Karlsruhe Institute of Technology

Assessing Hypotheses in Multi-Agent Systems for Natural Language Processing

Master Thesis of

Dominik Th. Fuchß

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr.-Ing. Anne Koziolek
Second reviewer:	Prof. Dr. Ralf H. Reussner
Advisor:	Jan Keim, M.Sc.
Second advisor:	Yves R. Kirschner, M.Sc.

30. April 2020 – 29. October 2020

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 29. October 2020

. . . .

(Dominik Th. Fuchß)

Abstract

In multi-agent systems (MAS) different agents work on a common problem. Such systems are also used in natural language processing (NLP). Agents of an MAS for natural language can generate results with confidence, so called hypotheses. These hypotheses reflect the ambiguity of natural language. If agents are dependent on each other, a wrong hypothesis can quickly lead to error propagation into the hypotheses of the dependent agents. The exploration of hypotheses offers the chance to improve the results of agents. This thesis improves the results of agents of a MAS for NLP by a controlled exploration of the hypothesis search space. Therefore, a framework for the exploration and evaluation of hypotheses is developed. In an evaluation with three agents promising results regarding the improvement could be achieved. For example, *Top-X Exploration* achieved an average improvement of the F1 score of the *Topic Detection* agent from originally 40% to now 49% and of the *Ontology Selection* agent from originally 74% to 79%.

Zusammenfassung

In Multi-Agenten Systemen (MAS) arbeiten verschiedene Agenten an einem gemeinsamen Problem. Auch im Bereich der natürlichen Sprachverarbeitung (NLP) werden solche Systeme verwendet. Agenten eines MAS für natürliche Sprache können neben Ergebnissen auch Ergebnisse mit Konfidenzen, s.g. Hypothesen generieren. Diese Hypothesen spiegeln die Mehrdeutigkeit der natürlichen Sprache wider. Sind Agenten abhängig voneinander, so kann eine falsche Hypothese schnell zu einer Fehlerfortpflanzung in die Hypothesen der abhängigen Agenten führen. Die Exploration von Hypothesen bietet die Chance, die Ergebnisse von Agenten zu verbessern. Diese Arbeit verbessert die Ergebnisse von Agenten eines MAS für NLP durch eine kontrollierte Exploration des Hypothesen-Suchraums. Hierfür wird ein Framework zur Exploration und Bewertung von Hypothesen entwickelt. In einer Evaluation mit drei Agenten konnten vielversprechende Ergebnisse hinsichtlich der Verbesserung erzielt werden. So konnte etwa mit der *Top-X Exploration* eine durchschnittliche Verbesserung des F1-Maßes des *Topic-Detection-Agenten* von ursprünglich 40% auf jetzt 49% und des *Ontology-Selection-Agenten* von ursprünglich 74% auf 79% erreicht werden.

Contents

List of Figures

List of Tables

1. Introduction

Together — one of the most inspiring words in the English language. Coming together is the beginning. Keeping together is progress. Working together is success.

— Edward Everett Hale [25]

Multi-Agent Systems (MAS) aim to work together on a commonly defined problem. According to Dorri, Kanhere, and Jurdak [7] MAS are one of the fundamental methods in *Distributed Artificial Intelligence*. In such systems, *Agents* cooperate with each other and exchange information about a task. The purpose of this exchange is to allow the agents to build on information from other agents. In order to accomplish natural language processing (NLP) tasks with such kind of system, *Programming ARchitecture for Spoken Explanations (PARSE)* [28] has been developed.

PARSE is an MAS for spoken natural language. Each agent of PARSE has its own task, like the detection of actions or the detection of the topics for a given input. Agents may need information from other agents for their work. Due to this, there can be dependencies between the results of agents. In particular, errors in the results of agents can directly influence the results of other agents. Especially when working with natural language, you have to deal with uncertainty in the results. One possibility to do so is that the agents or algorithms in NLP do not simply deliver a result. Instead, the algorithms provide multiple results and corresponding confidence levels that indicate the certainty of the agent for each result. A result and its confidence form a so-called *Hypothesis*. In addition to the existence of hypotheses, dealing with hypotheses is particularly important since existence does not yet prevent error propagation. This thesis presents an approach for the assessment of hypotheses with the goal to improve the results of agents of an MAS for NLP.

First, a simple example is used to show why hypotheses are important for the work with MAS for natural language. Two existing PARSE agents are considered: The Wikipedia-based Word-Sense Disambiguation (Wiki WSD) agent and the Topic Detection (TD) agent. The first one tries to identify the actual sense of ambiguous words for a given input. The second uses this information to retrieve topics for the given input. Currently, the most likely sense of a word is considered to be truth regardless of its confidence. This procedure can lead to error propagation as shown in the example in Figure 1.1. The example deals with the input “I want to make my guests happy by preparing a good bass”. This text is actually about me, who wants to prepare a nice meal (respectively a fish) for my guests. In Figure 1.1 you can see the disambiguation of two ambiguous words and the resulting topics for the input text. The Wiki WSD agent identifies two words that have to be disambiguated: “guests” and “bass”. The disambiguation of “guests” works well because “hospitality” expresses a reasonable meaning. However, if you look at the word “bass”, it is

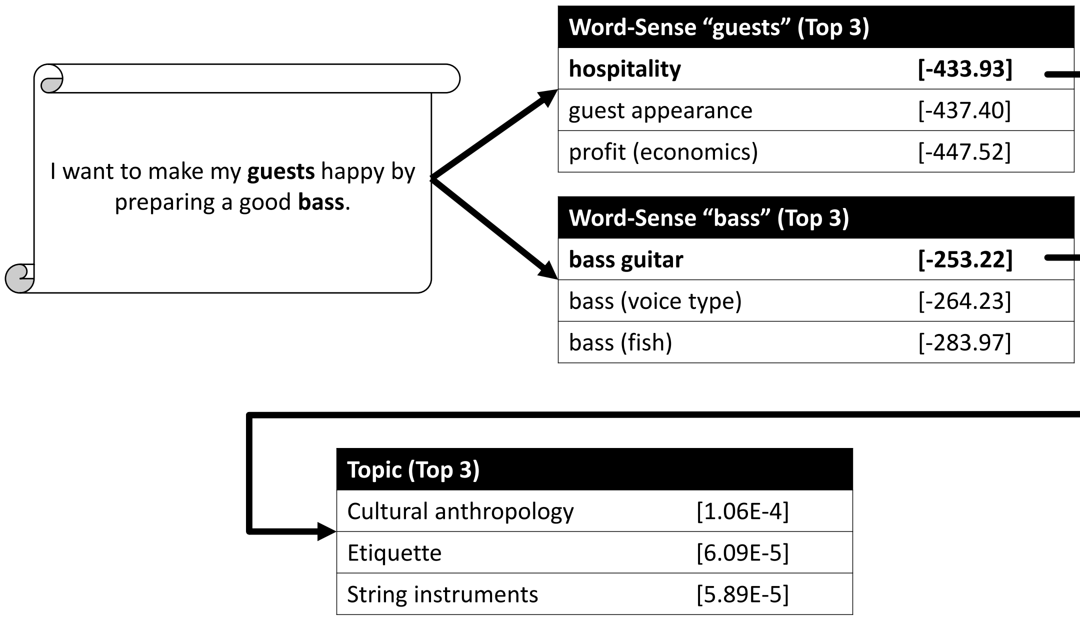

I want to make my **guests** happy by preparing a good **bass**.
Word-Sense "guests" (Top 3)
hospitality [-433.93]
guest appearance [-437.40]
profit (economics) [-447.52]
Word-Sense "bass" (Top 3)
bass guitar [-253.22]
bass (voice type) [-264.23]
bass (fish) [-283.97]
Topic (Top 3)
Cultural anthropology [1.06E-4]
Etiquette [6.09E-5]
String instruments [5.89E-5]

Figure 1.1.: Hypotheses of two PARSE agents: Word-Sense Disambiguation (WSD) and Topic Detection (TD)

noticeable that the "bass guitar" with the highest rating is chosen as sense. Only in third place the actually correct hypothesis is found: bass as a fish. The example showcases that a confidence is primarily seen as an arbitrary floating-point number. Wrong topics have been identified for the example because the PARSE agents use the best rated hypotheses as results. In particular, according to the agent, one topic of the example are "string instruments". The example shows that a consideration of the hypotheses must not be limited to the best hypotheses.

This thesis addresses the problem of error propagation in MAS for natural language. Therefore, the work is based on the Multi-Agent System PARSE [28]. At the moment, a common solution strategy in PARSE is to consider the best rated hypothesis to be correct. In the example you could see that this strategy does not always lead to correct results and especially facilitates error propagation. This thesis aims to explore the search space created by the hypotheses and to achieve better results by assessing the hypotheses found. The search space itself is a search tree that is spanned by different selections of hypotheses. If you look at the example from above, exactly one path is shown. This path is defined by the selection of "hospitality" and "bass guitar". A new path is created by making different selections of hypotheses, such as "hospitality" and "bass (fish)". Each selection opens a new branch in the search tree. The problem with this search space is that it grows exponentially. Therefore, strategies have to be developed how to efficiently explore this search space. In addition, after the search space has been explored, the correct result has to be identified. So the idea of this work is to partially explore the search space and then identify the best results by means of an assessment procedure. This ensures that several hypotheses can be examined efficiently. Since the hypotheses are no longer generated locally for an agent but are part of a search space, it is also possible to follow the progression of an agent's confidence over different branches. This can help to maximize confidence on a larger scale, rather than just looking at confidence locally for one agent at a time.

The following research questions shall be answered:

Research Questions

RQ1: What types of hypotheses are found in agents of an MAS for natural language?

RQ2: Can the results of agents be improved through partial exploration without changing the underlying mechanisms of the agents?

RQ3: Can correct hypotheses be identified after exploration?

The first research question deals with the different kinds of hypotheses. In particular, the focus is on different types of hypotheses that need to be found and identified in MAS for natural language. The second question addresses the issue of whether exploration respectively controlled partial exploration can improve the results of agents. It is particularly important to determine whether mechanisms to control the search space can be applied without losing the advantage of improving the results. The identification of the better results is not the focus of this research question. This identification of the correct hypotheses is the final goal of the work. The focus of the last research question is especially whether candidates for good results can be identified from exploration.

This thesis is structured as follows: First, the fundamentals necessary for understanding the work are introduced (see Chapter 2). Subsequently, I will look at related works in Chapter 3 that deal with similar topics to this work. After that, Chapter 4 introduces the approach of this thesis — the *Agent Analysis Framework*. The detailed development of this Framework is covered in the following three chapters (see Chapter 5, 6, and 7). These chapters are followed by the evaluation of this work (see Chapter 8). This thesis concludes with a summary of the results in Chapter 9.

2. Fundamentals

This chapter introduces the basic knowledge necessary to further understand the thesis. Concepts, projects, and terms are examined in more detail.

2.1. Multi-Agent Systems

According to Dorri, Kanhere, and Jurdak [7], algorithms for *Distributed Artificial Intelligence* can be separated to three categories: “Parallel AI”, “Distributed Problem Solving”, and “Multi-Agent Systems”. Multi-agent systems (MAS) involve *agents* to solve a certain problem. Dorri, Kanhere, and Jurdak define an agent as “an entity that is placed in an environment and senses different parameters that are used to make a decision based on the goal of the entity”. It is therefore crucial for MAS that agents pursue a precise goal. Agents are also characterized by the fact that their individual goals should be part of an overarching goal. This means that they perform individual tasks that together produce an aggregate result. Figure 2.1 summarizes the structure of an MAS. As shown in the

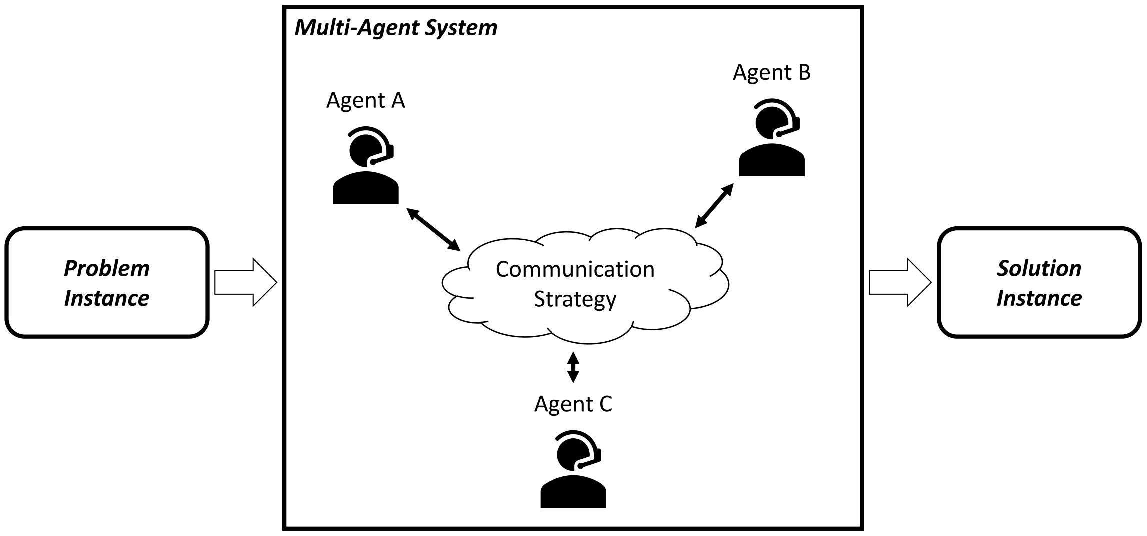


Figure 2.1.: A generic Multi-Agent System

picture, an MAS operates on a common problem instance. Within the MAS several agents communicate via a suitable communication strategy to solve the problem. The agents try to accomplish their tasks and try to find consensus in order to create a consistent result. After processing the problem instance, the MAS provides a solution that is built by the agents.

Following Dorri, Kanhere, and Jurdak, you can distinguish three communication strategies for MAS: "Speech Act", "Message Passing", and "Blackboard". The strategy "speech act" needs a certain language and the definition of two roles for the agents: "Speaker" and "Hearer". Using this strategy, the agents communicate by using the language and taking into account their respective roles. If agents communicate via "message passing", they are directly connected via a communication channel. Here, it is important that not all agents have to be directly connected. The "blackboard" strategy uses a common data structure to enable the communication of the agents. Thus, the agents are able to communicate by manipulating their common data. Since agents talk to other agents to solve their problems, dependencies between agents arise. Thus, the result of one agent can influence the results of other agents.

2.2. Natural Language Processing

Natural Language Processing (NLP) describes a field of research that deals with the machine processing of spoken and written natural language. Typically, this processing is divided into many successive processing steps. A realization of several of these steps is provided by the *Stanford CoreNLP Natural Language Processing Toolkit* [16]. In the following, some steps for processing natural language are introduced. Besides the steps implemented by Stanford Core NLP, I will discuss possible steps in general.

Part-of-speech Tagging (POS Tagging) One of the typical and important steps of natural language processing is *Part-of-speech Tagging* (POS Tagging). The individual parts of a sentence are assigned to word types. An example is shown in Figure 2.2.

Robo	,	go	to	the	kitchen	,	get	the	water	and	bring	it	to	John	.
NNP	,	VB	IN	DT	NN	,	VB	DT	NN	CC	VB	PRP	IN	NNP	.

Figure 2.2.: POS Tagging by *Stanford Core NLP 4.0.0*

For this example *Stanford Core NLP*, respectively, the web interface `https://corenlp.run` was used. As you can see, for instance words that have been identified as nouns (NN and NNP), verbs (VB and VBN), adverbs (RB), or adjectives (JJ). In addition, also punctuation marks have been annotated. Even articles (DT), personal pronouns (PRP), and prepositions (IN) have been labeled. POS Tagging thus represents a first possibility to enrich a sentence with information for interpretation. Simple algorithms could identify the tagged verbs as possible actions in a certain input text. Obviously, this would be a very simplified approach. Nevertheless, it shows how essential POS Tags are.

Named Entity Recognition (NER) A second important step in natural language processing is the Named Entity Recognition (NER). The Stanford CoreNLP recognizes mentioned people, locations, and organizations as named entities [16]. Furthermore, it also finds mentions of money, numbers, dates, and times. An example for the detection of named entities is shown in Figure 2.3.

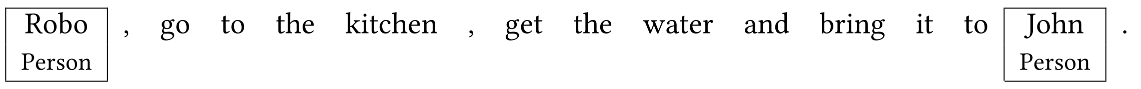


Figure 2.3.: NER by *Stanford Core NLP 4.0.0*

In the example, the named entity recognition identifies two mentioned people. On the one hand, the robot called “Robo” and on the other hand, the person “John”. Obviously, there is no distinction between a robot and a person. Both are classified as person. Like the POS Tagging before, the NER is a basic part of natural language processing. For example, simple algorithms may use the information about mentioned people, dates, locations, or points in time to determine the intent that is encapsulated in a given text. If you think of a booking machine for traveling tickets, the most important information is each person’s traveling destination. So the information needed would be a person and the location to travel.

Co-reference Resolution The resolution of co-references aims to connect referenced entities with their actual entity. An example is shown in Figure 2.4.

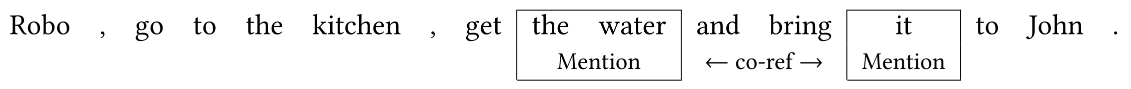


Figure 2.4.: Co-reference Resolution by *Stanford Core NLP 4.0.0*

In the example, the co-reference resolution tries to identify the actual entity that is referred by “it”. You can see that the resolution marks “the water” as entity referenced by “it”. In general, the co-reference resolution is useful whenever one needs to map pronouns to their corresponding entities. This resolution is also important to determine the equivalence of entities within multiple sentences.

Word-sense Disambiguation (WSD) Word-sense disambiguation (WSD) is a useful tool for more complex applications. In many applications that use natural language, it is important to find the actual sense of a word for a given text. Consider the word “bass” as an example. Most people might associate a musical instrument with it. Depending on the context of a sentence, “bass” can mean something completely different. If one considers the sentence “I want to eat a bass”, it is clear that in this case it would be the name of a fish. The WSD annotates the actual sense of different words to them. Thereby, the WSD has to analyze the existing context to disambiguate the words. The knowledge of word senses allows building more complex algorithms. For example, the topics of an input could be identified [14].

2.3. Result vs. Hypothesis

The difference between a result and a hypothesis is essential for further understanding of the work. If one considers algorithms, most of them return exactly one result. In case of a sorting algorithm this could be a sorted list. Considering such deterministic algorithms, there is usually exactly one result that is expected.

Regarding algorithms for natural language processing (NLP), the question of what the result of an algorithm should be is no longer clear. Especially in natural language, uncertainty and ambiguity must be dealt with. The example from the introduction (cf. Figure 1.1) shows that the topic of a sentence is usually more than one result. The sentence that has been used in the example is “I want to make my guests happy by preparing a good bass”. This sentence is about cooking, guest meetings, and especially about preparing a bass. Therefore, it would be difficult (even as a human) to formulate exactly one correct topic as result. Thereby, it becomes important to introduce hypotheses. A hypothesis in this work refers to an assumption. Hypotheses always consist of a resulting value and a confidence. A confidence or confidence level is a score that provides some insight into the certainty of a hypothesis. The structural difference between a result and a hypothesis is shown in Figure 2.5.

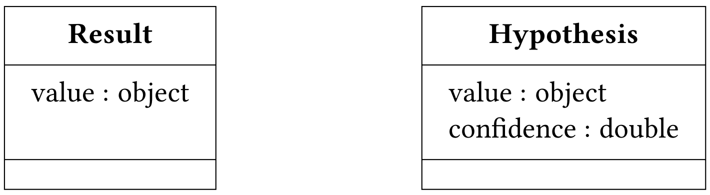


Figure 2.5.: Result vs. Hypothesis: A structural point of view

The idea of hypotheses is similar to probabilistic algorithms: An algorithm that uses hypotheses does not provide one result, but several hypotheses. In context of NLP, this allows working with uncertainties or multiple possibilities. In the introduction it was stated that the question is how to deal with hypotheses. The example of the introduction showed that simply considering the best hypotheses does not necessarily lead to the desired results. Thus, this thesis aims to facilitate the exploration and assessment of such hypotheses.

2.4. Hypotheses Graph & Hypotheses Paths

This section deals with the *Hypotheses Graph* or *Hypotheses Tree*. The Hypotheses Graph refers to a directed acyclic graph with exactly one start node — the *root*. As described in Chapter 1, this thesis aims to analyze different selections of hypotheses of an agent to examine the possible outcomes in the succeeding agents. Different selections open up the possibilities for other agents to find new hypotheses. Thereby, “new” refers to the original hypotheses that arise if the algorithm does not use selection mechanisms, but simply execute the agents. In Figure 2.6, you can see an illustration of such a Hypotheses Graph.

As you can see, the graph can be divided into layers that are each containing one agent. The figure also shows important properties of Hypotheses Graph. These properties are the absence of cycles and the existence of a unique root. The root of the graph is represented by the textual input that will be used by the agents. As a successor, a node containing the hypotheses of the first agent is appended to the root node. Subsequently, different hypotheses are selected by some selection mechanisms. These mechanisms lead

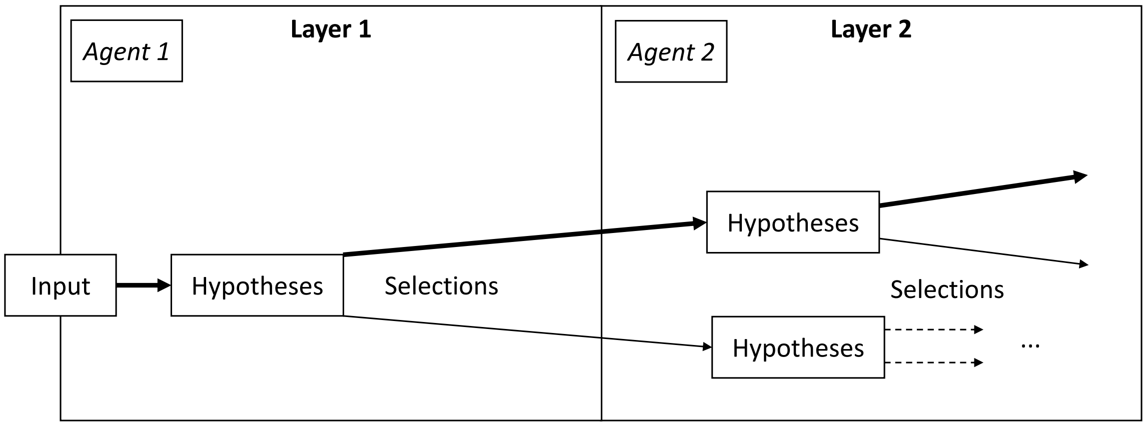


Figure 2.6.: Layered Hypotheses Graph

to new hypotheses of the second agent. Starting at those new hypotheses, new selections can lead to further branching for successive agents. After exploration, one has many hypotheses and possibilities of what the agents provide as final result. The possible results of the agents are the different paths within the Hypothesis Graph. One path is marked by thicker arcs in Figure 2.6. A path is defined by the selections of hypotheses along its edges. These selections determine what are assumed to be correct hypotheses for this path. Therefore, the final task that remains after an exploration is to select the best of all these paths (respectively the actual best result).

2.5. Programming Architecture for Spoken Explanations & Intent-Driven Requirements-to-Code Traceability

The *Programming Architecture for Spoken Explanations (PARSE)* [31] is a multi-agent system for spoken natural language. The architecture of PARSE is shown in Figure 2.7. In a first step, the speech is transformed into an input text. The actual processing of the input text starts with a pre-processing pipeline. This pipeline performs typical tasks of natural language processing, such as tokenization or POS Tagging. As a result, the pipeline generates the so-called *PARSE Graph*. The PARSE Graph represents the common data structure that is used by the agents. Therefore, the PARSE system uses a blackboard communication mechanism (cf. Section 2.1). In addition, the nodes and arcs of the PARSE Graph can contain arbitrary attributes. Thus, the agents communicate by modifying the arcs, nodes, and attributes in the graph. The idea behind PARSE is that different agents work on processing the input and successively add information to the PARSE Graph.

In the following, the actual PARSE Graph is examined in more detail. This is needed as the following chapters deal with agents that store information into the PARSE Graph. A simplified model of the PARSE Graph is shown in Figure 2.8. The graph itself is an instance of the `IGraph` interface. It contains all methods required to create, delete, or modify the arcs and nodes of the graph.

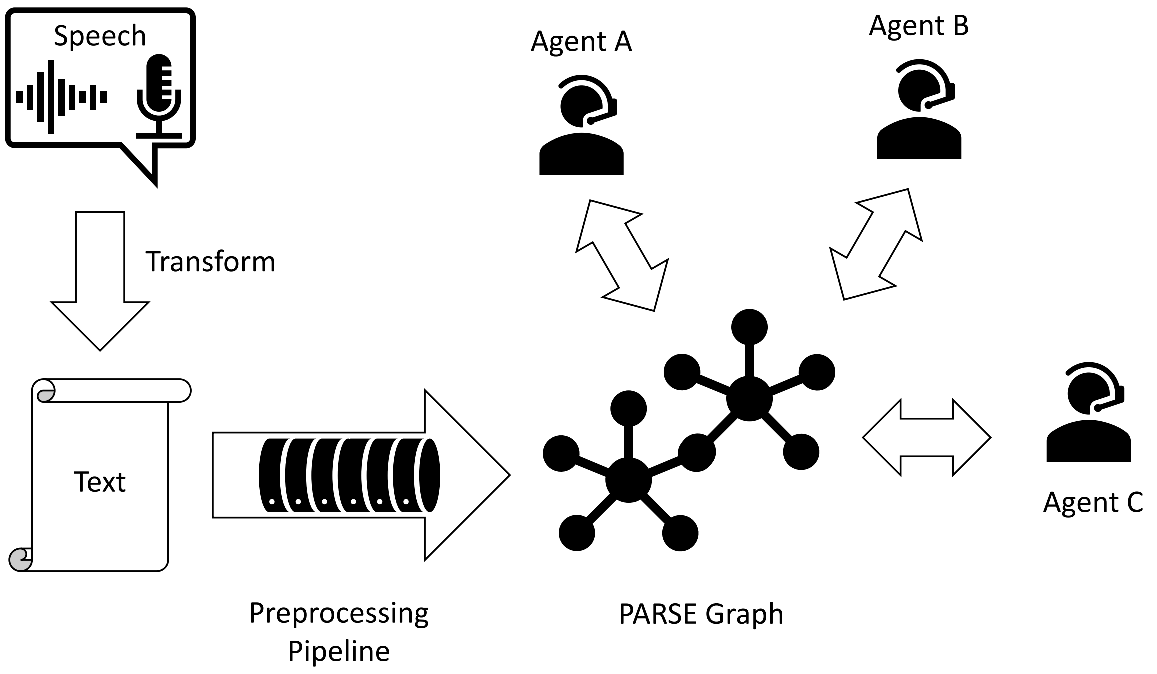


Figure 2.7.: The PARSE Architecture

The graph is a directed graph and consists of `INode` and `IArc`. Since the graph is directed, an arc has specific `source` and `target` nodes. The most important feature of nodes and arcs is the possibility of storing attributes into them. The types of attributes are defined within the node type and the arc type.

Information can be stored as values of a specific attribute in the corresponding arc or node. So far, information can encoded in two ways: On the one hand, by the connections between nodes — the edges. On the other hand, by the attributes or their values in the nodes and arcs of the graph.

The most essential type of nodes are `TokenNodes`. `TokenNodes` are generated by the pre-processing pipeline and represent the individual words (respectively tokens) of the input. For instance, the attributes of `TokenNodes` contain information about the word types. In addition, arcs (`Next`) connecting them to represent the word order from the original input.

The *Intent-Driven Requirements-to-Code Traceability (INDIRECT)* approach by Hey [11] is based on PARSE. INDIRECT uses the architecture of PARSE to perform tasks on written natural language documents. PARSE and INDIRECT differ significantly in the pre-processing pipeline they use. The differences are due to the fact that INDIRECT was designed for written and PARSE for spoken language. For example, INDIRECT can work with punctuation marks and uses a standalone agent for named entity recognition (cf. Chapter 5). PARSE cannot handle punctuation and recognizes named entities directly in the pre-processing pipeline.

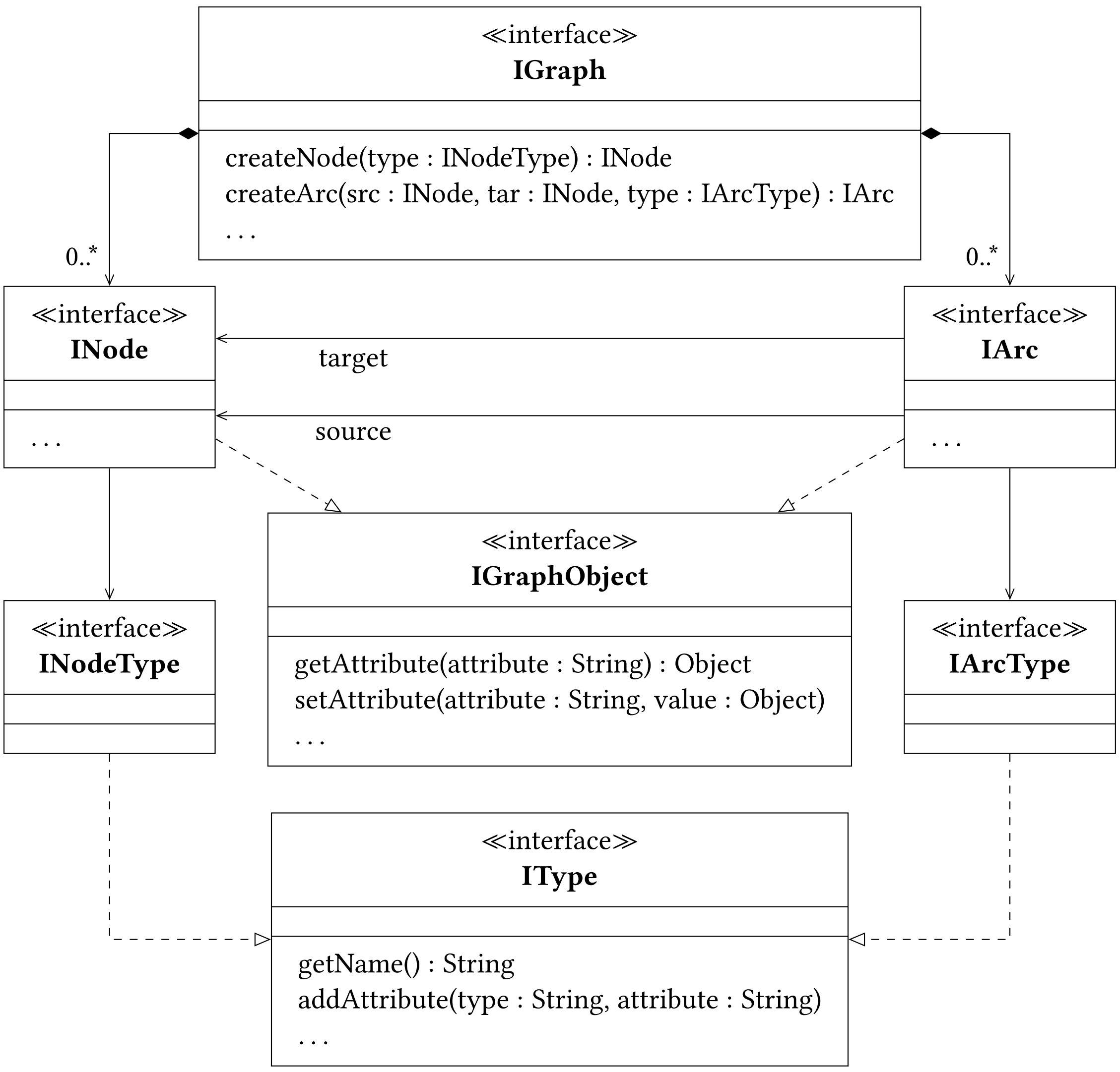

≪interface≫
IGraph
createNode(type : INodeType) : INode
createArc(src : INode, tar : INode, type : IArcType) : IArc
...
0..*
0..*
≪interface≫
INode
...
target
source
≪interface≫
IArc
...
≪interface≫
IGraphObject
getAttribute(attribute : String) : Object
setAttribute(attribute : String, value : Object)
...
≪interface≫
INodeType
≪interface≫
IArcType
≪interface≫
IType
getName() : String
addAttribute(type : String, attribute : String)
...

Figure 2.8.: Model: PARSE Graph (Simplified)

3. Related Work

After the fundamentals for understanding this work were established in the previous chapter, the following one now deals with related work. Since this work is aimed at improving the results of an MAS for natural language, several topics are addressed: The first area of research are confidences in multi-agent systems. This work primarily depends on the confidences of hypotheses generated by the agents of *PARSE*. Thus, it is important to understand how confidences are used in MAS. Furthermore, papers about uncertainty in natural language are discussed. Uncertainty is related to this thesis because confidences reflect some kind of uncertainty in the input text. Also, the research that about the handling of uncertainty in NLP is covered.

3.1. Trust & Confidence in Multi-Agent Systems

This section deals with research on confidences and their effects in MAS. In this context, a confidence is an indicator for the assumed correctness of the behavior of an agent or the correctness of certain data.

The first paper being discussed in the following paragraph, deals with so-called trust models in MAS. In their paper Ramchurn et al. [20] try to create a trust model for interactions in multi-agent systems based on confidences and reputations. Primarily, the authors deal with autonomous agents in "open environments". They define that in such open environments agents can also break their contracts. With the help of confidence and reputation they want to decide which agents are trustworthy for another agent. This shall lead to a decision which agent should be used by other agents for cooperation in the future. In their work they define confidence as being based on past interactions between agents. This work is not directly related to the confidence that an agent assigns to his results. The part relevant for this thesis is the handling of confidence using so-called "confidence levels". In particular, they have found that confidence can be subjective from the perspective of agents. In one example, they consider the levels: "Bad", "Average", and "Good" for a given task. How an actual event or a past execution is now classified depends on the agent performing the classification. For this work it can be concluded that the confidence of an agent can only be compared within the same agent.

After investigating a model for trust, the next paper covers confidence and its modeling for MAS in general. Basheer et al. [3] try to create a new model for confidence in multi-agent systems. They state that "confidence in multi-agent systems gives agents a form of control in making decisions". In particular, they try to model confidence by combining the reputation of an agent, experiences with other agents, and the behavior of an agent. Furthermore, they note that an agent may depend on multiple data sources. Especially in this case, they note the benefit of a model for confidence. Their model of confidence consists of three parts: "Trust", "Certainty", and "Evidence". In addition, they distinguish between "local confidence" and "global confidence". Local confidence refers to a confidence that is calculated by the agent itself, whereas a global confidence is calculated by considering the interactions between agents. In contrast to the paper, this thesis deals with an MAS for natural language. Furthermore, the agents of PARSE do not use mechanisms like trust or evidence so far. In future work it might be interesting to analyze whether some model of trust is applicable to the agents in PARSE.

Like the last paper, the paper of Becker and Corkill [4] also deals with confidence. In contrast to the previous paper, this one treats confidences from a theoretical point of view. In their work, they present analysis model that is able to "measure the sensitivity of a collaborative problem-solving system to potentially incorrect confidence-integration assumptions". This model has been designed as domain-independent analysis model. In a first step, they define their model for agents. They distinguish three types of models: First, a model that represents the sequential execution of the agents. Second, a model that is used to represent MAS that execute agents in parallel. The third type of model combines both models to represent both possibilities. In their models, they represent agents, the world's state and the dependencies between the agents. The model is parameterized by various probabilities: You can specify probabilities for certain events in the agent's world. In addition, you define probabilities for dependencies between certain agents. Furthermore, the correctness agents and the correction/introduction of errors by the agents can parameterized. By using their model, they found out that the incorrect assumption of independence of the agents can lead to significant errors. According to the authors, this effect occurs less strongly with good agents than with "mediocre" ones. In contrast to the work of Becker and Corkill, this thesis is less concerned with the theoretical combination of confidence. Nor can any assumptions of dependence or independence of results be made because the agents considered in this thesis each deal with a separate task. In particular, the agents in the paper had partly the same tasks.

3.2. Uncertainty in Natural Language & Natural Language Processing

In the following, related work that deals with the handling of uncertainty in NLP is introduced. Hypotheses and confidence are needed in natural language processing, since natural language itself contains uncertainty. Therefore, this section also covers work that deals with uncertainty in natural language in particular.

The first paper in this section analyzes the expression of uncertainty in linguistic data [2]. Therefore, Auger and Roy [2] classify uncertainty into two sub types. On the one hand "linguistic ambiguities" and on the other hand "referential ambiguities". Linguistic ambiguities mainly refer to different meaning of words. In contrast, referential ambiguities arises by missing context information. They state that the resolution of linguistic ambiguities are mainly resolvable using linguistic data. But they also find that the problems with referential ambiguities require more contextual models. They also deal with the mapping of uncertainty to probability values. Therefore, they refer to a so-called

"Kent Chart" introduced by S. Kent and mentioned in Brown and Shuford [5]. A Kent Chart combines formulations with a probability range that acts a score for the meant certainty. You have to note that the authors of the paper explain that the scope for interpretation of these mappings is too wide. According to them, this has been shown in several surveys. This paper is relevant for the thesis because it provides insight on how uncertainty in NLP arises. It also gives an estimation of uncertainties that are easier to resolve. This knowledge could help to assess hypotheses. Furthermore, the paper explains that the mapping of a formulation to a score that represents the certainty as numerical value is not obvious. Therefore, one should take care that when classifying hypotheses with respect to their correctness; the options for classification should be kept as clear as possible. In contrast to this thesis, the paper considers language in general and not MAS for natural language.

After having examined the expression of uncertainty in natural language in the last paper, the following paper deals with its modeling [21]. The work of Raskin and Taylor [21] also deals with the "modeling" and "representation" of uncertainty in natural language. Furthermore, they describe a methodology to the major type of uncertainty that is present in a given sentence. They try to build a formal representation of uncertainty. Therefore, they analyze four phenomena that refer to different "uncertainty types": "vagueness", "fuzziness", "possibility", and "probability". To classify a certain sentence, they classify the words of a sentence and associate one of these four types to the words. Like the paper of Auger and Roy [2], this paper deals with uncertainty in natural language in general. They provide a classification mechanism to identify the type of uncertainty of a sentence. In contrast to the work of Raskin and Taylor, this thesis will not take a closer look on the cause of the confidences that are related to some uncertainty.

After considering approaches to uncertainty in natural language in general, the next paper deals with a use case where this uncertainty must be dealt with. This use case is the so-called "Ontology Learning" [9]. According to Haase and Völker [9], "ontology learning aims at generating domain ontologies [. . .] by applying natural language processing and machine learning techniques". They aim at generating a consistent ontology. For this thesis the processes to eliminate and handle uncertainty is relevant as this thesis tries to improve the results of agents by using confidences. They explain that generally there are two ways to model uncertainty: First, the direct integration to the ontology data. This would be achieved by adding confidence values to the data. Second, it would be possible to keep other data on the side instead. For this thesis and the agents of PARSE, this means that two locations must be examined for confidence. One is the data itself (the PARSE Graph) and the other is the internal data of the agents. They generate their ontologies by applying changes that arise during the creation of an ontology consecutively. Thereby, they try to eliminate the inconsistencies in an ontology by removing the changes with the lowest confidence. Their experiments indicate that more logical contradictions arise, the worse the confidences are. In contrast to the work in the paper, this thesis operates on a multi-agent system. In addition, this thesis does not aim to generate ontologies.

Another use case where uncertainty has an important role is part-of-speech tagging. In their paper, Alba, Luque, and Araujo [1] deal with different algorithms for POS Tagging. Especially, they use genetic algorithms to create the tagging. Furthermore, they analyze different types of encoding the problem of tagging. They consider that POS Tagging is an important step for many NLP processes. In addition, they make it clear that POS Tags are associated with uncertainty, as the word type results from the context. As an example, they use the word “can”. This word can be a noun, an auxiliary verb or a transitive verb [1]. They further explain that the search for the correct tags is time-consuming, which is why parallel algorithms were also considered. In their experiments, they study different genetic algorithms, different population sizes and different corpora. Overall, their experiments have shown that genetic algorithms have the potential to improve the results of POS Tagging in some cases. In contrast, in this thesis the search space shall be searched by strategies. In particular, characteristics of hypotheses are to be worked out that help to reduce the search space.

After a use case for genetic algorithms was considered in the previous paper, the next paper deals with evolutionary algorithms in NLP in general. Bungum and Gambäck [6] describe that a lot of tasks are related to NLP and some of them can be approached with evolutionary algorithms. As an example, they mention the summarizing of texts. In particular, they note that genetic algorithms were used in the work of Litvak, Last, and Friedman [15] to explore a search space that was used to generate summaries. They further describe that the implementation of a genetic algorithm worked well in this example. Among other examples for application scenarios of evolutionary algorithms, they conclude that such algorithms work well for parameter optimization. However, they also believe that, depending on the application case, it must be examined whether other state-of-the-art approaches are more appropriate. In particular, they noticed that evolutionary algorithms show no significant improvement over other approaches. This thesis does not use evolutionary or genetic algorithms for the exploration of the search space. Instead, the aim is to investigate whether the search space can be searched specifically by properties of hypotheses.

The last paper of this section deals with an approach similar to that of this thesis, but in a different application area. Sperber et al. [22] aim to use “neural lattice-to-sequence” models to deal with uncertain natural language. A lattice defines several paths for a given input. Each path defines a possible recognition of an original input. Since every word that is represented as node in the lattice, has a confidence, the actual score of a path can be simply evaluated by combining the confidences. Thus, a lattice is comparable to a Hypotheses Graph as used in this thesis. By using these lattices they are able to retrieve different possibilities for a given input explicitly. In their paper, they combine lattices with neural networks. In contrast to the paper, this thesis does not operate on words but on hypotheses of agents in a multi-agent system. Furthermore, the thesis does not use machine learning during exploration or rating of the hypotheses.

3.3. Exploration of large Search Spaces

This last section covers work on exploring large search spaces in the context of NLP. A hypotheses graph defines a search space that grows exponentially in the number of hypotheses. Therefore, approaches to explore large search spaces in NLP are helpful for this thesis.

The first work, which is considered here, deals with a variation of the so-called *Beam Search*. This is a search method that is standard in the field of decoding in machine learning [26, 17]. Vijayakumar et al. [26] are interested in generating sequences from a neural sequence model. According to the authors the beam search is normally used for this purpose. They have noticed that the sequences that are generated by the beam search algorithm are only slightly different. Since the generated sequences are so similar, ambiguity is not well detected according to the authors. For this thesis the approach is relevant because the paper also deals with the exploration of search spaces that potentially become large and are created in the context of NLP. In contrast to this thesis the authors deal with algorithms of machine learning. In particular, they aim to show that their algorithms provides more diverse results and thus potentially generates sequences in a less one-sided way than beam search. One of their examples is the task of "image captioning" where headlines for images are to be found. In comparison to the beam search, they have succeeded in finding different formulations. With regard to the exploration of the search space of hypotheses explored in this thesis, there are also decisions that have to be made. For instance, the exploration strategy has to consider whether to continue the search with small changes or to pursue larger changes in the selection of hypotheses. The authors have shown that in some cases, larger changes are better than small changes to cover as much diversity as possible. In contrast to the approach from the paper, this thesis does not directly use neural models. Therefore, the approach is not directly applicable to this paper.

The next paper also deals with a variation of the beam search. In their paper, Meister, Vieira, and Cotterell [17] deal with a new search algorithm called "Best-First Beam Search". They state that the "decoding for many NLP tasks requires a heuristic algorithm for approximating exact search". According to the authors, this is necessary because the search space is too large to explore it efficiently. As explained above, the state-of-the-art approach for this case is a beam search. The authors of the paper present their algorithm as a replacement for this standard algorithm. In their experiments, they show that their algorithm for decoding neural models is faster than the usual ones. As before, the algorithm cannot be directly transferred to the approach of this thesis, since no neural models are used. However, the work clearly shows that the problem of search spaces in NLP does not disappear even if neural models are used.

The last paper in this section deals with an empirical analysis regarding search algorithms for a concrete problem in NLP. Yoo et al. [32] make an empirical analysis on generating NLP adversarial examples. For each analysis, they consider the algorithm, the search space, and the search budget. They aim to create a reproducible benchmark for different algorithms, search spaces, and budgets. For their work, they consider deterministic and non-deterministic search algorithms. In fact, they use a "beam search", several "greedy" searches, a "genetic algorithm" and a "particle swarm optimization". Through their experiments, the authors were able to show that greedy approaches are good for this application from a runtime perspective. At the same time, other methods such as beam search also yielded better results at longer runtimes. This last paper shows that different algorithms dealing with the exploration of search spaces in NLP are investigated. This thesis deals especially with NLP in the context of MAS. It also deals with the exploration of a search space of hypotheses and not with a specific NLP task.

4. Overview of the Approach — The Agent Analysis Framework

The goal of this thesis is to improve the results of agents that rely on each other using hypotheses. Therefore, this chapter introduces the *Agent Analysis Framework (AAF)*. This framework performs the task of managing agents, their respective hypotheses, as well as their dependencies. In the design of the AAF, particular focus was placed on extensibility. The expansion to include new agents, new methods of exploration, as well as new possibilities of evaluating hypothesis paths has been given special attention. The following sections describe the architecture of the framework and highlight the design decisions that were made.

4.1. Architecture

This section covers the architecture of the Agent Analysis Framework. Different module groups are introduced. Furthermore, the connections between the individual modules are explained. Figure 4.1 provides an overview. The picture shows the four different

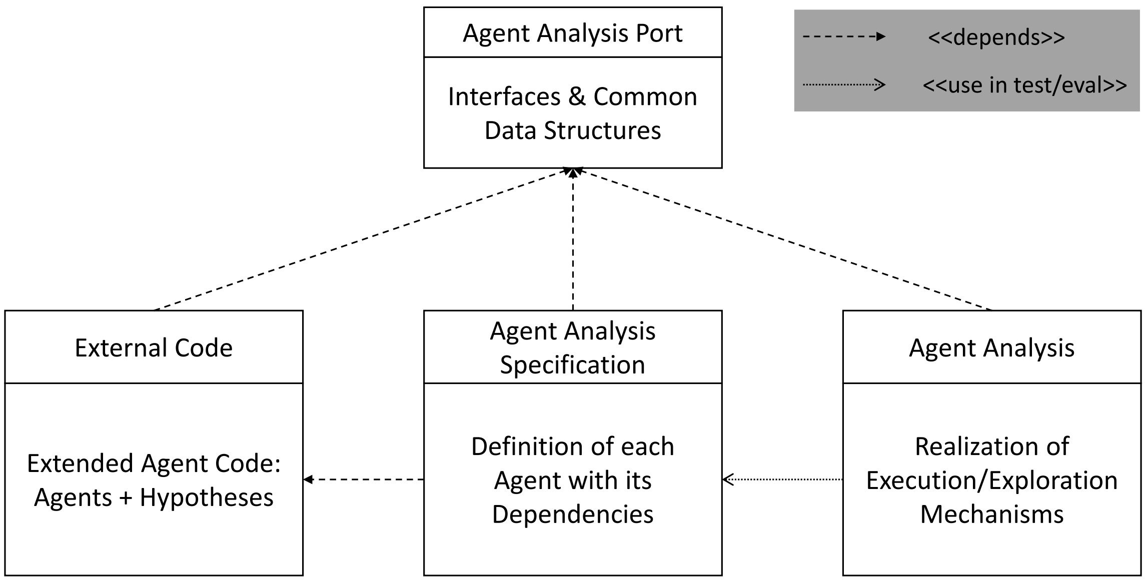

Agent Analysis Port
Interfaces & Common Data Structures
<<depends>>
<<use in test/eval>>
External Code
Extended Agent Code: Agents + Hypotheses
Agent Analysis Specification
Definition of each Agent with its Dependencies
Agent Analysis
Realization of Execution/Exploration Mechanisms

Figure 4.1.: Architecture of the Agent Analysis Framework

module groups of the project. A module group refers to one or more modules that have been grouped together. In the implementation, the modules are represented with Maven projects and modules. The framework is divided into four module groups:

1. Agent Analysis Port (cf. Section 4.1.1)
2. Agent Analysis Specification (cf. Section 4.1.2)
3. External Code (cf. Section 4.1.3)
4. Agent Analysis (cf. Section 4.1.4)

All module groups depend on the interfaces and common data structures in the Agent Analysis Port. In addition, the definitions of the agents uses the external code that contains the actual agents. In order that the Agent Analysis does not have dependencies to all agents, there is no direct dependency to the specifications. Instead, the specifications are only used for unit tests or in evaluation.

4.1.1. Agent Analysis Port

The Agent Analysis Port contains the central interfaces to the Agent Analysis Framework (AAF) to work with agents, hypotheses, and the framework itself.

The most important tasks of the module are the definition of . . .

- interfaces for PARSE and INDIRECT agents (cf. Section 4.1.2)
- information provided and needed for agent execution
- interfaces for hypotheses, and selections of hypotheses
- interfaces for the exploration of hypotheses graphs
- interfaces for rating functions
- basic data containers

The following paragraphs take a closer look at the different definitions.

Information & Agent Execution First, the Agent Analysis Port aims to define an interface for automatic execution of agents. This step is a preliminary work for the later specifications of the agents. The goal is to describe the dependencies between the agents. Therefore, two entities are defined: the `InformationId` and the `AgentSpecification`.

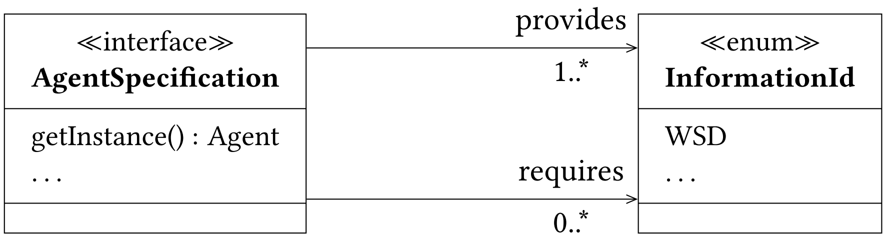


Figure 4.2.: Architecture: InformationId & AgentSpecification

In Figure 4.2 you can see the relation between an information and an agent specification. The idea is to achieve automation in the execution by explicitly defining dependencies. An agent specification encapsulates at least one instance of an agent. Furthermore, a specification has to define the required information of an agent as well as the provided information. An information then denotes an artifact that was written in the PARSE Graph. For example, the disambiguation of the input words (WSD). This structure with explicit dependencies is a first aid for this work. It can be used to avoid making mistakes during the execution of agents that are caused by a wrong execution order. An exact definition of the interfaces is then made in the implementation in Chapter 5.

Hypothesis and Hypotheses Selection The second block of the Agent Analysis Port deals with hypotheses and their selections. The Hypotheses Graph is introduced in Section 2.4. This graph is mainly created by different selections of hypotheses. Therefore, the port will provide structures to define selections of hypotheses from a fixed set of hypotheses.

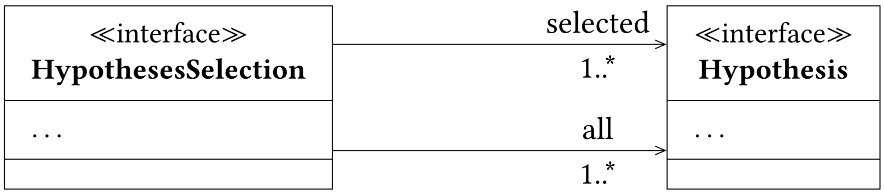


Figure 4.3.: Architecture: Hypothesis & HypothesesSelection

Figure 4.3 shows that the minimum requirement for the selection of hypotheses is a structure that combines the actual selection with the original fixed set of all hypotheses. This fixed set of hypotheses to choose from is called `all`. The chosen hypotheses from this set are called `selected`. The full definition of the relation between hypotheses and selections of hypotheses is in Chapter 6.

Layered Exploration Graph The exploration graph of hypotheses builds layers (cf. Section 2.4). Thus, some basic elements for the exploration can be defined.

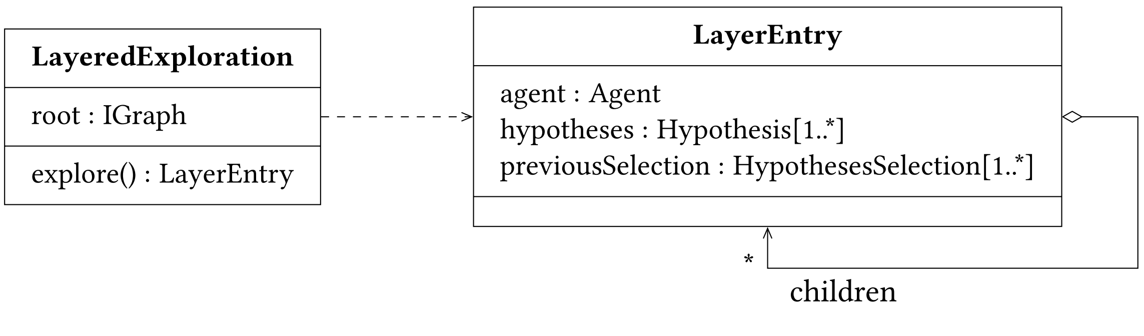


Figure 4.4.: Architecture: Layered Exploration

Figure 4.4 shows the preliminary structure of the exploration mechanism for hypotheses graphs. The exploration itself needs an initial PARSE Graph (`root`). This graph is the starting point for the exploration. The result of the exploration is at least the root entry of the exploration graph. The nodes of the exploration graph are called `LayerEntry`. These entries are coupled to an agent that is responsible for the generation of hypotheses. Furthermore, each step contains the hypotheses that are generated by this agent. As each LayerEntry has a parent (besides the root entry), the entry is a result of `HypothesesSelections` from the previous layer entries. Thus, the entry also encapsulates the selections that lead to this entry. This structure defines the information that is needed to represent the Layered Exploration and the exploration graph itself. The actual realization of this exploration will be discussed in Section 6.2.

Rating Functions The last artifact that needs to be defined in the port of the Agent Analysis Framework are Rating Functions for the exploration paths. As mentioned in Section 2.4, the approach uses the rating functions to determine good paths in the exploration. A *path* in an exploration graph is defined as a list of the corresponding Layer Entries. Thus, a rating function will provide a score for a collection of paths.

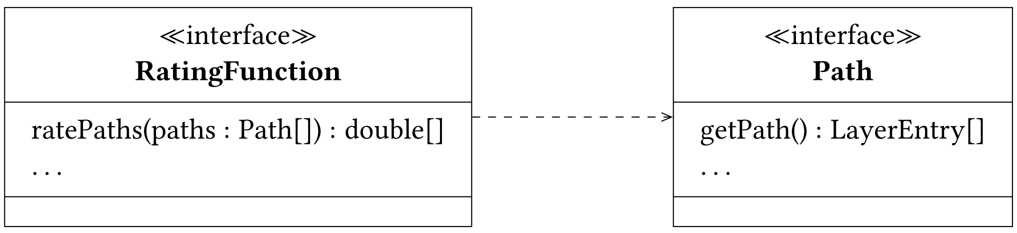


Figure 4.5.: Architecture: Hypothesis & HypothesesSelection

Figure 4.5 summarizes the connection between rating functions and paths. The image shows that a rating function provides multiple scores for multiple paths. The detailed implementation of these is discussed in Chapter 6.

4.1.2. Agent Analysis Specification

The Agent Analysis Specification Module Group consists of five modules (cf. Figure 4.6). These modules specify the available PARSE and INDIRECT agents. This separation into five modules is made so that projects dependent on it do not directly build dependencies to all existing agents.

The first module to mention is the Agent Analysis Specification Platforms (AAS Platforms) Module. It defines the different pre-processing pipelines of PARSE and Indirect. Thus, they provide the code to process textual input and generate PARSE Graphs. Furthermore, it defines super classes for the different types of Agent Specifications. Namely, PARSE agents, INDIRECT agents, and Hypothesis agents. The latter defines the base class for agents that can deal with hypotheses. More information on the different types of Agent Specifications can be found at the end of this section.

In addition, the AAS Module Group contains four further modules. These modules contain different specifications of agents. They build groups of specifications. The AAS PARSE Core Module contains the specification of the core agents of PARSE. The core agents are those agents that are directly referenced by the PARSE project page [28]. Similar

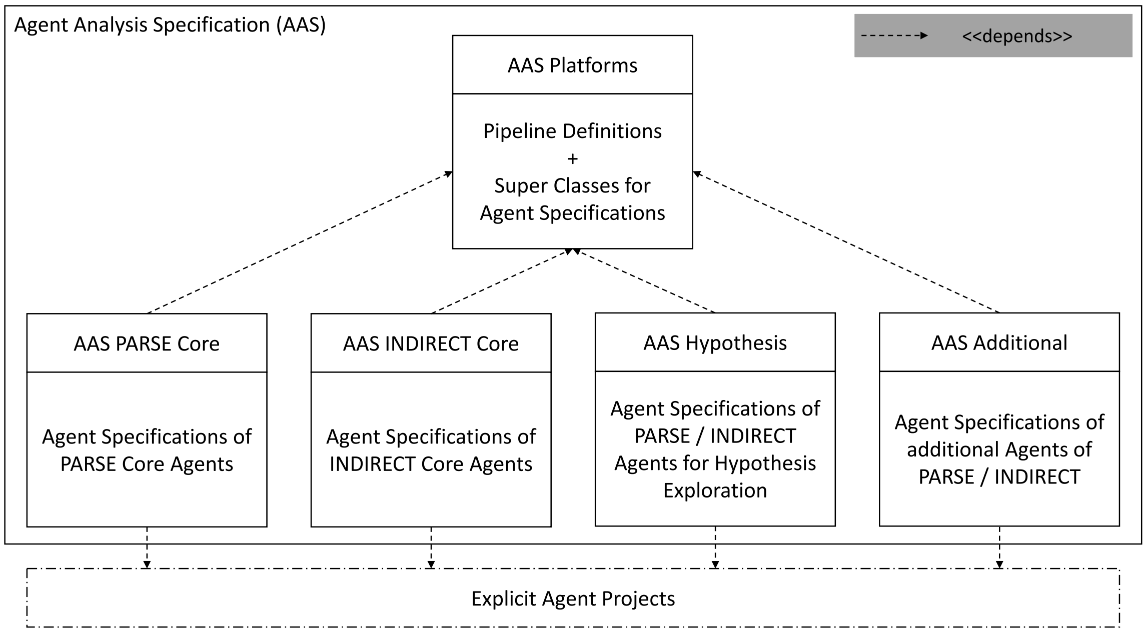

Agent Analysis Specification (AAS)
<<depends>>
AAS Platforms
Pipeline Definitions + Super Classes for Agent Specifications
AAS PARSE Core
Agent Specifications of PARSE Core Agents
AAS INDIRECT Core
Agent Specifications of INDIRECT Core Agents
AAS Hypothesis
Agent Specifications of PARSE / INDIRECT Agents for Hypothesis Exploration
AAS Additional
Agent Specifications of additional Agents of PARSE / INDIRECT
Explicit Agent Projects

Figure 4.6.: Architecture of the Agent Analysis Specification

to that, the AAS INDIRECT Core Module contains the specifications of the agents that are directly referenced by the INDIRECT project page [12]. Each module depends on the AAS Platforms Module as this module contains the base classes for the agents. Furthermore, the modules have explicit dependencies to the different agent projects. Thus, the modules do not need all agent projects but only the projects with the agents they specify.

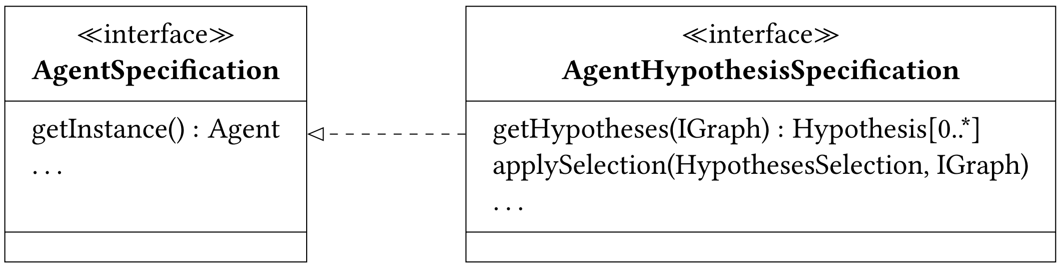

≪interface≫
AgentSpecification
getInstance() : Agent
...
≪interface≫
AgentHypothesisSpecification
getHypotheses(IGraph) : Hypothesis[0..*]
applySelection(HypothesesSelection, IGraph)
...

Figure 4.7.: Architecture: AgentSpecification & AgentHypothesisSpecification

In general, you distinguish two types of agent specification: The Agent Specification and the Agent Hypothesis Specification. The differences are shown in Figure 4.7. The normal Agent Specification is already known from Section 4.1.1. This type of specification is used to make the dependencies between the agents explicit. The Agent Hypothesis Specification extends a normal specification by providing methods to deal with hypotheses. On the one hand, you can extract the hypotheses stored in a PARSE Graph (IGraph) with an instance of the corresponding agent. Thus, the hypotheses for selection are retrieved. On the other hand, these hypotheses are used to build selections that can be applied to a PARSE Graph. In this way, the hypotheses are written in the graph and considered as chosen. The process of selection is fulfilled by so-called Selection Providers and is discussed in Chapter 6.

4.1.3. External Code

The explicit agent projects are already introduced (cf. Figure 4.6). These projects are the actual implementation of the agents. They do not define any dependency to other agents. The external code module group gathers all the agent code that have been modified in this thesis. Thus, the external code module contains the agents that are able to generate hypotheses. The external code modules depend on the Agent Analysis Port as the agents provide the methods for the `AgentHypothesisSpecification` interface. The external code modules are examined in detail in Chapter 6 during the implementation of the actual hypotheses gathering and application.

4.1.4. Agent Analysis

The last module is the actual Agent Analysis. It is responsible for the actual implementation of the execution and exploration of the agents, respectively the hypotheses of the agents. Therefore, the module is divided into the two parts: On the one hand, the *execution of agents*. This part manages the actual invocation of the PARSE agents. On the other hand, the *exploration of hypotheses*. The second part contains the different rating functions, selection providers, and the actual exploration management. The realization of the agent analysis is discussed in Chapter 5, 6, and 7.

4.2. 3-Phases Approach

As already mentioned in the introduction, this approach is divided into three phases. The three phases are covered in Chapter 5, 6, and 7. The overall goal of this thesis is the improvement of the results of existing agents in a multi-agent system for natural language. The underlying algorithms of the agents shall not be changed to achieve that goal. Instead, different combinations of generated hypotheses shall be generated and assessed to find new results for the agents.

Phase 1: Analysis of Agents In a first step, the existing agents shall be analyzed. This analysis provides information about the presence or absence of hypotheses. Furthermore, information about different kinds of hypotheses is gathered. Different types of confidences are particularly important. These types could be probabilities, real numbers, or ordinal scaled values. In addition, the mechanics to store results or hypotheses are investigated. Since the data must primarily be stored in PARSE Graph, the question arises as to how exactly hypotheses are stored or whether they are stored at all.

After examining and classifying the individual agents with respect to their suitability for the approach, the second step is to investigate how the agents are related. This is of particular interest for the approach, since the effects of different selections of hypotheses are investigated. Thus, groups of agents have to be found that depend on each other. Otherwise, an exploration of hypotheses are not realizable. Furthermore, all agents of such groups should be able to generate hypotheses. This ensures that selections can also be made in each layer (respectively for each agent) and thus a search space can be spanned.

Phase 1 consists of the following tasks:

Phase 1 — Tasks

T1.1: Analysis of existing agents regarding the use and generation of hypotheses instead of simple results without confidences

T1.2: Analysis of the generated hypotheses for similarities and differences

T1.3: Definition of all specifications of existing agents (dependencies between agents)

T1.4: Implementation of automated execution of agents according to their specifications

Phase 2: Exploration of Hypotheses In phase 1, the current state of the agents has been analyzed. The presence or absence of hypotheses has been studied. Furthermore, the agents and dependencies of the agents are specified. Thus, the automatic invocation of agents is operational. In phase 2 of this thesis, the actual Agent Analysis Framework is built. First, a model for hypotheses is created. In addition, a first exploration strategy — the *Layered Exploration* — is going to be implemented. To accomplish this, selection providers are implemented. Selection Providers control the growth of the exploration graph.

Besides these activities, hypotheses are also needed for exploration. Therefore, the handling of hypotheses for some agents is implemented during this phase. Thus, the external code module group is extended by chosen agents. These agents are extended by mechanisms to read and apply selected hypotheses to the PARSE Graph. To finish this phase, a GUI to show the results of an exploration, including the exploration graph, is created. Since the tool can be used to visualize the actual exploration graph, the tool can be used to get insights into explorations.

Phase 2 considers the following tasks:

Phase 2 — Tasks

T2.1: Creation of a model for Hypotheses

T2.2: Realization of the Layered Exploration

T2.3: Realization of Selection Providers

T2.4: Extension of chosen, existing agents by adding handlers for hypotheses

T2.5: Providing a GUI to visualize the exploration

Phase 3: Finding Good Rating Functions In the last phase of the approach, the rating functions (respectively the assessment mechanisms) are created. Rating functions provide an assessment for Hypotheses Paths generated by a Hypotheses Exploration. To rate the performance of the rating functions, the definition of what “good rating function” means has to be set. In order to measure the quality of rating functions, a classification of hypotheses is necessary, since the metrics for the measurement need them. For this purpose, a tool is to be created that enables a participant of a user study to classify hypotheses.

To sum up, the tasks of this phase are:

Phase 3 — Tasks

T3.1: Definition of “Good Rating Functions”

T3.2: Realization of Rating Functions

T3.3: Creation of a tool for classification of Hypotheses

5. Analysis of Agents

This chapter deals with the analysis of existing agents of PARSE and INDIRECT. The existing agents of PARSE and INDIRECT are inspected with respect to the use and generation of hypotheses. In particular, the identification of similarities and differences of these hypotheses is focused. This allows us to get a first insight into the present occurrence of hypotheses in natural language agents. In addition to this analysis of the hypotheses, the dependencies between agents are investigated.

This chapter is structured as follows: First, the agents of PARSE and INDIRECT are introduced. In this introduction, each agent will be analyzed according to its purpose and the kind of information provided by the agent. At the end of the introduction of PARSE agents or INDIRECT agents, the dependencies between the agents are summarized. After introducing the agents of PARSE and INDIRECT, the found information on hypotheses used or generated by the analyzed agents is gathered. In a last section, the gained information is used to realize the first parts of the Agent Analysis Framework. These parts include the specification of agents and the automatic execution mechanisms for their agent specifications.

5.1. PARSE Agents

In the following different PARSE [31] agents are analyzed. First, the core agents are examined. These agents are those that are directly referenced in the project page [28]. They represent the agents that are taken over into the PARSE project. Furthermore, additional PARSE agents are analyzed. These include the agents that have not been included in the project but are executable. All agents are listed in Table 5.1 together with a short description. As mentioned above, this table separates core agents and further agents.

5.1.1. Core Agents

There are the core agents in the PARSE project main page [28] that are listed and described briefly in the following.

Action Recognizer The Action Recognizer agent built by Ou [19] aims at the detection of actions in spoken natural language. In the work of Ou, an action is defined by the actor, the predicate, and the parameters of the action. Therefore, they uses POS Tags, Chunking, and an instruction number that is generated by the Shallow NLP (SNLP) during the pre-processing pipeline of PARSE. Furthermore, the agent uses semantic roles to extract the final actions. After finding the actions of an input, the agent modifies the PARSE Graph and stores the information in it. The agent does not pass hypotheses into the PARSE Graph.

Name	Description
Action Recognizer [19]	Detection of actions in a text
Concurrent Action [29]	Detection of concurrent actions in a text
Condition Detection [23, 27]	Detection of conditions in a text
Loop Detection [29]	Detection of loop control structures
Word-Sense Disambiguation	Disambiguation of words based on Babelfy
Context [30]	Generation of a context model from spoken utterances
CoRef [13]	Detection of co-references
Wiki WSD [14]	Disambiguation of words based on Wikipedia
Topic Detection [14]	Detection of Topics of an input
Ontology Selection [14]	Selection of Ontologies based on Topics
Method Synthesizer [24]	Synthesis of Method Definitions

Table 5.1.: An overview on the PARSE Agents.

No hypotheses are generated internally either. The obtained information is used by the agent to define a list of actions directly. The actions also have no assigned confidence levels. Thus, the agent does not represent a use case for this work.

Concurrent Action Based on the result of the Action Recognizer [19], the Concurrency Agent by Weigelt, Hey, and Steurer [29] determines that actions are concurrently executed. Therefore, the agent creates new nodes in the PARSE Graph for each concurrent action. A concurrent action encapsulates a set of actions that are performed concurrently. The agent optionally uses information on co-references provided by the CoRef agent (cf. CoRef). Like the actions determined by the action recognizer, the concurrent actions have no confidence or score. Therefore, the results of the concurrency agent are not usable to examine a search space of hypotheses.

Condition Detection Like the Concurrency agent, the Condition Detection agent [23, 27] is based on the results of the Action Recognizer [19]. The agent aims to detect conditional sentences in natural language text. Therefore, various heuristics are used [23]. The condition detector stores information on the *if statement*, the *then statement*, and the *else statement* into the PARSE Graph. Among these, the *else statement* is an optional part. Together, these two (or three) parts form a so-called "Condition Container" [23]. Similar to the concurrency agent, this agent can also use the information on co-references provided by the CoRef Agent. Like the actions and the concurrent actions, the conditions have no confidence or score. Therefore, the results of the Condition Detection Agent are not usable to examine a search space of hypotheses.

Loop Detection The work of [29] covers the detection of program loops in natural language text. Therefore, the agent is based on the results of the Action Recognizer [19]. The agent is able to detect different types of loops, such as *for loops*, *while loops* or *do until loops*.

This kind of loops are typical structures known from programming. The loop detection stores information on the loops directly into the PARSE Graph. Like the concurrency agent and the condition detection agent, this agent can also use the information on co-references provided by the CoRef Agent. This agent is not usable to examine a search space of hypotheses, as the agent neither uses confidences or scores internally, nor provide any confidences.

Word-Sense Disambiguation The default Word-Sense Disambiguation Agent of PARSE uses Babelfy [18] to disambiguate words. Therefore, the agent adds attributes to the nodes of the PARSE Graph. The agent stores the best sense of a word directly as an attribute. Furthermore, a score for this sense is stored. Apart from this information, the agent stores other senses together with a score as attribute of the actual PARSE Graph node. Thus, this agent already uses hypotheses in a basic manner. The agent selects the most probable sense from a set of possible senses, but also stores possible further senses of a word (together with a score) for subsequent agents.

Context The work of Weigelt, Hey, and Tichy [30] aims at the generation a context model from spoken utterances. Therefore, information like *concepts*, *actions*, *entities*, and *states* are added to the PARSE Graph. This allows other agents to capture the more precise relationships within a sentence. The Context Analyzer Agent uses some confidences for different relations between concepts, such as entities or actions. These confidences are used to find the most likely instance of such relations. The agent stores the information together with the confidence data in the graph, so agents that build on it can access the confidence data. The application of the hypothesis approach would therefore be conceivable, since the agent already uses confidences.

CoRef The CoRef Agent by Hey [13] aims to detect co-references in an input text. For this purpose, the agent uses the information stored in the context model provided by the Context Analyzer Agent. The references found by the agent are stored as arcs between nodes generated by the context analyzer. Internally, the agent generates candidates for possible relations. This set of candidates is then reduced based on various properties until it is stored in the graph structure. As with the context analyzer, confidences are also used here. In this case, the arcs that represent the relations have the confidences as an attribute. Thus, the application of the hypothesis approach would be conceivable, since the agent already uses confidences.

5.1.2. Further Agents

This subsection covers the agents that are not directly mentioned in the PARSE project main page [28]. The agents form the second part in the overview in Table 5.1. All mentioned agents are at least runnable on the current version of PARSE.

Wiki WSD The Wikipedia Word-Sense Disambiguation (Wiki WSD) Agent by Keim [14] provides a disambiguation of words based on Wikipedia articles. Similar to the default

Word-Sense Disambiguation Agent of PARSE, this agent stores the information on the actual meaning of a word into an attribute of the respective PARSE node. Internally, the agent uses logarithmic scores for each possible meaning of a word. After the agent has decided which sense is correct, the agent only stores the result without the confidence into the attribute. Special attention is paid to the fact that this is a logarithmic score, because you cannot simply compare such a score with a probability. For the further progress of this thesis it is important to know that there are various ranges of confidences. As the agent uses scores and generates multiple possible hypotheses internally, this agent is suitable for the hypothesis approach of this thesis.

Topic Detection Based on the Wikipedia Word-Sense Disambiguation (Wiki WSD) Agent, the Topic Detection Agent by Keim [14] provides possible topics of an input text. Therefore, the agent uses concepts from Wikipedia as topics of a text. The agent stores the information on the Top-8 topics into the PARSE Graph. This amount of the topics defaults to eight, but is configurable. Each topic contains a confidence and its name. The Topic Detection Agent already generates hypotheses including scores and stores them directly as hypotheses into the graph structure. Thus, the agent is compatible to the hypotheses approach of this thesis.

Ontology Selection The last agent by Keim [14] provides a selection of suitable ontologies for an input text. Keim distinguishes "Actor Ontologies" and "Environmental Ontologies". Based on the Topic Detection Agent, this agent chooses ontologies that represent the actor mentioned in a text, as well as ontologies that represent the environment, the text is about. Internally, the agent uses scores for all possible ontologies and selects the best ones according to internal heuristics and strategies. Similar to the Topic Detection Agent, this agent stores the information on the selected ontologies into a new type of node for the whole input graph. Since the agent uses scores, this agent is also suitable for the hypothesis approach of this thesis.

Method Synthesizer The Method Synthesizer by Steurer [24] aims to generate methods for a target system. This agent interprets statements in an input speech that aim to extend an existing system by new methods. Therefore, the agent has to identify *actions*, *sequences of actions*, as well as *instructions*. To identify these entities, the agent needs data about semantic roles, co-references and context information from the *Context* agent. The agent generates candidates for method calls. These candidates are stored as nodes into the PARSE Graph. Each candidate has a score that is used as confidence. In addition, candidates for possible function parameters are stored to the graph. The method and parameter candidates are connected by arcs. Since the agent stores candidates for the method calls and parameters, the agent is also suitable for the hypothesis approach of this thesis.

5.1.3. Dependencies in PARSE

The previous sections provided information about the existence and absence of hypotheses in the agents itself. In the following the mandatory dependencies between the agents are examined. Figure 5.1 shows these dependencies. Every agent is dependent on the

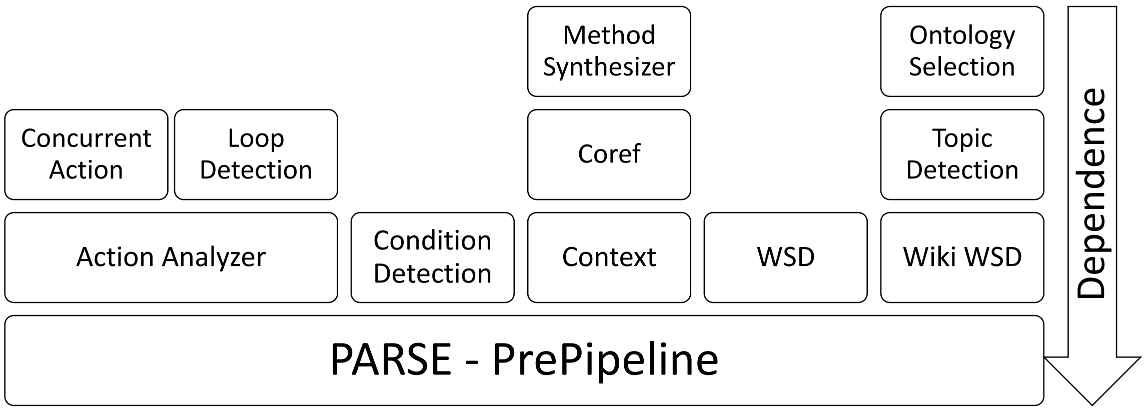


Figure 5.1.: Dependencies between PARSE agents

pre-processing pipeline execution of PARSE. Each agent that is located above another agent depends on the output of the agent below. Therefore, you can see three important dependency hierarchies that do not only consist of the pre-processing pipeline and one agent. First, the concurrent action agent and the loop detection agent depend on the action analyzer agent. This hierarchy is not usable for the approach of this thesis, as none of the agents generates or uses any hypothesis. The second hierarchy starts with the context agent, followed by the CoRef agent and the method synthesizer agent. These agents are suitable for the hypothesis approach of this thesis and could be considered later. The last hierarchy of agents to consider consists of the Wiki WSD, the Topic Detection, and the Ontology Selection agent. These three agents can also be used in the hypothesis approach of this thesis, as all agents can generate hypotheses. Furthermore, the Topic Detection agent already generates some kind hypotheses. In summary, you can see that no cyclic dependencies occurred. But in general one could build the hypothesis graph also with the presence of cyclic dependencies between agents. In this case a fixed number of runs through the cycle should be defined. By this an agent would appear in several layers. This way, especially the development of hypotheses could be observed. Also new variants of selectors and rating functions would be possible (cf. Chapter 6, 7).

5.2. INDIRECT Agents

After analyzing the PARSE agents in the previous section, this section covers the agents of INDIRECT [11] that were considered for this thesis. These agents are either directly referenced in the INDIRECT project [12] or agents that could be interesting for this work. The agents to consider are listed in Table 5.1. Like in the preceding section this table shows the agents together with a short description and separated into core agents and additional agents. This section is structured in the following way: First, the agents are analyzed in detail. After that, the dependencies between the agents are studied.

Name	Description
Dependency Parser	Port of Stanford Core NLP's Dependency Parser
Entity Recognizer	Detection of entities
Conceptualizer	Combination of actions and/or entities to concepts
Text NER	Port of Stanford Core NLP's NER
Const Parser	Port of Stanford Core NLP's Constituency Parser
Synset Mapping [10]	Links between Models and Text based on Synsets

Table 5.2.: An overview on the INDIRECT agents

5.2.1. Agents

The next paragraphs cover the agents of INDIRECT that were considered for this thesis (cf. Table 5.2). Possible sources of hypotheses as well as the structures used to store information are analyzed.

Dependency Parser The Dependency Parser agent ports the dependency information from Stanford Core NLP [16] to INDIRECT. Therefore, the agent simply invokes the Stanford Core NLP's dependency parser and stores the information to the PARSE Graph. Dependencies in Stanford Core NLP are characterized by the start word and end word as well as the type/name of the dependency. The agent transforms that information into an arc between the corresponding words in the PARSE Graph. The actual information like the name are stored as attributes into the arc. No score or confidence is provided by the detected dependencies itself.

Entity Recognizer Based on the work of Hey [13] and Weigelt, Hey, and Tichy [30] the entity recognizer detects mentioned entities in an input text. Therefore, the agent uses the context model as defined in [13]. This context model does also include actions. The different elements in the context model, namely entities, actions, and their sub types are important for the next agent to mention. The entity recognizer uses Wordnet [8] to detect the entities. Furthermore, it uses the dependencies detected by the dependency parser agent. The entities are stored as new type of node to the graph. The node contains different attributes, as the name of the entity. In addition, the node is connected by an arc to its original nodes. This detection of entities does not use any confidence, but on a second level the entities can be connected to other entities, states, or concepts. These connections are parameterized by a confidence. The next paragraph covers more information on these connections.

Conceptualizer Like the entity recognition of INDIRECT, the Conceptualizer agent is also based on the work of Weigelt, Hey, and Tichy [13, 30]. The agent has the task to combine actions, entities, and states to concepts. Furthermore, the agent can combine concepts to super concepts. Therefore, the agent uses the same context model as the entity recognition. Thus, a concept itself does not use scores or confidences. The candidates used during connecting concepts use confidences in relations and word similarities. Hence, one can generate multiple hypotheses for different relations or connection candidates. Currently, the agent uses the best candidates to connect different concepts or entities. Since candidates already have ratings, this agent is applicable for this approach.

Text Named Entity Recognition (NER) The Text Named Entity Recognition agent uses the NER information from Stanford Core NLP [16] and ports them to the PARSE Graph. Therefore, the agent invokes the toolkit to annotate the input text. After that, the information on named entities are stored to the utterance nodes (`TokenNode`) as attribute. The NER agent does not use any confidence or score. Thus, the agent is not suitable for this approach.

Constituency Parser The Constituency Parser agent ports the constituency information from Stanford Core NLP [16] to INDIRECT. Therefore, the agent invokes Stanford's constituency parser and uses the annotations to rebuild the constituency tree in the PARSE Graph. The tree itself is represented by a new node type and a new arc type. The information like labels or tree level are stored directly as attributes of the new node type. As the agent does not provide any confidences, it is not suitable for hypotheses exploration.

Synset Mapping (WSD Link Agent) The last agent of INDIRECT to mention is the Synset Mapping agent or WSD Link agent. The agent has been developed by Heine [10] and aims the connection of architectural models and textual documentation. To build links between model elements and elements mentioned in the documentation, the agent uses Babelfy [18]. Each detected model element will be added as node to the PARSE Graph. To connect model elements and textual elements, the linked elements are connected via an arc. Currently, the agent does not use confidences or scores for its results, but it uses word distance metrics to compare identifiers for model elements. It may be possible to use these distances to generate hypotheses for the mappings later.

5.2.2. Dependencies in INDIRECT

After the different agents of INDIRECT have been considered in detail, a closer look is taken to the dependencies between the agents. Figure 5.2 shows these dependencies. Every agent is dependent on the pre-processing pipeline execution of INDIRECT. Each agent that is located above another agent depends on the output of the lower agent. In contrast to the dependencies in PARSE you can see only one interesting dependency hierarchy. This hierarchy consists of the dependency parser, the entity recognizer, and the conceptualizer. As analyzed above, the only two agents in this hierarchy that uses some scores for internal calculations are the entity recognizer and the conceptualizer. Since the integration of hypotheses is rather complex, the next sections will mostly refer to other hierarchies.

5.3. Hypotheses in PARSE and INDIRECT

In the previous sections you have gained information about the agents of PARSE and INDIRECT. It has been shown that there are dependencies between the different agents. It

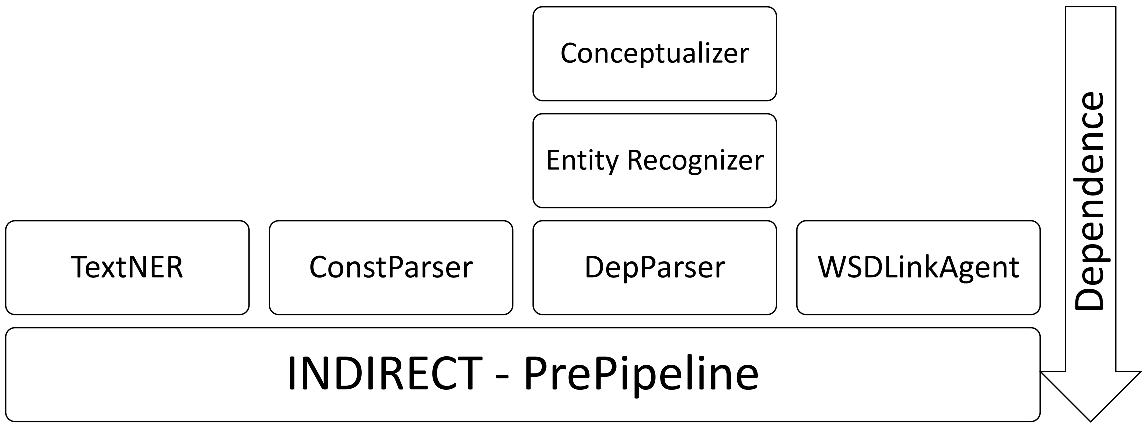


Figure 5.2.: Dependencies between INDIRECT Agents

became clear that only few agents actually use hypotheses. In most cases results are stored directly. Nevertheless, there are also some agents that already use hypotheses and can therefore potentially be used for the approach of this thesis. In the following, the current findings on hypotheses are discussed.

Range of Hypotheses The first result of the analysis of the agents of PARSE and INDIRECT is the so-called *Hypothesis Range*. There are two of them among the agents. On the one hand, there are the agents that generate results or hypotheses for single words or nodes. One of these is the Word-Sense Disambiguation. It generates possible means per word. Ont the other hand, results or hypotheses may refer to the entire input. This includes topic extraction, as an example. Because of the different ranges of influence it makes sense to record this property and to choose the selection procedure for hypotheses depending on it. A more detailed description of the procedure is carried out in Chapter 6.

Confidences of Hypotheses The second observation that emerged during the analysis of the agents is the different types of confidences or scores that were used. On the one hand, probabilities have occurred. These have a co-domain or value range of $[0, 1]$ and would allow the use of probabilistic methods. These have been noticed, for example, during topic extraction, or during the evaluation of words by Babelfy [18]. On the other hand, there are for example logarithmic scales of values too. These are used in the Wiki WSD agent. The realization that scores are not only to be equated with probabilities has an impact on evaluation procedures, because it is clear that one cannot necessarily use probabilistic methods directly but normalization must take place. This task is discussed in more detail in Chapter 7.

5.4. Conclusions for the Agent Analysis Framework

After the analysis of agents in the previous sections, the first two tasks of the first phase are achieved (cf. Section 4.2). The agents are analyzed regarding the use and generation of hypotheses (Task T1.1). Furthermore, the concepts of the range of a hypothesis and the different types of confidences of hypotheses have been found (Task T1.2). The two remaining tasks focus on the actual implementation. These tasks are discussed in detail in the following sections. In a first step, the definition of all specification of existing agents and their dependencies is realized in Section 5.4.1 (Task T1.3). After that, the automated execution of agents according to their specifications are finished in Section 5.4.2 (Task T1.4). This concludes the first phase of this work and in the following chapters the hypotheses of some agents will be utilized to evaluate the approach.

5.4.1. Agent Specification

In the following, a core feature of this thesis is discussed, the so-called *Agent Specifications*. The purpose of these specifications has already been introduced in Section 4.1.1. In general, the specification of a PARSE agent or INDIRECT agent accomplishes the task of defining the dependencies between the agents explicitly. The necessity for this is that without explicit tracking of the necessary dependencies and with manual execution of the agents, errors can easily be made.

With the knowledge from the analysis of the agents, the definition of an agent specification from Section 4.1.1 is refined. The final definition of a specification of an agent (`IAgentSpecification`) is shown in Figure 5.3. A specification is defined for a specific type

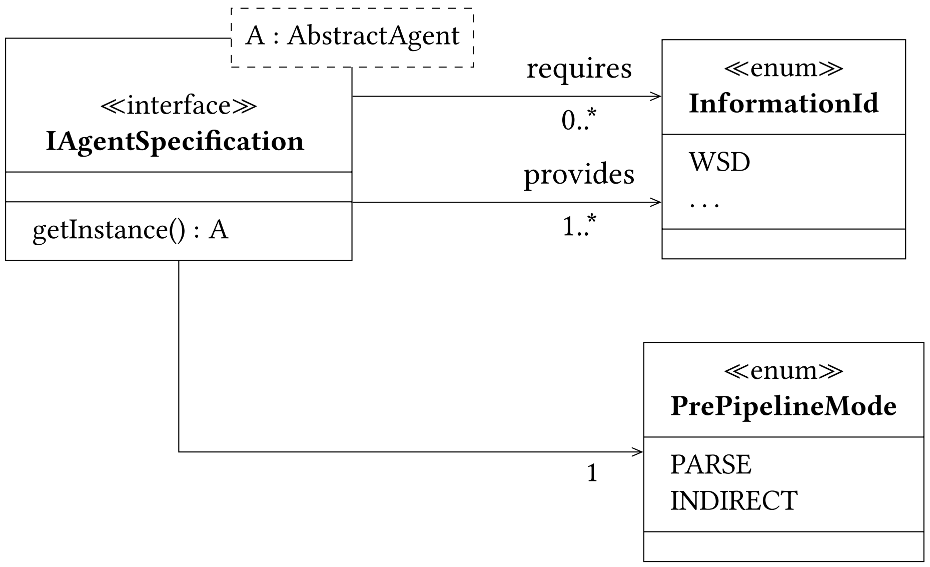


Figure 5.3.: Implementation: InformationId & IAgentSpecification (Agent Analysis Port)

of agent. PARSE and INDIRECT agents inherit from the class `AbstractAgent`. Therefore, the specification is parameterized by a generic type `A`, that is restricted to sub types of an abstract agent. As defined in the architectural chapter, the dependencies between the agents are characterized by so-called information identifiers (`InformationIds`). Each specification provides methods to get the required and provided information of an agent. If the agent only depends on the information provided by the pre-processing pipeline, the list of required information is empty. In contrast to that, the provided information identifiers may not be empty.

An information identifier contains no information about the actual type of information provided by the agent. Therefore, the information identifier is an enumeration of constants. No further methods are provided by the information identifier class (`InformationId`). In order to add new information, only the affected specifications and the constants in `InformationId` have to be extended. This ensures that changes in the hierarchy or agent dependencies can be handled mostly locally.

The agent specification (`IAgentSpecification`) also needs to provide access to the actual agent. The method to get the instance does not create a new instance but uses the configured instance of this specification.

The last information provided by the specification is the so-called `PrePipelineMode`. The `PrePipelineMode` defines different modes to generate a PARSE Graph. Currently two modes are implemented: *PARSE* and *INDIRECT*. Both modes refer to their corresponding default pre-processing pipeline configurations. These two are different since one is designed for spoken and the other for written language.

Pre-processing pipelines are represented by the `IPrePipeline` interface. They are part of the Agent Analysis Specification Platforms (AAS Platforms) module. The structure of pre-processing pipelines is shown in Figure 5.4. A pre-processing pipeline is characterized

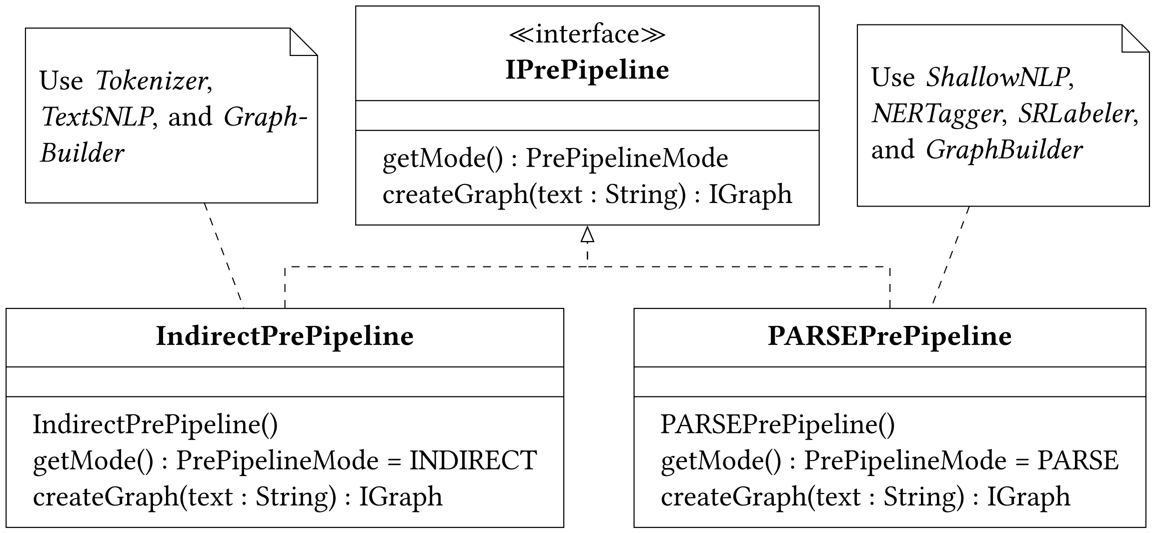

Use *Tokenizer*, *TextSNLP*, and *Graph-Builder*
≪interface≫
IPrePipeline
getMode() : PrePipelineMode
createGraph(text : String) : IGraph
Use *ShallowNLP*, *NERTagger*, *SRLabeler*, and *GraphBuilder*
IndirectPrePipeline
IndirectPrePipeline()
getMode() : PrePipelineMode = INDIRECT
createGraph(text : String) : IGraph
PARSEPrePipeline
PARSEPrePipeline()
getMode() : PrePipelineMode = PARSE
createGraph(text : String) : IGraph

Figure 5.4.: Implementation: AAS Platforms — Pipelines

by two methods. First, a getter method for the actual `PrePipelineMode`. This mode is used to compare a chosen pre-processing pipeline with the pipeline needed by a specification of an agent. Second, a creator method of a PARSE Graph. Since a PARSE Graph is represented as `IGraph`, the method uses an input string to generate an `IGraph`. The kind of the input text does not matter, because all unusable characters are removed. In case of PARSE these would be punctuation marks. Furthermore, as you can see, for both modes an implementation of the pre-processing pipeline was provided. These differ in the way PARSE Graphs are created, as already mentioned. The `IndirectPrePipeline` that represents the default pipeline for INDIRECT agents uses a `Tokenizer`, `TextSNLP` and in a last step the `GraphBuilder` to generate a suitable PARSE Graph. A tokenizer splits a certain text into tokens. After the input text is split into tokens, the TextSNLP (ShallowNLP optimized for textual input) performs basic tasks of NLP. Finally, the PARSE Graph is generated. In contrast to that, the `PARSEPrePipeline` uses `ShallowNLP`, a `NERTagger`, a `SRLabeler` and in the last step the `GraphBuilder` to generate the PARSE Graph. Thus, the default pipeline for PARSE also annotates named entities and semantic roles.

Now that the refined concept of specifications and pre-processing pipelines has been introduced, the actual structure of an agent specification has to be considered. Therefore, the class structure as shown in Figure 5.5 has been created. An abstract super class is

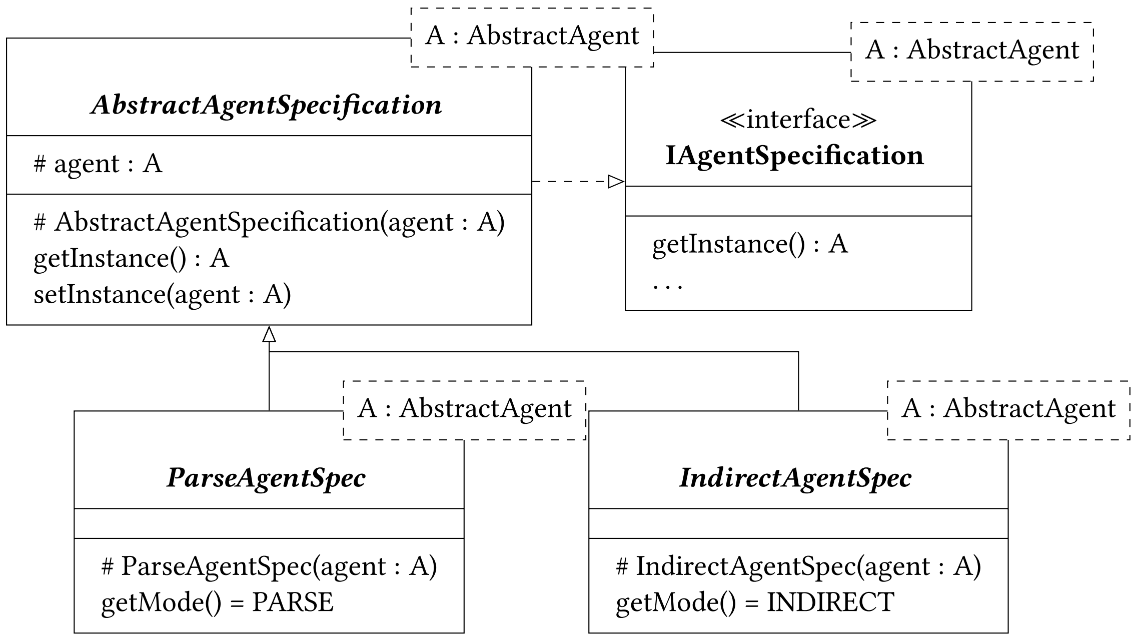


Figure 5.5.: Implementation: AAS Platforms — Agent Specifications

created for all specific actions, this super class first of all contains the actual instance of the agent. This super class `AbstractAgentSpecification` already implements the interface for agent specifications. As abstract sub types of the general agent specification, two abstract classes are defined — one for each `PrePipelineMode`. Thus, the creation of agent specifications for existing agents is easy to realize. Consider one of the core agents of PARSE — the Loop Detection agent. The specification for this agent has to be found in the *Agent Analysis Specification PARSE Core* module, as the Loop Detection agent is a core agent of PARSE. For the realization of the specification, the module needs a dependency on the implementation of the agent. Furthermore, the specification is completed by creating the specification class. This class is shown in Figure 5.5. The definition of the specification itself is simple. All that is needed is to inherit from the corresponding specification super class and instantiate the generic parameter with the corresponding agent. Afterwards, a new instance of the agent has to be created in the constructor and the required and provided information must be specified. In case of the Loop Detection agent, *Actions* are required and *Loops* are provided.

```
class LoopDetectionAgentSpec extends ParseAgentSpec<LoopDetectionAgent>
{
    public LoopDetectionAgentSpec() {
        super(new LoopDetectionAgent());
    }

    @Override
    public List<InformationId> getProvideIds() {
        return List.of(InformationId.LOOP);
    }

    @Override
    public List<InformationId> getRequiresIds() {
        return List.of(InformationId.ACTIONS);
    }
}
```

Figure 5.6.: Loop Detection Agent Specification (Code)

5.4.2. Agent Execution

The final task in this phase is the automatic execution of agents (Task T1.4). In order to accomplish this, the agent execution interface (`IAgentExecution`) has been defined. The structure for the automatic agent execution is shown in Figure 5.7. Since some

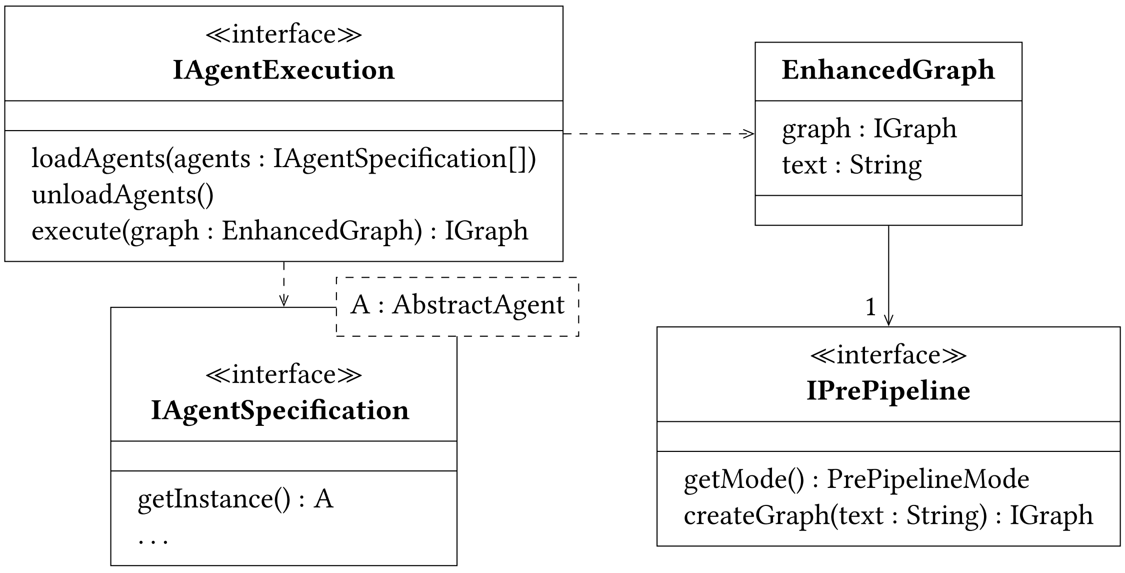


Figure 5.7.: Implementation: AgentExecution

preconditions for the execution must be checked, the agent execution does not work on normal PARSE Graphs (`IGraph`). Instead, the execution uses enhanced graphs that combines the actual textual input, the PARSE Graph, and the definition of the pre-processing pipeline. Thus, it can be verified whether the agents are suitable for execution on a graph generated by a certain type of pre-processing pipeline. The remaining functionality of the design is kept simple. It is possible to load and unload the different specifications of agents to be executed. The actual execution creates a normal new PARSE Graph that is created by executing the agents in a correct order. Here it is important that it is a new instance of the graph. The decision to use new instances is based on the fact that new graphs have to be created later during exploration. This ensures that the different explored variants do not influence each other in the common data structure.

The detailed process of sequential execution of agents is described in Algorithm 1. For the execution, an enhanced graph (`input`) and a set of specifications (`agents`) are required. In a first step, the execution seeks for invalid agents in the agent hierarchy. Agents can

Algorithm 1 Agent Execution

```
Require: EnhancedGraph $input$, Set of IAgentSpecification $agents$
 1: function EXECUTE($input$)
 2:     $graph \leftarrow$ input.getGraph()
 3:     $ppm \leftarrow$ input.getPrePipelineMode()
 4:     $invalid \leftarrow$ FINDINVALIDAGENTSINHIERARCHY($agents, ppm$)
 5:     if $invalid \neq []$ then
 6:         return $null$                // Fail iff missing information in agent hierarchy
 7:     end if
 8:     $orderedAgents \leftarrow$ FINDORDER($agents$)
 9:     for ordered $next \in orderedAgents$ do
10:         $nextGraph \leftarrow$ EXECUTEAGENT($next.getInstance(), graph$)
11:         $graph \leftarrow nextGraph$     // Replace current working instance
12:     end for
13:     return $graph$
14: end function
```

be invalid for two reasons. Firstly, the pre-processing pipeline used may be incompatible with that of the graph. In this case the agent cannot be used because the agent expects other information in the graph. The second reason for finding an invalid agent is that the required information of the agent is not generated by any other agent. In this case the agent cannot be executed either. If any invalid agent has been found, the execution aborts.

Otherwise, the execution first has to find a valid execution order of agents. A valid order is characterized by the fact that the necessary information of an agent is available through the already executed agents. Such a definition is possible because no cyclic dependencies occurred during the analysis of the agents. The execution of an agent leads to a new instance of the PARSE Graph. After execution, the current version of the graph is exchanged so that the next agent works on the modified graph. Finally, after all agents have been executed, the graph created in this way is returned as the result.

With this implementation, the last of the tasks (T1.4) in phase one has been achieved. The existing agents are analyzed regarding the use and generation of hypotheses (T1.1). The analysis of the generated hypotheses for similarities and differences within the hypotheses is finished (T1.2). Furthermore, the specifications of the existing agents are defined (T1.3) and they empower an automatic execution (T1.4).

6. Exploration of Hypotheses

In the previous chapter, the analysis of agents has been finished. Based on the specifications, this chapter deals with the actual exploration of hypotheses, according to the tasks set in Section 4.2.

First, a model for hypotheses, fixed sets of hypotheses and selections of hypotheses (T2.1) is created in Section 6.1. After that, the actual exploration of paths in the Hypotheses Graph (cf. Section 2.4) is discussed in Section 6.2. This so-called layered exploration represents the completion of the second task (T2.2). This exploration is mainly controlled by so-called *Selection Providers* (T2.3) that are discussed in Section 6.3. Furthermore, the implementation of the hypotheses of some agents is carried out. Due to the implementation of handlers for hypotheses of some agents, the fourth task (T2.4) is completed. In a last step, the *Agent Analysis Explorer*, a tool to visualize an exploration of an input text (T2.5), is introduced.

6.1. Hypotheses, Hypotheses Sets, and Hypotheses Selections

The first step to deal with hypotheses and to explore the search space created by them, is the creation of a suitable model of hypotheses. In the previous phase, more detailed information about the properties and occurrence of hypotheses in MAS for natural language has been collected. Thus, the basic concepts from Section 2.3 and Section 4.1.1 can be finalized. The actual model for hypotheses is shown in Figure 6.1. A hypothesis is defined as combination

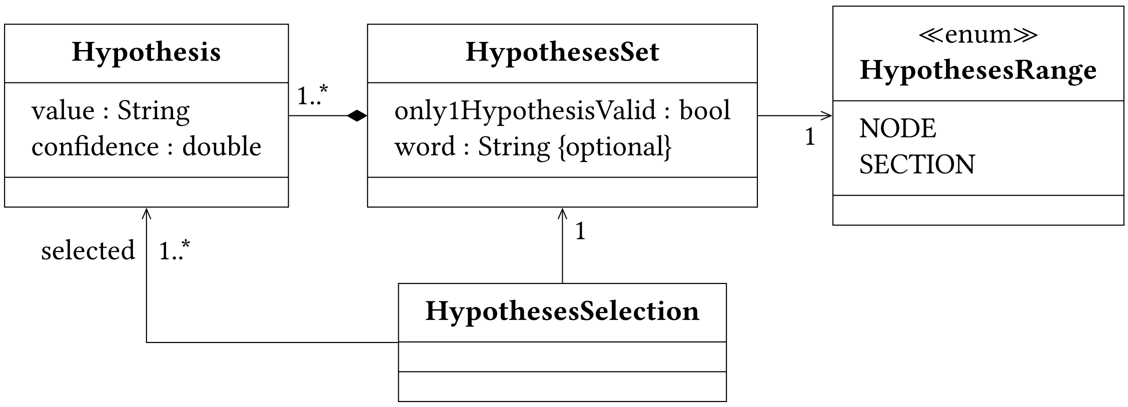


Figure 6.1.: Model: Hypotheses, Hypotheses Sets, and Hypotheses Selections

of a certain value and a specific confidence. In this thesis, the value is represented by a string that contains the necessary information. The actual type of representation depends on the agent. Confidences are represented as arbitrary floating-point values. This takes into account that, depending on the agent, the values of the confidence may include uncertain ranges. In contrast to the initial design during the approach, hypotheses are always part of a fixed set of multiple hypotheses. Such a set contains related hypotheses. For example, if you consider word sense disambiguation, you get exactly one set of hypotheses for each word that is disambiguated. Each set contains the possible meanings for the given word. Therefore, a property of such a set is the corresponding word. This property is optional, because the hypotheses of agents do not necessarily have to refer to only one word. Furthermore, the hypotheses in such a set always refer to a range of the input. Similar to the previous chapter, two types can be identified. On the one hand those hypotheses that refer to single words of the input. Since words in the PARSE project are stored in nodes, this range is called *Node* range. This includes the hypotheses of WSD. On the other hand, hypotheses can be related to the entire input text. In this case the range is called the *Section* range. A set of hypotheses always refer to exactly one type of hypotheses range. The last property of a set of hypotheses is an indicator called `only1HypothesisValid`. This indicator defines whether multiple hypotheses in a set might be correct. The indicator has an impact on the selection functions later in this chapter. In case that only one hypothesis of a set can be correct, there is no need to consider combinations within the sets. This reduces the search space for such sets. The last element of the model of hypotheses is the selection of certain hypotheses. Like in the first design in the approach, a selection contains selected hypotheses from a fixed set of hypotheses. Therefore, the actual realization of a selection refers to exactly one set of hypotheses. This set contains the eligible hypotheses. Furthermore, the selection points to the selected hypotheses from the set.

Pseudo-Hypotheses Concerning hypotheses, a last important term is *pseudo-hypothesis*. Path exploration aims to improve the results of agents without changing the way the agents work. In order to be able to check later on to what extent the results of agents have been improved by exploration, the results must be comparable regardless of the use of exploration. Therefore, it is necessary that the results obtained by an agent without the use of hypothesis-exploration can be transformed into hypotheses. This task is later taken over by the agents' specifications. Hypotheses derived from the results without exploration are called pseudo-hypothesis. In particular, these hypotheses have no specific confidence value.

Through this implementation of the model for hypotheses and the definition of pseudo-hypotheses, the first task of this second phase (T2.1) has been completed.

6.2. Layered Exploration

This section discusses the realization of the layered exploration. The layered hypotheses graph, the result of the layered exploration, has been introduced in Section 2.4. Before considering the actual exploration, the actual implementation and structure of the graph will be examined in detail. This realization is shown in Figure 6.2. The realization of the exploration graph is similar to the expected graph in the fundamentals of this thesis. It is still a graph that is divided into layers. Each layer is related to exactly one agent that is responsible for the creation of the hypotheses of the layer. In contrast to the

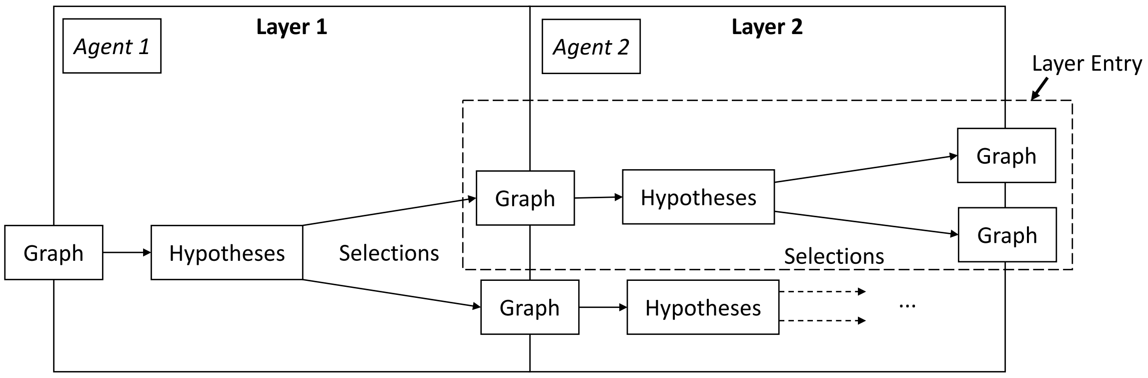


Figure 6.2.: Layers & Layer Entries in Exploration

representation at the beginning of this thesis (cf. Figure 2.6), the inputs of each layer are PARSE Graphs (*Graph*). These PARSE Graphs contain the information before the execution of the agent of the layer. The application of the agent leads to another instance of a PARSE Graph that is used to retrieve the hypotheses. Using selection functions, the layer creates new so-called layer entries in the succeeding layer. A layer entry contains information on the generated hypotheses, the input graph, all selections, and references to the next layer entries in the next layer. Therefore, the actual graph consists of layer entries that build a tree. After exploration, it can be decided to explore again differently, because the hypothesis graph contains the PARSE Graphs as copies. For example, other selection providers could be used to further extend the explored hypotheses graph. Because the expansion of the graph and the further exploration is preserved at runtime, exploration strategies are conceivable that could make use of already explored paths.

In Figure 6.3 you can see the structures needed to explore the Hypotheses Graph. The first important point to mention is the type of result of the exploration process.

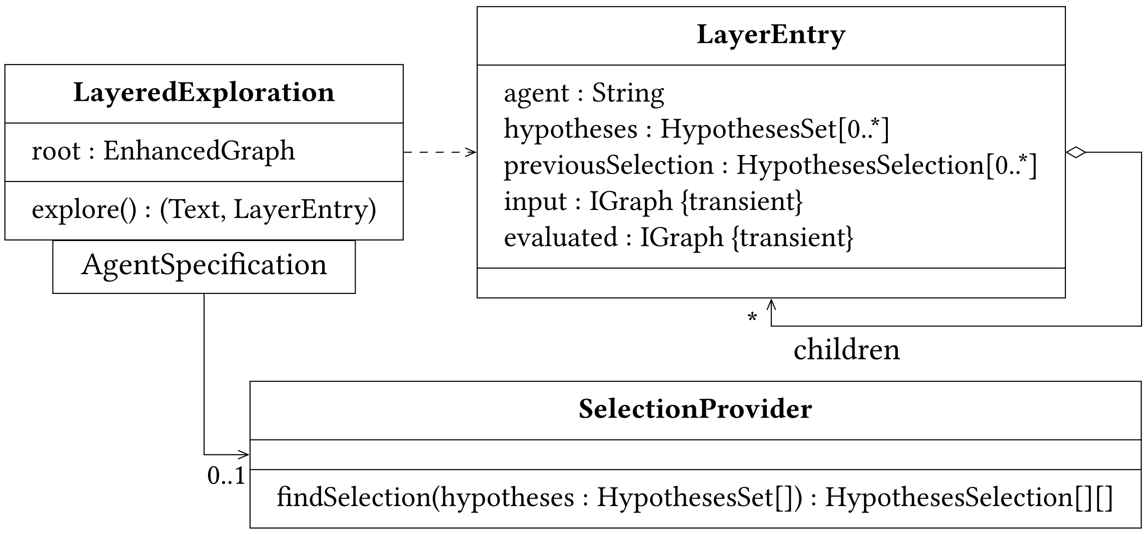

LayeredExploration
root : EnhancedGraph
explore() : (Text, LayerEntry)
AgentSpecification
LayerEntry
agent : String
hypotheses : HypothesesSet[0..*]
previousSelection : HypothesesSelection[0..*]
input : IGraph {transient}
evaluated : IGraph {transient}
*
children
SelectionProvider
findSelection(hypotheses : HypothesesSet[]) : HypothesesSelection[][]
0..1

Figure 6.3.: Implementation: Layered Exploration

The exploration result is a combination of input text and the first layer entry. Thus, it contains the whole exploration in the hypotheses graph based on its root layer entry. The layered exploration knows only the initial PARSE Graph (respectively `EnhancedGraph`, cf. Section 5.4.2) and a mapping from agent specification to selection provider. These mappings define the selection providers for the different layers of the exploration that are identified by an agent specification. Each Layer can have up to one selection provider. In the absence of a provider, the agent does not generate any hypotheses, i.e., the usual state without generating hypotheses. This way, the exploration is not restricted to agents that are able to generate hypotheses. Instead, the layered exploration is able to deal with a combination of agents that uses hypotheses, as well as agents that uses no hypotheses.

`SelectionProviders` are responsible for the creation of hypotheses selections. Therefore, a selection provider takes all sets of hypotheses for a certain layer and creates different combinations of selections per set. Thus, the return type of the `findSelection` function of the selection provider is a list of multiple selections (`HypothesesSelection[][]`). Each `HypothesesSelection[]` has the same size as the amount of sets of hypotheses (parameter `hypotheses`). Hence, every selection belongs to exactly one set of hypotheses. The realization of some selection providers is going to be discussed in Section 6.3.

The actual layer entry as stated in Figure 6.3 consists of serialized and transient properties. Properties that have to be serialized are those that represent the result of the exploration. The first property for serialization is the `agent`. The agent property refers to the agent that is responsible for layer of the layer entry. Secondly, the hypotheses sets that are generated by the corresponding agent are located in the layer entry. The selection from the previous layer entry that lead to this layer entry is also needed for the serialized version of the layer entry. In order to complete the serialized version of the exploration graph, the layer entry contains the successive layer entries as children. Information like the PARSE input graph or the resulting PARSE Graph (`evaluated`) are not stored in the serialized version of the exploration result.

The realization of the exploration process is illustrated in Algorithm 2. The layered exploration needs an initial PARSE Graph to start exploration (`root`). Furthermore, the mapping between the specification of agents and selection providers has to been known. The exploration process starts with the creation of the layers according to the configured agent specifications. During the creation of the layers, the steps already known from the agent execution (cf. Section 5.4.2) are performed. These include the checks for a valid specification hierarchy. Besides the valid specification hierarchy, the exploration has to check, whether the configured agents with hypotheses have suitable selection providers. Otherwise, an exploration cannot be executed.

After checking all constraints, the next step is the creation of the root layer entry in the first layer. This root layer entry is the first part of the exploration result at the end of the exploration. The creation of the root layer entry includes the invocation of the first agent on the root PARSE Graph.

With the completion of the generation of the first entry, the actual exploration begins. The actual steps of the exploration of the layers (`exploreLayers()`) is separately shown in Algorithm 2. The exploration of the layers iterates over all layers according to their order, starting with the root layer. This kind of exploration (layer by layer) is only possible, due the fact that no cycles have been found in the previous phase (cf. Chapter 5). As

Algorithm 2 Layered Exploration — explore

```
Require: EnhancedGraph $root$, AgentSpec × SelectionProvider ($agents \times selectors$)
 1: function EXPLORE($root, agents$)
 2:     $layers \leftarrow$ CREATELAYERS($agents$)
 3:     CREATEROOTLAYERENTRY($layers, root$)
 4:     EXPLORELAYERS($layers$)
 5:     return CREATEEXPLORATIONRESULT($root, layers$)
 6: end function
 7: function EXPLORELAYERS($layers$)
 8:     for ordered $layer \in layers$ do
 9:         $selectionProvider \leftarrow$ GETSELECTIONPROVIDER($layer, selectors$)
10:         for $entry \in layer.entries$ do
11:             $hypotheses \leftarrow$ GETHYPOTHESES($entry.input, layer.agent$)
12:             $selections \leftarrow$ FINDSELECTIONS($hypotheses, selectionProvider$)
13:             for $selection \in selections$ do
14:                 $graph \leftarrow$ CLONEGRAPH($entry.input$)
15:                 $graph \leftarrow$ APPLYSELECTION($layer.agent, selection, graph$)
16:                 CREATECHILDRENBYSELECTION($layers, entry, graph, selection$)
17:             end for
18:         end for
19:     end for
20: end function
```

already known, each layer that produces hypotheses using its agent has a specific selection provider to find the selections of hypotheses for the next layer.

Thus, the next step during the exploration of the current layer is the application of the suitable selection provider to the hypotheses of each layer entry. Therefore, the specification of the agent is used to retrieve the actual hypotheses from the input graph. The application of a selection provider then provides an amount of different selections for the current hypotheses. After retrieving the selections, the layered exploration clones the input graph of the current layer entry and applies the selection to it. The resulting new PARSE Graph is now used as input graph for a new child layer entry in the successive layer.

The exploration finishes after all layers have been explored. The result of the layered exploration is created by the actual layers and the input text in a last step. With this implementation of the exploration, the second task of the second phase (T2.2) was successfully completed.

6.3. Selection Provider

In the next step, the focus lies on the coordination of the exploration. The control of the exploration in the approach of this work is carried out by so-called *Selection Providers.*

Selection Providers take the sets of hypotheses of a certain layer entry and provide selections of hypotheses. The particular challenge here is that the providers can only rely on information from the hypotheses. The reason for this lies in the structure of the approach. One requirement during the design was that the mechanisms for exploring hypotheses should be reusable. This means that the selection provider is not adapted directly to a specific agent. Instead, the idea is that a certain selection strategy should be chosen for a certain agent. The finding of suitable strategies is partly covered in the following chapters. However, the final finding of strategies will go beyond the scope of this thesis and is rather to be seen in the future work. The strategies are thus related to aspects of the hypotheses themselves. Only the values, ranges, confidence and defined properties of the general hypothesis are available as information. In the following, various implemented providers are presented.

6.3.1. Full Exploration Strategy

The simplest variant of a selection is the complete exploration. A full exploration uncovers all combinations of hypotheses and uses the whole possible search space for the approach. However, the search space can be exponentially large. Depending on the size of the input, the search is then not practical. Especially the property `only1HypothesisValid` must be considered. In cases where more than one hypothesis of a fixed set of hypotheses may be valid, a complete exploration of the search space is difficult. Besides the actual selection, there would be a variable number of hypotheses to consider. Because of that, the full exploration strategy is restricted to sets of hypotheses that contain only one hypothesis that can be valid in a selection. Therefore, the strategy covers only the combination of all found hypotheses in the sets of hypotheses of a certain layer entry. To clarify what the strategy actually selects, Algorithm 3 shows the procedure reduced to the essential. As shown in

Algorithm 3 Selection Providers — Full Exploration

```
Require: List of IHypothesesSet sets (∀ sets: only1HypothesisValid = true)
1: function fullExplore(sets)
2:     counts : int[] ← map(λs. count(s.hypotheses), sets)
3:     selections : IHypothesesSelection[][] ← []
4:     for indices ∈ {l | at each index : l contains a value less than in counts} do
5:         selected : IHypothesis[] ← SelectByIndices(sets, indices)
6:         selections ← selections ∪ {HypothesesSelection(selected)}
7:     end for
8:     return selections
9: end function
```

the algorithm, all combinations of hypotheses are built. Therefore, the algorithm stores the amount of different hypotheses per set in `counts`. After that, it generates combinations of indices. Hence, the algorithm selects exactly one hypothesis from each set. These indices define the selected hypothesis per set and is used to generate the actual selection. At the end, the full exploration strategy returns all generated selections.

This realization allows the full exploration of hypotheses sets that contain only one correct hypothesis. The actual search space is not reduced. Therefore, the practical use of such kind of exploration has to be checked.

6.3.2. Top-X Confidence & Top-X Sliding Window

The full exploration strategy is only applicable if the actual hypotheses sets contain only one valid hypothesis. Furthermore, the full exploration strategy does not control the search space. In the following, two selection providers are introduced: *Top-X Confidence* and *Top-X Sliding Window*.

First, the *Top-X Confidence* provider. Like the full exploration provider, this provider is applicable to hypotheses sets that contain only one valid hypothesis. Instead of building all combinations of hypotheses and therefore not controlling the search space, the provider explores only a part of the search space. The provider is parameterized by a natural number that indicates the amount of rankings to generate. This number defines the maximum amount of selections provided by the selection provider. The *Top-X Confidence* selection provider generates selections according to the confidences of the hypotheses in the hypotheses sets. First, the provider selects the top scored hypothesis of each set of hypotheses. After that, the provider generates selections of the second best hypotheses, and so forth. The provider stops the generation of new selections either if it reaches the provided maximum amount of selections or if any of the sets of hypotheses does not contain any unused hypothesis anymore. Therefore, the provider is the first mentioned provider that actually restricts the search space. The strategy considers the presence of a ranking within the hypotheses. Thus, it selects not all hypotheses or just the best hypotheses. Rather, second-best (or third-best, etc.) findings are considered to be taken into account.

The second selection provider that has been created, is called *Top-X Sliding Window*. This provider focuses on sets of hypotheses that contain more than one correct hypothesis. It is parameterized by two values. First, the amount of selections to generate at maximum. Second, the amount of hypotheses per set that shall be considered as correct. The selection provider itself generates selections per set of hypotheses. Thus, you may build combinations of selections if more than one set of hypotheses is provided for the selection process. The selection process starts with the creation of a window that will shift through the sorted hypotheses of a hypotheses set. The size of the window is initially set to the provided amount of hypotheses that are considered as correct. After creating this window, the selector takes the hypotheses that lay inside the window as selection. Subsequently, the sliding window is shifted. A shift means that e.g. with a window of size two the top-1 and top-2 hypothesis is no longer covered but the top-2 and top-3 hypothesis. After shifting, the next selection is generated according to the hypotheses in the window. This procedure is executed until the amount of the selections reaches the maximum amount or no hypotheses are located inside the current window.

This two providers realizes two simple selection mechanisms that control the search space. The actual performance in contrast to other selectors or no exploration is going to be compared in the evaluation.

6.3.3. Random Hypothesis Selection Provider

The next selection provider to be mentioned is the *Random Hypothesis Selection Provider*. This provider empowers randomness to control the search space provided by the full exploration provider mentioned above. The provider takes advantage of the fact that with full exploration, the exploration itself is not practicable, but the definition of the possible paths is. Therefore, this selector uses the definitions of the paths as provided by the full exploration strategy but filters them. The provider is parameterized by two possible restrictions of the amount of selections provided by it. First, you can restrict the actual amount of selections that are provided by the selection provider. Second, you can specify a ratio within $(0, 1]$ that defines the amount of chosen selections from the full exploration strategy. In addition, you can specify both limits, where then the minimum of both is used as actual limit. Whenever a selection is requested, the provider generates all possibilities. After that, the provider calculates the actual maximum for this amount of selections. In a last step, the selection provider deletes random selections until the amount of selections is smaller than the maximum calculated. Thus, a random set of selections is provided by the selector. It is important to note that this provider, like the full exploration strategy, is only built for sets of hypotheses that contain only one correct hypothesis.

6.3.4. Same-Word-Same-Decision Decorator

After considering different selection providers in the previous subsections, this subsection deals with a specific *Decorator*. Decorators define a special type of selection provider. They are realized using the decorator pattern. Therefore, it is possible to combine different decorators. A decorator executes three steps:

1. Filtering of input hypotheses
2. Generation of selections by delegation to another selection provider
3. Remapping of selection to original hypotheses

In the first step, the decorator can change the hypotheses for the creation of the respective selections. This is the step of *filtering*. After filtering the hypotheses, the decorator delegates the hypotheses to another selection provider. This selection provider creates the actual selections. In the last step the decorator modifies the selections it has received. This post processing allows the decorator to create valid selections for the original hypotheses.

In the following, the so-called *Same-Word-Same-Decision Strategy* is introduced. This strategy is actually a special decorator for hypotheses that are related to single words. It aims to ensure that every occurrence of a word has the same selected hypotheses for a certain selection. Therefore, the decorator groups the sets of hypotheses by their respective words from the input text. After grouping the sets, new sets will be generated per group. A set for a certain word contains all the hypotheses from the occurrences of the word in the text. Thus, this decorator ensures that different hypotheses sets for different occurrences of a word in a text choose the same hypotheses in a selection. For example, for WSD you can use the assumption that the sense of a given word is always the same over the whole document. More details will be given in the next section.

With the realization of these different selection providers, the third task of the second phase (T2.3) is completed.

6.4. Hypotheses for Agents

Since the exploration of hypotheses is possible now, hypotheses have to be generated in the following. Therefore, existing agents have to be extended. In the first part of this section, the actual structure of the specifications for agents that provide hypotheses is introduced. After that, the concrete implementation for three agents is presented. The realization of handling mechanisms for hypotheses of three agents completes the fourth task of this phase (T2.4).

6.4.1. Agent Hypothesis Specification

In Section 4.1.2 the coarse structure of an Agent Hypothesis Specification has been introduced. The following paragraphs deal with the actual realization of this type of specifications. An overview is given in Figure 6.4. As you can see, the Agent Hypothesis

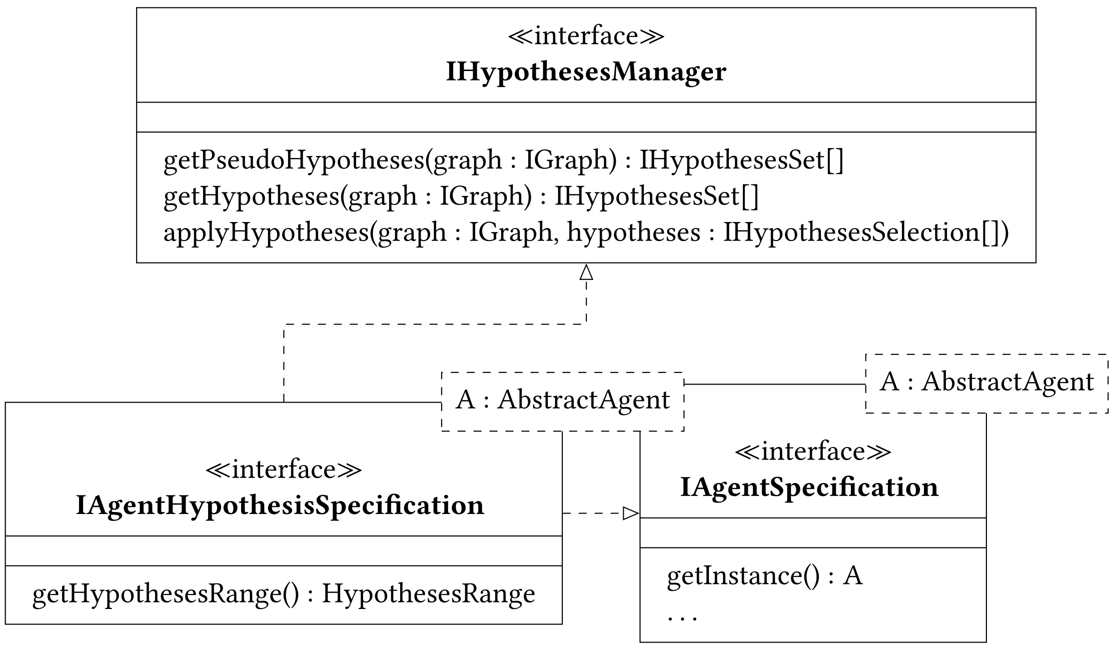

≪interface≫
IHypothesesManager
getPseudoHypotheses(graph : IGraph) : IHypothesesSet[]
getHypotheses(graph : IGraph) : IHypothesesSet[]
applyHypotheses(graph : IGraph, hypotheses : IHypothesesSelection[])
A : AbstractAgent
A : AbstractAgent
≪interface≫
IAgentHypothesisSpecification
getHypothesesRange() : HypothesesRange
≪interface≫
IAgentSpecification
getInstance() : A
...

Figure 6.4.: Implementation: Agent Hypothesis Specifications (Agent Analysis Port)

Specification is a specialization of a normal agent specification. Furthermore, the specification inherits the methods of a Hypotheses Manager. A Hypotheses Manager defines the interface of an agent that can handle hypotheses. Therefore, it provides three methods: First, a method to generate pseudo-hypotheses from the "old" version of the agent. Thus, it is possible to compare the exploration with the original results of an agent (cf. Section 6.1). Second, a method to retrieve the actual hypotheses from a PARSE Graph. To be able to read the hypotheses, the corresponding agent has to be executed. The last functionality provided by a Hypotheses Manager is the application of hypotheses selections to a PARSE Graph. Besides the functions of the Hypothesis Manager and the Agent Specification, the Agent Hypothesis Specification provides an auxiliary method to obtain the range of hypotheses generated by the specification. This function is a simplification for the implementation. The information about the type of hypothesis ranges can be obtained directly from the hypotheses.

After describing the structure, the construction of the super class for general agent hypothesis specifications is briefly discussed below. The class itself is shown in Figure 6.5. As shown in the definition of the class, this abstract super class fixes the generic agent

```
abstract class AbstractAgentHypothesisSpec<A extends AbstractAgent &
    IHypothesesManager> implements IAgentHypothesisSpec<A> {

    protected final A agent;
    private AbstractAgentSpecification<? super A> spec;

    protected AbstractAgentHypothesisSpec(
        AbstractAgentSpecification<? super A> spec, A agent) {
        this.agent = agent;
        this.spec = spec;
        this.spec.setInstance(this.agent);
    }

    @Override
    public final A getInstance() { return agent; }

    // Delegations to spec: getMode(), get[Provided|Required]Ids()

    @Override
    public final List<IHypothesesSet> getHypotheses(IGraph graph) {
        return agent.getHypotheses(graph);
    }

    // Further delegations to agent:
    // applyHypotheses(..), getPseudoHypotheses(..)
}
```

Figure 6.5.: Abstract Agent Hypothesis Specification (Code)

parameter to a class that extends an `AbstractAgent` (the super class of all PARSE agents) and implements a Hypotheses Manager. Furthermore, the specification combines the instance of an PARSE agent and a suitable (normal) Agent Specification, as it were created in Chapter 5. The agent specification used by the Agent Hypothesis Specification is parameterized so that the instance of the agent can be set to `A`. Thus, the agent specification is defined as `AbstractAgentSpecification<? super A>` (cf. Section 5.4.1). During creation of the Agent Hypothesis Specification, the instance of the inner agent specification is set to the same as the agent of this specification. As indicated in the code excerpt, the specification delegates the necessary methods to the instances of the old specification or the instance of the agent. The use of this super class is described in the following implementations for the agents.

6.4.2. Wiki WSD

In the following, the Wiki WSD agent will be extended with the handling of hypotheses. As already known from Chapter 5, the Wiki WSD agent [14] is a PARSE agent that uses information from Wikipedia to provide the senses of words in an input text. To classify an input, the agent first loads a pre-trained classifier for classification. After loading the classifier, the agent is initialized.

The interpretation of an input starts with the creation of an attribute type in the PARSE Graph. The attribute “wsd” is added to the token nodes in the PARSE Graph. As already introduced in Chapter 2, Token nodes represent the actual text within the PARSE Graph. Thus, the agent is able to add the actual sense of a word to the respective token as simple string attribute. For example for the word “bass” the meaning “bass (fish)” could be annotated to the token (cf. Chapter 1).

The agent simply creates a classification for each noun identified. A classification of the agent consists of the value (e.g. “bass (fish)”) and a score for that classification. The classification can produce multiple classifications per word but selects exactly one per noun. In the following the agent will be extended to provide the different classifications as hypotheses. Each noun is going to be represented by one fixed set of hypotheses.

Therefore, the current agent is used as base class for a new agent. This new agent is called `MultiHypothesisWSD`. Furthermore, this class implements the `IHypothesesManager` interface to provide the actual hypotheses. The following steps have to be taken to realize the `MultiHypothesisWSD`:

1. Preparation of the PARSE Graph
2. Latching into the classification mechanism
3. Storing of found senses as hypotheses
4. Realization of retrieval and application of hypotheses from the graph
5. Realization of retrieval of pseudo-hypotheses

In the first step, the new agent adds a further attribute to the token node type. This new attribute is used for storing multiple senses of a word in a serialized representation. The second step injects the necessary code to get the actual classifications of a word directly. Thus, the new agent calls the existing classification service of the old agent. After obtaining multiple classifications for one word, the classifications are serialized and stored into the new defined attribute of the token nodes. Afterwards, the classification of the old agent is resumed by redirecting the control flow to the old agent’s methods. In summary, the old functionality is retained and the Liskov substitution principle is not violated. Only further classifications are stored in a new attribute in the nodes of the words.

What is still missing are the mechanisms for processing the hypotheses of the WSD. As explained above, the different classifications of a word is stored in the corresponding token node in the PARSE Graph. Therefore, the retrieval of the actual hypotheses has to parse the information stored in the nodes to generate the fixed sets of hypotheses per word. In Figure 6.6 you can see the definitions of the hypotheses provided by the Wiki WSD agent. The specification of the agent, creates a `HypothesesSet` for each word that is annotated with WSD information. Such a fixed set of hypotheses contains the information about the actual word for disambiguation as well as a set of `WSDHypotheses`. In addition, the property `only1HypothesisValid` is set to `true` as a word has only one sense in a fixed context. As shown in the picture, a `WSDHypothesis` stores exactly the same information

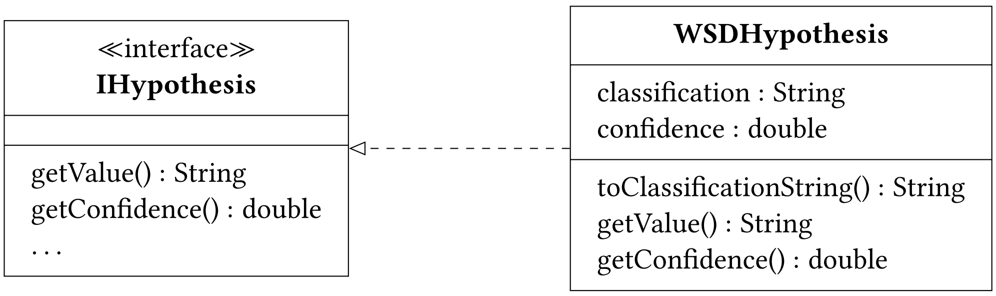


Figure 6.6.: Implementation: Specific Hypothesis for Wiki WSD

as a classification from the unmodified agent: the actual classification as string and the confidence of the classification as floating-point value. In addition to the needed methods of `IHypothesis`, the `WSDHypothesis` provide a function to generate the classification string that represents the value of the hypothesis. This classification string is needed as the original Wiki WSD agent stores the classification of a word as that kind of string into an attribute of the respective word. Thus, the agent hypothesis specification is able to apply a selection that selects specific word senses (as `WSDHypotheses`).

In order to compare the results of an exploration to the original behavior of the agent the specification has to provide pseudo-hypotheses for the executed original Wiki WSD agent. Therefore, the annotated sense of a word that is stored as attribute to the respective token node, is used as value of a `WSDHypothesis`. Since the information about the confidence is not stored, but only one value is possible per word, `NaN` is set as confidence. Thus, the mechanisms for the retrieval of real hypotheses can be used to get the pseudo-hypotheses.

```
class WikiWSDHypothesisSpec extends
    AbstractAgentHypothesisSpec<MultiHypothesisWSD> {

     public WikiWSDHypothesisSpec() { this(DEFAULT_HYPOTHESES); }

     public WikiWSDHypothesisSpec(int maxHypotheses) {
          super(new WikiWSDSpec(), new MultiHypothesisWSD(maxHypotheses));
     }

     public HypothesisRange getHypothesesRange() { return NODE; }
}
```

Figure 6.7.: Wiki WSD Agent Hypothesis Specification (Code)

The realization of the actual agent hypothesis specification is shown in Figure 6.7. As shown in the code excerpt, the specification allows to set the amount of hypotheses generated per word. Furthermore, the specification defines the base specification as `WikiWSDSpec` — the old specification of the original Wiki WSD agent. The actual implementation of the agent is set to `MultiHypothesisWSD` the new agent. In addition, the Wiki WSD agent does only provide hypotheses that belong to a node (Hypotheses Range: *Node*). For the following agents the specification is structured analogously. Therefore, they are not explicitly described in the following sections.

6.4.3. Topic Detection

As shown in Chapter 5, the Topic Detection agent [14] depends on the Wiki WSD agent. Thus, the following deals with the changes needed to use the hypotheses approach for the Topic Detection agent. The Topic Detection agent uses the senses in the attributes of the token nodes to determine a common topic of the input text. Concepts from Wikipedia are used to provide the topics.

By default, the Topic Detection agent uses online resources to obtain the necessary information about the concepts. After initializing the agent, the PARSE Graph is extended with a Topic Node Type. A topic node is a PARSE node that contains one attribute that stores the found topics. Exactly one node of this type is created by running the Topic Detection agent. Also, the node is not connected to anything. It therefore serves as data storage. The actual processing of the agent consist of three steps: First, the agent extracts all annotated senses of the Wiki WSD agent from the PARSE Graph. After that, the agent creates a so-called "Topic Graph" [14]. The topic graph, is created using the senses found in the PARSE Graph. Afterwards, the topic graph provides a list of topics that are related to the input text. Each topic consists of a label, a score, and the related senses used for this topic. The obtained topics are stored in the attribute of the topic node to provide the information for successive agents. This completes the execution process of the normal Topic Detection agent.

In the following the agent is extended to work with hypotheses. Similarly to the changes regarding the Wiki WSD agent, the extension of the Topic Detection agent is done by adding a sub class to the original agent. Like the changes for the Wiki WSD agent, the extended agent latches into the preparation of the PARSE Graph, and the storing mechanisms to obtain the necessary information. The first step of the new agent is the addition of a new attribute to the topic node type. This attribute contains the different topics for the hypotheses provided by the specification of the new Topic Detection agent. Using this attribute, the new agent stores a set of topics found during the creation of the topic graph. The actual processing of the topics is not changed. Therefore, the new agent behaves in the same way as the old agent. It only adds more information to the topic node. Thus, the Liskov substitution principle is not violated.

With this state, only the mechanisms that generate and process hypotheses from the available information are missing. For this purpose, Figure 6.8 shows how the hypotheses for the Topic Detection agent are structured. A topic hypothesis represents exactly one topic. Therefore, it contains this topic. The value of the hypothesis refers to the label of the topic. The hypothesis' confidence is equal to the topic's score.

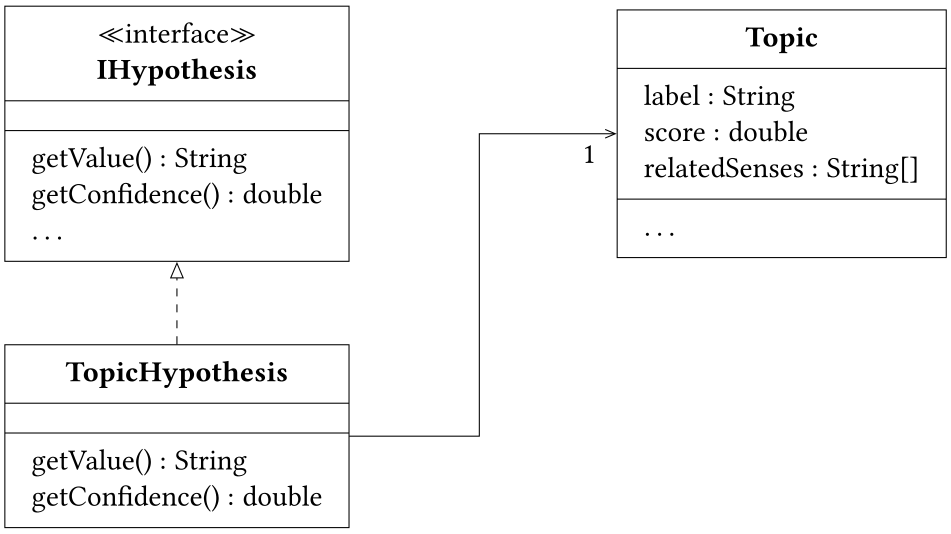


Figure 6.8.: Implementation: Specific Hypothesis for Topic Detection

The specification of the Topic Detection with hypotheses has to provide methods to generate these hypotheses from the topics stored in topic node. Therefore, the agent creates exactly one fixed set of hypotheses. This is defined by all topic hypotheses stored as topics in the topic node. The fixed set represents hypotheses with the Hypotheses Range *Section*. This is necessary, because the topics refer globally to one input. In addition, the fixed set contains the information that multiple hypotheses could be correct (`only1HypothesisValid` is set to false). Eventually, a sentence can cover several topics. For the extraction of the pseudo-hypotheses the same mechanisms can be used. Only the old attribute in the topic node must be used. The application of hypothesis selections then consists in describing the topic attribute with the topics in the selected hypotheses. With this implementation, the hypotheses of Topic Detection can now also be used for this approach.

6.4.4. Ontology Selection

The last agent in the hierarchy starting at the Wiki WSD agent is the Ontology Selection agent [14] (cf. Table 5.1). Therefore, in the following the handling of hypotheses of this agent is discussed. The Ontology Selection uses the found topics to determine suitable ontologies for the input text. Thereby, the ontologies are divided into *Actor Ontologies* and *Environment Ontologies*. As the names already imply, the first type of ontologies are those that represent different actors. Actors present would be for example a robot, a virtual assistant, or a drone. The environmental ontologies that exist so far are e.g. a kitchen, a garden, or a children's room.

The initialization of the agent starts with the loading of the configured ontologies. This is important because it limits the number of possible hypotheses for the selection of ontologies. Neither the WSD nor the TD agent have this characteristic. During the loading process, the topics of the ontologies are extracted. Therefore, the ontologies contain concepts that can be used by the internal mechanisms of the Topic Detection agent to obtain the topics by these concepts. Like other agents, the first step during execution is the preparation of the PARSE Graph. The agent adds a new node type — the Ontology Node Type. Similar to the Topic Node of the Topic Detection, this node type is only used as data storage. In the case of the Ontology Selection the node contains two attributes: First, the selected ontologies. In addition, the agent creates a merged ontology that contains all selected ontologies. This merged ontology is the second attribute of the ontology node.

After the preparation of the PARSE Graph is finished, the actual processing starts. First, the Ontology Selection detects the stored topics of the input text provided by the Topic Detection agent. In the next step, the agent calculates the conformities for the different ontologies. This leads to a score for each ontology. Afterwards, the ontologies are filtered by a selection method (e.g. a threshold for the score). In a last step, the selected ontologies are stored to the respective attribute, the ontologies are merged to one ontology, and the merged ontology is annotated to the ontology node. This completes the execution process of the normal Ontology Selection agent.

In the following the agent is extended to work with hypotheses. Similarly to the changes regarding the agents before, the extension of the Ontology Selection agent is done by adding a sub class to the original agent. The extended agent latches into the preparation of the PARSE Graph and the storing mechanisms to obtain the necessary information. After the creation of the Ontology Node Type, the updated agent adds two new attributes to the node type: An attribute to store the found actor ontologies and an attribute to store the found environmental ontologies. During the execution of the agent, the present ontologies are rated regarding the topics of the input text. Afterwards, the new agent stores the information about the ontologies and scores into the two new attributes. Thus, the information about the scores of the ontologies is saved to the PARSE Graph and is accessible for the generation methods of the hypotheses.

Like already known from the Topic Detection agent, the information stored to the respective node is used to generate the hypotheses. The Ontology Selection agent uses a specific model for its hypotheses. This model is shown in Figure 6.9. You can see that the

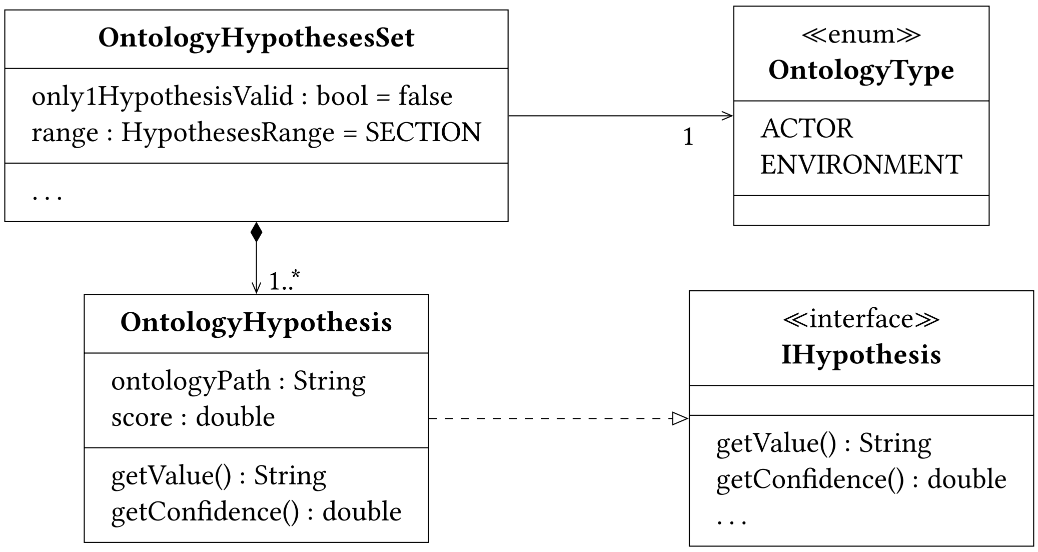


Figure 6.9.: Implementation: Specific Hypothesis for Ontology Selection

hypotheses of the Ontology Selection simply encapsulating the path to the ontology and its score. Furthermore, the Ontology Selection defines its own type of HypothesesSet. Besides the usual attributes of such a set, the set is also characterized by the type of ontology of the hypotheses. More precisely, Range and `only1HypothesisValid` are already defined. Since the information in ontologies might overlap, the sets are not restricted to contain only one valid hypothesis. The range of the hypotheses of the Ontology Selection is set to *Section*.

After its execution, the Ontology Selection agent provides exactly two fixed sets of hypotheses: First, a set that represents to possible actor ontologies. Secondly, a set that contains the possible environmental ontologies. If a selection of ontology hypotheses is given, the agent loads the selected ontologies by its path in a first step. After loading, the agent creates a merged ontology and sets the ontology attribute of the ontology node. Thereby, the application of the selection is completed.

The last functionality that is still missing is the generation of pseudo-hypotheses. This generation uses the attribute in the ontology node that contains all merged ontologies. Using that information the agent generates one fixed set of ontology hypotheses that contains all actor and environmental hypitheses. As no score is present, the agent uses NaN as score.

6.5. Agent Analysis Explorer

After the extension of existing agents has been completed in the previous section (T2.4), this section introduces the *Agent Analysis Explorer*. This explorer aims to complete the fifth task of this second phase — providing a GUI to visualize the exploration graph. The example for the whole section is the sentence “okay Armar go to the table grab popcorn come to me give me the popcorn which is in your hand”. That sentence is taken from the evaluation of Keim [14].

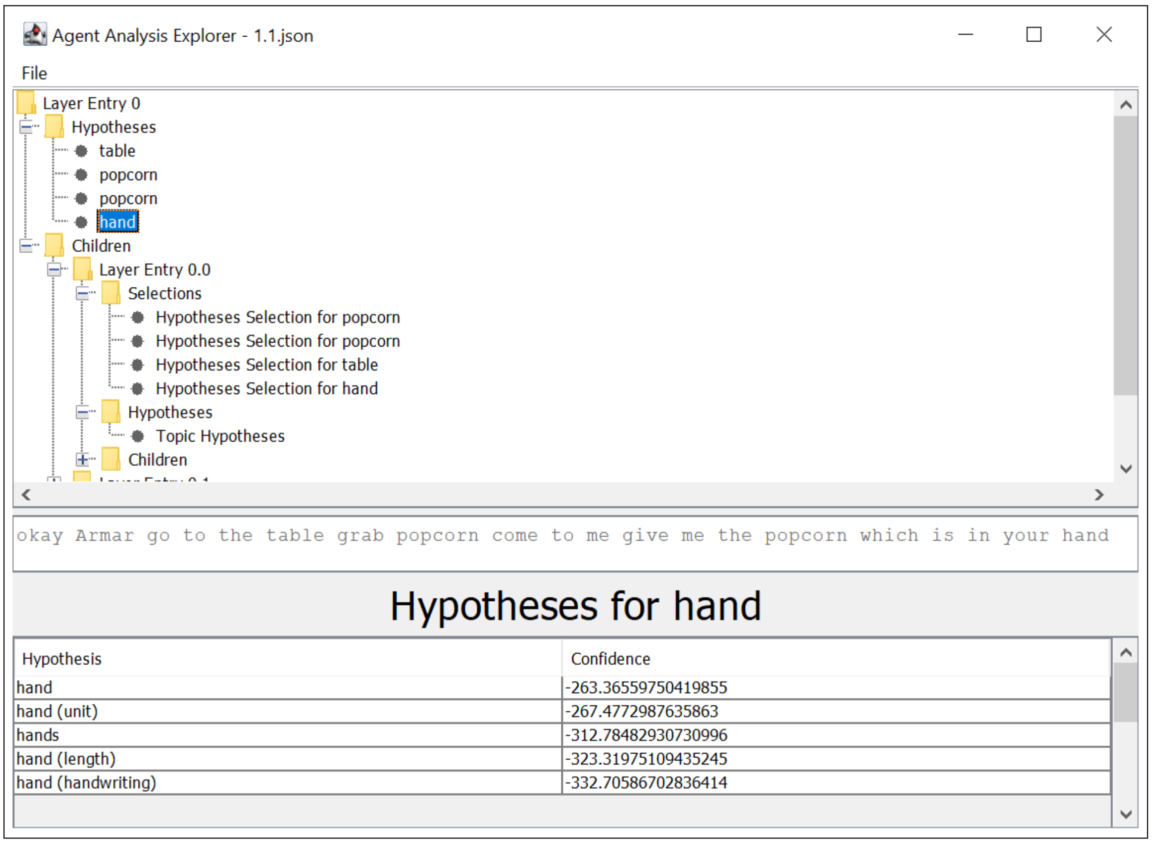
Before defining the actual graphical interface, the data that shall be visualized has to be considered. In the case of the Agent Analysis Explorer that data is an exploration result. An exploration result is the combination of the input text and the exploration graph (cf. Figure 6.3). In Figure 6.10 you can see an excerpt of the actual data for the example sentence. Like shown in the picture, the data is stored as JSON. The representation as JSON is only possible since the exploration graph is a tree. In general an exploration result has two root nodes: the input text (represented as string) and the root node of the exploration graph (`explorationRoot`). Each node of the exploration graph contains the declaration of the agent specification and a unique id that defines the node. Furthermore, it contains a set of `HypothesesSelection` called `selectionsFromBefore`. This set contains the selections of the previous layer that lead to this hypothesis. Therefore, the root layer entry (root node) does not contain any selection. In addition, a layer entry contains its actual hypotheses sets as a list. The example shown in Figure 6.10 contains exactly one hypotheses set. This set contains hypotheses for the word “hand”. Each hypothesis is characterized by the value, the confidence, and a human readable representation. The latter is omitted for clarity. In addition to the hypotheses, the hypotheses set contains the information about range,

```
{
    "inputText": "okay Armar go to the table ...",
    "explorationRoot": {
        "agent": "WikiWSDHypothesisSpec",
        "id": "WikiWSDHypothesisSpec-0",
        "selectionsFromBefore": null,
        "hypotheses-sets": [
            {
                "hypotheses": [
                    {
                        "value": "hand",
                        "prettyInformation": "...",
                        "confidence": -263.36559750419855
                    },
                    {
                        "value": "hand (unit)",
                        "prettyInformation": "...",
                        "confidence": -267.4772987635863
                    }, ...
                ],
                "only1HypothesisValid": true,
                "hypothesesRange": "NODE",
                "shortInfo": "hand",
                "wordOfHypotheses": "hand"
            }, ...
        ],
        "children": [...]
    }
}
```

Figure 6.10.: The exploration result of the sentence (Simplified)

word, and validity of multiple hypotheses (`only1HypothesisValid`). There is also a short description that contains in this case only the word the hypothesis refers to.

In summary, the GUI has to display the several information: The *Exploration Tree* (Layer Entries, Hypotheses, Selections), the *input text*, and the *information* about a specific element of the exploration tree (selected by a user in the tree). Figure 6.11 shows the actual GUI to represent the exploration of the Hypotheses Graph. The user interface is separated into several parts. First, the menu that allows the loading of exploration results via a File menu. Second, the graphical representation of the exploration graph. Since the exploration graph is a tree, the representation is a typical tree view as known from file explorers. Like the JSON data, the tree starts with the root layer entry ("Layer Entry 0") and contains two sub nodes. On the one hand, the hypotheses sets ("hypotheses") of the current layer. And on the other hand, the child entries of the current layer entry. The child nodes itself contain


Agent Analysis Explorer - 1.1.json

File

- Layer Entry 0
  - Hypotheses
    - table
    - popcorn
    - popcorn
    - hand
  - Children
    - Layer Entry 0.0
      - Selections
        - Hypotheses Selection for popcorn
        - Hypotheses Selection for popcorn
        - Hypotheses Selection for table
        - Hypotheses Selection for hand
      - Hypotheses
        - Topic Hypotheses
      - Children

```
okay Armar go to the table grab popcorn come to me give me the popcorn which is in your hand
```

Hypotheses for hand

Hypothesis	Confidence
hand	-263.36559750419855
hand (unit)	-267.4772987635863
hands	-312.78482930730996
hand (length)	-323.31975109435245
hand (handwriting)	-332.70586702836414

Figure 6.11.: The Agent Analysis Explorer

the successive layer entries. A third optional sub node of a layer entry is the "Selections" node that contain the selections from the previous layer that lead to the current layer entry. Like shown in the picture the root layer entry has no selections as no predecessor exists. The third part of the GUI simply shows the input text that is used for the loaded exploration. The last part of the interface is responsible to show the information from the node selected in the exploration graph. Different types of nodes must be distinguished: Layer Entries, Hypotheses Selections, and Hypotheses Sets. If you select a layer entry, the table shows information about the actual agent that is executed. Currently the only information shown is the name of the agent. If a selection of hypotheses is marked, the table shows the selected hypotheses along with their confidences. The last option of selections is the selection of a fixed set of hypotheses. Such a selection is shown in the figure. Like shown in the picture the selection of a set of hypotheses lead to the viewing of the hypotheses of this set together with its confidences. In the figure you can see the hypotheses for the word "hand" that are known from the JSON (cf. Figure 6.10). This GUI completes the work on the second phase of the approach.

7. Finding Good Rating Functions

In the last chapters, the exploration of hypotheses of some agents was implemented. It is therefore possible to generate paths for a combination of agents and selectors. Each path represents a possible *result* of the agents. The last question that is still unanswered is how to distinguish good paths from the bad ones. In particular, it is not yet clear what "good" actually means. In the following, I will first discuss what "good hypotheses" mean in this thesis. Subsequently, a class of rating functions is proposed, that was implemented in this thesis. Finally, the *Agent Analysis Evaluator* is introduced as a tool for the rating of hypotheses that is later necessary for the evaluation.

A path in a given Hypotheses Graph consists of the hypotheses of each layer and the selections within each layer. The *results* of the actual agents are defined by the selections and the hypotheses in the following way: If the hypotheses of an agent are filtered by a selection provider, the result of the agent is defined by the selected hypotheses. Thus, the definition of a result respects the actual selections within a Hypotheses Path. If the hypotheses of an agent are not filtered by a selection provider, the result of the agent is defined by all generated hypotheses of the agent. Typically, this is the case if the agent is the last agent in the hierarchy. This is so, because no selection on the hypotheses of this agent is executed. With this definition of how to derive *results* from paths, the definition of "good" paths is now quite clear.

Good Rating Functions identify **good paths**. Good paths are those paths whose **results can be interpreted as good** based on a **metric**.

With this definition the same metrics can be used to evaluate paths that are used for the original execution. By using the same metrics as the original execution, a Path (respectively the results in a Path) can be compared to the execution without hypotheses.

7.1. Rating Functions for Hypotheses Paths

After getting insight about the quality of Paths, this section deals with a group of rating functions. This type of rating functions is called *Normalized Aggregate.* Chapter 5 introduced that confidences of hypotheses have arbitrary ranges, depending on agents. For example, the *Wiki WSD* agent uses logarithmic scales, whereas the *Topic Detection* agent uses a probability. A rating function has to use some mechanism to handle the different types. In case of the *Normalized Aggregate* Rating Functions this is achieved by normalizing the confidence values.

The details of the algorithm is shown in Algorithm 4. As you can see, the rating function

Algorithm 4 Normalized Aggregate — ratePaths

```
Require: List of LayerEntry[] $paths$, LayerEntryEvaluation $lee$, LayerCombination $lc$
 1: function ratePaths($paths$)
 2:     if $paths = []$ then
 3:         return []
 4:     end if
 5:     $pathLength \leftarrow max(\lambda p.\ p.length, paths)$
 6:     $mins \leftarrow [\infty, \infty, \dots, \infty]$            // pathLength times $+\infty$
 7:     $maxs \leftarrow [-\infty, -\infty, \dots, -\infty]$      // pathLength times $-\infty$
 8:     findMinMaxPerEntryPerPath($mins, maxs, paths$)
 9:     $normalizedRatings \leftarrow []$
10:     for $path \in paths$ do
11:         $normalizedPath \leftarrow$ Normalize($path, mins, maxs$)
12:         $entryScores$ : double[] $\leftarrow$ LayerEntryEvaluation($normalizedPath, lee$)
13:         $pathScore$ : double $\leftarrow$ LayerCombination($entryScores, lc$)
14:         $normalizedRatings \leftarrow normalizedRatings + [pathScore]$
15:     end for
16:     return $normalizedRatings$
17: end function
```

needs a list of hypotheses paths that shall be rated. In addition, algorithm is parameterized by two parameters: First, the *Layer Entry Evaluation* and secondly the *Layer Combination*. Both parameters will be discussed in the successive paragraphs. For now the parameters define the actual aggregation of the scores of hypotheses. The overall idea behind the whole algorithm is that paths are evaluated according to the generated confidence. Actually, the evaluation of a path is done in relation to all other paths that are passed to the algorithm. The full details are explained in the following.

The algorithm starts with a check whether any path is supplied. If no path is supplied, the rating function skips any processing. Otherwise, the algorithms starts by determining the length of the paths. During exploration, it is possible, that a selection of hypotheses lead to no further hypotheses in the successive layers of the hypotheses graph. Thus, the length of the corresponding path would be less than the expected length. Therefore, the length of the paths is determined by finding the longest path within the supplied paths.

After determining the length of the paths, the algorithm starts its work on normalizing the confidences. To normalize the confidences, the confidence intervals are mapped to a specified range. The standard range in this procedure is $[0 + \varepsilon, 1 - \varepsilon]$ for a small ε. The small value ε is due to the *Layer Combination* mechanism. During the introduction of the *Layer Combination* it will be explained in more detail what this value is about. In the following it will be considered as 0 for simplicity.

Thus, the idea is to map any range of values that can be taken by the confidence of an agent, to the probability range $[0, 1]$. The *Normalized Aggregate* algorithm performs this transformation by a linear mapping from the original interval to the destination interval. To create this mapping, the algorithm determines the maximum and minimum values of the hypotheses per layer of all paths (line 6-8). This information is enough to establish the mapping. In detail, the maximum and minimum values are used to calculate the actual value in the interval $[0, 1]$ as follows:

$$v \mapsto \frac{v - min}{max - min}$$

As you can see that transformation ensures that any confidence value v is mapped to the target interval. For simplicity, a linear mapping was used in the algorithm. In general, however, the idea of the algorithm does not depend on it.

After determining, the maximum and minimum confidence values in each layer, the algorithm initializes the result list. The goal of the algorithm is to return a list of floating-point values that represents the rating of each path of the input.

The actual rating of a path consists of three steps: First the confidences of the hypotheses in the path are normalized using the stored minimum and maximum values of each layer. After normalizing, the *Layer Entry Evaluation* and *Layer Combination* are used to determine the score of the path. A *Layer Entry Evaluation* function provides a score for the hypotheses per layer. Therefore, the result of the *Layer Entry Evaluation* is a list of floating-point values that has the same length as the path. The actual functions to generate these values will be discussed in one of the following paragraphs. To obtain the final rating of the path a *Layer Combination* function combines the scores provided by the *Layer Entry Evaluation*. Some of these functions are discussed in one of the later paragraphs. Finally, the score of the path is added to the result list. After all paths are rated, the algorithm returns the score of each path.

Layer Entry Evaluation As already introduced above, a *Layer Entry Evaluation* refers to a function that takes the normalized confidences of the hypotheses of one layer. Thus, the function provides a rating for exactly one layer. In Table 7.1 you can see some functions that are considered for the *Layer Entry Evaluation*. A layer entry can consider multiple

Name	**Description**
AVERAGE	Build the **average** score of all normalized hypotheses in the layer entry.
MEDIAN	Build the **median** score of all normalized hypotheses in the layer entry.
MAXIMUM	Use the **maximum** score of all normalized hypotheses in the entry.
MINIMUM	Use the **minimum** score of all normalized hypotheses in the entry.
SIGMA	Use the **standard derivation** of all normalized hypotheses in the entry.

Table 7.1.: An overview on functions for Layer Entry Evaluation.

hypotheses. Therefore, a combination of the different confidences is necessary. If you consider WSD for example, you get a selected hypothesis for each disambiguated word. If you consider Topic Detection, you get a hypothesis for each selected topic. The first option in the table takes the normalized confidences of a layer entry and calculates their average value as score. The idea of this function is to consider a layer as good (in terms of confidence) if it is good on average. Instead of the average, the second function uses the median value. Although the idea is the same, the median is more robust against outliers. The next two functions only consider the extreme values of the confidence values in a layer. However, because the maximum and minimum of all normalized confidence values within an entry on the path are considered once, outliers are weighted significantly. Of both functions the minimum function is more interesting, because a good value (value near 1) here has a potentially strong importance: If the minimum value represents a good rating, all hypotheses in the current layer entry have a good normalized confidence. The last function differs slightly from the previous ones. Instead of combining the value of the confidence directly, the last function determines the standard deviation of the normalized confidence. In this context, the idea is that potentially the variance of the normalized values of the hypotheses of a layer entry provides an indication of the quality of the hypotheses. What kind of such a relation could be, will be part of the evaluation later.

Layer Combination After generating the ratings for the layer entries of a path, the *Layer Combination* function combines the ratings to a single rating for the whole path. In Table 7.2 you can see the desired functions to combine the scores to one score. The first

Name	Description
ADD	**Add** Layer Entry Evaluation Scores as final score of a path.
MULTIPLY	**Multiply** Layer Entry Evaluation Scores as final score of a path.

Table 7.2.: An overview on functions for Layer Combination.

possibility to combine the ratings is to sum up the ratings of a path. The idea behind adding the scores is that the different layers have equal priority on the final score. Finally, the score for paths of length l then lies somewhere in $[0, l]$. The second alternative combines the scores of the layer entries using multiplication. The idea here is that the individual values can influence the final result more. Since the individual scores of the entries are between zero and one, a score will be immediately at most low as soon as one of the entries is scored zero. To mitigate this fact, a small epsilon was introduced during normalization. This was already mentioned above. This means that the final value range for paths of length l lies in $[(0+\varepsilon)^l, (1-\varepsilon)^l]$.

7.2. Agent Analysis Evaluator

The last section of this chapter deals with the so-called *Agent Analysis Evaluator*. This tool provides an interface for the classification of hypotheses for evaluation. Furthermore, the evaluator is responsible for storing information about the classified hypotheses. Figure 7.1 shows the model for the classification of hypotheses. As you can see, the `EvaluationData` contains the actual classification of the hypotheses. Therefore, the instance holds multiple maps that map a specific `Classification` to multiple `Hypotheses`. Each map refers to a specific layer in the exploration result that is currently evaluated. The actual map contains a list of hypotheses for each type of classification. This classification data can be

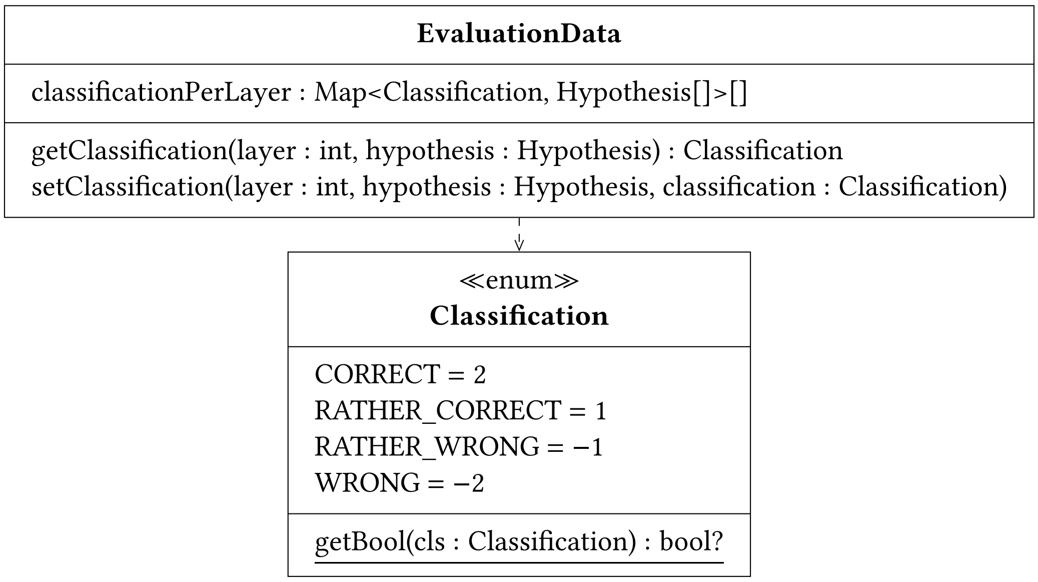

EvaluationData
classificationPerLayer : Map<Classification, Hypothesis[]>[]
getClassification(layer : int, hypothesis : Hypothesis) : Classification
setClassification(layer : int, hypothesis : Hypothesis, classification : Classification)
≪enum≫
Classification
CORRECT = 2
RATHER_CORRECT = 1
RATHER_WRONG = −1
WRONG = −2
getBool(cls : Classification) : bool?

Figure 7.1.: Implementation: Classification of Agent Analysis Evaluator

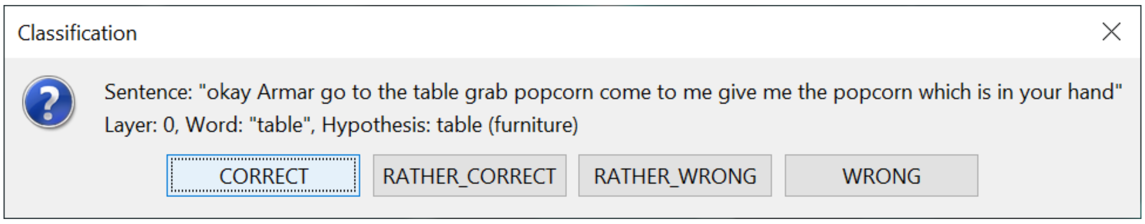
accessed by supplying the number of the current layer and the hypothesis from the current layer that shall be classified. To perform a new classification you have to set the layer, the hypothesis, and its actual classification. Currently, four classifications are possible: `CORRECT`, `RATHER_CORRECT`, `RATHER_WRONG`, and `WRONG`. Each type within the classifications is assigned a unique integer value. This value is used later in the evaluation to combine different `EvaluationData`.

The actual meaning of a classification is configurable. Decisive for the later metrics in the evaluation is whether the hypothesis is considered *correct* or *incorrect*. How this mapping to correct or incorrect is performed for the existing types in the classification is adjustable. The mapping is accessible via `getBool(..)`. This function has three possible values as return: `true`, `false`, and `null`. Where `true` means, that this kind includes correct hypotheses, `false` means, that this kind includes false hypotheses and `null` means, that this kind includes neither correct nor incorrect hypotheses. With this mapping, metrics can then be used in the evaluation that require a binary classification of the hypotheses. As in the previous chapters, the `EvaluationData` was designed to be stored as JSON. Therefore, the model does not contain any cyclic dependencies.

For the actual evaluation, the *Agent Analysis Evaluator* is designed to handle the classification of Exploration Results that are stored as JSON files (cf. Figure 6.10). As it is possible to explore a sentence multiple times, the evaluator needs the ability to handle multiple exploration result files of the same sentence. Therefore, you can supply a single exploration result as file or a folder that contains multiple exploration results as JSON.

First, the evaluator checks whether `EvaluationData` already exists. In this case, the evaluator loads the respective file and extends the data if necessary. Otherwise, the evaluator creates a new empty `EvaluationData` to start the classification.

The classification itself consists of multiple classification requests for a human. You can see such a request in Figure 7.1. As shown in the picture, the evaluator provides the


Classification

Sentence: "okay Armar go to the table grab popcorn come to me give me the popcorn which is in your hand"
Layer: 0, Word: "table", Hypothesis: table (furniture)

Figure 7.2.: Agent Analysis Evaluator — Request for Classification

sentence for the current classification, the layer of the hypotheses, and the actual value of the hypothesis. In addition, the respective word of the hypothesis is shown if a word is present. The user of the evaluator has to classify the current hypothesis by selecting the desired `Classification` with the buttons. After classification of all hypotheses in the exploration results, the evaluator stores the `EvaluationData`. This completes the classification process.

8. Evaluation

In this chapter, the individual components of the work are evaluated. The first step is the creation of a GQM plan based on the research questions. After that, the different experiments or experimental runs are introduced. Subsequently, a gold standard is created, which is needed for the evaluation of the experiments. After the results have been evaluated, threats to validity are discussed.

8.1. GQM Plan

The evaluation of the thesis reflects the research questions stated at the beginning of this thesis in Chapter 1. Before the definition of goals and questions of the GQM plan, I introduce the metrics that will be used. Nevertheless, a summary of the GQM Plan can be found in Table 8.1. The more detailed information can be found in the following paragraphs.

Goal G1: *Improvement of the result of agents*	
Question 1.A:	Are new correct results found?
→ Metric:	Recall of "normal" results
Question 1.B:	Do paths exist that show better results than the original variant?
→ Metric:	Precision, Recall, F1 score of the best paths
Goal G2: *Handling of the huge search space by partial explorations*	
Question 2:	How many correct hypotheses are found via partial exploration?
→ Metric:	Comparison of Recalls of partial explorations
Goal G3: *Identification of the good paths*	
Question 3:	Are there rating functions that predict the correct paths well?
→ Metric:	Normalized Rankings of the best Paths
Goal G4: *Applicability of the approach*	
Question 4:	Does the approach improve the agents' results?
→ Metric:	Precision, Recall, F1 score compared to base line

Table 8.1.: Goals, Questions, & Metrics of the GQM Plan

Metrics for Classification Tasks The first metric to mention is the *Recall* (Equation (8.1)). The *Recall* defines the ratio of identified relevant elements (so-called true positive) and all relevant elements (including true positive and false negative) in a certain classification task. Thereby, a true-positive (tp) is defined as element that has been correctly identified as relevant. A false-negative (fn) is defined as element that has not been recognized as relevant, but should have been recognized.

$$Recall = R = \frac{tp}{tp + fn} \tag{8.1}$$

As this thesis deals with hypotheses and their exploration using selection providers (cf. Section 6.3), the relevant elements are correct hypotheses. The determination of the correctness of hypotheses are introduced in Chapter 7 as part of the *Agent Analysis Evaluator*. The actual determination by using a user study is discussed in Section 8.2. In general, the *Recall* metric needs the amount of missed elements (fn) for its calculation. In the context of the thesis this value is not necessarily calculable. For example, if you consider topic detection agent, many correct topics may still be unknown after exploration. Therefore, the experiment setup includes a configuration that performs an an almost full exploration. Thus, the search space should be approximated as detailed as possible. Instead of calculating all possible misses, the recall is then determined by using the detected misses.

The *Precision* defines the ratio of identified relevant elements and all selected elements for a given classification. A selected element is either a true positive (tp) or a false positive (fp). False positives are those elements that are wrongly assumed to be relevant. In Equation (8.2) the precision metric is defined.

$$Precision = P = \frac{tp}{tp + fp} \tag{8.2}$$

In terms of hypotheses exploration, the metric defines the proportion of correct hypotheses within a given path or selection.

A typical way to combine precision and recall is the F1 Score. As defined in Equation (8.3), the F1 Score is the harmonic mean of precision and recall.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{8.3}$$

Goals and Questions After defining the necessary metrics, the goals and questions of the GQM plan will be stated. As a reminder, the research questions are listed again below:

Research Questions

RQ1: What types of hypotheses are found in agents of an MAS for natural language?

RQ2: Can the results of agents be improved through partial exploration without changing the underlying mechanisms of the agents?

RQ3: Can correct hypotheses be identified after exploration?

The first research question is not measurable and is going to be discussed in Chapter 9. Nevertheless, the goals of the GQM plan are directly related to the second and third research question. Two goals of the GQM plan are related to the improvement of the results of agents (RQ2): First, the actual achievement of improvements (G1). Second, the handling of the huge search space by partial explorations (G2). A general improvement without control of the search space would be difficult to use in practice.

Regarding the improvement, the first question (Q1.A) to answer is whether the exploration process unveiled new correct hypotheses in contrast to the execution of the agents without exploration of the search space. To evaluate the improvement, you measure the recall for the execution for some sentences without exploration of the search space. The lower the recall of these executions in relation to exploration performed, the more sense the exploration potentially makes, as new correct hypotheses are discovered. The second question regarding the improvement of the results is whether the exploration leads to paths that are better than the original "normal" result of the agents (Q1.B). These results can be directly extracted from the paths (cf. Chapter 7). To compare the results (generated by the paths) to the original result for a given input, you can compare the precision, the recall, and the F1 score of the different layers. As different inputs lead to different values for precision and recall, the interesting value is the trend of this metrics.

After considering the questions regarding the first goal in the GQM plan, the next goal has to be considered. The handling of the large search space of hypotheses (G2) is performed by using selection providers that build only some combinations of hypotheses and not all combinations. The question to be answered is how many correct hypotheses are found in relation to a gold standard (Q2). Different configurations for exploring the search space are then examined (cf. Section 8.2). The metric used to measure the quality of different types of exploration is the *Recall*. It indicates how many correct hypotheses can be found compared to all hypotheses that have been found during all explorations.

After defining the goals for the second research question, the next goal is related to the third research question: The third goal of the GQM plan is the identification of the good paths of an exploration (G3). Rating functions have been introduced in Chapter 7 to identify these paths. The question is whether one of the defined Rating Functions is able to identify the best paths (Q3). A metric to measure the classification performed by the rating functions is a *Rank*. If you consider the best paths according to the F1 score, you can determine the rank of this paths in the ranking performed by rating functions. The lower the rank (if you assume the best is 0), the better the rating function has identified the good paths. Since the number of paths can vary depending on the input, a normalized rank of a Path p is defined below:

$$NormalizedRank(p) \overset{def}{=} \frac{Rank(p)}{\#(Paths)} \tag{8.4}$$

As you can see the normalized Rank sets the actual rank in relation to the amount of paths that have been rated. If you consider a normalized Rank of 0.05 for the best path that would mean that the best path can be found by considering only the top 5% of the ranking. The lower this normalized rank is, the better the rating function is performing in terms of detection of the best path.

The last goal of the GQM plan is the actual applicability of the approach (G4). The question that shall be answered is whether the overall approach can improve the results of agents (Q4). To answer this, the precision, recall, and F1 measure shall be compared to a base line experiment (No Exploration). Thus, the selection providers and rating function are fixed.

8.2. Experiments

After defining the goals, questions, and metrics for evaluation, the actual experiments are stated in this section. In a first step, the machine that has been used for the evaluation is specified. After that, the configuration of the used agents is described. At the end of this section, the four experimental runs are explained.

Evaluation Machine Specs All the following experiments are performed on the same machine. The operating system of the machine is *Microsoft Windows 10 Version 2004 (OS Build 19031.572)*. It operates on an *Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz* and uses *32 GB* RAM. As Java Virtual Machine (JVM), the computer uses the pre-compiled version of *OpenJDK 14.0.1 x64*. The JVM has been allowed to use up to *30 GB* of the RAM (cf. `-Xmx30g`).

Configuration of the Agents For evaluation, the following agents have been used: *Wiki WSD*, *Topic Detection*, and *Ontology Selection*. The realization of the hypothesis specifications for these agents has been discussed in Section 6.4. The only parameter that needs to be defined for the *Wiki WSD (WSD)* agent is the amount of senses of a word that is going to be transformed to a hypothesis. For this experiment this value is set to 5. Considering the *Topic Detection (TD)*, the number of topics to be used as hypotheses has to be defined here. For the experiments, this value is set to 10. For the *Ontology Selection (OS)* the configuration is somewhat more complex. First the number of maximum hypotheses is limited to 10, as in the TD. Since less than ten ontologies (per ontology type) are loaded, this ensures that the number of ontologies is not limited. The value 10 was chosen because the OS agent uses an internal instance of the TD agent that should be initialized exactly like the instance for the experiment. The second part of the OS agent configuration concerns the loaded ontologies. You can find the used ontologies in the work of Keim [14]. The following *actor ontologies* were loaded: *robot.owl, virtual_assistant.owl, drone.owl, lego_mindstorms.owl*. In addition, the following *environmental ontologies* have been loaded: *kitchen.owl, bedroom.owl, bar.owl, laundry.owl, garden.owl, childrens_room.owl, heating.owl, music.owl*.

Experimental Runs There are four experimental runs. One of the runs is used to create the gold standard (cf. Section 8.3) and represents an almost full exploration. Furthermore, the data is used to answer the questions of the GQM plan in Section 8.4. All experimental runs use different selection providers to explore the exploration graph. The sentences for this evaluation originate from a evaluation by Keim [14]. In addition, they are listed in Appendix A.1.

The four experimental runs are used to gain some information about the actual performance of the exploration. Therefore, different levels of exploration are considered. Starting at completely no exploration and ending at an almost full exploration. As different amount of threads are used during the experimental runs, the runs are not completely comparable in terms of the actual run time. This will be discussed further in Section 8.4. In the following, the actual configuration of the different experimental runs are introduced.

Experimental Run 1: Base Line Experiment — No Exploration The first experimental run defines the base lines. Therefore, the original agents are executed and pseudo-hypotheses are stored. Thereby, one path per sentence is created. By default, the *Wiki WSD* agent annotates exactly one sense to each disambiguated word. The *Topic Detection* is configured to store the best three topics that have been found using the senses of the words of a sentence. Finally, the *Ontology Selection* agent selects multiple ontologies. The number of selected ontologies is not restricted.

Experimental Run 2: Gather Hypotheses — Full Exploration The second experimental run represents the most extensive exploration. This run is designed to find as many hypotheses as possible to create a suitable gold standard. For this second experimental run, the *WSD* agent uses the *Full Exploration Strategy* introduced in Chapter 6. The search space that originates from the hypotheses of the *WSD* agent is only restricted due to the use of the *Same-Word-Same-Decision Decorator*. The *Topic Detection* uses a *Top-X Sliding window*. This selection provider is configured to create a maximum of three combinations using three hypotheses (Topic Hypotheses). As the ontology selection is the last agent in the hierarchy, no specific selection provider needs to be applied. The agents themselves are configured like described in a previous paragraph. The goal of this exploration is to cover a large search space in order to make the recall metrics as accurate as possible. Due to the time required for this exploration, this exploration is not suitable for a regular run.

Experimental Run 3: Top-X Exploration After considering an almost full exploration, the third run restricts the exploration graph by using only Top-X candidates for the exploration. In comparison to the second experimental run, the only change is the use of another selection provider for the *Wiki WSD* agent: The *Wiki WSD* agent uses a *Top-X Confidence Strategy* that is decorated with the *Same-Word-Same-Decision Decorator*. The configuration of the *Top-X Confidence Strategy* defines a maximum of ten selections. This value is chosen to regard the amount of hypotheses per word, which is also set to ten. The configurations for the *Topic Detection* and *Ontology Selection* stay the same.

Experimental Run 4: Random Exploration The fourth experimental run uses randomness for the exploration of the exploration graph. Therefore, the *Wiki WSD* agent uses a *Random Hypothesis Selection Provider* that is again decorated with the *Same-Word-Same-Decision Decorator*. The restriction performed by the selection provider is set as follows: In total, a maximum of 250 paths can be created by *Wiki WSD*. In addition, the number of paths may not exceed 20% of the paths of the *Full Exploration*. To ensure reproducibility, the seed of the randomness generator is fixed. The other agents are configured as before.

8.3. Definition of the Gold Standard

In the previous sections, the GQM Plan and experiments that shall be performed, are introduced. All metrics that are considered have the need of a classification of hypotheses regarding the correctness of them. The acquiring of the gold standard has been performed after the actual execution of the explorations. Therefore, all generated hypotheses are known. The classification of the hypotheses has been realized by a user study. Two points are therefore important for the creation of the gold standard: Firstly, the conducting of the user study. Secondly, the creation of the gold standard by using the results of the user study.

The User Study The whole study took place virtually. In the appendix you will find questionnaires that had to be filled out by the participants (cf. Appendix A.2). The user study was performed by three masters students of computer science respectively media informatics. At the beginning, they had to fill in a short form about themselves. The actual results of this questionnaire is also available in the appendix (cf. Appendix A.3).

After the questionnaire has been answered, a participant had to download an archive file that contains all the necessary data for the evaluation. Directly after unpacking, the idea of hypotheses, layers, and the three agents were explained to the participants. This was made possible by a two-page introduction with text and illustrations. This introduction can be found in the appendix (cf. Appendix A.4). The instructions also explains how to use the *Agent Analysis Evaluator*.

After they have read the instructions, the evaluation actually starts. The participants have started the classification process for all hypotheses by using the *Agent Analysis Evaluator*. The evaluation has ended after the participants have submitted an archive containing the data of the evaluator.

The Aggregation of the Data The user study provided a unique classification of all hypotheses generated for the sentences for each participant. These classification data are represented as `EvaluationData` objects as introduced in Chapter 7. The goal of aggregating the results is to create exactly one gold standard for each sentence. For this purpose the following scheme has been used used to combine the classifications:

1. If two or more participants have classified a hypothesis equally, this classification is used for the hypothesis.
2. Otherwise, add the values of the classifications of a hypothesis and use the classification represented by the sum as the gold standard.

The creation of the gold standard thus considers majorities. If no majorities exist, trends are considered. Two examples show how the schema works: Assume that the following classifications are available for a hypothesis: 2x *CORRECT*, 1x *RATHER_CORRECT*. In this case, the first rule of the schema applies and the final classification is *CORRECT*. As a second example it is assumed that a hypothesis is classified as both *CORRECT* and *RATHER_CORRECT*, as well as *RATHER_WRONG*. The final classification has to be calculated by adding the values of these classifications (cf. Figure 7.1). The calculation would be $2 + 1 + (-1) = 2$, which represents *CORRECT*. It is important to note the calculation takes all opinions into account. If the last classification would be *WRONG* instead of *RATHER_WRONG* the result changes to *RATHER_CORRECT* because $2 + 1 + (-2) = 1$. This schema allows the creation of exactly one classification for all hypotheses of the sentences. Therefore, explorations can be evaluated in the next step.

8.4. Experimental Results

The preceding sections have defined the metrics, goals, and questions that shall be answered due to this evaluation. Furthermore, the different experiments or experimental runs have been introduced. This section follows the GQM plan and works through the goals and questions of the GQM plan one by one. However, before the GQM plan is elaborated, some general data on the experiments are first collected.

8.4.1. Statistical Data of the Exploration Experiments

The experimental setup allows to gain some statistical data about characteristic values like the number of paths of an exploration and the duration of the exploration process. A summary of the gathered values is shown in Table 8.2.

Experiment	Sentences	Threads	# Path (∅)	Duration $\left(\frac{Duration}{Sentence}\right)$
No	All Sentences	} 1	1	} 13 *min* (16 *s*)
Full	Num Sentences	} 3	10 – 2250 (647)	} 23.3 *h* (39 *min*)
	alexa1.1 bedroom1.1	} 2	1875 375	} 7.3 *h* (3.7 *h*)
	garden1.1 heating1.1 heating2.1 music1.1	} 2	375 10 375 375	} 1.6 *h* (24 *min*)
	drone2.1 mindstorm1.1 childrensroom1.1	} 2	9375 3000 9375	} 31.3 *h* (10.4 *h*)
	if4.1	} 1	9375	} 20.6 *h* (20.6 *h*)
	if5.1 bar1.1 drone1.1	} 1	≈ 84375 ≈ 46875 ≈ 328125	} –
Top-X	All Sentences	} 3	10 – 16 (15)	} 73 *min* (89 *s*)
Random	All Sentences	} 3	2 – 750 (213)	} 10.3 *h* (13 *min*)

Table 8.2.: Statistical Data for the Experimental Runs

In the first data line of the table you can see the statistical data of the base line experiment (No Exploration). As you can see, the experiment has been performed sequentially and produces exactly one path per sentence. The whole experimental run took 13 minutes for all 49 sentences, which equals 16 seconds per sentence.

The next groups of rows in the table represent the full exploration experimental run. To speed up the exploration process, multiple sentences have been analyzed in parallel using multiple threads. During the experiment a limiting factor has been identified: The available RAM. As the full exploration experiment creates a huge amount of paths, multiple instances of PARSE Graphs are created. Thus, the limit of 30 GB has been reached quickly. To prevent a memory overflow, the sentences have been grouped together. This is shown by the different blocks of rows that you can see in the table.

For example, you see that the full exploration experimental run for the sentences whose identifier consists of a number ("Num Sentences"), has been calculated on three cores in parallel (cf. data row 2). Furthermore, it is stated that the amount of paths for this group lies within 10 and 2250 paths per sentence. The average amount of paths that were generated in that group of sentences is 647. A total of 23.3 hours were needed for the exploration of these sentences.

One group of sentences is particularly important: The sentences that could not be explored with the full exploration strategy. This group consists of *if5.1*, *bar1.1*, and *drone1.1*. The exploration of these sentences were not possible on the machine specified above, as the memory has been exceeded during the exploration. For each of the sentences, the 30 GB of memory were not enough to hold the tree that was built up along with the PARSE Graphs. The number of paths was estimated by the amount of branches for the WSD layer and the configuration of the selection provider of Topic Detection. As you can see, the number of sentences in the comparison to the other sentences differs strongly.

The time for exploration can be reduced by restricting the exploration. You can see this for the two remaining experimental runs in the two last data rows. The best results in terms of time were thus achieved for the Top-X configuration, since on average the smallest number of paths has been explored.

In summary, the whole experiment had a computation time of approximately four days. Especially the memory consumption, which restricted the parallel execution of agents on the machine, was responsible for the overall long runtimes of most groups. In addition, the paths for a sentence are always only explored sequentially. For paths as well as for the three sentences that could not be fully explored, it should be checked in particular whether the exploration itself can parallelized. This is especially interesting from the point of view of the INDIRECT Project [12], because one idea at INDIRECT is to load whole documents and not just single sentences.

8.4.2. Results regarding the Improvement of Agents

The first goal of the GQM plan is the improvement of agents by using the exploration of hypotheses. Therefore, two questions have been formulated. The first one is whether new correct results can be found. This leads to the question of what correct and incorrect hypotheses are in this evaluation. For this evaluation, the correct hypotheses are those hypotheses that are classified as *CORRECT* according to the gold standard. Incorrect hypotheses are those that are classified as *WRONG*. The hypotheses that are classified as *RATHER_CORRECT* or *RATHER_WRONG* are not taken into account to calculate the classification metrics. This decision is based on the fact that in order to use these hypotheses, you would have to check the individual hypotheses; you would have to decide whether a hypothesis is really correct or false. Especially, the aggregation might have to be reconsidered.

Recall for No Exploration The first question is measured by the recall of the first experimental run (No Exploration). An excerpt of the precision, recall and F1 score for the base line experimental run is shown in Table 8.3. The whole table is available in the appendix (cf. Appendix A.5.1).

Sentence	**WSD**			**TD**			**OS**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
1.1	1.00	**1.00**	1.00	0.67	**0.40**	0.50	1.00	**0.67**	0.80
2.1	1.00	**1.00**	1.00	0.33	**0.33**	0.33	1.00	**1.00**	1.00
3.1	1.00	**0.60**	0.75	0.00	**0.00**	—	1.00	**1.00**	1.00
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
alexa1.1	0.67	**0.40**	0.50	0.00	**0.00**	—	1.00	**0.33**	0.50
bar1.1	0.50	**0.50**	0.50	0.00	**0.00**	—	1.00	**1.00**	1.00
bedroom1.1	0.67	**0.50**	0.57	0.00	**0.00**	—	1.00	**1.00**	1.00
childrensroom1.1	0.40	**0.50**	0.44	0.00	**0.00**	—	0.00	**0.00**	—
drone1.1	0.71	**1.00**	0.83	0.50	**0.33**	0.40	0.50	**0.50**	0.50
drone2.1	0.00	**0.00**	—	0.00	—	—	0.00	**0.00**	—
garden1.1	0.67	**0.50**	0.57	0.00	**0.00**	—	1.00	**1.00**	1.00
heating1.1	1.00	**0.50**	0.67	0.50	**0.33**	0.40	0.50	**0.50**	0.50
heating2.1	1.00	**0.33**	0.50	1.00	**0.75**	0.86	0.50	**0.50**	0.50
if.4.1	0.50	**0.50**	0.50	0.00	**0.00**	—	1.00	**1.00**	1.00
if.5.1	0.80	**0.57**	0.67	0.00	**0.00**	—	1.00	**0.67**	0.80
mindstorm1.1	0.50	**1.00**	0.67	0.00	—	—	0.00	**0.00**	—
music1.1	0.67	**0.40**	0.50	0.00	—	—	1.00	**1.00**	1.00
Average	0.84	**0.80**	0.82	0.28	**0.26**	0.40	0.63	**0.84**	0.74

Table 8.3.: Precision, Recall, and F1 for No Exploration [A.5.1]

As you can see, the value of the recall (R) varies depending on the different agents. On average, you see that about 80% of all correct hypotheses of the Wiki WSD agent can be found without exploration. Nevertheless, especially the more complex sentences have lower values for the recall.

A different picture emerges by looking at the Topic Detection agent. On average the recall value for all sentences is 26%. This means that the exploration has identified 74% new correct topics compared to no exploration. If you look at the more complex sentences at the end of the table, you can see that many do not have a single correct topic. This can be concluded from the zero values for the recall of the respective sentences.

Regarding the ontology selection, the recall values are similar to ones for the Wiki WSD agent. Especially note that the selection of the hypotheses of the last layer does not exist, because no further agent follows. The consequences of this are discussed in the next paragraph. Nevertheless, the recall value indicates that 84% of the correct hypotheses are found without exploration.

In summary, the simple answer regarding finding new correct hypotheses with the help of exploration is: Yes, one finds new correct hypotheses. However, the amount of found hypotheses depends largely on the agent. With WSD you consider an agent, where actually only one WSD hypothesis out of a set of hypotheses should be correct. Here the gain of new correct hypotheses is rather small. With regard to TD, the largest increase can be observed. This is especially due to the fact that several hypotheses can be correct. Also, it can be seen that there are sentences, where no correct topics were found before. The OS agent is an agent whose hypotheses are known in advance (besides the confidences) and only the selection is interesting. This is because the ontologies are all known. As with the WSD agent, many hypotheses have already been found without exploration. Nevertheless, there are some sentences where no correct hypothesis has been selected without exploration.

Precision, Recall, and F1 score of the best paths The second question regarding the improvement of the results of agents is whether paths exist that represent better results than the results without exploration. To answer this question, three paths are selected for each sentence. The selected paths are shown in Table 8.4 (Full Table: Appendix A.5.2).

Sentence	**WSD**			**TD**			**OS**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
1.1 F1 WSD	1.00	1.00	1.00	0.67	0.40	0.50	1.00	0.67	0.80
1.1 F1 TD	1.00	1.00	1.00	1.00	0.60	0.75	1.00	0.67	0.80
1.1 F1 OS	1.00	1.00	1.00	0.67	0.40	0.50	1.00	0.67	0.80
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
alexa1.1 F1 TD	0.50	0.40	0.44	1.00	0.67	0.80	1.00	0.33	0.50
alexa1.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
bar1.1 F1 WSD	1.00	0.50	0.67	0.00	0.00	—	0.25	1.00	0.40
bar1.1 F1 TD	0.20	0.17	0.18	1.00	0.33	0.50	1.00	1.00	1.00
bar1.1 F1 OS	0.50	0.50	0.50	0.00	0.00	—	1.00	1.00	1.00
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 8.4.: Precision, Recall & F1 of the best Paths [A.5.2]

The first path of a sentence represents the best path regarding the F1 scores for the Wiki WSD hypotheses of a sentence. For example this would be *1.1 F1 WSD* for the sentence with identifier *1.1*. The second path represents the best path according to F1 score for the Topic Detection (TD). The third one is the best path if you look at the F1 score for Ontology Selection (OS). This selection of paths should serve as an indication whether the newly found hypotheses are also present in the paths.

In the context of the OS the question arises, how the hypotheses of the ontology selection agent can be interpreted as selection. This was already mentioned above and is now explained here. Within the context of this evaluation, the selection of these hypotheses takes place through the use of confidence. The OS agent creates two fixed sets of hypotheses per path. The first set contains the hypotheses regarding the actor ontologies. In the second set, the hypotheses for the environmental ontologies are contained. As in most of the cases only one ontology fits to a certain sentence (one for the actor and one for the environment), the evaluation uses the best rated hypothesis as selected hypothesis. In the case that there are multiple best hypotheses with the same confidence, all those hypotheses get selected.

In order to get insights into how these best sentences from Table 8.4 behave in relation to the base line experiment without exploration, the changes in precision, recall and F1 are considered below. In Table 8.5 you can see the these changes.

Metric	**WSD**		**TD**		**OS**	
	Δ Range	Δ∅	Δ Range	Δ∅	Δ Range	Δ∅
P	[±0.00,+0.50]	**0.03**	[±0.00,+1.00]	**0.51**	[±0.00,+1.00]	**0.29**
R	[±0.00,+1.00]	**0.04**	[-0.25,+1.00]	**0.34**	[±0.00,+1.00]	**0.08**
F1	[±0.00,+0.29]	**0.02**	[-0.19,+0.60]	**0.31**	[±0.00,+0.50]	**0.23**

Table 8.5.: Changes for Precision, Recall, F1 Score for the best Paths (cf. Table 8.4)

For each value whose change is observed, i.e. the precision, the recall, and the F1 score for each of the layers, an interval and an average are calculated. The interval describes the range of the difference of the three best paths of exploration to the one without exploration. This interval is designated Δ *Range* in the table. If you consider these intervals for the Wiki WSD agent, you can see that neither the precision, nor the recall, nor the F1 score are negatively influenced for these best sentences. In particular, the +1.00 at the recall of the WSD's Δ *Range* shows that there is at least one sentence that has a maximum improvement for the recall value. The average describes the mean of these differences for all 49 sentences. This average of the differences is designated Δ∅ in the table.

To summarize the changes in of the precision, recall, and F1 score for the first layer, you can see that all scores slightly increased for the WikiWSD agent. The increase regarding the metrics for the Topic Detection is substantially higher on average for the best paths. For the OS layer, you can also see that especially the precision and the F1 measurement are potentially greatly improved. In particular, the values for the best paths do not deteriorate compared to the paths without exploration. In summary, the picture that emerges is that the consequences of exploration are agent dependent. The best paths were considered according to the F1 measure of the agents' hypotheses. In all measurements, improvements could be achieved for all three agents on average. Especially the F1 score for Topic Detection, which increased by 31 percentage points on average, is worth mentioning. Likewise, the improvements of 23 percentage points in Ontology Selection are worth mentioning. Thus, exploration itself offers the potential to improve agent results, substantially.

8.4.3. Results regarding the Handling of the Search Space

After analyzing the improvement of the results in general, the next paragraphs deal with the second goal in the GQM plan: The handling of the search space. Regarding this second goal, one question has been formulated that gives insights: "How many correct hypotheses are found via partial exploration?" Therefore, the recalls for the two different exploration strategies will be compared. In Table 8.6 you can see the actual recalls by agent and per sentence. The full table is part of the appendix (cf. Appendix A.5.3).

	Top-X Recall			**Random Recall**		
Sentence	**WSD**	**TD**	**OS**	**WSD**	**TD**	**OS**
1.1	1.00	0.60	0.67	1.00	0.60	0.67
2.1	1.00	0.67	1.00	0.50	0.33	1.00
3.1	1.00	0.17	1.00	1.00	0.50	1.00
4.1	1.00	0.29	1.00	1.00	0.71	1.00
	⋮	⋮	⋮	⋮	⋮	⋮
bar1.1	1.00	0.11	1.00	1.00	0.78	1.00
	⋮	⋮	⋮	⋮	⋮	⋮
Average	**1.00**	**0.59**	**0.92**	**0.82**	**0.69**	**0.93**

Table 8.6.: Recall per Experiment per Sentence [A.5.3]

As you can see, two experimental runs are represented as column groups in the table. Furthermore, the data is divided into the recalls for each layer. An important point to mention here is that the recall was calculated by using the selected hypotheses in the paths of the explorations. It can be seen that by using Top-X exploration all the correct hypotheses of WSD have already been covered. In contrast, some correct hypotheses for WSD are lost through Random Exploration. It also shows that for the final layer there is almost no difference in the recall of the two strategies. Regarding the TD, the Random strategy selects more correct topics on a path.

In summary, it can be stated that the combination of the WSD hypotheses influences the correct TD hypotheses found. At the same time, they play a rather minor role in hitting the correct ontologies on average over all the sentences. The search space covered by Random Exploration covers more of the topics and ontologies that had to be found than the simple combination of the Top-X word senses. For the control of the search space the combination of the hypotheses is crucial. This is clearly visible, since the Top-X strategy considers each hypothesis at least once, but still finds less correct topics than Random Exploration.

8.4.4. Results regarding the Identification of Good Paths

In the previous sections, it was found that exploration finds new correct hypotheses, the paths can also represent better results than the original results, and that the combination of hypotheses is crucial. The third goal of the GQM plan is the identification of good paths.

The question that is asked in relation to this goal is whether one of the rating functions can predict correct paths. To answer this question, the Random Exploration Strategy is used as a basis, since it is the strategy that generated more paths than Top-X. In Section 8.4.2 the best paths according to F1 score for the different layers have already been presented. In order to answer the question about the detection capability of rating functions, the ranks of these paths are determined in the following. Furthermore, the range of values of ranks as well as the average rank over all sentences are determined. In addition, the average size of the groups that arise by the rating is determined. A group contains several paths that have the same rating according to the Rating Function. Ideally this size would be one, because then every path would be assessed differently. The following tables are shown as excerpts. The full tables are available in the appendix (cf. Appendix A.5.4 & A.5.5).

In Table 8.7 you can see the (absolute) ranks for the paths with the best F1 score for the Wiki WSD agent. The column *Method (LC-LEE)* defines the *Layer Combina-*

Best WSD Paths (F1)						
Method (LC-LEE)	**Rank**		**Group-Rank**		**Group Size Avg**	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	$[0, 711]$	33.57	$[0, 706]$	32.86	$[1.00, 1.14]$	1.01
ADD-SIGMA	$[0, 714]$	103.53	$[0, 714]$	103.53	$[1.00, 1.33]$	1.01
ADD-AVERAGE	$[0, 598]$	32.45	$[0, 598]$	32.45	$[1.00, 1.00]$	1.00
ADD-MAX	$[0, 511]$	46.84	$[0, 503]$	46.10	$[1.00, 1.32]$	1.01
ADD-MIN	$[0, 604]$	41.57	$[0, 592]$	40.35	$[1.00, 2.00]$	1.11
	⋮	⋮	⋮	⋮	⋮	⋮

Table 8.7.: Rank, Group-Rank, & Group Size Avg (Absolute Ranks, WSD) [A.5.4]

tion (LC) and *Layer Entry Evaluation (LEE)* that is used to rate the sentences. The ε of the `NormalizedAggegate` Rating Function is set to 10^{-8}. As you can see, Table 8.7 shows three characteristics: First, the rank of the best path. As already done before, the data has been summarized over all sentences. Therefore, the range of ranks and the average rank is provided. Second, the rank of the group that contains the best path is stated. Again, you can see the its range and its average group rank. The third column deals with the average group size generated by Rating Functions.

The first finding is that ranks can vary greatly depending on the method. If you look more closely at the results, you will see that the most accurate evaluations are produced on average by using *ADD-AVERAGE*. In particular, the average group size in the evaluation is constantly 1.00. Thus, this procedure differentiates the paths well. A problem in considering the absolute ranks is the number of paths per sentence. As the number varies from sentence to sentence, an average makes no sense. Thus, the Normalized Rank was already introduced in the GQM Plan. In the following only normalized ranks are considered.

In Table 8.8 you can see the normalized ranks for the best WSD paths. You can see that regarding the normalized ranks, *ADD-AVERAGE* still performs well. On an average, you have to consider only the best 11% of the rating to find the best WSD path. If one regards

Best WSD Paths (F1)						
Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0.00, 0.95]	0.12	[0.00, 0.95]	0.12	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.95]	0.41	[0.00, 0.95]	0.41	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.92]	0.11	[0.00, 0.92]	0.11	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.91]	0.16	[0.00, 0.90]	0.16	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.95]	0.16	[0.00, 0.95]	0.16	[1.00, 2.00]	1.11
	⋮	⋮	⋮	⋮	⋮	⋮

Table 8.8.: Rank, Group-Rank, & Group Size Avg (Normalized Ranks, WSD) [A.5.5]

the values of the ranks, in particular also the absolute ranks, it becomes clear that there are quite possible candidates for good rating functions for the Wiki WSD agent. At the same time you can also see that these functions should rather be used as another filter to limit the possible paths from an exploration. But even this restriction is helpful in the variety of paths that are created by the full exploration strategy.

In Table 8.9 you can see the normalized ranks for the best paths according to the F1 score at the TD agent's hypotheses. According to the ranks, the best performing rating

Best TD Paths (F1)						
Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0.00, 0.68]	0.12	[0.00, 0.68]	0.12	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.99]	0.43	[0.00, 0.99]	0.43	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.68]	0.12	[0.00, 0.68]	0.12	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.94]	0.18	[0.00, 0.94]	0.18	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.64]	0.12	[0.00, 0.64]	0.12	[1.00, 2.00]	1.11
	⋮	⋮	⋮	⋮	⋮	⋮

Table 8.9.: Rank, Group-Rank, & Group Size Avg (Normalized Ranks, TD) [A.5.5]

functions are *ADD-MEDIAN*, *ADD-AVERAGE*, and *ADD-MIN*. The size of groups does not change as I use the same sentences as already for the WSD. You can see that on average, you have to take the best 12% into account to find the best path according to the F1 score for TD. Special attention should be paid to the *ADD-AVERAGE* function. This was the most precise for the WSD paths. For the TD, the range of normalized ranks is between 0.00 and 0.68. This means that there was at least one sentence for which this procedure performed the rankings rather bad. It becomes apparent that the rating functions may have to be adapted to the properties of the hypotheses or the agent to be optimized. Especially when using the Layer Combination `MULTIPLY` there is no combination that reaches the values that the Layer Combinations with `ADD` have achieved.

After considering the best paths for the F1 scores of Wiki WSD and Topic Detection, the remaining paths belong to the Ontology Selection agent. The normalized ranks for these paths are shown in Table 8.10. As shown in the table, *ADD-MIN* represents a combination

Best OS Paths (F1)						
Method (LC-LEE)	**Rank**		**Group-Rank**		**Group Size Avg**	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0.00, 0.95]	0.16	[0.00, 0.95]	0.16	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.99]	0.44	[0.00, 0.99]	0.44	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.86]	0.18	[0.00, 0.86]	0.18	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.88]	0.28	[0.00, 0.88]	0.28	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.88]	0.14	[0.00, 0.87]	0.14	[1.00, 2.00]	1.11
	⋮	⋮	⋮	⋮	⋮	⋮

Table 8.10.: Rank, Group-Rank, & Group Size Avg (Normalized Ranks, OS) [A.5.5]

that performs quite well on the given data for the OS agent. Using the *ADD-MIN* rating function leads to a localization of the best paths within the first 14% on average. However, it must also be emphasized that this is the average over 49 sentences. As the upper limit of the range indicates, there were also sentences where the best path was scored rather bad by the rating function.

In summary, it must be said that the choice of the rating function depends on what you want to optimize. For optimizing the OS you would probably use *ADD-MIN*. This function also works for the TD. For the WSD layer you would rather use *ADD-AVERAGE*. Especially good is that all paths could be separated and only small groups of sentences are created by the rating functions. Therefore, the paths can be differentiated. The number, how many paths must be considered, are similar for each layer. On average, the best results were achieved here for the WSD with 11%, the TD with 12%, and the OS with 14%. Extending the search range to the second and third best paths improves the results of the selection. In this case you would find one of the best three paths of the WSD on average in the 2% range, the TD on average in the 4% range, and the OS on average in the 4% range of best rated paths (cf. Appendix A.5.6). Nevertheless, the values also show that a refinement of the rating functions is necessary for successful detection. The existing functions show potential for an initial filtering of the results. It should also be remembered that the paths of Random Exploration were used for calculation.

8.4.5. Results regarding the overall Approach

The final goal of the GQM Plan is that the overall approach is applicable and also improves results. Thus, the entire approach is evaluated, from exploration, to the application of rating functions, to the selection of the best path. From the previous evaluations it was already found out that the rating function *ADD-MIN* should be used for the optimization regarding the ontology selection. Therefore, this Rating Function is used. With regard to the runtimes, the Top-X strategy is used for exploration. This strategy is more realistic than the runtime of the Random Exploration because the runtime is much shorter. The results for the 49 paths thus selected are shown in Table 8.11.

Sentence	**WSD**			**TD**			**OS**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
1.1	1.00	1.00	1.00	0.67	0.40	0.50	1.00	0.67	0.80
2.1	1.00	1.00	1.00	1.00	0.67	0.80	1.00	1.00	1.00
3.1	1.00	0.60	0.75	0.33	0.17	0.22	1.00	1.00	1.00
4.1	1.00	1.00	1.00	0.67	0.29	0.40	0.50	1.00	0.67
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
garden1.1	0.67	0.50	0.57	0.33	0.13	0.18	1.00	1.00	1.00
heating1.1	—	0.00	—	1.00	0.33	0.50	0.33	0.50	0.40
heating2.1	1.00	0.33	0.50	0.67	0.50	0.57	1.00	1.00	1.00
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Average	0.82	0.77	0.82	**0.50**	**0.37**	**0.49**	**0.69**	**0.86**	**0.79**

Table 8.11.: Precision, Recall, and F1 for the overall Approach [A.5.7]

On average for all sentences, the table shows that precision and recall of the WSD has slightly decreased compared to the base line experiment. The F1 measurement has remained the same on average. In contrast to this, you can see that all three metrics have improved for both the Topic Detection and the Ontology Selection. For the TD, the F1 score increased by 9 percentage points. Regarding the OS, the F1 score increased by 5 percentage points. If you look at *garden1.1*, you can see that there are at least some correct topics in the selected path. No correct topic has been found before for this particular sentence. On average over all sentences, it can be said that the overall approach already contributes to an improvement in the results at this stage. At the same time, it must also be said that the potential that can be seen in Table 8.5 has not yet been exhausted.

8.5. Threats to Validity

This section discusses the threats to validity of the evaluation. These are mainly to be found in the creation of the gold standard and the realization of the user study.

Threats to Construct Validity In context of construct validity, one point to consider is to keep the experimenter effect as low as possible. The experimenter effect refers to influences that the experimenter has on the participants of a study. These influences should be minimized. To make this possible, the interaction with the participants was kept to a minimum. The influence of the experimenter was tried to minimize by using textual instructions as well as by supporting the actual classification by a program. However, due to the online format, the experimenter was always available for the participants and also regularly checked the current progress. Therefore, influences of the experimental effect on the study cannot be completely eliminated.

The construct validity is threatened by sequence effects. For example, the participants had to perform virtually the same task for each sentence. In addition, learning effects may have occurred because some sentences were similar. This is not critical for this study, since it was a first estimation of the suitability of the approach. However, if the study is repeated, one possibility to reduce sequence effects is the permutation of sentences. Furthermore, the hypotheses that have to be classified definitely as *WRONG* could be classified before the actual study.

Threats to External Validity With regard to the external validity of this user study, there are two main issues to consider: First, the selection of participants. The selection was not randomized. Also, only students were used for the study. In general, the question is who uses such an MAS. Since this question depends strongly on which agents are considered, it is not clear which group of people is a representative group. In the most cases, students are not the representative group that use a multi-agent system for natural language.

Second, the number of participants and agents (in this study: three each) that were used in the evaluation are an issue for external validity in this thesis. For general conclusions, more agents and their hypotheses have to be considered. Regarding the amount of participants, only three classifications were used to define a common gold standard. Instead of calculating the common gold standard, strategies such as group sessions would also be possible. Through such sessions, the participants could exchange their opinions and thus find a common consensus. For a first estimation and with only three participants this was not necessary yet.

Threats to Internal Validity The most crucial threat to internal validity of the study and thus to the gold standard is found in the maturation of the participants. The study took several hours ($\approx 3\ hours/participant$) in total and the task of classification did not change over this time. As a result, the task was perceived as tiring and monotonous. Such a study has to be shortened for later repetitions. One last point that was mentioned by one participant was the conception of the GUI. On some screens the text was small. Also, it was easy to click the wrong way because they clicked *WRONG* much more often. This imbalance resulted from the fact that the majority of hypotheses had to be classified as *WRONG*. The hypotheses were therefore not balanced from the classification point of view. If this study will be repeated, the GUI should be adapted. In addition, a pre-classification of the hypotheses that are surely wrong should be done in order to shorten the evaluation time.

9. Conclusion

This thesis addresses the problem of error propagation within results of agents of multi-agent systems (MAS) for natural language. For this purpose, the thesis introduces the concept of hypotheses and the exploration of the hypotheses' search space. The exploration of the search space opened up three main questions that were examined in this thesis: Firstly, which types of hypotheses can be distinguished in an MAS for NLP (RQ1). Secondly, whether a partial exploration offers the potential to improve the results of agents (RQ2). Finally, it had to be clarified whether the correct results can be identified from an exploration (RQ3).

The analysis of the existing agents of PARSE and INDIRECT has led to insights regarding existing hypotheses. It became clear that hypotheses are always combined to a set of hypotheses that defines certain properties of the contained hypotheses. The most important property regarding the search space is the number of correct hypotheses within such a set. In general, one can distinguish between sets of hypotheses that contain at most one correct hypothesis and sets that contain an arbitrary number of correct hypotheses. Of these two type of sets, the exploration for exactly one correct hypothesis is easier to perform. Another property of hypotheses sets is their *Range*. A range describes the entity to which hypotheses refer. The hypotheses can refer either to the whole input (cf. `SECTION`) or to a single word (cf. `NODE`). In this thesis, the property has not yet been used extensively, since the effects on the search space are unknown. An investigation how the range affects the search space of the hypotheses is a possible future work. The last property of hypotheses is the co-domain or range of values of their confidences. This work has shown that the range of values vary, depending on the agent. In order to be able to handle different co-domains, a normalization for confidences was introduced. Regarding the first research question (RQ1), three characteristics of hypotheses have been identified for classification: The number of maximum correct hypotheses of a set (cf. one or arbitrary), the entity to which the hypothesis refers (cf. *Node* or *Section*), and the co-domain of the confidence.

With the information about hypotheses of an MAS for NLP, the mechanisms for the exploration and assessment of hypotheses has been developed. For this purpose, specifications have been created in order to specify the dependencies between different agents. An algorithm has been developed on the basis of these specifications to examine the different layers in the search space of the hypotheses. In addition, a group of assessment functions has been defined based on the normalization of hypotheses: The *Normalized Aggregate Rating Functions.* In summary, these functions normalize the confidence of a layer, combine the normalized confidences of hypotheses within a certain layer entry, and determine the rating of a path by combining the assessment of its layer entries.

Three PARSE agents were used to evaluate the approach: Wikipedia Word-Sense Disambiguation (Wiki WSD), Topic Detection (TD), and Ontology Selection (OS). The evaluation was designed to answer the last two research questions. As a starting point, 49 already existing sentences were used for the evaluation of the original agents. These sentences were used to explore the hypotheses graph in different ways and finally extract found hypotheses. Four experimental runs were performed: First, a base line experiment has been executed in order to determine the hypotheses and results without exploration. Subsequently, an almost complete exploration was conducted to obtain a large yield of possible hypotheses. However, this exploration does not represent a realistically feasible strategy, as it took several days to process the sentences. In order to produce results in a more feasible manner, two further runs were performed: *Top-X Exploration* and the *Random Exploration*. Both limit the search space and thereby they shorten the runtime.

During the evaluation, several questions regarding the second and third research question have been answered. It was found that the recall of the agents without exploration was between 26% and 84%. This means that up to 74% of the correct hypotheses of the explored hypotheses space are not found without exploration. This value differed significantly between the agents. Subsequently, the best paths within the exploration were examined and compared to the results without exploration. It was found that the results of each agent got improved on average across all sentences. The improvement was particularly noticeable for Topic Detection (F1: +0.31) and Ontology Selection (F1: +0.23). Considering the recall for the two exploration strategies (*Top-X* and *Random*), no significant difference could be identified for the final agent (Ontology Selection). If one considers the runtimes for the two explorations, one can conclude that in practice, *Top-X Exploration* is the better choice for the agents used. The *Top-X Exploration* was about ten times faster than *Random Exploration*. Regarding the identification of good paths (RQ3), it has been shown that a certain subset of rating function generates a rather good classification of exploration paths. Nevertheless, the results indicate that a rating function is just a further step of filtering the paths (at least for the random exploration).

After evaluating the individual building blocks of the approach, the last step was to evaluate the entire approach. *Top-X Exploration* and a fixed *Rating Function* were used for this purpose. This evaluation has shown a consistently good F1 score for the WSD. In addition, the approach improved F1 scores for the Topic Detection (+0.09) and Ontology Selection (+0.05). Therefore, it can be stated that the approach also leads to an overall improvement in the results.

During the realization of this work several future works have emerged. The evaluation showed that the exploration has further potential. Therefore, especially further *Selection Providers* should be developed that explore the search space differently. In particular, multiple ideas arose: On the one hand, one can observe the development of confidences of certain hypotheses during the exploration process. Such observations could facilitate the measurement of the impact of individual hypotheses and selection decisions. On the other hand, the exploration could consider similarity of paths. For this purpose, effects caused by changing hypothesis selections should be investigated. In particular, cross-layer evaluation would be possible. Such an evaluation could be used to control the exploration process. You could either trace similar paths or as different paths as possible. However, it is unclear which paths should be explored. In addition, the model of hypotheses could consider which hypotheses were used to generate subsequent hypotheses. For *Topic Detection*, such a mechanism is already possible in the original code since the used senses are stored to the topics. Tracking the hypotheses could provide a better understanding of the processes involved in exploration. Thereby, the decisions would become more comprehensible. In summary, future work can primarily work on exploring the search space more effectively. Another point for future work is the creation of new assessment functions. In this thesis, *Rating Functions* were created that operate purely on the confidences of hypotheses. In the future, the values of the hypotheses should also be considered. Furthermore, the use of tracking of hypotheses mentioned above would also be conceivable. In this thesis, the effectiveness of exploration could be demonstrated. Therefore, further agents should be used in the future to evaluate the approach. Especially the relevance of agents that form hypotheses for a fixed selection, such as *Ontology Selection (OS)*, may be important for future work. Since the paths are evaluated only on the basis of their confidence, one should check whether agents like *OS* are needed for this.

Bibliography

[1] Enrique Alba, Gabriel Luque, and Lourdes Araujo. “Natural language tagging with genetic algorithms”. en. In: *Information Processing Letters* 100.5 (Dec. 2006), pp. 173–182. ISSN: 00200190. DOI: `10.1016/j.ipl.2006.07.002`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S002001900600216X` (visited on 10/22/2020).

[2] Alain Auger and Jean Roy. “Expression of Uncertainty in Linguistic Data”. en. In: *11th International Conference on Information Fusion* (2008), p. 8.

[3] Ghusoon Salim Basheer et al. “Certainty, trust and evidence: Towards an integrative model of confidence in multi-agent systems”. en. In: *Computers in Human Behavior* 45 (Apr. 2015), pp. 307–315. ISSN: 07475632. DOI: `10.1016/j.chb.2014.12.030`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S074756321400747X` (visited on 10/09/2020).

[4] Raphen Becker and Daniel D. Corkill. “Determining confidence when integrating contributions from multiple agents”. en. In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS ’07*. Honolulu, Hawaii: ACM Press, 2007, p. 1. ISBN: 978-81-904262-7-5. DOI: `10.1145/1329125.1329212`. URL: `http://portal.acm.org/citation.cfm?doid=1329125.1329212` (visited on 10/11/2020).

[5] Thomas A. Brown and Emir H. Shuford. *Quantifying Uncertainty into Numerical Probabilities for the Reporting of Intelligence*. Santa Monica, CA: RAND Corporation, 1973. URL: `https://www.rand.org/pubs/reports/R1185.html`.

[6] Lars Bungum and Björn Gambäck. “Evolutionary Algorithmsin Natural Language Processing”. In: *2nd Norwegian Artificial Intelligence Symposium* (2010). URL: `https://www.researchgate.net/publication/276885888_Proceedings_of_the_second_Norwegian_Artificial_Intelligence_Symposium` (visited on 10/22/2020).

[7] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. “Multi-Agent Systems: A Survey”. en. In: *IEEE Access* 6 (2018), pp. 28573–28593. ISSN: 2169-3536. DOI: `10.1109/ACCESS.2018.2831228`. URL: `https://ieeexplore.ieee.org/document/8352646/` (visited on 04/13/2020).

[8] Christiane Fellbaum, ed. *WordNet: an electronic lexical database*. Language, speech, and communication. Cambridge, Mass: MIT Press, 1998. ISBN: 978-0-262-06197-1.

[9] Peter Haase and Johanna Völker. “Ontology Learning and Reasoning — Dealing with Uncertainty and Inconsistency”. en. In: *Uncertainty Reasoning for the Semantic Web I*. Ed. by Paulo Cesar G. da Costa et al. Vol. 5327. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 366–384. DOI:

10.1007/978-3-540-89765-1_21. URL: http://link.springer.com/10.1007/978-3-540-89765-1_21 (visited on 09/07/2020).

[10] Theresa Heine. "Verknüpfung von Textelementen zu Softwarearchitektur-Modellen mit Hilfe von Synsets". Bachelor's Thesis. Karlsruher Institut für Technologie (KIT) – IPD Koziolek, Nov. 2019.

[11] Tobias Hey. "INDIRECT: Intent-Driven Requirements-to-Code Traceability". en. In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. Montreal, QC, Canada: IEEE, May 2019, pp. 190–191. ISBN: 978-1-72811-764-5. DOI: 10.1109/ICSE-Companion.2019.00078. URL: https://ieeexplore.ieee.org/document/8802673/ (visited on 08/17/2020).

[12] Tobias Hey. *INtent-DrIven REquirements-to-Code Traceability*. URL: https://code.ipd.kit.edu/hey/indirect.

[13] Tobias Hey. "Kontext- und Korreferenzanalyse für gesprochene Sprache". MA thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Sept. 2016. URL: https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/hey_ma.

[14] Jan Keim. "Themenextraktion zur Domänenauswahl für Programmierung in natürlicher Sprache". MA thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Feb. 2018. URL: https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/keim_ma.

[15] Marina Litvak, Mark Last, and Menahem Friedman. "A New Approach to Improving Multilingual Summarization Using a Genetic Algorithm". en. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (2010), p. 10.

[16] Christopher Manning et al. "The Stanford CoreNLP Natural Language Processing Toolkit". en. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, 2014, pp. 55–60. DOI: 10.3115/v1/P14-5010. URL: http://aclweb.org/anthology/P14-5010 (visited on 08/25/2020).

[17] Clara Meister, Tim Vieira, and Ryan Cotterell. "Best-First Beam Search". en. In: *arXiv:2007.03909 [cs]* (July 2020). arXiv: 2007.03909. URL: http://arxiv.org/abs/2007.03909 (visited on 10/12/2020).

[18] Andrea Moro, Alessandro Raganato, and Roberto Navigli. "Entity Linking meets Word Sense Disambiguation: a Unified Approach". en. In: *Transactions of the Association for Computational Linguistics* 2 (Dec. 2014), pp. 231–244. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00179. URL: https://www.mitpressjournals.org/doi/abs/10.1162/tacl_a_00179 (visited on 08/24/2020).

[19] Yue Ou. "Erkennung von Aktionen in gesprochener Sprache". Bachelor's Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, Sept. 2016. URL: https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/ou_ba.

[20] Sarvapali D. Ramchurn et al. “DEVISING A TRUST MODEL FOR MULTI-AGENT INTERACTIONS USING CONFIDENCE AND REPUTATION”. en. In: *Applied Artificial Intelligence* 18.9-10 (Oct. 2004), pp. 833–852. ISSN: 0883-9514, 1087-6545. DOI: 10.1080/08839510490509045. URL: http://www.tandfonline.com/doi/abs/10.1080/08839510490509045 (visited on 10/09/2020).

[21] V. Raskin and J. M. Taylor. “Fuzziness, uncertainty, vagueness, possibility, and probability in natural language”. en. In: *2014 IEEE Conference on Norbert Wiener in the 21st Century (21CW)*. Boston, MA, USA: IEEE, June 2014, pp. 1–6. ISBN: 978-1-4799-4562-7. DOI: 10.1109/NORBERT.2014.6893868. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6893868 (visited on 09/07/2020).

[22] Matthias Sperber et al. “Neural Lattice-to-Sequence Models for Uncertain Inputs”. en. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017, pp. 1380–1389. DOI: 10.18653/v1/D17-1145. URL: http://aclweb.org/anthology/D17-1145 (visited on 04/25/2019).

[23] Vanessa Steurer. “Strukturerkennung von Bedingungen in gesprochener Sprache”. de. Bachelor’s Thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, 2016. URL: https://code.ipd.kit.edu/weigelt/parse/wikis/Theses/steurer_ba.

[24] Vanessa Steurer. “Synthese von Methodendefinitionen aus natürlichsprachlichen Äußerungen”. MA thesis. Karlsruher Institut für Technologie (KIT) – IPD Tichy, June 2019. URL: code.ipd.kit.edu/weigelt/parse/wikis/Theses/steurer_ma.

[25] United States. President. “Working Together”. In: *Addresses of the President of the U.S. and the Director of the Bureau of the Budget*. 1922, 14 (342). URL: https://books.google.de/books?id=Wg5MAQAAIAAJ.

[26] Ashwin K. Vijayakumar et al. “Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models”. en. In: *arXiv:1610.02424 [cs]* (Oct. 2018). arXiv: 1610.02424. URL: http://arxiv.org/abs/1610.02424 (visited on 10/12/2020).

[27] S. Weigelt, T. Hey, and V. Steurer. “Detection of Conditionals in Spoken Utterances”. In: *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*. Jan. 2018, pp. 85–92. DOI: 10.1109/ICSC.2018.00021.

[28] Sebastian Weigelt. *P.A.R.S.E. - Programming ARchitecture for Spoken Explanations*. URL: https://code.ipd.kit.edu/weigelt/parse.

[29] Sebastian Weigelt, Tobias Hey, and Vanessa Steurer. “Detection of Control Structures in Spoken Utterances”. In: *International Journal of Semantic Computing* 12.03 (2018). _eprint: https://doi.org/10.1142/S1793351X18400159, pp. 335–360. DOI: 10.1142/S1793351X18400159. URL: https://doi.org/10.1142/S1793351X18400159.

[30] Sebastian Weigelt, Tobias Hey, and Walter F. Tichy. “Context Model Acquisition from Spoken Utterances”. en. In: *The 29th International Conference on Software Engineering & Knowledge Engineering*, Pittsburgh, PA, July 2017, pp. 201–206. DOI: 10.18293/SEKE2017-083.

[31] Sebastian Weigelt and Walter F. Tichy. “Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Florence, Italy: IEEE, May 2015, pp. 819–820. ISBN: 978-1-4799-1934-5. DOI: `10.1109/ICSE.2015.264`. URL: `http://ieeexplore.ieee.org/document/7203085/` (visited on 09/23/2018).

[32] Jin Yong Yoo et al. “Searching for a Search Method: Benchmarking Search Algorithms for Generating NLP Adversarial Examples”. en. In: *arXiv:2009.06368 [cs]* (Sept. 2020). arXiv: 2009.06368. URL: `http://arxiv.org/abs/2009.06368` (visited on 10/12/2020).

A. Appendix

A.1. Evaluation — Sentences

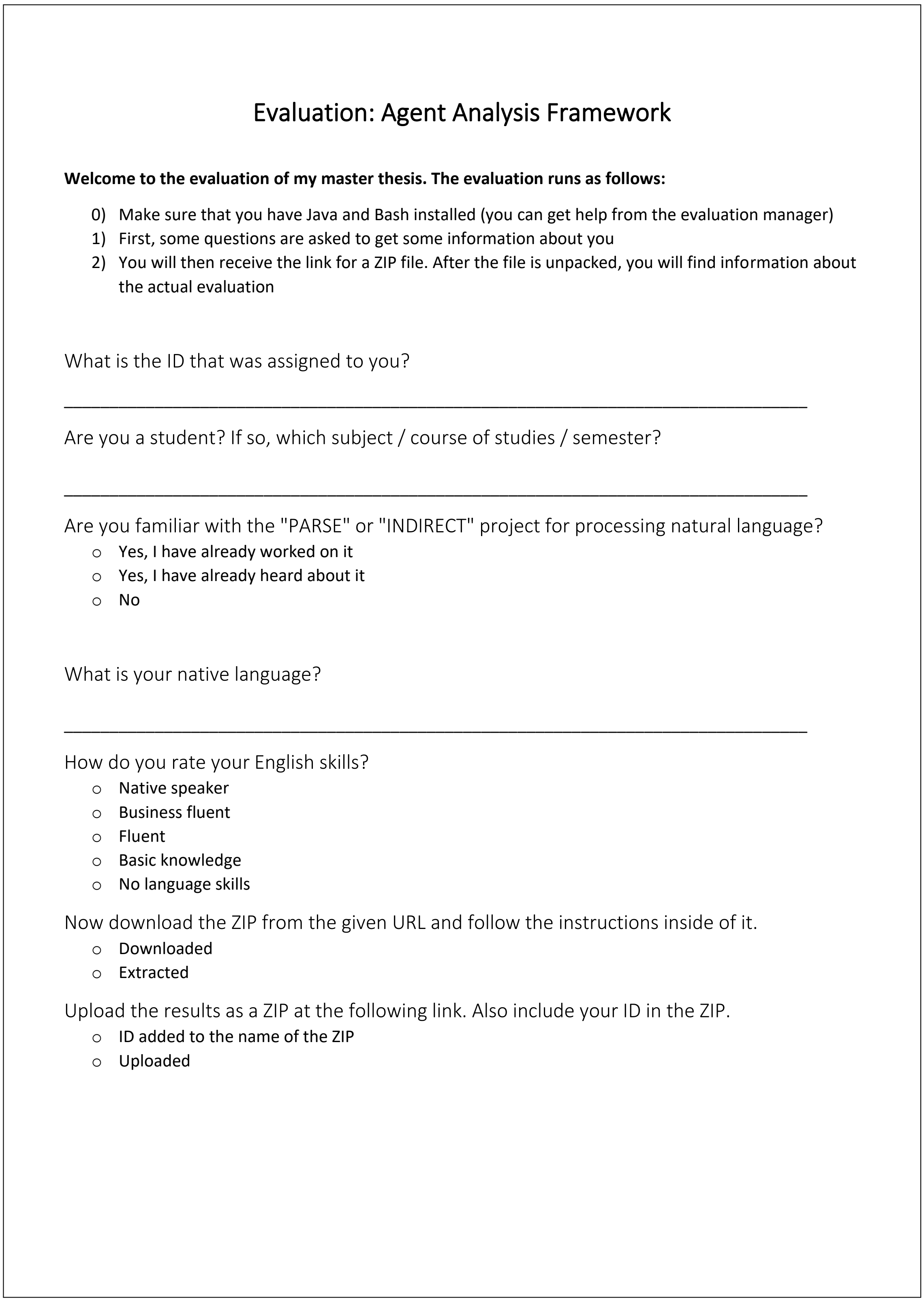
The sentences that are used for this evaluation (cf. Chapter 8). The sentences are taken from the work of Keim [14].

ID	Text
1.1	okay Armar go to the table grab popcorn come to me give me the popcorn which is in your hand
2.1	Armar can you please bring me the popcorn bag
3.1	Armar could you bring me the popcorn from the kitchen table please
4.1	Hey Armar can you go to the table and grab me the popcorn bag please and then yeah can you bring it back to me
5.1	Hey Armar go to kitchen table take the popcorn bag and bring it to me
6.1	Armar please bring me the popcorn bag from the table
7.1	Can you grab me the box of popcorn on the table it is next to the cereal box
8.1	Hey Armar bring me the popcorn bag from the table
9.1	Please go to the kitchen table and pick up the popcorn bag and bring it to me
10.1	Can you bring me the popcorn from the kitchen table
11.1	Hey Armar please go to the kitchen table on the right side and hand me the popcorn
12.1	okay go to the kitchen table ehm get the popcorn and bring it to me
13.1	go to the table in front of the window and take the popcorn and bring it to me
14.1	hey armar go to the table grab the popcorn and bring it to me
15.1	go to the table and grab the popcorn bag
16.1	Armar go to the table and give me the popcorn bag
17.1	Armar would you please pass me the popcorn
18.1	go to the table take the popcorn give it to me
19.1	can you please give me the popcorn it is uhm in front of you on the table
20.1	hey Armar go and get the popcorn bag for me please
21.1	can you bring me the popcorn from the table
22.1	hello Armar uhm I need some popcorn could you please go to the table grab the popcorn box turn to turn to me and then come back and hand it to me

23.1	Hey Armar i need some popcorn could you please bring me some popcorn thank you
24.1	Hello Armar bring me the popcorn bag from the table
25.1	Armar bring me the popcorn which is on the table please
26.1	Hello Robot bring me the popcorn
27.1	Hello Armar please bring me the popcorn from the table
28.1	Armar could you please bring me the popcorn which is on the left side of the green cup Thank you
29.1	Armar could you please bring me the popcorn it is on the black table between the cereals and the green cup
30.1	Hi Armar bring me the popcorn from the table
31.1	Hey Armar bring me the popcorn from the table
32.1	Hallo Armar bring me the popcorn bag from the table
33.1	Armar bring me the bag of popcorn from the table
34.1	Armar bring me the popcorn from the table
35.1	Hello Armar bring me the popcorn from the table He stands between Vitalis and cup Thank you very much
36.1	Hello Armar please bring me the popcorn from the table
alexa1.1	alexa turn up the temperature of the radiator by two degrees then start playing my favorite playlist
bar1.1	Go to the Fridge take the tonic water and mix it in the glass with the gin that is on the counter
bedroom1.1	go to the closet open it and grab the sweater and the trousers and bring them to me
childrens-room1.1	tidy up by grabbing the dolls and action figures and putting them on top of the cabinet
drone1.1	start and accelerate as fast as possible until you fly through the gate then slow down and turn left by sixty degree accelerate again and dodge the table by ascending first and descending afterwards fly through the greenhouse then turn left and fly above the pond if you crossed the pond break and descend down to the lawn and finally turn off
drone2.1	start and ascend to fifty meters then start the recording accelerate until you are over the target zone stop the recording and return
garden1.1	hey armar grab the lawn mower and use it to cut the grass
heating1.1	alexa turn on the radiator
heating2.1	go to the radiator and use the thermostat to increase the temperature
if.4.1	hey armar could you please have a look at these dishes if they are dirty put them into the dishwasher if they are not dirty put them into the cupboard
if.5.1	hey armar could you please open the fridge if there are fresh oranges bring them with the vodka else bring the orange juice and the vodka
mindstorm1.1	follow the line on the carpet at the end of the carpet turn until you see the rattle grab the rattle and afterwards release it again
music1.1	alexa please play my metal playlist in a random order

A.2. Evaluation — Questionnaire

Each participant in the evaluation completed the following questionnaire.



Evaluation: Agent Analysis Framework

Welcome to the evaluation of my master thesis. The evaluation runs as follows:

0) Make sure that you have Java and Bash installed (you can get help from the evaluation manager)
1) First, some questions are asked to get some information about you
2) You will then receive the link for a ZIP file. After the file is unpacked, you will find information about the actual evaluation

What is the ID that was assigned to you?

____

Are you a student? If so, which subject / course of studies / semester?

____

Are you familiar with the "PARSE" or "INDIRECT" project for processing natural language?

- o Yes, I have already worked on it
- o Yes, I have already heard about it
- o No

What is your native language?

____

How do you rate your English skills?

- o Native speaker
- o Business fluent
- o Fluent
- o Basic knowledge
- o No language skills

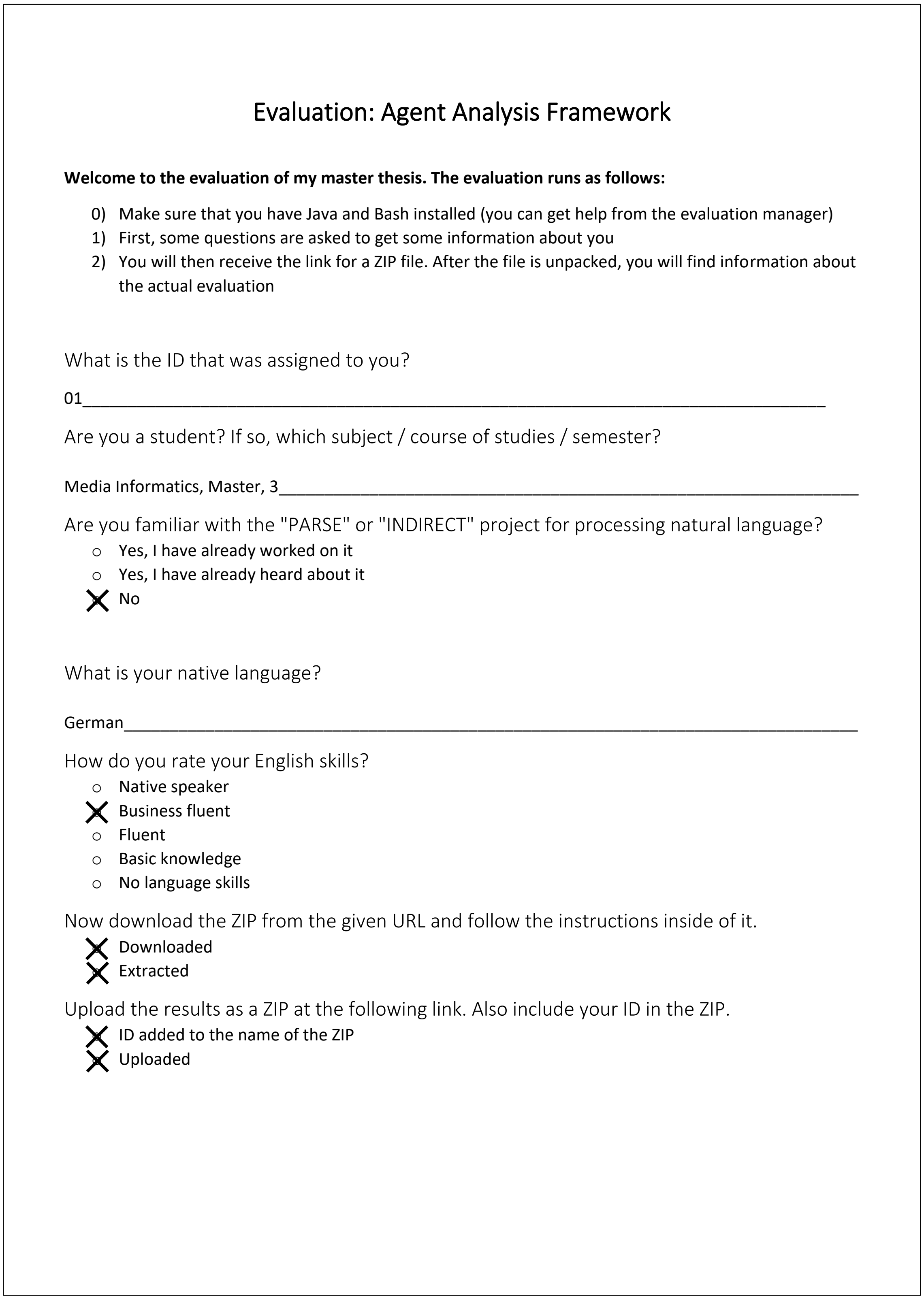
Now download the ZIP from the given URL and follow the instructions inside of it.

- o Downloaded
- o Extracted

Upload the results as a ZIP at the following link. Also include your ID in the ZIP.

- o ID added to the name of the ZIP
- o Uploaded

A.3. Evaluation — Results of Questionnaire



Evaluation: Agent Analysis Framework

Welcome to the evaluation of my master thesis. The evaluation runs as follows:

0) Make sure that you have Java and Bash installed (you can get help from the evaluation manager)
1) First, some questions are asked to get some information about you
2) You will then receive the link for a ZIP file. After the file is unpacked, you will find information about the actual evaluation

What is the ID that was assigned to you?

01

Are you a student? If so, which subject / course of studies / semester?

Media Informatics, Master, 3

Are you familiar with the "PARSE" or "INDIRECT" project for processing natural language?

- Yes, I have already worked on it
- Yes, I have already heard about it
- [✓ No]

What is your native language?

German

How do you rate your English skills?

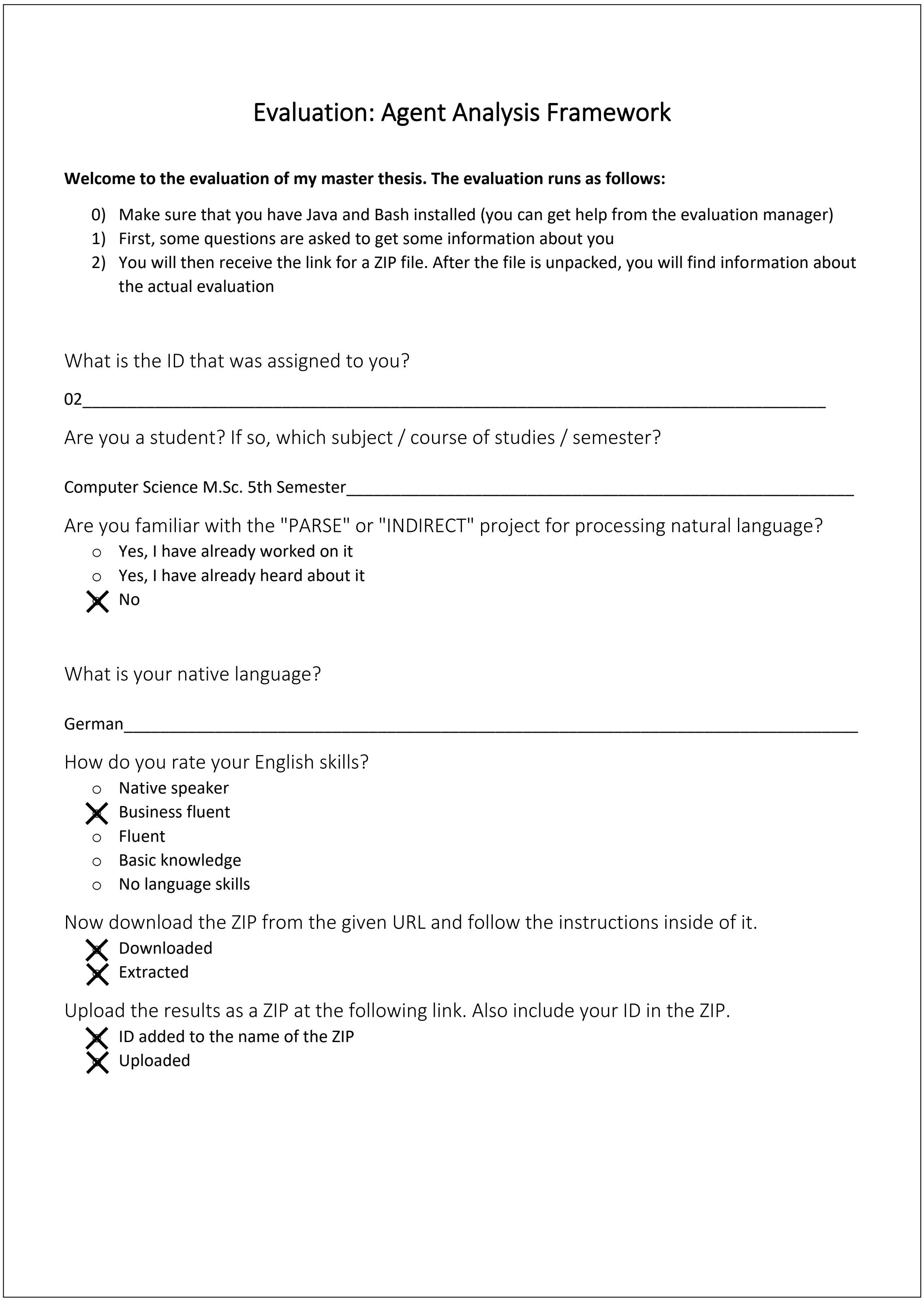
- Native speaker
- [✓ Business fluent]
- Fluent
- Basic knowledge
- No language skills

Now download the ZIP from the given URL and follow the instructions inside of it.

- [✓ Downloaded]
- [✓ Extracted]

Upload the results as a ZIP at the following link. Also include your ID in the ZIP.

- [✓ ID added to the name of the ZIP]
- [✓ Uploaded]



Evaluation: Agent Analysis Framework

Welcome to the evaluation of my master thesis. The evaluation runs as follows:

0) Make sure that you have Java and Bash installed (you can get help from the evaluation manager)
1) First, some questions are asked to get some information about you
2) You will then receive the link for a ZIP file. After the file is unpacked, you will find information about the actual evaluation

What is the ID that was assigned to you?

02

Are you a student? If so, which subject / course of studies / semester?

Computer Science M.Sc. 5th Semester

Are you familiar with the "PARSE" or "INDIRECT" project for processing natural language?

- Yes, I have already worked on it
- Yes, I have already heard about it
- [✓ No]

What is your native language?

German

How do you rate your English skills?

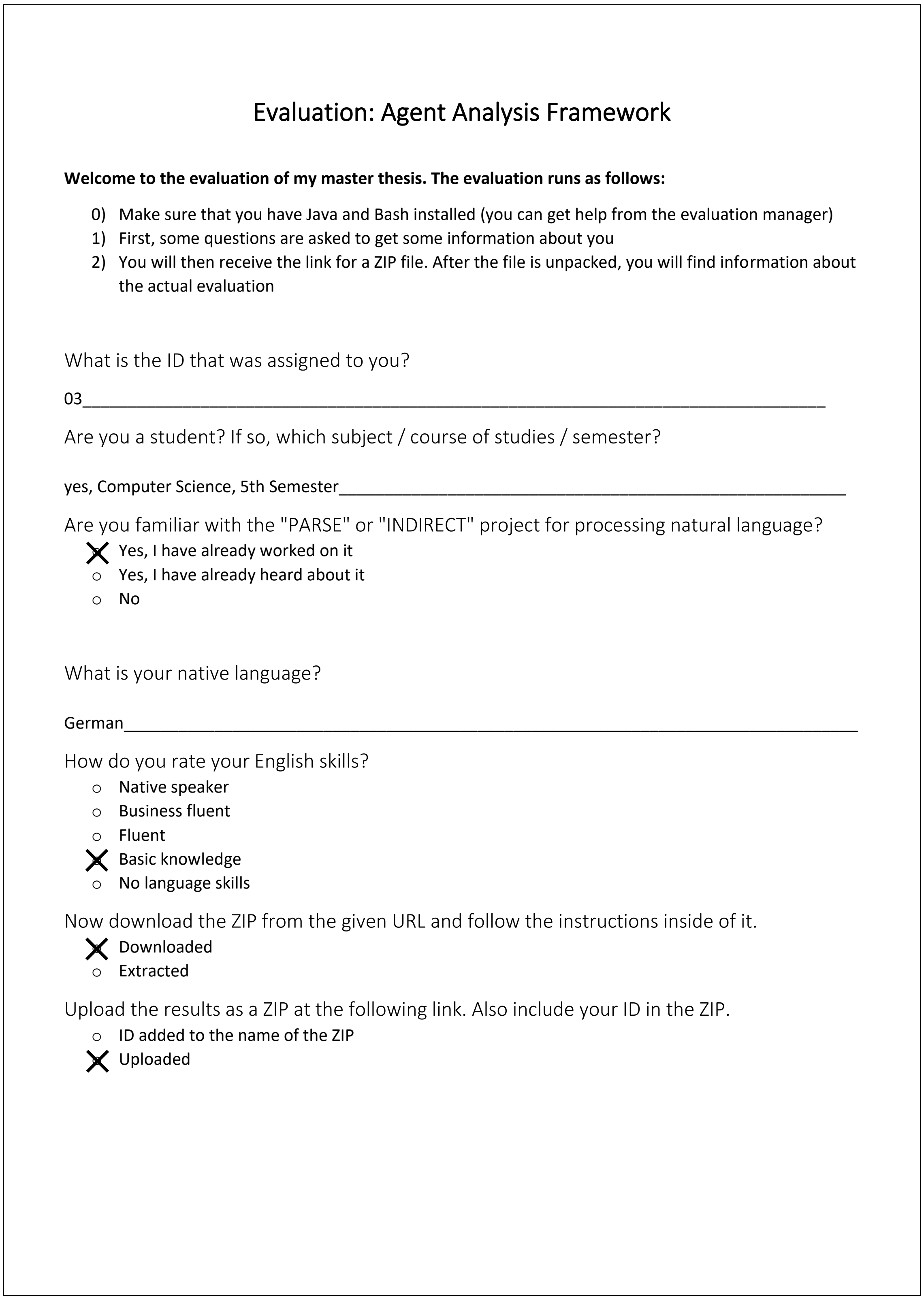
- Native speaker
- [✓ Business fluent]
- Fluent
- Basic knowledge
- No language skills

Now download the ZIP from the given URL and follow the instructions inside of it.

- [✓ Downloaded]
- [✓ Extracted]

Upload the results as a ZIP at the following link. Also include your ID in the ZIP.

- [✓ ID added to the name of the ZIP]
- [✓ Uploaded]



Evaluation: Agent Analysis Framework

Welcome to the evaluation of my master thesis. The evaluation runs as follows:

0) Make sure that you have Java and Bash installed (you can get help from the evaluation manager)
1) First, some questions are asked to get some information about you
2) You will then receive the link for a ZIP file. After the file is unpacked, you will find information about the actual evaluation

What is the ID that was assigned to you?

03

Are you a student? If so, which subject / course of studies / semester?

yes, Computer Science, 5th Semester

Are you familiar with the "PARSE" or "INDIRECT" project for processing natural language?

- [✓ Yes, I have already worked on it]
- Yes, I have already heard about it
- No

What is your native language?

German

How do you rate your English skills?

- Native speaker
- Business fluent
- Fluent
- [✓ Basic knowledge]
- No language skills

Now download the ZIP from the given URL and follow the instructions inside of it.

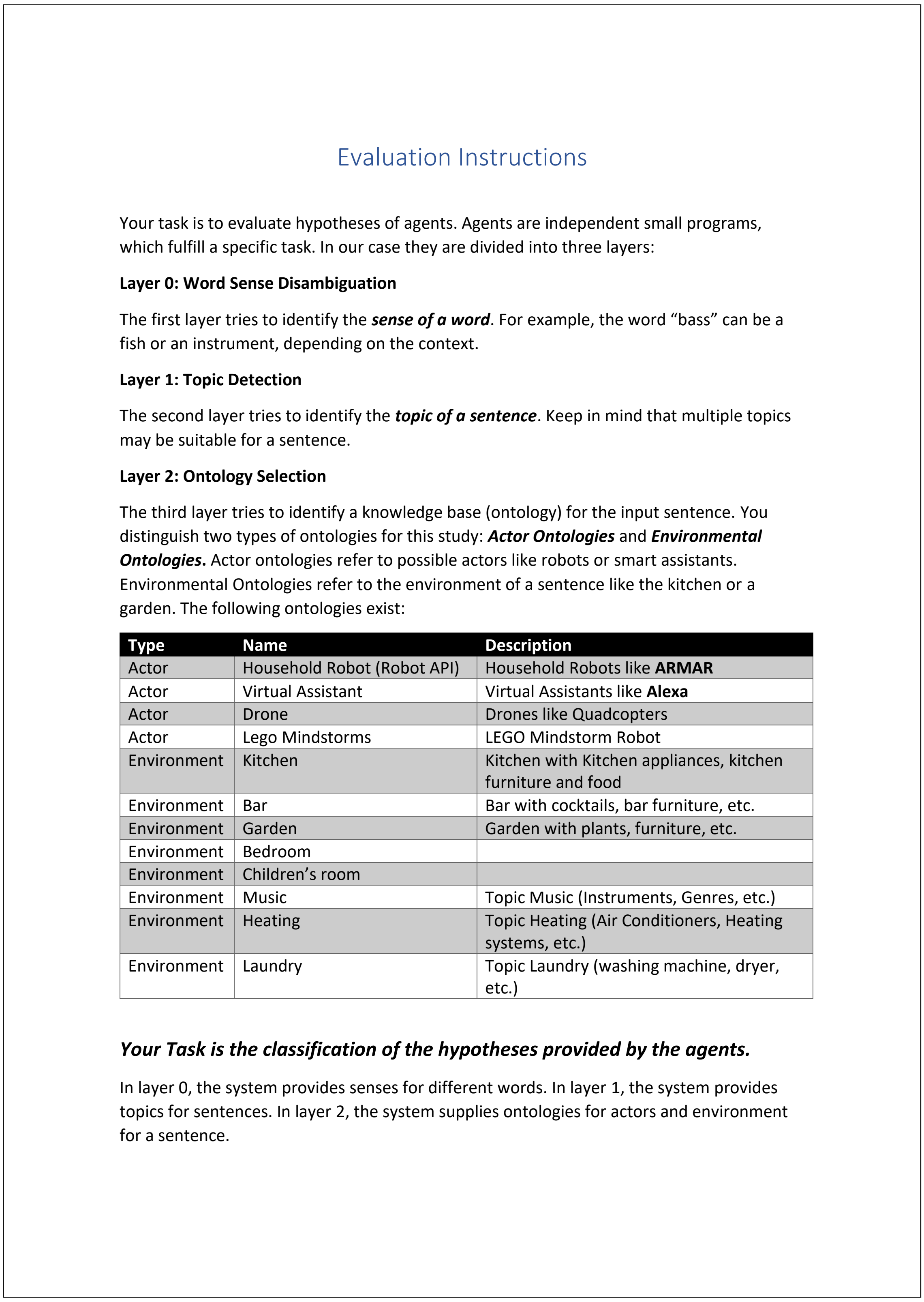
- [✓ Downloaded]
- Extracted

Upload the results as a ZIP at the following link. Also include your ID in the ZIP.

- ID added to the name of the ZIP
- [✓ Uploaded]

A.4. Evaluation — Instructions

Instructions for the participants. Information about the ontologies are from Keim [14].



Evaluation Instructions

Your task is to evaluate hypotheses of agents. Agents are independent small programs, which fulfill a specific task. In our case they are divided into three layers:

Layer 0: Word Sense Disambiguation

The first layer tries to identify the ***sense of a word***. For example, the word "bass" can be a fish or an instrument, depending on the context.

Layer 1: Topic Detection

The second layer tries to identify the ***topic of a sentence***. Keep in mind that multiple topics may be suitable for a sentence.

Layer 2: Ontology Selection

The third layer tries to identify a knowledge base (ontology) for the input sentence. You distinguish two types of ontologies for this study: ***Actor Ontologies*** and ***Environmental Ontologies.*** Actor ontologies refer to possible actors like robots or smart assistants. Environmental Ontologies refer to the environment of a sentence like the kitchen or a garden. The following ontologies exist:

Type	Name	Description
Actor	Household Robot (Robot API)	Household Robots like **ARMAR**
Actor	Virtual Assistant	Virtual Assistants like **Alexa**
Actor	Drone	Drones like Quadcopters
Actor	Lego Mindstorms	LEGO Mindstorm Robot
Environment	Kitchen	Kitchen with Kitchen appliances, kitchen furniture and food
Environment	Bar	Bar with cocktails, bar furniture, etc.
Environment	Garden	Garden with plants, furniture, etc.
Environment	Bedroom	
Environment	Children's room	
Environment	Music	Topic Music (Instruments, Genres, etc.)
Environment	Heating	Topic Heating (Air Conditioners, Heating systems, etc.)
Environment	Laundry	Topic Laundry (washing machine, dryer, etc.)

Your Task is the classification of the hypotheses provided by the agents.

In layer 0, the system provides senses for different words. In layer 1, the system provides topics for sentences. In layer 2, the system supplies ontologies for actors and environment for a sentence.

You can classify a hypothesis into the following categories:

- CORRECT: The hypothesis is correct.
- RATHER_CORRECT: The hypothesis is rather correct.
- RATHER_WRONG: The hypothesis is rather wrong.
- WRONG: The hypothesis is wrong.

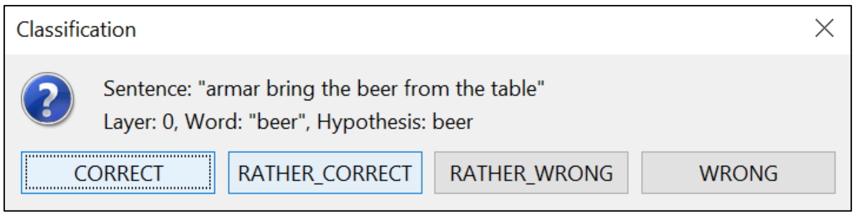
Example

Sentence: *"armar bring the beer from the table"*

Layer 0:
The systems provide the following hypotheses for "beer": "beer", "wheat beer", "dowry"

In this case "beer" could be classified as correct and "dowry" as wrong. Regarding "wheat beer" you have to decide whether you think that the correct or rather correct is a good classification.

The classification is done via a GUI:



Classification

Sentence: "armar bring the beer from the table"
Layer: 0, Word: "beer", Hypothesis: beer

Layer 1:

The system provides the following topics of the sentence: "beer", "furniture"
Again you have to decide whether a hypothesis is correct. In that case both hypotheses may be correct, as a user requests a robot to bring a beer from a table.

Layer 2:

The last layer provides the following ontologies:
"http://www.semanticweb.org/roboapibase",
"http://www.semanticweb.org/environment_music"

And again you have to decide which hypotheses are correct. In that case roboapibase is suitable as "ARMAR" is a robot. But the second hypothesis (music) is definitely wrong.

To start your task simply run "start.sh". After completion please create a ZIP and upload it.

A.5. Evaluation — Results

A.5.1. Precision, Recall & F1 for No Exploration

Sentence	WSD			TD			OS		
	P	R	F1	P	R	F1	P	R	F1
1.1	1.00	**1.00**	1.00	0.67	**0.40**	0.50	1.00	**0.67**	0.80
2.1	1.00	**1.00**	1.00	0.33	**0.33**	0.33	1.00	**1.00**	1.00
3.1	1.00	**0.60**	0.75	0.00	**0.00**	—	1.00	**1.00**	1.00
4.1	1.00	**1.00**	1.00	0.33	**0.14**	0.20	0.33	**1.00**	0.50
5.1	1.00	**0.80**	0.89	0.33	**0.14**	0.20	0.50	**0.50**	0.50
6.1	1.00	**1.00**	1.00	0.33	**0.17**	0.22	0.33	**1.00**	0.50
7.1	1.00	**0.80**	0.89	0.33	**0.09**	0.14	0.00	—	—
8.1	1.00	**1.00**	1.00	0.33	**0.20**	0.25	0.33	**1.00**	0.50
9.1	1.00	**1.00**	1.00	0.33	**0.17**	0.22	0.50	**0.50**	0.50
10.1	1.00	**0.75**	0.86	0.00	**0.00**	—	1.00	**0.67**	0.80
11.1	0.75	**1.00**	0.86	0.00	**0.00**	—	0.50	**0.50**	0.50
12.1	1.00	**0.75**	0.86	0.00	**0.00**	—	1.00	**1.00**	1.00
13.1	0.75	**0.75**	0.75	0.00	**0.00**	—	0.50	**1.00**	0.67
14.1	0.67	**1.00**	0.80	0.33	**0.33**	0.33	1.00	**1.00**	1.00
15.1	1.00	**1.00**	1.00	0.33	**0.25**	0.29	0.50	**1.00**	0.67
16.1	1.00	**1.00**	1.00	0.33	**0.20**	0.25	0.50	**1.00**	0.67
17.1	1.00	**1.00**	1.00	0.50	**1.00**	0.67	1.00	**1.00**	1.00
18.1	1.00	**0.67**	0.80	0.67	**0.67**	0.67	1.00	**1.00**	1.00
19.1	0.67	**0.67**	0.67	0.33	**0.33**	0.33	1.00	**1.00**	1.00
20.1	1.00	**1.00**	1.00	0.33	**0.50**	0.40	1.00	**1.00**	1.00
21.1	0.50	**1.00**	0.67	0.67	**0.67**	0.67	0.50	**1.00**	0.67
22.1	1.00	**1.00**	1.00	0.33	**0.20**	0.25	0.33	**1.00**	0.50
23.1	1.00	**1.00**	1.00	0.50	**1.00**	0.67	0.50	**1.00**	0.67
24.1	1.00	**1.00**	1.00	0.33	**0.14**	0.20	0.50	**1.00**	0.67
25.1	1.00	**0.67**	0.80	0.67	**0.67**	0.67	0.50	**1.00**	0.67
26.1	1.00	**1.00**	1.00	0.00	**0.00**	—	0.50	**1.00**	0.67
27.1	1.00	**1.00**	1.00	0.33	**0.17**	0.22	0.50	**1.00**	0.67
28.1	0.67	**1.00**	0.80	0.00	**0.00**	—	1.00	**1.00**	1.00
29.1	1.00	**0.67**	0.80	0.00	**0.00**	—	1.00	**1.00**	1.00
30.1	0.50	**1.00**	0.67	0.67	**0.67**	0.67	0.50	**1.00**	0.67
31.1	1.00	**1.00**	1.00	0.67	**0.67**	0.67	0.50	**1.00**	0.67
32.1	1.00	**1.00**	1.00	0.33	**0.20**	0.25	0.33	**1.00**	0.50
33.1	1.00	**0.75**	0.86	0.33	**0.20**	0.25	0.33	**1.00**	0.50
34.1	0.50	**1.00**	0.67	0.67	**0.67**	0.67	0.50	**1.00**	0.67
35.1	1.00	**0.80**	0.89	0.00	**0.00**	—	0.50	**1.00**	0.67
36.1	1.00	**1.00**	1.00	0.33	**0.20**	0.25	0.50	**1.00**	0.67
alexa1.1	0.67	**0.40**	0.50	0.00	**0.00**	—	1.00	**0.33**	0.50
bar1.1	0.50	**0.50**	0.50	0.00	**0.00**	—	1.00	**1.00**	1.00

bedroom1.1	0.67	**0.50**	0.57	0.00	**0.00**	—	1.00	**1.00**	1.00
childrensroom1.1	0.40	**0.50**	0.44	0.00	**0.00**	—	0.00	**0.00**	—
drone1.1	0.71	**1.00**	0.83	0.50	**0.33**	0.40	0.50	**0.50**	0.50
drone2.1	0.00	**0.00**	—	0.00	—	—	0.00	**0.00**	—
garden1.1	0.67	**0.50**	0.57	0.00	**0.00**	—	1.00	**1.00**	1.00
heating1.1	1.00	**0.50**	0.67	0.50	**0.33**	0.40	0.50	**0.50**	0.50
heating2.1	1.00	**0.33**	0.50	1.00	**0.75**	0.86	0.50	**0.50**	0.50
if.4.1	0.50	**0.50**	0.50	0.00	**0.00**	—	1.00	**1.00**	1.00
if.5.1	0.80	**0.57**	0.67	0.00	**0.00**	—	1.00	**0.67**	0.80
mindstorm1.1	0.50	**1.00**	0.67	0.00	—	—	0.00	**0.00**	—
music1.1	0.67	**0.40**	0.50	0.00	—	—	1.00	**1.00**	1.00
Average	0.84	**0.80**	0.82	0.28	**0.26**	0.40	0.63	**0.84**	0.74

A.5.2. Precision, Recall & F1 for best Paths

Sentence	**WSD**			**TD**			**OS**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
1.1 F1 WSD	1.00	1.00	1.00	0.67	0.40	0.50	1.00	0.67	0.80
1.1 F1 TD	1.00	1.00	1.00	1.00	0.60	0.75	1.00	0.67	0.80
1.1 F1 OS	1.00	1.00	1.00	0.67	0.40	0.50	1.00	0.67	0.80
2.1 F1 WSD	1.00	1.00	1.00	0.33	0.33	0.33	1.00	1.00	1.00
2.1 F1 TD	1.00	1.00	1.00	1.00	0.67	0.80	1.00	1.00	1.00
2.1 F1 OS	1.00	1.00	1.00	0.33	0.33	0.33	1.00	1.00	1.00
3.1 F1 WSD	1.00	0.60	0.75	0.33	0.17	0.22	1.00	1.00	1.00
3.1 F1 TD	0.67	0.40	0.50	0.67	0.33	0.44	1.00	1.00	1.00
3.1 F1 OS	1.00	0.60	0.75	0.33	0.17	0.22	1.00	1.00	1.00
4.1 F1 WSD	1.00	1.00	1.00	0.33	0.14	0.20	0.50	1.00	0.67
4.1 F1 TD	1.00	1.00	1.00	0.67	0.29	0.40	0.50	1.00	0.67
4.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
5.1 F1 WSD	1.00	0.80	0.89	0.33	0.14	0.20	1.00	1.00	1.00
5.1 F1 TD	1.00	0.80	0.89	1.00	0.43	0.60	0.50	0.50	0.50
5.1 F1 OS	1.00	0.80	0.89	0.33	0.14	0.20	1.00	1.00	1.00
6.1 F1 WSD	1.00	1.00	1.00	0.33	0.17	0.22	0.50	1.00	0.67
6.1 F1 TD	1.00	1.00	1.00	0.67	0.33	0.44	0.50	1.00	0.67
6.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
7.1 F1 TD	0.33	0.20	0.25	1.00	0.27	0.43	0.00	—	—
7.1 F1 OS	0.25	0.20	0.22	0.00	0.00	—	0.00	—	—
8.1 F1 WSD	1.00	1.00	1.00	0.33	0.20	0.25	0.50	1.00	0.67
8.1 F1 TD	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
8.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
9.1 F1 WSD	1.00	1.00	1.00	0.33	0.17	0.22	1.00	1.00	1.00
9.1 F1 TD	0.50	0.50	0.50	1.00	0.33	0.50	0.50	0.50	0.50
9.1 F1 OS	1.00	1.00	1.00	0.33	0.17	0.22	1.00	1.00	1.00

10.1 F1 WSD	1.00	0.75	0.86	0.33	0.14	0.20	1.00	0.67	0.80
10.1 F1 TD	0.33	0.25	0.29	1.00	0.29	0.44	0.50	0.33	0.40
10.1 F1 OS	1.00	0.75	0.86	0.33	0.14	0.20	1.00	0.67	0.80
11.1 F1 WSD	0.75	1.00	0.86	0.00	0.00	—	0.50	0.50	0.50
11.1 F1 TD	0.67	0.67	0.67	1.00	0.60	0.75	1.00	1.00	1.00
11.1 F1 OS	0.75	1.00	0.86	0.33	0.20	0.25	1.00	1.00	1.00
12.1 F1 WSD	1.00	0.75	0.86	0.33	0.20	0.25	1.00	1.00	1.00
12.1 F1 TD	0.67	0.50	0.57	1.00	0.40	0.57	0.50	0.50	0.50
12.1 F1 OS	1.00	0.75	0.86	0.33	0.20	0.25	1.00	1.00	1.00
13.1 F1 WSD	0.75	0.75	0.75	0.33	0.33	0.33	0.50	1.00	0.67
13.1 F1 TD	0.25	0.25	0.25	1.00	0.67	0.80	0.50	1.00	0.67
13.1 F1 OS	0.75	0.75	0.75	0.67	0.67	0.67	1.00	1.00	1.00
14.1 F1 WSD	0.67	1.00	0.80	0.33	0.33	0.33	1.00	1.00	1.00
14.1 F1 TD	0.33	0.50	0.40	1.00	0.67	0.80	0.50	1.00	0.67
14.1 F1 OS	0.67	1.00	0.80	0.33	0.33	0.33	1.00	1.00	1.00
15.1 F1 WSD	1.00	1.00	1.00	0.33	0.25	0.29	1.00	1.00	1.00
15.1 F1 TD	1.00	0.67	0.80	1.00	0.75	0.86	1.00	1.00	1.00
15.1 F1 OS	1.00	1.00	1.00	0.33	0.25	0.29	1.00	1.00	1.00
16.1 F1 WSD	1.00	1.00	1.00	0.33	0.20	0.25	1.00	1.00	1.00
16.1 F1 TD	1.00	1.00	1.00	0.67	0.40	0.50	1.00	1.00	1.00
16.1 F1 OS	1.00	1.00	1.00	0.33	0.20	0.25	1.00	1.00	1.00
17.1 F1 WSD	1.00	1.00	1.00	0.50	1.00	0.67	1.00	1.00	1.00
17.1 F1 TD	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
17.1 F1 OS	1.00	1.00	1.00	0.50	1.00	0.67	1.00	1.00	1.00
18.1 F1 WSD	1.00	0.67	0.80	0.67	0.67	0.67	1.00	1.00	1.00
18.1 F1 TD	1.00	0.67	0.80	1.00	1.00	1.00	1.00	1.00	1.00
18.1 F1 OS	1.00	0.67	0.80	0.67	0.67	0.67	1.00	1.00	1.00
19.1 F1 WSD	0.67	0.67	0.67	0.67	0.67	0.67	1.00	1.00	1.00
19.1 F1 TD	0.67	0.67	0.67	0.67	0.67	0.67	1.00	1.00	1.00
19.1 F1 OS	0.67	0.67	0.67	0.67	0.67	0.67	1.00	1.00	1.00
20.1 F1 WSD	1.00	1.00	1.00	0.33	0.50	0.40	1.00	1.00	1.00
20.1 F1 TD	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
20.1 F1 OS	1.00	1.00	1.00	0.33	0.50	0.40	1.00	1.00	1.00
21.1 F1 WSD	0.50	1.00	0.67	0.67	0.67	0.67	0.50	1.00	0.67
21.1 F1 TD	0.50	1.00	0.67	1.00	1.00	1.00	0.50	1.00	0.67
21.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
22.1 F1 WSD	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
22.1 F1 TD	0.33	0.33	0.33	1.00	0.60	0.75	0.50	1.00	0.67
22.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
23.1 F1 WSD	1.00	1.00	1.00	0.50	1.00	0.67	0.50	1.00	0.67
23.1 F1 TD	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
23.1 F1 OS	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
24.1 F1 WSD	1.00	1.00	1.00	0.33	0.14	0.20	0.50	1.00	0.67
24.1 F1 TD	1.00	1.00	1.00	0.67	0.29	0.40	0.50	1.00	0.67
24.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00

25.1 F1 WSD	1.00	0.67	0.80	0.67	0.67	0.67	0.50	1.00	0.67
25.1 F1 TD	1.00	0.67	0.80	1.00	1.00	1.00	0.50	1.00	0.67
25.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
26.1 F1 WSD	1.00	1.00	1.00	0.33	0.50	0.40	0.50	1.00	0.67
26.1 F1 TD	1.00	1.00	1.00	0.67	1.00	0.80	0.50	1.00	0.67
26.1 F1 OS	1.00	1.00	1.00	0.67	1.00	0.80	1.00	1.00	1.00
27.1 F1 WSD	1.00	1.00	1.00	0.67	0.33	0.44	0.50	1.00	0.67
27.1 F1 TD	1.00	1.00	1.00	0.67	0.33	0.44	0.50	1.00	0.67
27.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
28.1 F1 WSD	0.67	1.00	0.80	0.00	0.00	—	1.00	1.00	1.00
28.1 F1 TD	0.67	1.00	0.80	0.67	1.00	0.80	1.00	1.00	1.00
28.1 F1 OS	0.67	1.00	0.80	0.00	0.00	—	1.00	1.00	1.00
29.1 F1 WSD	1.00	0.67	0.80	0.67	0.15	0.25	1.00	1.00	1.00
29.1 F1 TD	1.00	0.67	0.80	1.00	0.23	0.38	1.00	1.00	1.00
29.1 F1 OS	1.00	0.67	0.80	0.67	0.15	0.25	1.00	1.00	1.00
30.1 F1 WSD	0.50	1.00	0.67	0.67	0.67	0.67	0.50	1.00	0.67
30.1 F1 TD	0.50	1.00	0.67	1.00	1.00	1.00	0.50	1.00	0.67
30.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
31.1 F1 WSD	1.00	1.00	1.00	0.67	0.67	0.67	0.50	1.00	0.67
31.1 F1 TD	1.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00	0.67
31.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
32.1 F1 WSD	1.00	1.00	1.00	0.33	0.20	0.25	0.50	1.00	0.67
32.1 F1 TD	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
32.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
33.1 F1 WSD	1.00	0.75	0.86	0.33	0.20	0.25	0.50	1.00	0.67
33.1 F1 TD	0.33	0.25	0.29	1.00	0.40	0.57	0.50	1.00	0.67
33.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
34.1 F1 WSD	0.50	1.00	0.67	0.67	0.67	0.67	0.50	1.00	0.67
34.1 F1 TD	0.50	1.00	0.67	1.00	1.00	1.00	0.50	1.00	0.67
34.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
35.1 F1 WSD	1.00	0.80	0.89	0.67	0.29	0.40	1.00	1.00	1.00
35.1 F1 TD	1.00	0.80	0.89	1.00	0.43	0.60	1.00	1.00	1.00
35.1 F1 OS	1.00	0.80	0.89	0.67	0.29	0.40	1.00	1.00	1.00
36.1 F1 WSD	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
36.1 F1 TD	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
36.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
alexa1.1 F1 TD	0.50	0.40	0.44	1.00	0.67	0.80	1.00	0.33	0.50
alexa1.1 F1 OS	0.00	0.00	—	0.00	0.00	—	1.00	1.00	1.00
bar1.1 F1 WSD	1.00	0.50	0.67	0.00	0.00	—	0.25	1.00	0.40
bar1.1 F1 TD	0.20	0.17	0.18	1.00	0.33	0.50	1.00	1.00	1.00
bar1.1 F1 OS	0.50	0.50	0.50	0.00	0.00	—	1.00	1.00	1.00
bedroom1.1 F1 WSD	0.67	0.50	0.57	0.00	0.00	—	0.25	1.00	0.40
bedroom1.1 F1 TD	0.67	0.50	0.57	0.33	1.00	0.50	1.00	1.00	1.00
bedroom1.1 F1 OS	0.67	0.50	0.57	0.00	0.00	—	1.00	1.00	1.00
childrensroom1.1 F1 WSD	0.60	0.75	0.67	0.00	0.00	—	0.00	0.00	—

childrensroom1.1 F1 TD	0.20	0.25	0.22	0.33	1.00	0.50	1.00	1.00	1.00
childrensroom1.1 F1 OS	0.50	0.50	0.50	0.00	0.00	—	1.00	1.00	1.00
drone1.1 F1 WSD	0.71	1.00	0.83	0.00	0.00	—	0.50	0.50	0.50
drone1.1 F1 TD	0.14	0.20	0.17	0.67	0.67	0.67	0.00	0.00	—
drone1.1 F1 OS	0.71	1.00	0.83	0.00	0.00	—	0.50	0.50	0.50
drone2.1 F1 WSD	0.25	1.00	0.40	0.00	—	—	0.00	0.00	—
drone2.1 F1 TD	0.00	0.00	—	0.00	—	—	0.00	0.00	—
drone2.1 F1 OS	0.00	0.00	—	0.00	—	—	0.00	0.00	—
garden1.1 F1 WSD	1.00	0.75	0.86	0.33	0.13	0.18	1.00	1.00	1.00
garden1.1 F1 TD	0.67	0.50	0.57	0.67	0.25	0.36	1.00	1.00	1.00
garden1.1 F1 OS	0.67	0.50	0.57	0.00	0.00	—	1.00	1.00	1.00
heating1.1 F1 WSD	1.00	0.50	0.67	0.50	0.33	0.40	0.50	0.50	0.50
heating1.1 F1 TD	—	0.00	—	1.00	0.33	0.50	0.33	0.50	0.40
heating1.1 F1 OS	1.00	0.50	0.67	0.50	0.33	0.40	0.50	0.50	0.50
heating2.1 F1 WSD	1.00	0.50	0.67	1.00	0.50	0.67	1.00	1.00	1.00
heating2.1 F1 TD	1.00	0.33	0.50	1.00	0.50	0.67	1.00	1.00	1.00
heating2.1 F1 OS	1.00	0.33	0.50	0.67	0.50	0.57	1.00	1.00	1.00
if.4.1 F1 WSD	0.75	0.75	0.75	0.00	0.00	—	1.00	1.00	1.00
if.4.1 F1 TD	0.25	0.25	0.25	0.00	0.00	—	1.00	1.00	1.00
if.4.1 F1 OS	0.20	0.25	0.22	0.00	0.00	—	1.00	1.00	1.00
if.5.1 F1 WSD	0.80	0.57	0.67	0.33	0.20	0.25	1.00	0.67	0.80
if.5.1 F1 TD	0.50	0.43	0.46	0.67	0.40	0.50	1.00	0.67	0.80
if.5.1 F1 OS	0.25	0.14	0.18	0.00	0.00	—	1.00	1.00	1.00
mindstorm1.1 F1 WSD	0.50	1.00	0.67	0.00	—	—	0.00	0.00	—
mindstorm1.1 F1 TD	0.00	0.00	—	0.00	—	—	0.00	0.00	—
mindstorm1.1 F1 OS	0.00	0.00	—	0.00	—	—	0.00	0.00	—
music1.1 F1 WSD	0.67	0.40	0.50	0.00	—	—	1.00	1.00	1.00
music1.1 F1 TD	0.33	0.20	0.25	0.00	—	—	0.00	0.00	—
music1.1 F1 OS	0.67	0.40	0.50	0.00	—	—	1.00	1.00	1.00

A.5.3. Recall in Experiments

	Top-X Recall			Random Recall		
Sentence	**WSD**	**TD**	**OS**	**WSD**	**TD**	**OS**
1.1	1.00	0.60	0.67	1.00	0.60	0.67
2.1	1.00	0.67	1.00	0.50	0.33	1.00
3.1	1.00	0.17	1.00	1.00	0.50	1.00
4.1	1.00	0.29	1.00	1.00	0.71	1.00
5.1	1.00	0.29	1.00	1.00	0.86	1.00
6.1	1.00	0.33	1.00	1.00	0.67	1.00
7.1	1.00	0.09	—	1.00	0.82	—
8.1	1.00	0.40	1.00	1.00	0.80	1.00
9.1	1.00	0.17	1.00	1.00	0.67	1.00
10.1	1.00	0.29	0.67	1.00	0.57	0.67

11.1	1.00	0.60	1.00	1.00	1.00	1.00
12.1	1.00	0.60	1.00	1.00	1.00	1.00
13.1	1.00	0.67	1.00	1.00	1.00	1.00
14.1	1.00	0.67	1.00	1.00	1.00	1.00
15.1	1.00	0.50	1.00	1.00	0.75	1.00
16.1	1.00	0.40	1.00	1.00	0.80	1.00
17.1	1.00	1.00	1.00	0.00	0.00	1.00
18.1	1.00	1.00	1.00	0.33	0.33	1.00
19.1	1.00	1.00	1.00	1.00	0.67	1.00
20.1	1.00	1.00	1.00	0.50	0.00	1.00
21.1	1.00	1.00	1.00	0.00	0.67	1.00
22.1	1.00	0.40	1.00	1.00	1.00	1.00
23.1	1.00	1.00	1.00	0.00	0.00	1.00
24.1	1.00	0.29	1.00	1.00	0.86	1.00
25.1	1.00	1.00	1.00	0.33	0.33	1.00
26.1	1.00	1.00	1.00	1.00	1.00	1.00
27.1	1.00	0.50	1.00	1.00	0.50	1.00
28.1	1.00	1.00	1.00	1.00	1.00	1.00
29.1	1.00	0.31	1.00	1.00	0.69	1.00
30.1	1.00	1.00	1.00	0.00	0.67	1.00
31.1	1.00	1.00	1.00	0.50	0.67	1.00
32.1	1.00	0.40	1.00	1.00	0.80	1.00
33.1	1.00	0.40	1.00	1.00	0.80	1.00
34.1	1.00	1.00	1.00	0.00	0.67	1.00
35.1	1.00	0.43	1.00	1.00	0.86	1.00
36.1	1.00	0.60	1.00	1.00	0.60	1.00
alexa1.1	1.00	1.00	1.00	1.00	1.00	1.00
bar1.1	1.00	0.11	1.00	1.00	0.78	1.00
bedroom1.1	1.00	1.00	1.00	1.00	1.00	1.00
childrensroom1.1	1.00	0.00	1.00	1.00	1.00	1.00
drone1.1	1.00	0.33	0.50	1.00	1.00	1.00
drone2.1	1.00	—	0.00	1.00	—	0.00
garden1.1	1.00	0.50	1.00	1.00	0.88	1.00
heating1.1	1.00	1.00	0.50	0.00	0.00	0.50
heating2.1	1.00	0.75	1.00	1.00	1.00	1.00
if.4.1	1.00	0.00	1.00	1.00	0.00	1.00
if.5.1	1.00	0.20	1.00	1.00	0.80	1.00
mindstorm1.1	1.00	—	0.00	1.00	—	0.00
music1.1	1.00	—	1.00	1.00	—	1.00
Average	**1.00**	**0.59**	**0.92**	**0.82**	**0.69**	**0.93**

A.5.4. Absolute Rank, Group-Rank, & Group Size Avg for best Paths

Best WSD Paths (F1)

Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0, 711]	33.57	[0, 706]	32.86	[1.00, 1.14]	1.01
ADD-SIGMA	[0, 714]	103.53	[0, 714]	103.53	[1.00, 1.33]	1.01
ADD-AVERAGE	[0, 598]	32.45	[0, 598]	32.45	[1.00, 1.00]	1.00
ADD-MAX	[0, 511]	46.84	[0, 503]	46.10	[1.00, 1.32]	1.01
ADD-MIN	[0, 604]	41.57	[0, 592]	40.35	[1.00, 2.00]	1.11
MUL-MEDIAN	[0, 723]	80.35	[0, 717]	79.00	[1.00, 1.14]	1.01
MUL-SIGMA	[0, 605]	85.04	[0, 605]	85.04	[1.00, 4.00]	1.11
MUL-AVERAGE	[0, 653]	71.59	[0, 653]	71.59	[1.00, 1.00]	1.00
MUL-MAX	[0, 598]	66.47	[0, 598]	65.63	[1.00, 1.32]	1.01
MUL-MIN	[0, 622]	57.22	[0, 610]	55.57	[1.00, 2.00]	1.11

Best TD Paths (F1)

Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0, 390]	34.14	[0, 387]	33.24	[1.00, 1.14]	1.01
ADD-SIGMA	[0, 695]	111.18	[0, 695]	111.18	[1.00, 1.33]	1.01
ADD-AVERAGE	[0, 401]	37.14	[0, 401]	37.14	[1.00, 1.00]	1.00
ADD-MAX	[0, 573]	55.49	[0, 565]	54.80	[1.00, 1.32]	1.01
ADD-MIN	[0, 481]	30.02	[0, 472]	28.86	[1.00, 2.00]	1.11
MUL-MEDIAN	[0, 681]	79.86	[0, 677]	78.59	[1.00, 1.14]	1.01
MUL-SIGMA	[0, 685]	121.55	[0, 685]	121.55	[1.00, 4.00]	1.11
MUL-AVERAGE	[0, 708]	84.78	[0, 708]	84.78	[1.00, 1.00]	1.00
MUL-MAX	[0, 722]	97.18	[0, 708]	96.14	[1.00, 1.32]	1.01
MUL-MIN	[0, 503]	54.16	[0, 491]	52.76	[1.00, 2.00]	1.11

Best OS Paths (F1)

Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0, 599]	33.71	[0, 591]	33.04	[1.00, 1.14]	1.01
ADD-SIGMA	[0, 695]	87.35	[0, 695]	87.35	[1.00, 1.33]	1.01
ADD-AVERAGE	[0, 629]	34.61	[0, 629]	34.61	[1.00, 1.00]	1.00
ADD-MAX	[0, 659]	48.02	[0, 637]	47.16	[1.00, 1.32]	1.01
ADD-MIN	[0, 479]	24.94	[0, 474]	24.04	[1.00, 2.00]	1.11
MUL-MEDIAN	[0, 485]	47.02	[0, 477]	46.18	[1.00, 1.14]	1.01
MUL-SIGMA	[0, 663]	75.47	[0, 663]	75.47	[1.00, 4.00]	1.11
MUL-AVERAGE	[0, 570]	49.45	[0, 570]	49.45	[1.00, 1.00]	1.00
MUL-MAX	[0, 602]	55.73	[0, 585]	55.08	[1.00, 1.32]	1.01
MUL-MIN	[0, 341]	27.41	[0, 339]	26.63	[1.00, 2.00]	1.11

A.5.5. Normalized Rank, Group-Rank, & Group Size Avg for best Paths

Best WSD Paths (F1)

Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0.00, 0.95]	0.12	[0.00, 0.95]	0.12	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.95]	0.41	[0.00, 0.95]	0.41	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.92]	0.11	[0.00, 0.92]	0.11	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.91]	0.16	[0.00, 0.90]	0.16	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.95]	0.16	[0.00, 0.95]	0.16	[1.00, 2.00]	1.11
MUL-MEDIAN	[0.00, 0.96]	0.21	[0.00, 0.96]	0.21	[1.00, 1.14]	1.01
MUL-SIGMA	[0.00, 0.87]	0.33	[0.00, 0.87]	0.33	[1.00, 4.00]	1.11
MUL-AVERAGE	[0.00, 0.89]	0.18	[0.00, 0.89]	0.18	[1.00, 1.00]	1.00
MUL-MAX	[0.00, 0.90]	0.19	[0.00, 0.90]	0.19	[1.00, 1.32]	1.01
MUL-MIN	[0.00, 0.89]	0.15	[0.00, 0.89]	0.15	[1.00, 2.00]	1.11

Best TD Paths (F1)

Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0.00, 0.68]	0.12	[0.00, 0.68]	0.12	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.99]	0.43	[0.00, 0.99]	0.43	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.68]	0.12	[0.00, 0.68]	0.12	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.94]	0.18	[0.00, 0.94]	0.18	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.64]	0.12	[0.00, 0.64]	0.12	[1.00, 2.00]	1.11
MUL-MEDIAN	[0.00, 0.91]	0.29	[0.00, 0.91]	0.29	[1.00, 1.14]	1.01
MUL-SIGMA	[0.00, 0.96]	0.47	[0.00, 0.96]	0.47	[1.00, 4.00]	1.11
MUL-AVERAGE	[0.00, 0.94]	0.29	[0.00, 0.94]	0.29	[1.00, 1.00]	1.00
MUL-MAX	[0.00, 0.96]	0.33	[0.00, 0.96]	0.33	[1.00, 1.32]	1.01
MUL-MIN	[0.00, 0.96]	0.26	[0.00, 0.96]	0.26	[1.00, 2.00]	1.11

Best OS Paths (F1)

Method (LC-LEE)	Rank		Group-Rank		Group Size Avg	
	Range	∅	Range	∅	Range	∅
ADD-MEDIAN	[0.00, 0.95]	0.16	[0.00, 0.95]	0.16	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.99]	0.44	[0.00, 0.99]	0.44	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.86]	0.18	[0.00, 0.86]	0.18	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.88]	0.28	[0.00, 0.88]	0.28	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.88]	0.14	[0.00, 0.87]	0.14	[1.00, 2.00]	1.11
MUL-MEDIAN	[0.00, 0.89]	0.25	[0.00, 0.89]	0.25	[1.00, 1.14]	1.01
MUL-SIGMA	[0.00, 0.99]	0.44	[0.00, 0.99]	0.44	[1.00, 4.00]	1.11
MUL-AVERAGE	[0.00, 0.91]	0.25	[0.00, 0.91]	0.25	[1.00, 1.00]	1.00
MUL-MAX	[0.00, 0.96]	0.31	[0.00, 0.96]	0.31	[1.00, 1.32]	1.01
MUL-MIN	[0.00, 0.63]	0.19	[0.00, 0.62]	0.20	[1.00, 2.00]	1.11

A.5.6. Normalized Rank, Group-Rank, & Group Size Avg for Top-3 Paths

Top-3 WSD Paths (F1)

Method (LC-LEE)	**Rank** Range	∅	**Group-Rank** Range	∅	**Group Size Avg** Range	∅
ADD-MEDIAN	[0.00, 0.33]	0.04	[0.00, 0.31]	0.04	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.41]	0.07	[0.00, 0.41]	0.07	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.27]	0.04	[0.00, 0.27]	0.04	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.17]	0.05	[0.00, 0.17]	0.05	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.33]	0.05	[0.00, 0.36]	0.05	[1.00, 2.00]	1.11
MUL-MEDIAN	[0.00, 0.41]	0.05	[0.00, 0.39]	0.05	[1.00, 1.14]	1.01
MUL-SIGMA	[0.00, 0.35]	0.08	[0.00, 0.35]	0.08	[1.00, 4.00]	1.11
MUL-AVERAGE	[0.00, 0.21]	0.04	[0.00, 0.21]	0.04	[1.00, 1.00]	1.00
MUL-MAX	[0.00, 0.17]	0.04	[0.00, 0.17]	0.04	[1.00, 1.32]	1.01
MUL-MIN	[0.00, 0.33]	0.02	[0.00, 0.36]	0.02	[1.00, 2.00]	1.11

Top-3 TD Paths (F1)

Method (LC-LEE)	**Rank** Range	∅	**Group-Rank** Range	∅	**Group Size Avg** Range	∅
ADD-MEDIAN	[0.00, 0.21]	0.04	[0.00, 0.22]	0.04	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.93]	0.19	[0.00, 0.93]	0.19	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.38]	0.04	[0.00, 0.38]	0.04	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.76]	0.07	[0.00, 0.76]	0.07	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.40]	0.05	[0.00, 0.46]	0.05	[1.00, 2.00]	1.11
MUL-MEDIAN	[0.00, 0.52]	0.06	[0.00, 0.52]	0.06	[1.00, 1.14]	1.01
MUL-SIGMA	[0.00, 0.79]	0.20	[0.00, 0.79]	0.20	[1.00, 4.00]	1.11
MUL-AVERAGE	[0.00, 0.59]	0.06	[0.00, 0.59]	0.06	[1.00, 1.00]	1.00
MUL-MAX	[0.00, 0.69]	0.08	[0.00, 0.69]	0.08	[1.00, 1.32]	1.01
MUL-MIN	[0.00, 0.33]	0.05	[0.00, 0.38]	0.05	[1.00, 2.00]	1.11

Top-3 OS Paths (F1)

Method (LC-LEE)	**Rank** Range	∅	**Group-Rank** Range	∅	**Group Size Avg** Range	∅
ADD-MEDIAN	[0.00, 0.25]	0.04	[0.00, 0.24]	0.04	[1.00, 1.14]	1.01
ADD-SIGMA	[0.00, 0.73]	0.11	[0.00, 0.73]	0.11	[1.00, 1.33]	1.01
ADD-AVERAGE	[0.00, 0.37]	0.05	[0.00, 0.37]	0.05	[1.00, 1.00]	1.00
ADD-MAX	[0.00, 0.33]	0.05	[0.00, 0.33]	0.05	[1.00, 1.32]	1.01
ADD-MIN	[0.00, 0.55]	0.06	[0.00, 0.56]	0.06	[1.00, 2.00]	1.11
MUL-MEDIAN	[0.00, 0.47]	0.08	[0.00, 0.47]	0.08	[1.00, 1.14]	1.01
MUL-SIGMA	[0.00, 0.66]	0.11	[0.00, 0.66]	0.11	[1.00, 4.00]	1.11
MUL-AVERAGE	[0.00, 0.48]	0.07	[0.00, 0.48]	0.07	[1.00, 1.00]	1.00
MUL-MAX	[0.00, 0.51]	0.07	[0.00, 0.51]	0.07	[1.00, 1.32]	1.01
MUL-MIN	[0.00, 0.33]	0.05	[0.00, 0.38]	0.05	[1.00, 2.00]	1.11

A.5.7. Precision, Recall & F1 for the overall Approach

Sentence	WSD			TD			OS		
	P	R	F1	P	R	F1	P	R	F1
1.1	1.00	1.00	1.00	0.67	0.40	0.50	1.00	0.67	0.80
2.1	1.00	1.00	1.00	1.00	0.67	0.80	1.00	1.00	1.00
3.1	1.00	0.60	0.75	0.33	0.17	0.22	1.00	1.00	1.00
4.1	1.00	1.00	1.00	0.67	0.29	0.40	0.50	1.00	0.67
5.1	1.00	0.80	0.89	0.33	0.14	0.20	0.50	0.50	0.50
6.1	1.00	1.00	1.00	0.67	0.33	0.44	0.50	1.00	0.67
7.1	1.00	0.80	0.89	0.33	0.09	0.14	—	—	—
8.1	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
9.1	1.00	1.00	1.00	0.33	0.17	0.22	0.50	0.50	0.50
10.1	1.00	0.75	0.86	0.33	0.14	0.20	1.00	0.67	0.80
11.1	0.75	1.00	0.86	0.33	0.20	0.25	1.00	1.00	1.00
12.1	1.00	0.75	0.86	0.33	0.20	0.25	1.00	1.00	1.00
13.1	0.75	0.75	0.75	0.33	0.33	0.33	0.50	1.00	0.67
14.1	0.67	1.00	0.80	0.67	0.67	0.67	1.00	1.00	1.00
15.1	1.00	1.00	1.00	0.67	0.50	0.57	1.00	1.00	1.00
16.1	1.00	1.00	1.00	0.67	0.40	0.50	1.00	1.00	1.00
17.1	1.00	1.00	1.00	0.50	1.00	0.67	1.00	1.00	1.00
18.1	1.00	0.67	0.80	1.00	0.67	0.80	0.50	1.00	0.67
19.1	0.67	0.67	0.67	0.67	0.67	0.67	1.00	1.00	1.00
20.1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
21.1	0.50	1.00	0.67	1.00	0.67	0.80	0.50	1.00	0.67
22.1	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
23.1	1.00	1.00	1.00	0.50	1.00	0.67	0.50	1.00	0.67
24.1	1.00	1.00	1.00	0.67	0.29	0.40	0.50	1.00	0.67
25.1	1.00	0.67	0.80	0.67	0.67	0.67	0.50	1.00	0.67
26.1	1.00	1.00	1.00	0.33	0.50	0.40	0.50	1.00	0.67
27.1	1.00	1.00	1.00	0.67	0.33	0.44	0.50	1.00	0.67
28.1	0.67	1.00	0.80	0.00	0.00	—	1.00	1.00	1.00
29.1	1.00	0.67	0.80	0.67	0.15	0.25	1.00	1.00	1.00
30.1	0.50	1.00	0.67	1.00	0.67	0.80	0.50	1.00	0.67
31.1	1.00	1.00	1.00	1.00	0.67	0.80	0.50	1.00	0.67
32.1	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
33.1	1.00	0.75	0.86	0.67	0.40	0.50	0.50	1.00	0.67
34.1	0.50	1.00	0.67	1.00	0.67	0.80	0.50	1.00	0.67
35.1	1.00	0.80	0.89	0.67	0.29	0.40	1.00	1.00	1.00
36.1	1.00	1.00	1.00	0.67	0.40	0.50	0.50	1.00	0.67
alexa1.1	0.33	0.20	0.25	0.00	0.00	—	1.00	0.67	0.80
bar1.1	0.50	0.50	0.50	0.00	0.00	—	1.00	1.00	1.00
bedroom1.1	0.67	0.50	0.57	0.00	0.00	—	1.00	1.00	1.00
childrensroom1.1	0.40	0.50	0.44	0.00	0.00	—	0.00	0.00	—

drone1.1	0.71	1.00	0.83	0.00	0.00	—	0.50	0.50	0.50
drone2.1	0.00	0.00	—	0.00	—	—	0.00	0.00	—
garden1.1	0.67	0.50	0.57	0.33	0.13	0.18	1.00	1.00	1.00
heating1.1	—	0.00	—	1.00	0.33	0.50	0.33	0.50	0.40
heating2.1	1.00	0.33	0.50	0.67	0.50	0.57	1.00	1.00	1.00
if.4.1	0.50	0.50	0.50	0.00	0.00	—	0.50	0.50	0.50
if.5.1	0.80	0.57	0.67	0.33	0.20	0.25	1.00	0.67	0.80
mindstorm1.1	0.00	0.00	—	0.00	—	—	0.00	0.00	—
music1.1	0.67	0.40	0.50	0.00	—	—	1.00	1.00	1.00
Average	0.82	0.77	0.82	**0.50**	**0.37**	**0.49**	**0.69**	**0.86**	**0.79**