Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,
Arbeitsgruppe Software Engineering

Entwicklung einer Infrastruktur für wechselbare Projektübertragungsformen und Weiterentwicklung Bestehender in Saros

Daniel Theus
Matrikelnummer: 4466096
daniel.theus@fu-berlin.de

Betreuer: Franz Zieris
Eingereicht bei: Prof. Dr. Lutz Prechelt
Zweitgutachter: Prof. Dr. Robert Tolksdorf

Berlin, 06. November 2015

Zusammenfassung

Saros ist ein Plugin für die Entwicklungsumgebung Eclipse und ermöglicht bis zu 5 Teilnehmern ortsunabhängig und gemeinsam an einem Projekt zu arbeiten, wobei jede Änderungen direkt für alle Teilnehmer sichtbar wird.
Grundlage dafür ist das zugrundeliegende Projekt. Dieses muss zu Beginn einer Sitzung jedem Teilnehmer verfügbar gemacht und in den Eclipse-Workspace integriert werden. Im Falle von Änderungen wird es dann ständig synchronisiert, wodurch Änderungen von jedem Teilnehmer übernommen werden.
Um diese Voraussetzung zu Bewerkstelligen existieren zwei Varianten zur Projektübertragung. Die erste versendet die Projektdaten in Form einer gebündelten Zip-Datei, die zweite befindet sich noch im Entwicklungszustand, in Form eines Prototyps und ist nur unter Einschränkungen nutzbar.

Das Ziel dieser Arbeit ist die Integrierung beider Varianter, inklusive einer Auswahlmöglichkeit und die Weiterentwicklung der Funktionalitäten des Prototyps.
Anschließend werden die Stärken und Schwächen beider Varianten in Tests ermittelt und miteinander verglichen.

Hiermit versichere ich,

Name:	Daniel Theus
Matrikelnummer:	4466096
Geboren am/ in:	14. August 1989 in Neuruppin

an Eides statt, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe Dritter verfasst wurde und ich keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie wörtliche und sinngemäße Zitate als solche kenntlich gemacht habe.
Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen und wurde bisher nicht veröffentlicht.

06. November 2015

Daniel Theus

Inhaltsverzeichnis

1. Einleitung

Projekte im Bereich der Softwareentwicklung finden heutzutage nicht mehr nur zentral an einem Ort statt, sondern es liegen mitunter große Entfernungen zwischen den Entwicklern. Ob externer Mitarbeiter einer Firma oder quelloffenes Großprojekt im Netz, die verteilte Entwicklung an Projekten ist gängige Praxis, nicht nur in der Welt der Informationstechnik und erlaubt eine hohe Flexibilität, sowie die Beteiligung einer großen Zahl an Entwicklern. Hieraus resultieren jedoch auch Nachteile, so findet die Koordination von Projekten verstärkt auf digitalem Weg über das Internet statt und eine enge Zusammenarbeit vor Ort ist nur noch selten oder auch gar nicht realisierbar. Praktiken wie die „Paarprogrammierung“ im Bereich des „Extreme Programming“ [1] werden dadurch stark eingeschränkt.

Mittel um die verteilte Projektdurchführung zu erleichtern existieren bereits, beispielsweise in Form von Versionskontrollsystemen zur standortunabhängigen Durchführung von Änderungen und Protokollierung dieser an einem Projekt oder Reviewsystemen zur Beurteilung von Änderungen durch Teammitgliedern, bevor diese in ein Projekt integriert werden.

Das Saros-Projekt geht hier, in Form des quelloffenen Saros-Plugins für die Entwicklungsumgebung Eclipse, noch einen Schritt weiter. Statt nur zu protokollieren oder fertigen Code zu analysieren, ermöglicht Saros verteilte kollaborative Gruppenprogrammierung in nahezu Echtzeit. Hierbei sind Sitzungen mit bis zu 5 Teilnehmern, völlig Unabhängig vom Standort, möglich. Allerdings befindet sich Saros noch immer in einem Entwicklungsstadium und wird stetig erweitert und optimiert. Hauptthemen sind hierbei unter anderem die Portierung auf weitere Entwicklungsumgebungen neben Eclipse, die permanente Steigerung der Robustheit und Funktionalität, sowie die Verbesserung der Benutzbarkeit.

Einschränkungen der Robustheit und Funktionalität finden sich beispielsweise beim Starten einer Sitzung. Die Teilnehmer wollen gemeinsam an einem Projekt arbeiten, doch dies ist nur möglich, wenn alle Teilnehmer über die Daten des Projektes verfügen und diese Daten einheitlich, in synchronisierter Form, vorliegen. Saros bewerkstelligt

diese gemeinsame Projektgrundlage indem die Projektdaten zu Beginn einer Sitzung an alle Teilnehmer verschickt werden.

1.1. Problemstellung

Um die Verteilung der Projektdaten an alle Teilnehmer zu ermöglichen existieren zwei Varianten. Die Erste versendet das Projekt gebündelt als eine Datei, was unter Umständen in einer langen Wartezeit resultieren kann, denn die Arbeit am Projekt wird erst möglich sobald die gesamte Datei vollständig übertragen ist. Um diese Wartezeit zu verkürzen wurde eine zweite Variante im Rahmen einer Bachelorarbeit eingeführt [2]. Diese versendet jede Datei des Projektes einzeln und im Hintergrund. Sobald eine Datei empfangen wurde, ist es möglich an dieser zu arbeiten, ohne auf den Rest des Projektes warten zu müssen. Diese Variante liegt bislang nur in Form eines Prototyps vor und unterliegt größeren Einschränkungen, so können beispielsweise zu große Projektdateien zum Absturz führen. Ein weiterer Nachteil liegt in der Konzipierung des Prototyps, denn dieser wurde so angelegt, die erste Variante zu ersetzen, was aufgrund der Defizite nicht in Frage kommt und Praxistests unmöglich macht. Ohne Tests ist es allerdings auch nicht möglich zu evaluieren, ob die Weiterentwicklung des Prototyps sinnvoll ist und der funktionale Vorteil bereits während der Übertragung arbeiten zu können tatsächlich zum Tragen kommt.

1.2. Zielstellung

Das Ziel ist nun eine Struktur einzuführen welche es erlaubt, beide Verfahren nebeneinander in Saros zu integrieren und beliebig auswählen zu können. So wäre es möglich den Prototypen schrittweise weiterzuentwickeln, Änderungen zu integrieren und verwenden zu können und die bereits eingesetzte Variante weiterhin als Standard beizubehalten. Die bestehenden Einschränkungen des Prototyps erlauben einen Einsatz nur unter speziellen, den Einschränkungen entsprechenden, Gegebenheiten.

Um die Vergleichbarkeit beider Varianten zu verbessern, muss die Robustheit des Prototyps zusätzlich gesteigert werden.
Anschließend können Tests durchgeführt werden um zu klären, ob eine Variante generell besser als die andere funktioniert und eine Koexistenz unnötig wird, oder beide gleichermaßen oder sogar kombiniert, situationsabhängig, eingesetzt werden können.

1.3. Gliederung der Arbeit

Der gesamte Inhalt der Arbeit bezieht sich auf das Saros-Plugin. Dieses Plugin wird im Rahmen des Saros-Projektes entwickelt. Somit muss im ersten Schritt geklärt werden, was Saros ist und an welche Gegebenheiten die Arbeit am Projekt geknüpft ist. Dazu zählen verwendete Technologien und generelle Arbeitsprozesse. Anschließend wird die technische Ausgangslage genauer beschrieben, da diese Arbeit auf Basis vorangegangener Arbeiten und bestehender Technologie aufbaut. Sind die Rahmenbedingungen geklärt erfolgt die Umsetzung der notwendigen Implementierungsschritte mit anschließenden Tests der Varianten. Im letzten Abschnitt werden die Testergebnisse diskutiert, ein Vergleich der Varianten durchgeführt und ein Ausblick auf mögliche Weiterführungen gegeben.

2. Grundlagen

Die Weiterentwicklung von Saros erfordert die Kenntnis über die Funktionsweise und verwendeten Konzepte, sowie Technologien. Im Nachfolgenden wird erklärt was Saros ist, wie ein Sitzungsaufbau funktioniert und welche Konzepte dabei Anwendung finden.

2.1. Saros

Saros [3] ist ein quelloffenes Plugin für die Entwicklungsumgebung Eclipse, entwickelt auf Basis der Programmiersprache Java und dient der verteilten kollaborativen Programmierung. Es ermöglicht Gruppen von bis zu fünf Teilnehmern gemeinsam, ad-hoc und ortsunabhängig an einem Projekt zu arbeiten und gemeinsame Reviews durchzuführen, wobei jede Veränderung am Quellcode direkt allen Beteiligten verfügbar gemacht wird.
Grundbedingung zur Nutzung ist ein installiertes Eclipse mit installiertem Saros, ein XMPP-Account, sowie eine bestehende Internetverbindung zur Authentisierung des Accounts und eine Verbindung zu möglichen Teilnehmern.
Saros wird im Rahmen des Saros-Projektes stetig weiterentwickelt, so wird beispielsweise an der Portierung zur Nutzung von Saros auch auf anderen Entwicklungsumgebungen gearbeitet und die Robustheit des Plugins ständig verbessert.

2.2. Eclipse

Eclipse ist eine integrierte Entwicklungsumgebung (integrated development environment, IDE), basierend auf der Programmiersprache Java. Ursprünglich ausgelegt auf die Entwicklung von Java Applikationen bietet Eclipse inzwischen auch Unterstützung für zahlreiche andere Programmiersprachen und Funktionen für die Softwareentwicklung.

Es liegt quelloffen vor und läuft unter der „Eclipse Public License“ [4], weshalb es möglich ist Eclipse beliebig zu verändern, weiterzuentwickeln und als Grundlage für eigene Software zu verwenden und zu vertreiben.

2.3. Extensible Messaging and Presence Protocol

Das „Extensible Messaging and Presence Protocol“ (XMPP) [5] ist ein Standard zur Übertragung von Nachrichten im XML-Format und bietet zusätzliche Dienste. So nutzt Saros diese zur Kommunikation auf Netzwerkebene und Authentifizierung/Zuordnung von Benutzern in Form von XMPP-Accounts.

2.4. Sitzungsaufbau in Saros

Der Verlauf einer Sitzung besteht aus zwei Phasen [3]. Phase Eins, der Einladungsprozess, beginnt unabhängig vom Verfahren zum Austausch der Projektdaten immer auf die gleiche Weise und beschreibt die Rahmenbedingungen der Sitzung, wie teilnehmende Benutzer und das zu teilende Projekt.
Phase Zwei vollzieht den Austausch der Projektdaten.

Im Folgenden werden die Schritte eines Sitzungsaufbaus dargestellt [6].
Ein Benutzer initiiert einen Sitzungsaufbau und nimmt die Rolle des Hosts ein. Dazu wählt er einen Klienten und das zu teilende Projekt. Zu diesem Projekt werden nun Informationen wie Projektname, Dateiliste etc. zusammengestellt. Anschließend wird eine Einladung an den Klienten versendet. Nimmt der Klient an, kann er ein bestehendes Projekt als Grundlage wählen, oder ein Neues anlegen. Verwendung eines bestehenden Projektes erfordert die Löschung überflüssiger Dateien und die Auswahl fehlender oder ungleicher Dateien. Das Anlegen eines neuen Projektes benötigt alle Dateien des Hosts. Eine Liste aller benötigten Dateien wird an den Host zurückgeschickt. Dieser erstellt dann ein Zip-Archiv und sendet dieses an den Klienten.

Der Klient muss nun noch das Archiv entpacken und die Dateien integrieren, dann ist der Aufbau abgeschlossen. Abb.1 stellt diese Schritte kompakt dar.

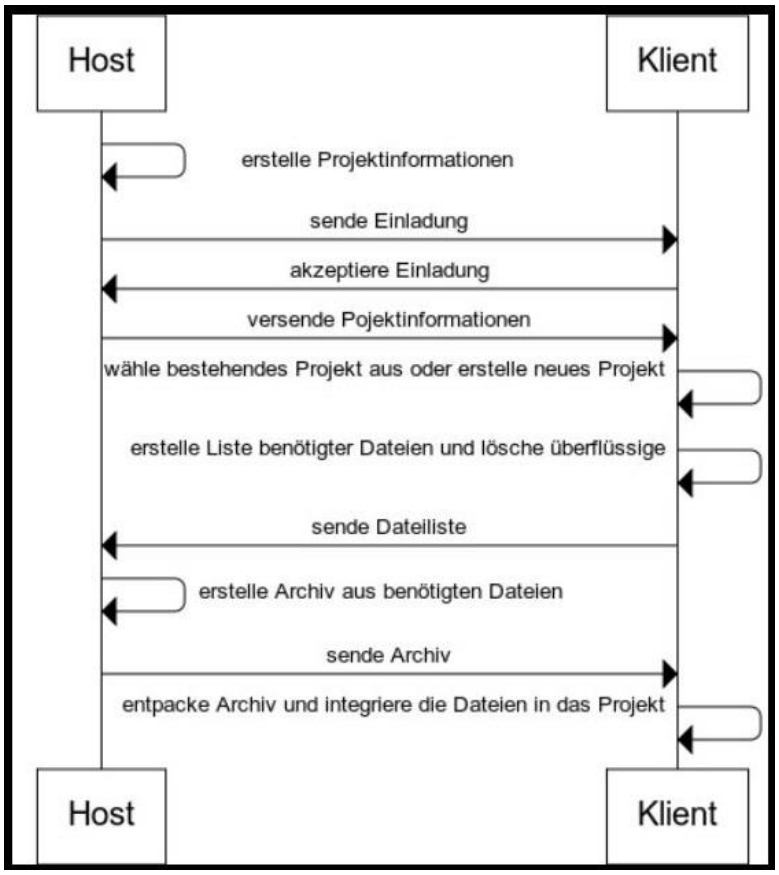

Abb. 1 – Prozessdiagramm des Sitzungsaufbaus

2.5. Kommunikationskanäle

Unterschieden werden in Saros zwei Kanäle zur Übertragung von Nachrichten. Der erste Kanal überträgt Nachrichten in Form von „Messages“, sie dienen der Organisation von Sitzungen und werden zur Ausführung von Aufgaben wie

Kompatibilitätsüberprüfungen, dem Abgleich kompatibler Saros-Versionen unter den Teilnehmern oder dem Einladungsvorgang von potentiellen Teilnehmern eingesetzt.
Der zweite Kanal überträgt Aktivitäten, welche den Zustand und Änderungen eines Projektes innerhalb einer laufenden Sitzung beschreiben. Veränderungen am Projekt werden somit den anderen Teilnehmern mittels Aktivitäten kenntlich gemacht.

2.5.1. Aktivitäten

Eine Aktivität beschreibt konkrete Veränderungen und Zustände am Projekt einer laufenden Sitzung. Sobald eine Aktivität bei einem Teilnehmer registriert wird, muss diese allen Teilnehmern gemeldet und übernommen werden. Aktivitäten gibt es in zahlreichen Varianten und Spezialisierungen. In den nachfolgenden Kapiteln werden FileActivities thematisiert.
Der Austausch von Aktivitäten erfolgt nebenläufig im Hintergrund und kann von dem Nutzer nur indirekt in Form von Darstellungs- und Codeänderungen wahrgenommen werden.

2.5.2. FileActivity

Bei FileActivities handelt es sich um spezielle Aktivitäten bezüglich Änderungen am Dateisystem. Dies umschließt das Anlegen neuer Dateien (Created), Verschieben (Moved) und Löschen (Removed) von Dateien. Erfolgt eine derartige Aktion bei einem Teilnehmer, wird die entsprechende Aktivität erstellt und an alle Teilnehmer gesendet. Zusätzlich existiert eine Spezialisierung, die RecoveryFileActivity, welche es erlaubt Änderungen an nur einen Teilnehmer zu senden. Kommt es zu Inkonsistenzen zwischen Teilnehmern können diese so gezielt behoben werden, ohne von allen Teilnehmern behandelt werden zu müssen.

2.6. Dummy File Creator

Um Tests mit Dateien beliebiger Größe durchführen zu können, werden diese mit Hilfe des Dummy File Creator generiert. Dieser ermöglicht die Generierung von Dateien, welche nahezu nicht komprimierbar sind, bestehend aus zufälligen Bytes.

3. Methoden

Die Mitarbeit im Saros-Projekt unterliegt festen Abläufen und Prozessen. Die Koordinierung erfolgt im Rahmen der Arbeitsgruppe „Software Engineering“ der Freien Universität Berlin. Die Entwicklung an Saros dient Studenten häufig als Grundlage für Abschlussarbeiten und unterliegt einer hohen Fluktuation unter den Mitgliedern des Teams. Da Teil dieser Arbeit auch Implementierungsarbeiten sind, folgen Erläuterungen zum typischen Ablauf einer Codeänderung.

Die Entwicklung von Saros wird nach der „Kontinuierlichen Integration“ [7] praktiziert. Dies bedeutet eine ständige Integrierung der laufenden Entwicklungen in das Endprodukt, im optimalen Fall tägliches Einfügen neuer Veränderungen durch die Entwickler. Unterstützt wird dieses Vorgehen in Form von automatisierten Tests der neuen Integrierungen und möglicher Codeprüfungen durch andere Entwickler (Reviews). Die Verwendung dieser Entwicklungsmethode vermeidet zusätzliche Integrierungsarbeiten gegen Ende eines Entwicklungszyklus und ermöglicht eine deutlich schnellere Veröffentlichung von neuen Produktversionen.
Organisiert wird der Saros Quellcode mithilfe der Versionkontrollsoftware für verteilte Projektentwicklung namens „Git“. Diese protokolliert Veränderungen, speichert, basierend auf Veränderungen, verschiedene Versionen des Quellcodes und erlaubt beliebiges Zurücksetzten und Zugriff auf ältere Versionen.
Um eine Änderung an Saros vornehmen zu können, muss zuerst der Quellcode von Git heruntergeladen werden. Wird anschließend eine Änderung durchgeführt, wird

diese wieder zurück zu Git hochgeladen. Eine derartige Änderung nennt sich Patch. Dies genügt jedoch nicht zur Integrierung in Saros. Auf Git setzt das Reviewsystem „Gerrit“ auf. Dieses bietet eine Weboberfläche und verschiedene Dienste. So listet es alle nicht integrierten Patches, wobei jeder neue Patch eine separate Nummer (ID) bekommt, anhand derer er identifizier- und aufrufbar ist. Auf diese Weise ermöglicht Gerrit die Begutachtung dieser und erlaubt zusätzliche Qualitätsicherungsmaßnahmen in Form weiterer Software.

Nach dem Hochladen eines Patches erfolgen mehrere Qualitätssicherungsprüfungen. Die Erste besteht aus automatisierten Tests des Codes, realisiert durch die Software „Jenkins“, sowie „Sonarqube“, welche mit Gerrit kombiniert arbeiten. Diese suchen nach Kompilierungsfehlern und der Einhaltung definierter Muster im Code.

Verlaufen diese Prüfungen erfolgreich, können die anderen Teammitglieder Gutachten über den Code erstellen. Erst wenn mindestens zwei Entwickler die Änderungen befürworten und keine Probleme am Code vorliegen, wird dieser Patch in Saros übernommen und die Integrierung ist abgeschlossen.

4. Die technische Ausgangssituation

Die Projektübertragung übernimmt die Aufgabe die Daten eines zu teilenden Projektes an alle Teilnehmer zu versenden und ist Teil des Sitzungsaufbaus, wie in Kapitel 2.4 beschrieben ist. Dieser Schritt ist notwendig, um eine gemeinsame Arbeitsgrundlage unter den Teilnehmern zu schaffen und stellt einen sehr zentralen Aspekt in Saros dar. Ausgangspunkt ist eine bereits übertragene Liste der benötigten Projektdateien.
Es stehen hierbei zwei Möglichkeiten eine Projektübertragung zu realisieren zur Verfügung. Diese werden im Folgenden beschrieben.

4.1. Projektübertragung per Zip-Datei

Der Sitzungsstart per Zip-Datei, nachfolgend als Zip-Variante bezeichnet, ist die bisher einzig in Saros enthaltene und vollständig implementierte Variante, um ein Projekt zu Beginn einer Sitzung zu übertragen.

Der Host beginnt eine Zip-Datei mit allen benötigten Dateien zusammenzusetzen. Hat der Host die Zip-Datei fertig gestellt, beginnt er mit der Übertragung an die Klienten. Nach Abschluss der Übertragung wird das Archiv von den Klienten entpackt und in ihren Workspace integriert.

Diese Variante erlaubt den Austausch beliebig großer Projekte. Während der Übertragung ist es dem Nutzer nicht möglich Eingaben zu tätigen, so werden Inkonsistenzen vermieden. Statt das Projekt bei jeder Sitzung komplett übertragen zu müssen, ist es den Klienten auch möglich, ein bereits Vorhandenes als Grundlage zu verwenden, so müssen nur noch fehlende Dateien übertragen und Überschuss gelöscht werden.
Prinzipiell ist es möglich mit dieser Variante Projekte zuverlässig auszutauschen. Dabei ergibt sich jedoch ein Problem, denn die Projektgröße nimmt massiven Einfluss auf die Zeit bis zur ersten möglichen Änderung am Projekt. Die Geschwindigkeit basiert auf

der eingesetzten Hardware, sowie auf der zur Verfügung stehenden Bandbreite in Abhängigkeit der Projektgröße und Art des Inhaltes. Unter ungünstigen Voraussetzungen können so sehr lange Wartezeiten entstehen, was vor allem im industriellen Sektor, wo der Zeitfaktor von großer Bedeutung ist, problematisch ist. So müssen Entwickler unter Umständen längere Zeit untätig zusehen bis der Projektaustausch vollzogen ist.
Eine weitere Einschränkung ergibt sich aus der gebündelten Übertragung. Im Falle einer Verbindungsunterbrechung während der Archiv-Übertragung kann der Klient die erhaltenen Daten nicht nutzen, da das Archiv nur teilweise und somit beschädigt übertragen wurde und nicht entpackt werden kann. In diesen Fällen ist eine vollständige Neuübertragung notwendig, was dieses Problem, gebündelt mit einer langen Wartezeit, verstärkt.
Diese Einschränkungen waren Anstoß um eine weitere Übertragungsform einzuführen.

4.2. Projektübertragung per FileActivity

Im aktuellen Saros-Release (Nummer: 14.10.31) befindet sich nur die Zip-Variante zur Projektübertragung. Eine Zweite wurde im Rahmen der Bachelorarbeit von David Damm entwickelt [1].

Diese zweite Variante verwendet eine Übertragung per Aktivitäten, eine detailgenaue Beschreibung findet sich in Referenz [1]. In diesem Fall beginnt der Host die benötigten Dateien der Reihe nach abzuarbeiten. Zu jeder Datei wird eine FileActivity erstellt und versendet. Diese Aktivität enthält Informationen zur Datei wie Name, Pfad im Projekt und den Dateiinhalt in Form eines Bytearrays. Nach Erhalt überprüft der Klient, ob die Datei bereits vorhanden ist. Falls ja wird geprüft, ob eine Ersetzung notwendig ist, ansonsten wird sie direkt in das Projekt integriert. So wird das Projekt dateiweise übertragen.
Dieses Vorgehen erlaubt die Beschränkungen der Zip-Variante zu überwinden. So ist es mit dieser Variante möglich, sofort nach Erhalt einer Datei Änderungen

vorzunehmen, ohne auf die vollständig abgeschlossene Projektübertragung zu warten, während der Host nun keinerlei Wartezeit zu absolvieren hat.

Die fehlende Integration der Aktivitäts-Variante beruht auf einer Vielzahl von Problemen. Das erste liegt in der Konzeption, denn diese Variante ist so konzipiert, dass sie die Bestehende ersetzt. Statt ergänzend zu agieren und den bestehenden Funktionsumfang zu erweitern, ersetzen die existierenden Patches Komponenten der bestehenden Variante.
Ein weiteres Problem liegt in der Übertragung großer Dateien. In Abhängigkeit vom Java-Heap führt die Übertragung einer großen Datei, und damit einer großen FileActivity, zu einem Speicherüberlauf und dem Absturz des Programms.
Weiterhin ist die Übertragung im Hintergrund problematisch. Während der Übertragung sind die Daten beim Host bereits verwendbar. Im Falle von Änderungen an Dateien, während diese Dateien noch nicht übertragen wurden, führt dies zur Ignorierung der Änderungen auf Seite der Klienten. Der Grund dafür ist, dass Änderungen an nicht vorhandenen Dateien nicht vorgenommen werden können. Dies resultiert in Inkonsistenzen.
Auch ist bekannt, dass das Hinzufügen von neuen Dateien in den Projektexplorer eine Blockierung des Benutzerinterfaces auslöst. Hierzu wurden von D.Damm im Nachgang an seine Ausarbeitung bereits Schritte unternommen, brachten allerdings einen Bug in Eclipse zum Vorschein, welcher unvorhersehbar zu einem Datei Ping Pong führen kann, einem ständigen hin- und herkopieren von einer Datei zwischen den Benutzern.

4.3. Lösungsansatz

Der vorliegende Ansatz fordert eine Entscheidung zugunsten einer Variante. In Anbetracht der bereits bewiesenen Einsatzbarkeit der Zip-Variante erscheint eine Ersetzung durch die Aktivitäts-Variante als höchst unwahrscheinlich. So wäre es zwingend notwendig sämtliche Defizite zu beheben, selbst dann müsste eine funktionierende Methode einer Ungeprüften weichen, nur um diese in der Praxis testen zu können.

Um dieses Problem zu lösen ist das erste Ziel eine Koexistenz zu ermöglichen und per Schalter eine Auswahlmöglichkeit zu schaffen. So muss keine Funktionalität eliminiert werden und eine Weiterentwicklung der Activity-Variante wird per kontinuierlicher Integration ermöglicht.

Im zweiten Schritt müssen die Probleme der Acitvity-Variante weiter eingegrenzt und behoben werden. Da die Übertragung großer Dateien zu einem Programmabsturz führt, sollte diese zuerst behoben werden. Ein Vorteil der Behebung des Problems ist zusätzlich, dass dieses auch beim Hinzufügen einer Datei während einer laufenden Sitzung auftreten kann und die generelle Robustheit, auch abgesehen von der Projektübertragung, erhöht wird.

Die Lösung der übrigen Probleme ist vermutlich gekoppelt, da der Datei Ping Pong Effekt erst durch die Möglichkeit zur nebenläufigen Übertragung ausgelöst wird. Eine mögliche Lösung bedarf weiterer Recherche und macht eine zeitliche Kalkulierung nicht möglich. Aus diesem Grund werden diese Problem am niedrigsten priorisiert.

5. Integrierung beider Übertragungsvarianten

5.1. Schalter zur Variantenauswahl

Die Möglichkeit beide Varianten ohne Änderungen am Quellcode testweise zu nutzen, erfordert einen Mechanismus zum Wechsel und Zuschalten der Varianten. Die Anforderung ist hierbei, dem Tester/Nutzer zu erlauben kontrolliert eine gewünschte Variante festlegen zu können, wobei als Standard die bisher genutzte Zip-Variante voreingestellt bleiben soll. So muss ein Benutzer über keine weiteren Kenntnisse verfügen und kann Saros wie bisher benutzen. Weitere Anforderungen gibt es nicht, weshalb viel Freiraum zur Umsetzung besteht. Da es sich hierbei nur um einen Zwischenschritt handelt fiel die Entscheidung auf einen möglichst simplen Entwurf und somit schnelle Umsetzung.
Um dem Tester eine Konfiguration von Außen zu ermöglichen, erscheint die Verwendung einer Konfigurationsdatei als naheliegend. Eine derartige Datei wird vom Saros-Plugin bereits verwendet und muss zu diesem Zweck lediglich um einen Eintrag erweitert werden. Dieser Eintrag regelt in Form einer Booleschen Variable die Aktivierung der neuen Activity-Variante und wird in Abbildung 2 dargestellt.

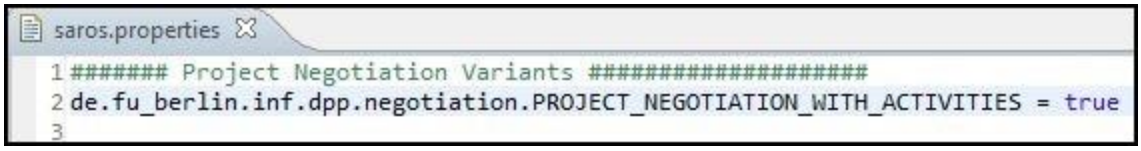

Abb. 2 , Konfigurationsdatei - saros.properties

Der Schaltmechanismus wird dagegen in den Programmcode integriert. Im Grunde eine if-else-Abfrage, welche den Eintrag der Konfigurationsdatei überprüft. Da die Rollen der Teilnehmer während der Übertragung in „Host“ und „Klienten“ aufgeteilt werden, befindet sich im Quellcode eine ähnliche Unterscheidung und Modularisierung in „Eingehenden“ und „Ausgehenden“, weshalb der Schalter auf beiden Seiten

eingefügt werden muss. Eine If-Bedingung prüft hierbei, ob ein anderes Verfahren genutzt werden soll, also die Activity-Variante, oder den Standard und somit Zip-Variante im else-Block. Auf diese Weise könnten auch weitere Verfahren sehr einfach hinzugefügt, kombiniert und priorisiert werden und erfordern lediglich zusätzliche/abgeänderte if-Bedingungen. Abbildung 3 zeigt den Schalter für den Ausgehenden Part, der Eingehende ist analog dazu implementiert.

```
if (PROJECT_NEGOTIATION_WITH_ACTIVITIES) {
    return new ProjectNegotiationActivityOutgoing(peer, session,
        projects, sarosContext);
} else {
    return new ProjectNegotiationZipOutgoing(peer, session, projects,
        sarosContext);
}
```

Abb. 3, Variantenschalter für die Ausgehende Projektübertragung

Zwar ist ein Schaltmechanismus nun vorhanden, da die Zip-Variante aber nach wie vor als fester Bestandteil im Programmcode vorliegt, ist es im nächsten Schritt notwendig diese Abzukapseln.

5.2. Kapselung der Zip-Variante

Wie beim Schalter, so existieren auch für die Durchführung der Variante separate Implementierungen für die Seite des Hosts und der Klienten, in Form der ausgehenden „OutgoingProjectNegotiation“ (Host), als auch Eingehender „IncomingProjectNegotiation“(Klient) Klassen. Abbildung 5 zeigt die vorliegende Klassenhierarchie inklusive der Methode zur Ausführung der Variante.

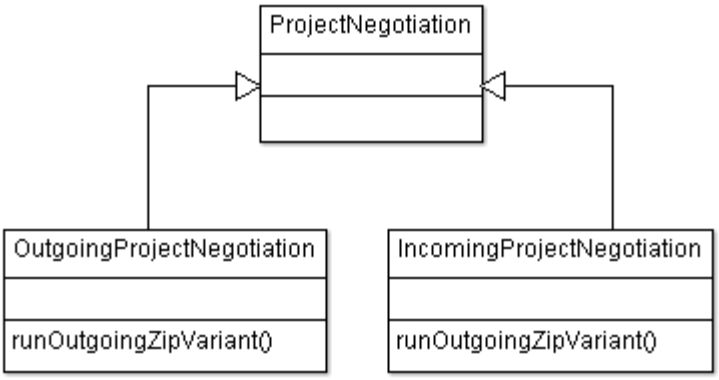

Abb. 5, Klassenhierarchie nach Integrierung beider Übertragungsvarianten

Eine Abkapselung erfordert die Umstrukturierung dieser Klassen.
Dazu werden beide in jeweils eine Oberklasse umgewandelt und dienen als Sammelklasse der von allen Varianten verwendeten Methoden und grundsätzlich benötigter Funktionen. Beispielsweise müssen unabhängig der verwendeten Übertragungsvariante die zu versendenden Dateien ermittelt werden, enthalten aber nicht mehr die konkrete Ausführung der Übertragung. Daneben werden die der Zip-Variante spezifischen Methoden und der Ausführungsblock dieser Variante als Unterklasse implementiert. In der Folge ergeben sich so neben den beiden Oberklassen pro Übertragungsvariante ebenfalls zwei Unterklassen, welche die „Outgoing/IncomingProjectNegotiation“ erweitern.
Übersichtlicher in Bezug auf die Klassenhierarchie wäre die Zusammenfassung zu einer Oberklasse, wodurch für jede Variante nur eine Unterklasse notwendig wäre. Allerdings sind diese Klassen bereits sehr komplex und die Überschneidungen, abseits von Methodennamen, sehr gering, weshalb eine zusammengefasste Klasse letztendlich doch in zwei Bereiche getrennt werden müsste und so erheblich komplexer und schwerer zu lesen wäre.

Die Resultate sind die veränderten Klassen „Outgoing/IncomingProjectNegotiation“, sowie die neueingeführten Klassen „IncomingZipProjectNegotiation“ und „OutgoingZipProjectNegotiation“. Auf die gleiche Weise könnten nun auch neue Varianten integriert werden und müssen lediglich dem Schalter bekannt gemacht werden.
Es ist jetzt also möglich durch den Schalter die Standard(Zip)-Variante zu verwenden, als auch die Aktivitäts-Variante zu aktivieren, wobei diese noch nicht existent ist und bei Auswahl einen Abbruch (Cancel) des Übertragungsprozesses auslöst.

5.3. Integrierung der Basis der Activity-Variante

Der Prototyp beinhaltet bereits große Teile der Activity-Variante, so ist auch die Basisfunktionalität, also eine Übertragung in Form von Aktivitäten, bereits vollständig vorhanden. Sie muss nun extrahiert und gekapselt werden.
Die Umsetzung erfolgt analog zur Zip-Variante. Hierzu werden zwei neue Klassen als Subklassen der „Outgoing/IncomingProjectNegotiation“ eingeführt und alle benötigten Methoden ebenfalls in die Unterklassen implementiert. Im nächsten Schritt wird dann überprüft, welche Überschneidungen mit der Zip-Variante existieren und entsprechende Methoden in die Oberklassen verschoben. Daraus ergibt sich eine veränderte Klassenhierarchie. Abbildung 5 zeigt die Klassenhierarchie mit Ort der Startmethode vor der Umstrukturierung, Abbildung 6 zeigt die veränderte Hierarchie nach der Umstrukturierung.

,

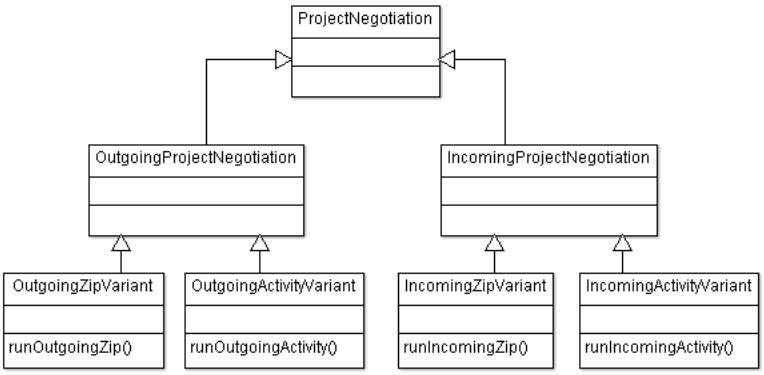

Abb. 6, die veränderte Klassenhierarchie nach der Integration beider Übertragungsvarianten.

5.4. Übertragung großer Dateien

Um eine neu erstellte Datei per Aktivität zu übertragen, wird eine FileActivity erzeugt. Diese wird mit dem gesamten Inhalt der Datei und einigen Zusatzinformationen ausgestattet. Wird dieser Inhalt zu groß, kommt es während der Umwandlung der Daten in das XML-Format zum Speicherüberlauf. Eine Lösung zur Vermeidung von diesem Problem ist die Aufteilung einer Datei auf mehrere Aktivitäten. So kann im Vorfeld eine Grenze des maximalen Inhalts einer FileActivity festgelegt werden und im Falle einer Überschreitung auf eine oder mehrere zusätzliche Aktivitäten verteilt werden.

Der naive Ansatz zur Lösung ist eine einfache Schleife, welche statt den ganzen Inhalt einer Datei einzulesen und in eine FileActivity zu verpacken, in jedem Durchlauf nur einen Teil, höchstens die festgelegte maximale Größe der FileActivity, einliest und diesen mit einer neuen Aktivität abschickt. Hierbei trat das erste Problem auf. Die bestehende Logik versendet eine FileActivty, prüft ob der Inhalt bereits vorliegt und überschreibt im Falle von Abweichungen eine bereits vorhandene Datei oder erstellt eine Neue. Überprüft wird dieser Inhalt durch Abgleich der Bytedarstellungen der Dateien. Ein solcher Abgleich funktioniert bei aufgeteilten Dateien nicht, da die Teile

keine Informationen über den Rest der Datei enthalten. Verschiedene Teile einer Datei würden sich somit immer direkt nacheinander ersetzen.
Eine Lösung erfordert einen umfangreicheren Eingriff. Da der Austausch der Daten über die FileActivity erfolgt, sollte diese mit weiteren Informationen bestückt werden. Dazu zählt die Information, ob die Aktivität der Anfang einer Datei ist und ein Teil einer aufgeteilten Datei. Hierzu wurde diese Zusatzinformation mit in die FileActivity übernommen.
Nun war es möglich auf Seite des Empfängers eine aufgeteilte Datei wieder zusammenzusetzen. Dazu wurde überprüft, ob eine FileActivity den Anfang einer Datei enthält, sollte eine Datei gleichen Pfades bereits vorliegen, wurde diese einfach Ersetzt, da so nicht prüfbar ist, ob die komplette Datei der vorliegenden entspricht, denn nachfolgende Teile könnten Abweichungen enthalten.
Um diesen Prozess weiter zu optimieren, wäre es möglich vor Versendung der Datei ihre Prüfsumme zu berechnen und jeder zugehörigen Aktivität mitzugeben. Bei Empfang müsste im Falle einer bereits existierenden Datei ebenfalls diese Prüfsumme berechnet und abgeglichen werden um festzustellen, ob eine Ersetzung notwendig ist.
Während der Veränderung der FileActivity stellten sich einige Fragen. Diese Aktivität wurde so konzipiert, dass sie mehrere Zustände darstellen kann. So ist es möglich neu erstellte, verschobene oder gelöschte Dateien zu repräsentieren. Diese Zustände werden mittels statischer Methoden erreicht, welche die Parameter des Klassenkonstruktors entsprechend anpassen. Ein derartiges Vorgehen bringt allerdings einige Nachteile mit sich. Beim Empfang einer FileActivity muss sehr häufig eine Fallunterscheidung durchgeführt werden, da nicht direkt klar ist, welcher Zustand vorliegt. Außerdem enthält die Klasse sämtliche Methoden und der Klassenkonstruktor die Parameter aller Zustände, obwohl diese je nach Fall nicht gebraucht werden, was den Nebeneffekt einer unnötig komplexen Klasse und eine gewisse Fehleranfälligkeit durch nicht durchgeführte Überprüfungen mit sich bringt.
Aus diesen Gründen wurde entschieden, die Klasse der FileActivity zu überarbeiten.

5.6. Überarbeitung der FileActivity

Die Klasse der FileActivity enthält drei statische Methoden zum Aufruf des Konstruktors mit festgelegten Parametern, um so zwischen drei Varianten von FileActivities zu unterscheiden. Hinzu kommt eine Kindklasse, die RecoveryFileActivity, welche ebenfalls zwei statische Methoden zur Verfügung hat und zwei weitere Formen der FileActivity repräsentiert. Die Folge sind Fallunterscheidungen im Programmfluss, wobei es beispielsweise nur um die Behandlung einer einzelnen Variante geht. Gleichzeitig sorgt dieser Aufbau für Unklarheiten und eine Intransparenz, sowie unnötig komplexe Klassen.

Eine Überarbeitung, durch Aufteilung in separate Klassen, der FileActivity würde die Lesbarkeit des Programmcodes deutlich erhöhen und gleichzeitig Erweiterung der FileActivity-Varianten vereinfachen, da nicht mehr alle Varianten betroffen wären. Dadurch könnte auch die notwendige Anpassung der FileActivity aus Kapitel 5.5. zur Versendung großer Dateien vereinfacht werden.

Zu diesem Zweck wird die FileActivity in ein Markierungsinterface umgewandelt, um eventuelle Prüfungen, ob überhaupt eine FileActivity vorliegt, weiterhin einfach zu halten. Zu jeder existierenden Variante wird dann eine separate Klasse angelegt. Diese enthält nur die für die Variante benötigten Methoden und Eigenschaften. Zusätzlich werden Empfangs- und Versandmethoden für Klient und Host angepasst und alle Fallunterscheidungen entfernt. Statt auf Fallunterscheidung wartet der Empfänger auf die spezialisiert Aktivität und muss nur im Empfangsfall auf diese reagieren.

6. Vergleichstests der Übertragungsvarianten

6.1. Testfälle

Es ist nun möglich beide Varianten zu nutzen und auszuwählen. Mithilfe von Testfällen soll gezeigt werden, unter welchen Umständen auf eine bestimmte Variante zurückgegriffen werden sollte und ob die offenen Punkte der Activity-Variante einen negativen Einfluss zeigen. Zur Konstruktion der Testfälle bedarf es einiger Vorüberlegungen.

Einflüsse auf die Projektübertragung können eingegrenzt und auf konkrete Ursachen und Bedingungen der Umgebung zurückgeführt werden. So bestimmt die zugrunde liegende Ausstattung, wie schnell Prüfsummen berechnet, sowie Daten erfasst, komprimiert und verschickt werden können. Die Dauer einer Übertragung hängt in erster Linie von der zur Verfügung stehenden Bandbreite ab und das zugrunde liegende Projekt bestimmt die Menge an zu verteilenden Daten. Der Nutzer nimmt nur indirekt Einfluss, indem er das Projekt auswählt, kann aber die Eigenschaften und Umstände nicht manipulieren. Unter gleichbleibenden Umgebungsbedingungen, der Nutzung der gleichen Ausstattung und Verbindung, ist es allein das ausgewählte Projekt, welches für die Dauer einer Übertragung entscheidend ist.

Die Wahl der Testumgebung beruht in erster Linie auf persönlichen Mitteln und ist deshalb eingeschränkt. Zur Verfügung stehen ein Laptop mit einem Halbleiterlaufwerk (Solid State Drive, SSD), 16 Gigabyte (gb) Hauptspeicher (Random-Access-Memory, RAM) und einem 4-Kern Intel-Prozessor i7 4770, sowie ein Desktop-PC mit einer magnetischen Festplatte, 8 gb RAM und einem AMD-FX-6100 6-Kernprozessor. Beide befinden sich in einem lokalen Netzwerk (Local Area Network, LAN) und verfügen über einen Internetzugang.

Da die Rahmenbedingungen aufgrund dieser Testumgebung festgesetzt sind, wird in den Tests der Einfluss unterschiedlicher Projekte ermittelt. Hierbei werden Projekte mit unterschiedlichem Inhalt, in Form verschieden großer Dateien, Anzahl von Dateien und Gesamtgröße eingesetzt.

Betrachtet wird die vollständige Übertragung der kompletten Projekte, als auch die Zeitspanne bis zur ersten möglichen Änderung am Projekt im Falle der Activity-Variante.

6.2. Erwartungen

Die Activity-Variante erfasst eine Datei als Bytearray. Ist der Inhalt dieser Datei kleiner als ihre Rahmendaten, dann transportiert eine Aktivität mehr Rahmendaten als den tatsächlichen Dateiinhalt. Die Folge sind ein größerer Aufwand der Erstellung und Versendung von Aktivitäten als die tatsächlich übertragene Nutzlast, auch als Overhead bezeichnet, und eine deutlich erhöhte Übertragungszeit des gesamten Projektes. Übersteigt die Dateigröße diese Grenze, das heißt die Nutzlast ist höher als die Rahmendaten, bedeutet ein linearer Anstieg von Daten auch einen linearen Anstieg von benötigten Aktivitäten. Optimalfall sind hier vermutlich eine permanente Vollauslastung der Aktivitäten, diese liegt bei 2 Megabyte (mb), was dann Maximalgeschwindigkeit entsprechen kann.
Da diese Variante erlaubt bereits während der Übertragung Änderungen am Projekt vorzunehmen, ist es nicht zwingend notwendig auf die Übertragung des gesamten Projektes zu warten. Die Zeitspanne bis zur ersten möglichen Änderung wird, entsprechend der Ergebnisse von D.Damm, als konstant angenommen.
Die Zip-Variante führt während der Erstellung einer Zip-Datei eine Kompression des Inhaltes durch. Je besser sich der Inhalt komprimieren lässt, umso kleiner ist die Größe der Zip-Datei. Im schlimmsten Fall ist der Inhalt der Dateien nicht komprimierbar und führt zu einer Größe der Zip-Datei, welche dem der Gesamtgröße der Originaldateien entspricht, in diesem Fall ist der gesamte Komprimierungsvorgang unnötig und führt zu einem unnötigen Overhead. Dieser schlechteste Fall bedeutet die gleiche Menge an zu übertragenen Daten beider Varianten, denn die Zip-Datei wäre genauso groß wie der Inhalt der Aktivitäten und erlaubt einen direkten Vergleich. Zeigt sich, dass die Zip-Variante an diesem Punkt schneller ist, wird sie vermutlich immer schneller sein, da sich die Bedingungen nur noch zu Gunsten dieser Variante entwickeln können.

6.3. Testdurchführung

Aufgrund der schnellen Verbindung im LAN umschließen die getesteten Projektgrößen einen Umfang von 64 bis zu 256 mb, um einen Effekt messen zu können.
Zu jeder Größe werden Tests mit unterschiedlichem Inhalt, in Form verschieden großer Dateien durchgeführt. Hierbei kommen Dateien der Größe 128 Kilobyte (kb), 256 kb, 512 kb, 1 mb, 2 mb und 4 mb zum Einsatz.
Die Dateien werden zufallsgeneriert. Hierzu wird auf das Programm „Dummy File Creator“ zurückgegriffen. Es ermöglicht die Generierung von Dateien mit zufälligem Inhalt, in Form zufälliger Bytes im Bereich 0 bis 255. Die Folge sind Dateien mit möglichst schlecht komprimierbarem Zustand, wodurch eine Annäherung an den schlimmsten Fall der Zip-Variante erreicht wird.
Gemessen werden die benötigten Zeiten von Hand mittels Stoppuhr in Sekunden und einem Messfehler von einer Sekunde. Der händische Ansatz beruht auf der Tatsache, dass es zusätzlichen Aufwand bedeutet hätte eine maschinelle Variante zu entwickeln und manuelles Vorgehen eine höhere Flexibilität erlaubt.
Um Unterschiede aufgrund der eingesetzten Arbeitsmittel feststellen zu können, wird der Sitzungsaufbau ausgehend vom PC, sowie vom Laptop durchgeführt und separat gemessen. Jeder Testfall wird zusätzlich dreimal durchgeführt und anschließend der Mittelwert gebildet, so können Ausreißer und Ungenauigkeiten eingegrenzt werden.

Die Tests sollen zeigen, ob eine Abhängigkeit zwischen Geschwindigkeit und Dateigröße, sowie der Dateianzahl besteht. Hierbei bleibt die Projektgröße in einer Testreihe die gleiche und entspricht 64 Mb. Begonnen wird mit einer Dateigröße von 128 Kb, was 512 Dateien im Projekt bedeutet. Im nächsten Schritt, werden die Dateien durch 256 Kb große Dateien ausgetauscht, die Dateianzahl reduziert sich somit auf 256 Dateien im Projekt. Dieses Vorgehen wird solange wiederholt bis kein Effekt mehr gemessen wird oder das Projekt nur noch aus einer einzelnen Datei besteht.
Analog werden Tests mit anderen Projektgrößen bis zu 256 Mb absolviert.

7. Diskussion der Testergebnisse und Ausblick

Die folgende Abbildung 6 zeigt die gemessenen Werte bei der fünfmaligen Übertragung eines 256 Mb großen Projektes vom PC zum Laptop mit der Zip-Variante. Zum Vergleich zeigt Abbildung 7 die Übertragung des gleichen Projektes mit der Activity-Variante.

Diese beiden Abbildungen stehen exemplarisch für alle durchgeführten Messungen, da die Tendenzen immer die gleichen sind. Die übrigen Messungen sind dem Anhang beigelegt.

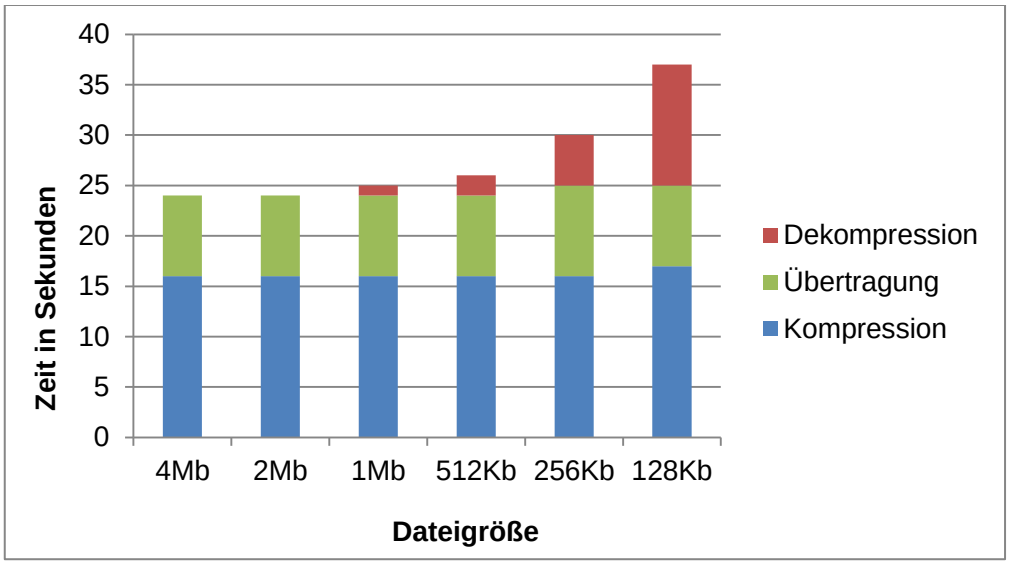

Abb. 6, Übertragung eines 256 Mb Projektes per Zip-Variante basierend auf unterschiedlichen Dateigrößen

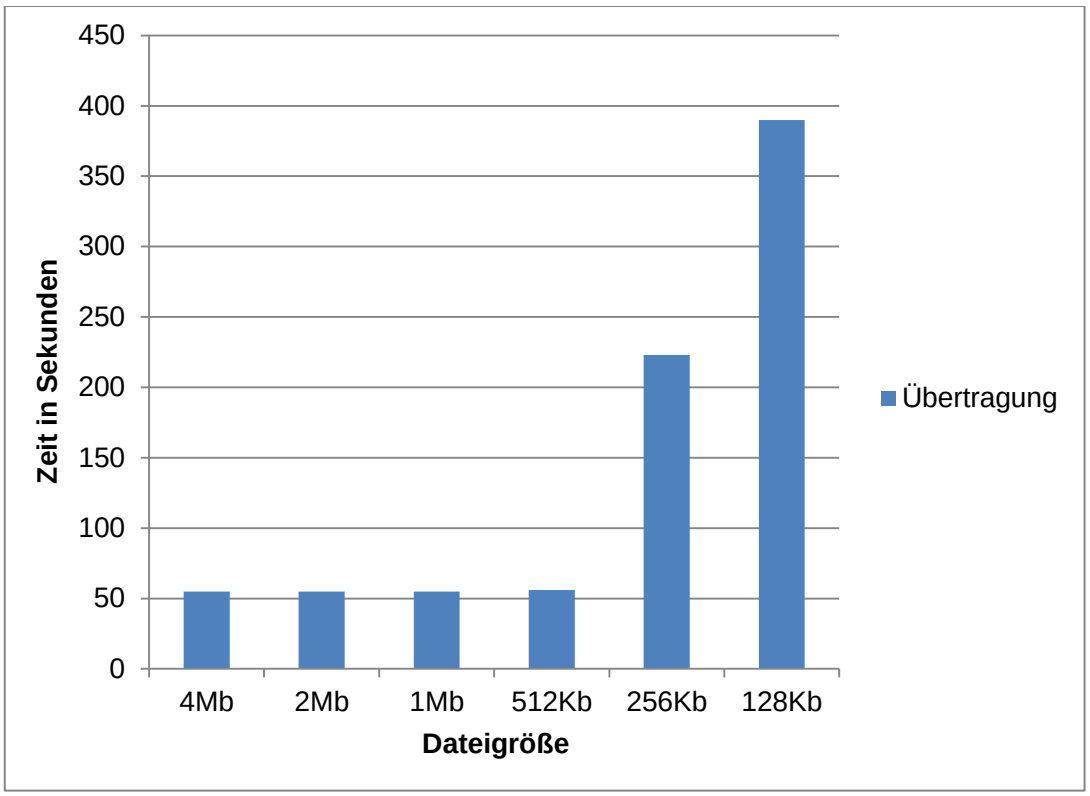

Abb. 7, Übertragung eines 256 Mb Projektes per Activity-Variante basierend auf unterschiedlichen Dateigrößen

7.1. Interpretation der Testergebnisse

Anhand der Testergebnisse lassen sich einige Eigenschaften der Varianten ableiten.
Die Zip-Variante besteht aus drei Phasen. Die erste ist die Kompression und Erstellung der Zip-Datei. Erstaunlich ist hier, dass die Kompression bei einer gleichbleibenden Projektgröße nahezu immer gleich bleibt und unabhängig von der Anzahl der Dateien im Projekt ist. Die reine Übertragungsdauer bleibt in diesen Fällen ebenfalls identisch, ist jedoch auf den Kompressionsgrad und damit der letztendlichen Größe der Zip-Datei zurückzuführen, welche aufgrund der zugrundeliegenden Dateien immer in etwa identisch ist, denn der Aufbau der Dateien sorgt für eine sehr niedrige Kompressionsrate. Der einzige Unterschied in der Gesamtübertragung wird bei der Dekompression, dem entpacken der Zip-Datei deutlich, denn je größer die Dateien

werden, umso schneller erfolgt das entpacken. Hier ist vermutlich der Grad an Kompression entscheidend. Zum einen muss jede vorhanden Datei einzeln entpackt und in den Workspace integriert werden, was aufwendiger ist als mit einer einzelnen großen Datei. Der Grad an Kompression spielt vermutlich auch eine Rolle, so benötigt eine höhere Komprimierung mehr Rechenleistung zum Entpacken, dafür wird allerdings Zeit während der Übertragung gespart, was sich letztendlich ausgleicht. Vergleicht man die Zeiten der Übertragung von unterschiedlich großen Projekten, zeigt sich ein nahezu linearer Anstiegt. Abweichungen sind auf Ungenauigkeiten der Messungen und kleinen Kompressionsrate zurückzuführen. Somit ist die benötigte Zeit extrapolierbar.

Während sich die Zip-Variante grundsätzlich linear bei Änderungen verhält, zeigen sich bei der Activity-Variante andere Zeichen. Das Verschieben von sehr vielen kleinen Dateien, beispielsweise im Testfall die Verschiebung von 2024 Dateien mit je 128 kb, zeigt eine überraschend hohe Wartezeit von mehreren Minuten und fällt mit größer werdenden Dateien massiv ab. Allerdings auch nur bis zum Erreichen von 1mb großen Dateien, anschließend stagniert der Wert der Übertragungsdauer. Im konkreten Fall des Projektes mit 256 mb, fiel die Zeitdauer beim Wechsel von 2024 Dateien mit jeweils 128 kb auf 256 Dateien mit je 1 mb im Extremfall von 590 Sekunden auf 50 herunter. Bedenkt man, dass Quellcodedateien häufig noch erheblich kleiner sind, ist davon auszugehen, dass sich diese Variante nicht für kleine Projekte mit großer Anzahl an Dateien eignet. Die Stagnation der Geschwindigkeitszunahme ab 1 mb großen Dateien, ist insoweit interessant, dass die Grenze der aufnehmbaren Nutzlast einer FileActivity bei 2 mb liegt. Die Vergrößerung einer FileActivity hat somit keinen Nutzen mehr und der Geschwindigkeitsanstieg bei größer werdenden Dateien verläuft linear, so genügen FileActivities mit bis zu 1 mb großem Bytearray.

Während der Durchführung konnten Teile der bekannten Probleme der Activity-Variante festgestellt werden. Nahezu bei jedem Durchgang kam es zu einem Datei Ping Pong. Allerdings fällt dieser einem Nutzer vermutlich nicht auf. Der einzige Hinweis ist ein kurz eingeblendeter Schriftzug über eine eintreffende Resource, was von einem Nutzer als normal aufgefasst werden könnte. Nur auf der Konsole, mit derer

die Tests simuliert wurden, war diese Phänomen deutlich sichtbar. Außerdem konnten Inkonsistenzen durch Arbeiten an noch nicht übertragenen Dateien erreicht werden.
Es konnten jedoch auch die Vorteile registriert werden, so war der Host immer arbeitsbereit und wurde durch die Verwendung dieser Variante nicht eingeschränkt, beim Verschicken vieler kleiner Dateien konnten nur gelegentliche Unterbrechung in der Benutzeroberfläche bemerkt werden. Der Klient war nach Ablauf einiger Sekunden arbeitsbereit, sofern die Übertragung nicht vorher beendet war. Dies deckt sich mit der Erkenntnis der Prototypentests.
Ein Einfluss der grundlegenden Ausstattung der verwendeten Computer zeigte nur in Extremfällen der Activity-Variante merkliche Unterschiede, ansonsten lagen die Unterschiede im Bereich weniger Sekunden und können vernachlässigt werden.

7.2. Vergleich der Varianten und Ausblick

Im direkten Vergleich schneidet die Zip-Variante generell besser ab. So erfolgt die Übertragung eines kompletten Projektes immer schneller als durch die Activity-Variante und es konnten, abgesehen von der Wartezeit bis zur vollständigen Übertragung, keine Probleme registriert werden.
Der Vorteil der Activity-Variante während der Übertragung arbeiten zu können, ist nur dann ein Vorteil, wenn die benötigte Datei auch frühzeitig, zu Beginn, übertragen wird. Erfolgt die Versendung einer benötigten Datei erst gegen Ende der gesamten Übertragung, wäre die Zip-Variante vermutlich bereits abgeschlossen. In Kombination der Abhängigkeit der Activity-Variante von der Dateigröße der Projektdaten, kann dies sogar extreme Ausmaße annehmen, indem die Übertragungsdauer extrem steigt und die benötigte Datei unter den letzten zu Übertragenen ist. Eine Möglichkeit diesen Punkt zu optimieren wäre die Einführung einer Priorisierungsmöglichkeit, beispielsweise Quellcode-Dateien zuerst zu übertragen oder eventuell konkret benötigte Dateien durch den Benutzer auswählen zu lassen.
Das Problem des Datei Ping Pong ist deutlich problematischer. Durch Reviews des Teams wird ein Bug in Eclipse als mögliche Ursache des Problems vermutet.

Möglichkeiten zur Vermeidung müssen gezielt recherchiert werden und benötigen unter Umständen eine lange Zeit bis zur Lösung, da dies nicht einschätzbar ist.
Allerdings bietet die Activity-Variante bereits jetzt einen riesigen Vorteil, denn der Abbruch einer Übertragung bedeutet keinen kompletten Datenverlust. Alle vollständig übertragenen Dateien können genutzt werden und durch die Funktion ein bestehendes Projekt zu nutzen, kann die Übertragung an genau dieser Stelle wieder aufgenommen werden, wogegen die Zip-Variante an dieser Stelle völlig versagen würde, da keine der übertragenen Daten nutzbar wären.
Wie sollten die Varianten nun eingesetzt werden?
Die Zip-Variante zeigt einen linearen Anstieg im Falle größer werdenden Daten, das heißt eine mögliche Zeitdauer zur Übertragung kann im Voraus kalkuliert bzw. nach Versendung eines kleineren Testpakets und Messung der Geschwindigkeit auch extrapoliert werden. Da durch die Prüfsummenberechnung auch alle Dateien und ihre Größe bekannt sind, wäre es denkbar ein Projekt vor der Übertragung zu analysieren und im Falle einer absehbar sehr langen Zeitspanne der Übertragung und günstiger Dateigröße für die Activity-Variante, diese auch einzusetzen, da Verlust durch Verbindungsabbruch und die Chance frühzeitig die Arbeit beginnen zu können einen Vorteil bieten können. Unter diesen Umständen wäre eine Kombination denkbar sonst überwiegt aufgrund der seltener auftretenden Probleme und größeren Unabhängigkeit der Vorteil der Zip-Variante. Um abzuwägen, ob eine weitere Entwicklung dieser Activity-Variante sinnvoll ist, wäre eine Evaluation, wie schwerwiegend die bestehenden Probleme sind, ob Optimierungen möglich sind und wie häufig Fälle der Projektformen mit günstigen Dateigrößen für die Activity-Variante in der Praxis auftreten, nötig.

8. Zusammenfassung und Fazit

Zum Abschluss der Arbeit erfolgt eine kurze Zusammenfassung der erreichten Ziele, sowie der offenen Punkte und eine persönliche Einschätzung der Arbeit am Saros-Projekt.

Im Rahmen dieser Arbeit wurde die Möglichkeit geschaffen, Übertragungsvarianten nebeneinander in Saros nutzen zu können, wobei diese so angelegt wurde, dass auch noch weitere Varianten, im Falle von Verwerfungen oder neuen Ansätzen, unproblematisch hinzugefügt und entfernt werden können. Die Fähigkeit Dateien im Bedarfsfall aufzuteilen und so stückweise zu verschicken, hat nicht nur die Activity-Variante verbessert, sondern war gleichermaßen eine Verbesserung der Robustheit von Saros, da auch neu hinzugefügte Dateien, innerhalb einer laufenden Sitzung, von dieser Verbesserung betroffen sind. Dies war so auch Auslöser für die Überarbeitung der FileActivity Klasse, wodurch eine bessere Übersicht im Code der Klassen gewährt, als auch die Fehleranfälligkeit durch große Fallunterscheidungen zukünftig vermieden wird. Der Test der verschiedenen Varianten konnte klare Eigenschaften filtern und Situationen aufzeigen in denen die Varianten eher ungeeignet sind, als auch gut angewendet werden können.

Das große Problem der Implementierungsarbeiten ist, dass sie sich alle noch in der Review-Phase befinden. Aufgrund der unter Umständen langen Zeit bis zum Eintreffen eines Reviews vergehen mitunter Wochen ohne einen Patch weiter bearbeiten zu können, da kein Feedback als Grundlage existiert. So wurde beispielsweise der Schalter für die Varianten bisher nicht einmal gereviewed, was, da er die Grundlage für die Varianten ist, auch diese bremst. So ist ein vollständiges Integrieren erst im Nachgang dieser Arbeit möglich. Daten der betreffenden Patches befinden sich im Anhang, dem Kapitel 11.1

Außerdem war es zeitlich nicht möglich, alle bekannten Probleme der Activity-Variante zu beheben, so sind die niedrig priorisierten Punkte nach wie vor offen.

Die generelle Arbeit im Saros Team habe ich als sehr angenehm empfunden. Auch wenn der Tonfall teils sehr direkt war, verlief die Arbeit recht sachlich und Fragen sowie

Probleme konnten häufig innerhalb kürzester Zeit aus dem Weg geräumt werden. Häufig waren Gespräche und Feedback die Inspiration für neue Lösungsansätze.
Dennoch gab es einige Hürden. So ist die Einarbeitung in den bestehenden Code und alle verwendeten Unterstützungswerkzeuge, trotz Kenntnis von Git und Eclipse, sehr umfangreich gewesen. Auch waren die Ansprüche an den geschriebenen Code ungewohnt hoch und nicht vergleichbar mit geforderten Arbeiten im alltäglichen Studiumsablauf.
Im Nachhinein betrachtet, war das Aufsetzen auf eine direkt vorlaufende Bachelorarbeit auch problembehaftet. So musste die Planung und Zielsetzung während der laufenden Arbeit mehrfach umgeändert werden, da sich auch die Voraussetzungen geändert haben.
Alles in allem möchte ich die Erfahrung der Arbeit an einem Projekt wie Saros nicht missen, so habe ich viel über die Koordinierung eines großen Projektes und generelle Softwareentwicklung gelernt.

9. Literatur

[1] Ian Sommerville, Software Engineering 9th. Ed, Addison Wesley, 2011

[2] David Damm, Schnellerer Sitzungsstart in Saros, Bachelor's thesis, Freie Universität Berlin, 2015

[3] Patrick Schlott, Analyse und Verbesserung der Architektur eines Nebenläufigen und verteilten Softwaresystems, Master’s thesis, Freie Universität Berlin, 2015, Url: http://www.inf.fu-berlin.de/inst/ag-se/theses/Schlott13-saros-concurrency.pdf

[4] Eclipse Foundation, Eclipse Public License - v.1.0, Url: https://eclipse.org/org/documents/epl-v10.php, zuletzt eingesehen am 04.11.2015

[5] XMPP Standards Foundation, About, http://xmpp.org/about-xmpp/, zuletzt eingesehen am 04.11.2015

[6] Saros-Project, Invitation, http://www.saros-project.org/invitation, zuletzt eingesehen am 04.11.2015

[7] Martin Fowler, Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html, zuletzt eingesehen am 04.11.2015

10. Anhang

10.1. Gerrit-Patches – Git Commits

Im Folgenden werden alle laufenden Patches aufgelistet. Zur Orientierung werden die Aufgabe, Patchnummer und Commit-Nachricht mit angegeben.

10.1.1. Aufteilung großer Dateien: Patch 2874

[INTERNAL][FIX] Marshalling large files without heap space error
This commit splits a new added file (if necessary) into several FileActivities. Afterwards the receiver puts them together to restore the file.
locally tested with:
- one 400mb file
- two 100mb files (folder)
- twenty 10mb files (folder)
Needs to be tested in real situations. Change-Id:
If7ced88f3cf741737964911c09325980d89a44fb

10.1.2. Umstrukturierung der FileActivity: Patch 2905

[API] FileActivities - creating separate classes
This change turns the FileActivity into a marker-interface and creates for all different types of FileActivities own classes.
Advantages:
- less complex classes
- decreased number of necessary parameters for each Activity
(removed purpose + type)
- simplified specialization of each type
- easier to distinguish
Disadvantages:

- bloat IActivityReceiver/AbstractActivityReceiver interfaces
- increased complexity of the FileActivity hierarchy
Change-Id: I82f0de7d0ad8452719dca09eebe55752da2beaab

10.1.3. Integrierung der Activity-Variante: Patch 2926

[API][FEATURE] second project negotiation variant

This patch integrates the basis of a second project negotiation variant.
It was already introduced in Patch 2094/8 (also the disadvantages) and uses FileActivities for file transmission. The difference to this patch is that the feature toggle (Patch 2910) allows to use this, without removing the existing negotiation variant and to switch to the preferred variant by configuring the saros.properties file, so it's possible to improve this negotiation step by step. Most changes done by this patch are copying methods from the zip-negotiation into the superclass, cause both variants use them. As already said, this is just the basis and has some limitations, for example: - the user has only read access during transmission and can't write anything (has to wait until transmission is complete) - works only with small projects (specific size depends on the heapsize -> some mb, no files > 5mb should normally work - inconsistency caused by changes during transmission is possible In the end it should be possible to work during the transmission is done in the background.

Actual benefits:
- negotiation starts immediately after checksum calculation (no zip/unzip)
- transmission of parts, not just one zip
-> connection loss during transmission not necessarily corrupt the whole transmitted data

Change-Id: I5cdd862417ecd716b0cb253cd097dfe5e829be83

10.1.4. Schaltmechanismus und Kapselung der Zip-Variante: Patch 2910

[API][FEATURE] feature toggle for project negotiation This change introduces a feature toggle for the project negotiation, controlled by the saros.properties file. So we are allowed to implement new negotiation variants into the masterbranch and testing them in the running system without changing consisting variants/functionality or using separate branches. This change implements the toggle into the SarosSessionManager. The Incoming/OutgoingProjectNegotiation classes are formed into abstract classes and work as parents for negotiation variants. The negotiation execution is extracted into subclasses. This causes a lot of changes, but mostly method movements or changes on the visibility without creating new lines. So the only new thing is the toggle itself and the changed class hierarchy. A new negotiation variant will follow in separate changes, so actually there is only the already existing zip variant.

Advantages: - possible coexistence of several negotiation variants - simplified adding of new variants - allows negotiation testing during development - reduced complexity of the Incoming/OutgoingProjectNegotiation classes

Disadvantages: - the toggle must be actually configured on both sides (client/host) - blows the class hierarchy - every negotiation variant needs separate incoming/outgoing parts - testing during development causes (normally) limitations on new introduced negotiation variants

Change-Id: Idfc30107692cb0570213f6fb86a70d52d236aacc

10.2. Messergebnisse

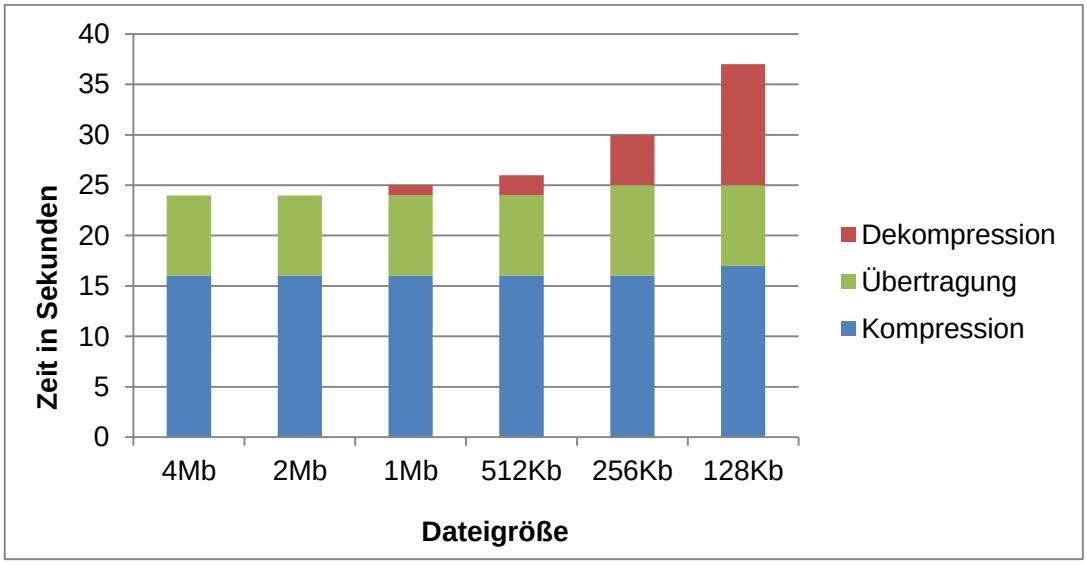

Abb. 1, Übertragung eines 256 mb Projektes per Zip-Variante vom PC zum Laptop

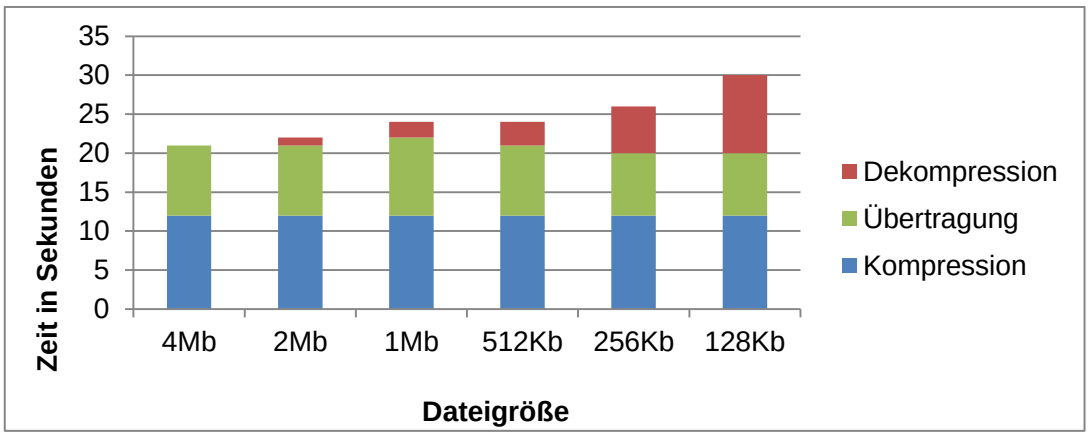

Abb. 2, Übertragung eines 256 mb Projektes per Zip-Variante vom Laptop zum PC

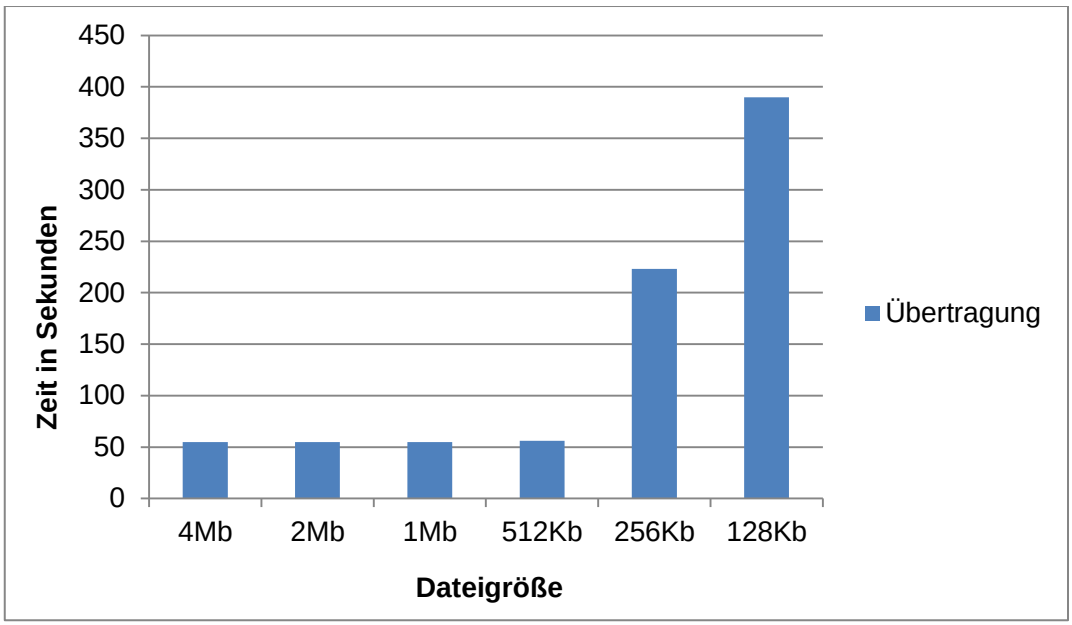

Abb. 3, Übertragung eines 256 Mb Projektes per Activity-Variante vom PC zum Laptop.

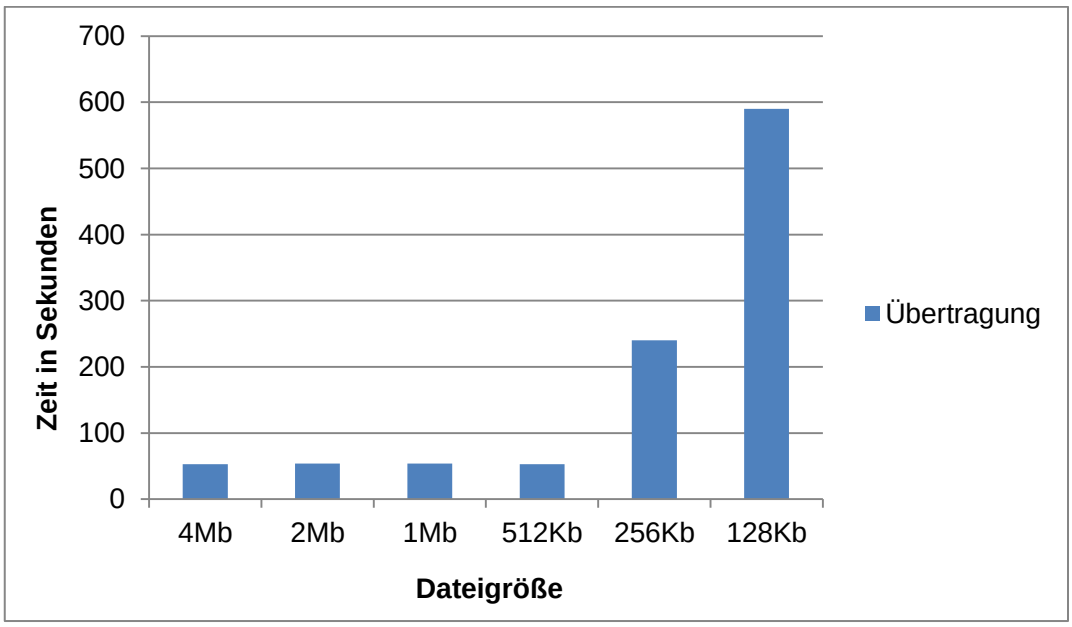

Abb. 4, Übertragung eines 256 Mb Projektes per Activity-Variante vom Laptop zum PC.

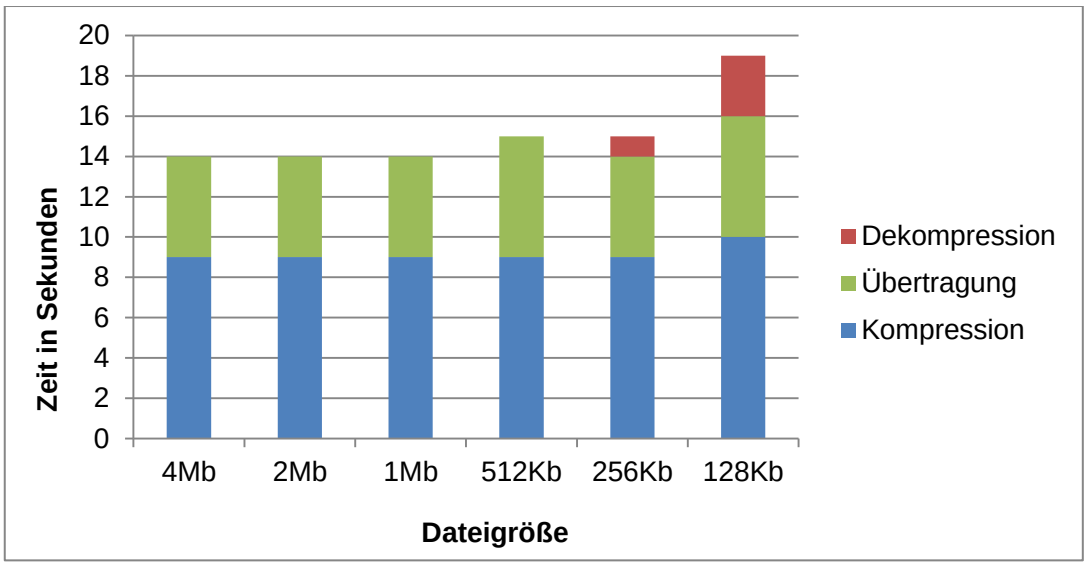

Abb. 5, Übertragung eines 128 Mb Projektes per Zip-Variante vom PC zum Laptop

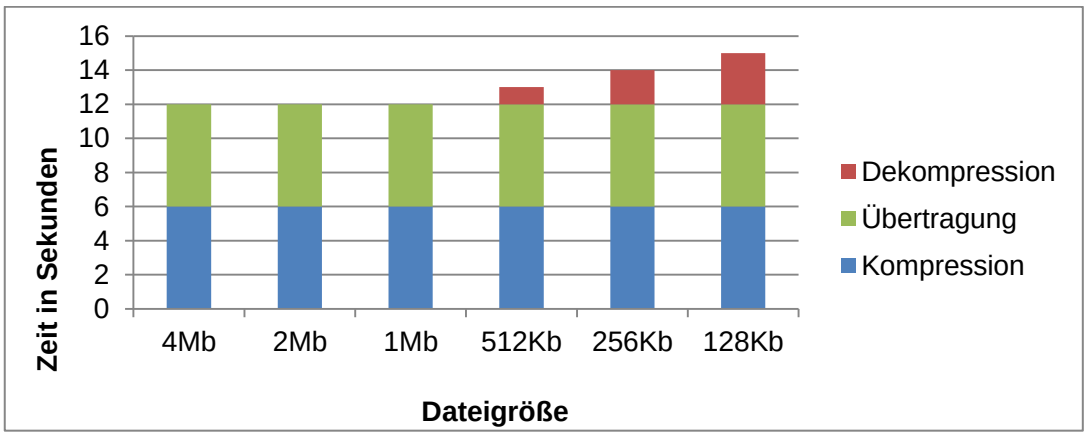

Abb. 6, Übertragung eines 128 Mb Projektes per Zip-Variante vom Laptop zum PC

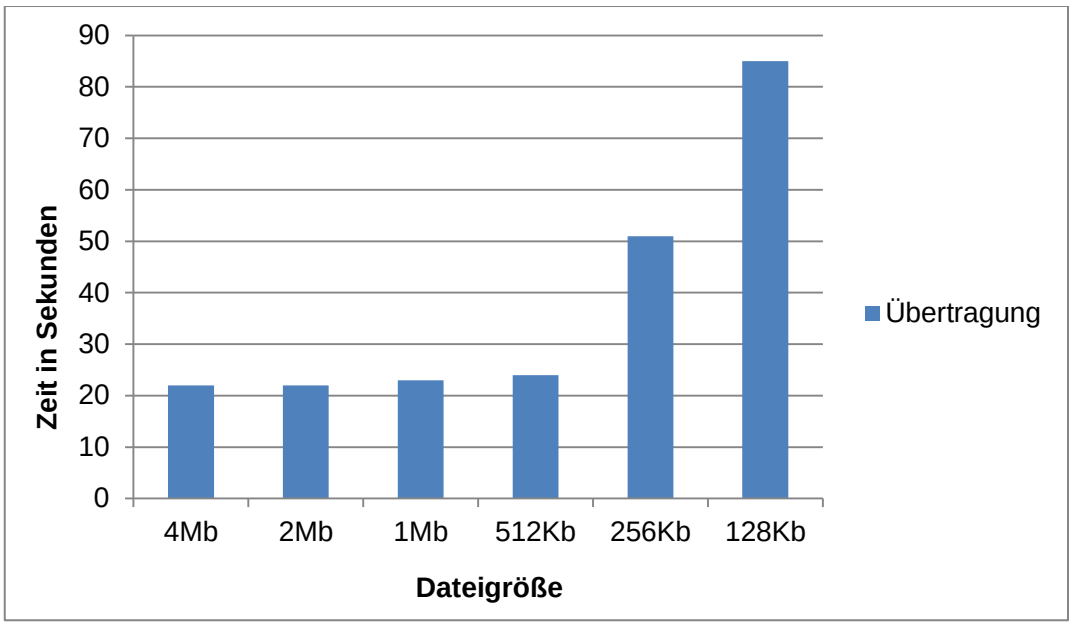

Abb. 7, Übertragung eines 128 Mb Projektes per Activity-Variante vom PC zum Laptop.

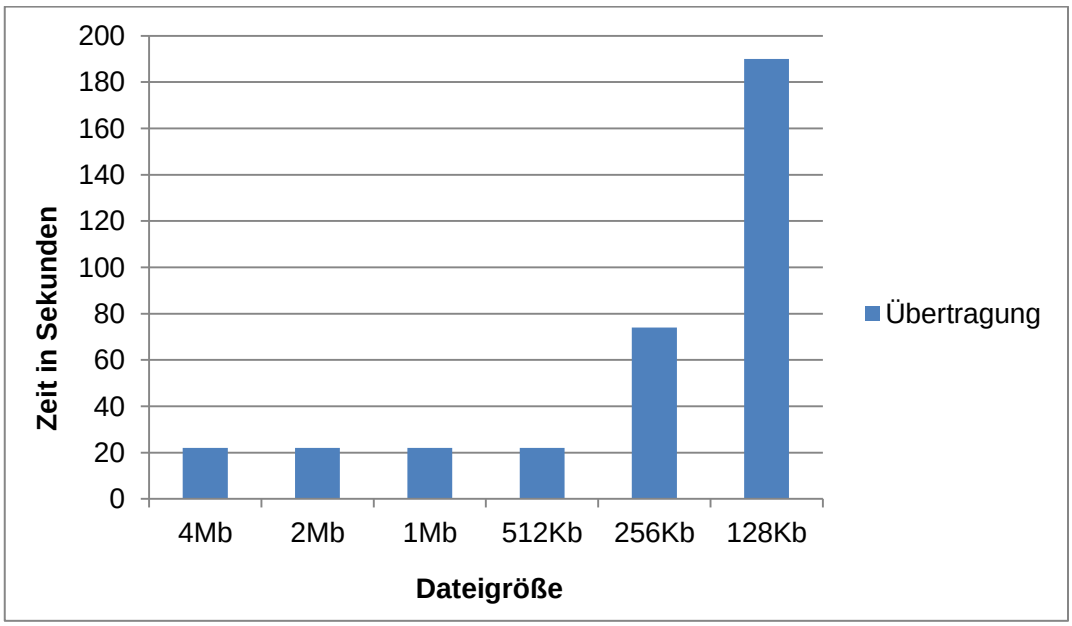

Abb. 8, Übertragung eines 128 Mb Projektes per Activity-Variante vom Laptop zum PC.

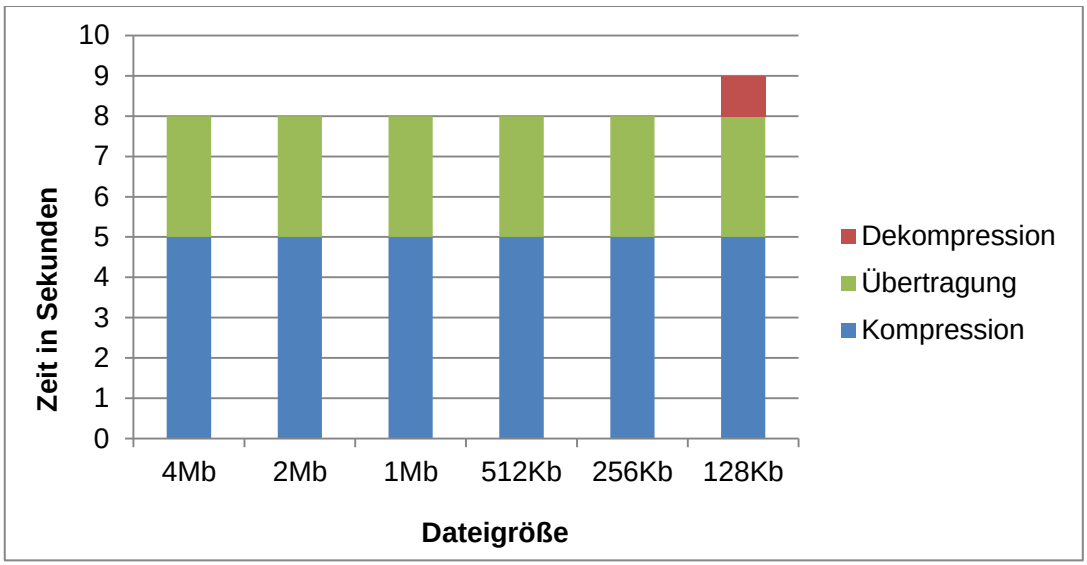

Abb. 9, Übertragung eines 64 Mb Projektes per Zip-Variante vom PC zum Laptop

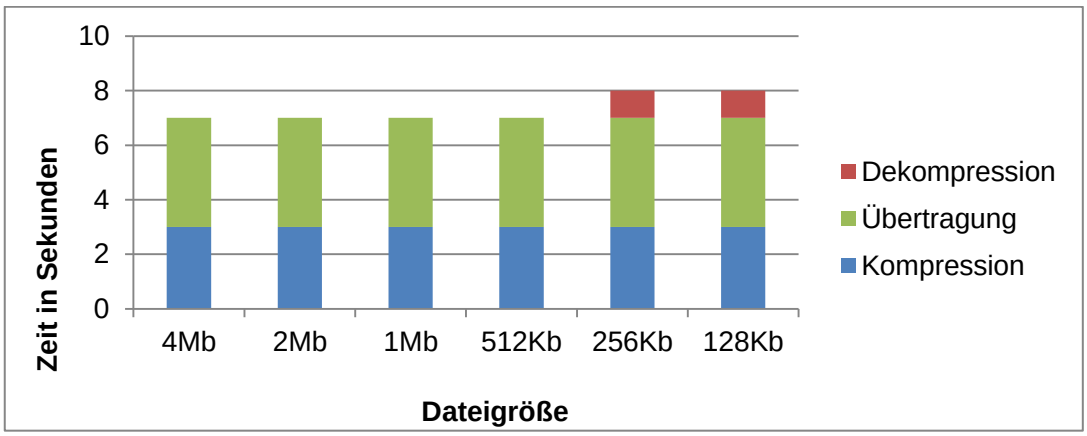

Abb. 10, Übertragung eines 64 Mb Projektes per Zip-Variante vom Laptop zum PC

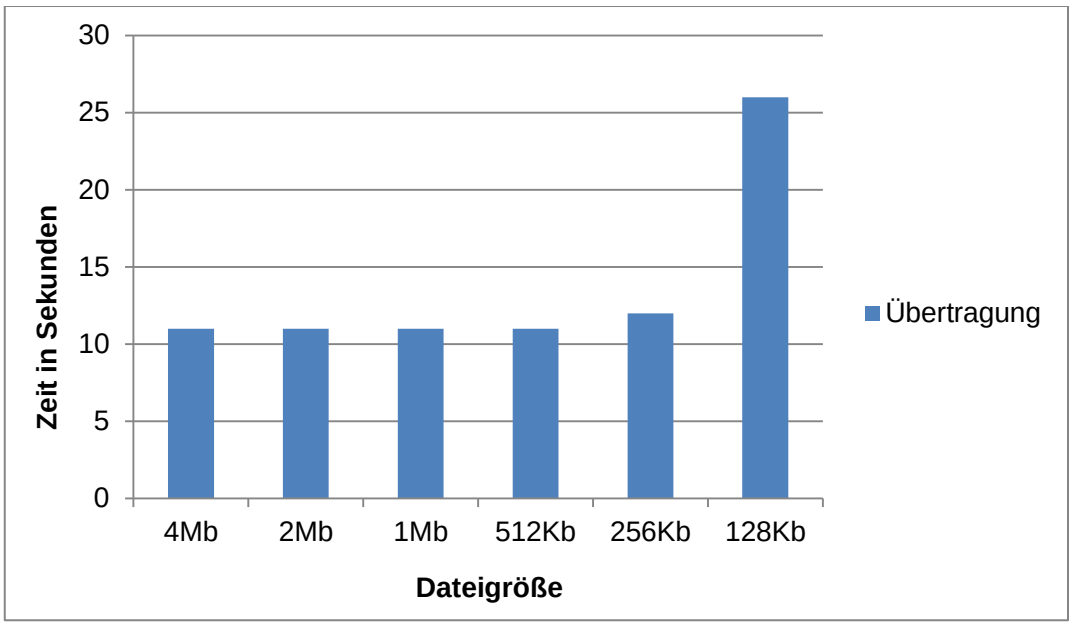

Abb. 11, Übertragung eines 64 Mb Projektes per Activity-Variante vom PC zum Laptop.

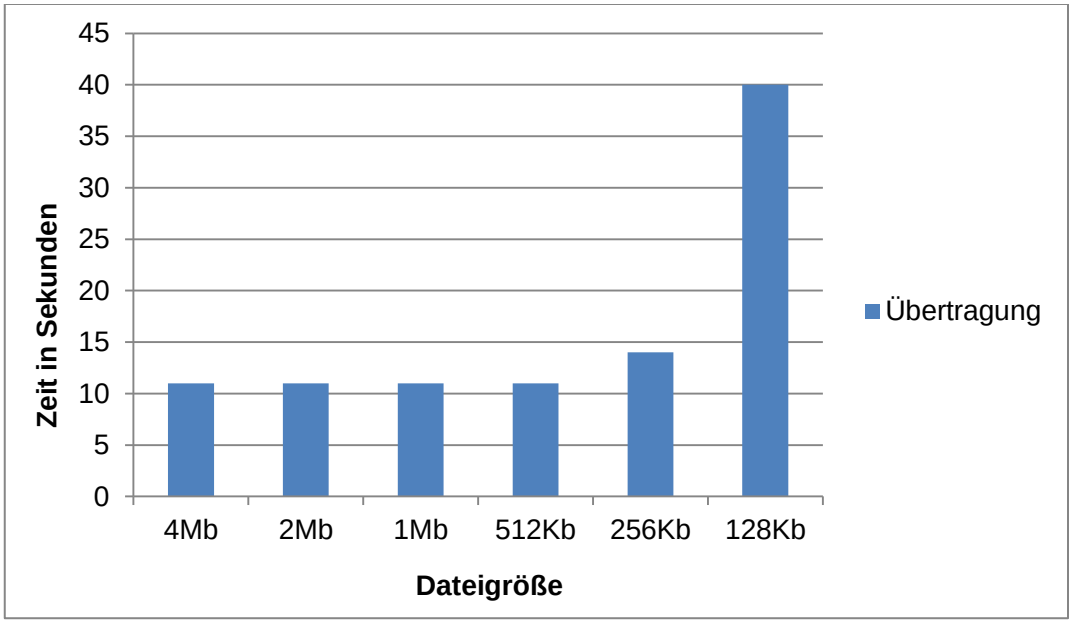

Abb. 12, Übertragung eines 64 Mb Projektes per Activity-Variante vom Laptop zum PC.