Arbeitsgruppe Software Engineering

Ermitteln des Aufmerksamkeitsfokus beim Durchsuchen von Code

Thorsten Wenzlaff
wenzlaff@inf.fu-berlin.de
Matrikel-Nr.: 3421289

Betreuer:
Prof. Dr. Lutz Prechelt
Christopher Oezbek

29. Juli 2006

Ich versichere, dass ich die vorliegende Arbeit mit dem Titel „Ermitteln des Aufmerksamkeitsfokus beim Durchsuchen von Code“ selbständig verfasst habe. Alle Stellen der Arbeit, die anderen Werken wörtlich oder sinngemäß entnommen sind, sind unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Berlin, 29. Juli 2006

Thorsten Wenzlaff

Zusammenfassung

Beim Durchsuchen von Quelltexten widmet der Entwickler bestimmten Teilen mehr Aufmerksamkeit als anderen. Aus der systematischen Aufzeichnung dieser Aufmerksamkeitsfoki des Entwicklers können sich Vorteile im praktischen Umgang mit Quelltexten ergeben. Dementsprechend wird hier nach einer begrifflichen und inhaltlichen Definition des Aufmerksamkeitsfokus eine Implementation – der „FocusTracker" – erarbeitet, die während der Betrachtung eines Quelltextes durch einen Entwickler den anzunehmenden Aufmerksamkeitsfokus ermittelt. Dabei werden die vom FocusTracker ermittelten Daten an ein bestehendes System zur Aufzeichnung von Entwickleraktivitäten exportiert. Während der Entwickler seinen eigenen Fokus jederzeit selber nachvollziehen kann, kann der FocusTracker dies jedoch nur mit einer gewissen Unsicherheit. Diese Unsicherheit im Sinne einer Abweichung von der Realität wird mit Hilfe einer empirischen Studie in dieser Arbeit untersucht und bewertet, um den Nutzen und die Einsetzbarkeit des FocusTracker zu überprüfen und erste Justierungen der Konfiguration vorzunehmen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Problemstellung und Ziel: FocusTracker als ECG-Sensor

Quelltexte sind heutzutage eine essentielle Basis von Softwareprojekten jeglicher Art. Die systematische Beobachtung, wonach ein Programmierer im eigenen oder fremden Quelltext zu einem bestimmten Zeitpunkt auf eine bestimmte Art und Weise sucht oder diesen bearbeitet, ist bereits aufgrund dieser großen Bedeutung des Quelltextes von starkem Interesse. Die Kenntnis, wie sich die Aufmerksamkeit des Entwicklers im Laufe seiner Arbeit an einem Quelltext verschiebt und worauf er sich fokussiert, soll daher im Laufe dieser Arbeit diskutiert und die Ermittlung des Aufmerksamkeitsfokus implementiert werden. Die Diskussion dieses Phänomens gewinnt vor dem Hintergrund weiter an Gewicht, dass sich zwar erste Arbeiten mit quelltextbezogenen Rahmenthemen befassen, jedoch bislang nicht mit dem hier betrachteten Problem auseinander setzen.[1]

Der Mikroprozess[2], der die Aktivität des Entwicklers beschreibt, steht demzufolge im Mittelpunkt dieser Arbeit. Er stellt die Grundlage für die Sammlung von Daten über den Aufmerksamkeitsfokus dar. Eine weiteres bedeutendes Themenfeld dieser Arbeit beschreibt, wie die durch den Mikroprozess generierten Ereignisse gesammelt, ausgewertet und an ein bestehendes System, das ECG[3], übermittelt werden. Dieses ECG (ElectroCodeoGram) zeichnet die Arbeitsschritte eines Programmierers in der Entwicklungsumgebung mit Hilfe von Sensoren automatisch auf. Die vorliegende Arbeit ergänzt die durch ECG-Sensoren beobachteten Aktivitäten um weitere wichtige, feingranularere Daten. Hierfür wird im Zuge dieser Arbeit der „FocusTracker“ im Sinne eines zusätzlichen ECG-Sensors implementiert. Die Daten sollen in voraggregierter Form an das ECG übergeben werden.[4] Der FocusTracker kommt damit der Forderung nach besseren Daten hinsichtlich spezifischer Programmieraktivitäten nach.[5] Um die Validität der Daten des FocusTracker empirisch zu stützen, erfolgt eine qualitative bzw. quantitative Datenerhebung über visuelle Auswertungen des Fokus in einer realen Programmierumgebung und einen Vergleich mit den Ergebnissen des FocusTracker.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in vier Teilbereiche. Der erste Teil „Einleitung“ führt in die Thematik ein und behandelt die zugrunde liegende Problemstellung. Große Bedeutung hat die Definition des Aufmerksamkeitsfokus, da er die begriffliche Basis der Arbeit darstellt und den Aufmerksamkeitsfokus von verwandten Begriffen wie dem

[1] vgl. [RM05], [KM05]
[2] vgl. Kapitel 1.3.2 Mikroprozess - ECG
[3] vgl. [Sch05], siehe auch Kapitel Kapitel 1.3.2 Mikroprozess - ECG dieser Arbeit
[4] bzgl. der Aggregation vgl. Kapitel 2.5.2 Abschnitt ECGExport
[5] vgl. http://projects.mi.fu-berlin.de/w/bin/view/SE/ThesisFuzzyFocusTracker

technischen Fokus abgrenzt. Die Definition erfolgt vor dem Hintergrund der Frage, wie mit Quelltexten gearbeitet wird bzw. welche Bedeutung dieser Arbeit im Softwareentwicklungsprozess zukommt. Der zweite Teil „Technische Umsetzung der Problemstellung“ beschreibt die Projektanforderungen und die Architektur der Implementierung des „FocusTracker“. Hierfür spielen insbesondere die Bewertungsregeln, die darüber entscheiden, zu welchem Grad sich ein Element im Aufmerksamkeitsfokus des Entwicklers befindet, eine elementare Rolle. Hier werden Unterschiede zu den in der Einleitung beschriebenen verwandten Lösungen zur Messung des Aufmerksamkeitsfokus deutlich. In dem darauf folgenden dritten Teil „Evaluation“ werden empirische Fallstudien ausgewertet, um die Validität der vom hier entwickelten „FocusTracker“ ermittelten Daten zu überprüfen. Die Ergebnisse liefern Hinweise darauf, in welcher Form die Konfiguration des FocusTracker modifiziert werden muss. Es wird deutlich, dass die einzelnen Teile dieser Arbeit nur bedingt in einem chronologischen Verhältnis zueinander stehen, sondern vielmehr einer permanenten Rückkopplung unterliegen.

1.3 Hintergrund

1.3.1 Aufmerksamkeitsfokus

Aufmerksamkeit im Allgemeinen bezeichnet nach Brockhaus die „selektive Ausrichtung des Wahrnehmens, Vorstellens und Denkens auf bestimmte gegenwärtige oder erwartete Erlebnisinhalte bei gesteigerter Wachheit und Aufnahmebereitschaft“. Unter Fokus versteht er gemeinhin den „Mittelpunkt im Bereich des Interesses“.

Die bloße Addition beider Begriffsdefinitionen des zusammengesetzten Begriffs des Aufmerksamkeitsfokus kann hier jedoch nur der groben Annäherung im Sinne dieser Arbeit dienen. Obgleich der Begriff des Aufmerksamkeitsfokus in anderen soziokulturellen Zusammenhängen Verwendung findet, kann hier dennoch nur von geringen Schnittmengen gesprochen werden. Aufmerksamkeit als Filter oder Selektor bzw. Konzentration wird zwar als eine Form der Fokussierung des Geistes auf einen Gegenstand unter Nichtberücksichtigung alles anderen verstanden - und bereits Leibnitz stellte fest, dass wir „auf diejenigen Gegenstände Aufmerksamkeit [richten], welche wir von den übrigen unterscheiden und ihnen vorziehen“[6] - aber Aufmerksamkeit und Fokus sind deshalb nicht automatisch identisch. Eine konkrete, auf die Thematik dieser Arbeit bezogene Definition können die vorgenannten Aspekte demnach nicht liefern. Sie stellen jedoch eine erste begriffliche Abgrenzung dar, auf der die folgende Betrachtung aufbauen kann.

[6] vgl. http://www.heise.de/tp/r4/artikel/6/6310/1.html; 09.07.2006

Definition

Diese Arbeit muss den Aufmerksamkeitsfokus im Sinne der Problemstellung definieren um somit eine Basis zur Lösung dieser zu schaffen. Die Definition stellt darauf ab, die Benutzeroberfläche von Entwicklungsumgebungen als komplexes Bild zu interpretieren, von denen eine oder einige der verschiedenen (Bildschirm)Komponenten[7] und deren Inhalte vom Entwickler selektiert und hinsichtlich der Frage, worauf er seinen Geist fokussiert, anderen Komponenten vorgezogen werden.

Nicht zu verwechseln ist der Aufmerksamkeitsfokus mit dem Begriff des technischen Fokus, wie er von Entwicklern von Benutzeroberflächen verwendet wird. Dieser ist hauptsächlich gleichzusetzen mit dem aktuell aktivierten Fenster, oder einer Komponente in diesem, das den Fokus üblicherweise durch das Anklicken erlangt. In der Eclipse-Entwicklungsumgebung, auf welcher der FocusTracker aufsetzt, wird dem Benutzer der Fokus einer Oberflächenkomponente durch die farbliche Hervorhebung ihres Rahmens visuell signalisiert. Tastatureingaben werden in den meisten Fällen von den aktiven Komponenten verarbeitet, ausgenommen hiervon sind globale Tastaturkürzel. Dieser Fokus, also die aktuell aktivierte Komponente, ist bei der Betrachtung des Aufmerksamkeitsfokus ebenso relevant, steht jedoch nicht im Mittelpunkt der Betrachtung.

Den Aufmerksamkeitsfokus eines Entwicklers zu bestimmen ist im Vergleich zum technischen Fokus die schwierigere Aufgabe, da der Fokus von vielen Faktoren innerhalb der Entwicklungsumgebung, jedoch auch von zahlreichen externen Faktoren abhängen kann.[8] Die Aufgabe wird dadurch weiter verkompliziert, dass der individuelle Stil des jeweiligen Entwicklers in einer großen Bandbreite möglichen Programmier- und somit Aufmerksamkeitsverhaltens mündet.[9] Weiterhin kann - sobald mehr als ein Element im Sichtbereich vorhanden ist - nicht mit Sicherheit bestimmt werden, welches hiervon sich momentan im Fokus des Entwicklers befindet. Dies kann nur anhand von weiteren Faktoren, die im Zuge dieser Arbeit diskutiert werden und die mit einer gewissen (quantifizerbaren) Unschärfe behaftet sind, bestimmt werden.

[7] so zählen der Quelltext-Editor, die Outline (welche mittels Java-Elementen der Navigation im Quelltext dient) oder die Konsole (welche u.a. die Ausgaben von Programmen anzeigt) zu den Bildschirmkomponenten, die gemeinsam ein komplexes, flexibles Ganzes bilden

[8] Faktoren innerhalb der Entwicklungsumgebung, die einen Hinweis auf den Aufmerksamkeitsfokus geben, können beispielsweise das Selektieren von Elementen, das Editieren von Elementen und deren Sichtbarkeit sein; externe Faktoren, die die zuvor genannten Faktoren u.U. relativieren, können Gespräche mit Dritten, Abwesenheit oder ähnliches sein: In diesen Fällen kann ein Element, welches selektiert und sichtbar ist, dennoch nicht im Aufmerksamkeitsfokus des Entwicklers sein

[9] der eine navigiert größtenteils über die Outline, während ein anderer die Navigation über Tastenkürzel vorzieht; manch Entwickler folgt seinem tatsächlichen Fokus mit der Maus, er führt den Mauszeiger über die aktuell fokussierten Stellen im Editor, selektiert sie eventuell sogar; andere studieren minutenlang eine Methode um sie zu verstehen, ohne eine einziges Mal die Maus zu bewegen: Diese Ereignisse werden jeweils vom FocusTracker über Regeln ausgewertet; vgl. Abschnitt 2.3 Bewertungsregeln

In dieser Arbeit wird der Aufmerksamkeitsfokus daher wie folgt definiert:

Der Aufmerksamkeitsfokus bezeichnet den aktuellen Mittelpunkt des Interesses eines Entwicklers in einer Entwicklungsumgebung. Er bezieht sich im Gegensatz zum technischen Fokus eines Entwicklers auf Projekte und Java-Elemente wie Pakete, Klassen, Methoden oder Felder, die der Entwickler mit einer quantifizierbaren Unschärfe anderen Elementen vorzieht.

Fuzzy Fokus

Wie oben beschrieben, stehen unter anderem Ereigniswahrscheinlichkeiten und -unschärfen im Mittelpunkt der Diskussion des Aufmerksamkeitsfokus.

Selbst unter der (vereinfachenden) Annahme, dass der Aufmerksamkeitsfokus über die Augenbewegungen des Entwicklers messbar wären, sind Aussagen über den Aufmerksamkeitsfokus immer mit Unsicherheiten behaftet. Auch bei der (im Vergleich zur Messung der Augenbewegung sicherlich noch ungenaueren) Erfassung des Aufmerksamkeitsfokus durch Sensoren innerhalb der Entwicklungsumgebung wird es ebenfalls zu Unsicherheiten kommen. Diese Unsicherheiten resultieren aus systematischen Messungenauigkeiten hinsichtlich der entwicklungsinternen Faktoren sowie externen Faktoren wie z.B. der Fokussierung auf entwicklungsunabhängige Faktoren wie Telefonate o.ä., die die Messungenauigkeiten verstärken. Vor diesem Hintergrund ist eine definitive Aussage darüber, ob sich ein Element sicher im Aufmerksamkeitsfokus eines Entwicklers befindet oder nicht, nicht möglich, da jegliche Sensoren (heutzutage) niemals Aufschluss darüber geben werden, was im Bewußtsein des Programmierers vor sich geht. Die vertrauensvollste Aussage hierüber kann allein vom Entwickler selber getroffen werden, da wir davon ausgehen, dass der Programmierer weiß (und auch er alleine wissen kann), was er gerade tut und auf welchen Teil eines Quelltextes er sich konzentriert. Der FocusTracker muss daher die Problematik dieser Unschärfe in der Messung angehen. Den Umstand, Unschärfen und Unsicherheiten durch Zwischenwerte darzustellen, greift wiederum die Fuzzy-Logik-Theorie auf, auf die nun eingegangen werden soll.

Die Fuzzy-Logik-Theorie beschäftigt sich auf Basis der „Fuzzy-Set-Theorie“, der unscharfen Mengenlehre, mit der Erweiterung der zweiwertigen Boole'schen Logik um Zwischenwerte, um auch Unschärfen darstellen zu können. Diese Arbeit fügt der Boole'schen Menge „Aufmerksamkeitsfokus“ mit den Mengenelementen „Element ist im Aufmerksamkeitsfokus“ und „Element ist *nicht* im Aufmerksamkeitsfokus“ diskrete Zwischenwerte zu, um die notwendige Unschärfe abzubilden. Diese Zwischenwerte, die sog. Wahrheitswerte, liegen zwischen der „falschen“ Aussage 0 (Element ist nicht im Aufmerksamkeitsfokus) und der „wahren“ Aussage 1 (Element ist im Aufmerksamkeitsfokus) und werden üblicherweise mittels Fuzzy-Funktionen unscharfen, qualitativen (verbalen) Skalenniveaus zugeordnet. Für den vorliegenden Fall bedeutet dies, dass ein Wahrheitswert von 0,8 einer qualitativen Ausprägung wie „Element ist stark

im Aufmerksamkeitsfokus“ zugeordnet werden könnte. Die Implementierung dieser Fuzzy-Funktionen erfolgt jedoch nicht im Zuge dieser Aufgabenstellung; der FocusTracker exportiert bewusst die diskreten quantitativen Wahrheitswerte, um Informationsverluste durch deren Abbildung auf qualitative Werte zu vermeiden.[10]

Zumindest die Bestimmung des Aufmerksamkeitsfokus auf der Ebene von Projekten in der Entwicklungsumgebung kann - unter Vernachlässigung oben genannter externer Faktoren - als sicher gelten. Wird ein Projekt oder eine zugehörige Klasse geöffnet bzw. ein Element in einer Java-Klasse betrachtet, so ist das Projekt per se im Fokus, da zu jedem Zeitpunkt nur ein Projekt in der Entwicklungsumgebung aktiv sein kann. Gleiches gilt für Pakete und Dateien. Der Fokus auf alle weiteren Java Elemente, wie zum Beispiel Felder oder Methoden, ist hingegen nicht eindeutig bestimmbar. Von ihnen sind in der Regel mehrere sichtbar und damit - in Abhängigkeit der zugrunde liegenden Bewertungsregeln[11] - möglicherweise im Fokus. Wenn eine Methode den sichtbaren Bereich vollständig ausfüllt, so ist es ziemlich sicher, dass sich diese im Fokus des Entwicklers befindet. Wenn sich jedoch auch weitere Elemente im sichtbaren Bereich befinden, muss erneut von einer gewissen Unsicherheit ausgegangen werden, auf die die erweiterte Boole’sche Menge des Fuzzy-Logik Ansatzes abzielt.

1.3.2 Mikroprozess - ECG

Der Mikroprozess, der die Aktivität des Entwicklers beschreibt, stellt die Grundlage für die Sammlung von Daten über den Aufmerksamkeitsfokus dar und wird daher an dieser Stelle beschrieben. Der Begriff „Mikroprozess“, wie er von der Arbeitsgruppe Software Engineering verwendet wird, wird von Sebastian Jekutsch für diese Arbeit wie folgt definiert: „A micro-process is the detailed, every-day process of programming (low-level designing, coding, comprehending, testing, understanding) described as a sequence of activities, i.e. Events.“[Jek05]

Als Softwareentwicklungsprozess werden üblicherweise eine Vielzahl von „High-Level“ Aufgaben und -Phasen wie Planung, Design oder Dokumentation zusammengefasst. Die Betrachtung dieser „Makro“-Phasen sei laut Jekutsch jedoch zu grob. Er begründet dies damit, dass auf der Makro-Ebene die Art der Ausführung sowie notwendige Details der betrachteten Phasen nicht dokumentiert bzw. beschrieben werden. Der bislang wenig beachtete „Mikroprozess“, wie er von der Arbeitsgruppe Software Engineering erforscht wird, füllt diese Lücke. Durch die Auswertung der einzelnen Entwicklungsschritte im Mikroprozess verspricht sich die Arbeitsgruppe, fehleranfällige Schwächen in Implementierungen aufzuzeigen, die ohne Kenntnis über den Mikroprozess möglicherweise unerkannt blieben. Hierzu gehört unter anderem das nachträgliche Ändern

[10] bzgl. der Fuzzy-Funktionen und des Umgangs mit den exportierten diskreten Zwischenwerten vgl. Kapitel 2.5.2 Abschnitt ECGExport

[11] vgl. Kapitel 2.3 Bewertungsregeln

von dupliziertem Quelltext, ohne die Änderung auch an der Dupliziervorlage vorzunehmen. Würde der Mikroprozess während der Entwicklungsarbeit erfolgreich aufgezeichnet und analysiert, könnte der Entwickler noch während der Arbeit vor einem solchen Fehler gewarnt werden. Da der Begriff des „Mikroprozesses“ bereits in vielen anderen Forschungsprojekten im Bereich der Softwareentwicklung zur Anwendung kommt, grenzt ihn die Arbeitsgruppe offiziell als „Actual Process“ ab um aufzuzeigen, dass Prozesse nicht durch diesen definiert, sondern beobachtet werden.

Um Mikroprozesse auszuwerten, entstand im Rahmen einer Diplomarbeit das ElectroCodeoGram (ECG)[12]; ein System zur Betrachtung des Mikroprozesses. Mithilfe dieser Software ist es möglich, den Mikroprozess der Softwareentwicklung über Sensoren aus den Programmierwerkzeugen heraus aufzuzeichnen und ad hoc zu analysieren.[13]
Die Arbeitsgruppe verwendet daher das ECG, um u.a. zu einem besseren Verständnis über die Ursachen der Entstehung von Softwaredefekten zu gelangen. Die Erweiterung des ECGs durch diese Arbeit unterstützt somit diesen Ansatz.

1.3.3 Verwandte Ansätze – aufgabenbasierte Entwicklungsumgebungen

Um Änderungen an einer Software vornehmen zu können, muss der Entwickler die Software untersuchen, um deren Quelltext und Architektur nachzuvollziehen. Aufgabenbasierte Entwicklungsumgebungen[14] sollen den Entwickler darin unterstützen, aufgabenbezogene von -unbezogenen Elementen zu unterscheiden. Eine Möglichkeit, Desorientierung während der Entwicklungsarbeit zu vermeiden, ist das Ausblenden von solchem Quelltext, der zur Lösung des Problems nicht notwendig ist: Eine „Informationsflut“ wird vermieden. Entwickler stehen des Weiteren einer Informationsflut gegenüber, wenn sie versuchen, sich in unbekannten Projekten zu orientieren. Hierbei wird der Entwickler zunächst durch viele Teile des Projekts navigieren und läuft dabei Gefahr, den Überblick zu verlieren. Informationen wie der Ausgangspunkt, die bereits besuchten Stellen im Quelltext und der Pfad, auf dem der Entwickler zur aktuellen Position gelangt ist, gehen in der Informationsvielfalt verloren.
Herkömmlichen Entwicklungsumgebungen sind nicht in der Lage, den Entwickler hierbei durch Aufzeichnung dieser Informationen zu entlasten. Um diesem Problem zu begegnen, wurden eine Reihe von Lösungsansätzen formuliert, die sich unter dem Begriff „Aufgabenbasierte Software-Entwicklungsumgebungen“ subsummieren lassen.
Die Theorie zu deren Umsetzung definiert u.a. Aufgaben (hier: Tasks), die vor Arbeitsbeginn in der Entwicklungsumgebung manuell angelegt oder geöffnet werden müssen. Die jeweiligen Aktivitäten werden in der Folge der aktuellen Aufgabe zugeordnet und gespeichert, um für die Aufgabe nicht relevanten Inhalte auszublenden und somit die Informationsflut zu reduzieren.

[12] das ECG ist auf http://projects.mi.fu-berlin.de/w/bin/view/SE/ElectroCodeoGram erhältlich
[13] vgl. [Sch05]
[14] vgl. [RM05]

Aufgabenbasierte Software-Entwicklungsumgebungen können im Idealfall selbst unterschiedlichen Entwicklern, die an identischen Aufgaben arbeiten, bereits einen Leitfaden vorgeben und so schon vom ersten Öffnen einer Aufgabe unnötige Pfade ausblenden. Der Entwickler gelangt direkt zu den aufgabenrelevanten Stellen im Quelltext. Mylar als Referenzimplementierung der hier beschriebenen Entwicklungsumgebungen befindet sich momentan in der Entwicklung durch Mik Kersten.[15] Mylar kann die für die Verwaltung von Aufgaben in vielen Projekten eingesetzten Bug-Tracker[16]-Systeme wie BugZilla[17] und Jira[18] in dessen Aufgabenverwaltung integrieren, wodurch eine doppelte Verwaltung der Aufgaben in der Entwicklungsumgebung und dem verwendeten Bug-Tracker-System entfällt.

Mylar beinhaltet bereits eine rudimentäre Betrachtung des Aufmerksamkeitsfokus im Sinne dieser Arbeit, umfasst jedoch nur einen geringen Anteil der hier erarbeiteten Bewertungsregeln, wann sich ein Element tatsächlich im Fokus des Entwicklers befindet.[19] Eine in Betracht zu ziehende Erweiterung von Mylar um die hier zugrunde liegenden Bewertungsregeln (und das Sammeln der für die Bewertung nötigen Daten, z.B. bzgl. der Sichtbarkeit von Elementen) lässt der aktuell noch instabile Entwicklungsstatus der Software allerdings nicht zu. Ein weiterer Unterschied in der Betrachtung des Aufmerksamkeitsfokus stellen Felder dar. Gemäß der Mylar-Spezifikation werden diese aus technischen Gründen (hauptsächlich aufgrund des typischerweise kleinen Anteils von Feldern im Quelltext) nicht betrachtet. Der FocusTracker behandelt dagegen auch Felder.

Bewertungen von Mylar werden anhand der Erhöhung oder Herabsetzung des so genannten „degree-of-interest"(DOI), einem Punktesystem für die Bewertung des Aufmerksamkeitsfokus auf den Elementen, vorgenommen. Für die Selektion eines Elements werden eine gewisse Anzahl an DOI-Punkten hinzugefügt, während die übrigen Elemente an DOI-Punkten verlieren, wodurch eine Gewichtung des Aufmerksamkeitsfokus über die Elemente generiert wird. Als visuelle Hilfestellung für den Entwickler wird der DOI für jedes Element anhand eines Dekorators[20] durch die Sättigung des Farbtons

[15] http://www.eclipse.org/mylar

[16] Bug-Tracker (engl. bugreport, bugtracker) sind in der Software-Entwicklung eingesetzte Computerprogramme, die als Werkzeug der Erfassung und Dokumentation von Programmfehlern dienen. Mit ihnen werden, oft interaktiv und im Internet, auch Status- oder Feature-Berichte geschrieben. (Wikipedia http://de.wikipedia.org/wiki/Bug-Tracker)

[17] http://www.bugzilla.org

[18] http://www.atlassian.com/software/jira

[19] Mylar betrachtet z.B. ausschließlich das Selektieren von Elementen, wohingegen hier auch die Sichtbarkeit auf dem Bildschirm als elementare Bewertungsregel angewendet wird; dieser Fall tritt beispielsweise dann ein, wenn eine Methode, in der sich der Cursor befindet, aus dem Sichtbereich gescrollt wird; obgleich sich der Cursor nach wie vor in der selektierten Methode befindet und Mylar die Methode weiterhin als im Fokus befindlich ansieht, wird der Aufmerksamkeitsfokus gemäß des hier entwickelten FocusTracker auf einem Element im Sichtbereich liegen. Vgl. Abschnitt 2.3 Bewertungsregeln

[20] vgl. Kapitel 2.5.3 Abschnitt FocusTrackerDecorator

der Hintergrundfarbe dargestellt. Ein höherer DOI führt zu einer starken Sättigung, ein niedriger lässt die Farbe verblassen. Durch diese visuelle Unterstützung - die ihrerseits eine Informationsflut, wie sie durch die bloße Angabe der DOI-Punkte sogar noch verstärkt würde, verhindert - ist eine adäquate Übersicht über die aufgabenbezogenen Elemente auch ohne Nutzung des Mylar-Filters, der alle Elemente ohne positiven DOI ausblendet, gegeben.

2 Technische Umsetzung der Problemstellung

Der nachfolgende Abschnitt gliedert sich in fünf Teile. Der Architektur und den Designentscheidungen, welche die zweite Hälfte dieses Abschnitts bestimmen, werden die inhaltlichen und technischen Anforderungen und die dem FocusTracker zugrundeliegenden Bewertungsregeln vorangestellt. Da sich insbesondere Designentscheidungen und Implementierung gegenseitig beeinflussen, sind diese nicht strikt als aufeinander folgende Arbeitsschritte zu verstehen, sondern vielmehr als in einigen Punkten miteinander verzahnte Phasen.

2.1 Inhaltliche Anforderungen

Die inhaltlichen Anforderungen ergeben sich sowohl direkt aus der Aufgabenstellung als auch aus den Erfahrungen in der Entwicklungsphase.

2.1.1 Fuzzy-Logik

Um den Aufmerksamkeitsfokus im Sinne der Definition[21] bestimmen zu können, muss der Fuzzy-Logik-Ansatz mit den notwendigen Wahrheitswerten zwischen „wahr“ und „falsch“[22] gewählt werden, da eine sichere Entscheidung, ob sich ein Element im Sinne eines Boole'schen System im Aufmerksamkeitsfokus befindet, häufig nicht möglich ist.[23]

2.1.2 Tiefe der Betrachtung

Eine weitere der Aufgabenstellung direkt entnommene Anforderung ist die Granularität der zu betrachtenden Daten. Es sollen Informationen über das aktuelle Projekt sowie die Java-Elemente Pakete, Klassen, Methoden und Felder betrachtet werden. Eine darüber hinaus gehende Betrachtung des Aufmerksamkeitsfokus, zum Beispiel eine zeilengenaue Betrachtung des Quelltextes, erfolgt nicht. Ein Grund hierfür liegt in der

[21] vgl. Abschnitt 1.3.1 Aufmerksamkeitsfokus
[22] vgl. Kapitel 1.3.1 Abschnitt Fuzzy Fokus
[23] vgl. Abschnitt 1.3.1 Aufmerksamkeitsfokus

Schwierigkeit, dass eine solch detaillierte Betrachtung ein noch höheres Maß an Unsicherheit hinsichtlich verlässlicher Aussagen aufgrund eingeschränkter Indikatoren[24] für deren Detektion birgt.

2.1.3 Inaktivität des Entwicklers

Das System muss einen Mechanismus besitzen, um zu ermitteln, ob der bestehende Fokus verloren gegangen ist. Da nur Ereignisse innerhalb des Computersystems betrachtet werden, soll dies über die Feststellung der Inaktivität des Entwicklers erfolgen.

2.1.4 Zeitnaher ECG-Export

Aus Erfahrungen in der Entwicklungsphase ergibt sich die Anforderung, dass Ereignisse bereits während ihres Auftretens ausgewertet werden sollten. Ein zweiphasiger, zeitlich versetzter Prozess, in dem zunächst die Ereignisse des Mikroprozesses gespeichert und deren Auswertung zur Bestimmung des Aufmerksamkeitsfokus zu einem späteren Zeitpunkt erfolgt, widerspricht zudem dem geforderten zeitnahen Exports der Daten an das ECG.

2.1.5 Konfigurierbarkeit

Gefordert ist des Weiteren die außerhalb des Quelltextes mögliche Konfigurierbarkeit von Parametern der Bewertungsfunktionen, die darüber entscheiden, mit welcher Unschärfe sich ein Element im Aufmerksamkeitsfokus des Entwicklers befindet. Um eine größtmögliche Konfigurierbarkeit hinsichtlich der Bewertungen zu gewährleisten, verfolgt diese Arbeit den Ansatz der regelbasierten Informationsverarbeitung.[25] Diese inhaltliche Anforderung gewährleistet, dass nicht nur die Änderungen von einzelnen Parametern möglich sind, sondern komplette Bewertungsfunktionen ohne tiefe Kenntnis über bzw. Änderung des FocusTracker-Quelltextes austauschbar sind. Die Möglichkeit der Änderung von Parametern ist m.E. grundsätzlich notwendig, da die gewählten Parameter, die einer Implementierung zugrunde liegen, anhand zukünftiger Forschungserkenntnisse einer ständigen Überprüfung und Anpassung unterzogen sein sollten.

[24] hier könnte lediglich die direkte Texteingabe Aufschluss über die fokussierte Zeile geben; denn selbst die Cursorposition lässt im Allgemeinen keine verlässliche Aussage darüber zu, ob der Fokus auf einer Zeile liegt, da der Cursor häufig zum Scrollen verwendet wird und sich bei der Navigation über die Outline nicht zwangsläufig im Sichtbereich befinden muss

[25] für den FocusTracker kommen hierfür eigens entworfene Bewertungsregeln zur Anwendung; so sollte das Schreiben von Quelltext ein guter Indikator dafür sein, dass der Aufmerksamkeitsfokus auf der aktuellen Zeile liegt; Maus- und Cursoraktivität wie Scrollen, Verschieben und Selektieren von Text sollten ähnliche Indikatorfunktionen ausüben; vgl. Abschnitt 2.3 Bewertungsregeln

2.2 Technische Anforderungen

Aus der Aufgabenstellung folgt, dass der FocusTracker in Form eines Eclipse-Plugins als ECG-Sensors implementiert werden muss. Er muss als Teil des ECG (unter Lizenz der GPL) frei bezogen werden können.[26]

2.2.1 Performanz

Die Erfahrungen aus der Evaluation von Mylar[27] offenbaren negative Performanzauswirkungen in der Entwicklungsumgebung. Dies setzt die Wahrscheinlichkeit herab, dass Mylar tatsächlich zum Einsatz gelangt, da der Entwickler im Allgemeinen nicht bereit sein dürfte, Nachteile durch Geschwindigkeitseinbußen in Kauf zu nehmen. Eine hieraus abgeleitete technische Anforderung an den FocusTracker ist, dass durch ihn keine spürbaren Nachteile für die Performanz entstehen dürfen. Im Zusammenspiel mit der inhaltlichen Anforderung, feingranulare Daten aufzuzeichnen und auszuwerten, resultiert jedoch die Problematik, dass mit steigendem Messvolumen und steigender Messfrequenz eine ebenso steigende Performanzbelastung einhergeht. Die Vermeidung unnötiger teurer Messungen ist daher eine wichtige technische Anforderung.

2.2.2 Sammeln von Daten - Eclipse JDT

Aus den inhaltlichen Anforderungen resultiert, dass der FocusTracker zur Betrachtung von Java-Elementen entwickelt wird. Die Grundlage für die Unterstützung von Java in der Entwicklungsumgebung Eclipse bilden die „Java Development Tools“-Plugins (JDT). Als technische Anforderung hieraus folgt, dass der FocusTracker Informationen, die aus Quellen wie den JDT-UI-Plugins (User Interface Plugin) erstellt werden, verarbeiteten können muss. Um den Aufmerksamkeitsfokus hinsichtlich Elementen in der Eclipse-Entwicklungsumgebung zu bestimmen, muss der FocusTracker daher die von der Entwicklungsumgebung bereit gestellten Daten nutzen können. Diese Daten werden von oben genannten Plugins der Entwicklungsumgebung erzeugt und auch für andere Anwendungsfälle innerhalb dieser genutzt.

Wird beispielsweise wie oben dargestellt ein Element selektiert, erscheint im so genannten „Ruler“[28] eine blaue Markierung vom Anfang bis zum Endpunkt des Elements, die auf der Grundlage der von der Entwicklungsumgebung bereit gestellten Daten generiert wird. Die Leistung des FocusTracker umfasst hierbei nicht allein die Bündelung der einzelnen, durch JDT-Plugins erstellten Daten, sondern vielmehr deren Anreicherung durch weitere Daten.[29]

[26] Quelle, über die der FocusTracker bezogen werden kann:
http://projects.mi.fu-berlin.de/w/bin/view/SE/ElectroCodeoGram

[27] vgl. Abschnitt 1.3.3 Verwandte Ansätze – aufgabenbasierte Entwicklungsumgebungen

[28] Leiste auf der linken Seite des Editors; siehe Abbildung 1

[29] vgl. Kapitel 2.5.2 Abschnitt Monitore

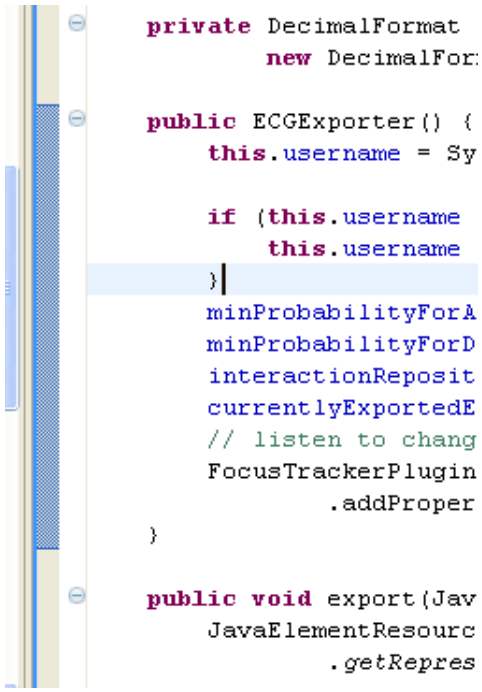

Abbildung 1: Darstellung einer Selektierung im Eclipse „Ruler“

2.2.3 Bestimmung des Aufmerksamkeitsfokus über Regeln

Die zuvor verbal-inhaltlich formulierte Definition darüber, wann sich ein Element mit einer gewissen Unschärfe im Aufmerksamkeitsfokus befindet, muss über Regeln[30] technisch umgesetzt werden. Grundlage hierfür muss die in den FocusTracker einzubindende „RulesEngine“[31] darstellen. Regeln und die zugehörigen relevanten Daten müssen von der verwendeten „RulesEngine“ zusammengeführt und verarbeitet werden können, um als Ergebnis die Unschärfe des Aufmerksamkeitsfokus für ein bestimmtes Element zu einem bestimmten Zeitpunkt auszugeben.

2.2.4 ECG-Anbindung

Der FocusTracker ist gemäß der Aufgabenstellung als ECG-Sensor zu implementieren. Die aufgezeichneten Daten bzgl. der Aufmerksamkeitsfoki müssen in konfigurierbaren zeitlichen Abständen in voraggregierter Form[32] an das ECG übermittelt werden. Da diese Daten in Form von XML-Fragmenten übertragen werden müssen, ist zunächst eine XML-Schema-Definition[33] (XSD) erforderlich, die die Struktur der zu übertragenden Daten definiert.

[30] vgl. Abschnitt 2.3 Bewertungsregeln
[31] vgl. Kapitel 2.5.2 Abschnitt RulesEngine
[32] vgl. Kapitel 2.5.2 Abschnitt ECGExport
[33] vgl. A Schemata der ECG-Sensordaten

2.3 Bewertungsregeln

Die Bewertungsregeln sind von zentraler Bedeutung für die Bestimmung des Aufmerksamkeitsfokus im Sinne dieser Arbeit. Diese Regeln beziehen sich auf Daten über Ereignisse aus unterschiedlichen Quellen,[34] die von Monitoren überwacht werden.[35] Durch die Konfigurierbarkeit der Bewertungsregeln Anpassungen durch zukünftige Erkenntnisse möglich. Nachfolgend werden einige der Grundlagen für die Formulierung der Bewertungsregeln des FocusTracker aufgeführt. Allen Regeln ist die Grundannahme gemein, dass lediglich sichtbare Elemente im Aufmerksamkeitsfokus liegen können.[36]

Quelle	**Element ist/wird...**	**Entwickler agiert durch...**	**Aufmerksamkeitsfokus 1.....0**
JavaEditor	selektiert (zuvor nicht selektiert)	Anklicken, Cursor Positionierung, Markieren	•
JavaEditor	selektiert (zuvor bereits selektiert)	Anklicken, Cursor Positionierung, Markieren	•
JavaEditor	Nachbarelement von Selektiertem	Anklicken, Cursor Positionierung, Markieren	•
JavaEditor	eingeklappt	ausklappen	•
JavaEditor	ausgeklappt	einklappen	•
JavaEditor	eingeklappt	Betrachtung (Element ist sichtbar)	•
JavaEditor	ausgeklappt	Betrachtung (Element ist sichtbar)	•
JavaEditor	im sichtbaren Bereich (füllt Editor komplett aus)	Scrollen, Änderung der Fenstergröße	•
JavaEditor	im sichtbaren Bereich (einzig sichtbares Element)	Scrollen, Änderung der Fenstergröße	•
JavaEditor	im sichtbaren Bereich (Anteil[37] gleich bleibend)	Scrollen, Änderung der Fenstergröße	•
JavaEditor	im sichtbaren Bereich (Anteil sinkend)	Scrollen, Änderung der Fenstergröße	•

[34] z.B. Maus (Selektion, Bewegung, Position), Tastatur (Selektion, Eingabe, Scrolling), Navigation (Outline, Scrolling) etc.

[35] vgl. Kapitel 2.5.2 Abschnitt Monitore

[36] hier unterscheidet sich der FocusTracker grundsätzlich von Mylar; vgl. Abschnitt 1.3.3 Verwandte Ansätze – aufgabenbasierte Entwicklungsumgebungen

[37] Anteil des aktuell sichtbaren Bereichs des Elements vom gesamten Sichtbereich

Quelle	**Element ist/wird...**	**Entwickler agiert durch...**	**Aufmerksamkeitsfokus 1.....0**
JavaEditor	im sichtbaren Bereich (Anteil steigend)	Scrollen, Änderung der Fenstergröße	•
JavaEditor	nicht im sichtbaren Bereich	Scrollen, Änderung d. Fenstergröße, Schließen bzw. Wechseln d. Editors	•
JavaEditor	textlich geändert	Tastatureingabe	•
JavaEditor	vom Mauszeiger berührt	Mausbewegung	•
Outline	selektiert (zuvor nicht selektiert)	Anklicken, Tastendruck	•
Outline	selektiert (zuvor bereits selektiert)	Anklicken, Tastendruck	•
Outline	Nachbarelement von Selektiertem	Anklicken, Tastendruck	•
Console	selektiert (zuvor nicht selektiert)	Anklicken, Tastendruck	•
Console	selektiert (zuvor bereits selektiert)	Anklicken, Tastendruck	•
Package Explorer	selektiert (zuvor nicht selektiert)	Anklicken, Tastendruck	•
Package Explorer	selektiert (zuvor bereits selektiert)	Anklicken, Tastendruck	•
Package Explorer	eingeklappt	ausklappen	•
Package Explorer	ausgeklappt	einklappen	•

Tabelle 1: Auswirkungen von ausgewählten Aktivitäten auf die Bestimmung des Aufmerksamkeitsfokus

Exemplarisch für die Entwicklung der Bewertungsregeln, denen u.a. die in der Tabelle 1 dargestellten Beziehungen zwischen Benutzeraktivitäten und Aufmerksamkeitsfokus zugrunde liegen, soll an dieser Stelle die Regel für die Sichtbarkeit von Elementen im Java-Editor hergeleitet werden. Die Grundüberlegung, wie die Sichtbarkeit bewertet werden soll, fußt auf der Relation des sichtbaren Bereichs des Elements im Verhältnis zum im Java-Editor sichtbaren Gesamtbereich. Füllt also ein Element den gesamten sichtbaren Bereich aus, soll es sich sicher im Aufmerksamkeitsfokus befinden. Ihm wird also der Wert 1,0 zugeordnet. Gleiches gilt, wenn es zwar nicht den komplett sichtbaren

Bereich ausfüllt, jedoch das einzige Element im sichtbaren Bereich des Java-Editors ist. Für den Fall, dass ein Element nicht den kompletten sichtbaren Editorbereich ausfüllt und weitere Elemente sichtbar sind, wird die Bewertung wie oben beschrieben über den Anteil geregelt. Liegt der sichtbare Anteil des Elements

- bei mehr als 90% vom sichtbaren Gesamtbereich, so wird dem Element ein Aufmerksamkeitsfokus von 1,0;
- zwischen 50% und unter 90%, soll der Aufmerksamkeitsfokus 0,5 betragen;
- zwischen 25% und unter 50%, soll der Aufmerksamkeitsfokus 0,25 betragen;
- bei unter 25%, wird ein Aufmerksamkeitsfokus von 0,1 zugewiesen.

Steigt also der sichtbare Anteil z.B. durch Scrollen oder die Änderung der Fenstergröße, steigt auch der zugewiesene Aufmerksamkeitsfokus und umgekehrt. Ist das Element nicht mehr im sichtbaren Bereich (z.B. durch Schließen oder Wechseln des Editors), so wird dem Element ein Aufmerksamkeitsfokus von 0,0 zugewiesen. Der Quelltext dieser Regel ist im Anhang A Regeldefinitionen unter „JavaEditor Visibility Changed“ ebenso wie alle anderen Regeln aufgeführt.

2.4 Designentscheidungen

Die nachfolgend dargestellten Entscheidungen bzgl. der Umsetzung des oben beschriebenen FocusTracker-Konzepts sind die wichtigsten Lösungsansätze für die im Zuge der Implementierung auftretenden Problemstellungen. Eine Aufzählung aller Designentscheidungen ist an dieser Stelle nicht möglich. Nicht aufgeführt sind für die Lösung unbedeutendere Designentscheidungen wie z.B. solche hinsichtlich der Klassenhierarchie und Paketstruktur.

2.4.1 Regelbezogene Designentscheidungen

Detektion von Inaktivität

Der FocusTracker muss die Inaktivität eines Benutzers ermitteln können. Diese ist jedoch abhängig vom Stil des Benutzers und kann daher nicht sicher bestimmt werden.

Die Regel, nach denen der FocusTracker Inaktivität detektieren soll, umfasst das Ausbleiben von Mausbewegungen und Tastatureingaben. Wird eine konfigurierte Zeit ohne messbares Ereignis überschritten, so soll die Sitzung als inaktiv deklariert und erst mit der nächsten Mausbewegung oder dem nächsten Tastendruck wieder aktiv werden. Aufgrund der Unsicherheiten, mit der die Ermittlung des Aufmerksamkeitsfokus einhergehen,[38] soll aus einer detektierten Inaktivität jedoch nicht augenblicklich

[38] vgl. Abschnitt 1.3.1 Aufmerksamkeitsfokus

ein nicht vorhandener Aufmerksamkeitsfokus resultieren. Vielmehr soll im Sinne der Fuzzy-Logik der Wahrheitswert zeitlich bedingt absinken, wie nachfolgend dargestellt wird.

Eine auf diese Art detektierte Inaktivität kann mehrere Ursachen haben. Sie muss nicht mit einer tatsächlichen Inaktivität des Entwicklers übereinstimmen. Die Ursachen hierfür reichen von konzentrierter Betrachtung eines Elements im Sinne eines sicheren Aufmerksamkeitsfokus, die zur Unterbrechung der Maus- und Tastaturaktivitäten führt, bis hin zur gänzlichen physischen Abwesenheit eines Benutzers im Sinne eines nicht vorhandenen Aufmerksamkeitsfokus. In beiden Extremfällen wird Inaktivität detektiert. Hier gilt es anhand von Studien oder gar durch die Lernfähigkeit der Software die Konfiguration der Zeit zu justieren, die vergehen muss, um eine tatsächliche Inaktivität mit geringer Unschärfe zu detektieren.[39]

Zeitliches Abfallen des Fokus bei Inaktivität

Eine wichtige Designentscheidung betrifft die Frage, wie mit der Inaktivität des Entwicklers vor dem Hintergrund oben aufgeführter Unsicherheiten umgegangen wird. Für diesen Fall betrachten wir ausschließlich Elemente, für die zum Zeitpunkt der eintretenden Inaktivität ein Aufmerksamkeitsfokus mit ungleich 0 bewertet wurde. Es ist davon auszugehen, dass nach einer gewissen Zeit ohne messbare Aktivität bzgl. dieser Elemente die vorherige Bewertung nicht mehr dem tatsächlichen Aufmerksamkeitsfokus entspricht. Es soll hier angenommen werden, dass er daher niedriger ist als vor einer detektierten Inaktivität. Dies kann aus oben genannten Gründen zu einem falschen Ergebnis führen, da der Entwickler im Extremfall durch hohe (lang andauernde) Konzentration im Sinne einer reinen Betrachtung ohne messbare Aktivität den Aufmerksamkeitsfokus tatsächlich erhöhen würde. Der FocusTracker hat jedoch keine Informationen hierüber, so dass diese Designentscheidung die genannte Unsicherheit nicht ausschließen kann. Aus diesem Grund soll nach einer (konfigurierbaren) Zeitspanne die Bewertung der Elemente bei detektierter Inaktivität stetig abfallen. Der FocusTracker setzt dies in Form einer abfallenden Gerade um, die als Funktion implementiert wird. Im produktiven Einsatz kann sich die Notwendigkeit einer Feinjustierung dieser Funktion herausstellen; für den größten Teil der Anwendungsfälle und den Beginn der Forschung sollte die gegenwärtig implementierte Funktion genügen.

Betrachtung von JavaDoc-Kommentaren

Des Weiteren von großer Bedeutung für die Auswertung der Regeln ist die Betrachtung von JavaDoc-Kommentaren. Hinsichtlich der Bewertungsregeln soll grundsätzlich gelten, dass das Selektieren von JavaDoc-Kommentaren mit dem Selektieren des eigentlichen Elements gleich zu setzen ist.

[39] vgl. Kapitel 4 Fazit und Ausblick - Erweiterungsmöglichkeiten

Bewertungspropagierung an Elementeltern

Eine ebenso nahe liegende wie auch sinnvolle Entscheidung betrifft die Bewertungspropagierung an Elementeltern. Elemente sollen jedoch nur dann die Bewertungen an ihre Eltern propagieren, wenn sich das Element im Sinne der messbaren Faktoren sicher im Aufmerksamkeitsfokus befindet, d.h. einen Wert von 1 aufweist. Grundlage für die Notwendigkeit dieser Propagierung stellt das Verhältnis des Elements zu den Elementeltern dar: das Element ist Bestandteil und Inhalt des Elternelements, so dass auch der Aufmerksamkeitsfokus Bestandteil des Elternelements sein muss.[40]

Elementgruppen

Für die Bewertungsregeln von Projekten, Paketen und Klassen sollen diese Elemente verallgemeinernd als Gruppen interpretiert werden. Für den Fall, dass sich ein Element aus einer Gruppe sicher im Aufmerksamkeitsfokus befindet, soll dies für kein anderes aus der gleichen Gruppe möglich sein.

Behandlung von Importstatements

Importstatements sollen in die Betrachtung nicht einbezogen werden, da diese durch Automatismen der Entwicklungsumgebungen für den Programmierer nebensächlich geworden sind: Es ist nicht mehr notwendig, sie manuell zu pflegen. Die Navigation bzw. die Betrachtung dieser ist daher extrem unwahrscheinlich. Die Designentscheidung, Importstatements nicht zu behandeln, begründet sich in einer optimalen Bewertung der anderen Elemente. Die Berücksichtigung von Importstatements in Bewertungsregeln würde die Ergebnisse hinsichtlich anderer Elemente verwässern.

Scrolling/Änderungen im Text

Aufgrund der oben genannten Performanzanforderungen ist es erforderlich, die zu verarbeitenden Datenmengen gering zu halten. Der FocusTracker soll daher nicht jedes Ereignis aufzeichnen und auswerten, sofern die Auswertung ein teurer Prozess ist.[41] Beim Scrollen müssen alle aktuell sichtbaren Elemente bestimmt werden, da jedoch für jede gescrollte Zeile eine Bewertung des Aufmerksamkeitsfokus aller Elemente erfolgen würde, stünde dies im Widerspruch zur oben genannten Performanzanforderung. Dieser Widerspruch wird in der vorliegenden Implementierung durch eine Verzögerung in der Bereitstellung von Informationen für die Bewertung von Ereignissen gelöst.

[40] befindet sich eine Methode mit Sicherheit im Aufmerksamkeitsfokus, so ist deren Elternelements, die zugehörige Klasse, mit der gleichen Sicherheit im Aufmerksamkeitsfokus des Entwicklers

[41] vgl. Kapitel 2.2.1 Abschnitt Performanz

2.4.2 Technische Designentscheidungen

Da viele der technischen Anforderungen[42] bereits Designentscheidungen implizieren, wird an dieser Stelle lediglich ein zentraler Aspekt behandelt.

Jobs vs Threads

Eclipse-Jobs als bereits bestehende Architekturen in der Eclipse-Plattform sind einmalig oder wiederkehrend auszuführende Aufgaben, die mit einer niedrigen Priorität abgearbeitet werden und daher kaum Einfluss auf die vom Benutzer wahrgenommene Performanz des Systems haben. Andere Aufgaben in der Entwicklungsumgebung, welche zum Beispiel die Interaktion mit dem Benutzer gewährleisten, haben dementsprechend Vorrang, so dass der Entwickler keine Verzögerungen durch rechen- oder zeitintensive Jobs hinnehmen muss. Jobs sind daher vor dem Hintergrund der Performanzanforderung für zeitunkritische Aufgaben die adäquate Wahl. Zu den zeitunkritischen Aufgaben zählen z.B. die Detektion von Inaktivität oder der Export von Daten an das ECG, welche dementsprechend im FocusTracker von periodisch ausgeführten Jobs übernommen werden sollen.

2.5 Implementierung und Architekur

2.5.1 Prototyp

Vor der Entwicklung des eigentlichen FocusTracker-Plugins habe ich einen experimentellen Prototypen zur Gewinnung erster Erkenntnisse über die grundsätzliche Erfüllbarkeit der Anforderungen entwickelt. Im Zentrum des Interesses lag dabei die Ereignisdetektion, d.h. die Messung der hier relevanten Mikroprozesse[43] des Entwicklers in der Entwicklungsumgebung.

Erst nach einer erfolgreichen Testphase, in dem der Prototyp erste zur Erfüllung der Anforderungen nötige Ereignisse[44] tatsächlich erfassen konnte, erstellte ich die Regeln zur Bewertung des Aufmerksamkeitsfokus, woraufhin wiederum der Umfang an zu erfassenden Ereignissen festgelegt werden konnte: Regeln und zu erfassende Daten bedingen sich somit gegenseitig. Oben genannten Schritten folgte der Entwurf der Architektur der übrigen Komponenten, worauf nun eingegangen werden soll.

[42] vgl. Kapitel 2.2 Technische Anforderungen

[43] relevante Mikroprozesse, die nicht bereits durch vorhandene ECG-Sensoren erfasst werden, werden in Kapitel 2.3 Bewertungsregeln dieser Arbeit beschrieben

[44] der Prototyp erfasste lediglich Scrollereignisse und Selektierungen in Quelltexten

ViewPart-Komponente	vom PartMonitor zuzuweisender Monitor
JavaEditor	JavaEditorMonitor
ClassFileEditor	JavaEditorMonitor
Outline	OutlineMonitor
PackageExplorer	PackageExplorerExpansionMonitor

Tabelle 2: Zuordnungen des PartMonitors

2.5.2 Kernkomponenten

Eine der Kernkomponenten des FocusTracker ist der „PartMonitor“, der die Monitore registriert. Diese Monitore bewerten wiederum die relevanten Ereignisse über eine weitere Kernkomponente, die „RulesEngine“. Die Monitore leiten die Bewertungen an den „EventDispatcher“ weiter, welcher wiederum die Ereignisse mittels „InteractionRepository“ vorhält und an registrierte Empfänger[45] verteilt. Der ECG-Export als Schnittstelle für das ECG ist die Ausgabekomponente des FocusTracker, die ihre Daten aus dem „InteractionRepository“ bezieht.

PartMonitor

Der PartMonitor überwacht die Oberfläche der Entwicklungsumgebung. Er erkennt Ereignisse, die Oberflächenkomponenten betreffen. Der PartMonitor reagiert auf die Ereignisse Öffnen, Schließen und Deaktivieren von ViewParts,[46] indem er entsprechende Monitore auf die ViewParts registriert bzw. deregistriert. Der PartMonitor gewährleistet demnach jederzeit eine dynamische Zuordnung für alle ViewParts. Für jede geöffnete ViewPart-Instanz existiert genau eine Monitor-Instanz über deren gesamten Lebenszyklus. Diese Instanzen werden anhand der Monitorklasse, die dem Ereignis zugrundeliegenden ViewPart zugeordnet ist, erzeugt.

Monitore

Monitore sind Komponenten, die andere Komponenten beobachten und bei Bedarf auf Ereignisse reagieren können.

Die Monitore des FocusTracker kapseln einen oder mehrere Listener und melden sie bei den zu beobachtenden Komponenten in Eclipse an. Empfangene Ereignisse werden wie nachfolgend beschrieben aufbereitet und mit Informationen der zugrundeliegenden Komponente angereichert. Dieses Konzept bietet den Vorteil, bei isolierter Betrachtung wenig aussagekräftige Ereignisse mit Informationen aus der Umgebung

[45] wie z.B. die LoggingView, siehe Kapitel 2.5.3 Abschnitt LoggingView

[46] siehe 2.5.3 Abschnitt GUI Komponenten

Monitor-Ebene	Monitor	Betrachtetes Ereignis
Benutzer	JavaEditorMonitor	Ein- bzw. Ausklappen
		Selektierung
		Position des Cursors
		Sichtbarkeit
		Textänderung (editieren)
		Mauszeigerposition
	JavaClassFileMonitor	Ein- bzw. Ausklappen
		Selektierung
		Position des Cursors
		Sichtbarkeit
		Mauszeigerposition
	PackageExplorer	Ein- bzw. Ausklappen
		Selektierung
	OutlineMonitor	Selektierung
	ConsoleMonitor	Selektierung
	ProjectLifecycleMonitor	Öffnen/Schliessen von Projekten
System	SystemMonitor	Tastatur & Maus
		Hauptfenster Aktiv - Inaktiv

Tabelle 3: Monitore und betrachtete Ereignisse

zu kombinieren. Zur Verdeutlichung dieses Konzepts sei nachfolgend der JavaEditor-Monitor betrachtet. Dieser implementiert u.a. einen Listener, der über Scrollereignisse oder Größenänderungen des Editorfensters informiert wird. Andere Listener werden u.a. über Textänderungen, Ein- bzw. Ausklappen von Elementen oder deren Selektierung informiert. Weiterhin referenziert der Monitor die Inhalte des aktuellen Editor-Dokuments. Nun wird bei der Änderung des Sichtbereichs von der Eclipse-Plattform als einziges Änderungsindiz die absolute Position des ersten sichtbaren Zeichens im Dokument angegeben. Diese Information ist isoliert betrachtet relativ wertlos, wird jedoch um Informationen durch den JavaEditorMonitor ergänzt. Anhand dieser Position des ersten sichtbaren Zeichens, ergänzenden Informationen aus dem referenzierten Dokument und weiteren Informationen darüber, welche Elemente ein- bzw. ausgeklappt sind, wird eine für die Weiterverarbeitung unerlässliche Liste von aktuell sichtbaren JavaElementen generiert.

Die implementierten Monitore arbeiten dabei auf zwei Ebenen: Auf der Benutzer-Ebene, wo Monitore direkt vom Benutzer initiierte Ereignisse[47] betrachten, und auf der System-Ebene, wo Monitore vom System erzeugte Ereignisse[48] beobachten.

[47] z.B. Selektierung
[48] z.B. Inaktivität

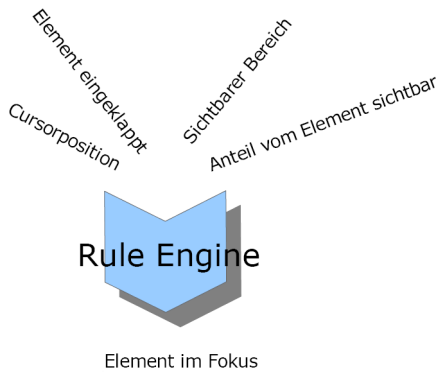

Abbildung 2: Funktionsweise der RulesEngine

RulesEngine

Als „RulesEngine“, die die von den Monitoren bereitgestellten Informationen anhand von Regeln[49] auswerten, kommt JBoss Rules (auch bekannt als Drools 3.0)[50] zum Einsatz. Die Gründe, warum JBoss Rules als Implementierung einer RulesEngine im FocusTracker zur Anwendung kommt, sind u.a.

- volle Kompatibilität zur JSR-94 Java Rule Engine API,[51]
- Zukunftssicherheit im Vergleich zu proprietären Produkten,
- Open-Source-Verfügbarkeit,
- Ausgereiftheit (aktuelle Version 3.0),
- produktiver Einsatz in kommerziellen Produkten sowie
- Anerkennung in der Java Rule Community.[52]

Der FocusTracker bindet die JBoss Rules Engine bewusst an exakt einer Stelle ein[53], wodurch gewährleistet ist, dass diese bei Bedarf ohne aufwändige Umstrukturierungen durch andere Implementierungen bzw. sogar grundlegend andere Konzepte substituierbar ist. Abbildung 2 visualisiert die Zusammenführung und Auswertung isolierter Daten in der RulesEngine anhand der vom JavaEditorMonitor bereitgestellten Daten.[54]

[49] bzgl. der für den FocusTracker relevanten Regeln vgl. Kapitel 2.3 Bewertungsregeln
[50] http://labs.jboss.com/portal/jbossrules
[51] http://www.jcp.org/aboutJava/communityprocess/review/jsr094/
[52] z.B. http://www.javarules.org
[53] die Klasse `Rating` kapselt die JBoss RulesEngine vollständig und bildet eine davon unabhängige Schnittstelle
[54] vgl. Kapitel 2.3 Bewertungsregeln und A Regeldefinitionen

EventDispatcher

Der EventDispatcher stellt den Mittelpunkt der Kommunikation und Ereignisverarbeitung des FocusTracker dar. Er empfängt die Ereignisse von den Monitoren und leitet diese an die registrierten Listener weiter. Zu diesen gehört das zentrale InteractionRepository, der Dekorator sowie ViewParts wie die FocussedElementsView und die LoggingView. Der EventDispatcher entspricht dem SingletonPattern, so dass systemweit lediglich eine Instanz des EventDispatchers auftritt.

InteractionRepository

Interaktionen sind alle vom Benutzer oder dem System initiierten Ereignisse; dementsprechend ist das InteractionRepository eine temporäre Datensenke für alle Interaktionen, die zur Laufzeit des Systems aufgezeichnet und zur Auswertung herangezogen werden. In dieser Senke wird jedem Java-Element, das im Laufe einer Sitzung zu einem Teil einer Interaktion wird, eben diese Interaktion zugeordnet. Es handelt sich hierbei um eine 1:n-Zuordnung: Alle Interaktionen auf diesem Element werden so lange vorgehalten, bis die relevanten Informationen an das ECG exportiert werden. So wird eine unnötige Datenmenge vermieden und der Resourcenbedarf des FocusTracker auf dem niedrigst möglichen Niveau gehalten.

Interaktionen dienen der Bewertung von Elementen. Allerdings können Elemente auch ohne direkte Interaktion neu bewertet werden. Dieses ist beispielsweise der Fall, sofern ein Element mit vollständiger Sicherheit in den Aufmerksamkeitsfokus gerückt ist, da sich dann kein weiteres Element auf gleicher Ebene ebenfalls im Aufmerksamkeitsfokus befinden kann.[55] Konkret ist dies beim Öffnen von Javadateien der Fall. In der Eclipse-Entwicklungsumgebung kann maximal ein Java-Editor zu einem Zeitpunkt geöffnet sein. Daraus folgt, dass sich ausschließlich die aktuell geöffnete Datei im Fokus befinden kann.

Interaktionen haben also Einfluss auf die betrachteten Elemente. Die Stärke des Einflusses hängt von der Interaktion und deren Bewertung im konkretem Fall ab. Interaktionen haben einen Zeitpunkt, einen Ursprung und beliebige, vom Ereignis abhängige weitere Parameter. Analog zur Aufteilung der Monitore unterscheidet der FocusTracker auch die Interaktionsebenen des Benutzers[56] und des Systems.[57]

[55] vgl. Kapitel 2.4.1 Abschnitt Elementgruppen

[56] Benutzerinteraktionen werden vom Benutzer aktiv herbeigeführt, zum Beispiel durch das Selektieren von Elementen, das Öffnen von Projekten/Dateien oder das Scrollen

[57] wird z.B. eine Inaktivität des Benutzers erfasst, wird diese als Systeminteraktion interpretiert

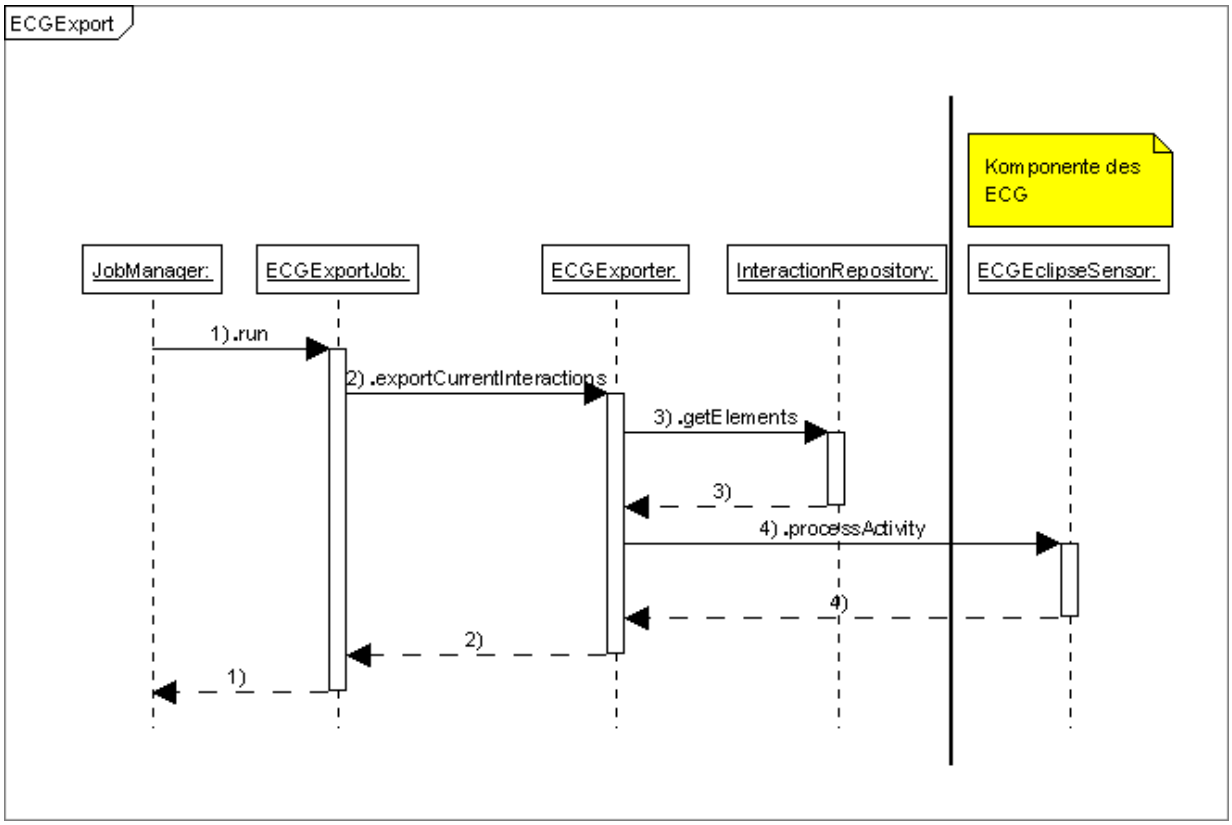

Abbildung 3: ECGExport

ECGExport

Der ECG-Export wird über einen Eclipse-Job[58] realisiert, der in konfigurierbaren Abständen aufgerufen wird. Er bereitet die Daten für den Export auf, indem er redundante Informationen entfernt und kombinierbare Daten zusammenfasst, und übergibt sie dem ECG. Die Schwelle für Ein- und Austritt von Elementen, die sich potentiell im Aufmerksamkeitsfokus befinden, ist anhand konfigurierbarer Unschärfewerte ebenso einstellbar. Die abschließende Auswertung der exportierten Daten, die anhand von Fuzzy-Funktionen[59] erfolgen soll, ist nicht Bestandteil des FocusTracker. Es werden daher keine Fuzzy-Funktionen implementiert, sondern lediglich deren Eingangsparameter an das ECG übergeben, welche diese wiederum durch Fuzzy-Funktionen auswerten kann.

[58] vgl. Kapitel 2.4.2 Abschnitt Jobs vs Threads
[59] vgl. Kapitel 1.3.1 Abschnitt Fuzzy Fokus

Der ECG-Export exportiert stetige, quantitative Unschärfewerte.[60] Der Grund, warum nicht bereits durch im FocusTracker implementierte Fuzzy-Funktionen diskrete, qualitative Unschärfewerte[61] ermittelt werden, liegt in dem damit verbundenen Informationsverlust. Nur hierdurch besteht die Möglichkeit, die Fuzzy-Funktionen zu einem späteren Zeitpunkt zu ändern und die bereits gesammelten (exportierten) Daten nachträglich neu zu verarbeiten.

2.5.3 GUI Komponenten

Die Monitor-Kernkomponenten greifen auf zahlreiche bestehende GUI-Komponenten, welche in der Ecplipse-Plattform „Views“ oder „ViewParts“ genannt werden, zurück. Diese „ViewParts“ unterstützen Editoren und stellen alternative Sichten sowie alternative Navigationsmöglichkeiten dar. Die nachfolgend aufgeführten Komponenten jedoch sind Ergebnisse dieser Arbeit: Sie wurden im Zuge der Entwicklung des FocusTracker implementiert und dienen der Transparenz der Arbeitsweise des FocusTracker durch visuelle Ausgabe. Ihre Informationen beziehen sie über den EventDispatcher.

LoggingView

In dieser View werden alle Interaktionen, die zu einer Bewertung von Elementen beigetragen haben, ausgegeben. Die Ausgabe beinhaltet den Zeitstempel, an dem die Interaktion detektiert wurde, den Ursprung[62], die Aktion[63], das zugrunde liegenden Java-Element und die Bewertung der Interaktion, wie sie die RulesEngine[64] ausgegeben hat.

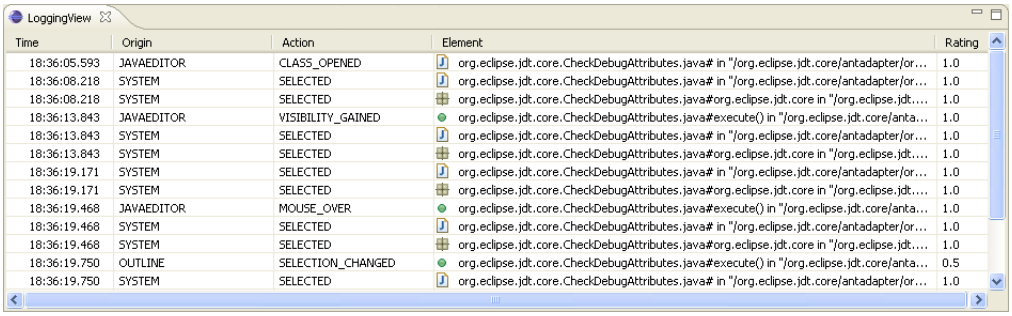

Abbildung 4: Screenshot der LoggingView

[60] hier wird, auch wenn keine Fuzzy-Funktionen implementiert werden, dennoch bereits auf den Fuzzy-Ansatz zurückgegriffen, vgl. 1.3.1 Abschnitt Fuzzy Fokus; Beispiele für stetige, quantitativen Unschärfewerte sind 0; 0,1; 0,23; 0,566 oder 1

[61] Beispiele für diskrete, qualitative Unschärfewerte sind „Element ist ein wenig im Aufmerksamkeitsfokus“ oder „Element ist stark im Aufmerksamkeitsfokus“

[62] z.B. JavaEditor, Outline, PackageExplorer usw.

[63] z.B. Markierung erfolgt, Sichtbarkeit erlangt usw.

[64] vgl. Kapitel 2.5.2 Abschnitt RulesEngine

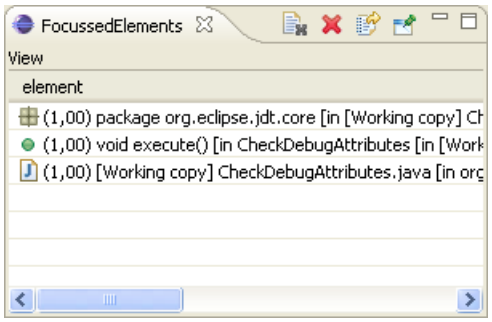

Abbildung 5: Screenshot der FocussedElementsView

FocussedElementsView

Die „Focussed-Elements-View“ zeigt Informationen zu den aktuell bewerteten Elementen in absteigender Reihenfolge ihrer Bewertungen bzgl. des Aufmerksamkeitsfokus an. Ähnlich wie der nachfolgend dargestellte Dekorator ermöglicht sie eine schnelle Übersicht über alle aktuell bewerteten Elemente. Der wesentliche Vorteil dieser View liegt im Gegensatz zum Dekorator jedoch darin, dass hier ausschließlich die bewerteten Elemente angezeigt werden. Bei großen Projekten, in denen der „Package Explorer“ oder die „Outline“ - eben jene Views, die der Dekorator durch Statusinformationen erweitert - über den im Bildschirm darstellbaren Bereich hinausgehen, sind häufig nicht mehr alle dekorierten Objekte sichtbar.

FocusTrackerDecorator

Dekoratoren sind visuelle Hinweise, die Statusinformationen von Ressourcen oder Objekten in Eclipse-Views wiedergeben. Sie werden in der gesamten Eclipse-Plattform benutzt, um einen schnellen Überblick der Stati von Objekten in Übersichten wie dem „Package Explorer“ oder der „Outline“ bereitzustellen. Dies kann durch das Hinzufügen von textuellen Informationen zu den Bezeichnungen, aber auch durch das Verändern deren Icons erfolgen. Standardmäßig ist der „Java Method Override Indicator“, der dem Icon von überschriebenen Methoden einen Pfeil hinzufügt, aktiviert. Der wohl jedem Entwickler bekannte „ProblemDecorator“ fügt dem Icon einer Javaklasse ein kleines rotes Kreuz hinzu, sobald ein Kompilierfehler festgestellt wurde. Wird eine Versionsverwaltung wie CVS[65] benutzt, so zeigt der „CVS-Decorator“ anhand eines Symbols in den Icons die Verfügbarkeit dieser Ressource in der Versionsverwaltung an. Der Bezeichner erhält zudem Informationen über den Dateityp und den Status der Datei. Es versteht sich von selbst, dass die Verhaltensweise von Dekoratoren frei konfiguriert oder sie komplett ausgeschaltet werden können.

[65] *Concurrent Versions System*, weit verbreitetes System für die Versionsverwaltung

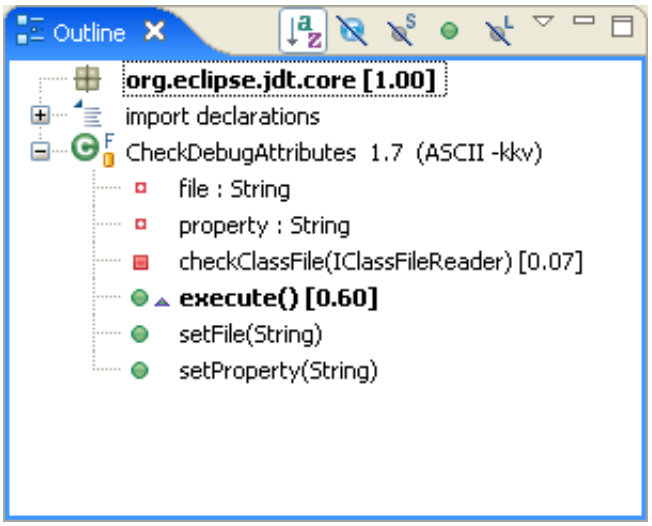

Abbildung 6: Screenshot des FocusTrackerDecorator in der Outline

Der hier implementierte FocusTrackerDecorator fügt dem Bezeichner die aktuelle Bewertung des Elements als Suffix hinzu, sofern das Element bereits bewertet wurde und die Bewertung des Aufmerksamkeitsfokus größer als 0 ist. Um Elemente mit hohen Bewertungen von solchen mit geringeren deutlich abzugrenzen, werden Elemente mit einer Bewertung von > 0.5 (konfigurierbar) durch Setzen des Schrifttyps auf „fett“ optisch hervorgehoben.

In der Architektur des FocusTracker tritt der FocusTrackerDecorator als ein weiterer InteractionListener auf, der sich beim EventDispatcher registriert, um über alle Änderungen von Bewertungen unterrichtet zu werden. Auf diese Änderungen wird inkrementell reagiert, d.h. es werden ausschließlich die Bezeichner von betroffenen Elementen aktualisiert.

Der FocusTrackerDecorator war eine geeignete Komponente, um die Validität von Bewertungen in der Entwicklungsphase, in den ersten Testläufen und in der Evalutationsphase zu bestimmen, da die Bewertungen in Echtzeit angezeigt werden. Somit konnten zeitaufwändige Umwege über Logging-Ausgaben oder Vergleiche von Videoaufzeichnungen von Sitzungen und den Daten aus dem ECG-Export für offensichtlich falsche Bewertungen vermieden und zeitnah nachgespielt werden.

2.5.4 Klassendiagramme

Nachdem die vorherigen Abschnitte die Grundsätze der Implementierung des FocusTracker dargelegt haben, stellt dieser Abschnitt die FocusTracker-Architektur anhand der zentralen Klassendiagramme dar.

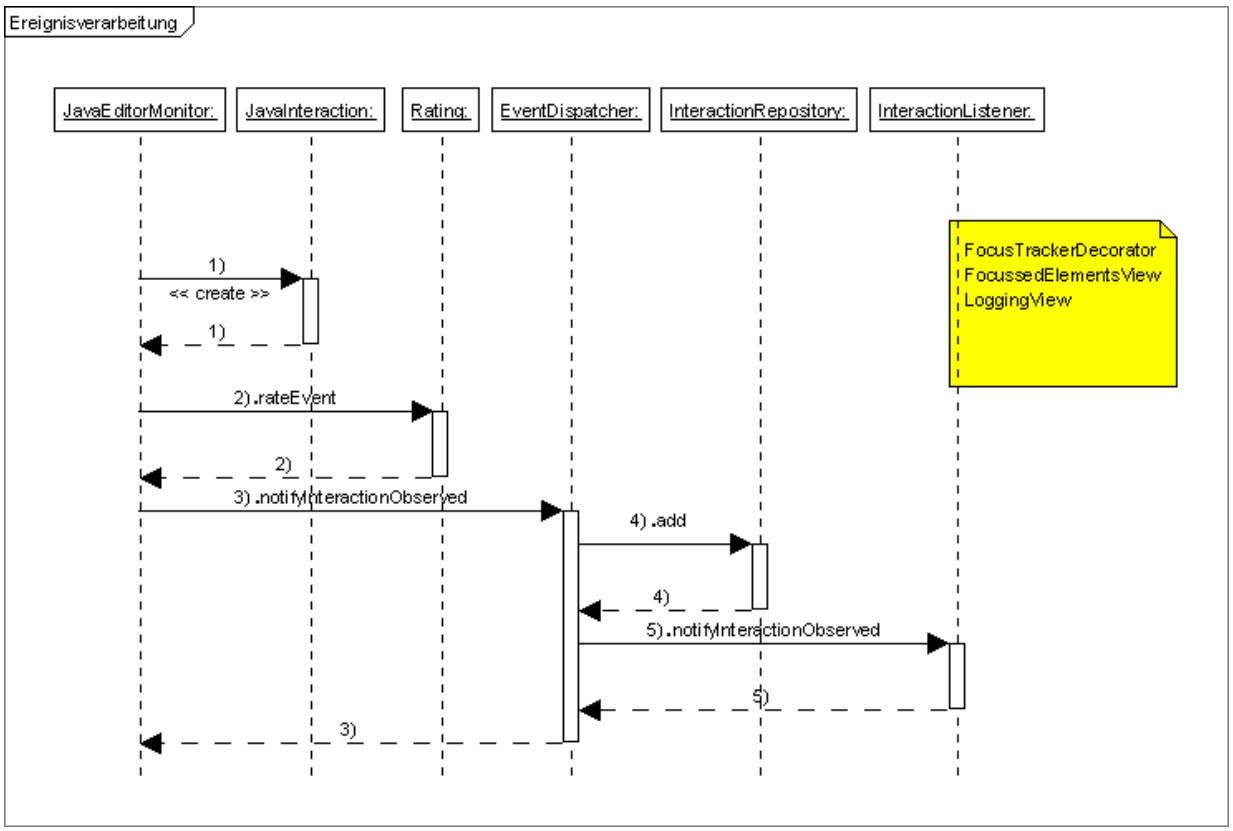

Abbildung 7: Schaubild der Ereignisverarbeitung

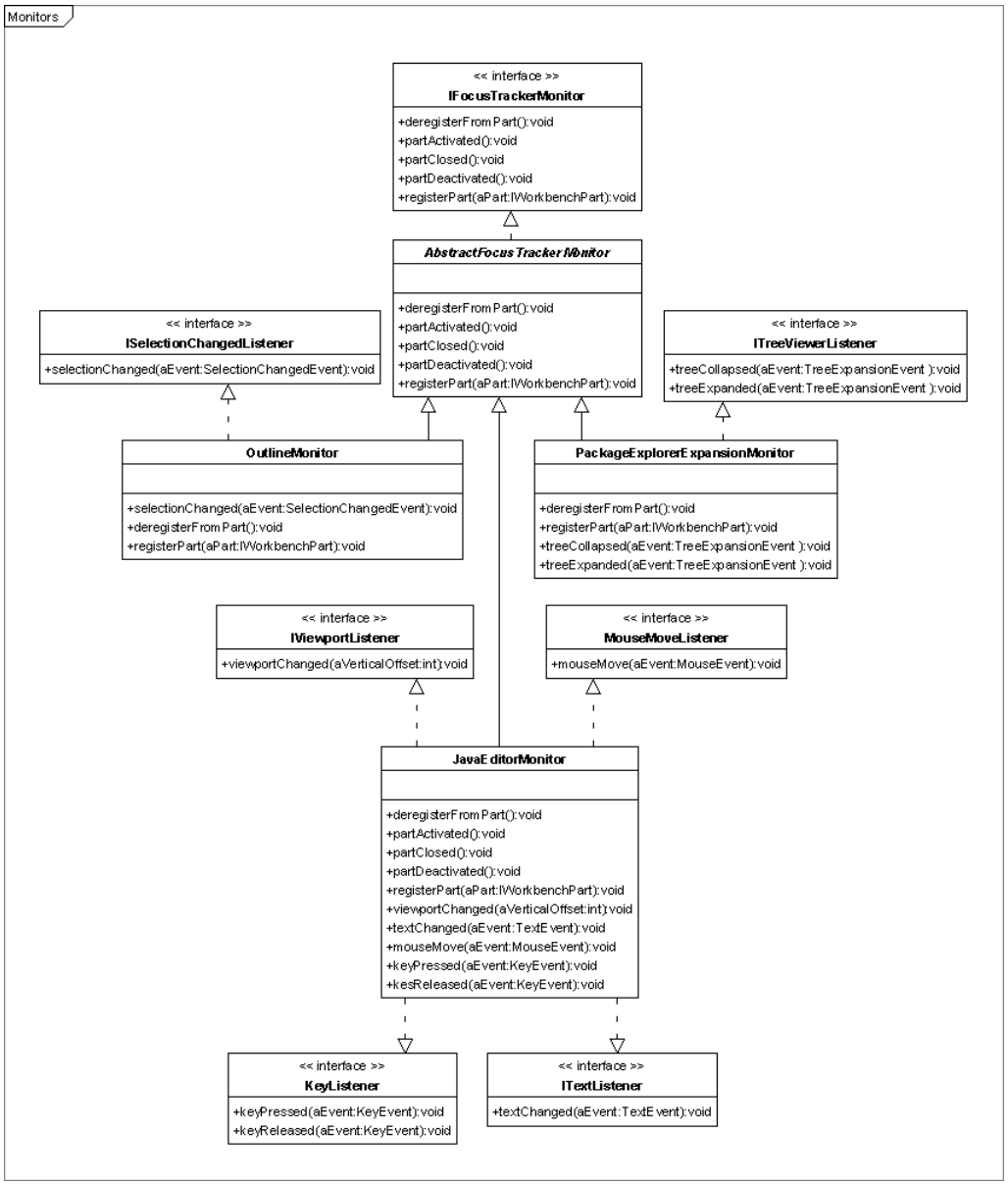

Abbildung 8: Klassenhierarchie der Monitore

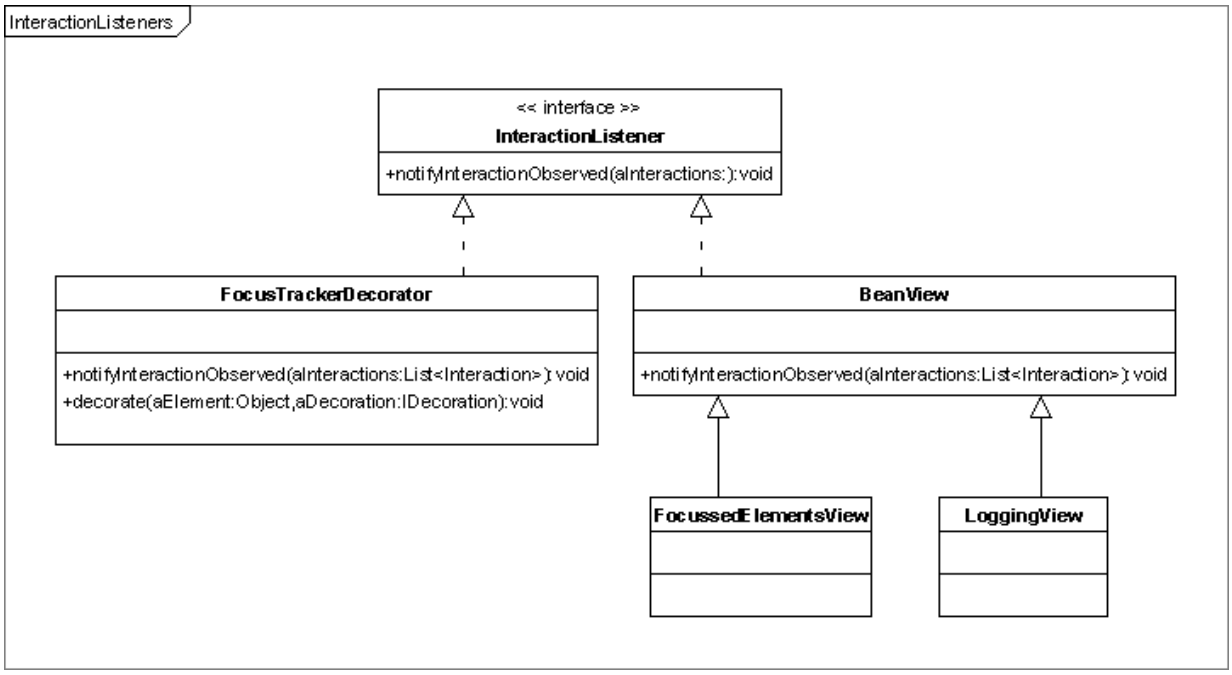

Abbildung 9: Klassenhierarchie der implementierten InteractionListener

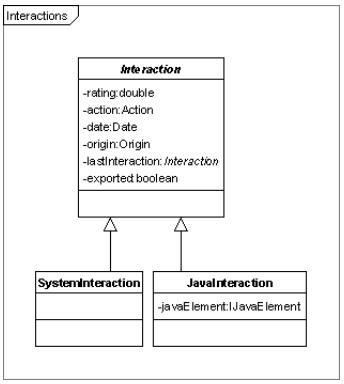

Abbildung 10: Klassenhierarchie der Interaktionen

3 Evaluation

Die Evaluation dient der Überprüfung der Funktionsweise und der Ergebnisse des FocusTracker nach der ersten finalen Version. Der Evaluation geht der Test der Funktion der einzelnen Komponenten und deren Zusammenspiel durch umfassende UnitTests voraus. UnitTests ersetzen jedoch keine fundierte empirische Evaluation im Sinne einer Validierung der Ergebnisse (zumindest sofern sie mit vertretbarem Aufwand implementiert werden sollen), da sie in einem relativ starren Umfeld arbeiten. Die Repräsentativität dieser Testumgebungen ist häufig nicht gegeben, zumal der Entwickler der UnitTests meist mit dem Entwickler der zu testenden Software identisch ist und somit die zu testenden Aspekte zuvor bekannt sind. Dabei muss auch hinsichtlich einer weiterreichenden Evaluation die Kritik geäußert werden, dass grundsätzlich niemals alle Faktoren der realen Welt abgebildet werden können. Die eingangs beschriebenen unterschiedlichen Stile der Entwickler, Quelltext zu modifizieren oder zu durchsuchen, sind hierfür nur ein Beispiel. Aus diesem Grund musste sich der FocusTracker im Zuge seiner „Evolution“ bereits in sehr frühen Entwicklungsstadien in realen Umgebungen beweisen. Nachdem ein Mindesmaß an Stabilität erreicht war, habe ich den FocusTracker daher in einem nächsten Schritt auch in meiner produktiven Entwicklungsumgebung aktiv eingesetzt, um bereits während der Entwicklung des FocusTracker Erkenntnisse über die Aussagekraft der Ergebnisse zu erhalten. Probesitzungen mit Videoaufzeichnungen wurden von Christopher Özbeck und Sebastian Jekutsch bereits in der Entwicklungsphase des FocusTracker vorgenommen, um schon in einem frühen Stadium den Aspekt unterschiedlicher Programmierstile zu würdigen und Fehler in der Auswertung der Ereignisse festzustellen.

3.1 Ziele

Die Evaluation dient dem Ziel, den FocusTracker hinsichtlich der hier erarbeiteten Anforderungen zu überprüfen. Sie offenbart inhaltliche Probleme, technische Schwächen oder zu ändernde Konfigurationen (vor allem hinsichtlich der Bewertung), die es in einem weiteren Schritt, der zweiten finalen Version des FocusTracker, zu beheben gilt. Der FocusTracker gewinnt durch eine gewissenhafte Evaluation an Stabilität und Aussagekraft.

3.2 Methodik

Eine wissenschaftlich fundierte Untersuchung wird im Rahmen dieser Arbeit nur schwer zu erreichen sein. Um dennoch verlässliche Aussagen über die Funktionsweise des FocusTracker zu erhalten, wird eine empirische Untersuchung durchgeführt, welcher nachfolgende methodische Ansätze zugrundeliegen:

- Beobachtung der Entwicklungstätigkeit einer Versuchsperson mit aktiviertem FocusTracker durch Videoaufzeichnung (1,5 Stunden) der Bildschirminhalte,
- Auswertung der Videoaufzeichnung über die chronologische Auflistung der tatsächlichen Aufmerksamkeitsfoki in zuvor definierter Form durch mich,
- eigene, automatisierte Auswertung der festgestellten (vom FocusTracker exportierten) Aufmerksamkeitsfoki anhand der Logging-Datei des ECG und
- Vergleich der vom FocusTracker exportierten Aufmerksamkeitsfoki mit den von mir tatsächlich festgestellten Aufmerksamkeitsfoki.

Vergleich der Aufmerksamkeitsfoki gemäß FocusTracker und Versuchsperson

Es wird deutlich, dass dem Vergleich der gemäß FocusTracker detektierten und der tatsächlichen Aufmerksamkeitsfoki der Versuchsperson eine zentrale Bedeutung in der Evaluation zukommt. Somit ist auch die Methodik der Auswertung von großer Bedeutung.

Gemäß der Definition des Aufmerksamkeitsfokus kann immer nur ein Element (einer Elementgruppe[66]) im Aufmerksamkeitsfokus des Betrachters liegen. Dieses Element wird durch mich für jeden Zeitraum anhand der Videoauszeichnung deklariert. Für diese Auswertung wird ihm (manuell) über die Länge des jeweils angegebenen Zeitraums ein numerischer Wert von 1,0 zugewiesen. Der FocusTracker hingegen arbeitet Fuzzy-Logik-basiert, so dass vom FocusTracker für jeden Zeitraum mehrere Elemente mit Unschärfen von 0,0 bis 1,0 als potenziell im Aufmerksamkeitsfokus befindlich ausgegeben werden. Der Vergleich der auf die jeweiligen Zeiträume bezogenen Werte je Quelle erfolgt nach Maßgabe des tatsächlichen Aufmerksamkeitsfokus der Versuchsperson: Diesem werden die vom FocusTracker zum gleichen Zeitpunkt detektierten Elemente gegenüber gestellt. Alle Elemente, die der FocusTracker im Vergleichszeitraum detektiert, werden mit einer Unschärfe von 0,0 belegt, es sei denn, es handelt sich um das im Aufmerksamkeitsfokus befindliche Element (für diesen Fall wird dem Element wie oben beschrieben der Wert 1,0 zugeordnet). Da der FocusTracker sensibel auf Aktivitäten des Entwicklers reagiert, besteht die Möglichkeit, dass der FocusTracker innerhalb eines Zeitraums, in dem sich ein Element tatsächlich im Aufmerksamkeitsfokus des Entwicklers befindet, für Teilzeiträume jeweils spezifische Werte für das Element erzeugt.[67] Für die Evaluation soll dies insofern berücksichtigt werden, als dass die spezifischen Werte innerhalb eines solchen Zeitraums über die Länge des Teilzeitraums gewichtet werden. Stellt der Entwickler also für einen Zeitraum von 60 Sekunden ein

[66] vgl. Kapitel 2.4.1 Abschnitt Elementgruppen

[67] oder im ungünstigen Fall für andere „falsche“ Elemente, die sich tatsächlich nicht im Aufmerksamkeitsfokus befinden

Element als im Aufmerksamkeitsfokus befindlich fest, wird diesem für diese 60 Sekunden der Wert 1,0 zugewiesen. Wenn der FocusTracker innerhalb dieses Zeitraums während der ersten 45 Sekunden einen Wert von 1,0 und für die übrigen 15 Sekunden einen Wert von 0,0 erzeugt, ergibt sich ein über den gesamten Zeitraum gewichteter Wert von 0,75.

Für jeden Zeitraum werden beide Werte gegenübergestellt. Über die Summe der (nochmals über den gesamten Zeitraum gewichteten) Abweichungen ergibt sich eine durchschnittliche Abweichung von der tatsächlichen Situation. Je geringer diese Abweichung ausfällt, desto besser arbeitet der FocusTracker. Ein Beispiel für diese Vorgehensweise ist in Tabelle 4 dargestellt.

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	FocusTracker	abs.	gew.
00:00:00	11	a b c	1,0 0,0 0,0	0,8 0,3 0,2	0,2 0,3 0,2	0,04 0,06 0,04
00:00:11	27	a b e	0,0 1,0 0,0	0,1 0,7 0,5	0,1 0,3 0,5	0,05 0,14 0,23
00:00:38	21	f	1,0	1,0	0,0	0,00
$\sum$	**59**					**0,56**

Tabelle 4: Beispiel für die Auswertung der Evaluation

Die Auswertung soll in mehreren Stufen erfolgen: In einer ersten Stufe sollen sämtliche vom FocusTracker detektierten potenziellen Aufmerksamkeitsfoki in die Auswertung einfließen. In weiteren Stufen werden die detektierten Aufmerksamkeitsfoki für die Auswertung gefiltert, d.h., dass nur Aufmerksamkeitsfoki mit Unschärfewerten oberhalb festgelegter Grenzen betrachtet werden. Diese Vorgehensweise entspricht dem Konzept der konfigurierbaren Exportschwellen[68] und liefert demnach wichtige Erkenntnisse für eine erste Justierung dieser.
Die Ergebnisse in Tabelle 5 verdeutlichen, dass für den vorliegenden (fiktiven) Fall der Schwellenwert 0,25 zu wählen ist, um die Abweichung von der tatsächlichen Situation zu reduzieren.

[68] vgl. Kapitel 2.5.2 RulesEngine

Auswertung Nr.	Schwellenwert, ab dem die FocusTracker-Ergebnisse in die Auswertung einfließen	Summe der gewichteten Differenzen
1	0,00	0,56
2	0,25	0,47

Tabelle 5: Auswirkungen unterschiedlicher Schwellenwerte anhand der fiktiven Werte aus Tabelle 4

3.3 Ergebnisse

Die Evaluation verdeutlicht, dass der FocusTracker von der realen Situation abweicht und abweichen muss, da nur der Entwickler selbst den Aufmerksamkeitsfokus eindeutig bestimmen kann. Dennoch wird deutlich, das der FocusTracker weit überwiegend mit der realen Situation übereinstimmt oder nur gering hiervon abweicht. Die geringe gewichtete durchschnittliche Abweichung, die je nach Art Schwellenwert zwischen 0,2226 und 0,2316 liegt, unterstreicht dies. Als Ergebnis der Evaluation soll als voreingestellter Schwellenwert für den ECG-Datenexport der Wert 0,25 gewählt werden, da dieser die Abweichungen reduziert. Des Weiteren wird eine große Bedeutung unterschiedlicher Progammierstile vermutet, welche hier jedoch nicht untersucht werden kann. Dies soll im Hinblick auf die Erweiterung des FocusTracker erneut aufgegriffen werden.

Auswertung Nr.	Schwellenwert, ab dem die FocusTracker-Ergebnisse in die Auswertung einfließen	Summe der gewichteten Differenzen
1	0,00	0,2316
2	0,25	0,2226

Tabelle 6: Auswirkungen unterschiedlicher Schwellenwerte anhand der real ermittelten Werte aus Tabelle 7

4 Fazit und Ausblick - Erweiterungsmöglichkeiten

Anhand der Art von Entwickleraktivitäten besteht ggf. die Möglichkeit, eine Sitzung einem bestimmten Profil zuzuordnen. Mögliche Profile könnten das Durchsuchen von Quelltext, das Bearbeiten bzw. Dokumentieren von Quelltext sowie das Debuggen sein. Jedem Profil könnte ein spezifischer Regelsatz zugewiesen werden, um eine auf die jeweilige Aktivität bezogene niedrigst mögliche Fehlerquote zu erreichen. Insbesondere während des Debuggens liegt der Aufmerksamkeitsfokus aufgrund der geführten Navigation durch den Programmablauf im Gegensatz zu anderen Profilen nicht mit der gleichen Sicherheit auf jedem (automatisch) selektierten Element. Analog hierzu besteht die Möglichkeit, den Stil des Entwicklers während einer Sitzung zu analysieren und die Regeln daraufhin anzupassen.

Eine weitere mögliche Modifikation betrifft die Detektion von Inaktivität. Die gegenwärtig starre Detektionsfunktion könnte durch einen dynamischen Ansatz ersetzt werden, welcher die Abstände zwischen Interaktionen erfasst. Sind diese auffallend groß, jedoch noch unterhalb der Inaktivitätsgrenze, so könnte ein dynamischer Ansatz die Grenze nach oben verschieben. Sehr kurze Inaktivitätsphasen deuten auch auf ein zu klein gewähltes Intervall hin. Stellt der dynamische Ansatz hingegen über längere Zeit viele Interaktionen mit kurzen Abständen fest, würde er die Grenze nach unten korrigieren.

Durch Studien und zukünftige Forschungserkenntnisse könnte sich die Notwendigkeit einer erweiterten Definition des Aufmerksamkeitsfokus hinsichtlich sich nicht im Sichtbereich befindlicher Elemente ergeben. Ein Beispiel hierfür könnte die Arbeit an einer Methode sein, für die aus einer anderen Methode Quelltext kopiert wird. Obgleich gegenwärtig der Aufmerksamkeitsfokus zwischenzeitlich für die Methode festgestellt wird, aus der kopiert wird, könnte bei einer erweiterten Definition davon ausgegangen werden, dass der Aufmerksamkeitsfokus permanent auf der Methode, in die kopiert wird, befindet – auch wenn sich diese kurzzeitig nicht im visuellen Fokus befand.

Nichtsdestotrotz stellt der hier im implementierte FocusTracker ein erstes weiterführendes Werkzeug hinsichtlich der Beobachtung und Auswertung des Mikroprozesses dar. Wünschenswert wäre, wenn er die Tür zu weiteren Forschungen öffnet.

Literatur

[Jek05] Sebastian Jekutsch. *Definition Mikroprozess*. 2005.

[KM05] Mik Kersten and Gail C. Murphy. *Mylar: a degree-of-interest model for IDEs*. 2005.

[RM05] Martin P. Robillard and Gail C. Murphy. *Program Navigation Analysis to Support Task-aware Software Development Environments*. 2005.

[Sch05] Frank Schlesinger. *Protokollierung von Programmiertätigkeiten in der Eclipse-Umgebung*. 2005.

A Regeldefinitionen und Schemata der ECG-Sensordaten

Regeldefinitionen

```
package de.fu_berlin.inf.focustracker.rating

import de.fu_berlin.inf.focustracker.rating.event.EditorSelectionEvent;
import de.fu_berlin.inf.focustracker.rating.event.ElementVisibiltyEvent;
import de.fu_berlin.inf.focustracker.rating.event.ElementFoldingEvent;
import de.fu_berlin.inf.focustracker.interaction.Action;
import de.fu_berlin.inf.focustracker.interaction.Origin;

rule "JavaEditor Selection Changed"
  no-loop true
  when
    event : EditorSelectionEvent( action == Action.SELECTED) ||
      EditorSelectionEvent( action == Action.SELECTION_SAME_ELEMENT)
  then
    double p = 0;
    if(event.getElementRegion().isFillingCompleteView()) {
      p = 1;
    } else {
      double percentageVisible = event.getElementRegion().
          getPercentageVisible();
          if(percentageVisible >= 0.5) {
            p = 1;
          } else if (percentageVisible >= 0.25) {
              p = 0.50;
          } else {
              p = 0.25;
          }
      }
    event.setRating(p);
    modify( event );

end

rule "JavaEditor Visibility Changed"
  no-loop true
  when
    event : ElementVisibiltyEvent( action == Action.VISIBILITY_GAINED ) ||
      ElementVisibiltyEvent( action == Action.VISIBILITY_LOST ) ||
      ElementVisibiltyEvent( action == Action.VISIBILITY_CHANGED )
  then

      double p = 0;
      if(event.isVisible()) {
        if(event.getElementRegion().isFillingCompleteView()
          || event.getNumberOfElementsVisible() == 1
          || (event.getLastInteraction() != null &&
            event.getLastInteraction().getRating() == 1)) {
```

```
        p = 1;
        } else {
        double percentageVisible = event.getElementRegion().
            getPercentageOfView();
            if(percentageVisible >= 0.9) {
              p = 1;
            } else if (percentageVisible >= 0.5) {
                p = 0.5;
            } else if (percentageVisible >= 0.25) {
                p = 0.25;
            } else {
                p = 0.1;
            }
          }
      }
    event.setRating(p);
    modify( event );

end

rule "JavaEditor Folding Expanded"
  no-loop true
  when
    event : ElementFoldingEvent( action == Action.FOLDING_EXPANDED,
      origin == Origin.JAVAEDITOR,
      javaElementCurrentRating < 0.5)
  then
    System.out.println("JavaEditor Folding Expanded");
    event.setRating(0.5);
    modify( event );

end

rule "JavaEditor Folding Collapsed"
  no-loop true
  when
    event : ElementFoldingEvent( action == Action.FOLDING_COLLAPSED,
      origin == Origin.JAVAEDITOR )
  then
    event.setRating(0.1);
    modify( event );

end

rule "PackageExplorer Folding Expanded"
  no-loop true
  when
    event : ElementFoldingEvent( action == Action.FOLDING_EXPANDED,
      origin == Origin.PACKAGE_EXPLORER,
      javaElementCurrentRating < 0.5 )
  then
```

```
    System.out.println("PackageExplorer Folding Expanded");
    event.setRating(0.5);
    modify( event );

end

rule "PackageExplorer Folding Collapsed"
  no-loop true
  when
    event : ElementFoldingEvent( action == Action.FOLDING_COLLAPSED,
      origin == Origin.PACKAGE_EXPLORER )
  then
    event.setRating(0.1);
    modify( event );

end
```

Schemata der ECG-Sensordaten

msdt.focus.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:annotation>
    <xs:documentation>
      This XML schema defines the structure of the "Focus"
      MicroSensorDataType.
    </xs:documentation>
  </xs:annotation>

  <xs:include schemaLocation="msdt.common.xsd"/>

  <xs:element name="microActivity">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="commonData" type="commonDataType"/>
        <xs:element name="focus">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="resourcename" type="xs:token"/>
              <xs:element name="element" type="xs:token"/>
              <xs:element name="elementtype" type="xs:token"/>
              <xs:element name="hasfocus" type="xs:boolean"/>
              <xs:element name="rating" type="xs:double"
                minOccurs="0"/>
              <xs:element name="detectedtimestamp"
                type="xs:dateTime"/>
              <xs:element name="comment" type="xs:token"
                minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

msdt.user.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:annotation>
    <xs:documentation>
      This XML schema defines the structure of the "User" MSDT.
    </xs:documentation>
  </xs:annotation>

  <xs:include schemaLocation="msdt.common.xsd"/>

  <xs:element name="microActivity">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="commonData" type="commonDataType"/>
        <xs:element name="user">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="activity">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="user_active"/>
                    <xs:enumeration value="user_inactive"/>
                    <xs:enumeration value="copy"/>
                    <xs:enumeration value="cut"/>
                    <xs:enumeration value="paste"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="param1" type="xs:string"
                minOccurs="0"/>
              <xs:element name="param2" type="xs:string"
                minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

B Protokolle und Auswertungen der Sitzungen

Die Videoaufzeichnung sowie die ECG-Log-Datei und Tabelle der Deklaration der Aufmerksamkeitsfoki auf denen die folgende Auswertung basiert können unter http://www.focustracker.org/thesis/ bezogen werden.

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
21:07:27	68	compute	1,0	0,1109	0,8891	0,0122
21:08:35	9	print_script	1,0	0,6875	0,3125	0,0015
		print_hunk	0,0	0,1250	0,1250	0,0000
		compute	0,0	0,0417	0,0417	0,0000
21:08:44	21	print_hunk	1,0	0,7951	0,2049	0,0025
		analyze_hunk	0,0	0,0164	0,0164	0,0000
		print_script	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:09:05	56	analyze_hunk	1,0	0,7161	0,2839	0,0034
		print_number_range	0,0	0,0085	0,0085	0,0000
21:10:01	3	compute	1,0	1,0000	0,0000	0,0000
		analyze_hunk	0,0	0,0000	0,0000	0,0000
21:10:04	2	analyze_hunk	1,0	1,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:10:06	15	print_hunk	1,0	0,8333	0,1667	0,0015
		analyze_hunk	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:10:21	10	compute	1,0	0,5909	0,4091	0,0009
		analyse	0,0	0,1818	0,1818	0,0000
		print_hunk	0,0	0,0000	0,0000	0,0000
21:10:31	29	analyse	1,0	0,7000	0,3000	0,0021
		compute	0,0	0,0000	0,0000	0,0000
21:11:00	25	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0400	0,0400	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
21:11:25	5	compute	1,0	0,0000	1,0000	0,0010
21:11:30	34	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
21:12:04	21	compute	1,0	0,0476	0,9524	0,0040
21:12:25	185	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,1945	0,1945	0,0000
21:15:30	6	compute	1,0	0,8333	0,1667	0,0002
21:15:36	6	LineChangeIterator	1,0	0,5909	0,4091	0,0009
		compute	0,0	0,0455	0,0455	0,0000
21:15:42	23	compute	1,0	0,6962	0,3038	0,0016
		LineChangeIterator	0,0	0,0000	0,0000	0,0000
21:16:05	2	analyse	1,0	0,8333	0,1667	0,0001
		compute	0,0	0,4000	0,4000	0,0000
21:16:07	4	compute	1,0	0,7083	0,2917	0,0007
		LineChangeIterator	0,0	0,0417	0,0417	0,0000
		analyse	0,0	0,0417	0,0417	0,0000
21:16:11	5	LineChangeIterator	1,0	0,7000	0,3000	0,0009
		compute	0,0	0,1000	0,1000	0,0000
		analyse	0,0	0,0667	0,0667	0,0000
21:16:16	149	analyse	1,0	0,9205	0,0795	0,0031
		getCode	0,0	0,0051	0,0051	0,0000
		initialize	0,0	0,0051	0,0051	0,0000
		propertyChanged	0,0	0,0051	0,0051	0,0000
		getLines	0,0	0,0051	0,0051	0,0000
		update	0,0	0,0051	0,0051	0,0000
		getHashKey	0,0	0,0282	0,0282	0,0000
		compute	0,0	0,0000	0,0000	0,0000
		LineChangeIterator	0,0	0,0000	0,0000	0,0000
21:18:45	6	getHashKey	1,0	0,6714	0,3286	0,0005
		analyse	0,0	0,0357	0,0357	0,0000
21:18:51	66	analyse	1,0	1,0000	0,0000	0,0000
		compute	0,0	0,0152	0,0152	0,0000
		getHashKey	0,0	0,0000	0,0000	0,0000
21:19:57	11	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:20:08	22	compute	1,0	0,5409	0,4591	0,0020
21:20:30	15	analyse	1,0	1,0000	0,0000	0,0000

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
		compute	0,0	0,0185	0,0185	0,0000
21:20:45	34	compute	1,0	0,8500	0,1500	0,0012
		analyse	0,0	0,0750	0,0750	0,0000
21:21:19	25	analyse	1,0	1,0000	0,0000	0,0000
		compute	0,0	0,0370	0,0370	0,0000
21:21:44	144	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:24:08	172	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:27:00	238	compute	1,0	0,7234	0,2766	0,0133
21:30:58	14	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:31:12	50	compute	1,0	0,0372	0,9628	0,0097
21:32:02	252	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,8746	0,8746	0,0000
21:36:14	56	compute	1,0	0,1571	0,8429	0,0095
21:37:10	20	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
21:37:30	95	compute	1,0	0,3053	0,6947	0,0133
		LineChangeIterator	0,0	0,0105	0,0105	0,0000
21:39:05	33	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,9855	0,9855	0,0000
21:39:38	41	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	1,0000	1,0000	0,0000
21:40:19	20	analyse	1,0	0,7542	0,2458	0,0030
		compute	0,0	0,0000	0,0000	0,0000
		computeNewNeighborhood	0,0	0,0833	0,0833	0,0000
21:40:39	64	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0833	0,0833	0,0000
		compute	0,0	0,0000	0,0000	0,0000
		computeNewNeighborhood	0,0	0,0833	0,0833	0,0000
21:41:43	3	analyse	1,0	0,7478	0,2522	0,0005
		compute	0,0	0,0000	0,0000	0,0000
		computeNewNeighborhood	0,0	0,0833	0,0833	0,0000

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
21:41:46	132	compute	1,0	0,3303	0,6697	0,0185
		initialize	0,0	0,0018	0,0018	0,0000
		propertyChanged	0,0	0,0201	0,0201	0,0000
		size	0,0	0,0073	0,0073	0,0000
		getLines	0,0	0,0018	0,0018	0,0000
		getHashKey	0,0	0,0000	0,0000	0,0000
		LineChangeIterator	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,1204	0,1204	0,0000
21:43:58	10	<kein Element>	0,0	0,0000	0,0000	0,0000
		propertyChanged	0,0	0,5000	0,5000	0,0000
21:44:08	32	analyse	1,0	0,8394	0,1606	0,0029
		initialize	0,0	0,0000	0,0000	0,0000
		propertyChanged	0,0	0,0000	0,0000	0,0000
21:44:40	64	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	1,0000	1,0000	0,0000
21:45:44	51	analyse	1,0	0,8275	0,1725	0,0018
21:46:35	72	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,7800	0,7800	0,0000
21:47:47	16	analyse	1,0	0,3700	0,6300	0,0020
21:48:03	18	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,3700	0,3700	0,0000
21:48:21	15	analyse	1,0	0,3700	0,6300	0,0019
21:48:36	71	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,3648	0,3648	0,0000
21:49:47	9	analyse	1,0	0,0000	1,0000	0,0018
21:49:56	7	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
21:50:03	3	analyse	1,0	0,0000	1,0000	0,0006
21:50:06	109	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
21:51:55	28	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
21:52:23	75	analyse	1,0	0,0000	1,0000	0,0151
21:53:38	6	<kein Element>	0,0	0,0000	0,0000	0,0000

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
		analyse	0,0	0,0000	0,0000	0,0000
21:53:44	4	analyse	1,0	1,0000	0,0000	0,0000
21:53:48	245	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	1,0000	1,0000	0,0000
21:57:53	31	analyse	1,0	1,0000	0,0000	0,0000
21:58:24	44	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
21:59:08	77	print_hunk	1,0	0,9429	0,0571	0,0009
		analyse	0,0	0,0000	0,0000	0,0000
22:00:25	43	getHashKey	1,0	0,8360	0,1640	0,0041
		getLines	0,0	0,0820	0,0820	0,0000
		print_hunk	0,0	0,0000	0,0000	0,0000
22:01:08	24	<kein Element>	0,0	0,0000	0,0000	0,0000
		getLines	0,0	0,0833	0,0833	0,0000
		getHashKey	0,0	0,1667	0,1667	0,0000
		print_hunk	0,0	0,0000	0,0000	0,0000
22:01:32	6	getHashKey	1,0	0,7778	0,2222	0,0008
		analyse	0,0	0,0556	0,0556	0,0000
		getLines	0,0	0,0417	0,0417	0,0000
		print_hunk	0,0	0,0000	0,0000	0,0000
22:01:38	76	print_hunk	1,0	0,5343	0,4657	0,0077
		print_number_range	0,0	0,0030	0,0030	0,0000
		print_header	0,0	0,0030	0,0030	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
		getHashKey	0,0	0,0000	0,0000	0,0000
		getLines	0,0	0,0000	0,0000	0,0000
22:02:54	3	analyse	1,0	0,7500	0,2500	0,0006
		print_hunk	0,0	0,0000	0,0000	0,0000
		print_number_range	0,0	0,0625	0,0625	0,0000
		print_header	0,0	0,0625	0,0625	0,0000
22:02:57	39	compute	1,0	0,4151	0,5849	0,0063
		print_hunk	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
		print_number_range	0,0	0,0189	0,0189	0,0000

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
		print_header	0,0	0,0189	0,0189	0,0000
22:03:36	12	print_hunk	1,0	0,7609	0,2391	0,0011
		compute	0,0	0,0000	0,0000	0,0000
22:03:48	8	compute	1,0	0,6667	0,3333	0,0016
		print_hunk	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,0833	0,0833	0,0000
22:03:56	54	print_hunk	1,0	0,6667	0,3333	0,0109
		analyse	0,0	0,0015	0,0015	0,0000
		compute	0,0	0,0000	0,0000	0,0000
22:04:50	6	analyse	1,0	0,8125	0,1875	0,0005
		print_hunk	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,2500	0,2500	0,0000
22:04:56	17	print_hunk	1,0	0,8438	0,1563	0,0010
		compute	0,0	0,0313	0,0313	0,0000
22:05:13	82	analyze_hunk	1,0	0,9512	0,0488	0,0008
22:06:35	112	compute	1,0	0,9634	0,0366	0,0009
		analyse	0,0	0,0186	0,0186	0,0000
		analyze_hunk	0,0	0,0083	0,0083	0,0000
22:08:27	19	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,8195	0,8195	0,0000
22:08:46	18	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,8700	0,8700	0,0000
22:09:04	122	compute	1,0	0,9444	0,0556	0,0014
22:11:06	45	analyse	1,0	0,0000	1,0000	0,0000
		compute	0,0	1,0000	1,0000	0,0000
22:11:51	35	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	1,0000	1,0000	0,0000
22:12:26	15	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,3182	0,3182	0,0000
		compute	0,0	0,3636	0,3636	0,0000
22:12:41	24	analyse	1,0	1,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
22:13:05	63	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000

Beginn Zeitraum	**Dauer Zeitraum (in s)**	**Element**	**Aufmerksamkeitsfokus (gewichtet nach relativer Dauer) gemäß**		**Differenz der Unschärfen**	
			Deklaration	Focus-Tracker	abs.	gew.
		analyse	0,0	0,5000	0,5000	0,0000
22:14:08	121	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,5000	0,5000	0,0000
22:16:09	4	analyse	1,0	0,5950	0,4050	0,0007
		compute	0,0	0,0000	0,0000	0,0000
22:16:13	5	analyze_hunk	1,0	1,0000	0,0000	0,0000
		compute	0,0	0,2500	0,2500	0,0000
		analyse	0,0	0,0000	0,0000	0,0000
22:16:18	14	compute	1,0	1,0000	0,0000	0,0000
22:16:32	94	<kein Element>	0,0	0,0000	0,0000	0,0000
		compute	0,0	1,0000	1,0000	0,0000
22:18:06	2	compute	1,0	1,0000	0,0000	0,0000
22:18:08	41	analyse	1,0	0,9917	0,0083	0,0002
		computeNewNeighborhood	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
22:18:49	95	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	0,1667	0,1667	0,0000
		computeNewNeighborhood	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
22:20:24	14	analyse	1,0	0,8363	0,1637	0,0008
		computeNewNeighborhood	0,0	0,0000	0,0000	0,0000
		compute	0,0	0,0000	0,0000	0,0000
22:20:38	199	<kein Element>	0,0	0,0000	0,0000	0,0000
		analyse	0,0	1,0000	1,0000	0,0000
22:23:57	365	analyse	1,0	0,5641	0,4359	0,0569
$\sum$	**4955**					**0,2316**

Tabelle 7: Auswertung der Sitzung

C Installation und Konfiguration

Konfiguration

Parameter	Beschreibung	Standard
ecgExportInterval	Abstand zwischen zwei ECG-Exporten (in s)	10
ecgExportMinRatingForApperance	Minimale Bewertung eines Elements um exportiert zu werden	0,25
ecgExportMinRatingForDisapperance	Minimale Bewertung eines bereits exportierten Elements um als „nicht bewertet“ exportiert zu werden	0,1
userInactivityDetectionTimeout	Zeit ohne Ereignisse ab der Inaktivität festgestellt wird (in s)	60
enableJavaEditorMouseMoveListener	Betrachtung von Mausbewegungen über Java-Elementen im Editor	true
enableFocusTrackerDecorator	De- bzw. aktiviert den FocusTrackerDecorator	true

Tabelle 8: Konfigurationsparameter

Die Konfiguration des FocusTracker ist über das Menü „Window/Preferences...“ zu erreichen. Nach Auswahl der Option „FocusTracker“ im Konfigurationsfenster erscheint die Konfigurationsseite (siehe Abbildung 11). Der FocusTrackerDecorator kann über den Verweis „Label Decorations“ de- oder aktiviert werden.

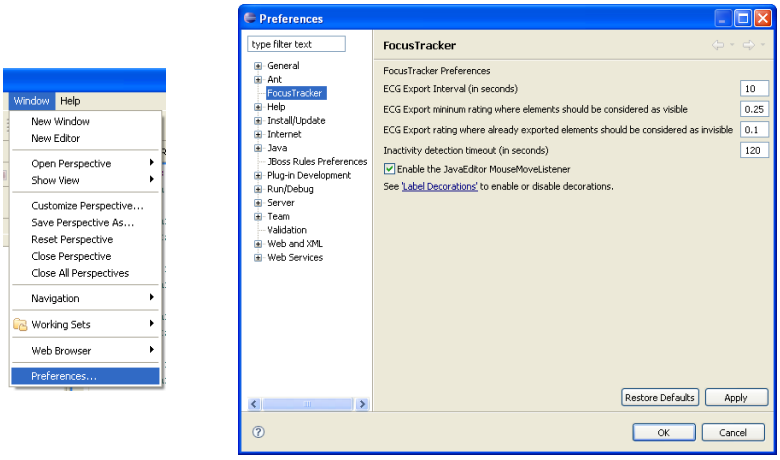

Abbildung 11: Öffnen des Konfigurationsfensters

Installation

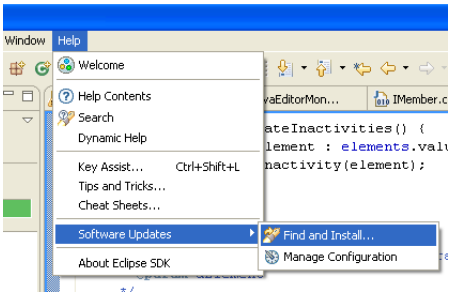

Abbildung 12: Starten der Installation

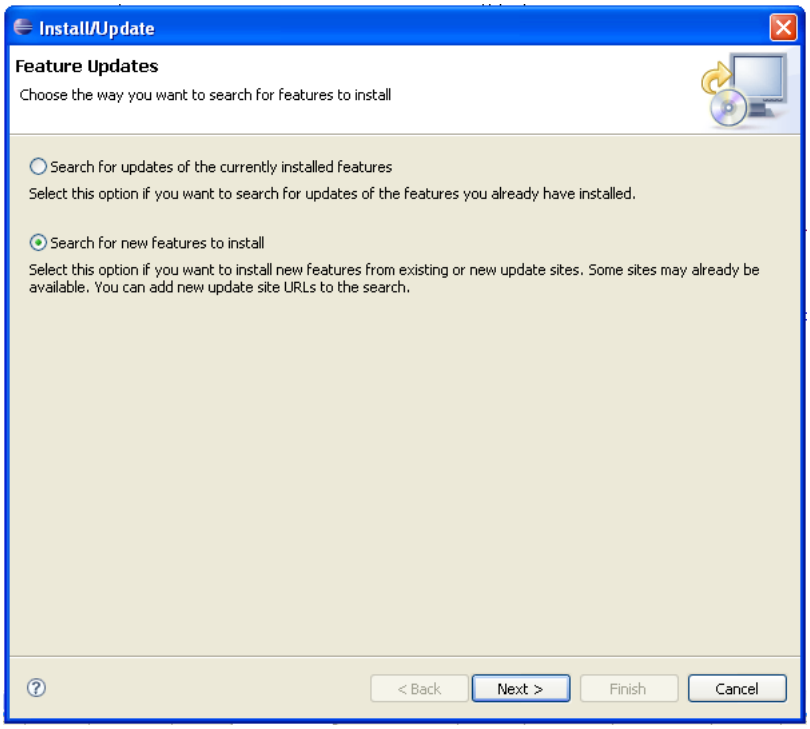

Abbildung 13: Neue Features installieren

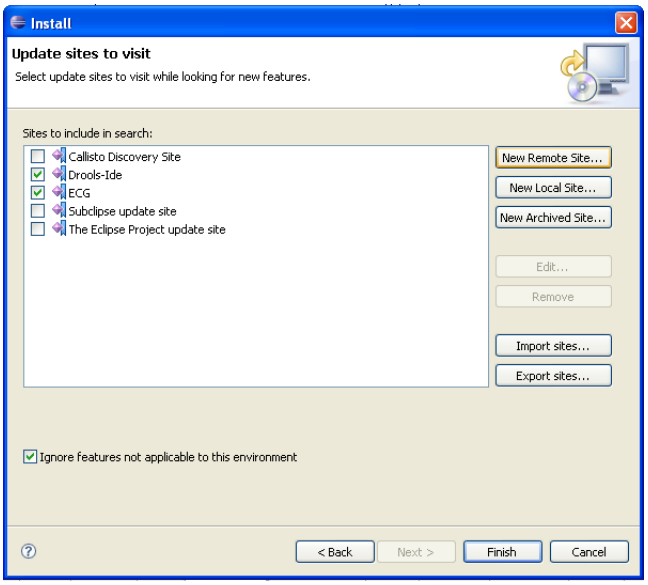

Abbildung 14: Ansicht der verfügbaren Update-Seiten

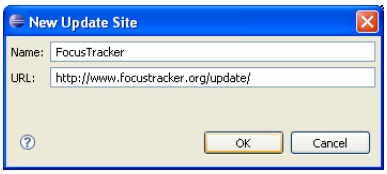

Abbildung 15: Update-Seite hinzufügen

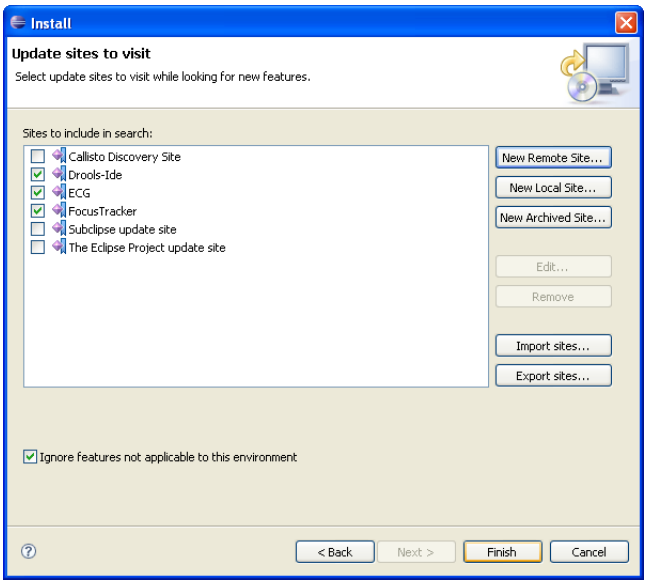

Abbildung 16: Auswahl der Update-Seite

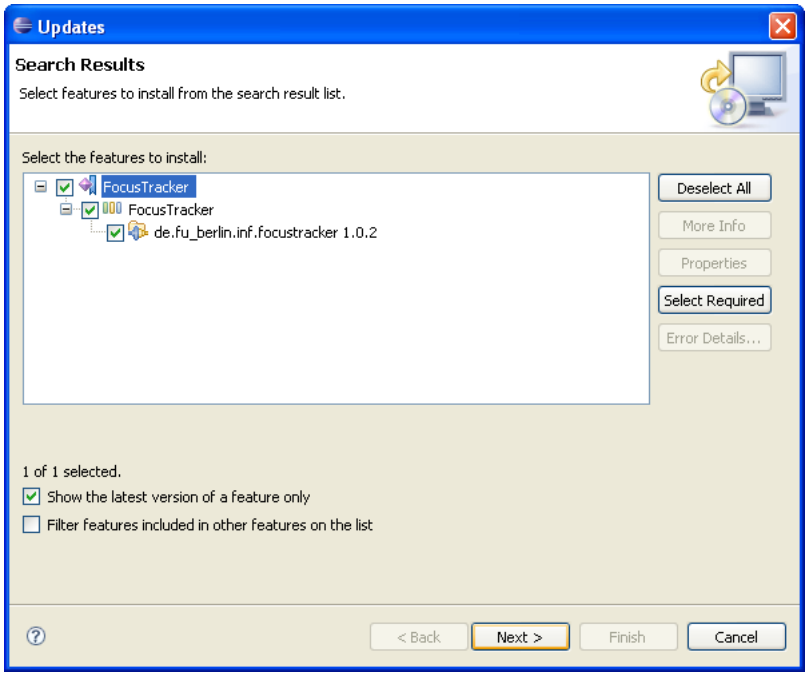

Abbildung 17: Auswahl des FocusTracker

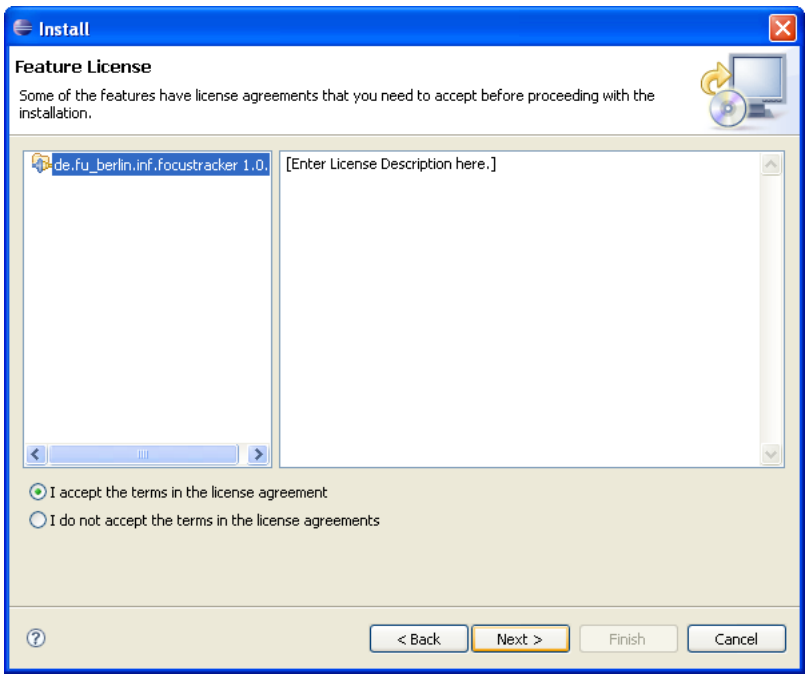

Abbildung 18: Akzeptieren der Lizenz

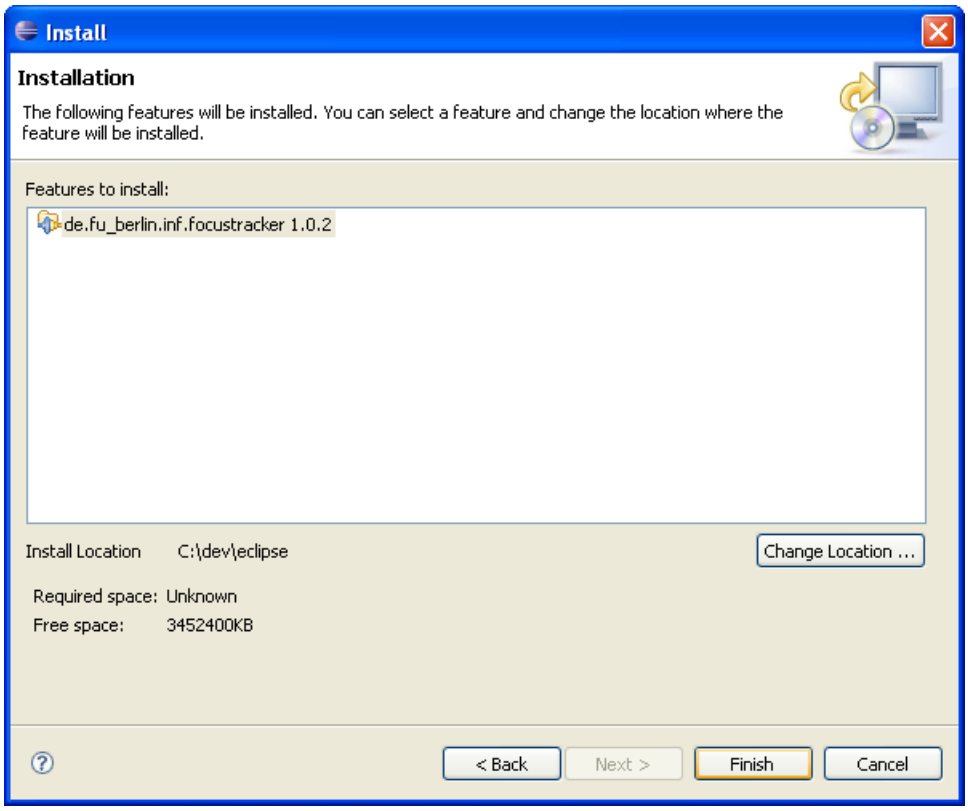

Abbildung 19: Auswahl bestätigen

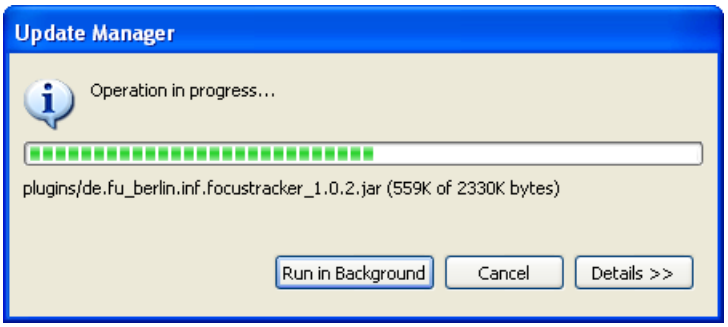

Abbildung 20: Plugin wird übertragen

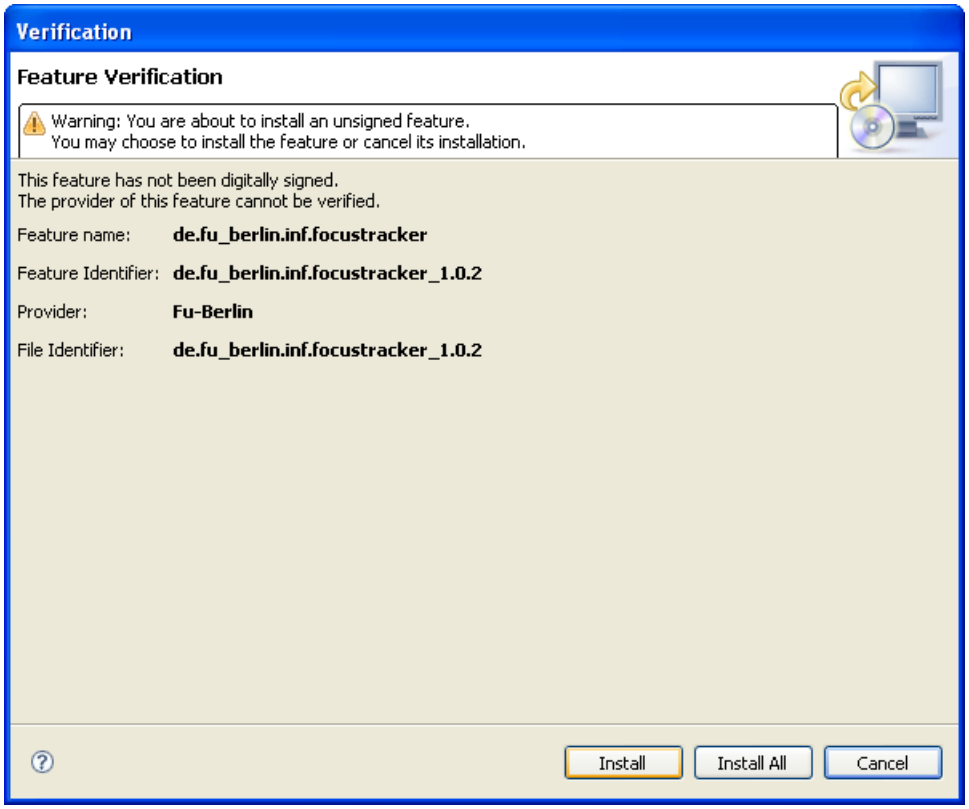

Abbildung 21: Installation des Plugins bestätigen

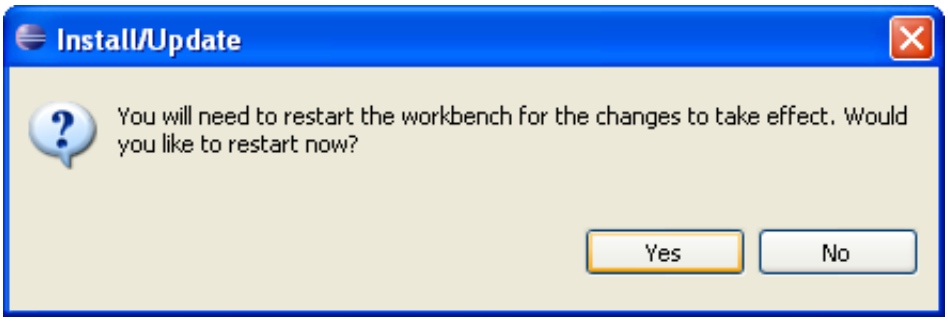

Abbildung 22: Neustarten der Entwicklungsumgebung