Search

Searching a B-Tree is straightforward and very similar to the search operations performed on classical BSTs.
If a key X is searched and a node consisting the key-value pairs $V_1 \dots V_k$ is currently observed, then four possible cases can arise:

- If $X < V_1$, recursively search for X in the left subtree of V_1.
- If $X > V_k$, recursively search for X in the right subtree of V_k.
- If $X = V_i$, for some i, then we are done and X has been found.
- If for some i, $V_i < X < V_{(i+1)}$, recursively search for X in the subtree that is in between V_i and $V_{(i+1)}$.

Suppose one is searching for the value associated with the key 42 in the tree depicted in Figure 4.6. Then the search would work as follows: At the root, the second case would apply, so the search is continued in the right subtree. At the root of this subtree, the forth case applies, as 42 is between 37 and 50, and thus the search is continued in the subtree between them. Finally, the third case applies as the searched key was found.

If 47 would have been searched, exactly the same processing steps would have been performed. However, at the last node, the second case would apply, but the subtree to be searches is empty. Therefore the algorithm would conclude that 51 is not in the tree.

The other algorithms for binary search trees, such as insertion and deletion, are generalized in a similar way, but far more complicated as one wants to ensure a perfectly balanced tree. As with binary search trees, inserting values in ascending order will result in a degenerate search tree; i.e. a tree whose height is O(N) instead of O(logN).

Insertion

The insertion process for a new key-value pair X into a B-Tree with balancing factor M consists of three steps:

1. Perform a search for X to find the leaf node n to which X should be added.
2. Add X to n such that $V_1 \dots V_i < X < V_{i+1} \dots V_{m-1}$.
3. Check the size s (number of keys) stored i n
   (a) If $s \leq M - 1$ after adding X, then the algorithm is finished,
   (b) Else n has "overflowed", which needs to be countered by
      i. Splitting n into three parts:
         - the first $(M-1)/2$ key-value pairs (`left`)
         - the key-value pair at position $((M-1)/2)+1$ (`middle`)
         - the last $(M-1)/2$ key-value pairs (`right`)
      ii. Making `left` and `right` to children of `middle`, which is added to n's parent node p according to Step 2.
      iii. Rerunning Step 3 for p.

The example shown in Figure 4.7 will be used to clarify this algorithm. Assume a B-Tree with balancing factor $M = 4$. Adding the values 3, 7 and 10 successively to an empty tree results in a tree consisting of a single node of size 3. However, as an additional value, e.g. 8, is added, the node overflows and is split into 3 nodes as described in Step 3b. Adding more values eventually results in an overflow, and again Step 3b is applied. However, in this case the node in the middle is added to the parent node of the overflowing one.

FIGURE 4.7: An example for B-Tree insertion. **Create new picture, this one is too blurred.**

Delete

The deletion operation on B-Trees is more complex than adding new values, because one can delete a key from any node, not only leafs. Thus, one must ensure that the deletion does not violate the B-tree properties, meaning that we must ensure that a node does not underflow, i.e. all nodes except the root must contain at least the minimum number $\lceil \frac{m}{2} \rceil$ of keys.
The deletion algorithm for a key X formally works as follows:

1. If X is in node n and n is a leaf, delete X from it.
2. If X is in node n and n is an internal node, check
   (a) If child c preceding X in n has at least $\lceil \frac{m}{2} \rceil$ keys, then find the predecessor X' of X in the sub-tree rooted at c. Recursively delete X', and replace X by X' in n.
   (b) If c has less than $\lceil \frac{m}{2} \rceil$ keys, then, symmetrically, examine the child c' following X in n. If c' has at least $\lceil \frac{m}{2} \rceil$ keys, then find the successor X' of X in the subtree rooted at c' and recursively delete X', and replace X by X' in n.
   (c) Else, if both c and c' have only $\lceil \frac{m}{2} \rceil - 1$ keys, merge X and all of c' into c. Then free c' and recursively delete X from c.
3. If the X is not in node n and n is an internal node, determine the root c of the appropriate subtree that must contain X.
   If c has only $\lceil \frac{m}{2} \rceil - 1$ keys, execute Steps 3a or 3b as necessary to guarantee descends to a node containing at least $\lceil \frac{m}{2} \rceil$ keys. Finish by recursively repeating this step on the appropriate child of n.
   (a) If c has only $\lceil \frac{m}{2} \rceil - 1$ keys but also has an immediate sibling with at least $\lceil \frac{m}{2} \rceil$ keys, borrow c an extra key by moving a key from n down into c, moving a key from cs immediate left or right sibling up into n, and moving the appropriate child pointer from the sibling into c.
   (b) If c and both of its immediate siblings have $\lceil \frac{m}{2} \rceil - 1$ keys, merge c with one sibling, which involves moving a key from n down into the new merged node to become the median key for that node.

As this formal definition is not that intuitive, here a short explanation using an example based on the tree shown in Fig 4.8. By removing 7 one triggers an underflow as the node n former containing 7 now contains less then 1 ($\lceil \frac{m}{2} \rceil - 1$) keys. This leads to a merge of its child notes (case 2c). Additionally the key 10 is borrowed from the root node, which was then replaced by the smallest key of n's right sibling (case 3a). Finally all pointer are updated.

Implementation & Usage

As seen in the previous sections, it can be quite expensive to add or remove values to/from a B-Tree. Thus, a B-Tree should only be used in scenarios where (i) the indexed data base is static or only slowly evolving and (ii) many search operations

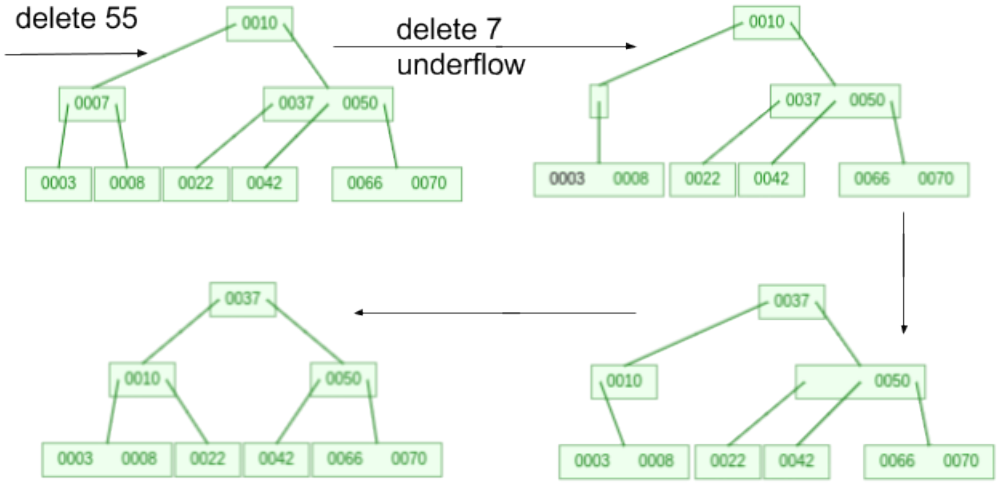


FIGURE 4.8: An example for B-Tree deletion.**Create new picture, this one is too blurred.**

are performed.

The implementation in this work is inspired by the one published by Robert Sedgewick and his group at Princeton[3]. However, it was adapted such that key value pairs are used to store the data in inner and leaf nodes, instead of storing them only in the leaf nodes. Additionally, a function performing simple range lookups was added. The implementation of the range lookups is quite simple and works as follows:
Standard B-Tree searches for the nodes that should contain the start and end of the range respectively are performed. Whenever the algorithm visits a node during these searches all values, including values of subtrees to the right, when searching the start, or the left, when searching the end of the range, are collected. In case both searches visit the same node, this collection process is only performed once. This way all entries in the tree between start and end can be covered.

4.1.3 B+Tree

A B+Tree[28] is a B-Tree in which each node contains only keys (not key values pairs), and where all leaf nodes are linked, similar to a linked list.
A benefit of B+Trees compared to classical B-Trees is the fact that linked lists between the leaf nodes support range queries. The B+Tree must satisfy the following additional properties, compared to a B-Tree:

- All data is stored in the leaf nodes.
- All leaf nodes have links to their left and right neighbors.

For example Figure 4.9 shows an B+Tree satisfying these properties.

[3]http://algs4.cs.princeton.edu/code/

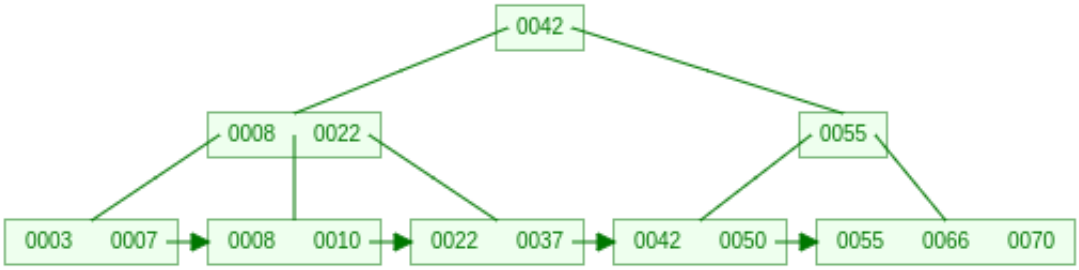


FIGURE 4.9: A valid B+Tree.**Create new picture, this one is too blurred.**

Search

As for B-Trees, searching B+Trees is straightforward. Starting form the root, the algorithm is looking for the leaf which may contain the value associated with key X. At each node, it figures out which internal pointer we should follow. This leads it to the searched leaf and so to the searched value.

If a key X is searched and currently a node n consisting the keys $V_1 \dots V_k$ is observed, then five possible cases can arise:

- If $X < V_1$, recursively search for X in the left subtree of V_1.
- If $X > V_k$, recursively search for X in the right subtree of V_k.
- If $X = V_i$ for some i and n is a leaf, then X has been found.
- If $X = V_i$ for some i and n is **no** leaf, recursively search for X in the right subtree of V_i.
- If for some i, $V_i < X < V_{(i+1)}$, recursively search for X in the subtree that is in between V_i and $V_{(i+1)}$.

Insertion

The insertion process for a new key-value pair X into a B+Tree with balancing factor M is formally defined as:

1. Perform a search for X to find the leaf node n to which X should be added.
2. Add X to n such that $V_1 \dots V_i < X < V_{i+1} \dots V_{m-1}$.
3. Check the size s of n
   (a) If $s \leq M - 1$ after adding X, then we are finished,
   (b) Else n has "overflowed", which is countered by
      i. Splitting n into two halves of same size
      ii. Allocating a new leaf node and move half the keys to the new node, link the node to its neighbors
      iii. Inserting the new leaf's smallest key into the n's parent node p.

iv. If the p is full, split it too, and repeat the split process above until a parent is found that need no split.

v. If the root splits, create a new root which has one key and two children..

Again, an example will be used to visualize the algorithm, compare Figure 4.10. In this example the values 3, 7, 10, 8, 6 and 5 will successively be added to an empty B+Tree. Adding the first three values results in a single node containing the values 3, 7 and 10. As soon as 8 is added, the node overflows resulting in two new subtrees produced by splitting the overflowing node. The new root of the tree contains the key 8 which is used as guide when searching the tree. Remember, unlike in B-Trees, inner nodes in B+Trees do not contain key value pairs. Therefore a key-value pair for 8 can be found in the right subtree. Adding the value 6 is now straightforward, as it is simply added to the left child node. However, adding 4 triggers an overflow that is handled by splitting the node and adding the middle key to the parent node.

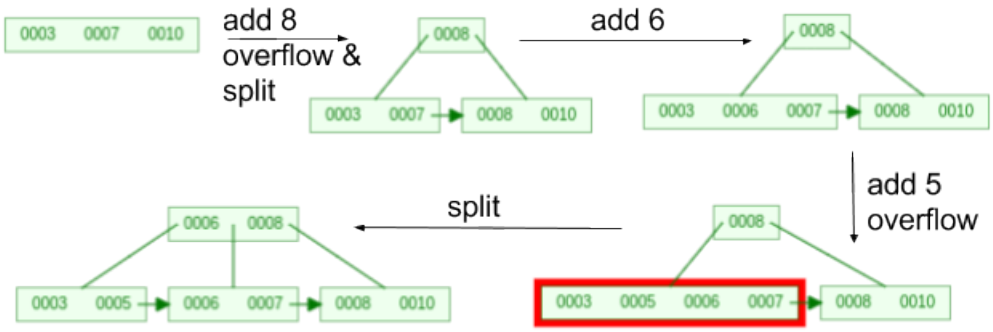


FIGURE 4.10: Adding values 3,7,10,8,6 and 5 to an empty B+Tree. **Create new picture, this one is too blurred.**

Delete

The deletion algorithm for an existing key X from a B+Tree with balancing factor M is formally defined as:

1. Perform a search to determine which leaf node l contains the key.

2. Remove the key-value pair from l, if the removed key was used as a navigation key in the parent node, replace the navigation key with the smallest one from l

3. If for the number of nodes m in l it holds that $m \geq \lceil \frac{m}{2} \rceil - 1$, i.e., it is at least half-full, done!

4. Else, try to borrow a key from a sibling node s (adjacent node with same parent)

   (a) If s is the left sibling of l, then borrow the last key of s, and replace their parent's navigate key with this borrowed key.

(b) If s is the right sibling of l, then borrow first key of s, and replace their parent's navigate key with s's second key.

(c) If borrowing a key from s is not possible, then merge l and s, then delete their parent's navigate key and proper child pointer.

5. Repeat step 4 on parent node, if necessary.

The deletion algorithm of B+Trees is not that different from the one for B-Trees, as demonstrated in the short example shown in Figure 4.11. Assume one successively remove the keys 66, 55, 70 and 50 from the tree shown in Figure 4.9. Removing 66 is no problem, as it is not used as an guide in any inner node (Step2). Removing 55 also is not that problematic, however the guide in the parent node must be replaced by 70, which is the last available node in the leave node(Step 2.a). When removing 70 an underflow in the effected leave node is triggered. Therefore, the greatest key value pair, i.e. 50, from the left sibling node is borrowed and the key for 70 in the parent node is replaced with the borrowed key (Step 4.a). Finally, when removing 50, the tree becomes unbalanced, which is fixed by borrowing a node from the left subtree and successively updating the parent nodes. Therefore, the new rule node contains 22, whereas the root of the right subtree contains the guide 42 (Step 4.c).

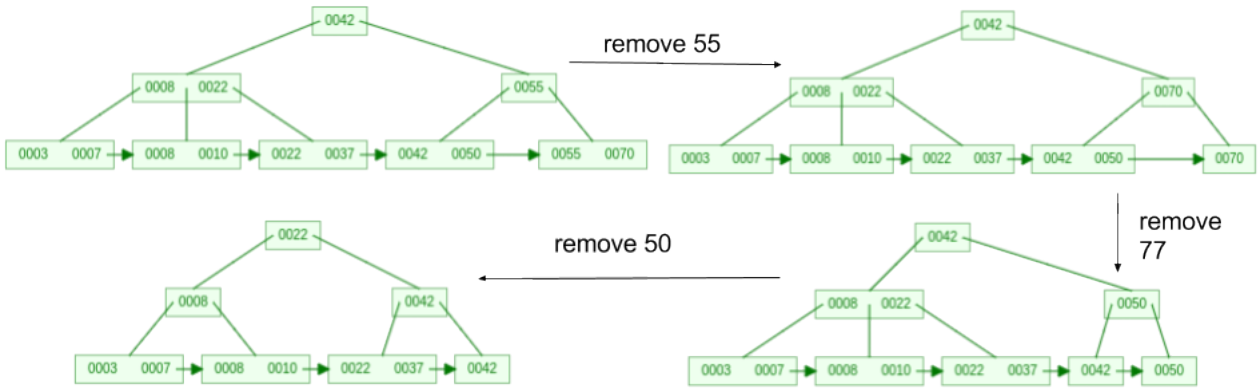


FIGURE 4.11: Removing keys 66, 55, 70 and 50 from the tree shown in Figure 4.9.**Create new picture, this one is too blurred.**

Usage & Implementation

The B+Tree implementation in this work follows the definition presented above. However, not only the leaf nodes are interconnected, but each inner node also knows its left and right neighbours. This makes it easier to borrow nodes in the event of an underflow.
The B+Tree is an advanced indexing structure suitable for single key lookups. However, due to the linked leaf nodes it should also be capable of performing range searches with an reasonable performance. Intervals can be looked up by searching the leaf node that contains or should contain the start key of the interval. Then starting from the start key or the next higher key in this node traverse to the right and collect the associated value for each key until the end key of the interval or a higher value was found.

4.1.4 Interval-Tree

The Interval Tree was first introduced by Cornen et al. [28] and is a (self-balancing) Binary Search Tree to maintain a set of intervals. It is an ordered data structure whose nodes represent the intervals and are therefore characterized by a start and an end key. The start key of an interval is used as key to maintain order in BST. Several implementation of Interval Trees possible, as it is a variant of self balancing BSTs. Thus implementations could be based on every self balancing BST, e.g. Red-Black or B-Trees.
The Interval Tree follows the properties of (self-balancing) BSTs, with the difference that every node of Interval Tree stores the following information.

- An interval which is represented as a pair [start, end]
- Maximum high value in subtree rooted with this node.

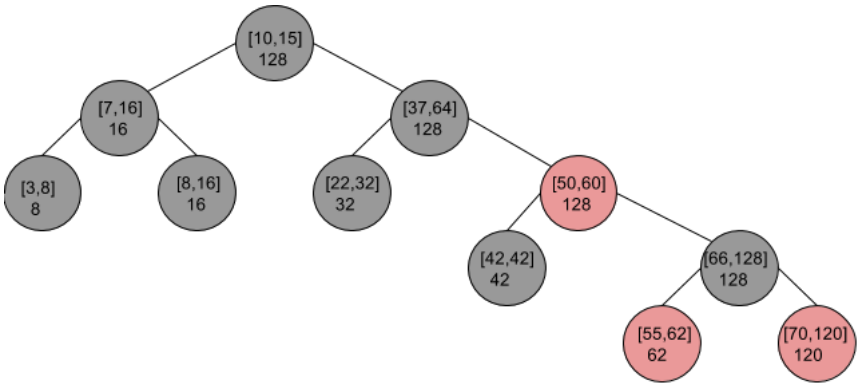


FIGURE 4.12: A valid Interval Tree based on a Red-Black Tree implementation.**Create new picture, this one is too blurred.**

Search

The algorithm for searching overlapping intervals works as follows: Assuming we observe a node n and search intervals overlapping with an given interval $X = [x_l, x_h]$ then

1. Check if X overlaps with n's interval $N = [n_l, n_h]$, i.e. whether $x_l <= n_h$ && $n_l <= x_h$. If true return N.
2. Else, check if left child c_l of n is not empty and the max key in left child is greater than x_l value, perform Step 1 on c_l
3. Else, perform Step 1 on the right child c_r.

If, for example, an interval overlapping with $[17, 28]$ is searched in the tree shown in Figure 4.12, then starting at the root, the algorithm first checks whether the interval stored in the node overlaps with the searched one. This is not the case, so it traverses into the right subtree as the max value of the left one (16) is

lower than the start value of the searched interval (Step 3). The next visited note contains the interval $[37, 64]$. Again no overlapping is found. However, as the max value of the left subtree is higher than the min value of the searched interval Step 2 can be applied leading to the overlapping interval $[22, 32]$.

Insertion & Deletion

The Insertion and Deletion algorithms for Interval Trees are very similar to the ones used in their basic structure, e.g. Red-Black Tree. The only difference is that one has to ensure that the max keys stored in the parent nodes are correctly updated, whenever a new node introduces a new max key or a node providing a max key was removed. This can be easily achieved using recursive approaches.

Implementation & Usage

Interval Trees were introduced in this paper to index valid time statements. In addition, they should enable an effective calculation of Allen's interval relations on these statements.
For this work an Interval Tree based on the Red-Black Tree design was implemented. However, it is planned to also implement a B-Tree based one in future versions of the Stream Reasoner/HFC.

4.1.5 Implementation/IndexStore

All the described data structures were added to an indexing framework, which is incorporated into HFC. The diagram depicted in Figure 4.13 shows the structure of the new indexing framework and its place within HFC[4]. The IndexStore is a

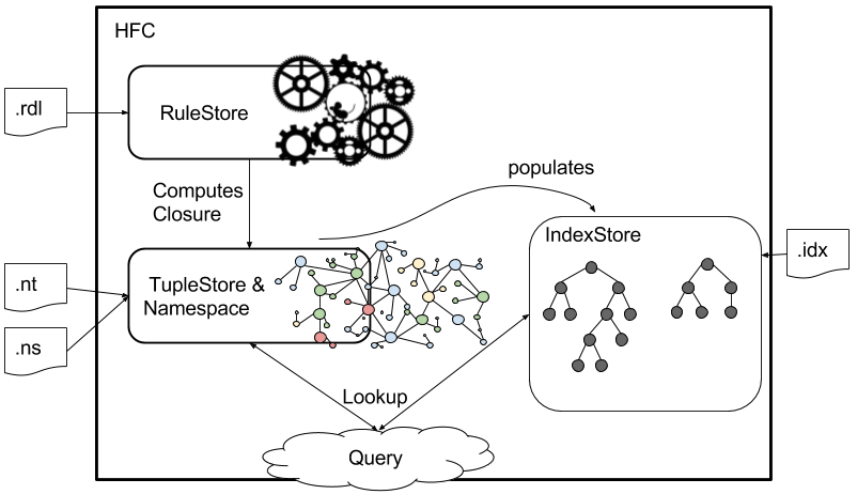


FIGURE 4.13: HFC's structure including IndexStore.

structure used to maintain the indexes. It is an complementing addition to HFC meaning that HFC can also be used without a defined IndexStore and thus without

[4]Note this section is not about the actual Java implementation of the indexing framework. Readers interested into the implementation might take a look in the code**ref zu repo.**

indexing structures. These date structures can be configured using so called .idx files. For example, the .idx file given in Figure 4.14 would define two different indexes. The first one uses an B-Tree to index all occurrences of atoms with type `<xsd:date>` at the first position of a tuple and the second one utilizes an Interval Tree to index intervals of type `<xsd:Date>` defined by atoms located at the 5th and 6th position of an tuple.

```
1 #Comment indicating that the index defined below corresponds to - transaction time
2 Key_Type1: XsdDate
3 Key_Position1: 0
4 Structure1: BTree_Index
5 #Comment indicating that the index defined below corresponds to - valid time
6 Key_Type2: XsdDate
7 Key_Position2: 4,5
8 Structure2: IntervalTree_Index
```

FIGURE 4.14: Example for .idx file.

The IndexStore is theoretically able to maintain an arbitrary number of indexing structures. However, the number of supported structures was restricted to up to two structures, as I think the cost for maintenance and the memory consumption of more indexes would reduce the overall effectiveness. Two structures should be more than sufficient for a wide variety of task. For example one index could be used for valid time indexing, whereas the other one can be used for transaction time.
Both, the TupleStore and IndexStore, have a connected population process to ensure consistency between both structures. The algorithm used to populate the indexes is straightforward:
Whenever a new tuple is added to the TupleStore, it is also checked whether the tuple matches any of the indexing keys (correct type at the correct position in the tuple). If this is the case, an appropriate entry is added to the corresponding index. For example assume the tuple

```
1 "2017-08-01"^^<xsd:date> <bio:Peter> <bio:marriedTo> <bio:Liz>
2         "2001-01-01"^^<xsd:date> "2017-08-01"^^<xsd:date>
```

is added to the ontology. Assume further that the IndexStore is defined according to the .idx file shown in Figure 4.14. First, the TupleStore is updated with this information, if this is done and the tuple was internalized, the internalized tuple is used to update the IndexStore. This results in the fact that a new key `"2017-08-01"^^<xsd:date>` associated with the internalized tuple as value is added to the B-Tree and the interval `["2001-01-01"^^<xsd:date>, "2017-08-01"^^<xsd:date>]` is added as key to the Interval Tree, again associated with the internalized tuple as value. Unfortunately the Id representations of indexed elements as keys could not be used, as they do not provide the absolutely necessary comparability of keys. Thus all newly introduced structures are Java Objects representing XSD:types as keys and sets of Integer arrays representing tuples as values.
The newly introduced indexing framework can be used as an complementing addition for HFC's original indexing structure and should be especially useful when querying the ontology. To benefit from these indexing structures the query language QDL of HFC was also extended, which we will discuss in the next section.

4.2 Query-Extension

The QDL (Query Description Language) of HFC was extended to take advantage of the newly introduced indexing capabilities. The extension includes support for Allen's Interval Relations [25], as well as syntactic sugar for looking up all values associated with keys in a given range or interval. During the course of this section a short introduction to these new features, their implementation and the overall structure of the extended QDL and HFC will be presented.

4.2.1 Range Lookups

As mentioned in the introduction to this section intervals were introduced as syntactic sugar for looking up values that are in a given range. A short example will introduce the new feature before its syntax, semantics and the actual computation will be explained.
So, lets have a look at the example shown in Listing 4.1.

```
1 SELECT ?s WHERE ?s <bio:hasAge> [18,64) & ?s <rdf:type> <bio:Person>
```

Listing 4.1: Query featuring a simple interval.

The query presented in Listing 4.1 will return all persons `?s` that are in an age between `18` and `64`. This is implemented using the interval in the object position of the `<bio:hasAge>` statement, restricting the age to be greater or equal to `18` and less than `64`.

Syntax

Syntactically these intervals are defined as follows:

```
1         <interval>      ::= {"["|"("} <atom> "," <atom> {"]"|")"}
```

Each interval must either start with an `[` or `(`, followed by two atoms which are separated by an `,`. The interval is closed by either `]` or `)`.
These intervals can be used to restrict an element of an WHERE clause and may be used at every position of such a clause.

```
1         <where>         ::= "WHERE" <tuple> {"&" <tuple>}
2         <tuple>         ::= <token>^+
3         <token>         ::= <var> | <uri> | <atom> | <interval>
```

Semantics

This intervals work same as the well known mathematical intervals: Open intervals, which are indicated by parentheses, do not include their endpoints Closed intervals however include endpoints, and are denoted with square brackets. For example, [0,1] means greater than or equal to 0 and less than or equal to 1. Also combinations of open and closed endpoints are possible, implying that the interval includes only one of its endpoints.
To ensure proper, computation the atoms used in an interval must be comparable and of the same type. Otherwise the interval can not be processed.

Definition 4.1. Given a strict partial ordering $O = \texttt{<}O\texttt{,}<\texttt{>}$, an interval in O is an ordered pair s,e enclosed by parentheses P_1, P_2 such that $s, e \in O$, $s \leq e$, $P_1 \in \{(, [\}$, and $P_2 \in \{),]\}$. An atom a belongs to an interval $I = P_1\ s,\ e\ P_2$ if

$$
\begin{aligned}
s \leq a \leq e \ \texttt{in case}\ I &= [s, e] \\
s < a \leq e \ \texttt{in case}\ I &= (s, e] \\
s \leq a < e \ \texttt{in case}\ I &= [s, e) \\
s < a < e \ \texttt{in case}\ I &= (s, e)
\end{aligned}
$$

Computation

The observant reader might have noticed that these intervals can easily be replaced using FILTER clauses. actually, these intervals were only introduced as syntactic sugar to avoid these FILTER clauses and make queries more readable. Instead of relying on FILTER clauses, one could use this opportunity to examine whether it is feasible to run range searches directly on the indexing structures. A comparison of both approaches might prove interesting. To enable these simple range lookups two of the basic indexing structures, i.e. B+Tree and Red-Black Tree, were extended by a corresponding function, as described in sections 4.1.1, 4.1.2 and 4.1.3.
However, if no index is defined the intervals will be rewritten by replacing them with a new variable and introducing new filter clauses mimicking the interval's behaviour. Let an interval I be defined according to Definition 4.1, then the rewriting algorithm for interval statement in queries is defined as:

1. Replace all occurrences of I with a newly introduced variable v.
2. Create new filter clauses f_1 and f_2 depending on P_1 and P_2:
   (a) if $P_1 ==$ (then $f_1 =>\ v\ s$ else $f_1 =\geq\ v\ s$
   (b) if $P_2 ==$) then $f_2 =<\ v\ e$ else $f_1 =\leq\ v\ e$

where $<, \leq, \geq, >$ are the comparing operations of ordering O and where v is a new unique variable.
Rewriting the interval used in 4.1 would result in the following query (4.2):

```
1 SELECT ?s  WHERE ?s <bio:hasAge> ?ref & ?s <rdf:type> <bio:Person>
2           FILTER IGREATEREQUAL ?ref 18 & ILESS ?ref 64
```

Listing 4.2: Rewriting the query from Listing 4.1.

4.2.2 Allen's Interval Relations

Allen's Interval Relations (AIR) were also introduced as a new feature to the QDL. Thereby the temporal changing information in the stream reasoning context can be respected more easily. The relations can be used to search for all entries in an

ontology, where for example the valid times are related in a distinct matter, e.g. during.
First, the interval relations will be explained. Then an example will be used to explain the syntax, semantics and the integration of this new feature into the QDL framework.

Background

In 1983 James F. Allen [25] introduced an algebra for reasoning about temporal relations. The calculus defines 13 possible relations between time intervals that can be used as a basis for reasoning about temporal descriptions of events, see Table 4.1.

Relation	Inverse	Illustration	Interpretation
X <Y	Y >X		X takes place before Y.
X = Y	Y = X		X is equal to Y.
X m Y	Y mi X		X meets Y.
X o Y	Y oi X		X overlaps with Y.
X d Y	Y di X		X takes place during Y.
X s Y	Y si X		X starts Y.
X f Y	Y fi X		X finishes Y.

TABLE 4.1: Allen's Interval Relations.

Note, that these 13 relations are mutually exclusive and jointly exhaustive, meaning that exactly one of Allens Interval Relations holds between any given pair of strict intervals. Formal definitions on Allen's Interval Relations will be presented in the following section. However, these definitions use the notion of AIR used in the extended QDL.

Integration in QDL

A valid query utilizing one of Allen's relations is shown in Listing 4.3.

```
1 SELECT ?s ?o WHERE ?s <bio:marriedTo> ?o D "1991"^^<xsd:int> "2010"^^<xsd:int>
2                  & ?s <rdf:type> <bio:Person>
3                  & ?o <rdf:type> <bio:Person>
```

LISTING 4.3: Example query using Allen's Interval Relations.

The query returns a mapping between all persons ?s and ?o who were married during `"1991"^^<xsd:int>` and `"2010"^^<xsd:int>`. This is implemented using the statement `D "1991"^^<xsd:int> "2010"^^<xsd:int>` in the first WHERE clause. This statement represents the *during* relation and matches all intervals that are contained in this interval.

Syntax

As can be seen from this example, the syntax for Allen's Interval Relation in QDL is quite simple. An relational statement is initialized using a keyword describing the relation. This keyword is followed by two atoms.

```
1 <relation>      ::= <relKey> <atom> <atom>
2 <relKey>        ::= "Bf" | "Af" | "Eq" | "S" | "Si" | "M" | "Mi" | "D" |
3                     "Di" | "O" | "Oi" | "F" | "Fi"
```

Listing 4.4: EBNF describing the syntax of AIR in QDL.

These interval relations can be used to restrict two elements of an `WHERE` clause according to the given relation and values. They may be used in every suitable position of the `WHERE` or `FILTER` clause.

```
1 <where>         ::= "WHERE" <tuple> {"&" <tuple>}
2 <tuple>         ::= <token>^+
3 <token>         ::= <var> | <uri> | <atom> | <relation>
```

Listing 4.5: EBNF describing the positioning of AIRs in QDL statements.

Definition 4.2. Given a strict partial ordering $O = <O, <>$, an Allen Interval Relation (AIR) is defined as $AIR = RK\ s\ e$, where
$RK \in$ `BF, AF, EQ, S, Si, M, Mi, D, Di, O, Oi, F, Fi` and $s, e \in O$. O is a strict partial ordering on xsd types.

Semantics

An Allen Interval Relation $AIR = RK\ s\ e$ works similar to the operators already integrated into QDL, such as Count or Min. They can be used to ensure that two values, which in combination e.g. encode the valid time of a statement, have a distinct relation to the values in a query's AIR.

Definition 4.3. Based on Definition 4.2 the AIR between $RKs\ e$ and two values $v_s\ v_e$, where $v_s, v_e \in O$, is true if

$$
\begin{aligned}
s < v_s, e < v_s \ &\texttt{in case}\ RK = Bf \\
s > v_e, e > v_e \ &\texttt{in case}\ RK = Af \\
s == v_s, e == v_e \ &\texttt{in case}\ RK = EQ \\
s == v_s, e < v_e \ &\texttt{in case}\ RK = S \\
s == v_s, e > v_e \ &\texttt{in case}\ RK = Si \\
s < v_s, e == v_s \ &\texttt{in case}\ RK = M \\
s > v_s, v_e == s \ &\texttt{in case}\ RK = Mi \\
v_s < s, e < v_e \ &\texttt{in case}\ RK = D \\
s < v_s, e > v_e \ &\texttt{in case}\ RK = Di \\
s < v_s, e < v_e, e > v_s \ &\texttt{in case}\ RK = O \\
s > v_s, s < v_e, e > v_e \ &\texttt{in case}\ RK = Oi \\
s > v_s, e == v_e \ &\texttt{in case}\ RK = F \\
s < v_s, e == v_e \ &\texttt{in case}\ RK = Fi
\end{aligned}
$$

Again, $<, \leq, \geq, >$ are the comparing operations for ordering O and where v is a new unique variable.

Computation

For the newly introduced interval clauses new specific lookup strategies for Allen's Interval Relations were integrated into HFC, compare Table 4.2. The lookups are designed such that they can be performed using a single querying of an Interval Tree based index without any postprocessing to filter results. To achieve this the following lookup methods are used:

- `searchInterval`: Returns all intervals that are within the specified range
- `searchIntervalIncluding`: This method receives a range, as well as two boolean flags (`includeLow` and `includeHigh`) as input parameter and returns the values associated with all intervals that cover the given range and that fulfill the constraints.
- `searchIntervalWithEquality`: This method receives a range, as well as two boolean flags (`lowIsEqual` and `highIsEqual`) as input and returns all values associated with intervals that match the specified range and that fulfill the equality constraints.

All these lookups can only performed on instances of Interval Trees, as all others data structures do not support the necessary range lookups.

We will take a detailed look on how these lookups are triggered and how they are integrated in the QDL framework in section 4.2.3.

If no index is defined, the AIRs will be rewritten by replacing them with a new

Operator	Description	Lookup Method	Lookup
Bf	Before	`searchInterval`	start = $minValue$ end = $s - 1$
Af	After	`searchInterval`	start = $e + 1$ end = $maxValue$

EQ	Equal	`searchIntervalWithEquality`
		`lowIsEqual = true`
		`highIsEqual = true`
			start = s
			end = e

S	Starts	`searchIntervalWithEquality`
		`lowIsEqual = true`
		`highIsEqual = false`
			start = s
			end = $maxValue$

Si	inverse Starts	`searchIncluding`
		`includeLow = true`
		`includeHigh = false`
			start = s
			end = $e - 1$

M	Meets	`searchIntervalWithEquality`
		`lowIsEqual = false`
		`highIsEqual = true`
			start = $minValue$
			end = s

Mi	inverse Meets	`searchIntervalWithEquality`
		`lowIsEqual = true`
		`highIsEqual = false`
			start =s
			end = $maxValue$

O	Overlaps	`searchInterval`	start =$minValue$ end = $e - 1$
Oi	inverse Overlaps	`searchInterval`	start =$s + 1$ end = $maxValue$
D	During	`searchInterval`	start =$s + 1$ end = $e - 1$

Di	inverse During	`searchIncluding`
		`includeLow = false`
		`includeHigh = false`
			start =s
			end = e

F	Finishes	`searchIntervalWithEquality`
		`lowIsEqual = false`
		`highIsEqual = true`
			start =$s + 1$
			end = e

Fi	inverse Finishes	`searchIntervalWithEquality`
		`lowIsEqual = false`
		`highIsEqual = true`
			start =$minValue$
			end = e

TABLE 4.2: Lookup specifications for Allen's Interval Relations

variable and introducing new `FILTER` clauses mimicking the behavior of the interval relations. Let an interval I be defined according to the Definitions 4.2 and 4.3, then the rewriting algorithm for queries is defined as:

1. Remove the RK from the query.
2. Replace s and e with new variables $?v_s$ and $?v_e$.
3. Create new filter clauses $f_1 \dots f_n$ on RK according to the criteria defined in definition 4.3.

where $<, \leq, \geq, >$ are the comparing operations of the ordering O.
For example, rewriting the query in Listing 4.3 will result in the query shown in 4.6.

```
1  SELECT ?s ?o WHERE ?s <bio:marriedTo> ?o ?ref1 ?ref2
2                  & ?s <rdf:type> <bio:Person>
3                  & ?o <rdf:type> <bio:Person>
4          FILTER LGREATER ?ref1 "1991"^^<xsd:int>
5                  & LLESS ?ref2  "2010"^^<xsd:int>
6                  & LGREATER ?ref2 ?ref1
```

Listing 4.6: Rewriting Allen's Interval Relations

4.2.3 Summary and Framework

Combining both new features results in the extended EBNF depicted in Figure 4.15.

```
1          <query>          ::= <select> <where> [<filter>] [<aggregates>]
2          <select>         ::= "SELECT" ["DISTINCT"] {"*" | <var>^+}
3          <var>            ::= "?"<nwchar>^+
4          <nwchar>         ::= any non-whitespace character
5          <where>          ::= "WHERE" <tuple> {"&" <tuple>}
6          <tuple>          ::= <token>^+
7          <token>          ::= <var> | <uri> | <atom> | <interval> | <relation>
8          <uri>            ::= "<" <nwchar>^+ ">"
9          <atom>           ::= "\"" <char>^* "\"" ["@" <langtag> | "^^" <xsdtype>]
10         <interval>       ::= {"["|"("} <atom> "," <atom> {"]"|")"}
11         <relation>       ::= <relKey> <atom> <atom>
12         <relKey>         ::= "Bf" | "Af" | "Eq" | "S" | "Si" | "M" | "Mi" | "D" |
13                             "Di" | "O" | "Oi" | "F" | "Fi"
14         <char>           ::= any character incl. whitespaces, numbers
15         <langtag>        ::= "de" | "en" | ...
16         <xsdtype>        ::= "<xsd:int>" | "<xsd:long>" |"<xsd:float>" |
17                             "<xsd:double>" | "<xsd:dateTime>" |"<xsd:string>" |
18                             "<xsd:boolean>" ...
19         <filter>         ::= "FILTER" <constr> {"&" <constr>}^*
20         <constr>         ::= <ineq> | <predcall>
21         <ineq>           ::= <var> "!=" <token>
22         <predcall>       ::= <predicate> <token>^*
23         <predicate>      ::= <nwchar>^+ (e.g., isUri, NoValue)
24         <aggregate>      ::= "AGGREGATE" <funcall> {"&" <funcall>}^*
25         <funcall>        ::= <var>^+ "=" <function> <token>^*
26         <function>       ::= <nwchar>^+ (e.g., Count, Identit, ...)
27
```

Figure 4.15: EBNF defining syntactically legal QDL queries.

This enables us to use very short and easy-to-understand queries for rather complex facts. For example, assuming an ontology representing sensor data produced

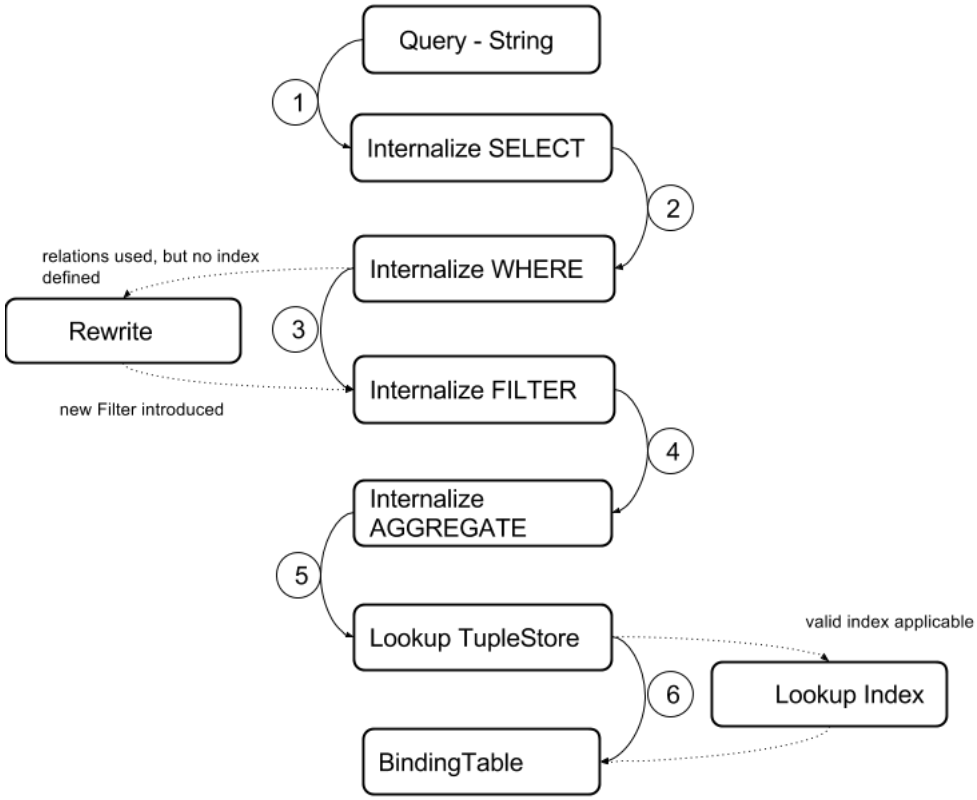


FIGURE 4.16: The extended QDL workflow.

by an ADAS where the last two positions encode the valid time of the statements. Then, the query shown in Listing 4.7 will return all sensors that returned a temperature value between `"42"^^<xsd:int>` and `"64"^^<xsd:int>` during the time `"199923"^^<xsd:long> and "201039"^^<xsd:long>`.

```
1 SELECT ?s WHERE ?s <test:hasTemp> ["42"^^<xsd:int>, "64"^^<xsd:int>]
2                       D "199923"^^<xsd:long> "201039"^^<xsd:long>
3            &  ?s <rdf:type> <test:Sensor> "0"^^<xsd:int> "0"^^<xsd:int>
```

LISTING 4.7: A query that uses both the new interval notion and Allen's Interval Relations.

In the following it will be discussed how QDL is connected and incorporated to the newly introduced IndexStore, as discussed in Section 4.1. At the end this section, it was mentioned that the IndexStore was introduced to speed up the query processing to hopefully enable lookups with the shortest possible delay for the stream reasoner.

Whenever a query is issued, the IndexStore might be accessed if valid indexes are defined. The notion of a valid index in this context means that the indexed key is present at the defined position of the tuple. To explain how this works in detail the already known processing of QDL queries will be extended. The new framework is depicted in Figure 4.16.

The work flow remained largely unchanged, except for the processing of WHERE clauses and the strategy which is implemented to search the database.

The internalization process for WHERE clauses changed, as now intervals and AIRs are taken into account. Whenever one of those is internalized while processing a WHERE clause, it is checked whether an applicable index is defined. If this is not the case the interval or AIR is directly rewritten as described in Sections 4.2.1 and 4.2.3. If a clause is completely processed, it is also checked whether it matches any defined index. If so, a corresponding lookup statement, else a pattern. In case a clause matches multiple indexes a combined lookup featuring all matched indexes is creates. These lookups are objects defining the exact behaviour of an index lookup including restrictions for the results.
Recall, in the original QDL, the ontology was queried using the patterns extracted during the processing of the WHERE clauses. Then the results were joined in case multiple clauses were used. This lookup strategy for the database changed such that index lookups are now performed before the patterns are used to query the TupleStore.
These lookups work according to the algorithm shown in Algorithm 2. First, the actual index lookup for the specified key is performed. Thereby, either a relation specific strategy, e.g. for Allen's Interval Relations, or a standard search is performed, in case no relation is used in the `WHERE` clause. Further, the constants from the `WHERE` clause are applied to the results from the lookup, which decreases the size of matches in every iteration step. If at any time during the algorithm no matching tuples can be found, the algorithm immediately stops and returns an empty set of tuples as a result.

Later both results from index the lookups and pattens are combined and processing continues as usual.
Now consider a simple example processing the query shown in Listing 4.7. For this scenario assume that the ontology is maintained by an instance of HFC which is equipped with an Interval Tree based index for intervals defined by `<xsd:long>` values at position 3 and 4 of the tuples.
When processing the query above the following happens:

1. The `SELECT` clause is parsed resulting in the internalized variable assignment `?s` $\rightarrow -1$.

2. While parsing the first WHERE clause the algorithm proceeds into the interval `["42"^^<xsd:int>, "64"^^<xsd:int>]`. As there is no index defined for the second position of the tuples this interval is rewritten and two new FILTER clauses are introduced. Continuing the processing the algorithm reaches the AIR, which we internalize. As there is an index defined for the third and forth position of the tuple nothing more needs to be done at this point. Now, as we have finished the internalization of the first clause we check whether the whole clause has applicable indexes, and indeed the defined index can be applied. Thus we create a lookup for the Interval Tree and the keys `"199923"^^<xsd:long> "201039"^^<xsd:long>`.
   While parsing the second WHERE clause we find neither intervals or AIRs. However, the defined index matches again the clause and thus we create a lookup for the Interval Tree with keys `"0"^^<xsd:long>, "0"^^<xsd:long>`.

Algorithm 2 lookup Where-clause in index.

```
1: procedure indexLookup(clause,key,relation)
2:     result = empty Set of tuples
3:     // Lookup key
4:     if relation != null then
5:         result = relation specific lookup
6:     else
7:         result = index.search(key)
8:     end if
9:     // Check whether there were tuples associated with key, else done.
10:    if result is empty then
11:        return;
12:    end if
13:    // Apply constants from the Where clause
14:    for i = 0; i < clause.length; i++ do
15:        temp = empty set of tuple
16:        ID = clause[i];
17:        if ID >= 0 then // ID represents a constant.
18:            for tuple in result do
19:                if tuple[i] == ID and tuple.length = clause.length then
20:                    temp.add(tuple)
21:                end if
22:                if temp is empty then
23:                    return
24:                else‘
25:                    result.addAll(temp)
26:                end if
27:            end for
28:        end if
29:    end for
30:    return result
31: end procedure
```

3. In the next step, we internalize the FILTER clauses introduced by rewriting the interval.

4. Processing AGGREGATES can be skipped as non are present.

5. Now, the actual data is retrieved. This can be done by either looking up the tuple store according to patterns and/or using the indexing lookups. As we have no patterns, the lookups are performed. As a result we retrieve two sets of tuples each matching one of the WHERE clauses. These are joined and the FILTER statements are applied as usual.

6. In a final step the actual variable assignments (SELECT) are extracted and combined to a binding table.

4.3 Stream-Reasoner

Finally, all prerequisites are covered, such that the stream reasoner itself can be discussed.
The developed stream reasoner SHFC is capable of processing inputs (rdf- or n-tuple streams) from different sources, detecting important events in these streams and deriving new facts based on the observed ones. These facts can then either be used to update the background ontology and/or fed back into the stream where they potentially trigger new derivations.
In addition, SHFC is able to periodically create binding tables with variable assignments, similar to the window-based SELECT queries of the other RSPs. The inference computation and event detection is based on a rule based approach with rules similar to the ones used by HFC for closure computation.

The remaining part of this section will start with an overview on the structure and functionality of the stream reasoner. Later on, details on the definitions of the used rules and how these rules process the input will be covered.

4.3.1 Overview

The stream reasoner is not integrated into HFC, instead it is an extension to it. Figure 4.17 shows a simplified version of the reasoner's framework. The elements

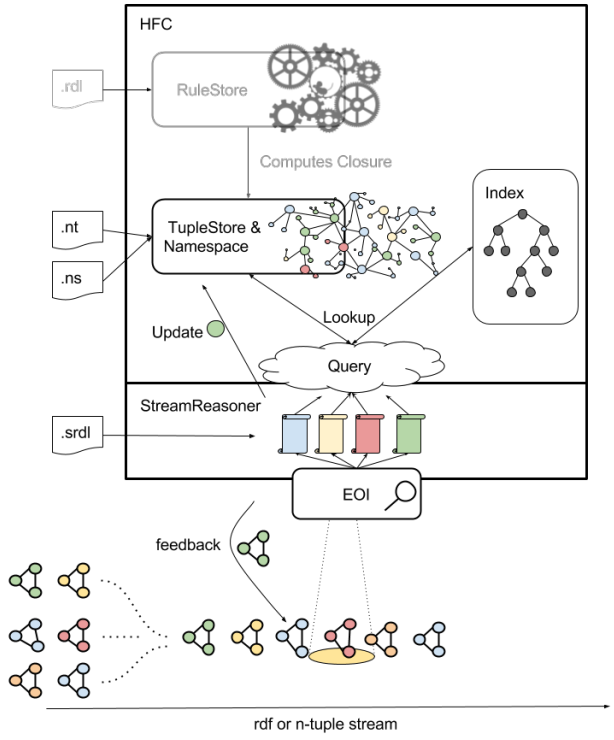


FIGURE 4.17: General structure of the stream reasoner framework.

of the input streams are channelled through a blocking queue, which serves as an interface between the input sources and the actual reasoner. This queue can be

seen as a bottleneck, where the streams generated by the different input sources (producers) are combined to a single stream.
The reasoner itself works according to the algorithm shown in Algorithm 3.

Algorithm 3 The stream reasoning work flow.

```
1: procedure PROCESS
2:     queue = a queue of events (OWL-triples)
3:     rules = a set of SRDL rules,
4:     t = time to wait for new input
5:     intializeSystem();
6:     while not killed do
7:         if q.isEmpty() then
8:             wait(t)
9:         else
10:            EoI = first element of queue
11:            for each r in rules do
12:                run r.apply(EoI) in a separate thread
13:            end for
14:            wait for all rules to finish
15:            remove first element from queue
16:        end if
17:    end while
18: end procedure
```

SHFC is a reactive system, meaning that events are processed as soon they arrive, resulting in an instantaneous rule execution. Whenever there are elements in the queue all rules are triggered, which then process the first element, also called event of interest (EoI), of the queue (the stream) in parallel[5]. While processing the EoI rules might access the background knowledge by issuing queries to HFC. Once all rules have finished processing, the first event is released from the queue. However, if the buffer is empty the reasoner idles until new events arrive.
SHFC supports 3 different outputs. On the one hand, newly derived information - n-tuple - can be used to expand or update the ontology background. On the other hand, this information can also be introduced back into the stream, where they either trigger new rules or can be processed by other tools. Last but not least, rules can also be used to periodically create binding tables including data from observed and derived facts. These can be used by other tools who have registered as an observer at the corresponding rule. Now as you are more or less familiar with the mode of operation of the stream reasoner, let's have a more detailed view on the rules, their definition and their way of working.

4.3.2 Rules

It was mentioned initially that the rules are defined using a new language which is similar to the RDL of HFC. The new language is called SRDL, which is short

[5]Each rule runs in an own thread, if possible.

for Stream Rule Description Language, and it is syntactically and semantically similar to its predecessor.
The rules are read in by the so-called `RuleModel` which parses the `.srdl` file containing the definitions and maintaining the resulting rules, meaning that it provides the rules with information on how to access the background knowledge (via. `KnowledgeGate`) and the EoI. The EoI is used as an interface between stream and reasoner determining the current event of interest. The `KnowledgeGate` is used as an adapter for accessing the background ontology. This way the rules and HFC are encapsulated and problems due to concurrent accesses are minimized. However, the missing support for parallel background access might cause a reduced performance in certain scenarios.

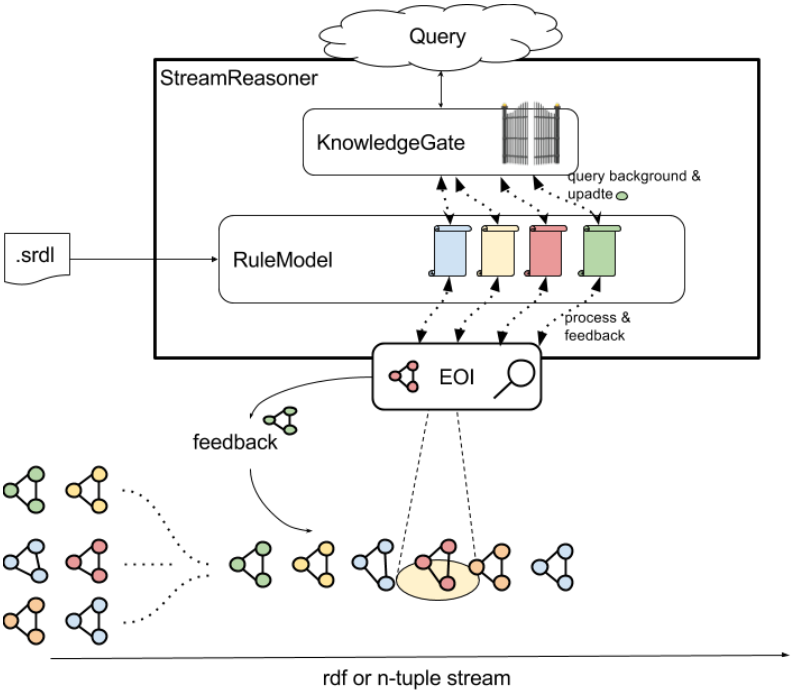


FIGURE 4.18: Detailed structure of the stream reasoner.

SRDL - Stream Rule Description Language

Similar to the original RDL, the SRDL features extended RDF n-tuple notation, functional and relational variables, complex tests and actions. Additional features such as static and dynamic knowledge lookups are supported, as well. An EBNF of the SRDL is shown in Figure 4.19. Each rule must start with a name and consists of one or more *antecedent* clauses and at least one transition operator followed by consequence clauses. This name is a unique identifier of the rules, which allows other tools to register as an observer for the rule's results. The transitions between antecedent and the consequences are defined by either `->`(update the ontology), `=>`(update the stream), `=>>`(update both stream and ontology), or <*X time*> (periodically create a binding table). Both antecedent and consequence clauses are represented as a sequence of tuples. Each tuple element is either a URI, an XSD atom, or a variable which is interpreted locally within the scope of the

```
1         <rule>          ::= <name> <ante> {<imply> <cons>}^+
2                             [<test>] [statKnow] [dynKnow] [<action>] <cr>
3         <name>          ::= "$" <nwchar>^+ <cr>
4         <ante>          ::= { <tuple> <cr>}^+
5         <imply>         ::= { "->" | "=>" | "=>>" | "<" <+int0>^+ <time>">")} <cr>
6         <cons>          ::= { <tuple> <cr> |
7                              "{" {<lvar>|<lfunc>} ":" <tuple> "}" <cr> }^+
8         <test>          ::= "@Test" <cr> {<pred> <cr>}^*
9         <statKnow>      ::= "@StaticKnowledge" <cr> {<qfunc> <cr>}^*
10        <dynKnow>       ::= "@DynamicKnowledge" <cr> {<qfunc> <cr>}^*
11        <action>        ::= "@Action" <cr> {<func> <cr>}^*
12        <pred>          ::= <var>  "!=" {<var> | <uri> | <atom>| <lvar> | <lfunc>} |
13        <nwchar>^+ {<var> | <uri> | <atom>| <lvar> |<lfunc>}^*
14        <func>          ::= <var> "=" <nwchar>^+ {<var>| <uri> | <atom>}^*
15        <qfunc>         ::= {<lvar> "="| <lfunc> ":="} <target> ":" {<var>| <uri> | <atom>}^+
16        <target>        ::= {<var> | "TABLE" | "TRUE" | "FALSE"}
17        <var>           ::= "?"<nwchar>^+
18        <lvar>          ::= "!"<nwchar>^+
19        <lfunc>         ::= "!"<nwchar>^+"("{<var> | <lvar>}")"
20        <tuple>         ::= {<var>| <uri> | <atom>| <lvar>}^+
21        <uri>           ::= "<" <nwchar>^+ ">"
22        <atom>          ::= "\"" <char>^* "\"" ["@" <langtag> | "^^" <xsdtype>]
23        <time>          ::= { "ms" | "s" | "m"}
24        <char>          ::= any character incl. whitespaces, numbers, even '\"'
25        <nwchar>        ::= any non-whitespace character
26        <+int0>         ::= any positive intege, including 0
27        <cr>            ::= the carriage return/newline character
28
```

FIGURE 4.19: EBNF defining syntactically valid SRDL rules.

rule. However, in case binding tables are created, each tuple in the consequence clauses must only consist of variables, as each of these tuples defines the variables for one binding table.
SRDL also features `@Test` clauses, supporting inequality constraints, but also arbitrary, possibly custom predicates. These clauses are SHFC's equivalent to the FILTER statements in the query-based RSPs, but much more versatile due to the custom predicated. Additionally, so called `@Action` clauses can be used to define function performing small computations such as shown in the example below. The `@Action` clauses are similar to aggregates in queries, however performed operation statements are completely customizable.
In addition to these features, which should already be known from the original RDL, knowledge statements were introduced. We distinguish between different kinds of knowledge statements, namely variable assignments and functions. The first one assigns the result of the query specified at the right hand side to the local variable, e.g. `!x`, on the left hand side. Local variables can only be introduced by knowledge statements and are used to signalize that their value is derived from the local background knowledge.
The syntax of a knowledge statement k is formally defined as:

$$k = v := t : q$$

where v is a valid and unique local variable;
t is either a variable used in q, TRUE, FALSE or TABLE;
q is a valid set of WHERE, FILTER and AGGREGATE clauses.
A valid variable assignment would look as follows:

```
1 !threshold = ?o : <test:sensor1> <test:threshold> ?o "0"^^<xsd:long>
```

Queries have a preceding statement t, which is virtually defining the SELECT clause of the query and thus the projected wars. Using a variable assigns the entries associated with this variable to this local variable. TRUE and FALSE, indicate that the query is used to check whether the statement defined in q holds, e.g. whether a tuple is part of the ontology. Thereby TRUE and FALSE define the expected outcome of this check. Finally, the key word TABLE can be used to bind the whole binding table resulting from the query to the local variable.
The only difference between assignments and functions is that the left hand side is not a simple local variable, but a local variable with parameters, i.e. `!x(?v1)`. They can be used to lookup information with respect to the current state of the rule, i.e. already assigned vars etc. When declaring such a function, arguments are specified using placeholder variables, i.e., variables not used in the remaining query. When applying the function these variables are then replaced by respective variables and their assigned values. A valid function definition would look as follows:

```
1 !ref(?x) = ?o : ?x <test:threshold> ?o "0"^^<xsd:long>
```

Both, variable assignments and functions can be used in a static (`@StaticKnowledge`) or dynamic (`@DynamicKnowledge`) fashion. If used statically, the associated query will only be performed once while the rule is initialized, whereas dynamic knowledge statements are performed whenever the result of the associated query is needed to process an event. Thus, variable assignments are predestined for use in a static matter and functions for use in a dynamic matter.
An example for a valid rule could look as follows:

```
1  $isOverheating
2  ?s <test:hasValue> ?v ?t
3  ?s <test:hasValue> ?v1 ?t1
4  =>>
5  ?s <test:hasState> <overheating> ?max
6  @Test
7  LGreater ?v !ref(?s)
8  LGreater ?v1 !ref(?s)
9  LLess ?dif "5"^^<xsd:long>
10 @DynamicKnowledge
11 !ref(?x) := ?o : ?x <test:threshold> ?o "0"^^<xsd:long>
12 @Action
13 ?dif = LAbsoluteDifference ?t ?t1
14 ?max = LMax2 ?t ?t1
```

Listing 4.8: A rule checking whether a sensor is overheating.

This rule checks whether a motor overheats, by observing a stream of sensor data. The first line defines the name of the rule. Lines 2 and 3 define the antecedent of the rule. Here, two events have to occur that have the same subject, and the relation `<test:hasValue>`. However, the values (`?v, ?v1`) and timestamps (`?t, ?t1`) can be different.
Line 4 defines the way in which the consequence defined in line 5 is achieved. In this case both the ontology and the stream should be updated with the corresponding tuple.
The `@Test` block (line 6-9) ensures that (i) the events are within a time span of 5 seconds, (ii) both values `?v1` and `?v2` are higher than the threshold set for this

sensor. To retrieve this threshold the function `!ref(?x)` defined in line 12 was instantiated with `?v1` and `?v2` respectively.
The `@DynamicKnowledge` block on lines 10 and 11 is used to assign the value on the right to the function on the left hand side. These queries are performed every time the rule is triggered.
The `@Action` block computes the absolute difference between and the maximum of the timestamps and assigns the result to the variable `?dif` and `?max`, respectively.

On the other hand, a simple rule that periodically creates binding tables for sensor values is shown in Listing 4.9.

```
1  $sensorValues_thermal
2  ?s <test:hasValue> ?v ?t
3  <1000ms>
4  ?s ?v
5  @Test
6  !isThermalSensor{?s} != "false"^^<xsd:boolean>
7  @StaticKnowledge
8  !isThermalSensor(?x) := TRUE : ?x <rdf:type> <test:ThermalSensor> "0"^^<xsd:long>
9
```

Listing 4.9: A rule periodically creating binding tables.

Each second this rule creates a binding table containing all thermal sensors and their value observed in the last second. Here the `StaticKnowledge` statement returns `"true"^^<xsd:boolean>` iff the given `?x` is of type thermal sensor, else `"false"^^<xsd:boolean>` is returned.

Rule Processing

This section will discuss in detail how the events are processed internally by the rules. But beforehand, the internal data structure of the rules will be presented, see Figure 4.20.

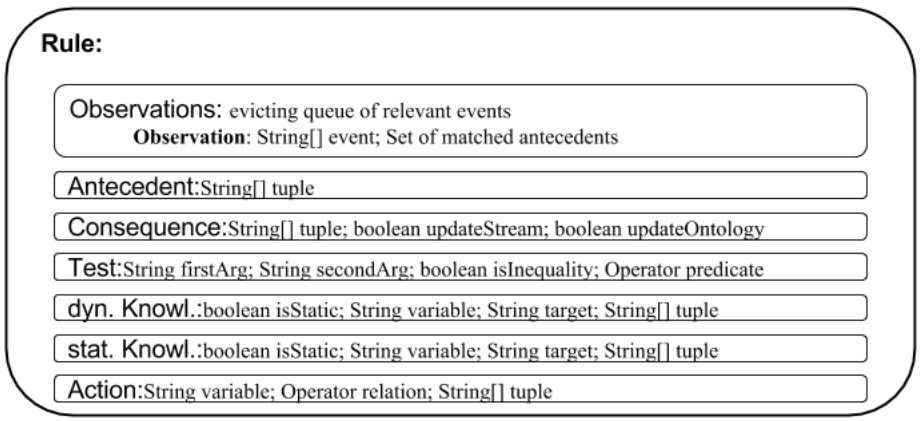


Figure 4.20: The data structure of a rule.

Internally, the behaviour of a rule is defined by the actual rule components (`antecedent, consequence, test, action, dynamic` and `static knowledge`) and an evicting queue of size x containing the last observations of relevant events. Here "relevant" means, that only events that match at least one of the antecedent

clauses of the rule is considered, where x is a use case specific number. These observations contain the actual event, as well as the set of antecedents matched by this event. SHFC does not use timed windows for the consideration of events and instead uses windows determined by a fixed size of events, unlike for example C-SPARQL and C-QELS.
While processing new incoming events so-called rule state objects are used to track the different possible variable assignments. A rule state is defined by a map of variable assignments for all variables (general and local) used by the rule. This assignments are initialized in a lazy fashion. In addition, the antecedent clauses fulfilled by the represented assignments are tracked. The rule components are represented using collections of the following objects:

- `antecedent`: A rule contains an `Antecedent` object for each antecedent clause. Each of these objects consists of a `tuple` representing the associated clause.

- `consequence`: A rule contains a `Consequence` object for each consequence clause. Each of these objects consists of boolean representations for the associated operations (`updateOntology`, `updateStream`). In addition a simple `tuple` representing the associated clause is used.

- `test`: A rule contains a `Test` object for each test clause. These Test objects consist of a boolean value indicating whether the test is an in-equality or an relational constraint (`isInEquality`). A FunctionalOperator `relation` is also included, however if the test is an in-equality statement this field is assigned to `null`. In addition there are two Strings (`firstArg`,`secondArg`) either representing the left and right hand side of the inequality constraint or the arguments of the binary predicate.

- `dynamic/static Knowledge`: `Knowledge` objects contain a boolean value (`isStatic`) indicating whether the object represents a static or dynamic knowledge lookup, a String (`var`) representing the local variable on the left hand side, an additional String (`target`) representing the target variable of the lookup, and a String `tuple` representing the Query statements.

- `action`: `Action` objects simply contain a String (`var`) representing the variable on the left hand side, a FunctionalOperator `relation` describing the relation to be processed, and a String `tuple` representing the arguments for the relation.

With this structure in mind, we can finally take a look on how the rules actually process events, compare Figure 4.21 and Algorithm 4.

A rule idles until a new event of interest (EoI) can be processed. Then it is checked whether the event matches any of the antecedent clauses. This is done by first comparing the length of clause and event and, in case these match by checking whether all constants from the antecedent clauses appear at the respective position in the EoI. If no antecedent is matched the rule returns to its idle state. In case the new event matches any antecedent statement of the rule, a set of rule states - one for each matched antecedent - is created. These states feature the variable

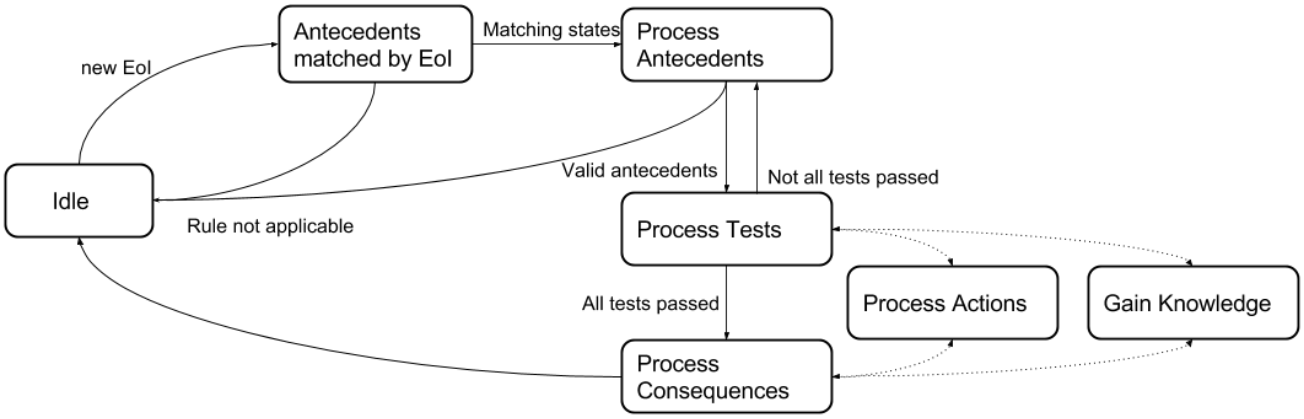


FIGURE 4.21: State diagram of the rule processing framework.

Algorithm 4 A rule processing an event of interest (`String[]`).

```
1: procedure applyRule(String[] EoI)
2:     results = empty set of tuples (String[])
3:     observations = evicting queue of past observations
4:     // compute the antecedents fulfilled by the EoI
5:     matches = checkAntecedents(EoI) // set of rule states
6:     if matches are empty then
7:         return, nothing to do
8:     else
9:         for each state match in matches  do
10:            //compute the rule state triggering the rule, if it exists, else null
11:            trigger = ruleTriggered(match)
12:            if trigger! = null then
13:                results = applyConsequences(trigger)
14:                observations.add(EoI)
15:                return results
16:            end if
17:        end for
18:        observations.add(EoI)
19:    end if
20:    return results
21: end procedure
```

assignment introduced by the new event as well as the matched antecedent. The states are then used as a starting point to compute whether the new event triggers the rule in combination with the past observations. This is achieved by examining, for each of these states separately, whether the past observations fulfil the missing antecedent clauses while respecting established variable assignments, e.g. the one introduced by the new event, or later in the process by other observations. The examining process exits as soon as it recognizes that an antecedent cannot be fulfilled by the existing observations, as in this case further investigations are pointless. On the other hand, as soon as a combination of observations, i.e. a rule

Algorithm 5 Checking whether an event matches antecedent clauses.

```
1: procedure CheckAntecedents(EoI)
2:     matches = empty list of rule states
3:     for each antecedent in the antecedents of the rule do
4:         // apply antecedent to EoI, returns null if not applicable
5:         state = antecedent.apply(EoI)
6:         if state! = null then
7:             matches.add(state)
8:         end if
9:     end for
10:    return matches
11: end procedure
12:
13: procedure ruleTriggered(ruleState)
14:    if ruleState fulfills all antecedents then
15:        if ( thenruleState fulfills all tests)
16:            return ruleState
17:        end if
18:        return null
19:    else
20:        for each not fulfilled antecedent in antecedents do
21:            // find states based on the given one such that antecedent is fulfilled
22:            // this states are all versions of the given one,
23:            // extended by new variable assignments
24:            possibleMatches = findMatches(state, antecedent);
25:            if possibleMatches is empty then
26:                // no need to continue, as the rule cannot be triggered
27:                // if one antecedent is not valid
28:                return null
29:            end if
30:            // continue search recursively
31:            for match in possibleMatches do
32:                result = ruleTriggered(potentialMatch)
33:                if result! = null then
34:                    return result
35:                end if
36:            end for
37:        end for
38:    end if
39:    return null
40: end procedure
```

state with variable assignments based on these observations, is found such that the whole antecedent statement holds, the test statements are applied on the rule state. During this step it might be necessary to perform small computations (actions) or background knowledge lookups to retrieve reference values, for example. If the tests are not passed, the examining process is continued until, either a new state is found where all antecedents are valid or all combinations have been checked. However, if all tests are passed the last step of the rule processing, applying the consequences, is performed. In this step the variables in the consequence clauses are instantiated with their associated values, thereby knowledge lookups or actions might be used to retrieve the necessary values. The resulting clauses are added to the ontology or feed back into the stream depending on the transitions associated with the consequence clause. If this is done the rule returns to its idle state.

4.4 Summary

During the last sections, the implementation of an n-tuple based stream reasoner SHFC and the necessary prerequisites have been presented. SHFC is capable of processing streams of RDF-triples or n-tuples in (almost) real time, using a rule-based stream-processing and reasoning-approach, with rules that are structurally similar to the RDL rules used by HFC. The background knowledge is represented in an HFC framework and thus can be queried using an extended version of HFCs query language.
Compared to the other RSP discussed RSP systems SHFC can be classified as seen from Table 4.3.

Table 4.3: Classification of presented RSP Systems.

	C-SPARQL	CQELS	ETALIS	SHFC
structure	black-box	white-box	white-box	white-box
strategy	periodically	reactive	reactive	reactive
input	rdf	rdf	rdf + timestamp	n-tuple
task description	SPARQL-like query	SPARQL-like query	SPARQL-like query or rule	rule
background	RDF/OWL	RDF/OWL	RDF/OWL Prolog	n-tuple
time aware	no	no	yes	yes
output	binding tables, instantaneous RDF graphs, RDF stream	stream of RDF or SPARQL Result format	data, RDF stream	n-tuple stream binding tables ontology update
others	supports timed windows	supports timed windows	/	/

The following section will take a look on whether a n-tuple based background ontology provides benefits in the context of stream reasoning compared to classical RDF/OWL triples. Therefore, exhaustive evaluations of the extended QDL and

the stream reasoner were performed.

Chapter 5
Evaluation

In this section the evaluations performed during this work will be presented and discussed. The first evaluation is a comparison between the original indexing used by HFC and the extended framework utilizing the advanced indexing structures discussed in Section 4.1. This evaluation will show whether integrating these advanced structures is beneficial. The second evaluation is a comparison between the developed stream reasoner and existing frameworks such as CQELS, ETALIS and CSPARQL. This evaluation will show whether n-tuple based background knowledge is useful in the context of stream reasoning.
Both evaluations were performed using the following system:

- Java 1.8.0_144
- Manjaro Linux 17.0.5
- Kernel 4.11-rt-x86_64
- Haswell Core`I7-4700 @2.40GHz`
- 12GB DDR3 with 1600MHz

5.1 Evaluating the Indexing Framework

In this evaluation we will compare the original indexing framework of HFC, which is situated in the TupleStore, against each of the new frameworks.

5.1.1 Setup

This evaluation uses three different types of ontologies, from now on referred to as `Valid`, `Transaction` and `TransactionAndValid`. Each of these is based on the LUBM[29] benchmark ontologies. However, they are extended by transaction or valid time depending on the evaluated index and feature.

- `Transaction`: The baseline, as well as all new indexing structures were evaluated with respect to Transaction time. Therefore the LUBM ontologies were extended by a single `<xsd:long>` value representing the transaction time of a statement. This resulted in the following tuple structure
  `<tt> <sub> <pred> <obj>`.
  All advanced structures were configured to index the value at the first position of a tuple, as shown in Listing 5.1.

```
1         Key1: XsdLong
2         Position1: 0
3         Structure1: <Index>
4
```

LISTING 5.1: Index configuration for transaction time setup. `Index` is either BTreeIndex, BPlusTreeIndex, RedBlackTreeIndex or IntervalTreeIndex.

- `Valid`: Here the baseline and all newly integrated structures were evaluated with respect to valid time. Therefore the LUBM ontologies were extended by two `<xsd:long>` values representing the lower and upper bound of the valid time of an statement. This resulted in the following tuple structure `<sub> <pred> <obj> <vt1> <vt2>`.
  The Interval Tree was configured to index intervals described by two last positions in a tuple (Listing 5.2), whereas the standard structures used `<xsd:long>` values at the last position of the tuples as key (Listing 5.3).

```
1         Key1: XsdLong
2         Position1: 3,4
3         Structure1: IntervalTreeIndex
4
```

LISTING 5.2: IntervalTree configuration for valid time setup.

```
1         Key1: XsdLong
2         Position1: 4
3         Structure1: <Index>
4
```

LISTING 5.3: Index configuration for valid time setup. `Index` is either BTreeIndex, BPlusTreeIndex or RedBlackTreeIndex.

For the evaluation, the following 6 query patterns were applied to versions of the modified LUBM ontologies with 20000, 25000 and 50000 tuples each. The issued queries can be found in Appendix 6.2.

`Q1` Query with two WHERE clauses and fixed transaction and/or valid time. Fully utilizes the indexing structures.

`Q2` Query with two WHERE clauses and variables for transaction and/or valid time in the second WHERE clause. Indexing structures are not utilized for a where clause if variables are used for the indexed keys. Thus this query will check how effective lookups in the indexing structure and the baseline implementation can be combined.

`Q3` Query with two WHERE clauses, where only the indexed keys are constants and every other part of the clause is represented by variables. It is assumed that the indexing structures clearly outperform the baseline in this scenario.

`Q4` Query with two WHERE and one FILTER clause and fixed transaction and/or valid time. This query will show the impact of filters on the performance. Fully utilizes the indexing structures.

`Q5` Query with two WHERE and one AGGREGATE clause and fixed transaction and/or valid time. This query will show the impact of aggregates on the performance. Fully utilizes the indexing structures.

`Q6` Query with two WHERE clauses featuring interval and/or AIR statements. This query will show the performance with respect to the new features. It is assumed that the indexing structures perform better than the baseline for this queries. Also fully utilizes the indexing structures.

For benchmarking the throughput, average time per operation, and memory usage per operation were measured using the JMH, short for Java micro benchmark harness[30][1]. JMH is a toolkit that helps implementing Java micro benchmarks. These benchmarks were executed using the JMH defaults, meaning 20 warmup iterations, 20 measurement iterations, and 10 forks of each.
The calls can be gathered from the `jmh-run.sh` script in the repository.[2]

5.1.2 Results

The results we achieved are quite puzzling, as none of the newly integrated structures was able to outperform the baseline in a significant manner. In some cases the baseline was even better than the more elaborated structures. Possible explanations for this behavior will be presented while discussing the results in detail.

TransactionTime The results for throughput, average runtime per operation and memory consumption per operation in the transaction time scenario are presented in Figures 5.1, 5.2, and 5.3 respectively.
Taking a look at the measurements for the throughput one quickly realizes that the baseline and the new indexing structures, except the Interval Tree, deliver roughly the same performance.

Generally, the throughput is dropping for all structures with increasing size of the queried ontology. However, Q1 seems to be an exception here. It is assumed that there were no performance drops for this query due to its simplicity. For queries Q4 and Q5 the overall performance is worse compared to the Q1 and Q3, however that was to be expected as these queries use FILTER and AGGREGATE clauses. Interestingly, the performance drops for these queries are not that deep when the ontology size is increased, especially compared to the drops observed for Q3.
On the other hand, the performance for Q2 and Q6 was significantly worse for all structures compared to the other queries. This was to be expected as (i) in query Q2 variables where used for the transaction time and thus large sets of possible matches had to be processed and (ii) in query Q6 the newly introduced interval notion was used for the transaction time statement, which inevitably results in more processing to be performed. The results for Q2 are identical for all

[1] Additional information as well as a good introduction into the JMH can be found in this turorial: https://github.com/guozheng/jmh-tutorial/blob/master/README.md
[2] **link**

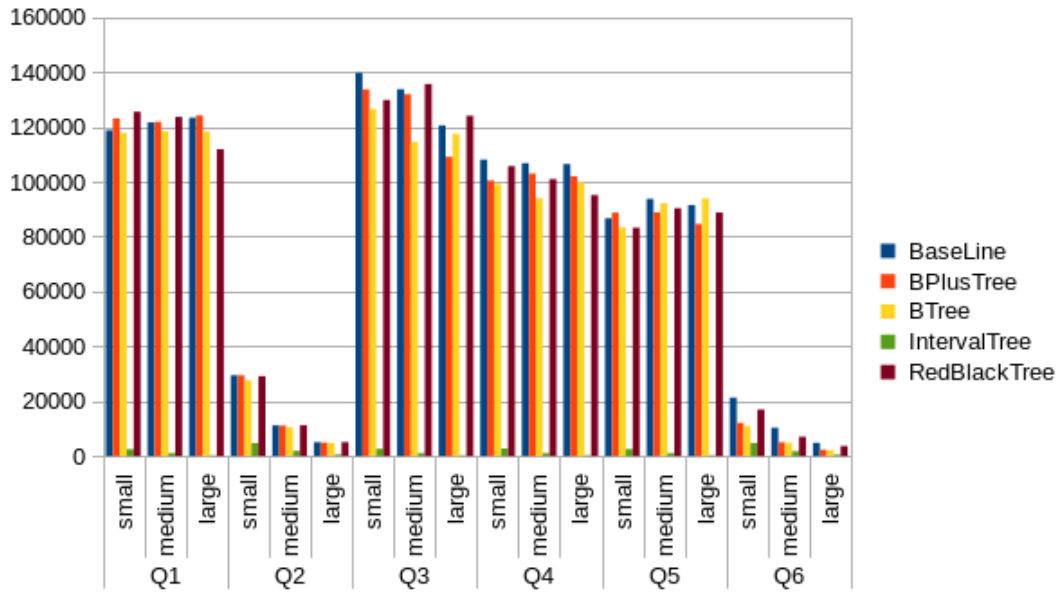


FIGURE 5.1: Throughput in op/s for the TransactionTime-Ontology.

structures as here the baseline was used for all lookups, as described in evaluation setup. The results for Q6 clearly show that implementing simple range lookups into the standard data structure (see Section 4.1) to support the new interval feature in QDL has not paid off. The baseline implementation, which implements the range lookups by means of internally rewriting the range statement into FILTER clauses, outperforms all other data structures. For the Red-Black Tree the performance decreased by around one third compared to the baseline and for B-Tree and B+Tree more than half.
Since the range to be searched in Q6 was very large [3] the query was evaluated again using smaller ranges. However, the overall picture has remained the same. Only the Red-Black Tree was able to catch up in comparison to the baseline. So we can say that the interval/range extension for QDL should be used as syntactic sugar and should therefore be rewritten internally into appropriate filters. Custom lookup functions for the new indexing structures provide no advantage.
For all other queries the picture remains the same, there the baseline and Red-Black Tree perform best with almost identical results, leaving the other structures and especially the Interval Tree somewhat behind.
It should be noted that the Interval Tree performed extremely bad throughout all queries. This is even more clear when taking a look at the average runtime per operation in Figure 5.2.

Here, the values for the Interval Tree are that high such that the values for all other structures are barely visible. Only for Q2 and Q6 there are peeks reflecting the higher processing effort for these scenarios.
Taking a look at the memory footprint (Figure 5.3) of the evaluated structures we see some inconsistent results.

It is common that with increasing size of the ontology the overall used memory increases. However there are some exceptions, where the memory footprint for

[3] `[ "0"^^<xsd:long> , "6666"^^<xsd:long> ]`

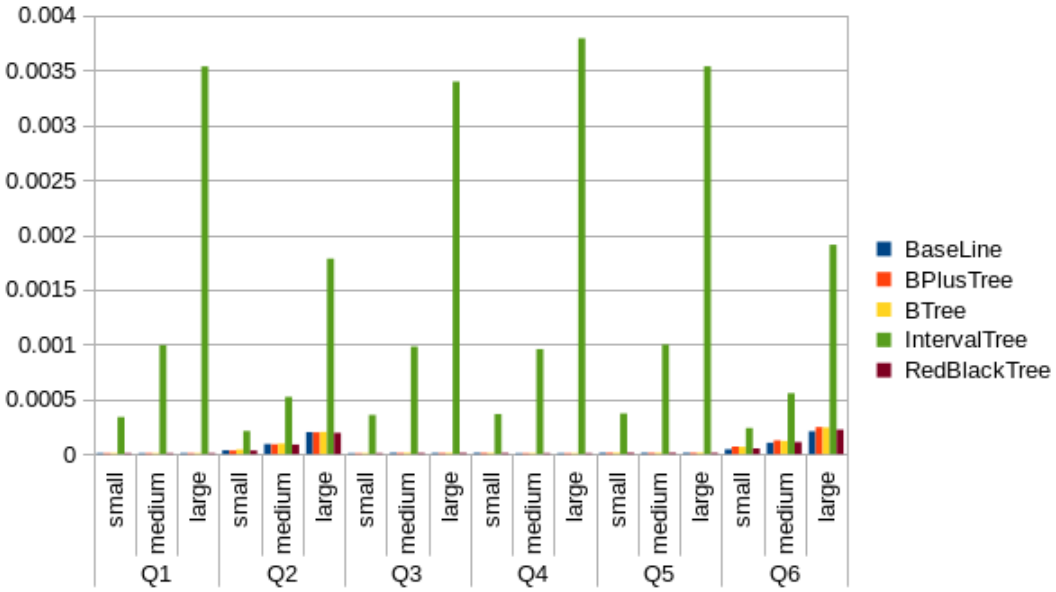


FIGURE 5.2: Average runtime per operation for the TransactionTime-Ontology.

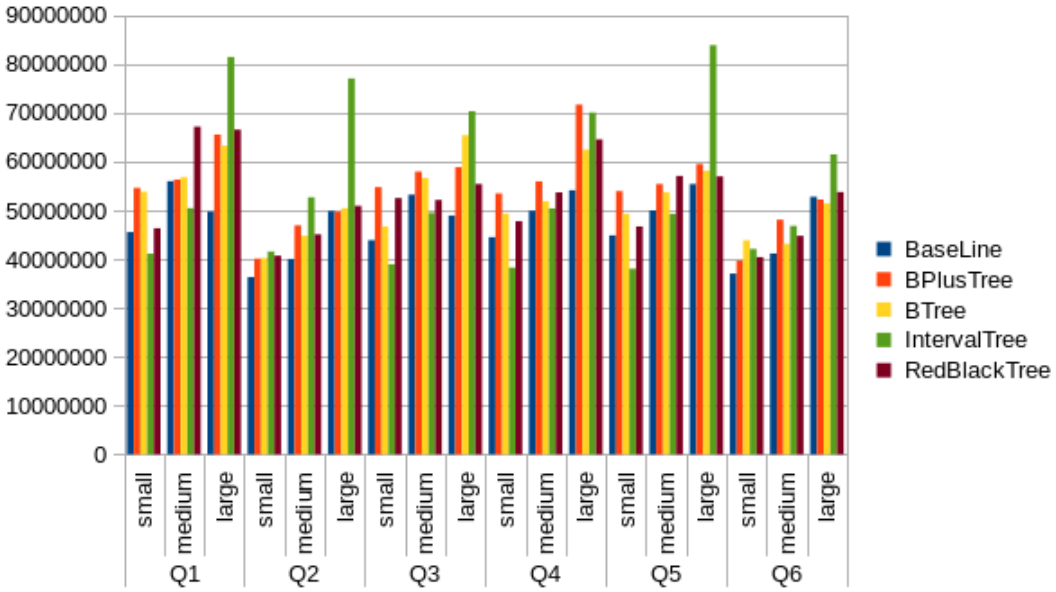


FIGURE 5.3: Memory consumption per operation for the TransactionTime-Ontology.

querying the large ontology was cheaper than querying the big one, which has only half the size of the large one. It is assumed that this is due to inaccuracies in the measuring process[4].

For the transaction time we can conclude that the Interval Tree performs worst for all scenarios/queries, which indicates a flaw in its implementation and a need

[4]Here a custom JMH profiler performing forced garbage collections before each run and measure the memory after the run was performed.

for optimizations. The other structures however perform the same, at large, even if it was originally expected that the more elaborated structures outperform the baseline.

ValidTime The measurements for the valid time setup are depicted in Figures 5.4, 5.5, and 5.6 respectively.
Generally the throughput is not that high than for the transaction time setup, with a maximum of approx. 110000 operations per second compared to the maximal 140000 operations in the transaction time setup. This behavior was expected, due to the longer tuples used in this setup.
Surprisingly, the newly integrated indexing structures, except the Interval Tree,

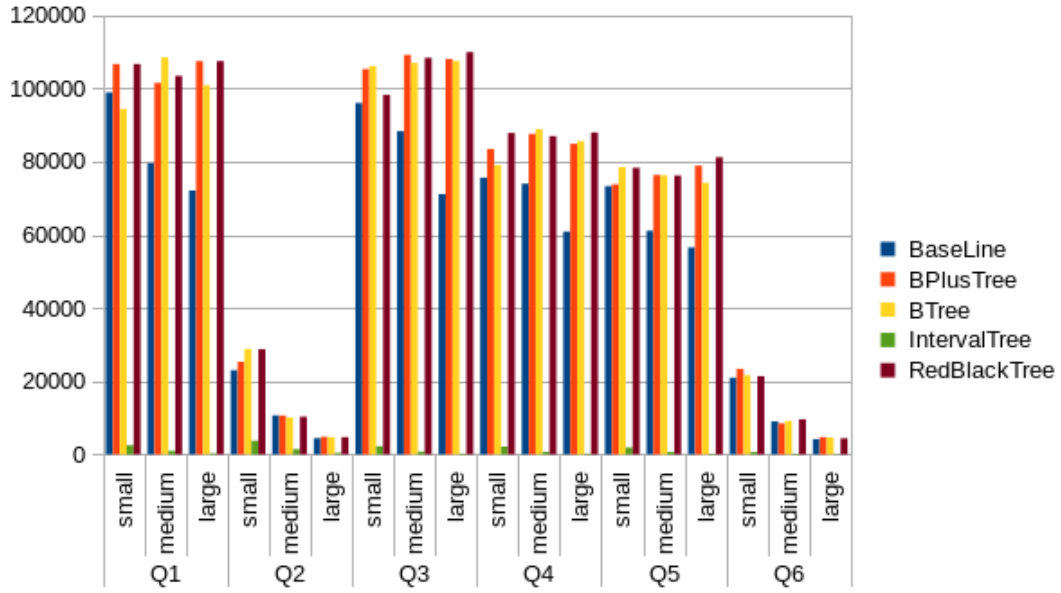


FIGURE 5.4: Throughput in op/s for the ValidTime-Ontology.

significantly outperform the baseline in queries Q1, Q3, Q4, Q5. Remember, for the ValidTime setup, B-Tree, B+Tree, and Red-Black Tree were configured to use the `<xsd:long>` values at the last position of the tuple as keys. This allows them to quickly narrow the search space in the given queries, which in turn speeds up the calculation of the results by applying the remaining constants to the search space (cf. Section 4.2.3). In contrast, the baseline in this scenario must iterate over all 5 positions of the tuple to compute the results (see Section 2.2.2). Some quick experiments with longer tuples have confirmed that the indexing structures scale much better with increasing tuple size than the baseline implementation. It seems, that in the TransactionTime scenario, where tuples of length 4 were processed, the performance of baseline and indexing structures was rather coincidentally very similar.
The identical performance in queries Q2 and Q6 has the same explanation as in the transaction time setup. In this queries the baseline implementation was used as the indexes were not supporting the necessary features, i.e. variables at indexed positions in Q2, support for AIR in Q6.
However, the Interval Tree has again a very poor performance throughout all

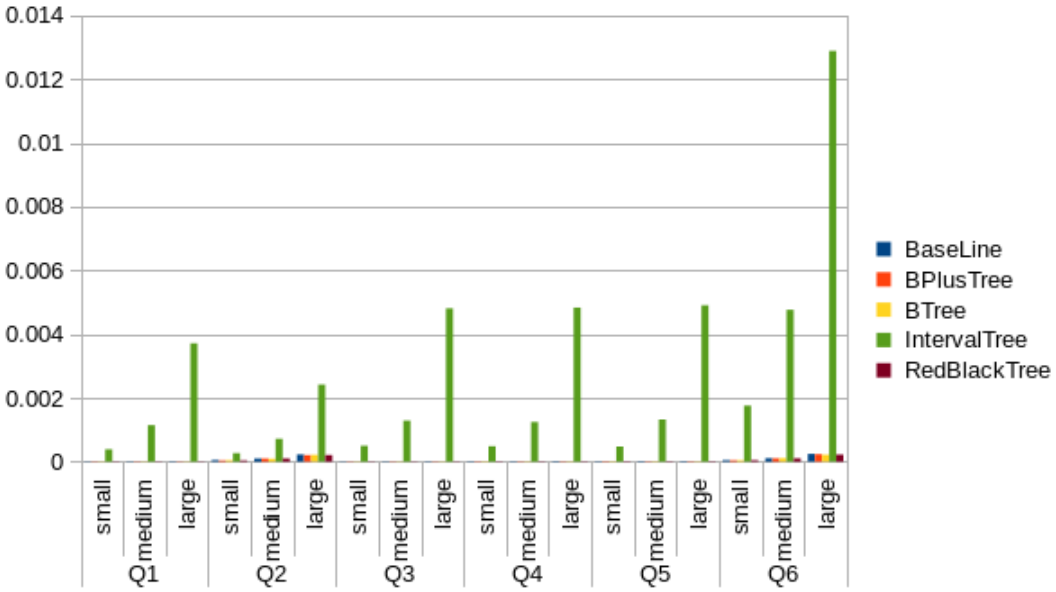


FIGURE 5.5: Average runtime per operation for the ValidTime-Ontology.

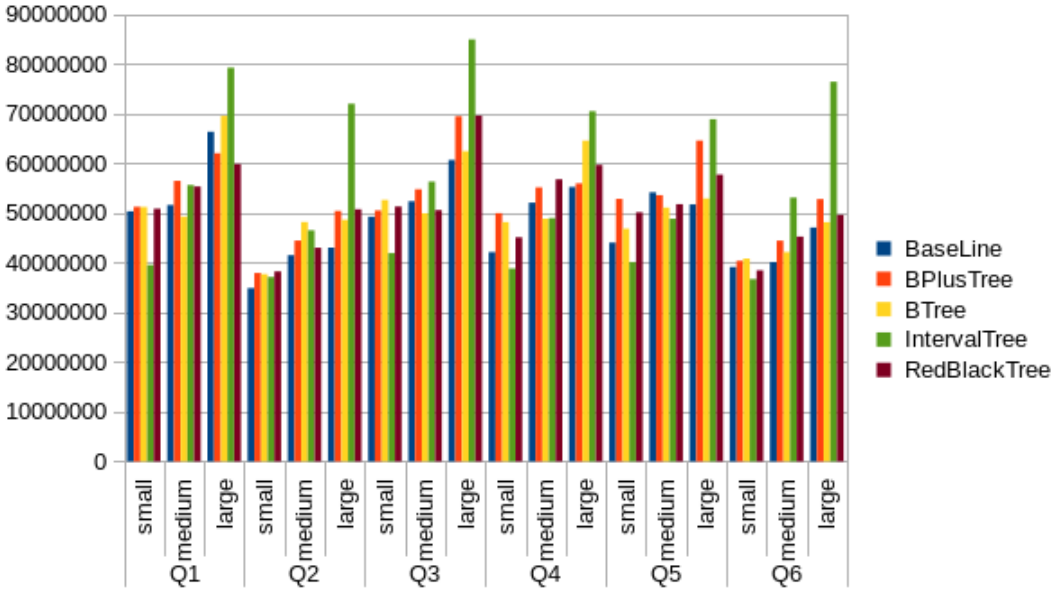


FIGURE 5.6: Memory consumption per operation for the ValidTime-Ontology, Interval Tree excluded.

queries, which confirms the presumption that there is a flaw in its implementation, as this structure was initially integrated to perform exactly the task of indexing valid time statements.
Observations of the average runtime per operation (Figure 5.5) complete the overall picture and confirm that in this scenario the indexing structures are more effective than the baseline. However, this is at the cost of higher memory consumption as shown in Figure 5.6.

5.1.3 Summary

This quick evaluations indicate that the performance of the baseline index implementation of HFC can compete against the new indexing structures for ontologies with short tuples, i.e. triple or quatuple. However the new indexing structures are superior when the size of the tuples increases. The performance of these indexing structures could be increased when moving from general Java object based structures to even more specialized ones, which are restricted towards an single `<xsd:type>`. This would allow the usage of for example long values as keys instead of Java objects, which is far more effective.
The specialized lookups for Allen's Interval relations implemented for the Interval Tree could not be evaluated correctly due to the overall very poor performance of the tree. Here further investigations are definitively necessary.
The new range lookups integrated in QDL perform best when using the baseline implementation. Custom range lookups for the standard indexing structures, i.e. B-Tree, B+Tree and Red-Black Tree, do not bring any advantages. This is most likely due to the fact that these structures are not designed to perform these tasks and thus the range lookups are very cost intensive.

Based on this results, it can be said that the usage of the new indexing structures is (with the current implementation) only encouraged when longer tuples are used. B+Trees should be preferred for indexing single values in case simple range lookups are used, otherwise are Red-Black Trees to be preferred.

5.2 Evaluating the Stream Reasoner

This section is dedicated to the evaluation of SHFC by comparing its performance against other popular and widely used frameworks, namely CQELS, EP-SPARQL, and C-SPARQL, all of which where presented in section 3.2, 3.3 and 3.4, respectively.
Several benchmarks for comparing stream reasoners have been proposed, including

- SRBench [31]: functional coverage of the query languages
- LSBench [32]: throughput correctness by comparing and quantifying the mismatch
- CSRBench [33]: correctness based on the operational semantics
- CityBench [34]: capture the performance metrics using smart-city datasets

Despite all of these benchmarks, comparing the performance of stream reasoning systems is still difficult, as they highly differ on syntactic and semantic levels. For example, CQELS and EP-SPARQL/ETALIS use a reactive execution strategy, meaning that queries are executed as soon as new data becomes available. C-SPARQL on the other side executes the queries periodically.
First, it was planned to use the LSBench as basis for the evaluation, as it covers all three presented RSP systems. However, the LSBench software is poorly documented and thus it was not possible to get it fully up and running. Contacting the developers to get the source code and obviously missing input files was also

not successful.
Therefore, it was decided to use the CityBench, which supports CQELS and C-SPARQL.

5.2.1 Setup

The CityBench aims on evaluating RSP system within a smart city context using smart city data provided by the CityPulse project[35], which include real-time data streams generated from various sensors deployed within the city of Aarhus, Denmark.
These data sets are associated with Vehicle Traffic, Parking, Weather, Pollution, Cultural Event, Library Events and User Locations. It provides a set of continuous queries covering a variety of data- and application- dependent characteristics associated with the smart city context. The queries relevant for this evaluation are [5]:

`Q1` *What is the traffic congestion level on each road of my planned journey?*
A query that monitors the traffic congestions from all traffic sensors located on the roads which are part of a planned journey. This query involves simple lookups for background knowledge on every execution.

`Q5` *What is traffic congestion level on the road where a given cultural event X is happening? Notification for congestion level should be generated every minute starting from 10 minutes before the event X is planned to end, till 10 minutes after.*
This is a conditional query which should be deployed at the occurrence of an event and has a predefined execution duration.

`Q8` *Which parking places are available nearby library event X?*
This query combines parking data streams with the static information about the library events to locate parking spaces nearby the library.

`Q10` *Notify me every 10 minutes, about the most polluted area in the city.*
`Q10` is an analytical query executed over the pollution data streams to find out which area in the city in most polluted and how this information evolves. This query does not require any background knowledge.

In order to compare SHFC against C-SPARQL and CQELS the evaluation conducted [34] by Ali et al. was repeated. There, they compared C-SPARQL and CQELS with respect to (i) Latency, (ii) Memory Consumption, and (iii) Completeness. For all evaluations the streams were configured to send 5 events every 1000ms.

Latency refers to the time consumed by the RSP engine between the input arrival and output generation. It was evaluated by

[5] Detailed description of the data, data streams and queries can be found in [34]

(i) increasing the number of input streams within a query (`Q10` with 2,5 and 8 inputs) and

(ii) by increasing the number of concurrent queries executed (`Q1`, `Q5` and `Q8` with 1,10,20 for number of concurred queries),

to simulate different workloads.

Memory Consumption of the RSP engines is evaluated by observing the usage of system memory during a time window of 15 minutes while

(i) performing a concurrent execution of an increasing number of queries (`Q1` and `Q5` with 1, 10, 20 concurrent query instances), and

(ii) performing query executions with increasing the size of background data (`Q5`, increasing the size from 3MB to 20MB and 30 MB).

Completeness is evaluated by executing Query `Q1` with variable input rate of data streams. The correctness is then computed by letting the RSP produce x outputs. These outputs include y unique IDs resulting in a completeness $c = y/x$.

Ali et al. decided to use these configurations as these provide the most representative and simplest to compare results for CQELS and C-SPARQL, especially as some of the other CityBench queries are not supported by either of the two RSP systems. To evaluate SHFC these queries had to be transformed into SHFC rules. This turned out to be a none trivial task as SHFC does not support rule specific input streams, but only one combined stream. However it also showed the high diversity and expressiveness of SHFC, as queries could be represented by rules in multiple ways.

```
1  select ?obId1 ?obId2
2  where {
3  stream <SampleEventService:AarhusPollutionData182955> [range 3000ms slide 1s]
4  {?obId1 a ?ob.
5  ?obId1 <ssn:observedProperty> ?p1.
6  ?obId1 <sao:hasValue> ?v1.
7  ?obId1 <ssn:observedBy> <SampleEventService:AarhusPollutionData182955>.
8  }
9  stream <SampleEventService:AarhusPollutionData158505> [range 3000ms slide 1s]
10 {?obId2 a ?ob.
11 ?obId2 <ssn:observedProperty> ?p2.
12 ?obId2 <sao:hasValue> ?v2.
13 ?obId2 <ssn:observedBy> <SampleEventService:AarhusPollutionData158505>.
14 }
15 }
```

Listing 5.4: CQELS representation of Q10 of the CityBench benchmark. Short namespaces were used for better readability.

For example, the CQELS query shown in Listing 5.4 can be rewritten by

1. Creating one rule for each input stream specific computation in the query. Results are collected into one binding table per rule and are published according to the slide statement in the original query.

```
# stream <SampleEventService:AarhusPollutionData182955>
# stream <SampleEventService:AarhusPollutionData158505>

$Q10_1
?obId1 <rdf:type> ?ob
?obId1 <ssn:observedProperty> ?p1
?obId1 <sao:hasValue> ?v1
?obId1 <ssn:observedBy> <SampleEventService:AarhusPollutionData182955>
<1000ms>
?obId1

$Q10_2
?obId2 <rdf:type> ?ob
?obId2 <ssn:observedProperty> ?p2
?obId2 <sao:hasValue> ?v2
?obId2 <ssn:observedBy> <SampleEventService:AarhusPollutionData158505>
<1000mms>
?obId2
```

2. Combining the stream specific parts of the original query in one huge rule, which also collects results in a binding table and publishes them according to the slide statement of the original query. This might result in longer computation times per rule, but would offer the benefit that the results are collected in a single table.

```
# stream <SampleEventService:AarhusPollutionData182955>
# stream <SampleEventService:AarhusPollutionData158505>

$Q10_1
?obId1 <rdf:type> ?ob
?obId1 <ssn:observedProperty> ?p1
?obId1 <sao:hasValue> ?v1
?obId1 <ssn:observedBy> <SampleEventService:AarhusPollutionData182955>
?obId2 <rdf:type> ?ob
?obId2 <ssn:observedProperty> ?p2
?obId2 <sao:hasValue> ?v2
?obId2 <ssn:observedBy> <SampleEventService:AarhusPollutionData158505>
<1000ms>
?obId1 ?obId2
```

3. Combining both aforementioned approaches by creating a rule for each stream specific computation in the query. In contrast to approach (1) the results are not collected and published as one binding table but fed back into the stream. An additional rule collects the results of the other ones and combines them to a binding table, which is published according to the slide statement in the original query.

```
# stream <SampleEventService:AarhusPollutionData182955>
# stream <SampleEventService:AarhusPollutionData158505>

$Q10_1
?obId1 <rdf:type> ?ob
?obId1 <ssn:observedProperty> ?p1
?obId1 <sao:hasValue> ?v1
?obId1 <ssn:observedBy> <SampleEventService:AarhusPollutionData182955>
=>
"?obId1" ?obId1

$Q10_2
?obId2 <rdf:type> ?ob
?obId2 <ssn:observedProperty> ?p2
?obId2 <sao:hasValue> ?v2
?obId2 <ssn:observedBy> <SampleEventService:AarhusPollutionData158505>
=>
```

```
18  "?obId2" ?obId2
19
20  $Consumer
21  "?obId1" ?obId1
22  "?obId2" ?obId2
23  <1000ms>
24  ?obId1 ?obId2
```

The measurements for SHFC were performed using the rule composition that best reflects the behaviour of the original rule.
The CQELS and SHFC versions of the queries can be found in Appendix **add appendix.**.

All experiments presented in this thesis are reproducible. Systems, queries and datasets used in this evaluation, as well as the produced results, are available at **link**.

5.2.2 Results

Remark: The RSPs will not be compared directly against each other, as all evaluated RSPs use entirely different semantics, functionalities and follow different strategies in submitting results, which results in non-comparable measurements for the latency. Instead, this evaluation will focus on the general behaviour in the measurements that can be observed for each RSP separately.

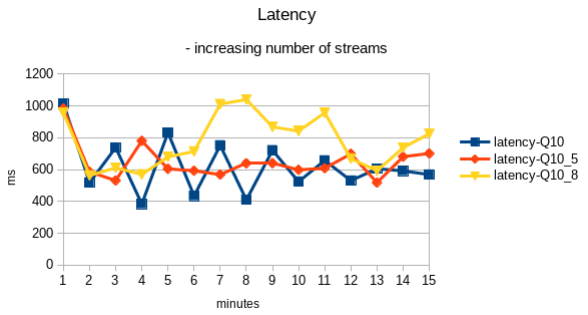
Latency - increasing number of streams

When evaluating the latency according to the first setup (increasing number of streams) we achieved the results shown in Figures 5.7. The overhead for C-SPARQL was minimal with increasing number of streams, in all scenarios the latency fluctuated around 800ms with peeks at ca. 1100ms and lows at 600ms. The peeks where always at the beginning of the experiment, indicating that C-SPARQL either needs some warm-up time or query specific processing was applied at the beginning of the measurements.

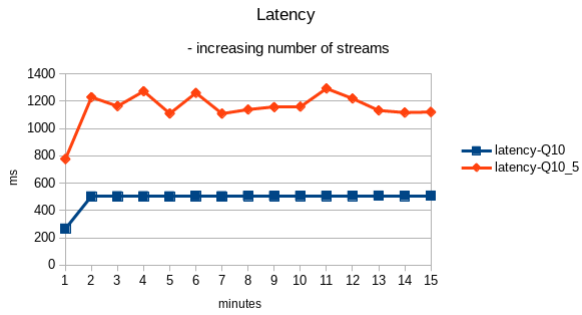
However, the results for CQELS were quite different with extremely low latencies at the beginning. Later on, after approximately 1 to 2 minutes the latency commuted around 500ms for the setup with two streams and around 1000ms for the setup with 5 streams. No results could be gathered for the setup with 8 streams, which was also observed by Ali et al.[34] in their evaluation.

In this scenario, SHFC achieved rather consistent results for all three setups. The latency increased with the number of stream to be processed, which might be due to the higher processing effort for the single rules, as well as, due to the increased traffic on SHFC's buffer. It also worth mentioning that the latency also increases over time, which could indicate that the buffer continuously fills up, resulting in the fact that events cannot be processed the moment they arrive.
The lower latencies compared to the other tested systems can be explained with the implementation of the query in SHFC. As the stream specific processes were

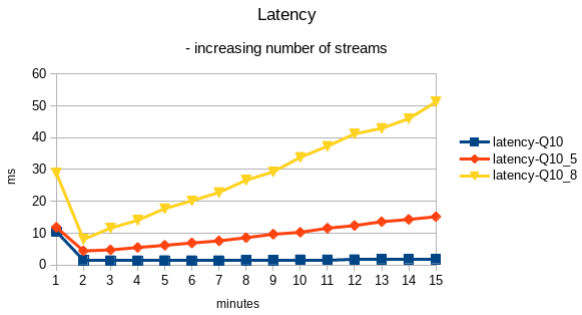
separated in different rules, the workload was distributed in 2, 5, or 8 parallel threads, respectively. This seams reasonable, if one considers that for this query only simple pattern matching and no background lookups were necessary.



(a) C-SPARQL



(b) CQELS



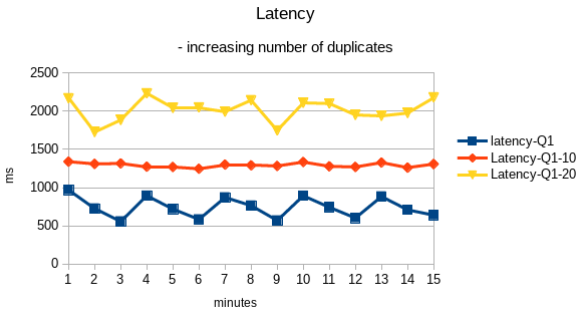
(c) SHFC

FIGURE 5.7: Latency with increasing number of streams.

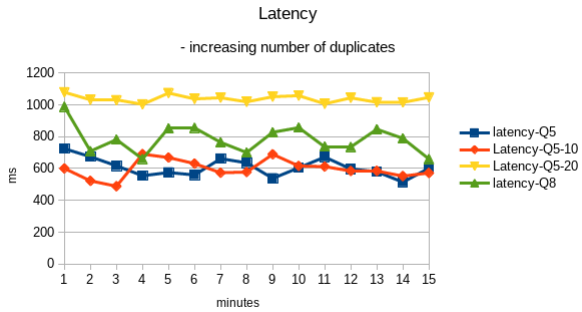
Latency - increasing number of concurrent executions

For `Q1` a constant size of overhead for the latency with the increasing number of concurrent queries can be observer for CSPARQL. The latency peeks at the beginning of the measurements are also present in this scenario, which makes it implausible that the peeks in the first scenario were due to query specific matters, as here other rules were used, compare Figure 5.8(a).
The measurements for `Q5`, Figure 5.8(b), look similar, however the increase for this queries seams to be not as high compared to `Q1`. For `Q8` no statements can be made, as the evaluation did not provide any measurements for the runs with 10 and 20 concurrent queries, which was also observed by Ali et al., however they were not able to provide any explanations for this.

For CQELS the data (Figure 5.9) shows that the latency increases significantly for all three queries, when increasing the number of concurrent queries from 0 to 10. However, the performance suffers not much when the number of runs is increased from 10 to 20. Additionally, high fluctuations in the measurements fo query `Q1` can be observed, especially for the scenarios with 10 and 20 concurrent



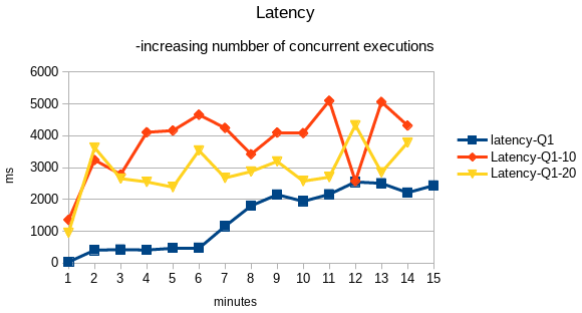
(a) C-SPARQL- Q1



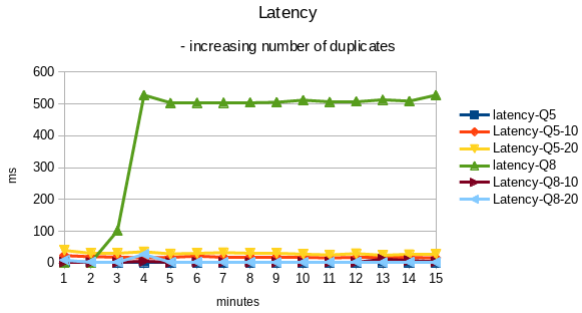
(b) C-SPARQL - Q5 & Q8

FIGURE 5.8: C-SPARQL's latency with increasing number of concurrent executions.

executions. It should also be mentioned that the latency for Q1 is steadily increasing over time. Query Q8 showed some strange behaviour, as for the setup without duplicates the latency jumped form around 2ms to 500ms. The latency stayed this high for the rest of the experiment. This could not be observed for the other experiments involving Q8. The log files of the experiments did not provide any clue on what trigged this behaviour.



(a) CQELS- Q1

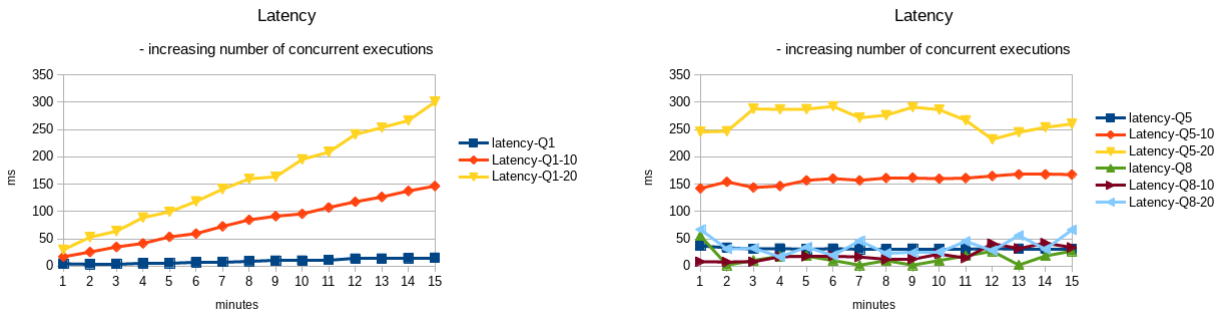


(b) CQELS - Q5 & Q8

FIGURE 5.9: CQEL's latency with increasing number of concurrent executions.

The measurements conducted for SHFC are quite interesting. For Q1 one can observe a steady increase of the latency over time, compare Figure 5.10(a). This effect strengthens with the increasing number of concurrent executions. This could indicate a problem in the implementation of Q1 in SHFC. Here, two rules were used to collect the requested information, which were then fed back into the stream where they are collected and published by a third rule. This means that, with increasing number of concurrent executions more and more tuples are fed back into the stream, resulting in higher latencies. This highlights the importance of good rule design in SHFC.
However, the data for Q5 and Q8 look quite different, compare Figure Figure 5.10(b). Both queries have a almost constant latency over time. This emphasizes the assumption made for Q1 as both queries are implementing a single SHFC rule. The effect of increasing the number of concurrent executions on both queries

is quite different. For Q8 only small latency increases can be observed, whereas for Q5 the increases are very big. This can be traced back to the different background lookups performed by the queries. Q8 performs rather simple queries, whereas Q5 requires complex queries with multiple WHERE and FILTER clauses. As SHFC does not support parallel querying of the background ontology to prevent concurrency issues, it is obvious that Q5 suffers from an increase of concurrent executions, as rules have to wait until the queries of the others have been processed. This is obviously a point to address in further versions of SHFC.



(a) CQELS- Q1 (b) CQELS - Q5 & Q8

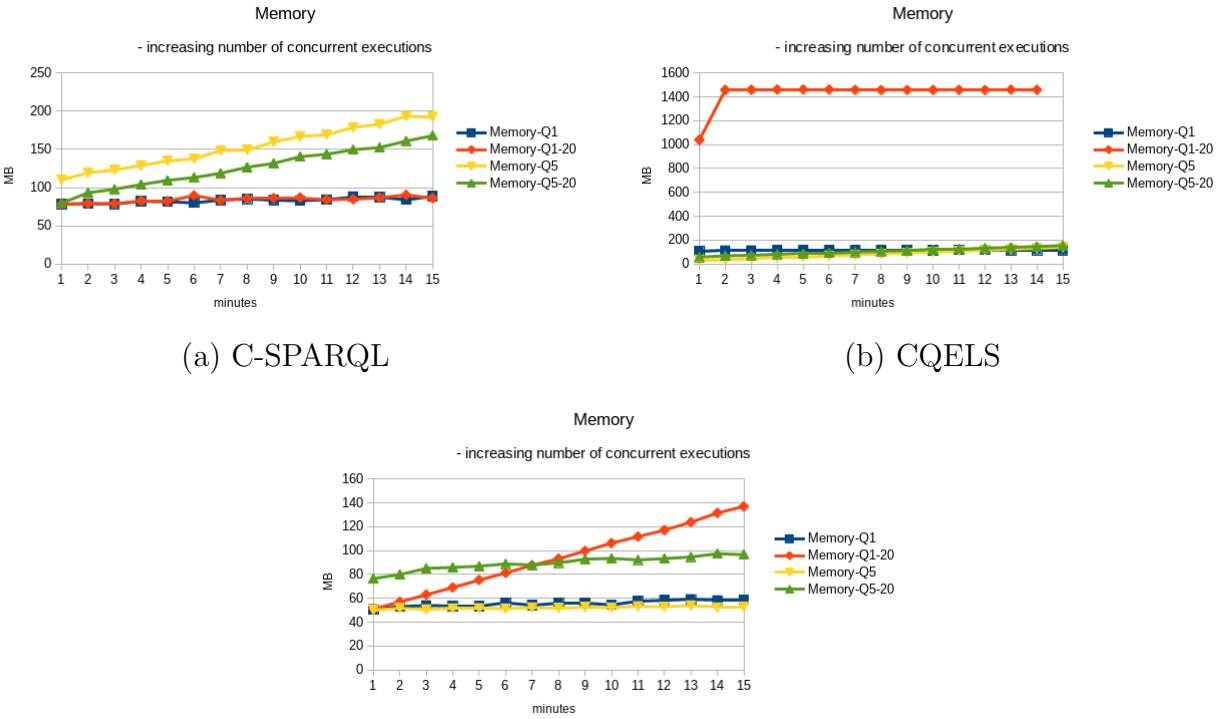
FIGURE 5.10: SHFC's latency with increasing number of concurrent executions.

Memory - Increasing number of concurrent executions

For C-SPAQRL, a constant increase in memory consumption for Q5 can be observed, compare Figure 5.11(a). However, the rate of increase in memory is similar for both the execution of a single and of 20 concurrent queries. Overall, increasing the number of concurrently executed queries had no significant impact for either Q1 or Q5.
In contrast, the memory consumption of CQELS, Figure 5.11(b), was quite stable over the duration of the experiments. For Q5 the memory consumption increased only slightly overtime, whereas for Q1 no increase over time could be observed. However, increasing the number of concurrently running threads had a major impact on the memory consumption when running Q1, where the memory consumption increased to a value of around 1500MB.
On a first glance one notices that for SHFC the basic memory consumption for Q1 and Q5 is nearly the same, see Figure 5.11(c). However, when increasing the number of concurrent executions the memory consumption of Q5 increases by a constant factor and the one for Q1 increases drastically over time. This shows the effect of the rule design on the memory consumption quite nice, as Q1 implements a fed back based approach, which leads to higher load on the stream, whereas Q5 uses a single rule approach.

Memory - Increasing size of Background Knowledge

Here, the results for all measured RSPs are similar, see Figure 5.12. With increasing the size of the background ontologies from 3MB to 20 and 30MB, the memory



(a) C-SPARQL

(b) CQELS

(c) SHFC

FIGURE 5.11: Memory consumption while increasing size of concurrent executions.

used by the RSPs is increased. Nevertheless, there are slight differences between the RSPs. The memory footprint of SHFC initially increases by an constant factor and has only a small increase during the experiment, whereas the other approaches have a rapid increase of memory consumption during the course of the experiment. This difference can be explained by the fact that SHFC indexes all information of the ontology at compile time.

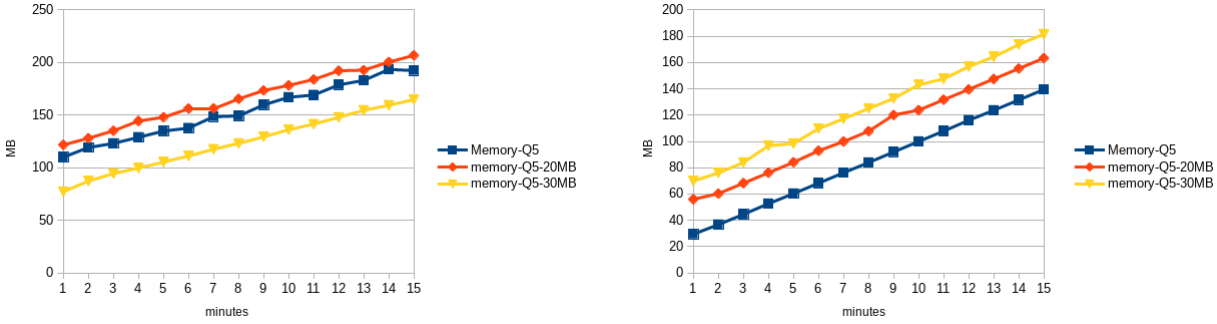
Completeness

The results for completeness were quite interesting. The completeness of CQELS continuously dropped with increasing streaming rates, while the completeness of C-SPARQL and SHFC was mostly close to 100% in all setups, compare Figure 5.13.

5.2.3 Summary

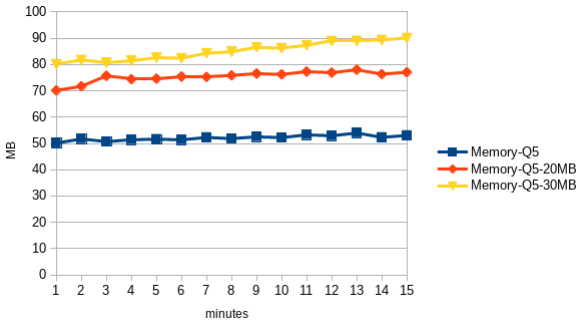
During the setup and execution of the evaluation some points regarding SHFC came up.

1. Support for rule specific streams, would be an nice addition which in fact could result in performance increases.



(a) C-SPARQL

(b) CQELS



(c) SHFC

FIGURE 5.12: Memory consumption while increasing size of background ontology.

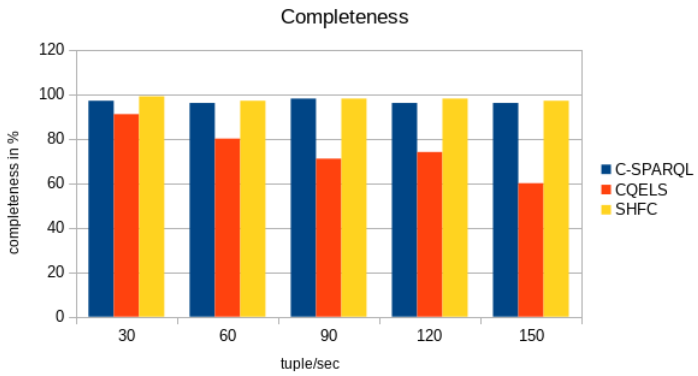


FIGURE 5.13: Completeness with increasing number of tuples per second.

2. The missing support for timed windows made it a none trivial task to exactly model the behaviour of the CityBench queries. Thus, an extension of SRDL with this feature should be considered.

3. The ability to create custom operations for @Test and @Action clauses was a great help. This feature makes SHFC versatile and flexible, which could especially come in handy in scenarios with lots of complex and specific computation and tasks.

However, one should pay attention to the application environment when designing the rules. The option to feed back derived elements into the stream might not always be an reasonable option, as seen from the evaluation of `Q1`.

Unfortunately non of the benchmarked queries features more complex, e.g. time dependent reasoning, which is possibly due to the fact that neither CQELS nor C-SPARQL support such queries. Thus, the real strength of (S)HFCs n-tuple support could not be fully presented.
However, combining the findings from the index structure evaluation, the short lookup times when using indexing structures on tuples of higher length and the present evaluation one can strongly assume that n-tuple support is beneficial in those scenarios. This assumption is based on the observation that SHFC performed well in all queries were background knowledge was accessed. It is thus only reasonable to say that the performance will increase with the usage of indexing structures in case larger tuples are used, e.g. when transaction and valid time are important. It is also clear that the performance of the other RSP systems will drop when background knowledge tagged with transaction and or valid time will be queried, as the container based triple representation promoted by the w3c will cause higher computation efforts.

Chapter 6
Conclusion & Future Work

This work was born from the intuition that a stream reasoner will most likely benefit from an n-tuple based background ontology. In theory, this should grant us faster and easier lookups of background knowledge, which is essential for real-time stream reasoning.

6.1 Conclusion

During the course of this work, a stream reasoner, SHFC, was developed and presented. An integral part of SHFC is HFC, a semantic repository and rule-based stream reasoner that not only supports classic RDF/OWL triple but also n-tuple based ontologies. The presented reasoner uses a rule-based approach to extract relevant events from a stream of events and derive new information from these, which can then likewise be utilized to perform one or all following actions:
They can be collected using binding tables which are published in user defined periods. This is the standard use-case for RSP systems and supported by all discussed systems. In addition, the information can be fed back in the stream where they possibly trigger new rules. This approach is also used by the complex event processing tool ETALIS. Finally, the information can be directly used to update the background ontology, which is not supported by any other RSP system.
During the derivation process one or more ontologies managed by HFC might be used as background knowledge. In order to speed up the necessary process querying of background knowledge, HFC has been extended with additional indexing structures and new features for the query language QDL. The evaluation results suggest that these queries perform effectively, even on large ontologies. This good performance is mainly due to the fact that HFC pre-calculates all inferences and therefore reduces queries to an indexing problem. This also means that the memory consumption of SHFC is much higher compared to other RSP systems.
The evaluation conducted on the basis of the citybench, using smart city data provided by the CityPulse project[35], which includes real-time data streams generated from various sensors, has shown that SHFC seems quite practicable. With the correct rule designs it provides reliable latencies and a high completeness in most scenarios. However, the evaluations carried out have also shown that it was not the best idea to combine all the input streams into a single one, which is then observed by all rules, as this becomes a bottleneck in scenarios with high throughput.

6.2 Future Work

The presented reasoning framework is only a first step to an effective and versatile system. There are several new features and extension to be introduced in future versions.
First and foremost, code optimizations in the rule processing and an integration of continuous queries are planned to increase the performance of the reasoner and make it compatible to existing formats, such as C-SPARQL and CQELS. The new query language will then implement Allen's Interval Relations, too.
As mentioned during the course of this thesis the implementation of the Interval Tree has to be revisited and improved to fully benefit from the integration of Allen's Interval Relations. It is also planned to experiment with additional indexing structures and other, optimized, implementations of the presented structures. By introducing these new structures the background ontology could be reviewed and maintained more effective. In addition, these new structures could make the reasoner more versatile.
One could also think about an integration of the RCC8**ref** relations to the extended QDL to provide a better support for geo-spatial queries.
Last but not least, it is planned to extend the SRDL with several new options: The first one would be the possibility to feed derived facts into a separate stream or add them to an ontology different from the one used as the background ontology. This way the reasoner could be utilized to populate an entirely new ontology. Another, addition that could be useful in future is the possibility to use the rules to collect data with respect to a given time interval and then returning the collected data in a binding table. This way the timed sliding windows of other RSP like C-SPARQL or C-QELS could be realized in SHFC.
As one can imagine, so far only the surface of what is possible and imaginable with HFC based stream reasoning was scratched.

Appendix 1: Queries used to evaluate the new Indexing Structures

```
SELECT ?s ?o
WHERE "0"^^<xsd:long> ?s <rdf:type> <univBench:FullProfessor>
        & "5457"^^<xsd:long> ?s <univBench:emailAddress> ?o

SELECT ?s ?o
WHERE "0"^^<xsd:long> ?s <rdf:type> <univBench:FullProfessor>
        & ?t ?s <univBench:teacherOf> ?o

SELECT ?s ?o
WHERE "0"^^<xsd:long> ?s <rdf:type> <univBench:FullProfessor>
        & "572"^^<xsd:long> ?s ?p ?o

SELECT ?s ?o
WHERE "0"^^<xsd:long> ?s <rdf:type> <univBench:FullProfessor>
        & "572"^^<xsd:long> ?s <univBench:teacherOf>  ?o
FILTER ?o != <http://www.Department0.University0.edu/GraduateCourse14>

SELECT ?s ?o
WHERE "0"^^<xsd:long> ?s <rdf:type> <univBench:FullProfessor>
        & "572"^^<xsd:long> ?s <univBench:teacherOf> ?o
AGGREGATE ?count = CountDistinct ?o

SELECT ?s ?o
WHERE "0"^^<xsd:long> ?s <rdf:type> <univBench:FullProfessor>
        & [ "0"^^<xsd:long> , "6666"^^<xsd:long> ] ?s <univBench:teacherOf> ?o
```

Listing 1: Evaluated queries. Using ontology encoding the transaction time at the first position of each tuple.

```
SELECT ?s ?o
WHERE ?s <rdf:type> <univBench:FullProfessor> "0"^^<xsd:long> "0"^^<xsd:long>
        & ?s <univBench:emailAddress> ?o "8934"^^<xsd:long>  "9186"^^<xsd:long>

SELECT ?s ?o
WHERE ?s <rdf:type> <univBench:FullProfessor> "0"^^<xsd:long> "0"^^<xsd:long>
        & ?s <univBench:teacherOf>  ?o ?t1 ?t2

SELECT ?s ?o
WHERE ?s <rdf:type> <univBench:FullProfessor> "0"^^<xsd:long> "0"^^<xsd:long>
        & ?s ?p  ?o "2955"^^<xsd:long>  "4865"^^<xsd:long>

SELECT ?s ?o
WHERE ?s <rdf:type> <univBench:FullProfessor> "0"^^<xsd:long> "0"^^<xsd:long>
        & ?s <univBench:teacherOf>  ?o "2955"^^<xsd:long>  "4865"^^<xsd:long>
FILTER ?o != <http://www.Department0.University0.edu/GraduateCourse14>

SELECT ?s ?o
WHERE ?s <rdf:type> <univBench:FullProfessor> "0"^^<xsd:long> "0"^^<xsd:long>
        & ?s <univBench:teacherOf> ?o "2955"^^<xsd:long>  "4865"^^<xsd:long>
AGGREGATE ?count = CountDistinct ?o

SELECT ?s ?o
WHERE ?s <rdf:type> <univBench:FullProfessor> "0"^^<xsd:long> "0"^^<xsd:long>
        & ?s <univBench:teacherOf> ?o D "5000"^^<xsd:long>  "9999"^^<xsd:long>
```

Listing 2: Evaluated queries using ontology with valid time at the two last positions of each tuple

Bibliography

[1] Patrick Hayes. RDF Semantics, 2004. URL http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

[2] Herman J. ter Horst. Completeness, decidability and complexity of entailment for {RDF} schema and a semantic extension involving the {OWL} vocabulary. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(23):79 – 115, 2005. ISSN 1570-8268. doi: http://doi.org/10.1016/j.websem.2005.06.001. URL http://www.sciencedirect.com/science/article/pii/S1570826805000144. Selcted Papers from the International Semantic Web Conference, 2004 ISWC, 20043rd. International Semantic Web Conference, 2004.

[3] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993. ISSN 1042-8143. doi: http://dx.doi.org/10.1006/knac.1993.1008. URL http://www.sciencedirect.com/science/article/pii/S1042814383710083.

[4] Rudi Studer, V.Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, 1998. ISSN 0169023X. doi: 10.1016/S0169-023X(97)00056-6.

[5] Frank Manola, Eric Miller, and B. McBride. RDF primer. *W3C recommendation*, 10(February 2004):1–107, 2004. doi: citeulike-article-id:498572. URL http://www.uazuay.edu.ec/bibliotecas/mbaTI/pdf/RDFPrimer.pdf{%}5Cnfile://localhost/Users/enrico/Documents/Papers/2004/Manola/W3Crecommendation2004Manola.pdf{%}5Cnpapers://e411e0f6-a7ed-4df0-b2a1-432575887559/Paper/p3877.

[6] W C Owl Working Group. OWL 2 Web Ontology Language Document Overview, 2009.

[7] Guus Schreiber and Yves Raimond. Rdf 1.1 Primer, 2014. URL http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140225/.

[8] Steffen Staab and Rudi Studer. Handbook on Ontologies. *Decision Support Systems*, page 654, 2007. ISSN 10744770. doi: 10.1007/978-3-540-92673-3. URL http://www.gbv.de/du/services/toc/bs/368354474.

[9] Hans-Ulrich Krieger and Christian Willms. Extending owl ontologies by cartesian types to represent n-ary relations in natural language. In *In Proceedings of the IWCS Workshop on Language and Ontologies.*, 2015.

[10] Pat Hayes, Jeremy Carroll, Chris Welty, Michael Uschold, Bernard Vatant, Frank Manola, Ivan Herman, and Jamie Lawrence. Defining N-ary Relations on the Semantic Web, 2006. URL http://www.w3.org/TR/swbp-n-aryRelations/.

[11] Herman J. ter Horst. Combining RDF and part of OWL with rules: Semantics, decidability, complexity. In *The Semantic Web - ISWC 2005, 4th International*

Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings, pages 668–684, 2005. doi: 10.1007/11574620_48. URL http://dx.doi.org/10.1007/11574620_48.

[12] Geert-Jan Kruijff and Shanker Keshavdas. Functional Mapping for Human-Robot Cooperative Exploration. *International Journal of Computer and Applications (IJCA)*, 36(1):o.A., 2014. ISSN 1206212X. URL http://www.dfki.de/web/forschung/publikationen/renameFileForDownload?filename=keshavdas_kruijff_ijca.pdf&file_id=uploads_2135.

[13] Hans-Ulrich Krieger. An Efficient Implementation of Equivalence Relations in OWL via Rule and Query Rewriting. In *2013 IEEE Seventh International Conference on Semantic Computing*, pages 260–263. IEEE, sep 2013. ISBN 978-0-7695-5119-7. doi: 10.1109/ICSC.2013.51. URL http://ieeexplore.ieee.org/document/6693526/.

[14] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83, New York, NY, USA, 2004. ACM. ISBN 1-58113-912-8. doi: http://doi.acm.org/10.1145/1013367.1013381. URL http://portal.acm.org/citation.cfm?doid=1013367.1013381.

[15] Barry Bishop, Atanas Kiryakov, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. Owlim: A family of scalable semantic repositories. Technical report, 2010.

[16] Hans-Ulrich Krieger. A Description of HFC DRAFT. URL http://www.dfki.de/lt/onto/hfc/hfc.pdf.

[17] Jan Grant & Dave Beckett. Rdf test cases, 2004. URL http://www.w3.org/TR/rdf-testcases/.

[18] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. *W3C Recommendation*, 2009(January):1–106, 2008. doi: citeulike-article-id:2620569. URL http://www.w3.org/TR/rdf-sparql-query/.

[19] Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. Stream Reasoning: Where We Got So Far. In *Proceedings of the NeFoRS2010 Workshop, co-located with ESWC2010*, 2010. URL http://wasp.cs.vu.nl/larkc/nefors10/paper/nefors10_paper_0.pdf.

[20] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-sparql : A continuous query language for rdf data streams. *International Journal of Semantic Computing*, 04(01):3–25, 2010. ISSN 1793-351X. doi: 10.1142/S1793351X10000936.

[21] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, pages 370–388, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-25072-9. URL http://dl.acm.org/citation.cfm?id=2063016.2063041.

[22] Darko Anicic, Sebastian Rudolph, Paul Fodor, and Nenad Stojanovic. Stream reasoning and complex event processing in ETALIS. *Semantic Web*, 3(4):397–407, 2012. doi: 10.1080/08839514.2012.636616.

[23]

[24] Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic, and Rudi Studer. A rule-based language for complex event processing and reasoning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6333 LNCS, pages 42–57, 2010. ISBN 3642159176. doi: 10.1007/978-3-642-15918-3_5.

[25] J Allen. Maintaining knowledge about temporal intervals. *Communications of ACM*, 26(11):832–843, 1983. URL `http://scholar.google.com/scholar?q=related:SfE4Y6t68aMJ:scholar.google.com/{&}hl=en{&}num=20{&}as{_}sdt=0,5`.

[26] Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, 1972. ISSN 1432-0525. doi: 10.1007/BF00289509. URL `http://dx.doi.org/10.1007/BF00289509`.

[27] Robert Sedgewick and Leo J. Guibas. A dichromatic framework for balanced trees. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 00:8–21, 1978. ISSN 0272-5428. doi: doi.ieeecomputersociety.org/10.1109/SFCS.1978.3.

[28] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1992.

[29] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semant.*, 3(2-3):158–182, October 2005. ISSN 1570-8268. doi: 10.1016/j.websem.2005.06.005. URL `http://dx.doi.org/10.1016/j.websem.2005.06.005`.

[30] Oracle Corporation. Jmh - java micro benchmark harness, 2017. URL `http://openjdk.java.net/projects/code-tools/jmh/`.

[31] Ying Zhang, Pham Minh Duc, Oscar Corcho, and Jean-Paul Calbimonte. *SRBench: A Streaming RDF/SPARQL Benchmark*, pages 641–657. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35176-1. doi: 10.1007/978-3-642-35176-1_40. URL `https://doi.org/10.1007/978-3-642-35176-1_40`.

[32] Danh Le-Phuoc, Minh Dao-Tran, Minh-Duc Pham, Peter Boncz, Thomas Eiter, and Michael Fink. *Linked Stream Data Processing Engines: Facts and Figures*, pages 300–312. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35173-0. doi: 10.1007/978-3-642-35173-0_20. URL `https://doi.org/10.1007/978-3-642-35173-0_20`.

[33] Daniele Dell'Aglio, Jean-Paul Calbimonte, Marco Balduini, Oscar Corcho, and Emanuele Della Valle. *On Correctness in RDF Stream Processor Benchmarking*, pages 326–342. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-41338-4. doi: 10.1007/978-3-642-41338-4_21. URL `https://doi.org/10.1007/978-3-642-41338-4_21`.

[34] Muhammad Intizar Ali, Feng Gao, and Alessandra Mileo. *CityBench: A Configurable Benchmark to Evaluate RSP Engines Using Smart City Datasets*, pages 374–389. Springer International Publishing, Cham, 2015. ISBN 978-3-319-25010-6. doi: 10.1007/978-3-319-25010-6_25. URL https://doi.org/10.1007/978-3-319-25010-6_25.

[35] Dan Puiu, Payam Barnaghi, Ralf Tnjes, Daniel Kumper, Muhammad Intizar Ali, Alessandra Mileo, Josiane Parreira, Marten Fischer, efki Kolozali, Nazli Farajidavar, Feng Gao, Thorben Iggena, Thu-Le Pham, Cosmin-Septimiu Nechifor, Daniel Puschmann, and Joao Fernandes. Citypulse: Large scale data analytics framework for smart cities. 4:1086–1108, 01 2016.