Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht Home-Office2.0

Matthias Holsten, Benjamin Kirstgen, Karsten Panier

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Matthias Holsten, Benjamin Kirstgen,
Karsten Panier

Projektbericht Home-Office2.0

LuPanKu Projekt - Second Milestone

Projektbericht eingereicht im Rahmen von PR2
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor: Prof. Dr. Kai von Luck

Abgegeben am 27. Februar 2011

Inhaltsverzeichnis

1 Einleitung

Das Szenario *Home-Office2.0* beschreibt Entwicklerteams, die zeitweise über mehrere Standorte verteilt arbeiten. Diese Projekt-Struktur erschwert die Kommunikation und Kollaboration. Dadurch kann es zu Vertrauenseinbußen zwischen den Teammitgliedern kommen und den Projekterfolg gefährden. [10]

Eine Möglichkeit diesen Problemen zu begegnen ist die Verwendung von *Enterprise2.0*-Technologien zur Unterstützung der Zusammenarbeit.[34] Im Bereich der Softwareentwicklung existieren hierfür lediglich Insellösungen. Deshalb wurde das `LuPanKu`-System entworfen, um eine bessere Integration in den Entwicklungsprozess zu ermöglichen und dadurch die Kollaboration zu verbessern. [33]

Diese Arbeit beschreibt die Weiterentwicklung des `LuPanKu`-Systems. Zu Beginn dieses zweiten Releases wurde mit einer Bewertung des bisherigen Vorgehens und den aktuell erreichten Zielen begonnen. Daraus wurde der Ansatz entwickelt die zu erreichenden Szenarien zu konkretisieren, um daraus neue Anforderungen abzuleiten (siehe Kapitel 2).

Zur Realisierung dieser Anforderungen wurde eine Restrukturierung der Architektur notwendig um neue Technologien integrieren zu können (siehe Kapitel 3).

Diese Technologien und die Kriterien auf deren Basis sie ausgewählt wurden, wird in Kapitel 4 erläutert. Im Anschluss wird die Integration der Technologien in das bestehende `LuPanKu`-System im Kapitel 5 dargelegt. Die während der Weiterentwicklung entstandenen Erfahrungen werden im Kapitel 6 veranschaulicht. Zum Abschluss werden im Kapitel 7 die nächsten Schritte des Projektteams und die geplante Weiterentwicklung des `LuPanKu`-Systems skizziert.

2 Masterthemen

Zu Beginn der Veranstaltung Projekt 2 wurden die offenen Anforderungen an das *LuPan-Ku*-System überarbeitet und in Hinblick auf die sich konkretisierenden Masterarbeiten neu bewertet. So wurden unter anderem Eclipse[16]-basierte Entwicklungsumgebungen als zentrale Bausteine identifiziert. Aus diesen Vorarbeiten wurden die Szenarien der Masterthemen entwickelt, die nachfolgend vorgestellt werden.

2.1 Masterarbeit Expertensuche

Die Masterarbeit *Expertensuche* setzt sich zum Ziel, zu einer Fragestellung den bestmöglichen Ansprechpartner zu finden. Im Gegensatz zu einer Dokumentation bietet ein Ansprechpartner die Möglichkeit, gezielt auf die Bedürfnisse des Fragestellers einzugehen und dessen Wissensstand zu berücksichtigen. Ferner können in der direkten Kommunikation Informationen effektiver ausgetauscht werden.

Die Expertensuche ist damit ein Ansatz, die Informationssuche weiter zu entwickeln. Durch die vermehrte Nutzung von Web-basierten Informationssystemen (Wiki, Blogs, Foren und klassischen Webseiten) ist eine große Menge an Informationen vorhanden. Dies betrifft zum einen das Internet aber auch die Systeme in Unternehmen. Nutzer haben daher das Problem, die für sie relevanten Informationen aus der Flut herauszufiltern.

Dieses wird sowohl durch eine Studie[29] von 2006 über Suchverhalten von Internetnutzern, als auch Informationen, die der Provider AOL ins Internet stellte und die von Mario Fischer analysiert wurden, belegt. Danach wird das erste Suchergebnis bei einer Suchanfrage zu ca 45% angeklickt und nur noch 10% der Suchenden blättern auf die zweite Seite der Ergebnisse[14]. Neben dem Finden von Informationen gibt es weitere Konsequenzen, denn die Menge der verfügbaren Informationen ist von einem Menschen nicht mehr zu verarbeiten. Eine ausschließlich oberflächliche Betrachtungsweise der Informationen ist die Folge. Die Überforderung des Gehirns macht eine klare Differenzierung und Strukturierung der Daten laut David Kirsh sehr kompliziert[27]. Kirsh ergründet des Weiteren die Qualität der Informationen als Problem für die Überforderung des Menschen:

> „The second source of cognitive overload, then, stems from our effort to cope with the uncertain quality and relevance of our information supply. There is more search and less satisfaction.“[27]

Der Ansatz der Masterarbeit ist die Verknüpfung der klassischen Dokumenten-basierten Suche mit den Autoren und ihren sozialen Strukturen. Damit rückt der Mensch, in der

Rolle des Autors oder Lesers, in den Vordergrund und das Dokument in den Hintergrund. Für die Identifikation eines Experten wird dabei auf zwei Säulen gebaut. Zum einen werden die Artefakte eines Wissensarbeiters untersucht und passend zu dem Suchbegriff die Kompetenz des Autoren gemessen. Durch das Sichtbarmachen des sozialen Netzwerkes wird der Anreiz geschaffen, sich besonders in die Gemeinschaft einzubringen und seinen Beitrag am sozialen Miteinander zu leisen. Durch das aktive Bewerten der Beiträge eines Benutzers wird der Ansporn qualitativ hochwertige Informationen zu liefern geschaffen und so die Qualität gesteigert.

Mit der Erhebung der Daten über die Nutzung der sozialen Informationen und Bewertung der personalisierten Beiträge im sozialen Netzwerk lassen sich Wissensprofile erstellen und die Benutzer einem Themen-basierten Lernlevel zuordnen. Die Lernlevel geben darüber Auskunft, wie weit ein Autor sich in einen Themenbereich eingearbeitet hat[8].

Durch die Gliederung in Lernlevel, kann einem Fragesteller der für ihn passende Ansprechpartner vorgeschlagen werden. Bei einer großen Abweichung des Lernlevels zwischen Fragesteller und Ansprechpartner kann es zu Kommunikationsproblemen kommen. Dies liegt daran, dass gerade Anfänger kontextfreie Anweisungen benötigen. Ein Experte denkt hingegen in kontextsensitiven Strukturen und kann diese nur schwer auf eine kontextfreie Form herunterbrechen. [22]

Wenn das System einen Experten vorschlägt sollte es daher den Lernlevel berücksichtigen und dem Fragesteller einen passenden Kontakt anbieten.

Communities im Internet bilden sich um bestimmte Themenbereiche. Innerhalb dieser Bereiche kann man einen Experten finden, sein Wissen zu anderen Themen ist aber nicht ableitbar, da die Communities für sich separiert stehen. Dieses steht konträr zum Unternehmenskontext. Durch eine standardisierte Benutzerkennung sind Mitarbeiter in verschiedenen Systemen identifizierbar. Durch die Zusammenführung der Daten mit einem sozialen Medium entsteht ein Enterprise 2.0 System. Innerhalb eines Unternehmens ist zudem das soziale Netzwerk begrenzt durch die Belegschaft und für die Informationsobjekte gelten Regeln, die eine Auswertung erleichtern. Diese Einschränkungen definieren den Rahmen in dem die Masterarbeit die *Expertensuche* betrachtet. Zudem ergibt sich aus dem Unternehmenskontext ein valides Testbett zur Verifikation des Systems.

2.2 Masterarbeit Diskursverfolgung

Das Ziel der Masterarbeit *Diskursverfolgung* ist, dass Entwickler bei ihrer Arbeit unterstützt werden, indem wichtige Informationen über den Entwicklungsprozess der aktuellen Ressource aggregiert dargestellt werden.

Dazu werden dateispezifische Informationen aus verschiedenen Werkzeugen (Wiki, Versionsverwaltung, Ticketsystemen) zusammengeführt. Aus diesen wird ein Diskurs erstellt, der den Entwicklungsprozess der zu bearbeitenden Ressource darstellt. Dabei werden nicht nur die chronologischen Veränderungen des Codes ersichtlich, sondern auch zusätzliche Informationen zu Wikiseiten, Arbeitspaketen und Tickets, die während der Ent-

wicklung erstellt wurden. Dieser Diskurs wird dem Entwickler transparent und bedarfsorientiert in seiner Entwicklungsumgebung bereitgestellt.

Ein Szenario, das durch diese Arbeit unterstützt werden kann, bezieht sich auf das Phänomen *Old Software Never Dies* (siehe Abbildung 2.1).

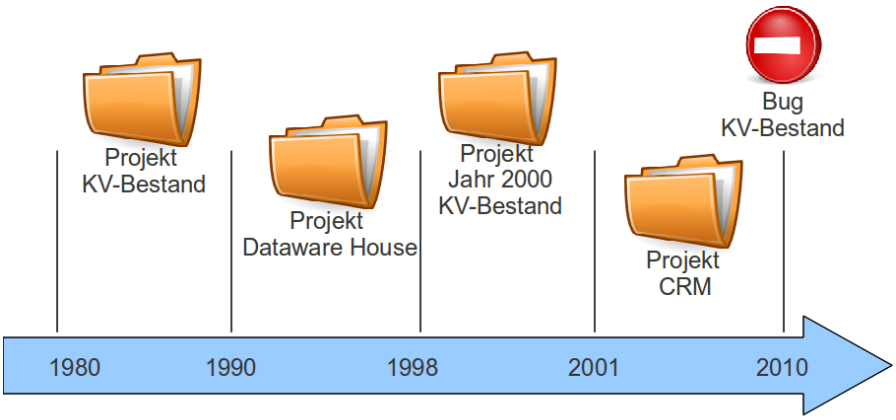

Abbildung 2.1: Szenario für die Arbeit in alten Projekten

Dieses Beispiel betrachtet die Bearbeitung eines Fehlers an einer Klasse des abgeschlossenen Projekts KV-Bestand. Das Projekt wurde bereits zur Jahrtausendumstellung wiedereröffnet. Allerdings muss sich der Bearbeiter, der den Fehler beheben soll, erst in das Projekt einarbeiten. Das geplante Tool soll dem Entwickler dabei helfen, indem der Diskurs der zu bearbeitenden Klasse aus der IDE abrufbar wird. Der Diskurs enthält Informationen über vorherige Bearbeiter, mit der Klasse verbundene Arbeitsaufgaben und die Änderungen der Klasse während der Entwicklung. Zudem werden Hinweise auf Einträge im Wiki mit hilfreichen Informationen dargestellt. Dadurch kann sich der Mitarbeiter besser den Kontext der Klassen aufbauen und schneller an die effektive Fehlerbehebung gehen.

In einem agilen Softwareentwicklungsteam mit *weak* oder *collectiv Code ownership* [18] ist nicht nur der vergangene Diskurs einer Ressource interessant, sondern auch deren aktuelle Entwicklung. Daher wird das Werkzeug der *Diskursverfolgung* als *peripheral awareness*-System realisiert, um die Entwickler frühzeitig über mögliche Überschneidungen zu informieren. Bei der Entwicklung von *peripheral awareness*-Systemen wird die Verwendung der unterbewussten Wahrnehmung gefördert, um das Bewusstsein nur geringfügig von der aktuellen Aufgabe abzulenken. [20] Der Aufwand zur Aufnahme der Informationen sollte dabei für den Benutzer so gering wie möglich gehalten werden. [21] Das Werkzeug lehnt sich an die Darstellungsform bekannter IM (Instant Messaging)-Systeme[49] an. Allerdings wird nicht der Status anderer Benutzer angezeigt, sondern ob und wie diese eine Ressource verwenden. Dabei können folgende Unterscheidungen gemacht werden:

Geöffnet – Dieser Status gibt an, dass die Ressource zum Lesen geöffnet wurde.

Modifiziert – Dieser Status zeigt, dass die Ressource verändert wurde. Wird dies von mehreren Entwicklern gemacht entstehen Merge-Konflikte.

Aufgabenrelevant – Bei diesem Status wird eine Vorhersage getroffen, wie wahrscheinlich eine Ressource auf Grund aktueller Tätigkeit verändert wird.

Weitere Information zur Status-Anzeige können beispielsweise aus der Anzahl von *Commits* und der Anzahl der verschiedenen Bearbeiter gewonnen werden. Wodurch mögliche Konflikte ermittelt und der Entwickler vorgewarnt werden kann. Eine gängige Möglichkeit der Darstellung der *peripheral awareness* in der Entwicklungsumgebung ist das Ampel-System, wobei Rot eine hohe Aktivität und Grün die freie Bearbeitung signalisiert.

Die beiden Hauptaufgaben für diese Arbeit sind das Erstellen eines Eclipse Plug-Ins [6] und eines Analyzers für das `LuPanKu`-System (siehe Abb. 2.2).

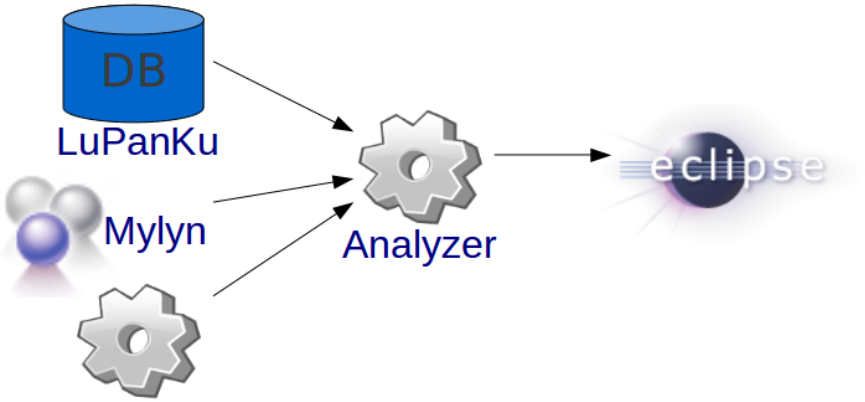

Abbildung 2.2: Übersicht Masterarbeit Diskursverfolgung

Das Eclipse Plug-In dient der Darstellung des Diskurses und der *peripheral awareness* in der Entwicklungsumgebung Eclipse. Der Analyzer ist verantwortlich für die Aggregation der relevanten Daten aus der Fülle der Informationen. Aus diesen erstellt er den Diskurs der aktuellen Ressource und leitet diesen an die IDE weiter.

2.3 Masterarbeit Virtual Project Office

Das Ziel der Masterarbeit ist die Entwicklung eines *Virtual Project Office* für verteilte agile Teams. In agilen Teams ist die direkte Kommunikation ein wesentlicher Bestandteil. Diese funktioniert am besten von Angesicht zu Angesicht. [1] Daher wird für agile Teams

ein Raum empfohlen, in dem das gesamte Team zusammenarbeitet. Neben der Möglichkeit jederzeit direkt miteinander zu kommunizieren, bekommen die Mitglieder direktes Feedback über die Situation der Kollegen und den Stand des Projektes. [5] Gerade bei agilen Teams herrscht ein erhöhter Geräuschpegel, da sich die Entwickler verstärkt über ihre Aufgaben austauschen. Auch wenn ein Entwickler nicht direkt an einem Gespräch beteiligt ist, bekommt er dies unbewusst mit und kann darauf reagieren. [10]

Gerade in global arbeitenden Projekten kann das nicht umgesetzt werden, da mehrere Standorte involviert sind. Bereits die Verteilung des Teams auf mehrere Büros erschwert die Kommunikation und verhindert das Bewusstsein für die Tätigkeiten im Team. Das *Virtual Project Office* soll hierbei die Kommunikation erleichtern und das Bewusstsein für die Aktivitäten im Team verbessern.

Das Leitbild für das System ist ein Großraumbüro, in dem das Team zusammen sitzt. Das Team erzeugt dabei eine geschäftige Atmosphäre, in der gemeinsam auf das Projektziel hingearbeitet wird. Diese Atmosphäre trägt dazu bei, ein Gemeinschaftsgefühl und Vertrauen zu erzeugen. [5] So registrieren die Teammitglieder welche Anforderungen die Kollegen bearbeiten.

Ein weiterer Aspekt, der unterschwellig im Raum kommuniziert wird, ist die Situation der einzelnen Entwickler. Diese senden durch Haltung und Gestik unbewusst Signale aus, die anzeigen, ob sie konzentriert arbeiten und besser nicht gestört werden oder vor einem kniffligen Problem stehen, bei dem sie Unterstützung benötigen. [37] Die Kollegen registrieren diese Signale und können sich entsprechend verhalten.

Gerade für die Wissensarbeit des Teams sind das wichtige Informationen. Unterbrechungen bedeuten einen Verlust der aktuell gewonnen Erkenntnisse, die dann wieder erarbeitet werden müssen. [22] Neben dem zeitlichen Verlust belastet es auch den Entwickler, da so vermeidbarer Stress erzeugt wird. [27]

Für ein verteiltes Team sollen diese Informationen virtuell verfügbar gemacht werden. Daher wird das *Virtual Project Office* als ein *peripheral awareness*-System entwickelt. [36] Die Tätigkeiten und Situationen der Teamkollegen soll ein Entwickler in seiner Entwicklungsumgebung angezeigt bekommen, ohne dass diese Informationen ihn bei der Arbeit unterbrechen oder stören.

Für die Realisierung soll zunächst eine Kontaktliste mit den Teammitgliedern entwickelt werden. In dieser Kontaktliste wird dann der Status der Entwickler und der aktuellen Aufgabe angezeigt. Die aktuell bearbeitete Aufgabe soll aus den Informationen des Mylyn Plug-Ins extrahiert werden.

2.4 Zusammenfassung

Aus den beschriebenen Szenarien für die Masterarbeiten ergeben sich folgende zusätzliche Anforderungen an das `LuPanKu`-System:

Eclipse als Client
- Für die Szenarien des *Virtual Project Office* und der *Diskursverfolgung* muss die Eclipse IDE mit dem `LuPanKu`-Server Informationen über aktuelle Entwicklungen der Projektmitglieder und Ressourcen austauschen.

Identifikation der Entwickler Aufgabe
- In Kapitel 2.3 wurde gefordert die aktuelle Aufgabe eines Entwicklers als Information dem Team verfügbar zu machen. Dies soll zu einem besseren Verständnis der Team-Aktivitäten führen.

Identifikation der genutzten Ressourcen
- Für die Schaffung einer *peripheral awareness* muss identifiziert werden, welche Ressourcen ein Entwickler verwendet. Diese Informationen müssen an den `LuPanKu`-Server übermittelt werden, damit sie den Teamkollegen zur Verfügung stehen.

Ästhetische Benutzeroberfläche
- Der Mehrwert von *Enterprise 2.0*-Systemen besteht in den Informationen, die die Nutzer in diesen hinterlegen. Dabei ist die Eigeninitiative des Nutzers gefordert, welche Informationen er als relevant erachtet und verfügbar stellt. Dies legt einen besonderen Fokus auf die Usibility der Anwendung, sowohl bei der Bedienung als auch bei der Gestaltung. [45]

Metriken für Experten Suche
- Damit ein Softwaresystem mögliche Experten identifizieren kann, müssen Metriken gefunden werden. Für die *Expertensuche* bei der Softwareentwicklung basieren diese Metriken auf Informationen aus dem Entwicklungsprozess. Ein Weg diese zu ermitteln ist ein Socio-Technical-Congruence-Network. [47]

Visualisierung von Metriken
- Metriken ermitteln für einzelne Komponenten eines Systems Kenngrößen. Deren Aussagekraft gewinnt, wenn sie im Zusammenhang mit dem Gesamtsystem gebracht werden. Für das bessere Verständnis eignet sich eine graphische Darstellung. [23]

Diese Anforderungen machen den Einsatz weiterer Technologien (siehe Kapitel 4) notwendig. Dafür sind Anpassungen der Architektur vorgenommen worden.

3 Architektur-Änderungen

Durch die in Kapitel 2.4 erarbeiteten Anforderungen auf Basis der konkretisierten Überlegungen der Masterthemen, wurde der Bedarf für Anpassungen an die Architektur des `LuPanKu`-Systems deutlich.

Der bisherige Web-basierte Zugriff auf den Server reicht nicht mehr aus, da eine Integration des Systems in Eclipse-basierte Entwicklungsumgebungen notwendig ist. Dies bedeutet auf der Serverseite eine Kommunikationsschnittstelle, die Services für andere Clients verfügbar macht. Für die Eclipse wird daher ein Plug-In notwendig, welches die Interaktion mit dem `LuPanKu`-Server ermöglicht.

Neben diesem Plug-In sind weitere notwendig, die die benötigten Informationen aus der Entwicklungsumgebung extrahieren. Die Connector-Schnittstelle des `LuPanKu`-Servers erhält dadurch die Entwicklungsumgebung als einen weiteren Connector, der Informationen für die Datenbasis bereitstellt.

Die bestehende Web-Oberfläche zur Konfiguration und Auswertung des `LuPanKu`-Systems muss für die Nutzung im Kontext von Enterprise 2.0 überarbeitet und erweitert werden. Die bisherige Oberfläche orientierte sich an einer Eclipse-ähnlichen Darstellung und war für die Konfiguration eines Plug-In-basierten Systems ausgelegt. Für eine nahtlose Integration mit anderen Ajax-basierten Anwendungen, wie Wikis, ist ein Redesign der Oberfläche notwendig. Für ein besseres Verständnis über die Aktivitäten im Team ist gefordert, die aktuelle Aufgabe eines Entwicklers als Information dem Team verfügbar zu machen. Dafür muss diese identifiziert werden.

Aus diesen Anforderungen wurde die Architektur weiterentwickelt (siehe Abbildung 3.1). In dieser sind die neuen Elemente rot hervorgehoben.

Die in Projekt 1 entwickelte Architektur, mit der Connector-Schnittstelle und der Infrastruktur zur Speicherung der Meta-Informationen, wurde übernommen. [33] Die Benutzerschnittstelle wurde neu entwickelt, da zum einen die Web-Oberfläche den neuen Anforderungen gerecht werden muss und zum anderen Eclipse-basierte Entwicklungsumgebungen als Client identifiziert wurden.

Zur Realisierung der in Kapitel 2 beschriebenen Masterarbeiten wurde die Analyzer-Komponente restrukturiert. Die Analyzer können nun verteilt auf dem Server und den Clients entwickelt werden. Dies ermöglicht eine Vorverarbeitung auf Seiten des Clients und reduziert die Netzwerklast auf das Notwendige.

Der modulare Aufbau des Systems bietet sowohl die Möglichkeit der separierten Analyzer-Entwicklung zur Realisierung der individuellen Mastertheorien, als auch die Wiederverwendung von Komponenten mit überschneidender Funktionalität.

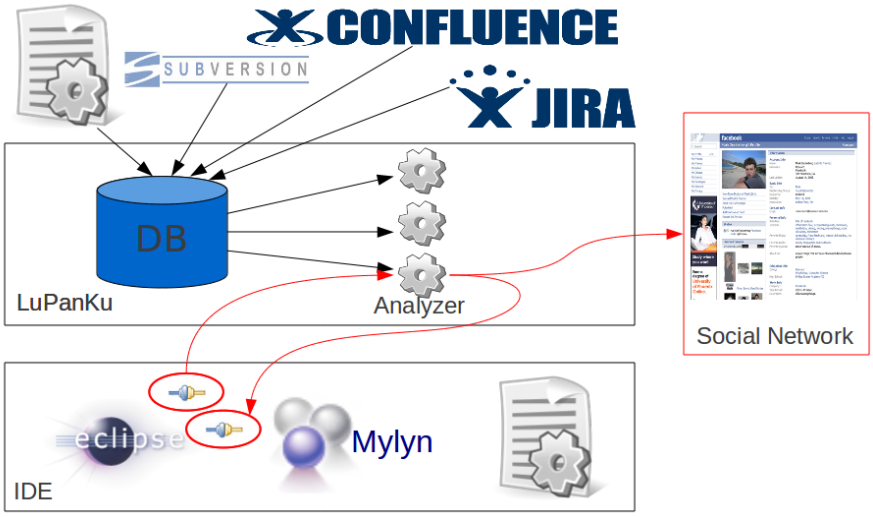

Abbildung 3.1: Übersicht neue Architektur

4 Technologien

In diesem Kapitel werden die Technologien beschrieben, die aufgrund der Anforderungen aus Kapitel 2.4, in das `LuPanKu`-System neu aufgenommen wurden.

In diesem Zusammenhang wurde für die bestehenden Technologien geprüft, ob für diese Weiterentwicklungen existieren, die einen Mehrwert für das System bieten. Das Ticketsystem Jira [31] ist in einer neuen Major Version 4.x erschienen. Zu den für das `LuPanKu`-Projekt relevanten Neuerungen gehören zum einen eine bessere Unterstützung von agilen Teams durch die nahtlose Integration der GreenHopper Komponente [30] und die Einführung der JiraQueryLanguage (JQL, siehe 4.4). Diese erlaubt präzisere und einfachere Abfragen des Datenbestandes. Dadurch kann der Jira-Connector performanter implementiert werden.

Die identifizierten neuen Technologien und die Kriterien aufgrund dessen diese ausgewählt wurden, werden im Folgenden beschrieben.

4.1 Kommunikation

Für die Szenarien des *Virtuel Project Office* (siehe Kapitel 2.3) und der *Diskursverfolgung* (siehe Kapitel 2.2) ist es notwendig, eine Kommunikation zwischen einer Eclipse IDE und dem `LuPanKu`Server aufzubauen. Beide Anwendungen sind auf Basis der OSGi Technologie [3] entwickelt. Daher wurde geprüft ob sich für die Kommunikation zwischen den beiden Anwendungen die OSGi Remote Services nutzen lassen. [2] OSGi bietet für seine Komponenten eine Serviceschicht an, um die verschiedene Komponenten mit geringer Kopplung zu verbinden. Darüber hinaus stellt die Serviceschicht eine dynamische Laufzeitumgebung bereit, in der Komponenten zu Laufzeit hinzugefügt, aktualisiert oder entfernt werden können. Die Komponenten können dabei über Services miteinander kommunizieren. Die Serviceschicht stellt dafür eine zentrale Service-Registry zur Verfügung um Services zu finden. Weiterhin werden Mechanismen für das dynamische An- und Abmelden der Services bereitgestellt. Diese Services werden lokal in der Java-VM ausgeführt und über Referenzen angesprochen.

Mit der Spezifikation 4.2 für OSGi ist das Konzept um Remote-Services erweitert worden. Diese erlauben eine Kommunikation mit Komponenten außerhalb der Java VM. Die Spezifikation beschreibt folgende Eigenschaften.

Einheitlicher Zugriff auf Services – Beide Servicetypen werden in der Serviceschicht verwaltet und können über diese bezogen werden.

Die Definition eines Service erfolgt mittels eines Java Interface. Ein Remote-Service wird, analog zu einem lokalen, in der Service-Registry unter einem Namen angemeldet. Bei der Anmeldung werden Properties übergeben. Wenn ein Service außerhalb der Java-VM angeboten werden soll, werden entsprechende Export Properties gesetzt. [2]

Abstraktion der Transportschicht – Innerhalb des OSGi Containers werden lokale und remote Services als Implementation eines Java Interfaces zur Verfügung gestellt. Dies verbirgt die Transportschicht für den entfernten Aufruf. Es ist Aufgabe des Containers den Java Aufruf auf eine Übertragungsschicht zu mappen. Die Spezifikation sieht dabei die Implementation von verschiedenen Technologien vor, wie RMI oder SOAP. Die Auswahl des Transportmediums wird über die Konfiguration des Remote-Services erreicht. Dadurch ist es möglich auch andere als OSGi-Systeme mit einzubeziehen. [2]

Sicherheit – Verteilte Services stellen einen Angriffspunkt auf das System dar. Die Spezifikation sieht vor, dass eine Sicherung, wie Verschlüsselung, innerhalb der Transportschicht zu erfolgen hat. Innerhalb des OSGi Containers bietet der *Conditional Permission Admin* Service die Möglichkeiten Berechtigungen für Services zu steuern. So kann ein Service prüfen, welche Signatur eine aufrufende Komponente hat. [2]

Die Funktionalität von Remote Services wird durch zusätzliche Komponenten im OSGi-Container zur Verfügung gestellt. Für `LuPanKu` wurde im Projekt 1 die OSGi-Implementation Equinox [17] aus dem Eclipse-Projekt gewählt. [33] Daher wurden Frameworks auf Basis von Equinox untersucht, um die gewünschte Funktionalität zu erhalten.

Riena – Das Ziel des Projektes Riena [15] ist das zur Verfügung stellen einer Client-Service-Architektur für Geschäftsanwendungen auf Basis von OSGi. Dafür stellt das Projekt zum einen eine Remote-Service-Implementation bereit und erweitert das bestehende Oberflächenkonzept der Eclipse-Rich-Client-Plattform. [44]

ECF – Das Eclipse Communication Framework (ECF) Projekt [11] erlaubt es verteilte Anwendungen zu entwickeln. Dabei werden zum einen Client-Server und Peer to Peer Architekturen unterstützt. Zudem unterstützt es synchrone und asynchrone Kommunikation, wobei der Schwerpunkt auf Werkzeuge zur Kommunikation wie Chat oder gemeinsame Editoren gelegt wird. [40]

Mit beiden Frameworks wurden Prototypen erstellt, um die Handhabung und Funktionalität zu bewerten. Die Entscheidung fiel zu Gunsten des ECF-Frameworks aus, da dieses eine höhere Flexibilität für die Remote-Services erlaubt und die in Kapitel 2.3 geforderte Unterstützung von Chats ermöglicht.

Für die Realisierung von verteilten Services definiert das ECF-Framwork zwei Schnittstellen. [9]

Discovery-API – Die Discovery-API definiert eine Schnittstelle auf welche Weise Services den OSGi-Containern bekannt gegeben werden. Dafür werden im ECF-Framework verschiedene Implementationen angeboten. So kann in einer Konfigurationsdatei dem System mitgeteilt werden welche Services auf welchem Server verfügbar sind. Eine weitere Implementation ist Apache Zookeeper [4]. Diese ermöglicht es mehrere Service Anbieter und Konsumenten dynamisch miteinander zu konfigurieren, ohne dass ein Knoten das gesamte Netzwerk kennen muss.

Remote-Service-API – Die Remote-Service-API abstrahiert von der zugrunde liegenden Transporttechnologie. Im Framework sind verschiedene Implementationen für SOAP, Rest, RMI und andere vorhanden. Die API unterstützt asynchrone Kommunikation indem der Serviceanbieter das Interface *IAsyncRemoteServiceProxy* implementiert.

Die Verwendung von OSGi-Remote-Services bietet den Entwicklern ein einheitliches und transparentes Programmiermodell für ein Komponentensystem, das auf Basis von lokalen und verteilten Services funktioniert. Die konkrete Technologie für den Aufruf von entfernten Services ist weggekapselt und austauschbar. Die Entwickler benötigen kein Wissen über die konkrete Übertragungstechnologie. Dafür steigt der Aufwand der Konfiguration und Initialisierung der Anwendung.

4.2 Mylyn

Das im Kapitel 2.2 beschriebene Szenario benötigt zur Erstellung einer *peripheral awareness* Informationen über die aktuelle Verwendung von Ressourcen eines Entwicklers. Diese Informationen kann das Eclipse Plug-In Mylyn bereitstellen. Es erzeugt auf Basis der Tätigkeiten eines Entwicklers in der Entwicklungsumgebung einen Kontext, in dem die Ressourcen nach ihrer Verwendung und Relevanz erfasst werden.

Dieser Kontext wird mit einer Aufgabe assoziiert. Diese Aufgabeninformation ist wiederum für das in Kapitel 2.3 beschriebene *Virtual Project Office* notwendig.

Die Zielsetzung des Plug-Ins Mylyn ist die Unterstützung des Entwicklers, indem die Informationsflut in der Entwicklungsumgebung reduziert wird. Dies wird durch das Ausblenden nicht benötigter Ressourcen und die Hervorhebung von relevanten Informationen

erreicht. Die Relevanz der Informationen ergibt sich aus der Arbeit des Entwicklers und wird mit seiner aktuellen Aufgabe verknüpft. [26]

Der Arbeitsablauf für die Verwendung von Mylyn gestaltet sich folgendermaßen. Zunächst wählt der Entwickler eine ihm zugewiesene Aufgabe aus einem Ticketsystem aus und aktiviert deren Bearbeitung. Dafür stellt Mylyn diverse Connectoren zu etablierten Ticketsystemen bereit. [43]

Während der Bearbeitung seiner Aufgabe, werden die Interaktionen des Entwicklers mit der Entwicklungsumgebung von Mylyn aufgezeichnet. Diese Interaktionen geben Auskunft über die Verwendung der Ressourcen. So kann beispielsweise zwischen Lesen und Verändern einer Ressource differenziert werden. Zusätzlich kann durch die Häufigkeit von Interaktionen einer Ressource deren Relevanz für die aktuelle Aufgabe gemessen werden. Diese wird durch den Degree-Of-Interest-Algorithmus (DOI) ermittelt. [26]

Der von Mylyn erstellte Aufgaben-Kontext kann als XML-Datei anderen Kollegen über das Ticketsystem zur Verfügung gestellt werden.

Dieser Mylyn Aufgaben-Kontext enthält die zur Erstellung der *peripheral awareness* Informationen bereit. Es werden die benutzen Ressourcen aufgezeigt und es wird die Art der Verwendung unterschieden.

Durch das explizite Auswählen an welcher Aufgabe der Entwickler arbeitet, lassen sich die Aktivitäten des Teams im *Virtual Project Office* darstellen. Mit Hilfe der von Mylyn registrierten und bearbeiteten Ressourcen kann die *peripheral awareness* der *Diskursverfolgung* (siehe Kap. 2.2) realisiert werden.

4.3 Rap Theming

Im `LuPanKu`-Projekt wurde entschieden, die Benutzeroberfläche als Webanwendung zu erstellen, wie im ersten Projekt-Bericht [33] in Abschnitt 6.5 dokumentiert. Dieses bietet die Vorteile, dass es eine zentrale Anwendungsstruktur gibt und dass keine Software auf den Clients installiert werden muss. Für die Umsetzung wurde das Framework RAP[28] (Rich Ajax Platform) gewählt. Die Plug-In basierte Struktur der RAP-Oberfläche unterstützt ein flexibles Entwickeln. Die Entwicklung von Oberflächen mit dem RAP-Framework gleicht dem programmieren von Desktopanwendungen. Auf Grund dessen ist der Lernaufwand eher gering. Durch geringen Programmieraufwand wird eine Anwendung im Web-Browser entwickelt, die im Design ebenfalls an eine Desktopanwendung erinnert (siehe Abbildung 4.1). Das Standardlayout basiert auf Fenstern, die die Funktionalität von Desktopfenstern haben und sich schließen, vergrößern und verkleinern lassen. Diese Optionen sind für Webanwendungen allerdings nicht optimal und vom `LuPanKu`-Projekt nicht erwünscht.

Um das Design der Anwendung anzupassen bringt das RAP-Framework den Mechanismus des Themings mit. Dadurch kann ein einheitliches Look and Feel[50] erzeugt werden[12], um die Arbeitsunterstützung für den Entwickler zu optimieren. Mit Verwendung des Themings können die Anwendungs-Bausteine in ihrem Aussehen verändert

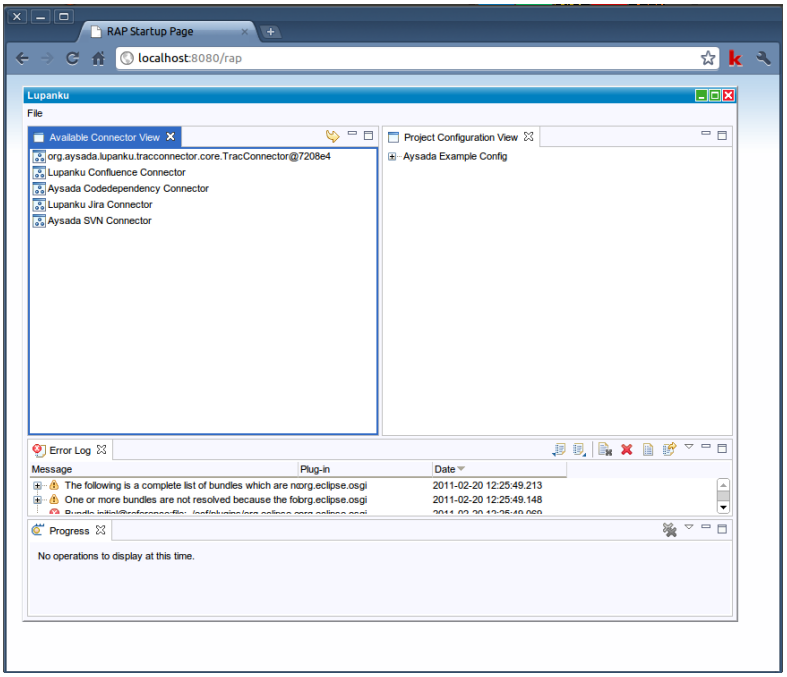

Abbildung 4.1: Desktop basiertes Look and Feel von RAP

werden, bspw. können die Farbe, die Schriftart oder auch die Umrandung benutzerdefiniert dargestellt werden. Geringfügige Umgestaltungen des Themings können einen Dialog verändern, wie in Abbildung 4.2 zu erkennen.

Das RAP-Framework setzt eine programmierte Desktopanwendung mit Hilfe der Javascript-Bibliothek qooxdoo[7] in HTML- und Javascript-Code um. Javascript-basierte Anwendungen werden vorwiegend auf zwei Weisen implementiert, einmal als *Thin Client* oder als *Fat Client*.

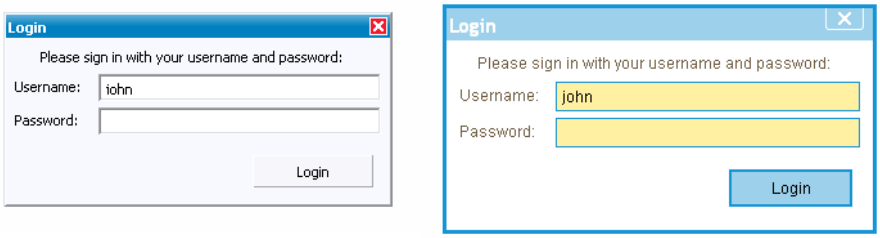

Abbildung 4.2: Theming veränderung eines Login Dialogs

Thin Client – Bei einem *Thin Client* wird die Geschäftslogik auf dem Server ausgeführt und nur die Darstellung wird im Browser vorgenommen und manipuliert.

[51]

Fat Client – Beim *Fat Client*-Modell wird die gesamte Logik auf den Client ausgelagert, so dass sowohl die Darstellung als auch die Geschäftslogik auf dem Client berechnet wird. Ein *Fat Client* fragt von dem Server nur die Daten ab, die er zu manipulieren und darzustellen hat. [48]

Das RAP-Framework basiert auf dem Konzept des *Thin Client*, somit wird die Geschäftslogik auf dem Server durch den Java-Quellcode berechnet und im Browser mit Hilfe von Javascript dargestellt. Die Trennung von Geschäftslogik und Präsentation ist durch dieses Konzept gewährleistet. An dieser Stelle wird kurz beschrieben wie ein Theme erstellt wird.

Erstellung eines RAP - Themes

RAP-Themes basieren seit der Version *RAP 1.1M4* auf CSS 2.1 (Cascading Style Sheets), somit lässt sich eine RAP - Anwendung mit denselben Techniken wie herkömmliche Web-Anwendungen im Look and Feel anpassen.

1. Erstellen eines Themes – Es ist eine `*.css`-Datei zu erstellen, um in dieser die Themeveränderungen einzutragen. Damit die Datei in der Anwendung vorhanden ist, muss sie in der `build.properties`-Datei eingebunden werden. Das Theme selbst muss nach der CSS-Syntax Version 2.1 erstellt werden, allerdings unterstützt RAP nicht die komplette CSS-Syntax, so sind bspw. alle *@-Regeln* nicht unterstützt. Ein kleines Beispiel eines Themes wird im Listing 4.1 dargestellt.

2. Theme registrieren – Damit das Theme der Anwendung bekannt ist, muss es in der `plugin.xml` registriert werden. Dieses ist mit dem Code aus Listing 4.2 zu bewerkstelligen. Eine Erweiterung bereits bestehende Themes ist möglich, in der Hilfe der Eclipse Foundation ist hierzu mehr zu erfahren [12].

3. Theme aktivieren – Um ein default-Theme anzugeben, muss ein Branding in der `plugin.xml` registriert werden, dieses kann mit dem Code aus dem Listing 4.3 erzeugt werden. Themes können auch mit Hilfe der *URI* umgestellt werden, so ist das Theme, falls mehrere Plug-Ins installiert sein sollten, mit der URL `http://hostname:port/lupanku` zu aktivieren.

```
Button[PUSH], Button[TOGGLE] {
  border: 2px solid blue;
  color: rgb( 17, 23, 103 );
  background-color: #f9f9f9;
}

Button[PUSH]:hover, Button[TOGGLE]:hover {
  background-color: white;
}
```

Listing 4.1: RAP Theme Beispiel

```
<extension
    point="org.eclipse.rap.ui.themes">
  <theme
    file="theme/lupankutheme.css"
    id="org.aysada.lupanku.ui.lupankutheme"
    name="Lupnaku Theme">
  </theme>
</extension>
```

Listing 4.2: Theme-Registrierung

```
<extension
      point="org.eclipse.rap.ui.branding">
   <branding
      defaultEntrypointId="org.aysada.lupanku.ui.Application"
      id="org.aysada.lupanku.ui.branding1"
      servletName="lupanku"
      themeId="org.aysada.lupanku.ui.lupankutheme">
   </branding>
</extension>
```

Listing 4.3: Theme-Aktivierung

4.4 Jira 4 und JiraQueryLanguage (JQL)

Im Jahr 2009 hat Atlassian mit der neuen Version 4 des BugTracking-Systems JIRA das größte Release in der Firmengeschichte veröffentlicht.[35] Zu den zahlreichen Neuerungen gehört eine komplette Überarbeitung der Benutzeroberfläche, eine Möglichkeit der Einbindung von OpenSocialGadgets und die Einführung der JIRA Query Language (JQL).[35]

Das letzte Feature ist für das Projekt vor allem von Interesse, da es die Abfrage über das SOAP erweitert. JQL vereinfacht das Erstellen von komplexen und individualisierten Suchanfragen.[35]

Dabei handelt es sich um eine an SQL angelehnte Abfrage-Sprache, die auf die Anforderungen zur Abfrage eines BugTracking-Systems angepasst ist. Wie bei SQL wird eine Abfrage aus verschiedenen Klauseln zusammengesetzt, die durch Verbindungsworte (AND oder OR) verbunden werden.(siehe Listing 4.4) Die Klauseln sind wie im bekannten Format aus einem Feld, einem Operator und einem Operanden zusammengesetzt.[32]

```
updated > 12/12/2010 AND assignee = currentUser()
```

Listing 4.4: JQL-Beispiel

Bei dem Feld handelt es sich um das Attribut auf das der Filter angewendet werden soll (`updated`). Der Vergleichsoperator legt die Art des anzuwendenden Filters fest

(>). Bei den Operanden kann es sich um Literale oder um JQL-Funktionen handeln (`12/12/2010`).[32] Eine für Entwickler interessante Erweiterung sind diese JQL-Funktionen. Da sie neben einer Reihe vorgegebener Funktionen, dem Entwickler die Möglichkeit gewähren, eigene Funktionen zu implementieren und so noch unbekannte Operanden zur Laufzeit zu ermitteln.[25]

4.5 Eclipse - GEF (Graphical Editing Framework)

In Abschnitt 2.4 wurde die Anforderung formuliert die Beziehungen zwischen technischen und sozialen Netzwerken zu visualisieren, um ein besseres Verständnis für deren Wechselwirkungen zu erlangen [13]. Die bestehende Implementation von `LuPanKu` basiert zu großen Teilen auf Frameworks der Eclipse-Plattform. Zur besseren Integration der Visualisierung von graphischen Darstellungen in die bestehende Infrastruktur, wurden die Möglichkeiten der Eclipse-Plattform untersucht.

Im Eclipse Umfeld werden graphische Editoren, auf Grund der leichten Einbindung in die Infrastruktur, standardgemäß auf Basis des Frameworks GEF (Graphical Editor Framework) [41] realisiert. Dieses nutzt die Draw2D-Bibliothek[39] für die graphische Darstellung, die auf SWT-Komponenten aufbaut. Die Wahl fiel auf GEF, da es ein höheres Abstraktionslevel zu Draw2D darstellt. In diesem wird das Pattern *Model-View-Controller* implementiert. Das erleichtert die Integration von Visualisierungen in die Anwendung, ohne das Datenmodell zwingend in ein GEF-Spezifisches umzuwandeln.

Modell – Das Modell kann durch beliebige *PoJos* repräsentiert werden.

Controller – Ein GEF-Controller wird durch eine Unterklasse von `EditPart` realisiert. Dieser Controller hält die Referenzen auf das Modell und den View. Er synchronisiert diese miteinander und verarbeitet die Benutzerinteraktionen des Editors. Ein Controller kann zusätzlich Kindelemente enthalten vom Typ `EditPart`, womit das Composite Pattern [19] realisiert wird.

View – Der View eines Modells wird durch Unterklassen von `Figure` repräsentiert. Wobei jedes `Figure` eine Modell-Objekt oder deren Beziehung visualisiert. Die `Figure` nutzt dabei die Funktionalität der Draw2D-Bibliothek.

Die Basis jeder GEF Darstellung ist der `GraphicalViewer`, dieser bildet den Rahmen für die Registrierung des `EditParts` und kümmert sich um das Layout dieser.

Der Lebenszyklus des Contollers ist ein wichtiger Bestandteil für das Verständnis des Frameworks. Selbiger besteht aus folgenden Phasen:

Erzeugung – Ein `EditPart` wird entweder vom `GraphicalViewer` direkt mittels einer Fabrik erzeugt oder durch das Composite Pattern bei der Bearbeitung eines anderes `EditPart`. Dieser ist dann sein Vaterknoten. Danach wird diesem Controller sein Modell bekannt gegeben.

Hinzufügen zum Diagramm – In dieser Phase wird zunächst die Vater-Sohn-Beziehung des Controller-Objekts sichergestellt. Daraufhin wird die graphische Repräsentation, die `Figure`, erzeugt und eine Verbindung zum Event-Handler hergestellt. Nach dieser Initialisierung wird die Komponente aktiviert.

Nutzungsphase – In der Nutzungsphase kann das Objekt manipuliert werden, indem es bspw. verschoben, umbenannt oder verändert wird.

Abräumen – Beim Abräumen eines Contoller-Objekts löst dieses automatisch all seine Abhängigkeiten auf. Dabei löscht es unter anderem sein Modell, den View und alle verwendeten Ressourcen.

Das Framework GEF folgt etablierten Entwurfsmustern und realisiert mit einfachen Konzepten eine effektive Technologie zur Umsetzung von komplexen graphischen Editoren.

5 Implementation

5.1 PID - Metrik

Ein Ansatz der *Expertensuche* besteht in dem Auffinden von Programmschnittstellen und den dazugehörigen Ansprechpartnern. Die automatisierte Identifikation von Ansprechpartnern für Schnittstellen ist über das Aufbauen eines *Socio-Technical-Conguence Network*[47] möglich. Dabei wird der technische Abhängigkeitsgraph in Beziehung zu dem sozialen Netzwerk der Softwareentwickler gesetzt.

Zur Abbildung eines sozialen Netzwerks innerhalb eines Unternehmens lassen sich die hierarchischen Organisationsstrukturen des Betriebes nutzen. Des Weiteren können Informationen über gemeinschaftliche Projekte verschiedener Abteilungen in dem sozialen Netzwerk skizziert werden.

Durch die Aufzeichnung der Aufruf-Hierarchie von Mainframeprogrammen lassen sich ihre technischen Abhängigkeiten darstellen. Dieses ist sinnvoll, weil Mainframprogramme meist in Unterprogrammen organisiert sind.

Die Darstellung des sozialen Netzwerks und der Programm-Hierarchie lassen sich durch gerichtete Graphen visualisieren.

Zur Validierung der Metrik wird ein Versicherungsbestandssystem mit seinen Nachbarsystemen verwendet. Dieses zeichnet sich dadurch aus, dass über Abteilungsgrenzen hinweg entwickelt wurde.

Die Anwendung und ihre Nachbarsysteme wurden auf Basis von Mainframe-Technologien entwickelt. Für die Identifikation des Programmgraphen analysiert ein Scanner Source-Code auf Abhängigkeiten. Als Scanner wurde XINFO [24] gewählt, da er sowohl Module zur Prüfung von Cobol als auch PL1 Programm-Code bereitstellt. XINFO speichert die Ergebnisse in einer relationalen Datenbank der Programm Informations DB (PID). Die Anbindung an `LuPanKu` geschieht über den Plug-In Mechanismus, allerdings wurde die Implementation für die PID-Metrik aus lizenzrechtlichen Gründen Closed Source entwickelt. Zur Darstellung der Ergebnisse werden Graphen auf Basis der in Kapitel 4.5 beschriebenen Technologie GEF erzeugt.

Zur Erzeugung des sozialen Netzwerkes werden Daten aus firmeneigenen Organisations-Datenbank (OrgDB) genutzt. Dieses soziale Netzwerk kann mit dem technischen Netzwerk der Programme verbunden werden, indem die Informationen aus dem Buildmanagementsystem ChangeMan [38] genutzt werden. Dieses speichert die Beziehung zwischen dem Programm-Modul und den verantwortlichen Entwicklern.

Zum Aufspüren der Programmschnittstelle und dessen Verantwortlichen wird ein Graph über alle Programmaufrufe erstellt. Dieser wird mit Informationen zu den jeweiligen Programm-Verantwortlichen erweitert. Dafür wird aus dem Changeman Datenbestand die Beziehung zwischen Programm und Verantwortlichem und in einem zweiten Schritt die Organisationseinheit des Mitarbeiters auf Basis der OrgDB ermittelt.

Unter Verwendung dieser Informationen wird aus dem komplexen Programmaufruf-Graphen (Abbildung 5.1) ein anschauliches Socio-Technical-Congruence-Network (Abbildung 5.2). In diesem Netzwerk lassen sich die Programme einer Organisationseinheit zusammenfassen. Daraus entsteht ein Graph, indem die Knoten die Abteilungen repräsentieren und die Kanten Aufrufe von Programmen anderer Abteilungen darstellen. Kandidaten für Schnittstellen ergeben sich aus Programmen, die von anderen Abteilungen aufgerufen werden.

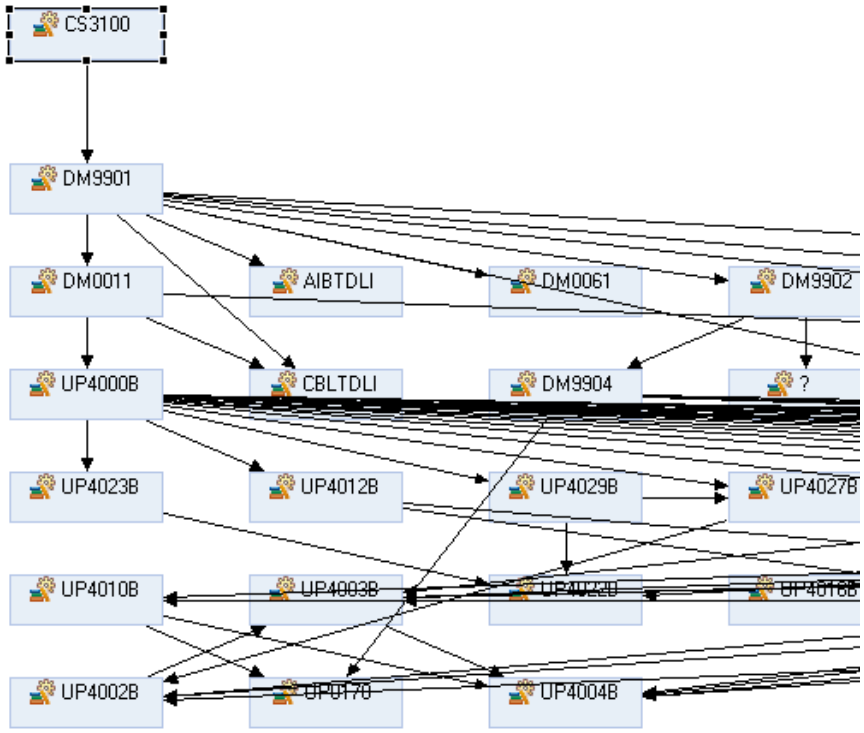

Abbildung 5.1: Programmhierarchie

Der Verantwortliche eines Programms, das als Schnittstelle identifiziert wurde, ist für diese der Experte und wird somit als Ansprechpartner identifiziert. Die Ergebnisse, die bei der Analyse des Versicherungsbestandssystem ermittelt werden konnten, wurden mit den beteiligten Entwicklern diskutiert. Die Resultate wurden von ihnen bestätigt und die Metrik konnte auf Schnittstellen hinweisen, die vorher nicht als solche definiert waren.

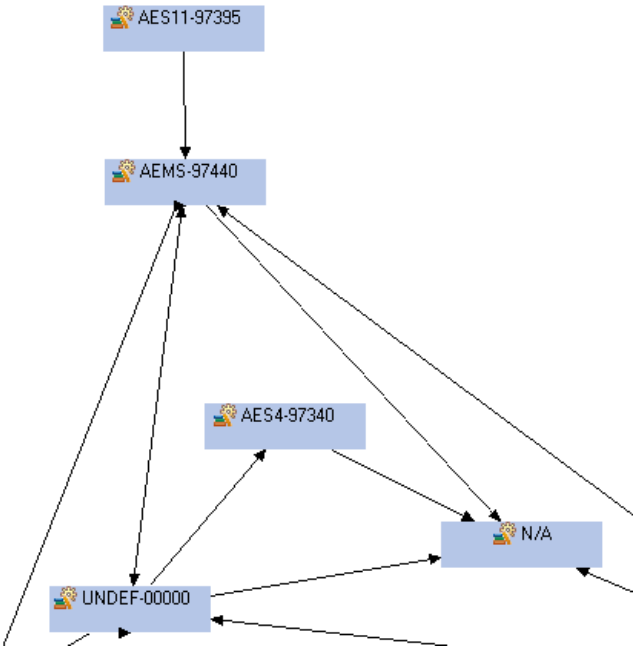

Abbildung 5.2: Socio-Technical Network

5.2 OSGi-Remote-Services

Für die Kommunikation zwischen dem `LuPanKu`-Server und den Eclipse-basierten Entwicklungsumgebungen wurden *OSGi-Remote-Services* gewählt. Diese werden, wie in Kapitel 4.1 beschrieben, von dem Riena- und dem ECF-Projekt bereitgestellt. Da die ECF-Implementation den vollen Umfang verwendet und auch für den in Kapitel 2.3 geforderten Chat-Server verwendet werden soll, wurde das ECF-Projekt gewählt.

Die in den Masterthemen beschriebenen Szenarien verwenden `LuPanKu` als zentralen Server. Aus diesem Grund wurde auf einen Discovery-Dienst verzichtet. Dadurch muss jeder Client den Server selbst konfigurieren, aber die technologische Komplexität reduziert sich und erleichtert den Betrieb der Anwendung.

Die Remote-Services können auf zwei Arten im OSGi-Kontext registriert werden. Entweder deklarativ über XML-Dateien oder programmatisch im Activator des Bundles. Die Entscheidung fiel hier zu Gunsten der Programmierung, da hierdurch das Debuggen erleichtert wird.

Für den Zugriff auf die Services werden, wie in Kaitel 4.1 beschrieben, die Wege über die OSGi-Service-Registry oder der Verwaltung eines eigenen ECF-Containers angeboten. Beide Wege erfordern bei der Handhabung von Remote-Services zusätzlichen Aufwand, so dass der Entwickler explizit zwischen lokalen und verteilten Services unterscheiden muss.

Bei einem Zugriff über die OSGi-Registry muss der Client zuvor das ECF-Framework initialisiert haben. Dadurch müssen mehr Plug-Ins in der IDE installiert und initialisiert werden, als für die Kommunikation notwendig sind. Daher wurde die Entscheidung für das Erzeugen eines eigenen ECF-Containers getroffen. Der so notwendige Mehraufwand an Programmierung wurde, für den geringeren Aufwand der Konfiguration und Initialisierung der Umgebung, in Kauf genommen.

Als Übertragungsprotokoll wurde R-OSGi gewählt. Dieses Protokoll ist nicht proprietär wie das ECF-Protokoll und wird von mehreren OSGi-Containern unterstützt. Ein weiterer Vorteil ist der geringe Verbrauch von Ressourcen und der geringere Overhead bei der Datenübertragung, den beispielsweise ein Webservice verursachen würde.

Für die Implementierung der Remote-Services wurden drei Bundles implementiert:

Services API – In diesem Bundle befinden sich die Service Definitionen als Java-Interfaces beschrieben. Diese Schnittstellenkomponente wird sowohl auf dem Server als auch dem Client installiert. Sie enthält zwei Interfaces, den IClientListener, den der Server für den Client anbietet und den ICallBackListener, den der Server nutzt, um dem Client Nachrichten zuzusenden. Durch diesen Mechanismus wird ein Pollen der Clients vermieden.

Server Remote Service – In diesem Bundle befindet sich die Implementation des Service. Dieser wird über den Activator beim Server angemeldet (siehe Listing 5.1).

Remote Service Client – Der Client speichert keine Referenz auf den Remote-Service, sondern macht vor jedem Zugriff einen neuen Lookup (siehe Listing 5.2). Dadurch kann das System auf Unterbrechungen der Netzwerkverbindung besser reagieren.

```
public void start(BundleContext bundleContext) throws Exception
{
  // Create R-OSGi Container
  IContainerFactory factory = ContainerFactory.getDefault();
  IContainer container = factory.createContainer("ecf.r_osgi.peer");
  // Get remote service container adapter
  IRemoteServiceContainerAdapter containerAdapter =
    (IRemoteServiceContainerAdapter) container
      .getAdapter(IRemoteServiceContainerAdapter.class);
  // Register remote service
  containerAdapter.registerRemoteService(
    new String[] { IClientListener.class.getName() },
      new ClientListener(), null);
}
```

Listing 5.1: Remote Service veröffentlichen

```
IRemoteServiceContainerAdapter containerAdapter =
  (IRemoteServiceContainerAdapter) container
  .getAdapter(IRemoteServiceContainerAdapter.class);
IRemoteServiceReference[] references = containerAdapter
```

```
    .getRemoteServiceReferences(IDFactory.getDefault().createID(
        container.getConnectNamespace(), "r-osgi://localhost:9280"),
        IClientListener.class.getName(), null);
  IRemoteService remoteService = containerAdapter
    .getRemoteService(references[0]);
  IClientListener proxy = (IClientListener) remoteService.getProxy();
```

Listing 5.2: Remote Service auflösen

Die gewählte Implementation erwies sich im Prototyp als robust.

5.3 Jira-Connector

Nachdem der bisherige Jira-Connector des Projekts *HomeOffice 2.0* auf die Version 3 ausgelegt war (siehe [33]), wurde auf Grund der umfassenden Funktionserweiterung in der aktuellen Version des Jira-Systems (siehe Kapitel 4.4) beschlossen, auch einen Connector für die Version 4 zu implementieren.

Wie im Projekt1-Bericht beschrieben, stellen Jira-Systeme RPC-Plugins zur Verfügung, welche einen Zugriff auf die Jira-Datenbank mit SOAP oder XMLRPC ermöglichen. Wie auch in der ersten Version des Jira-Connectors wird der WSDL-Deskriptor der Jira-Installation verwendet, um ein Paket mit den Klassen und Methodenaufrufen zu erstellen, die für die Verbindung zu Jira verantwortlich sind.[33]

Der Unterschied zur ersten Version des Jira-Connectors besteht darin, dass der Methodenaufruf zur Abfrage JQL verwendet. Dies ermöglicht eine Vorselektion der zu lesenden Daten. In der alten Version wurde eine allgemeine Abfrage der Daten in der Jira-Datenbank durchgeführt und nach der Übertragung zum `LuPanKu`-Server die benötigten Daten ausgewählt. In der aktuellen Version werden die zu übertragenden Daten schon bei der Abfrage aus der Jira-Datenbank eingegrenzt. Dadurch kann ein Teil der Verarbeitung auf dem Server der Jira-Datenbank durchgeführt werden und der Datentransfer über SOAP reduziert werden. Dies wiederum führt zu einer stark verbesserten Performance des `LuPanKu`-Systems.

5.4 Mylyn Adapter

Für die Extraktion der Informationen aus Mylyn heraus, wurde das Plug-In *Mylyn-Adapter* implementiert. Diese Komponente dient als Fassade für das Mylyn Plug-In. Dem `LuPanKu`-Server werden die benötigten Informationen in einem eigenen Datenmodell übermittelt, um den Server unabhängig von dem Taskkontext-System zu gestalten. Im Rahmen von Projekt 2 wurden zunächst folgende Funktionen entwickelt:

Aktuell verwendete Dateien – Der *Mylyn Adapter* ermittelt den aktuellen Arbeitskontext des Entwicklers. Zur Erstellung des Kontextes zeichnet Mylyn die Interaktionen des Entwicklers mit

der Entwicklungsumgebung auf und speichert diese in einer *InteractionHistory*. Jede Interaktion betrifft eine Datei im *Workspace*. Aus diesen Daten werden dann die für die aktuelle Aufgabe verwendeten Dateien ermittelt (siehe Listing 5.3).

Aktuell bearbeitete Dateien – Neben den Dateien wird in der *InteractionHistory* bei jeder Interaktion gespeichert, wie der Entwickler mit dieser interagiert hat. Die Interaktionen werden dabei nach verschiedenen Arten unterschieden wie Selektion, Befehl oder Manipulation [26]. Auf Basis dieser Interaktionsarten lassen sich die aktuell für die Aufgabe veränderten Dateien ermitteln (siehe Listing 5.4).

Aktueller Arbeitsauftrag – Für eine erweiterte Statusanzeige im *Virtual Project Office* soll die aktuelle Aufgabe des Entwicklers angezeigt werden. Diese entspricht der in Mylyn ausgewählten *Task*. Daher wird sie von Mylyn abgefragt und als Information dem `LuPanKu`-Server zur Verfügung gestellt. Dafür wird, wie in Listing 5.5 gezeigt, zunächst der aktuelle Kontext ermittelt. Dieser verfügt über eine Referenz-ID der *Mylyn Task*.

```
public Set<String> getAllUsedFiles() {
  Set<String> contextInfos = new HashSet<String>();
  IInteractionContextManager cxtManager = ContextCore.getContextManager();
  IInteractionContext context = cxtManager.getActiveContext();
  List<IInteractionElement> interesting = context.getInteresting();
  for (IInteractionElement iInteractionElement : interesting) {
      contextInfos.add(iInteractionElement.getHandleIdentifier());
  }
  return contextInfos;
}
```

Listing 5.3: Verwendete Dateien ermitteln

```
public Set<String> getAllUsedFiles() {
  Set<String> contextInfos = new HashSet<String>();
  IInteractionContextManager cxtManager = ContextCore.getContextManager();
  IInteractionContext context = cxtManager.getActiveContext();
  List<IInteractionElement> interesting = context.getInteresting();
  for (IInteractionElement iInteractionElement : interesting) {
      IDegreeOfInterest doi = iInteractionElement.getInterest();
      List<InteractionEvent> events = doi.getEvents();
    for (InteractionEvent event : events) {
      if(Kind.MANIPULATION == event.getKind())
        contextInfos.add(iInteractionElement.getHandleIdentifier());
    }
  }
  return contextInfos;
}
```

Listing 5.4: Bearbeitete Dateien ermitteln

```
public String getActualTask() {
  IInteractionContextManager cxtManager = ContextCore.getContextManager();
  IInteractionContext context = cxtManager.getActiveContext();
  String handleIdentifier = activeContext.getHandleIdentifier();
  TaskList taskList = TasksUiPlugin.getTaskList();
  return taskList.getTask(handleIdentifier).getSummary();
}
```

Listing 5.5: Ermitteln der aktuellen Aufgabe

Die vorgestellten Methoden des *Mylyn Adapters* reichen für eine prototypische Implementation der *peripheral awareness* in Kapitel 5.5 aus.

5.5 Peripheral Awareness

Sowohl für das *Virtual Project Office* als auch für die *Diskursverfolgung* ist es notwendig aktuelle Informationen über die Aktivitäten der Kollegen und den verwendeten Ressourcen zeitnah an alle Mitglieder zu verteilen (siehe Kapitel 2.4). Diese Informationen sind verfügbar im Aufgabenkontext von Mylyn (siehe Kapitel 4.2). Mittels eines Prototypen wurde überprüft, ob auf Basis dieser Kontextinformationen das gewünschte Szenario umgesetzt werden kann. Der Prototyp verifiziert dabei die Implementation der OSGi-Remote-Services (siehe Kapitel 5.2) im `LuPanKu`-Server sowie den in Kapitel 5.4 beschriebenen *Mylyn-Adapter*.

Die `LuPanKu`-Service-API wurde zunächst so einfach wie möglich gestaltet. Sie beinhaltet lediglich das *ClientListener*- und das *CallBackService*-Interface.

Das *ClientListener*-Interface (siehe Listing 5.6) umfasst die Deklaration der *registerClient*-Methode und der *submitContext*-Methode.

Die *registerClient*-Methode ist verantwortlich für die Registrierung des Clients beim `LuPanKu`-Server. Hierbei identifiziert sich der Client beim Server und registriert seine Implementation des *CallBackService*-Interfaces. Anhand dieses Interfaces kann der Server bei Bedarf den Clients Nachrichten schicken und diese müssen nicht regelmäßig auf neue Nachrichten prüfen. Diese Funktion wird genutzt, wenn ein Client dem Server mittels der *submitContext*-Methode einen Auszug des Mylyn-Kontextes überträgt. Der Server verteilt dann über das *CallBackService*-Interface diese Informationen an die anderen Clients.

```
  public void registerClient(String clientId, String clientServiceURL);
  public void submitContext(Set<String> contextInfos);
```

Listing 5.6: ClientListener Interface

Das *CallBackService*-Interface (siehe Listing 5.7) beinhaltet, in der aktuellen Ausbaustufe des Prototypen, die Deklaration der *handleContextOfColleague*-Methode. Diese wird vom Client implementiert, um den erhaltenen Mylyn-Kontext zu verarbeiten.

```
public void handleContextOfColleague(Set<String> contextInfos);
```

Listing 5.7: CallBackService Interface

Der Prototyp wurde durch ein einfaches Test-Szenario überprüft. In diesem Szenario arbeiten zwei Entwickler mit ihren lokalen Arbeitskopien an einem gemeinsamen Projekt. Deren Eclipse-basierte Entwicklungsumgebung beinhaltet das Mylyn Plug-In und das LuPanKu-Connector Plug-In. Der Arbeitsablauf gestaltet sich wie folgt:

Anmelden am LuPanKu-Server – Der Prototyp integriert das LuPanKu-System durch einen Eclipse-View in die Entwicklungsumgebung. Von diesem View ausgehend ist die Anmeldung an den LuPanKu-Server mittels eines Anmeldedialoges möglich.

Auswahl einer Mylyn-Aufgabe – Für die *peripheral awareness* ist es notwendig zu wissen welche Ressourcen vom Entwickler genutzt werden. Diese Informationen sind im Mylyn-Aufgaben-Kontext vorhanden [26]. Damit dieser bei der Entwicklung erzeugt wird, muss der Entwickler eine Aufgabe aktivieren. Dies geschieht, indem er eine ihm zugewiesene Aufgabe aus dem Task-View auswählt und diese aktiviert. [42]

Aufgabe bearbeiten – In diesem Schritt bearbeitet der Entwickler im gewohnten Vorgehen die ihm zugeteilte Aufgabe. Dabei werden eine oder mehrere Ressourcen des gemeinsamen Projekts modifiziert. Diese Tätigkeiten werden von Mylyn erfasst und im Kontext kontinuierlich aufbereitet.

Übermitteln des Kontextes – In der ersten Ausbaustufe des Prototypen übermittelt dieser nicht automatisch die Kontextinformationen zum LuPanKu-Server. Der Entwickler entscheidet selbst, zu welchem Zeitpunkt er diese Informationen für das Team freigibt. Das LuPanKu-Eclipse Plug-In extrahiert dann aus dem Mylyn-Kontext die relevanten Informationen und überträgt sie in einem Mylyn-unabhängigen Datenmodell an den Server.

Benachrichtigung des Teams – Sobald der LuPanKu-Server den Kontext eines Clients erhält, verteilt er diese wiederum an die bei ihm angemeldeten Clients. Das LuPanKu-Eclipse Plug-In gleicht den übermittelten Kontext mit dem aktuellen Kontext des Entwicklers ab. Werden bei dem Abgleich Dateien in beiden Kontexten gefunden, wird eine Warnung im Eclipse-Log-View ausgegeben. Dabei

unterscheidet das Plug-In zwischen zwei Hinweisleveln. Der erste Level ist durch die Konstellation definiert, dass ein Entwickler die Datei zur Ansicht geöffnet und der andere diese bereits modifiziert hat. Bei dieser Konstellation kann es zu einem Merge-Konflikt kommen. Der zweite Level beschreibt die Konstellation das beide Entwickler die gleiche Datei modifiziert haben. In diesem Fall besteht ein Merge-Konflikt, aus welchem Abstimmungsbedarf resultiert.

Durch den Prototypen wurde die Tragfähigkeit der gewählten Technologien überprüft. Das Test-Szenario wurde mehrmals mit unterschiedlichen Konstellationen durchlaufen, dabei wurden die Interpretation der Mylyn-Kontextinformationen verbessert. Die gesammelten Erfahrungen legen es nahe, den Prototypen im Rahmen der Masterarbeiten weiterzuentwickeln.

6 Erfahrungen

Wie in Kapitel 2 wurden zu Beginn der zweiten Projektphase neue Anforderungen aus den spezifizierten Master Themen für das `LuPanKu`-System abgeleitet. Für deren Realisierung notwendige Komponenten liessen sich durch den modularen Aufbau der OSGi-Architektur nahtlos integrieren. Dies wird erleichtert durch den Zwang die Schnittstellen einer Komponente explizit zu definieren. Zusätzlich leitet dieses Vorgehen die Entwickler an gemeinsame Module klar in ihrer Verantowrtung zu spezifizieren und erleichtert damit die Arbeit im Team.

Der wie im Projektbericht [33] beschriebene Prozess der Teambildung hat sich weiter entwickelt. Nach Tuckmann [46] hat das Team das Stadium *Performing* erreicht. Dies zeigte sich bei der gemeinsamen Arbeit im Projektraum daran, dass sich das Team bei Herausforderungen spontan zu einer konstruktiven Diskussion zusammenfand. In diesem Rahmen wurde dann ein gemeinschaftlicher Lösungsansatz gefunden.

Diese Herausforderungen ergaben sich zum Teil aus der verwendeten Technologie. So verbirgt zwar die Implementation der Remote Services das zugrunde liegende Transport-Framework. Dafür ist allerdings ein hoher Konfigurationsaufwand der OSGi Umgebung notwendig. Dies hat zur Folge, dass sich Fehler innerhalb dieser Konfiguration nur schwer untersuchen lassen.

Diese Problematik bestand nicht bei der RAP Komponente, jedoch stellte die Dokumentation das Team vor eine Herausforderung. Das Konzept zum Theming von RAP Obeflächen ist in der Vergangenheit mehrfach grundlegegend geändert worden. Dennoch sind veraltete Dokumentationen hierfür in verschiedenen Wikis und Blogs vorhanden. Diese erschwerten die Recherche nach der aktuellen Beschreibung. Für künftige Recherchen wäre eine Anfrage in der Newsgroup bei den Entwicklern ein schnellerer Weg.

Der in Kapitel 5.1 beschriebene Ansatz Programmschnittstellen auf Basis der sozialen Strukturen zu identifizieren, wurde im Kontext der Signal Iduna überprüft. Dabei bestätigten die Rückmeldungen der Entwickler den Nutzen des Ansatzes.

Das für die Visualisierung verwendete Framework GEF hat verwertbare Darstellungen ermöglicht, die weiterentwickelt werden können. Die Einarbeitungszeit in diese Technologie offenbarte allerdings eine hohe Lernkurve.

Die Migration von Jira auf die aktuelle Version 4 zeigte einen nicht erwarteten Anstieg der Hardwareanforderung. Hierfür musste zunächst zusätzliche Serverkapazität bereitgestellt werden. Dies verögerte die Implementation des Conntectors.

7 Ausblick

Die Zielsetzung für Projekt 2 war es die Infrastruktur im Hinblick auf die Masterthemen aufzubauen und erste Erfahrungen mit dieser zu sammeln. Durch die Rückmeldung der Entwickler zu der Programschnittstellen Metrik und den Erfahrungen mit dem Perepherie Awarness Prototypen scheint die Basis für die Masterarbeiten gelegt.

Bei der Bearbeitung der Masterthemen wird der Fokus der einzelnen Projektmitglieder auf der Entwicklung seperater Analyzerkomponenten liegen. Dadurch wird die bisherige enge Zusammenarbeit im Team aufgelockert. Allerdings wird der gemeinsame Projektrahmen HomeOffice 2.0 zum einen auf der theoretischen sowie auf der Implementationsebene eine lose Zusammenarbeit erforderlich machen.

Für die Masterarbeit Expertensuche wurde zum einen die Möglichkeiten einer ansprechenden Benutzeroberfläche untersucht und zum anderen einer erste Metrik entworfen die Experten für bestimmte Konstellationen liefert. Im ersten Schritt soll ein Enterprise 2.0 System entworfen werden, was die Informationen mit anderen Werkzeugen venretzt und eine Bewertung durch das soziale Netzwerk ermöglicht. Auf dieser Basis werden dann weitere Metriken untersucht, die sich für die Identifikation von Experten nutzen lassen. Diese sollen dann in Hinblick auf ihren Nutzen für die Komunikation und den Wissensaustauch untersucht werden.

Für die Masterarbeit Diskursverfolgung wurde mit dem Peripheral-Awareness-Prototypen gezeigt wie aktuell verwendete Ressourcen im Team sichtbar gemacht werden können. Im Rahmen der Masterarbeit muss zunächst überlegt werden, welche Informationen für den Diskurs einer Ressource von Interesse sind. Dabei sollen Informationen aus anderen Werkzeugen wie beispielweise Wiki und Ticketsystemen mit einbezogen werden. Hierbei ergibt sich die Herausforderung wie aus diesen Werkzeugen die relevanten Teilinformationen ermittelt werden können. Abschließend muss über eine adequate Darstellung innerhalb der Entwicklungsumgebung nachgedacht werden.

Für das *Virtual Project Office* wurden verschiedene Möglichkeiten untersucht, um die Atkivitäten der Entwickler im Team zu identifizieren. Dabei wurde zum einen der Arbeitsauftrag als Information erkannt und zum anderen könmnen aktuelle Arbeiten an den Sourcen transparent gemacht werden. Hierbei wird eine Herausferdung darin bestehen die Informationen zu verdichten um auf die Situation des Entwicklers hinzuweisen. Für die angedachte Visualisierung des Teams in Form einer Kontaktliste wurde das Kommunikationsframework ECF untersucht. Auf dieser Basis kann ein Instant Messanging System entworfen werden, das die Kommuniktation unterstützt und Informationen über die Situation der Entwickler darstellt. Bei diesen Betrachtungen soll das methodische

Vorgehen des Teams und welche Auswirkungen sich auf diesem durch das *Virtual Project Office* ergeben untersucht werden.

Aufgrund der Erfahrungen mit den Teamdiskussionen und des hohen Anteils gemeinsam genutzter Komponenten ist bei der Bearbeitung der Masterarbeiten eine Weiterführung des regen Austausches zu erwarten.

Literaturverzeichnis

[1] Agile: *Manifesto for Agile Software Development.* 02 2010. – URL http://agilemanifesto.org/

[2] Alliance, OSGi: *OSGi Service Platform Service Compendium.* August 2009. – URL http://www.osgi.org/Download/Release4V42

[3] Alliance, OSGi: *OSGi.* 2010. – URL http://www.osgi.org/Main/HomePage. – Zugriffsdatum: 04. July 2010

[4] Apcache: *Apache ZooKeeper.* Apache Fondation. – URL http://hadoop.apache.org/zookeeper/

[5] Beck, Kent: *Extreme Programming Das Manifest.* Addison Wesley, 2000

[6] Clayberg, Eric ; Rubel, Dan: *Eclipse: Building Commercial-Quality Plug-ins (2nd Edition) (Eclipse).* Addison-Wesley Professional, 2006. – ISBN 032142672X

[7] Community qooxdoo: *Qooxdoo.* – URL http://qooxdoo.org/. – Zugriffsdatum: 11. January 2011

[8] Dreyfus, Hubert L. ; Dreyfus, Stuart E. ; Athanasiou, Tom: *Mind over machine: the power of human intuition and expertise in the era of the computer.* New York, NY, USA : The Free Press, 1986. – ISBN 0-02-908060-6

[9] ECF: *OSGi 4.2 Remote Services and ECF.* Eclipsepedia. – URL http://wiki.eclipse.org/OSGi_4.2_Remote_Services_and_ECF

[10] Eckstein, Jutta: *Agile Softwareentwicklung mit verteilten Teams.* dpunkt.verlag, 2009

[11] Eclipse: *Eclipse Communication Framework.* – URL http://www.eclipse.org/ecf/. – Zugriffsdatum: 20. Dezember 2010

[12] eclipse.org: *RWT Theming.* 2011. – URL http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.rap.help/help/html/advanced/theming.html. – [Online; Stand 12. Februar 2011]

[13] Ferstl, Otto K. ; Sinz, Elmar J.: *Grundlagen der Wirtschaftsinformatik 1.* Oldenbourg, 2006. – URL http://www.worldcat.org/isbn/9783486579420. – ISBN 9783486579420

[14] Fischer, Mario: *Website Boosting 2.0: Suchmaschinen-Optimierung, Usability, Online-Marketing*. 2. Auflage. mitp, 2008

[15] Fondation, Eclipse: *Riena*. Eclipse Fondation. – URL `http://www.eclipse.org/riena/`

[16] Foundation, Eclipse: *Eclipse*. – URL `http://www.eclipse.org/`

[17] Foundation, Eclipse: *Equinox*. – URL `http://www.eclipse.org/equinox/`. – Zugriffsdatum: 04. July 2010

[18] Fowler, Martin: *Code Ownership*. 2006. – URL `http://martinfowler.com/bliki/CodeOwnership.html`

[19] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns*. Reading, MA : Addison Wesley, 1995

[20] Gennaro, Dr. Rocco J.: Representationalism, peripheral awareness, and the transparency of experience. In: *Philosophical Studies* (2007). – URL `http://www.theassc.org/files/assc/repperiph.pdf`

[21] Gross, Michael: *Computer-Supported Cooperative Work*. Oldenbourg Wissenschaftsverlag GmbH, 2007

[22] Hunt, Andy: *Pragmatic Thinking and Learning: Refactor Your Wetware (Prag- matic Programmers)*. Pragmatic Bookshelf, 2008. – ISBN 1934356050, 9781934356050

[23] Irwin, Warwick ; Churcher, Neville: Object Oriented Metrics: Precision Tools and Configurable Visualisations. In: *Proceedings of the 9th International Symposium on Software Metrics*. Washington, DC, USA : IEEE Computer Society, 2003, S. 112–. – URL `http://portal.acm.org/citation.cfm?id=942804.943757`. – ISBN 0-7695-1987-3

[24] IT, Horizont: *XINFO*. – URL `http://www.horizont-it.com/`. – Zugriffsdatum: 2011.02.09

[25] Jameson, Rosie: *JQL Function Plugin Module*. Januar 2011. – URL `http://confluence.atlassian.com/display/JIRA/JQL+Function+Plugin+Module`

[26] Kersten, Mik: *Focusing knowledge work with task context*. Vancouver, BC, Canada, Canada, Dissertation, 2007

[27] Kirsh, David: A Few Thoughts on Cognitive Overload. In: *Intellectica* 30 (2000)

[28] Lange, Fabian: *Eclipse Rich Ajax Platform: Bringing Rich Client Into the Web*. 1. Auflage. Springer Verlag Gmbh, 2008

[29] Laura A. Granka, Geri G.: Eye-Tracking Analysis of User Behavior in WWW Search. In: *Cornell University* (2006)

[30] Ltd., Atlassian P.: *GreenHopper*. – URL http://www.atlassian.com/software/greenhopper/. – Zugriffsdatum: 20. August 2010

[31] Ltd., Atlassian P.: *Jira*. – URL http://www.atlassian.com/software/jira. – Zugriffsdatum: 20. August 2010

[32] Maddox, Sarah: *Plugin Tutorial - Adding a JQL Function to JIRA*. Januar 2011. – URL http://confluence.atlassian.com/display/DEVNET/Plugin+Tutorial+-+Adding+a+JQL+Function+to+JIRA

[33] Matthias Holsten, Karsten Panier Daniel W.: *Projektbericht Home-Office 2.0*. 2010. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-proj1/panier.pdf

[34] McAfee, Andrew P.: Enterprise 2.0: The Dawn of Emergent Collaboration. In: *MITSloan* 47 (2006), Spring, Nr. 3, S. 21–28

[35] Olofsen, Ken: *JIRA 4 - The centre of your development team*. Juni 2009. – URL http://blogs.atlassian.com/jira/2009/10/jira-4-now-available.html

[36] Pedersen, Elin R.: People presence or room activity supporting peripheral awareness over distance. In: *CHI 98 conference summary on Human factors in computing systems*. New York, NY, USA : ACM, 1998 (CHI '98), S. 283–284. – URL http://doi.acm.org/10.1145/286498.286763. – ISBN 1-58113-028-7

[37] Rickheit, Gert (Hrsg.) ; Deutsch, Werner (Hrsg.) ; Hermann, Theo (Hrsg.): *Psycholinguistik / Psycholinguistics*. Berlin : De Gruyter Mouton, 2003 (Handbücher zur Sprach-und Kommunikationswissenschaft / Handbooks of Linguistics and Communication Science 24). – URL get-book.cfm?BookID=7654

[38] Software, Serena: *ChangeMan ZMF*. – URL http://www.serena.com/products/changeman-zmf/. – Zugriffsdatum: 2011.02.09

[39] Team, Draw2D: *Draw2D Extensions*. Eclipse Fondation. – URL http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.draw2d.doc.isv/guide/guide.html

[40] Team, ECF: *Eclipse Communication Framework Project*. Eclipse Fondation. – URL http://wiki.eclipse.org/ECF

[41] Team, GEF: *GEF Extensions*. Eclipse Fondation. – URL http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.gef.doc.isv/guide/guide.html

[42] Team, Mylyn: *Mylyn / User Guide*. Eclipse Fondation. – URL http://wiki.eclipse.org/index.php/Mylyn/User_Guide

[43] Team, Mylyn: *Mylyn Extensions*. Eclipse Fondation. – URL http://wiki.eclipse.org/index.php/Mylyn_Extensions

[44] Team, Riena: *Riena Project*. Eclipse Fondation. – URL `http://wiki.eclipse.org/Riena_Project`

[45] Tractinsky, Noam: Aesthetics and apparent usability: empirically assessing cultural and methodological issues. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 1997 (CHI '97), S. 115–122. – URL `http://doi.acm.org/10.1145/258549.258626`. – ISBN 0-89791-802-9

[46] Tuckman, B.: Developmental sequence in small groups. In: *Psychological Bulletin* 63 (1965), S. S. 384–389

[47] Valetto, Giuseppe ; Helander, Mary ; Ehrlich, Kate ; Chulani, Sunita ; Wegman, Mark ; Williams, Clay: Using Software Repositories to Investigate Socio-technical Congruence in Development Projects. (2007), S. 25. ISBN 0-7695-2950-X

[48] Wikipedia: *Fat Client — Wikipedia, Die freie Enzyklopädie*. 2011. – URL `http://de.wikipedia.org/wiki/Fat_Client`. – [Online; Stand 13. Februar 2011]

[49] Wikipedia: *Instant Messaging — Wikipedia, Die freie Enzyklopädie*. 2011. – URL `http://de.wikipedia.org/w/index.php?title=Instant_Messaging&oldid=83733164`. – [Online; Stand 6. Februar 2011]

[50] Wikipedia: *Look and Feel — Wikipedia, Die freie Enzyklopädie*. 2011. – URL `http://de.wikipedia.org/wiki/Look_and_Feel`. – [Online; Stand 17. Februar 2011]

[51] Wikipedia: *Thin Client — Wikipedia, Die freie Enzyklopädie*. 2011. – URL `http://de.wikipedia.org/wiki/Thin_Client`. – [Online; Stand 13. Februar 2011]