Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterprojekt 2

Benjamin Lindemann

Stress am IT-Arbeitsplatz - *Projektbericht 2*

4. Februar 2013

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

Kurzzusammenfassung

Das Ziel meiner Masterarbeit ist die Entwicklung eines Software-Frameworks, das bei der Messung von Stresszuständen am Menschen unterstützt. Hierzu bietet das Framework Schnittstellen an, um Sensoren mit der zentralen Steuerungseinheit zu verbinden. Die zentrale Steuerungseinheit empfängt dabei die ermittelten Stresslevel pro Sensor und akkumuliert die Menge von Stressleveln pro Zeiteinheit zu einem globalen Stresslevel.

Die zentrale Steuerungseinheit des Frameworks wird in Java implementiert. Die Anbindung der Sensoren erfolgt dabei über eine ActiveMQ. Die Implementierungen einzelner Sensoren können durch diese Art der Anbindung in jeder beliebigen Programmiersprache erfolgen. Das global ermittelte Stresslevel kann als Grundlage dienen, um in späteren Arbeiten eine Antwort auf die Fragestellung „Was ist Stress am IT-Arbeitsplatz und wann tritt er auf?“ zu ermitteln.

In diesem Bericht wird der aktuelle Implementierungsfortschritt des Frameworks dargestellt. Die grundlegende Struktur der Kommunikationsschnittstelle aus *Projekt 1* wurden stark erweitert. Zusätzlich ist es nun möglich sämtliche Komponenten des Frameworks über ein Kontextmenü eines Systemtray-Icons zu konfigurieren.

1 Einleitung

Stress ist ein Zustand des Menschen, der aus verschiedenen Gründen entstehen kann. Zum Beispiel kann eine Reizüberflutung durch hohe Informationsdichte, auch *Cognitive Overload*[8] genannt, negativen Stress beim Menschen auslösen. Daher muss ein System entwickelt werden, das den Menschen beim Fokussieren auf seine Arbeit unterstützt. Das Ziel meiner Masterarbeit ist die Entwicklung eines Software-Frameworks, das bei der Messung von Stresszuständen am Menschen unterstützt.

Mein Framework soll in erster Linie keine ermittelten Daten bewerten[1], sondern eine Verbindungskomponente zwischen den Sensoren darstellen. Die von den Sensoren gemeldeten Stresslevel werden im Framework nur akkumuliert, um ein globales Stresslevel pro Zeiteinheit wiedergeben zu können. Hierauf aufbauend können Plugins entwickelt werden, die die arbeitende Person aktiv bei der Stressprävention unterstützen.

Meine Arbeit hat weder das Ziel konkrete Sensoren zur Ermittlung eines Stresszustandes beim Menschen zu implementieren noch konkrete Plugins zu entwickeln. Der Fokus der Arbeit liegt auf der Entwicklung eines Frameworks, das die einzelnen Sensoren miteinander verbindet. Hierzu wird die zentrale Steuerungseinheit in Java implementiert. Die Verbindung zu den einzelnen Plugins, die die Sensoren repräsentieren, erfolgt über den Open Source Message Broker ActiveMQ[2] von Apache.

Der folgende Bericht unterteilt sich in drei Abschnitte. Zunächst stelle ich ein paar Vorarbeiten und relevante Arbeiten, die meine Arbeit bis hierhin geprägt haben, vor. Im Hauptteil beschreibe ich den aktuellen Stand der Entwicklung des Frameworks. Im Abschluss gebe ich einen Ausblick über weitere Schritte zur Erweiterung und Verbesserung meines Frameworks.

2 Vorarbeiten und relevante Arbeiten

In den folgenden zwei Abschnitten gebe ich einen kurzen Überblick über meine bereits geleistete Arbeit und stelle kurz einige verwandte Arbeiten vor.

2.1 Vorarbeiten

Zu Beginn der Arbeit habe ich mir im Rahmen der Ringvorlesung *Anwendungen 1* Gedanken über einen Softwareagenten gemacht, mit dessen Hilfe ein am Computer Arbeitender vor Stressfaktoren geschützt werden kann. Hierbei sollte ein Softwarekonstrukt entstehen, das

[1] *Bewerten* steht hier für die Funktionalität der Software, vorhandene Daten in einem höheren Sinn zu verknüpfen und den Daten so eine abstraktere Bedeutung zu geben.

[2] `http://activemq.apache.org/` (besucht am 30.01.2013)

automatisch Stress beim Arbeitenden erkennt, in einer Stresssituation gezielt Informationen zurückhält und darauf reagiert, indem dieses erst zeitlich verzögert Informationen präsentiert. Die zeitliche Verzögerung orientiert sich an dem Stresslevel. Das heißt, dass die Informationen angezeigt werden, sobald sich das Stresslevel gesenkt hat. Hierbei beschäftigte ich mich mit dem Thema Stress sowie der Betrachtung des aktuellen Aufgabenkontextes des Arbeitenden. [11]

Hierauf aufbauend habe ich andere Forschungsarbeiten betrachtet, die sich mit einer ähnlichen Problemstellung befasst haben. Schon dort stellte ich fest, wie komplex und umfangreich das Thema Stresserkennung ist. [13]

Im *Projekt 1* konnte ich dann erste Tests mit konkreten Sensoren durchführen. Dabei sollten die Sensoren auf ihre Nutzbarkeit zur Ermittlung eines Stresszustandes beim Menschen begutachtet werden. Außerdem habe ich hier erste Teile des Frameworks spezifiziert und implementiert. Zusätzlich konnte ich mir erste Gedanken machen, wie aufgabenbasiertes Arbeiten (siehe [7]) für den Menschen einfacher zu gestalten ist. [12] Der Stand am Ende des *Projekts 1* deutete erneut auf den enormen Umfang meiner Aufgabenstellung hin.

Die Erkenntnisse aus diesen Vorarbeiten haben dazu geführt, dass ich das Ziel meiner Masterarbeit genauer spezifizieren musste. So werde ich nun keinen Softwareagenten entwickeln, der automatisch Stress beim Menschen erkennt und aktiv darauf reagiert, sondern ein Framework bereitstellen, das zur Entwicklung eines solchen Softwareagenten verwendet werden kann.

2.2 Relevante Arbeiten

Malte Kantak hat in seiner Arbeit einen Softwareagenten konzipiert, der automatisch die Erreichbarkeit einer Person in einer Smart Home Umgebung ermittelt. Hierzu wurden neben der Drools Engine[3] verschiedene Sensoren zur Bestimmung des aktuellen Zustandes der Person zu Hilfe genommen. [4, 5, 6] Der ermittelte Erreichbarkeitsstatus kann als ein Input für die Stresserkennung genutzt und als Plugin an mein Framework angebunden werden. Die Arbeit von Malte Kantak ist also als mögliche Erweiterung für mein Framework zu sehen.

Sven Klaholz hat sich in seiner Arbeit mit verschiedenen Softwarelösungen zum Betrieb einer Enterprise 2.0 Landschaft in Unternehmen beschäftigt [9]. Hierbei hat er unter anderem das Projektverfolgungstool Jira[4] betrachtet [10], welches ich als Kontext-Plugin an mein Framework angebunden habe.

Arne Bernin beschäftigt sich in seiner Doktorarbeit mit dem Aufbau eines Open Source Frameworks zur Verarbeitung von Daten zur Interpretation von Benutzeremotionen und

[3] http://www.jboss.org/drools/ (besucht am 02.02.2013)
[4] http://www.atlassian.com/software/jira/overview (besucht am 30.01.2013)

-aktivitäten. Seine Ziele basieren auf dem Finden und Entwickeln der nächsten Generation von Möglichkeiten zur Interaktion mit Computern und Ubiquitous Entertainment. [1] Die Erkenntnisse aus der Arbeit von Arne Bernin könnten später zur Evaluierung und Erweiterung meines Frameworks herangezogen werden.

Eine Arbeit aus Schweden hat mir tiefere Einblicke in die Verwendung von Biosensoren gegeben. Diese Arbeit aus der Runde einer Forschungsgruppe beschäftigt sich mit dem Aufbau eines Biofeedback Systems, welches Stress für den Anwender visualisiert. Dabei wird, wie auch in meinem Framework, keine Bewertung der Daten vorgenommen. Die gemessenen Daten werden nur für den Anwender visualisiert, damit dieser sie dann selbst bewerten kann. [2] Diese Arbeit hat mit gezeigt, dass es wichtig ist den Menschen mit dem Zustand Stress zu konfrontieren. Denn so hat der Mensch die Möglichkeit, seine Selbsteinschätzung zu verbessern. Weiterhin sollte der Mensch dabei unterstützt werden, weniger Stress ausgesetzt zu sein.

Beim Aufbau meines Frameworks habe ich auf die Ergebnisse der Forschungsarbeit von Shamsi T. Iqbal und Brian P. Bailey aufbauen können. Sie haben sich mit dem Aufbau eines Frameworks zur Steuerung der Benachrichtigungen von Programmen (Desktop Popups) beschäftigt. Dabei ist ein System entstanden, das über Plugins die Steuerung der Anzeige von Benachrichtigungen direkt in den Programmen regelt. Weitere Plugins überwachen die aktuelle Tätigkeit des Anwenders, zum Beispiel beim Erstellen eines Diagramms in Microsoft Visio[5]. Die zentrale Steuerungseinheit berechnet dann aus dem aktuellen Arbeitskontext Unterbrechungsstufen. Abhängig von der Stufe dürfen dann bestimmte Benachrichtigungen angezeigt werden. [3] Das Konzept der Arbeit hat den Entstehungsprozess der Archtitektur meines Frameworks stark beeinflusst, da das Konzept von Iqbal und Bailey für mich sehr ausgereift scheint. Die Struktur deren Frameworks habe ich in ähnlicher Form adaptiert und auf meine Bedürfnisse angepasst. Besonders die Plugins, die von außerhalb an OASIS angebunden werden, sind eine Grundlage meiner Pluginstruktur.

3 Aktueller Stand

Im folgenden Abschnitt beschreibe ich den aktuellen Stand der Implementation meines Frameworks.

3.1 Spezifikation der Schnittstellen

Um eine klare Schnittstelle für die Kommunikation zwischen der zentralen Steuerungseinheit und den Plugins aufzubauen, mussten zunächst die Topics auf der ActiveMQ und die Nach-

[5] http://office.microsoft.com/de-de/visio/ (besucht am 02.02.2013)

richtenformate formal spezifiziert werden. Ich gebe hier einen Überblick über den Aufbau der Namen der Topics und die bereits spezifizierten Topics sowie deren Bedeutung.

3.1.1 Topic-Namen

Die Topic-Namen folgen einem einheitlichen Format. Diese baut sich wie folgt auf:

Prefix Alle Topics haben ein einheitliches Prefix, damit sie auf einer ActiveMQ Umgebung, auf der weitere Anwendungen laufen, als zusammengehörig erkannt werden können.

Name Nach dem Prefix folgt direkt der Name des jeweiligen Topics.

CamelCase Die Benennung der Topics erfolgt im CamelCase Stil.

Beispiel Prefix: *StressCompanion*,
Name: *InputTopic*,
vollständiger Topic-Name: *StressCompanionInputTopic*

3.1.2 Aktuelle Topics

Die aktuell spezifizierten und implementierten Topics sind:

StressCompanionConfigTopic Sämtliche Komponenten können hierüber Änderungen an der aktuellen Konfiguration der laufenden Anwendung durchführen.

StressCompanionInputTopic Plugins senden über diese Topic ihre Stresslevel pro Zeiteinheit an die zentrale Steuerungseinheit.

StressCompanionResultTopic Die zentrale Steuerungseinheit stellt hier das aggregierte Stresslevel pro Zeiteinheit bereit.

StressCompanionSystemTrayMenuTopic Plugins können über diese Topic ein Untermenü im Kontextmenü des System Tray Icons anlegen und aktualisieren.

StressCompanionSystemTrayPopupTopic Zur Anzeige von Benachrichtigungen per System Tray Popup können Plugins eine Nachricht über diese Topic an die zentrale Steuerungseinheit senden.

StressCompanionLoginRequestTopic Sollte ein Plugin sich an einem externen System anmelden müssen und benötigt es hierzu eine Benutzerinteraktion zur Eingabe der Logindaten, so kann das Plugin diese über die *StressCompanionLoginRequestTopic* anfordern.

Sämtliche Nachrichten, die über die ActiveMQ gesendet werden, sollen im JSON-Format[6] verteilt werden. Jede JSON-Nachricht kann in Java als Klasse hinterlegt werden. Alle definierten Nachrichten sind im Maven *Util* Repository enthalten. So kann jedes Plugin auf Java-Basis diese Nachrichten nutzen.

3.1.3 Plugins in anderen Programmiersprachen

Neben der Hauptarbeit an der Java-Implementation habe ich ein kleines Beispiel-Plugin in C# entwickelt. Dieses stellt ein Beispiel der ActiveMQ-Anbindung in C# dar. Weitere Programmiersprachen, wie zum Beispiel Python[7], sind auf Grund der programmiersprachenunabhängigen Schnittstelle durch die ActiveMQ möglich.

3.2 Maven Repository

Über ein Apache Maven[8] Repository werden die einzelnen Java-Pakete für das Framework bereitgestellt. Neben der zentralen Steuerungseinheit sind aktuell ein Util-, ein Beispiel-Plugin- und ein Jira-Plugin-Paket im Maven Repository verfügbar.

Durch die Umstellung auf diese Art der Entwicklung des Frameworks können die Abhängigkeiten zwischen den einzelnen Paketen dynamisch aufgelöst werden. Hierdurch erfolgt schon bei der Implementierung eine klare Trennung der einzelnen Komponenten. Später ist es möglich, dass die zentrale Steuerungseinheit auf einer Maschine und jedes Plugin auf einer jeweils anderen Maschine läuft.

3.3 Implementationsstruktur

Der Aufbau der einzelnen Teile des Frameworks folgt einer vorgegebenen Struktur. Alle Komponenten sind unter einem einheitlichen Packagenamen strukturiert:

`de.hawhamburg.stresscompanion`

Unter diesem Package sind dann alle Komponenten aufgeteilt. Die folgende Baumstruktur soll einen Überblick über die aktuell implementierten Pakete geben. Dabei ist zu beachten, dass die Pakete in unterschiedlichen Kombinationen in verschiedenen Teilimplementationen des Projekts enthalten sein können.

Beispiel: Die Implementation des Jira-Plugins beinhaltet nur die Paketstruktur `de.hawhamburg.stresscompanion.plugin.jira`

[6] http://de.wikipedia.org/wiki/JavaScript_Object_Notation (besucht am 30.01.2013)
[7] http://www.python.org/ (besucht am 30.01.2013)
[8] http://maven.apache.org/ (besucht am 30.01.2013)

Während der Kompilierung kommen die Pakete aus der im Jira-Plugin genutzten Util-Implementation hinzu:

- `de.hawhamburg.stresscompanion.config`
- `de.hawhamburg.stresscompanion.messages`
- `de.hawhamburg.stresscompanion.utils`

Packagestruktur

```
de.hawhamburg.stresscompanion  // Klassen der zentralen Steuerungseinheit
    .config    // Konfigurationsklassen des gesamten Frameworks
    .messages  // ActiveMQ Nachrichten
    .plugin    // Plugins der Anwendung
        .jira  // Jedes konkrete Plugin befindet sich in einem Unterpaket
    .utils     // Hilfsklassen des Frameworks
    .system    // Die System spezifischen Klassen
        .tray  // Implementation des Systemtray Icons
```

Die Plugins liegen unter dem Paket `de.hawhamburg.stresscompanion.plugin`. Sie sollen der folgenden Struktur folgen:

```
de.hawhamburg.stresscompanion.plugin
    .<pluginName>                 // Beispiel: myContext
        <PluginName>Plugin.java   // Hauptdatei des Plugins,
                                  // Beispiel: MyContextPlugin.java
        <PluginName>Worker.java   // Thread, der zur Laufzeit des Plugins
                                  // ausgeführt wird.
                                  // Beispiel: MyContextWorker.java
```

Alle weiteren Dateien, die das Plugin benötigt, können beliebig unter dem Plugin-Paket abgelegt werden.

Beispiel: Ein Plugin mit dem Namen *Feedback Buzzer* soll implementiert werden. Das Plugin benötigt eine Konfigurationsdatei und eine Library Anbindung zum Auslesen der Sensordaten der Hardwareimplementation. Wie jedes Plugin startet das Plugin einen Worker als Thread. Der Aufbau des Pakets würde dann wie folgt aussehen:

```
de.hawhamburg.stresscompanion.plugin
```

```
.feebackBuzzer
    .config
        Config.java
    .lib
        BuzzerApi.java
    FeedbackBuzzerPlugin.java
    FeedbackBuzzerWorker.java
```

3.4 Zentrale Steuerungseinheit

Wie im *Projektbericht 1* beschrieben, habe ich unter anderem bereits die Basisklassen der zentralen Steuerungseinheit, eine Utility-Klasse zur Anbindung an die ActiveMQ, ein Jira-Input-Plugin und das Systemtray Icon, inklusive Kontextmenü und der Anzeige von Popup-Nachrichten, implementiert [12]. Durch die komplette Umstellung der Kommunikationsschnittstelle auf die Apache ActiveMQ mussten im *Projekt 2* jedoch einige dieser Komponenten umgeschrieben werden.

Die aktuelle Systemarchitektur ist in Abbildung 2, der Stand aus *Projekt 1* in Abbildung 1 dargestellt. Im Gegensatz zum Stand am Ende des *Projekts 1* wurde der Scheduler konzeptionell in den Bereich der Plugins verschoben. Der Scheduler implementiert ein Verhalten, das indirekt auf die Stresssituation wirkt, jedoch nichts mit der Aggregation oder der direkten Verarbeitung der Sensordaten zu tun hat. Daher gehört der Scheduler nicht in die zentrale Steuerungseinheit. Zusätzlich wurde das SystemTrayIcon hinzugefügt und die Verbindung zu den Plugins visualisiert.

Konzeptionell hinzugekommen ist die Kernkomponente für die Interaktion zwischen dem Framework und dem Anwender. Die Komponente `SystemTrayIcon` hatte ich im *Projekt 1* bereits zu programmieren begonnen. Im *Projekt 2* habe ich diese nun vollständig auf die Verwendung der ActiveMQ umgestellt. Die Eingabe von Logindaten, die Anzeige und Aktualisierung eines Kontextmenüeintrages pro Plugin und die Anzeige von Popup-Benachrichtigungen über die ActiveMQ können nun verwendet werden. Hierzu habe ich die benötigten Topics und Nachrichten definiert und die `SystemTrayIcon` Komponente entsprechend umgeschrieben.

Im *Projekt 2* wurde der Collector implementiert. Am Collector können sich interessierte Prozesse aus der zentralen Steuerungseinheit, sogenannte Listener, registrieren, um über neue Ergebnismengen per Event benachrichtigt zu werden. Der Collector ruft alle Stressleveldaten vom *StressCompanionInputTopic* ab und ordnet sie anhand des in der Nachricht enthaltenen Zeitstempels in Intervalle ein. Diese Intervalle bilden Mengen von Stressleveln, die in diesem

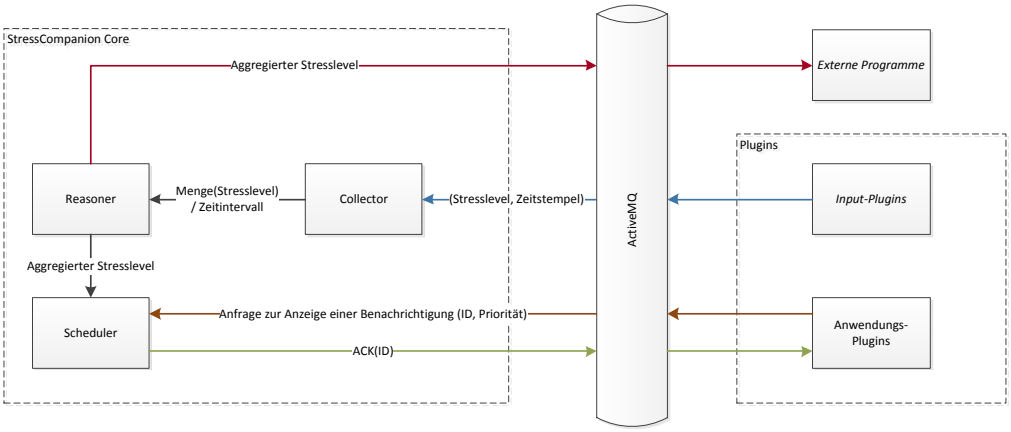

Abbildung 1: Übersicht der Systemarchitektur des Frameworks zum Stand des *Projekts 1* vom 18. August 2012 [12]

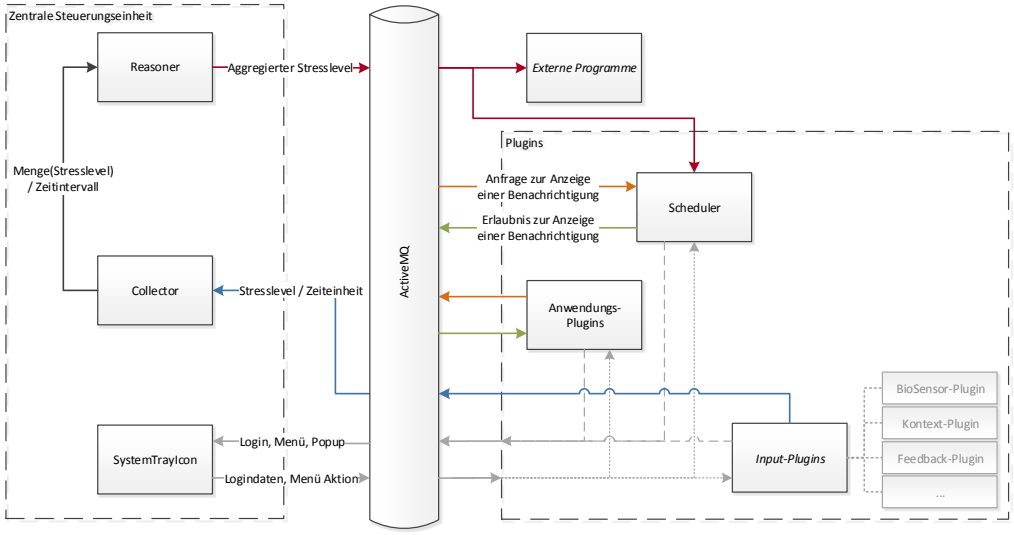

Abbildung 2: Übersicht über den aktuellen Stand der Systemarchitektur des Frameworks vom 4. Februar 2013

Intervall gemessen wurden. Sobald ein Intervall zeitlich abgelaufen ist und zu diesem Intervall keine Stressleveldaten mehr auf der ActiveMQ vorhanden sind, schließt der Collector das Intervall ab. Dann benachrichtigt der Collector alle Listener, dass eine neue Menge von Stressleveldaten bereitsteht.

Aktuell hängt an dem Event, welches der Collector an seine Listener schickt, direkt die Menge von Stressleveldaten. Zur Optimierung werde ich, bei ausreichender Zeit, hier eine zusätzliche Persistenzebene zwischenschalten, damit alle Listener immer auf die komplette Datenmenge Zugriff haben.

3.5 Konfiguration

Zur globalen Konfiguration des Frameworks habe ich eine Konfigurationsklasse entwickelt und implementiert, die die aktuelle Konfiguration des Frameworks repräsentiert. Konfigurationsänderungen erhält die Konfigurationsklasse über die ActiveMQ. Die Konfigurationsklasse stellt die aktuelle Konfiguration des Frameworks ebenfalls über die ActiveMQ allen Komponenten zur Verfügung. Da die Konfigurationsklasse im Utils-Paket enthalten ist, kann jede auf Java basierende Komponente die einheitliche Implementation nutzen.

Eine der wichtigsten Konfigurationvariablen ist der aktuelle Ausführungszustand des Frameworks. Hierüber wird gesteuert, ob das Framework gerade läuft, pausiert oder gestoppt wurde. Jede Komponente sollte diesen Zustand abfragen und sich entsprechend des aktiven Zustandes verhalten, damit nicht unnötig Ressourcen verwendet werden oder Informationen verloren gehen.

3.6 Plugins

Das Jira-Plugin musste auf Grund der neuen Anforderungen umgebaut werden. Sämtliche Kommunikation mit der zentralen Steuerungseinheit erfolgt nun über die ActiveMQ. So habe ich zum Beispiel die Abfrage der Logindaten zur Anmeldung am Jira-Repository über die ActiveMQ integriert. Ebenfalls konnte ich die Einbindung und Aktualisierung des Pluginmenüs und die Anzeige von Popup-Nachrichten über die ActiveMQ realisieren.

Neben dem Jira-Plugin habe ich ein weiteres Plugin begonnen, dass als Beispiel-Plugin mit dem Framework ausgeliefert werden soll. In diesem Beispiel-Plugin sollen sämtliche Nachrichten beispielhaft implementiert werden, so dass Wissenschaftler, die das Framework nutzen wollen, schnell und einfach Zugang zur Implementierung einzelner Plugins bekommen. Das Beispiel-Plugin wird mit einfachen Strukturen und umfangreichen Kommentaren aufgebaut.

4 Fazit

Das Ziel meiner Masterarbeit ist die Entwicklung eines Software-Frameworks, das bei der Messung von Stresszuständen am Menschen unterstützt. Das Framework soll definierte Daten über den ermittelten Stresszustand pro Sensor entgegennehmen und über die Menge an Daten pro Zeiteinheit einen globalen Stresslevel berechnen.

Im *Projekt 2* konnte ich das Ziel meiner Masterarbeit genauer ausrichten und weitere konkrete Schritte zur Erreichung dieses Ziels einleiten. Die Implementation des Frameworks habe ich weiter vorangetrieben. Es ist eine Software entstanden, die bereits die meisten grundlegenden Anforderungen bietet. So können Sensoren über eine ActiveMQ an das Framework angebunden werden und Stresszustände an die zentrale Steuerungseinheit übermitteln. Zur Interaktion mit den Komponenten und Konfiguration der Komponenten gibt es eine Schnittstelle, die es sämtlichen Komponenten erlaubt, Menüeinträge im Kontextmenü eines Systemtray Icons anzulegen und zu aktualisieren.

5 Ausblick

Wenn die vollständige Implementation meines Frameworks beendet ist und ich noch Zeit habe, möchte ich folgende Arbeitsschritte einleiten:

Der Reasoner sollte eine konkrete Implementation bekommen, so dass ein einfacher aber robuster Algorithmus die Menge an Stressleveln pro Zeiteinheit auswertet.

Der Scheduler könnte als Plugin realisiert werden, um so beispielhaft zu zeigen, wie das Framework im Echtzeitbetrieb zur Stressprävention eingesetzt werden könnte. Hierzu müssten sämtliche Popup-Nachrichten von laufenden Programmen auf dem Desktop des Anwenders abgefangen werden. Die Anzeige der Popup-Nachrichten sollte dann mit Hilfe des Schedulers gesteuert werden.

Ein weiteres konkretes Plugin könnte ein Buzzer sein. Dieser soll vom Anwender gedrückt werden, sobald er sich gestresst fühlt. Hierdurch wird dann ein Abbild der aktuellen Messdaten gespeichert, um später Rückschlüsse aus den Daten ziehen zu können. Ein erneutes Drücken des Buzzers hebt den Zustand „Ich bin gestresst“ wieder auf.

Außerdem sollte pro Pluginart zumindest ein Plugin vollständig programmiert sein, um den Zweck der verschiedenen Plugins zu verdeutlichen.

Um das Framework auf Dauer sinnvoll einsetzen zu können, muss die vom Anwender getätigte Konfiguration dauerhaft gespeichert werden. Aktuell geht die Konfiguration beim Beenden der Anwendung verloren. Außerdem sollten die verschiedenen Messdaten von den Sensoren, dem Collector und dem Reasoner in einer zusätzlichen Persistenzebene gespeichert

werden. Hierdurch kann eine nachträgliche Auswertung oder Evaluierung verschiedener Situationen durchgeführt werden.

Literatur

[1] Arne Bernin. »A framework concept for emotion enriched interfaces«. In: *Proceedings of the 11th international conference on Entertainment Computing*. ICEC'12. Bremen, Germany: Springer-Verlag, 2012, S. 482–485. ISBN: 978-3-642-33541-9. DOI: 10.1007/978-3-642-33542-6_59. URL: http://dx.doi.org/10.1007/978-3-642-33542-6_59.

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

[3] Shamsi T. Iqbal und Brian P. Bailey. »Oasis: A framework for linking notification delivery to the perceptual structure of goal-directed tasks«. In: *ACM Trans. Comput.-Hum. Interact.* 17.4 (Dez. 2010), 15:1–15:28. ISSN: 1073-0516. DOI: 10.1145/1879831.1879833.

[4] Malte Kantak. *Erreichbarkeit in Smart-Homes*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.

[5] Malte Kantak. *Erreichbarkeit in Smart-Homes - Related Work*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.

[6] Malte Kantak. *Vorarbeit für einen Erreichbarkeitsagenten*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.

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

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

[9] Sven Klaholz. *Collaboration in distributed Scrum*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.

[10] Sven Klaholz. »Enterprise 2.0 & Home Office 2.0 - Collaboration in distributed Teams«. Bericht zur Ringvorlesung Anwendung 2 an der Hochschule für angewandte Wissenschaften Hamburg. 2013.

[11] Benjamin Kuska. *CoSEC - A Stress Companion - Ein Software-Agent zur Unterstützung von hochkonzentrierter Arbeit*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.

[12] Benjamin Lindemann. *Stress am IT-Arbeitsplatz - Projektbericht*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.

[13] Benjamin Lindemann. *Stress am IT-Arbeitsplatz - Related Work*. Techn. Ber. Hochschule für angewandte Wissenschaften Hamburg, 2012.