Experimental Evaluation and Modelling of the Comprehension of Indirect Anaphors in a Programming Language

Sebastian Lohmeier

Version 1.1

A Master's thesis supervised by
Prof. Dr.-Ing. Sebastian Möller (TU Berlin),
Prof. Dr. Lutz Prechelt (FU Berlin), and
Prof. Dr. Arthur Jacobs (FU Berlin)

Abstract

In this thesis I evaluate experimentally, how programmers understand indirect anaphors in source code and attempt to predict the comprehension of indirect anaphors in a cognitive model. Because indirect anaphors underspecify a relation, they might ease comprehension, but they will be hard to comprehend for those who do not know the underspecified relation. Indirect anaphors could thus be used as an input method only. Alternatively, indirect anaphors could also be shown during reading, if (a) they benefit readers sufficiently often, and (b) such cases can to be identified before the code is presented to the reader. Prior experiments in psycholinguistics indicate that condition (a) can be fulfilled with regard to on-line and off-line effects comprehension and the design of the experiment in this thesis follows the design of the existing studies. Existing cognitive models of text comprehension render the prediction of the comprehension of indirect anaphors in source code achievable based on the activation values computed in such models.

An experiment was performed, in which programmers read source code while having their eye movements tracked and answered comprehension questions afterwards. Extensive post-processing of the eye tracking data was performed to validate the data and to reduce the error in the data. It seems advisable to use required fixations in the experimental materials next time, to have a better basis for post-hoc error correction. Even after error correction exceptionally many data had to be removed. The ANOVA for the eye tracking data could not be performed as within-subject analysis, probably due to missing data. The post-test questionnaire provided input to further the development of anaphors.

A cognitive model was implemented in jACT-R that receives knowledge representations derived from the AST of Eclipse as well as a sequence of experimentally derived knowledge representations. The model constructs a text-world model based on these two inputs and computes activation values that determine the generated regression-path reading times. Even though there are errors during the construction of the text-world model, the model processes all input and generates regression durations, but will need further optimiziation.

Contents

Preface

This Master's thesis concludes 2 1/2 years of work towards using eye tracking and ACT-R to model how programmers read source code with indirect anaphors. It documents an empirical view on indirect anaphors in source code, a topic I had tackled from a more computational perspective in my Bachelor's thesis. During the Master's, I had the chance to take many different perspectives on language and programming languages that took maximal benefit from the intellectual abundance that a city with four universities provides. Horst Zuse, maintaining his father's legacy, offered a seminar that provided room for a historical perspective on the development of programming languages, in which I benefitted from the work of Dirk Siefkes and colleagues on the social history of informatics. The freies Anwendungsfach and Studium Generale of Technische Unversität offered me the academic freedom to largely ignore the study plans that good-willing others had made up and enabled me to choose 2/5 of my courses outside of informatics, in order to get into eye tracking, cognitive modelling, and cognitive linguistics at Technische Universität and into psycholinguistics at Humboldt Universität.

My Master's has been a rewarding social experience twice: in the beginning, when I was part of the organising team of the 52nd Student's Conference of Linguistics organised by students for students and when I sought and worked with the participants of the experiment for this thesis. I also received a lot of support without which this thesis would not have been possible and for which I am most thankful. Monika Schwarz-Friesel, Thomas Green and Hiltrud Kramm did never hesitate to encourage me to continue my work in the way I find useful. Monika Schwarz-Friesel, Nele Russwinkel, Lutz Prechelt and Sebastian Möller promptly offered and provided support over the course of my studies. Monika Schwarz-Friesel, Helge Skirl, Nele Russwinkel and Sebastian Möller asked for proposals or were open to my proposals of how to customize their courses to my needs, which is not self-evident. Stacy Birch of CUNY was most kind to promptly provide materials for me to use as the basis of my first cognitive model of reading. SMI made an eye tracker available to me as part of the ECEM 2013 student competition and The Eye Tribe offer a development kit at a price that I could afford. Both enabled my work on eye tracking with Eclipse. Freunde der Technischen Universität Berlin and the Psychology of Programming Interest Group (PPIG) supported me financially so I was able to participate in the 25th Workshop of PPIG. I also thank the participants of my experiment, as well as Benny Briesemeister, Teresa Busjahn, Marc Halbrügge, Arthur Jacobs, Lutz Prechelt, Nele Russwinkel, and Sascha Tamm, who all contributed to the design, execution, or analysis of the experiment. Anthony Harrison implemented jACT-R used for the cognitive model in Chapter 5. Martin Konvička was so nice to proof-read the thesis. I thank my family for their patience and financial support and finally Martin, who has always been interesting, lovely and kind over the years.

Sebastian Lohmeier
sl@monochromata.de

1 Introduction

Indirect anaphors are well known in linguistics and have previously been added to the Java programming lanuage Lohmeier (2011a). In this thesis I evaluate experimentally, how programmers understand indirect anaphors in source code and attempt to predict the comprehension of indirect anaphors in a cognitive model. This chapter introduces indirect anaphors, experiments used to study indirect anaphors and models of the comprehension of indirect anaphors. It compiles the information that provides the basis for the introduction to indirect anaphors in source code, experiment and the cognitive model that follow in subsequent chapters. (Please note that this text uses page notes.To keep the text plain, there are no potentially-distracting super-script indices within the text, but there are notes starting on page 99 that provide fine-grained acknowledgements, technical details, traces to data, source code and scripts related to the statements made in the thesis. Each note starts with a page number a quote of the text that it belongs to. In the PDF, the quote links to the page to which the note belongs.)

In texts, nouns and other words are used to bring objects and ideas to the attention of the reader. The notional relation between linguistic forms appearing in a text and the non-linguistic objects or ideas they identify is called *reference*. The objects or ideas referred to by words are called *referents*. Words come with a (reader-specific) *reference potential*: they do not automatically bring referents to attention; instead, words need to be preceived and mentally processed by the individual reader to bring a referent to her attention. When reading sample (1) below, “Ken”, “pressed”, “the power button”, “his” and “computer” are typically understood as referring to some entity or action in the mind of the reader, as long as she possesses a concept of what the referring expressions mean.

(1) Ken pressed the power button of his computer.

Even a short sentence like the one in (1) shows that reference alone is not sufficient to explain how meaning is constructed during the comprehension of text: ideas referred to do not stand in isolation. Instead, ideas are connected to each other, also by a multitude of functions of language, anaphors being one of them.

1.1 Direct and Indirect Anaphors

Means of relating backwards to what was previously mentioned in a text are in linguistics said to function as *anaphors*. When an expression functions as anaphor that is related to another previously-mentioned expression, it has a referent that is identical (co-referential) or related to the referent of the previously-mentioned expression. Pronouns like “she”, “this”, and definite noun phrases like “the hard disk” in (2) can function as anaphors. While words like “this” exist both in English and in programming languages, “this” in programming languages cannot,

unlike in English, be used to indicate to the compiler that "this" is meant to refer to a referent freely chosen by the programmer. Lopes, Dourish, Lorenz, and Lieberherr (2003) and Knöll and Mezini (2006) proposed to include anaphors in general-purpose programming languages but did not cover *indirect anaphors* like "the hard disk" in

(2) Ken pressed the power button of his computer. The computer did not boot. The hard disk had been formatted.

that refers to a part of "the computer". Indirect anaphors (also: *bridging*, *accommodation*, or *associative anaphors*) underspecify information (e.g. that computers have hard-disks): the underspecified relation is not encoded in the text, but is taken from world knowledge and added to the reader's mental model of the text's meaning. In the above example "the hard disk" functions as indirect anaphor with the *related expression* "the computer" in the previous sentence. The related expressions of an indirect anaphor is also called *anchor* because the referent of the indirect anaphor is related to the referent of anchor by means of a part-of-relation (computers have hard-disks). *Direct anaphors* like "his" in the above example, have a related expression that is also called *antecedent*, "Ken" in the above example. Direct anaphors are often *co-referential* to their antecedent (both refer to the same referent).

Because indirect anaphors underspecify relations, indirect anaphors in programming languages could shorten source code in cases in which programmers know the underspecified relation but would make source code reading harder to comprehend for those who do not know the underspecified relation. They can therefore either only be used as part of an input method that automatically adds the underspecified relation to the text, when the indirect anaphor has been entered. Alternatively, indirect anaphors might be displayed during code reading, too, but only to programmers who know the underspecified relation. To all other programmers, a variant of the source code would be displayed that explicates the otherwise underspecified relation. Both variants of the source code would be generated automatically (not as part of this thesis, though). The display of indirect anaphors is viable subject to two conditions:

(a) indirect anaphors need to benefit the reader sufficiently often, and
(b) such cases need to be identified before the code is presented to the reader.

Assuming that the comprehension of indirect anaphors in source code is comparable to the comprehension of indirect anaphors in natural language, experiments from psycholinguistics can provide directions on whether condition (a) can be met and how to design an experiment to test the condition for indirect anaphors in source code. Process models that were constructed to explain the comprehension of indirect anaphors might serve as a basis for meeting condition (b). Experimental results relevant to condition (a) are reviewed in the following.

1.2 Experimental Results on Indirect Anaphors

(3) The teacher was busy writing an exercise on the blackboard. However, she was disturbed by a loud scream from the back of the class and the chalk dropped on the floor. (Garrod & Terras, 2000, 529)

Example (3) shows an excerpt of one of the 24 texts that Garrod and Terras (2000) constructed

to consolidate earlier research by comparing the understanding of indirect and direct anaphors that refer to dominant and non-dominant instruments to a verb (besides further variables). A dominant instrument of a specific verb is the instrument that was most frequently mentioned by subjects of an association pretest when the verb was given. The corresponding non-dominant instrument was only infrequently mentioned given the verb. (3) is an example of a non-dominant indirect anaphor (an indirect anaphor referring to a non-dominant instrument). If *with chalk* is appended to the first sentence in (3), the anaphor *the chalk* in the second sentence will pick up that mention of chalk and be a direct anaphor instead of an indirect one. The texts for the conditions with dominant anaphors used a pen instead of chalk because writing is more commonly performed with a pen than with a chalk. In the study, first fixation durations and regression-path reading-times were analysed. The *first fixation duration* of a word in a text is the duration for which a participant's eye looked at (fixated) a word for the first time. The *regression-path reading-time* of a word in a text is the sum of the durations of all fixations on a regression path from the word. The regression path includes the first fixation on the word, subsequent fixations that do not exceed the word in reading order (i.e. to the left and/or above the word and on the word). Garrod and Terras (2000) found that for dominant instruments like the one between *writing* and *pen*, subjects' first fixation durations and regression-path reading-times on the anaphor and the following verb were not significantly longer for indirect anaphors than for direct anaphors. For non-dominant instruments, like the one between *writing* and *chalk*, first fixation durations and regression-path reading-times on the verb following the anaphor were 30 ms and 48 ms longer on average for indirect anaphors than for direct anaphors and both results were significant. An interpretation of this result is that the comprehension effort for indirect anaphors is comparable to the effort to understand direct ones if the instrument referred to by the anaphor is dominant for the verb, but harder for non-dominant instruments. This result may also be generalised: a relation from verb to instrument may be assumed that is strong for dominant instruments and weak for non-dominant instruments. There are further relations comparable to those between verbs and instruments, e.g. part-whole-relations (e.g. between FACE and NOSE) and associations between entities commonly occuring together (e.g. NURSE and DOCTOR). Indirect anaphors can also be based on these relations.

Lavigne-Tomps and Dubois (1999) describe an eye tracking study testing indirect anaphors based on association and part-whole relations. The study is problematic, though, for a number of reasons: (1) it is not clear whether the fixation durations mentioned in the text belong to the reported experiment or a previous one, (2) no statistical tests are reported, leaving open the reliability of the results, (3) the tested anaphors are in object position of the sentence, relating to the subject of the same sentence, while prototypical anaphors are in subject position, relating to referents from prior sentences, (4) there are differences in anchor-anaphor distance exhibited in the examples provided by the authors but not discussed by them that could confound the results.

Before eye tracking was available, reading times were used to operationalise difficulty of comprehension. Reading times are measured in a setup in which participants press a button to display a text line by line. The reading time is the difference between the time at which the sentence with the anaphor appears on the screen and the time when the participant presses a button to advance to the next sentence or signals completion. Reading times are assumed to be proportional to difficulty of comprehension. Haviland and Clark (1974) report an average

increase of reading time by 181 ms for indirect anaphors based on association relations compared to their direct counterparts. Singer (1979) reports an average increase in reading time of 116 ms for indirect anaphors based on verb-instruments relations compared to corresponding direct anaphors. Garrod and Sanford (1982) report an additional 50 ms reading time on average for indirect anaphors based on a verb-instrument and an associative relation compared to direct anaphors. Note that they found a reading-time difference when the associative relation had been added. No difference was found without the additional associative relation. This early differenced result is reflected in the differences between indirect anaphors based on dominant and non-dominan instruments that Garrod and Terras (2000) had reported for eye tracking.

I did not find dependable eye tracking results on the comprehension of indirect anaphors based on relations other than verb-instrument relations. There were, however, dependable results with significant reading time differences for some indirect anaphors based on verb-instrument and associative relations. It is therefore not implausible to assume that first fixation duration and regression path duration differences comparable to those found by Garrod and Terras (2000) for indirect anaphors based on verb-instrument relations can occur for indirect anaphors based on association relations or on part-whole relations. Note that source code can be used to represent all of the kinds of relations mentioned up to now, as will be shown in Chapter 2. It can thus be expected to find significant differences for indirect anaphors based on non-dominant relations and it is possible that indirect anaphors based on dominant relations as processed as quickly as the corresponding direct anaphors. All the Above mentioned studies focussed on the on-line comprehension of indirect anaphors. But how does the comprehension of direct and indirect anaphors operate off-line, after reading has been completed? How is memory affected? These questions has been addressed in research on cohesion.

1.3 Experimental Results on Cohesion

Because indirect anaphors are based on an underspecified relation, they reduce the *cohesion* of a text, i.e. they does not show in the text the underspecified relation between the referent of the anchor and the referent of the indirect anaphor. Note that indirect anaphors do not impact *coherence*, which describes the relations between referents established during reading, because the underspecified relation between the referent of the anchor and the referent of the indirect anaphor is established during the comprehension of the indirect anaphor.

In a study to replicate the so-called *reverse cohesion effect*, O'Reilly and McNamara (2007) manipulated the cohesion in a biology text on cell mitosis presented to college students. The high-cohesion materials were constructed by modifying an existing low-cohesion text: pronouns were replaced by noun phrases, elaborations and sentence connectives were added. Besides other tasks, participants answered comprehension questions that asked for information from a single sentence (text-based questions) or required integration of information from two sentences (inference-based questions). O'Reilly and McNamara (2007) found that readers with high prior knowledge and low reading skill answered text-based questions better for low-cohesion texts than for high-cohesion texts. The authors explain this reverse cohesion effect by hypothesising that low-skill readers skim more and therefore likely miss more detail in high-cohesion texts compared to low-cohesion texts. The study confirmed the possibility that reduced cohesion

can, depending on prior knowledge and reading skill, potentially improve comprehension as measured by comprehension questions.

The previous section summarised prior research of the on-line comprehension of indirect anaphors that showed that there are indirect anaphors that can be understood as quickly as direct anaphors. Above study showed that the reduction of cohesion (that could also be achieved via the use of indirect anaphors) can in some cases facilitate memory for a text as operationalised by answers to comprehension questions. Comparable effects may occur when cohesion is reduced in source code through the use of indirect anaphors, rendering condition (a) from Section 1.1 fulfillable. Next, the design and the hypotheses of the experiment of this thesis are derived from the above mentioned studies.

1.4 Design of the Experiment

The experiment of this thesis is to investigate whether condition (a) from Section 1.1 can be satisfied, i.e. whether there is a sufficient number of cases where programmers benefit from reading source code with indirect anaphors compared to normal source code. The Java programming language is chosen for normal source code and will be augmented with indirect anaphors, as has been done in Lohmeier (2011a). The experiment combines on-line and off-line measures to provide a copious picture of the comprehension of indirect anaphors in source code.

The experiment comprises five factors (condition, estimated on-line and off-line activation and question type are within-subject factors; program comprehension skill is a between-subject factor).

Condition The test condition uses code in which qualified expressions and local variables had been replaced by direct and indirect anaphors, the control condition uses the original code with qualified expressions and local variables.

Program comprehension skill Comparable to reading skill in O'Reilly and McNamara (2007), low and high program comprehension skill are contrasted, using a median split based on the score obtained from the program comprehension skill questionnaire (see Section 3.2).

Estimated on-line activation As will be explained in Section 1.6 below, the availability of relations either underspecified in indirect anaphors or explicated in qualified expressions or local variables exerts an on-line influence on eye movements during the comprehension of indirect anaphors that is modelled with the psychological construct of mental *activation*. The activation of relations immediately before the presentation of a target expression that underspecifies or explicates the relation is quantitatively estimated using the procedure documented in Section 4.6.1. A median split was used to divide target expressions into two groups of expressions with low vs. high underlying activation. This factor is equivalent to the distinction between dominant and non-dominant instruments in Garrod and Terras (2000).

Estimated off-line activation After the code will have been read, activation of relations underspecified or explicated in Java code with or without anaphors will influence recall from memory while answering comprehension questions (off-line). Quantitative estimation of

these activation values is documented in Section 3.6. Again, a median split into relations with low vs. high underlying activation was performed such that this within-subject factor reflects the between-subject factor that O'Reilly and McNamara (2007) used to tell subjects with high background knowledge apart from subjects with low background knowledge. Note that because on-line activation is only used in the analysis of eye movement data and off-line activation is only incorporated in the analysis of the comprehension questions, the word "activation" might at times be used in both cases in the following without further differentiation.

Question type The comprehension questions presented to participants after reading source code (see Section 3.6) were categorised as being either text-based or inference-based, similar to the question types in O'Reilly and McNamara (2007).

A double-balanced design was chosen, in which each participant receives two independent sequences of source code (1 and 2): one in the control (C) and the other one in the test condition (C). Four groups were designed to balance both the sequence in which test and control condition occurred as well as the sequence of source code materials: A (1T, 2C), B (2T, 1C), C (1C, 2T) and D (2C and 1T).

The following three dependent variables are used:

Comprehension question score A score is computed by the experimenter based on applying the scheme defined in Section C to participants' answers to the comprehension questions. The construction of the questions is documented in Section 3.6.

Regression-path reading-time on target expression For each target expression (an indirect anaphor in the test condition, or the corresponding local variable or qualified expression in the control condition, as defined on page 23), eye-movement data is used to compute for each participant's first encounter with the target expression a regression-path reading time, "which sums all fixation durations from first fixation of the region until the eye goes beyond that region" (Garrod & Terras, 2000, 534) and covers initial repair processes during comprehension. Note that the source codes I present are longer than the texts shown by Garrod and Terras. When determining the first regression path reading-time on a target expression I therefore ignore regressions that merely cross the target expression and started over words below or to the right of the target expression in question.

Task duration Task duration captures the overall time required to complete a task in the experiment. Each of the two sequences of source code presented to a participant contains 20 pages of source code, each of which is followed by a yes-no question. A task starts when a participant opens a page of source code and lasts until she clicked a button to answer the yes-no question that follows the page of source code. Participants are able to re-view source code introduced during earlier tasks. It is also possible to answer a question multiple times. In such a case, only the time until the first answer is considered.

1.5 Hypotheses for the Experiment

Based on the prior work and the design above, I expect the factors to exercise the following effects on the dependent variables.

A Regression-path reading-times on an indirect anaphor will be shorter, the higher the on-line activation of the underspecified relation – i.e. the more recently and frequently presented the relation – analogous to the dominance effect found by Garrod and Terras (2000).

B Regression-path reading-times on target words in control and test condition will be identical for relations with high on-line activation, analogous to the results of Garrod and Terras (2000) for dominant instruments.

C For relations with high off-line activation, the comprehension question score for text-based comprehension questions is expected to be lower for the test condition with indirect anaphors than for the control condition with local variables and qualified expressions, in line with the findings of O'Reilly and McNamara (2007) for readers with high background knowledge and low reading skill.

D1 At least for sub-tasks that involve relations with high on-line activation, task duration could be lower for the test condition with indirect anaphors than for the control condition with local variables and qualified expressions because under-specification reduces the amount of text to be read and indirect anaphors for dominant relations are not gazed at longer than direct anaphors.

D2 Alternatively, task duration could be higher for the test condition than for the control condition, if indirect anaphors are generally harder to understand than local variables and qualified expressions.

The hypotheses and the experiment will provide answers as to if and when indirect anaphors support comprehension, the subject of condition (a) from Section 1.1. If indirect anaphors support comprehension, these cases need to be identified up front, as put forward in condition (b). If up-front identification is possible, source code, that can be parsed and modified automatically, can be adjusted to show indirect anaphors only when they are beneficial. To be able to predict the comprehensibility of indirect anaphors, models of their comprehension are consulted next.

1.6 Processing Accounts of Indirect Anaphors

The following sections detail psychological and cognitive models of the processing of indirect anaphors. Formal models are not covered for they do not typically model psychological effects that can be observed during the comprehension of indirect anaphors and are therefore not expected to help forecasting anaphor comprehensibility. Neurolinguistic models were not included due to time constraints and because the model in Chapter 5 was chosen to be based on a cognitive architecture that implements theories from cognitive psychology. Note that the models below will be presented in a uniform terminology derived from three-level semantics that is presented first. The presentation of the models is focused towards the goals to get ideas of how to implement the resolution of direct an indirect anaphors in the cognitive model in Chapter 5 and how to be able to predict whether a programmer will be able to understand an indirect anaphor before the source code that contains the anaphor is displayed.

1.6.1 Three-level semantics

Three-level semantics (Schwarz, 1992) is a cognitive theory of meaning construction that has been applied to systematise indirect anaphors (Schwarz, 2000; Schwarz-Friesel, 2007) based on the knowledge structures indirect anaphors are anchored in. Three-level semantics describes knowledge representations and processes operating on them at a notional mental level that seeks compatibility with neuro-linguistic evidence (Schwarz, 1992, 39ff.), but does not model the brain. Three-level semantics is based on the idea that a reader constructs a mental representation while reading a text that integrates context-independent conceptual and lexical-semantic information from long-term memory in a text-world model (TWM) that represents context-specific current meanings in nodes and relations between them. While the conceptual level is mode- and language-independent, the lexical-semantic level contains language-specific graphemic, syntactic and phonetic information. The text-world model constitutes the third of the levels of three-level semantics. The TWM is a localist connectionist network: unlike artificial neural network, the nodes of a TWM represent units of meaning. The nodes have an activation that influences whether and if so how quickly the nodes can be retrieved. Activation can spread along the relations that connect the nodes. It is an important aspect of three level semantics that meaning is represented in the form of semantic features at all three levels. A conceptual schema can represent a type or an instance of a type (a token) and contains all features related to a concept. Lexicon entries are related to the subset of features of related schemata that are features of all instances of the concept. TWM nodes have a limited capacity for features so they can be used to represent working-memory limitations. The nature of semantic features themselves is not defined in three-level semantics. I find it convenient to assume that semantic features are themselves conceptual schemata because this adds a recursive relation to semantic features and conceptual schemata that may correctly reflect the difficulty of defining atomic semantic features. It also adds the possibility to have semantic features that cannot be named, i.e. semantic features that are concepts that have no lexicon entry and hence no phonologic or graphemic representation that would describe how to pronounce or spell them.

The nodes in the TWM in three-level semantics are the referents of words whose reference potential has been used by the reader to *referentialise* the words. Example (2) can be used to describe the anchoring of an indirect anaphor in three-level semantics. The definite article "the" of the noun phrase "the hard disk" signals to the reader that the phrase should refer to a referent existing in the text-world-model. A lexicon entry HARD DISK is activated by "the hard disk". This also activates the HARD DISK DEFAULT that is part of the lexicon entry of COMPUTER which has previously been activated when the anchor "the computer" was read. The exact details of this have not yet been described, but a new TWM node is created from the HARD DISK DEFAULT and the HARD DISK lexicon entry activated by "the hard disk". Semantic features from both lexicon entries and potentially from the text (consider reading "the external hard drive" that would contribute the feature EXTERNAL) are added to the newly instantiated TWM node via a process called *specification* (see Schwarz 1992, 127). Because the capacity of a TWM node is limited, not all features from the HARD DISK DEFAULT and the HARD DISK lexicon entry are added to the new TWM node and relevant ones are *selected* (see Schwarz 1992, 125ff.). The representation of a part-whole relation between the referent of the anchor and the newly instantiated TWM node which is the referent of the indirect anaphor is

created to permit activation to spread between the two (such coherence relations are the reason why activation is retained longer during text reading than in list reading).

Schwarz (2000) was relevant for the construction of the compiler for indirect anaphors in Java (Lohmeier, 2011b), because the symbolic knowledge structures it describes are similar to the way knowledge is represented in Java source code, as will be detailed in Section 2.3. Consequently, her classification of indirect anaphors is mirrored in Section 2.4. The resolution of indirect anaphors with activation and other psychological properties laid out in Schwarz (2000) has not yet been implemented in a computer, besides a rudimentary ACT-R model in a term paper (Lohmeier, 2013). One important reason was the absence of knowledge representations to use in the model. This hindrance is not present for source code that is a knowledge representation that is relatively complete because the compiler requires it. Three-level semantics lacks a number of details (e.g. a specific algorithm to compute activation values and a direct connection to eye movement parameters) but provides a general framework of knowledge representations and cognitive processes as well as the general concept of activation that influences availability and retrieval times for TWM nodes. These provide a basis to work towards satisfying condition (b) from Section 1.1.

1.6.2 Bonding and resolution

Garrod and Terras (2000, 540ff.) textually outline a model to explain their experimental results on eye movements during the comprehension of anaphors related to verb instruments. For the sake of simplicity, I do not reproduce their terminology and reasoning as would be required in a critical examination of their model, but merely reproduce the results of their argument using the terminology of three-level semantics. I also limit the description to the effect relevant to the experiment of this thesis and omit the context effect that Garrod and Terras also explain.

Garrod and Terras distinguish between lexical activation processes they call *bonding* and processes operating on the text-world model that they call *resolution*. When a word functioning as indirect anaphor is fixated by the reader's eye, the graphemic representation of the word's entry in the mental lexicon of the reader is used to retrieve the lexicon entry of the word from the mental lexicon. When this process has been completed, the eye proceeds to fixate the next word. All subsequent *post-access* complications of processing will thus be reflected in first-fixation and regression-path reading-times of the following word, but not – which is why Garrod and Terras did not find a significant effect on first-fixation reading times – on the indirect anaphor. After lexical access has been completed, the lexicon entry of the noun that functions as indirect anaphor receives activation that is passed on to semantically-related lexicon entries using forward-relationships and is returned (at no time) from these lexicon entries via backward-relationships. Traversal of each relationship reduces or amplifies the activation in a way that reflects the strength of the relation between the two lexicon entries. As Garrod and Terras had determined using an association pre-test, the lexicon entries of nouns that function as indirect anaphors and are to refer to dominant and non-dominant instruments are linked to the antecedent verb's lexicon entry using equally-active relations. However, the relation from the antecedent verb's lexicon entry to the lexicon entry of the indirect anaphor is more active for nouns that are to refer the dominant instruments of the verb than for nouns that are to refer to non-dominant instruments of the verb. A new TWM node is then instantiated from the de-

fault instrument stored in the lexicon entry of the verb and the lexicon entry of the noun that functions as an anaphor. Instantiation of the default takes less time, the higher the activation of the default and the indirect anaphor's lexicon entry. For dominant instruments, the activation is higher because they receive more of the activation that they sent to the verb's lexicon entry than non-dominant instruments do. In the case of non-dominant instruments, this at least delays access to the lexicon entry of the following word, prolonging the first-fixation reading-time on the following word and thus also its regression-path reading-time that at least contains the first-fixation reading time. Note that Garrod and Terras (2000, 539) did not find a significant increase in the frequency of regressions for indirect anaphors referring to non-dominant instruments. The processing of direct anaphors differs from the processing of indirect anaphors in that no default is instantiated because the referent of the antecedent has a higher activation than the default instrument and the lexicon entry of the lexicon entry of the anaphor and therefore the referent of the antecedent is re-activated.

Garrod and Terras provide a detailed account of the on-line processing of direct and indirect anaphors based on differently activated knowledge relations that exceeds what had been detailed in Schwarz (2000) and unlike Schwarz, Garrod and Terras connect their model to observable eye movements. Their model explains how lower activation of the relation between the lexicon entry of a verb and the lexicon entry of an indirect anaphor causes longer first-fixation and regression-path reading times on the following word. The experiment of this thesis does not use dominant and non-dominant instruments, but instead varies the activation of the relation underspecified by the indirect anaphor by manipulating how often and how recently it has been explicated in the source code presented before the anaphor is read. I therefore do not model the bi-directional activation spreading that Garrod and Terras used, but directly vary the activation of the instrument default.

With regard to condition (b) from Section 1.1, the model of Garrod and Terras contributed the interpretation that long first-fixation and regression-path reading-times on the word following an anaphor are caused by low activation that slows processing of the anaphor. Using a cognitive model that computes activation values, it is possible to use the activation values of potential referents of indirect anaphors when a new source code is opened in an editor to determine whether it will be too hard to comprehend certain indirect anaphors and to display a local variable or a qualified expression instead of the anaphor.

1.6.3 Construction-integration and predecessors

Schmalhofer, McDaniel, and Keefe (2002) modelled a naming-task experiment of Keefe and McDaniel (1993) in which participants read sentences and were later asked to pronounce a probe word that either occurred in the sentences, or that described a consequence of the situation described in the sentences. The probe word was thus comparable to direct and indirect anaphors, even though the probe word itself was not embedded in a sentence. The latency to pronounce the probe word was taken as an indicator of processing difficulty. Schmalhofer et al. conceptualised processing in the form of inferences. Inferences comparable to what is required to anchor indirect anaphors they called *bridging inferences*. The model implemented by Schmalhofer et al. (2002) belongs to the construction-integration family of models (Kintsch, 1988) that is comparable to three-level semantics in the sense that there are localist networks as symbolic knowledge

representations that are modified by a set of rule-like productions. The model exhibits qualities not available in the previously described model of Garrod and Terras (2000): Schmalhofer et al. created representations of the text surface, propositions and the situation model manually but systematically. The representations were created for two representative texts, yielding two different models only, because further modelling would have required too much time, according to the authors (Schmalhofer et al., 2002, 119). Activation processes were simulated on the representations to compute activations at the end of each clause of the modelled texts. A formalized definition of activation is used to compute activation values iteratively at the end of each clause, until activation values for the nodes in the knowledge representation stop changing (Schmalhofer et al., 2002, 115). Assuming that activation translate to word naming latency, the authors qualitatively compared activation and naming latency and found them sufficient to regard the model as an explanation of the empirical results (Schmalhofer et al., 2002, 118).

Predecessors of Kintsch (1988) have already been used decades before the work of Schmalhofer et al. by Theo S. Mandel to compare cognitive models to experimentally derived eye movements during reading. Mandel (1979) reported that more important propositions in a propositional representation of a text constructed according to Kintsch (1974) receive more regressions and fixations in a reading experiment. Later, in Mandel (1984), he compared eye movement records to memory operations performed by the computational model of Miller and Kintsch (1980) that followed Kintsch and van Dijk (1978) when it processed the materials read by participants of the experiment. Mandel (1984, 31ff.) highlighted that regressions between sentences happened in the experiment when the model performed more memory operations to access information introduced in previous sentences. Running the model before another experiment would have provided a forecast for the processing difficulty of the text instead of an explanation for the difficulty found in the experiment.

Before this thesis I was wondering why I was unable to find works that input eye movement records into cognitive models. Now that I completed the thesis, I know that it is quite some work to get done that cannot be converted into applications as long as the knowledge representations used in the models cannot be derived from naturally occurring text automatically and as long as eye tracking is not widely used outside labs. However, it has been shown that construction-integration models and their predecessors can explain experimental data, so this can be expected for the similar three-level semantics as well. It is furthermore suitable to apply the approach to source code from which knowledge structures can be derived automatically. Moreover, eye tracking is currently becoming cheaper, so it makes sense to base a model on eye movements.

1.6.4 The landscape model with LSA vectors

Yeari and van den Broek (2013) applied a comparable approach based on the landscape model (Tzeng, van den Broek, Kendeou, & Lee, 2005). Again a graph-based knowledge representation is constructed not from propositions manually extracted from a text, but using the vectorial representation of latent semantic analysis (LSA; Landauer 2007) that captures the corpus-based similarity of words. The similarity value from the LSA vectors is used as strength of the edges in the knowledge representation and activation is spread among the nodes of the knowledge representation. Activation is again computed in cycles (after each clause or sentence) until it settles. Yeari and van den Broek (2013) found activation to correspond to reaction time data

from a bridging inferences study. While this model does not, unlike the previous one, represent cohesive relations expressed in the text, these relations are implied by the similarities obtained from LSA. While LSA vectors are not available off-the-shelf for all words, them being based on corpus data reduces the influence the modeller can exercise over the model. The landscape model also shows that it is possible to model knowledge-based effects on behavioural data during reading without modelling high-level cognitive processes as is done in construction-integration models and in cognitive architectures in general.

1.7 Hypotheses for the Cognitive Model

While three-level semantics provides a general framework for meaning construction during text comprehension, bonding and resolution connects activation during text comprehension to first-fixation and regression-path reading times. Construction models and their predecessors, that are similar to three-level semantics, have been implemented in computers and were used to explain empirical data, even eye movements. Applications of the landscape model has shown that knowledge representations in models can be derived automatically and models can provide explanations for effects in text comprehension even when no high-level cognitive processes are modelled. It thus seems feasible to implement a comparable model for the source code reading experiment of this thesis to explain the results of the experiment and to forecast comprehensibility of source code.

Besides the variables defined in Section 1.4, two further factors are added to be able to formulate hypotheses concerning the ACT-R-based model of the experiment described in Chapter 5.

Computed on-line activation Besides being computed by the ACT-R model, this is identical to the estimated on-line activation variable defined in Section 1.4.

Computed off-line activation Besides being computed by the ACT-R model, this is identical to the estimated off-line activation variable defined in Section 1.4.

Based on the review of the existing models, the following hypotheses are put forward regarding the ACT-R model.

E The model re-generates the regression-path reading-times measured during the experiment.

F The computed on-line activation predicts the on-line comprehension of indirect anaphors: Referents of indirect anaphors that depend on an underspecified relation for which the model computed a low activation when the indirect anaphor is read, lead the model to re-generate a longer regression-path reading-time than in the case of indirect anaphors based on a highly active relation. (This hypothesis depends on whether the experiment shows a difference in regression-path reading-times for indirect anaphors based on relations with high vs. low estimated on-line activation.)

G The model predicts whether an indirect anaphor, or a local variable or qualified expression should be shown: When a new source code is opened in the editor, the computed on-line activation of the underspecified relations of indirect anaphors contained in the source code is lower for indirect anaphors with low estimated on-line activation than it is for indirect anaphors with high estimated on-line activation.

H The off-line activation computed by the model predicts the off-line comprehension effects of indirect anaphors: The activation of an underspecified relation computed at the end of the execution of the model of a participant correlates with the answer score the participant obtained for the comprehension question that targets the underspecified relation.

1.8 Summary

In this thesis I evaluate experimentally how programmers understand indirect anaphors in source code and attempt to predict the comprehension of indirect anaphors in a cognitive model. Because indirect anaphors underspecify a relation, they might ease comprehension, but they will be hard to comprehend for those who do not know the underspecified relation. Indirect anaphors could thus be used as an input method only. Alternatively, indirect anaphors could also be shown during reading, if (a) they benefit readers sufficiently often, and (b) such cases can be identified before the code is presented to the reader. Prior experiments in psycholinguistics indicate that condition (a) can be fulfilled with regard to on-line and off-line effects of comprehension. The design of the experiment in this thesis follows the design of the existing studies. Existing cognitive models of text comprehension render the prediction of the comprehension of indirect anaphors in source code achievable based on the activation values computed in such models. Before proceeding to the experiment and the model, the next chapter details indirect anaphors in source code.

2 Anaphors in a Cognitive Linguistics of Programming

The anaphors in source code that I examine in the experiment, are modelled after the conception of anaphors in cognitive linguistics in line with the models discussed in Section 1.6. Note that I conceptualise the anaphors as part of a cognitive linguistics of programming – not of programming languages – to highlight that programmers do not only actively type when writing source code, but even actively construct mental representations during program comprehension. This approach is likely most remote to common conceptualisations of programming languages in theoretical computer science that do not include a reader in their models of meaning of code and assume that the meaning is in the code itself.

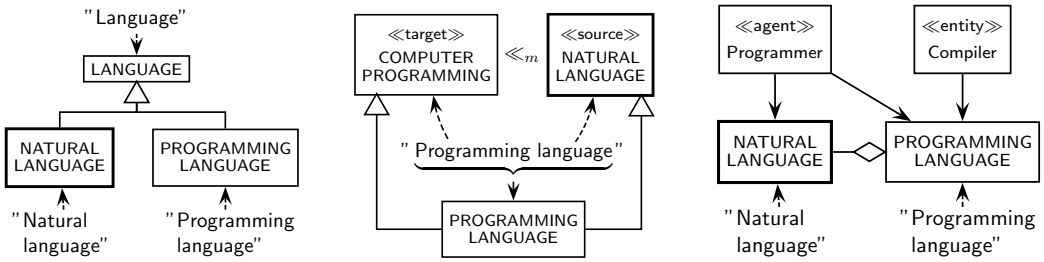
To motivate anaphors in source code, this chapter starts with a discussion of relations between natural language and programming languages. I also treat the cognitive gap between compilers and programmers and its consequences for anaphors in source code, as well as the experiment and the cognitive model of this thesis. The representation of knowledge in Java source code is explained, as well as the classification of indirect anaphors in source code that rests upon it.

2.1 Relations between Natural Language and Programming Languages

From a constructivist position, there is no unalterable external reality. What humans perceive is at least partly fabricated by their individual minds. Hence, different ontological relations between natural language and programming languages can be constructed and can be shown to make sense, as will be done in this section. I will not distinguish between different natural languages and will ignore spoken language. Hence, the following sections are based on the idea that the language that humans read and write is called *natural language*. There are three relations between natural language and programming languages that are related to ontological status, i.e. related to what programming languages are and to what natural language is. Each relation is introduced in one of the following subsections.

2.1.1 Sibling Concepts

Natural language and programming languages can be thought of as two sub-concepts of the same superordinate concept of language as illustrated in Figure 2.1(a). The figure illustrates relation in the style of the unified modelling language (UML) as well as boxes with capitalized labels that show idealised conceptual schemata referred to by the words in quotation marks.



(a) Sibling Concepts (b) Is-Like (c) Part-Whole

Figure 2.1: Relations between natural language and current programming languages

It is hard to find a definition of the meaning of the word *language* that is sufficiently abstract to encompass verbal *natural language*, sign language and *programming languages*, yet is specific enough to exclude telephones. Schank (1999, 4) gave an interesting one: "Language is a memory-based process. It is a medium by which thoughts from one memory can, to some extent, be communicated in order to influence the contents of another memory." This definition of language covers programming languages in that a human programmer can write a program that a compiler transfers into an abstract syntax tree (AST). An AST can be regarded as a kind of memory that the compiler modifies according to further instructions in the source code that reflects what the programmer had in mind when writing the program. Hence, one can say that natural language and programming languages are sibling concepts given Schank's definition of language. Even if Schank's definition is not known in informatics, the idea that natural language and programming languages are sibling concepts, seems to be the most common conception of the relation between them. (I do not assert that informaticists are conscious about this conceptualisation.)

Note that natural language is the most well-known language, which means that humans will typically regard it as the prototypical language. That means that I assume that when the word *language* is mentioned in a neutral context, one typically thinks of natural language and not of an abstract concept of language. The prototypicality of natural language is in Figure 2.1(a) expressed by the broader margin.

2.1.2 Is-Like

The compound noun *programming language* can function as a compound metaphor whose second part *language* is metaphorical, as illustrated in Figure 2.1(b). The parts of a compound metaphor refer to different concepts and the use of the metaphor (if it is not a dead or lexicalised metaphor) implies an IS-A relation between the target concept (here: COMPUTER PROGRAMMING) and the the source concept (here: NATURAL LANGUAGE, the prototypical meaning of *language*). Because there is no IS-A relation between COMPUTER PROGRAMMING and NATURAL LANGUAGE, the relation is interpreted as an IS-LIKE relation and semantic features of NATURAL LANGUAGE are transferred to COMPUTER PROGRAMMING, as illus-

trated by $\ll_m$ in the figure. To understand *programming language* as compound metaphor might typically happen in the case of novices at computer programming. Since a metaphor does not particularise the relationship between the two concepts, it is up to the reader to infer specifics, i.e. that in this case both COMPUTER PROGRAMING and NATURAL LANGUAGE involve writing, that programming can be seen as trying to "tell" the compiler what to "do", using "statements" and "expressions", etc. After a while the metaphor becomes lexicalised, i.e. is added to the mental lexicon and a concept of PROGRAMMING LANGUAGE may be created that draws parts of it semantic features from the COMPUTER PROGRAMMING and NATURAL LANGUAGE concepts. The subconscious comprehension process triggered when an expression is read that functions as metaphor is comparable to the conscious interpretation process engaged when reflecting about the features of natural language that are found in or should be added to programming languages. Anaphors are such a feature.

2.1.3 Part-Whole

Finally, fragments of natural language occur in programming language, as illustrated in Figure 2.1(c). A form of this appearance that programmers are well aware of, are keywords like `if`, `class`, and `new`, that are implemented by compilers in a way that at least partially matches their use in natural language. String literals, comments and identifiers used to name variables, methods, classes and other entities in programming languages are problematic, though. They contain fragments of natural language whose meaning is not accessible to the compiler, but to the programmer only. This creates a gap between programmer and compiler: The programmer can be described as an autonomous intelligent agent who is able to understand both natural language and programming languages; the compiler on the other hand is merely an entity that is able to process programming languages according to given rules. Because above differential characterisations use the terminology of cognitive science, I call the gap a *cognitive* one.

2.2 The Cognitive Gap between Programmer and Compiler

Identifiers are well suited to illustrate the cognitive gap between programmer and compiler that is caused by natural language occurring in programming languages. (The following statements are restricted to Java and comparable object-oriented programming languages.) The Java language specification recommends the following naming conventions for identifiers in Java source code.

1. "Names of class types should be descriptive nouns or noun phrases, not overly long, in mixed case with the first letter of each word capitalized." (Gosling, Joy, Steele, & Bracha, 2005, 147)
2. "Method names should be verbs or verb phrases" (Gosling et al., 2005, 149)
3. "Fields should have names that are nouns, noun phrases, or abbreviations for nouns." (Gosling et al., 2005, 150)
4. "Local variable and parameter names should be short, yet meaningful. They are often short sequences of lowercase letters that are not words." (Gosling et al., 2005, 151)

These conventions reflect programming practice and thus identifiers typically contain combinations of words from natural language that have a meaning to programmers. How are identifiers processed, e.g. identifiers that refer to variables, be them local variables or the fields of classes? Given the identifier, a compiler will perform a case-sensitive search for a declaration of a variable with the given identifier. How do programmers understand identifiers? I commit to the plausible assumption that three-level semantics is a suitable model of source code comprehension just as much as it is a suitable model of text comprehension. I further assume a situation in which a programmer has read either a declaration or a prior occurrence of the identifier before the occurrence of the identifier that is now being processed. The programmer will then activate a lexicon entry whose graphemic representation fits the identifier and the semantic features of the lexicon entry re-activate an existing TWM node that shares at least some of the semantic features of the lexicon entry. The difference between the processing in the compiler and in the programmer is that the programmer can resolve identifiers like anaphors that refer to referents that have not been declared but simply been read before. Consider the code with the class instance creation expression below:

```
1 new Voucher(id, creator, title, description, value, location);
```

Listing 1: Creating a voucher

If comprehension of this expression is modelled according to three-level semantics, a new TWM node VOUCHER-1 is created that represents a voucher with a number of potential parts and associations provided as arguments. Putting aside thoughts about whether or not the programming language works like this, a programmer would easily be able to comprehend that an identifier "voucher" encountered in the next line refers to the voucher from the first line:

```
1 new Voucher(id, creator, title, description, value, location);
2 writeVoucher(space, voucher);
```

Listing 2: Creating a voucher and referring to it

In three-level semantics, "voucher" functions as direct anaphors and the comprehension of "voucher" is explained as a re-activation of the existing VOUCHER-1 node representing the voucher created in the first line. A Java compiler does however require the local variable to be declared that has a declared type and declares the identifier of the local variable:

```
1 Voucher voucher = new Voucher(id, creator, title, description, value,
      location);
2 writeVoucher(space, voucher);
```

Listing 3: Voucher in a declared local variable

If direct anaphors like "voucher" were added to source code, the rules applied by the compiler to resolve referents of identifiers would need to get closer to human comprehension. This would in Listing 2 even work without considering semantic features. Instead, a case-insensitive match of the identifier to the type name used in the class instance creation expression would suffice.

Since programmers who read the previous source code snippets do by now know that a voucher has an ID, they might also be able to understand which ID is meant in the following snippet.

```
1 new Voucher(100, creator, title, description, value, location);
2 write(id);
```

Listing 4: Referring to the ID of a voucher

Because the creation of the TWM node that represents the voucher in the first line of above snippet involves a lexicon entry abstracted from previous encounters of vouchers in this series of examples, the VOUCHER-1 TWM node contains a semantic feature that represents the ID of VOUCHER-1. Human programmers are able to understand that the "id" in the next line refers to the ID that is part of the VOUCHER-1 TWM node. The "id" thus functions as an indirect anaphor. The compiler is again not able to resolve this anaphor, because it does not consider the semantic features of previously processed variables when searching for the referent of the "id" identifier. To enable the compiler to process indirect anaphors, it would need to consider knowledge representations and potentially activation values.

A compiler able to resolve indirect anaphors needs to access the knowledge encoded in the source that specifies which relations exist that may be underspecified by indirect anaphors. The representation of this knowledge is described in the upcoming Section 2.3. It is clear that a compiler and also a programmer can only process relations available to them. Programmers might have a relation in mind when writing an indirect anaphor that is not represented in the source code. If there is no other relation that the indirect anaphor can be based on, the compiler will show an error message. If there is another relation available to the compiler that differs from the relation the programmer has in mind, this error will go unnoticed if the compiler does not show to the programmer what referent it anchored the indirect anaphor in and the programmer does not pay attention to the signal of the compiler.

There are a number of ways in which a compiler can determine possible anchors for the indirect anaphor "id" in Listing 4. The compiler could visit the referents of all words occurring before the indirect anaphor, i.e. the referents of "100", "creator", "title", "description", "value", "location", and of "new Voucher(100, creator, title, description, value, location)". If more than one of these referents provided an ID, an error could be reported by the compiler to avoid any possible referential ambiguity. Participants of the experiment of this thesis were told that indirect anaphors are compiled like this. Alternatively, a compiler that narrows the cognitive gap further, could compute activation values and search those referents for an ID, that are active enough to consider the possibility that the programmer might have anchored the "id" anaphor using those referents. To illustrate this strategy, consider a scenario in which not only VOUCHER-1 has an ID, but also the LOCATION-1 TWM node referred to by "location" in Listing 4 has an ID. In this scenario, activation is computed based on eye-tracking measures and "location" has not been fixated, but "new Voucher(" has. Then the VOUCHER-1 instantiated when "new Voucher(" was fixated is active as would be the ID that is part of the VOUCHER-1 node. Because the referent of "location" has not been fixated and thus did not receive further activation, the ID that is part of the LOCATION-1 referent could have an activation too low to be retrieved so that there is no referential ambiguity in this scenario. (There would be referential ambiguity if the ID of LOCATION-1 is sufficiently active to be retrievable.) The cognitive model described in Chapter 5 computes activation in the way described.

Because activation is not stable over time, it is not correct to mention the compiler when potential anchors of an indirect anaphor are to be selected based on their activation. Currently

there are two different phases in programming: compile-time and run-time. At compile-time, the compiler converts the source code into an executable format. At run-time, the code is executed. Both phases are independent of the cognition of the developer and of the time course of activation of referents in a model of the developer's cognition. It is therefore appropriate to add a new phase before compile-time: *edit-time*, the time when the programmer edits or reads the source code. During this editing process, activation values can be computed continuously and can be used to aid decisions on where to anchor an indirect anaphor or whether to display indirect anaphors, or local variables or qualified expressions. It should be at edit-time that the indirect anaphor is translated into a local variable or a qualified expression that preserves the meaning of the indirect anaphor. This avoids compile-time errors analogous to the following scenario based on Listing 4: only the VOUCHER-1 TWM node has an ID, the LOCATION-1 node does not and the anaphor "id" has been anchored in "new Voucher" and refers to the ID of VOUCHER-1. While the indirect anaphor is still displayed in the editor, the code actually submitted to the compiler stores the referent of the VOUCHER-1 TWM node in a local variable and accesses the ID using the qualified expression "voucher.id" as shown in Listing 5.

```
1 Voucher voucher1 = new Voucher(100, creator, title, description, value,
      location);
2 write(voucher1.id);
```

Listing 5: Referring to the ID of a voucher after edit-time transformation

Now an ID is added to the LOCATION-1 node by modifying the underlying class declaration. While this would introduce referential ambiguity were the indirect anaphor anchored at compile time, anchoring at edit-time leaves previously entered indirect anaphors unaffected by later changes to the structure of the code. Such edit-time translations are supposedly implementable in all major integrated development environments (IDEs) at least for Java that all come with editors that update an abstract syntax tree while the displayed code is modified.

While this section mainly dealt with identifiers, it is relevant to the estimation of activation that comments and string literals can also contain anaphors. The cognitive model in Chapter 5 will thus handle such expressions that will also be contained in the materials of the experiment.

To summarise, anaphors in source code should be anchored at edit-time by the editor of the IDE to avoid changes to the code affect previously entered indirect anaphors. At edit time it is also possible to anchor indirect anaphors based on computed activation values should it be desired. The editor needs access to the knowledge represented in the source code, which should be in all major Java IDEs. Care needs to be taken to design the editor in a way that enables developers to detect when the editor determined an incorrect referent. The cognitive model implemented in this thesis computes activation values from eye movements. The experiment will contain anaphors in comments and string literals besides identifiers and the cognitive model will determine referents for such anaphors, too.

2.3 Knowledge Representation in Java Source Code

Declarations of classes and interfaces in Java like those in Listings 6 and 7 are equivalent to conceptual schemata in three-level semantics. Note that I do not model lexicon entries along with

the conceptual schemata, because source code is typically monolingual which removes the most obvious need for lexicon entries separate from conceptual schemata for now. Because synonymy is unusual in source code, conceptual schemata carry a single graphemic representation of the word they belong to. The nodes of the TWM are constructed on demand by the model when fixations are processed by the model that are assigned to expressions inside method declarations.

```
1  public class OutriggerImpl implements OutriggerServer {
2
3    private OutriggerServer ourRemoteRef;
4    private static SpaceProxy SpaceProxy;
5
6    public static void main(String[] args) {
7      if (System.getSecurityManager() == null)
8        System.setSecurityManager(new SecurityManager());
9      Configuration config = ConfigurationProvider.getInstance(args,
           OutriggerImpl.class.getClassLoader());
10     LookupLocator locator = new LookupLocator("localhost", 4160);
11     ServiceRegistrar registrar = locator.getRegistrar();
12     Log.log(Level.INFO, "Found registrar: "+registrar);
13   }
14 }
```

Listing 6: A class declaration in Java

```
1  public interface JavaSpace extends Service {
2    public Lease write(Entry entry, long lease)
3      throws RemoteException;
4    public Entry take(Entry tmpl, long timeout)
5      throws RemoteException;
6    public EventRegistration notify(Entry tmpl, RemoteEventListener listener,
         long lease)
7      throws RemoteException;
8  }
```

Listing 7: An interface declaration in Java

The following kinds of relations typical for lexical and conceptual knowledge representations can be identified in Java source code. These relations are among the semantic features that are used in three-level semantics.

Hypernymy The relation from a concept to its super-ordinate concept is expressed using the "extends" and "implements" keywords. Listing 7 shows that "Service" is the name of the concept super-ordinate to the concept named "JavaSpace". In Listing 6 "OutriggerServer" is the name of the super-ordinate concept of the concept named "OutriggerImpl".

Thematic roles The parameters of methods and constructors are comparable to verb arguments. A verb argument can have a thematic role, one of them being INSTRUMENT, the role that the anaphors of Garrod and Terras (2000) re-activated. Unlike verb arguments, the parameters of methods and constructors in Java do not have declared default values. Invocations of methods thus need to specify arguments to all parameters of the method or constructor. Because they lack defaults, it makes no sense to anchor indirect anaphors in

the parameters. It is however possible to anchor indirect anaphors in the value returned by invocations of methods and in the object created by constructor invocations like "new SecurityManager()" in Listing 6.

Meronymy The relation from parts to wholes is in Java expressed via declarations of field of a class like "private OutriggerServer ourRemoteRef" in Listing 6 that represents that instances of the concept named "OutriggerImpl" have an instance of the concept named "OutriggerServer" that is also called "ourRemoteRef". Part-whole relations can also be represented via accessor methods like "System.getSecurityManager()" that provides access to the security manager that is part of the system and "System.setSecurityManager(SecurityManager)" that sets the security manager instance provided as argument of invocations of this method (both accessor method appear in Listing 6). But see the next entry.

Association Concepts can be related to other concepts, even if there is no part-whole relation between them. E.g. NURSE and DOCTOR often occur together but they are not inseparable. In this case an association relation is modelled between the two. In Java, associations are also modelled as fields and accessors and are thus indistinguishable in the knowledge representations generated from Java source code.

Using representations of the above mentioned relations in source code, it is possible to construct indirect anaphors in an extension of the Java programming language that can be anchored in fields, accessor methods or invocations of methods that return a value.

2.4 Anaphors in Java source code

The kinds of anaphors used in the experiment of this thesis are based on the kinds of anaphors used in Lohmeier (2011a) that were developed from Schwarz (2000). Unlike the examples in Section 2.2, these anaphors are marked with a dot preceding a type name, e.g. ".Service.ID". The dot is equivalent to the definite article "the" in English and tells anaphors in source code apart from references to types. The named type is used to find a referent for the anaphor. This approach works both for humans as well as for the compiler implemented in Lohmeier (2011a). While that compiler processed only indirect but no direct anaphors, the experiment of this thesis also contained direct anaphors that can be used to continuously refer to existing referents or referents that have been brought to foreground via an indirect anaphor. Like in natural language, a single syntactic form is used for both direct and indirect anaphors. E.g. ".Service.ID" can be a direct anaphor whose antecedent is a local variable of the type named "Service.ID". Alternatively, ".Service.ID" can be an indirect anaphor anchored e.g. in a method invocation that returns an instance of the type named "Service.ID". There are six kinds of anaphors: three direct ones and three indirect ones.

2.4.1 DA1R: Recurrence

The clearest form of direct anaphors is one that repeats a type name that is also present in its antecedent. This form of direct anaphor can be related to parameter declarations, local variable declarations, and other anaphors. Note that invocations of factory methods like "Uuid.generate()"

and “Configuration.getInstance()” are also subsumed under this kind because the name of the type of the anaphor recurs in the qualifiers “Uuid.” and “Configuration.” of the qualified method invocations.

A local variable used to store the value created by a factory method can (e.g. `id`)

```
Uuid id = Uuidy.generate();
Voucher voucher = new Voucher(id, creator, title, description, value,
    location);
```

can be replaced by a direct anaphor of kind DA1R (e.g. `.Uuid`)

```
Uuid.generate();
new Voucher(.Uuid, creator, title, description, value, location);
```

Note that direct anaphors of kind DA1R typically refer to declarations or uses of parameters or local variables or to other anaphors, as can be seen from Table B.1 on page 90 that lists all direct anaphors of kind DA1R that occur in the test condition of the experiment.

The two code samples mentioned above also serve to illustrate the different forms of *target expressions* that occur in the experiment. Target expressions are expressions that are anaphors or that anaphors were created from. These expressions will be areas of interest for the eye-tracking analyses of the experiment. In the control condition, the “id” from the upper code sample functions as *replaced target expression* or *control target expression*. The code of the test condition shown in the lower code sample has been derived from the code in the upper sample by replacing the replaced target expression by the anaphor “.Uuid” that functions as a *test target expression* during eye tracking.

2.4.2 DA1C: Newly Constructed Instance

This kind of direct anaphors is psychologically identical to the DA1R kind in that the type used in the anaphor recurs in the antecedent. Because only class instance creation expressions are suitable antecedents for this DA1C kind of direct anaphors, this kind is technically different from the DA1R kind and hence distinguished.

A local variable used to make available the instance created by a class instance creation expression (here: `c`)

```
Client c = new Client(args);
c.init();
```

can be replaced by a direct anaphor of kind DA1C (here: `.Client`)

```
new Client(args);
.Client.init();
```

Table B.2 on page 91 lists all direct anaphors of kind DA1C that occur in the experimental materials of the test condition.

Note that the categorisation of anaphors introduced in this section slightly differs from the categorisation used in Lohmeier (2011a), where categorisation was based on how a compiler processes source code. Here, categorisation is based on how programmers comprehend source code. For instance kind DA1C was in Lohmeier (2011a) regarded as an indirect anaphor, because a local variable opaque to the programmer may need to be introduced during compilation to save

the created instance for later reference to it. Here, DA1C is considered a direct anaphor on the other hand, because the type name used in the anaphor also occurs in the constructor invocation that functions as antecedent. The recurrence (i.e. repetition) of the type name is why DA1C is considered a direct anaphor in this work.

2.4.3 DA1Gr: Getter return value

The return value of the invocation of a getter method can be referred to using a direct anaphor of kind DA1Gr. A getter method has no parameters and a name that is identical to the name of its return type prefixed by "get" and potentially by further prefixes. The DA1Gr kind of direct anaphors is slightly different from the DA1R kind because it refers to the return value of a method and the recurrence is only partial (the prefix "get" and the further prefixes do not recur).

A local variable used to make available the value returned by a getter method (here: `location`)

```
Location location = getMyLocation();
new Voucher(null, null, null, null, null, location);
```

can be replaced by a direct anaphor of kind DA1Gr (here: `.Location`)

```
getMyLocation();
new Voucher(null, null, null, null, null, .Location);
```

Table B.3 on page 91 lists all direct anaphors of kind DA1Gr that occur in the experimental materials of the test condition.

2.4.4 IA1Mr: Method return value

For methods that are not getter methods, the method name is not (largely) identical to the name of the return type of the method as in the case of direct anaphors of kind DA1Gr introduced above. Because the method invocation that is the related expression of this kind of anaphor does not invoke a method whose name recurs the name of the type of the anaphor, the anaphor is an indirect anaphor.

A local variable used to make available the value returned by a method (here: `registration`)

```
ServiceRegistration registration = registrar.register(item, Lease.ANY);
Log.log(Level.INFO, "Got service id:"+ registration.getServiceID());
```

can be replaced by an indirect anaphor of kind IA1Mr (here: `.ServiceRegistration`)

```
registrar.register(item, Lease.ANY);
Log.log(Level.INFO, "Got service id:"+.ServiceRegistration.getServiceID());
```

Table B.4 on page 92 lists all indirect anaphors of kind IA1Mr that occur in the experimental materials of the test condition.

2.4.5 IA2F: Field declaration

The value of a qualified field access expression like `Instance.spaceProxy` can be stored in a local variable

```
Instance = new OutriggerImpl(config);
SpaceProxy spaceProxy = Instance.spaceProxy;
ServiceInfo info = new ServiceInfo("Outrigger", "Sun Microsystems, Inc.",
    "2.2.2");
ServiceItem item = new ServiceItem(null, spaceProxy, new Entry[] { info });
```

or be accessed directly

```
Instance = new OutriggerImpl(config);
ServiceInfo info = new ServiceInfo("Outrigger", "Sun Microsystems, Inc.",
    "2.2.2");
ServiceItem item = new ServiceItem(null, Instance.spaceProxy, new Entry[] {
    info });
```

In both cases, the qualified field access can be substituted by an indirect anaphor of kind IA2F (e.g. `.SpaceProxy`)

```
Instance = new OutriggerImpl(config);
ServiceInfo info = new ServiceInfo("Outrigger", "Sun Microsystems, Inc.",
    "2.2.2");
ServiceItem item = new ServiceItem(null, .SpaceProxy, new Entry[] { info });
```

Table B.5 on page 92 lists all indirect anaphors of kind IA2F that occur in the experimental materials of the test condition.

2.4.6 IA2Mg: Getter declaration

The return value of a qualified getter method invocation expression like `locator.getRegistrar()` can be stored in a local variable

```
LookupLocator locator = new LookupLocator("localhost", 4160);
ServiceRegistrar registrar = locator.getRegistrar();
Log.log(Level.INFO, "Found registrar: "+registrar);
```

or be accessed directly.

```
LookupLocator locator = new LookupLocator("localhost", 4160);
Log.log(Level.INFO, "Found registrar: "+locator.getRegistrar());
```

In both cases, qualified getter method invocation expression can be replaced by an indirect anaphor of kind IA2Mg (e.g. `.ServiceRegistrar`).

```
new LookupLocator("localhost", 4160);
Log.log(Level.INFO, "Found registrar: "+.ServiceRegistrar);
```

Table B.6 on page 93 lists all indirect anaphors of kind IA2Mg that occur in the experimental materials of the test condition.

Note that a IA2Ms kind is also possible that does not invoke a getter method to obtain a value, but a setter method, to set a value, e.g. “serviceRegistration.setAttributes(new Entry[]);” could technically be replaced by “.Entry[] = new Entry[] ;” if there is a prior mention of “serviceRegistration” that can serve as related expression of “.Entry”. Because assignments to anaphors have not yet been fully implemented in the compiler and to keep further complexity out of the experiment, the IA2Ms kind of indirect and other kinds of anaphors used to assign values are not tested in the experiment.

The psychological stance towards anaphors taken in this thesis makes the kind of anaphor that an expression is said to function as, dependent on reading order. In item 1-13 on page 79, there are three ".ServiceRegistrar" anaphors in lines 16, 21 and 24. For an idealised reader, the anaphor on line 16 is said to be an indirect anaphor of kind IA2Mg that obtains the ServiceRegistrar instance via the getter "LookupLocator.getServiceRegistrar()" from the instance of "LookupLocator" created on line 15. The anaphors on lines 21 and 24 are direct anaphors accessing the "ServiceRegistrar" instances obtained during anchoring of the indirect anaphor on line 16. If reading order differs, the kind assigned to the anaphors changes: if a reader starts reading on line 21, the anaphor on that line will be indirect and the ones on lines 16 and 24 will be direct anaphors in case the reader fixates them.

2.5 Summary

Three relations between natural language and programming languages have been reviewed: that both are siblings of an abstract concept of language, that the understanding of programming languages is based on ideas about natural language and that natural language occurs in programming languages. The latter relation was found to lead to a cognitive gap between programmer and compiler who process the natural-language parts of source code differently as has been shown for the case of Java identifiers. The discussion of the cognitive gap showed that anaphors in programming languages are best anchored at edit-time, i.e. by the editor of the IDE, instead of by the compiler to avoid structural changes in the future from breaking existing indirect anaphors. Referents of indirect anahors might be found by examining all potential anchors or by restricting anchoring via activation based on eye movements. Especially in the latter case that will be implemented in the cognitive model of this thesis care needs to be taken to help programmers detect situations when the editor choses a referent for the indirect anaphor that the programmer did not intend. The chapter also summarised how knowledge is represented in Java, how anaphors in source code were choosen to look like and how they are classified. With this basis, the materials for the experiment can be constructed.

3 Construction of Experimental Materials

The following sections detail the process of constructing the materials used in the experiment and explicate the choices made during that process. Materials are presented in the order in which subjects got in contact with them.

3.1 Instructions

The instructions were read to the subjects by the experimenter until subjects were seated and facing the computer screen targeted by the eye tracker. From then on, subjects read on-screen instructions. Instructions were written in English to use a single language for both experimental materials and instructions. All instructions are listed in Appendix A.1.

3.2 Program comprehension skill questionnaire

Similar to reading skill rated by O'Reilly and McNamara (2007), I want to identify subjects' individual differences in program comprehension. Individual differences could be assessed based on programming experience, e.g. how long participants have been programming or how many programming languages they master. Because the results from programming experience questionnaires typically do not correlate well with programming skill (Feigenspan, Kästner, Liebig, Apel, & Hanenberg, 2012; Prechelt, 2000; Prechelt & Unger, 1999), a custom program comprehension skill score was created instead. The score combines a simple comparative self-assessment question from Prechelt (2000, 27) with questions related to the prior experience in comprehending design-level structures in largely undocumented source code of reasonably complex programs because the experiment poses a similar task. The complete questionnaire including instructions is listed in Appendix A.2.

The questions after the self-assessment question target participants' experience with maintenance, design patterns (Gamma, Helm, Johnson, & Vlissides, 1995), refactoring (Fowler, 1999) and the documentation of software designs. Questions target quantity and quality of the experience and the ease that participants felt about their experience. The list of design patterns was adapted from Prechelt, Unger, Tichy, Brössler, and Votta (2001). I added two more design patterns to the list: singleton and dependency injection. Singleton was added because I consider it well-known, to give participants with little knowledge of patterns a chance to give an answer. Dependency injection was added to update the list.

The program comprehension skill questionnaire is used to calculate a score that is used to balance program comprehension skill among the groups. The answers to the single-choice questions of the questionnaire are coded as integers ranging from 1 for the answer expressing the lowest skill level, to, e.g. 4, for the answer expressing the highest skill level. Multiple-choice

questions are coded by the number of options checked in the answer. During the experiment, subjects are assigned to groups with the aim that (1) in the end all groups have the same number of subjects and (2) all groups have close average program comprehension scores.

3.3 Introduction to anaphors

Participants complete the introduction to anaphors that is reproduced in Appendix A.3. The different kinds of anaphors are not named during the introduction, but are merely introduced using examples in an attempt to make the anaphors look simple. This is also meant to stress bottom-up subconscious processing of anaphors instead of top-down conscious reasoning processes when subjects learn to comprehend anaphors in code. The introduction to anaphors does not explicate that anaphors in source code are modelled after the identically named phenomenon that everybody comprehends as part of natural language. Instead, anaphors are introduced in source code by contrasting code with and without anaphors as well as examples of anaphors that work and those that a compiler would reject with an error message. I chose not to mention natural language to avoid activation of traditional resentments against the use of natural language in programming as, e.g. expressed by Dijkstra (1978). Although the experiment is designed to test indirect anaphors only, direct anaphors are also part of the introduction, test, reference and main part because at least in top-down comprehension they complicate understanding compared to a situation in which there are only indirect anaphors. More importantly, direct anaphors are the mechanism required for subsequent references to a referent that has initially be referred to using an indirect anaphor.

During the introduction and the experiment, all kinds of anaphors described in Sections 2.4.1 to 2.4.6 are introduced except for kind DA1Gr described in Section 2.4.3. Kind DA1Gr is not introduced to avoid confusion about whether or not only invocations of getter methods can be underspecified via indirect anaphors like those of kind IA2Mg, while normal methods whose return value can be referred to using kind IA2Mr cannot be underspecified. Subjects are told that all anaphors appearing in the main part have been compiled and there are no issues like referential ambiguity that is exemplified during the introduction to anaphors using a negative example to make sure subjects develop a correct understanding of the concept of direct anaphors.

3.4 Anaphors reference

When last page of the introduction to anaphors is displayed, an anaphors reference appears in the lower left of the user interface. The reference visualises a direct and two indirect anaphors so participants can turn to the reference should they be unsure about the comprehension of anaphors during the experiment. The reference is depicted in Appendix A.4.

3.5 Items

After participants have been introduced to anaphors in source code, they are prepared for the main part of the experiment in which they read source code items while their eye movements

are recorded. Appendix A.5 lists the instructions presented to participants before the items, as well as samples of the items. The instructions also tell participants when eye tracking starts. The complete set of items is available online and on the attached CD-ROM (see Appendix D).

The items of the main task were derived from source code of the Apache River project (`http://river.apache.org/`) that is sufficiently specialised not to be known by subjects and to enable an effective manipulation of activation levels. The source code of each item fit into the editor without requiring scrolling. Lines were up to 100 characters long. Items could contain a class or interface declaration, but some class declarations were spread over multiple items. For each class or interface declaration, a package declaration is provided but import statements have been removed. Almost all comments had been removed. Statements were inserted or modified in order to adapt the existing source code to the demands of the experiment. Some classes were also written from scratch. It was also possible that code from a class or interface declaration was shown in multiple items. If that was the case, at least the yes-no questions following the code (see below) were different. The instructions given to participants informed the about omissions in the code, but not about other modifications performed to create the materials.

Seven of the items for the test group contained a total of 19 indirect anaphors. 15 of the items contained a total of 81 direct anaphors. There were 4 indirect anaphors that were anchored in the return value of a method invocation (kind IA1Mr), 3 were anchored based on a field declaration (kind IA2F) and 12 were anchored based on the declaration of a getter method (kind IA2Mg).

Two sets of 20 items each were created that contained normal Java code for an infrastructure service and an application. From these two sets that were used in the control condition, two further sets were created for the test condition by replacing local variables and qualified expressions by direct and indirect anaphors. Subjects of all groups received 40 items each, 20 items in the control condition and 20 items in the test condition (see Section 1.4). While it was originally intended to create indirect anaphors such that each low-activation relation is explicated only once, three items before the item with the indirect anaphor under-specifying the relation and each high-activation relation is explicated in all three items immediately preceding the item containing the indirect anaphor, it was not possible to create such code in a reasonable amount of time. Instead, the on-line activation of underspecified relations (see Section 1.4) was estimated using the distribution of participants over groups as documented in Section 4.6.1.

Each item was followed by a yes-no question with feedback on the correctness of the response to keep subjects focussed on the task. Identical yes-no-questions are shown in both control and test group, i.e. questions targeting indirect anaphors must make sense in the control group as well, that reads local variables and qualified expressions instead. The correct answer to 22 of the yes-no questions was “yes”, 18 questions had a correct answer of “no”. The following list shows some types of the yes-no-questions.

Method declaration (MDecl) Ask whether a type declares a method whose name is provided, e.g. *Can a Lease be cancelled?* when the interface Lease declares a cancel() method.

Method inheritance (MInherit) Ask whether or not a type provides a method when the method is declared by a super type, e.g. *Does a JavaSpace has a ServiceID?* when the interface JavaSpace extends the interface Service that declares a method getServiceID().

Field declaration (FDecl) Like MDecl, but for fields.

Field type (FType) Ask about a field name which is related to the actual field type, e.g. *Does a ServiceTemplate contain an array of service types?* when the class ServiceTemplate declares a field named "serviceTypes" of type Class[].

Hyponymy (Hypo) Ask whether a type is the subtype of another one, e.g. *Does DiscoveryListener extend Service?*, when the DiscoveryListener interface does not extend the Service interface.

Detail (Det) Ask about an interaction of objects in the source code of the item, e.g. *Exporters make an object invocable to a remote host: can an OutriggerImpl be exported by a BasicJeriExporter?* when an instance of OutriggerImpl.

3.6 Comprehension questionnaire

The comprehension questionnaire contained 20 open-ended questions, 10 text-based and 10 inference-based ones. 5 text-based and 5 inference-based questions targeted or included knowledge of the relations under-specified in indirect anaphors for the test group or explicated for the control group. The questions are listed in Appendix A.6. The construction of inference-based and text-based questions roughly followed the scheme applied by O'Reilly and McNamara (2007, 130): "We classified text-based and bridging-inference questions based on whether the answer to a question could be found in a single sentence. Text-based questions could be found in a single sentence within the passage. The bridging-inference questions required the reader to integrate information from two or more sentences. A sample text-based question is 'During which phase do the chromatids become aligned at the midregion, or equator, of the cell?' An example of a bridging-inference question is 'How does cytokinesis differ for plant and animal cells?"

Off-line activation was estimated for the comprehension questions as the sum of the number of times the relations targeted by the comprehension questions are explicated in the source code of the items and the number of times the relations are the target of a yes-no question. For comprehension questions that targeted more than one relation, the counts obtained for its relations were summed. A median split was performed that resulted in a category of comprehension questions targeting low-activation relations (N=11, M=3, SD=1, min=2, max=4) and a category of comprehension questions targeting high-activation relations (N=9, M=16.78, SD=16.56, min=5, max=50). Off-line activation is used in the analysis of the answers to comprehension questions in Section 4.6.5.

3.7 Post-test questionnaire

A number of questions were used after the experiment to check possible confounding variables and to gather participants opinions and comments on anaphors in source code. These questions are reproduced in Appendix A.7.

Now that all materials are complete, the next chapter documents the experiment that put them to use.

4 Experiment

As put forward in condition (a) in Section 1.1, the goal of the experimental part of this thesis is to figure out whether indirect anaphors enhance or impede programming. The experiment also provides empirical data to base the cognitive model in Chapter 5 on. To be able to study the phenomenon in a limited time, the experiment is restricted to a program comprehension task, because program comprehension can be studied in a controlled way and is a part of software engineers' daily work. To that end, subjects are instructed to read source code to be able to summarise high-level structures of the source code. The program comprehension task combines eye movement measures and a questionnaire using the experimental design set up in Section 1.4. The effect of the activation of background knowledge on the understanding of indirect anaphors in source code is investigated and is to be compared to reading source code with local variables and qualified expressions as laid down in the hypotheses in Section 1.5.

The following sections describe the final details of the experiment. During data analysis, the post-processing of the eye tracking data turned out to be especially tricky and is therefore covered in depth before the results of the experiment are analysed and finally discussed.

4.1 Participants

There were 19 participants (3 female, 16 male, aged 24 to 46) who took part in the experiment: students of informatics, human factors as well as professional programmers. Eight participants (42 %) had normal, uncorrected vision, eight (42 %) used glasses and three (16 %) used contact lenses. The native language of one participant was Dutch, all others were native speakers of German. Participants were rewarded with lemonade, ice cream and the chance to win one out of four vouchers for the movies, each worth 25 euros.

4.2 Apparatus

The experiment was conducted in a dedicated laboratory room at basement level. Windows were shaded to create constant illumination conditions from fluorescent tubes. The experimenter was present until participants started to answer the comprehension questions. Participants were seated in front of an SMI iViewX HiSpeed tower-based eye tracker that was operated at 500Hz for binocular eye tracking using the corneal-reflection dark-pupil method. The manufacturer reports a typical accuracy of $0.25°$–$0.5°$, a typical precision of $0.01°$. The eye tracker directed participants' sight onto a 19 inch LCD display that showed 1280x1024 px on an area covering 379x304 mm at a distance of 600 mm to the eyes of the participants. The eye tracker was put into operation at least 20 minutes before each experiment to avoid drift that might occur during early operation. The stimuli were displayed in a customised version of the Eclipse IDE that used

a plug-in to control the eye tracker and correlate eye movement data with words displayed in the IDE's editor with syntax highlighting. The Eclipse IDE was customised to hide the application toolbar and disable markers and annotations in the editors (e.g. for errors shown because the editor is not able to parse indirect anaphors). The editors were put into a disabled state that removed the context menu and prevented text selection and code navigation like jumping to the declaration of a source code element at the current position of the text cursor. The height of the table carrying eye tracker, monitor, mouse and (when during the comprehension questionnaire) keyboard was adjusted so that participants would sit comfortably. A chair without casters was used.

4.3 Procedure

After being welcomed, participants declared written consent to participate in the study. They were offered to have ice tea or a caffeinated local lemonade popular among students and computer programmers. The table and eye tracker were adjusted so that their eye movements could be tracked while they sit comfortably (no calibration was performed yet). The experimenter read the instructions shown in Appendix A.1 out loud to the participant and subsequent instructions were displayed on screen (see Appendix A.1). The mouse was made accessible to the participant so that she was able to start answering the multiple-choice questions of the program comprehension skill questionnaire (see Sections 3.2 and A.2). A score was computed and displayed automatically from the results of the program comprehension skill questionnaire and the experimenter assigned the participant to one of the groups (A–D) in order to balance programming comprehension skill between the groups. Participants were introduced to anaphors (see Sections 3.3 and A.3). The eye tracker was calibrated using a 13-point calibration, a 13-point validation (using the calibration points) and a 4-point extended validation using different points distributed around the centre of the screen. Calibration could be repeated until satisfactory results were obtained.

After two warm-up tasks, participants started to work through the first of two series of 20 items (each item comprised a page of source code and a page with a yes-no question). Feedback was displayed for answers to yes-no questions. For correct answers, "Correct" was displayed for 1 second. For incorrect answers an explanation of the correct answer was provided and participants were able to read until they clicked a button to proceed. Participants were able to answer a yes-no question multiple times but only the first answer was taken into consideration. Unlike in psycholinguistics, participants were able to view the source code from previous items again at their choice. This reduces experimental control but increases ecological validity. This is a trade-off between psycholinguistics on which the experiment is based and that typically exercise greatest control over materials and between experimental work in informatics that emphasizes ecological validity as can be seen from the routine discussion of threats to internal and external validity in papers from this discipline that is not found in psycholinguistics. Because subjects were able to return to the source code after reading the yes-no question for the code, both subjects who are used to work after getting a task as well as subject how are used to gain an overview over the code before seeing the task were able work in the way they are used to. An attempt was made to standardise the behaviour with regard to this by showing the button that leads to the

question not before 5 seconds after the item was displayed.

Upon completion of the first 20 items, validation and extended validation were performed again. During a short break, participants were able to lean back, have a drink and relax. Before the second series of 20 items, calibration, validation and extended validation were performed once again in the way described above. Validation and extended validation were repeated after the second series of items. Participants were allowed to relax and stretch, before being given the keyboard to start completing 20 open-ended comprehension questions (see Sections 3.6 and A.6), writing a summary in bullet points and completing post-test questions (see Sections 3.7 and A.7). Each comprehension question had a time limit of 40 seconds that was counted down on the screen. The summary had a time limit of 5 minutes. The experimenter went to a control room next door during this last part of the experiment. After the experiment was complete, participants were thanked for participating, de-briefed and offered ice cream of a quality local brand as a thank-you. The entire procedure was scheduled to take 90 minutes per subject on average.

4.4 Materials

The materials used in the experiment were constructed as described in Chapter 3 and are listed in Appendix A.

4.5 Post-Recording Processing

Because data delivered by eye tracking devices is subject to error from a number of sources, post-recording processing is required to reduce error to a limit that permits further analysis of the data. The following sub-sections describe error in eye tracking data, survey previous algorithms for reducing error in eye tracking data and finally describe and evaluate a new algorithm for reducing error in eye tracking data that is applicable to the experiment conducted for this thesis.

4.5.1 Sources and types of error in eye tracking data

The colour-sensitive cones of the retina of the human eye required for sharp vision are concentrated in an area of the eye called the fovea that allows humans to e.g. read words within an area covering less than 2° of visual angle at the center of the visual field (Holmqvist et al., 2011, 21). Information processing is also possible within the parafovea that spans 5° visual angle of the visual field, even though processing of normal-size text takes more time in the parafovea if no saccade movement is made that re-positions the eye to bring the object of interest into the location sensed by the fovea that permits faster visual processing (Rayner, 1998, 374). While the eye may be expected to fixate the vertical center of words, the biology of the eye hinted at above will not slow down reading if the fixation is not exactly on the vertical middle of the word but the word is still processed in the fovea. In an eye-tracking system in which estimates of the on-screen position the eyes are directed at (i.e. what position they fixate) are subject to measurement errors caused by the eye-tracking system it is possible that the fixation position reported by the eye tracker diverges from the actual position of the eye. It can therefore not, due to above

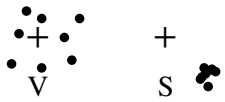


Figure 4.1: The common distinction between variable (V) and systematic (S) error or a lack of precision (V) and a lack of accuracy (S) using the terminology of Holmqvist et al. (2011). The cross shows the position a subject is actually looking at, the dots show subsequent raw data samples with coordinates determined by the eye tracking device. While both variable and systematic error co-occur, different algorithms tackle these two kinds of error. The figure is inspired by Figure 1 provided by Hornof and Halverson (2002).

flexibility of the eye, be said whether, e.g. a placement of the eye 0.5° above or below a word means that the word was not read or it was read but the position reported by the eye tracker was incorrect.

The eye tracker I used, applies a technique called *dark-pupil and corneal-reflection eye tracking*: Two infrared light sources are placed next to a camera, all pointing at the eyes of a subject. The camera is adjusted to capture images showing the (dark) pupil and the reflections of the infrared lights on the transparent cornea of the eyes. Automatic image analysis is performed to estimate the centers of the pupil and the corneal reflections to relate them to each other and thereby calculate the on-screen coordinates the eyes are found to look at. Image analysis may fail, e.g. if there are other infrared light source like the sun that lead to additional reflections in the eye. Saccades and fixations are detected from the raw coordinates.

Eye tracking equipment samples images of a participant's eyes at a fixed frequency. Each of theses images is processed, yielding a pair of on-screen coordinates representing what the eye tracker estimated the user might be looking at. This estimate is called *raw data sample*. If these raw events occur for a certain time around a single point, a *fixation* is detected that lasts for as long as the eyes are assumed to be directed at that point.

During and after the processing of image samples to fixations, error related to different dimensions can occur: temporal or spatial error. Temporal error describes the difference between the time a saccade or fixation is performed by the eye and the point in time the eye tracker reports it. Temporal error is most relevant for dynamic stimuli but is not taken in consideration in this experiment where stimuli are mostly static – participants navigate through different displays but do not scroll the display. This project will only treat spatial error, i.e. a difference between the location that the eyes fixate and the location that the eye tracker reports as being fixated. The most common distinction of spatial error is shown in Figure 4.1: variable and systematic error caused by a lack of precision or accuracy. The name *variable* error is slightly misleading, because systematic error, too, can be expected to vary over time. Variable error is most likely caused by image processing and is effectively reduced by fixation detection that computes the

fixation location from the locations of a number of raw data samples. Systematic error may caused by a number of factors that may impact how images are turned into raw data samples: changes in brightness of the environment that cause changes of pupil size, changes in head position (in any of the three dimensions). Spatial error can occur in both x and y dimension and is typically larger in the y dimension. That was also the case in this experiment. The error is at best fixed over time and display location, but may also be linearly related to the y dimension or even be related to both x and y dimensions in a linear or non-linear way. Since spatial error is a frequent problem in eye tacking, a number of automatic correction algorithms have been implemented to reduce spatial error or approximations of it.

4.5.2 Error-correction algorithm

There are a number of existing post-processing algorithms that were not applied for the following reasons: Cohen (2013) used 120px of space between lines whereas my experiment uses lines that are 17px high without any artificial spacing inbetween rendering Carl's algorithm inapplicable. Carl (2013) is specific to translation tasks in which text is written on-line in a window, but correction is off-line using an alignment of translated words to words in the source text whose meaning they represent. The experiment of this thesis does not involve a translation task and does not involve annotating all eye movements by hand. The algorithm designed by Hyrskykari (2006) is suitable for text read line after line without skimming. Since skimming was expected and was present in the experiment, this algorithm was also not applicable. The algorihm of Drewes, Masson, and Montagnini (2012) to correct in-accuracy caused by changes in pupil size was not applied because lightning conditions were kept constant and the algorithm would add another calibration step with was infeasible given the 13-point calibration and the 4-point extended calibration already used. The algorithm proposed by Cerrolaza, Villanueva, Villanueva, and Cabeza (2012) to compensate changes in the distance between eyes and screen was not required because that distance was fixed using a chin- and forehead rest. John, Weitnauer, and Koesling (2012) is suitable for few fixation locations spread over the screen, while the experiment used many words that are not necessarily fixated at their center repeatedly. Zhang and Hornof (2011) is applicable only when fixation targets are not arranged in a grid but source code, like text, is arranged in a grid with respect to the vertical dimension. Required and probable fixation locations as used in Hornof and Halverson (2002); Špakov and Gizatdinova (2014); Zhang and Hornof (2014) could not be applied across the entire screen given that the experiment used a reading task during eye tracking where it is not known which words are fixated and which ones are not.

Even though existing error correction algorithms are not applicable, the error needs to be corrected automatically for three reasons: there is too much data to correct it by hand, there is a methodological concern formulated further below, and there is an application in mind for continuing the work of this thesis that requires on-line error correction.

One further point needs to be considered: error correction algorithms cannot be based on spatial error, i.e. the distance between where the eyes fixate and where the eye tracker reports the fixation, because there is typically no report on the location of the fixation except the one provided by the eye tracker. Correction algorithms are thus mostly used to reduce approximations of spatial error that are based on knowledge about the visual system. These approximations are

based on knowledge established with equipment of greater accuracy than the equipment whose errors are to be reduced. One such approximation is the distance between the fixation location reported by the eye tracker and the center of the nearest object. This approximation is based on the fact that the eyes typically fixate the center of an object.

Which sources of error may apply to the experiment? Illumination was kept constant so that effects on pupil size have been minimised to the extent possible without additional calibration and validation. While a chin- and forehead-rest was used to reduce head movements, it could not remove these movements entirely because participants' heads were not fixed (and not supposed to be). Participants repositioning their head therefore caused disparities between fixation locations reported by the eye tracker and fixation locations assumed to be plausible when inspecting the materials overlaid with the fixation data after the experiment. In an iterative process, I decided to have the algorithm correct fixed and linear offsets along the y axis and to correct fixed offsets along the x axis by hand and to discard data with more complex forms of disparities (a displacement on the y-axis that seemed to grow like a parabola with increasing x-values, could be seen in the data a number of times).

The error correction algorithm applied to the data obtained in the experiment, is based on the fact that source code, unlike normal text, is not laid out en bloc with even edges left and right but is indented with irregular left and right edges as partly exemplified in Figure 4.2. The irregular layout permits an automatic correction along the y-axis: there are typically lines that are longer to the right than others and if there are fixations on those lines, they can be visually and algorithmically identified as belonging to those lines.

Error correction starts with computation of error values for correction parameters before parameters yielding the (globally) minimum error are selected and applied to the data.

Error computation

Throughout error computation and later analyses of the eye tracking data, a *fixation assignment window* is used. The fixation assignment window stretches $\pm$ 10 px above and below (i.e. 20 px, i.e. 5.9375 mm, i.e. 1.13°), 20 px to the left and 50 px to the right of the fixation location reported by the eye tracker (i.e. 70px, i.e. 20.727 mm, i.e. 3.96°). The dimensions of the window have been chosen to reflect research results on the *perceptual span* and *word identification span* in reading. Rayner (1998, 378) provides a somewhat fuzzy characterisation of *perceptual span* as the "effective visual field" which determines "[how] much useful information [...] a reader [can] obtain during eye fixations". For languages read from left to right, like English, the perceptual span is asymmetric and typically spans 3–4 letter spaces to the left and 14–15 letter spaces to the right of a fixation (Rayner, 1998, 380). It has also been found that the "*word identification span* (or area from which words can be identified on a given fixation" (Rayner, 1998, 380) typically extends not more than 7–8 letter spaces to the right (Rayner, 1998, 380). The perceptual span has also been found to be smaller for difficult text (Rayner, 1998, 381). The font used for the experiment is a mono-spaced one in which each character (or space) is 7 px wide. The 20 px to the left thus span 2.9 characters, the 50 px to the right span 7.1 characters. The width and placement of the fixation assignment window is thus comparable to a word identification span for text that is not easier than typical, which may be assumed of the source code used in the experiment. It should be noted however, that, when words are assigned to a fixation based on

the fixation assignment window, only entire words are assigned to the fixation, regardless of the extent of the word that is covered by the fixation assignment window.

The following steps are performed for all pairs of parameter values that are to be considered when searching for the combination of parameters that yield the lowest remaining error.

The correction algorithm uses the parameters o_x (horizontal offset in px), o_y (vertical offset in px) and a (a linear factor applied vertically). The parameter o_x is chosen by the experimenter to reduce computational complexity. The fixation locations for 111 of 756 individual tasks performed (14.7 %) were corrected using a horizontal offset determined by the experimenter. The parameters o_y and a are chosen such that they minimise an average error function $E(T, o_x, o_y, a)$ below. The parameter values that yield minimum average error are computed individually for each set of fixations T that occur while a participant performs one of the 40 main tasks of the experiment. This simple division of the stream of fixations is to ensure that each set of fixations used to compute optimum parameter values has at least one page of source code that participants should read entirely because it is new and needs to be read to complete the yes-no question of the task. Only fixations on pages with the source code of tasks and the accompanying yes-no question are used to compute the minimum error. Fixations on pages with additional instructions, calibration or validation points are ignored. Fixations not assigned to a target (see below) using any of the pairs of parameter values tested are considered outliers and are also ignored.

$$
\begin{aligned}
(o_y^*, a^*) &= \arg \min_{o_y \in O_y, a \in A} E(T, o_x, o_y, a) \\
E(T, o_x, o_y, a) &= \sum_{(f_x, f_y) \in T} \frac{1}{|T|} E(f_x, f_y, o_x, o_y, a) \\
E(f_x, f_y, o_x, o_y, a) &= |t_x(c_x(f_x, o_x)) - c_x(f_x, o_x)| + |t_y(c_y(f_y, o_y, a)) - c_y(f_y, o_y, a)| \\
c_x(f_x, o_x) &= f_x + o_x \\
c_y(f_y, o_y, a) &= a \cdot f_y + o_y
\end{aligned}
$$

The average error is computed from the error values for each fixation $(f_x, f_y) \in T$ obtained from the individual error function $E(f_x, f_y, o_x, o_y, a)$ that sums the absolute disparities along the x-axis and y-axis between the corrected fixation c and a target location t. The target location is computed in three steps described below: control assignment, word assignment and target assignment.

1. *Control assignment* The graphical user interface of the Eclipse IDE that participants operate during the experiment, is composed of user interface controls like buttons, labels, text elements and the source code editor. It is initially necessary to find the user interface control that is at the corrected position of the fixation. The user-interface control to be searched for words is chosen based on the fixation assignment window described above. If there is more than one control in the fixation assignment window, the control with the closest edge is chosen. If there are two closest controls, the one farther to the right and to the top is chosen.
2. *Word assignment* Inside the assigned control, words are identified. Based on the location of the words inside the control, the word clostest to the corrected fixation position c is

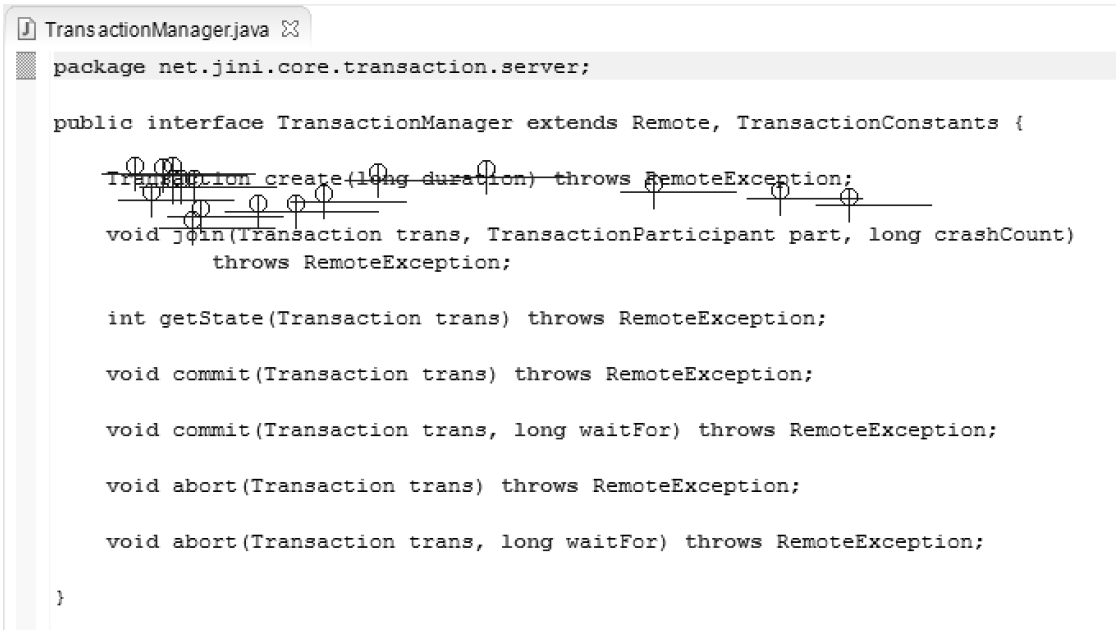


Figure 4.2: Warm-up task 0-02 of trial 19 with fixation locations reported by the eye tracker (crosses) and after correction (circles) shows a scan-path with ambiguous interpretation.

identified

3. *Target assignment* The target is a location in the line of text closest to the corrected fixation location. The line is described by the x-coordinate of the first non-whitespace character in the line (line start, s_x), the y-coordinate of the vertical center of the line (l_y) and the x-coordinate of the last non-whitespace character of the line (line end, e_x). Given a word assigned to the corrected fixation location (c_x, c_y), the target of a corrected fixation location is:

   a) the point (s_x, l_y), if $c_x < s_x$
   b) the point (c_x, l_y), if $s_x \leq c_x \leq e_x$
   c) the point (e_x, l_y), if $c_x > e_x$

   Note that the decision to compare corrected fixation locations to positions inside the line and not to words on the line removes the need to consider potential fixation locations inside words. This decision still leverages the fact that left and right edges of source code are irregular and yield irregular fixation positions when the complete source code is read. This assumes that no fixations are placed on whitespace during introspective throught-processes with low environmental intake that are to be isolated from the processing of visual input.

When the parameters with minimum error have been identified, they are applied to the fixations to reduce the error in the eye tracking data.

4.5.3 Evaluation of the algorithm and discarded data

Figure 4.2 shows a scan path that exemplifies why I do not want to correct the eye tracking data manually – either for the mere purpose of correction or to obtain a "gold standard" to compare the automatic error reduction to. The scan path in the figure was recorded after the participant had read the yes-no question for this task, that asks whether or not the `join` method (the second method defined in the depicted class) returns a Transaction. The first fixation is the one closest to the method name `join`. Then follows a fixation that is closest to the method name `create` in the line above. After two regressive fixations to the declaration of the return type `Transaction` of the `create` method, there is a sequence of fixations over the remainder of the declaration of the `create` method that is increasingly offset towards the bottom. The next fixation is on the method name `join` again, then on the `Transaction` return type of the `create` method, its name (`create`) and finally a number of times on its return type again before the participant switches to the yes-no question and answers it correctly (`join` does not a Transaction). There are a number of possible causes for the observed eye tracking data: (a) there may be fluctuating spatial error in the eye tracking data that distorts that the subject is actually just reading a single line, (b) there may be an offset towards the top of the screen that conceils that the participant is solely reading the declaration of the `join` method or (c) the participant may be contrasting the declaration of the methods `join` and `create` to ensure she submits the correct answer. While it may be theoretically possible to weight these possibilities manually, that would be too time consuming and would also depart from the idea to use eye movement data as objective measurements.

The above mentioned example illustrates why I did not perform a manual error correction based on scan-paths that potentially moves individual fixations or small groups of fixations in order to not impose my interpretation of what might have caused participants eye movements onto the data. While this still leaves the data in a state where it is not at all times clear whether fixation positions are correctly or slightly erroneously reported by the eye tracker, I might be correct to assume that the error distribution of the eye tracker is unbiased towards any of the experimental conditions and that the same applies to the error correction algorithm I applied.

Since I do not intend to craft a *gold standard* of manually corrected data that the results of the error correction algorithm could be compared to: How have the other algorithms listed above been evaluated? Three other algorithms (Carl, 2013; Cohen, 2013; Hyrskykari, 2006) were compared to a manual correction. Drewes et al. (2012) compared fixations reported by the corneal-reflection eye tracker to those simultaneously measured by a search coil system that has higher accuracy. Cerrolaza et al. (2012); John et al. (2012); Špakov and Gizatdinova (2014) instructed subjects to look at points appearing on the screen and compared error after correction to error in the absence of correction or using a simpler correction algorithm. Hornof and Halverson (2002); Zhang and Hornof (2011, 2014) compared corrected fixations to positions of mouse clicks at positions that participants had to look at in order to find out where to click (required fixation locations). Additionally, Zhang and Hornof (2011) compared how often assignments to the nearest fixation target resulted in correct assignments to known screen locations when error-correction was used vs. when it was not used.

Because no other error correction was performed except used the algorithm introduced above, the algorithm was not evaluated by comparison. Instead, the error gradients that resulted from

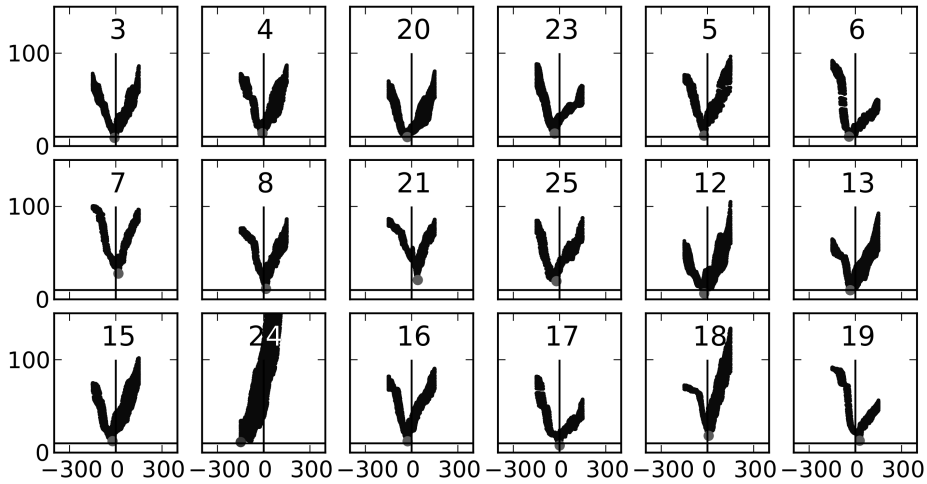


Figure 4.3: Provisional error gradients for item 1-01 for all trials (trial numbers are shown at the top of each sub-figure). The horizontal axis shows the offset along the y-axis applied (the vertical bar marks a y-offset of 0 px), the vertical axis shows the resulting average error value (the horizontal bar marks an error of 10 px). Each sub-figure shows errors for all linear factors that have been used in conjunction with the y-offset shown by the horizontal axis.

the brute force test of parameters was visually inspected to ensure that optimal parameters were found. Afterwards a visual inspection of all corrected data was performed, during which data for all task performances was discarded in which the error correction had not reduced error to a sufficient degree. (A task performance comprises all eye movement data captured for a single participant while performing one of the 42 tasks – including the two warm-up tasks – set during the experiment.) Then the corrected fixation data was compared to the uncorrected data using descriptive statistics.

The error gradient that resulted from the brute force test of parameters was plotted to confirm that the optimum of the tested parameters was chosen and that the choice appears to be sound. Errors were computed twice to make sure the process is reliable. For all corrections found minima were at a distance of at least 10% of the range of parameters tested, making it very likely that the selected ranges contained the global minima and that the search for global minima was not cut off by the selected ranges. The brute-force approach of testing all combinations of parameters in a set range was initially chosen because at the start only y-offset was chosen by the algorithm which was quite feasible computationally. At that point variants of gradient descent or more efficient search algorithms were unnecessary computationally and undesired given the limited duration of the project. Having the algorithm determine the linear factor, too, makes it plausible to use efficient search algorithms the next round.

All trials were corrected using y-offsets in intervals between $[-100; 100]$ and $[-140; 140]$ and

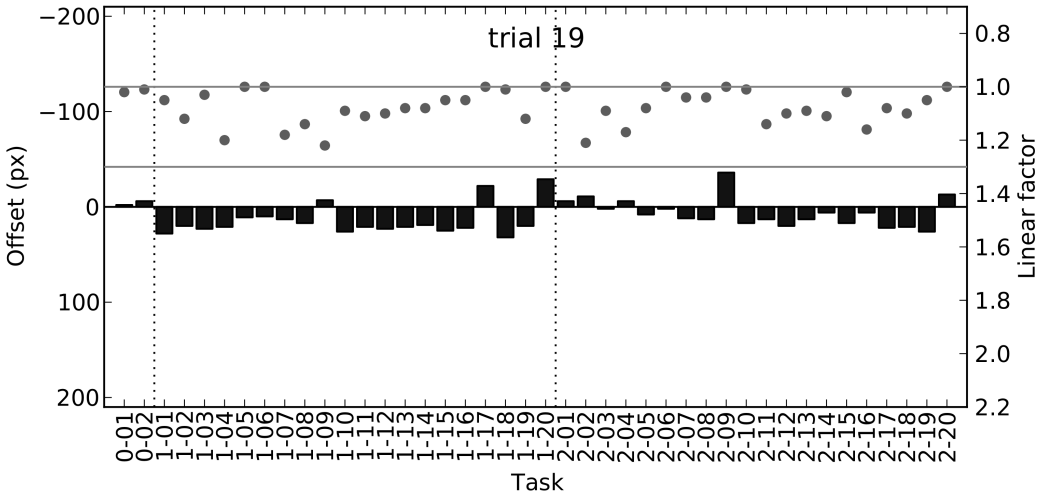


Figure 4.4: Error correction parameters for trial 19 found to lead to minimum errors for each of the tasks shown along the horizontal axis (0-XX are warm-up-tasks, 1-XX and 2-XX tasks from the main two sequences of main tasks). The black bars show the y-offset determined by the algorithm, the gray dots depict the linear offset determined by the algorithm.

linear factors in intervals between $[1, 1.01, 1.02, ..., 1.3]$ and $[1, 1.01, 1.02, ..., 1.6]$. Figure 4.3 shows error gradients for all participants while performing task 1-01 (the first task, 01, in block 1): there is variance both in the minimum error and in the optimal y-offset of the different trials (i.e. the different participants). Graphs like the one in Figure 4.4 were used to visually inspect the y-offset and linear factor chosen by the algorithm. The horizontal lines at 1 and 1.3 (on the right vertical axis) show the bounds of the parameter range tested for linear factors (note that no linear factors below 1 were tested). Figures 4.5 shows the error and its components before and after correction for all task performances of a single trial.

All corrected data was reviewed manually in groups of all fixations on a page to gauge whether there were obvious error retained by the error correction algorithm. As a result of the review, 259 (32.5 %) of all 798 ($= 19 \cdot 42$) task performances including warm-up tasks were removed due to problematic eye movement records. 116 task performances (14.5 %) were removed because sequences of non-regressive fixations that could be assigned to reading a horizontal line of words formed a rising, falling or bent curve instead of an approximately straight line. 35 task performances (4.4 %) were discarded because the participant switched from the question page to the source code and back to the question to answer it (correctly) and the fixations on the source code were close to, but not on the words that provided the information to obtain or re-assure the correct answer to the question. That is to say, in these case it was likely that fixations were assigned to the wrong line. There were 80 task performances (10.0 %) that had to be excluded from further analysis because there were too many fixations that were too far away from the words

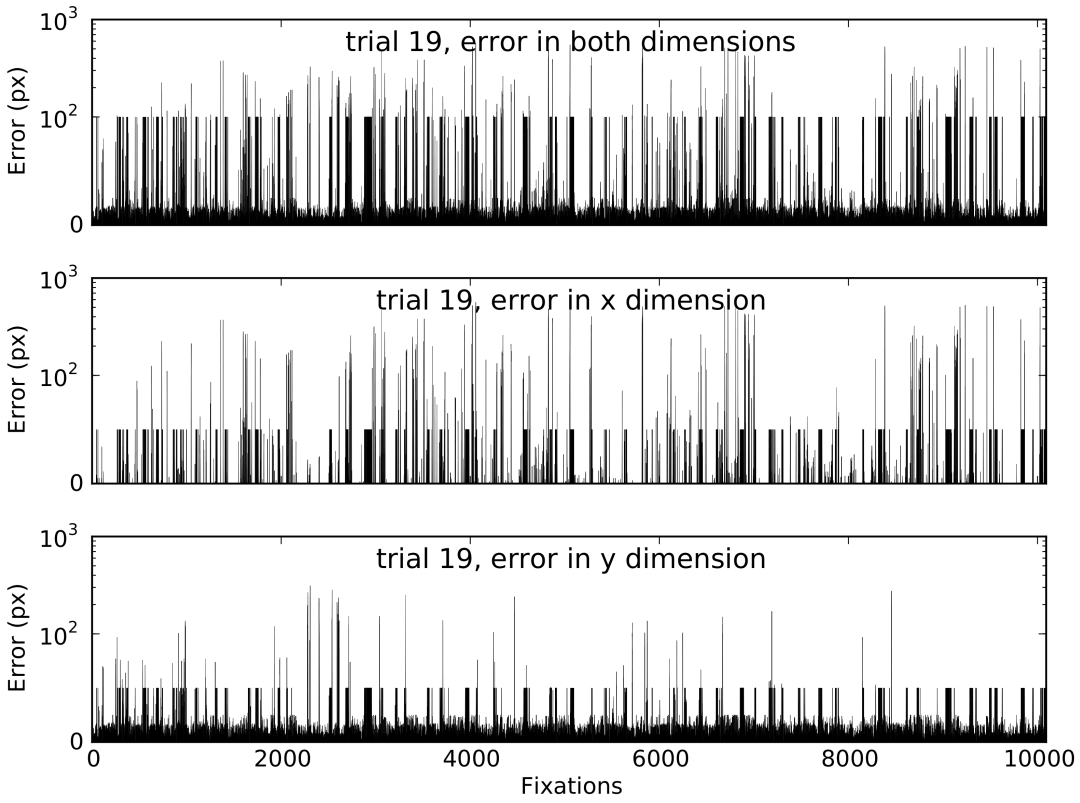


Figure 4.5: Error in both, x- and y-dimensions for trial 19 for all fixations of the right eye. The vertical axis has a split scale: error between 0 and 100 px is plotted linearly, error between 101 and 1000 px is plotted logarithmically. The gray lines show error before application of the correction algorithm, black lines show error after the correction algorithm has been applied. If the algorithm was not able to assign a word to the fixation, an error of 50 px for x- and y-dimensions and 100 px for the combined error of x- and y-dimensions is assigned to the fixation and depicted here. Average error is 27.0 px, 13.2 px and 13.9 px along both axes, x-axis and y-axis before correction and 14.5 px, 3.8 px and 10.7 px along both axes, x-axis and y-axis after correction.

of a line. 28 task performances (3.5 %) were excluded from further analysis due to missing eye tracking data (because there were very long blinks or no fixations reported between saccades). There were no participants who had all their task performances retained. Five participants had up to five task performances removed. Four participants had 6 to 10 task performances removed. Three participants had 11 to 20 task performances removed. Four participants had 21 to 30 task performances removed. One participant had 31 task performances removed, one participant had all 42 task performances removed. The distribution of removed task performances over groups and conditions are listed in Table 4.1. As can be seen, removal of data is uneven among groups

Subset	Task perf.	Removed task perf.	Removed/subset	Removed/total
Group A	160	62	39%	9%
Group B	240	54	23%	8%
Group C	160	76	48%	11%
Group D	160	61	38%	9%
Test condition	360	87	24%	12%
Control condition	360	166	46%	23%
All	720	253	35%	35%

Table 4.1: Total number of task performances, number of removed task performances, and ratio of removed task performances relative to the total number of task performances of the group (removed/subset) and to all task performances (removed/total) for groups and conditions. The numbers of task performances in this table do not include task performances of the warm-up tasks.

Processing stage	min	max	M	SD
Uncorrected	21.60	50.73	33.75	9.01
Corrected	11.83	32.51	19.50	4.77
Corrected, without unassigned fixations	9.12	18.00	12.53	2.76
Corrected, without unassigned and discarded fixations	8.90	16.98	11.77	2.37

Table 4.2: Minimum, maximum, mean and standard deviation of the per-participant mean errors (in px) computed from the fixation data. Rows represent all fixations before correction (1st row), corrected fixations (2nd row), corrected fixations remaining after those fixations have been discarded, that were not assigned to a word (3rd row), and the data from the 3rd row without fixations in discarded task performances (4th row).

and conditions, with one group (C) having half of its data discarded. The ratio of data removed overall (35%) is higher than usual in analyses of eye tracking data. Any results from the analyses of the eye tracking data are therefore not expected to be stronger than hypotheses.

Table 4.2 lists descriptive statistics for the error at different stages of correction: error correction reduced the mean error by about 1/3. Turning to fixations on words, which is the error that affects subsequent analyses, mean error is again smaller (which does not imply that error correction removed more error, though). Error for fixations on words is slightly reduced by ignoring discarded target performances. The final error does not look unacceptably high, given that in the experiment characters were 8 px wide (including a distance of 1 px to the next character) and lines were 17 px high. This does not qualify the above stated remark that the discarded data turns results from subsequent analyses on the eye tracking data into hypotheses. Instead, the results reported in Table 4.2 are qualified by what has been pointed out above: The error computed here is not the "real" distance between where the eye looks and what the eye tracker reported (spatial error), but only an approximation of spatial error. Nonetheless, there is now data ready for the

analysis.

4.6 Results

Analyses and their results are reported roughly in the order in which the respective data was acquired during the experiment.

4.6.1 Estimated on-line activation

The on-line activation of relations underspecified by indirect anaphors was estimated manually. The estimate was calculated for each indirect anaphor and its underspecified relation. This means that for two indirect anaphors that underspecify the same relation but occur at different positions in the source code, separate on-line activation values of the underspecified relation were estimated. The estimated on-line activation e was calculated as $e = a - (dr + daa)$ where a is the average relation activation, dr is the distance to the last relation activation in number of items (items succeeding each other have a distance of 1) and daa is the distance between anchor and anaphor in expressions that lie between them. The average relation activation is averaged over participants. For each participant, the relation activation is computed based on how often the relation was (a) explicitly presented, (b) required to be able to answer a yes-no-question, or (c) required to comprehend another indirect anaphor in items up to the one in which the indirect anaphor occurs for which the on-line activation of the relation is computed. In the calculation, a reflects that activation is proportional to the number of mental operations involving the relation, dr reflects that activation decreases with time and daa reflects that the lexicon entry of the anaphor will less likely spread activation to the lexicon entry of the anchor (cf. Section 1.6.2) when both are far away, i.e. have likely been read with a temporal distance.

A median split based on the estimated on-line activation produced a category of low-activation relations (N=9, M=-13.24, SD=5.38, min=-22.58, max=-6.16) and a category of high-activation relations (N=10, M=-1.9, SD=3.23, min=-6, max=2.42) that will be used for further analyses.

4.6.2 Program comprehension skill questionnaire

While all answers of the program comprehension skill questionnaire (PCSQ, see Sections 3.2 and A.2) were taken into account to compute a score used to assign participants into groups balanced by program comprehension skill, the computation of the score was re-considered after the experiment had been performed and a number of modifications to the score were considered that would further the score's target at design-level program comprehension skill and removed redudancy:

1. Answers to question 5 were not taken into account because they judge open source software instead of the participant.
2. The number of answers chosen for the multiple-choice questions 8 and 10 on known design patterns and refactorings was removed to delete a bias towards design patterns and refactorings that the inclusion of these numbers created.

	SA	PCSQ	PCSQ-13	PCSQ-123	PCSQ-1234	CQ-TB	CQ-IB
PCSQ	.82 **	–					
PCSQ-13	.85 **	.99 **	–				
PCSQ-123	.90 **	.90 **	.94 **	–			
PCSQ-1234	.87 **	.89 **	.93 **	1 **	–		
CQ-TB	.50 *	.58 *	.60 *	.57 *	.57 *	–	
CQ-IB	.43	.48 *	.48 *	.43	.42	.37	–
CQ	.57 *	.65 **	.66 **	.61 *	.61 *	.89 **	.76 **

Table 4.3: Correlations between program comprehension skill scores and comprehension question scores similar to Table 3 of O'Reilly and McNamara (2007, 132). The table lists rounded Pearson's product-moment correlation coefficients (SA: self-assessment, PCSQ: complete PCSQ score, PCSQ-13/123/1234: PCSQ score with modifications 1,3 or 1,2,3 or 1,2,3,4, CQ: comprehension score for all questions, CQ-TB/IB: comprehension score for text-based/inference-based questions, * $p < .05$, ** $p < .005$).

	Group							
	A (N=4)		B (N=6)		C (N=5)		D (N=4)	
	M	SD	M	SD	M	SD	M	SD
PCSQ	49.25	16.46	41.83	12.77	46.5	18.74	47.5	7.68
PCSQ-13	46	18	38	14	46	19	41	4

Table 4.4: Mean and standard deviation per group for the original program comprehension skill questionnaire (PCSQ) score used to distribute participants over groups and the PCSQ-13 score with modifications 1 and 3 used for further analysis.

3. The final answer on how people document non-trivial software designs was ignored if people told in the answer to the previous question that they never documented a non-trivial software design.
4. Answers to the first question were removed because the first question is a summarising self-assessment question that should rather be compared to the sum of the answers to the remaining questions that detail skills related to the comprehension of design-level structures in previously unknown source code.

To decide on a formulation of the score to be used in the median split performed before further analysis, correlations between the different formulations of the score and the scores obtained from the comprehension questions (CQ) participants answered at the end of the experiment were calculated. This way of validating the score had also been taken by O'Reilly and McNamara (2007, 132). Table 4.3 lists these correlations. The program comprehension skill questionnaire score with modifications 1 and 3 (PCSQ-13) correlates most with the score obtained from all comprehension questions (CQ). The correlation is significant, with a 95 percent confidence in-

terval of $[0.297, 0.858]$. The PCSQ-13 score was therefore chosen to perform the median split to separate participants with low program comprehension skill (N=10, M=31.2, SD=7.79, min=18, max=42) from those with high program comprehension skill (N=9, M=54.33, SD=8.03, min=43, max=66). Per-group PSCQ and PSCQ-13 scores are listed in Table 4.4.

4.6.3 Eye movements

Eye movement data was obtained from 18 subjects. (Calibration was impossible for one subject that continued the experiment without eye tracking.) For all other participants, the eye to use during data analysis was chosen to be the one for which all validation runs before and after the two parts of the main experiment yielded the best accuracy, i.e. lowest error in degrees of visual angle along the y-axis. These values are listed in the row "pre- and post-task accuracy y-axis" of Table 4.5. The validation runs used the same points as the calibration runs. Extended validation runs that used different points were not suitable for this purpose for the eye tracker only provided data averaged over both eyes for extended validation runs. Based on validation accuracy, the left eye was chosen for data analysis for 5 participants (28 % of those participants for which eye tracking data was obtained, validation accracy was min=$0.98°$, max=$4.33°$, M=$2.62°$, SD=$1.55°$ before post-recording processing), the right eye was chosen for 13 participants (72 % of the participants whose eyes were tracked, accuracy was min=$0.53°$, max=$2.68°$, M=$1.20°$ SD=$0.56°$ before post-recording processing).

Figure 4.6 shows regression-path reading-times of anaphors and the respective control target expression. Both in the test- and in the control condition there were outliers that exceeded a threshold of 4 sec. They might be caused by top-down comprehension processes like comparisons and were removed. Table 4.6 summarises the descriptive statistics for target expressions (i.e. anaphors in the test condition and local variables and qualified expressions in the control condition). The relations whose on-line activation levels structure the table are the relations underspecified by indirect anaphors in the test condition; these relations are explicated in the control condition with local variables and qualified expressions. Note that for qualified expressions that function as target expression in the control condition reading-times were computed for the qualified part only, excluding the qualifier (i.e. of `service.getServiceID()` only `getServiceID` would be used in the calculation). Presentation counts (N) for words following target expressions are lower because eight of 19 indirect anaphors (42 %) did not have a subsequent word on the same line or as part of the same statement wrapped around and continued at the next line. The differences between the test and control conditions listed in the table are visualised in Figure 4.7.

The regression-path reading-times for the target expressions were submitted to a 2×2 ANOVA with estimated on-line activation and condition as within-subject variable but no model could be constructed – potentially because not all participants had fixated indirect anaphors in all factor combinations. Alternatively, the variables were treated as between-subject variables which lead to significant main effects of estimated on-line activation (high=595 ms, low=1104 ms, F=12.155, p=.0007) and condition (control=388 ms, test=904 ms, F=9.121, p=.003) as well as a significant interaction between estimated on-line activation and condition (F=4.058, p=0.047; mean values are given in Table 4.6). The regression-path reading times for words following the target expression were submitted to an identically-designed between-subjects ANOVA but did

Subset	min	max	M	SD
Left eye chosen (N=5, 28 %)				
pre-task accuracy x- and y-axis	0.33°	2.58°	1.03°	0.92°
pre- and post-task accuracy y-axis	0.98°	4.33°	2.62°	1.55°
Right eye chosen (N=13, 72%)				
pre-task accuracy x- and y-axis	0.3°	2.8°	0.98°	0.75°
pre- and post-task accuracy y-axis	0.53°	2.68°	1.20°	0.56°

Table 4.5: Minimum, maximum, mean and standard deviation of the accuracy determined during validation runs, separated by the eye of the participant that was used in the analyses. The percentages are relative to the total number of 18 participants for which eye tracking data was acquired. Pre-task accuracy was determined after calibration before a series of items was presented. Post-task accuracy was determined immediately after the last item of a series had been completed.

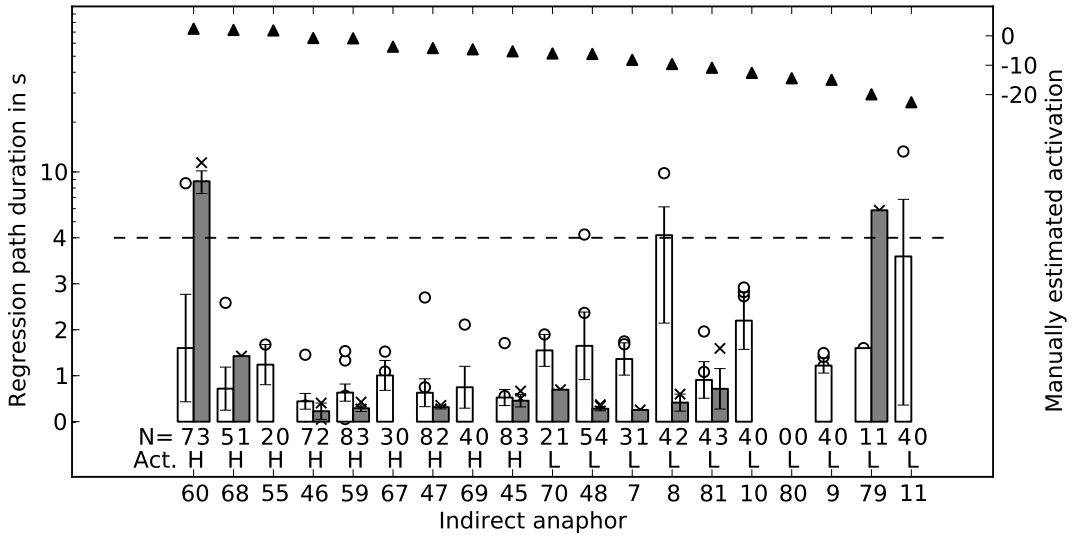


Figure 4.6: Regression path durations (in seconds) for all indirect anaphors (white bar) and their corresponding local variables and qualified expressions (gray bar), along with the estimated on-line activation of the underlying relation. Anaphors are ordered by their estimated on-line activation, the activation level (Act) and the 1-digit number of fixations (N, 1-digit per anaphor) are given at the bottom of the figure. The left vertical scale is linear until 4 sec and logarithmic above this value. Durations above 4 sec are excluded from further analysis (see text). The error bars show the standard error of the mean.

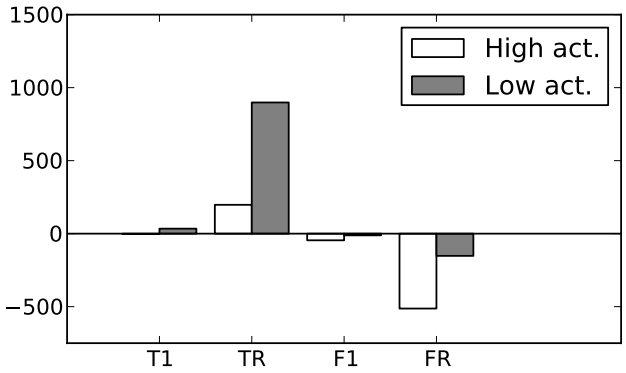


Figure 4.7: Differences between test- and control condition with regard to mean (in ms) first-fixation- (T1) and regression-path (TR) reading-times of target expressions, and mean first-fixation (F1) and regression-path (FR) reading-times of words immediately following target expressions for high estimated on-line activation (white) and low estimated on-line activation (gray). Durations above 4 sec were excluded.

not yield any significant effects or interactions. This might confirm hypothesis A from Section 1.5, that stated that regression-path reading-times for indirect anaphors that underspecify relations with high estimated on-line activation will be shorter than the reading-times of indirect anaphors for low-activation relations. This potential confirmation is constrained by the conclusion from Section 4.5.3 that the large amount of discarded data might alleviate this result and by the fact that a between-subject ANOVA was computed instead of a within-subject ANOVA. Hypothesis B might potentially be rejected: regression-path reading-times for relations with high on-line activation might differ in control and test condition subject to constraints applied to hypothesis A above.

4.6.4 Task durations

Task durations were analysed only cursorily because I am unsure how reliable the data is, given the high individual differences that can be expected for task durations. Figure 4.8(a) shows a potentially significant difference between conditions, i.e. tasks with anaphors might take longer to complete. Figure 4.8(b) differentiates further by PCSQ-13 level and shows larger differences between highly skilled participants than between less skilled participants. Figure 4.9(a) differentiates by group and condition and also adds the part of the code (1: infrastructure service, 2: application) as well as the sequence (1.: presented first during the experiment, 2.: presented afterwards during the experiment). There seems to be a strong sequence effect that I cannot yet integrate with the result shown in Figure 4.8(a). Finally 4.9(b) compares durations of selected tasks in the control condition (gray) to durations of the corresponding tasks the the test condition (white) that use direct anaphors (DA), direct anaphors and high-activation anaphors (DA+high IA), and direct anaphors and low-activation anaphors (DA+low IA). There I leave hypotheses D1 and D2 from Section 1.5 untested for now, even though Figure 4.9(b) seems to

	Relation activation									
	High					Low				
	Dur.	SD	n	N	f	Dur.	SD	n	N	f
Target expression										
First-pass reading-time										
Test	233	91.3	60	69	87%	293	182.9	39	52	75%
Control	236	90.4	20	32	63%	259	75.6	17	34	50%
Regression-path reading-time										
Test	630	639.7	51	69	74%	1456	1072.0	29	52	56%
Control	433	352.7	11	32	34%	460	390.2	11	34	32%
Word following target expression										
First-pass reading-time										
Test	228	108.3	25	45	56%	202	52.2	18	23	78%
Control	273	100.6	11	21	52%	213	71.4	7	19	37%
Regression-path reading-time										
Test	323	184.9	14	45	31%	736	781.5	12	23	52%
Control	836	453.2	6	21	29%	1679	1433.1	4	19	21%

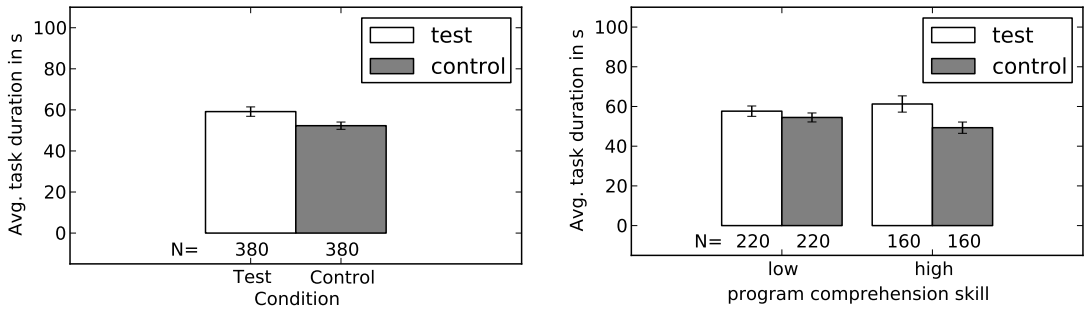
Table 4.6: Mean and standard deviation of first-fixation and regression-path reading times (in ms), for target expressions and words following target expressions by condition and by estimated on-line relation activation. The structure of the table is oriented towards Tables 3 to 6 in Garrod and Terras (2000) to facilitate comparison to their results. Three additional columns are included for presentation count (N), and fixation-/regression count (n) and frequency (f=n/N). Discarded task performances do not contribute to these columns. Note that regression-path reading durations longer than 4 sec have been excluded from the analysis.

support hypothesis D1 and Figure 4.8(a) seems to support its alternative D2.

4.6.5 Comprehension questionnaire

The comprehension questionnaire (see Sections 3.6 and A.6) comprises 20 open-ended questions. For two participants, answers to seven questions (a total of 14 answers) were lost due to a malfunction of the experimental software. While averaged results did at first sight look equivalent with/without the remaining answers of these two participants, the remaining answers for the two were removed from subsequent analysis to avoid any side-effects. The analysis was thus based on 340 answers (an answer being a potentially empty response to a comprehension question).

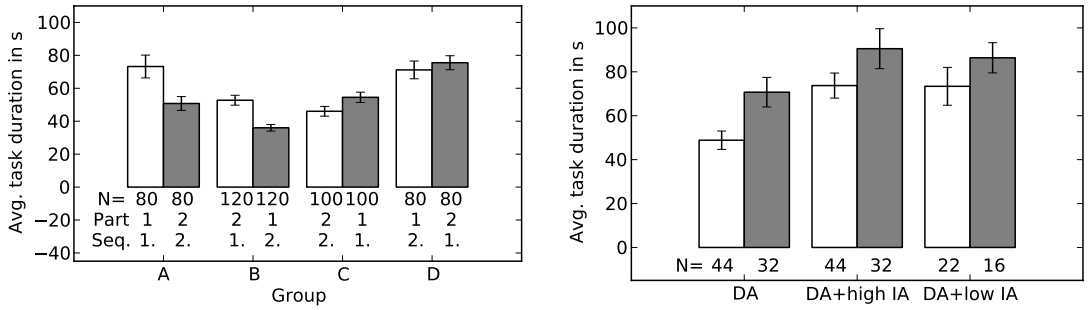
The answers to the comprehension questionnaire were scored by the experimenter according to the scheme reproduced in Section C on page 95. The scoring scheme is comparable to the one



(a) Condition

(b) PCSQ-13 and condition

Figure 4.8: Task durations (in ms) with standard error of the mean



(a) Group and condition

(b) Anaphors and condition

Figure 4.9: Task durations (ms) for test condition (white) and control condition (gray) with standard error of the mean

of O'Reilly and McNamara (2007, 130): "Each question was worth 1 point; however, partial credit was awarded for components of the correct answer. Full credit was awarded if the student provided all the necessary information; partial credit (i.e., with .25 or .5 increments depending on the question) was given if the student provided only a portion of the correct answer." Unlike O'Reilly and McNamara, the results of this experiment were scored by a single annotator.

The descriptive results for participants' answer scores are summarised in Figure 4.10 and Table 4.7. Comparable to O'Reilly and McNamara (2007), the result were submitted to a $2 \times 2 \times 2 \times 2$ ANOVA with condition, estimated off-line activation and question type as within-subject factors and PCSQ-13 level as a between-subject factor. There was a significant main effect of PCSQ-13 level (high=0.3021, low=0.1551, F=9.057, p=0.003), a significant main effect of estimated off-line activation (high=0.30515, low=0.14338, F=25.78, p=0.0001), and a significant main effect of question type (inference-based=0.15074, text-based=0.29779, F=11.37, p=0.004). There were no further significant effects. An additional $2 \times 2 \times 2$ ANOVA was performed for questions based on relations with a high estimated off-line activation. The ANOVA involved

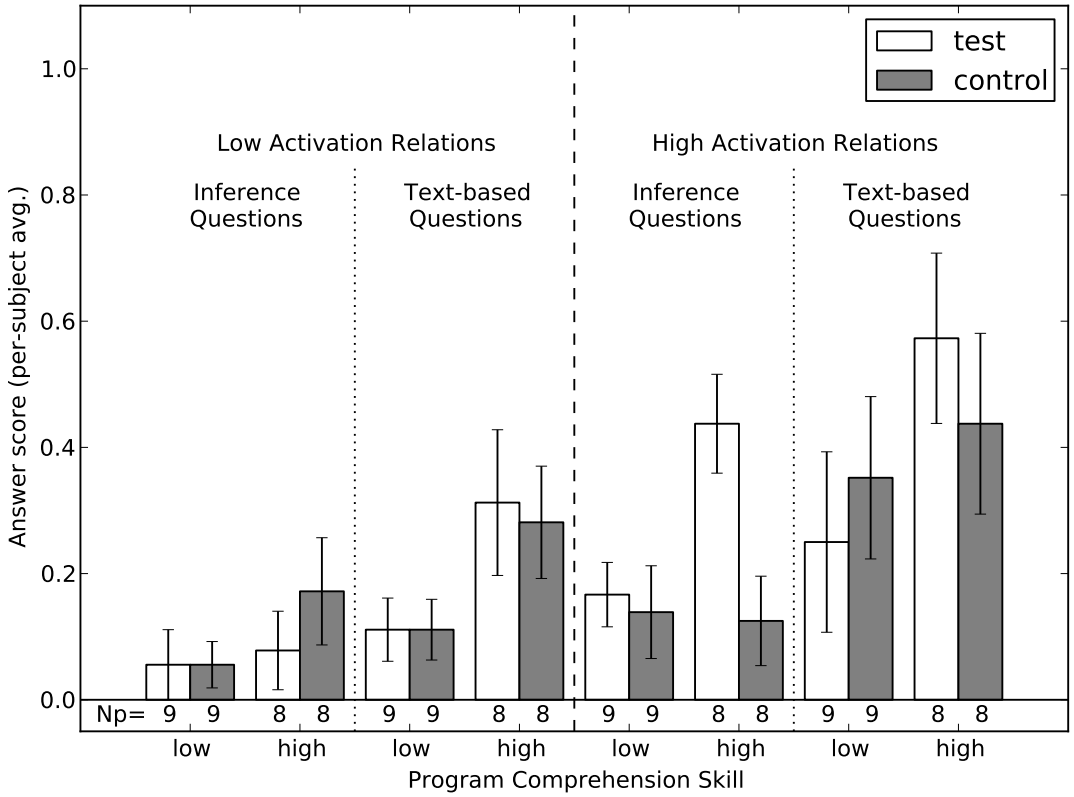


Figure 4.10: Answer scores based on per-subject means over all answers that belong to a factor combination with standard error of the mean

condition and question type as within-subject factors and PCSQ-13 level as a between-subject factor. There was a main effect of PCSQ-13 level (high=0.3932, low=0.2269, F=4.331, p=0.04), and a main effect of question type (inference-based=0.2132, text-based=0.3971, F=9.739, p=0.007). No further effects were significant. This might be caused by the many 0 answer scores. These results do no confirm hypothesis from Section 1.5: the reverse cohesion effect might not be reliable given the data. (Note that after the analyses had been computed, a typo was noticed in comprehension question 2-09, a text-based question based on a relation with a low estimated off-line activation.)

4.6.6 Post-test questionnaire

The post-test questionnaire (see Sections 3.7 and A.7) comprised seven questions to detect confounding, to see whether there were difficulties in the task and what participants thought about the use of anaphors in programming.

None of the participants knew the Apache River/Jini project used to create the main experimental materials before they took part in the experiment.

	Relation activation											
	Low						High					
	Test			Control			Test			Control		
	M	SD	N_a	M	SD	N_a	M	SD	N_a	M	SD	N_a
Low program comprehension skill (N_p=9)												
Inf.	.06	.16	18	.06	.10	18	.17	.14	30	.14	.21	24
Text	.11	.14	27	.11	.14	27	.25	.40	15	.35	.36	21
High program comprehension skill (N_p=8)												
Inf.	.08	.16	16	.17	.23	16	.44	.21	26	.13	.19	22
Text	.31	.31	24	.28	.24	24	.57	.36	14	.44	.38	18

Table 4.7: Answer scores based on per-subject means over all answers that belong to a factor combination (N_p: number of participants, N_a: number of answers)

The second question evaluated difficulties rooted in participants' English language proficiency: 11 participants (58%) faced no such difficulties, 2 participants (11%) mentioned they lacked knowledge of some English words, 2 participants (11%) reported comprehension problems at the sentence level, 3 participants (16%) found the yes-no or the comprehension questions difficult because they were in English, 1 participant (5%) found it difficult to write the answers to comprehension questions and/or the summary in bullet points in English. Note that the analysis of comprehension question scores was not qualitatively different from the reported results when answers of the latter four participants were exluded from the analysis.

When asked to evaluate the workload during the experiment (question 3), 1 participant (5%) described it as low to medium, 5 participants (26%) found it high or very high in general, a number of participants had difficulties with a certain aspect of the experiment: 5 (26%) due to recall during comprehension questions and/or summary in bullet points, 2 (11%) due to the duration of the experiment, 1 (5%) due to the amount of source code to read, 1 due to the long position in the eye tracker, 1 due to the unnatural coding situation that lacked documentation and familiar features of the Eclipse IDE, 1 due to lack of familiarity with Java, 1 gave no answer and 1 did not understand the question correctly, most likely due to a typo in the question. Question 4 asked participants to judge workload differences between the two parts of the code they had seen. Analysed by presentation sequence, 6 (32%) participants found the first part easier than the second, 10 (53%) found it more demanding and 3 (16%) felt no difference. Analysed by condition, 7 (37%) found the test condition with anaphors easier than the control condition without anaphors, 9 (47%) found it more demanding and 3 (16%) felt no difference. Analysed by the kind of source code, 7 (37%) found the infrastructure service easier to understand than the application service, 9 (47%) found it more demanding and 3 (16%) felt no difference. Apart from the sequence effect, I am reluctant to conclude a difference in workload between the two parts.

All participants reported that they understood the concept of anaphors. Presented with a binary choice, 7 (37%) of the participants responded that they think that anaphors would be useful for their programming tasks, 12 (63%) responded that anaphors would not be useful. Among the participants who found anaphors in source code useful, three did not provide further comments. The other four remarked that one needs to get used to comprehending anaphors, that anaphors are useful while writing but that underspecification might be problematic during reading, that modifications of source code that introduce referents above anaphors can make anaphors referentially ambiguous and thus not compile, and that not all variables are identical to type names and thus variables are at times preferable to anaphors. Three of those participants who did not find anaphors in source code useful did not provide comments, the remaining 9 provided comments: two participants stated that anaphors made code harder to comprehend and lead to confusion, five participants criticised that knowledge of relations underspecified in anaphors is required to comprehend the code, one participant pointed out that comprehension might be hard if there are many potential related expressions, and yet another participant found anaphors to be similar to positional arguments and expressed that named arguments/variables are easier to understand.

4.7 Summary

Extensive post-processing of the eye tracking data was performed to validate the data and to reduce the error in the data. It seems advisable to use required fixations in the experimental materials next time to have a better basis for post-hoc error correction. Even after error correction exceptionally many data had to be removed. The ANOVA for the eye tracking data could not be performed as within-subject analysis, probably due to missing data. On this basis, there was support for hypothesis A and lack of support for hypothesis B. While there were effects in line with hypothesis C, they were not significant, potentially due to many cases in which participants were unable to answer the comprehension questions because they were too difficult. Task durations seemed to support both hypothesis D1 and D2, that were formulated as alternatives. The post-test questionnaire provided input to further the development of anaphors. The experiment also provides the data for the cognitive model in the next chapter.

5 Cognitive Model

This thesis was also used to develop a cognitive model that reads the sequences of fixations obtained from the experiment and reproduces the regression-path reading times of individual participants. The model is based on jACT-R and uses a visual processing module (REMMA) implemented for this thesis, that unlike other visual modules of ACT-R and jACT-R does not generate eye movements. The basic task of the module is to pass words encoded by the REMMA module to the TWM module that constructs a text-world module. Besides the visual input, the jACT-R model uses pre-coded knowledge representations that are turned into memory chunks.

5.1 jACT-R model

The model is implemented in jACT-R, a Java-based variant of ACT-R. The model uses the two modules REMMA and TWM that are connected by two productions that are expressed in XML

Listing 8: Productions of the jACT-R model

```
<!-- get-next-word-from-remma -->
<production name="get-next-word-from-remma">
 <conditions>
  <query buffer="remma">
   <slot name="state" equals="free"/>
   <slot name="buffer" equals="empty"/>
  </query>
 </conditions>
 <actions>
  <add buffer="remma" type="get-next-word"/>
 </actions>
</production>

<!-- pass-word-from-remma-to-twm -->
<production name="pass-word-from-remma-to-twm">
 <conditions>
  <match buffer="remma" type="tlsWord"/>
  <query buffer="remma">
   <slot name="state" equals="free"/>
  </query>
  <query buffer="twm">
   <slot name="state" equals="free"/>
  </query>
 </conditions>
 <actions>
  <add buffer="twm" chunk="=remma"/>
  <add buffer="remma" type="get-next-word"/>
 </actions>
```

29 `</production>`

5.2 Visual processing

A new REMMA module is implemented that is backed by a log of fixations whereby each fixation has a start timestamp in milliseconds, a duration in milliseconds, x and y coordinates in pixels and in degrees of visual angle, and a string of 1 or multiple words that are in foveal or para-foveal sight of the reader.

When the model starts or when a saccade generated by the model ends, a new fixation is taken from the log, its contents are made available to the REMMA module and a counter for the duration of the generated fixation is started. When the non-cancellable phase of a new saccade is started, the currently generated fixation is finished, the generated duration is appended to a log file. The measured duration of the saccade is read from the log file and the saccade is modelled to take the respective amount of model time. Then the procedure starts again.

The remainder of the model works like the EMMA model of Salvucci (2001) except that the strings read by the fixations are obtained from the log file and that saccade locations/distances are not computed by the model, but are given.

Real data will be input in the following way, globally:

$$E^e = \sum_{i=0}^{n} \left|s_{i_m} - s_{i_g}\right| + \left|d_{i_m} - d_{i_g}\right| \tag{5.1}$$

E^e error in terms of eye movement durations
n number of fixations
s_{i_m} measured start of i'th fixation
d_{i_m} measured duration of i'th fixation
s_{i_g} generated start of i'th fixation
d_{i_g} generated duration of i'th fixation

Note that the error includes measured both differences in duration and differences in start time!

Note further, that error for deviating saccade durations is implicit: if a saccade is too short, this is caused by an overly long previous fixation that leads to an increased error already.

$$t_{sacc_i} = t_{sprep1} + t_{sprep2} + t_{sexec_i} \tag{5.2}$$
$$t_{exec_i} = 20ms + 2ms \cdot e_i \tag{5.3}$$

t_{sacc_i} Duration of the saccade following the i'th fixation
t_{prep1} Fixed duration of the cancellable preparation phase of 135ms
t_{prep2} Fixed duration of non-cancellable preparation of 50ms (in Salvucci 2001 this is part of what is here called t_{exec_i})

t_{exec_i} Execution duration (computed as $20 + 2 \cdot e_i$ ms by Salvucci 2001 but is read from a log file in the case of REMMA)
e_i the distance between start and end of the saccade in degrees of visual angle

Related to fixation durations generated by the model are encoding durations generated by the model. Encoding durations are directly related to visual attention. The following formula and descriptions are taken from Salvucci (2001).

$$T_{enc_{i,j}} = K \cdot [-\log f_j] \cdot e^{k\epsilon_{i,j}} \tag{5.4}$$

$T_{enc_{i,j}}$ time taken to encode the j'th object given the i'th fixation
K Used to scale the encoding time, Estimated by Salvucci to be 0.006
f_j the frequency of the j'th object, normalised to $[0, 1]$
k Used to scale the exponent, estimated by Salvucci to be 0.4
$\epsilon_{i,j}$ the distance of the center of the j'th object to the location of the i'th fixation by means of visual angle

5.3 Knowledge Representation

The ASTConversionFilter in the PagedExperiment Eclipse plugin reads the abstract syntax tree (AST) maintained by the Java editor of Eclipse and transforms it into a JSON (JavaScript Object Notation) file that includes information about conceptual relations expressed in the AST as well as the locations in the Java source code where the words are located that define the conceptual relations.

5.4 Source-Code Comprehension Processes

A number of processes are defined in the TWM jACT-R module that operate on the knowledge representations and the visual input received from the REMMA module.

5.4.1 Concept formation

Representations of conceptual relations are read from the JSON file and are turned into chunks in the declarative memory of the jACT-R model when the visual input received from REMMA indicate that the word that declared a relation has been read for the first time (subsequent reads will re-activate the concepts).

5.4.2 Referentialisation

Each schema has a scope that is an empty string for all schemata except for TWM nodes. The score of TWM nodes models the scopes of the Java source code to ensure that e.g. local variables defined in one method are not activated when another method is read from which the TWM nodes referred to by these local variables cannot be referred to as per the JLS and the

model of the Java programming language that Java programmers (and programmers of other programming lanaguges) are expected to have learnt.

Source code is a knowledge representation outside of the bodies of invocables and a text whose comprehension uses the represented knowledge inside the bodies of invocables (see Lohmeier 2011b). Thus, all simple names inside the bodies of invocables need to be encoded as reference potentials.

Because referentialisation requires a scope for each reference potential, all simple names in the source code need to be encoded as referencePotentials in the ast.json file so each of them has a scope to be used during referentialisation.

Referents can be grouped according to the following scheme, that also details the programming language constructs that have a potential for such referents to be realised during referentialisation processes.

Preconditions

Before a referentialisation process can be executed, the following conditions must hold.

1. There is a reference potential that has just been read.
2. The lexicon entry for the reference potential has been activated via the grahemic information, leading to the activation of the conceptual features of the lexicon entry and its conceptual schema.
3. The reference potential does not belong to the type name of an instanceof expression but to one of the following
   a) a simple name identical to the name of a declared variable (i.e. a declared field, parameter, or local variable)
   b) a simple name identical to an annotated anaphor
   c) the type name of a cast expression
   d) the method name of a method invocation expression
   e) Was ist mit: class instance creation expression?

5.4.3 Co-reference resolution

When a word is read, the reference potential for the word identified by line, column, word and source file name is read from the ast.json file.

```
1 {
2   ID="referencePotential$10",
3   SCOPE="...",
4   CT="ReferencePotential",
5   ACT=0.0,
6   TEC=false,
7   ATTR={ scope="...", referent=null,
        coReferenceChain="referencePotential$9", isDefinite=true/false,
        declaredIn="methodTypeSchema", roleId="methodInvocation",
        roleIn="methodInvocation$ID" },
8   SF={ uri="...", line=100, column=120, length=10 },
```

```
 9    LF={ graphemic="..." },
10    CF=[ "featuresOf:..." ]
11  }
```

1. *Assignment*
    a) *LHS has no referent yet, RHS does* The conceptual features activated by (the type and) the name of the LHS are joined with the conceptual features of the referent of the RHS and the most active features are retained. The referent of the RHS is marked in the reference potential of the LHS. -¿ Spezifikationsprozess
    b) *LHS has referent, RHS not yet*
    c) *Both LHS and RHS have referent already* A new TWM node is created that combines the conceptual features with highest activation. The new referent replaces the previous ones in all reference potential that mentioned the previous referents. -¿ Naja, das ist ja dann nur Re-Aktivierung ...
2. *Local variable declaration (with initializer)* reference potential for the type of the variable and for the name of the variable are marked with mutual coReferentTo= references
3. *Field declaration (with initializer)*

Co-reference is resolved in the following way

Preconditions

Before a co-reference resolution process can be executed, the following conditions must hold.

1. In addition to preconditions X to Y listed for instantiation.
2. Auch bei Knoten, die TWM-Knoten erstellen knnen und schon referent haben
3. Was ist, wenn kein TWM-Knoten gefunden wird: Re-Aktivierung schlgt fehl, es wird ein leerer Knoten erstellt ... nein, dann wird der Knoten erstellt, aber seine Bestandeteile, die nur in der Deklaration genannt wurden, haben weiterhin eine Aktivierung von null, ... dieser Aspekt muss aber unter Instantiation stehen ... eben fr die Ausdrcke, die nicht selbst TWM-Knoten erstellen
4.

5.4.4 Instantiation

(In the jACT-R model. The description describes the steps that follow the activation of the lexicon entry / conceptual schema of a reference potential.)

1. If a reference potential has no referent yet and no (transitive) co-referent reference potential that has a referent yet (regardless of whether the co-reference reference potential has been activated already, or not), a new TWM node for the reference potential is instantiated and saved as the referent of the reference potential and all its co-referent reference potentials.
2. If a reference potential has a referent already, the conceptual features activated by its lexicon entry are used to update the conceptual features of the referent.

A TWM node is instantiated in the following way

Preconditions

Before an instantiation process can be executed, the following conditions must hold.

1. The reference potential has no referent, yet.
2. There is no token schema that matches the reference potential, i.e. there is no token schema, that, including the activation spread by the lexicon entry of the reference potential, has an activation value above the activation threshold. The activation value of the token schema is the sum of the activation values of its conceptual features. (Note that a token schema may have been created by a prior instantiation procedure or may be part of a type schema, i.e. as a part or role of a type schema.)
   If this precondition is not met because there is a matching token schema, the re-activation procedure may be applied.
3. There is a type schema that matches the reference potential, i.e. including the activation spread by the lexicon entry of the reference potential, the activation value of the type schema exceeds the activation threshold. The activation value of the type schema is the sum of the activation values of its conceptual features.
   If more than one type schema exceeds the activation theshold, the type schema with the hightest activation value is used. If more than one type schema reaches the same maximum value, one of them is chosen.
4. If the reference potential was created from one of the following types of AST nodes: Assignment, FieldDeclaration, SingleVariableDeclaration, VariableDeclarationFragment, VariableDeclarationExpression, or VariableDeclarationStatement, the reference potential is indefinite (comparable to an indefinite noun phrase like *a house* in natural language, that will lead the reader to construct a new TWM node, when she reads the phrase for the first time).
5. Alternatively, if the reference potential was not created from one of the types of AST nodes listed under above point 4 of this enumeration, the reference potential is definite (comparable to a definite noun phrase like *the house* in natural language, that typically signals that an existing referent is to be recovered).

Implemented procedure

An instantiation process comprises the following steps.

1. All conceptual features of the type schema are copied into a new token node. The copies are deep in the sense that all tokens schemata that are contained in the type schema as conceptual features are copied and for those token schemata that are marked as technical , the copy procedure is recursively applied to their conceptual features until no further technical token schemata are found.
2. The referent is added to the reference potential as well as to all reference potentials that refer to this reference potential using the technical attribute coReferentTo.
   Note that for definite reference potentials, this step has to be performed both for the reference potential that has been read as well as for the reference potentials that belong to

an AST node of one of the types listed in precondition 4, if the latter exists (which is not always the case in the experiment because declarations for a number of types and methods used in the experiment were not introduced in the experiment).

3. If the reference potential belongs to the name of a class instance creation expression or a method invocation expression, the type schema determined according to the preconditions, contains one of the conceptual features CS#Java$isConstructor, CS#Java$AccessorDeclaration, or CS#Java$MethodDeclaration. If this is the case, arguments of the class instance creation or method invocation might have been instantiated already. The instantiated arguments are referred to by reference potentials that have a technical attribute roleIn that specifies the ID of the reference potential of the name of the class instance creation expression or the method invocation to which the argument belongs. This reference potential has a technical attribute declaredIn that contains the ID of the method type schema of the method declaration underlying the method invocation. The reference potentials of the arguments carry an additional technical attribute named roleID that specifies the ID of the method parameter in the type schema that belongs to the class instance creation or method invocation. Before assigning the instantiated arguments to the instantiated class instance creation or method invocation, the roleID attributes of the reference potentials of all (instantiated and not instantiated) attributes of the method invocation are set to the ID of the parameter in the method token schema created during instantiation (the declaredIn attribute can be used to find IDs of the parameters in the method token schema corresponding to the IDs of the parameters in the method type schema).
   All instantiated arguments are token schemata. Using the roleId that now occur in both the reference potential that refers to the instantiated argument as well as in a property describing the parameter in the token schema referred to by the reference potential of the name of the class instance creation expression or method invocation expression, the argument token replaces the value token in the property that represents the parameter in the MethodDeclaration.
4. If the reference potential has a roleID and a roleIn technical attribute, it belongs to an argument provided to a method invocation .
   a) If the reference potential whose ID is given in the roleIn technical attribute has no referent (i.e. has not yet been instantiated), no further processing happens.
   b) Alternatively, if the reference potential whose ID is given in the roleIn technical attribute has a referent, the roleID attribute in the reference potential of the argument points to the parameter in the token schema pointed to by the roleIn attribute of the reference potential of the argument. The parameter is then replaced by the argument.

5.4.5 Re-activation

A TWM node is re-activated in the following way

Preconditions

Before a re-activation process can be executed, the following conditions must hold.

1. The reference potential has a referent already, or
2. the reference potential is definite, has no referent yet and there is a token schema that matches the reference potential and has an activation (after including the activation received from the lexicon entry of the reference potential) that exceeds the retrieval threshold.
   If more than one token schema matches the reference potential and has an activation that exceeds the retrieval threshold, the one with the highest activation is chosen. If more than one type schema reaches the same maximum value, one of them is chosen.

Implemented procedure

A re-activation process comprises the following steps.

1. If the reference potential have a referent already, continue at step 5. Alternatively, there is a definite reference potential that has no referent yet and there is a token schema that matches the reference potential and has an activation that exceeds the activation threshold.
2. referent setzen
   a) ggf. auch coreference chain id bernehmen aus der er kommt,
   b) bzw. wenn dieses reference potential eine co-reference chain id hat, das token schema auch in allen anderen reference potentials des co-reference chains setzen
   c) wenn beides zutrifft: erst referent im eigenen coreference chain setzen und dann die chain id des eigenen chains durch die des chains des tokens ersetzen),
3. Identical to step 4 the implemented instantiation procedure, if the reference potential has a roleID and a roleIn technical attribute, it belongs to an argument provided to a method invocation .
   a) If the reference potential whose ID is given in the roleIn technical attribute has no referent (i.e. has not yet been instantiated), no further processing happens.
   b) Alternatively, if the reference potential whose ID is given in the roleIn technical attribute has a referent, the roleID attribute in the reference potential of the argument points to the parameter in the token schema pointed to by the roleIn attribute of the reference potential of the argument. The parameter is then replaced by the argument.
4. conceptual features aus dem Lexikon entry in den TWM-Knoten bernehmen, sofern sie noch nicht enthalten sind
5. Increment activation counters
6.

5.5 Results

The model was executed for input from trial 3. The model executed 33,704 jACT-R processing cycles and created 16,360 chunks. For part 1 of the experiment, the model generated the following regression-path reading times on indirect anaphors (the last three columns are: the measured time in ms, the generated time in ms and the difference):

```
1 IA2Mg low ServiceRegistrar     1746 3742 1996
2 IA2F  low SpaceProxy           3418 3873  455
3 IA2Mg low Lease                2728 4694 1966
4 IA1Mr low ServiceRegistration  1210 1716  506
5 IA2F  low LeaseRenewalManager   250  514  264
```

Aggregated values for all regression-path reading times in both parts of the experiment:

```
1 relationActivation empiricalDurationMs generatedDurationMs
     generated-empirical
2 high     1812 3863 2051
3 low      1957 3981 2024
4 low-high  145  118 -27
```

While the model re-generates regression-path durations, there are still errors when the text-world model is constructed and the generated regression-path durations can still be optimized (so far the standard parameters of jACT-R have not been modified).

5.6 Summary

A cognitive model was implemented in jACT-R that receives knowledge representations derived from the AST of Eclipse as well as a sequence of experimentally derived knowledge representations. The model constructs a text-world model based on these two inputs and computes activation values that determine the generated regression-path reading times. Even though there are errors during the construction of the text-world model, the model processes all input and generates regression durations, but will need further optimiziation.

A Experimental Materials

The following materials were presented to a participant during a session of the experiment.

A.1 Instructions

Welcome to the experiment! The experiment compares a new programming feature to existing programming features in Java. There are two parts: one uses the existing features, the other works with the new feature. You will be assigned to a group that has the new feature first or last in a couple of minutes. The course of the experiment is as follows: you will be asked to fill out an on-screen questionnaire on your programming skills. Then I will assign you to one of the groups to make sure that participants in all groups have equivalent skills. Afterwards, the new feature will be introduced to you, the eye tracker is calibrated to you. Then the task starts in which you are going to read a sequence of source code passages to learn the basic structures in the source code. After the main part, I will remove the eye tracker and provide you with a set of questions of another developer who read the code but did not fully understand it. I will also ask you to write a short summary of the code (in bullet points). Both the questions and the summary focus on the high-level structures in the source code. That's the experiment. Any questions so far?

A.2 Program comprehension skill questionnaire

Welcome to the experiment *Eye Movements with a new Programming Feature*!

Please use the mouse to move through the experiment: the *Next* button on the upper left leads you to the next screen. This will always be the case, except if the button is placed below the text in this part of the screen.

First, please fill out the programming skill questionnaire. Please click *Next* to begin.

Question 1 Overall, I tend to rate myself as follows compared to all other programmers

- among the upper 10 percent
- upper 11 to 25 percent
- upper 25 to 40 percent
- upper 40 to 60 percent
- lower 25 to 40 percent
- lower 11 to 25 percent
- lower 10 percent.

Question 2 I continue the work of others on software projects with at least 10,000 lines of code that lack a specification of the design of the software

- often
- sometimes
- rarely
- never so far.

Question 3 When I have to extend software with more than 10,000 lines of code, which is sparsely documented and lacks a documentation of the software design,

- I get myself a drink first, because I am used to it, but it takes time
- I am annoyed/demotivated because it is hard to understand undocumented code
- I am excited because that is what I always wanted to do

Question 4 I program with open-source APIs/frameworks

- often
- sometimes
- rarely
- never so far.

Question 5 In my experience, open-source APIs/frameworks are in general documented in a way so that I need to look at the source code

- often
- sometimes
- rarely
- never so far.

Question 6 When the documentation of the design of an open-source API was insufficient, finding the missing aspects in the source code was usually

- easy
- okay
- hard
- impossible.

Question 7 I use design patterns when writing code

- often
- sometimes
- rarely
- never so far.

Question 8 I identified or implemented the following design patterns at least once

- Abstract factory
- Abstract machine
- Adapter
- Command
- Observer
- Visitor
- Bridge
- Dependency Injection
- Decorator
- Factory method
- Flyweight

- Singleton
- Composite
- Memento
- Model-view-controller
- Pipeline
- Iterator
- Proxy
- Template method
- Layers
- Strategy
- Mediator
- Chain of responsibility.

Question 9 I use refactorings during programming

- often
- sometimes
- rarely
- never so far.

Question 10 I have applied the following refactorings to code that I or others had written

- Add Parameter
- Rename Method
- Extract or Inline Class or Interface
- Extract or Inline Method
- Move Field or Method
- Pull Up or Push Down Field or Method
- Remove Middle Man
- Replace Foreign Method
- Replace Conditional with Polymorphism
- Replace Constructor with Factory Method
- Replace Data Value with Object
- Replace Inheritance with Delegation or vice versa
- Replace Method with Method Object
- Replace Type Code with State/Strategy
- Separate Domain from Presentation.

Question 11 I find refactoring

- easy
- okay
- hard
- impossibly hard.

Question 12 I have written code for software projects with more than 10,000 lines of code

- often
- sometimes
- rarely
- never so far.

Question 13 I document the design of software projects with more than 10,000 lines of code

- often
- sometimes
- rarely
- never so far.

Question 14 I typically document the design of software projects with more than 10,000 lines of code

- in a publicly available text
- in a text available to other project members or stakeholders
- by giving a presentation at a meeting or conference
- in conversations with a single dialogue partner
- using physical or digital notes, for myself
- in my mind.

Thank your for completing the programming skill questionnaire.
Your score is <*PCSQ-SCORE*>.
The experimenter will now assign you to one of the groups in the experiment, to keep their structure balanced.

A.3 Introduction to anaphors

Below is a reproduction of the introduction to indirect anaphors given to subjects in the test group. After this sentence, *italicised* text is used for comments on and classification of the introduction materials that is not presented to the subjects and all non-italicised text is shown to subjects.

The experiment uses a new language feature called anaphors. Anaphors expression that are used inside method bodies to refer back to an expression that occurred earlier in the method body or to a parameter declared in the method header. Anaphors can also occur in constructors and initialisers. An anaphor has the following syntax.

```
.TypeName
```

The initial dot is used to indicate that the anaphor refers to an expression of type `TypeName` that was mentioned earlier. Array types like `HelloWorld[]` and generic types like `Hello<World>` work as well. E.g. in the following code snippet

Positive example of DA1C

```
private void connected(Socket socket) {
  new ConnectionMenuItem(socket.getInetAddress());
  .JMenuItem.addActionListener(new Listener(socket));
}
```

the anaphor `.JMenuItem` refers to the instance of `ConnectionMenuItem` instantiated in the line above the anaphor. The anaphor compiles, because `JMenuItem` is a supertype of `ConnectionMenuItem`. The anaphor `.ConnectionMenuItem` would have the same effect as `.JMenuItem`, though.

Just to make it absolutely clear: Above code is equivalent to the following code that stores the created instance in a local variable.

```
private void connected(Socket socket) {
  JMenuItem con = new ConnectionMenuItem(socket.getInetAddress());
  con.addActionListener(new Listener(listener));
}
```

It is also possible to keep the declaration of the local variable and use an anaphor.

Positive example of DA1C and alias analysis

```
private void connected(Socket socket) {
  JMenuItem con = new ConnectionMenuItem(socket.getInetAddress());
  .JMenuItem.addActionListener(new Listener(socket));
}
```

The compiler performs alias analysis and is able to detect that the value of the local variable `con` is identical to the `ConnectionMenuItem` instance created by the constructor invocation. The following example does not work, though.

Negative example of DA1C with referential ambiguity

```
private void connected(Socket socket) {
  new ConnectionMenuItem(socket.getInetAddress());
  new JMenuItem("Demo");
  .JMenuItem.addActionListener(new Listener(socket));
}
```

There are two instances of `ConnectionMenuItem`; therefore, the anaphor `.JMenuItem` cannot be compiled, because it is ambiguous. The anaphor `.ConnectionMenuItem` in the following snippet has a unique referent, though, and can be compiled.

Positive example of DA1C with subtype to avoid referential ambiguity

```
private void connected(Socket socket) {
  new ConnectionMenuItem(socket.getInetAddress());
  new JMenuItem("Demo");
  .ConnectionMenuItem.addActionListener(new Listener(socket));
}
```

If there are no side effects, the following snippet provides another way of avoiding referential ambiguity.

Positive example of DA1C with position to avoid referential ambiguity

```
private void connected(Socket socket) {
  new ConnectionMenuItem(socket.getInetAddress());
  .JMenuItem.addActionListener(new Listener(socket));
  new JMenuItem("Demo");
}
```

Referential ambiguity is avoided here because only expressions occurring the anaphor are considered. Expressions occurring after the anaphor cannot cause referential ambiguity.

It is not only possible to have anaphors refer to instances created by an constructor invocation. Anaphors can generally refer to values of arbitrary expressions, like the return value of a method invocation.

Positive example of IA1Mr

```
private void connected(Socket socket) {
  new File(".").list();
  System.out.println(.String[]);
}
```

E.g. `.String[]` referring to an array with element type `String` returned by `list()` declared by `File`. Anaphors can also be used to refer to values of parameters by using the type of the parameter instead of its name.

Positive example of DA1R related to parameter declaration

```
private void connected(Socket socket) {
  InputStream inStream = .Socket.getInputStream();
  int bytesAvailable = inStream.available();
}
```

Putting things one step further, it is even possible to refer "around the corner".

Positive example of IA2Mg

```
private void connected(Socket socket) {
  int bytesAvailable = .InputStream.available();
}
```

I.e. the anaphor `.InputStream` compiles because the parameter `socket` provides an instance of `Socket` that declares a getter method whose name starts with `get`, that takes no argument and returns a `InputStream` instance, i.e. the `getInputStream()` method.

Note that the anaphor `.InputStream` would not compile if the type `Socket` defined two getter method that return a `InputStream` or if there was another expression of a type that defines a getter method that returns a `InputStream`. If at run-time the variable `socket` is null, it is not possible to invoke the `getInputStream()` getter method and therefore a `NullPointerException` will be thrown when the anaphor `.InputStream` is executed. Above anaphor `.InputStream` would also compile, if `Socket` declared a field of type `InputStream` instead of a getter that returns a `InputStream`. I.e. above would also work as a replacement for the following code, if the `inputStream` field is declared public.

```
private void connected(Socket socket) {
  int bytesAvailable = socket.inputStream.available();
}
```

If the `inputStream` field is declared private instead of public, compilation of the anaphor `.InputStream` in the previous snippet will fail because the locator is not accessible.

It is also possible to refer to a referent using more than one anaphor.

Positive example of IA2F followed by DA1R

```
private void connected(Socket socket) {
  int bytesAvailable = .InputStream.available();
  .InputStream.mark(1000);
}
```

In that case the underlying getter method will be invoked only once and the `InputStream` instance will be re-used, equivalent to the following code.

```
private void connected(Socket socket) {
  InputStream inStream = socket.getInputStream();
  int bytesAvailable = inStream.available();
  inStream.mark(1000);
}
```

Control flow will affect the compilation of anaphors, too. If an anaphor follows an if-then-else statement, it can only be compiled if he type specified in the anaphor matches the type of expressions from both the if and the else branch of the if-then-else statement.

```
private void connected(Socket socket) {
  if(socket.getInputStream().available() != 0) {
    socket.getInetAddress();
  } else {
    InetAddress.getByName("localhost");
  }
  System.out.println(.InetAddress);
}
```

In contrast, an anaphor occurring after an if-then statement referring to an expression inside the if-then statement cannot be compiled, because there is no value if the body of the if-then statement is not executed, as in the following example.

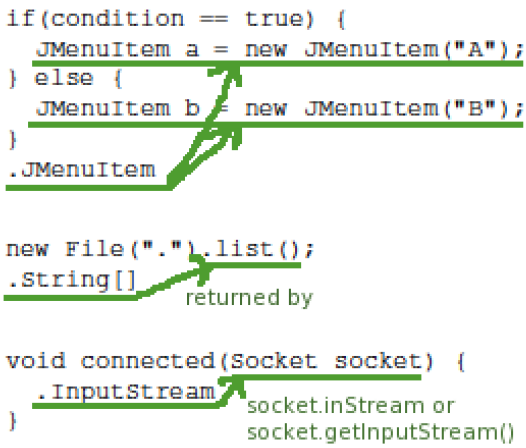
```
private void connected(Socket socket) {
  if(socket.getInputStream().available() != 0) {
    socket.getInetAddress();
  }
  System.out.println(.InetAddress);
}
```

Likewise, an anaphor following a loop cannot be compiled if it refers to an expression inside the loop because for every execution of the loop body there is a different instance, like below.

```
private void connected(Socket[] sockets) {
  for(Socket socket: sockets) {
    if(socket.getInputStream().available() != 0) {
      socket.getInetAddress();
    } else {
      InetAddress.getByName("localhost");
    }
  }
  System.out.println(.InetAddress);
}
```

Now you have a complete picture of anaphors. On the lower left, there is a short summary of how they work. It is obvious that anaphors cannot replace all uses of local variables. During the experiment, you are going to read two sequences of source code, one of which is going to contain anaphors. All anaphors in that sequence have been compiled and there are no issues like referential ambiguity.

A.4 Anaphors reference



A.5 Items

This section documents a provisional version of a subset of the items used in the experiment. When the variant of an item presented to the test group is identical to the variant presented to the test group, a note indicates this and only the variant of the control group is provided.

Indentation is not reproduced here. While the original items have an indentation of 4 spaces per tabulator character, 2 spaces are used here. The items shown to the subjects did also have the standard Java syntax highlighting of Eclipse, whereas no highlighting is used here. Note that the source code contained up to 100 characters per line which leads to additional line breaks in the source code representation below that were not in the source code seen by the subjects. These line breaks can be identified from the line numbers on the left of each source code listing: they have not been assigned a line number.

Item 01

Code for control and test group

```
1  package net.jini.core.lookup;
2
3  public interface Service {
4    public ServiceID getServiceID();
5  }
6
7  public class ServiceID implements Serializable {
8    public long mostSig;
9    public long leastSig;
10
11   public ServiceID(long mostSig, long leastSig) {
12     this.mostSig = mostSig;
```

```
13      this.leastSig = leastSig;
14    }
15
16    public long getMostSignificantBits() {
17      return mostSig;
18    }
19
20    public long getLeastSignificantBits() {
21      return leastSig;
22    }
23  }
```

Yes-no-question

Does ServiceID provide a method getMostSignificantBytes()?
No, ServiceID provides a method getMostSignificantBits().

Item 02

Code for control and test group

```
1  package net.jini.core.lookup;
2
3  public interface ServiceRegistrar extends Service {
4
5    public ServiceRegistration register(ServiceItem item, long leaseDuration)
6      throws RemoteException;
7
8    public ServiceMatches lookup(ServiceTemplate tmpl, int maxMatches)
9      throws RemoteException;
10
11   public EventRegistration notify(ServiceTemplate tmpl, RemoteEventListener
         listener,
12         long leaseDuration)
13     throws RemoteException;
14
15   public LookupLocator getLocator()
16     throws RemoteException;
17 }
```

Yes-no-question

Do implementations of ServiceRegistrar provide a method getServiceID()?
Yes, because ServiceRegistrar extends Service.

Item 03

Code for control and test group

```
1  package net.jini.core.lookup;
2
3  public class ServiceItem implements Serializable {
```

```
4
5     public ServiceID serviceID;
6     public Service service;
7     public Entry[] attributeSets;
8
9     public ServiceItem(ServiceID serviceID, Service service, Entry[] attrSets)
          {
10      this.serviceID = serviceID;
11      this.service = service;
12      this.attributeSets = attrSets;
13    }
14
15  }
```

Yes-no-question

Does ServiceItem have a field of type Entry?
No, ServiceItem has a field of type Entry[].

Item 04

Code for control and test group

```
1  package net.jini.core.entry;
2
3  public interface Entry extends Serializable {
4  }
5
6  public class ServiceInfo implements Entry {
7
8    public String name;
9    public String vendor;
10   public String version;
11
12   public ServiceInfo(String name, String vendor, String version) {
13     this.name = name;
14     this.vendor = vendor;
15     this.version = version;
16     }
17
18 }
```

Yes-no-question

Is ServiceInfo serializable?
Yes, because ServiceInfo implements Entry, which extends Serializable.

Item 05

Code for control and test group

```
1  package net.jini.core.lookup;
2
3  public class ServiceTemplate implements Serializable {
4    public ServiceID serviceID;
5    public Class[] serviceTypes;
6    public Entry[] attributeSetTemplates;
7
8    public ServiceTemplate(ServiceID serviceID, Class[] serviceTypes, Entry[]
         attrSetTemplates) {
9      this.serviceID = serviceID;
10     this.serviceTypes = serviceTypes;
11     this.attributeSetTemplates = attrSetTemplates;
12   }
13 }
14
15 public class ServiceMatches implements Serializable {
16   public ServiceItem[] items;
17   public int totalMatches;
18
19   public ServiceMatches(ServiceItem[] items, int totalMatches) {
20     this.items = items;
21     this.totalMatches = totalMatches;
22   }
23 }
```

Yes-no-question

Does ServiceTemplate declare a field called serviceType?

No, ServiceTemplate declares a field called serviceTypes.

Item 06

Code for control and test group

```
1  package net.jini.core.lookup;
2
3  public interface ServiceRegistration {
4
5    public ServiceID getServiceID();
6
7    public Lease getLease();
8
9    public void addAttributes(Entry[] attrSets) throws RemoteException;
10
11   public void setAttributes(Entry[] attrSets) throws RemoteException;
12 }
```

Yes-no-question

Are all ServiceRegistrations serializable?

No, ServiceRegistration does not extend Serializable.

Item 07

Code for control and test group

```
1  package net.jini.core.lease;
2
3  public interface Lease {
4
5    public long ANY = -1;
6    public long FOREVER = Long.MAX_VALUE;
7
8    public long getExpiration();
9
10   public void cancel() throws RemoteException;
11
12   public void renew(long duration) throws LeaseDeniedException,
         RemoteException;
13
14 }
```

Yes-no-question

Does Lease provide a method cancel()?
Yes, Lease declares a cancel() method.

Item 08

Code for control and test group

```
1  package net.jini.core.event;
2
3  public class EventRegistration implements Serializable {
4    protected long eventID;
5    protected Lease lease;
6
7    public long getID() {
8      return eventID;
9    }
10
11   public Lease getLease() {
12     return lease;
13   }
14 }
```

Yes-no-question

Does an EventRegistration have a Lease?
Yes, EventRegistration declares a field lease of Type Lease.

Item 09

Code for control and test group

```
1  package net.jini.core.event;
2
3  public class RemoteEvent implements Serializable {
4    protected long eventID;
5    protected long seqNum;
6
7    public RemoteEvent(long eventID, long seqNum) {
8      this.eventID = eventID;
9      this.seqNum = seqNum;
10   }
11
12   public long getID() {
13     return eventID;
14   }
15
16   public long getSequenceNumber() {
17     return seqNum;
18   }
19 }
20
21 public interface RemoteEventListener {
22   public void notify(RemoteEvent theEvent) throws RemoteException;
23 }
```

Yes-no-question

Does RemoteEvent declare a field eventNum?
No, RemoteEvent declares a field eventID.

Item 10

Code for control and test group

```
1  package net.jini.core.discovery;
2
3  public class LookupLocator implements Serializable {
4    protected String host;
5    protected int port;
6
7    public LookupLocator(String host, int port) {
8      if (host == null)
9        throw new NullPointerException("null host");
10     if (port <= 0 || port >= 65536)
11       throw new IllegalArgumentException("port number out of range");
12     this.host = host;
13     this.port = port;
14   }
15
16   public String getHost() {
17     return host;
18   }
19
20   public int getPort() {
```

```
21      return port;
22    }
23
24    public ServiceRegistrar getRegistrar() throws IOException,
          ClassNotFoundException {
25      int timeout = 60 * 1000;
26      return getRegistrar(timeout);
27    }
28  }
```

Yes-no-question

Does LookupLocator provide a getRegistrar() method?
Yes, LookupLocator declares a method getRegistrar().

Item 11

Code for control and test group

```
1  package net.jini.space;
2
3  public interface JavaSpace extends Service {
4
5    public Lease write(Entry entry, long lease)
6        throws RemoteException;
7
8    public Entry take(Entry tmpl, long timeout)
9        throws RemoteException;
10
11   public EventRegistration notify(Entry tmpl, RemoteEventListener listener,
         long lease)
12       throws RemoteException;
13 }
```

Yes-no-question

Does JavaSpace declare a method named listen?
No, JavaSpace declares a method named notify.

Item 12

Code for control and test group

```
1  package com.sun.jini.outrigger;
2
3  interface OutriggerServer extends JavaSpace, Remote {
4  }
5
6  public class OutriggerImpl implements OutriggerServer {
7
8    public static final String COMPONENT_NAME = "com.sun.jini.outrigger";
9    private static final Logger Log = Logger.getLogger(COMPONENT_NAME);
```

```
10    private static OutriggerImpl Instance;
11
12    private LeaseRenewalManager lrm;
13    private Exporter exporter;
14    private OutriggerServer ourRemoteRef;
15    private SpaceProxy spaceProxy;
16
17    // ...
18
19  }
```

Yes-no-question

Does OutriggerImpl declare a field of type Lease?
No, OutriggerImpl declares no field of type Lease.

Item 13

Code for control group

```
1  package com.sun.jini.outrigger;
2
3  public class OutriggerImpl implements OutriggerServer {
4
5    // ...
6
7    public static void main(String[] args) {
8      if (System.getSecurityManager() == null)
9        System.setSecurityManager(new SecurityManager());
10
11     try {
12       Configuration config = ConfigurationProvider.getInstance(args,
13         OutriggerImpl.class.getClassLoader());
14
15       LookupLocator locator = new LookupLocator("localhost", 4160);
16       ServiceRegistrar registrar = locator.getRegistrar();
17       Log.log(Level.INFO, "Found registrar: "+registrar);
18
19       Instance = new OutriggerImpl(config);
20       ServiceInfo info = new ServiceInfo("Outrigger", "Sun Microsystems,
             Inc.", "2.2.2");
21       ServiceItem item = new ServiceItem(null, Instance.spaceProxy, new
             Entry[] { info });
22       ServiceRegistration registration = registrar.register(item, Lease.ANY);
23       Log.log(Level.INFO, "Got service id: "+registration.getServiceID());
24       Instance.lrm.renewUntil(registration.getLease(), Lease.FOREVER, null);
25       Log.log(Level.INFO, "Joined registrar: "+registrar);
26     } catch(Throwable t) {
27       Log.log(Level.SEVERE, "Exception while starting Outrigger", t);
28     }
29   }
30 }
```

Code for test group

```
1  package com.sun.jini.outrigger;
2
3  public class OutriggerImpl implements OutriggerServer {
4
5    // ...
6
7    public static void main(String[] args) {
8      if (System.getSecurityManager() == null)
9        System.setSecurityManager(new SecurityManager());
10
11     try {
12       Configuration config = ConfigurationProvider.getInstance(args,
13         OutriggerImpl.class.getClassLoader());
14
15       new LookupLocator("localhost", 4160);
16       Log.log(Level.INFO, "Found registrar: "+.ServiceRegistrar);
17
18       Instance = new OutriggerImpl(config);
19       new ServiceInfo("Outrigger", "Sun Microsystems, Inc.", "2.2.2");
20       new ServiceItem(null, .SpaceProxy, new Entry[] { .ServiceInfo });
21       .ServiceRegistrar.register(.ServiceItem, Lease.ANY);
22       Log.log(Level.INFO, "Got service id:
             "+.ServiceRegistration.getServiceID());
23       .LeaseRenewalManager.renewUntil(.Lease, Lease.FOREVER, null);
24       Log.log(Level.INFO, "Joined registrar: "+.ServiceRegistrar)
25     } catch(Throwable t) {
26       Log.log(Level.SEVERE, "Exception while starting Outrigger",
             .Throwable);
27     }
28   }
29 }
```

Anaphors in code of test group

#	Kind	Line: Related expression	Line: Anaphor, Underspecified relation

Yes-no-question

Is the LeaseRenewalManager used to renew the Lease part of the ServiceRegistrar?
No, the LeaseRenewalManager is part of the OutriggerImpl instance.

Item 14

Code for control and test group

```
1  package net.jini.export;
2
3  public interface Exporter {
4    public Remote export(Remote impl) throws ExportException;
```

```
5     public void unexport();
6   }
7
8   package net.jini.id;
9
10  import java.io.Serializable;
11
12  /**
13   * A universally unique identifier
14   */
15  public class Uuid implements Serializable {
16    private final long bits0;
17    private final long bits1;
18
19    public Uuid(long bits0, long bits1) {
20      this.bits0 = bits0;
21      this.bits1 = bits1;
22    }
23
24    public final long getMostSignificantBits() {
25      return bits0;
26    }
27
28    public final long getLeastSignificantBits() {
29      return bits1;
30    }
31  }
```

Yes-no-question

Are Uuids serializable?

Yes, Uuid implements Serializable.

Item 15

Code for control and test group

```
1   package net.jini.id;
2
3   public class UuidFactory {
4
5     private static final Object lock = new Object();
6     private static SecureRandom secureRandom;
7
8     public static Uuid generate() {
9       synchronized (lock) {
10        if (secureRandom == null) {
11          secureRandom = new SecureRandom();
12        }
13      }
14      long bits0 = secureRandom.nextLong();
15      long bits1 = secureRandom.nextLong();
16
```

```
17      // set "version" field to 0x4
18      bits0 &= 0xFFFFFFFFFFFF0FFFL;
19      bits0 |= 0x0000000000004000L;
20
21      // set "variant" field to 0x2
22      bits1 &= 0x3FFFFFFFFFFFFFFFL;
23      bits1 |= 0x8000000000000000L;
24
25      return new Uuid(bits0, bits1);
26    }
27  }
```

Yes-no-question

Does UuidFactory declare a field of type SecureRandom?
Yes, UuidFactory declares a field of type SecureRandom.

Item 16

Repetition of item 10

Yes-no-question

Does OutriggerImpl declare a field of type ServiceRegistrar?
No, OutriggerImpl does not have a ServiceRegistrar.

Item 17

Code for control group

```
1  package com.sun.jini.outrigger;
2
3  public class OutriggerImpl implements OutriggerServer {
4
5    // ...
6
7    public OutriggerImpl(Configuration config) {
8      lrm = new LeaseRenewalManager();
9
10     Exporter basicExporter = new
           BasicJeriExporter(TcpServerEndpoint.getInstance(0),
11                           new BasicILFactory(), false, true);
12     exporter = (Exporter)Config.getNonNullEntry(config, COMPONENT_NAME,
           "serverExporter",
13                 Exporter.class, basicExporter);
14     ourRemoteRef = (OutriggerServer)exporter.export(this);
15
16     long maxServerQueryTimeout = Config.getLongEntry(config, COMPONENT_NAME,
17                                  "maxServerQueryTimeout", Long.MAX_VALUE);
18
19     Uuid topUuid = UuidFactory.generate();
```

```
20      Log.log(Level.INFO, "Uuid: "+topUuid);
21      spaceProxy = new SpaceProxy(ourRemoteRef, topUuid,
            maxServerQueryTimeout);
22    }
23
24    // ...
25  }
```

Code for test group

```
1   package com.sun.jini.outrigger;
2
3   public class OutriggerImpl implements OutriggerServer {
4
5     // ...
6
7     public OutriggerImpl(Configuration config) {
8       lrm = new LeaseRenewalManager();
9
10      new BasicJeriExporter(TcpServerEndpoint.getInstance(0), new
            BasicILFactory(), false, true);
11      exporter = (Exporter)Config.getNonNullEntry(config, COMPONENT_NAME,
            "serverExporter",
12                  Exporter.class, .BasicJeriExporter);
13      ourRemoteRef = (OutriggerServer)exporter.export(this);
14
15      long maxServerQueryTimeout = Config.getLongEntry(config, COMPONENT_NAME,
16                                    "maxServerQueryTimeout", Long.MAX_VALUE);
17
18      UuidFactory.generate();
19      Log.log(Level.INFO, "Uuid: "+.Uuid);
20      spaceProxy = new SpaceProxy(ourRemoteRef, .Uuid, maxServerQueryTimeout);
21    }
22
23    // ...
24  }
```

Anaphors in code of test group

#	Kind	Line: Related expression	Line: Anaphor, Underspecified relation

Yes-no-question

Is a BasicILFactory created in the OutriggerImpl(String[]) constructor?
Yes, a BasicILFactory is passed of the BasicJeriExporter.

Item 18

Code for control and test group

```
1  package com.sun.jini.outrigger;
2
3  class SpaceProxy implements JavaSpace, Serializable {
4    private OutriggerServer space;
5    Uuid spaceUuid;
6    long maxServerQueryTimeout;
7
8    SpaceProxy(OutriggerServer space, Uuid spaceUuid, long
         maxServerQueryTimeout) {
9      if (space == null)
10       throw new NullPointerException("space must be non-null");
11     if (spaceUuid == null)
12       throw new NullPointerException("spaceUuid must be non-null");
13     if (maxServerQueryTimeout <= 0)
14       throw new  IllegalArgumentException("serverMaxServerQueryTimeout must
             be positive");
15     this.space = space;
16     this.spaceUuid = spaceUuid;
17     this.maxServerQueryTimeout = maxServerQueryTimeout;
18   }
19
20   // ...
21 }
```

Yes-no-question

Does a SpaceProxy have a Uuid?

Yes, SpaceProxy declares a field spaceUuid of type Uuid.

A.6 Comprehension questionnaire

The comprehension questionnaire contains the following questions. Question type, targeted relations, the estimated off-line activation of the relations and the correct answer given in parentheses were not shown to participants.

Question 1-1 Which service is used to find other services and make a service available to others? (text-based, targeting relations `ServiceRegistrar.register(...)` and `ServiceRegistrar.lookup(...)` that yield a high activation in total, correct answer: ServiceRegistrar)

Question 1-2 Instances of which class are returned by a ServiceRegistrar when a new service has been added? (text-based, targeting the low-activation relation `ServiceRegistrar.register(...): ServiceRegistration` underspecified in indirect anaphors, correct answer: ServiceRegistration)

Question 1-3 Which service is used to store and retrieve data? (inference-based, targeting relations `JavaSpace.write(Entry, long)`, `JavaSpace.take(Entry, long)` and `Entry extends Serializable` that yield a high activation in total, correct answer: JavaSpace)

Question 1-4 What are remote events used for? (inference-based, targeting the low-activation relation `ServicRegistrar.addListener(...)`, correct answer: inform about new ServiceRegistrars)

Question 1-5 How do service objects become invocable remotely? (text-based, targeting the low-activation relation `Exporter.export(...)`, correct answer: they are exported)

Question 1-6 Instances of which class are used to find service registrars? (inference-based, targeting the low-activation relation `RegistrarLocator.getRegistrar(): ServiceRegistrar` underspecified in indirect anaphors, correct answer: RegistrarLocator)

Question 1-7 Name the two classes that are used to represented identity. (text-based, targeting `Service.ID` and `Uuid` that yield a high activation in total, correct answer: Service.ID and Uuid)

Question 1-8 Which class does OutriggerImpl use to keep track of leases? (text-based, targeting the low-activation relation `OutriggerImpl.lrm` underspecified in indirect anaphors, correct answer: LeaseRenewalManager)

Question 1-9 An instance of which class is used by OutriggerImpl to communicate to clients? (inference-based, targeting the low-activation relation `OutriggerImpl.spaceProxy` underspecified in indirect anaphors, correct answer: SpaceProxy)

Question 1-10 Which object is obtained from the ServiceRegistration to renew OutriggerImpl'[sic!] listing in the ServiceRegistrar? (text-based, targeting the low-activation relation `ServiceRegistration.getLease()` underspecified in indirect anaphors, correct answer: Lease)

Question 2-1 What was the central entity in the part of the experiment that showed an application? (inference-based, targeting the highly activated class `Voucher`, correct answer: Voucher)

Question 2-2 What were GeoCoordinates used for? (text-based, targeting `public GeoCo-`

`ordinate latitude; public GeoCoordinate longitude;` in `Location` and `public Location location;` in `Voucher` that yield a high activation in total , correct answer: to find nearby vouchers)

Question 2-3 Which service contains the vouchers? (inference-based, targeting targeting `space.write(voucher, leaseDurationMs)` and `voucher = space.take( voucherTemplate)` that yield a low activation in total, correct answer: JavaSpace)

Question 2-4 Which object informs the VoucherDistribution object about new ServiceRegistrar services? (inference-based, targeting high-activation relation `DiscoveryEvent. getRegistrars(): ServiceRegistrar[]` underspecified in indirect anaphors, correct answer: DiscoveryEvent)

Question 2-5 What is the easiest and most reliable way to distinguish two Voucher objects programmatically? (inference-based, targeting `Voucher.id:Uuid` and the comment "A universally unique identifier" in JavaDoc displayed for the the `Uuid` class that yield a high activation in total, correct answer: compare their Uuid)

Question 2-6 How does the client use its own location to find vouchers? (inference-based, targeting the high-activation relation `Voucher.location` underspecified in indirect anaphors, correct answer: it includes it in an Entry template)

Question 2-7 What object is returned when a new listener was added to a JavaSpace? (text-based, targeting the high-activation relation `JavaSpace.addListener(): EventRegistration` underspecified in indirect anaphors, correct answer: an EventRegistration)

Question 2-8 How does the client decide when to stop taking vouchers from the JavaSpace? (text-based, targeting the low-activation information `if(vouchersObtained++ >= numberOfVouchers)` in the method `VoucherConsumer.notify` (the `Client` uses a VoucherConsumer), correct answer: it maintains a counter)

Question 2-9 To which object does the lease that VoucherConsumer cancels [sic!] when it has received enough vouchers? (text-based, targeting the low-activation relation `EventRegistration.getLease()` underspecified in indirect anaphors, correct answer: the Lease belongs to the EventRegistration)

Question 2-10 Instances of which class are used in log entries to identify a specific JavaSpace service? (inference-based, targeting the high-activation relation `Service. getServiceID()` underspecified in indirect anaphors, correct answer: instances of ServiceID)

A.7 Post-test questionnaire

Question 1 Did you know Apache River/Jini before the experiment?

- Yes
- No.

Question 2 Which parts of the task did you find difficult because the experiment was in English?

Question 3 How to [sic!] do evaluate the workload during the experiment.

Question 4 What do you think about the workload of the two parts of the code?

- Both were equally demanding.

- The first was more demanding than the last.
- The first was easier than the last.

Question 5 Do you think you understood the concept of anaphors?

- Yes
- No.

Question 6 Do you think that anaphors would be useful for your programming tasks?

- Yes
- No.

Question 7 Do you have a comment on anaphors in source code?

B Anaphors in the Experimental Materials

Please see next page.

#	Item	Line: Related expression	Line: Anaphor	Replaced target expression
1	1-13	13: Configuration.getInstance(...)	18: .Configuration	config
2	1-13	16: .ServiceRegistrar	21: .ServiceRegistrar	registrar
3	1-13	21: .ServiceRegistrar	24: .ServiceRegistrar	registrar
6	1-13	25: Throwable t	26: .Throwable	t
13	1-14	19: .Uuid	20: .Uuid	topUuid
14	1-14	18: Uuid.generate()	19: .Uuid	topUuid
15	1-15	201: EntryRep tmpl	207: .EntryRep	tmpl
16	1-15	207: EntryHandle handle	208: .EntryHandle	handle
17	1-15	208: .EntryHandle	209: .EntryHandle	handle
18	1-15	201: QueryCookie cookie	211: .QueryCookie	cookie
19	1-15	211: .QueryCookie	214: .QueryCookie	cookie
20	1-16	269: UUid.generate()	273: .Uuid	uuid
21	1-16	266: RemoteEventListener listener	273: .RemoteEventListener	listener
23	1-16	275: .EventRegistrationWatcher	277: EventRegistrationWatcher	reg
24	1-16	266: EntryRep tmpl	277: .EntryRep	tmpl
25	1-16	273: .Uuid	280: .Uuid	uuid
26	1-16	277: .EventRegistrationWatcher	280: .EventRegistrationWatcher	reg
27	1-17	333: EntryRep rep	335: .EntryRep	rep
28	1-17	335: .EntryRep	336: .EntryRep	rep
29	1-17	336: .EntryRep	337: .EntryRep	rep
30	1-17	337: .EntryRep	340: .EntryRep	rep
31	1-17	337: new EntryHandle(...)	342: .EntryHandle	handle
32	1-17	336: .Result	344: .Result	r
33	1-17	340: .EntryRep	344: .EntryRep	rep
34	1-17	344: .EntryRep	345: .EntryRep	rep
35	1-19	7: Entry tmpl	17: .Entry	tmpl
36	1-20	10: Entry tmpl	12: .Entry	tmpl
37	1-20	10: RemoteEventListener listener	13: .RemoteEventListener	listener
38	1-20	15: Entry entry	16: .Entry	entry
39	1-20	16: .Entry	19: .Entry	entry
41	2-09	70/73: discoveryManager	76: .DiscoveryManager	discoveryManager
42	2-13	11: .RegistrarLocator	11: .RegistrarLocator	registrar.getLocator()
43	2-13	8: ServiceRegistrar registrar	13: .ServiceRegistrar	registrar
44	2-13	20: RemoteException re	21: .RemoteException	re
49	2-14	13: (Location[])...	14: .Location[]	locations
50	2-14	14: Location location	16: .Location	location
52	2-14	19: Throwable t	20: .Throwable	t
53	2-14	7: JavaSpace space	17: .JavaSpace	space
54	2-14	15: Uuid.generate()	16: .Uuid	id
56	2-15	7: Voucher voucher	10: .Voucher	voucher
57	2-15	12: .Service.ID	14: .Service.ID	space.getServiceID()
58	2-15	13: RemoteException re	14: .RemoteException	e
62	2-17	8/11: discoveryManager	15: .DiscoveryManager	discoveryManager
63	2-17	83: .RegistrarLocator	83: .RegistrarLocator	registrar.getLocator()
65	2-17	79: ServiceRegistrar registrar	86: .ServiceRegistrar	registrar
66	2-17	90: RemoteException re	91: .RemoteException	re
72	2-18	7: JavaSpace space	11: .JavaSpace	space
74	2-18	11: .JavaSpace	12: .JavaSpace	space
75	2-18	11: .Voucher	12: .Voucher	voucherTemplate
77	2-18	12: .VoucherConsumer	13: .VoucherConsumer	voucherConsumer
78	2-18	10: .Location	14: .Location	location

Table B.1: Anaphors of kind DA1R in the experimental materials

#	Item	Line: Related expression	Line: Anaphor	Replaced target expression
4	1-13	19: new ServiceInfo(...)	20: .ServiceInfo	info
5	1-13	20: new ServiceItem(...)	21: .ServiceItem	item
12	1-14	10: new BasicJeriExporter(...)	12: .BasicJeriExporter	basicExporter
22	1-16	273: new EventRegistrationWatcher(...)	275: .EventRegistrationWatcher	reg
40	2-08	13: new Server(...)	14: .Server	serverImpl
51	2-14	16: new Voucher(...)	17: .Voucher	voucher
61	2-16	12: new Client(...)	13: .Client	c
64	2-17	85: new ServiceTemplate(...)	86: .ServiceTemplate	template
73	2-18	10: new Voucher(...)	11: .Voucher	voucherTemplate
76	2-18	11: new VoucherConsumer(...)	12: .VoucherConsumer	voucherConsumer

Table B.2: Anaphors of kind DA1C in the experimental materials

#	Item	Line: Related expression	Line: Anaphor	Replaced target expression
71	2-18	9: getMyLocation()	10: .Location	location

Table B.3: Anaphors of kind DA1Gr in the experimental materials

#	Item	Line: Related expression	Line: Anaphor, Underspecified relation	Replaced target expression
9	1-13	21: .ServiceRegistrar.register(...)	22: .ServiceRegistration, register(...):ServiceRegistration	registration
48	2-13	13: .ServiceRegistrar.lookup(...)	14: .ServiceItem[], lookup(...):ServiceItem[]	items
70	2-17	86: .ServiceRegistrar.lookup(...)	87: .ServiceItem[], lookup():ServiceItem[]	items
79	2-18	12: .JavaSpace.addListener(...)	13: .EventRegistration, addListener(...):EventRegistration	eventRegistration

Table B.4: Anaphors of kind IA1Mr in the experimental materials

#	Item	Line: Related expression	Line: Anaphor, Underspecified relation	Replaced target expression
8	1-13	18: new OutriggerImpl(...)	20: .SpaceProxy, OutriggerImpl.spaceProxy	Instance.spaceProxy
11	1-13	18: new OutriggerImpl(...)	23: .LeaseRenewalManager, OutriggerImpl.lrm	Instance.lrm
59	2-15	10: .Voucher	11: .Location, Voucher.location	voucher.location

Table B.5: Anaphors of kind IA2F in the experimental materials

#	Item	Line: Related expression	Line: Anaphor, Underspecified relation	Replaced target expression
7	1-13	15: new RegistrarLocator(...)	16: .ServiceRegistrar, RegistrarLocator.getRegistrar()	registrar
10	1-13	22: .ServiceRegistration	23: .Lease, ServiceRegistration.getLease()	registration.getLease()
45	2-13	7: DiscoveryEvent e	8: .ServiceRegistrar[], DiscoveryEvent.getRegistrars()	e.getRegistrars()
46	2-13	8: ServiceRegistrar registrar	10: .Service.ID, Service.getServiceID()	registrar.getServiceID()
47	2-13	8: ServiceRegistrar registrar	11: .RegistrarLocator, ServiceRegistrar.getLocator()	registrar.getLocator()
55	2-14	17: .JavaSpace	20: .Service.ID, Service.getServiceID()	space.getServiceID()
60	2-15	7: JavaSpace space	12: .Service.ID, Service.getServiceID()	space.getServiceID()
67	2-17	78: DiscoveryEvent e	79: .ServiceRegistrar[], DiscoveryEvent.getRegistrars()	e.getRegistrars()
68	2-17	79: ServiceRegistrar registrar	82: .Service.ID, Service.getServiceID()	registrar.getServiceID()
69	2-17	79: ServiceRegistrar registrar	83: .RegistrarLocator, ServiceRegistrar.getLocator()	registrar.getLocator()
80	2-18	12: .JavaSpace	14: .Service.ID, Service.getServiceID()	space.getServiceID()
81	2-20	15: this.eventRegistration	17: .Lease, EventRegistration.getLease()	this.eventRegistration.getLease()

Table B.6: Anaphors of kind IA2Mg in the experimental materials

C Comprehension Question Scoring Scheme

1 point is assigned, if:

- LIT The answer was literally identical to the correct answer. (Differing or lack of capitalization, typos and unfinished words are tolerated for answers were due within 40 seconds.)
- COR The answer correctly paraphrases the correct multi-word answer.
- OPN The question was posed in a way in which the answer is correct.

0.75 point is assigned, if:

- NHD The answer diverges from the correct answer by omitting the modifier of a compound noun given in the question and common in the source code or in Java.
- CHD The answer diverges from the correct answer by using a noun common in the source code or Java instead of the head of the compound in the correct answer.
- GER The head noun of the compound used in the answer is a noun made up by a German native speaker that can be imaged to be supposed to resemble the head noun in the correct answer.

0.5 point is assigned if:

- SRP A semantically related part of the object identified by the correct answer is named in the answer. The named object exists in the source code.
- ONE The answer correctly states one object in case of a question that asks for two objects.
- DET The first part of the answer correctly paraphrases the correct multi-word answer, but the second part of the answer adds an incorrect detail.

0.25 point is assigned, if:

- CNA An object is named in the answer that does not exist in the presented source code but is conceptually associated to the object named in the correct answer. The parts of the given name are no uncommon words occuring in the question.
- CRL An object is named in the answer that exists in the source code and is directly conceptually associated to the object named in the correct answer.
- PLC The named compound contains a placeholder besides common words or words from the question that indicated that the participant was looking for the missing word to replace the placeholder.

0 point is assigned, if:

- NOA No answer was provided.

DON	The participant expressed that she does not know an answer.
REP	The provided answer rephrases the question as a declarative statement.
CMB	The provided answer is a combination of uncommon words occuring in the question and potentially further common words from the source code presented. (Words like *service* and *manager* are common words in the source code presented.)
INC	The provided answer is incomprehensible and possibly incomplete.
WRG	The provided answer is wrong for other reasons.
MIS	The data for the answer is missing.

D Index of Materials Online

The following materials are provided on the website accompanying this thesis (`http://www.monochromata.de/master_thesis/`).

Thesis document

1. `ma.pdf`

Software

The software created or extended for this thesis is available in the repository `https://srv7.git-repos.de/dev785/ecem.git` (user: public, password: public). The repository contains the following Eclipse projects relevant to this thesis.

1. `ActivationGraph` A jACT-R module implemented for this thesis to export information about chunks and their activation to be fed into the dmGraph.r script that plots networks of chunks and plots the time course of activation of chunks.
2. `EyeTrackerAPI` A modified version of the Java Eye Tracker API that the EyeTrackingPlugin is based on.
3. `EyeTrackingPlugin` The EyeTrackingPlugin that controls eye trackers and receives event-based data from it that is distributed to filters.
4. `MAModel` The jACT-R model that uses the REMMA and TWM modules to process input created using eye tracking.
5. `PagedExperiment` An Eclipse plugin implemented for the thesis that runs the experiment and exports data for statistical analyses as well as for the jACT-R model.
6. `REMMA` The REMMA module for jACT-R that was written for this thesis and re-generates durations for given eye tracking input.
7. `TWM` The TWM module for jACT-R that was written for this thesis and implements three-level semantics.

Notes

The notes contain acknowledgements and provide traces to classes that make up the software used to generate and analyse the experimental data. Appendix D details locations from which to obtain the classes and scripts named here.

Chapter 1

1 *"Please note that this text uses page notes."*: This is a page note.

3 *"FACE and NOSE"*: I follow the convention to typeset words that refer to concepts all caps.

6 *"A double-balanced design was chosen"*: Both Lutz Prechelt and Marc Halbrügge proposed to use this design instead of a simpler-to-implement one that would not have balanced sequence effects.

6 *"Comprehension question score"*: Marc Halbrügge proposed not to call this score an *error*, because it involved human judgement instead of mere counting.

7 *"show indirect anaphors only when they are beneficial"*: Being beneficial might not necessarily mean quick to read, but could also mean to have longer-lasting memory effects

Chapter 2

19 *"If there is another relation available to the compiler that differs from the relation the programmer has in mind"*: Sebastian Möller pointed out the importance of such errors. I did not yet consider them in depth so I am not sure how often they will occur. They need to be treated, though.

22 *"these anaphors are marked with a dot preceding a type name, e.g. ".Service.ID"."*: Sascha Tamm proposed to use anaphors without the dot prefix. This lead me to try out this approach.

Chapter 3

27 *"The score combines a simple comparative self-assessment question"*: Lutz Prechelt proposed to combine the self-assessment question with custom questions targeted at the specific skill required in the experiment.

Chapter 4

32 *"Unlike in psycholinguistics, participants were able to view the source code from previous items again at their choice."*: It was Lutz Prechelt who initially suggested that participants be enabled to review at least a limited number of earlier items.

33 *"Post-Recording Processing"*: The section title is taken from Holmqvist, Nyström, and Mulvey (2012, 51).

37 *"The following steps are performed for all pairs of parameter values that are to be considered when searching for the combination of parameters that yield the lowest remaining error."*: Error computation and choice of parameters with least error is implemented in the `de.monochromata.eclipse.eyetracking.filter.correction.DisparityComputatio` class. In the filter chain, filters of this class need to be configured after a `de.monochromata.eclipse.eyetracking.filter.ControlAssignmentFilter` and a `de.monochromata.eclipse.eyetracking.filter.WordAssignmentFilter`.

37 *"a linear factor applied vertically"*: Sascha Tamm suggested to add such a linear factor to the correction algorithm.

37 *"Control assignment"*: Control assignment is implemented in the `de.monochromata.eclipse.eyetracking.filter.ControlAssignmentFilter` class.

37 *"Word assignment"*: Word Assignment is implemented in the `de.monochromata.eclipse.eyetracking.filter.WordAssignmentFilter` class.

37 *"Inside the assigned control"*: The current implementation of the `WordAssignmentFilter` class is able to assign words inside StyledText, Text, Label and Button controls to fixations.

38 *"the word clostest to the corrected fixation position c is identified"*: While the default configuration of the `WordAssignmentFilter` class (inherited from the `AbstractAssignmentFilter` class) is to use a `RectangleMatcher`, like the `ControlAssignmentFilter` class, all filter configuration files (`batchEventsChain.json`, `disparityComputationChain.json` and `disparityCorrectionChain.json` replace this setting by a `PointMatcher`, that matches the nearest word (based on the bounds of the word). The `PointMatcher` is used instead of the `RectangleMatcher` because the former is able to assign a closest word even in case the closest word is outside the rectangle within which the latter is able to assign closest word.

38 *"Target assignment"*: Target assignment is also implemented in the `WordAssignment` class.

38 *"When the parameters with minimum error have been identified, they are applied to the fixations to reduce the error in the eye tracking data."*: Application of the found parameters is implemented in the filter class `de.monochromata.eclipse.eyetracking.filter.correction.DisparityCorrection`.

40 *"(trial numbers are shown at the top of each sub-figure)"*: The trial numbers are irregular because trial numbers were not reset after two pre-tests that used different materials, because no eye tracking data could be obtained for trial 9 which required restarts of the experimental software that assigned two further trial numbers, because trial 14 did not proceed beyond the introduction to anaphors and had to be aborted, and because trial number 22 was assigned during a test of the equipment and trial numbers were not reset thereafter.

40 *"Errors were computed twice to make sure the process is reliable."*: Early attempts to recompute a previously-obtained minimum error led to the discovery that error computation misses events if the Eclipse window is put into the background while error computation is in progress. Hence, there was no user interaction with the computer while error computation was in progress during later stages.

41 *"All corrected data was reviewed manually"*: An initial analysis of the data not obtained via eye tracking resulted in task durations that neither fit, nor contradicted hypotheses D1 and D2, Lutz Prechelt encouraged me to look for potential sources of error or other explanations of the results. He later kept advising caution when I am analysing and interpreting my data.

44 *"a number of modifications to the score were considered"*: See the pcsqScore.ods spreadsheet.

45 *"Correlations between program comprehension skill scores and comprehension question scores"*: The correlations and statistical tests were computed using R, as have been all statistical tests reported in the following. The script pcsqCorrelations.r was used.

Chapter 5

56 *"A new REMMA"*: The REMMA module is a modified implementation of the EMMA module from Salvucci (2001) that abbreviated *Eye Movements and Movement of Attention* the additional *R* stands for *Reconstructing* to express that .

60 *"If more than one type schema exceeds the activation theshold, the type schema with the hightest activation value is used."*: This constitutes an implicit specification process

60 *"following types of AST nodes"*: The listed names a class names from the package `org.eclipse.jdt.core.dom` that defines the document object model (DOM) used to represent the abstract syntax tree (AST) of Java source code files.

60 *"All conceptual features of the type schema are copied into a new token node."*: Note that in the case of method type schemata this step also copies all roles i.e. the return role and the parameter roles of the method. Therefore, instantiation makes the return role available for recovery through indirect anaphors (subject to the activation of the return role exceeding the retrieval threshold).

61 *"the argument token replaces the value token in the property that represents the parameter in the MethodDeclaration."*:

61 *"The parameter is then replaced by the argument."*:

62 *"The parameter is then replaced by the argument."*:

References

Carl, M. (2013). Dynamic programming for re-mapping noisy fixations in translation tasks. *Journal of Eye Movement Research*, *6*(2), 1–11.

Cerrolaza, J. J., Villanueva, A., Villanueva, M., & Cabeza, R. (2012). Error characterization and compensation in eye tracking systems. In *Proceedings of the symposium on eye tracking research and applications* (pp. 205–208). New York, NY, USA: ACM.

Cohen, A. L. (2013). Software for the automatic correction of recorded eye fixation locations in reading experiments. *Behavior Research Methods*, *45*(3), 679–683.

Dijkstra, E. (1978). On the foolishness of 'natural language programming'. *Unpublished Report*.

Drewes, J., Masson, G. S., & Montagnini, A. (2012). Shifts in reported gaze position due to changes in pupil size: ground truth and compensation. In *Proceedings of the symposium on eye tracking research and applications* (pp. 209–212). New York, NY, USA: ACM.

Feigenspan, J., Kästner, C., Liebig, J., Apel, S., & Hanenberg, S. (2012). Measuring programming experience. In *ICPC 2012* (pp. 73–82).

Fowler, M. (1999). *Refactoring: Improving the design of existing code*. Boston: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Boston: Addison-Wesley.

Garrod, S., & Sanford, A. J. (1982). Bridging inferences and the extended domain of reference. In A. Baddeley & J. Long (Eds.), *Attention and performance* (Vol. XI, pp. 331–346). Hillsdale, NJ: Erlbaum.

Garrod, S., & Terras, M. (2000). The contribution of lexical and situational knowledge to resolving discourse roles: Bonding and resolution. *Journal of Memory and Language*, *42*, 526–544.

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2005). *The Java language specification* (3rd ed.). Boston: Addison Wesley.

Haviland, S. E., & Clark, H. H. (1974). What's new? acquiring new information as a process in comprehension. *Journal of Verbal Learning and Verbal Behavior*, *13*(5), 512–521.

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., & van de Weijer, J. (Eds.). (2011). *Eye tracking: A comprehensive guide to methods and measures*. Oxford University Press.

Holmqvist, K., Nyström, M., & Mulvey, F. (2012). Eye tracker data quality: what it is and how to measure it. In *Proceedings of the symposium on eye tracking research and applications* (pp. 45–52). New York, NY, USA: ACM.

Hornof, A. J., & Halverson, T. (2002). Cleaning up systematic error in eye-tracking data by using required fixation locations. *Behaviour Research Methods, Instruments, and Computers*, *34*(4), 592–604.

Hyrskykari, A. (2006). Utilizing eye movements: Overcoming inaccuracy while tracking the focus of attention during reading. *Computers in Human Behavior*, *22*(4), 657–671.

John, S., Weitnauer, E., & Koesling, H. (2012). Entropy-based correction of eye tracking data for static scenes. In *Proceedings of the symposium on eye tracking research and applications* (pp. 297–300). New York, NY, USA: ACM.

Keefe, D., & McDaniel, M. (1993). The time course and durability of predictive inferences. *Journal of Memory and Language*, *32*(4), 446–463.

Kintsch, W. (1974). *The representation of meaning in memory*. Hillsdale: Erlbaum.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction integration model. *Psychological Review*, *92*(5).

Kintsch, W., & van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological Review*, *85*, 363–394.

Knöll, R., & Mezini, M. (2006). Pegasus: first steps toward a naturalistic programming language. In *Companion to the 21st ACM SIGPLAN symposium on object-oriented programming, systems, languages & applications (OOPSLA '06)* (pp. 542–559). New York: ACM.

Landauer, T. K. (Ed.). (2007). *Handbook of latent semantic analysis*. Mahwah, New Jersey: Erlbaum.

Lavigne-Tomps, F., & Dubois, D. (1999). Context effects and associative anaphora in reading. *Journal of Pragmatics*, *31*, 399–415.

Lohmeier, S. (2011a). *Continuing to shape statically-resolved indirect anaphora for naturalistic programming.* Retrieved 2014/06/01, from `http://monochromata.de/shapingIA/`

Lohmeier, S. (2011b). *Continuing to shape statically-resolved indirect anaphora for naturalistic programming: A transfer from cognitive linguistics to the Java programming language.*

Lohmeier, S. (2013). *Modellierung von Anaphernverstehen und Drei-Ebenen-Semantik in ACT-R.* Hausarbeit im Seminar "Modellierung und Simulation in Mensch-Maschine-Systemen" bei Prof. Dr. Nele Russwinkel, TU Berlin, WS2012/13.

Lopes, C. V., Dourish, P., Lorenz, D. H., & Lieberherr, K. (2003). Beyond AOP: toward naturalistic programming. *SIGPLAN Notices*, *38*(12), 34–43.

Mandel, T. S. (1979). Eye movement research on the propositional structure of short texts. *Behaviour Research Methods & Instrumentation*, *11*(2), 180–187.

Mandel, T. S. (1984). *An eye movement investigation of a process model of comprehension* (Tech. Rep. No. 84-132). University of Colorado, Institute of Cognitive Science.

Miller, J. R., & Kintsch, W. (1980). Readability and recall of short prose passages: A theoretical analysis. *Journal of Experimental Psychology: Human Learning and Memory*, *6*(335–354).

O'Reilly, T., & McNamara, D. S. (2007). Reversing the reverse cohesion effect: Good texts can be better for strategic, high-knowledge readers. *Discourse Processes*, *43*(2), 121–152.

Prechelt, L. (2000). *An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl for a search/string-processing program* (Technical Report No. 2000-5). Universität Karlsruhe, Fakultät für Informatik.

Prechelt, L., & Unger, B. (1999). *A controlled experiment on the effects of psp training: De-*

tailed description and evaluation (Technical Report No. 1999-01). Universität Karlsruhe, Fakultät für Informatik.

Prechelt, L., Unger, B., Tichy, W. F., Brössler, P., & Votta, L. G. (2001). A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE Transactions on Software Engineering*, *27*(12), 1134–1144.

Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, *124*(3), 372–422.

Salvucci, D. D. (2001). An integrated model of eye movements and visual encoding. *Cognitive Systems Research*, *1*(4), 201–220.

Schank, R. C. (1999). *Dynamic memory revisited*. Cambridge, UK: Cambridge University Press.

Schmalhofer, F., McDaniel, M. A., & Keefe, D. (2002). A unified model for predictive and bridging inferences. *Discourse Processes*, *33*(2), 105–132.

Schwarz, M. (1992). *Kognitive Semantiktheorie und neuropsychologische Realität: Repräsentationale und prozedurale Aspekte der semantischen Kompetenz*. Tübingen: Niemeyer.

Schwarz, M. (2000). *Indirekte Anaphern in Texten: Studien zur domänengebundenen Referenz und Kohäsion im Deutschen*. Tübingen: Niemeyer.

Schwarz-Friesel, M. (2007). Indirect anaphora in text: A cognitive account. In M. Schwarz-Friesel, M. Consten, & M. Knees (Eds.), *Anaphors in text : cognitive, formal and applied approaches to anaphoric reference* (pp. 3–20). Amsterdam: Benjamins.

Singer, M. (1979). Processes of inference during sentence encoding. *Memory & Cognition*, *7*(3), 192–200.

Tzeng, Y., van den Broek, P., Kendeou, P., & Lee, C. (2005). The computational implementation of the landscape model: Modeling inferential processes and memory representations of text comprehension. *Behavior Research Methods*, *37*(2), 277–286.

Špakov, O., & Gizatdinova, Y. (2014). Real-time hidden gaze point correction. In *Proceedings of the symposium on eye tracking research and applications* (pp. 291–294). New York: ACM.

Yeari, M., & van den Broek, P. (2013). *Computational modeling of inference generation during reading comprehension.* Poster at the 35th Annual Cognitive Science Conference.

Zhang, Y., & Hornof, A. J. (2011). Mode-of-disparities error correction of eye-tracking data. *Behavior Research Methods*, *43*, 834-842.

Zhang, Y., & Hornof, A. J. (2014). Easy post-hoc spatial recalibration of eye tracking data. In *Proceedings of the symposium on eye tracking research and applications* (pp. 95–98). New York: ACM.

Changelog

Version	Description
1.1	Proofread
1.0	Initial version