Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Benjamin Lindemann

Realisierung eines Frameworks zur netzwerkgestützten Erkennung von potentiellen Ablenkungsfaktoren

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Benjamin Lindemann

Realisierung eines Frameworks zur netzwerkgestützten Erkennung von potentiellen Ablenkungsfaktoren

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Gunter Klemke

Eingereicht am: 07. August 2013

Benjamin Lindemann

Thema der Arbeit
Realisierung eines Frameworks zur netzwerkgestützten Erkennung von potentiellen Ablenkungsfaktoren

Stichworte
Ablenkung, Wissensarbeit, Kontext, verteiltes System, Middleware, Framework, ActiveMQ

Kurzzusammenfassung
Wissensarbeiter haben einen enormen Konzentrationsbedarf. Jede Ablenkung kann zu zeitlichen Verzögerungen in ihrem Arbeitsablauf führen. Eine mögliche Folge ist Stress, der auf Dauer krank machen kann. Ablenkung sollte somit auf jeden Fall vermieden werden.

In dieser Arbeit wird ein Framework zur netzwerkgestützten Erkennung von potentiellen Ablenkungsfaktoren entwickelt. Das Framework wurde auf Basis der Architektur von verteilten Systemen aufgebaut. Das entwickelte Framework bietet Schnittstellen zur Integration von Plug-ins an. Zur Realisierung dieser Schnittstellen werden Technologie-Empfehlungen ausgesprochen.

Das entwickelte Framework bietet eine gute Grundlage, um Systeme zur Erkennung von potentiellen Ablenkungsfaktoren aufzubauen.

Title of the paper
Realization of a framework for network-based detection of potential distraction factors

Keywords
distraction, knowledge worker, context, distributed system, middleware, framework, ActiveMQ

Abstract
Knowledge workers have a huge requirement in concentration. Any distraction can lead to delays in their workflow. A possible consequence is stress which could make sick in the long run. Distraction should therefore be avoided in any case.

In this work, a framework for network-based detection of potential distraction factors is developed. The framework was built upon the basis of the architecture of distributed systems. The developed framework provides interfaces for the integration of plug-ins. For the realization of these interfaces technology recommendations are pronounced.

The developed framework offers a good fundament to setup systems for the detection of potential distraction factors.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

1.1 Motivation

Unsere Gesellschaft entwickelt sich immer weiter von einer Industriegesellschaft hin zu einer Wissensgesellschaft (Heidenreich 2003). Dabei ist jeder Mitarbeiter, der in einem Unternehmen Wissen verarbeitet, ein Wissensarbeiter. Wissensarbeiter haben einen erhöhten Konzentrationsbedarf, da sie ständig mit enormen Mengen an Informationen umgehen müssen. Die Informationen des aktuellen Aufgabenkontextes werden dabei im Kurzzeitgedächtnis gespeichert. Jede Unterbrechung kann zum Verlust der Informationen im Kurzzeitgedächtnis führen. Die Folge: Der Wissensarbeiter muss sich erneut in die Aufgabe hineindenken.[1]

Einer Studie aus dem Jahr 2005 zufolge verliert der Wissensarbeiter 28% seiner täglichen Arbeitszeit aufgrund von Unterbrechungen (Spira und Feintuch 2005). Neben den wirtschaftlichen Folgen dieses Arbeitszeitverlustes können gesundheitliche Probleme auftreten, wie beispielsweise dauerhafter Stress durch ständigen Termindruck.[2] Ablenkungen stellen somit ein großes Problem in der Wissensarbeit dar. Neue Lösungswege zu finden, um Ablenkung am Arbeitsplatz automatisch zuerkennen und den Wissensarbeiter gezielt davor zuschützen, stellt die Motivation für meine Forschung dar.[3]

Die verlässliche Erkennung von Ablenkungen im Bereich der Wissensarbeit ist bisher noch nicht vollständig realisiert. Einzelne Projekte konnten Teillösungen für konkrete Laborumgebungen entwickeln und hier erfolgreich körperliche Reaktionen auf Ablenkung und Stress messen (siehe Kapitel 2.4). Die direkte Erkennung von Ablenkungsfaktoren stand dabei jedoch nicht im Vordergrund.

In dieser Arbeit wird ein Framework entwickelt, mit dem ein Sensornetzwerk zur Erkennung von potentiellen Ablenkungsfaktoren aufgebaut werden kann. Die Architektur des Frameworks muss dabei so flexibel aufgebaut sein, dass unterschiedliche Sensoren sowie Aktoren leicht anbindbar sind. Die Architektur des Frameworks muss dabei unabhängig von spezifischen Pro-

[1] Für weitere Informationen zum Thema „Wissensarbeit“ sei beispielsweise auf (Göddel 2008) verwiesen.
[2] Siehe hierzu auch (Lohmann-Haislah 2012).
[3] Andere Forschungsgruppen beschäftigten sich mit der Optimierung von Management-Strategien zur Verbesserung der Arbeitsbedingungen von Wissensarbeitern (siehe hierzu (Davenport, Jarvenpaa und Beers 1996)).

grammiersprachen oder Betriebssystemen sein, damit die verschiedenen Systemanforderungen der unterschiedlichen Sensoren bewältigt werden können. Die Kommunikation zwischen den Komponenten soll so ausgelegt sein, dass es allen Komponenten ermöglicht wird gemeinsam kollaborativ zuarbeiten.

Das Ziel der vorliegenden Arbeit ist es, dass das Framework als Grundlage für weitere Forschungsarbeiten verwendet werden kann und so einen Beitrag zur Optimierung der Arbeitsbedingungen von Wissensarbeitern beiträgt.

1.2 Aufbau dieser Arbeit

Diese Arbeit ist in fünf Kapitel unterteilt. Im Anschluss an dieses einleitende Kapitel folgt im Kapitel 2 zunächst die Vorstellung von drei Szenarien, die die Einsatzmöglichkeiten des zu entwickelnden Frameworks und die Probleme, die dabei auftreten können, darstellen. Aus diesen Szenarien ergeben sich einige Begriffe, die näher erläutert und deren Bezüge zu dieser Arbeit deutlich gemacht werden sollen. Um die Anforderungen und Ziele, die sich aus diesen Szenarien an das Framework ergeben, aufzustellen und von bereits vorhandenen Lösungen abzugrenzen, folgt die Vorstellung von verschiedenen Forschungsarbeiten. Das Kapitel schließt mit der Aufstellung der Anforderungen und Ziele an das zu entwickelnde Framework sowie der Ableitung der für ein konkretes Problem benötigten Komponenten.

Aus den Anforderungen und Zielen kann nun die Architektur des Frameworks aufgebaut werden. Die Grundlagen hierfür sind im Kapitel 3 ausführlich dargestellt. Hier folgen ebenfalls die Erläuterungen der verwendeten Technologien, die zur Realisierung des Frameworks notwendig sind. Die Spezifizierung der Kommunikationsschnittstelle und der Persistenzebene schließen hieran an. Die Erstellung der Architektur und eine erste Übersicht über diese leiten zum nächsten Kapitel über.

Das Kapitel 4 beschreibt die Umsetzung der vorangegangenen Designentscheidungen. Dabei werden spezifische Technologien für die Realisierung der Infrastruktur eingeführt und festgelegt. Die funktionale Übersicht des Frameworks stellt die verschiedenen Komponenten und ihre Kommunikationskanäle dar. Es folgen Beschreibungen der Kernkomponente „zentrale Steuerungseinheit“, der möglichen Plug-ins und der Kommunikationsschnittstellen.

Die Evaluierung des Konzepts wird im Kapitel 4.5 durchgeführt. Am Ende der vorliegenden Arbeit erfolgt im Kapitel 5 eine Zusammenfassung des Inhalts und eine Bewertung der Ergebnisse. Der Ausblick beinhaltet verschiedene Ansatzpunkte, um das Framework und die Intention dieser Arbeit weiter auszubauen.

2 Analyse

Laut Duden steht der Begriff „Ablenkung“ für eine „Richtungsänderung“, „Abwechslung“ oder eine „Zerstreuung“.[4] „Ablenkung“ ist also etwas, dass eine Veränderung mit sich bringt und damit eine Überleitung von einem Zustand in einen anderen darstellt. Spricht man beim Menschen von „Ablenkung“, so ist häufig die Veränderung der Konzentration gemeint. Man richtet dabei seinen Fokus von einer Aufgabe auf eine andere. Dabei vergisst man sehr schnell den Kontext der vorherigen Aufgabe beziehungsweise lässt diesen außer Acht.

Ablenkungen können schwerwiegende Folgen haben.[5] Unfälle im Straßenverkehr entstehen beispielsweise häufig durch abgelenkte Fahrer. So ziehen zum Beispiel die elektronischen Geräte des Fahrzeugs, wie das Navigationsgerät, die Klimaanlage, das Radio und diverse Anzeigen, die Aufmerksamkeit des Fahrers auf sich. Neben diesen im Fahrzeug verbauten elektronischen Geräten gibt es auch noch die tragbaren Geräte, wie Musikplayer und Mobiltelefone, die ebenfalls vom Straßenverkehr ablenken können. Auch eine Unterhaltung mit anderen Fahrgästen stellt eine „Ablenkungsursache“ dar. (Kutila u. a. 2007)

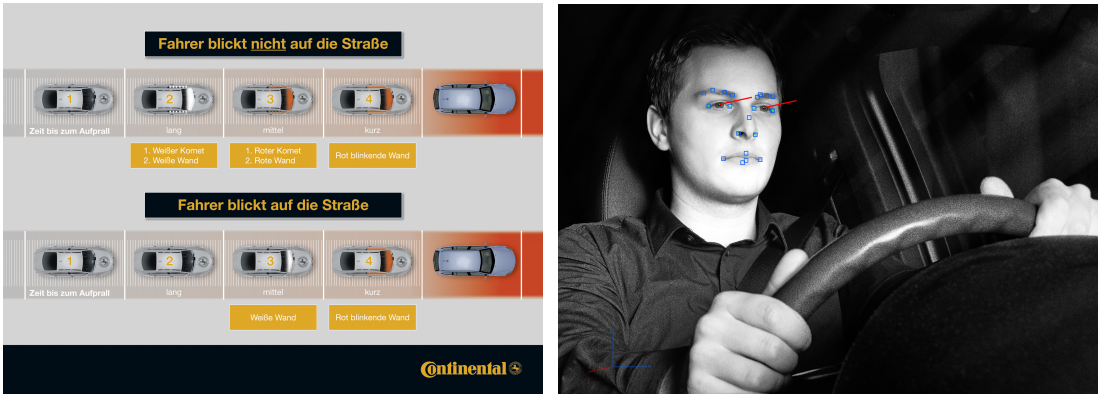
Zusätzlich zu diesen äußeren Einflüssen können auch innere Einflüsse die Aufmerksamkeit des Autofahrers beeinträchtigen. Tagträumerei oder stark konzentriertes Nachdenken können die Konzentration ebenfalls vom Straßenverkehr ablenken. Hierbei handelt es sich um kognitive Ablenkung. (Kutila u. a. 2007)

Um den Menschen vor den Folgen von „Ablenkung“ beim Steuern von Fahrzeugen zu schützen (hier: vor einem Verkehrsunfall), sollte das Fahrzeug die Ablenkung des Menschen erkennen und darauf reagieren können. Das Fahrzeug der Zukunft könnte den Fahrer zum Beispiel darin unterstützen, seine Aufmerksamkeit wieder auf die Straße zu lenken. Erste Umsetzungen eines solchen Systems sind in dem Projekt „PRORETA 3“[6] (Abbildung 2.1) entwickelt worden (Lotz 2013). Dies zeigt auch, wie wichtig es ist, den Anwender bei seiner Konzentration zu unterstützen.

[4] siehe http://www.duden.de/node/726323/revisions/1282921/view, zuletzt besucht am 19.07.2013

[5] Kurzzeitige Ablenkung wiederum kann die Konzentration sogar steigern. (Ariga und Lleras 2011)

[6] siehe auch http://www.proreta.tu-darmstadt.de/proreta_1/startseite_proreta/index.de.jsp, zuletzt besucht am 17.07.2013



a: Warnstufen des LED-Warnsystems mit und ohne Ablenkung (© Continental[7])

b: Aufnahme mit Infrarotkamera zur Erkennung der Blickrichtung des Fahrers (© Continental[8])

Abbildung 2.1: LED-Warnsystem von Continental

Ein anderer Bereich, in dem Ablenkung negative Folgen haben kann, ist die Wissensarbeit. Ablenkung entsteht in diesem Bereich häufig durch eingehende E-Mails, Anrufe oder Kollegen. So zeigt zum Beispiel das E-Mail-Programm eine Benachrichtigung auf dem Desktop des Benutzers an, sobald eine neue E-Mail verfügbar ist. Da die Wissensarbeit ein hohes Maß an Konzentration erfordert, muss sich der Wissensarbeiter nach einer Unterbrechung seiner Arbeit erneut in die Aufgabe einarbeiten. Das kostet Arbeitszeit und somit Geld. Die Gefahren konzentrieren sich hier also auf wirtschaftliche Verluste, die durch häufige Ablenkungen der Wissensarbeiter entstehen können. Außerdem wird der Wissensarbeiter gestresst, da ihm durch die Unterbrechungen weniger Zeit für seine Aufgabe zur Verfügung steht. (siehe auch (Heidenreich 2003))

Die Gründe für die Ablenkung sind sehr unterschiedlich, wie sich an den zwei vorangegangenen Beispielen gezeigt hat, und müssen je nach Kontext genauer betrachtet werden. Ob Autofahrer oder Wissensarbeiter: Ablenkung sollte in jedem Fall vermieden werden!

In der vorliegenden Arbeit liegt der Schwerpunkt auf der Wissensarbeit und den Ablenkungsfaktoren in diesem Bereich. Bevor näher auf das zu entwickelnde Framework eingegangen wird, folgt eine Einführung in den Begriff „Kontext“. Dazu wird in dem folgenden Kapitel beschrieben, wie ein Kontextwechsel durch Ablenkung hervorgerufen und zu Arbeitsverzögerungen führen

[7] Quelle Abbildung (a): http://mediacenter.conti-online.com/internet/generator/MAM/index,templateId=Folder_2FrenderDefault.jsp.html?method=show&action=/details.do&oid=7433056, zuletzt besucht am 17.07.2013

[8] Quelle Abbildung (b): http://mediacenter.conti-online.com/internet/generator/MAM/index,templateId=Folder_2FrenderDefault.jsp.html?method=show&action=/details.do&oid=7433040, zuletzt besucht am 17.07.2013

kann. Da dies häufig zu Stress führt und Stress, wie sich zeigen wird, die häufigste Folge von Ablenkung ist, wird im Kapitel 2.2.1 das Thema „Stress“ näher betrachtet.

Anschließend zeigen verschiedene Einsatzszenarien mögliche Bedingungen, die an das Framework gestellt werden. Hier findet dann jeweils eine kurze Darstellung dieser Anforderungen statt. Diese zeigen, welche Flexibilität das zu entwickelnde Framework erreichen muss.

Im letzten Teil dieses Kapitels werden drei Forschungsarbeiten vorgestellt, die sich mit der Stressprävention und Vermeidung von Ablenkung beschäftigt haben. Hier wird sich zeigen, dass weitere Forschungsgruppen „Stress“ als Ausgangspunkt für ihre Arbeiten genommen haben und das Thema immer noch sehr aktuell ist.

2.1 Der Kontext

Der Kontext, in dem eine Information steht, gibt der Information ihren aktuellen Sinn. Im Folgenden wird der Begriff „Kontext“ näher erläutert. Zusätzlich erfolgt die Einführung in den Begriff „Kontextwechsel“ und in die Folgen, die ein Kontextwechsel mit sich bringt.

2.1.1 Kontext und Kontextwechsel

Spricht man vom *Arbeiten*, kann das viele Bedeutungen haben. In erster Linie denkt man vielleicht an die *Arbeit* an einem Büroarbeitsplatz. Was passiert aber, wenn man sagt: „Jemand arbeitet in der Küche.“? Plötzlich assoziiert man *Arbeiten* mit *Kochen*. Der Kontext, in dem „gearbeitet“ wird, hat sich verändert. Und mit ihm auch die Assoziationen, die wir mit dem Begriff *Arbeit* verbinden.

Das Beispiel zeigt, dass jedes einzelne Teil unserer Welt einen Bezug zu den anderen Dingen hat, die es umgeben. Man kann dennoch einzelne Teile in einem eingeschränkten Rahmen, einem Kontext, betrachten. In dem Beispiel wird der Begriff „Arbeit“ als einzelnes Teil hervorgehoben und mit weiteren Informationen, wie dem Begriff „Küche“, verknüpft. Zu dem Kontext des Teils gehört dann jede Information, die genutzt werden kann, um einen Sach- und Situationsbezug zu der Einheit zu definieren. So wird hier der Einheit „Arbeit“ durch die Aussage „Jemand arbeitet in der Küche.“ ein inhaltlicher Sinnzusammenhang gegeben.

Als Einheit wird beispielsweise eine Person, ein Ort oder ein Objekt bezeichnet. Bei der Betrachtung der Einheit werden dann gezielt einzelne Bezüge hervorgehoben und andere

ausgeblendet. Der so aufgebaute Kontext gibt den Dingen einen speziellen Sinn und sagt etwas über ihren aktuellen Sach- und Situationsbezug aus, in dem sie verstanden werden müssen.[9]

Das kleine Beispiel zeigt auch, dass es sehr wichtig ist, den Kontext zu beachten. Eine einheitliche Definition für den Begriff „Kontext“ gibt es jedoch nicht. In dieser Arbeit wird der Begriff *Kontext* im Zusammenhang mit der Wissensarbeit verwendet. Andy Hunt hat in seinem Buch *Pragmatisches Denken und Lernen - Refactor Your Wetware!* folgende Definition genannt:

> „Bei Kontext [...] handelt es sich um die Menge von Informationen, die sich im Zusammenhang mit der vor Ihnen liegenden Aufgabe aktuell in Ihrem Kurzzeitgedächtnis befindet.“ (Hunt 2009, S.211)

Das bedeutet, dass die Informationen im Kurzzeitgedächtnis ausgetauscht werden, sobald man den Kontext wechselt. Im Beispiel passiert dies, sobald verdeutlicht wird, dass es um das Arbeiten in der Küche geht. Die Informationen, die mit dem vorangegangenen Kontext zusammenhingen, sind unwiderruflich verloren. Ein *Kontextwechsel* beim Menschen ist somit nichts anderes, als die Wiederherstellung eines vorherigen bekannten Zustandes oder die Erstellung eines völlig neuen Zustandes einer Informationsbasis im Kurzeitgedächtnis. Im Durchschnitt dauert es zwanzig Minuten, um sich in erneut den Kontext einer komplexen Aufgabe komplett einzuarbeiten. (Hunt 2009, S.212)

Es ist also enorm wichtig, dass der Wissensarbeiter vor einem Kontextwechsel soweit wie möglich bewahrt und bei einem zwingenden Kontextwechsel unterstützt wird. Ein System sollte dabei potentielle Ablenkungsfaktoren erkennen, einstufen und gegebenenfalls unterdrücken.

Andy Hunt gibt in *Pragmatisches Denken und Lernen - Refactor Your Wetware!* verschiedene Tipps, um mit Unterbrechungen, die zwingend zu einem Kontextwechsel führen, umzugehen. Er rät unter anderem dazu, sämtliche Benachrichtigungen abzuschalten oder zumindest zu filtern, so dass nur noch die wichtigen Benachrichtigungen (zum Beispiel von Vorgesetzten und dem Lebenspartner) angezeigt werden (Hunt 2009, S.220).

Außerdem könnten Kontextwechsel, die direkt am Bildschirm stattfinden, minimiert werden. Stellt man sich einen Schreibtisch vor, breitet man häufig alle Informationen, die man zu einer Thematik hat, vor sich aus. So erhält man schnell einen Überblick über alles, was zu dem aktuellen Aufgabenkontext gehört. Am Bildschirm ist dies aufgrund der begrenzten Fläche nur bedingt möglich. Durch einen Mehrbildschirm-Arbeitsplatz kann diese Problematik zumindest verringert werden, da so Informationen, die sonst nur durch den Wechsel von Fenstern sichtbar würden, auf einen anderen Bildschirm ausgelagert und dort angezeigt werden könnten. Hierbei

[9] Siehe auch http://www.duden.de/node/651291/revisions/1204503/view, zuletzt besucht am 18.07.2013.

muss darauf geachtet werden, dass man nur die Informationen auf den zusätzlichen Monitoren anzeigt, die auch zum aktuellen Aufgabenkontext gehören. Beispielweise könnte auf dem Hauptbildschirm ein Texteditor zur Bearbeitung eines Textes geöffnet sein. Auf einem zweiten Monitor ist dann ein PDF-Viewer dargestellt, in dem ein wissenschaftliches Papier geöffnet ist, auf das man sich im Text beziehen möchte.

Eine Erweiterung dieses Vorgehens ist die Nutzung von virtuellen Desktops. Dabei können die benötigten Informationen pro Aufgabe auf je einem virtuellen Desktop gruppiert werden. Auf diese Weise bleibt bei einem Kontextwechsel zumindest der Kontext am Bildschirm erhalten. (Hunt 2009, S.223ff)

2.1.2 Context-aware computer application

Bezieht man nun den Begriff „Kontext“ auf Computerapplikationen, so entstehen *context-aware computer applications*[10]. Hierzu gibt es jedoch, wie auch schon beim Begriff „Kontext“, keine einheitliche Definition. Im Folgenden werden einige Arbeiten vorgestellt, die sich mit der Begriffsbildung *context-aware* beschäftigt haben. In Bezug auf diese Arbeiten schließt die Beschreibung, was *context-aware* in der vorliegenden Arbeit bedeutet, an.

Über allen Definitionen steht ein Grundprinzip. Dieses Grundprinzip besagt, dass eine Applikation, die mit einem Kontext arbeitet, die Eigenschaften „Wer“, „Wo“, „Wann“ und „Was“ der beobachteten Einheit auswerten muss. Aus diesen Informationen kann dann das „Warum“ der Situation, also der Grund für den Kontext, bestimmt werden. (Anind K. Dey und Abowd 1999)

Schilit, Adams und Want haben *context-aware computer applications* als Softwares beschrieben, die über den Kontext informiert werden oder sich selbst über den Kontext informieren können. Zusätzlich kann sich die Software an diesen Kontext anpassen. In »Context-aware computing applications« (Schilit, Adams und Want 1994) haben die Autoren ein System entwickelt, das auf tragbaren Computern unter anderem die Drucker anzeigt, die in der räumlichen Nähe zum Benutzer stehen. An diesem Beispiel können die vier Eigenschaften des Grundprinzips noch einmal beispielhaft dargestellt werden:

Wer ist in den Kontext involviert?
Der Benutzer.

Wo sollen die Informationen über den Kontext eingeholt werden?
Am aktuellen Aufenthaltsort des Benutzers.

[10] zu deutsch sinnhaft: Computerapplikationen, denen der aktuelle Kontext bewusst ist

Wann sollen die Informationen gesammelt werden?
Unmittelbar.

Was für Informationen sind für den Kontext relevant?
Die verfügbaren Drucker in der Nähe des Benutzers.

In ihrer Arbeit haben sie vier Kategorien ausgearbeitet, in die *context-aware computer applications* eingeteilt werden können. Diese vier Kategorien sollen unterschiedliche Funktionsweisen darstellen, die die jeweilige Applikation aufweist. Die Kategorien können dabei in zwei Varianten, die manuelle und die automatische Ausführung, unterteilt werden. Die Kategorien sind im einzelnen:

Proximate selection: Die Applikation kann *manuell* Informationen sammeln, die auf dem aktuellen Benutzerkontext beruhen.

Automatic contextual reconfiguration: Die Applikation sammelt *automatisch* Informationen, die auf dem aktuellen Benutzerkontext beruhen.

Contextual commands: Die Applikation führt *manuell* Kommandos aus, die auf dem aktuellen Benutzerkontext beruhen.

Context-triggered actions: Die Applikation kann *automatisch* Kommandos ausführen, die auf dem aktuellen Benutzerkontext beruhen.

Eine ähnliche Einteilung hat Pascoe in »Adding Generic Contextual Capabilities to Wearable Computers« (Pascoe 1998) aufgestellt. Er befasste sich mit tragbaren Computersystemen im Bereich der Ökologie. Sein Ziel war es, einen kontextabhängigen Informationsservice zu entwickeln. Den Begriff „kontextabhängiger Informationsservice“ beschrieb Pascoe als ein System, das es dem Benutzer ermöglicht, kontextabhängige Informationen zu erhalten, diese zu bearbeiten und neue Informationen anzulegen. Die Informationen konnte der Benutzer bearbeiten und mit anderen Informationen verknüpfen. Die Verknüpfungen ergaben dann den Kontext. Das Einpflegen neuer Informationen wurde dem Benutzer ebenfalls ermöglicht. Die kontextabhängigen Informationen sollten dabei für den Benutzer unabhängig von den darunterliegenden Datenformaten und Sensormechanismen aufbereitet sein. Pascoe hat in seiner Arbeit die folgenden Kategorien aufgestellt:

Contextual sensing: Die Applikation kann kontextabhängige Informationen erkennen und sie dem Benutzer anzeigen.

Contextual adaption: Die Applikation kann, auf Basis des aktuellen Kontextes, automatisch einen Service ausführen oder diesen anpassen.

Contextual resource discovery: Ressourcen und Services, die relevant in Bezug auf den Kontext des Benutzers sind, können von der Applikation lokalisiert und genutzt werden.

Contextual augmentation: Digitale Daten können von der Applikation automatisch mit dem aktuellen Benutzerkontext verknüpft werden.

Zwischen den Ansätzen von *Schilit, Adams und Want* und *Pascoe* können direkt einige Parallelen gezogen werden. Diese Parallelen haben Anind K. Dey und Abowd in »Towards a better understanding of context and context-awareness« (Anind K. Dey und Abowd 1999) unter drei Funktionskategorien zusammengefasst:

***Präsentation* von Informationen und Services:** Der erste Punkt setzt sich aus den Kategorien *proximate selection* und *contextual commands* aus der Arbeit von Schilit sowie *contextual sensig* aus der Arbeit von Jason Pascoe zusammen. Diese Funktionskategorie beschreibt die Ermittlung und Darstellung von kontextabhängigen Informationen und kontextabhängigen Services durch die Applikation für den Benutzer.

***Automatische Ausführung* eines Services:** Diese Funktionskategorie fasst Schilits Kategorie *context-triggered actions* und Pascoes Kategorie *contextual adaption* zusammen. Sie beschreibt die Fähigkeit einer Applikation, auf Basis des aktuellen Kontextes automatisch Reaktionen des System hervorzurufen, indem Services ausgeführt werden.

***Taggen* von Kontext in Bezug auf Informationen zur späteren Verwendung:** Zu den ersten beiden Funktionskategorien kommt die Kategorie *contextual augmentation* von Pascoe hinzu, die hier unter einem allgemeineren Namen beschrieben wird. Sie besagt, dass eine Applikation selbstständig digitale Daten mit dem Benutzerkontext verknüpfen kann.

Weitere Informationen zu *context-aware computing applications* können in den Arbeiten (Anind K Dey 1998; Anind K. Dey, Abowd und Wood 1998) sowie (Dragunov u. a. 2005) nachgelesen werden.

Bezug zu dieser Arbeit

Wie sich in den drei Arbeiten zeigte, gibt es keine einheitliche Definition des Begriffs *context-aware*. Dennoch zeichnen sich Eigenschaften ab, die den Begriff allgemein umschreiben. Je nach

Einsatzgebiet der entwickelten Applikationen werden diese Eigenschaften von den Autoren entsprechend auf ihre eigene Definition abgebildet.

Das Framework, das in dieser Arbeit entwickelt wird, bietet eine Grundlage für die Entwicklung einer *context-aware computer application*. Das exakte Einsatzgebiet des Frameworks und damit der Kontext, in dem das Framework läuft, sind jedoch nicht festgelegt. Es kann daher an dieser Stelle keine einheitliche Definition in Bezug auf das zu entwickelnde Framework gegeben werden.

Die drei beschriebenen Funktionskategorien bieten in meinen Augen aber ein gute Grundlage, um zu einem späteren Zeitpunkt eine Definition zu bilden. Dies kann erfolgen, sobald das Framework in einem definierten Kontext eingesetzt wird. Es muss dann untersucht werden, wie die Komponenten arbeiten:

Präsentieren sie kontextabhängige Informationen?

Verknüpfen sie Informationen mit einem Kontext?

Oder führen sie sogar selbstständig Services auf der Grundlage der kontextabhängigen Informationen aus?

Als Beispiel sei ein System angenommen, das Ablenkung bei einem Arbeitenden erkennen und potentielle Ablenkungsfaktoren verhindern soll. Zusätzlich sei angenommen, dass dieses System auf dem Framework aus der vorliegenden Arbeit basiert. Die eingesetzten Sensoren erheben Messwerte, verknüpfen diese mit dem aktuellen Aufgabenkontext des Benutzers und veröffentlichen sie im Netzwerk für die anderen Komponenten des Systems. Wird eine potentielle Ablenkung erkannt, werden Prozeduren ausgeführt, um diese Ablenkung einzudämmen. Eine mögliche Definition für dieses kontextabhängige System auf der Basis des Frameworks wäre dann:

> Eine *context-aware computer application* stellt ein System dar, das Informationen erhebt, diese mit dem aktuellen Kontext verknüpft und auf der Grundlage dieser Verknüpfungen der Informationen auf Ereignisse im aktuellen Kontext reagiert.

2.2 Stressempfinden und Überlastung geistiger Fähigkeiten

Der Aufgabenkontext eines Wissensarbeiters ist komplex, da der Wissensarbeiter ständig mit vielen Informationen arbeitet. Sobald in seinem Arbeitsablauf ein Kontextwechsel stattfindet, muss sich der Wissensarbeiter in den geänderten Aufgabenkontext einarbeiten. Häufige Kontextwechsel können schnell zu Stress führen, da viel Zeit aufgewendet werden muss, um sich jedes Mal in den geänderten Aufgabenkontext einzuarbeiten. Dies geschieht meist unter

Zeitdruck, da dem Wissensarbeiter in der Regel keine uneingeschränkte Zeit zur Verfügung steht. Zusätzlich kann häufiger Kontextwechsel einen Wissensarbeiter kognitiv überfordern.

2.2.1 Der Begriff „Stress“

Im allgemeinen wird „Stress“ als „eine Belastung, Störung oder [...] Überforderung der psychischen und/oder physischen Anpassungskapazitäten“ (Rensing u. a. 2005, S. 4) verstanden. Das bedeutet, dass „Stress der [...] gespannte Zustand ist, in dem man sich unter Stressoren wie hohem Lärmpegel, Verkehrsdichte oder Arbeitsüberlastung befindet und der mit Gefühlen von Ärger, Angst, Aggressivität, Hilflosigkeit und ihren physischen Korrelaten wie Herzklopfen, Magendrücken oder Schweißausbrüchen einhergeht“ (Rensing u. a. 2005, S. 4). Man kann dabei zwischen positivem Stress (*Eustress*) und negativem Stress (*Disstress*) unterscheiden. *Eustress* beschreibt dabei einen Stress, der meist kurzzeitig ist und die positiven und anregenden Aspekte darstellt. Diese Form des Stresses ist immer vom Menschen kontrollierbar. *Disstress* hingegen steht für Situationen wie Verzweiflung, Erschöpfung oder Elend, in denen der Mensch nicht mehr aktiv reagieren kann. Hier befindet sich der Mensch in einem krankheitsbegünstigten Zustand. (Rensing u. a. 2005, Kapitel 1)

Die Abbildung 2.2 zeigt die einzelnen Organisationsebenen des Menschen. In Stresssituationen werden häufig alle Organisationsebenen angeregt. Für die vorliegende Arbeit sind die Organisationsebenen *somatische Systeme*, *zelluläre / molekulare Systeme* und *psychische Systeme* besonders interessant. Aus der Anregung der Ebene der *somatischen Systeme* folgen die körperlichen Reaktionen in Form von Gestik und Mimik. Diese körperlichen Reaktionen können gemessen werden. Außerdem ist die Organisationsebene der *zellulären / molekularen Systeme* von Interesse. Die Anregungen dieser Systeme können zum Beispiel Veränderung auf der Haut (Schweiß, Temperatur) hervorrufen, die ebenfalls messbar sind. Die Auswirkungen, die auf die Anregung der *psychischen Systeme* folgt, stellen das subjektive Erleben und damit die Individualität des Stressempfindens dar. Sie sind nicht messbar, müssen aber bei der Auswertung beachtet werden.

Stress am Arbeitsplatz

„Stress am Arbeitsplatz“ gilt im Bereich der wissenschaftlichen Untersuchungen als eine Konkretisierung der Forschungsrichtung „Stress“. Es gibt verschiedene Definitionen von „Stress am Arbeitsplatz“, die Rensing u. a. zusammengefasst haben:

> „Arbeitsplatzstress kann als schädliche physikalische und emotionale Reaktion definiert werden, die auftritt, wenn die Arbeitsbedingungen nicht mit den Kompe-

> tenzen, Ressourcen oder Bedürfnissen des Arbeiters übereinstimmen. Arbeitsstress kann zu mangelhafter Gesundheit und sogar Verletzungen führen."(Rensing u. a. 2005, S. 30)

> Arbeitsplatzstress ist „die emotionale, kognitive, verhaltensbedingte und physiologische Reaktion auf aversive und gesundheitsschädliche Aspekte der Arbeit, der Arbeitsumgebung und der Arbeitsorganisation. Es ist ein Zustand, der durch hohe Erregung und Leiden sowie häufig durch Gefühle des Nicht-Zurechtkommens charakterisiert ist."(Rensing u. a. 2005, S. 30)

Beide Definitionen drücken Negativität aus. Stress am Arbeitsplatz sollte nach den Definitionen von Rensing u. a. demnach in jedem Fall vermieden werden, um die Gesundheit des Arbeitenden nicht zu gefährden und seine Arbeitskraft aufrecht zu erhalten.

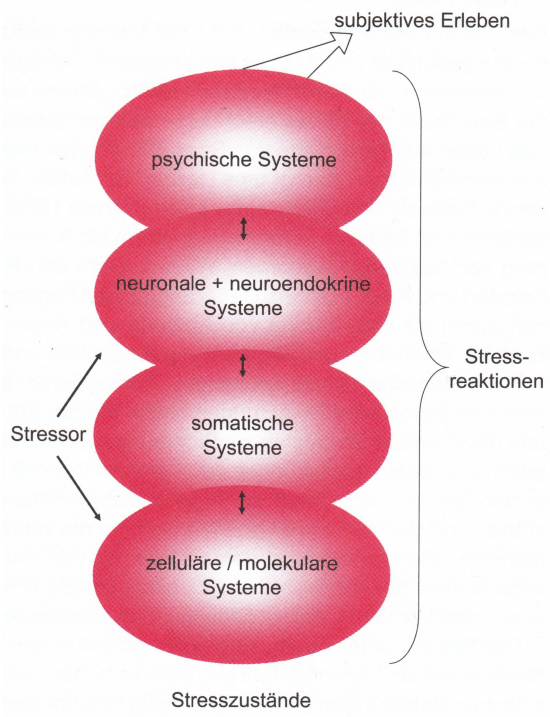


Abbildung 2.2: Organisationsebenen des Menschen mit Bezug zu Stressoren und Stresswahrnehmung (Rensing u. a. 2005, S. 2)

Eine Studie der DAK, die im Jahr 1997 durchgeführt wurde, zeigt zudem, dass Termin- und Zeitdruck die am häufigsten empfundenen Stressoren sind (Rensing u. a. 2005, S. 31). Das Problem, das sich aus Termin- und Zeitdruck ergibt, kann Mangel an Vorbereitungszeit für die

folgenden Termine sein. Hierdurch besteht die Gefahr, dass der Mensch in eine unkontrollierbare Stresssituation (Disstress) gerät.

2.2.2 Was uns stresst - Die Stressoren

Stress wird durch Stressoren, also Ereignisse oder Einflüsse, die auf den Menschen einwirken, ausgelöst. Natürliche Stressoren sind Situationen, in denen der Mensch einer Gefahr ausgesetzt ist. Als Beispiel sei hier der Angriff eines Tigers genannt. Stressoren können dabei sowohl von außerhalb unseres Körpers (exogen - zum Beispiel: Angriff durch Tiger) wie auch von innen (endogen - zum Beispiel: Hohe Selbsterwartung) wirken.

Rensing u. a. haben festgestellt, dass in Industriegesellschaften die Hauptstressoren Termindruck, Angst vor Arbeitslosigkeit, Belastung am Arbeitsplatz und in der Familie sowie der sozioökonomische Status sind. Hinzu kommen physikalische Stressoren wie UV- und ionisierende Strahlung, Hitze und Kälte, mechanische Beanspruchung und Lärm. (Rensing u. a. 2005, Kapitel 2)

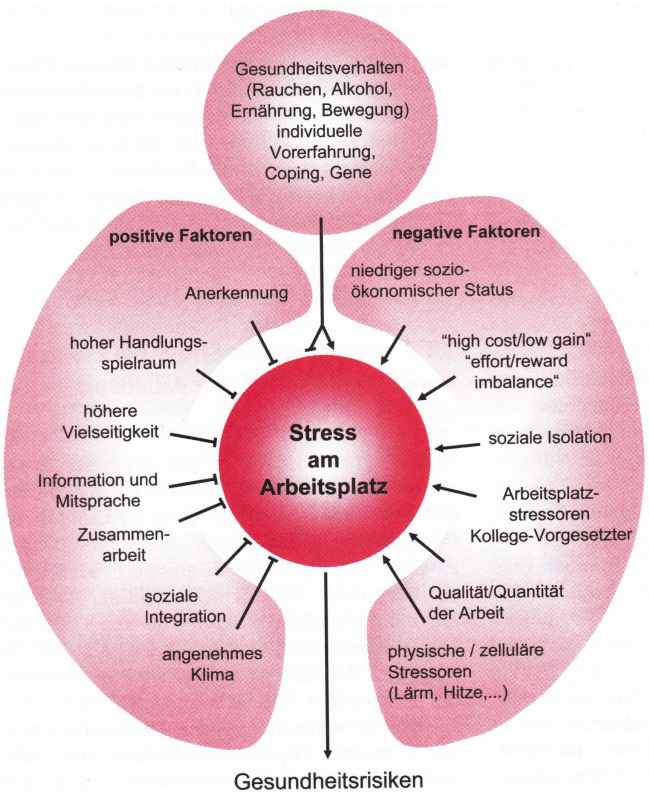


Abbildung 2.3: Stressoren, die Stress am Arbeitsplatz beeinflussen (Rensing u. a. 2005, S. 30)

Die Abbildung 2.3 zeigt einige Stressoren, die Stress am Arbeitsplatz beeinflussen können. Die Abbildung ist in positive und negative Faktoren unterteilt. Die Folgen von Stress am Arbeitsplatz sind Gesundheitsrisiken, wobei die positiven Faktoren die Gesundheitsrisiken mildern und die negativen Faktoren die Risiken verstärken.

2.2.3 Stressempfinden und Stressmessung

Wie stark Stressoren den Menschen beeinflussen, hängt vom jeweiligen Individuum ab. Jeder Mensch hat einerseits durch seine Gene, andererseits durch seine Lebenserfahrung, seine Lebensgewohnheiten, sein Alter und Geschlecht sowie seine Ethnie unterschiedliche Empfindlichkeiten gegenüber einzelner Stressoren. So wird beispielsweise laute Musik eher dem älteren Menschen Stress zuführen als dem jungen.

Besonders stark wird Stress empfunden, sobald der Mensch den Stressor nicht mehr durch aktives Verhalten vermeiden oder kontrollieren kann. Die Schwelle, ab der eine Stresssituation unkontrollierbar wird, ist ebenfalls individuell unterschiedlich.

Das Stresserleben läuft dabei auf sieben Ebenen ab (Rensing u. a. 2005, S. 68):

1. Bewusste Eigenwahrnehmung von psychischem und körperlichem Erleben.
2. Aktivierung der Mimik, die von anderen als Emotionsausdruck wahrgenommen wird.
3. Aktivierung von Gestik und Körperhaltung, die den mimischen Emotionsausdruck begleitet.
4. Aktivierung motorischer Handlungsbereitschaften zur Regulierung unlustvoller und lustvoller Situationen (Reaktionsverhalten wie *abwehren* und *festhalten*).
5. Physiologische Aktivierung des vegetativen und endokrinen Systems (somatischer Anteil des Affekts).
6. Psychophysiologische Abläufe und ihre Wahrnehmung (Herzklopfen, Schwindel, Atemnot).
7. Sprachliche Benennung der psychisch und körperlich erlebten Emotionen.

Bei der Stressmessung ist es wichtig, das persönliche Stresserleben des Menschen zu beachten. In der Abbildung 2.4 ist das subjektive Erleben der naturwissenschaftlichen Beobachtung einer Stresssituation gegenübergestellt. Der Doppelpfeil zwischen der *Psychoanalyse* und dem *„Ich“* auf der linken Seite deutet einen Dialog zwischen dem Psychotherapeuten und dem Patienten an, der bei der Psychoanalyse stattfindet. Bei der wissenschaftlichen Analyse hingegen

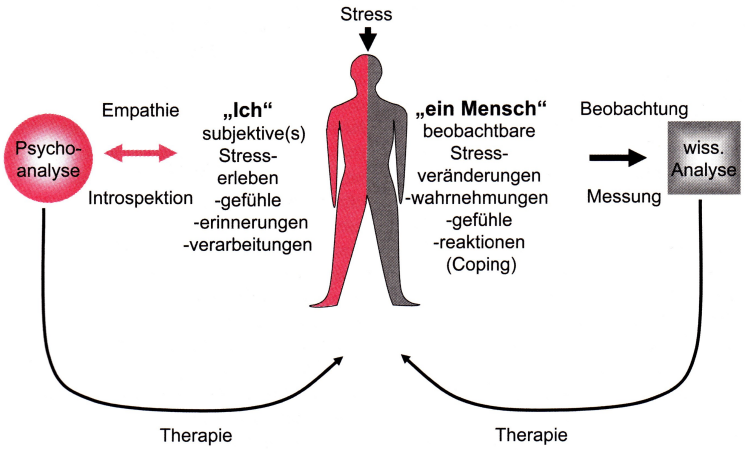


Abbildung 2.4: Subjektives Erleben und naturwissenschaftliche Beobachtung einer Stresssituation (Rensing u. a. 2005, S. 16)

findet ein Monolog statt (einseitiger Pfeil rechts). Die wissenschaftliche Analyse nimmt die Informationen, die aus der Situation und der Reaktion des Menschen beobachtet und gemessen werden können, auf und verarbeitet sie. Die Naturwissenschaft muss sich bei der Auswertung der Informationen somit auf vorherige Erkenntnisse stützen. Erschwerend kommt hinzu, dass jeder Mensch ganz individuell auf Stress reagiert. Dies ist darauf zurückzuführen, dass der Mensch im Laufe seines Lebens verschiedene Strategien mit unterschiedlichen Stressreaktionen entwickelt hat, um mit der Belastung umzugehen. Diese Reaktionen (auch Coping-Strategien genannt) können unter anderem

- Abwehr oder Verdrängung von Konflikten,
- Regression auf kindliche Schutzsuche sowie
- Rückzug oder aktive Auseinandersetzung

sein. (Rensing u. a. 2005, Kapitel 3)

Die Messung von Stress ist ein extrem schwieriges Unterfangen. Es ist ein Forschungsgebiet für sich und kann daher in dieser Arbeit nicht behandelt werden. Zum Einstieg in das Thema sei an dieser Stelle auf die Arbeiten (Gulhane, Rode und Ladhake 2011) und (Serva u. a. 2011) verwiesen. Andere Forschungsgruppen haben sich mit ähnlichen Themen, wie der Messung von Frustration (siehe Fernandez und R.W. Picard 1998; Kapoor, Burleson und R. W. Picard 2007) und der Erkennung von Emotionen (siehe Hook 2008; Müller 2010; Valenti, Sebe und Gevers 2007), beschäftigt. Ein ebenfalls sehr aktiver Forschungsbereich ist die Stresserkennung beim Steuern von Fahrzeugen (siehe Benoit u. a. 2009; Healey, Seger und Rosalind Picard 1999).

2.2.4 Stressprävention

Stressprävention zielt auf die Minderung von Stressoren und die Reduktion der Anzahl potentieller Stressoren ab. Dies ist besonders bei endogenen, physischen Stressfaktoren wichtig, da hier bei längerer Belastung ein hohes Maß an Schäden zu erwarten ist. Exogene Stressoren wie Lärm, extreme Temperaturen und Temperaturschwankungen sowie hohe körperliche Belastungen können ebenfalls Schäden nach sich ziehen und sollten, besonders am Arbeitsplatz, schnell erkannt und vermieden werden. (Rensing u. a. 2005, Kapitel 2)

Da die benannten exogenen Stressoren gut messbar sind, bietet sich hier eine Automatisierung der Erkennung und automatische Prävention oder Warnung vor der Stresssituation an. Es können zum Beispiel Temperatursensoren und Mikrofone die Umgebung überwachen. Auf der Grundlage der Messwerte dieser Sensoren wären Aktoren dann in der Lage, automatisch auf Veränderungen oder extreme Werte zureagieren.

2.2.5 Überlastung der kognitiven Fähigkeiten

Ein weiterer Ursprung für Stress ist die Überlastung der kognitiven Fähigkeiten. Diese Überlastung entsteht zum Beispiel durch häufige Unterbrechungen bei der Arbeit. Eine Unterbrechung ist hierbei definiert als „der Wechsel von einer Umgebung in einer andere“ (Kirsh 2000, S. 32). Dies ist zum Beispiel genau dann der Fall, wenn man seinen aktuellen Aufgabenkontext wechselt, um eine andere Aufgabe zu bearbeiten. Gerade Wissensarbeiter müssen häufig zwischen ihren verschiedenen komplexen Aufgabenkontexten wechseln und sind daher besonders gefährdet, ihre kognitiven Fähigkeiten zu überlasten. Multi-Tasking ist zum alltäglichen Lebensstil von Wissensarbeitern geworden. (Kirsh 2000)

Die Quellen, aus denen die enorme Menge an Informationen täglich auf den Wissensarbeiter einprasseln, sind zum Beispiel: E-Mails, Anrufe, elektronische Diskussionsgruppen, Webseiten, gepushte Intranetnews, Briefe, Memos, Faxe, Kalender, Pager und natürlich die direkte physikalische Kommunikation. Hierzu ein kleines Beispiel:

> Ein Wissensarbeiter einer Einrichtung hatte zehn Tage Urlaub. Er kommt zurück in sein Büro und bringt einen Haufen Post mit, den er an der Poststelle abgeholt hat. Auf seinem Stuhl liegt ein weiterer Stapel an Post, der nicht in den Postkasten passte. Er startet seinen Computer und sieht sich mit 260 E-Mails, die er während seiner Abwesenheit erhalten hat, konfrontiert. Auf seinem Anrufbeantworter sind zudem 14 Nachrichten während seiner Abwesenheit eingegangen. Er beginnt die E-Mails durchzugehen und anhand des Betreffs auszusortieren. 21 E-Mails sieht er sich genauer an und beantwortet sechs dringende E-Mails sofort. Danach

> hört er die Nachrichten auf dem Anrufbeantworter ab und ruft bei vier Anrufern zurück. Zu guter Letzt setzt er sich mit seinem Papierkorb und der Post auf den Fußboden und beginnt die Post zu sortieren. Er legt dabei verschiedene Stapel an. Die Werbung wirft er direkt weg. Den Stapel mit der Post, die an Personen aus seiner Abteilung gerichtet sind, nimmt er danach direkt mit. (Kirsh 2000)

Dieses kleine Beispiel zeigt, dass es häufig nur wenige relevante Informationen in der Masse gibt. Um nicht mit Informationen überfrachtet zu werden, müssen die Informationen vom Wissensarbeiter entsprechend gefiltert werden. Andernfalls droht die kognitive Überlastung. Es kann zwischen vier verschiedenen Fällen von Überlastung unterschieden werden (Kirsh 2000):

Zu viel Informationsangebot: Es gibt immer mehr Informationen, gleichzeitig aber nicht immer bessere Informationen. Die Abbildung 2.5 stellt diese Entwicklung graphisch dar.

Zu viel Informationsnachfrage: Viele Wissensarbeiter wollen immer mehr Informationen, da sie der Meinung sind, dass es andauernd noch bessere Informationen zu ihrer Aufgabe geben muss. Ein Problem hierbei ist, dass man in der Regel nicht wissen kann, ob es überhaupt noch Informationen zu der Aufgabe gibt.

Durchgehendes Multi-Tasking und Unterbrechen der Aufgabe: Die Bearbeitung von komplexen Aufgaben, denen eine große Menge an Informationen zu Grunde liegt, führt zur Unterteilung der Arbeit in mehrere Teilaufgaben. Zwischen diesen Teilaufgaben muss jedoch häufig gewechselt werden. Jeder Wechsel führt zu einer Unterbrechung, nach der man sich in den neuen Aufgabenkontext zunächst einarbeiten muss.

Inadäquate Arbeitsplatz-Infrastruktur: Die verfügbaren Tools helfen nur geringfügig bei der Überwachung von Informationen. Neue Tools sollten den Arbeitsfluss des Wissensarbeiters während seiner Arbeit mit Informationen nicht mehr stören oder unterbrechen.

Wie verschiedene Arbeiten zeigen, kann die aktuelle kognitive Auslastung aber auch als Messwert gesehen werden. Eine hohe kognitive Auslastung bedeutet, dass der Wissensarbeiter gerade sehr konzentriert an einer komplexen Aufgabe arbeitet. Aus den Messergebnissen können dann Unterbrechungen zeitlich auf den Arbeitsablauf eines Wissensarbeiters abgestimmt werden (siehe hierzu unter anderem Haapalainen u. a. 2010).

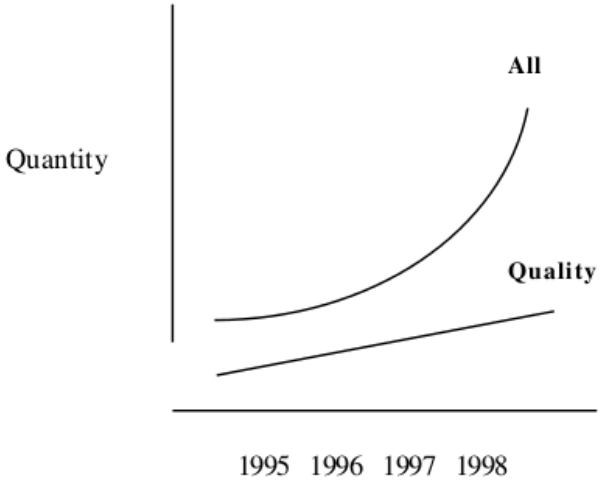


Abbildung 2.5: Entwicklung des Informationsangebots - Masse und Qualität (Kirsh 2000)

Bezug zu der vorliegenden Arbeit

Über einen Punkt ist sich die Literatur einig: Stress kann negative Folgen für die Gesundheit des Menschen haben. Was genau Stress auslöst, ist jedoch von Mensch zu Mensch unterschiedlich. Deswegen muss auf diesem Gebiet noch mehr Forschung betrieben werden.

Stress ist lediglich eine Folge von Ablenkung oder Überforderung. Wissensarbeiter sind, durch die Menge an Informationen mit denen sie arbeiten, besonders anfällig für eine kognitive Überlastung und somit auch einer Überforderung ihrer Kompetenzen. Diese kognitive Überlastung tritt besonders häufig auf, wenn der Wissensarbeiter während seiner Arbeit abgelenkt wird. Um den Wissensarbeiter davor zu schützen, muss sein Arbeitskontext überwacht, Ablenkung vermieden und der Wissensarbeiter bei einem zwingenden Kontextwechsel unterstützt werden.

Um hier Systeme zu entwickeln, die den Wissensarbeiter unterstützen können, wird allerdings eine Infrastruktur benötigt, die es ermöglicht unterschiedliche Sensoren zusammenarbeiten zu lassen. Auf diese Weise könnte ein großes System zur Messung körperlicher Reaktionen bei Ablenkung unter hochkonzentrierter Arbeit gebaut werden. Eine solche Infrastruktur soll mein Framework anbieten. Potentielle Plug-ins könnten dann spezielle Sensoren implementieren und diese so auf einfache Weise an das Framework anbinden.

Das zu entwickelnde Framework soll verschiedene Szenarien abdecken können. In den folgenden Abschnitten wird auf drei mögliche Szenarien genauer eingegangen.

2.3 Anwendungsszenarien

Die folgenden drei Anwendungsszenarien sollen beispielhaft die vielfältigen Anforderungen, die an das zu entwickelnde Framework gestellt werden müssen, aufzeigen. Hieraus lässt sich dann der benötigte Funktionsumfang des Frameworks bestimmen.

2.3.1 Szenario 1: IT-Projektleiter am Arbeitsplatz

Es ist Donnerstagnachmittag. Bob, IT-Projektleiter, sitzt an der Ausarbeitung eines komplexen Pflichtenheftes. Bob arbeitet gerade höchst konzentriert an einem Absatz, zu dem er einige wichtige Gedanken im Kopf hat.

Plötzlich öffnet sich am unteren rechten Bildschirmrand ein kleines Pop-up-Fenster mit einer Benachrichtigung über eine neue E-Mail. Die Benachrichtigung hat die Überschrift „1 neue Mail“ und den Inhalt „15 neue Tickets im Bugtracker eröffnet“. Bobs Blick wandert nur kurz auf das Pop-up-Fenster. Dieser kurze Blick reicht jedoch aus, um zu erkennen, dass im Bugtrackingsystem verhältnismäßig viele neue Tickets eröffnet wurden. Bob versucht sich dennoch auf den aktuellen Absatz im Pflichtenheft zu konzentrieren. Er wird nach der Fertigstellung des Absatzes ins Bugtrackingsystem gucken.

Einen Moment später ruft eine Freundin namens Alice über Skype an. Skype stellt Bob ein Fenster zur Annahme des eingehenden Anrufes direkt in der Mitte des Bildschirms dar. Bob hatte vergessen vor Beginn seiner Arbeit an dem Pflichtenheft seinen Skype-Status von „online“ auf „beschäftigt“ zu stellen. Durch den Anruf wird Bob nun endgültig aus seinem Arbeitskontext gerissen. Er hat den Großteil seiner Gedanken zu dem aktuellen Absatz vergessen und wird so in seiner Arbeit um einige Zeit zurück geworfen.

Fazit

Dieses Szenario zeigt, dass es am IT-Arbeitsplatz viele Störfaktoren gibt, die den Arbeitenden bei seiner Arbeit unterbrechen können. Auf Bob haben hier zwei Störfaktoren eingewirkt, auf die im Folgenden noch einmal genauer eingegangen wird.

Die erste Störung beinhaltete wichtige Informationen für ihn als IT-Projektleiter. Sie zeigte ihm durch die Anzahl der neuen Tickets, dass irgendetwas in seinem Projekt nicht gut läuft. Er konnte anhand dieser Information entscheiden, ob er sich sofort oder später mit der Problematik auseinandersetzt.

Die zweite Störung hatte weniger berufliche Relevanz. Den Anruf von Alice konnte Bob nur anhand des Namens der Anruferin einordnen. Zusätzlich wurde ihm der Anruf durch Skype vor seinen Arbeitsfokus gesetzt. Er hatte keine Möglichkeit diese Störung zu übersehen.

Jede Störung hat eine Informationsquelle als Ausgangspunkt. Die dahinterliegende Information und deren Wichtigkeit sind jedoch nicht immer direkt ersichtlich. Zusätzlich muss die Relevanz einer Information im aktuellen Kontext bewertet werden. Eine Nachricht eines Freundes bezüglich eines Termins hat am Arbeitsplatz weniger Relevanz, als die Nachricht vom Vorgesetzten, der ebenfalls einen Termin absprechen möchte. Einige der Informationsquellen sind jedoch unabdingbar. So würde kein IT-Professional am Arbeitsplatz komplett auf die Benachrichtigungen seines E-Mail-Programms verzichten wollen.

Die Folgen der Störfaktoren sind unter anderem der Verlust von Gedanken im Kurzzeitgedächtnis. Außerdem kann sich durch Störung ein möglicher Wechsel des emotionalen Zustandes ergeben. So kann sich der Unterbrochene über die ihm unterbreitete Information freuen (zum Beispiel über die Nachricht eines Lottogewinns) oder aber über die Störung ärgern, da sie zum Beispiel völlig unrelevante Informationen für ihn beinhaltet. Der Ärger kann sich bei wiederholter Störung weiter steigern und in Wut übergehen. Wichtig ist dabei auch, dass jeder Mensch ganz individuell mit den verschiedenen Störfaktoren umgeht. Jemand, der seit mehreren Jahren in einem Großraumbüro arbeitet, kommt gegebenenfalls besser mit Störungen am Arbeitsplatz zurecht als jemand, der in einem Einzelbüro in völliger Stille arbeitet.

Ziel ist es, die Leistung des Arbeitenden dauerhaft zu erhöhen und ihn vor den Folgen von Stress zu bewahren. Dies könnte in diesem Beispiel durch eine automatische Erkennung potentieller Störfaktoren und der Auswirkung dieser auf den aktuellen Zustand des Arbeitenden erreicht werden. Dazu ist ein System notwendig, dass durch verschiedene Sensoren an unterschiedlichen Stellen den aktuellen Zustand der gesamten Arbeitsumgebung des Projektleiters überwacht. Neben dem Desktop und den Anwendungen auf diesem sollten auch das Bugtracking-System und der aktuelle emotionale Zustand des Projektleiters überwacht werden. Nur so kann ein System flexibel auf die aktuelle Arbeitssituation des Projektleiters reagieren. Das entstehende System könnte durch eine lernende Komponente erweitert werden, die das Verhalten des Systems automatisch an die individuellen Eigenschaften des jeweiligen Nutzers anpasst.

2.3.2 Szenario 2: Laboruntersuchungen zur körperlichen Reaktion bei Ablenkung unter hoher Arbeitsbelastung

Die Forscherin Carol und ihr Team haben in einem Labor einen Versuchsaufbau eingerichtet. Sie möchten die körperlichen Reaktionen, die beim hochkonzentrierten Arbeiten durch Ablenkung ausgelöst werden können, untersuchen. Außerdem möchten sie die Auswirkungen der individuellen Persönlichkeit des Menschen auf Ablenkungsfaktoren bestimmen. Ihr Ziel ist es, neue Erkenntnisse auf diesen Gebieten zu gewinnen. Hierzu soll jeweils ein Proband vor einem

Computer sitzen und Aufgaben bearbeiten, die einer realen Arbeitssituation nachempfunden sind. Der jeweilige Proband wird dabei immer wieder kontrolliert auf unterschiedliche Arten unterbrochen.

Während des gesamten Testverfahrens messen Carol und ihr Team verschiedene körperliche Reaktionen der Probanden. Zusätzlich zeichnen sie Messwerte der Umgebung der Probanden, wie zum Beispiel die aktuelle Raumtemperatur, die Luftfeuchtigkeit, die Tageszeit und das Wetter, auf. Zur Bestimmung des individuellen Empfindens von Ablenkung wird zu jedem Probanden ein Profil angelegt. Das Profil umfasst unter anderem den aktuellen Ausbildungsstand, das Alter, die Berufserfahrung, einen Wert zum persönlichen Stressempfinden sowie die aktuelle Position im Unternehmen. Es soll helfen, aus den persönlichen Daten einer Person, die Belastungsgrenze gegenüber Ablenkungen zu bestimmen.

Zur Messung der körperlichen Reaktionen hat das Team verschiedene Sensoren aufgebaut. Um den Hautleitwert zu ermitteln, wurden Sensoren vorbereitet, die direkt am Körper der Probanden angebracht werden. Zusätzlich haben sie Kameras aufgestellt, die die Bewegungen der Probanden aufzeichnen sollen. Eine zusätzliche Hochgeschwindigkeitskamera misst den Puls (siehe Abbildung 2.6).[11] Ein Eyetracker erfasst die Augenbewegungen und die Pupillendurchmesser der Probanden. Weiterhin setzen Carol und ihr Team Temperatur- und Luftfeuchtigkeitssensoren sowie Sensoren zur Ermittlung der aktuellen Uhrzeit und des aktuellen Wetters ein. Die Sensoren sind an unterschiedlichen Computern angeschlossen. Diese Computer sind wiederum untereinander mittels eines Local Area Networks (LANs) miteinander verbunden. Zusätzlich steht dem Probanden ein Buzzer zur Verfügung, durch den er signalisieren kann, dass er sich durch die Störfaktoren stark von seiner Arbeit abgelenkt fühlt.

Nachdem die Tests mit einem Probanden durchgeführt wurden, müssen Carol und ihr Team die Messdaten von den unterschiedlichen Computern zusammentragen. Die gesammelten Messdaten speichern sie dann gemeinsam ab, um diese später auszuwerten.

Fazit

Carol und ihr Team haben ein Sensornetzwerk aufgebaut, dass keine Möglichkeit bietet, die Messdaten über das vorhandene Netzwerk zu verteilen. Stattdessen mussten sie nach jedem Testdurchlauf alle Daten per Hand von den einzelnen Computern einsammeln.

Um diesen Umstand zu lösen, wird ein System benötigt, das die einzelnen Computer und die Sensor-Implementationen miteinander verbindet. Es muss dabei unabhängig von der Architektur des Computers, des Betriebssystems und dem speziellen Application Programming Interface (API) des Sensors sein. Die Messdaten sollen, während der Test noch läuft, an einer

[11] Für mehr Informationen zu diesem Verfahren siehe (Wu u. a. 2012).

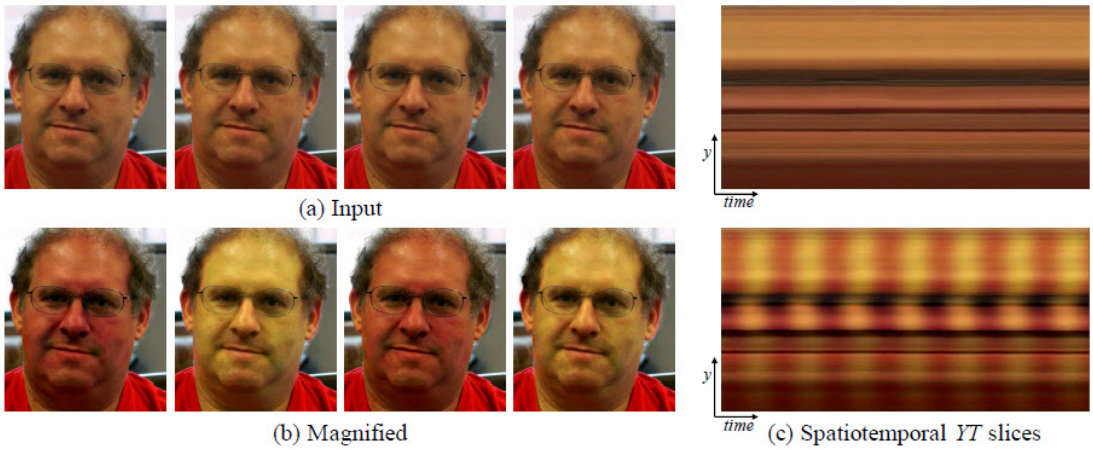


Abbildung 2.6: Eulersche Videoverstärkung (siehe Wu u. a. 2012)

zentralen Stelle gesammelt werden. Dies würde zusätzlich zu der Speicherung der Daten sogar eine Echtzeitauswertung der Daten ermöglichen.

Außerdem sollte es von einem zentralen Punkt aus möglich sein, alle Sensoren im Netzwerk mit den jeweiligen spezifischen Eigenschaften zu konfigurieren oder ihnen Befehle zu schicken. Zum Beispiel könnte ein Befehl eine neue Testreihe ankündigen und einen Tag mitschicken, der die Testreihe markiert. Alle Sensoren würden dann ihre Messdaten mit diesem Tag versehen können. Eine händische Gruppierung der Messdaten pro Testreihe wäre dann nicht mehr notwendig.

2.3.3 Szenario 3: Verteiltes Benachrichtigungssystem

Zur Überwachung einer technischen Anlage werden verschiedene Sensoren eingesetzt. Ziel ist es, die Funktionsfähigkeit der technischen Anlage sicherzustellen. Die Sensoren sind dabei nicht Teil der technischen Anlage. Sie beobachten lediglich die technische Anlage und speichern ihre Messwerte ab.

Im Einsatz befinden sich verschiedene Hardware-Sensoren wie Temperatursensoren, Luftfeuchtigkeitssensoren und Kameras. Ein Temperatursensor überwacht beispielsweise die Betriebstemperatur von Motoren. Zusätzlich werden unterschiedliche Software-Sensoren zur Überwachung von E-Mail-Servern und Authentifizierungssystemen eingesetzt. Sämtliche Sensoren sind über ein Netzwerk miteinander verbunden.

Der Administrator Dave möchte gerne einen raschen Überblick über die aktuelle Funktionsfähigkeit der technischen Anlage bekommen. Hierzu kann er die Messwerte der Sensoren verwenden, da diese die technische Anlage beobachten. Es gibt allerdings keine Software,

die die Messwerte aller Sensoren abgreift und diese für Dave anzeigt. Stattdessen muss er jeden einzelnen Sensor überprüfen und sich selbst ein Gesamtbild über die technische Anlage machen.

Fazit

Damit Dave einen Überblick über die technische Anlage bekommt, muss er viel Zeit investieren. Dave muss jeden Sensor einzeln untersuchen, um einen Gesamtüberblick über die technische Anlage zu erhalten. Je mehr Sensoren im Einsatz sind, desto länger wird die Überprüfung der Anlage dauern.

Für eine mögliche Lösung dieses Problems sollten alle Sensoren ihre Messdaten an einer zentralen Stelle ablegen. So wäre es möglich, dass eine Software diese Daten gebündelt abgreift und für Dave anzeigt. Dave müsste so nur noch an einer zentralen Stelle die Daten prüfen.

Dieser Ansatz bringt noch einen weiteren Vorteil mit sich. Sämtliche Messdaten wären unabhängig von der Aktivität von Dave für jede beliebige Software im System vorhanden. Somit kann ebenfalls eine Software entwickelt werden, die automatisch den Zustand der technischen Anlage überwacht. Sollten Messwerte außerhalb eines Toleranzbereichs liegen, wird der Administrator Dave durch die Software benachrichtigt. Dem Administrator wird es so ermöglicht, jederzeit schnell auf Probleme der technischen Anlage zu reagieren.

2.4 Themennahe Arbeiten

Die folgenden drei Arbeiten haben großen Einfluss auf die vorliegende Arbeit genommen. Es konnten Erkenntnisse aufgegriffen und verarbeitet sowie Abgrenzungen zu der vorliegenden Arbeit gemacht werden. Diese Arbeiten zeigen ebenfalls, wie umfangreich die Problemstellung der automatischen Erkennung von emotionalen Zuständen des Menschen ist.

2.4.1 Oasis - Omniscient Automated System for Interruption Scheduling

Shamsi T. Iqbal und Brian P. Bailey haben ein System zur Steuerung von Benachrichtigungen auf einer Desktopumgebung entwickelt. In ihrer Arbeit »Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks« (Shamsi T. Iqbal und Brian P. Bailey 2010) beschreiben sie ihr System und grundlegende Erkenntnisse aus anderen Arbeiten, die die Entwicklung ihres Systems geprägt haben.

Die Abkürzung Oasis steht für *Omniscient Automated System for Interruption Scheduling*. Iqbal und Bailey haben das System für eine Windows-Desktopumgebung entwickelt. Hier fungiert Oasis als Planungskomponente und steuert die zeitliche Anzeige von Benachrichtigungen.

Die Anzeige einer Benachrichtigung soll dabei auf Unterbrechungen im aktuellen Arbeitsablauf des Arbeitenden abgestimmt werden. Hierzu werden zwei Arten von Plug-ins eingesetzt. Die einen überwachen den aktuellen Arbeitskontext (zum Beispiel die Integrated Development Environment (IDE)) und versuchen Unterbrechungspunkte zu erkennen und einzuordnen. Die anderen Plug-ins steuern das Benachrichtigungsverhalten anderer Anwendungen. Sie stimmen dabei die Anzeige der Benachrichtigung mit Oasis ab. Oasis arbeitet dabei vollständig automatisiert.

Benachrichtigungen

Um die Steuerung der Anzeige von Benachrichtigungen zu übernehmen, muss zunächst definiert werden, was Benachrichtigungen überhaupt sind. Für Iqbal und Bailey sind sämtliche visuelle Hinweise, akustische Signale und haptische Warnungen, die auf einer Desktopumgebung auftauchen können, Benachrichtigungen.

Benachrichtigungen bringen viele Vorteile mit sich, wie zum Beispiel die nahezu unmittelbare Kommunikation zwischen einem Programm und dem Anwender (Czerwinski, Cutrell und Horvitz 2000b; Latorella 1999). Somit ist es auch kaum verwunderlich, dass alle diese Benachrichtigungen normalerweise sofort, also sobald die dahinterliegende Information verfügbar ist, angezeigt werden. Sie werden aber hierdurch häufig genau dann angezeigt, wenn der Arbeitende gerade sehr in seiner Arbeit vertieft ist. Dies führt zu einer verringerten Arbeitsleistung, da der Arbeitende durch die Anzeige der Benachrichtigung aus seinem Arbeitskontext gerissen werden kann (Czerwinski, Cutrell und Horvitz 2000a; Kreifeldt und McCarthy 1981; Latorella 1998). Die Abschaltung der Benachrichtigungen wäre aber der falsche Ansatz. Studien haben gezeigt, dass Benutzer Benachrichtigungen haben wollen, damit sie nicht ständig selbst nachsehen müssen, ob neue Informationen zur Verfügung stehen (Shamsi T. Iqbal und Horvitz 2010). Eine Abstimmung des Zeitpunktes der Anzeige einer Benachrichtigung auf die Unterbrechungen im Arbeitsablauf des Arbeitenden erscheint somit sinnvoll.

Unterbrechungen von Aufgaben

Die menschliche Wahrnehmung unterteilt aufgezeichnete Aktivitäten in hierarchische Strukturen, bestehend aus diskreten Aufgabenteilen (Newtson und Engquist 1976). Somit ist es prinzipiell möglich, Aufgabenteile und den Wechsel zwischen diesen Teilen zu erkennen. Wie Shamsi T. Iqbal und Brian P. Bailey in »Understanding and developing models for detecting and differentiating breakpoints during interactive tasks« (Shamsi T. Iqbal und Brian P. Bailey 2007) herausgefunden haben, können Unterbrechungen von interaktiven Aufgaben am Computer in drei Granularitäten unterteilt werden:

Fein: Tritt zum Beispiel beim Wechsel von der Bearbeitung eines Quellcodes zum Kompilieren des Quellcodes auf.

Mittel: Stellt den Wechsel von einer Quellcode Datei X zu einer anderen Quellcode Datei Y dar, wobei die Dateien inhaltlich nicht zusammenhängend sind.

Grob: Beschreibt einen Wechsel von völlig unzusammenhängenden Aufgaben, wie dem Wechsel von einer IDE zu einem Media Player.

Das Framework

Oasis versucht im aktuellen Arbeitsablauf des Arbeitenden Aufgabenwechsel zu erkennen und diese einer der drei Granularitäten zu zuordnen. Dabei arbeitet das System mit statistischen Modellen. Diese werden auf Basis von aufgezeichneten Arbeitsabläufen durch Lern-Algorithmen[12] erstellt. Die Modelle wendet der *Breakpoint Detector* auf die aktuellen Events an. Die Events werden von den Add-ins bereitgestellt, die zur Überwachung des Arbeitskontextes direkt in den Applikationen integriert wurden, mit denen der Anwender arbeitet. So haben Iqbal und Bailey zum Beispiel Add-Ins für *Visual Studio* und *Microsoft Visio* entwickelt. Die Abbildung 2.7 zeigt den Aufbau des Frameworks sowie die externen Komponenten.

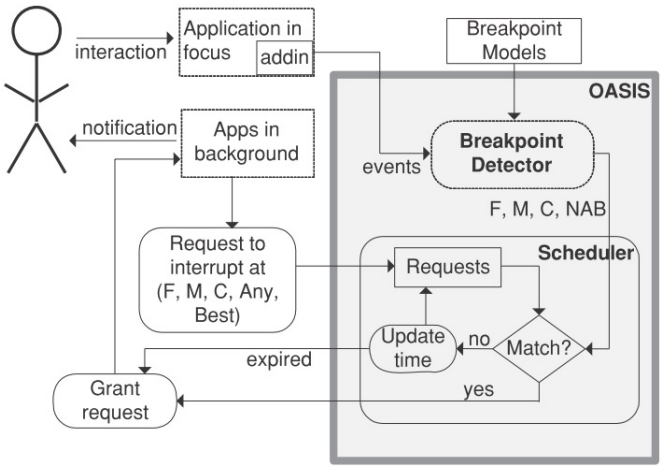


Abbildung 2.7: Überblick über die Komponenten (Shamsi T. Iqbal und Brian P. Bailey 2010)

[12] In der Arbeit wurden „Entscheidungsbäume“, „Bayes'sche Netze“, „Multilayer Perceptrons“ und „IB1“ als eingesetzte Lernalgorithmen genannt.

Unterbrechender

Anwendungen, die eine Benachrichtigung anzeigen wollen, sollen nun zunächst bei Oasis anfragen, wann die Benachrichtigung angezeigt werden darf. Neben der Granularität, zu der die Benachrichtigung angezeigt werden soll, kann auch eine Zeit angegeben werden, nach der die Benachrichtigung angezeigt werden muss. Erkennt Oasis eine Unterbrechung mit der angegebenen Granularität, benachrichtigt er die Anwendung, so dass diese die Benachrichtigung auf ihre Weise anzeigen kann.

Fazit

Oasis dient als Vermittler zwischen einem Arbeitenden und einem Unterbrechenden. Hierzu arbeiten einzelne Komponenten, die auf verschiedene Applikationen verteilt sind, mit einer Kernkomponente zusammen. Dieser verteilte Ansatz ist gut, aber in diesem Fall auf den Desktop des Anwenders beschränkt. Einflüsse, die von anderen Rechnern, wie zum Beispiel einem Ticketsystem auf einem Server, kommen, können nicht betrachtet werden. Die Architektur des in dieser Arbeit zu entwickelnden Frameworks bietet, durch die Verteilung der Komponenten über ein Netzwerk, deutlich mehr Möglichkeiten.

Die Auswertung der Daten zur Erkennung einzelner Unterbrechungspunkte ist bei Oasis sehr detailliert auf die jeweilige Applikation zugeschnitten. Externe Sensoren werden nicht weiter betrachtet. Stattdessen beschränkt sich das System auf die laufenden Applikationen auf der Desktopumgebung des Anwenders. Einflüsse, die von außerhalb dieses Kontextes kommen und sich zum Beispiel durch eine körperliche Reaktion des Anwenders äußern, können nicht betrachtet werden. Durch die angezielte Flexibilität des zu entwickelnden Frameworks der vorliegenden Arbeit soll es möglich sein, verschiedene Sensoren in Form von Input-Plug-ins zu entwickeln und diese mit dem Framework zu verbinden. Die gelieferten Daten könnten durch Aktor-Plug-ins weiter verarbeitet und zusammengefasst werden. Somit soll das in der vorliegenden Arbeit zu entwickelnde System in keiner Weise auf einen Kontext beschränkt, sondern hochflexibel einsetzbar sein. Dies hätte den Vorteil, dass jede denkbare Art eines Sensors einen Beitrag zur Erkennung des aktuellen Arbeitskontextes leisten könnte.

2.4.2 Stresserkennung bei Computernutzern mit Hilfe von psychophysiologischen Signalen

Jing Zhai und Armando Barreto beschäftigten sich mit der Erkennung von Stress mit Hilfe von psychophysiologischen Signalen bei Computernutzern. Ihre Arbeit basiert dabei auf der Erkenntnis, dass Reaktionen auf Stress immer im Zusammenhang mit Veränderungen im

vegetativen Nervensystem stehen. Hieraus resultieren körperliche Veränderungen, die mit entsprechenden Sensoren gemessen werden können. (A. B. Barreto und Zhai 2003)

Verwendete Sensoren

Um diese emotionale Situation am besten zu beschreiben, setzten Zhai und Barreto verschiedene Sensoren parallel ein. Dies waren zunächst Sensoren zur Messung des Hautleitwerts, des Blutvolumenpulses und des Pupillendurchmessers (A. B. Barreto und Zhai 2003). Später kam noch ein Sensor zur Messung der Körpertemperatur hinzu (Zhai, A. B. Barreto u. a. 2005). Der Hautleitwert, der Blutvolumenpuls und die Körpertemperatur wurden von der linken Hand abgenommen. Diese musste während des Tests still auf dem Tisch liegen.

Der Hautleitwert: Bereits kleine Veränderungen der Feuchtigkeit der Hautoberfläche haben Einfluss auf den elektrischen Widerstand der Haut. In Stresssituationen oder bei Nervosität fangen die Hände des Menschen an zu schwitzen. Hierdurch verändert sich der Hautleitwert. Durch das Anlegen eines geringen Stroms an zwei Fingern kann der Hautleitwert gemessen werden. (A. B. Barreto und Zhai 2003)

Der Blutvolumenpuls: Durch die Aktivierung des sympathischen Nervensystems verändert sich die Herzschlagrate und das Schlagvolumen des Herzens sowie die äußere kardiovaskuläre Resistenz. Über die Messung des Blutdurchflusses an einer Extremität, wie dem Fingerende, kann der Blutvolumenpuls gemessen werden. Hieraus ist es dann möglich, die Aktivität des sympathischen Nervensystems abzuleiten und somit auf Stress zu schließen. (A. B. Barreto und Zhai 2003)

Die Körpertemperatur: Die Körpertemperatur haben Zhai und A. B. Barreto mit einem Sensor an der Hautoberfläche gemessen. Sie stellten dabei die Vermutung an, dass die Temperatur am Finger im Falle einer Stresssituation abnimmt. Dies sei auf die Schweißbildung zurückzuführen, da diese den Körper kühle. Die Temperatur konnte sehr gut gemessen werden. (Zhai und A. B. Barreto 2006a)

Der Pupillendurchmesser: Das vegetative Nervensystem regt einige der Muskeln im Auge, ganz speziell den Muskel, der die Iris öffnet und schließt, an. Über die Veränderung der Pupillengröße kann auf den Zustand „Stress“ geschlossen werden. Den Pupillendurchmesser haben Zhai und Barreto über ein Eye-Gaze-System ermittelt. (A. B. Barreto und Zhai 2003)[13]

[13] siehe auch (A. Barreto, Gao und Adjouadi 2008; Beatty 1982; Hoeks und Levelt 1993; Shamsi T Iqbal, Zheng und Brian P Bailey 2004) sowie (Meißner 2013, Kap. 5.3)

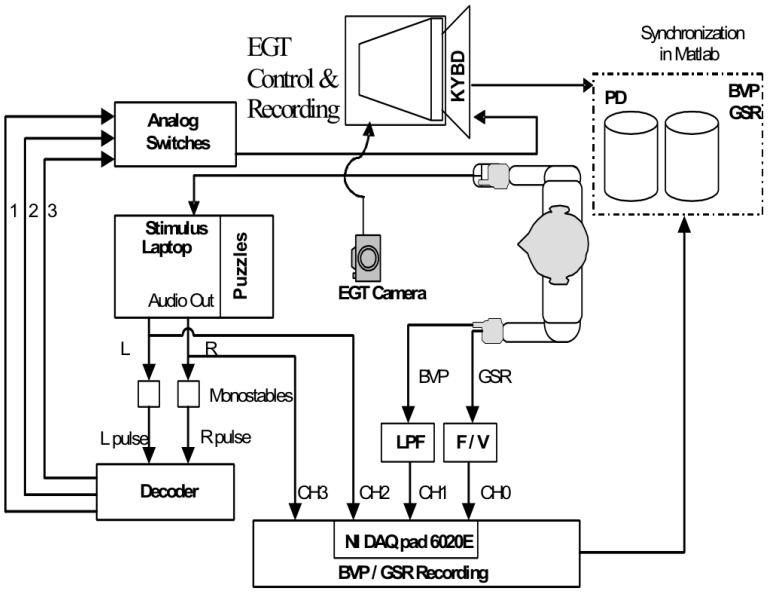


Abbildung 2.8: Aufbau der Testumgebung (Zhai, A. B. Barreto u. a. 2005)

Synchronisierung der Sensoren: In der Abbildung 2.8 ist der Aufbau der Testumgebung von A. B. Barreto und Zhai abgebildet. Hierauf ist zu sehen, dass der *Stimulus Laptop* über den Audioausgang mit den Aufnahmeeinheiten der Sensoren verbunden ist. Über diese Schnittstelle werden die Sensordaten mit dem aktuellen Stand des Spiels (*Puzzle*) synchronisiert. Der Laptop spielt an Schlüsselpunkten einen bestimmten Sinuston ab. Dieser indiziert einen bestimmten Zustand im Spiel, wodurch A. B. Barreto und Zhai später die Auswertungen der Daten vornehmen können. (A. B. Barreto und Zhai 2003)

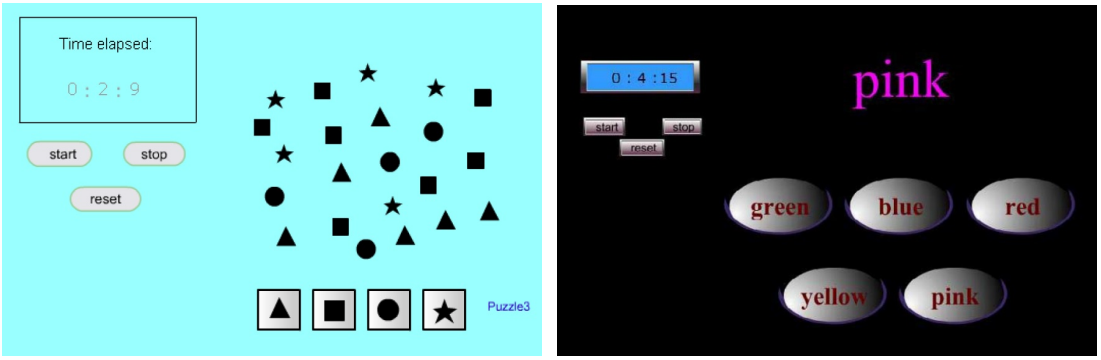
Testverfahren

Um ihr System zu testen, entwickelten Zhai und Barreto zwei verschiedene Spiele. Sie versuchten auf diesem Weg den Probanden gezielt unter Stress zu setzen.

Das erste Spiel ist in der Abbildung 2.9a abgebildet. Der Proband sollte hierbei für 30 Bilder jeweils angeben, welches Symbol am häufigsten vorkommt. Dazu klickte der Proband mit der Maus auf eines der Symbole am unteren Bildschirmrand. Hierzu hatte er je Bild nur fünf Sekunden Zeit. Um den Probanden zusätzlich unter Stress zu setzen, konnte man eine Highscore erreichen, bei der sowohl die benötigte Zeit wie auch die Anzahl der Fehler bewertet wurde. (A. B. Barreto und Zhai 2003)

Das zweite Spiel wurde dem Stroop Color-Word Interference Test (siehe Stroop 1935) nachempfunden. Bei diesem Test sah der Proband ein Wort, das eine Farbe beschreibt (zum Beispiel das Wort „rot“). Dieses Wort war in einer Farbe eingefärbt, die inkongruent zu der Bedeutung des Wortes stand (zum Beispiel: rot). Hierzu hatten Zhai und Barreto eine interaktive Version des Tests als Spiel implementiert (Abbildung 2.9b). Der Proband musste auf die richtig Antwort klicken und hatte je Aufgabe nur drei Sekunden Zeit. Bei diesem Spiel gab es drei sich wiederholende Abschnitte mit je vier Segmenten (siehe (Zhai, A. B. Barreto u. a. 2005)):

1. Einführungsphase
2. Übereinstimmende Phase: Hier stimmte die Einfärbung des Wortes mit der Bedeutung des Wortes überein.
3. Nicht-übereinstimmende Phase: Hier weichte die Einfärbung des Wortes von seiner Bedeutung ab. In dieser Phase entstand der Stroop-Effekt.
4. Ruhephase: Der Proband sollte sich wieder entspannen, um seinen emotionalen Zustand in den Ruhewert zu bewegen.



a: Spiel 1: 30 Puzzle (A. B. Barreto und Zhai 2003) b: Spiel 2: Der Stroop Test (Zhai, A. B. Barreto u. a. 2005)

Abbildung 2.9: Implementierte Spiele zur Erzeugung von Stress beim Probanden

Lernendes System

Affective Computing benötigt eine robuste und praktische Implementierung von emotionaler Erkennung. Um den emotionalen Zustand eines Probanden zu erkennen und wiederzuerkennen, haben Zhai, A. B. Barreto u. a. ein lernendes System auf Basis einer Support Vector Machine (SVM) entwickelt. (Zhai, A. B. Barreto u. a. 2005)

Die SVM wurde verwendet, um zwischen den Zuständen „Stressed“ und „Relaxed“ zu unterscheiden. Um die Maschine zu trainieren, wendeten Zhai und A. B. Barreto auf die Messdaten zunächst eine Merkmalsextraktion an. Sie unterschieden dabei zwischen den Stresssituationen (nicht übereinstimmende Phase im Spiel 2) und den normalen Zuständen (übereinstimmende Phase im Spiel 2). Für jede der zwei Gruppierungen konnten sie anhand der Daten elf Merkmale identifizieren. Diese sind in der Abbildung 2.10 zu sehen. Die Wertemengen wurden dann als Trainingsdaten für die SVM benutzt. Spätere Befragungen der Probanden zeigten, dass die Messergebnisse in starker Korrelation zu den empfundenen, emotionalen Zuständen der Probanden standen. (Zhai und A. B. Barreto 2006a,b)

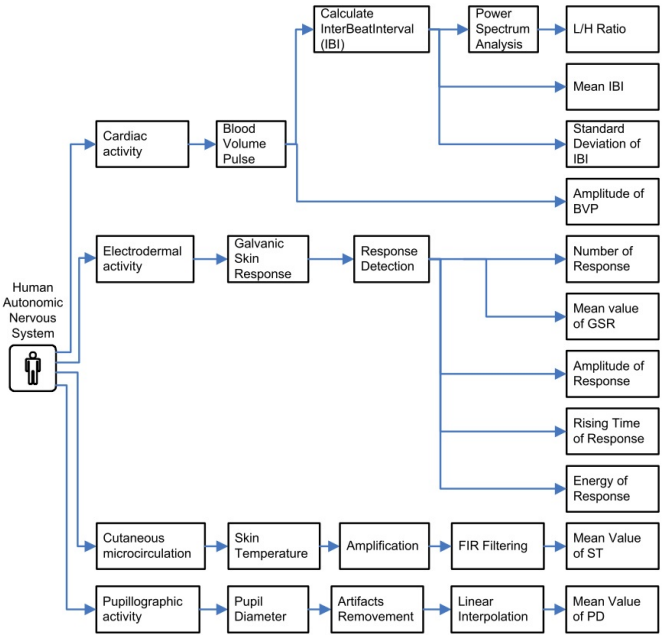


Abbildung 2.10: Feature Extraktion der physiologischen Merkmale (Zhai und A. B. Barreto 2006a)

Weitere Erkenntnisse

Im Kontext der Mensch-Computer-Interaktion ist der empfundene Stress meistens mental und mäßig in seiner Intensität. Um dennoch verlässliche Aussagen über den emotionalen Zustand einer Person treffen zu können, müssen die Schlüsselsensoren genauer bestimmt werden. Zhai und A. B. Barreto haben gezeigt, dass der Pupillendurchmesser solch eine Schlüsselrolle bei der Erkennung von Stresszuständen darstellt. Durch die Abschaltung der Erkennung des

Pupillendurchmessers sank die Erkennungsrate um ca. 30%. Bei allen anderen eingesetzten Sensoren war die Auswirkung wesentlich geringer. (Zhai und A. B. Barreto 2006a)

Fazit

Zhai und A. B. Barreto haben sich in ihrer Arbeit mit der Erkennung des emotionalen Zustands „Stress“ beschäftigt. Sie haben dabei ein System entwickelt, mit dem sie Stresssituationen verlässlich von Nicht-Stresssituationen unterscheiden können. Hierzu setzten sie verschiedene Sensoren ein, die allerdings die Bewegungsfreiheit des Benutzers einschränkten.[14] Zhai und A. B. Barreto gingen dabei nicht weiter auf die Echtzeitfähigkeit ihres Systems ein. Außerdem fehlt eine Beschreibung für die Möglichkeiten der Erweiterung ihres Systems um weitere Sensoren. Zusätzlich erscheint die Konstruktion ihres Systems noch nicht vollständig ausgereift. Zum Beispiel erfolgt die Synchronisierung der Sensoren mit dem Testspiel auf dem Laptop über die Ausgabe von Sinustönen. Besser wäre hier eine direkte Ausführung einer Prozedur, um so die einzelnen Abschnitte des Tests anzukündigen.

Die Ergebnisse der Arbeit von Zhai und A. B. Barreto konnten zeigen, dass der Pupillendurchmesser eines der wichtigsten Merkmale zur Erkennung des emotionalen Zustandes „Stress“ ist.

Ein wichtiger Unterschied zwischen der Architektur ihrer Arbeit und der geplanten Architektur des Frameworks der vorliegenden Arbeit ist die Anbindung der Sensoren. Sämtliche Sensoren sind bei Zhai und A. B. Barreto mit einem einzigen Computer verbunden. Dieser empfängt die Rohmessdaten der Sensoren und muss diese dann entsprechend verarbeiten. Dies könnte, bei einer Echtzeitauswertung der Daten, zu Performanzproblemen führen.

In dem zu entwickelnden Framework aus der vorliegenden Arbeit kann jeder Sensor an einen einzelnen, unabhängigen Computer angeschlossen sein. Dieser unabhängige Computer stellt dann selbstständig eine Verbindung zur Kommunikationsschnittstelle des Frameworks her. Somit wird die Verarbeitung der Rohmessdaten, und damit auch die Rechenlast, auf mehrere Computer verteilt.

2.4.3 Home Office 2.0 Projekt

Karsten Panier hat sich in seiner Arbeit *Home Office 2.0* mit der Problematik der Wissensarbeit im Bereich „Home Office 2.0“ befasst. Er stellte dabei den Informationsaustausch beim Arbeiten in verteilten Teams in den Vordergrund. Verteilte Teams benötigen die Informationen, die direkt

[14] Andere Arbeiten zeigen, dass die Bewegungsfreiheit eine große Rolle spielt, wenn die Sensoren im Alltag getragen werden sollen (Ferreira 2008; Ferreira u. a. 2008; Holmquist u. a. 2007; P. Sanches 2008; Pedro Sanches, Höök u. a. 2010; Pedro Sanches, Vaara u. a. 2010; Vaara, Höök und Tholander 2009; Vaara, Silvşan u. a. 2010).

mit dem Projekt zusammenhängen, zur gleichen Zeit an unterschiedlichen Standorten. Diese Informationen werden als „das Wissen“ des Projektteams angesehen. Liegen die Informationen in digitaler Form vor, können alle Wissensarbeiter des Projektes auf das Wissen zugreifen.

Ein großes Problem bei der Wissensarbeit ist, dass durch Unterbrechungen ca. 20% - 40% der Produktivität verloren gehen, da sich der Wissensarbeiter erneut in die Aufgabe einarbeiten muss. Hilfe gegen dieses Problem können kontextsensitive Anwendungen bieten. Sie erkennen Informationen, die zu einem Kontext gehören, und versuchen diese Informationen zusammenzufassen. Es kann jedoch kein globales System geben, das alle Probleme bei der Wissensarbeit behebt, da je nach Einsatzzweck ein unterschiedlicher Informationsbedarf besteht.

Karsten Panier hat sich im weiteren Verlauf seiner Arbeit auf die Wissensarbeit fokussiert und die entstehenden Geschäftsprozesse sowie die technische Infrastruktur der Projektarbeit in den Hintergrund rücken lassen.[15] Ein von ihm behandeltes Problem war, dass in einem Aufgabenkontext häufig zu viele Informationen auf einmal dargestellt werden, was unweigerlich zu einem *Information overload* führt. Die Folge davon ist häufig die kognitive Überforderung (siehe auch Kapitel 2.2.5 sowie Kirsh 2000). Er stellte im Folgenden dar, dass unterstützende Systeme, die in der Lage sind den Aufgabenkontext des Anwenders zu erfassen und zu speichern, bei der Wissensarbeit extrem hilfreich sein können. Informationen, die zum Aufgabenkontext gehören, sind laut Kirsh zum Beispiel:

- Die vom Benutzer verwendeten Dokumente.
- Die ausgeführten Aktionen des Benutzers.
- Die vom Benutzer verwendeten Werkzeuge und Anwendungen.
- Die Ziele der Aufgabe.
- Die zeitliche Reihenfolge der Interaktionen.
- Das *Socio-Technical Network*, welches die Arbeitsbeziehungen zwischen den Mitarbeitern auf Basis von Konfigurationsmanagement-Werkzeugen aufbaut.

Einige dieser Informationen, wie zum Beispiel die Information welche Dokumente der Benutzer angefasst hat, lassen sich sehr einfach verfolgen. Andere Informationen, wie die Ziele der Aufgabe, sind hingegen nur sehr schwer automatisch zu erkennen. (Holsten u. a. 2010; Panier 2010)

[15] Karten Panier hat seine Arbeit unter anderem im Bereich eines virtuellen *Project Office* weiter ausgebaut (Panier 2011).

Fazit

Die Wissensarbeit erfordert einen erhöhten Konzentrationsbedarf. Ablenkungen sollten bei der Wissensarbeit in jedem Fall vermieden werden, da sonst massiver Produktivitätsverlust auftreten kann. Systeme, die die Konzentration des Arbeitenden fördern und Ablenkungen vermeiden, können dem Produktivitätsverlust entgegen wirken. Hier setzt das geplante System aus der vorliegenden Arbeit an und erweitert die Sichtweise auf das Problem. Das zu entwickelnde Framework soll die Überwachung des Kontextes auch außerhalb der Desktopumgebung des Arbeitenden ermöglichen. Hierdurch könnten weitere Ablenkungsfaktoren erkannt und Präventionsmaßnahmen eingeleitet werden.

Der von Karsten Panier beschriebene Aufgabenkontext könnte als eine Komponente des zu entwickelnden Frameworks realisiert werden. Diese Komponente würde Funktionen zur Erstellung des aktuellen Kontextes bereitstellen. Einerseits könnte der Aufgabenkontext Informationen zum aktuellen Zustand des Benutzers liefern. Andererseits wäre eine Reaktion auf Störungen denkbar. Hierzu könnten der Aufgabenkontext analysiert und die Informationen, die nicht zum Aufgabenkontext gehören, gezielt ausgeblendet werden.

Weitere Ansätze zur Prävention vor und Unterstützung bei Kontextwechseln können in (Kersten 2007) und (Voida und Mynatt 2009) nachgelesen werden.

2.5 Anforderungen und Ziele

Um den Rahmen der vorliegenden Arbeit näher zu spezifizieren, wurden aus den genannten Szenarien und den themennahen Arbeiten Anforderungen an das zu entwickelnde System definiert. Diese Anforderungen und die dazu gehörenden Ziele sollen folgend vorgestellt werden.

Verwendung verschiedener Sensorarten: Durch die unterschiedlichen Anforderungen, die sich aus den vorgestellten Szenarien ergeben, muss das Framework maximal flexibel bleiben. Es soll möglich sein, ein sehr breites Spektrum an Sensoren an das Framework anzubinden. Die Anbindung soll zusätzlich unabhängig von der präferierten Programmiersprache des APIs des Sensors sein. Als Sensor wird hierbei alles angesehen, was Hardware und Software vereint, auf diese Weise Messdaten erfasst und diese Messdaten in standardisierter Form den anderen Komponenten des Frameworks bereitstellt. Ein Sensor kann somit sowohl eine Hardwarekomponente sein, die den Puls des Anwenders misst, als auch eine Softwarekomponente, die den aktuellen Zustand eines Bugtracking-Systems auswertet.

Das Ziel ist es also, die Architektur des Frameworks so zu entwerfen, dass Sensoren unabhängig von ihren eigenen Implementationsvoraussetzungen an das Framework angebunden werden können.

Plug-ins sind über das Netzwerk angebunden: Es können neben den Sensoren weitere Komponenten hinzukommen, die keine Sensoren, sondern zum Beispiel Aktoren sind. Zur Vereinfachung werden alle Komponenten des Frameworks, die eine bestimmte Funktionalität bereitstellen, unter dem Begriff *Plug-ins* zusammengefasst.

Alle diese unterschiedlichen Plug-ins müssen mit den anderen Komponenten des Frameworks kommunizieren können. Außerdem werden die meisten Plug-ins mehr oder weniger komplexe Berechnungen durchführen, um beispielsweise den aktuell zu ermittelnden Zustand aus den Rohmessdaten zu ermitteln. Die Rechenlast kann durch die verteilte Architektur auf mehrere Rechner ausgelagert werden. Hierfür wird eine Schnittstelle entworfen, die die Kommunikation der Plug-ins mit den anderen Komponenten sicherstellt.

Das Framework soll also eine Kommunikationsschnittstelle bereitstellen, die es den Plug-ins ermöglicht, über ein Netzwerk miteinander zu kommunizieren.

Verschlüsselung: Manche Daten sind privat, wie zum Beispiel ein Passwort. Diese Daten müssen vor ungewollten Zugriffen geschützt werden. Daher sollte es möglich sein, Nachrichteninhalte zu verschlüsseln, bevor Sie über das Netzwerk verschickt werden.

Das Ziel ist also, eine standardisierte Form der Verschlüsselung anzubieten, mit der auf einfache Weise die Daten vor dem Versenden der Nachricht geschützt werden können.

Lose Kopplung: Sämtliche Komponenten des Frameworks sollten unabhängig voneinander funktionieren. Diese Anforderung trifft besonders auf die Plug-ins zu, die im Netzwerk verteilt sind. Mit Hilfe der losen Kopplung soll es möglich sein, die einzelnen Komponenten zur Laufzeit einzubinden, abzuschalten oder auszutauschen. Es muss also bei der Implementierung darauf geachtet werden, dass einzelne Plug-ins keine Abhängigkeiten von anderen Plug-ins haben. Im Kontext des Frameworks bedeutet Abhängigkeit, dass ein Plug-in A ohne die Aktivität eines Plug-ins B nicht ausführbar wäre.

Dieses Ziel kann nicht einfach durch die Architektur des Frameworks erfüllt werden. Jeder Entwickler, der ein Plug-in oder eine andere Komponente für das Framework entwickelt, muss darauf achten, dass seine Komponente zu den anderen Komponenten des Frameworks eine lose Kopplung aufweist.

Freie Konfigurierbarkeit: Aufgrund der Unabhängigkeit der Komponenten kann jede Komponente für sich frei konfiguriert werden. Da die einzelnen Plug-ins im Netzwerk verteilt liegen, muss dem Anwender eine einheitliche Schnittstelle zur Konfiguration angeboten werden. Jedes Plug-in soll dazu in der Lage sein, über die angebotene Netzwerkschnittstelle einen Menüeintrag anzulegen, worüber ein Konfigurationsdialog angezeigt werden kann.

Ein weiteres Ziel besteht also darin, den Komponenten des Frameworks eine Schnittstelle zur Kommunikation mit dem Benutzer anzubieten.

Persistierung: Zur Bestimmung bestimmter Fakten ist es notwendig, Daten in einem Verlauf zu betrachten. Nur so ist es möglich eine Entwicklung von Daten festzustellen. Betrachtet man beispielsweise einen Börsenkurs, so sagt dieser nur etwas aus, wenn man den Verlauf des Kurses einsehen kann. So kann direkt eine Aussage darüber getroffen werden, ob der Kurs steigt oder fällt. Hat man hingegen nur einen einzelnen Wert zur Verfügung, kann man diese Aussage nicht treffen.

Damit eine Software einen Datenverlauf auswerten kann, müssen die Daten gespeichert werden und von der Software abrufbar sein. Diese Persistierung der Daten kann zum Beispiel mit Hilfe einer Datenbank erfolgen. Das Ziel ist es also, dass das Framework eine Persistenzschicht anbietet.

Ableitung der benötigten Komponenten

In den Szenarien (siehe Kapitel 2.3) wurden sehr unterschiedliche Einsatzzwecke dargestellt. Diese sollten zeigen, wie flexibel das zu entwickelnde Framework sein muss. Da in dieser Arbeit nicht auf alle möglichen Einsatzszenarien eingegangen werden kann, liegt bei weiteren Erläuterungen in dieser Arbeit die Konzentration auf dem *Szenario 2*.

In den folgenden Absätzen soll ein erster Überblick über die Architektur des Frameworks gegeben werden. Zunächst soll an dieser Stelle aber noch der Aufbau des *Szenarios 2* erweitert werden:

> Carol und ihrem Team stehen nun Software-Komponenten zur Verfügung, die automatisch versuchen ein individuelles Profil pro Proband zu erstellen. Diese Komponenten sind in einem Subsystem, dem **Profiling-Subsystem**, gebündelt. Durch das Profiling soll ein System entstehen, das sich automatisch an die Individualität des jeweiligen Benutzers anpassen kann. Wichtig für das Profiling ist die Ermittlung von vielen Daten über die Zeit. So kann beispielsweise anhand des zeitlichen Verlaufs der körperlichen Reaktionen ein Profil des Probanden erstellt werden. Damit eine Betrachtung von Daten über die Zeit überhaupt möglich wird,

müssen die Daten gespeichert werden. Hierzu wurde eine Persistenzschicht an das Profiling-Subsystem angebunden. Ebenfalls wichtig für die Erstellung eines Profils sind die aktuellen Einflüsse, die aus der Umgebung auf den Benutzer wirken. Hierzu sollen Sensoren eingesetzt werden, die den Kontext der Umgebung erfassen. Diese **Kontextinformationen** könnten zum Beispiel das aktuelle Wetter, die Tageszeit, die Umgebungstemperatur und die Umgebungsluftfeuchtigkeit sein. Sie werden auf der netzwerkbasierten Kommunikationsschnittstelle bereitgestellt.

Um zusätzlich persönliche Daten beim Benutzer abfragen zu können, ist eine **Benutzerinteraktionsschnittstelle** vorgesehen. Hierüber kann der Benutzer auf Nachfrage persönliche Daten, wie beispielsweise seine berufliche Erfahrung, seine berufliche Verantwortung, seine berufliche Position, sein Alter, sein Geschlecht oder Anmeldeinformationen für externe Systeme an das Framework übermitteln. Da einige dieser Daten, zum Beispiel Passwörter, geheim sind und nur einer bestimmten Komponente des Systems zugänglich sein sollen, muss es eine Möglichkeit geben, diese Daten zu verschlüsseln. Die Verschlüsselung an sich und der Schutz der Privatsphäre des Benutzers sollen allerdings nicht Bestandteil dieser Arbeit sein. Zum Einstieg in dieses Thema sei auf die Arbeit (Kobsa und Schreck 2003) verwiesen.

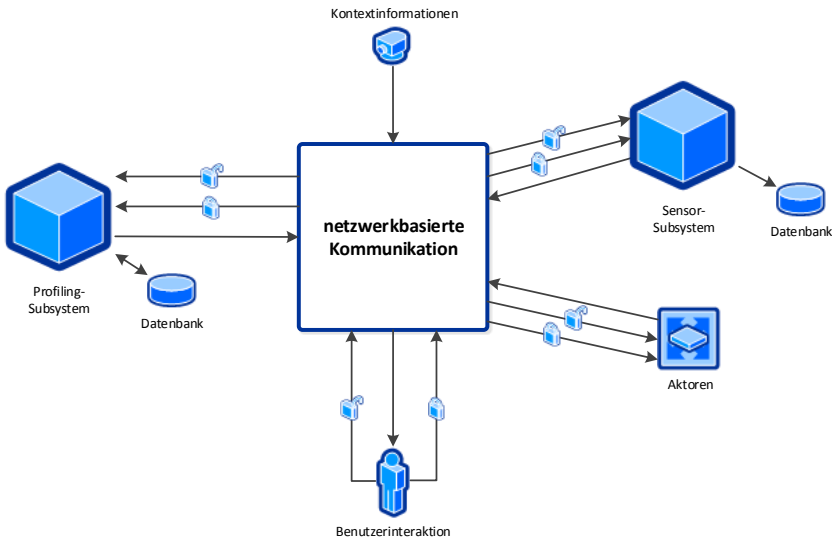


Abbildung 2.11: Überblick über die Architektur des Frameworks mit potentiellen Komponenten

In der Abbildung 2.11 ist eine beispielhafte Architekturübersicht aus den Anforderungen und Zielen des *Szenarios 2* dargestellt. Hier sind einzelne Komponenten und komplette Subsysteme[16] abgebildet. Der Kern der Architektur des Frameworks ist die **netzwerkbasierte Kommunikationsschnittstelle**. Hierüber können alle Komponenten des Frameworks miteinander kommunizieren. Damit wirklich alle Komponenten Zugriff auf diese Daten haben, darf es keine Ende-zu-Ende-Verbindung zwischen einzelnen Komponenten geben. Viel mehr sollte hier die Möglichkeit eines Art globalen Speichers geschaffen werden. Darum soll die netzwerkbasierte Kommunikationsschnittstelle auf einer *Blackboard-Architektur* basieren. Bezogen auf das *Szenario 2* bedeutet dies, dass hier alle Sensoren ihre Messdaten den anderen Komponenten zur Verfügung stellen.

Zur Messung der körperlichen Reaktionen setzten Carol und ihr Team verschiedene Sensoren ein. Diese lieferten Messdaten, die von den anderen Komponenten des Systems ausgewertet und zur späteren Einsicht gespeichert werden sollten. Für das Framework kann hier ein **Sensor-Subsystem** entwickelt werden, das sämtliche Sensoren bündelt. Hieran müsste eine Datenbank angeschlossen sein, in der die Messdaten zur späteren Verwendung gespeichert werden. Die Entwicklung eines solchen Subsystems liegt allerdings außerhalb des Rahmens dieser Masterarbeit.

Während des Testverlaufs werden die Probanden immer wieder unterbrochen. Diese Unterbrechungen sollen kontrolliert und nachvollziehbar durchgeführt werden. **Aktoren**, die an das Framework angeschlossen werden, könnten so konfiguriert sein, dass sie zu bestimmten Zeitpunkten durch unterschiedliche Ablenkungsfaktoren den Probanden gezielt versuchen zu unterbrechen. Zusätzlich könnten die Aktoren die Ablenkungsfaktoren auf den aktuellen Zustand des Probanden abgleichen, um so noch gezielter Ablenkungen zu produzieren. Hierzu greifen die Aktoren auf die vorhandenen Messdaten zu und werten diese für sich aus.

Die hier beschriebenen Komponenten stellen lediglich ein mögliches Einsatzszenario dar und sollen ein Beispiel für die Flexibilität des Frameworks darstellen. Das Framework stellt dabei die Kommunikationsschnittstelle, die Persistierung und verschiedene Interaktionsmöglichkeiten für den Benutzer bereit. Alle weiteren Komponenten, die in einem bestimmten Szenario verwendet werden sollen, können als Plug-ins an das Framework angebunden werden. In dem folgenden Kapitel werden die Kernkomponenten des Frameworks genauer beschrieben.

[16] Subsysteme eignen sich zur Integration komplexer Komponentenstrukturen, die eine bestimmte Gesamtfunktionalität anbieten. Für das Framework und die anderen Komponenten des Frameworks erscheint das komplexe Subsystem allerdings wie eine einfache Komponente. Das Subsystem selbst kann aus beliebig vielen Komponenten bestehen, die alle einzeln mit der Kommunikationsschnittstelle kommunizieren können.

3 Design

In diesem Kapitel wird die Architektur des zu entwickelnden Frameworks beschrieben. Zunächst gibt es eine kurze Einführung, was ein *verteiltes System* ist und wie dieses Modell auf das zu entwickelnde Framework zu übertragen ist. Hieraus ergibt sich die Einführung in den Begriff *Middleware* und die Wahl des Modells, das in dem Framework eingesetzt werden soll. Es folgt eine Einführung in die Begriffe *Complex Event Processing* und *Blackboard* sowie eine Herstellung des Bezugs dieser Begriffe zu dem Framework.

3.1 Verteiltes System

In dieser Arbeit wird die Definition von A. Tanenbaum und Steen für ein verteiltes System verwendet. Diese lautet:

> „Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen." (A. Tanenbaum und Steen 2003, S. 18)

Aus dieser Definition gehen zwei wichtige Punkte für ein verteiltes System hervor. Zunächst basiert das verteilte System auf *voneinander unabhängigen Computern*. Diese müssen also eigenständig arbeiten und dürfen sich nicht gegenseitig beeinflussen. Wenn ein Computer ausfällt, kann das verteilte System trotzdem und ohne Funktionseinbußen weiterarbeiten. Der zweite Punkt ist die *Sicht des Benutzers* auf das System. Der Benutzer soll ein System sehen, das wie ein einziges (lokales) System aussieht. Die Applikation, die auf dem System läuft, verrät also nichts über die eigentliche (verteilte) Architektur des Systems.

Zur Realisierung dieser Anforderungen wird häufig eine Softwareschicht in die Architektur des Systems eingefügt. Diese Schicht steht zwischen der Applikationsschicht und der Hardwareschicht (siehe Abbildung 3.1). Sie wird Middleware genannt.

In der vorliegenden Arbeit wird eine verteilte Architektur gewählt, da nur so alle Komponenten des Systems unabhängig voneinander auf unterschiedlichen Computern laufen können. Die Komponenten, die es in dem zu entwickelnden Framework geben soll, sind die Kommunikationsschnittstelle, der Datenspeicher zur Persistierung der Tupel, die Systemteile

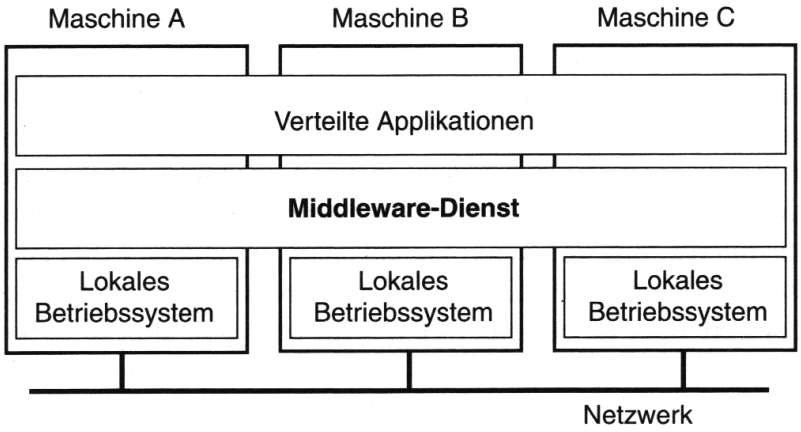


Abbildung 3.1: Einordnung der Middleware-Schicht (A. Tanenbaum und Steen 2003, S. 19)

zur Benutzerinteraktion und die Plug-ins. Der Einsatzzweck des Frameworks wird durch die eingesetzten Plug-ins definiert. Diese bilden letztlich die funktionalen Komponenten, um zum Beispiel Ablenkungsfaktoren zu erkennen.

Ein erster Architekturentwurf des zu entwickelnden Frameworks ist in der Abbildung 3.2 dargestellt. Das Modell entspricht dabei nicht mehr ganz dem, welches A. Tanenbaum und Steen aufgestellt haben (vergleiche Abbildung 3.1). In der Architektur des zu entwickelnden Frameworks besteht die verteilte Applikation aus der Middleware, der zentralen Steuerungseinheit und den verschiedenen Plug-ins, die auf unterschiedlichen Maschinen laufen. Auf einer Maschine können eine einzelne oder mehrere Komponenten laufen. Im Gegensatz zum herkömmlichen Modell eines verteilten Systems erleidet das zu entwickelnde Framework eine Funktionseinbuße sobald ein Plug-in ausfällt. Die Funktion des Plug-ins steht dem Framework dann nämlich nicht mehr zur Verfügung. Das liegt daran, dass jedes Plug-in für eine bestimmte Aufgabe zuständig ist und jedes Plug-in auf nur einer Maschine laufen soll und somit nicht repliziert wird. Es kann also keine Ausfallsicherheit bezüglich einzelner Plug-ins geben. Dies bedeutet allerdings nicht, dass das zu entwickelnde Framework an Stabilität verliert. Denn die Kernkomponenten sollen auch ohne die Plug-ins lauffähig sein und von dem Ausführungszustand einzelner Plug-ins nicht beeinflusst werden.

Die verteilte Applikation verbirgt sich hinter der zentralen Steuerungseinheit, die als Schnittstelle zum Benutzer dienen soll. Die zentrale Steuerungseinheit wird auf einer einzelnen Maschine, dem Computer des Benutzers, ausgeführt. Sie kommuniziert dann im Hintergrund mit der Middleware, die irgendwo im Netzwerk liegen und die Anbindung der verteilten Komponenten ermöglichen wird. Für den Benutzer sieht das gesamte System letztlich aus, als

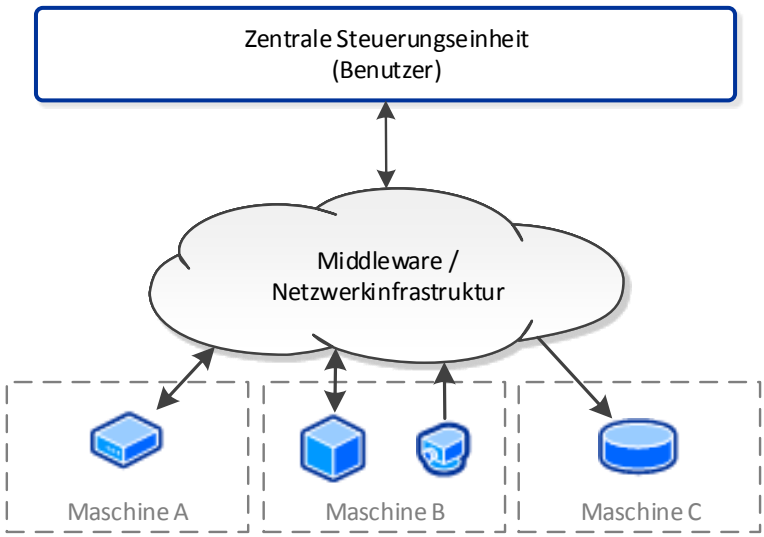


Abbildung 3.2: Architekturübersicht des zu entwickelden Frameworks

würde es auf seinem Computer laufen, obwohl es im Netzwerk auf unterschiedliche Maschinen verteilt ist.

3.2 Middleware

Eine Middleware ist eine Softwareschicht, die die technischen Details des Systems vor der Applikation verheimlicht. Die Middleware muss hierzu verschiedene Schnittstellen anbieten, damit die Applikation sämtliche Funktionen des Systems verwenden kann. Für die Applikation ist es dabei nicht von Relevanz, welche Hardware oder welches Betriebssystem von den einzelnen Komponenten eingesetzt wird.

A. S. Tanenbaum stellt in seinem Buch *Moderne Betriebssysteme* verschiedene Arten von Middleware vor. Die Beschreibung der Middleware-Arten sowie die Auswahl einer Art für das zu entwickelnde Framework wird in den folgenden Absätzen durchgeführt.

Dokumentenbasierte Middleware: Die erste Art von Middleware ist die dokumentenbasierte Middleware. Dabei ist alles, was vom Benutzer verwendet wird, ein Dokument. Einzelne

Dokumente können auf andere Dokumente verweisen. Hierdurch entsteht ein großer, gerichteter Graph.

Ein Beispiel für eine dokumentenbasierte Middleware ist das World Wide Web (WWW). Jeder Rechner im Netzwerk kann einzelne Dokumente anbieten, die sogenannten Webseiten. Diese Dokumente können Verweise (Hyperlinks) auf andere Dokumente beinhalten. Zur Anzeige der Dokumente wird der Webbrowser verwendet.

Das WWW ist ein klassisches Client-Server-System. Der Client, dargestellt durch den Webbrowser, fordert die Dokumente beim Server an und lädt sie herunter. (A. S. Tanenbaum 2009, S. 684f)[17]

Dateisystembasierte Middleware: Ein ähnliches Modell stellt die dateisystembasierte Middleware dar. Dabei wird ein globales Dateisystem gebildet, das tatsächlich im Netzwerk verteilt ist. Der Benutzer kann die Verteilung im Netzwerk aber nicht sehen, da ihm das Dateisystem durch die Middleware wie ein lokales Dateisystem dargestellt wird.

Bei dieser Art von Middleware kommen verschiedene Übertragungsmodelle in Frage. Zum Einen kann die Übertragung durch ein **Upload/Download-Modell** realisiert werden. Die Vorteile dabei sind die einfache Realisierung und der Geschwindigkeitsvorteil. Der Geschwindigkeitsvorteil ergibt sich allerdings nur bei der Übertragung von ganzen Dateien. Das Problem ist jedoch, dass bei der Anforderung einer Datei der lokale Speicher ausreichen muss, um die Datei zu speichern. Außerdem kann die Übertragung von großen Dateien sehr lange dauern. Möchte man zum Beispiel nur einen kleinen Teil einer Datei bearbeiten, ist dieses Vorgehen sehr ineffizient. Neben diesen Problemen kann es noch Schwierigkeiten bei der Konsistenz geben, sobald mehrere konkurrierende Benutzer auf einer Datei arbeiten wollen.

Ein anderer Weg zur Übertragung von Daten ist hier das **Remote-Access-Modell**. Die Dateien werden dann nicht mehr übertragen, sondern durch verschiedene Techniken direkt an ihrem Speicherort bearbeitet. (A. S. Tanenbaum 2009, S. 686ff)[18]

Objektbasierte Middleware: Im Gegensatz zu den zwei vorangegangenen Middleware Arten wird nun alles als Objekt angesehen. Der Unterschied besteht darin, dass ein Objekt aus Variablen und Prozeduren besteht. Auf die Variablen darf aber nur per Prozedur zugegriffen werden.

Die Middleware ist in diesem Fall für die Kommunikation zwischen dem anfragenden Prozess und dem betreffenden Objekt zuständig. Ruft ein Prozess eine Prozedur eines Objekts auf,

[17] für weitere Details siehe auch (A. Tanenbaum und Steen 2003, S. 57)
[18] für weitere Details siehe auch (A. Tanenbaum und Steen 2003, S. 56f, S. 645ff)

baut die Middleware die Client-Server-Kommunikation auf und ermittelt dabei den Ort des Servers, auf dem das Objekt liegt. Ein Beispiel für diese Art von Middleware ist *CORBA*. (A. S. Tanenbaum 2009, S. 691f)[19]

Koordinationsbasierte Middleware: Das Prinzip einer koordinationsbasierten Middleware ist die Bereitstellung eines Services zur Kommunikation und Synchronisation von Informationen zwischen Prozessen. Angestoßen wurde das Prinzip von einem akademischen Versuchsprojekt Namens **Linda**.

David Gelernter hatte 1985 die Idee eine Kommunikationsgrundlage für Prozesse zu schaffen. So könnten die Prozesse miteinander kommunizieren ohne identifizierende Informationen voneinander zu haben. Die Prozesse A und B könnten also sowohl zeitlich, als auch räumlich völlig unabhängig voneinander existieren, aber mit Hilfe des Systems von Gelernter Informationen austauschen. Mit seinem Studenten Nick Carriero entwickelte er *Linda*, ein neuartiges System zur Kommunikation und Synchronisation. *Linda* kann als Programmiersprache für die verteilte Programmierung angesehen werden. Hierzu werden nachfolgend beschriebene Operationen in eine vorhandene Programmiersprache integriert. Der Programmierer muss sich dann nicht mehr mit den technischen Details, wie zum Beispiel der Hardware oder der Verteilung der Daten, beschäftigen. Er kann die vorhandenen Operationen nutzen und so ohne weiteres Zutun seine Software verteilt implementieren. (Carriero und Gelernter 1986, 1989; Gelernter 1985)

Die Kommunikation zwischen den Prozessen erfolgt bei *Linda* über einen Tupelraum. Jeder Prozess kann Tupel zum Tupelraum hinzufügen oder aus dem Tupelraum entfernen. Der Tupelraum selbst ist global und somit für alle Prozesse gleich. Er sieht für die Prozesse wie ein globaler Speicher aus, auf den alle Prozesse zugreifen können.

Zur Interaktion mit dem Tupelraum gibt es vier Grundoperationen. Zunächst gibt es eine Operation zum *Hinzufügen eines Tupels* zum Tupelraum. Das gleiche Tupel kann dabei auch mehrfach zum Tupelraum hinzugefügt werden. Es existieren dann mehrere Kopien des Tupels im Tupelraum. Die zweite Operation *entfernt ein Tupel* aus dem Tupelraum. Diese ist eine besondere Art des Lesens. Sie wird Konsumieren genannt, da das Tupel nach dem Lesen nicht mehr im Tupelraum vorhanden ist. Andere Prozesse haben bei dieser Art des Lesens keinen Zugriff mehr auf das Tupel. Eine weitere Operation bietet ebenfalls das *Lesen eines Tupels* an, wobei hier das Tupel nicht aus dem Tupelraum entfernt wird. Beide Operationen zum Lesen von Tupeln blockieren die Ausführung des lesenden Prozesses. Es ist also ein synchroner Funktionsaufruf. Der Prozess bleibt dabei solange blockiert, bis das angeforderte

[19] für weitere Details siehe auch (A. Tanenbaum und Steen 2003, S. 57, S. 554ff)

Tupel verfügbar ist und gelesen werden konnte. Die letzte der vier Operationen erlaubt es, direkt im Tupelraum *Berechnungen ausführen zu lassen.* Das Ergebnis der Berechnung wird als neues Tupel im Tupelraum veröffentlicht.

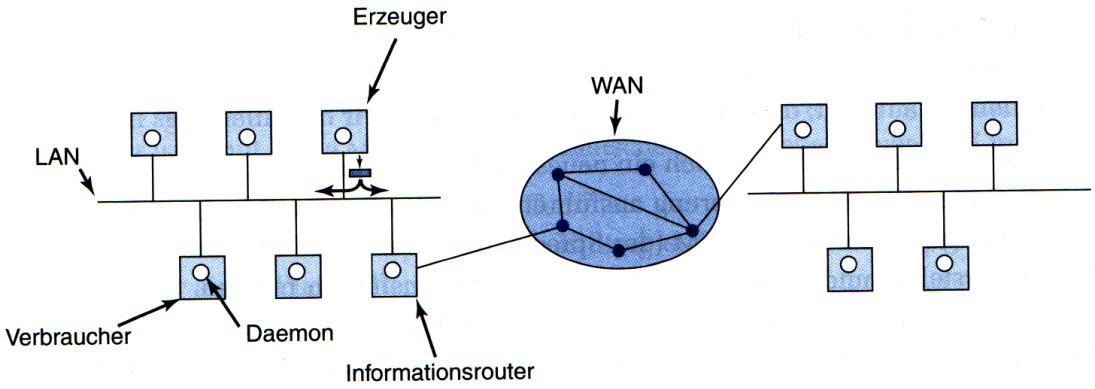


Abbildung 3.3: Publish/Subscribe-Modell (A. S. Tanenbaum 2009, S. 695)

Durch *Linda* inspiriert wurde das **Publish/Subscribe-Modell** entwickelt. Es wurde 1993 von Oki u. a. in »The Information Bus: an architecture for extensible distributed systems« (Oki u. a. 1993) eingeführt. Das **Publish/Subscribe-Modell** beschreibt eine Middleware in einem Netzwerk, über die Informationen zwischen Erzeugerprozessen und Verbraucherprozessen ausgetauscht werden können. Jeder Prozess im Netzwerk kann dabei Erzeuger, Verbraucher oder beides sein. A. S. Tanenbaum hat in seinem Buch *Moderne Betriebssysteme* (A. S. Tanenbaum 2009) zur Verdeutlichung des Modells die Abbildung 3.3 angeführt. Sie zeigt alle wichtigen Teile des Modells, inklusive der Anbindung anderer Netze über ein WAN. Möchte nun ein Erzeuger eine neue Information im Netzwerk bereitstellen, verpackt er die Information in einem Tupel. Das Tupel besitzt im Header eine hierarchisch aufgebaute Betreffzeile. Die einzelnen Teile der Betreffzeile sind durch Punkte getrennt. Insgesamt gibt die Betreffzeile das Themengebiet an, zu dem die Information gehört. Dieses Tupel publiziert der Erzeuger dann im Netzwerk (**Publishing**). Mit Hilfe der Betreffzeile können die einzelnen Tupel gefiltert werden. Hierdurch ist es möglich, dass ein Verbraucher ein bestimmtes Thema abonniert (**Subscribing**). Ein Tupel-Daemon-Prozess empfängt dafür alle im Netzwerk publizierten Tupel und filtert sie anhand der Betreffzeile und des vorgegebenen Themas. Im *Publish/Subscribe-Modell* können verschiedene Arten von Semantiken eingesetzt werden. Die wichtigsten sind die garantierte und die zuverlässige Zustellung. Bei der *garantierten Zustellung* ist es wichtig, dass die Tupel auch nach einem Absturz noch an die Subscriber ausgeliefert werden können, die das jeweilige Tupel noch nicht erhalten haben. Hierzu wird eine Persistierung der Tupel vorgenommen. Aus

der Persistierung kann, zum Beispiel nach einem Absturz, der alte Zustand wieder hergestellt werden.[20] Die *zuverlässige Zustellung* garantiert nur eine Zustellung, solange kein Fehler am Server auftritt.

Bezug zu der vorliegenden Arbeit

Die hier vorgestellten Middleware-Arten geben einen Überblick über das breite Spektrum an Möglichkeiten, die eine Middleware bietet. Für den Einsatz in dem zu entwickelnden Framework eignet sich speziell der Ansatz der koordinationsbasierten Middleware und hierbei explizit das Publish/Subscribe-Modell. Jede Komponente kann so ihre Daten unter einem definierten Themengebiet veröffentlichen. Die anderen Komponenten erhalten diese Daten bei Veröffentlichung automatisch, in dem sie das Themengebiet abonnieren. Es gibt keine Konsistenzprobleme bei der Bearbeitung eines Datums durch mehrere Komponenten. Außerdem werden in dem Framework der vorliegenden Arbeit voraussichtlich keine großen Daten publiziert, sondern viele kleine. Daher ist auch die Übertragung der kompletten Daten kein Problem.

Weitere Informationen zu koordinationsbasierter Middleware und verteilten koordinationsbasierten Systemen können in (A. S. Tanenbaum 2009, S. 692ff) und (A. Tanenbaum und Steen 2003, S. 779ff) nachgeschlagen werden.

3.3 Dynamische Anbindung der Komponenten

Die geplante Middleware-Architektur mit dem Publish/Subscribe-Modell erlaubt die dynamische Anbindung von Komponenten zur Laufzeit. Diese Art der Anbindung gleicht einer losen Kopplung in Softwarearchitekturen. Ludewig und Lichter beschreiben die lose Kopplung in *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken* anhand eines Beispiels wie folgt:

> „Man kann die Module auch mit Knödeln vergleichen, die beim Kochen dazu neigen, zusammenzukleben oder zu zerfallen. Das ideale Modul klebt nicht (geringe Kopplung) und zerfällt nicht (hoher Zusammenhalt)." (Ludewig und Lichter 2010, S. 413)

In dem Kontext der vorliegenden Arbeit soll „lose Kopplung" die Unabhängigkeit einzelner Komponenten (zum Beispiel Plug-ins) gegenüber anderer Komponenten des Frameworks bedeuten. Zwar wird sich herausstellen, dass einzelne Komponenten nur dann Informationen

[20] Die Apache ActiveMQ nutzt zum Beispiel eine KahaDB Datenbank zur Persistierung der Daten (siehe http://activemq.apache.org/kahadb.html, zuletzt besucht am 23.06.2013).

verarbeiten, wenn ihnen bestimmte Daten geliefert werden, aber diese Komponenten werden nicht abhängig von einem bestimmten Erzeuger der Daten sein.

Die dynamische Anbindung der Komponenten bietet zudem die Möglichkeit zur Laufzeit einzelne Komponenten an das Framework anzubinden oder aus dem Framework zu entfernen, ohne dass das Framework dabei an Stabilität verliert. Wenn die Kernkomponenten des Frameworks nicht von den lose angebundenen Komponenten (den Plug-ins) abhängen, können die einzelnen Plug-ins zur Laufzeit gewartet, getauscht oder abgeschaltet werden.

3.4 Complex Event Processing

Das *Complex Event Processing*, also die Verarbeitung von komplexen Ereignissen, ist ein Konzept der Informationsverarbeitung. Hierbei werden voneinander abhängige Events betrachtet und verarbeitet. Die Verarbeitung erfolgt dabei kontinuierlich und zeitnah zum Auftritt des Ereignisses. Das Ziel des *Complex Event Processing* ist die Ableitung von höherem Wissen aus der Kombination mehrerer Ereignisse (siehe auch (Eckert und Bry 2009)).

Das *Complex Event Processing* ist ein Verfahren, das auch in der Wirtschaft und in der Industrie bereits weit verbreitet ist. So wird es zum Beispiel im Bereich des *automatisierten Handels*[21] eingesetzt. Hier werten Software-Agenten Börsenkurse aus und verknüpfen verschiedene Ereignisse mit dem Verlauf der Börsenkurse. Auf Grundlage dieses kontinuierlichen Informationsstroms können die Software-Agenten automatisch Prognosen erstellen. Auf dieser Basis handeln die Software-Agenten dann automatisch mit Aktien. (Eckert und Bry 2009)

Die Abfrage der Ereignisse aus dem kontinuierlichen Informationsstrom ist dabei eines der wichtigsten Merkmale des *Complex Event Processing*. Der Abfrage unterliegen einige Anforderungen, die Eckert und Bry in »Complex Event Processing (CEP)« genauer beschreiben. Der folgende Absatz soll einen kurzen Überblick über diese Anforderungen geben.

Die **Extraktion** von Daten aus den Ereignissen ist der erste wichtige Punkt. Denn die Daten werden benötigt, um das Ereignis zu analysieren und zu entscheiden, ob und wie auf das Ereignis reagiert werden soll. Die **Komposition** von Ereignissen ist ebenso wichtig. Hierüber können mehrere Ereignisse verbunden und so ihr gemeinsames Auftreten über die Zeit als komplexes Ereignis betrachtet werden. Hieraus ergibt sich direkt eine weitere Anforderung: Der **zeitliche Zusammenhang** von Ereignissen. So muss jedes Ereignis in einem zeitlichen Zusammenhang zu seinem Auftritt stehen. Auf diese Weise können Ereignisanfragen gestellt werden, die bestimmte Ereignisse in einem Zeitraum abrufen oder Ereignisse in einer zeitlichen Reihenfolge anordnen. Diese Anforderung ist Grundlage für die **Akkumulation**. Sie beschäftigt sich mit

[21] http://de.wikipedia.org/wiki/Automatisierter_Handel, zuletzt besucht am 24.06.2013

der negierten Anfrage und der Aggregation von Ereignisdaten. Die Negation drückt aus, dass ein bestimmtes Ereignis in einem bestimmten Zeitraum nicht aufgetreten ist. Die Aggregation von Ereignisdaten schafft die Möglichkeit Ereignisdaten zu kombinieren.

Neben diesen Anforderungen gibt es noch zwei Arten von Regeln, die Eckert und Bry benannt haben. Die **deduktiven Regeln** definieren neue Ereignisse auf der Basis von aufgetretenen Ereignissen. Dabei gilt das aufgetretene Ereignis als Auslöser der Regel. Das Ergebnis der Abarbeitung der Regel ist ein neues Ereignis. Diese Funktionalität ist der von *Sichten* in Datenbanken sehr ähnlich. Die zweite Art von Regeln sind die **reaktiven Regeln**. Diese beschreiben die Reaktion oder die explizite Nichtreaktion auf Ereignisse oder komplexe Ereignisse. Eine Reaktion kann zum Beispiel der Aufruf einer Prozedur sein.

Weitere Informationen zum Thema *Complex Event Processing* werden unter anderem auch in *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems* (Luckham 2002) dargestellt.

Bezug zu der vorliegenden Arbeit

Das *Complex Event Processing* ist ein etablierter Ansatz zur Auswertung von vielen, kontinuierlich auftretenden und zusammenhängenden Ereignissen. Das zu entwickelnde Framework soll eine Grundlage zur Verarbeitung und Auswertung von Messdaten darstellen. Die Messdaten können zur Analyse von und zur Reaktion auf potentielle Ablenkungsfaktoren am Arbeitsplatz genutzt werden. Gerade die Analyse der Messdaten bedarf der Kombination und Abstraktion verschiedener Messdaten zu einem Ganzen. Hier setzt das *Complex Event Processing* an und bietet Konzepte, um diesen Prozess der Kombination, Abstraktion und Reaktion durchzuführen.

3.5 Blackboard-Architektur

Wie sich im Abschnitt 3.2 bereits gezeigt hat, soll eine Middleware mit dem Publish/Subscribe-Modell die Grundlage für die Kommunikation zwischen den Komponenten des zu entwickelnden Frameworks darstellen. Diese Architektur soll es ermöglichen, dass alle Komponenten des Frameworks auf sämtliche Informationen, die mit Hilfe des Frameworks gesammelt werden, zugreifen können. Man kann sich die Kommunikationsschnittstelle wie eine Tafel *(im Englischen: Blackboard)* vorstellen. Zur Veranschaulichung sei ein größeres Gesamtproblem angenommen, welches fünf Wissenschaftler gemeinsam lösen wollen:

> Die Wissenschaftler stehen vor einer Tafel. Sie befassen sich alle mit einem gemeinsamen Gesamtproblem, das sie unter sich in Teilprobleme aufgeteilt haben. So

kann sich jeder einzelne auf sein Teilproblem konzentrieren und seine Ansätze können zur Lösung des Gesamtproblems beitragen. Hierzu hat jeder Wissenschaftler einen Stift und einen Block mit Haftnotizzetteln vor sich. Wichtige Informationen und Ansätze zur Lösung ihres Teilproblems bringen die Wissenschaftler dann mit den Haftnotizen an der Tafel an. Um eine bessere Übersicht zu gewinnen, haben die Wissenschaftler die Tafel in unterschiedliche Themengebiete aufgeteilt, die nicht zwingend an ein Teilproblem gebunden sind. Passt eine Haftnotiz in kein Themengebiet, kann der Wissenschaftler ein neues Themengebiet anlegen. Jeder der Wissenschaftler ist somit in der Lage, die Informationen der anderen Wissenschaftler zu verwenden, um sein eigenes Teilproblem zu lösen. Gleichzeitig trägt jeder einzelne zur Lösung des Gesamtproblems bei.

Die Tafel bildet also die Kommunikationsschnittstelle zwischen den Wissenschaftlern. Jeder kann auf die vorhandenen Informationen zugreifen und selbst Informationen bereitstellen.

Die Abbildung 3.4 stellt das Software-Architekturmuster „Blackboard“ (im folgenden *Blackboard-Architektur* genannt) exemplarisch dar. Sie ist an die Arbeit von Ellenberg u. a. angelehnt.[22] In der Abbildung 3.4 sind die Publisher und die Subscriber durch verschiedene Komponenten, wie Sensoren und Recheneinheiten, repräsentiert. Das Blackboard fungiert als die zentrale Kommunikationsschnittstelle. Jede Komponente kann lesend und schreibend auf das Blackboard zugreifen. Die Blackboard-Architektur von Ellenberg u. a. gibt keine Vorgaben zur Formatierung der Daten. Das in der vorliegenden Arbeit zu entwickelnde Framework setzt auf eine einheitliche Formatierung der Daten im JavaScript Object Notation (JSON)-Format. Jede Komponente kann sich zudem zur Laufzeit an das Blackboard anbinden oder von diesem entfernen. Keine Komponente weiß, wer außer ihr Daten auf das Blackboard schreibt oder von ihm liest.

Zur Auswahl einer geeigneten Implementation einer Blackboard-Architektur wird auf vorhandene Forschungsergebnisse zurückgegriffen. Otto und Voskuhl haben in *Entwicklung einer Architektur für den Living Place Hamburg* verschiedene Blackboard-Architektur-Implementationen untersucht. Ziel war es, eine Kommunikationsarchitektur für das Living Place Hamburg (siehe Luck u. a. 2010) aufzubauen, über die die einzelnen Teilsysteme der intelligenten Umgebung ihre Daten austauschen können. Ein wichtiger Vorsatz war dabei, dass alle Teilsysteme die gleiche Sicht auf die Kontextdaten haben (damit Reaktionen zum Gesamtkontext passen) und die verwendete Implementation schnell, zuverlässig und gut do-

[22] Weitere Verwendungen der Blackboard-Architektur werden unter anderem in diesen Arbeiten beschrieben: (Ellenberg 2011; Ellenberg u. a. 2011; Voskuhl 2012)

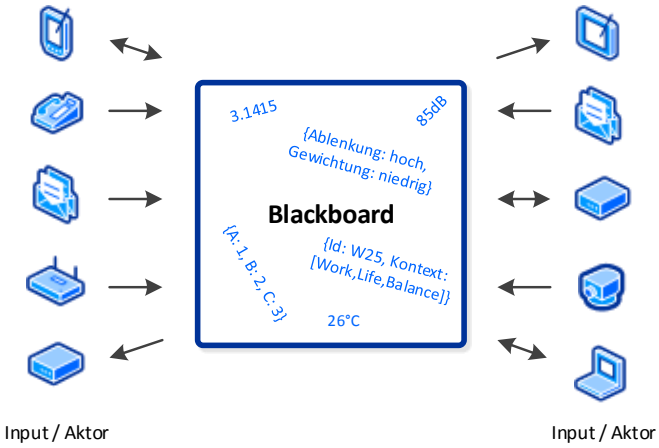


Abbildung 3.4: Blackboard-Architektur

kumentiert ist. Otto und Voskuhl haben sich für den *ActiveMQ Message Broker* von Apache entschieden, weil dieser sämtlichen Anforderungen genügt und sehr flexibel eingesetzt werden kann.[23] In dieser Arbeit wird daher ebenfalls der ActiveMQ als Referenzarchitektur verwendet.

Bezug zu der vorliegenden Arbeit

In der vorliegenden Arbeit kann auf die Erfahrungen von Otto und Voskuhl zurückgegriffen und hierdurch ebenfalls eine Blackboard-Architektur als Kommunikationsschnittstelle eingesetzt werden. Die Blackboard-Architektur bietet den Komponenten große Flexibilität bei der Wahl der Kommunikationskanäle. Jede Komponente kann zur Laufzeit eigene Themengebiete erstellen oder beliebige Themengebiete verwenden und abonnieren. Zusätzlich bietet diese Architektur eine hohe Flexibilität beim Einsatz der Nachrichtenformate zum Austausch der Daten. So soll in dieser Arbeit zwar JSON als einheitliches Nachrichtenformat verwendet werden, es könnte aber zum Beispiel auch die eXtensible Markup Language (XML) zum Einsatz kommen.

3.6 Persistierung

Eine Persistierung von Daten wird immer dann notwendig, wenn diese Daten zu einem späteren Zeitpunkt ausgewertet werden sollen. Im Kontext des kontinuierlichen Auftretens

[23] Vergleiche auch die Arbeit Hollatz 2010.

von Ereignissen bildet die Persistierung der Ereignisdaten zudem eine Möglichkeit, eine zeitlich beschränkte Sichtweise auf die Daten zu erhalten. Ohne die Persistierung wäre es zum Beispiel nicht möglich, die Entwicklung der Messdaten über eine Zeitspanne zu betrachten.

Manchmal ist es notwendig, Messreihen zu betrachten und auszuwerten, um hieraus abstrakte Informationen abzuleiten. Dies soll an einem kurzen Beispiel verdeutlicht werden:

> Es macht einen bedeutenden Unterschied, ob jemand jetzt gerade seine Entwicklungsumgebung fokussiert hat, oder dies bereits seit einer Stunde tut. In diesem Fall wäre vielleicht zusätzlich relevant, wie häufig in der gleichen Zeitspanne die Maus und die Tastatur benutzt wurden. Ein Aktor könnte aus dieser Menge an Informationen beispielsweise ableiten, dass der Benutzer seit längerer Zeit an einer Aufgabe arbeitet und sehr vertieft in diese ist.

Das Beispiel zeigt, dass es also unabdingbar ist, dass die Daten persistiert werden, um bestimmte Situationen überhaupt erkennen zu können. Nun stellt sich jedoch die Frage, wann welche Daten persistiert werden sollten. Da jedoch keine verlässliche Aussage über die zukünftige Verwendung der Daten getroffen werden kann, sollen in dem zu entwickelnden Framework prinzipiell alle Daten persistiert werden. Nur so kann später garantiert werden, dass eine bestimmte Menge an Informationen abrufbar ist. Der einfachste Weg besteht darin, die Tupel, die über die Kommunikationsschnittstelle verschickt werden, dauerhaft zu speichern. Um festzulegen, an welchem Punkt die Persistierung der Daten angestoßen wird, sind im Folgenden drei unterschiedliche Ansätze aufgestellt, die für das zu entwickelnde Framework in Frage kommen.

Der **erste Ansatz** konzentriert sich auf den Publisher-Prozess. Jeder Publisher wäre hierbei dazu verpflichtet, das zu publizierende Tupel an einer zentralen, globalen Stelle zu speichern. Dieser Vorgang sollte jedoch vereinfacht und durch eine weitere Abstraktionsschicht, zum Beispiel durch einen Wrapper, automatisiert werden. Der Wrapper übernimmt dann für den Publisher-Prozess die Veröffentlichung und die Speicherung des Tupels. Der Vorteil dieses Ansatzes ist, dass jede Nachricht automatisch persistiert werden würde. Zudem ist der Aufwand der Speicherungsanfrage auf die einzelnen Publisher verteilt. Jedoch muss jeder Publisher den Wrapper verwenden. Außerdem wird für jede mögliche Programmiersprache ein Wrapper benötigt. Der **zweite Ansatz** sieht die Persistierung durch einen externen Prozess vor. Dieser abonniert sämtliche Themengebiete des Tupelraums und speichert alle veröffentlichten Daten an einer globalen Stelle. Der Vorteil ist, dass sich nur eine Komponente mit der Persistierung auseinandersetzen muss. Zudem ist dieser Ansatz unabhängig von der Anbindung einzelner Publisher an den Tupelraum. Die Persistierung unterliegt hierbei allerdings zwei Schwierigkeiten. Zum einen könnte die Persistierungskomponente ausfallen oder unter zu hohem Datenaufkom-

men überlastet sein. Zum anderen könnte eine andere Komponente des Frameworks ein Tupel konsumierend lesen, bevor die Persistierungskomponente das Tupel gelesen hat. In diesem Fall würde das Tupel nicht gespeichert werden. Die Persistierung der Tupel könnte auch direkt durch den Tupelraum erfolgen. Dieser **dritte Ansatz** hätte den Vorteil, dass keine externe Komponente die Persistierung übernehmen müsste. Zum Teil nehmen die Tupelraum-Server diese Aufgabe bereits wahr. Häufig werden die Daten jedoch nach einer bestimmten Zeit wieder gelöscht. Zusätzlich kann nicht immer von außen einfach auf diesen Datenbestand zugegriffen werden.

In der vorliegenden Arbeit wurde der erste Ansatz gewählt. Dieser bietet die Möglichkeit, die Anfragen zur Persistierung direkt von den einzelnen Komponenten des Frameworks abzuschicken. So würden die Daten auch dann gespeichert werden, wenn die Kommunikationsschnittstelle nicht mehr erreichbar ist.

Ein weiterer Vorteil repräsentiert sich in der Form eines Wrappers in Java, der verwendet werden könnte. Dieser Wrapper ist aus der Arbeit von Otto und Voskuhl hervorgegangen.[24] Auf den Wrapper wird im weiteren Verlauf dieser Arbeit noch einmal eingegangen (vergleiche Kapitel 4.1) werden.

3.7 Benutzerinteraktion

Alle bisher angesprochenen technischen Details sind primär für den Einsatz in autonomen Systemen vorgesehen. Diese Systeme sehen keine Interaktion mit einem Benutzer vor. Das Konzept des zu entwickelnden Frameworks hat jedoch den Anspruch, den Benutzer zu involvieren. Hierzu sollen der Benutzer und seine Umgebung nicht nur überwacht werden, der Benutzer soll auch aktiv in das System eingreifen können. Dazu muss er die Möglichkeit haben, Einfluss auf jede Komponente zu nehmen. Dieser Einfluss kann dem Benutzer zum Beispiel durch Konfigurationsparameter und entfernte Prozeduraufrufe eingerichtet werden.

Parameter

Parameter sind Variablen, die die Ausführung einer Software beeinflussen. In dem zu entwickelnden Framework sollen einzelne Komponenten Parameter anbieten, die vom Benutzer zur Laufzeit verändert werden können. Da es sich bei dem Framework um ein verteiltes System handeln wird, muss diese Freigabe von Parametern unabhängig von dem Standort der Kompo-

[24] Otto hat später die Deploymentstruktur des Living Place Hamburg weiter ausgebaut (Otto 2013). Voskuhl hat die Architektur genutzt, um eine Architektur für *context-aware* Systeme im Bereich von intelligenten Wohnungen aufzubauen (Voskuhl 2012).

nenten stattfinden. Einzige Voraussetzung soll sein, dass der Benutzer über die Existenz der Komponenten und ihre jeweiligen Konfigurationsparameter informiert wird. Es muss also ein Konzept vorgestellt werden, dass es erlaubt, Änderungen der Parameter von Komponenten in einem verteilten System vorzunehmen.

Veröffentlichung von Parametern

Wie zuvor beschrieben, wird das Framework auf eine Blackboard-Architektur als zentrale Kommunikationsschnittstelle setzen. Diese kann, neben dem Austausch von Messdaten, auch für die Konfiguration der Komponenten genutzt werden. Jede konfigurierbare Komponente muss hierzu ihre Konfigurationsparameter auf einem zuvor spezifizierten Themengebiet veröffentlichen. Dieses Themengebiet kann von einer Komponente, die lediglich zur Interaktion mit dem Benutzer dient, abonniert werden. Zusätzlich abonnieren alle konfigurierbaren Komponenten selbst dieses Themengebiet. Jedes Tupel mit Konfigurationsparametern muss dann eindeutig identifizierbar sein, damit die Komponenten ihr Tupel mit Konfigurationsparametern wiedererkennen können. Nimmt der Benutzer Änderungen an der Konfiguration einer Komponente vor, wird das veränderte Tupel auf dem Themengebiet veröffentlicht. Das Tupel kann dann von der Komponente gelesen werden.

3.8 Architekturentwurf des Frameworks

Aus der Analyse (Kapitel 2) und den Designentscheidungen (Kapitel 3) kann nun die Architektur des zu entwickelnden Frameworks bestimmt werden. Wichtig hierbei ist, dass die Architektur möglichst allgemein und flexibel gehalten wird, damit sie verschiedene Szenarien (vergleiche Abschnitt 2.3) abdecken kann. Die Abbildung 3.5 zeigt die grundlegende Architektur des zu entwickelnden Frameworks.

Die Kommunikationsschnittstelle soll durch einen ActiveMQ realisiert werden. Die Publisher, also die Plug-ins, die Informationen auf der ActiveMQ veröffentlichen, sind für die Persistierung der Tupel verantwortlich und müssen die Tupel in einer MongoDB speichern. Der ActiveMQ und die MongoDB stellen gemeinsam die Middleware und somit die Kommunikationsschnittstelle dar. Der Computer des Benutzers soll eine besondere Rolle einnehmen und wurde daher auch in der Abbildung gesondert dargestellt. Auf diesem Computer läuft später die Applikation, die die anderen Komponenten steuert und konfigurieren kann. Daher wird sie auch *zentrale Steuerungseinheit* genannt. Die Plug-ins und Subsysteme können mit Hilfe der ActiveMQ zusammenarbeiten, um potentielle Ablenkungsfaktoren, den aktuellen Kontext oder andere Gruppen von Datenmengen zu bestimmen.

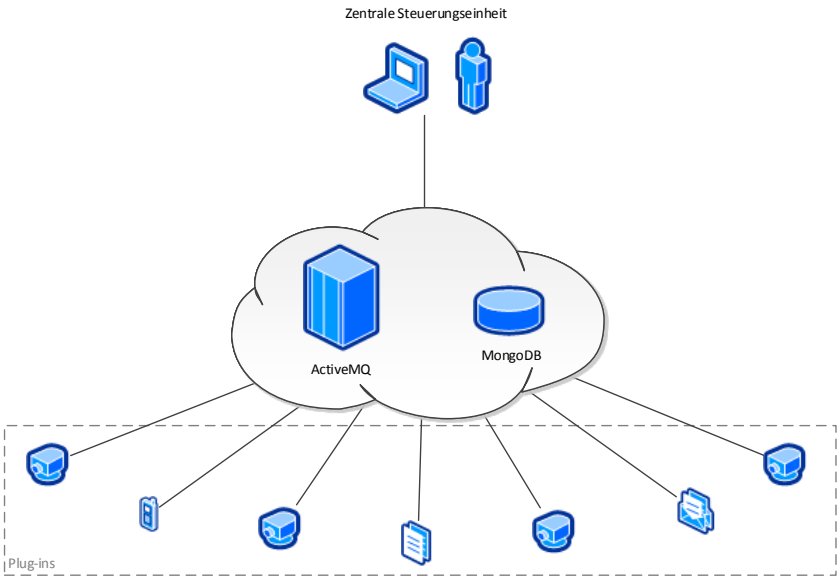


Abbildung 3.5: Schematische Architekturübersicht des Frameworks

Im nächsten Kapitel wird auf die einzelnen Komponenten genauer eingegangen und beschrieben, welche Möglichkeiten die zentrale Steuerungseinheit bietet. Die Ausführungen umfassen zusätzlich die unterschiedlichen Arten von Plug-ins und die verschiedenen Themengebiete, die für die Konfiguration und die Steuerung der Plug-ins verwendet werden sollen.

4 Architekturübersicht und Evaluierung

Ausgehend von der Analyse und dem Design wurde die Architektur beispielhaft implementiert. Hierbei ist unter anderem ein *Beispiel-Plug-in* entstanden, das alle definierten Schnittstellen[25] verwendet.

In den folgenden Abschnitten werden Empfehlungen zur Umsetzung der Architektur des vorangegangenen Designs beschrieben. Die Erläuterungen zeigen, welche Komponenten für welchen Zweck verwendet werden können und wie sich das verteilte System im Netzwerk darstellt.

4.1 Infrastruktur

Zur Realisierung des verteilten Systems wird eine zentrale Stelle zur Kommunikation benötigt. An dieser wird ein Server eingerichtet, der sowohl die Blackboard-Architektur als auch eine Datenbank zur Persistierung bereitstellt.

Die Verwendung eines zentralen Servers hat mehrere Gründe. Zunächst wird so auf einfache Weise sichergestellt, dass jede Komponente des Frameworks die Netzwerkadresse, unter der die Kommunikationsschnittstelle erreichbar ist, kennt. Zum anderen bleibt das System leicht wartbar.

Eine weitere Möglichkeit zur Realisierung des verteilten Systems bietet eine verteilte Serverstruktur. Diese ermöglicht eine bessere Lastenverteilung. Für den geplanten Einsatzzweck wird ein zentraler Server dennoch völlig ausreichen (siehe hierzu auch Otto und Voskuhl 2011).

In der Abbildung 3.5 ist neben dem Kommunikationsserver und den verteilten Komponenten auch der Rechner des Benutzers dargestellt. Der Benutzer an sich steht im Mittelpunkt sämtlicher Messungen. Der primäre Einsatzzweck des entwickelten Frameworks ist die Erkennung von körperlichen und geistigen Zuständen des Benutzers, die den Einfluss von potentiellen Ablenkungsfaktoren verstärken. Außerdem sollen die potentiellen Ablenkungsfaktoren selbst ermittelt werden. Der Rechner des Benutzers nimmt jedoch in der Architektur des Frameworks eine Sonderrolle ein, da von diesem Rechner die Konfiguration durch den Benutzer zur Laufzeit

[25] Siehe Anhang A

ermöglicht wird. Der Benutzerrechner gehört damit zu den Kernkomponenten des Frameworks. Außerdem dient dieser Rechner zur Darstellung von Informationen, die die Sensoren dem Benutzer übermitteln wollen.

Ausgehend von dieser Annahme ist in die Komponente, die auf dem Rechner des Benutzers läuft, auch ein zusätzlicher Publisher eingebaut. Dieser ist für die globale Konfiguration des Frameworks verantwortlich. Die globale Konfiguration gibt den Komponenten Auskunft über den aktuellen Ausführungszustand des Frameworks und kann weitere globale Parameter enthalten. Damit jede Komponente immer auf dem aktuellen Stand ist und auch neue Komponenten die globale Konfiguration erhalten, wird die aktuelle Konfiguration periodisch[26] veröffentlicht. Somit dient sie gleichzeitig als eine Art *Keep-Alive-Ticker*.

4.1.1 Blackboard durch einen ActiveMQ-Server

Als Tupelraum beziehungsweise Blackboard-Server soll ein Apache ActiveMQ-Server eingesetzt werden. Er unterstützt den Java Messaging Service 1.1, ist schnell und OpenSource. Zudem ist der ActiveMQ sehr gut dokumentiert.

Bestärkt wurde die Entscheidung auch durch die Untersuchungen, die Otto und Voskuhl bezüglich einer geeigneten Blackboard-Architektur durchführten. In ihren Untersuchungen haben Otto und Voskuhl den ActiveMQ als ihre favorisierte Blackboard-Architektur ausgewählt. Die Entscheidung fiel insbesondere wegen der einfachen und systemunabhängigen Installation, der umfangreichen Dokumentation und der hohen Geschwindigkeit auf dem ActiveMQ.

Ein weiterer Pluspunkt ist der vorhandene Wrapper. In ihrer Arbeit *Weiterentwicklung der Architektur des Living Place Hamburg* (Otto und Voskuhl 2011) haben Otto und Voskuhl einen Wrapper in Java geschrieben, der die Anbindung an den ActiveMQ kapselt. Außerdem bietet er einen weiteren Vorteil: Der Wrapper für den ActiveMQ beinhaltet zusätzlich die automatische Persistierung der publizierten Nachrichten in einer MongoDB. Der Vorteil dieser gegebenen Infrastrukturkomponente ist, dass sie von den Autoren bereits ausgiebig getestet wurde.

Allerdings sind alle auf einem ActiveMQ veröffentlichten Tupel für jeden, der Zugang zum Netzwerk hat, zugänglich. Und die Persistierung der Tupel innerhalb des ActiveMQ ist nicht hinreichend, um die Tupel später einer Analyse zu unterziehen. Um dieses Problem zu beheben, wird eine extra Datenbank eingesetzt, die folgend beschrieben wird.

Der ActiveMQ-Server soll zusammen mit dem MongoDB-Server auf einem Rechner im Netzwerk ausgeführt werden. Während der Entwicklungsphase war dies meist der Rechner des Benutzers, auf dem auch die zentrale Steuerungseinheit lief. In einem produktiven Setup sollte

[26] zum Beispiel einmal pro Minute

dies jedoch ein dedizierter Server sein. Nur so kann garantiert werden, dass den Serveranwendungen genügend Ressourcen zur Verfügung stehen und diese ihre volle Leistung ausschöpfen können. Insbesondere der ActiveMQ muss mit höchster Geschwindigkeit sämtliche Nachrichten auf den einzelnen Themengebieten verarbeiten können. Andernfalls kommt es bei der Kommunikation zwischen den Komponenten des Frameworks zu stärkeren Verzögerungen, was sich in einer geringeren Reaktionszeit des Systems niederschlägt.

Einführung in die Funktionsweise des ActiveMQ Message Brokers

Der ActiveMQ besteht aus verschiedenen Komponenten, die in der Abbildung 4.1 dargestellt sind. Es folgen Erläuterungen zu den für das entwickelte Framework wichtigen Teilen des ActiveMQ-Servers.

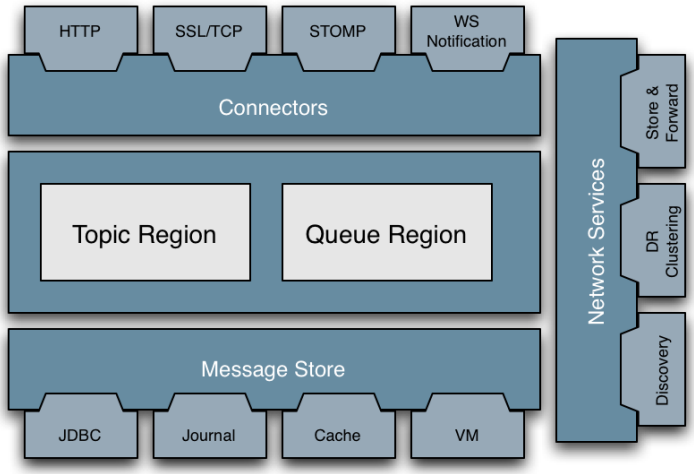


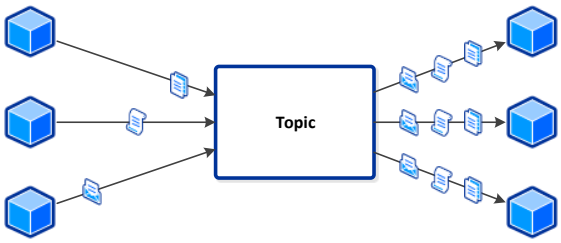
Abbildung 4.1: Komponentenübersicht des ActiveMQ Message Brokers[27]

Topic Region und Queue Region: Das Kernstück des ActiveMQ besteht aus den zwei Regionen *Topic* und *Queue*. Zusammen bilden sie einen Tupelraum, wie er im Kapitel 3.2 mit dem System *Linda* eingeführt wurde. Der Unterschied der beiden Regionen liegt in der Art des Zugriffs auf die Tupel. Die **Topic Region** stellt allen Subscribern das nicht-konsumierende Lesen zur Verfügung. Es erhalten somit alle Subscriber eines Themengebiets alle Tupel (Abbildung 4.2a). Die **Queue Region** hingegen lässt nur das konsumierende Lesen zu. Jedes Tupel kann also von nur genau einem Subscriber gelesen werden (Abbildung 4.2b). Danach ist das Tupel nicht mehr im Tupelraum vorhanden.

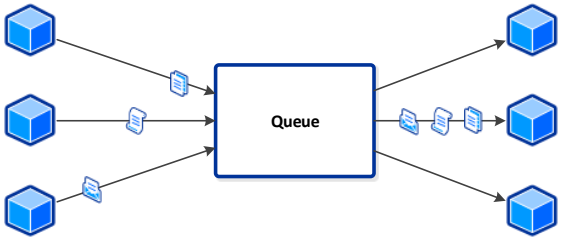
[27] Quelle: `http://activemq.apache.org/code-overview.html`, zuletzt besucht am 11.07.2013

Connectors: Damit die Publisher und die Subscriber mit der ActiveMQ kommunizieren können, bietet die Architektur verschiedene Konnektoren an. In der Abbildung 4.1 sind nur vier mögliche Konnektoren abgebildet. Tatsächlich gibt es noch einige mehr.[28] Hinter dem *HTTP-Connector* verbirgt sich beispielsweise ein Representational State Transfer (REST)-API. Über dieses können alle Komponenten angebunden werden, die internetfähig sind. Ein wichtiger Konnektor für das Framework ist in der Abbildung 4.1 nicht aufgeführt. Es ist der Konnektor zur Anbindung von Java Programmen. Hier erfolgt die Anbindung über das Java Native Interface (JNI). Dieser Konnektor ist sehr wichtig, da das in der vorliegenden Arbeit entwickelte Framework in Java umgesetzt wurde.

Weitere Informationen zum ActiveMQ können auf der Projektseite `http://activemq.apache.org/`[29] oder in *ActiveMQ in action* (Snyder, Bosnanac und Davies 2011) nachgelesen werden.



a: Tupelfluss bei einer Topic Region



b: Tupelfluss bei einer Queue Region

Abbildung 4.2: Unterschied des Tupelflusses zwischen einer *Topic Region* und einer *Queue Region*

[28] Siehe hierzu `http://activemq.apache.org/connectivity.html`, zuletzt besucht am 15.07.2013

[29] zuletzt besucht am 12.07.2013

4.1.2 Persistierung durch eine MongoDB

Wichtig bei der Entscheidung für eine geeignete Datenbank ist die Art der Speicherung der Daten. In dem entwickelten Framework sollen Tupel eines Tupelraums gespeichert werden. Diese Tupel folgen keiner Struktur, die mit herkömmlichen relationalen Datenbanken abgebildet werden können. Daher musste eine andere Art der Speicherung gefunden werden.

Wie sich in *Weiterentwicklung der Architektur des Living Place Hamburg* (Otto und Voskuhl 2011) herausgestellt hat, ist die neuere Technologie des NoSQL-Ansatzes besser geeignet. Hierbei werden die Daten als Dokumente betrachtet und entsprechend als Dokumente gespeichert. Otto und Voskuhl haben zwei verschiedene NoSQL-Datenbanken einem Test unterzogen und danach die MongoDB als Testsieger gekürt. Sie konnte sich in Sachen Performanz bei kleinen Daten durchsetzen. Und da sich diese Anforderung auch in der vorliegenden Arbeit wiederfindet, soll die Persistierung in diesem Framework ebenfalls durch eine MongoDB realisiert werden.

4.2 Funktionale Übersicht des Frameworks

Die Abbildung 4.3 stellt eine Übersicht über die Funktionalität des Frameworks dar. Die farbigen Pfeile geben die Kommunikationswege und ihre Zusammenhänge wieder. Auf der linken Seite der Abbildung sind die Teile der zentralen Steuerungseinheit abgebildet. Hier kann zum Beispiel die Komponente für den *Konfigurationsdialog* eine Nachricht *Dialog anzeigen* empfangen und eine Nachricht *Konfiguration speichern* veröffentlichen.

Auf der rechten Seite sind sämtliche Plug-ins angeordnet. In dieser Abbildung sind die Arten der Plug-ins dargestellt. Die *Input-Plug-ins* veröffentlichen ihre Sensordaten. Diese Sensordaten werden von den *Aktor Plug-ins* empfangen und verarbeitet. Als Beispiel für ein *Aktor-Plug-in* sind die zwei Komponenten *Collector* und *Reasoner* in der Abbildung 4.3 dargestellt. Der *Collector* sammelt Sensordaten in bestimmten Zeitabschnitten und gibt diese Menge an Sensordaten an den *Reasoner* weiter. Der *Reasoner* stellt eine lernende Komponente, zum Beispiel eine SVM dar. Er soll die vorhandenen Daten abstrahieren und einen höheren Abstraktionsgrad der Sensordaten erzeugen.

In der Mitte ist die Kommunikationsschnittstelle dargestellt, welche die Middleware repräsentiert. Sie besteht aus dem ActiveMQ-Server und dem MongoDB-Server und kann auf einem Rechner oder mehreren beliebigen Rechnern installiert sein.

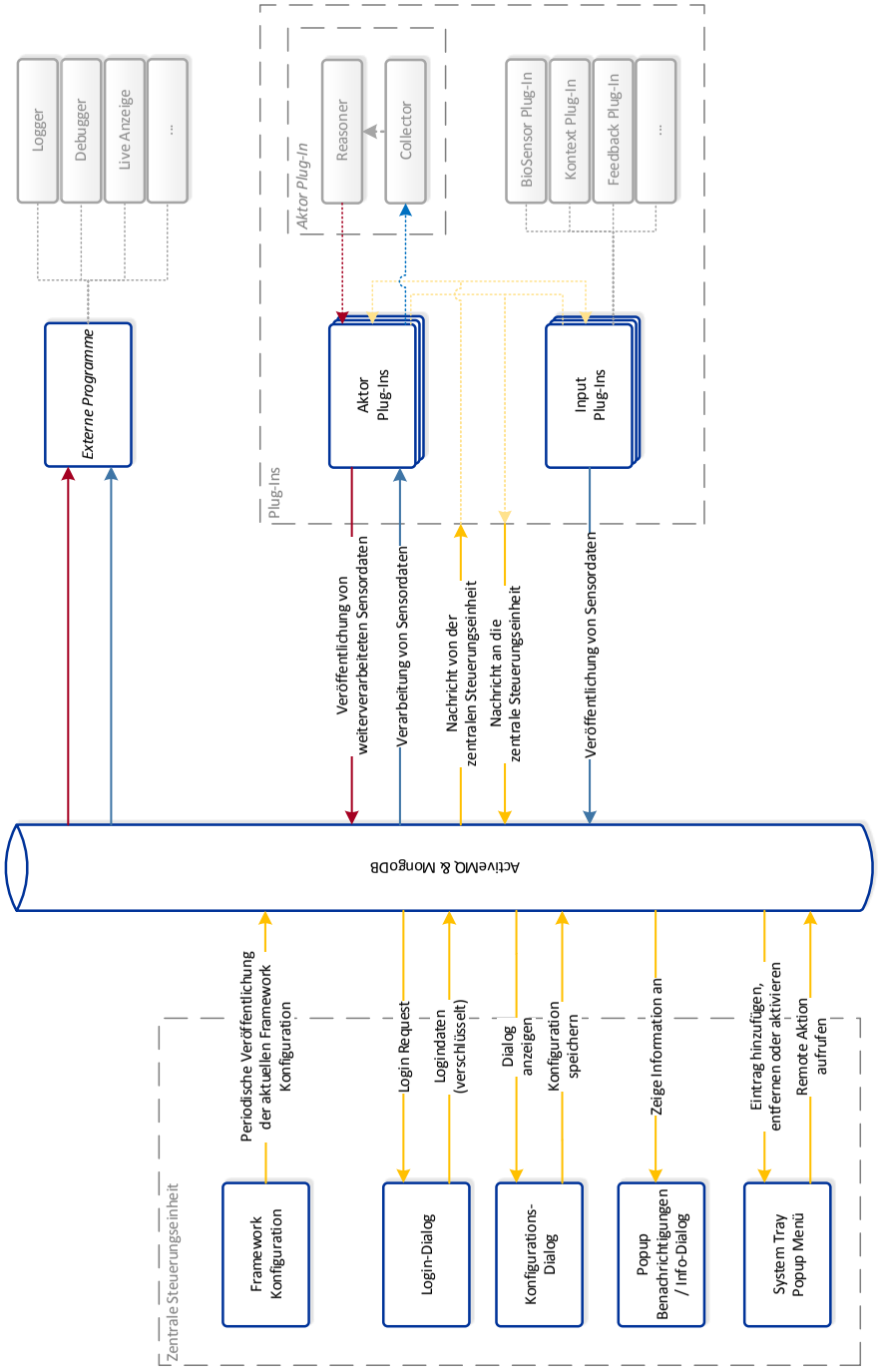


Abbildung 4.3: Komponentenübersicht des Frameworks

4.3 Zentrale Steuerungseinheit

Die Kommunikation zwischen dem Framework und einem Benutzer erfolgt über eine zentrale Steuerungseinheit. Dem Benutzer wird hierüber ermöglicht, Informationen von einzelnen Komponenten des Systems zu betrachten und einzelne Komponenten zu konfigurieren. Die Benutzerinteraktion ist ein wichtiger Punkt in jedem System.

Verschiedene mögliche Benutzerinteraktionen sind in unterschiedliche Komponenten unterteilt. Funktionen, die zu einer bestimmten Benutzerinteraktion oder einer Gruppe von Benutzerinteraktionen gehören, sind in einer Komponente gekapselt. Das System bleibt so leicht wartbar. Außerdem können einzelne Komponenten hierdurch leicht ausgetauscht werden. Die zentrale Steuerungseinheit teilt sich in die folgenden Komponenten auf:

Taskbar Notification Area Symbol

Im Jahr 1997 haben Oran u. a. die Taskbar Notification Area (TNA) im US-Patent 5,617,526 spezifiziert. Die TNA beschreibt einen Benachrichtigungsbereich auf der graphischen Benutzeroberfläche des Betriebssystems, der zur Anzeige von Benachrichtigungen reserviert ist. Die Anzeige der Benachrichtigungen kann auch von Programmen erfolgen, die aktuell im Hintergrund laufen. Ein Systemprogramm nimmt dafür Anfragen zur Anzeige von Benachrichtigungen entgegen. Ein wichtiger Punkt ist dabei, dass die Anzeige einer Benachrichtigung ein aktuell laufendes Programm nicht unterbricht. Zusätzlich können Symbole in der TNA angezeigt werden. Jedes Symbol repräsentiert eine Applikation, wobei die Applikation in Hintergrund ausgeführt werden kann. Für jedes Symbol besteht die Möglichkeit einen Tooltip anzuzeigen, sobald der Mauszeiger über das Symbol gehalten wird. (Oran u. a. 1997)

Abbildung 4.4: Darstellung des Symbols in der Taskbar Notification Area (TNA)

Über das Symbol einer Applikation in der TNA kann der Benutzer zusätzlich mit der Applikation interagieren. Ein Klick oder Doppelklick auf das Symbol kann die zugehörige Applikation öffnen. Meist werden über das Symbol weitere Interaktionsmöglichkeiten angeboten. Dem Benutzer können kurze Informationen über den aktuellen Zustand der Applikation gegeben

werden (Tooltip). Außerdem besteht die Möglichkeit, die Applikation über ein Menü zu steuern (Kontextmenü).

Das Framework soll ebenfalls über ein Symbol in der TNA repräsentiert werden (siehe Abbildung 4.4). Die zentrale Steuerungseinheit legt das Symbol in der TNA an. Zusätzlich bindet die zentrale Steuerungseinheit ein Kontextmenü an das Symbol (siehe Abbildung 4.5), über das der Benutzer mit dem Framework interagieren und es herunterfahren kann. Jede Komponente im Framework kann ihren eigenen Eintrag in dem Kontextmenü erhalten. Es bekommt dabei einen Menüpunkt im Hauptmenü zugewiesen, welcher ein Untermenü aufruft, das von der Komponente nach Belieben aufgebaut werden kann. Die Einbindung erfolgt mit der Nachricht `JsonMenuEntry` über das Themengebiet `BeLi.SystemTray.Menu`.

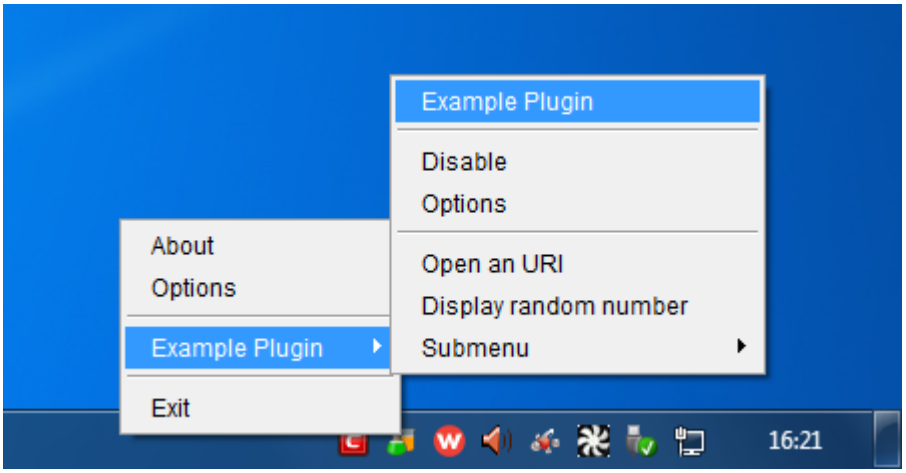


Abbildung 4.5: Darstellung des Kontextmenüs in der Taskbar Notification Area (TNA)

Das Framework bietet verschiedene Arten von Menüpunkten an. Jedem Menüpunkt kann dabei eine Funktion zugewiesen sein. Im Folgenden werden die unterschiedlichen Funktionen genauer beschrieben.

Darstellung eines Textes: Die einfachste Funktion ist die Darstellung eines Textes (siehe Abbildung 4.5, „Example Plugin“ rechts oben). Hierüber kann zum Beispiel der Name einer Komponente oder der Name einer Gruppe von Funktionen im Menü dargestellt werden. Ein Klick auf diesen Menüeintrag löst keine weitere Aktion aus.

Untermenü: Es ist möglich, ein weiteres Untermenü zu öffnen, in dem dann weitere Menüpunkte dargestellt werden (siehe Abbildung 4.5, „Example Plugin“ links). Ein Untermenü ist hilfreich, um Gruppen von Funktionen auszulagern und so die Benutzerfreundlichkeit der Anwendung zu erhöhen.

Lokale Funktionsaufrufe: Es können lokale Funktionen ausgeführt werden. Das *ExamplePlugin* hat beispielsweise einen Menüpunkt (siehe Abbildung 4.5, „Open an URI“) mit dem Aufruf einer Universal Resource Identifier (URI) im Standard Webbrowser des Betriebssystems des Benutzers verknüpft.

Entfernte Funktionsaufrufe: Neben den lokalen Funktionen können auch entfernte Funktionen aufgerufen werden. Hierzu wird eine `JsonRemoteAction` Nachricht auf dem Themengebiet `BeLi.SystemTray.Menu.RemoteAction` veröffentlicht. Diese Nachricht beinhaltet die Identifikationsnummer (ID) des JSON-Objekts, das den Menüpunkt definiert. Diese ID ist bei der Komponente, die den Menüeintrag erstellt hat, bekannt. Somit weiß diese Komponente, das der Aufruf an sie gerichtet ist und kann den beinhaltenden Funktionsaufruf ausführen.

Anzeige eines Konfigurationsdialogs: Eine spezielle Form des entfernten Funktionsaufrufs ist die Anzeige eines Konfigurationsdialogs (siehe Abbildung 4.5, „Options“). Dieser Menüpunkt führt ebenfalls einen entfernten Funktionsaufruf durch. Hierbei wird jedoch eine spezielle Funktion aufgerufen, die die aktuelle Konfiguration der Komponente anfordert, um diese im Konfigurationsdialog anzuzeigen. Auf den Konfigurationsdialog wird im Abschnitt *Dialoge* (Kapitel 4.3) noch genauer eingegangen.

Das Symbol in der TNA kann dem Benutzer über das Kontextmenü bereits viele Informationen und Interaktionspunkte anbieten. Zusätzlich dient es noch einem weiteren Zweck: Der Anzeige von Benachrichtigungen.

Anzeige von Benachrichtigungen

Die Anzeige von Benachrichtigungen ist in Betriebssystemen meist an ein Symbol in der TNA gebunden. Daher können Benachrichtigungen in dem Framework auch nur über die zentrale Steuerungseinheit angezeigt werden. Benachrichtigungen gelten dabei als kleine Fenster, die im Bereich der TNA auftauchen und sich nach kurzer Zeit von selbst wieder schließen. Die Abbildung 4.6 zeigt so eine Benachrichtigung.

Benachrichtigungen dienen Applikationen zur Anzeige von aktualisierten Informationen. Beispielsweise zeigen E-Mail-Programme eine Benachrichtigung an, wenn neue E-Mails empfangen wurden. Das Fenster weist den Benutzer auf diese Tatsache hin und gibt ihm die Anzahl der E-Mails als Zusatzinformation. Wichtig ist, dass Benachrichtigungen nicht unbedingt vom Benutzer wahrgenommen werden müssen. Gerade auf großen Bildschirmen oder Multimonitor-Systemen können Benachrichtigungen leicht übersehen werden, da sie zum Zeitpunkt der Anzeige nicht im Sichtbereich des Benutzers liegen müssen.

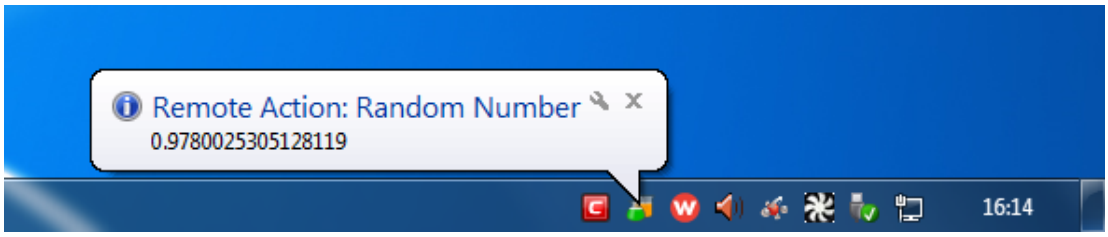


Abbildung 4.6: Darstellung einer Benachrichtigung in der Taskbar Notification Area (TNA)

Das Framework bietet jeder Komponente die Möglichkeit Benachrichtigungen anzuzeigen. Dabei unterscheidet es zwischen drei Kategorien von Benachrichtigungen. Die erste Kategorie beinhaltet die **Informations-Benachrichtigungen**. Sie umfasst alle Benachrichtigungen, die Aktualisierungen von Informationen beinhalten. Die zuvor genannte Benachrichtigung über den Eingang neuer E-Mails würde in diese Kategorie fallen. Als nächste Kategorie sind die **Warnungs-Benachrichtigungen** vorgesehen. Sie zeigen Informationen an, die im Zusammenhang mit einem Problem in der Applikation stehen. Diese Probleme beeinträchtigen jedoch noch nicht die Stabilität der Applikation, könnten aber zu einer Instabilität führen. Läuft die Applikation in einen instabilen Zustand, kann sie eine **Fehler-Benachrichtigung** anzeigen (dies ist die dritte Kategorie). Diese gibt dem Benutzer einen Hinweis darüber, dass die Applikation einen Fehler in ihrer Ausführung festgestellt hat und nicht mehr stabil läuft. Gegebenenfalls bindet die Applikation auch einen technischen Hinweis in die Benachrichtigung mit ein, die einem Administrator oder Programmierer bei der Behebung des Problems helfen kann.

Jede Komponente kann die Anzeige einer Benachrichtigung mit Hilfe der Veröffentlichung einer `JsonPopupMessage` Nachricht auf dem Themengebiet `BeLi.SystemTray.Popup` anfordern.

Dank der offenen, verteilten Architektur kann natürlich auch jede andere Komponente das Themengebiet zur Anzeige von Benachrichtigungen abonnieren. So wäre zum Beispiel eine Komponente, die auf einem Smartphone läuft, denkbar. Diese würde die `JsonPopupMessage` Nachricht empfangen und eine (zusätzliche) Benachrichtigung auf dem Smartphone anzeigen.

Wie zu Beginn gesagt: Benachrichtigungen können vom Benutzer schlicht übersehen werden. Daher bietet das entwickelte Framework eine weitere Möglichkeit an, dem Benutzer Informationen darzustellen: Die Dialoge.

Dialoge

Im Gegensatz zu den Benachrichtigungen sind Dialoge Fenster, die immer in der Mitte des Bildschirms dargestellt werden. Jeder Dialog muss vom Benutzer bestätigt werden, damit das

Dialog-Fenster sich wieder schließt. Ein Dialog-Fenster kann dabei theoretisch alles mögliche anzeigen. Von einem einfachen Text über ein Formular zur Eingabe von Daten bis hin zu einem Video ist alles, was im Desktopbereich bekannt ist, denkbar. In dem Framework wird sich auf drei Arten von Dialogen, die im Folgenden näher beschrieben werden, konzentriert.

Die einfachste Form ist die Anzeige eines **Informationsdialogs**. Ein Beispiel ist in der Abbildung 4.7 abgebildet. Dieser beinhaltet einen Text und einen Bestätigungsbutton. Der Informationsdialog dient der Anzeige von Beschreibungstexten oder Inhalten, die der Nutzer in jedem Fall lesen soll. Hier wären auch Lizenztexte oder Datenschutzbedingungen denkbar. Ein Informationsdialog wird von einer Komponente mit der `JsonInfoDialog` Nachricht über das Themengebiet `BeLi.InfoDialog` bei der zentralen Steuerungseinheit angefordert.

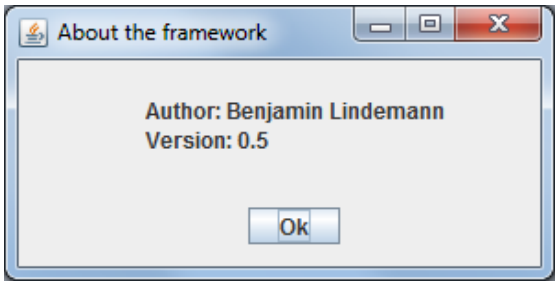


Abbildung 4.7: Darstellung eines Informationsdialogs

Eine erweiterte Form eines Dialogs ist der **Logindialog**, wie er in der Abbildung 4.8 abgebildet ist. Der Logindialog zeigt dem Benutzer ein Formular zur Eingabe von Anmeldedaten, bestehend aus einem Benutzernamen und einem Passwort, an. Zusätzlich sind in dem Fenster zwei Buttons eingebracht. Der eine Button bietet dem Benutzer die Möglichkeit den Anmeldeprozess abzubrechen. Der andere Button bestätigt die Eingabe des Benutzers und publiziert die Anmeldedaten im Tupelraum. Damit nicht jeder Subscriber die Anmeldedaten mitlesen kann, können die Anmeldedaten verschlüsselt angefordert werden. Dazu muss die Komponente, die die Daten anfordert, einen öffentlichen Schlüssel nach dem RSA-Kryptoverfahren mit der Anfrage zur Anzeige des Logindialogs in die `JsonLoginRequest` Nachricht über das Themengebiet `BeLi.LoginRequest` an die zentrale Steuerungseinheit übermitteln. So ist es nur noch der anfragenden Komponente möglich, die Anmeldedaten zu entschlüsseln.

Die letzte Art von Dialog ist der **Konfigurationsdialog**. Die zentrale Steuerungseinheit stellt einen Konfigurationsdialog zur Verfügung, indem für jede Komponente ein Tab angezeigt werden kann. Die Abbildung 4.9 zeigt einen Konfigurationsdialog, bei dem der Tab des Beispiel Plug-ins geöffnet ist. Die Erstellung und Anzeige des Tabs wird über die `JsonConfigDialog` Nachricht bei der zentralen Steuerungseinheit angefragt. Über den Konfigurationsdialog wird

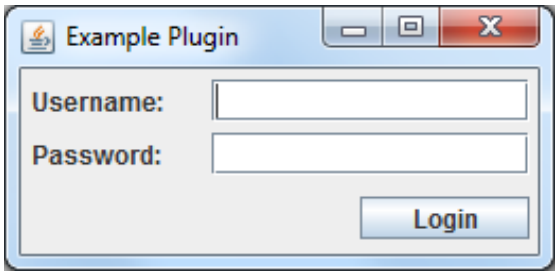


Abbildung 4.8: Darstellung eines Logindialogs

dem Benutzer die Möglichkeit gegeben, Änderungen an der aktuellen Konfiguration der einzelnen Komponenten vorzunehmen. Jede Änderung wird nach der Speicherung auf dem Themengebiet `BeLi.ConfigDialog` veröffentlicht. Alle Komponenten, die einen Tab in den Konfigurationsdialog integrieren, müssen dieses Themengebiet abonnieren, um hierüber die Änderungen an ihrer Konfiguration zu empfangen. Die Zuordnung der Nachrichten erfolgt anhand der ID. Diese ist die gleiche ID wie die ID, die in der Anfrage zur Integration des Tabs und der Anzeige des Konfigurationsdialogs verwendet wurde.

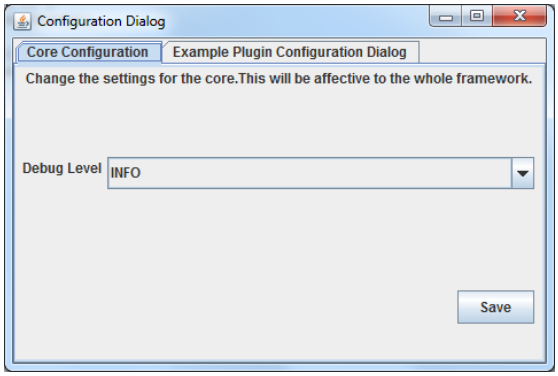


Abbildung 4.9: Darstellung eines Konfigurationsdialogs

Globale Frameworkskonfiguration

Die zentrale Steuerungseinheit übernimmt neben der Benutzerinteraktion noch eine weitere Funktion. Zur globalen Steuerung des Ausführungszustands des Frameworks ist in die zentrale Steuerungseinheit ein Konfigurations-Publisher eingebaut. Dieser publiziert periodisch die aktuelle globale Konfiguration des Frameworks. Die Konfigurationsparameter sind dabei in der Nachricht `JsonFrameworkConfiguration` verpackt. Die globale Konfiguration kann neben dem aktuellen Ausführungszustand natürlich noch weitere Optionen beinhalten. Es ist

beispielhaft ein Debuglevel eingebaut, der ebenfalls über die globale Konfiguration gesteuert werden kann.

Jede Komponente muss die globale Konfiguration empfangen und auswerten. Nur so kann garantiert werden, dass sich das Framework in einem definierten und konsistenten Ausführungszustand befindet. Der Benutzer des Frameworks könnte zum Beispiel über einen angebotenen Menüeintrag den globalen Ausführungszustand ändern. Ändert er ihn auf „Pausiert“, müssen alle Komponenten ihre Arbeit vorübergehend einstellen.

Damit ein Entwickler die Übernahme der globalen Konfigurationsparameter in seine Komponente leicht integrieren kann, wurde ein Konfigurations-Subscriber geschrieben. Dieser abonniert automatisch das Themengebiet `BeLi.Config`, über das die globale Konfiguration des Frameworks veröffentlicht wird, und übernimmt automatisch die Konfiguration für die Komponente.

4.4 Plug-ins

Ein Plug-in ist ein Softwaremodul. Es beinhaltet bestimmte Funktionalitäten. Das Plug-in wird von einer Software zur Laufzeit erkannt und eingebunden. Hierdurch ist es also möglich, die Funktionalität einer Software zur Laufzeit zu erweitern.

Das Plug-in-Architekturmuster wird bei der Entwicklung von Software-Systemen eingesetzt, die auf einem einzelnen Computer laufen. Jeder Entwickler ist dadurch in der Lage über vorgegebene Schnittstellen die Funktionalität eines bestehenden Software-Systems zu erweitern. Der Kern des Software-Systems muss dabei nicht quelloffen sein und benötigt vor allem keinerlei Modifikation, da die Erweiterungen nur an den Schnittstellen ansetzen. Wichtig ist, dass immer der Kern des Software-Systems die Plug-ins findet und einbindet. (Ludewig und Lichter 2010, S. 436)

In dem in dieser Arbeit entwickelten Framework wird das Konzept der Plug-in-Architektur erweitert. Die Plug-ins sind hier unabhängiger vom Kern des Software-Systems. Sie können selbstständig starten und ihren Beitrag zu der Funktionalität des Frameworks liefern. Jedes Plug-in benötigt, um starten zu können, lediglich die Kommunikationsschnittstelle in Form des ActiveMQ-Servers. Ein Vorteil an diesem Vorgehen ist, dass, wenn ein Plug-in abstürzt, dies keinen Einfluss auf die Stabilität des Frameworks hat.

Die zentrale Steuerungseinheit ist in der Kombination mit dem Kommunikationsserver der Kern des Frameworks. Ohne die Kommunikationsschnittstelle könnten die Komponenten des Frameworks nicht miteinander kommunizieren. Sie ist somit eine Voraussetzung für die Funktionalität des Frameworks. Die zentrale Steuerungseinheit publiziert in periodischen

Abständen die globale Konfiguration des Frameworks. Ohne diese kann nicht garantiert werden, dass alle Komponenten des Frameworks den gleichen Ausführungszustand haben und sich das Framework somit in einem konsistenten Zustand befindet. Dieser Kern des Frameworks benötigt allerdings kein Wissen darüber, welche Plug-ins zur Laufzeit in das Framework integriert werden sollen oder wo sich diese Plug-ins im Netzwerk befinden. Es sind die Plug-ins, die sich zur Laufzeit über ein Themengebiet in die Funktionalität des Frameworks integrieren und sich so automatisch an das bestehende und laufende System anbinden können.

Arten von Plug-ins

In dem in der vorliegenden Arbeit entwickelten Framework können Plug-ins prinzipiell alles Mögliche sein. Einzige Voraussetzung für ein Plug-in ist, dass es in irgendeiner Form Daten liefert oder verarbeitet, die im Zusammenhang mit der Erkennung von oder der Reaktion auf Ablenkungsfaktoren am Arbeitsplatz stehen. In den folgenden Abschnitten werden einige Plug-ins vorgestellt und ihre Anbindung an das Framework dargestellt. Diese Übersicht stellt lediglich theoretische Annahmen zu möglichen Plug-ins dar und hat keinen Anspruch auf Vollständigkeit.

Input-Plug-in

Ein Input-Plug-in publiziert Daten auf dem ActiveMQ, die zur Erkennung von potentiellen Ablenkungsfaktoren beitragen können. Dabei versucht jedes Input-Plug-in aus seinen eigenen Messdaten einen Ablenkungsfaktor zu bestimmen. Dieser wird mit den Messdaten und einer Gewichtung des Ablenkungsfaktors in einer `JsonMeasurmentData` Nachricht auf dem Themengebiet `BeLi.Input` publiziert.

Zur Erkennung von potentiellen Ablenkungsfaktoren müssen Einflüsse und Reaktionen von sehr unterschiedlicher Art erfasst werden. Die folgende Beschreibung von Input-Plug-in-Kategorien soll einen Überblick über die unterschiedlichen Quellen von potentiellen Ablenkungsfaktoren bieten.

Bio-Sensoren: Bio-Sensoren messen körperliche Reaktionen des Menschen. Dabei kann eine körperliche Reaktion, je nach Art der Reaktion invasiv oder nichtinvasiv ermittelt werden. Ein Eyetracker beispielsweise zeichnet die Augenbewegungen und den Durchmesser der Iris nichtinvasiv durch Infrarotkameras auf. Die Analyse von Blutwerten hingegen kann nur auf invasive Weise erfolgen, da hierfür zwingend Blut aus dem Körper entnommen werden muss.

Die Bio-Sensoren, die in dem Framework eingesetzt werden könnten, sollten durch die körperliche Reaktion des Benutzers auf dessen emotionalen Zustand schließen. Aus diesem könnte dann die Anfälligkeit des Benutzers gegenüber verschiedener Ablenkungsfaktoren ermittelt werden. Eine solche Erkennung des emotionalen Zustandes eines Benutzers ist jedoch sehr komplex und noch nicht hinreichend erforscht. Bernin hat hierzu in »A framework concept for emotion enriched interfaces« (Bernin 2012) erste Ansätze gemacht und verfolgt dieses Thema aktiv weiter.

Software-Sensoren: Neben den Bio-Sensoren sollen Software-Sensoren nach potentiellen Ablenkungsfaktoren suchen oder aus dem aktuellen Kontext und den Kontextwechseln des Benutzers seine Anfälligkeit gegenüber Ablenkungsfaktoren bestimmen. Unter der „Anfälligkeit des Benutzers gegenüber Ablenkungsfaktoren“ ist hier gemeint, dass jede potentielle Ablenkung den aktuellen Arbeitsablauf des Benutzers stark stören würde.

Im Folgenden werden zwei verschiedene Szenarien dargestellt, die den Einsatz von Software-Sensoren verdeutlichen sollen. Zunächst wird ein Software-Sensor betrachtet, der eine Quelle von potentiellen Ablenkungsfaktoren beobachtet. Dieser Software-Sensor könnte beispielweise ein externes System (zum Beispiel einen Bugtracker) überwachen. Wenn hier starke Veränderungen stattfinden, die einen Bezug zu dem Benutzer haben und seine aktuelle Arbeit beeinflussen könnten, publiziert der Sensor dies als hohen Ablenkungsfaktor. Ein anderer Software-Sensor könnte die Desktopumgebung des Benutzers und die Interaktionen des Benutzers auf dieser überwachen. Dieser Software-Sensor überwacht somit die aktuelle Anfälligkeit des Benutzers gegenüber potentieller Ablenkungsfaktoren. Stellt der Sensor fest, dass der Benutzer seit langer Zeit in einem festen Kontext arbeitet, sollte das Framework versuchen jede Ablenkung zu vermeiden. Um dies zu erreichen, publiziert der Software-Sensor eine hohe Anfälligkeit für Ablenkung. Dies bewirkt, dass die anderen Komponenten des Frameworks versuchen, jede Quelle potentieller Ablenkung zu dämpfen und den Benutzer so vor Ablenkung zu schützen. Denn jede Ablenkung würde den Benutzer aus seinem Kontext reißen und er müsste sich später erneut einarbeiten.

Umgebungssensoren: Neben den Sensoren, die direkt den Benutzer und sein Handeln überwachen, könnte es auch Sensoren geben, die die Faktoren aus der Umwelt des Benutzers ermitteln. Umgebungssensoren ermitteln ihre Messdaten durch die Überwachung der räumlichen Umwelt des Benutzers. Temperatur-, Luftfeuchtigkeit- und Lautstärke-Sensoren stellen hierbei nur drei der möglichen Sensoren dar. Alles, was von außen auf den Benutzer einwirken könnte und einen direkten Einfluss auf seine Konzentration

am Arbeitsplatz hat, sollte von dieser Art von Sensor überwacht werden. Ein hohes Ablenkungspotential würde bestehen, sobald ein Wert eine Toleranzgrenze überschreitet. So wäre zum Beispiel eine laute Geräuschkulisse für die Konzentration des Benutzers sicherlich nicht förderlich.

Feedback-Sensoren: Die letzte Kategorie von Input-Plug-ins, die in dieser Arbeit vorgestellt wird, sind die Feedback-Sensoren. Sie ermitteln ihre Messdaten nicht automatisch, sondern nehmen Feedback vom Benutzer entgegen. Die einfachste Form eines Feedback-Sensors wäre ein großer roter Button. Der Benutzer könnte diesen Button drücken, sobald er sich in einer Situation befindet, in der er sich aufgrund zu starker Ablenkung nicht mehr konzentrieren kann. Der Feedback-Sensor würde dann einen sehr hohen Ablenkungswert mit einer sehr hohen Gewichtung auf dem ActiveMQ publizieren.

Kein einzelner Sensor kann alleine eine verlässliche Aussage über den Zustand des Benutzers oder der Umwelt des Benutzers machen. Je mehr Sensoren Messdaten beitragen könnten, desto verlässlicher würde eine Aussage werden. Doch um die Messdaten zu kombinieren und höhere Abstraktionsebenen aus den Messdaten zu erstellen, benötigt man weitere Komponenten. Dies sind die Aktor-Plug-ins, die nun genauer vorgestellt werden.

Aktor-Plug-ins

Eine Erweiterung zu den Input-Plug-ins sind die Aktor-Plug-ins. Ein Aktor-Plug-in verwendet die publizierten Daten anderer Plug-ins und wertet diese aus. Auf diese Weise können neue Daten einer höheren Abstraktionsebene entstehen. Beispielsweise könnte eine lernende Komponente die Messdaten analysieren und die typischen, individuellen körperlichen Reaktionen des Benutzers auf die aufgetretenen Ablenkungsfaktoren ermitteln. Die Erkenntnisse hieraus würden dann einen Beitrag zum Gesamtergebnis leisten und könnten die Fehlerrate bei der Erkennung von Ablenkungspunkten im Arbeitsablauf des Benutzers verringern.

Um die Aktor-Plug-ins besser zu unterscheiden, bietet sich eine Einteilung in Kategorien an. Es folgt die Vorstellung von drei möglichen Kategorien, die für das entwickelte Framework besonders interessant sind:

Aggregator: Der eben vorgestellte Fall, bei dem ein Plug-in Sensordaten zusammenfasst, beschreibt bereits einen Aggregator. Dieser veröffentlicht seine Informationen in Form einer `JsonHighLevelMeasurmentInformation` Nachricht auf dem Themengebiet `BeLi.HighLevelInput`. Der Aggregator verwendet vorhandene Daten und versucht aus ihnen neue, abstraktere Daten zu produzieren.

Controller: Führt das Aktor-Plug-in keine Abstraktion vorhandener Daten durch, sondern verwendet die bereits vorhandenen Daten, um den aktuellen Zustand der Situation zu erfassen und auf diesen zu reagieren, handelt es sich um einen Controller. Hierzu greift der Controller zum Beispiel in den Workflow bestimmter Programme ein, um einen potentiellen Ablenkungsfaktor zu dämpfen. Ein Plug-in, das anhand der aktuellen Anfälligkeit für Ablenkung des Benutzers die Anzeige von Benachrichtigungen steuert, würde Controller genannt werden. Ein Controller kann aber auch die Steuerung des Telefons oder der Türklingel übernehmen.

Logger: Sollen die Messdaten lediglich persistiert werden, kommt ein Logger in Frage. Dieser abonniert verschiedene Themengebiete und speichert jede Nachricht ab. Hierdurch wird es unter anderem möglich, zeitliche Verläufe von Messdaten zu betrachten.

Diese drei Kategorien von Aktor-Plug-ins zeigen, dass selbst hier eine hohe Flexibilität vorhanden ist. Das entwickelte Framework kann durch Plug-ins in seiner Funktionalität flexibel erweitert werden. Jedes Plug-in ist in der Lage, sich selbst zur Laufzeit in den aktuellen Ablauf des Frameworks einzubinden und so einen Beitrag zum Gesamtergebnis zu leisten.

4.5 Evaluierung

Zur Evaluierung des Konzepts des Frameworks wird zunächst ein weiteres Szenario vorgestellt. Anhand dieses Szenarios wird dann auf die Möglichkeiten, die das entwickelte Framework zur Erkennung der potentiellen Ablenkungsfaktoren bietet, eingegangen.

4.5.1 Szenario

Ein Programmierer ist dabei, ein umfangreiches Projekt in Eclipse zu bearbeiten. Er hat hier mehrere Dateien mit unterschiedlichen Klassen geöffnet. Neben Eclipse hat er ein weiteres Fenster, einen Chat, geöffnet. In dem Chatfenster kommuniziert er mit einer anderen Person. Das Szenario ist in der Abbildung 4.10 dargestellt.

Der Programmierer in diesem Szenario ist der Benutzer des Frameworks. An das Framework ist unter anderem ein Eyetracker angeschlossen. Das Plug-in, das den Eyetracker verwendet, erkennt, dass der Benutzer regelmäßig mit seinem Blick zwischen der IDE Eclipse und dem geöffneten Chatfenster hin und her wechselt. Zusätzlich registriert ein Software-Plug-in diesen Wechsel, durch die abwechselnde Aktivierung der Fenster und das Schreiben in jeweils einem der Fenster. Alle anderen Sensoren, die hier nicht weiter erwähnt sind, erkennen keine besonderen Aktivitäten. Der Benutzer ist ruhig und konzentriert.

Die Frage in diesem Szenario, die sich aus der Sicht des Frameworks stellt, lautet: *Wie hoch ist der potentielle Ablenkungsfaktor, der sich durch den Chat darstellt?* Aus dieser Frage ergibt sich direkt eine weitere: *Welche Sensoren können zur der Bestimmung des potentiellen Ablenkungsfaktors beitragen?*

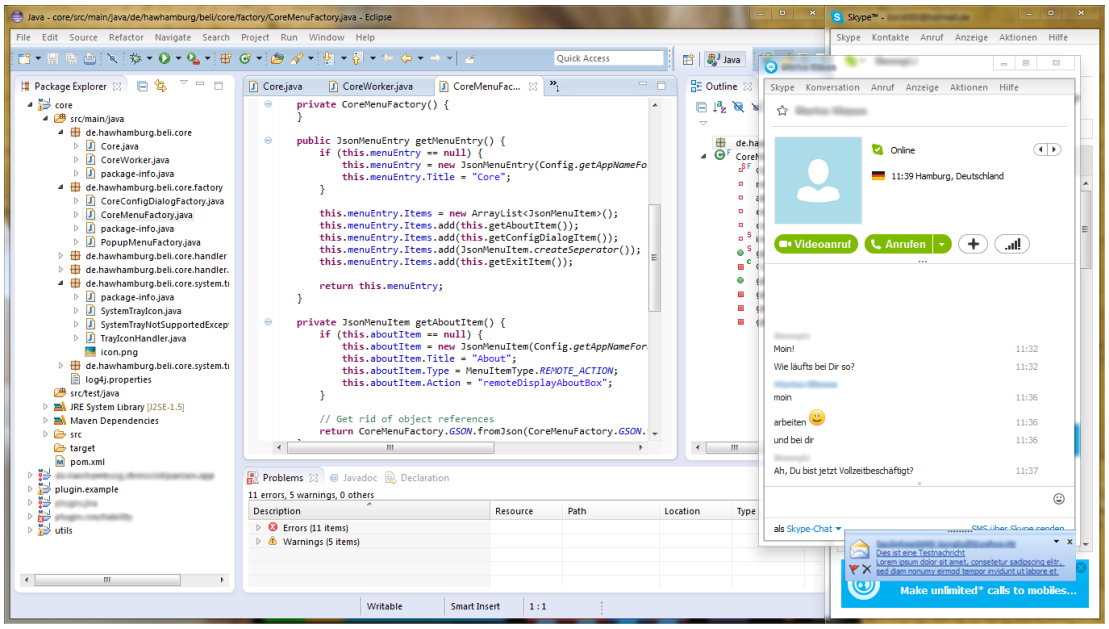


Abbildung 4.10: Darstellung des Desktops eines Entwicklers

4.5.2 Lösungsansatz durch das entwickelte Framework

Das Szenario beschreibt eine Situation, in der auf den Programmierer potentielle Ablenkungsfaktoren wirken. Der Benutzer scheint dieser Ablenkung nachzugeben, in dem er zwischen seiner IDE und dem Chatfenster hin und her wechselt. Dieser Wechsel wird durch den Einsatz eines Eyetrackers und eines Software-Plug-ins eindeutig erkannt.

In den folgenden Abschnitten wird, mit Bezug zu dem beschriebenen Szenario, auf den Eyetracker und die Erkennung des Kontextes, eingegangen. Es wird sich zeigen, wie das Framework eingesetzt werden kann, um die dargestellten Probleme durch Ablenkung zu erkennen und darauf zu reagieren.

Verwendete Plug-ins

Das zuvor beschriebene Szenario sieht einen **Eyetracker** vor, der die Augenbewegungen des Benutzers aufzeichnet. Dies soll durch ein Input-Plug-in realisiert werden, das den Eyetracker

verwendet. Um aus den Augenbewegungen des Benutzers seinen Blick auf einzelne Fenster abzuleiten, muss das Input-Plug-in zusätzlich den Desktop des Benutzers überwachen. So kann das Input-Plug-in den aktuellen Blickpunkt des Benutzers auf die geöffneten und sichtbaren Fenster auf dem Desktop abbilden. Das Plug-in kann also eine Aussage darüber treffen, auf welches Fenster der Benutzer gerade guckt.

Möchte man nicht nur den momentanen Blickpunkt des Benutzers erfassen, sondern bestimmen, wie lange er ein bestimmtes Fenster mit seinem Blick fokussiert, wird der zeitliche Verlauf der Daten benötigt. Nur so ist es möglich, eine Aussage über die Fokussierung des Blickes des Benutzers zu treffen. Hier wird deutlich, dass eine Persistierung der Messdaten zwingend erforderlich ist. Im Abschnitt 3.6 wurde bereits auf die Persistierung in dem entwickelten Framework eingegangen. Es zeigt sich somit, dass das Framework diese Anforderung erfüllt.

Mit Hilfe der gespeicherten Daten ist das Plug-in nun in der Lage einen Wechsel zwischen den geöffneten Fenster festzustellen. Voraussetzung ist dabei, dass der Benutzer eine bestimmte Zeit mit seinem Blick auf einem Fenster geblieben ist. Andernfalls könnte es ein flüchtiger Blick oder Messfehler sein, der nicht beachtet werden darf.

Um die Erkennung des **Fensterwechsels** zu unterstützen, kann ein weiteres Input-Plug-in integriert werden. Im Szenario wurde beschrieben, dass ein Input-Plug-in die Aktivierung der Fenster auf dem Desktop des Benutzers überwacht. Hierdurch kann eindeutig ein Wechsel zwischen den Fenstern festgestellt werden.

Die Kombination der beiden Input-Plug-ins ermöglicht es nun, sowohl aktive Fensterwechsel als auch inaktive Fensterwechsel, die nur durch das Betrachten eines Fensters entstehen, zu erkennen. Hierzu müssen die beiden Plug-ins allerdings miteinander kommunizieren. Um diese Kommunikation von der „normalen“ Kommunikation im System zu unterscheiden, sollten die Plug-ins ein neues Themengebiet für die zusätzliche Kommunikation untereinander erstellen und verwenden. Hierüber wäre dann zum Beispiel der Austausch von IDs der aktuell aktiven Fenster denkbar.

Erkennung des Kontextes

Die Augenbewegung und die Fokussierung einzelner Fenster allein reicht jedoch nicht aus, um eine verlässliche Aussage über einen Kontextwechsel treffen zu können. Die Komponenten müssen auch etwas über den Inhalt der Fenster, also den tatsächlichen Kontext, wissen.

Der Begriff „Kontext“ wurde im Abschnitt 2.1 beschrieben. Zur Erkennung eines Kontextes müssen spezielle Plug-ins entwickelt werden, die jeweils einen Teil des gesamten Kontextes bestimmen können. Der Gesamtkontext im beschriebenen Szenario wäre beispielsweise die aktuelle (Programmier-)Aufgabe des Benutzers. Teilkontexte können sich zum Beispiel in

den einzelnen Fenstern, mit denen der Benutzer arbeitet, befinden. So wären die aktuell geöffneten Klassen in der IDE ein Teilkontext der Aufgabe. Ist nebenher beispielsweise noch ein Internetbrowser geöffnet, in dem nur die Seite der Dokumentation der gerade verwendeten Programmiersprache angezeigt wird, wäre dies ebenfalls ein Teilkontext der Aufgabe.

Um den gesamten Kontext einer Aufgabe zu bestimmen, müsste der Inhalt sämtlicher, vom Benutzer verwendeter Desktopfenster analysiert werden. Mit Hilfe des Frameworks ist dies ein lösbares Problem. Hierzu müsste für jedes Fenster ein Input-Plug-in geschrieben werden, das auf einem definierten Themengebiet den Kontext des jeweiligen Fensters publiziert. Damit verschiedene Fenster einem gemeinsamen Kontext zugeordnet werden können, sollte der Kontext jedes Fensters zusätzlich mit verschiedenen Tags markiert sein. So ist es möglich, Fenster mit einem Kontext, der nicht unbedingt eindeutig ermittelbar ist, mit anderen Fenstern zu verknüpfen. Beispielsweise kann ein Fenster mit dem Kontext „Chat mit Alice“ nicht dem Arbeitskontext zugeordnet werden. Sind diesem Kontext aber Tags wie „Java“, „Arbeitskollegin“ und „Bugtracker-ID #0815“ zugewiesen, ist es wesentlich einfacher das Fenster mit anderen Fenstern in Verbindung zubringen. Die zuvor beschriebenen Input-Plug-ins könnten dann diese Daten (Kontext und Tags) verwenden, um den Kontext des jeweils als aktiv erkannten Fensters mit dem aktuellen Aufgabenkontext abzugleichen. Auf diese Weise wäre eine verlässlichere Aussage über einen Kontextwechsel möglich.

Reaktion bei einem Kontextwechsel

Wie in dieser Arbeit bereits verdeutlicht wurde, sollte Ablenkung bei der Arbeit auf jeden Fall vermieden und somit jeglicher Kontextwechsel verhindert werden. Wenn doch ein Kontextwechsel stattfindet, muss der Benutzer bei diesem Wechsel unterstützt werden. Dies kann zum Beispiel durch die Speicherung des alten Kontextes geschehen, um diesen dem Benutzer später wieder zur Verfügung zu stellen.

Es sind hier verschiedene Reaktionen denkbar. Beispielsweise wäre es möglich ein Aktor-Plug-in zu entwickeln, das Fenster, die nicht zum aktiven Kontext gehören, im Hintergrund hält. So würde potentielle Ablenkung durch unnötige Darstellung von unrelevanten Informationen vermieden werden. Jedoch müsste dafür eindeutig ersichtlich sein, welche Fenster zum aktuellen Aufgabenkontext gehören. Durch die zuvor beschriebenen Input-Plug-Ins zur Erkennung des Kontextes von Fenstern kann dies mit dem in dieser Arbeit entwickelten Framework realisiert werden.

Ein anderer Ansatz wäre die Speicherung eines Kontextes, um diesen nach einem Kontextwechsel wieder aktivieren zu können. Zu diesem Kontext gehören unter anderem alle Fenster und Cursorpositionen, die zur Bearbeitung der aktuellen Aufgabe beigetragen haben. Der

Benutzer wird so bei einem Kontextwechsel unterstützt, indem er beim Zurückkehren zum verlassenen Kontext seine komplette Arbeitsumgebung wieder vorfindet. Er muss sich somit keine Gedanken mehr darüber machen, welche Fenster und Dateien geöffnet waren und an welcher Stelle er genau aufgehört hat.

4.5.3 Fazit

Wie dieses kurze Beispiel gezeigt hat, ist das Framework in der Lage, verschiedene Szenarien abzudecken. Die Flexibilität erlangt das Framework durch den Einsatz von Plug-ins und einer Blackboard-Architektur als zentrale Kommunikationsschnittstelle.

Die Lösungsansätze für das beschriebene Szenario zeigen, wie flexibel das Framework eingesetzt werden kann. Die einzelnen Lösungsansätze lassen sich sehr einfach mit Hilfe des Frameworks umsetzen. Dabei kann jede Teillösung (hier: Eyetracker, Fensteraktivierung, Bewertung der fokussierten Fenster) zunächst für sich, in Form eines Plug-ins, betrachtet werden. Anschließend ist es möglich, die Teillösungen über den ActiveMQ zu verbinden und so ein Subsystem zu erstellen, dass für die Erkennung von Kontextwechseln allgemein zuständig ist.

Mit der vorliegenden Arbeit wurde somit eine Grundlage geschaffen, um ein System zur Erkennung von potentiellen Ablenkungsfaktoren aufzubauen. Dabei konnte jedoch nicht weiter auf die tatsächliche Erkennung von Ablenkungsfaktoren, von körperlichen Reaktionen oder von Kontexten eingegangen werden, da dies den Rahmen der vorliegenden Arbeit überschritten hätte. Auf dem Framework aufbauend können nun unterschiedliche Systeme entworfen werden, die unter anderem potentielle Ablenkungsfaktoren erkennen und darauf reagieren können.

Um dies umzusetzen, müssen zunächst *Sensoren* evaluiert werden, die die Erkennung der entsprechenden Merkmale ermöglichen. Jeder Sensor kann dann als Plug-in an das Framework angebunden werden und seine Messdaten so den anderen Komponenten des Frameworks zur Verfügung stellen. *Aktoren* hätten dann die Möglichkeit auf der Grundlage dieser Daten Auswertungen vorzunehmen und so auf die Ablenkungsfaktoren zu reagieren.

Damit sich die Erkennung der potentiellen Ablenkungsfaktoren besser an das persönliche Empfinden des Arbeitenden anpasst, könnten lernende Komponenten (*Reasoner*) eingesetzt werden. Diese neue Art von Plug-ins nimmt dann sämtliche verfügbaren Daten als Grundlage und versucht hieraus Muster zu erkennen. Die Muster würden dann wiederum zur Bewertung von bestimmten Situationen (Ist die aktuelle Ablenkung gut oder schlecht?) eingesetzt werden. Gleichzeitig wäre es möglich hierüber ein *Profiling* stattfinden zu lassen. Dieses würde ein Profil pro Benutzer anlegen, in dem persönliche Informationen des Benutzers gespeichert

sind, die bei der Bewertung von Ablenkungsfaktoren hilfreich sein können (Berufserfahrung, berufliche Position, etc.).

Für eine verbesserte Benutzbarkeit sollte zusätzlich eine *Speicherung der Konfiguration* ermöglicht werden. Aktuell können zwar die einzelnen Plug-ins über Dialoge konfiguriert werden, aber eine Speicherung dieser Konfiguration über die Ausführungszeit des Frameworks hinaus erfolgt nicht. Um allerdings beispielsweise spezifische Konfigurationen für unterschiedliche Test-Szenarien zu entwickeln, wäre es hilfreich einzelne Konfigurationen speichern und zu einem späteren Zeitpunkt laden zu können.

5 Schluss

In dieser Arbeit wurde das Konzept der Architektur eines Frameworks entwickelt, das zur netzwerkgestützten Erkennung von potentiellen Ablenkungsfaktoren eingesetzt werden kann. Die Architektur hat dabei den Anspruch, möglichst flexibel auf verschiedene Anforderungen, die von zu integrierenden Komponenten gestellt werden, reagieren zu können.

5.1 Zusammenfassung

Unsere Gesellschaft entwickelt sich von einer Industriegesellschaft hin zu einer Wissensgesellschaft. Die Wissensarbeiter müssen dabei mit enormen Mengen an Informationen arbeiten. Durch Ablenkung werden sie aus ihrem Aufgabenkontext gerissen und müssen sich daraufhin erneut in die Aufgabe einarbeiten.

Die Analyse hat gezeigt, dass Ablenkung ein großes Problem darstellt. Besonders Wissensarbeiter sind gefährdet. Die Erkennung von potentiellen Ablenkungsfaktoren ist jedoch schwierig, da sehr viele verschiedene Einflussfaktoren, beispielsweise der aktuelle Aufgabenkontext, die individuelle Stressresistenz und der Ablenkungsgrund, beachtet werden müssen. Um alle Faktoren erkennen zu können, werden unterschiedliche Sensoren benötigt. Es fehlt jedoch ein System, das die gemeinsame Nutzung der unterschiedlichen Sensoren allgemein ermöglicht und die Messdaten automatisch zusammentragen kann. Auf diese Weise könnten die Sensoren eine gemeinsame Datenbasis aufbauen und so zusammen an der Lösung des Problems arbeiten. Ein Framework zur netzwerkgestützten Erkennung von potentiellen Ablenkungsfaktoren muss zudem extrem flexibel auf die technischen Anforderungen verschiedener Sensoren reagieren können (vergleiche Kapitel 2).

Aus den in der Analyse aufgestellten Anforderungen und Zielen (siehe Kapitel 2.5) konnten dann verschiedene Designentscheidungen getroffen werden. Die Architektur des Frameworks ist dabei so ausgelegt, dass Plug-ins, unabhängig von ihrer Programmiersprache und dem Betriebssystem, auf dem sie laufen, angebunden werden können. Dies wurde durch ein verteiltes System (Kapitel 3.1) mit einer kommunikationsbasierten Middleware (Kapitel 3.2) gelöst. Sämtliche Komponenten kommunizieren hier über ein Netzwerk mit der Middleware. Die Middleware selbst ist ein Tupelraum, der speziell durch einen ActiveMQ-Server realisiert

wurde (Kapitel 3.5). Die Middleware umfasste zusätzlich eine Persistenzschicht in Form einer MongoDB, um so eine spätere Auswertung der Daten über die Zeit zu ermöglichen (Kapitel 3.6). Es ist auch die Middleware, die eine lose Kopplung der Komponenten ermöglicht. In dem entwickelten Framework können sich die Komponenten hierdurch zur Laufzeit an das System anbinden oder von diesem entfernen. Zur Interaktion mit dem Benutzer sind verschiedene Schnittstellen, wie beispielsweise ein Kontextmenü in der TNA, vorgesehen (Kapitel 3.7). Hierüber können zum einen die Plug-ins konfiguriert und zum anderen persönliche Daten abgefragt und verschlüsselt übertragen werden.

Die im Design entwickelte Architektur (Kapitel 2.5) konnte durch konkrete Technologieempfehlungen und der Aufstellung der wichtigsten Komponenten weiter verfeinert werden (siehe Kapitel 4). Hier wurde noch einmal genauer auf die Infrastruktur der Middleware, bestehend aus der ActiveMQ und der MongoDB, eingegangen (Kapitel 4.1). Die Beschreibung der zentralen Steuerungseinheit (Kapitel 4.3) und die Möglichkeit zur Anbindung von Plug-ins (Kapitel 4.4) rundeten das Konzept ab.

Neben der rein theoretischen Evaluierung (Kapitel 4.5) wurde eine beispielhafte Implementation der entwickelten Architektur vorgenommen. Ein zusätzliches *Beispiel-Plug-in* demonstriert die Verwendung der in dieser Arbeit spezifizierten Kommunikationsschnittstellen. Die Implementation zeigt, dass das Konzept umsetzbar ist. Teile aus der Implementation können im Anhang nachgeschlagen werden.

5.2 Fazit

Im Kapitel 2.5 wurden verschiedene Anforderungen an das Framework aufgestellt. Diese waren:

1. Verwendung verschiedener Sensorarten
2. Plug-ins sind über das Netzwerk angebunden
3. Verschlüsselung
4. Lose Kopplung
5. Freie Konfigurierbarkeit
6. Persistierung

Insbesondere mit Hilfe der kommunikationsbasierten Middleware konnten viele der aufgestellten Anforderungen abgedeckt werden. Das Framework gewinnt hierdurch an Unabhängigkeit gegenüber der Implementation einzelner Plug-ins. Diese werden **über das Netzwerk**

angebunden (Anforderung 2). Auf diese Weise kann das Framework **verschiedene Sensorarten verwenden** (Anforderung 1).

Die Implementation jeder Komponente sollte dabei möglichst keine Abhängigkeiten gegenüber anderer Komponenten des Frameworks haben. Da diese Anforderung jedoch nicht nur von dem Framework erfüllt werden kann, muss jeder Plug-in-Entwickler selbst darauf achten, dass sein Plug-in eine **lose Kopplung** (Anforderung 4) zu den anderen Plug-ins aufweist.

Sollen Plug-ins mit dem Benutzer interagieren und Informationen austauschen, werden diese Informationen über den Tupelraum verschickt. Da hier jeder alle Informationen lesen kann, musste eine Möglichkeit geschaffen werden, die Daten zu verschlüsseln. Die Offenheit der Kommunikationsschnittstelle bezüglich der Nachrichtenformate lässt genug Raum, um einzelne Nachrichten oder Nachrichtenteile zu **verschlüsseln** (Anforderung 3). Hierfür wurde in dieser Arbeit das RSA-Kryptoverfahren vorgeschlagen.

Ebenfalls mit Hilfe der Middleware kann die **freie Konfiguration** (Anforderung 5) der einzelnen Komponenten erfolgen. Durch Konfigurationsnachrichten teilen die Komponenten der zentralen Steuerungseinheit ihre aktuelle Konfiguration mit. Die zentrale Steuerungseinheit zeigt die aktuelle Konfiguration dann dem Benutzer an und dieser kann die Konfiguration anpassen. Durch die Verwendung der gleichen Nachrichten mit derselben ID erfolgt die Mitteilung der geänderten Konfiguration an die entsprechende Komponente.

Eine zusätzliche **Persistenzschicht** (Anforderung 6), die ebenfalls in die Middleware integriert und durch eine MongoDB realisiert wurde, ermöglicht die Auswertung von zeitlichen Verläufen der Messdaten. So können später Algorithmen die Daten auswerten und versuchen Muster zu erkennen.

In der vorliegenden Arbeit konnte somit gezeigt werden, dass die entwickelte Architektur nahezu alle gestellten Anforderungen erfüllt. Lediglich die lose Kopplung kann nicht vollständig durch das entwickelte Framework garantiert werden. Diese Anforderung muss zusätzlich jeder Plug-in-Entwickler selbst beachten. In einer beispielhaften Implementation (siehen Anhang C) wurde zusätzlich gezeigt, dass eine Umsetzung des Konzepts möglich ist.

5.3 Ausblick

„Aktuelle technologische Entwicklungen, insbesondere in der Sensorik, ermöglichen eine präzise Wahrnehmung der Umgebung, der Intention oder des kognitiven oder emotionalen Zustands des Nutzers. Dadurch entstehen technische Lösungen, die maßgeschneidert auf den jeweiligen Kontext und den individuellen Nutzer reagieren können und ihm so ein besseres

Nutzungserlebnis und eine angemessene Unterstützung bieten."[30] Die vorliegende Arbeit konnte das vom Bundesministerium für Bildung und Forschung (BMBF) aufgestellte Ziel noch nicht vollständig erfüllen. Das entwickelte Framework macht aber einen Schritt in diese Richtung und bietet eine Grundlage für weitere Forschung. Mit dem entwickelten Framewok kann direkt in die Arbeitsabläufe des Benutzers eingegriffen und so die Konzentration des Benutzers unterstützt werden. Dies kann jedoch auch als Eingriff in die Privatsphäre gesehen werden. An dieser Stelle muss weitere Forschung betrieben werden, um auf interdisziplinärer Ebene die Akzeptanz und die rechtlichen Möglichkeiten einer solchen Software zu bestimmen.

Besonders in der Automobilbranche ist diese Grenze zur Autonomie von Software (hier im Fahrzeug) für den Fahrer nicht mehr deutlich sichtbar. Bremssysteme beispielsweise agieren selbstständig und regeln die Bremskraft individuell pro Reifen, um dem Fahrzeug die höchst mögliche Stabilität auf der Straße zu geben. Assistenzsysteme bremsen sogar bereits komplett selbstständig, wenn eine Gefahrensituation naht und der Fahrer nicht selbst reagiert. Den meisten Fahrern ist diese Autonomie ihres Fahrzeugs jedoch nicht bewusst. Die Frage, die sich an dieser Stelle stellt, ist: *Wenn dem Anwender nicht bewusst ist, dass ein System seine Arbeitsabläufe anhand von externen Einflüssen oder seines aktuellen emotionalen Zustandes beeinflusst, wird er das System dann eher akzeptieren?*

Geht man hier noch einen Schritt weiter und nimmt ein adaptives System an, das sich ein Profil seines Benutzers erstellt, um so später angepasst reagieren zu können, wird es dann Probleme mit Vorratsdatenspeicherung und Datenschutz geben? Oder ist dies durch eine Vereinbarung zwischen dem Benutzer und der Software abgegolten?

Das Bundesministerium für Bildung und Forschung hat im Dezember 2012 geschrieben: „[Das] Eingreifen der Technik in die Selbstbestimmung des Menschen und die Verletzung seiner Privatsphäre [müssen] von vornherein ausgeschlossen werden."[31] Hieran ist schon deutlich zu erkennen, dass selbst der Bund dieses Verhalten von Software nicht dulden will.

Dies sind nur einige wenige Denkanstöße, die aus der vorliegenden Arbeit hervorgehen und bei weiteren Entwicklungen des Frameworks bedacht werden müssen. Andere Wissenschaftler mögen anders denken und so auf neue Ideen kommen. Denn wie Albert Einstein einst sagte:

„Eine neue Art von Denken ist notwendig, wenn die Menschheit weiterleben will."

[30] Aus http://www.bmbf.de/foerderungen/20972.php, zuletzt besucht am 24.07.2013
[31] Aus http://www.bmbf.de/foerderungen/20972.php, zuletzt besucht am 24.07.2013

A ActiveMQ Themengebiete

Je mehr Plug-ins an das Framework angeschlossen werden, desto mehr Nachrichten werden verschickt. Damit nicht jede Komponente sämtliche Nachrichten verarbeiten muss, um aus der Masse die für sich relevanten Nachrichten herauszufiltern, wurde die Kommunikation in Themengebiete eingeteilt. Diese Themengebiete heißen auf der ActiveMQ „Topics“[32].

Im Folgenden werden alle in dieser Arbeit implementierten Themengebiete beschrieben. Der Name des Themengebiets setzt sich immer aus einem Prefix (*BeLi*) und dem Namen des Themengebiets zusammen. Die jeweilige Kurzbeschreibung soll helfen, einen schnellen Überblick über das entsprechende Einsatzgebiet des jeweiligen Themengebiets zu bekommen.

A.1 Themengebiet für die Frameworkskonfiguration

Name: BeLi.Config

Kurzbeschreibung: Dient der Veröffentlichung der globalen Frameworkskonfiguration.

Auf dem Themengebiet `BeLi.Config` werden Nachrichten veröffentlicht, die globale Konfigurationsvariablen enthalten. Jede Konfigurationsnachricht ist mit einem Zeitstempel versehen, der sich erst ändert, wenn sich die Konfiguration geändert hat. Dadurch kann beim Empfang einer Konfigurationsnachricht schnell entschieden werden, ob Änderungen gegenüber der aktuellen Konfiguration vorliegen oder ob diese Nachricht verworfen werden kann.

A.2 Themengebiet für Messdaten

Name: BeLi.Input

Kurzbeschreibung: Dient der Veröffentlichung der Messdaten von Input-Plug-ins.

Jedes Input-Plug-in soll seine Messdaten auf dem Themengebiet `BeLi.Input` veröffentlichen. Damit könnte garantiert werden, dass alle Komponenten des Frameworks sämtliche

[32] englisch für „Themengebiete“

Messdaten erhalten können. Wichtig ist hier, dass jede Nachricht einen Zeitstempel mit dem Zeitpunkt der Erfassung des Messwertes enthält. Nur so kann später eine Synchronisierung der Messwerte stattfinden.

A.3 Themengebiet für abstrahierte Messdaten

Name: BeLi.HighLevelInput

Kurzbeschreibung: Dient der Veröffentlichung der weiterverarbeiteten Messdaten.

Einige Komponenten des Frameworks sollen die erfassten Messdaten analysieren und auswerten. Die Ergebnisse dieser Analysen und Auswertungen können ebenfalls als „Messdaten“ angesehen werden. Dies sind allerdings Messdaten einer höheren Abstraktionsebene und sollten daher getrennt von den Messdaten der Input-Plug-ins behandelt werden. Für die Veröffentlichung von abstrahierten Messdaten steht das Themengebiet `BeLi.HighLevelInput` zur Verfügung.

Der Vorteil an abstrahierten Messdaten wäre, dass die Analyse und Auswertung der Messdaten der Input-Plug-ins nur von wenigen Komponenten des Frameworks durchgeführt werden müsste. Alle anderen Komponenten bekämen einen direkten Zugriff auf das Ergebnis der Auswertung. Die Rechenlast, die benötigt würde, um die Auswertung durchzuführen, könnte so auf einen bestimmten oder auf mehrere Rechner verteilt werden.

A.4 Themengebiet für die Menü Integration

Name: BeLi.SystemTray.Menu

Kurzbeschreibung: Dient der Veröffentlichung von Menüstrukturen zur Einbindung eines Menüpunktes inklusive Untermenüpunkten für Plug-ins.

Durch die Verteilung der Komponenten des Frameworks in einem Netzwerk, verliert der Benutzer schnell die Möglichkeit, mit einzelnen Komponenten zu interagieren oder diese gezielt zu konfigurieren. Deshalb wird dem Benutzer von der zentralen Steuerungseinheit ein Symbol in der TNA inklusive eines Kontextmenüs zur Verfügung gestellt (siehe Abbildung A.1). Um nun auch mit den einzelnen Plug-ins zu interagieren oder diese zu konfigurieren, kann jedes Plug-in einen eigenen Menüpunkt in das Kontextmenü integrieren. Dieser Menüpunkt öffnet dann ein Untermenü, das von dem jeweiligen Plug-in nach Belieben befüllt werden kann. Die Integration in das Menü kann über das Themengebiete `BeLi.SystemTray.Menu` angefordert werden.

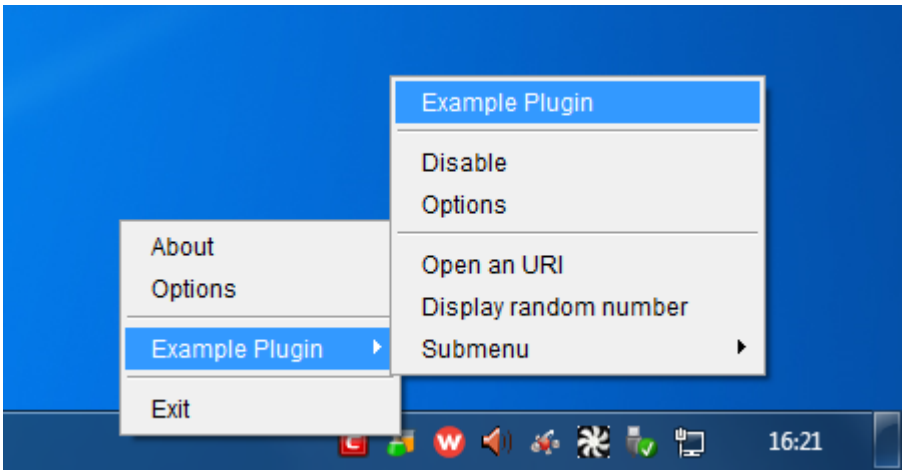


Abbildung A.1: Darstellung des Kontextmenüs in der Taskbar Notification Area (TNA)

A.5 Themengebiet zur Ausführung von Remote-Funktionen

Name: BeLi.SystemTray.Menu.RemoteAction

Kurzbeschreibung: Dient der Veröffentlichung von Funktionsaufrufen, die bei einem Plug-in ausgeführt werden sollen. Dies sind in der Regel Funktionen, die das Plug-in zuvor mit einem Menüpunkt verknüpft hat.

Die Plug-ins sind im Netzwerk verteilt und sollen lose an das Framework angebunden werden. Das bedeutet, dass ein Plug-in seinen Standord im Netzwerk ändern kann, ohne dass es eine andere Komponente davon in Kenntnis setzen muss. Um in diesem Kontext Funktionen von Plug-ins durch den Benutzer aufrufen zu lassen, muss ein entfernter Funktionsaufruf stattfinden. Dieser Funktionsaufruf soll allerding unabhängig von der aktuellen Position des Plug-ins im Netzwerk sein. Hierzu kann der ActiveMQ eingesetzt werden. Das Themengebiet `BeLi.SystemTray.Menu.RemoteAction` stellt dabei den Bereich auf dem ActiveMQ bereit, auf dem die Remote-Funktionsaufrufe veröffentlich werden. Jedes Plug-in, das entfernte Funktionsaufrufe zulässt, muss dieses Themengebiet abonnieren. Auf diese Weise können die Plug-ins die entfernten Funktionsaufrufe empfangen.

A.6 Themengebiet für die Konfigurationsdialoge

Name: BeLi.ConfigDialog

Kurzbeschreibung: Dient der Veröffentlichung von Nachrichten zum Anzeigen und Speichern von Konfigurationsdialogen.

Die Benutzer sollen die Möglichkeit haben, Plug-ins zu konfigurieren. Die zentrale Steuerungseinheit stellt dem Benutzer hierfür einen Konfigurationsdialog dar (siehe Abbildung A.2). Jedes konfigurierbare Plug-in kann hier einen Tab integrieren, in dem die Konfigurationsoptionen des jeweiligen Plug-ins angezeigt werden. Zur Übermittlung der Konfigurationsoptionen soll das Themengebiet `BeLi.ConfigDialog` verwendet werden.

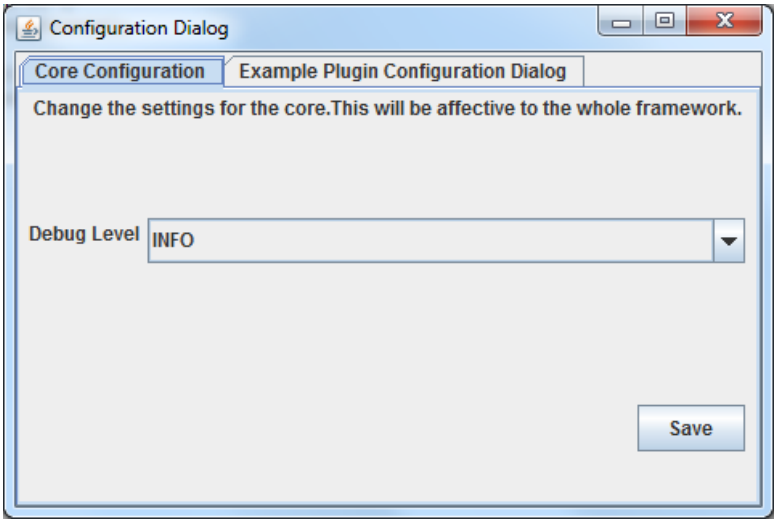


Abbildung A.2: Konfigurationsdialog mit Tabs

A.7 Themengebiet für die Informationsdialoge

Name: BeLi.InfoDialog

Kurzbeschreibung: Dient der Veröffentlichung von Nachrichten, die dem Benutzer in einem Informationsdialog (Fenster) angezeigt werden sollen.

Um dem Benutzer eine Information anzuzeigen, kann beispielweise ein Fenster angezeigt werden, dass der Benutzer bestätigen muss, damit es wieder verschwindet (siehe Abbildung A.3). Diese Möglichkeit der Informationsdarstellung sollte jedoch nur für wirklich wichtige oder

durch den Benutzer angeforderte Informationen genutzt werden, da diese Form der Darstellung den Arbeitsprozess des Benutzers unterbrechen kann. Das Themengebiet `BeLi.InfoDialog` bietet die Schnittstelle für diese Art der Informationsübermittlung. Hier publizierte Nachrichten werden von der zentralen Steuerungseinheit verarbeitet und als Informationsdialog dargestellt.

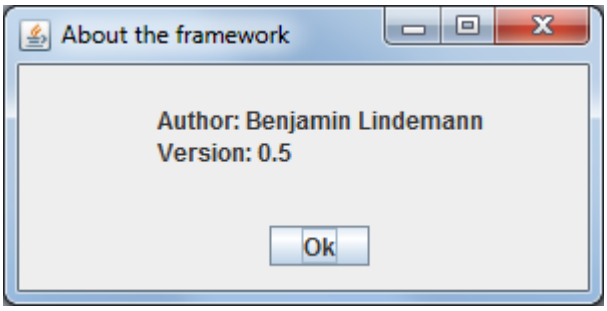


Abbildung A.3: Informationsdialogfenster

A.8 Themengebiet für Pop-up-Nachrichten

Name: BeLi.SystemTray.Popup

Kurzbeschreibung: Dient der Veröffentlichung von Nachrichten, die als Systembenachrichtigung (Pop-up) angezeigt werden sollen.

Die Darstellung einer Systemmeldung in Form eines Pop-ups (siehe Abbildung A.4) ist in der Regel nicht im direkten Sichtbereich des Benutzers. Diese Informationsdarstellung hat somit weniger Einfluss auf den Arbeitsfluss des Benutzers. Zudem verschwindet die Nachricht nach kurzer Zeit von selbst. Über das Themengebiet `BeLi.SystemTray.Popup` können Plug-ins drei verschiedene Arten von Meldungen anzeigen lassen: Informationsmeldungen, Warnungen und Fehlermeldungen.

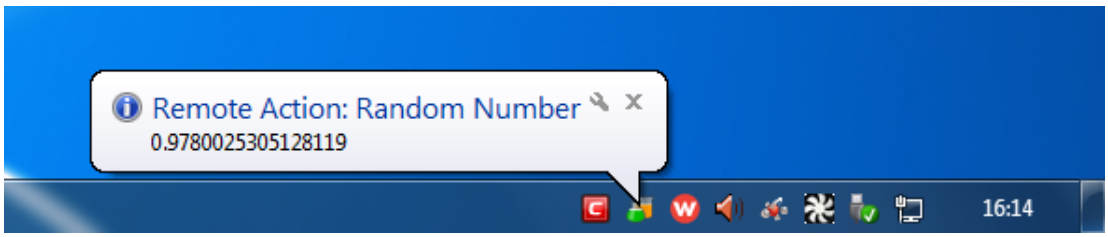


Abbildung A.4: Informations-Pop-up

A.9 Themengebiet für die Anmeldedialoge

Name: BeLi.LoginRequest

Kurzbeschreibung: Dient der Veröffentlichung der Anfrage zur Anzeige eines Logindialogs und der Antwort mit den (verschlüsselten) Logindaten.

Manche Input-Plug-ins müssen sich an einem externen System (zum Beispiel einem Bugtracker) anmelden, um Auswertungen über die dort gespeicherten Informationen machen zu können. Um diese Daten zur Laufzeit bei den Benutzern abzufragen, kann ein Dialog angezeigt werden, in den der Benutzer seine Anmeldedaten einträgt (siehe Abbildung A.5). Die Anmeldedaten werden dann von der zentralen Steuerungseinheit verschlüsselt publiziert, so dass nur das anfragende Plug-in Zugriff auf diese Daten erhällt. Die Anfrage zur Anzeige des Dialogs und die Übertragung der verschlüsselten Anmeldedaten erfolgt über das Themengebiet `BeLi.LoginRequest`.

Zur einfacheren Nutzung dieser Funktionalität wurde ein `LoginHandler` implementiert. Dieser liegt in dem Paket `de.hawhamburg.beli.handler`.

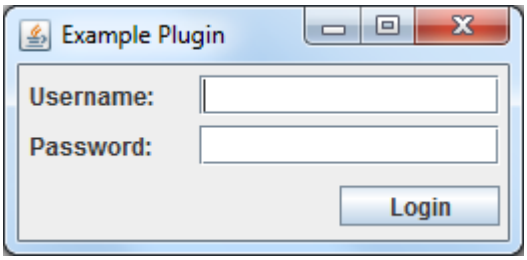


Abbildung A.5: Anmeldedialogfenster

B JSON-Nachrichten

Zur Übermittlung von Informationen über den ActiveMQ werden Nachrichten verwendet. Das entwickelte Framework ist auf die Verarbeitung von Nachrichten im JSON-Format ausgelegt. Jede Nachricht muss dabei eine ID und eine Versionsnummer haben. Diese sind in der Standardnachricht `JsonMessage` implementiert. Jede JSON-Nachricht im System sollte von dieser Standardnachricht abgeleitet sein.

Nachfolgend werden die einzelnen JSON-Nachrichten und die jeweiligen Zugehörigkeiten zu den Themengebieten definiert und kurz der Einsatzzweck der Nachricht beschrieben. Mehr zur Implementierung kann in dem Tutorial im Anhang C nachgelesen werden.

B.1 Nachricht zur Übermittlung der Frameworkskonfiguration

Name: JsonFrameworkConfiguration

Themengebiet: BeLi.Config

Kurzbeschreibung: Diese Nachricht beinhaltet die aktuellen Konfigurationsoptionen des Frameworks, die für alle Komponenten gelten.

Eine Konfigurationsoption des Frameworks, die für alle Komponenten gelten muss, ist zum Beispiel der aktuelle Ausführungszustand des Frameworks. Er beschreibt, ob die Komponenten des Frameworks aktuell arbeiten oder pausieren sollen oder ob sich das Framework gerade vollständig herunterfährt.

Die Nachricht `JsonFrameworkConfiguration` wird von der zentralen Steuerungseinheit periodisch auf dem Themengebiet `BeLi.Config` veröffentlicht. Auf diese Weise sind immer alle, auch neu hinzugekommene, Plug-ins auf dem aktuellen Stand.

B.2 Nachrichten zur Veröffentlichung von Messdaten

Name: JsonMeasurmentData, JsonHighLevelMeasurmentInformation

Themengebiete: BeLi.Input, BeLi.HighLevelInput

Kurzbeschreibung: Diese Nachrichten beschreiben einen berechneten potentiellen Ablenkungslevel.

Jedes Input-Plug-in soll versuchen aus den ihm zur Verfügung stehenden Informationen einen potentiellen Ablenkungslevel zu berechnen. Dieser Wert beschreibt die Möglichkeit, dass sich (1) der Benutzer von einer dem Input-Plug-in verfügbaren Informationsquelle ablenken lässt oder (2) der Benutzer eine körperliche Reaktion gezeigt hat, die ihn für potentielle Ablenkungsfaktoren anfällig werden lässt. Mit Hilfe der Nachricht `JsonMeasurmentData` können Input-Plug-ins diese Informationen auf dem Themengebiet `BeLi.Input` bereit stellen. Neben dem potentiellen Ablenkungslevel kann eine Gewichtung zu dem Messwert angegeben werden. Die Gewichtung soll eine Aussage darüber treffen, wie verlässlich der Wert des potentiellen Ablenkungslevels ist.

Aus einer Menge von Messdatennachrichten können dann weitere Berechnungen angestellt werden. Aktor-Plug-ins können diese Daten nutzen, um eine höhere Abstraktionsebene der Messdaten zu erstellen. Zum Beispiel könnten sie aus der Menge an Messdaten in einem definierten Zeitabschnitt einen aggregierten Messwert berechnen. Das Ergebnis der Berechnung veröffentlicht das Plug-in dann auf dem Themengebiet `BeLi.HighLevelInput` in Form der Nachricht `JsonHighLevelMeasurmentInformation`.

B.3 Nachricht zur Einbindung von Menüeinträgen

Name: JsonMenuEntry

Themengebiet: BeLi.SystemTray.Menu

Zugehörige Nachricht: JsonMenuItem

Kurzbeschreibung: Diese Nachricht definiert einen Menüeintrag mit zugehörigem Untermenü und den hierin enthaltenen Menüpunkten.

Jede Komponente des Frameworks kann einen eigenen Menüpunkt im Kontextmenü haben, das mit dem Symbol in der TNA verknüpft ist. Hierzu muss eine `JsonMenuEntry`-Nachricht,

die die Menüstruktur definiert, auf dem Themengebiet `BeLi.SystemTray.Menu` veröffentlicht werden. Die Nachricht `JsonMenuEntry` ist dabei ein Container für die Untermenüpunkte. Jeder Untermenüpunkt ist über eine `JsonMenuItem`-Nachricht definiert, die in dem Container enthalten ist.

B.4 Nachricht zur Ausführung von Remote-Funktionen

Name: JsonRemoteAction

Themengebiet: BeLi.SystemTray.Menu.RemoteAction

Kurzbeschreibung: Diese Nachricht übermittelt die Anfrage zur Ausführung einer Funktion eines Plug-ins.

Jedes Plug-in kann in jedem Menüeintrag eine Aktion hinterlegen, die auf dem Plug-in ausgeführt werden sollen. Zum Beispiel könnte ein Menüpunkt mit dem Titel „Über das Plug-in“ mit der Anzeige eines Dialogfensters verknüpft sein, in dem Informationen über das Plug-in angezeigt werden. Um die Anzeige der Informationen anzufordern, muss eine Funktion beim Plug-in ausgeführt werden, die die Anzeige des Dialogfensters auslöst. Mit Hilfe der Nachricht `JsonRemoteAction`, die auf dem Themengebiet `BeLi.SystemTray.Menu.RemoteAction` veröffentlicht wird, kann eine solche entfernte Funktion aufgerufen werden.

B.5 Nachricht zur Anzeige und Speicherung von Konfigurationsdialogen

Name: JsonConfigDialog

Themengebiet: BeLi.ConfigDialog

Zugehörige Nachricht: JsonConfigDialogItem

Kurzbeschreibung: Diese Nachricht repräsentiert einen Konfigurationsdialog und die Konfigurationsvariablen eines Plug-ins, sowie die aktuellen Werte der Variablen.

Möchte ein Plug-in einen Konfigurationsdialog anzeigen lassen, muss es auf dem Themengebiet `BeLi.ConfigDialog` eine `JsonConfigDialog`-Nachricht veröffentlichen. Empfängt

die zentrale Steuerungseinheit diese Nachricht, zeigt sie den Konfigurationsdialog an, wobei der Tab für das entsprechende Plug-in ausgewählt ist.

Die Nachricht `JsonConfigDialog` dient als Container für die einzelnen Konfigurationsoptionen. In dieser Nachricht können neben den einzelnen Optionen zusätzlich der Titel des Tabs und ein Beschreibungstext angegeben werden. Jede Option wird als eine eigene Nachricht angegeben, die jeweils in dem Container enthalten sind. Die Nachricht `JsonConfigDialogItem` definiert eine solche Konfigurationsoption.

B.6 Nachrichten zur Anzeige von Informationen

Name: JsonInfoDialog, JsonPopupMessage

Themengebiete: BeLi.InfoDialog, BeLi.SystemTray.Popup

Kurzbeschreibung: Diese Nachrichten veranlassen jeweils die Anzeige einer Information für den Benutzer.

Jedes Plug-in kann dem Benutzer auf unterschiedlichen Wegen Informationen über sich und seinen Zustand zur Verfügung stellen. Zwei dieser Wege sind (1) die Anzeige eines Dialogfenster mit Informationen, den der Benutzer aktiv bestätigen muss und (2) die Anzeige eines Pop-up-Fensters im Bereich der TNA, dass nach einer bestimmten Zeit von selbst wieder verschwindet. Der direkte Weg über das Dialogfenster wird den Benutzer voraussichtlich in seiner Arbeit unterbrechen und sollte nur selten benutzt werden. Der indirekte Weg sollte immer dann gewählt werden, wenn die Information für den Benutzer interessant sein kann, es aber nicht zwingend notwendig ist, dass er sie auch bewusst wahrnimmt.

Für jeden der beiden Wege gibt es eine Nachricht, mit der die Anzeige der Information bei der zentralen Steuerungseinheit angefordert werden kann. Die Anzeige des Dialogfensters ist mit der Nachricht `JsonInfoDialog` implementiert, die auf dem Themengebiet `BeLi.InfoDialog` veröffentlich wird. Damit die zentrale Steuerungseinheit ein Pop-up-Fenster im Bereich der TNA einblendet, erwartet sie eine `JsonPopupMessage`-Nachricht auf dem `BeLi.SystemTray.Popup`-Themengebiet.

B.7 Nachricht zur Anzeige eines Anmeldedialogs und zur Übermittlung der Anmeldedaten

Name: JsonLoginRequest

Themengebiet: BeLi.LoginRequest

Kurzbeschreibung: Diese Nachricht stellt die Anfrage zur Einforderung von Anmeldedaten vom Benutzer und übermittelt die (verschlüsselten) Anmeldedaten.

Manche Plug-ins benötigen Anmeldedaten, um sich an einem externen System, zum Beispiel einem Bugtracker, anzumelden. Damit diese Anmeldedaten nicht im Plug-in gespeichert werden müssen, können diese individuell vom Benutzer abgefragt werden. Empfängt die zentrale Steuerungseinheit über das Themengebiet `BeLi.LoginRequest` eine `JsonLoginRequest`-Nachricht, zeigt sie dem Benutzer einen Anmeldedialog an. Die eingegebenen Anmeldedaten werden verschlüsselt wieder in die eingegangene `JsonLoginRequest`-Nachricht gepackt und erneut auf dem Themengebiet `BeLi.LoginRequest` veröffentlicht. Von dort kann das anfragende Plug-in die Anmeldedaten dann abgreifen.

C Einführung in die Plug-in-Entwicklung für das Framework

In den folgenden Abschnitten wird erläutert, wie man ein Plug-in für das in dieser Arbeit entwickelte Framework in Java programmiert. Als Beispiel dient hier ein Plug-in mit dem Namen *Example*.

C.1 Einstiegspunkt und Worker-Thread

Als Einstiegspunkt sollte jedes Plug-in eine Klasse mit dem Namen des Plug-ins enthalten. Diese Klasse trägt das Postfix *Plugin* im Namen. Für das Beispiel Plug-in sieht das wie folgt aus:

```
Package:    de.hawhamburg.beli.plugin.example
Klasse:     ExamplePlugin
```

Die Klasse `ExamplePlugin` sollte lediglich die *Main*-Methode beinhalten und einen Worker-Thread starten. Es empfiehlt sich eine zweite Klasse zu erstellen, die den Worker-Thread und die eigentliche Logik implementiert und diese in das gleiche Paket zu legen. Diese Klasse implementiert das Java-Interface `java.lang.Runnable`. Im Beispiel folgt also:

```
Package:     de.hawhamburg.beli.plugin.example
Klasse:      ExamplePluginWorker
Implements:  java.lang.Runnable
```

```
package de.hawhamburg.beli.plugin.example;

public class ExamplePlugin {
  public static void main(String args[]) {
    new Thread(new ExamplePluginWorker()).start();
  }
}
```

Listing C.1: *ExamplePlugin*-Klassenkonstrukt

Die Verteilung der Aufgaben des Plug-ins ist im Worker-Thread implementiert. Während der Initialisierung erstellt der Worker-Thread Objekte der benötigten Klassen und ruft während seiner Lebenszeit deren Methoden zyklisch auf. Der Lebenszyklus des Worker-Threads[33] ist an den Ausführungszustand des Frameworks gekoppelt. Zusätzlich gibt es einen lokalen Ausführungszustand, der nur für das Plug-in und seine Komponenten gilt. Dieser lokale Ausführungszustand muss zum Ende des Lebenszyklus des Worker-Threads auf STOPPED gesetzt werden. Die run-Methode des Worker-Threads hat also das folgende Grundkonstrukt:

```
1  public void run() {
2    try {
3      /****** ###### EXECUTING PHASE ###### ******/
4      // execute until the framework is shutting down completely
5      while (Config.getGlobalExecutionState() != ExecutionState.
           STOPPED) {
6        // Execute the main stuff only, if the framework is in an
             active
7        // state
8        if (Config.getGlobalExecutionState() == ExecutionState.RUNNING
             ) {
9          // Business logic comes here
10       }
11     }
12     /****** ###### END EXECUTING PHASE ###### ******/
13   } catch (Exception e) {
14     ExamplePluginWorker.DEBUGGER.exception(e);
15   }
16
17   /****** ###### SHUTDOWN PHASE ###### ******/
18   Config.setLocalExecutionState(ExecutionState.STOPPED);
19   ExamplePluginWorker.DEBUGGER.info("Exiting Example Plugin Worker")
       ;
20   System.exit(0);
21   /****** ###### END SHUTDOWN PHASE ###### ******/
22 }
```

Listing C.2: Grundkonstrukt der *run*-Methode eines *Plug-in Worker-Threads*

[33] und aller anderen Thread Klassen im Plug-in

C.2 Menüintegration und Anzeige von Konfigurationsdialogen

Jedes Plug-in kann seinen eigenen Menüpunkt einrichten. Unter diesem Menüpunkt können dem Benutzer verschiedene Informationen oder Konfigurationsmöglichkeiten angeboten werden (siehe auch Kapitel 4.3). Zusätzlich gibt es die Möglichkeit, über einen Menüpunkt einen Konfigurationsdialog anzeigen zu lassen.

Zur einfacheren Implementierung werden im *Utils*-Paket verschiedene Interfaces und abstrakte Klassen bereitgestellt. Die Interfaces sind:

PluginConfig: Dies ist die einfachste Form einer Plug-in Konfiguration. Es müssen zumindest der Plug-in Name und die aktuelle Version per Getter erreichbar sein.

PluginConfigWithMenu: Die Erweiterung der einfachen Form bietet Funktionen zum Erstellen des Menüpunktes und der Prüfung auf valide Remote-Funktionsaufrufe.

PluginConfigWithConfigDialog: Die Menü-Integration kann um die Darstellung eines Konfigurationsdialogs erweitert werden. Das Interface definiert dafür zwei Funktionen. Zunächst kann ein Konfigurationsdialog erstellt werden. Die zweite Funktion prüft dann, ob die Nachricht zur Speicherung einer Konfiguration für das entsprechende Plug-in bestimmt ist.

Zusätzlich zu den angebotenen Interfaces, wurden zwei abstrake Klassen implementiert. Diese sollen bei der schnellen Umsetzung von neuen Implementationen helfen. Die abstrakten Klassen sind:

SimplePluginConfigWithMenu: In dieser abstrakten Klasse wird eine einfache Implementation der Funktion `isValidRemoteAction` angeboten.

SimplePluginConfigWithConfigDialog: Für die direkte Verwendung bietet diese Klasse eine Implementation der Funktion `isValidConfigDialogSaveAction` an. Da diese Klasse von `SimplePluginConfigWithMenu` erbt, bietet sie auch deren Implementationen an.

Für das Beispiel-Plug-in wurde der volle Funktionsumfang (Menü und Konfigurationdialog) gewählt. Die Erstellung des Menüpunktes und des Konfigurationsdialogs wurde jeweils in eine Factory-Klasse ausgelagert. Zudem wurde die Konfigurationsklasse als Singleton implementiert. Die Klassendeklaration der Konfigurationsklasse sieht wie folgt aus:

```java
public class ExamplePluginConfig extends
    SimplePluginConfigWithConfigDialog {

  /**
   * The Menu Builder is responsible for the creation of the items
       for the
   * system tray menu in json format
   */
  private ExamplePluginMenuBuilder pluginMenuBuilder;

  /**
   * The Config Dialog Builder is responsible for the creation of
       the items
   * for the config dialog frame in json format
   */
  private ExamplePluginConfigDialogBuilder configDialogBuilder;

  /**
   * The one and only instance of this class
   */
  private static ExamplePluginConfig instance;

  /**
   * The one and only instance of this class
   *
   * @return The singleton
   */
  public static ExamplePluginConfig getInstance() {
    if (ExamplePluginConfig.instance == null) {
      ExamplePluginConfig.instance = new ExamplePluginConfig();
    }

    return ExamplePluginConfig.instance;
  }

  private ExamplePluginConfig() {
    this.pluginMenuBuilder = new ExamplePluginMenuBuilder();
    this.configDialogBuilder = new ExamplePluginConfigDialogBuilder
        ();
  }
```

```
37
38    public JsonConfigDialog getConfigDialog() {
39      return this.configDialogBuilder.getConfigDialog();
40    }
41
42    public String getPluginName() {
43      return ExamplePluginConfigVariables.NAME;
44    }
45
46    public String getVersion() {
47      return ExamplePluginConfigVariables.VERSION;
48    }
49
50    public JsonMenuEntry getMenuEntry() {
51      return this.pluginMenuBuilder.getMenu();
52    }
53  }
```

Listing C.3: Klassendeklaration der Konfigurationsklasse

C.2.1 Erstellung von Menüeinträgen

Die Erstellung des Menüpunktes mit allen Unterpunkten ist in einer Factory-Klasse realisiert. Diese liegt im Paket `de.hawhamburg.beli.plugin.example.factory` und ist die Klasse `ExamplePluginMenuBuilder`.

Im Folgenden werden alle möglichen Menüpunkte beispielhaft gezeigt. Jedes Objekt, das einen Menüpunkt repräsentiert, ist dabei als Singleton implementiert. So wird sichergestellt, dass sich die ID des Menüpunktes nicht ändert. Denn nur so kann der Handler die Nachrichten der verschiedenen Plug-ins unterscheiden. Es können sich trotzdem definierte variable Eigenschaften des Menüpunktes bei einer erneuten Erstellung des Menüpunktes ändern.

Der Plug-in Menüpunkt

Auf der obersten Ebene befindet sich das Hauptmenü. In diesem wird für jedes Plug-in ein Untermenü erstellt. Das Untermenü wird durch die Klasse `JsonMenuEntry` implementiert. Das folgende Beispiel zeigt, neben der Erstellung des Menüpunktes, auch die Nutzung des lokalen Ausführungszustandes des Plug-ins. Bestimmte Teile des Menüs sollen nämlich nur angezeigt werden, wenn das Plug-in ausgeführt wird.

```
/**
 * Sets up the menu message
 *
 * @return
 */
public JsonMenuEntry getMenu() {
  // Static stuff
  if (this.menu == null) {
    this.menu = new JsonMenuEntry(this.namePrefix + "Menu",
        ExamplePluginConfigVariables.VERSION);

    this.menu.Title = "Example Plugin";
  }

  // Variable stuff > The items
  this.menu.Items = new ArrayList<JsonMenuItem>();
  this.menu.Items.add(this.getInfoMenuItem());
  this.menu.Items.add(this.getStateToggleMenuItem());

  // Display the following menu items only if this plugin is in
      running
  // state
  if (Config.getLocalExecutionState() == ExecutionState.RUNNING) {
    this.menu.Items.add(this.getConfigDialogItem());

    this.menu.Items.add(this.getSeperator());

    this.menu.Items.add(this.getUriMenuItem());
    this.menu.Items.add(this.getRandomNumberMenuItem());
    this.menu.Items.add(this.getMoreMenuItem());
  }

  // Only give a copy out to the world and not the reference!
  return (JsonMenuEntry) this.menu.clone();
}
```

Listing C.4: Erstellung von Menüeinträgen - Einhängepunkt

Einfache Text Menüpunkte zur Darstellung von Informationen

Sollen lediglich einfache Informationen, wie der Name des Plug-ins, angezeigt werden, so bietet sich diese Form des Menüpunktes an. Dabei wird nur ein Text angezeigt. Der Menüpunkt hat ansonsten keine weitere Funktion.

```
1  /**
2   * Simple text item with the plugin name
3   * @return The menu item
4   */
5  public JsonMenuItem getInfoMenuItem() {
6    // Static stuff
7    if (this.infoMenuItem == null) {
8      this.infoMenuItem = new JsonMenuItem(this.namePrefix + "Info",
           ExamplePluginConfigVariables.VERSION);
9      this.infoMenuItem.Type = MenuItemType.TEXT;
10     this.infoMenuItem.Title = "Example Plugin";
11   }
12   return (JsonMenuItem) this.infoMenuItem.clone();
13 }
```

Listing C.5: Erstellung von Menüeinträgen - Text

Trennlinie zwischen zwei Menüpunkten

Zur Unterteilung der Menüstruktur in verschiedene Funktionsgruppen kann eine Trennlinie zwischen zwei Menüpunkten eingefügt werden. Trennlinien können für den Benutzer hilfreich sein, um sich schnell einen Überblick über verschiedene Funktionsgruppen zu verschaffen.

```
1  /**
2   * Just a little seperator line
3   * @return The menu item
4   */
5  public JsonMenuItem getSeperator() {
6    // Static stuff
7    if (this.seperator == null) {
8      this.seperator = JsonMenuItem.createSeperator();
9    }
10   return (JsonMenuItem) this.seperator.clone();
11 }
```

Listing C.6: Erstellung von Menüeinträgen - Trennlinie

Untermenü

Eine weitere Möglichkeit Menüpunkte zu gruppieren, ist die Erstellung eines Untermenüpunktes. Dieser ist wie der Hauptmenüpunkt des Plug-ins aufgebaut und kann auf die gleiche Weise gefüllt werden.

```
1  /**
2   * A submenu item
3   *
4   * @return The menu item
5   */
6  public JsonMenuItem getMoreMenuItem() {
7    // Static stuff
8    if (this.moreMenuItem == null) {
9      this.moreMenuItem = new JsonMenuItem(this.namePrefix + "More",
           ExamplePluginConfigVariables.VERSION);
10
11     this.moreMenuItem.Title = "Submenu";
12     this.moreMenuItem.Type = MenuItemType.SUBMENU;
13   }
14
15   this.moreMenuItem.Items = new ArrayList<JsonMenuItem>();
16   this.moreMenuItem.Items.add(this.getSubMenuWarningPopupMenuItem
       ());
17   this.moreMenuItem.Items.add(this.getSubMenuUriMenuItem());
18   this.moreMenuItem.Items.add(this.getSubMenuRandomNumberMenuItem
       ());
19
20   return (JsonMenuItem) this.moreMenuItem.clone();
21 }
```

Listing C.7: Erstellung von Menüeinträgen - Untermenü

Lokaler Funktionsaufruf

Manchmal kann es hilfreich sein, auf dem Rechner des Benutzers eine Funktion auszuführen. Diese Art der Aufrufe ist allerdings auf bestimmte, vorgegebene Funktionen beschränkt und zur Zeit limitiert auf den Aufruf einer URI.

```
/**
 * By clicking at this item a configured URI will be opened
 *
 * @return The menu item
 */
public JsonMenuItem getUriMenuItem() {
  // Static stuff
  if (this.uriMenuItem == null) {
    this.uriMenuItem = new JsonMenuItem(this.namePrefix + "Uri",
        ExamplePluginConfigVariables.VERSION);

    this.uriMenuItem.Type = MenuItemType.LOCAL_ACTION;
    this.uriMenuItem.Title = "Open an URI";
    this.uriMenuItem.ActionType = MenuItemActionType.URI;
  }

  this.uriMenuItem.Action = ExamplePluginConfigVariables.MENU_URI;

  return (JsonMenuItem) this.uriMenuItem.clone();
}
```

Listing C.8: Erstellung von Menüeinträgen - Lokaler Funktionsaufruf

Remote-Funktion

Für jedes Plug-in sollte es zumindest eine Remote-Funktion geben: `remoteToggleState()` Diese schaltet zwischen dem lokalen Ausführungszustand `RUNNING` und `SLEEPING` hin und her. So können einzelne Plug-ins zur Laufzeit des Systems ausgesetzt werden.

```
  /**
   * Activate and deactivate this plugin.
   *
   * @return The menu item
   */
  private synchronized JsonMenuItem getStateToggleMenuItem() {
      // Static stuff
      if (this.stateToggleMenuItem == null) {
          this.stateToggleMenuItem = new JsonMenuItem(this.
              namePrefix + "State",
                  ExamplePluginConfigVariables.VERSION);

```

```
12              this.stateToggleMenuItem.Type = MenuItemType.
                    REMOTE_ACTION;
13              this.stateToggleMenuItem.Action = "remoteToggleState";
14          }
15
16          if (Config.getLocalExecutionState() == ExecutionState.
                RUNNING) {
17              this.stateToggleMenuItem.Title = "Disable";
18          } else {
19              this.stateToggleMenuItem.Title = "Enable";
20          }
21
22          return (JsonMenuItem) this.stateToggleMenuItem.clone();
23      }
```

Listing C.9: Erstellung von Menüeinträgen - Remote-Funktion

Jede Remote-Funktion muss von dem jeweiligem Plug-in verarbeitet werden. Hierzu bietet es sich an, eine extra Klasse zu implementieren. Diese Klasse soll die Funktionen bereitstellen, die Remote aufgerufen werden können. Die Klasse, die die Verarbeitung der Remote-Funktionen durchführt, muss das Interface `RemoteActions` aus dem Paket `de.hawhamburg.beli.handler` implementieren.

Im Beispiel Plug-in ist die `ExamplePluginRemoteActions`-Klasse für die Verarbeitung der Remote-Funktionen zuständig. Damit diese Klasse automatisch aktiviert wird, sobald eine Remote-Funktion aufgerufen wird, kann die Hilfsklasse `RemoteActionHandler` verwendet werden. Sie bietet hierfür die statische Funktion `listenToRemoteActions` an:

```
1   /**
2    * Create a remote action listener on the given topic.
3    *
4    * @param pluginConfig
5    *            The plugin config to check the remote actions
6    * @param remoteActions
7    *            The offered remote actions of the plugin
8    */
9   public static void listenToRemoteActions(
10                      PluginConfigWithMenu pluginConfig,
11                      RemoteActions remoteActions)
```

Listing C.10: Die Funktion des *Remote Action Handlers*

Die Klasse zur Verarbeitung der Funktionsaufrufe muss nun nur noch die eigentlichen Funktionen implementieren. Im obigen Beispiel (Listing C.9) wurde die Remotefunktion für den Menueintrag auf

```
this.stateToggleMenuItem.Action = "remoteToggleState";
```

gesetzt. Die Klasse `ExamplePluginRemoteActions` implementiert die entsprechende Funktion:

```
public class ExamplePluginRemoteActions implements RemoteActions {
  private static final Debugger DEBUGGER =
    new Debugger(ExamplePluginRemoteActions.class.getSimpleName());

    /**
     * Changes the local exection state between running and sleeping
        and
     * republishes the menu entry.
     */
    public final void remoteToggleState() {

        JsonConfigDialog configDialog = this.pluginConfig.
            getConfigDialog();

        if (Config.getLocalExecutionState() == ExecutionState.
            RUNNING) {
            ExamplePluginRemoteActions.DEBUGGER.info("Getting to
                sleep...");
            Config.setLocalExecutionState(ExecutionState.SLEEPING);
            configDialog.RemoveFromConfigDialog = true;
        } else {
            ExamplePluginRemoteActions.DEBUGGER.info("Awake!");
            Config.setLocalExecutionState(ExecutionState.RUNNING);
        }
        ExamplePluginConfig.getInstance().publishMenu();

        ConnectionUtil.setMessageOnTopic(ConfigVariables.
            SYSTEM_TRAY_CONFIG_DIALIG_TOPIC, configDialog);
    }
}
```

Listing C.11: Erstellung von Menüeinträgen - Implementierung einer Remote-Funktion

Konfigurationsdialog anzeigen

Der Konfigurationsdialog des Frameworks zeigt je aktivem Plug-in einen Tab an. Um den Tab für ein Plug-in zu aktivieren muss der komplette Konfigurationsdialog des Plug-ins an den zuständigen Handler übermittelt werden. Hierbei gibt es zwei Varianten:

1. Der Tab wird nur registriert, der Konfigurationsdialog aber nicht angezeigt.
2. Der Tab wird registriert und der Konfigurationsdialog wird direkt angezeigt.

Um einen Konfigurationdialog anzuzeigen, muss ein Menüeintrag mit einer speziellen Remote-Funktion verknüpft werden. Die Remote-Funktion fordert beim Plug-in die aktuelle Konfiguration zur Anzeige des Konfigurationsdialogs an.

```
1   /**
2    * Request the config dialog window.
3    *
4    * @return The menu item
5    */
6   private synchronized JsonMenuItem getConfigDialogItem() {
7       // Static stuff
8       if (this.configDialogItem == null) {
9           this.configDialogItem = new JsonMenuItem(
10                  this.namePrefix + "ConfigDialog",
11                  ExamplePluginConfigVariables.VERSION);
12
13          this.configDialogItem.Type = MenuItemType.OPTIONS;
14          this.configDialogItem.Title = "Options";
15          this.configDialogItem.Action = "remoteGetConfigDialog";
16      }
17
18      return (JsonMenuItem) this.configDialogItem.clone();
19  }
```

Listing C.12: Erstellung von Menüeinträgen - Konfigurationsdialog

Wie bei den Remote-Funktionen wird auch der Aufruf des Konfigurationsdialogs von einer `RemoteActions`-Implementation verarbeitet. An dieser Stelle muss nun allerdings die Funktion `remoteGetConfigDialog()` implementiert werden:

```
1     /**
2      * Sets up the config dialog message and publishes it to the
3      * message broker.
4      */
5     public final void remoteGetConfigDialog() {
6         JsonConfigDialog configDialog = this.pluginConfig.
              getConfigDialog();
7         configDialog.OnlyAppend = false;
8         ConnectionUtil.setMessageOnTopic(ConfigVariables.
              SYSTEM_TRAY_CONFIG_DIALIG_TOPIC, configDialog);
9     }
```

Listing C.13: Erstellung von Menüeinträgen - Remote-Funktion für die Anzeige des Konfigurationsdialogs

C.2.2 Anzeige und Verarbeitung eines Konfigurationsdialogs

Um die Anzeige des Konfigurationsdialogs beim Benutzer hervorzurufen, wird vom Plug-in die Nachricht `JsonConfigDialog` gesendet. Hierbei ist zu beachten, dass die Flags

`JsonConfigDialog.OnlyAppend`

und

`JsonConfigDialog.RemoveFromConfigDialog`

beide auf **`false`** gesetzt sind.

Im Beispiel-Plug-in ist die Factory-Klasse `ExamplePluginConfigDialogFactory` für die Erstellung der JSON-Nachricht verantwortlich. Zu beachten ist, dass sich die ID der Nachricht nicht verändern darf. Nur so kann der Handler die Nachrichten der verschiedenen Plug-ins unterscheiden. Die Erstellung einer Nachricht zur Anzeige des Konfigurationsdialogs sieht ähnlich wie die Erstellung einer Nachricht für die Anzeige des Menüeintrags aus. Im folgenden Beispiel gibt es genau einen Konfigurationspunkt, der angepasst werden kann: Eine URI. Diese URI ist im Beispiel-Plug-in mit einem Menüpunkt verknüpft, um so die Auswirkung der Änderung für den Benutzer direkt sichtbar zu machen. Das Listing C.13 zeigte, wie die Nachricht dann an den Handler übermittelt wird.

```
public class ExamplePluginConfigDialogFactory {

    /**
     * The config dialog message (Singleton).
     */
    private JsonConfigDialog configDialog;

    /**
     * The URI which will be opened by the corresponding menu item (
        Singleton).
     */
    private JsonConfigDialogItem menuUriConfigItem;

    /**
     * Sets up the configuration dialog message.
     *
     * @return The configuration dialog message to request the
        configuration
     *         dialog window
     */
    public final JsonConfigDialog getConfigDialog() {
        // Static stuff
        if (this.configDialog == null) {
            this.configDialog = new JsonConfigDialog("
                ExamplePluginConfigDialog",
                ExamplePluginConfigVariables.VERSION);

            this.configDialog.Title = "Example Plugin Configuration
                Dialog";
            this.configDialog.DescriptionText = "In this dialog you
                can change configuration settings for the plugin.";
        }

        // Variable stuff > The items
        this.configDialog.Items = new ArrayList<JsonConfigDialogItem
            >();
        this.configDialog.Items.add(this.getMenuUriConfigItem());

        return (JsonConfigDialog) this.configDialog.clone();
    }
```

```
34
35      /**
36       * The URI which will be opened by the corresponding menu item.
37       *
38       * @return The config item
39       */
40      private JsonConfigDialogItem getMenuUriConfigItem() {
41          // Static stuff
42          if (this.menuUriConfigItem == null) {
43              this.menuUriConfigItem = new JsonConfigDialogItem("
                    ExamplePluginMenuUri",
44                          ExamplePluginConfigVariables.VERSION);
45
46              this.menuUriConfigItem.Type = ConfigDialogItemType.TEXT;
47              this.menuUriConfigItem.Title = "URI to open";
48              this.menuUriConfigItem.FieldName = "MENU_URI";
49          }
50
51          // Variable stuff > The URI
52          this.menuUriConfigItem.Value = ExamplePluginConfigVariables.
                menuUri;
53
54          // Get rid of object reference issues
55          return (JsonConfigDialogItem) this.menuUriConfigItem.clone()
                ;
56      }
57
58  }
```

Listing C.14: Anzeige von Konfigurationsdialogen - Erstellung der JSON-Nachricht

Im Konfigurationsdialog kann der Benutzer dann die Einstellungen ändern und zum Abschluss den „Speichern-Button“ klicken. Hierdurch wird die JSON-Nachricht, mit den veränderten Werten zurück gesendet. Das Plug-in muss diese Nachricht empfangen und verarbeiten, um die Einstellungen zu übernehmen. Die Utility-Klasse `ConfigDialogSaveListener` bietet zum empfangen der Nachricht die folgende Funktion an:

```
1  /**
2   * Create a config dialog save action listener on the given topic.
3   *
4   * @param pluginConfig
5   *            The plugin config to check the config dialog save
        actions
6   * @param remoteActions
7   *            The offered remote actions of the plugin
8   */
9  public static <T> void listenToConfigDialogSaveActions(Class<T>
       configClass,
10      PluginConfigWithConfigDialog pluginConfig)
```

Listing C.15: Verarbeitung von Konfigurationsdialogen - *ConfigDialogSaveListener*

Literatur

Ariga, Atsunori und Alejandro Lleras (2011). »Brief and rare mental "breaks" keep you focused: Deactivation and reactivation of task goals preempt vigilance decrements«. In: *Cognition* 118.3, S. 439–443. ISSN: 0010-0277.

Barreto, Armando B. und Jing Zhai (2003). »Physiologic Instrumentation for Real-time Monitoring of Affective State of Comuter Users«. In: *WSEAS Transactions on Circuits and Systems*. Bd. 3, S. 469–501.

Barreto, Armando, Ying Gao und Malek Adjouadi (2008). »Pupil diameter measurements: Untapped potential to enhance computer interaction for eye tracker users?« In: *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*. ACM, S. 269–270.

Beatty, Jackson (1982). »Task-evoked pupillary responses, processing load, and the structure of processing resources.« In: *Psychological bulletin* 91.2, S. 276.

Benoit, Alexandre u. a. (2009). »Multimodal focus attention and stress detection and feedback in an augmented driver simulator«. In: *Personal and Ubiquitous Computing* 13.1, S. 33–41.

Bernin, Arne (2012). »A framework concept for emotion enriched interfaces«. In: *Proceedings of the 11th international conference on Entertainment Computing*. ICEC'12. Bremen, Germany: Springer-Verlag, S. 482–485. ISBN: 978-3-642-33541-9.

Carriero, Nicholas und David Gelernter (Mai 1986). »The S/Net's Linda kernel«. In: *ACM Trans. Comput. Syst.* 4.2, S. 110–129. ISSN: 0734-2071.

– (Apr. 1989). »Linda in context«. In: *Commun. ACM* 32.4, S. 444–458. ISSN: 0001-0782.

Czerwinski, Mary, Edward Cutrell und Eric Horvitz (2000a). »Instant Messaging and Interruption: Influence of Task Type on Performance«. In: *OZCHI 2000 Conference Proceedings*. Association for Computing Machinery, Inc., S. 356–361.

– (2000b). »Instant Messaging: Effects of Relevance and Timing«. In: *HUMAN-COMPUTER INTERACTION*, S. 71–76.

Davenport, Thomas H, Sirkka L Jarvenpaa und Michael C Beers (1996). »Improving Knowledge Work Processes«. In: *Sloan management review* 37.4, S. 53–65.

Dey, Anind K (1998). »Context-aware computing: The CyberDesk project«. In: *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*, S. 51–54.

Dey, Anind K. und Gregory D. Abowd (1999). »Towards a better understanding of context and context-awareness«. In: *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. Springer-Verlag, S. 304–307.

Dey, Anind K., Gregory D. Abowd und Andrew Wood (1998). »CyberDesk: a framework for providing self-integrating context-aware services«. In: *Proceedings of the 3rd international conference on Intelligent user interfaces*. IUI '98. San Francisco, California, USA: ACM, S. 47–54. ISBN: 0-89791-955-6.

Dragunov, Anton N. u. a. (2005). »TaskTracer: a desktop environment to support multi-tasking knowledge workers«. In: *Proceedings of the 10th international conference on Intelligent user interfaces*. IUI '05. San Diego, California, USA: ACM, S. 75–82. ISBN: 1-58113-894-6.

Eckert, Michael und François Bry (2009). »Complex Event Processing (CEP)«. German. In: *Informatik-Spektrum* 32.2, S. 163–167. ISSN: 0170-6012.

Ellenberg, Jens (2011). »Ontologiebasierte Aktivitätserkennung im Smart Home Kontext«. Masterarbeit. Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.

Ellenberg, Jens u. a. (2011). »An environment for context-aware applications in smart homes«. In: *International Conference on Indoor Positioning and Indoor Navigation (IPIN), Guimaraes, Portugal*.

Fernandez, R. und R.W. Picard (1998). »Signal processing for recognition of human frustration«. In: *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*. Bd. 6, 3773–3776 vol.6.

Ferreira, Pedro (2008). »Dealing with Stress: Studying experiences of a real-time biofeedback system«. Magisterarb. Departement of Computer und Systems Sciences, Stockholm University/KTH.

Ferreira, Pedro u. a. (2008). »License to chill!: how to empower users to cope with stress«. In: *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*. NordiCHI '08. Lund, Sweden: ACM, S. 123–132. ISBN: 978-1-59593-704-9.

Gelernter, David (Jan. 1985). »Generative communication in Linda«. In: *ACM Trans. Program. Lang. Syst.* 7.1, S. 80–112. ISSN: 0164-0925.

Göddel, Jens (2008). »Herausforderung der Wissensarbeit: Bewältigen von Unterbrechungen bei nebenläufigen Arbeiten«. In: *Readings on Knowledge Management* 1. Hrsg. von Marcus Liwicki Andreas Dengel und Thomas Roth-Berghofer, S. 73.

Gulhane, YH, SV Rode und SA Ladhake (2011). »Application of fuzzy logic in stress analysis«. In: *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*. ACM, S. 679–685.

Haapalainen, Eija u. a. (2010). »Psycho-physiological measures for assessing cognitive load«. In: *Proceedings of the 12th ACM international conference on Ubiquitous computing*. ACM, S. 301–310.

Healey, Jennifer, Justin Seger und Rosalind Picard (1999). *Quantifying driver stress: Developing a system for collecting and processing bio-metric signals in natural situations*. Techn. Ber. M.I.T Media Laboratory Perceptual Computing Section.

Heidenreich, Martin (2003). »Die Debatte um die Wissensgesellschaft«. German. In: *Wissenschaft in der Wissensgesellschaft*. Hrsg. von Stefan Böschen und Ingo Schulz-Schaeffer. VS Verlag für Sozialwissenschaften, S. 25–51. ISBN: 978-3-531-13996-8.

Hoeks, Bert und Willem JM Levelt (1993). »Pupillary dilation as a measure of attention: A quantitative system analysis«. In: *Behavior Research Methods, Instruments, & Computers* 25.1, S. 16–26.

Hollatz, Dennis (2010). »Entwicklung einer nachrichtenbasierten Architektur für Smart Homes«. Masterarbeit. Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.

Holmquist, Lars Erik u. a. (2007). *Mobile Life: A Research Program for Mobile Services*. Techn. Ber. DSV – Department of Computer und Systems Sciences, Stockholm University.

Holsten, Matthias u. a. (2010). *Projektbericht Home-Office 2.0*. Projekt 1. Hochschule für angewandte Wissenschaften Hamburg.

Hook, Kristina (2008). »Knowing, communication and experiencing through body and emotion«. In: *Learning Technologies, IEEE Transactions on* 1.4, S. 248–259.

Hunt, Andrew (2009). *Pragmatisches Denken und Lernen - Refactor Your Wetware!* München: Hanser Verlag. ISBN: 3446416439.

Iqbal, Shamsi T. und Brian P. Bailey (2007). »Understanding and developing models for detecting and differentiating breakpoints during interactive tasks«. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '07. San Jose, California, USA: ACM, S. 697–706. ISBN: 978-1-59593-593-9.

– (Dez. 2010). »Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks«. In: *ACM Trans. Comput.-Hum. Interact.* 17.4, 15:1–15:28. ISSN: 1073-0516.

Iqbal, Shamsi T. und Eric Horvitz (2010). »Notifications and awareness: a field study of alert usage and preferences«. In: *Proceedings of the 2010 ACM conference on Computer supported*

cooperative work. CSCW '10. Savannah, Georgia, USA: ACM, S. 27–30. ISBN: 978-1-60558-795-0.

Iqbal, Shamsi T, Xianjun Sam Zheng und Brian P Bailey (2004). »Task-evoked pupillary response to mental workload in human-computer interaction«. In: *CHI'04 extended abstracts on Human factors in computing systems*. ACM, S. 1477–1480.

Kapoor, Ashish, Winslow Burleson und Rosalind W Picard (2007). »Automatic prediction of frustration«. In: *International Journal of Human-Computer Studies* 65.8, S. 724–736.

Kersten, Mik (2007). »Focusing knowledge work with task context«. AAINR26735. Diss. Vancouver, BC, Canada, Canada: University of British Columbia. ISBN: 978-0-494-26735-6.

Kirsh, David (2000). »A Few Thoughts on Cognitive Overload«. In: *Intellectica* 30, S. 19–51.

Kobsa, Alfred und Jörg Schreck (Mai 2003). »Privacy through pseudonymity in user-adaptive systems«. In: *ACM Trans. Internet Technol.* 3.2, S. 149–183. ISSN: 1533-5399.

Kreifeldt, J.G. und M.E. McCarthy (1981). »Interruption as a test of the user-computer interface«. In: *Proceedings of the 17th Annual Conference on Manual Control*. Jet Propulsion Laboratory. California Institute of Technology: JPL Publication, 81–95, 655–667.

Kutila, M. u. a. (2007). »Driver Distraction Detection with a Camera Vision System«. In: *Image Processing, 2007. ICIP 2007. IEEE International Conference on*. Bd. 6, pages.

Latorella, Kara A. (1998). »Effects of modality on interrupted flight deck performance: implications for data link«. In: *Proceedings of the 42nd Annual Meeting of the Human Factors and Ergonomics Society*, S. 87–91.

– (1999). »Investigating Interruptions: Implications for Flightdeck Performance«. In: *National Aviation and Space Administration*.

Lohmann-Haislah, Andrea (2012). *Stressreport Deutschland 2012. Psychische Anforderungen, Ressourcen und Befinden*. Bundesanstalt für Arbeitsschutz und Arbeitsmedizin.

Lotz, Felix (2013). »System Architectures for Automated Vehicle Guidance Concepts«. English. In: *Automotive Systems Engineering*. Hrsg. von Markus Maurer und Hermann Winner. Springer Berlin Heidelberg, S. 39–61. ISBN: 978-3-642-36454-9.

Luck, Prof. Dr. Kai von u. a. (2010). *A place for concepts of IT based modern living*. Techn. Ber. Hamburg University of Applied Sciences. URL: http://livingplace.informatik.haw-hamburg.de/content/LivingPlaceHamburg_en.pdf (besucht am 24. 06. 2013).

Luckham, D.C. (2002). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. ADDISON WESLEY Publishing Company Incorporated. ISBN: 9780201727890.

Ludewig, J. und H. Lichter (2010). *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt-Verlag. ISBN: 9783898646628.

Meißner, Tina (2013). »Untersuchung physiologischer Signale kognitiv-psychologischer Belastungen hinsichtlich berechenbarer Merkmale«. Magisterarb. Fachhochschule Brandenburg.

Müller, Larissa (2010). *Ausarbeitung AW 1 - Context Awareness - Affective Computing*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg.

Newtson, Darren und Gretchen Engquist (1976). »The perceptual organization of ongoing behavior«. In: *Journal of Experimental Social Psychology* 12.5, S. 436–450. ISSN: 0022-1031.

Oki, Brian u. a. (1993). »The Information Bus: an architecture for extensible distributed systems«. In: *Proceedings of the fourteenth ACM symposium on Operating systems principles*. SOSP '93. Asheville, North Carolina, USA: ACM, S. 58–68. ISBN: 0-89791-632-8.

Oran, Daniel P u. a. (Apr. 1997). *Operating system provided notification area for displaying visual notifications from application programs*. US Patent 5,617,526.

Otto, Kjell (2013). »Aktuelle Entwicklungskonzepte zur Projektintegration in einem Smart Home anhand von Maven, OSGi und Drools Fusion«. Masterarbeit. Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.

Otto, Kjell und Sören Voskuhl (2010). *Entwicklung einer Architektur für den Living Place Hamburg. Projektbericht Sommersemester 2010*. Projektbericht. Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.

– (2011). *Weiterentwicklung der Architektur des Living Place Hamburg. Projektbericht Wintersemester 10/11*. Projektbericht. Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.

Panier, Karsten (2009). *Home Office 2.0*. Anwendungen 1. Hochschule für angewandte Wissenschaften Hamburg.

– (2010). *Home Office 2.0 Related Work*. Anwendungen 2. Hochschule für angewandte Wissenschaften Hamburg.

– (2011). *Home Office 2.0 - Virtual Project Office*. Seminar. Hochschule für angewandte Wissenschaften Hamburg.

Pascoe, Jason (1998). »Adding Generic Contextual Capabilities to Wearable Computers«. In: *Proceedings of the 2nd IEEE International Symposium on Wearable Computers*. ISWC '98. Washington, DC, USA: IEEE Computer Society, S. 92–. ISBN: 0-8186-9074-7.

Rensing, Ludger u. a. (2005). *Mensch im Stress - Psyche, Körper, Moleküle*. 1. Aufl. München: Spektrum-Akademischer Vlg. ISBN: 382741556X.

Rivest, Ronald L, Adi Shamir und Len Adleman (1978). »A method for obtaining digital signatures and public-key cryptosystems«. In: *Communications of the ACM* 21.2, S. 120–126.

Sanches, P. (2008). »Supporting Self-Reflection in Everyday Life: An exploratory review of physiological input methods for the Affective Health system«. Magisterarb. Departement of Computer und Systems Sciences, KTH.

Sanches, Pedro, Kristina Höök u. a. (2010). »Mind the body!: designing a mobile stress management application encouraging personal reflection«. In: *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. DIS '10. Aarhus, Denmark: ACM, S. 47–56. ISBN: 978-1-4503-0103-9.

Sanches, Pedro, Elsa Kosmack Vaara u. a. (2010). »Affective Health - designing for empowerment rather than stress diagnosis«. In: *At the workshop, Know thyself: monitoring and reflecting on facets of one's life at CHI 2010*. Atlanta, GA, USA.

Schilit, B., N. Adams und R. Want (Dez. 1994). »Context-aware computing applications«. In: *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on*, S. 85–90.

Serva, Mark A u. a. (2011). »In times of stress, be bold and valiant: a preliminary exploration of the psychosocial and physiological measures of stress and suggestions for future MIS research«. In: *Proceedings of the 49th SIGMIS annual conference on Computer personnel research*. ACM, S. 14–19.

Snyder, Bruce, Dejan Bosnanac und Rob Davies (2011). *ActiveMQ in action*. Manning.

Spira, Jonathan B und Joshua B Feintuch (2005). »The Cost of Not Paying Attention: How Interruptions Impact Knowledge Worker Productivity«. In: *Analyst*.

Stroop, J. Ridley (1935). »Studies of interference in serial verbal reactions«. In: *Journal of Experimental Psychology*. Bd. 18. George Peabody College, S. 643–662.

Tanenbaum, Andrew S. (2009). *Moderne Betriebssysteme*. 3., aktualisierte Auflage. Pearson Studium. ISBN: 3827373425.

Tanenbaum, A.S. und M. van Steen (2003). *Verteilte Systeme: Grundlagen und Paradigmen*. I : Informatik. Pearson Education Deutschland GmbH. ISBN: 9783827370570.

Vaara, Elsa Kosmack, Kristina Höök und Jakob Tholander (2009). »Mirroring bodily experiences over time«. In: *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*. CHI EA '09. Boston, MA, USA: ACM, S. 4471–4476. ISBN: 978-1-60558-247-4.

Vaara, Elsa Kosmack, Iuliana Silvşan u. a. (Okt. 2010). »Temporal Relations In Affective Health«. In: *Proceedings of the 6th Nordic Conference on Human-Computer Interaction 2010*. Reykjavik, Iceland: ACM Press, S. 833–838.

Valenti, R., N. Sebe und T. Gevers (2007). »Facial Expression Recognition: A Fully Integrated Approach«. In: *Workshop on Visual and Multimedia Digital Libraries*.

Voida, Stephen und Elizabeth D. Mynatt (2009). »It feels better than filing: everyday work experiences in an activity-based computing system«. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, S. 259–268. ISBN: 978-1-60558-246-7.

Voskuhl, Sören (2012). »Modellunabhängige Kontextinterpretation in einer Smart Home Umgebung«. Masterarbeit. Hochschule für angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik, Department Informatik.

Wu, Hao-Yu u. a. (Juli 2012). »Eulerian video magnification for revealing subtle changes in the world«. In: *ACM Trans. Graph.* 31.4, 65:1–65:8. ISSN: 0730-0301.

Zhai, Jing und Armando B. Barreto (2006a). »Stress Detection in Computer Users Based on Digital Signal Processing of Noninvasive Physiological Variables«. In: *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, S. 1355–1358.

– (2006b). »Stress detection in computer users through non-invasive monitoring of physiological signals«. In: *Biomedical sciences instrumentation*. Bd. 42, S. 495–500.

Zhai, Jing, Armando B. Barreto u. a. (Apr. 2005). »Realization of stress detection using psychophysiological signals for improvement of human-computer interactions«. In: *SoutheastCon, 2005. Proceedings. IEEE*, S. 415–420.

Glossar

ActiveMQ Ein freier Message Broker, der von Apache entwickelt wird und unter der Apache Lizenz 2.0 veröffentlicht wird (siehe http://activemq.apache.org/, zuletzt besucht am 13.06.2013). vi, 44, 48, 51, 54–57, 65, 66, 68, 73, 75, 76, 79, 81, 85

Middleware Die Middleware (Dienstschicht) ist eine Zwischenschicht in Programmen, die zwischen der darüber und darunter liegenden Schicht vermittelt und die Komplexität und die Infrastruktur der Applikationsschicht verborgen bleiben (siehe auch Kapitel 3.2). vi, 38–44, 51

MongoDB Dokumentenorientierte Datenbank, die Daten im JSON-Format speichert (siehe auch http://www.mongodb.org/, zuletzt besucht 22.06.2013). 51, 54, 57, 76, 77

RSA-Kryptoverfahren Ein asymmetrisches Verschlüsselungsverfahren von Rivest, Shamir und Adleman, bei dem die Verschlüsselung und Entschlüsselung mit einem Schlüsselpaar, bestehend aus privatem und öffentlichem Schlüssel, durchgeführt wird. (siehe auch (Rivest, Shamir und Adleman 1978)). 63, 77

Tab „(Bürowesen) der Kenntlichmachung bestimmter Merkmale dienender, vorspringender Teil am oberen Rand einer Karteikarte“ (aus dem Duden http://www.duden.de/node/695839/revisions/1074920/view, zuletzt besucht am 26.06.2013). vi, 63, 64, 82, 88, 101

Tag aus dem englischen: Auszeichner; Beschreibt zusätzliche Informationen an Daten, die das Datum näher beschreiben. 22, 72

Wissensarbeiter Wissensarbeiter werden nicht für ihre körperliche Arbeit entlohnt. Sie wenden stattdessen ihr erworbenes Wissen an. 1, 2, 4, 6, 10, 11, 16–18, 32, 75

Wrapper Auch: Adapter. Umhüllt die Funktionalität einer anderen Klasse oder eines ganzen Programms und bietet meist vereinfachte Schnittstellen zur Nutzung des darunter liegenden Systems an.. 49, 50, 54

Akronyme

Application Programming Interface Eine Programmierschnittstelle eine Programms. 21, 33, 56

ID Identifikationsnummer. 61, 64, 71, 77, 85, 94, 102

IDE Integrated Development Environment. 24, 25, 69, 70, 72

JNI Java Native Interface. 56

JavaScript Object Notation Ein Datenformat zur Repräsentation von Informationen, die für den Computer lesbar sind (siehe auch http://json.org/json-de.html, zuletzt besucht am 15.06.2013). vii, 47, 48, 85, 102, 104, 113

LAN Local Area Network. 21

REST Representational State Transfer. 56

SVM Support Vector Machine. 29, 30, 57

Taskbar Notification Area Ein Benachrichtigungsfeld von grafischen Benutzeroberflächen. Die TNA befindet sich meistens in der Taskleiste. vi, 59–62, 76, 80, 81, 86, 88

URI Universal Resource Identifier. 61, 97, 102

WWW World Wide Web. 41

eXtensible Markup Language Ein Datenformat zur Repräsentation von Informationen, die für den Computer lesbar sind (siehe auch http://www.w3.org/XML/, zuletzt besucht am 07.07.2013). 48

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 07. August 2013 Benjamin Lindemann