Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Bachelorarbeit

Agentenbasierte Erfassung von Nutzungsdaten zur Evaluierung von Usability und Nutzungsverhalten

Autor:

Florian Koch

09. Oktober 2017

Betreuer:

Prof. Dr.-Ing. habil. Bernhard Preim

Institut für Simulation und Graphik

Dr.-Ing. Konrad Mühler

SelectLine Software GmbH

SelectLine®

Kaufmännische Software

Koch, Florian:
Agentenbasierte Erfassung von Nutzungsdaten zur Evaluierung von Usability und Nutzungsverhalten
Bachelorarbeit, Otto-von-Guericke-Universität Magdeburg, 2017.

Zusammenfassung

Diese Arbeit beschäftigt sich mit einem System zur Erfassung von Nutzungsdaten, mit dem Ziel des Einsatzes für die SelectLine Programmfamilie. Dabei liegt ein besonderer Fokus auf der Sicherstellung des Datenschutzes sowie der Flexibilität des Einsatzes. Die Auswertung der gesammelten Daten soll dabei helfen, Usability-Probleme zu finden und die weitere Entwicklung der Programme zu planen.

Die präsentierte Lösung setzt sich aus zwei Bestandteilen zusammen: einem Client zur Erfassung und Vorverarbeitung der Daten, sowie einem zentralen Server, der die Clients verwaltet und die Daten zur Auswertung sammelt. Um eine einfache Integration in die SelectLine-Applikationen zu gewährleisten, kommt clientseitig Aspektorientierte Programmierung zum Einsatz. Spezialisierte Agenten, die vom Server dynamisch bereitgestellt werden, sind für die Verarbeitung der so gesammelten Daten verantwortlich, und bieten die nötige Flexibilität.

Inhaltsverzeichnis

1. Einführung

Um Software gezielt und gewinnorientiert entwickeln zu können, muss unter anderem die Arbeitsweise der Nutzer berücksichtigt werden. Das sogenannte *Usability Engineering* gewinnt daher immer mehr an Bedeutung [Fer03][SK10]. Auch bei bestehenden Produkten spielt dies für die weitere Planung eine große Rolle. Durch die Evaluierung des Verhaltens einiger Testpersonen in vorbereiteten Szenarien lassen sich Probleme der Usability und mögliche Verbesserungen für ausgewählte Aspekte eines Programmes gezielt ermitteln.Um jedoch umfassende Daten zur langfristigen Planung und Priorisierung von Entwicklungszielen zu erhalten, reichen derartige Tests in kleinem Rahmen nicht aus. Hierzu bedarf es einer Lösung, die durch eine große Anzahl von Nutzern repräsentative Ergebnisse liefern kann.

Existierende Systeme dieser Art zielen im Allgemeinen auf Webseiten ab, allerdings konzentrieren sich die SelectLine-Produkte hauptsächlich auf Windows Desktop-Anwendungen. Das Ziel dieser Arbeit ist daher, eine Lösung zur Nutzungsdatenerfassung für den Einsatz in den bestehenden und künftigen SelectLine-Applikationen zu entwickeln. Ein solches System muss die Daten direkt beim Kunden sammeln sowie anonymisieren, ohne dabei dessen Arbeit mit dem Programm zu beeinträchtigen. Weiterhin ist eine zentrale Sammlung der Informationen zur späteren Auswertung erforderlich.

SelectLine entwickelt Software für den Bereich des Enterprise-Resource-Planning (ERP) und vertreibt diese über Fachhändler an kleine und mittelständische Unternehmen. Auch der Support steht im Allgemeinen nur für die Fachhändler zur Verfügung, die ihrerseits für ihre eigenen Kunden als Ansprechpartner dienen. Nur selten wird direkt mit den sogenannten *Endkunden* interagiert, wodurch der Aufwand für den Support reduziert wird. Allerdings erschwert dies auch, Erkenntnisse über häufig in der Praxis auftretende Probleme zu gewinnen.

Die Programmfamilie umfasst aktuell die Hauptprodukte *Warenwirtschaft*, *Rechnungswesen* und *Plantafel*, die jeweils mit diversen Modulen erweitert werden können, sowie kleinere Produkte, die meist mit der Warenwirtschaft zusammenarbeiten. Dazu gehört beispielsweise die Lösung für den mobilen Zugriff, *SL.mobile*. Zur Demonstration soll das in dieser Arbeit entwickelte System in dem neuesten Produkt,

SelectLine CRM, zum Einsatz kommen. Die Möglichkeit der leichten Integration in andere Applikationen stellt allerdings eine wichtige Anforderung dar.

Die Auswertung der gesammelten Daten dient der Identifizierung von Problemen der Usability sowie der Einschätzung des durchschnittlichen Nutzungsverhaltens. Die gewonnenen Informationen sollen die Grundlage für die weitere Planung der Entwicklung bilden. Da die Auslieferung neuer Programmversionen, die das System zur Nutzungsdatenerfassung verwenden, sowie die Sammlung einer ausreichenden Menge an Daten einen erheblichen Zeitfaktor darstellen, umfasst diese Arbeit lediglich einen Ausblick auf Möglichkeiten der Auswertung anhand von intern gesammelten Nutzungsdaten.

Bevor das entwickelte System vorgestellt wird, findet in Kapitel 2 eine Analyse der Ausgangslage und der resultierenden Anforderungen an die Lösung statt. Kapitel 3 beschäftigt sich mit den theoretischen Grundlagen, darauf aufbauend beschreibt Kapitel 4 das entwickelte Umsetzungskonzept. In Kapitel 5 wird auf ausgewählte Schwerpunkte der Implementierung eingegangen. Abschließend werden in Kapitel 6 Ergebnisse in Bezug auf das System selbst, seine Anwendung sowie die Auswertung der gesammelten Daten präsentiert.

2. Problemstellung

2.1 Ausgangssituation

SelectLine hat in den letzten 25 Jahren eine breite Produktpalette für den ERP-Bereich geschaffen und bedient damit eine Vielzahl von Kunden in unterschiedlichen Bereichen. Während eine Erweiterung der Produktpalette meist auf Basis der aktuellen Marktlage entschieden wird, ist das für die Weiterentwicklung bestehender Produkte nicht immer ausreichend. Die Planung der zukünftigen Entwicklungsziele besitzt viele Einflussfaktoren. In jedem Fall müssen sich die Ergebnisse jedoch an den Kundenwünschen in Bezug auf die existierende Software orientieren.

2.1.1 Kundenfeedback

Damit die Planung an den Kundenwünschen ausgerichtet werden kann, muss zuerst entsprechendes Feedback gesammelt werden. Da SelectLine seine Produkte nicht direkt sondern über Fachhändler vertreibt, konzentrieren sich die Bemühungen dabei auf deren Wünsche. Für sie wurden bereits mehrere Möglichkeiten geschaffen, SelectLine diese Wünsche mitzuteilen:

Hotline

An Wochentagen steht für Fachhändler telefonischer Support durch SelectLine-Mitarbeiter zur Verfügung. Abgesehen von der Hilfe bei spezifischen Umsetzungsproblemen, die sie dort erhalten, können auch Fehler gemeldet werden, die von den Fachhändlern oder ihren Kunden gefunden wurden. Daraufhin wird ein entsprechender Vorgang im *HelpDesk* angelegt.
Der HelpDesk ist eine eigens entwickelte Software zur zentralen Verwaltung der auswärtigen Kommunikation und aller Arbeitsaufgaben. Ursprünglich wurde der komplette Arbeitsablauf in der Software abgebildet, inzwischen wurden die Verantwortlichkeit für Planung und Entwicklung allerdings in den *Team Foundation Server*

(TFS)[1] verlagert. Für den Support ist der HelpDesk jedoch weiterhin das Hauptwerkzeug.
Fehler stellen grundsätzlich einen Sonderfall des Kundenfeedbacks dar, da die Korrektur von falschem Verhalten der Programme Priorität über etwaigen Wünschen der Kunden genießt. Kritische Fehler werden in Form eines *Hotfixes*[2] sofort behoben, andere werden direkt in die Planung für die nächste Version eingeordnet und entsprechend priorisiert.
Aus einer Support-Anfrage geht unter Umständen direkt ein Wunsch eines Fachhändlers bzw. eines seiner Kunden hervor. In diesem Fall wird ein Eintrag im UserVoice Portal angelegt und auf selbigen verwiesen (siehe nächster Abschnitt). Eine Ausnahme stellen hier jedoch individuelle Dienstleistungen dar: Wenn der Fachhändler bzw. Kunde eine Funktion zwingend benötigt, besteht die Möglichkeit einer Individualprogrammierung. In diesem Fall wird ein Auftrag für kostenpflichtige Dienstleistungen aufgesetzt, der die erstmalige Umsetzung sowie unter Umständen die Wartung umfasst. Das gewünschte Feature wird daraufhin zeitnah entsprechend der Anforderungen umgesetzt.

UserVoice Portal

Auf der Firmenwebseite besteht für Fachhändler und Kunden die Möglichkeit, Wünsche in einem UserVoice Portal[3] einzureichen. Jeder Nutzer hat 25 Stimmen zur Verfügung, von denen jeweils maximal 3 für einen speziellen Wunsch abgegeben werden können. Wenn dieser umgesetzt oder die Umsetzung explizit abgelehnt wird, werden die Stimmen ”rückerstattet”, sodass diese auf neue Wünsche verteilt werden können. Das UserVoice Portal ermöglicht SelectLine nicht nur, die Wünsche kennenzulernen, sondern erleichtert durch den Vergleich der Stimmanzahl auch, Prioritäten zuzuordnen. Weiterhin bietet es für den Kunden den Vorteil, den aktuellen Status seines Wunsches einsehen zu können. Dieser kann als ”wird überprüft”, ”geplant”, ”gestartet” (in Bearbeitung), ”fertiggestellt” oder ”abgelehnt” markiert sein.
Zusätzlich existiert eine Kommentarfunktion für jeden Wunsch, die eine Diskussion der Nutzer untereinander sowie mit Mitarbeitern ermöglicht. Dadurch können zusätzliche Vorschläge unterbreitet, kritische Aspekte hervorgehoben und auch alternative Lösungsvorschläge gemacht werden.

Umfragen & Interviews

Gelegentlich ergreift SelectLine die Initiative und geht auf die Fachhändler und Endkunden zu, anstatt auf ihr Feedback zu warten. Zu diesem Zweck werden unter anderem Umfragen durchgeführt. Jährlich findet die Partnerumfrage statt, wodurch die allgemeine Zufriedenheit der Fachhändler mit den Produkten und Dienstleistungen ermitteln werden soll. Relevanter für den Kontext dieser Arbeit sind jedoch die Umfragen zur Betaversion.

---

[1] Plattform für kollaborative Softwareentwicklung, siehe `https://www.visualstudio.com/de/tfs/`
[2] außerplanmäßiges Update
[3] Das UserVoice Portal ist eine Dienstleistung der gleichnamigen Firma. `https://www.uservoice.com/product/feedback-collection/portal/`

Seit 2014 wird den SILBER- und GOLD-Partnern, d.h. den wichtigsten Fachhändlern, die Möglichkeit geboten, eine Betaversion des nächsten Updates bereits einige Wochen vor dem eigentlichen Release zu testen. Für sie hat das den Vorteil, sich bereits vor dem Update einen Überblick über eventuell nötige Anpassungen an bestehenden Konfigurationen verschaffen zu können. In den Umfragen, die an alle Teilnehmer der aktuellen Beta geschickt werden, geht es hauptsächlich um die Bewertungen der neuen Funktionen in der jeweiligen Version. Es besteht zusätzlich die Möglichkeit, Anmerkungen zu den einzelnen Funktionalitäten zu machen. Falls Fehler auffallen können diese zudem direkt an den Support gemeldet werden, damit sie noch vor dem Release korrigiert werden. Abbildung A.5 (im Anhang) zeigt beispielhaft eine Seite der aktuellsten Beta-Umfrage, die sich auf das SelectLine CRM bezieht.
Vor der Konzeption neuer Funktionen werden außerdem, sofern möglich, Interviews mit Endkunden durchgeführt, um deren Bedürfnisse und Arbeitsweise bezogen auf einen ausgewählten Aspekt zu ermitteln. Auf diese Art und Weise werden wertvolle Einblicke in die Praxis erlangt, die großen Einfluss auf das Konzept haben können.

Hands-On Sessions & Feature-Demonstrationen für Fachhändler

Wenn neue Module oder umfangreiche Überarbeitungen geplant sind, werden ausgewählte Fachhändler zu Beginn oder während der Entwicklung eingeladen, um das Konzept oder ausgewählte Teile des bisherigen Stands, z.B. in Form eines Prototypen, zu begutachten. Das Feedback so früh zu bekommen bedeutet, dass größere Änderungen der Planung noch immer möglich sind. Dadurch lassen sich Fehler in der Einschätzung des Bedarfs und der Erwartungen der Kunden vermeiden.

2.1.2 Planung & Entwicklung

Das Produktmanagement ist für die Planung und Konzeption neuer Features verantwortlich. Die Basis für neue Funktionalitäten sind aufgenommene Tickets im HelpDesk, die Wünsche der Fachhändler im UserVoice Portal und Ideen, die direkt aus dem Produktmanagement stammen. Dabei kommt das szenariobasierte Design zum Einsatz (vgl. [BTT05, Kapitel 8.4], [RC09]).

Das Hauptmerkmal dieser Arbeitsweise besteht darin, Szenarien für die Nutzung des Programms zu erstellen. Diese stellen die Grundlage für die Planung von neuen Funktion bzw. Änderungen dar. Ein Szenario beschreibt mögliche Handlungsabfolgen und resultierende Ereignisse aus Sicht eines Nutzers. Der Detailgrad variiert dabei sehr stark, je nach Wichtigkeit der beleuchteten Aspekte und Fortschritt innerhalb des Planungsprozesses. So sind die Szenarien anfänglich noch sehr unspezifisch und umreißen lediglich grob, welche Aktionen der Nutzer durchführen kann. Spätere Ausarbeitungen betrachten nur kleine Ausschnitte und gehen dabei mehr ins Detail, z.B. wie genau der Nutzer mit dem System interagiert und in welcher Form Rückmeldungen stattfinden.

Szenarien können als arbeitsorientierte Designobjekte sogar noch effektiver werden, wenn Nutzer direkt an ihrer Erstellung beteiligt sind [RC09][4]. Die Interviews von

[4] freie Übersetzung, original: „Scenarios can be made even more effective as work-oriented design objects when users are directly involved in creating them.“ [RC09, S. 5]

Endnutzern zielen darauf ab, diesen Aspekt auszunutzen. Indem das Produktmanagement sie besucht und bei ihrer Arbeit beobachtet und befragt, können klarere Einsicht in sowie neue Perspektiven auf ihre Aufgaben gewonnen werden, welche sich wiederum in den Szenarien widerspiegeln.

Wenn ein Konzept umgesetzt werden soll, müssen konkrete Arbeitsaufgaben aus den Szenarien erstellt werden. Dies fällt in den Zuständigkeitsbereich der Entwickler, da es sich um den technischen Aspekt der Planung handelt. Sobald die Entwicklung weit genug fortgeschritten ist, besteht auch die bereits erwähnte Möglichkeit, Fachhändler einen Prototypen oder Teile des aktuellen Stands begutachten zu lassen. Diese Möglichkeit wird allerdings aufgrund des nötigen Zeitaufwandes für beide Seiten nur selten genutzt. Die Flexibilität gegenüber Änderungen ist in dieser Phase der Entwicklung bei Weitem nicht so groß wie in der Konzeption, allerdings werden einige Probleme oft erst während der Umsetzung deutlich.

Ist die Entwicklung für eine Programmversion abgeschlossen, wird die Beta-Version bereitgestellt. Dies hängt nicht notwendigerweise mit der Fertigstellung eines bestimmten Features zusammen, da die Versionen regelmäßig und mit mehreren Neuerungen geplant sind. Feedback von Nutzern der Betaversion wird hauptsächlich genutzt, um Fehler in neuen Features zu korrigieren, da die Änderungsmöglichkeiten zu diesem Zeitpunkt schon stark eingeschränkt sind. Für den Fall, dass größere Änderungen notwendig werden, wird dies in die Planung für die nächste Version einbezogen.

Abbildung 2.1 visualisiert den Einfluss des Feedbacks auf den Planungs- und Entwicklungsprozess, der sich wie folgt zusammenfassen lässt:

- Die Grundlage für die Konzeption bilden im Allgemeinen Wünsche der Fachhändler und Endkunden, meist aus dem UserVoice Portal.
- Zu Beginn der Planung größerer Projekte werden Nutzerinterviews durchgeführt, um geeignete Szenarien zu identifizieren.
- Während der Entwicklung werden Fachhändler eingeladen, um Zwischenstände bzw. Prototypen zu begutachten.
- Vor der offiziellen Auslieferung wird die Beta-Version freigegeben, um Fehler zu finden und zu korrigieren.
- Nach der Auslieferung einer Version werden etwaige Fehler in der Hotline gemeldet, Verbesserungswünsche finden ihren Weg in das UserVoice Portal.
- Kritische Fehler werden sofort behoben, die Korrektur anderer wird durch das Produktmanagement in den bestehenden Zeitplan eingeordnet.

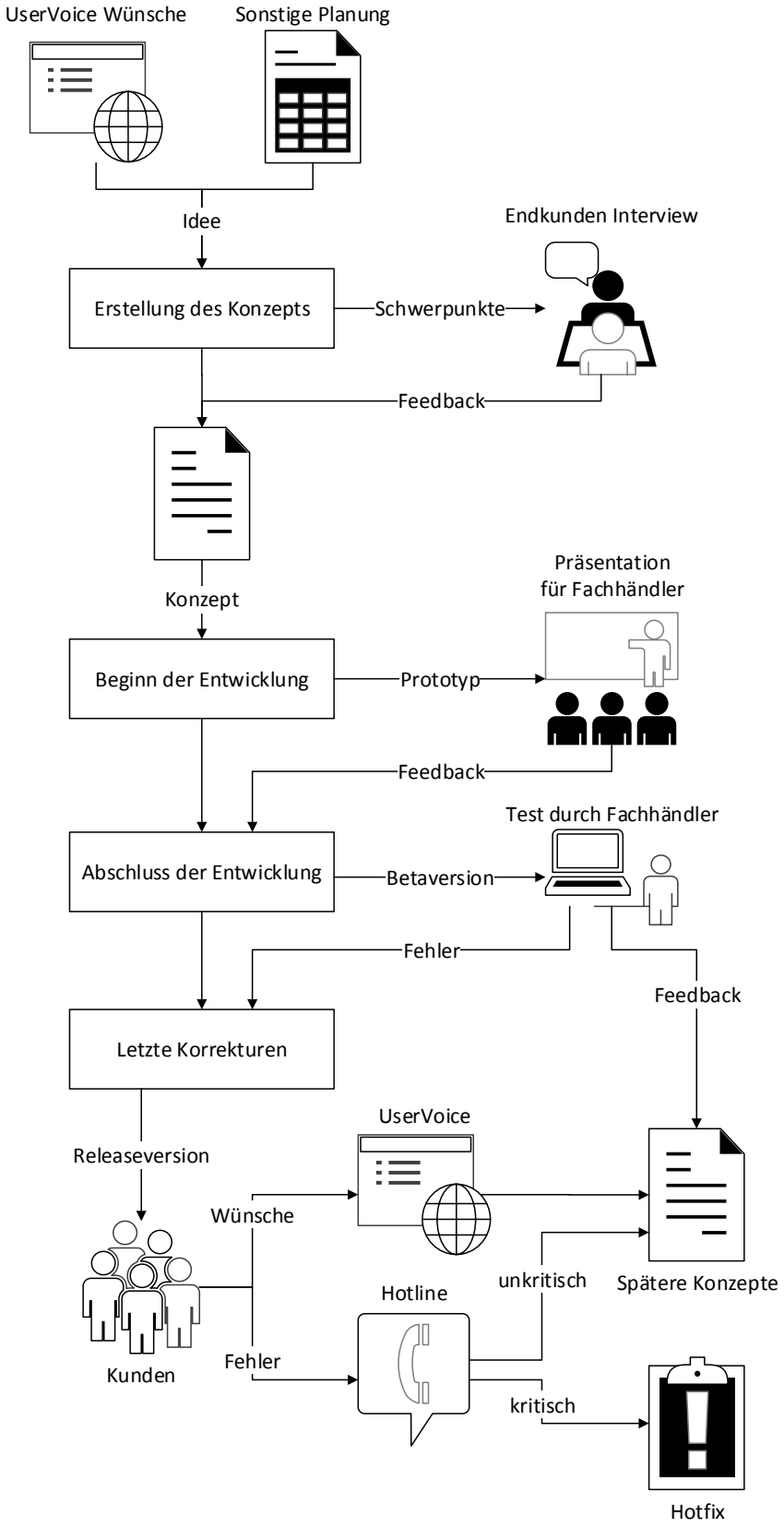

Abbildung 2.1: Einfluss von Feedback auf den Planungs- und Entwicklungsprozess

2.2 Motivation

Die bisher beschriebenen Möglichkeiten zur Sammlung von Feedback sind zwar umfangreich, allerdings handelt es sich größtenteils um *qualitatives Feedback* – detailliert, jedoch aus Sicht einzelner Personen und daher nicht repräsentativ. Im Gegensatz dazu steht *quantitatives Feedback*, d.h. repräsentative Daten bezüglich einer großen Anzahl von Nutzern, die unter Umständen jedoch weniger spezifisch sind. Entscheidungen anhand dieser Informationen bieten den Vorteil, dass eine größere Sicherheit in Bezug auf den allgemeinen Kundennutzen besteht.

Umfragen sind theoretisch in der Lage, quantitatives Feedback zu liefern, allerdings nur bei ausreichender Beteiligung. Sowohl bei der Betaversion, als auch bei den Neukunden und verlorenen Kunden, ist lediglich ein sehr kleiner Teil der Nutzer betroffen. Außerdem nehmen selten alle beteiligten Personen an der Umfrage teil, oder es werden nicht alle Fragen beantwortet. In Summe führt dies dazu, dass die Umfragen zwar Hinweise geben, allerdings nicht als Quelle für quantitatives Feedback betrachtet werden können.

Eine weitere Möglichkeit ist die Analyse der Verkaufszahlen, da anhand der verkauften Programm- und Modullizenzen die Beliebtheit bzw. der Bedarf der entsprechenden Produkte geschätzt werden kann. Durch die Lizenzierung für eine gewisse Anzahl von Arbeitsplätzen im Fall der Programme sowie der größeren Module können dadurch bereits relative genaue Erkenntnisse in Bezug auf die Nutzung gewonnen werden. Eine wichtige Information, die jedoch fehlt, ist, welche Version der Programme eingesetzt wird. Weiterhin ist nicht bekannt, welche Betriebssystem- und SQL-Server-Versionen bei den Kunden im Einsatz sind. Dadurch fällt es schwer, Änderungen, die zu Problemen mit der Kompatibilität älterer Versionen führen, einzuplanen. Es wurden bereits Bemühungen unternommen, diese Informationen durch Umfragen zu ermitteln, jedoch greifen auch hier die bereits erwähnten Probleme.

Grundsätzlich liegt das Problem im aktuellen Vorgehen somit darin, dass oft die Informationen fehlen, um begründet Entscheidungen mit Einfluss auf die komplette Kundenbasis zu treffen. Es muss daher ein Weg geschaffen werden, Daten von möglichst allen Nutzern zu erhalten, die in Folge dessen zur Identifizierung von Problemen sowie Analyse von durchschnittlichen Arbeitsabläufen genutzt werden können. Anhand dieser soll der Planungs- und Entwicklungsprozess weiter optimiert werden.

Ein weiterer Faktor, der für eine Ergänzung der bisherigen Maßnahmen spricht, wird in Abbildung 2.2 deutlich. Ferre [Fer03] beschreibt die Integration von Techniken zur Verbesserung der Usability in den Entwicklungsprozess. Zwar sind die bisherigen Maßnahmen nicht nur auf die Verbesserung der Usability, sondern auch auf die Integration neuer Funktionen ausgerichtet, allerdings stellt gute Usability einen Kernaspekt für die Zufriedenheit der Nutzer mit einer Applikation dar. Während große Teile der durch Ferre präsentierten Möglichkeiten bereits innerhalb der Planung und Umsetzung eines Projektes genutzt werden, fehlt der Aspekt der ”Usability Evaluation” jedoch fast vollständig. Bisher wird nach der Auslieferung lediglich Feedback in Form von Rückmeldungen der Kunden und Fachhändler erfasst. Das Ziel sollte daher sein, auch nach Abschluss der Entwicklung weiterhin aktiv Daten zu sammeln.

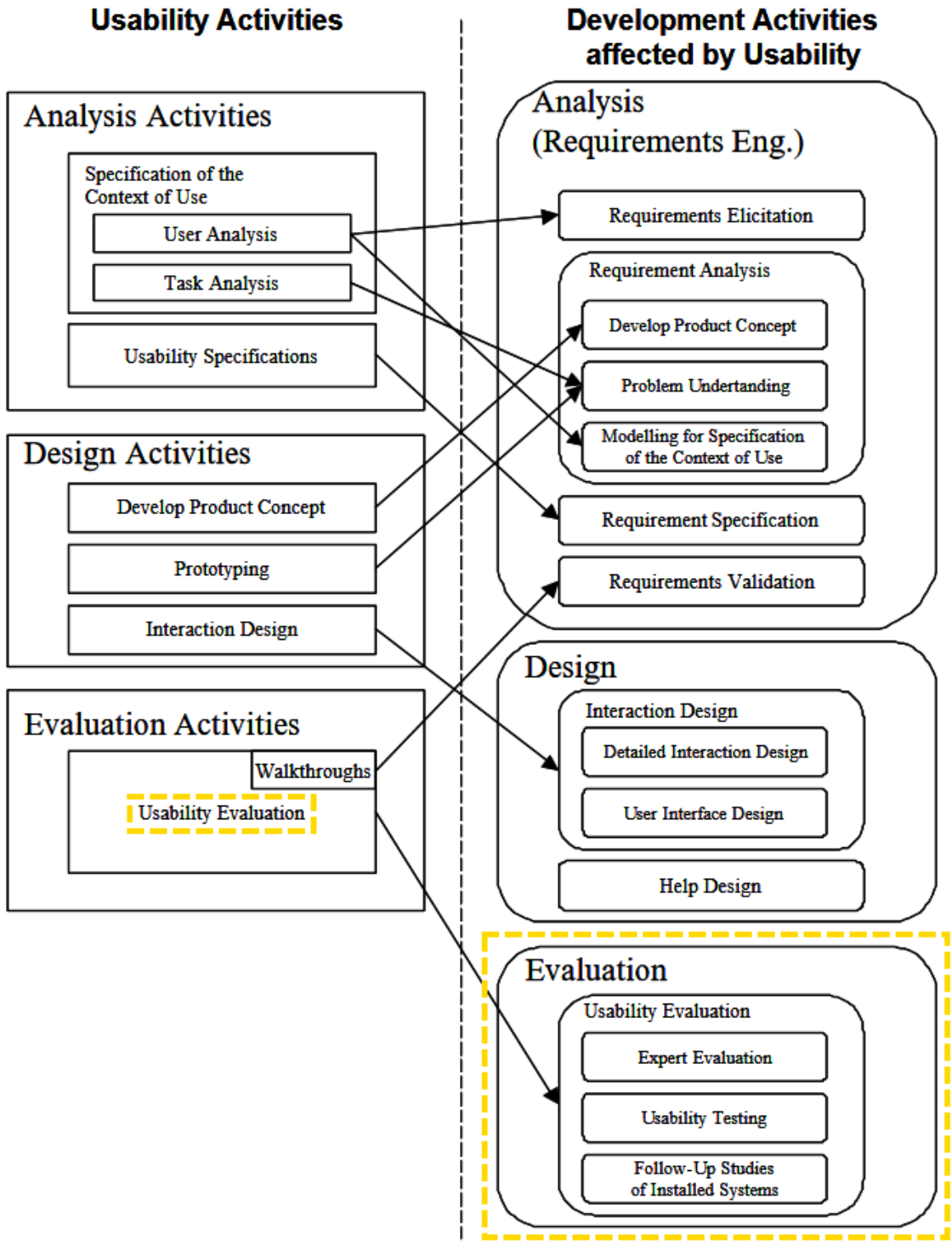

Abbildung 2.2: Zusammenhang zwischen dem Entwicklungsprozess (rechts) und Aktionen in Bezug auf Usability (links), aktuell fehlende Aspekte sind hervorgehoben. Original übernommen von Ferre [Fer03, Figure 2], Hervorhebung hinzugefügt.

2.3 Anforderungen

Wie das Beispiel der Umfragen zeigt, ist es äußerst schwierig, quantitatives Feedback zu erhalten, wenn ein Handlungszwang für den Nutzer besteht, oder nur ein Teil der Kunden erreicht werden kann. Das Ziel muss daher eine automatische Lösung sein, die bei allen Nutzern zum Einsatz kommt. Hierzu wird eine Erfassung von Nutzungsdaten während des alltäglichen Einsatzes der SelectLine Software angestrebt. Diese muss folglich auf den Systemen der Nutzer geschehen, woraufhin die Daten für die spätere Auswertung zentral gesammelt werden können. Im Gegensatz zum bisherigen Vorgehen, bei dem der Nutzer aktiv Feedback geben musste, stellt eine solche Lösung einen passiven Ansatz dar.

2.3.1 Umfang der Lösung

Die zu sammelnden Daten lassen sich grob in zwei Kategorien einteilen: allgemeine Statistiken, die ständig erfasst werden sollen, sowie situativ ermittelte Informationen, die der gezielten Analyse von spezifischen Programmelementen sowie der Überprüfung von Erwartungen dienen.

Zur ersten Kategorie gehören unter anderem folgende Daten:

- **Eingesetzte Hard- und Software**
  Eine sehr wichtige, jedoch bisher nicht verfügbare, Information ist die Version der SelectLine Software. Derzeit ist nicht abschätzbar, wie viele Kunden die aktuellen Updates einspielen und wie schnell dies vonstatten geht. Eine Statistik zu genutzten Versionen von Betriebssystem und SQL-Server kann zudem einen Eindruck vermitteln, wann eine Erhöhung der geforderten Mindestversionen tragbar ist. In Bezug auf die Hardware interessiert vor allem die Leistungsfähigkeit der PCs sowie die Bildschirmauflösung, da beides relevante Aspekte für das Programmdesign sind.

- **Größe des Datenbestandes**
  Optimierungen im Bereich der Geschwindigkeit der Programme sind meist sehr zeitaufwendig. Mit Informationen darüber, welche Datenbanktabellen wie viele Datensätze enthalten, lassen sich kritische und weniger wichtige Ansatzpunkte für solche Optimierungen identifizieren.

- **Art der Bedienung**
  Im Bereich der ERP-Software spielt die Effizienz beim Arbeiten eine große Rolle [AMAMZ03]. Daher ist die Bereitstellung von Schnellzugriffen, wie z.B. Tastenkombinationen, besonders relevant. Einen Überblick darüber, wie auf Funktionen zugegriffen wird (Kontextmenü, Toolbar, Hotkeys[5], KeyTips[6]), kann wertvolle Einsichten in die Arbeitsweise liefern und Ansätze zur Optimierung der häufigsten Prozesse aufzeigen.

[5] Hier im Sinne einer applikationsspezifischen Tastenkombination.
[6] Abfolge von Tastendrücken bei gehaltener `Alt`-Taste, dient der schnellen Navigation durch Menüs. Die entsprechenden Tasten werden meist durch unterstrichene Buchstaben indiziert.

- **Modulnutzung**
  Statistiken darüber, wie viele Nutzer welches Programm bzw. Modul verwenden, lassen sich bereits durch die Verkaufszahlen der Lizenzen abschätzen. Allerdings besteht durch die Nutzungsdatenerfassung die Möglichkeit, auch die Häufigkeit und Dauer der Nutzung zu ermitteln. Mit diesen Informationen kann der Fokus für zukünftige Entwicklungen auf die wichtigsten Elemente gelegt werden.

- **Häufigkeit der Nutzung von Funktionen**
  Eine oft gestellt Frage ist, wie häufig bestimmte Programmfunktionen genutzt werden. Für die Planung ist relevant, ob Fehler bzw. Wünsche ein viel genutztes Feature betreffen, oder nur auf eine kleine Nutzergruppe Einfluss haben. Solche Informationen müssen allerdings mit Vorsicht behandelt werden, da die Häufigkeit der Verwendung nicht zwingend ein Indikator für die Wichtigkeit einer Funktion darstellt[7].

Situativ gesammelte Daten lassen sich, der Natur ihrer Ausrichtung entsprechend, nur schwer genau beschreiben. Ein Anwendungsfall, in dem solche Informationen gesammelt werden könnten, stellt beispielsweise die Betaversion dar. Der Fokus der Datensammlung liegt hier auf der Nutzung der neuen bzw. geänderten Funktionen. Es ist daher von Interesse, möglichst genaue Informationen in dieser Hinsicht zu sammeln. Grundsätzlich geschieht jede Auswertung von Daten auf Basis einer Hypothese, in einem solchem Fall könnte diese allerdings sehr spezifisch sein (z.B. „Zuordnungen von Adressen werden erst vorgenommen, wenn die grundlegenden Daten ausgefüllt wurden“). Die Erfassung muss daher wesentlich umfangreicher und genauer geschehen, als das im normalen Gebrauch der Fall ist. Beispielsweise könnte ein Aufzeichnung der Mausbewegungen und -klicks dabei helfen, Probleme im Layout der Oberfläche zu identifizieren.

Es wird bereits deutlich, dass die Erfassung weniger eine umfassende Funktion, sondern eher eine Menge spezialisierter Vorgänge darstellt. Sie gliedert sich in funktionale Einheiten, die jeweils Informationen in Bezug auf einen ausgewählten Aspekt sammeln. Eine Herausforderung stellen Änderungen an diesen Einheiten dar. Für die allgemeinen Informationen ist dies weniger relevant, da das Sammeln der entsprechenden Daten universell umgesetzt wird und dementsprechend nur selten Anpassungen erfordern sollte. Bei der zweiten Kategorie hingegen sind häufige Änderungen zu erwarten. Die Erfassung und Auswertung so spezifischer Informationen ist ein iterativer Prozess, bei dem die Veränderung der zu sammelnden Daten einen essentiellen Bestandteil darstellt. Entscheidend ist dabei auch, dass die Iterationen in kurzen Zeitabschnitten ablaufen können. Wenn eine Änderung der Erfassung nur durch eine Programmaktualisierung vonstatten gehen kann, verzögert dies den Prozess enorm. Außerdem ist nicht gewährleistet, dass alle Anwender die Updates zeitnah einspielen. Es muss daher eine Möglichkeit geschaffen werden, die Erfassung unabhängig von den normalen Updates anzupassen.

Dieser Aspekt bringt aus technischer Sicht eine Einschränkung mit sich: es kann nur auf ein gewisses Kontingent an Daten zurückgegriffen werden, welches die Anwen-

[7] Ein gutes Beispiel ist der Jahreswechsel, der nur einmal jährlich verwendet werden muss aber eine essentielle Funktionalität darstellt. Ein weiteres Beispiel ist die Inventur.

dung zu diesem Zweck bereitstellt. Diese Daten sollten umfangreich und detailliert sein, um die Möglichkeiten der Auswertung nicht zu beeinträchtigen. Allerdings muss dafür gesorgt werden, dass das Abstraktionslevel nicht zu niedrig ist. Beispielsweise lässt sich mit einer reinen Sammlung von Events auf der Betriebssystemebene nur schwer ein Kontext ermitteln oder ein Zusammenhang herstellen. Eine genauere Betrachtung, warum der Zugriff eingeschränkt ist und welche Daten wie erfasst werden, folgt in Abschnitt 4.2.

Die so erfassten Daten sollen zudem nicht komplett gesammelt, sondern vorher verarbeitet und reduziert werden. Im Folgenden wird daher immer von *Rohdaten* gesprochen. Der wichtigste Grund für diese Entscheidung ist der Datenschutz. Eine ausführliche Auseinandersetzung mit diesem Thema folgt in Abschnitt 4.3. Es gibt allerdings auch weitere Einflussfaktoren, wie z.B. das Datenvolumen, auf die ebenfalls später eingegangen wird.

Eine weitere wichtige Eigenschaft der Erfassung ist, dass niemand beeinträchtigt werden darf, für den die Erfassung keine direkte Relevanz hat. Zu dieser Gruppe gehört auch der Nutzer selbst. Im Optimalfall nimmt dieser den Prozess der Erfassung nicht wahr. Nichtsdestotrotz muss er einmalig darüber informiert werden, dass eine Datensammlung stattfindet, sowie jederzeit die Möglichkeit haben, diese abzuschalten. Ebenfalls der Gruppe zuzuordnen sind die meisten Entwickler. Bei der Implementierung neuer Funktionen sollte der Fakt, dass für diese Nutzungsdaten erfasst werden könnten, keinen bzw. minimalen Einfluss auf ihre Arbeit haben. Lediglich für Entwickler, die direkt an der Erfassung arbeiten, und Produktmanager, die an den Daten und ihrer Auswertung interessiert sind, darf ein signifikanter Aufwand entstehen.

2.3.2 Zieldefinition

Auf Basis dieser Anforderungen lässt sich aus dem abstrakten Konzept der Nutzungsdatenerfassung ein Ansatz für eine praktische Lösung erarbeiten. Ziel ist die Entwicklung eines Systems, das:

- flexibel zur automatischen Erfassung verschiedenartiger Daten über die Verwendung einer Applikation genutzt werden kann,
- eine Grundlage zur Auswertung der gesammelten Daten bereitstellt,
- eine möglichst große Nutzerbasis umfasst,
- den Nutzer der Anwendung und unbeteiligte Entwickler nicht beeinträchtigt,
- bei der Erfassung ein besonderes Augenmerk auf Datenschutz legt,
- mit wenig programmiertechnischem Aufwand in die SelectLine-Programme integriert werden kann,
- unabhängig von den Updates der zugrunde liegenden Anwendung aktualisiert werden kann.

Im Folgenden wird auf die theoretischen Grundlagen sowie das Konzept der Umsetzung eingegangen.

3. Theoretische Grundlagen

Die Zieldefinition schränkt die Möglichkeiten der Umsetzung bereits stark genug ein, um eine grobe Vorstellung von der Lösung zu bekommen. Allerdings müssen Grundlagen bezüglich der Nutzungsdatenerfassung geschaffen werden, damit die resultierenden Daten eine sinnvolle Analyse erlauben. Dazu muss zum Einen geklärt werden, wie Probleme der Usability identifiziert werden können, und zum Anderen, wie die Gliederung der Datenerfassung in funktionale Einheiten möglich ist.

3.1 Usability Evaluation

Um sich mit dem Thema der Evaluierung auseinandersetzen zu können, muss zuerst die Bedeutung der Begriffe *Usability* (Gebrauchstauglichkeit bzw. Benutzerfreundlichkeit) und *User Experience* geklärt werden.
Usability ist definiert als „das Ausmaß, in dem ein System, Produkt oder Dienst durch bestimmte Benutzer verwendet werden kann, um bestimmte Ziele in einem spezifischen Anwendungskontext effektiv, effizient und zufriedenstellend zu erreichen“[1].
User Experience ist ein weiter gefasster Begriff, der „die Wahrnehmung und Reaktionen einer Person, die sich aus der Nutzung und/oder erwarteten Nutzung eines Produkts, Systems oder Dienstes ergeben“[2], beschreibt.
Die Abgrenzung dieser Begriffe ist wichtig, da für diese Arbeit im Wesentlichen die Usability relevant ist. Der umfassendere Begriff der User Experience schließt viele Faktoren ein, die nicht in Betracht gezogen werden sollen. Das Ziel ist lediglich, Probleme zu finden, welche die Arbeit mit dem Programm erschweren.

Die Analyse und Verbesserung der Usability gehört zur Forschung im Bereich der *Human-Computer Interaction* (HCI) und begleitet diesen Forschungszweig praktisch

[1] frei übersetzt aus ISO 9241-210:2010(en), 2.13, original: „Extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.“

[2] frei übersetzt aus ISO 9241-210:2010(en), 2.15, original: „A person's perceptions and responses resulting from the use and/or anticipated use of a product, system or service.“

seit seinen Anfängen [HAW01]. Im Folgenden werden verschiedene Möglichkeiten der Evaluation beleuchtet, um unterschiedliche Herangehensweisen zu analysieren. Eine Auseinandersetzung mit diesen Ansätzen ermöglicht ein besseres Verständnis der Aspekte der Usability, welches wiederum die Konzipierung einer Lösung für die konkrete Zielstellung erleichtert.

Bei der Evaluierung wird in der Theorie zwischen zwei verschiedenen Ansätzen unterschieden: dem *Usability Testing* (auch *empirische Methoden* genannt), bei dem Nutzer der entsprechenden Zielgruppe die Applikation verwenden und die Nutzungsdaten analysiert werden, und der *Usability Inspection* (*analytische* oder *Inspektionsmethoden*), bei der Experten für Usability direkt nach Problemen suchen [GS98][FIA11][FLH12]. Diese Begriffe werden im Folgenden so verwendet, allerdings wird kein Anspruch auf die Allgemeingültigkeit dieser Klassifizierung erhoben. Beispielsweise werden modellbasierte Systeme häufig als eigene Klasse betrachtet, wohingegen sie hier mit in die Gruppe der Usability Inspection eingeordnet werden, da sie einen Usability-Experten zur Durchführung benötigen.

In der Praxis ist die Trennung in empirische und analytische Methoden sogar nur schwer möglich. Eine Umfrage unter Usability-Praktikern hat gezeigt, dass die Mehrheit Usability Testing verwendet, jedoch zusätzlich Expertenwissen mit einbezieht [FLH12]. Solano et al. empfehlen, Kombinationen von UEMs einzusetzen, wobei jeweils mind. eine Inspektionsmethode und eine empirischen Methode gewählt werden sollte [SCRF16]. Grundsätzlich scheint das Feld durch einen sehr dynamischen und flexiblen Ansatz geprägt zu sein, der sich auch in einer Mischung von etablierten Konzepten und eigenen Ansätzen manifestiert.

Im Folgenden wird eine Auswahl von *Usability Evaluation Methods* (UEMs) vorgestellt, d.h. Prozeduren, die durch eine Menge von Aktivitäten zur Datensammlung in Bezug auf die Interaktion eines Endnutzers mit einem Softwareprodukt, und/oder die Eigenschaften desselben, die zur Usability beitragen, definiert sind [FIA11]. Dies sind theoretische Ansätze, daher ist nicht gewährleistet, dass sie in genau dieser Form zum Einsatz kommen. Wie Følstad et al. [FLH12] gezeigt haben, divergieren Theorie und Praxis sehr stark. Es sind häufig Abwandlungen und Mischformen der UEMs im Einsatz. In diesem Kontext besitzt das allerdings keine große Bedeutung, da vorrangig die zugrundeliegenden Ideen in Bezug auf die Analyse der Arbeitsweise und die Identifikation von Usability-Problemen beleuchtet werden sollen.

3.1.1 Usability Testing

Als Grundlage für UEMs im Bereich des Usability Testings dient im Allgemeinen ein Szenario, in dem ein Nutzer in der Applikation ein definiertes Ziel erreichen soll. Dabei ist meist ein Evaluator anwesend, der Notizen macht und eventuelle Fragen klären kann. Während und nach diesem Vorgang werden UEMs angewandt. Einige Beispiele sind:

”Thinking aloud”-Protokoll

Beim ”Laut Denken” ist der Nutzer dazu angehalten, seine Gedanken zu jedem Arbeitsschritt auszusprechen. Der Evaluator schreibt dabei mit, oft wird zusätzlich ein

Video von der Sitzung für eine spätere Auswertung aufgenommen.
Der genaue Ablauf kann dabei unterschiedlich sein. Ursprünglich stammt die Technik aus der Psychologie, und zielt auf eine einseitige Kommunikation der Testperson mit dem Evaluator ab, da Unterbrechungen den Denkprozess beeinflussen und damit verfälschen könnten. Dementsprechend kann die Methode als UEM nach den gleichen Regeln durchgeführt werden. Da der Kontext ein anderer ist, kann dies allerdings dazu führen, dass die Daten nicht aussagekräftig genug sind, z.B. weil der Evaluator dem Nutzer bei Verständnisproblemen nicht helfen darf. Andererseits können die Regeln etwas gelockert werden, wodurch zwar mehr Daten gesammelt werden, jedoch die Glaubwürdigkeit der Daten verletzt werden könnte, da die Unterbrechungen Einfluss auf die folgenden Gedankengänge des Nutzer haben. Alternativ kann auch eine abgewandelte Methode verwendet werden, wie z.B. das "framework of speech communication" von Boren & Ramay, in dem eine gewisse Form eines Dialoges zwischen Testperson und Evaluator explizit erwünscht ist [KU04].
Eine weitere Entscheidung, die bei der Durchführung von Usability Tests mit dem "Thinking aloud"-Protokoll getroffen werden muss, ist die Anzahl der Testpersonen. Dies ist natürlich auch bei anderen Methoden der Fall. Die meisten Usability-Probleme werden nur von einem kleinen Teil der Nutzer gefunden, allerdings finden verschiedene Personen meist unterschiedliche Probleme. Nielsen [Nie94a] empfiehlt 4 ± 1 Subjekte und zielt damit aus Kostengründen auf ~75% gefundene Probleme ab, wohingegen Hwang und Salvendy [HS10] auf Basis der Forschung der letzten Jahre 10 ± 2 vorschlagen, um diesen Anteil wirklich zu erreichen.
Der Erfolg des "Thinking Aloud" hängt zudem stark davon ab, wie gut die Testpersonen in der Lage sind, ihre Gedanken zur aktuellen Tätigkeit in Worte zu fassen. Schwierigkeiten entstehen durch Probleme, die schwer verbalisierbar sind, sowie durch Personen, denen das Ausformulieren ihrer Gedanken generell schwerfällt.

Performance-Messung

Ein Aspekt der Usability ist, dass Aufgaben schnell und effizient erledigt werden können[3]. Daher bietet es sich an, die Leistung der Testnutzer in einem Szenario zu messen. Diese kann durch verschieden Kriterien definiert werden, z.B. die Dauer für die Ausführung einzelner Schritte, die Anzahl der falschen Aktionen oder wie oft der Nutzer Hilfe benötigt hat.
Bei dieser Form der Evaluation hat das Vorwissen bzw. die Erfahrung des jeweiligen Testnutzers einen wesentlich größeren Einfluss als bei anderen Testmethoden. Daher ist es wichtig, Informationen zum Kenntnisstand zu erfassen und in der Auswertung mit einzubeziehen. Dies sollte generell getan werden, ist allerdings gerade im Kontext einer Performance-Messung von besonderer Bedeutung. Eine gute Diversität in der Gruppe der Nutzer ist nötig, damit allgemeingültige Ergebnisse ermittelt werden können. Aber auch die gezielte Auswertung von Nutzergruppen mit ähnlichen Voraussetzungen bietet Vorteile, wie z.B. eine Einschätzung der Eignung für ungeschulte Anwender.

[3] lt. ISO 9241-210:2010(en), 2.13

Eye-Tracking

Das Eye-Tracking bietet die Möglichkeit, eine sogenannte *Heatmap* von der Oberfläche zu erstellen, d.h. eine graphische Repräsentation davon, wie lange der Nutzer welchen Bereich des UI betrachtet. Heatmaps sind eine zeitliche Zusammenfassung der sogenannten *Gaze Points* (Blickpunkte), die von der Kamera mit einer gewissen Frequenz erfasst werden. Eine genauere Auswertung der Daten liefert *fixations* und *saccades*, d.h. die Zeitspanne über die der Blick an einem Punkt verweilt, sowie die Bewegungen zwischen diesen Punkten [JK03].
Diese Daten ermöglichen es, ein UI auf ein intuitives Design zu überprüfen. Wichtige Elemente müssen in der Heatmap auffallen, fixations und saccades (fixation sequences) sollten sich an das Layout und den gewünschten Arbeitsfluss anpassen. Eine relativ zufällige Verteilung von fixations mit vielen schnellen saccades dazwischen deutet beispielsweise darauf hin, dass der Nutzer nach etwas sucht und nicht sofort fündig wird. Das kann unter Umständen auf ein unübersichtliches Layout zurückzuführen sein.
Das Eye-Tracking liefert Daten, die keine andere Methode in dieser Form bereitstellen kann. Es bietet sich daher sehr gut zur Kombination mit anderen Verfahren an. Beispielsweise kann es eingesetzt werden, um Zeitabschnitte im “Thinking Aloud“-Protokoll zu untersuchen, in denen der Nutzer schweigt. Außerdem kann erfasst werden, ob und wie lange gezögert wird, bevor eine Aktion ausgeführt wird [CC05].

Fragebögen

Eine sehr verbreitete Methode, um Feedback nach Abschluss des Testszenarios zu sammeln, sind vorgefertigte Fragebögen. Dies erleichtert die Auswertung, da gut vergleichbare Ergebnisse von allen Testpersonen entstehen. Weiterhin lassen sich einige Feststellung erst im Nachhinein, rückblickend auf den gesamten Ablauf, tätigen. Die resultierenden Daten sind weniger detailliert als beispielsweise das Protokoll der Sitzung, können jedoch einen schnellen Überblick von Problemen und der allgemeinen Zufriedenheit geben.
Ein großes Problem der Fragebögen besteht darin, dass der Nutzer nicht während, sondern nach der Benutzung seine Meinung abgibt, wodurch die Genauigkeit der Angaben leiden kann. Die Fragebögen sollten daher immer in Kombination mit anderen Methoden, wie beispielsweise dem ”Thinking Aloud”, eingesetzt werden. Weiterhin können schlechte formulierte Fragestellungen zu Verständnisproblemen oder Einschränkungen des Ausdrucksspielraums führen. Der Zeitaufwand zur Erstellung guter Fragebögen darf daher nicht unterschätzt werden.
Eine Alternative stellt der Einsatz von existierenden generischen Fragebögen dar, die der Evaluierung spezieller Aspekte dienen. Beispielsweise fokussieren sich ISO-NORM und ISOMETRICS auf die durch die ISO-Norm 9241-110 vorgegeben Kriterien für Usability, während der NASA TLX auf die Belastungserfassung des Anwenders abzielt [PD15].

3.1.2 Usability Inspection

Methoden der Usability Inspection kommen hauptsächlich zum Einsatz, wenn früh im Designzyklus Erkenntnisse gewonnen werden sollen oder die Ressourcen für Usa-

bility Testing zu knapp sind. Obwohl diese kostengünstigeren Methoden gute Ergebnisse erzielen können, darf Usability Testing nicht komplett vernachlässigt werden. Ein wesentlicher Unterschied besteht darin, dass bei der Usability Inspection lediglich Experten für UI zum Einsatz kommen, die jedoch nicht notwendigerweise Fachkenntnisse im Anwendungsbereich der Applikation besitzen. Ein weiterer Grund ist, dass der Einfallsreichtum von Personen, die eine für sie neuartige Applikation verwenden, die Vorstellungen von Experten bei weitem übersteigt [JD92][4].

Bekannte UEMs aus der Klasse der Usability Inspection sind unter anderem folgende [Nie94b]:

Heuristische Evaluation

Bei diesem Ansatz bewertet ein Experte einzelne UI-Elemente, wie z.B. Dialoge, anhand etablierter Konzepte der Bedienbarkeit, den sogenannten Heuristiken. Häufig kommen dabei die "Acht goldenen Regeln des Oberflächenentwurfs" von Shneiderman [SPC+16] zum Einsatz:

- nach Konsistenz streben
- häufigen Nutzern die Möglichkeit von Abkürzungen bieten
- informative Rückmeldungen geben
- in sich abgeschlossene Dialoge entwerfen
- simple Fehlerbehandlung anbieten
- einfache Umkehrung von Aktionen erlauben
- Kontrollgefühl vermitteln
- Anforderungen an das Kurzzeitgedächtnis reduzieren

Ein weiteres Beispiel sind die neun grundlegenden Usability Prinzipien von Nielsen und Molich [NM90]. Es existieren auch wesentlich umfangreichere und detailliertere Sammlungen von Usability-Richtlinien, wie beispielsweise von Smith und Mosier [SM86], die allerdings gerade aufgrund ihrer hohen Komplexität selten genutzt werden.
Ein häufig angewandte Vorgehensweise ist außerdem, mehrere Experten unabhängig voneinander eine Applikation mit den gleichen Heuristiken bewerten zu lassen. Der Grund dafür ist, dass verschiedene Personen meist auch unterschiedliche Probleme finden. Dies wird als *Evaluator Effekt* bezeichnet und ist Gegenstand weiterer Forschung [Cap06][HMJ14]. Nielsen und Landauer [NL93] schlagen ein Modell vor, anhand dessen das Kosten-/Nutzen-Verhältnis abhängig von der Anzahl der Evaluatoren abgeschätzt werden kann.
Ein typisches Problem der heuristischen Evaluation liegt, wie auch bei den anderen

[4] freie Übersetzung, original: „the ingenuity of people using a novel application greatly exceeds the imaginations of experts"[JD92]

Methoden der Usability Inspection, darin, dass die gefundenen Probleme lediglich die Sicht der Experten für Usability repräsentieren. Tatsächliche Nutzer neigen dazu, andere Aspekte als problematisch einzuschätzen. Die Schnittmenge der von beiden Gruppen gefundenen Problemen ist meist sehr gering, wobei sich zusätzlich die Einschätzung des Schweregrades dieser Probleme unterscheidet [PP12].

Kognitive & Pluralistische Walkthroughs

Bei einem Walkthrough wird ein typischer Prozess eines Nutzers des Programms (ein Szenario) simuliert und entsprechend schrittweise nachvollzogen. Dabei wird bei jedem Schritt evaluiert, ob der aktuelle Kenntnisstand sowie das UI Design zur richtigen nächsten Aktion leiten. Beim pluralistischen Ansatz geschieht diese Simulation mit einem Team aus Nutzern, Entwicklern und Experten für Usability, die sich austauschen sollen, wohingegen der ursprüngliche kognitive Ansatz nur den Experten umfasst.
Im Gegensatz zur heuristischen Evaluation, die sich auf die einzelnen UI-Elemente konzentriert, steht hier deren Zusammenwirken im Kontext eines Prozesses im Vordergrund. Ein besonderes Augenmerk wird auf die Unterstützung von *explorativem Lernen* gelegt, d.h. wie gut die erstmalige Nutzung ohne formales Training möglich ist [RFR95].
Ein Walkthrough ist ein äußerst aufwendiger Prozess, der zudem sehr viel Kompetenz des Usability Experten erfordert. Im Fall des kognitiven Walkthroughs benötigt dieser zusätzlich Kenntnisse im entsprechenden Fachgebiet der Anwendung, um gute Einschätzungen liefern zu können [Hol05]. In Folge dessen handelt es sich um eine sehr kostspielige und daher selten genutzte UEM. Es existieren zudem diverse modifizierte Versionen, die unterschiedliche Probleme des ursprünglichen Ansatzes adressieren, um die Methode besser nutzbar zu machen [TMSK10].

Standards-, Konsistenz- und Featureinspektion

Inspektionen sind Untersuchungen nach vorgegebenen Kriterien, wie z.B. der Übereinstimmung des UI mit vorgegebenen Standards (z.B. ISO 9241-12 bis 9241-17, ISO 14915, IEC 61997 [Bev06]) oder der Konsistenz zwischen mehreren Elementen. Bei der Featureinspektion werden für typische Arbeitsaufgaben die nötigen Arbeitsschritte aufgelistet, woraufhin die resultierenden Sequenzen auf übermäßige Komplexität sowie unnatürliche oder aufwendige Aktionen untersucht werden.

3.1.3 Automatisierungspotential

Eine vollständige Automatisierung von UEMs ist in den meisten Fällen nicht möglich oder praktikabel, unter anderem da viele Methoden subjektive Elemente einbeziehen, wie beispielsweise die Zufriedenheit der Nutzer oder die persönlichen Einschätzungen von Experten. In Folge dessen werden die Methoden im Allgemeinen lediglich durch Tools unterstützt, die bestimmte Teile des Prozesses übernehmen.

Ivory und Hearst [IH01] schlagen eine Taxonomie vor, die den Automatisierungsgrad sowie den zusätzlich nötigen menschlichen Aufwand klassifiziert. Eine UEM kann dabei eine beliebige Kombination der Automatisierung in den Kategorien ”Aufnahme”,

”Analyse” und ”Kritik”, oder keine Automatisierung unterstützen. Der Menschliche Aufwand ist dabei entweder ”minimal” oder erfordert die ”Entwicklung eines Modells”, ”informelle Nutzung” oder ”formelle Nutzung”. Ihre Analyse verschiedener UEMs zeigt, dass nur wenige Methoden eine Automatisierung ermöglichen, und dabei nie alle drei Kategorien unterstützt werden. Außerdem ist lediglich die Analyse mit Hilfe eines ”Guideline Reviews” mit minimalem menschlichem Aufwand möglich. Dabei ist allerdings zu beachten, dass alle Methoden aus dem Bereich des Usability Testing notwendigerweise eine formelle oder informelle Nutzung erfordern. In dieser Hinsicht ist daher auch die Analyse von Log-Dateien, die Aufnahme von Performance-Daten und die Erfassung der Nutzungsdaten beim *Remote Testing*[5] ohne zusätzlichen Aufwand automatisierbar. Tatsächlich sind dies auch Methoden, die ohne die Automatisierung praktisch nicht funktionieren würden.

In einer aktuelleren Studie haben Fernandez et al. [FIA11] ermittelt, dass lediglich 31% der im Web-Bereich genutzten UEMs einen signifikanten Automatisierungsgrad besitzen. Diese fokussieren sich hauptsächlich auf die Simulation eines Nutzers, Auswertung von Log-Dateien oder Überprüfung des Quellcodes. Sie stellen fest, dass die Automatisierung auf die Sammlung objektiver Daten orientiert ist und somit die Perspektive eines Nutzers vernachlässigen. Sie empfehlen daher, dass diese immer durch nicht-automatisierte Methoden unterstützt werden sollten. Eine ihrer abschließenden Feststellungen ist, dass es einen Mangel an automatisierten UEMs gibt, speziell solcher, die früh im Entwicklungsprozess eingesetzt werden können.

3.1.4 Schlussfolgerungen

Im Fokus von UEMs steht im Allgemeinen, die Usability einer Applikation von Personen evaluieren zu lassen, die nicht Teil der Entwicklung waren. Dabei wird entweder auf wenige Usability-Experten oder mehrere Nutzer aus der Zielgruppe der Anwendung gesetzt. Die gesammelten Daten sind dabei meist nur wenig strukturiert und umfassen Nutzungsdaten verschiedener Form (verbale Beschreibung, Videoaufnahme, Log-Dateien, ...) sowie direktes Feedback (unstrukturiert, oder beispielsweise semistrukturiert durch Umfragen). Einige Verfahren umfassen weiterhin das Sammeln von Verbesserungsvorschlägen, allerdings ist dies hauptsächlich bei Inspektionsmethoden der Fall. Beim Usability Testing geschieht dies seltener. Das Ziel ist dort vorrangig, möglichst diverse und umfangreiche Daten zum Nutzungsverhalten und aufgefallenen Problemen zu erhalten. Der limitierende Faktor ist dabei die Anzahl der Nutzer bzw. der Aufwand, der pro Nutzer entsteht, da die Kosten-Nutzen-Funktion einen logarithmischen Anstieg besitzt. Wie bereits erwähnt, existieren daher mathematische Modelle zur Abschätzung des Trade-Offs zwischen Anzahl der Nutzer und gefundenen Problemen [NL93] [Nie94a][HS10].

Automatisierung setzt an dieser Stelle an, indem versucht wird, den Aufwand und damit die Kosten zu verringern. Für die Analyse der Daten ist das schwer umsetzbar. Das Sammeln von Daten ist allerdings ein typischer Anwendungsfall für automatische Systeme. Je nach Art der gesammelten Daten steigt der Analyseaufwand

[5] Als Remote Testing werden alle Methoden bezeichnet, bei dem die Anwesenheit des Testsubjekts in einem Usability Lab nicht erforderlich ist.

verschieden stark an. Optimal sind Daten, die sich leicht aggregieren lassen oder bereits vorab gefiltert, reduziert oder anderweitig modifiziert werden, um eine Auswertung zu erleichtern. Protokolle von"Think Aloud" Sitzungen liefern beispielsweise umfangreiche Daten, die Analyse lässt sich jedoch nicht signifikant vereinfachen. Eye-Tracking-Daten hingegen sind gut aggregierbar, wodurch allgemeine Aussagen getroffen werden können.

Für die Nutzungsdatenerfassung in den SelectLine Applikationen ergeben sich daraus einige Schlussfolgerungen. Insbesondere der bereits beleuchtete Aspekt, dass die Daten von möglichst vielen Nutzern gesammelt werden sollten, wird bestätigt. Weiterhin wird klar, dass viele Ansätze mit einem System zur automatischen Erfassung in einem großen Analyseaufwand resultieren. Es muss daher bei der Wahl der zu erfassenden Daten darauf geachtet werden, dass diese reduziert und aggregiert werden können, bevor sie zentral gesammelt werden. Daten, für die dies nicht möglich ist, sollten nur in einem begrenztem Rahmen erfasst werden, beispielsweise um eine spezielle Hypothese zu überprüfen. Für solche Fälle wäre eine Möglichkeit, die Erfassung auf Nutzer mit bestimmten Kriterien einzuschränken, von Vorteil. Auch in Bezug auf die Art der zu sammelnden Daten lassen sich Rückschlüsse ziehen. So sind beispielsweise Informationen zum Verlauf der Nutzung von Funktionen in einem Kontext sinnvoll, da anhand dessen der übliche Handlungsablauf identifiziert und mit dem erwarteten Verhalten abgeglichen werden kann. Dies gleicht einem pluralistischen Walkthrough. Auch Performance-Messungen lassen sich mit einem automatischen System einfach durchführen, und liefern gut auswertbare Daten.

3.2 Erfassung durch Agenten

Um die Nutzungsdatenerfassung in funktionale Einheiten aufzugliedern, bietet sich das von Hilbert und Redmiles [HR98] [HR01] vorgestellte Verfahren der *agentenbasierten Erfassung* an. Ihr System fällt in die Kategorie des Remote Testing und hat das Ziel, die Limitationen von klein angelegten Tests in Usability Labs durch den Einsatz des Internets zur Einbindung vieler Nutzern zu umgehen. Die Vorteile bestehen darin, dass die Nutzer in ihrer normalen Arbeitsumgebung alltägliche Aufgaben erledigen können, sowie dass die Erfassung über einen längeren Zeitraum stattfinden kann.

Der Ansatz zeichnet sich durch den Einsatz von *Agenten* aus, die jeweils die Verantwortung für einen spezifischen Aspekt der Erfassung übernehmen. Jeder dieser Agenten arbeitet eigenständig und wertet die beim Nutzer gesammelten Daten aus, d.h. es findet eine Vorverarbeitung der Rohdaten statt. Abbildung 3.1 zeigt den Ablauf eines Entwicklungsprozesses, der durch ein solches System unterstützt wird.

Entscheidend ist dabei das iterative Vorgehen: Agenten werden gemeinsam mit der Applikation entwickelt (1) und sammeln daraufhin beim Nutzer Daten (2 bis 4), woraufhin sie die Daten zurücksenden (5) und diese genutzt werden, um sowohl Anwendung als auch Agenten weiterzuentwickeln (6), womit der Prozess sich wiederholt. Dabei ist zu beachten, dass der Nutzer in Schritt 2 nicht jedes Mal eine neue Version der Anwendung erhält – der Prozess beschreibt zwar die Verbesserung

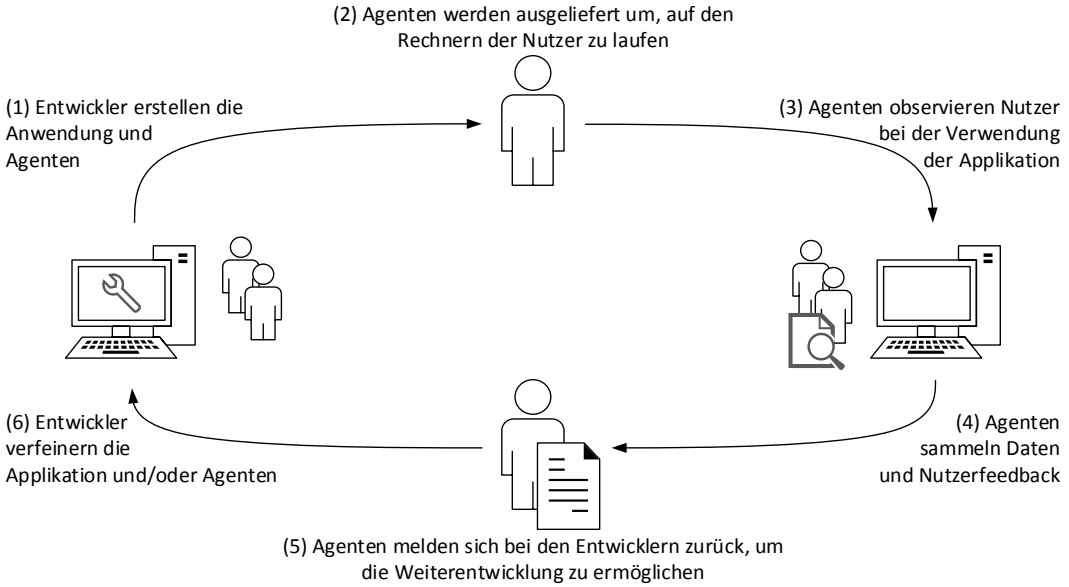

Abbildung 3.1: Entwicklungsprozess mit agentenbasierter Nutzungsdatenerfassung, adaptiert von Hilbert und Redmiles [HR98, Figure 1]

der Applikation, die Auslieferung selbiger ist jedoch unabhängig vom Zyklus der Agenten.

Abbildung 3.2 zeigt die Architektur des Prototypen für das ”expectation driven even monitoring system” (EDEM) [HR98]. Das System selbst setzt sich dabei aus zwei Komponenten zusammen: dem Server, der die Agenten bereitstellt und die gesammelten Daten empfängt und speichert, sowie dem Client, der den Kern des EDEM darstellt und durch den die Agenten mit der Anwendung interagieren. Der Client lädt über eine URL die Definitionen der Agenten in Form von ASCII-Zeichenketten herunter und überträgt die gesammelten Daten per E-Mail zurück.

Den Agenten stehen zur Erfassung lediglich von der Applikation speziell bereitgestellte Daten zur Verfügung. EDEM liefert zu diesem Zweck alle UI-Events sowie Informationen zum geöffneten Fenster. Dies erfordert nur einen minimalen Aufwand für die Integration in die Applikation (es handelt sich um lediglich zwei Zeilen Javacode [HR01, Section 4.2]). Bei der Implementierung der Agenten muss dementsprechend ausgewählt werden, auf welche Events reagiert werden soll. An diesem Punkt wird die Besonderheit des Ansatzes deutlich: statt die erfassten Event-Daten direkt zu sammeln, werden diese zuerst durch die Agenten verarbeitet. Der Grund dafür liegt hauptsächlich darin, dass Redundanz vermieden und das Datenvolumen eingeschränkt werden soll. Aufgrund des Transfers über das Internet sowie einer zentralen Speicherung der Daten sehr vieler Nutzer ist es essentiell, die Menge der Daten zu senken. Zusätzlich zur Reduktion der benötigten Bandbreite bzw. des Speicherplatzes sinkt auch der Aufwand bei der Auswertung, der bei großen Datenmengen sonst eine besondere Herausforderung darstellt [MGT+04].

Ein Problem bei der Sammlung der Rohdaten liegt darin, dass Events eine niedrige Abstraktionsebene darstellen. Um Aktionen des Nutzers erfassen zu können,

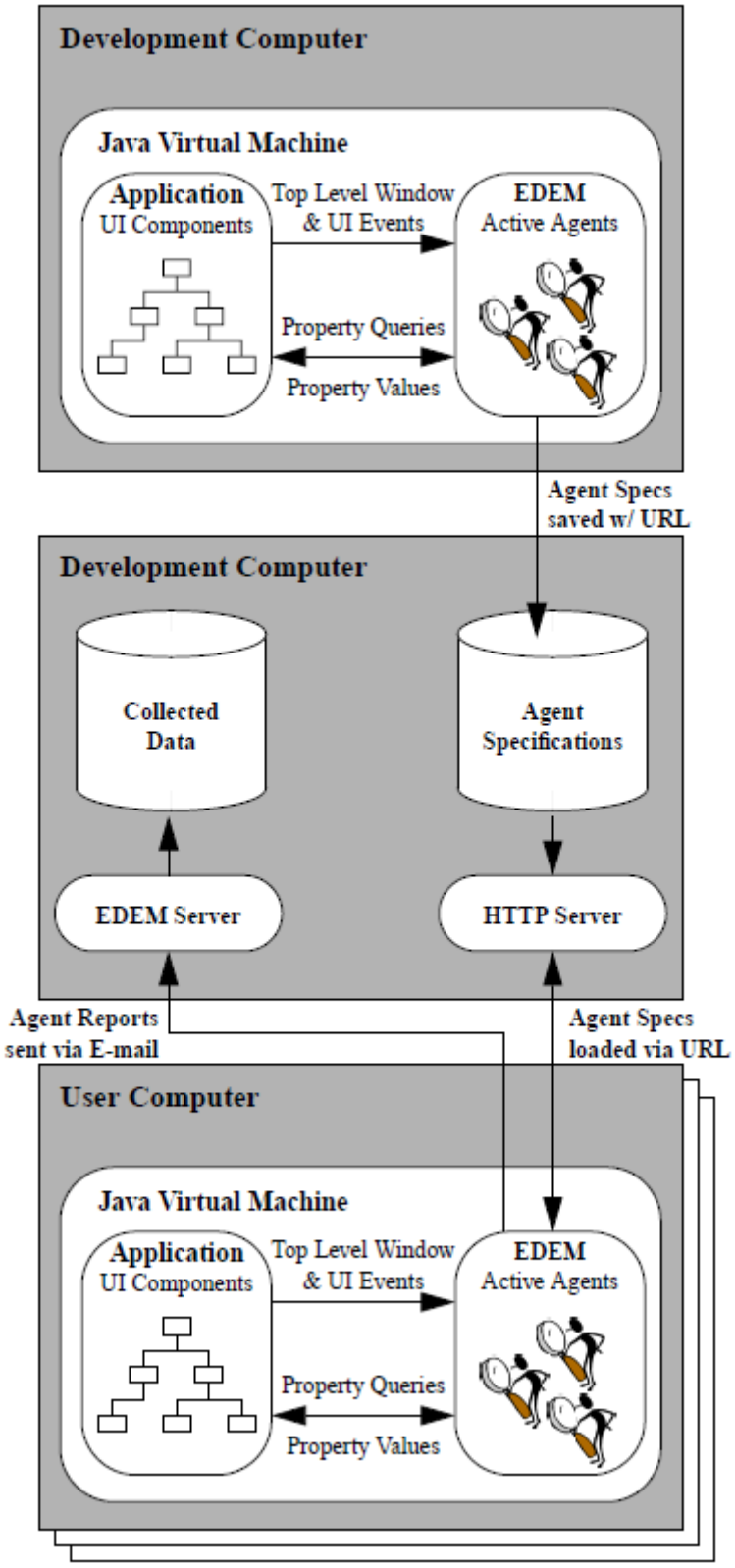

Abbildung 3.2: EDEM Architektur [HR98, Figure 5]

ist daher zuerst eine Interpretation der Events notwendig, aus denen sich diese zusammensetzen. Renaud und Gray [RG04] beschreiben die Herausforderungen und Lösungsansätze im Umgang mit Daten, die von einem System gesammelt werden, das lediglich die grundlegenden Windows-Events aufzeichnet. Schwierigkeiten beim Einsatz des "Generic Remote Usage Measurement Production System" (GRUMPS) stellen insbesondere die korrekte Kontextzuordnung sowie Phantom-Daten, wie z.B. durch den Start das Bildschirmschoners oder durch wiederholte Events bei längerem Druck von Tasten, dar.

Agenten müssen somit zuerst Informationen ableiten, die mehr Aussagekraft besitzen, um verwertbare Daten für die Auswertung bereitstellen zu können. Hilbert und Redmiles [HR01] schlagen dazu eine geschichtete Architektur vor, bei der Agenten zuerst die Rohdaten in Form von Events selektieren und abstrahieren, woraufhin diese rekodiert werden, so dass andere Agenten die vorverarbeiteten Daten zur Verfügung haben. Abbildung 3.3 verdeutlicht die Schichten einer Applikation, die ein solches System zur Nutzungsdatenerfassung einsetzt.

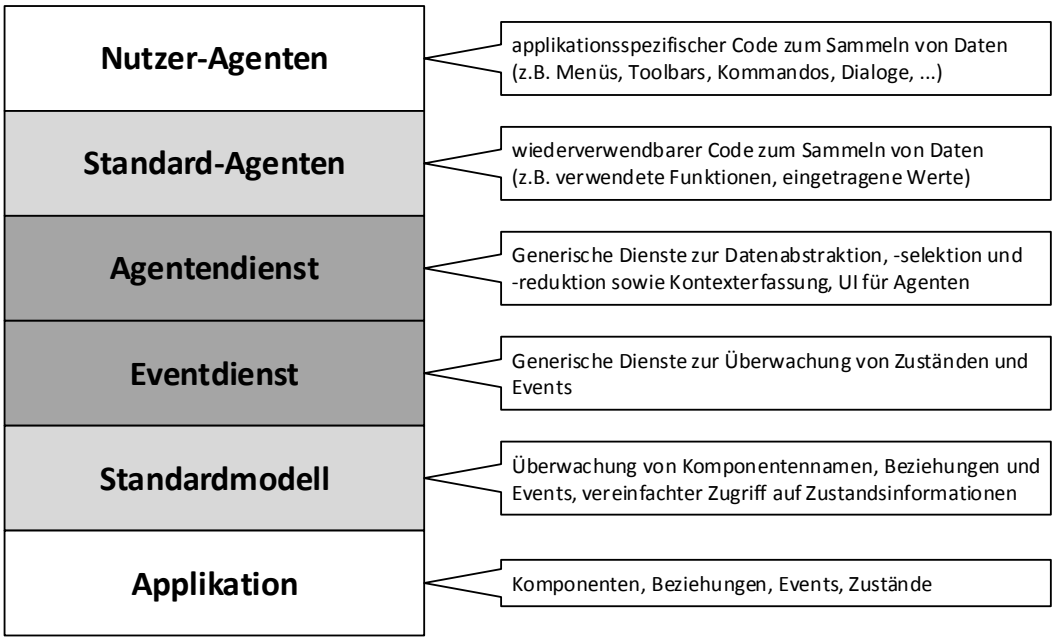

Abbildung 3.3: Die Beziehungen der geschichteten Architektur. Schattierungen zeigen den Grad der Wiederverwendbarkeit der entsprechenden Komponente. Adaptiert von Hilbert und Redmiles [HR01, Figure 2]

Hilbert und Redmiles [HR00] heben im Rahmen der Untersuchung verschiedener Systeme zur Auswertung von Event-Daten hervor, dass eines der Hauptprobleme vieler automatisierter Ansätze die fehlende Möglichkeit der Transformation, d.h. Abstraktion, Selektion und Rekodierung, der Daten ist. Außerdem fehlt Methoden, die diese Transformationen nicht direkt zum Zeitpunkt der Erfassung erlauben, der Zugriff auf Kontextdaten zur Unterstützung der Selektion und Abstraktion durch weitere Zustandsinformationen. Der Einsatz von Agenten erlaubt es, beide Probleme zu umgehen, allerdings muss der Zugriff auf entsprechende Kontextdaten vorbereitet

werden. Weiterhin stellen sie fest, dass ein Zugriff auf zusätzliche Events höherer Abstraktionsebenen äußerst hilfreich sein kann. Diese müssen von der Applikation zusätzlich bereitgestellt werden. Im Gegensatz dazu stehen Windows- und UI-Events im Allgemeinen ohne speziellen Aufwand zur Verfügung. Daher muss dieser Gewinn an Informationen gegen den nötigen Aufwand in der Implementierung sowie den Einflusses auf die weitere Entwicklung abgewogen werden. Für das EDEM System kommen, wie bereits erwähnt, lediglich die vorhandenen UI-Events zum Einsatz.

Existierende Systeme zur Nutzungsdatenerfassung konzentrieren sich aufgrund dieser Schwierigkeiten meist auf Webseiten. Dort ist es wesentlich einfacher, benötigte Daten zu erfassen. Grundlegende Informationen können bereits durch einen Proxy-Server ermittelt werden, der die gesendeten und empfangenen Daten analysiert. Weiterhin ist es mit geringem Aufwand möglich, eine ausgelieferte Seite um Skripts zu ergänzen, die einen erweiterten Funktionsumfang der Erfassung bieten. Im Allgemeinen kommt dabei JavaScript zum Einsatz [AWS06].

Das wohl bekannteste Beispiel ist Google Analytics[6], welches neben der Analyse des Traffics sowie der Interaktion mit externen Diensten auch diverse Funktionen zur Visualisierung der Interaktionen der Nutzer bietet. Ein gutes Beispiel ist die Darstellung des Benutzerflusses, d.h. des Weges, den die Nutzer auf der Webseite nehmen. Informationen dieser Art sind sehr wertvoll um zu prüfen, wie übersichtlich und intuitiv die Navigation ist. Auf einer Webseite sind diese Daten verhältnismäßig leicht zu erfassen, da jeder Seitenwechsel einen expliziten Aufruf darstellt, der leicht aufgezeichnet werden kann. In normalen Applikationen hingegen ist es sehr schwer, ohne eigens implementierte Events eine solche Visualisierung zu ermöglichen.
Beispiele für kommerzielle Lösungen sind AppDynamics[7], New Relic[8] oder SPM[9]. Es existieren auch diverse Open Source Systeme, wie beispielsweise Scouter[10].

Ein System für den Einsatz in den SelectLine Programmen muss jedoch für Desktop-Applikationen funktionieren. Der Ansatz von Hilbert und Redmiles stellt daher eine gute Grundlage dar. Die Agenten bieten die Möglichkeit, die Erfassung in sinnvolle Einheiten zu gliedern. Gleichzeitig deckt sich der iterative Ansatz in Verbindung mit dem Download der Agenten sehr gut mit der Anforderung, die Erfassung unabhängig von Programmupdates anpassen zu können.

[6] https://www.google.com/analytics/
[7] https://www.appdynamics.com
[8] https://www.newrelic.com
[9] http://sematext.com/spm/
[10] https://github.com/scouter-project/scouter

4. Konzept

In Anbetracht der Diversität der Anforderungen, sowohl in Bezug auf die hergeleitete Zieldefinition, als auch auf die herkömmlichen Methoden der Usability Evaluation, muss die Lösung flexibel gestaltet sein. Das geplante System stellt dementsprechend in erster Linie einen Rahmen zur Umsetzung spezifischerer Anforderungen dar, und deckt selbst lediglich die grundlegenden bzw. geteilten Verantwortlichkeiten. Dazu wird ein Server sowie ein Client-Modul entwickelt, die einer auf Agenten basierenden Nutzungsdatenerfassung nach Vorbild des EDEM-Systems dienen.

Im Folgenden wird zuerst auf den groben Aufbau der Lösung und daraufhin auf einige Schwerpunkte eingegangen.

4.1 Funktionsumfang

4.1.1 Client

Die Verantwortlichkeit des Client-Moduls besteht darin, die Rohdaten zu sammeln und diese den Agenten zur Verfügung zu stellen. Dabei ist es wichtig, dass das Modul mit geringem Aufwand in einer Applikation eingesetzt werden kann. Die Agenten müssen vom Server heruntergeladen werden und die gesammelten Daten der Agenten müssen zurück zum Server gesendet werden. Zusätzlich muss für den Nutzer die Möglichkeit bestehen, die Erfassung an- oder auszuschalten. Gleichzeitig sollte transparent sein, welche Daten erfasst werden. Daher muss das Client-Modul ein User Interface (UI) anbieten, über das alle aktiven Agenten und ihre Aufgabe einsehbar sind. Diese benötigen somit jeweils einen Namen und eine Beschreibung. Abbildung 4.1 und Abbildung 4.2 zeigen den Entwurf für die Oberfläche.

Damit das Client-Modul Agenten verwenden kann bzw. diese Zugriff auf die Rohdaten erhalten, muss es eine gemeinsame Schnittstelle geben. Diese sollte flexibel genug sein, um die Verarbeitung der Rohdaten und die Zugriffsmöglichkeiten auf Kontextdaten nicht einzuschränken. Die Komplexität sollte jedoch gering gehalten werden, um die Implementierung der Agenten nicht zu erschweren. Weiterhin wird

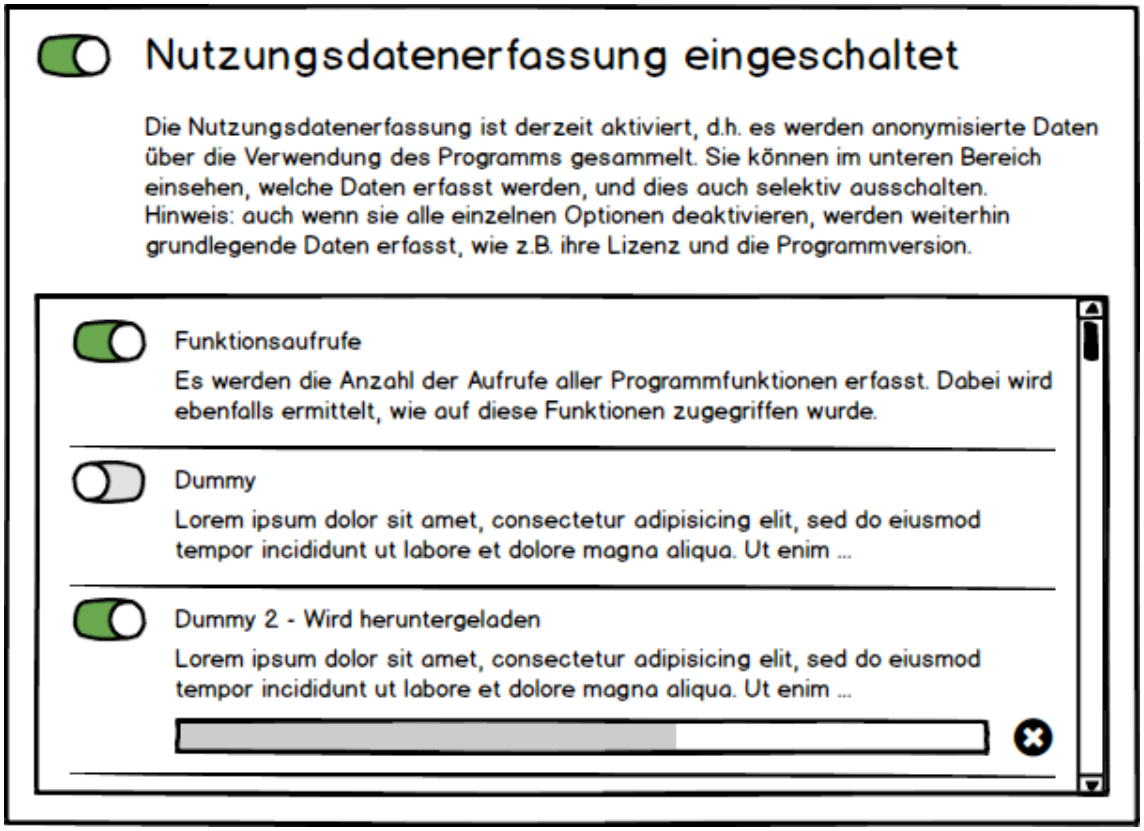

Abbildung 4.1: Mockup des Client UI bei aktivierter Nutzungsdatenerfassung

ein "Basis-Agent" benötigt, der diese Schnittstelle verwendet, sodass für die Agenten keine Funktionalität dupliziert werden muss.

4.1.2 Server

Der Server setzt sich aus drei Komponenten zusammen: der REST[1]-Schnittstelle (im folgenden API[2] genannt), der Datenbank und einer Weboberfläche (*Frontend*).

Die API ist die Schnittstelle, mit der das Client-Modul arbeitet. Sie muss Funktionen zum Senden der Agenten sowie zum Empfangen der gesammelten Daten bereitstellen. Clients müssen sich authentifizieren, damit die Daten nicht durch unautorisierte Zugriffe verfälscht werden können. Das genaue Vorgehen wird im Rahmen der Betrachtung der Sicherheit in Abschnitt 4.3.2 erläutert.

Das Frontend dient der Verwaltung der Agenten sowie der Auswertung der gesammelten Daten. Es werden daher Übersichten zu den Agenten und Clients benötigt, sowie die Möglichkeit, die gesammelten Daten anzusehen und für die weitere Auswertung zu exportieren (z.B. als CSV-Datei). Der Server muss eine Nutzerverwaltung besitzen, um den Zugriff für unautorisierte Personen einzuschränken. Dabei sollten Nutzern verschiedene Rechte zugewiesen werden können, so dass beispielsweise

[1] Representational State Transfer, ein Programmierparadigma, das sich insbesondere durch *Zustandslosigkeit* auszeichnet.
[2] Application Programming Interface (Schnittstelle zur Anwendungsprogrammierung)

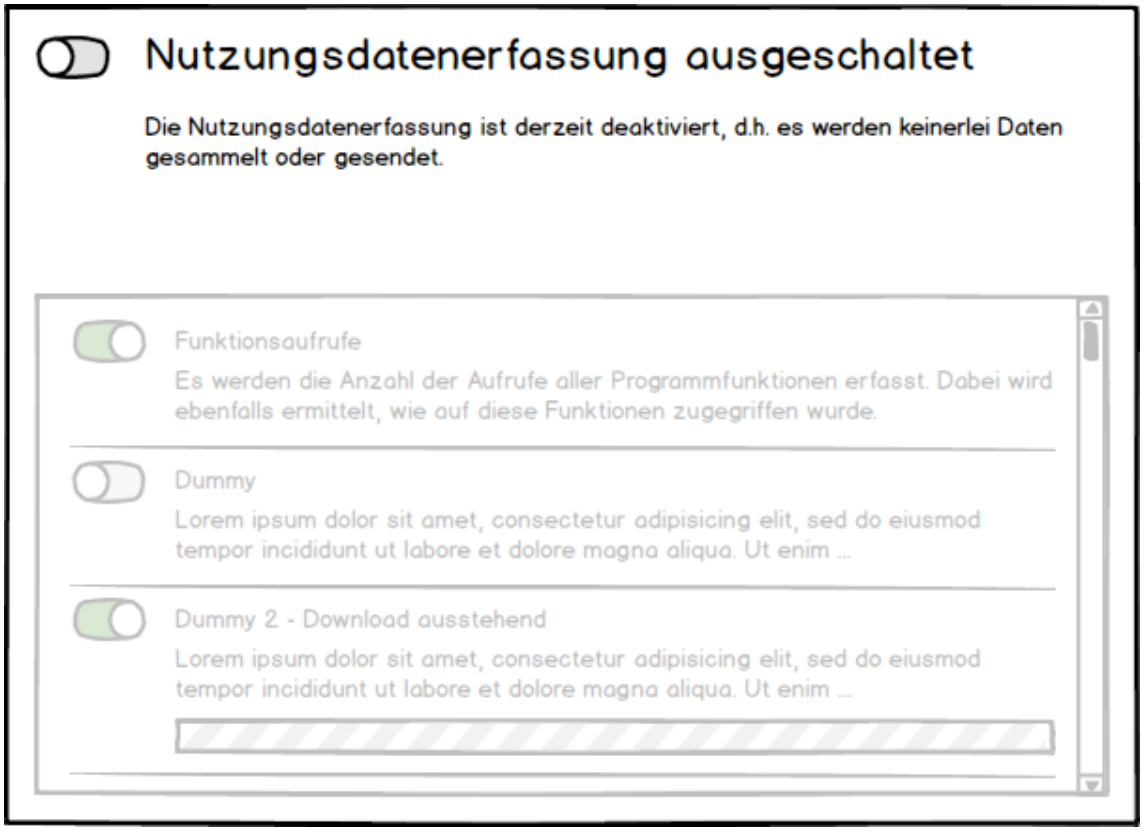

Abbildung 4.2: Mockup des Client UI bei deaktivierter Nutzungsdatenerfassung

für einen spezifischen Nutzer zwar die Auswertung der Daten, allerdings nicht das Hochladen oder Löschen von Agenten möglich ist.

Die Datenbank speichert die Agenten sowie alle gesammelten Daten. Außerdem enthält sie zusätzliche Tabellen für die Funktionalität des Frontends. Dabei sollte das Datenbankschema so gestaltet sein, dass bei einer direkten Auswertung mit anderen Tools die Möglichkeit besteht, die Daten in verwertbarer Form direkt aus der Datenbank zu beziehen.

Abbildung 4.3 fasst den Zusammenhang zwischen dem Server, seinen Komponenten und den Clients zusammen.

4.2 Sammlung der Rohdaten

Die Erfassung der Rohdaten stellt eines der Kernprobleme in der Umsetzung des Systems dar. Die Schwierigkeit liegt dabei darin, eine Grundlage für eine umfassende Auswertung zu schaffen. Dazu müssen in erster Linie genug Daten vorhanden sein, allerdings müssen diese auch aussagekräftig sein.

Die Notwendigkeit, Rohdaten zu sammeln, entsteht aus dem Ansatz der agentbasierten Erfassung. Da die Agenten unabhängig von der Applikation sind, besteht für sie keine Möglichkeit, direkt auf den Anwendungscode zuzugreifen, um Informationen zu sammeln. Sie sind daher auf eine Datenquelle angewiesen, die von der Anwendung

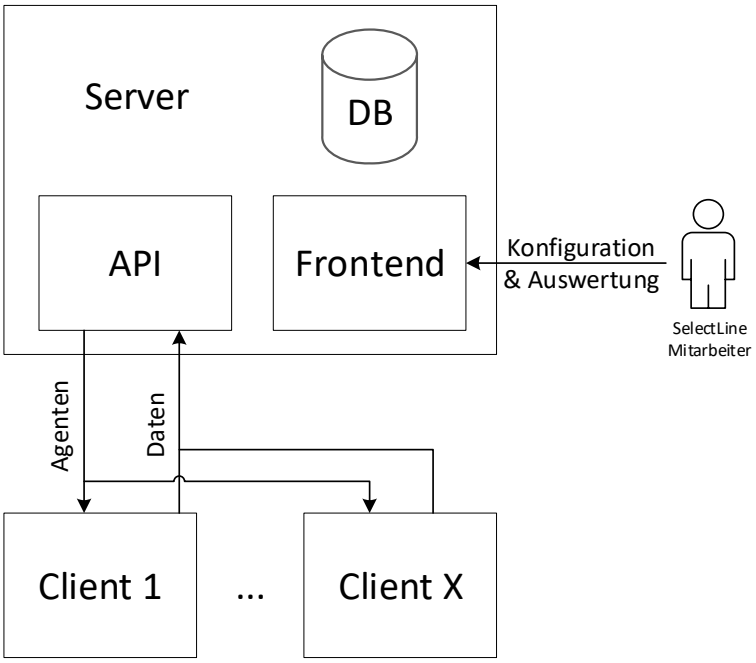

Abbildung 4.3: Client-Server-Architektur

speziell zu diesem Zweck bereitgestellt wird. Um die Flexibilität der Erfassung nicht einzuschränken, muss die Datenquelle möglichst viele und detaillierte Informationen liefern.

Die SelectLine Programme werden in Delphi und C# implementiert. Für den ersten Einsatz des Systems ist die Wahl auf eine C# WPF[3]-Applikation gefallen, da die Rohdatenerfassung in diesem Fall einfacher ist. Das liegt darin begründet, dass WPF umfangreiche UI-Events bereitstellt, die eine gute Grundlage darstellen. Im Gegensatz dazu müsste in Delphi auf die reinen Windows-Events zurückgegriffen werden, die weniger detailliert sind, oder manuell eine Erfassung aller Events der VCL[4] implementiert werden. Ein weiterer Vorteil besteht darin, dass in C# geschriebene Anwendungen sogenannter *managed code* sind. Der Code wird zu einer *Intermediate Language* (IL) kompiliert, die erst zur Laufzeit durch das .NET-Framework zu echtem Maschinencode umgewandelt wird. Das ermöglicht umfangreiche automatische Anpassungen an Anwendungen, ohne den Originalcode zu verändern, indem die IL in den durch das Kompilieren erzeugten Assemblies modifiziert wird. Typischerweise wird diese Möglichkeit genutzt, um zusätzliche Sicherheitsmaßnahmen umzusetzen, wie z.B. Obfuscation[5] des Codes. Es ist allerdings auch möglich, dies zu nutzen, um *aspektorientierte Programmierung* (AOP) anzuwenden.

AOP adressiert das Problem von Aufgaben bzw. Verantwortlichkeiten eines Programmes, die sich schlecht in die übrige Abstraktionsstruktur (z.B. Domänenobjekte) integrieren lassen. Während sich eine normale Applikation aus Komponenten

[3] Windows Presentation Foundation, ein Oberflächenframework für Windows.
[4] Visual Component Library, Kapselung des Windows Forms Oberflächenframeworks.
[5] Eine Methode zur Erschwerung des Reverse Engineering von Applikationen.

zusammensetzt, die meist in definierten Schichten organisiert sind (z.B. MVVM[6]), lassen sich solche Funktionen nicht in diese Architektur einordnen und sind sogenannte *cross-cutting concerns*. An dieser Stelle greift das Konzept der Aspekte: diese existieren neben den normalen Komponenten und werden nachträglich durch den *Aspect Weaver* in das Programm integriert. Diese Integration kann zur Compile-Zeit stattfinden, oder per *Reflection*[7] zur Laufzeit geschehen. Dabei gibt es bestimmte *Join Points* im normalen Code (üblicherweise Funktionsaufrufe), an denen der Aspektcode eingreift und die Funktionalität anpassen kann [KLL+02]. So lassen sich beispielsweise Decorators umsetzen, die vor und nach dem Aufruf einer spezifischen Funktion eine Aktion durchführen.

Der größte Vorteil besteht darin, dass übergreifende Funktionalitäten, wie beispielsweise Logging oder Fehlerbehandlung, zentral abgehandelt werden können, ohne wiederholt in den normalen Code integriert werden zu müssen. Die Verantwortlichkeiten sind wesentlich besser getrennt und es entstehen weniger Abhängigkeiten, wodurch sich die Wartbarkeit der Software erhöht. Zusätzlich reduziert der aspektorientierte Ansatz den Aufwand weiterer Entwicklung, da Änderungen an den Komponenten keine Änderung an den Aspekten erfordern.

Die größte technische Schwierigkeit beim Einsatz von AOP stellt das *Code Weaving* dar, d.h. das Injizieren des zusätzlichen Codes an den Join Points. Programmiersprachen wie C#, deren Code zu einer IL compiliert werden, erleichtern diesen Prozess, da der Bytecode der IL viele Meta-Informationen bereitstellt und leichter modifiziert werden kann. Weiterhin ist die Reflection in solchen Sprachen meist vielseitiger einsetzbar. Bei Programmiersprachen wie Delphi, die direkt zu nativem Code compiliert werden, ist das Weaving zur Compilezeit nahezu unmöglich, und auch die Umsetzung zur Laufzeit gestaltet sich durch reduzierte Möglichkeiten der Reflection als schwierig.

Die Nutzung von AOP zur Nutzungsdatenerfassung wurde bereits in der Forschung thematisiert. Lettner und Holzmann [LH12] präsentieren ein Toolkit zur Entwicklung von mobilen Applikationen, das die Sammlung von Nutzungsdaten ohne Änderung des Applikationscodes ermöglicht. Nach dem Compilieren wird ihr Framework durch einen zweiten Compile-Vorgang in die Applikation integriert. Dieser Ansatz hat große Ähnlichkeiten mit dem System dieser Arbeit, allerdings findet keine Vorverarbeitung durch Agenten statt. Zu kritisieren ist daher, dass die kompletten Daten direkt an einen Server gesendet werden, ohne dabei den Datenschutz zu bedenken.

4.3 Sicherheit & Datenschutz

Speziell in Anbetracht des gestiegenen öffentlichen Bewusstseins stellt der Datenschutz ein extrem wichtiges Thema für diese Arbeit dar. Insbesondere durch den Fokus der SelectLine Applikationen – kaufmännische Anwendungsfälle in kleinen bis mittelständische Unternehmen – arbeiten die Kunden mit äußerst sensiblen Daten. Beispielsweise kann es sich um Kontaktinformationen sowie Zahlungsdaten von

[6] Model-View-ViewModel, ein Programmierparadigma für die Applikationsarchitektur.
[7] Methoden zur Modifikation des bereits compilierten Codes zur Laufzeit.

Privatpersonen und Firmen handeln, mit denen der Kunde in Kontakt steht. Eine der wichtigsten Funktionen des Systems besteht daher darin, die Erfassung komplett abschalten zu können. In diesem Fall muss gewährleistet sein, dass keine Daten an den Server gesendet werden.

Ebenfalls wichtig ist die Sicherheit im Sinne von Security – das System darf nicht missbraucht werden können. In Bezug auf die Architektur umfasst das den Schutz aller gesammelten Daten auf dem Server und auf dem Kommunikationsweg vor unberechtigtem Zugriff, sowie das Verhindern des Missbrauchs der Agentenschnittstelle für Schadsoftware. Außerdem sollte die Manipulation der gesendeten Daten bzw. das Einspeisen von falschen Informationen verhindert werden. Dies hat allerdings eine geringere Priorität, da das Opfer in diesem Fall nicht der Kunde ist. Die Security ist nicht immer klar vom Datenschutz zu trennen, da viele Maßnahmen letztlich den Zugriff auf die Daten einschränken.

Der Begriff Sicherheit umfasst auch den Aspekt der Safety. Diese ist im Kontext dieser Arbeit allerdings kaum relevant, da das System keinen Einfluss auf die Arbeitsweise oder die Daten des Kunden nimmt, sondern lediglich einen beobachtenden Charakter besitzt. Im Folgenden wird der Begriff der Sicherheit daher synonym mit Security verwendet.

4.3.1 Datenschutz

Wie bereits erwähnt, kann die Nutzungsdatenerfassung komplett abgeschaltet werden. Es können allerdings stattdessen auch einzelne Agenten deaktiviert werden (siehe Entwurf der Oberfläche, Abbildung 4.1). Auch wenn die Erfassung für jeden Agenten ausgeschaltet ist, kommuniziert der Client weiterhin mit dem Server, beispielsweise um neue Agenten herunterzuladen. Im Gegensatz dazu stellt das Deaktivieren der gesamten Erfassung sicher, dass jegliche Verbindung zum Server verhindert wird. Zusätzlich wird der Nutzer beim ersten Start aufgefordert, sich explizit für oder gegen die Aktivierung der Nutzungsdatenerfassung zu entscheiden. Der Einsatz geschieht dementsprechend ”opt-in”. Das System unterstützt allerdings auch den ”opt-out”-Ansatz. In diesem Fall entfällt die Abfrage oder wird durch einen Hinweis darauf, dass die Erfassung aktiv ist und abgeschaltet werden kann, ersetzt.

Client-seitig obliegt die Verantwortung für alle weiteren Maßnahmen den Agenten selbst. Die Rohdatenerfassung ist sehr generisch und darf zu keinen Performance-Einbußen führen, daher ist eine Filterung von sensiblen Daten an dieser Stelle nicht möglich. In Folge dessen müssen die Agenten müssen dafür sorgen, dass ihre Vorverarbeitung die Daten entsprechend anonymisiert, bevor sie an den Server gesendet werden. Dabei ist insbesondere zu beachten, dass keine Eingaben in Formularfelder gesammelt werden dürfen.

Sobald die Daten vom Client an den Server gesendet werden, wird davon ausgegangen, dass korrekt anonymisierte Information vorliegen. Trotzdem dürfen diese nicht öffentlich verfügbar sein, da auch aus den übrigen Daten potentiell persönliche Informationen gewonnen werden können. Die Verbindung zwischen Client und Server wird daher durch den Einsatz von HTTPS bei der Verwendung der API vor unberechtigtem Zugriff geschützt. Die API soll außerdem nicht dazu genutzt werden,

Daten auszulesen. Da die Clients dies nicht benötigen, müssen keine entsprechende Funktionen angeboten werden. In Bezug auf den Datenschutz bleiben somit nur Bedenken bezüglich des Zugriffs auf die gesammelten Daten über das Frontend oder den SQL-Server übrig. Dies sowie Angriffs- bzw. Manipulationsszenarien gehören zum Bereich der Security.

4.3.2 Sicherheit

Für die Serverseite sind diverse Sicherheitsmaßnahmen vorstellbar. Eine sehr grundlegende Möglichkeit zum Schutz der Daten besteht darin, dass der SQL-Server keinen Zugriff von außen erlaubt. Zusätzlich sollte eine Nutzerverwaltung aktiv sein, die dafür sorgt, dass lediglich das Backend der Serversoftware sowie ein Admin-Account zur Wartung Zugriff haben. Die entsprechenden Login-Daten dürfen aus Datenschutzgründen auch innerhalb der Firma nicht frei zur Verfügung stehen.

Weiterhin müssen alle an den Server gesendeten Daten (sowohl via Frontend als auch via API) so verarbeitet werden, dass kein Schadcode injiziert werden kann (z.B. SQL Injections).

Das Frontend muss ebenfalls eine Nutzerverwaltung besitzen, wobei diese ein Rechtesystem umfassen sollte. Dadurch kann gewährleistet werden, dass nur autorisierte Mitarbeiter Zugriff auf Teile des Systems haben, die für sie relevant sind. So dürfen beispielsweise User der Gruppe "Analyst" zwar die gesammelten Daten einsehen, allerdings keine neuen Agenten hochladen oder die Rechte anderer Nutzer verwalten.

Die API erfordert eine spezielle Authentifizierung, die sicherstellen kann, dass es sich bei der Gegenseite wirklich um einen Client handelt. Ohne diese Maßnahme könnte jeder, der die ausgehenden Verbindungen prüft und so die Funktionen der API ermittelt, falsche Daten in das System einspeisen. Zusätzlich müssen sich Clients eindeutig identifizieren lassen, wobei allerdings keine Rückschlüsse auf die Identität des zugehörigen Kunden möglich sein dürfen. Jeder Client benötigt daher eine ID. Der Aufbau dieser folgt gewissen Regeln und kann daher auf Authentizität geprüft werden, sodass ein potentieller Angreifer keine zufällige ID verwenden kann. Es können daher lediglich die Daten von Clients, deren ID bekannt ist, manipuliert werden.

4.4 Verwaltung & Verwendung der Agenten

Die Agenten stellen als eigenständige funktionale Einheiten die Besonderheit der in dieser Arbeit vorgestellten Lösung dar. Das System dient im Wesentlichen dazu, die Rahmenbedingungen für die Agenten zu schaffen. im Folgenden werden die einzelnen Bestandteile in Bezug auf ihre Arbeit mit den Agenten betrachtet.

4.4.1 Client

Das Client-Modul liefert die Rohdaten und bietet für die Agenten alle notwendigen Funktionen, um mit dem Server zu kommunizieren. Um die Flexibilität in der

Verwendung möglichst groß zu halten, sind Agenten eigenständige Binärdateien, die vom Client eingebunden werden können. Dadurch können die Agenten komplett frei implementiert werden. Der Client muss allerdings eine Schnittstelle anbieten, damit alle benötigten Funktionen bereitgestellt werden können.

Ein besonders wichtiger Aspekt der Nutzungsdatenerfassung ist, dass sie den Anwender nicht beeinträchtigt. Die Performance spielt daher eine große Rolle. Das Client-Modul muss, soweit möglich, unabhängig von der eigentlichen Applikation laufen. Für die Erfassung selbst ist das nicht möglich, daher muss insbesondere an diesem Punkt auf eine effiziente Implementierung geachtet werden. Die Vorverarbeitung sowie das Senden der Daten können hingegen asynchron ablaufen und dementsprechend auch rechenintensiver sein. Da dies Aufgabe der Agenten ist, liegt die Verantwortlichkeit nicht ausschließlich beim Client-Modul. Dies muss bei der Implementierung der Agenten beachtet werden.

4.4.2 API

Die API stellt das Bindeglied zwischen Client und Server dar. Sie gliedert sich in die Verwaltung der Agenten, sowie das Schreiben der gesammelten Daten. Zusätzlich müssen die bereits erwähnten Sicherheitsmaßnahmen für den Zugriff existieren. Die Verwaltung umfasst Funktionen zum Herunterladen der Informationen über alle Agenten, die für den Client relevant sind, Download dieser Agenten, sowie Meldung darüber, welche von ihnen aktiv sind. Für das Senden der Daten gibt es mehrere Möglichkeiten. Unterscheiden lässt sich zwischen Kombinationen aus einer von drei verschiedenen Schreibstrategien (Create, Increment, Update) sowie einer der beiden Sendestrategien (Single, Batch).

Die verschiedenen Ansätze beim Schreiben der gesammelten Daten ergeben sich aus unterschiedlichen Anforderungen vorab identifizierter Szenarien für den Einsatz von Agenten. Beispielsweise erfordert das Speichern der Hardwarespezifikationen das Überschreiben bestehender Einträge (Update) anstelle neuer Datensätze (Create), wie sie beispielsweise beim Logging verwendet werden. Sollen Funktionsaufrufe gezählt werden, reicht auch das nicht aus, da ohne Kenntnis der bisherigen Menge ein gewisser Wert hinzugefügt können werden muss (Increment). Sowohl Update als auch Increment erfordern dabei einen zusätzlichen Schlüssel, falls mehr als ein Datensatz pro Client gespeichert werden soll. Dieser wird zur Identifikation des zu bearbeitenden Eintrags genutzt. Beim Create ist dies nicht nötig, da immer nur neue Datensätze hinzugefügt werden.

Die Möglichkeit, statt einzeln (Single) auch zusammengefasst (Batch) Daten senden zu können, dient der Reduzierung des Overheads in der Kommunikation. Agenten sollten Sendungen bündeln und in regelmäßigen Zeitabständen bzw. vor Beendigung der Applikation verschicken.

Erneut hervorzuheben ist, dass aus Sicherheitsgründen keine Funktionen angeboten werden, um gesammelte Daten vom Server auszulesen (siehe Abschnitt 4.3.1). Die einzigen an den Client gesendeten Daten betreffen die Agenten selbst. Eine komplette Übersicht aller verfügbaren Funktionen (API-Referenz) findet sich in Abschnitt A.3

im Anhang. Neben diesem Dokument existiert zusätzlich eine interaktive und detailliertere Version der API-Referenz, sowie ein Sequenzdiagramm, welches den Ablauf der Kommunikation über die API verdeutlicht.

4.4.3 Server

Auf der Serverseite besteht die Herausforderung darin, eine Architektur zu schaffen, welche die Arbeit mit individuellen Agenten und ihren Daten einfach gestaltet. Unterschiedliche Anwendungsfälle resultieren in verschiedenartige Daten. Es wird somit eine entsprechende Struktur in der Datenbank nötig, um diese zu speichern. Zu Auswertungszwecken sind dedizierte Spalten mit verschiedenen Datentypen wünschenswert, allerdings ist dies nicht möglich, wenn alle Agenten ihre Daten in einer gemeinsamen Tabelle speichern. Das Ziel ist daher, dass jeder Agent eine eigene Tabelle mit selbst definierten Spalten besitzt.

Das Problem dabei ist, dass die Agenten zur Laufzeit hinzugefügt werden. Dadurch ist es nicht möglich, das Datenbankschema im Voraus zu definieren. Gleiches gilt für das Frontend, da auch bei der Verwaltung der Agenten sowie der Auswertung ihrer Daten eine Trennung erforderlich ist. Insgesamt wird daher eine dynamische Lösung notwendig. Dies ist mittels *Metaprogrammierung* möglich, welche es erlaubt, neuen Code für jeden Agenten zu erzeugen. Abschnitt 5.4.2 beschreibt genauer, wie dies vonstatten geht.

5. Umsetzung

Das vorgestellte Konzept wurde im Rahmen dieser Arbeit auch implementiert, um später in den SelectLine-Produkten zum Einsatz zu kommen. Dementsprechend muss zuerst ein Prototyp entwickelt und getestet werden, sodass die Funktionalität gewährleistet werden kann. Ein Einsatz in der Praxis bedeutet eine Auslieferung an einen großen Kundenstamm. Daher muss sichergestellt werden, dass das System keine Probleme verursacht. Im Fokus liegt daher die Erstellung dieses Prototypen sowie dessen Test. Der Einsatz wird exemplarisch im SelectLine CRM demonstriert und getestet.

Im Folgenden wird beleuchtet, welche grundlegenden technischen Entscheidungen getroffen und wie diverse Schwerpunkte in der Implementierung umgesetzt wurden.

5.1 Technische Grundlagen

Die im Konzept hervorgehobene Trennung von Server und Client wird auch bei allen technischen Entscheidungen deutlich. Da es sich um zwei eigenständige Systeme handelt, die lediglich durch eine Schnittstelle (die API) kommunizieren, ist auch die Technik unabhängig voneinander. Demzufolge lassen sich auch alle Entscheidungen einer der beiden Komponenten zuordnen.

Server

Bei der Server-Komponente ist die erste Frage, auf welcher Technik die drei Bestandteile (Datenbank, Frontend und API) basieren sollen. Die Wahl fiel dabei auf Ruby On Rails für Frontend und API, sowie einen MySQL-Server als Datenbank. Der Grund dafür ist hauptsächlich, dass es mit Ruby on Rails sehr leicht ist, APIs zu erstellen. Frontend und API mit einer Lösung abzudecken reduziert zudem den Aufwand, da die Datenanbindung nur einmal implementiert werden muss. Der MySQL-Server könnte dabei durch praktisch jeden anderen SQL-Server ersetzt werden.

Die Entscheidung für MySQL liegt lediglich darin begründet, dass es der Standard zur Verwendung mit Ruby On Rails ist.

Ein weiterer Vorteil von Ruby on Rails ist der in Ruby integrierte Paketmanager und die daraus resultierende Vielzahl an leicht einzusetzenden "Gems", d.h. Bibliotheken, für die meisten Standardprobleme. Somit kann der Aufwand, speziell im Bereich des Frontends, stark reduziert werden. Beispielsweise wird die Benutzerverwaltung über das populäre Gem "Devise"[1] abgehandelt und dabei durch "CanCan"[2] sowie "RoleModel"[3] zur Umsetzung von Benutzerrechten ergänzt. Dadurch bleibt mehr Zeit für die Entwicklung der Kernaspekte des Systems.

Ebenfalls maßgeblich durch ein Gem beeinflusst ist das Design der Oberfläche. Da das Frontend hauptsächlich als Überblick für viele verschiedene Daten sowie zur Verwaltung der Agenten dient, wird die komplette Seite mit "ActiveAdmin"[4] umgesetzt. Dies ist keine übliche Vorgehensweise, da das Gem eher als Ergänzung zu einer existierenden Seite gedacht ist. Für das Frontend ist es in diesem Fall jedoch absolut ausreichend und daher auch als eigenständige Lösung gut einsetzbar. ActiveAdmin ist (genau wie Devise) eine sogenannte "Engine", d.h. ein ergänzendes Modul für eine Ruby on Rails Applikation, und bietet die Möglichkeit, mit wenig Code Seiten zum Einsehen, Anlegen, Bearbeiten und Entfernen von beliebigen Objekten zu erstellen. Das reduziert den Aufwand für das UI und ermöglicht es, mehr Zeit in die Strukturen des Backends, d.h. die Verwaltung der Agenten und der gesammelten Daten, zu investieren.

Client

Für das Client-Modul ist in erster Linie entscheidend, welche Programmiersprache verwendet wird. Maßgebend ist dabei die Verwendbarkeit für die SelectLine Produkte, die hauptsächlich in Delphi und C# entwickelt werden. Wie bereits erwähnt soll der Prototyp vorerst nur in eine Applikation integriert werden. Die Wahl ist dabei auf das neueste Produkt der Familie, das SelectLine CRM[5], gefallen, da dieses die modernste Codebase besitzt und in C# mit WPF implementiert ist. Diese Kriterien waren ausschlaggebend, da die Erfassung der Rohdaten unter diesen Bedingungen am vielseitigsten ist. Die Gründe dafür wurden bereits in Abschnitt 4.2 erläutert. Um die Verknüpfung zwischen Applikation und Client-Modul einfach zu gestalten, wird für das Modul ebenfalls C# verwendet. Dies schließt allerdings eine spätere Verwendung in Delphi-Applikationen nicht aus, da die DLL auch dort verwendet werden könnte, wenn auch mit zusätzlichem Aufwand.

Das Client-Modul stellt im Wesentlichen die Verknüpfung zwischen Agenten und Anwendung her. Dazu müssen entsprechende Schnittstellen zur Applikation und zu den Agenten bereitgestellt werden. Die Kommunikation zwischen Client-Modul und Anwendung umfasst zum einen die Übermittlung der gesammelten Rohdaten (von

[1] https://github.com/plataformatec/devise
[2] https://github.com/CanCanCommunity/cancancan
[3] https://github.com/martinrehfeld/role_model
[4] https://github.com/activeadmin/activeadmin
[5] https://www.selectline.de/selectline-crm/

Anwendung zu Client-Modul) und zum Anderen die UI-Interaktionen (beidseitig). Für beide Fälle muss eine möglichst simple Möglichkeit bereitgestellt werden, um den Aufwand bei der Integration des Modulls in eine Applikation gering zu halten.

Für die Übertragung der Rohdaten kommen ”TraceSources” zum Einsatz. Diese sind Teil des .NET Frameworks und werden üblicherweise für Tracing und Logging genutzt. Auch die existierenden Event-Daten bezüglich der WPF Oberflächen werden in dieser Form bereitgestellt. Das Client-Modul erwartet dementsprechend eine TraceSource-Instanz, mit der es arbeiten kann. Über diese müssen sämtliche erfasste Rohdaten eingespeist werden. Dies bietet den Vorteil, dass das existierende System der ”TraceListener” zur Auswertung durch die Agenten genutzt werden kann. Eine TraceSource kann mit beliebig vielen TraceListenern verknüpft werden, die den kompletten Datenstrom nach entsprechenden Filterkriterien auswerten und passende Elemente zur weiteren Verarbeitung erhalten. Dementsprechend muss jeder Agent lediglich einen TraceListener bereitstellen, um die Datenverarbeitung umzusetzen. Durch diese Herangehensweise vereinfacht sich auch die innere Struktur des Client-Moduls, da die Verteilung der Rohdaten an die Agenten lediglich das Verknüpfen der TraceListener erfordert.

Für die Anbindung des UI stellt das Client-Modul ein Interface mit allen nötigen Informationen sowie Events und Funktionen bereit. Zusätzlich werden bereits eine View (nach dem Entwurf im Konzept, siehe Abbildung 4.1 sowie Abbildung 4.2) und ein passendes ViewModel, welches das Interface implementiert, bereitgestellt. Die Nutzung dieser ist optional, alternativ kann eine Ableitung erstellt oder eine komplett eigene Implementierung gewählt werden. Dies erlaubt sehr viel Freiraum in der Umsetzung, allerdings auch eine einfache Einbindung, wenn keine speziellen Anpassungen notwendig sind.

Die Verbindung des Client-Moduls mit dem Server geschieht über den Zugriff auf die API via HTTPS. Solche Verbindungen lassen sich mit den von .NET bereitgestellten Klassen umsetzen, allerdings gibt es fertige 3rd Party Bibliotheken, die dies deutlich vereinfachen können. In diesem Fall wird die Kommunikation mit der API über ”RestSharp”[6] abgebildet, gestützt durch eine Fehlerbehandlung mit ”Polly”[7]. Das ermöglicht eine einfache Verwendung der API, wobei eventuell auftretende, vereinzelte Netzwerkprobleme durch automatische Wiederholung im Fehlerfall umgangen werden können. Einer der größten Vorteile RestSharps ist dabei die Unterstützung des async-await-Patterns[8], die andernfalls manuell hätte implementiert werden müssen. Eine Ausnahme stellt hier lediglich der Download der Agenten dar, der manuell umgesetzt werden musste, um dort ebenfalls eine zufriedenstellende Umsetzung der Asynchronität zur Verfügung zu haben.

Die Verwendung des async-await-Patterns zieht sich durch das komplette Client-Modul, um der Anforderung gerecht zu werden, dass die Arbeit des Nutzers nicht durch die Nutzungsdatenerfassung beeinträchtigt werden darf. Alle Aufrufe innerhalb des Moduls, die mit API-Anfragen in Verbindung stehen, werden asynchron ausgeführt, um den Programmfluss nicht zu blockieren. Weiterhin geschieht die komplette Auswertung der Daten durch die Agenten in einem separaten Worker-Thread.

[6] https://github.com/restsharp/RestSharp
[7] https://github.com/App-vNext/Polly
[8] Spezielle Form der asynchronen Programmierung in C#, verfügbar seit der Spezifikation 5.0.

Lediglich die Filterung der interessanten Daten der TraceSource sowie die Beschaffung von Kontextdaten muss synchron erfolgen, da andernfalls nicht sichergestellt werden kann, dass die benötigten Informationen noch bereitstehen. Zu beachten ist hierbei, dass die Agenten dadurch eine Teilverantwortung für die Performance-Einbußen tragen. Es muss daher sichergestellt werden, dass die Filterung der Daten keine zeitintensiven Aktionen beinhaltet. In Folge dessen sollten im Zweifelsfall mehr Daten akzeptiert und erst später aussortiert werden, wenn die Verarbeitung asynchron abläuft und daher keinen wesentlichen Einfluss auf die Geschwindigkeit der Applikation nimmt.

5.2 Sammlung der Rohdaten

Wie bereits erwähnt, müssen die Daten für die Agenten durch die Anwendung über eine TraceSource bereitgestellt werden. Da WPF bereits umfangreiche Informationen in dieser Form liefert, konnte ohne großen Aufwand eine grundlegende Erfassung umgesetzt werden. Es stellte sich jedoch heraus, dass diese Herangehensweise nicht optimal war. Bei Tests mit einer Prototyp-Applikation, deren UI dem des SelectLine CRM ähnelt, wurden alle Daten der WPF TraceSource direkt in eine Textdatei geschrieben. Die Ergebnisse zeigen, dass innerhalb von 10 Sekunden aktiver Nutzung ca. 100MB aufgezeichnet werden. Obwohl der eigentliche Schreibprozess asynchron ablief, führte dies zu einer merklichen Verlangsamung des Programms. Einer der Hauptgründe für die große Menge an Daten besteht darin, dass jede Aktion auf einem UI-Element nicht nur zu einem Eintrag für die entsprechende Klasse, sondern auch jeweils zu einem Eintrag für alle Vorfahren führt. Durch den Einsatz von 3rd Party Komponenten entstehen lange Vererbungsketten, die wiederum zu einer Vervielfachung der über die TraceSource gesendeten Informationen führen. Diese zusätzlichen Daten sind redundant und stellen daher lediglich einen Faktor dar, der die Auswertung erschwert.

Ein weiteres Problem ergab sich daraus, dass die Daten durch WPF in Textform bereitgestellt werden. Dadurch besteht kein Zugriff auf die Elemente, auf die sich die Informationen beziehen, wodurch keine Kontextdaten bezogen werden können. Da auch der Name des betroffenen Elements nicht mitgeliefert wird, ist eine Identifizierung nicht möglich. Klicks auf verschiedene Buttons in Menüs lassen beispielsweise nicht unterscheiden. Dadurch sinkt die Aussagekraft der Daten beträchtlich.

Der zweite im Konzept vorgestellte Ansatz, die aspektorientierte Programmierung, stellte sich als bessere Wahl heraus. Nach einiger Recherche und mehreren Tests mit einer Prototyp-Applikation und dem SelectLine CRM fiel die Wahl auf *PostSharp*[9] als Framework zur Umsetzung der AOP. Kostenlose Alternativen, wie beispielsweise *SheepAspect*[10], wurden seit Jahren nicht mehr weiterentwickelt und funktionierten entweder nur in der Prototyp-Applikation, oder mit keinem der beiden Programme. PostSharp erlaubt die Anwendung von Aspekten durch den Einsatz von Attributen, wobei diese entweder direkt an einer Funktion oder Klasse stehen, oder durch Angabe eines Namespaces und zusätzlichen Optionen über mehrere Klassen hinweg angewendet werden.

[9] https://www.postsharp.net/
[10] https://sheepaspect.codeplex.com/

```
1 using PostSharp.Extensibility;
2 using SelectLine.Erp.UsageDataCollection;
3
4 [assembly: CrmUDCAspect(AttributeTargetTypes = "SelectLine.Erp.Crm.Wpf.*",
      AttributeTargetMemberAttributes = MulticastAttributes.Public,
      AttributePriority = 1)]
5 [assembly: CrmUDCAspect(AttributeTargetTypes =
      "SelectLine.Erp.Crm.Wpf.*ViewModel*", AttributePriority = 2)]
6 [assembly: CrmUDCAspect(AttributeTargetTypes =
      "SelectLine.Erp.Crm.Wpf.UsageDataCollection*", AttributeExclude = true,
      AttributePriority = 2)]
7 [assembly: CrmUDCAspect(AttributeTargetTypes = "SelectLine.Erp.Crm.Wpf.App",
      AttributeExclude = true, AttributePriority = 2)]
```

Quelltext 5.1: Anwendung des Aspekts im SelectLine CRM

Quelltext 5.1 zeigt die Attribute für die Haupt-Assembly[11] des SelectLine CRM. Der Aspekt `CrmUDCAspect` wird auf allen public-Methoden innerhalb der Assembly angewandt. Bei ViewModels wird die Einschränkung auf öffentliche Funktionen aufgehoben. Weiterhin wird alles im `UsageDataCollection`-Namespace ausgeschlossen, da die Funktionszugriffe bei der Erfassung sonst ebenfalls aufgezeichnet werden würden, wodurch ein StackOverflow[12] entsteht. Außerdem werden alle Funktionen, die für den Start und die Initialisierung der Applikation verantwortlich sind, ignoriert, da zu diesem Zeitpunkt noch keine Agenten zur Verfügung stehen.

Quelltext A.1 (im Anhang) zeigt den Code des verwendeten Aspekts. In der Funktion `CompileTimeValidate` wird dafür gesorgt, dass der Zugriff auf Property-Getters nicht aufgezeichnet wird, da diese für die Erfassung nicht interessant sind. Kern des Aspekts ist die Funktion `OnEntry`, die beim Zugriff auf jedes Element, auf das der Aspekt angewendet wird, ausgeführt wird. An dieser Stelle werden die Informationen über den Aufruf an die TraceSource weitergegeben.

5.3 Sicherheit

Wie im Konzept erläutert, benötigt die API eine Authentifizierung, um unautorisierte Zugriffe zu verhindern. Da es sich um eine REST API handelt, ist das System ”stateless”, d.h. jede Anfrage ist in sich abgeschlossen und der Server muss keine Sessions verwalten. Die Authentifizierung muss daher immer Teil der Anfrage sein. Zu diesem Zweck kommen *JSON Web Tokens* (JWTs)[13] zum Einsatz.

JWTs werden häufig für Authentifizierungen in REST APIs verwendet, da sie leicht einsetzbar und sehr kompakt sind. Weiterhin existieren fertige Implementierungen für die meisten Programmiersprachen, die den Aufwand weiter verringern. Zusätzlich

[11] Das SelectLine CRM setzt sich aus mehreren Bestandteilen zusammen, die separat als Assemblies kompiliert werden. Die Haupt-Assembly wird dabei zu der ausführbaren Applikation, alle weiteren zu DLLs.
[12] Ein Fehler, der meist durch zu häufige rekursive Aufrufe entsteht.
[13] entsprechend RFC 7519, siehe https://tools.ietf.org/html/rfc7519

dienen die Erweiterungen JSON Web Signature (JWS)[14] zur Signierung sowie JSON Web Encryption (JWE)[15] zur Verschlüsselung des Inhalts. Die entsprechenden Algorithmen sind durch die JSON Web Algorithms (JWA) Spezifikation[16] definiert. [Pey16]

Im Wesentlichen kommen für die API zwei verschiedene Token zum Einsatz, die beide der Authentifizierung dienen. Weiterhin gibt es einen speziellen Token für den Download von Agenten, der allerdings keine Relevanz für die Sicherheit besitzt. Einer der für die Authentifizierung verwendeten Token wird vom Client erstellt, wobei ein "shared secret" bei der Signierung zum Einsatz kommt, d.h. ein Schlüssel, den sowohl Client als auch Server kennen. Abbildung 5.1 zeigt den Aufbau eines solchen JWT. Die Signierung dient dazu, dem Server die Überprüfung der Authentizität des Inhalts (Payload) zu ermöglichen. In diesem Fall muss es sich um die ausgehende IP sowie die ID des Clients handeln. Zudem wird ein sehr kurzer Gültigkeitszeitraum durch einen "expiration claim" festgelegt. Dieser Token muss gemeinsam mit den anderen grundlegenden Daten des Clients, d.h. ID und Programm- sowie Lizenzinformationen, in einer initialen Anfrage an die API gesendet werden (`/access_token`, siehe Abschnitt A.3 im Anhang). Das Kopieren und Wiederverwenden dieser Anfrage durch einen Angreifer ist durch die Einbeziehung der ausgehenden IP und der kurzen Gültigkeit des signierten JWT nicht möglich, außer der Angreifer verwendet den selben Internetzugang wie das Opfer und agiert sehr schnell. Ohne das shared secret zu kennen ist es nicht möglich, die IP oder den expiration claim im JWT auszutauschen, da andernfalls die Signatur ungültig wird. Grundsätzlich wird jeder Manipulationsversuch jedoch bereits dadurch erschwert, dass die Verbindung zur API über HTTPS stattfindet und somit nicht mitgelesen werden kann. Da die JWTs on-demand erzeugt werden und sich lediglich im Arbeitsspeicher befinden, entfallen somit alle einfachen Möglichkeiten, existierende Token zu erlangen. Übliche Angriffsszenarien nutzen meist in Cookies gespeicherte JWTs. Da diese nur in Browsern zum Einsatz kommen, tritt dieser Fall hier jedoch nie ein.

Erfolgt die initiale Anfrage korrekt, liefert der Server einen weiteren Token zurück. Dieser enthält lediglich die ID des Clients und ist mit einer Signatur versehen, die auf einem Geheimnis des Servers basiert. Dadurch kann der JWT nicht mehr verändert werden, ohne ungültig zu werden. Weiterhin ist er mit einem Ablaufdatum versehen, sodass die vollständige Authentifizierung regelmäßig wiederholt werden muss. Alle folgenden Anfragen an die API müssen diesen Token zur Validierung beinhalten. Der Server kann dadurch mit weniger Aufwand als bei der erstmaligen Authentifizierung sicherstellen, dass eine Anfrage korrekt ist, und sie dem Client zuordnen.

Ein weiteres übliches Angriffsszenario, bei dem die Signatur entfernt wird und die Header-Informationen entsprechend angepasst werden, wird dadurch verhindert, dass explizit mit einer bestimmten Methode (HMAC und SHA256) signierte Token gefordert werden. Unsignierte JWTs werden dadurch automatisch abgewiesen.

Es ist zu beachten, dass keine dieser Maßnahmen perfekt ist. Durch den Einsatz eines shared secret besteht die Gefahr, dass dieses durch Decompilierung des Client-

[14] entsprechend RFC 7515, siehe `https://tools.ietf.org/html/rfc7515`
[15] entsprechend RFC 7516, siehe `https://tools.ietf.org/html/rfc7516`
[16] RFC 7518, siehe `https://tools.ietf.org/html/rfc7518`

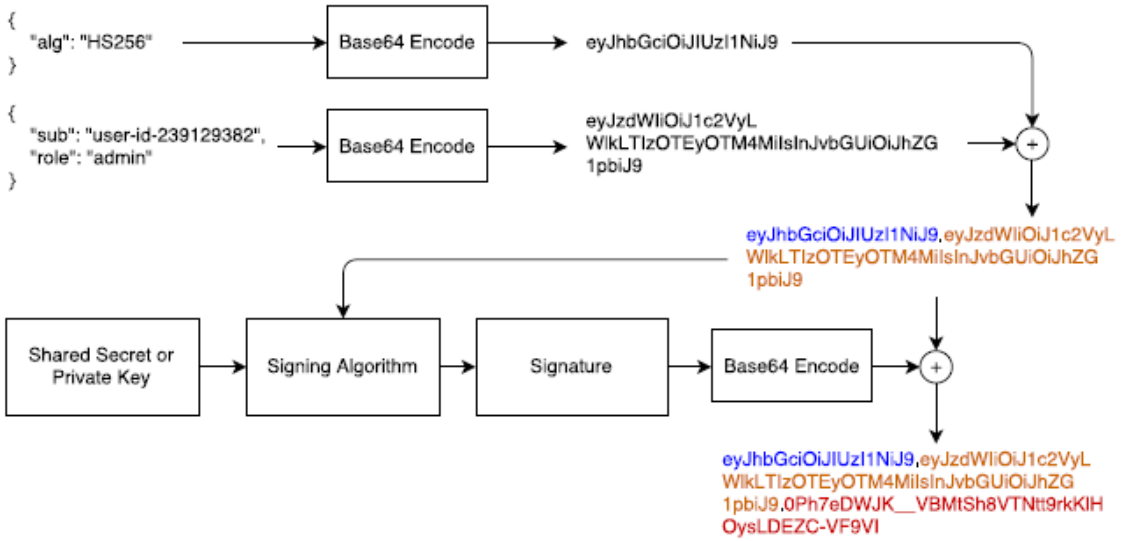

Abbildung 5.1: Aufbau eines signierten JWT [Pey16, Figure 4.1]

Moduls erlangt wird, wodurch die Signierung des ersten Tokens kompromittiert werden kann. Durch Obfuscation lässt sich zwar das Risiko senken, allerdings nicht komplett verhindern. Weiterhin kann vom PC des Nutzers bzw. aus dem selben Netz der gleiche Token zur erstmaligen Authentifizierung verwendet werden. Der resultierende zweite Token kann sogar von beliebigen System genutzt werden. Beide lassen sich jedoch nur dazu einsetzen, falsche Daten für den jeweiligen Client einzuspielen. Dadurch wird eine groß angelegte Manipulation erschwert.

Die Probleme ließen sich womöglich durch weitere Maßnahmen weiter reduzieren. Jedoch muss an dieser Stelle die Frage gestellt werden, ob der Aufwand gerechtfertigt wäre. Bereits durch den Einsatz der vorgestellten Maßnahmen werden Manipulationen erheblich erschwert. Realistisch ist lediglich die Einspeisung von falschen Daten für wenige Client IDs, wodurch der Einfluss auf die Analyse der Daten relativ klein bleibt. Ein umfangreicher Angriff würde nicht nur den Besitz des shared secret, sondern auch die Möglichkeit der Erzeugung gültiger Client IDs erfordern. Weiterhin lassen sich Clients, die durch überdurchschnittlich viele oder stark abweichende Daten auffallen, in der Auswertung leicht herausfiltern. Nicht zuletzt muss auch bedacht werden, dass die Manipulation der Daten kaum einen Vorteil für einen potentiellen Angreifer mit sich bringt. Auf weitergehende Maßnahmen wurde daher verzichtet. Wesentlich wichtiger ist der Schutz der gesammelten Daten vor unberechtigtem Lesezugriff, für den bereits umfassende Maßnahmen präsentiert wurden.

Ein weiterer Aspekt der Security ist der Schutz der Agentenschnittstelle vor Missbrauch. Vorstellbar wäre beispielsweise eine DLL mit einer Klasse, die zwar das Interface implementiert und daher korrekt geladen wird, allerdings zusätzlich Schadcode ausführt. Wie bereits erwähnt, ist der Zugriff auf die Funktionen zum Bereitstellen von Agenten stark limitiert. Zusätzlich wird kein direkter Zugriff auf die entsprechenden Dateien auf dem Server gewährt. Clients müssen stattdessen ein Download-Ticket in Form eines JWT anfordern, wobei überprüft wird, ob der Client diesen Agenten verwenden soll. Das Ticket kann daraufhin genutzt werden, um eine Dateiübertragung zu initiieren. Das Ersetzen der DLLs ist daher nur clientseitig möglich.

Dagegen kann ebenfalls vorgegangen werden, indem nur signierte DLLs akzeptiert werden oder eine Prüfsumme gebildet wird, die mit dem Original vom Server verglichen werden kann.

5.4 Verwaltung & Verwendung der Agenten

5.4.1 Client

In Folge der Entscheidung für komplette DLLs als einzelne Agenten agiert das Client-Modul gewissermaßen als Plugin-System, dass lediglich die Schnittstelle zwischen Programm und den Agenten bietet. Dies bringt große Flexibilität mit sich, da beliebige Funktionen implementiert werden können. Trotzdem hängen die Möglichkeiten von Umfang und Form der gelieferten Rohdaten sowie den weiteren Fähigkeiten der angebotenen Schnittstelle ab.

Das Interface ist nicht sehr umfangreich, da das Client-Modul, um eine einfache Integration in Anwendungen zu gewährleisten, ebenfalls nur eine sehr eingeschränkte Schnittstelle zur jeweiligen Applikation besitzt. Abbildung 5.2 zeigt ein vereinfachtes Klassendiagramm, das alle für die Zusammenarbeit der Module wesentlichen Funktionen beinhaltet.

Den Großteil der Kommunikation stellt die Bereitstellung der Rohdaten durch die Applikation an das Client-Modul dar. Dazu werden eine oder mehrere TraceSources genutzt, die in eine entsprechende Collection (1) eingefügt werden müssen. Das Client Modul leitet diese Daten an alle Agenten weiter, indem der jeweilige TraceListener (2) mit allen TraceSources verknüpft wird. Die einzige weitere Schnittstelle zwischen Agenten und Client-Modul besteht in der AgentAPIConnection (3), die zum Senden der Daten an den Server verwendet wird. Die UI-Interaktionen stellen den letzten Teil der Schnittstelle da. Zu diesem Zweck existieren zwei Interfaces (4), für welche die Applikation entsprechende Instanzen bereitstellen muss.

Wie bereits im Klassendiagramm ersichtlich, müssen Agenten das Interface `IAgent` implementieren. Bei der Initialisierung wird eine Instanz der `IAgentAPIConnection` bereitgestellt, mit der der Agent arbeiten kann. Da das allgemeine Vorgehen, insbesondere in Bezug auf die grundlegende Arbeit mit dem TraceListener und der API, für alle Agenten relativ ähnlich aussehen sollte, existiert als Grundlage die abstrakte Klasse `BaseAgent`. Von dieser gibt es bereits drei weitere, ebenfalls abstrakte, Ableitungen, die die verschiedenen Schreibstrategien (siehe Abschnitt 4.4.2) umsetzen. Alle Agenten sollten daher von einer dieser Klassen erben, um dupliziertes Verhalten zu vermeiden.

Beim Programmstart beginnt die Initialisierung des Client-Moduls, sofern die Nutzungsdatenerfassung aktiviert ist. Im Falle des ersten Starts erscheint stattdessen ein Dialog, der den Nutzer auffordert, sich explizit für oder gegen die Erfassung zu entscheiden. Grundsätzlich wird vor dem Start der einzelnen Agenten geprüft, ob diese aktuell sind und ob sich die Liste der für den Client verfügbaren Agenten geändert hat. Danach werden eventuell notwendige Updates durchgeführt. Sobald dies abgeschlossen ist, werden die TraceListener der Agenten in die TraceSource(s) eingehängt und die Start-Prozeduren der Agenten aufgerufen.

Abbildung 5.2: Vereinfachtes Klassendiagramm des Client-Moduls

Bei Programmende oder Abschaltung der Nutzungsdatenerfassung werden analog alle TraceListener deaktiviert und für die Agenten die jeweilige Stop-Funktion aufgerufen. Sofern notwendig, wird das Programmende verzögert, bis alle Operationen abgeschlossen wurden. Dies ist notwendig, um Datenverlust und inkonsistente Zustände zu vermeiden. Üblicherweise dient die Stop-Funktion der Agenten dazu, übrige gepufferte Daten an den Server zu senden.

5.4.2 Server

Eine übliche Vorgehensweise bei der Implementierung des Servers wäre, alle Agenten als Instanzen einer allgemeinen Klasse umzusetzen, wodurch sie gemeinsam in

einer Tabelle der Datenbank gespeichert werden. Da jedoch jeder Agent eine eigene Tabelle zur Speicherung seiner Daten besitzt, ist dies problematisch. Zwar könnte bei jedem Agenten jeweils ein Verweis auf die Tabelle hinterlegt werden, allerdings steht eine solche Herangehensweise mit Prinzipien der Objektorientierten Programmierung (OOP) sowie im speziellen den Best Practices von Ruby on Rails im Konflikt.

Ruby on Rails verwendet standardmäßig "Active Record" als *ORM*[17]. Das heißt, für jede Tabelle in der Datenbank gibt es eine eigene Klasse (ein *Model*), die von einer entsprechenden Basisklasse (`ActiveRecord::Base`) erbt und alle Spalten als Attribute besitzt. Ein Datensatz bzw. Objekt in der Tabelle ist daher eine Instanz der jeweiligen Klasse, deren Attribute entsprechend der Werte in der Datenbank gefüllt sind.

Das Ziel ist daher, für die Daten jedes Agenten ein entsprechendes Model zu besitzen, welches die zugehörige Tabelle abbildet. Die Schwierigkeit besteht darin, dass die Agenten erst zur Laufzeit erzeugt werden und die entsprechenden Objekte daher nicht vorab implementiert werden können. An dieser Stelle kommt Metaprogrammierung zum Einsatz. Das System wird dadurch in die Lage versetzt, für neue Agenten entsprechende Klassen und Tabellen anzulegen, die daraufhin regulär zum Einsatz kommen. Durch die Verwendung von Ruby gestaltet sich dies verhältnismäßig einfach, da es eine sogenannte Skriptsprache ist und der Code daher nicht compiliert, sondern direkt interpretiert wird. Es ist somit lediglich nötig, eine Datei mit dem Code für den Agenten anzulegen und zu laden, sowie die entsprechende Tabelle in der Datenbank zu erzeugen.

```
1 def create_table
2   ActiveRecord::Migration.create_table table_name do |t|
3     capture_columns.each do |cc|
4       t.column cc.column_name, cc.column_datatype
5     end
6   end
7 end
```

Quelltext 5.2: Erzeugung der Tabelle für die Daten eines Agenten

Der nötige Code für die Erstellung ist nicht sehr umfangreich. Wie in Quelltext 5.2 ersichtlich, benötigt das Erzeugen der Tabelle lediglich die vorhandene Migrationsfunktion von ActiveRecord sowie eine einfache Iteration über die gewünschten Spalten. Auch das Model ist sehr kompakt (siehe Quelltext 5.3), da es nur dem Mapping der Tabelle auf das Objekt dient. Durch das Ableiten von `ActiveRecord::Base` sowie dem Setzen der entsprechenden Tabelle geschieht das automatisch. Alle weiteren benötigten Funktionen werden durch die Basisklasse `Capture` bereitgestellt.

Eine Besonderheit ist allerdings, dass die einzelnen Models nicht direkt von der Klasse erben, sondern lediglich durch das Gem "ActiveRecord::ActsAs"[18] verknüpft wer-

[17] Object-Relational Mapping, in diesem Fall ein Framework für selbiges.
[18] https://github.com/krautcomputing/active_record-acts_as

```ruby
def create_model
  File.open(model_filename, 'w+') do |f|
    put_notice f    # remark that this file shouldn't be changed manually
    f.puts 'module Capturing'
    f.puts " class #{capture_class_name} < ActiveRecord::Base"
    f.puts '   acts_as :capture'
    f.puts "   self.table_name = '#{table_name}'"
    f.puts ' end'
    f.puts 'end'
  end
end
```

Quelltext 5.3: Erzeugung des Models für die Daten eines Agenten

den. Der Grund dafür ist, dass ActiveRecord keine multi-table-inheritance (MTI)[19] unterstützt. Da alle Models für die von Agenten gesammelten Daten von der `Capture`-Klasse erben sollen, ihre Daten jedoch in eigenen Tabellen gespeichert werden, ist MTI allerdings unerlässlich. Über eine Instanz vom Typ `Capture` kann mit Hilfe von ActsAs durch den Aufruf `.specific` das entsprechende spezifische Model bezogen wurden, umgekehrt ist dasselbe durch `.acting_as` möglich.

```ruby
def create_aa_resource
  File.open(aa_resource_filename, 'w+') do |f|
    put_notice f    # remark that this file shouldn't be changed manually
    f.puts "ActiveAdmin.register Capturing::#{capture_class_name}, as:
        '#{self.name}' do"
    f.puts ' actions :show, :index, :destroy'
    f.puts ' menu parent: \'Captures\''
    f.puts ''
    f.puts ' index do'
    f.puts '   selectable_column'
    f.puts '   column \'Main Id\', :capture_id'
    f.puts '   id_column'
    f.puts '   column :client'
    capture_columns.each do |cc|
      f.puts "   column :#{cc.column_name}"
    end
    f.puts '   actions'
    f.puts ' end'
    f.puts ''
    capture_columns.each do |cc|
      f.puts " filter :#{cc.column_name}"
    end
    f.puts 'end'
  end
end
```

Quelltext 5.4: Erzeugung der ActiveAdmin-Ressource für die Daten eines Agenten

[19] Abbildung von Vererbung in der Datenbank, bei der eine Tabelle für die Basisklasse sowie je eine Tabelle für jede Ableitung exisitert. Im Gegensatz dazu steht die single-table-inheritance, bei der alle Daten in einer gemeinsamen Tabelle gespeichert werden.

Die Verfügbarkeit von individuellen Modellen für alle Agenten bringt auch Vorteile für das Frontend mit sich. So lassen sich die Daten neuer Agenten durch wenige Zeilen Code, die ebenfalls automatisch erzeugt werden, mit entsprechenden Seiten in ActiveAdmin versehen (siehe Quelltext 5.4), wodurch der Zugriff auf die gesammelten Daten auch im Frontend gewährleistet wird. Dies ist ein weiterer Grund, der für den unkonventionellen Einsatz von ActiveAdmin für das Frontend spricht.

5.5 UsageDataCollector und UDC Webplattform

Das entwickelte System wird UsageDataCollector (UDC) genannt, wobei dieser Begriff allerdings hauptsächlich das Client-Modul umfasst, und die Serverkomponente als UDC Webplattform bezeichnet wird. Beide sind als Prototypen zu betrachten und daher noch nicht veröffentlicht, sondern lediglich zu Testzwecken im Einsatz. Das Client-Modul wurde in eine grundlegende Prototyp-Applikation sowie eine gesonderte Version des SelectLine CRM integriert. Die Webplattform läuft aktuell auf einem internen Server. Weiterhin wurden bereits einige Agenten zur Sammlung grundlegender Daten implementiert.

5.5.1 UDC im SelectLine CRM

Für den Anwender wird die Nutzungsdatenerfassung unter dem Namen ”Programm zur Verbesserung der Nutzerfreundlichkeit” präsentiert. Entsprechend der Anforderungen können die Details der Nutzungsdatenerfassung eingesehen und diese einzeln oder übergreifend deaktiviert werden. Das UI dafür, dargestellt in Abbildung 5.3, entspricht im Groben dem Mockup aus dem Konzept (siehe Abbildung 4.1), ist allerdings an den Stil der Applikation angeglichen. Weiterhin wurde der Hinweistext angepasst und erweitert. Außer diesen Einstellungen existiert lediglich der Hinweisdialog, der beim ersten Start des Programmes abfragt, ob die Erfassung eingeschaltet werden soll (siehe Abbildung 5.4). Alles weitere läuft für den Anwender unbemerkt im Hintergrund.

5.5.2 UDC Webplattform

Die Webplattform stellt die zentrale Instanz zur Verwaltung der Agenten und der von ihnen gesammelten Daten dar. Der Zugriff geschieht, eingeschränkt durch eine Nutzerverwaltung, über einen Webbrowser und die Adresse des Servers. Es existiert ein Dashboard mit einer Auswahl der wichtigsten Daten (siehe Abbildung A.1 im Anhang), um einen schnellen Überblick zu bieten. Je nach Berechtigungen des angemeldeten Nutzers besteht weiterhin Zugriff auf die Agenten, verknüpfte Clients, erfasste Daten und auch die Nutzerverwaltung. Für jeden dieser Punkte gibt es jeweils eine Listenansicht (siehe Anhang, Abbildung A.2 und Abbildung A.3), die sortiert und gefiltert werden kann, sowie Detailansichten für die einzelnen Elemente (siehe Anhang, Abbildung A.4). Aus der Listenansicht lassen sich außerdem alle Daten, die auf den aktuellen Filter passen, als CSV oder JSON exportieren.

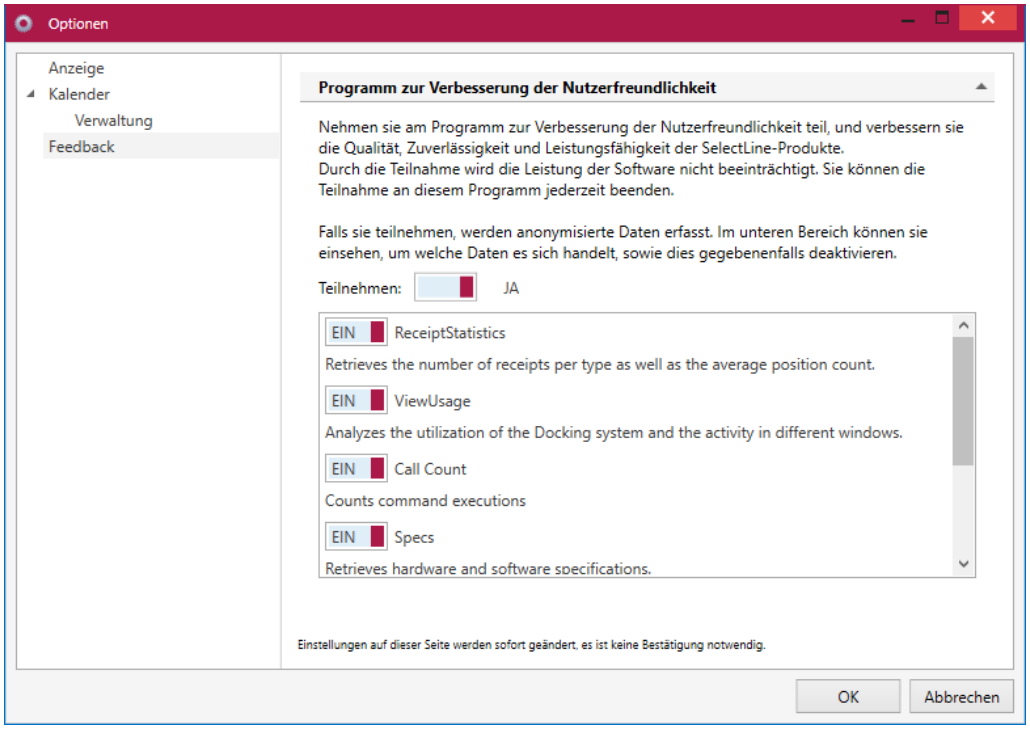

Abbildung 5.3: Optionsseite für die Einstellungen der Nutzungsdatenerfassung

5.5.3 Agenten

Bisher wurden vier Agenten entwickelt, die permanent zum Einsatz kommen sollen. Diese dienen zur Erfassung von grundlegenden Daten in Bezug auf die Nutzung des Programmes.

”Specs” Agent

Der ”Specs” Agent erfasst relevante Details zur eingesetzten Hard- und Software des PCs. Dazu gehören das Betriebssystem, der Prozessor, der Arbeitsspeicher, die Bildschirmauflösung, der verfügbare Speicherplatz auf den Laufwerken und ob bzw. welche Version von Microsoft Outlook installiert ist. Um diese Informationen zu erhalten, kommt die *Windows Management Infrastructure*(WMI) bzw. *Management Instrumentation* (MI) zum Einsatz. Quelltext A.3 im Anhang zeigt beispielhaft die resultierenden Daten für einen Client, die im JSON-Format aus dem Webinterface exportiert wurden.
In Bezug auf die erläuterten Schreibstrategien der Agenten lässt sich der ”Specs” Agent in die Kategorie Update einordnen, da die Daten bei jedem Start des Programms durch den Agenten überprüft und gegebenenfalls aktualisiert werden.

”TableSize” Agent

Der ”TableSize” Agent dient dazu, die Größe des Datenbestandes beim Kunden zu ermitteln. Dazu werden von allen Tabellen in den relevanten Datenbanken die Anzahl der Datensätze sowie der verwendete Speicherplatz erfasst. Um diese Daten zu

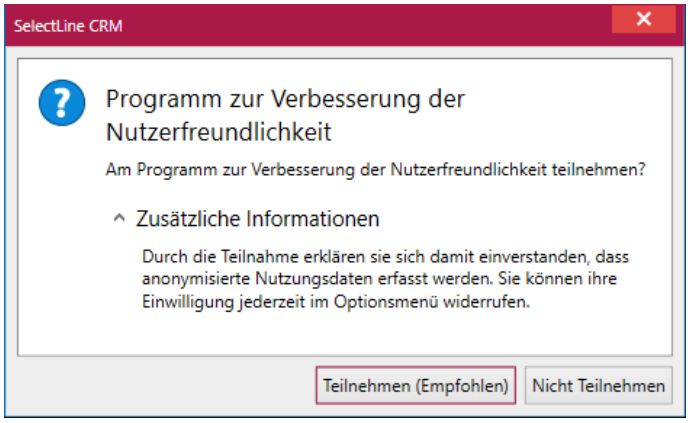

Abbildung 5.4: Hinweisdialog bei erstem Start

ermitteln, wird mittels SQL auf die Systemtabellen der jeweiligen Datenbank zugegriffen. Das entsprechende Skript findet sich in Quelltext A.2 im Anhang.
Auch dieser Agent arbeitet mit der Update Strategie, da die existierenden Datensätze jeweils nur aktualisiert werden. Um den Datenschutz zu gewährleisten, werden die Datenbanknamen durch einen Hash ersetzt. Dadurch wird eine Zuordnung der Tabellen zu unterschiedlichen Datenbanken ermöglicht, ohne dabei deren Namen preiszugeben.

”ReceiptStatistics” Agent

Der ”ReceiptStatistics” Agent stellt eine Spezialisierung des ”TableSize” Agenten dar und liefert genauere Informationen zum Inhalt der Beleg-Tabellen. Da alle Belege gemeinsam in einer Tabelle gespeichert werden, lässt sich allein durch die Größe der Tabellen nicht ermitteln, welche Belegtypen wie viele Datensätze besitzen. Es muss daher eine separate SQL-Abfrage durchgeführt werden, die nach Belegtypen gruppiert und dementsprechend die Anzahl der Belege je Typ sowie die durchschnittliche Positionsanzahl bestimmen kann.
Auf der Serverseite existiert eine eigene Tabelle für die resultierenden Daten, wobei für das Schreiben ebenfalls die Update Strategie eingesetzt wird.

”CallCount” Agent

Der ”CallCount” Agent zählt die Anzahl der Aufrufe aller Funktionen, die direkt durch den Nutzer angestoßen werden. Damit lässt sich eine Übersicht der Häufigkeit der Nutzung aller Programmfunktionen erstellen. Ein Einblick in die Art des Zugriffs (Toolbar, Menü, Hotkey, ...) ist aktuell noch nicht möglich, wäre allerdings vorstellbar, wenn noch mehr Aufwand investiert wird.
Bei der Kommunikation mit der API kommt die Increment Schreibstrategie zum Einsatz, da bestehende Werte erhalten bleiben und lediglich durch die Anzahl erneuter Aufrufe ergänzt werden.

6. Ergebnisse

Das entwickelte System ist lauffähig und wurde mit einigen Agenten bereits als Prototyp getestet. Abschließend soll nun beispielhaft der Entwicklungsprozess eines komplexeren Agenten betrachtet sowie die Möglichkeiten der Auswertung der gesammelten Daten beleuchtet werden.

6.1 Fallbeispiel

Die bereits vorgestellten Agenten liefern allgemeine Daten, die einfach erfasst und kaum vorverarbeitet werden müssen. Das System ist jedoch so konzipiert, dass auch kompliziertere Anwendungsfälle möglich werden. An dieser Stelle soll daher exemplarisch der Aufbau und die Arbeitsweise eines Agenten präsentiert werden, der die Rohdaten live auswertet und die resultierenden Informationen an den Server sendet.

6.1.1 Ziel

Aufgabe des Agenten soll es sein, Daten zum Nutzungsverhalten in Bezug auf Fenster zu ermitteln. Dabei ist von Interesse, welche Elemente wie oft pro Sitzung geöffnet werden und wie lange sie durchschnittlich aktiv sind. Die resultierenden Daten sollen Informationen darüber liefern, welche Funktionen des Programms häufig bzw. lange in Verwendung sind.

6.1.2 Vorgehensweise

Die Grundlage zur Ermittlung der Informationen stellen die Rohdaten, d.h. die durch Aspektorientierte Programmierung gewonnen Informationen über Funktionsaufrufe, dar. Um den Zeitraum zu bestimmen, in dem ein Fenster geöffnet ist, müssen die Zeitpunkte der Erstellung und des Schließens der entsprechenden View ermittelt werden. Weiterhin muss erfasst werden, wann der Fokus geändert wird, damit auch

der aktive Zeitraum eines Fensters bekannt ist. Die Basisklasse für die Views enthält Funktionen, die bei den entsprechenden Aktionen ausgeführt werden. Dementsprechend muss der Agent bei allen Ableitungen dieser Basisklasse auf diese Aufrufe reagieren.

Anhand des Klassennamens lassen sich die Views identifizieren. Mehrere Instanzen der selben Klasse sind allerdings nicht voneinander zu unterscheiden, da die Rohdatenerfassung diese Informationen nicht sinnvoll bereitstellen kann. Der Agent muss daher pro Klasse die Anzahl der Aufrufe auf das Öffnen und Schließen mitzählen, um zu wissen, ob mindestens ein Fenster des selben Typs geöffnet ist. Solange dies der Fall ist, gilt die entsprechende View als geöffnet. Die aktive Zeit lässt sich durch die Differenz zwischen dem letzten Wechsel auf ein bestimmtes Fenster und dem nächsten Aufruf auf ein anderes berechnen. Wenn alle Instanzen einer View geschlossen wurden, wird ein entsprechender Datensatz mit dem Namen sowie der aktiven und geöffneten Zeit erstellt. Dabei kommt die Create Schreibstrategie zum Einsatz. Um den Fakt, dass mehrere Instanzen nicht unterschieden werden können, auszugleichen, wird außerdem die maximale Anzahl geöffneter Fenster gleichen Typs mit abgespeichert.

Um diese Arbeitsweise umzusetzen, erbt der "ViewUsage" Agent vom Basisagenten und reagiert auf alle Traces, die dem Namensschema der Funktionen des Öffnens, Schließens und der Aktivierung auf einer View entsprechen. Intern wird eine Sammlung von Aktivitäten verwaltet, die jeweils eine Art von aktuell geöffneten Fenstern repräsentieren. Der Code für den Agenten ist in Quelltext A.4 (im Anhang) zu finden. Quelltext A.5 zeigt die Klasse für die Aktivitäten. Beide sind mit Kommentaren zum besseren Verständnis versehen.

6.1.3 Resultat

Die gesammelten Daten liegen in Form von Datensätzen vor, die jeweils den Zeitraum zwischen dem Öffnen des ersten Fensters eines Typs und dem Schließen der letzten Instanz beschreiben. Quelltext 6.1 zeigt beispielhaft zwei dieser Einträge (als JSON aus dem Webinterface exportiert). Ersichtlich ist, dass die Listenansicht der Personen (`PersonListeView`) einmal geöffnet und nach eineinhalb Minuten wieder geschlossen wurde, in dieser Zeit jedoch nur 21 Sekunden aktiv war. Die Detailansicht einer Person (`PersonView`) war hingegen zeitweise für 5 Datensätze gleichzeitig geöffnet, wobei der Fokus für einen großen Teil der Zeit (79 von insgesamt 93 Sekunden) auf einem dieser Fenster lag. Daraus lässt sich schlussfolgern, dass der Nutzer die Listenansicht während der Arbeit mit einzelnen Datensätzen geöffnet lässt, sowie mehrere Detailansichten parallel bzw. abwechselnd bearbeitet, anstatt diese sofort zu schließen.

Ein Problem der aktuellen Umsetzung liegt darin, dass Pausenzeiten in der Erfassung nicht berücksichtigt werden. Es ist unwahrscheinlich, dass ein Anwender vor seiner Pause alle offenen Fenster schließt. Daher ist davon auszugehen, dass die Zeiten verfälscht werden. Eine mögliche Maßnahme wäre, durch die Rohdatenerfassung eine regelmäßige Aktivitätsbenachrichtigung zu senden. Sollte innerhalb eines gewissen Zeitraums keine Interaktion durch den Nutzer stattfinden, oder beispielsweise

```
[{
  "id":2,
  "client_id":7,
  "created_at":"2017-08-24T06:49:50.000Z",
  "updated_at":"2017-08-24T06:49:50.000Z",
  "agent_id":15,
  "active":79,
  "maxinstancecount":5,
  "open":93,
  "view":"SelectLine.Erp.Crm.Wpf.Personen.Views.PersonView"
},
{
  "id":1,
  ...
  "active":21,
  "maxinstancecount":1,
  "open":98,
  "view":"SelectLine.Erp.Crm.Wpf.Personen.Views.PersonListeView"
}]
```

Quelltext 6.1: JSON Export von Beispieldaten für den "ViewUsage" Agent. Redundante Abschnitte wurden gekürzt.

der Rechner gesperrt sein, würde diese Nachricht ausbleiben. Der Agent könnte in Folge dessen den entsprechenden Zeitraum aus der Berechnung ausschließen, und dadurch die Daten repräsentativ halten. Damit könnte auch der Fall, dass der Nutzer parallel ein anderes Programm verwendet, abgedeckt werden.

6.2 Ausblick: Auswertung der Daten

Grundsätzlich sollen die erfassten Daten der Unterstützung des Planungsprozesses dienen. Dies soll in einer ähnlichen Form stattfinden, wie es bereits durch die bisherigen Maßnahmen zur Sammlung von Feedback geschieht. Dafür müssen die Daten in eine verwertbare Form gebracht werden. Wie bereits erwähnt, kann das Frontend dazu genutzt werden, die Informationen je Agent direkt einzusehen. Damit kann sich schnell ein Überblick verschafft werden. Außerdem liefert das Dashboard die wichtigsten Informationen darüber, wie viele Daten in letzter Zeit erfasst wurden (siehe Abbildung A.1). Für eine umfassende Auswertung der Daten reicht dies jedoch nicht aus. Daher können die Daten zusätzlich als CSV oder JSON exportiert werden, sodass sie beispielsweise mit Microsoft Excel[1] weiter verarbeitet werden können.

Eine weitere Möglichkeit stellt der direkte Zugriff auf die Datenbank dar. Da bei der Implementierung ein Augenmerk darauf gelegt wurde, dass dort korrekt typisierte Informationen in sinnvoll getrennten Strukturen vorliegen, ist die Auswertung auf diesem Weg sehr mächtig. Ein Ansatz ist die Verwertung mittels sogenannter *Business Intelligence* Software, wie beispielsweise Microsoft PowerBI[2]. Abbildung 6.1 zeigt beispielhaft, wie die Erstellung eines *Reports* auf Basis der intern gesammelten

[1] Tabellenkalkulationsprogramm
[2] https://powerbi.microsoft.com/de-de/

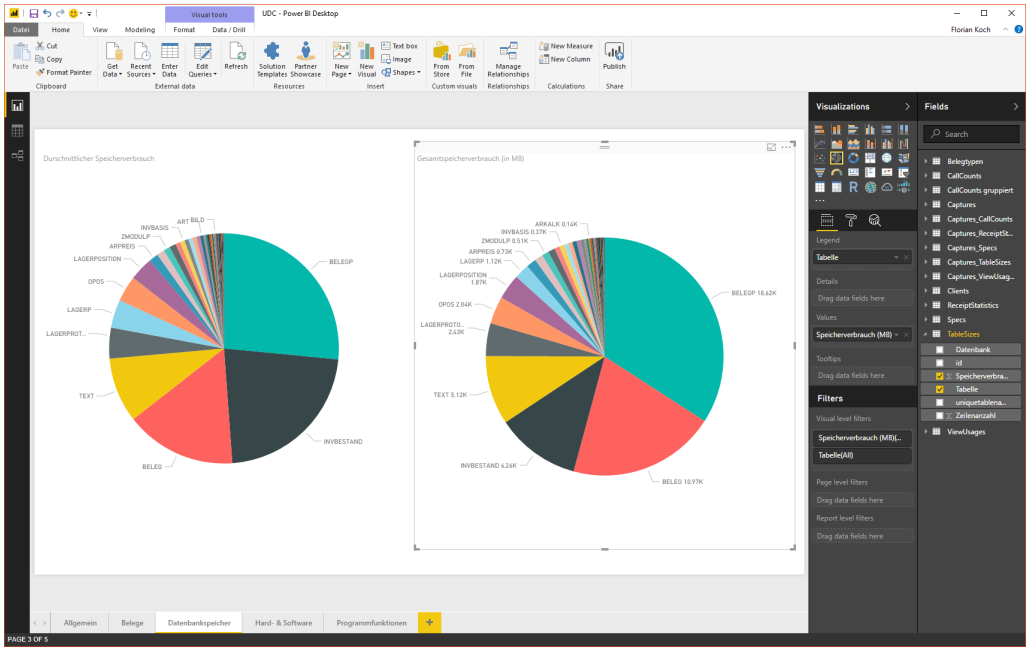

Abbildung 6.1: Arbeit mit PowerBI auf den Daten des UDC.

Testdaten aussehen könnte. Dieser setzt sich aus mehreren Seiten zusammen, die wiederum beliebige Visualisierungen enthalten. Aus solchen Reports können auch Dashboards erstellt werden, indem einzelne Elemente oder komplette Seiten übernommen werden. Alle so erstellten Auswertungen sind, sofern gewollt, auch online verfügbar.

Reports müssen nur einmalig für einen Anwendungszweck zusammengestellt werden. Da Visualisierungen nicht statisch sind, sondern auf Basis einer Datenquelle dynamisch aufgebaut werden (Abbildung A.6 im Anhang zeigt beispielhaft ein Datenmodell), sind daraufhin immer aktuelle Informationen in leicht verständlicher Form verfügbar. Ein Produktmanager hat dadurch beispielsweise jederzeit einen Überblick darüber, welche Versionen der Programme aktuell im Einsatz sind. Der Planungsprozess lässt sich somit durch die Bereitstellung von sinnvollen Statistiken unterstützen, ohne den gesamten Workflow anpassen zu müssen.

Eine weitere Möglichkeit der Verwertung der Nutzungsdaten ist das *Data Mining*. Das Ziel besteht dabei darin, durch Mustererkennung neue Erkenntnisse aus den Daten zu ziehen. Da jeder Client eindeutig identifizierbar ist (wobei allerdings kein Rückschluss auf die Identität des Nutzers möglich ist), lassen sich auch die Datensätze verschiedener Agenten miteinander in Beziehung setzen. Dadurch können beispielsweise Korrelationen zwischen verschiedenen Aspekten des Nutzungsverhaltens oder Eigenschaften des Datenbestandes ermittelt werden.

Die Durchführung von Data Mining ist ohne einen echten und ausreichend großen Datenbestand nicht sinnvoll möglich. Da diese Arbeit lediglich die Implementierung eines Prototypen sowie interne Tests umfasst, stehen solche Daten nicht zur Verfügung. Der Ansatz wurde daher vorerst nicht weiter verfolgt. In Zukunft ist allerdings nicht auszuschließen, dass Techniken des Data Minings zum Einsatz kommen

werden. In welcher Form die resultierenden Informationen in den Planungsprozess eingehen, ist allerdings schwer abschätzbar.

Sowohl die Auswertung mit Techniken der Business Intelligence, als auch durch Data Mining, liefern zwar Daten zum Nutzungsverhalten, adressieren jedoch nicht die Zielstellung, Usability-Probleme zu finden. Hierfür wird ein spezielleres Vorgehen notwendig. Ein zentrales Element ist dabei der iterative Ansatz, der durch die Agenten ermöglicht wird. Auf Basis einer Hypothese kann ein Agent entwickelt werden, der Daten in Bezug auf die gestellten Annahmen sammelt. Anhand der Ergebnisse muss entschieden werden, ob die Daten ausreichend sind, um die Hypothese zu belegen oder widerlegen. Wenn dies nicht der Fall ist, kann der Agent entsprechend verfeinert werden. Die Analyse der Daten erfordert manuellen Aufwand und muss, im Gegensatz zu den Reports, jedes Mal von Neuem durchgeführt werden. Wie genau das abläuft, hängt vom jeweiligen Anwendungsfall ab. Voraussichtlich werden jedoch Excel-Tabellen zur Verarbeitung der Daten zum Einsatz kommen.

6.3 Fazit

Ziel dieser Arbeit war die Entwicklung eines Systems, das:

- flexibel zur automatischen Erfassung verschiedenartiger Daten über die Verwendung einer Applikation genutzt werden kann,
- eine Grundlage zur Auswertung der gesammelten Daten bereitstellt,
- eine möglichst große Nutzerbasis umfasst,
- den Nutzer der Anwendung und unbeteiligte Entwickler nicht beeinträchtigt,
- bei der Erfassung ein besonderes Augenmerk auf Datenschutz legt,
- mit wenig programmiertechnischem Aufwand in die SelectLine-Programme integriert werden kann,
- unabhängig von den Updates der zugrunde liegenden Anwendung aktualisiert werden kann.

Die Implementierung von Agenten für beliebige Einsatzzwecke bietet die gewünschte Flexibilität. Gleichzeitig erfüllt das Agentensystem durch den Download zur Laufzeit die Anforderung, unabhängig von den Anwendungsupdates aktualisieren zu können. Die große Nutzerbasis ist automatisch gegeben, da die Erfassung bei allen Kunden stattfindet, die sich beim ersten Start der Anwendung für die Aktivierung der Nutzungsdatenerfassung entscheiden. Für den Kunden entsteht bis auf diese Abfrage kein weiterer Handlungszwang.

Durch die Rohdatenerfassung mittels aspektorientierter Programmierung wird zum Einen sichergestellt, dass der Aufwand der Integration in die SelectLine-Programme gering ist, und zum Anderen, dass Entwickler nicht in ihrer normalen Arbeit beeinträchtigt werden.

Die Auswertung ist über die Webplattform als zentrales Element der Datensammlung möglich. Dazu können die Informationen im Frontend direkt eingesehen werden oder zu CSV bzw. JSON exportiert werden. Alternativ ist auch Einbindung des SQL-Servers in spezialisierte Tools zur weiteren Verarbeitung, wie beispielsweise Power BI, möglich.

Für die Einhaltung des Datenschutzes wurden wichtige Grundlagen gelegt, allerdings verbleibt die endgültige Verantwortung darüber, genau wie für die Vermeidung von Beeinträchtigungen der Arbeit des Nutzers, bei den Entwicklern der Agenten. Aspekte, die in diesem Kontext zu beachten sind, wurden bereits in Abschnitt 4.3.1 erläutert.

Zusammengefasst stellt das präsentierte System der agentenbasierten Erfassung von Nutzungsdaten somit eine Lösung dar, die alle in der Problemanalyse identifizierten Anforderungen abdecken kann. Die Integration des Prototypen in das SelectLine CRM hat weiterhin bewiesen, dass dies nicht nur ein theoretischer Ansatz ist, sondern tatsächlich umgesetzt werden kann. Für den Einsatz beim Kunden ist lediglich eine komplette Testabdeckung der modifizierten Version des SelectLine CRM (sowohl durch automatische Unittests, als auch manuell durch die Testabteilung) notwendig. Sobald dies geschehen ist und das System freigegeben wurde, steht einer Auslieferung mit dem nächsten Update nichts im Wege. Zukünftig kann die Nutzungsdatenerfassung auch in weitere C#-Applikationen eingebunden werden.

Für den Einsatz in Delphi-Anwendungen ist eine Alternative für die Rohdatenerfassung erforderlich. Ob die Zeit, um eine Lösung für dieses Problem zu finden, aufgewendet werden wird, ist aktuell nicht abschätzbar. Falls die Auswertung der in den anderen Applikationen erfassten Daten erhebliche Vorteile mit sich bringt, ist eine Investition von Aufwand allerdings nicht unwahrscheinlich.

Die Entwicklung weiterer Agenten sowie die Auswertung der erfassten Daten bilden einen Prozess, der sich zuerst entwickeln und in den etablierten Workflow integrieren muss. Voraussichtlich fällt die Verarbeitung der Daten sowie die Konzeption von neuen Agenten in die Verantwortlichkeiten des Produktmanagements, wohingegen die tatsächliche Implementierung Aufgabe einiger Mitarbeiter des Entwicklungsteams sein wird. Eine komplette Spezialisierung eines Teams auf das Thema ist allerdings unwahrscheinlich.

Für die zukünftige Entwicklung des UsageDataCollectors sind einige Funktionen vorstellbar. Wie bereits erwähnt, wäre eine Möglichkeit der Zuweisung von Agenten basierend auf gewissen Bedingungen sinnvoll. Die Grundlage dafür wurde bereits gelegt, indem die Lizenzinformationen mit gespeichert werden, die als Kriterium dienen könnten. Dadurch wäre diese Weiterentwicklung möglich, ohne eine neue Programmversion auszuliefern. Weiterhin wäre eine Erweiterung des Systems um eine automatische Fehleraufzeichnung vorstellbar. Auch die Integration eines Dialogs, mit dem der Nutzer Feedback direkt aus dem Programm senden kann, würde sich anbieten. Der Vorteil gegenüber anderen Möglichkeiten liegt darin, dass dabei Informationen zu aktuell geöffneten Fenstern oder andere Zustandsinformationen des Programms mitgesendet werden könnten. Diese beiden Erweiterungen würden allerdings Änderungen an den Clients erfordern.

Sobald die erste Änderung am Client-Modul erfolgt, muss zusätzlich eine erweiterte Versionierung der Agenten auf der Serverseite implementiert werden. Hintergrund ist hier, dass neuere Agenten ausschließlich mit dem aktualisiertem Client-Modul zusammenarbeiten können, und daher nicht immer an alle Anwendungen verteilt werden dürfen. Bisher wird jedoch automatisch immer die neueste Version jedes Agenten an den Client geschickt. Außerdem werden ältere Versionen der Binärdaten nicht auf dem Server gespeichert.

A. Anhang

A.1 Codebeispiele

```
namespace SelectLine.Erp.UsageDataCollection
{
    using System;
    using System.Collections.Generic;
    using System.Diagnostics;
    using System.Reflection;

    using PostSharp.Aspects;
    using PostSharp.Serialization;

    [PSerializable]
    public class CrmUDCAspect : OnMethodBoundaryAspect
    {
        public static TraceSource TraceSource { get; set; }

        public override Boolean CompileTimeValidate(MethodBase method)
        {
            if (method.Name.StartsWith("get_"))
                return false;
            return base.CompileTimeValidate(method);
        }

        public override void OnEntry(MethodExecutionArgs args)
        {
            if (TraceSource == null)
                return;
            var traceData = new List<Object>
            {
                args.Method.DeclaringType + "." + args.Method.Name,
                args.Instance != null ? args.Instance.GetType().FullName : String.Empty
            };
            var parameters = args.Arguments?.ToArray();
            if (parameters != null && parameters.Length >= 0)
            {
                traceData.Add(parameters);
            }
            TraceSource.TraceData(TraceEventType.Verbose, 42, traceData.ToArray());
        }
    }
}
```

Quelltext A.1: Aspekt zur Nutzungsdatenerfassung im SelectLine CRM.

```
SELECT sOBJ.name AS [TableName],
SUM(sdmvPTNS.row_count) AS [RowCount],
CAST(ROUND(((SUM(sdmvPTNS.used_page_count) * 8) / 1024.00), 2) AS NUMERIC(36,
    2)) AS [UsedSpaceMB]
FROM sys.objects AS sOBJ
INNER JOIN sys.dm_db_partition_stats AS sdmvPTNS
ON sOBJ.object_id = sdmvPTNS.object_id
WHERE sOBJ.type = 'U'
AND sOBJ.is_ms_shipped = 0x0
AND sdmvPTNS.index_id < 2
GROUP BY sOBJ.schema_id,
sOBJ.name
ORDER BY [TableName]
```

Quelltext A.2: SQL zur Ermittlung der Tabellengrößen.

```
[{
  "id":6,
  "client_id":5,
  "created_at":"2017-08-10T14:31:00.000Z",
  "updated_at":"2017-08-10T14:31:00.000Z",
  "agent_id":8,
  "key":"Outlook",
  "value":"Microsoft Outlook MUI (German) 2016; "
},
{
  ...
  "key":"ScreenResolutions",
  "value":"1920x1200; 1080x1920; "
},
{
  ...
  "key":"DiskSpace",
  "value":"D: (33,3GB of 232,9GB free); C: (664,3GB of 931,4GB free); "
},
{
  ...
  "key":"Memory",
  "value":"16GB"
},
{
  ...
  "key":"Processor",
  "value":"Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (x64); "
},
{
  ...
  "key":"OS",
  "value":"Microsoft Windows 7 Enterprise : de-DE; "
}]
```

Quelltext A.3: JSON Export der Daten eines Clients für den ”Specs” Agent. Redundante Abschnitte wurden gekürzt.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using UsageDataCollector.Agents;

namespace ViewUsageAgent
{
    public class ViewUsageAgent : BaseAgent
    {
        // Sammlung der derzeit geöffneten Fenster in Form von "Aktivitäten"
        private Dictionary<String, ViewActivity> Activities = new Dictionary<String, ViewActivity>();

        private const String CreateIdentifier = "OnCreating";
        private const String ActivateIdentifier = "OnActivating";
        private const String CloseIdentifier = "OnClosing";

        protected override Boolean ShouldCapture(TraceEventCache cache, String source,
             TraceEventType eventType, Int32 id, String formatOrMessage, Object[] args, Object
             data1, Object[] data)
        {
            // überprüft auf korrektes Format der Trace-Daten
            return data != null && data.Length >= 1 && data[0] is String &&
                 IsFunctionRelevant((String)data[0]);
        }

        private Boolean IsFunctionRelevant(String name)
        {
            // prüft, ob der aufgerufene Funktionsname dem einer der drei relevanten View-Funktionen
                 entspricht
            return name.EndsWith("View." + CreateIdentifier) || name.EndsWith("View." +
                 ActivateIdentifier) || name.EndsWith("View." + CloseIdentifier);
        }

        protected override async Task<TraceListener> CreateUsageListener()
        {
            // erzeugt einen Buffer zur asynchronen Abarbeitung der Traces, für die ShouldCapture
                 true zurückgibt
            return new AsyncBufferedTraceListener<Object>(ProcessTrace);
        }

        public override async Task Stop()
        {
            // beim Beenden der Erfassung werden alle verbleibenden Aktivitäten erfasst
            foreach (var activity in Activities)
            {
                await CaptureActivity(activity.Key, activity.Value);
            }
            // und danach entfernt
            Activities.Clear();
            await base.Stop();
        }

        private async Task CaptureActivity(String view, ViewActivity activity)
        {
            // aus der Aktivität wird ein Datensatz für die Erfassung erstellt
            var data = new List<KeyValuePair<String, Object>>
            {
                new KeyValuePair<String, Object>("view", view),
                new KeyValuePair<String, Object>("open",
                     Convert.ToInt32(activity.OpenDuration.TotalSeconds)),
                new KeyValuePair<String, Object>("active",
                     Convert.ToInt32(activity.ActiveDuration.TotalSeconds)),
                new KeyValuePair<String, Object>("maxinstancecount", activity.MaxInstanceCount)
            };
            // dieser wird über die API an den Server gesendet
            await ApiConnection.CreateCapture(data.ToArray());
        }

        // wird asynchron bei der Abarbeitung der Traces aufgerufen
        private void ProcessTrace(TraceWrapper<Object> wrapper)
```

```
65          {
66              if (wrapper.LoggingObjects == null || wrapper.LoggingObjects.Length < 2)
67              {
68                  return;
69              }
70              var view = (String)wrapper.LoggingObjects[1];
71              // anhand des Funktionsnamens wird entschieden, wie weiter vorgegangen werden muss
72              switch (((String)wrapper.LoggingObjects[0]).Split('.').Last())
73              {
74                  case CreateIdentifier:
75                      CreatedView(view);
76                      break;
77                  case ActivateIdentifier:
78                      ActivatedView(view);
79                      break;
80                  case CloseIdentifier:
81                      ClosedView(view);
82                      break;
83                  default:
84                      break;
85              }
86          }
87
88          private void CreatedView(String view)
89          {
90              // wenn eine View geöffent wird, wird entsprechend eine Aktivität erzeugt bzw. eine
                    Instanz hinzugefügt, falls die Aktivität für die View bereits existiert
91              if (Activities.TryGetValue(view, out var activity))
92              {
93                  activity.AddInstance();
94              }
95              else
96              {
97                  activity = new ViewActivity();
98                  Activities.Add(view, activity);
99              }
100         }
101
102         private void ActivatedView(String view)
103         {
104             // wenn sich der Fokus ändert, wird die entsprchende Aktivität benachrichtigt
105             if (Activities.TryGetValue(view, out var activity))
106             {
107                 activity.MakeActive();
108             }
109         }
110
111         private void ClosedView(String view)
112         {
113             // wenn eine View geschlossen wird, wird eine Instanz aus der Aktivität entfernt
114             if (Activities.TryGetValue(view, out var activity))
115             {
116                 activity.RemoveInstance();
117                 // sollte die Anzahl der Instanzen auf 0 gefallen sein, ist kein Fenster des Typs
                        mehr geöffnet
118                 if (activity.CanRemove())
119                 {
120                     // die Aktivität wird daraufhin als Datensatz erfasst
121                     CaptureActivity(view, activity);
122                     // und danach gelöscht
123                     Activities.Remove(view);
124                 }
125             }
126         }
127     }
128 }
```

Quelltext A.4: ”ViewUsage” Agent

```
using System;
using System.Threading;

namespace ViewUsageAgent
{
    public class ViewActivity
    {
        public ViewActivity()
        {
            _openSince = DateTime.Now;
            _activeDurationStored = TimeSpan.Zero;
            _instanceCount = 1;
            MaxInstanceCount = 1;
        }

        // maximale Anzahl gleichzeitig geöffneter Instanzen
        public Int32 MaxInstanceCount { get; private set; }

        private readonly DateTime _openSince;

        // Gesamtzeit, in der mind. ein Fenster des Typs geöffnet war
        public TimeSpan OpenDuration => DateTime.Now.Subtract(_openSince);

        private DateTime? _activeSince;

        private TimeSpan _activeDurationStored;
        // Gesamtzeit, in der eins der Fenster des Typs fokussiert war
        public TimeSpan ActiveDuration => _activeDurationStored + (_activeSince.HasValue ?
             DateTime.Now.Subtract(_activeSince.Value) : TimeSpan.Zero);

        private static ViewActivity _activeView;

        public void MakeActive()
        {
            if (_activeView == this) return;
            if (_activeView?._activeSince != null)
            {
                // angelaufene aktive Zeit des letzten Fensters speichern
                _activeView._activeDurationStored +=
                     DateTime.Now.Subtract(_activeView._activeSince.Value);
                _activeView._activeSince = null;
            }
            _activeView = this;
            this._activeSince = DateTime.Now;
        }

        private Int32 _instanceCount;

        public void AddInstance()
        {
            _instanceCount++;
            // aktualisiert die maximale Anzahl an gleichzeitig geöffneten Fenstern des Typs
            MaxInstanceCount = Math.Max(_instanceCount, MaxInstanceCount);
        }

        public void RemoveInstance()
        {
            _instanceCount--;
        }

        public Boolean CanRemove()
        {
            return _instanceCount <= 0;
        }
    }
}
```

Quelltext A.5: `ViewActivity` Helfer-Klasse für ”ViewUsage” Agent

A.2 Abbildungen

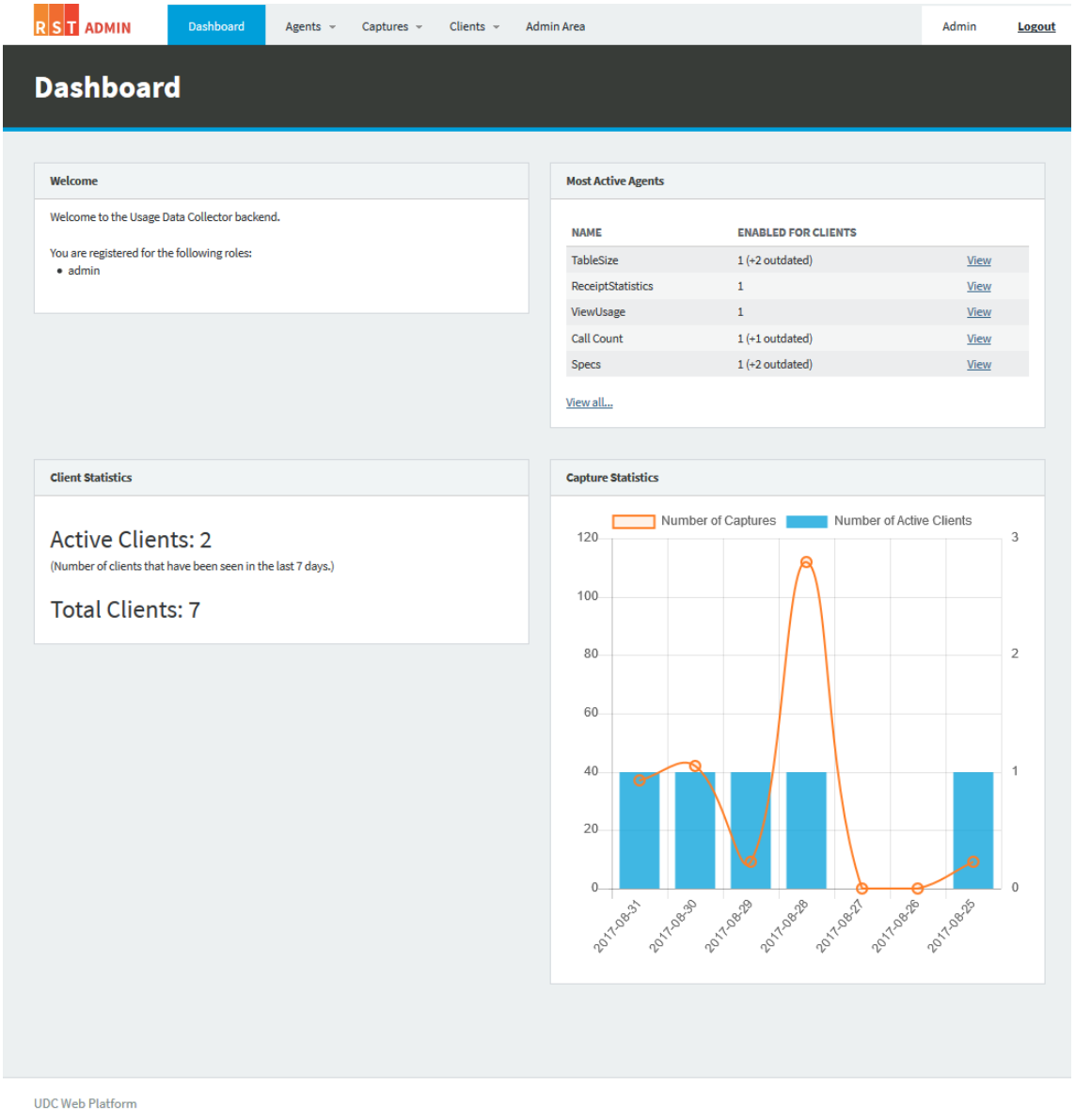

Abbildung A.1: Dashboard der Webplattform

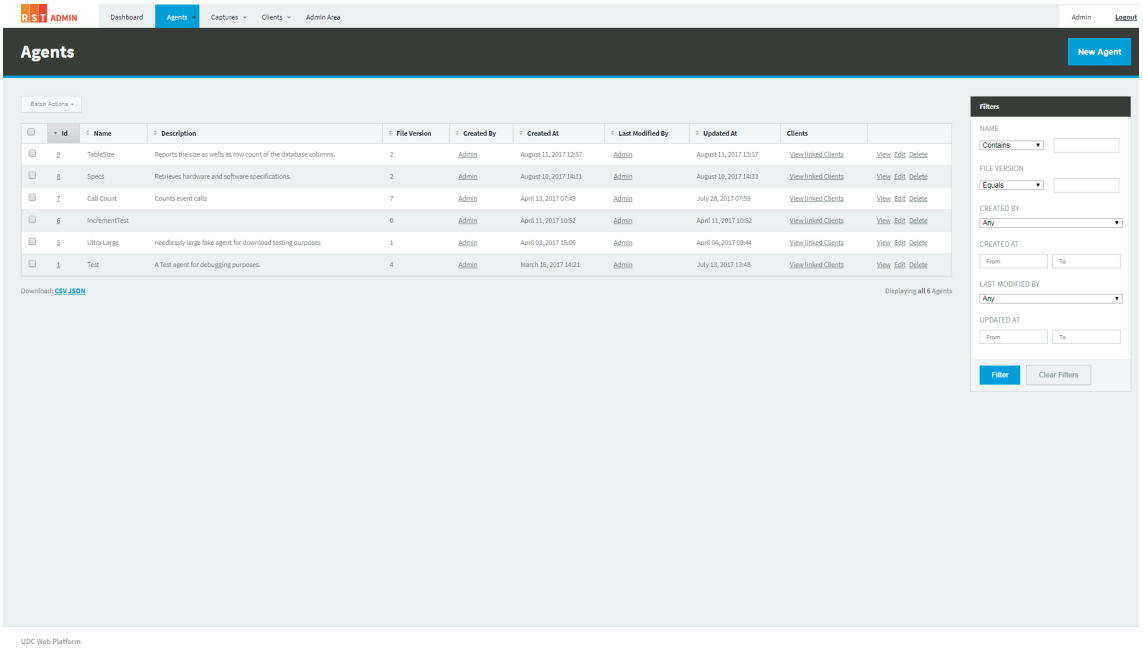

Abbildung A.2: Verwaltung der Agenten im Frontend

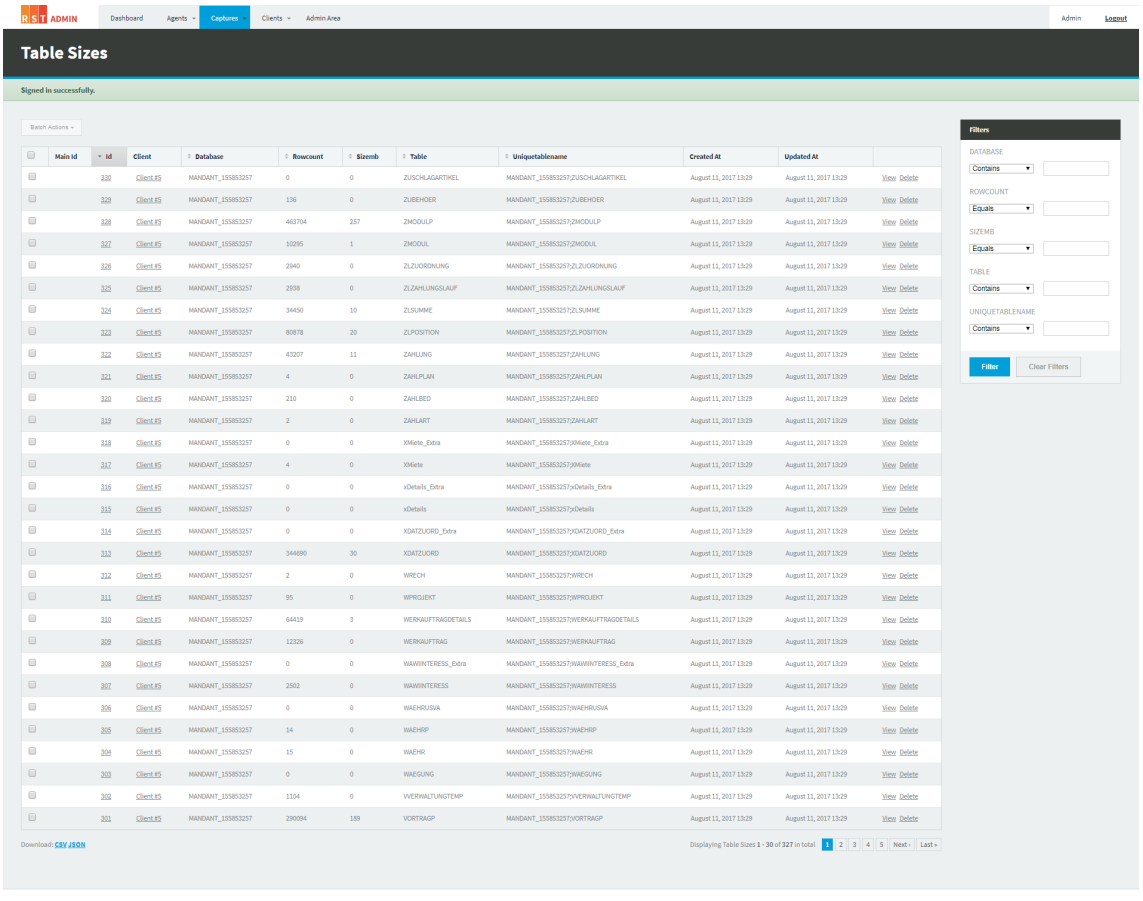

Abbildung A.3: Listenansicht der erfassten Daten für einen spezifischen Agenten

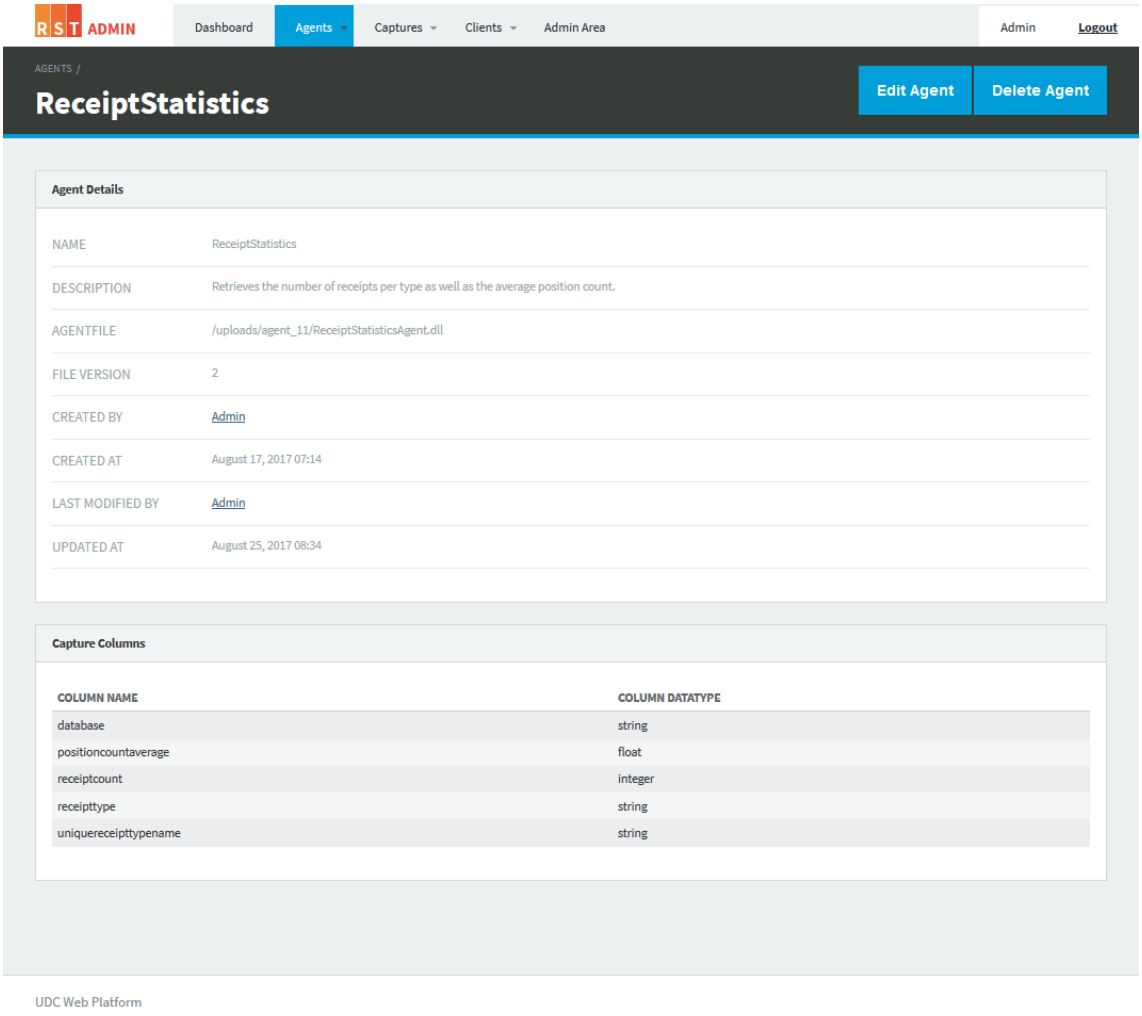

Abbildung A.4: Detailansicht für einen Agenten

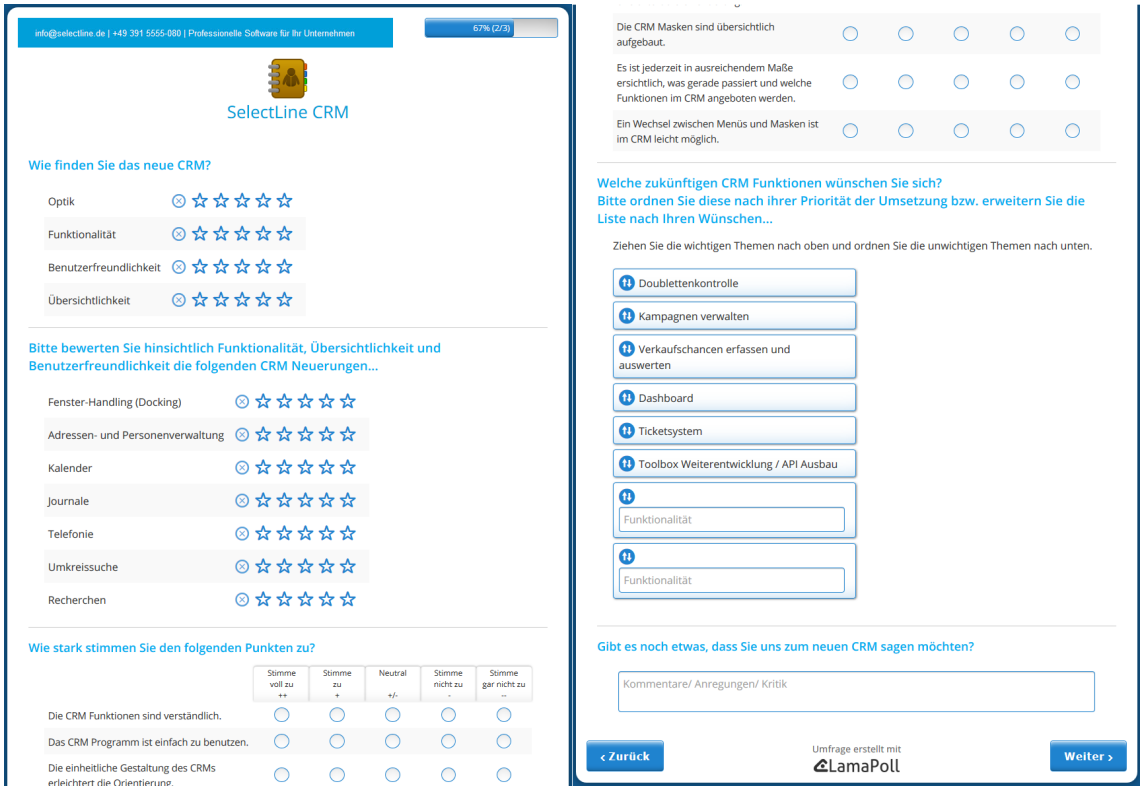

Abbildung A.5: Beta-Umfrage zur aktuellen Version, Bereich CRM

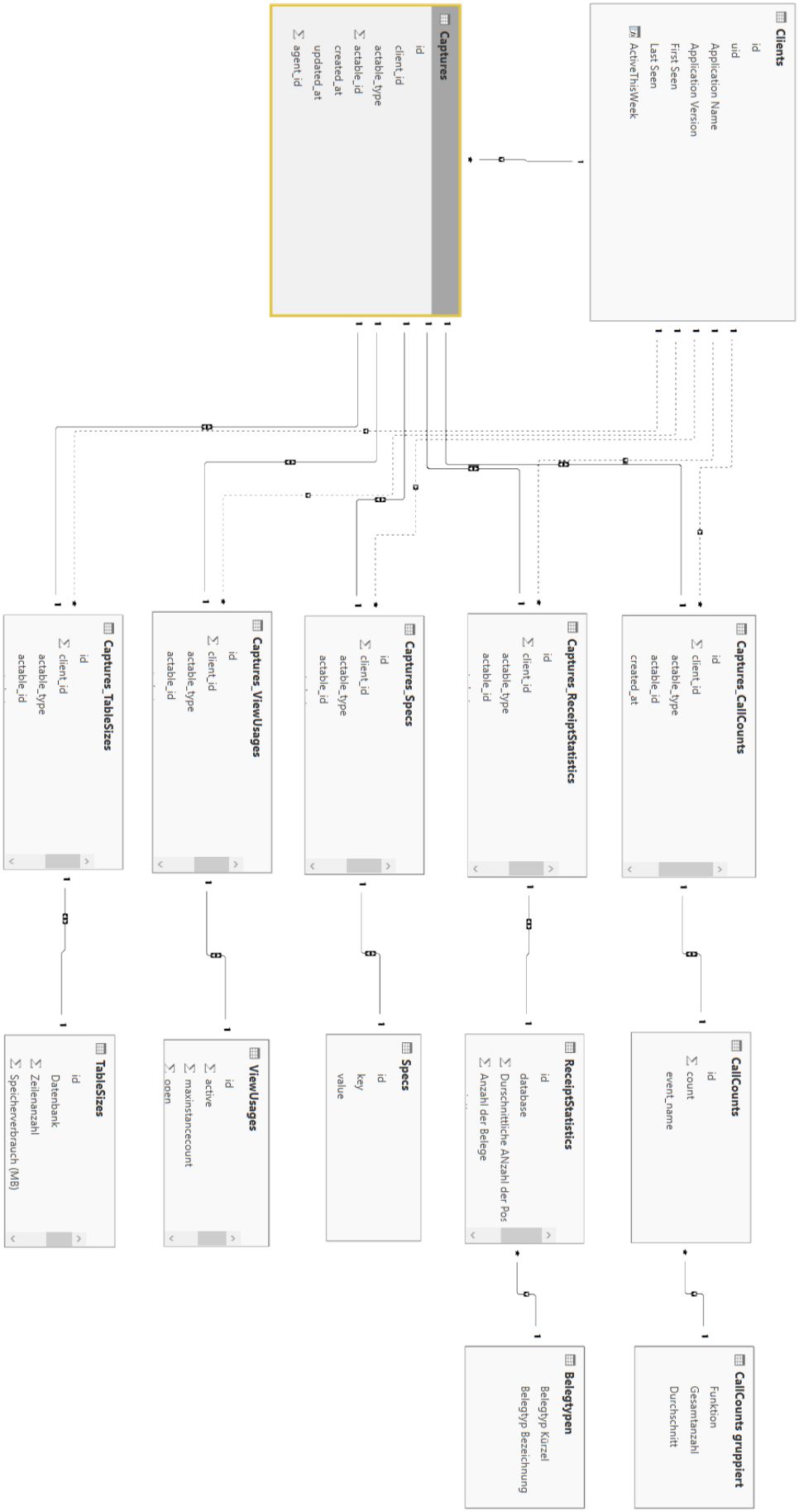

Abbildung A.6: Datenmodell eines PowerBI Reports

A.3 API Referenz

Zugriffs-Token

/access_token

- **post**: Fordert einen Zugriffs-Token vom Server an.

Agenten

/agents

- **get** *(secured)*: Ruft alle Agenten ab, die für den aktuellen Client vorgeshen sind.
- **post** *(secured)*: Speichert die Liste der Agenten und ob diese eingeschaltet sind.

/agents/{agent_id}

- **get** *(secured)*: Initiiert den Download der Binärdaten des Agenten, der die ID *{agent_id}* besitzt.

Download-Ticket

/download_ticket/{agent_id}

- **post** *(secured)*: Fordert ein Download-Ticket für den Agenten mit der ID *{agent_id}* an.

Erfassungsdatensatz

/capture/{agent_id}/create

- **post** *(secured)*: Erzeugt einen neuen Erfassungsdatensatz für den Agenten mit der ID *{agent_id}*.

/capture/{agent_id}/update

- **post** *(secured)*: ändert einen existierenden Erfassungsdatensatz für den Agenten mit der ID *{agent_id}* oder erzeugt einen neuen, wenn noch keiner existiert.

/capture/{agent_id}/increment

- **post** *(secured)*: Inkrementiert die Werte eines existierenden Erfassungsdatensatzes für den Agenten mit der ID *{agent_id}* oder erzeugt einen neuen, wenn noch keiner existiert.

Literaturverzeichnis

[AMAMZ03] Majed Al-Mashari, Abdullah Al-Mudimigh, and Mohamed Zairi. Enterprise resource planning: A taxonomy of critical factors. *European Journal of Operational Research*, 146(2):352–364, apr 2003. (zitiert auf Seite 10)

[AWS06] Richard Atterer, Monika Wnuk, and Albrecht Schmidt. Knowing the user's every move - User Activity Tracking forWebsite Usability Evaluation and Implicit Interaction. In *Proceedings of the 15th international conference on World Wide Web - WWW '06*, page 203, 2006. (zitiert auf Seite 24)

[Bev06] Nigel Bevan. International Standards for HCI. *Encyclopedia of Human Computer Interaction*, pages 362–372, 2006. (zitiert auf Seite 18)

[BTT05] David Benyon, Phil Turner, and Susan Turner. *Designing interactive systems: People, activities, contexts, technologies*. Pearson Education, 2005. (zitiert auf Seite 5)

[Cap06] Miranda G. Capra. *Usability problem description and the evaluator effect in usability testing*. PhD thesis, Virginia Polytechnic Institute and State University, 2006. (zitiert auf Seite 17)

[CC05] Lynne Cooke and Elisabeth Cuddihy. Using eye tracking to address limitations in think-aloud protocol. In *IEEE International Professional Communication Conference*, pages 653–658. IEEE, 2005. (zitiert auf Seite 16)

[Fer03] Xavier Ferre. Bridging the gaps between software engineering and human-computer interaction. *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 28–35, 2003. (zitiert auf Seite 1, 8 und 9)

[FIA11] Adrian Fernandez, Emilio Insfran, and Silvia Abrahão. Usability evaluation methods for the web: A systematic mapping study. *Information and Software Technology*, 53(8):789–817, 2011. (zitiert auf Seite 14 und 19)

[FLH12] Asbjørn Følstad, Effie Law, and Kasper Hornbæk. Analysis in practical usability evaluation. *Proceedings of the 2012 ACM annual conference*

on Human Factors in Computing Systems - CHI '12, page 2127, 2012. (zitiert auf Seite 14)

[GS98] Wayne D. Gray and Marilyn C. Salzman. Damaged Merchandise? A Review of Experiments That Compare Usability Evaluation Methods. *Human–Computer Interaction*, 13(3):203–261, 1998. (zitiert auf Seite 14)

[HAW01] H. Rex Hartson, Terence S. Andre, and Robert C. Williges. Criteria for evaluating usability evaluation methods. *International Journal of Human-Computer Interaction*, 13(4):373–410, dec 2001. (zitiert auf Seite 14)

[HMJ14] Morten Hertzum, Rolf Molich, and Niels Ebbe Jacobsen. What You Get Is What You See: Revisiting the Evaluator Effect in Usability Tests. *Behaviour & Information Technology*, pages 144–162, 2014. (zitiert auf Seite 17)

[Hol05] Andreas Holzinger. Usability Engineering Methods for Software Developers. *Commun. ACM*, 48(1):71–74, 2005. (zitiert auf Seite 18)

[HR98] David M Hilbert and David F Redmiles. An approach to large-scale collection of application usage data over the Internet. *Proceedings of the 20th international conference on Software engineering*, pages 136–145, 1998. (zitiert auf Seite 20, 21 und 22)

[HR00] David M Hilbert and David F Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys*, 32(4):384–421, 2000. (zitiert auf Seite 23)

[HR01] David M Hilbert and David F Redmiles. Large-scale collection of usage data to inform design. *IFIP Conf. on Human Computer Interaction (INTERACT)*, pages 569–576, 2001. (zitiert auf Seite 20, 21 und 23)

[HS10] Wonil Hwang and Gavriel Salvendy. Number of people required for usability evaluation. *Communications of the ACM*, 53(5):130, may 2010. (zitiert auf Seite 15 und 19)

[IH01] Melody Ivory and Marti Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516, 2001. (zitiert auf Seite 18)

[JD92] Robin Jeffries and Heather Desurvire. Usability testing vs. heuristic evaluation. *ACM SIGCHI Bulletin*, 24(4):39–41, oct 1992. (zitiert auf Seite 17)

[JK03] Robert J K Jacob and Keith S Karn. Eye Tracking in Human-Computer Interaction and Usability Research. Ready to Deliver the Promises. *Mind*, 2(3):4, 2003. (zitiert auf Seite 16)

[KLL+02] Gregor J Kiczales, John O Lamping, Cristina V Lopes, James J Hugunin, Erik A Hilsdale, and Chandrasekhar Boyapati. Aspect-oriented programming. *U.S. Patent Documents, United States Patent*, 2002. (zitiert auf Seite 29)

[KU04] Emiel Krahmer and Nicole Ummelen. Thinking About Thinking Aloud: A Comparison of TwoVerbal Protocols for Usability Testing. *IEEE Transactions on Professional Communication*, 47(2):105–117, 2004. (zitiert auf Seite 15)

[LH12] Florian Lettner and Clemens Holzmann. Automated and unsupervised user interaction logging as basis for usability evaluation of mobile applications. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia - MoMM '12*, page 118, 2012. (zitiert auf Seite 29)

[MGT+04] Rebecca Mancy, Phil Gray, Richard Thomas, Gregor Kennedy, and Steve Draper. Using database technologies to transform low-level stream data for client analysis. 2004. (zitiert auf Seite 21)

[Nie94a] Jakob Nielsen. Estimating the number of subjects needed for a thinking aloud test. *International Journal of Human-Computer Studies*, 41(3):385–397, sep 1994. (zitiert auf Seite 15 und 19)

[Nie94b] Jakob Nielsen. Usability inspection methods. In *Conference companion on Human factors in computing systems - CHI '94*, pages 413–414. ACM, 1994. (zitiert auf Seite 17)

[NL93] Jakob Nielsen and Thomas K Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '93*, pages 206–213, 1993. (zitiert auf Seite 17 und 19)

[NM90] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, pages 249–256, New York, New York, USA, 1990. ACM Press. (zitiert auf Seite 17)

[PD15] Bernhard Preim and Raimund Dachselt. *Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces*. Springer-Verlag, 2015. (zitiert auf Seite 16)

[Pey16] Sebastián E Peyrott. The JWT Handbook. Technical report, 2016. (zitiert auf Seite 40 und 41)

[PP12] Helen Petrie and Christopher Power. What do users really care about?: a comparison of usability problems found by users and experts on highly interactive websites. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 2107–2116, 2012. (zitiert auf Seite 18)

[RC09] Mary Beth Rosson and John M Carroll. Scenario based design. *Human Computer Interaction. Boca Raton*, pages 145–162, 2009. (zitiert auf Seite 5)

[RFR95] John Rieman, Marita Franzke, and David F Redmiles. Usability Evaluation with the Cognitive Walkthrough. *Conference companion on Human factors in computing systems*, pages 387–388, 1995. (zitiert auf Seite 18)

[RG04] Karen Renaud and Phil Gray. Making Sense of Low - Level Usage Data to Understand User Activities. *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, 2004. (zitiert auf Seite 23)

[SCRF16] Andrés Solano, César A Collazos, Cristian Rusu, and Habib M Fardoun. Combinations of methods for collaborative evaluation of the usability of interactive software systems. *Advances in Human-Computer Interaction*, 2016, 2016. (zitiert auf Seite 14)

[SK10] Osama Sohaib and Khalid Khan. Integrating usability engineering and agile software development: A literature review. In *2010 International Conference On Computer Design and Applications*, volume 2, pages V2–32–V2–38, 2010. (zitiert auf Seite 1)

[SM86] Sidney L Smith and Jane N Mosier. Guidelines for Designing User Interface Software. *Guidelines for Designing User Interface Software*, ESD-TR-86-(ESD-TR-86-278), 1986. (zitiert auf Seite 17)

[SPC+16] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016. (zitiert auf Seite 17)

[TMSK10] Publisher Taylor, Thomas Mahatody, Mouldi Sagar, and Christophe Kolski. State of the Art on the Cognitive Walkthrough Method, Its Variants and Evolutions. *International Journal of Human-Computer Interaction*, 26(924789933):741–785, 2010. (zitiert auf Seite 18)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 09. Oktober 2017