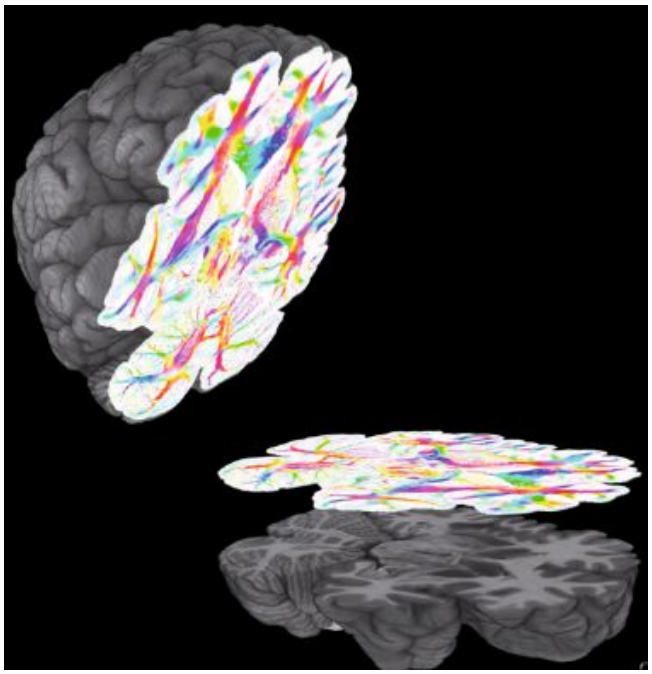
Konzeption und Implementierung eines Software-Deployment Systems für Analysesoftware in den Neurowissenschaften



Christian Krause
Matrikelnummer 1956616
Heinrich-Heine Universität Düsseldorf
Mail: christian.krause@hhu.de

Bachelor of Science
Erstgutachter: Dr. Markus Axer
Zweitgutachter: Prof. Dr. Axel Görlitz
Betreuer: Dipl. Inf. (FH) Tim Hütz

Dormagen, den 12. Januar 2015

Inhaltsverzeichnis

1 Einleitung und Motivation

1.1 Zusammenfassung

Innerhalb der Arbeitsgruppe Faserbahnarchitektur im Institut für Neurowissenschaften und Medizin 1 (INM-1) des Forschungszentrum Jülich werden zur Analyse neuroanatomischer Daten eigenentwickelte Programme herangezogen. Die Programme werden von Einzelpersonen oder kleinen Arbeitsgruppen programmiert und dabei fortlaufend weiterentwickelt. Als Programmiersprache werden primär C, Python oder Cython genutzt. In Abhängigkeit von der aktuellen Problemstellung bzw. der Menge an zu verarbeitendem Datenmaterial wird der Programmcode auf unterschiedlichen Rechnern ausgeführt. Dazu gehören neben dem Arbeitsplatzrechner auch Rechenknoten sowie der High Performance Computer JuDGE, bereitgestellt durch das Jülich Supercomputing Center. Der bisherige Ablauf eines Softwareentwicklers bestand darin, jede veröffentlichte Programmversion auf jedem Zielrechner auszurollen, d.h. den Programmcode auf den Maschinen manuell mit dem Repository abzugleichen und den Kompiliervorgang zu starten. Ziel der Arbeit ist die Entwicklung und Bereitstellung eines Programmes, welches diesen Vorgang automatisiert.

1.2 Einleitung

Die Bachelorarbeit wurde innerhalb des Institut für Neurowissenschaften und Medizin 1 (INM-1) des Forschungszentrum Jülich durchgeführt. Der Institutsbereich "Strukturelle und funktionelle Organisation des Gehirns" des INM-1 entwickelt eine dreidimensionale Karte der Nervenfaserbahnverbindungen in Gehirnen verschiedener Spezies. Diese von Nagetieren-, Primaten- und Menschen-Gehirnen erstellte Karte wird auch Konnektom genannt.[1] Auf den Prozess zur Erstellung eines Konnektoms soll im Folgenden näher eingegangen werden.

Im ersten Schritt wird das Gehirn mit Formalin präpariert und anschließend eingefroren. Nun wird das Hirn in etwa 60 μm dünne Scheiben geschnitten. Vor jedem Schnitt wird ein Aufsicht-Bild (*Blockface*) des Gehirns angefertigt, sowie ein Auflicht-Bild des zugehörigen Schnittes. Aus den Unterschieden zwischen Blockface und Auflicht-Bild lassen sich später aus dem Schnittvorgang entstehende Deformationen herausrechnen.

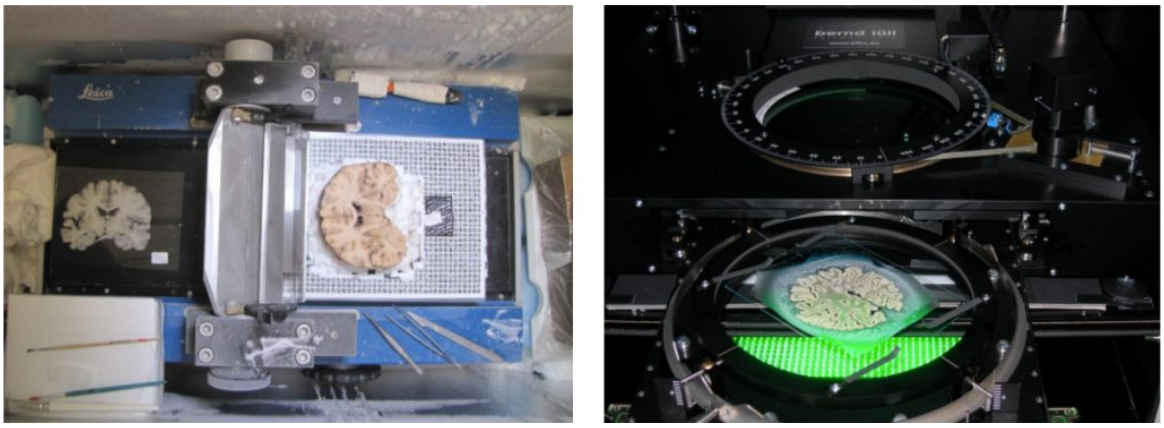


Abbildung 1: Links: Mit dem Mikrotom wird das Gehirn in Scheiben geschnitten, Rechts: Polarimeter zur Digitalisierung der Schnitte, Quelle: M. Huysegoms[2]

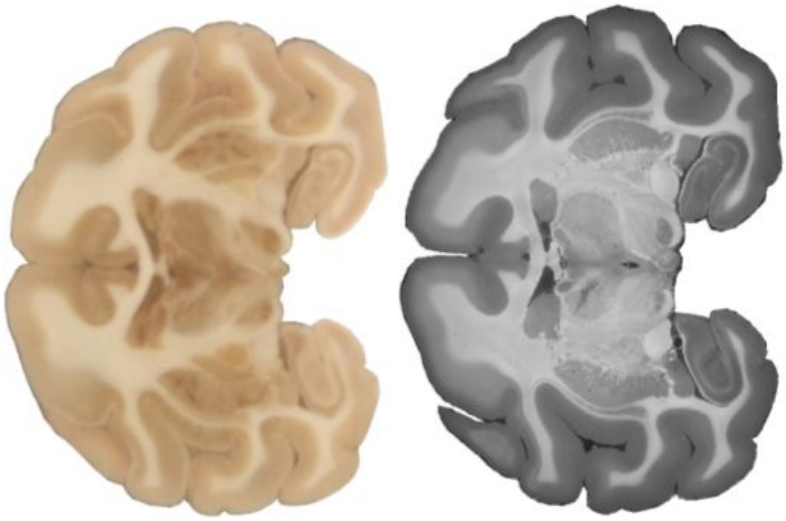


Abbildung 2: Blockface (links) und Auflicht-Bild des Schnittes (rechts), Quelle: M. Huysegoms[2]

Der zweite Schritt besteht in der Bestimmung der Raumrichtung der Nervenfaserbahnverbindungen. Zentrale Bedeutung für diesen Vorgang ist die uniaxiale[1], doppelbrechende Eigenschaft der die Nervenzellfortsätze umgebende Myelinscheide. Hierfür wird der Hirnschnitt im Polarisator in 10°-Schritten um insgesamt 180° gekippt. Die Helligkeiten einzelner Bildbereiche ergeben über alle Aufnahmen ein sinusförmiges Intensitätsprofil, aus welchem sich die lokale Richtung der Nervenfaserbahnen für jeden Bildbereich berechnen lässt. Auf die konkrete Berechnungsmethode soll in dieser Einleitung nicht weiter eingegangen werden.

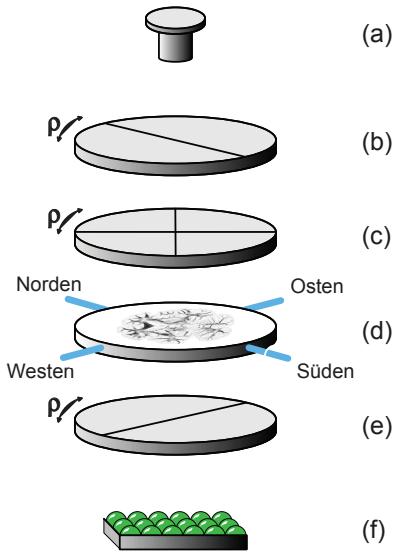


Abbildung 3: Aufbau: f) Lichtquelle, e) linearer Polarisator, d) kippbare Gewebebühne, c) $\frac{\lambda}{4}$-Verzögerungsplatte, b) linearer Polarisator, a) Kamera, Quelle: Diplomarbeit T. Hütz

[1] in *eine* Raumrichtung

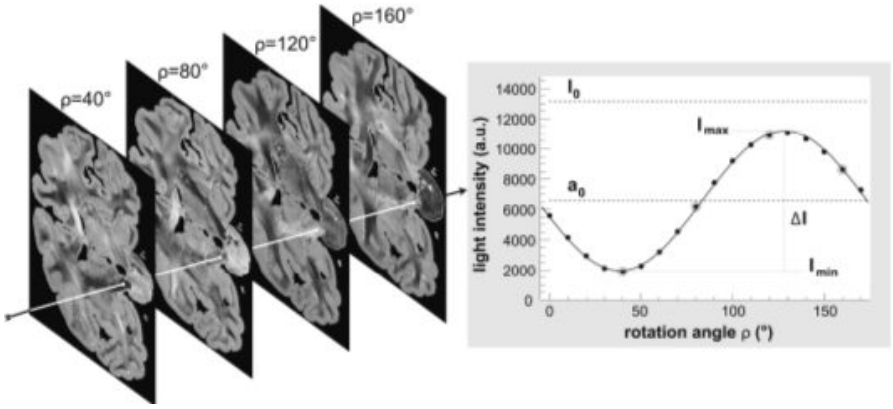


Abbildung 4: Aufnahme eines Hirnschnitt unter verschiedenen Winkeln, Quelle: M. Huysegoms[2]

Wird jeder Raumrichtung der Nervenfaserbahnen in einem Hirnschnitt eine spezifische Farbe im Bild zugewiesen, ergibt sich eine Fiber Orientation Map (FOM).

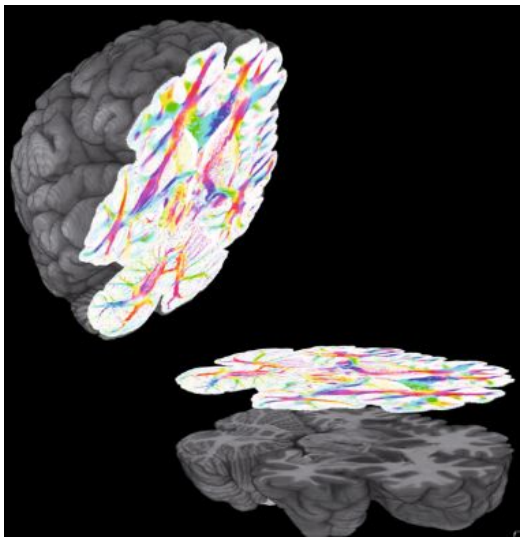


Abbildung 5: Künstlerische Aufbereitung einer Fiber Orientation Map, Quelle: M. Axer

Der dritte Schritt besteht in der Erweiterung der FOMs auf drei Dimensionen. Es reicht an dieser Stelle nicht aus, die Schnittbilder in der korrekten Reihenfolge übereinanderzulegen, um ein 3D-FOM zu erhalten. Die Deformationen des Schnittprozesses müssen analysiert und korrigiert werden, um Verzerrungen bei der Rekonstruktion zu vermeiden. Für diesen Prozess wird das *Blockface*-Bild genutzt. Der Prozess wird Registrierung genannt. Dieser Registrierung widmet sich die Arbeit von M. Huysegoms, aus der viele der hier vorgestellten Bilder entnommen sind[2].

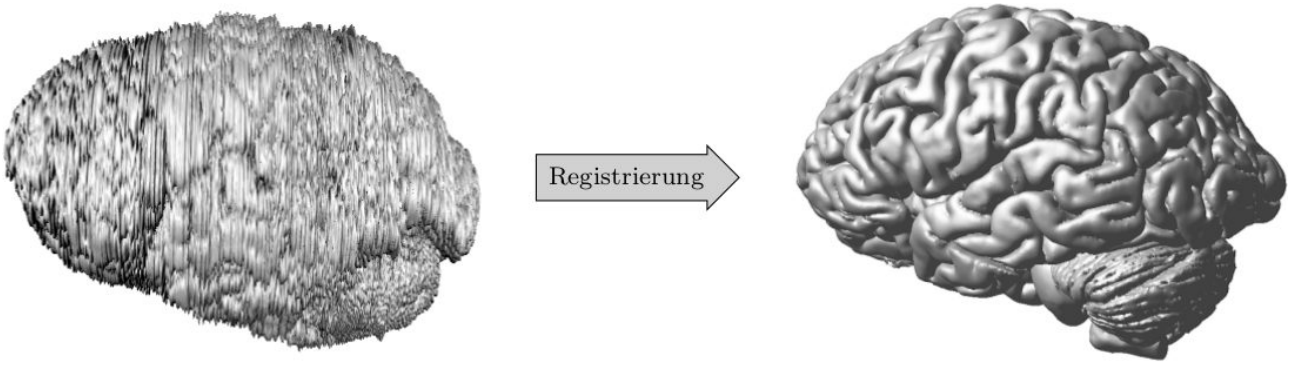


Abbildung 6: Gestapelte Hirnschnitte ohne (links) und mit Registrierung (rechts), Quelle: Amunts[4]

1.3 Software-Projekte

Alle oben aufgeführten Schritte werden mithilfe von eigener, stetig weiterentwickelter Software durchgeführt. Die folgende Projekte werden nach derzeitigem Stand in RTK implementiert:

Calibration Normalisierung der Messwerte jeder Winkelmessung

ICA "Independent Component Analysis", Bildentstörung durch Komponentenanalyse

EPA "Extended Polarization Analysis": Berechnung der Faserrichtung innerhalb eines 3D-PLI Hirnschnitt-Bildes (in-plane) über Direktions- und Inklinationsabbild

MEPA EPA auf Basis von MPI für parallele Berechnung auf JuDGE

calcIncl Berechnung der out-of-plane Faserrichtungen zwischen verschiedenen Schnittbildern

composeFOMs Erstellung von Karten der Faserrichtungen in Farbe

Stitching Zusammensetzen von Kachelbildern aus dem Microscope Polarimeter

BildverarbeitungsTools Tools zur Bildverbesserung

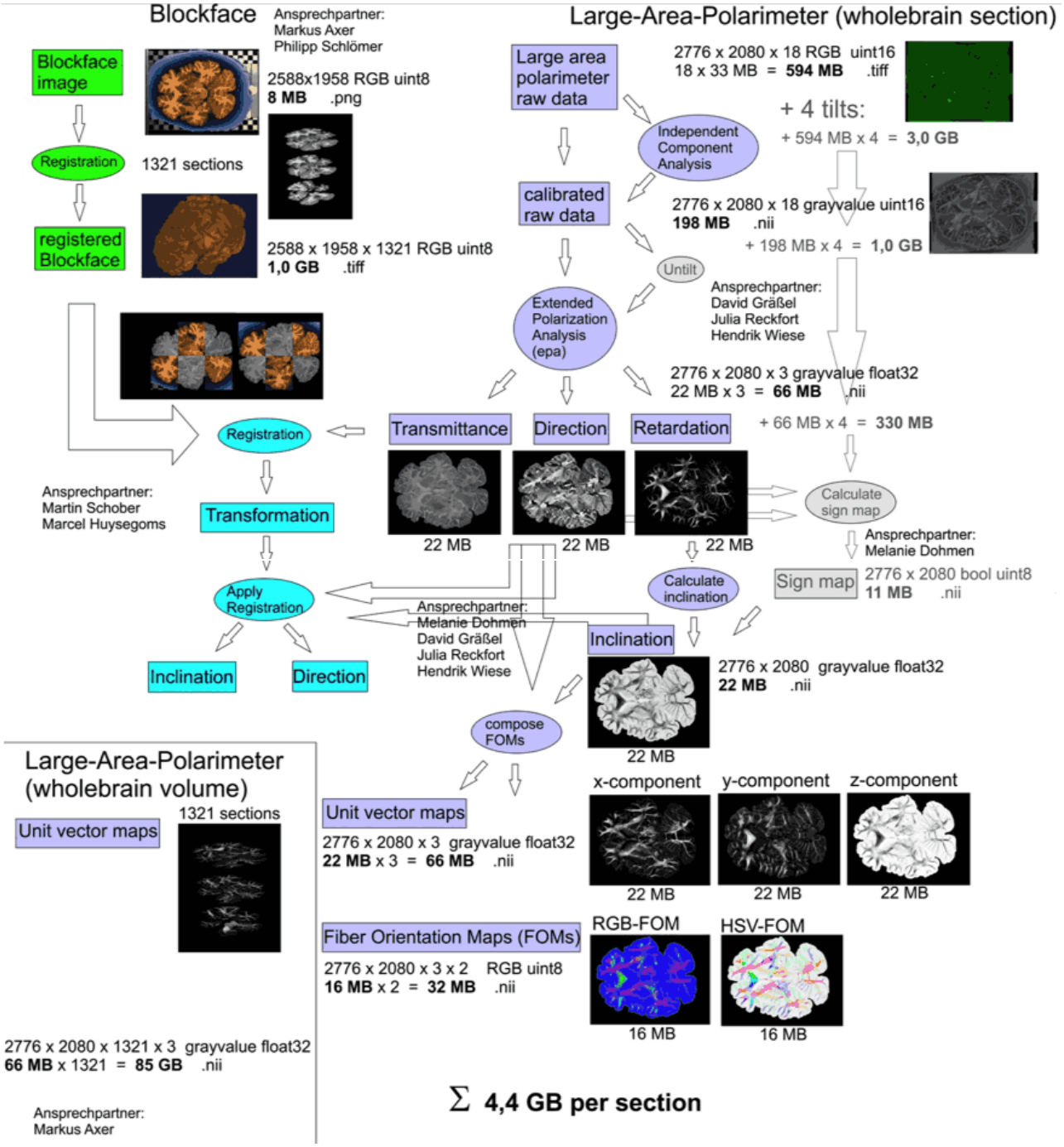


Abbildung 7: Die folgende Übersichtstafel zeigt zuständige Mitarbeiter sowie die anfallenden Datenmengen jedes Prozessschrittes von der 3D-PLI-Messung zum Konnektom. Quelle: M. Axer

1.4 Hardware-Umgebung

Die oben beschriebenen Berechnungsschritte werden je nach Umfang der zu verarbeitenden Daten auf verschiedenen Rechnerarchitekturen ausgeführt. Die folgenden Rechner kommen dabei zum Einsatz:

- lokale Arbeitsstation: Single-Node, typische Daten
  - Quad-Core Intel 64 Bit Prozessor mit 2,5 GHz
  - 8 GB RAM
- hausinterner Rechenknoten ime263: Single-Node mit
  - 2× Hexa-Core Intel 64 Bit Prozessor mit 3,47 GHz
  - 190 GB RAM
- High-Performance Computer JuDGE[2][5]: 206-Nodes mit jeweils
  - 2× Hexa-Core Intel 64 Bit Prozessor mit 2,66 GHz
  - 96 GB RAM
  - 2× Nvidia Tesla 1,15 GHz Graphikkarten (Fermi-Architektur) mit je 3 GB (54 Nodes) bzw. 6 GB (152 Nodes) RAM

1.5 Ist-Situation

Zurzeit gleichen die Entwickler ihre Software mit jeder fertigen Programmversion manuell mit den Zielrechnern ab. Dabei wird der aktuelle Softwarestand auf den Zielrechner heruntergeladen und die Programmkompilierung mittels `cmake` und `make` gestartet. Wird das Ausrollen der Software vernachlässigt, arbeiten Anwender der Software mit alten Versionsständen, die unter Umständen Fehler enthalten, oder aufgrund geänderter Schnittstellen nicht mehr mit in späteren Verarbeitungsschritten genutzten Programmen zusammenarbeiten. Solche Fehler werden typischerweise erst nach einer späteren Analyse der gewonnenen Daten festgestellt.

1.6 Projektziel

Ziel dieser Bachelorarbeit ist die Erstellung eines Programmes, welches die von der Arbeitsgruppe eigenentwickelte Software auf allen benötigten Maschinen automatisch verteilt. Hierzu gehört das Herunterladen der aktuellen Version des Software-Projektes auf den Zielrechner sowie die Kompilierung der Software auf dem und für das Zielsystem. Dabei ist wichtig, dass das in dieser Bachelorarbeit erstellte Programm für spätere Software-Projekte und Maschinen erweiterbar ist und sich grundlegende Änderungen (z.B. Abhängigkeiten von weiteren Programmen und Bibliotheken) einfach implementieren lassen.

[2] "Juelich Dedicated GPU Environment"

2 Grundlagen

2.1 Computerarchitektur

2.1.1 Symmetrische Multiprozessorsysteme (SMP)

Die im Heim-Bereich eingesetzten Computersysteme sind Single-Node Systeme mit einem oder mehreren Prozessorkernen. Für die Berechnungsgeschwindigkeit ist unerheblich, ob die Prozessoren in einem Computersystem auf einem einzigen Prozessor-Socket liegen (Mehrkernprozessor) oder auf mehrere Sockets aufgeteilt sind (Mehrprozessorsysteme). Relevant ist hingegen die Anbindung der Recheneinheiten an den Arbeitsspeicher (RAM). Gängige Architekturen, die heute in Heim-Computersystemen und Mobiltelefonen zu Einsatz kommen, sind sogenannte Symmetrische Multiprozessorsysteme (SMP), die Anbindung an den RAM ist symmetrisch, d.h. jeder Prozessor kann auf jeden Speicherbereich mit gleicher Berechtigungsstufe zugreifen. Die Programmierung eines solchen Systems ist einfach, skaliert aber schlecht mit einer großen Anzahl von Prozessoren bzw. Prozessorkernen.[7]

2.1.2 Non-Uniform Memory Access-Systeme (NUMA)

Demgegenüber steht die NUMA Architektur. Ziel der NUMA Architektur ist die Erhöhung der Skalierbarkeit beim Einsatz einer großen Anzahl an Prozessoren. Die Anbindung an den Speicher ist hier asymmetrisch, d.h. jeder Prozessor besitzt einen eigenen Speicherbereich, auf den dieser bevorzugt zugreifen kann. Auf den Speicher der anderen Prozessoren kann ebenfalls zugegriffen werden, der Zugriff geschieht demgegenüber langsamer als auf den eigenen Speicherbereich. Bei der Programmierung eines solchen Prozessors muss der Speicherzugriff dahingehend optimiert werden, dass Zugriffe überwiegend auf den prozessoreigenen Speicherbereich erfolgen. Die Konsequenz aus dieser Optimierung ist, dass jeder Prozessor mit einer eigenen, unabhängigen Aufgabe beschäftigt ist. Der Aufwand zur Synchronisation der Aufgaben nach Beendigung muss möglicht gering gegenüber dem Aufwand für die Berechnung sein.[7]

2.1.3 High Performance Computing (HPC)

Bei JuDGE handelt es sich um einen High Performance Computer. JuDGE gehört zu den Multi-Node Systemen. Die Untergliederung eines Großrechners in verschiedene Rechenknoten stellt eine Erweiterung des NUMA-Prinzips dar. Jeder Rechenknoten ist ein für sich abgeschlossenes Rechnersystem. Die Prozessoren eines Knoten haben ausschließlich Zugriff auf den eigenen Arbeitsspeicher. Die Steuerung der einzelnen Knoten geschieht von einem zentralen (Einstiegs-)Knoten. Daten- und Programmcode können über Netzwerkspeichersysteme ausgetauscht werden. Sinnvolle Einsatzzwecke ergeben sich bei der Nutzung gut parallelisierbarer Anwendungen. Hierzu gehört die Verarbeitung großer Bilddaten, wie sie im INM-1 anfallen. Je nach Umfang der Berechnungen und Auslastung des HPC kann die Arbeit auf eine unterschiedlich große Anzahl an Rechenknoten aufgeteilt werden. Organisatorisch kann jede interne Abteilung Rechenkapazität an dem System buchen, die Wartedauer auf Rechenzeit steigt mit der Menge an angeforderter Rechenleistung.

2.1.4 Single Instruction, Multiple Data (SIMD)

JuDGE besitzt je Rechenknoten zwei leistungsfähige Graphikkarten, die bei den Berechnungen unterstützen. Optimal nutzen lässt sich ein solches System durch eine Programmierung, die sowohl die Hauptprozessoren (CPU) als auch die Graphikprozessoren (GPU) vollständig auslastet. Moderne GPU basieren auf dem SIMD Verfahren. Im Gegensatz zur Arbeitsweise einer CPU, bei der jeder Datensatz durch eine individuelle Operation (Instruktion) in der Recheneinheit verarbeitet wird, wird bei SIMD Systemen eine Instruktion auf eine große Menge an Daten angewendet. Dieses Verfahren wird in der Multimediaverarbeitung angewendet, u.a. für Filter in der Bildverarbeitung, für die ein Filter auf jeden Pixel eines Bildes oder jeden Frame eines Videos angewendet wird. GPUs besitzen viele Recheneinheiten, sodass viele Berechnungen parallel ablaufen. Die Komplexität einzelner Befehle ist dagegen geringer als bei einer CPU. Es lassen sich daher nicht alle Arten von Berechnungen optimal auf einer GPU ausführen.[8]

2.2 Versionsverwaltung

Versionsverwaltungen sind Systeme, welches Änderungen eines Datenbestandes fortlaufend protokollieren. Ein bekanntes Beispiel eines versionsverwalteten Systems ist Wikipedia. Alle durchgeführten Änderungen an einem Artikel in der Zeitgeschichte sind protokolliert und einsehbar, Änderungen zwischen verschiedenen Artikeln lassen sich vergleichend darstellen.

Versionsverwaltungen haben sich aufgrund ihrer Nützlichkeit als Grundvoraussetzung in der Softwareentwicklung etabliert, um Änderungen am entwickelten Programmcode nachverfolgen und im Problemfall rückgängig machen zu können. Sie ermöglichen darüber hinaus, dass mehrere Softwareentwickler an verschiedenen Stellen des gleichen Programmes entwickeln und ihre getätigten Änderungen regelmäßig mit dem zentralen Versionsverwaltungssystem (Repository) abgleichen[3] können. Viele Versionsverwaltungssysteme werden daher als Serverdienst betrieben. In vielen Open-Source Projekten ist der Server frei erreichbar, sodass jeder Internetnutzer die aktuelle Version auschecken und Entwickler mit der notwendigen Authorisierung ihre Änderungen einchecken können.

Für die bisherigen Entwicklungen findet in unserer Arbeitsgruppe die Versionsverwaltung Git Verwendung.

Bei Git handelt es sich um ein dezentrales Versionsverwaltungssystem. Die Änderungshistorie wird im Stammordner des versionierten Datenbestandes[4] auf der lokalen Festplatte geschrieben. Es ist möglich, dass mehrere Nutzer mit dem gleichen Repository arbeiten. Hierfür wird ein bestehendes Repository mittels `git clone` geklont. Änderungen beider Klone lassen sich später wieder zusammenführen. Organisatorisch bietet es sich an, ein zentrales Haupt-Repository zu erstellen. Diese Vorgehensweise ähnelt den älteren, zentralen Systemen wie Subversion. Dieses zentrale Repository ist für alle Nutzer über ein Netzwerk erreichbar. Für diesen gängigen Fall beherrscht Git einen “bare“-Modus, der keine Arbeitskopie des Stammordners, sondern lediglich die Versionierungsinformation enthält[5]. Als Dateiübertragungsprotokoll eignen sich beliebige Dateiübertragungsprotokolle wie SSH (SCP), Samba, Webdav oder FTP.

[3] Das Hochladen des eigenen Programmcodes heißt einchecken, das Herunterladen des Programmcodes auschecken
[4] genauer: im Unterordner “.git“ des Stammordners
[5] Aus der Versionierungsinformation lässt sich jederzeit ein beliebiger Versionsstand rekonstruieren

2.3 Grundlagen der Programmiersprache

Programmiersprachen werden unterteilt in Skriptsprachen, Interpretersprachen, Compilersprachen und Maschinensprachen.

2.3.1 Maschinensprache

Maschinensprache stellt die älteste Programmiersprache dar. Programme werden direkt in für den Prozessor verständlichen Befehlen eingegeben. Durch die hardwarenahe Programmierung lassen sich - bei guter Optimierung - optimale Rechengeschwindigkeiten erreichen. Maschinensprache wird aufgrund der schlechten Wartbarkeit und architekturübergreifenden Portierbarkeit des Programmcodes, wenn überhaupt, ausschließlich für hochoptimierte Software-Bibliotheken wie z.B. Videoencoder, Kompressionsalgorithmen oder Verschlüsselungssoftware verwendet.

2.3.2 Compilersprache

Compilersprachen wurden aus der Not heraus entwickelt, die durch den Einsatz von Maschinensprachen entstand. Mithilfe eines Compilers lässt sich Code in für Menschen gut lesbarer Form ("Hochsprache") schreiben und mittels Compiler in Maschinensprache übersetzen. Verbreitete Compiler sind typischerweise sehr gut in Optimieren von Code, d.h. der vom Compiler erzeugte Maschinencode kommt bezüglicher seiner Ausführungsgeschwindigkeit optimalem Maschinencode sehr nahe. Der Code lässt sich weiterhin, mit einigen Einschränkungen, auch für andere Prozessorarchitekturen oder Betriebssysteme erzeugen, sodass die vollständig parallele Entwicklung für unterschiedliche Zielsysteme entfällt, wie sie bei Maschinensprache nötig ist.[10]

2.3.3 Interpretersprache

Interpretersprachen übersetzen den Code während der Ausführung in Maschinensprache. Statt eines Compilers wird hier ein Interpreter eingesetzt, der diese Arbeit übernimmt. Ein Vorteil ist, dass der Interpreter auf das Zielsystem angepasst ist, d.h. Architekturabhängigkeiten des Codes entfallen nahezu vollständig, indem sie vom Interpreter vereinheitlicht werden. Hierdurch lässt sich einmal entwickelter Code auf völlig unterschiedlichen Computer-Architekturen einsetzen. Weiterhin sind Interpretersprachen typischerweise abstrakter als Compilersprachen, sodass für die gleiche Implementierung (deutlich) weniger Codezeilen benötigt werden. Durch den kürzeren Quellcode und den fehlenden Zwischenschritt der Compilierung ist die Wartbarkeit vereinfacht. Ein Nachteil ist die geringere Ausführungsgeschwindigkeit gegenüber Compilercode.[10]

2.3.4 Skriptsprachen

Skriptsprachen ergeben sich aus der Möglichkeit von vielen Eingabeaufforderungen, Eingaben über eine Textdatei linear abzuarbeiten. Dabei haben die verschiedenen Betriebssysteme mit ihren Eingabeaufforderungen Erweiterungen implementiert, welche grundlegende Programmierstrukturen aus Interpretersprachen bieten. Hierzu gehören Schleifen, Bedingungen, Variablen, logische Vergleichsoperatoren

sowie in Grundzügen die Fehlerbehandlung. Viele Strukturen werden über externe, zum Betriebssystem gehörende Standardprogramme abgedeckt, die die Ausführungsgeschwindigkeit gegenüber Interpretersprachen nochmals reduzieren und die Programmierung teilweise trickreich machen. Ein weiterer Nachteil ist die Betriebssystem- respektive Shell-Abhängigkeit. Ein Vorteil ist, dass kein zusätzlicher Interpreter benötigt wird, sondern die Programmierung vollständig mit Betriebssystemfunktionen abgebildet werden kann.

3 Anforderungen und Planungen

3.1 Namensgebung

Der Name des Programmes leitet sich aus seinen Eigenschaften ab: “RTK“, für “Rollout-Toolkit“.

3.2 Versionsverwaltung

Als Versionsverwaltungssystem zur Entwicklung von RTK wurde auf das bereits für andere Projekte im Einsatz befindliche Git gesetzt. Jedes Projekt besitzt ein bare-Repository auf dem ime263 Datei-Server, entwickelt wird an einer lokalen Kopie auf dem Arbeitsplatzrechner. Der Abgleich der Repositories erfolgt via SSH. Zum Rollout der verschiedenen Softwareprojekte auf die Zielmaschinen wird ebenfalls auf SSH als Übertragungsprotokoll zum Zugriff auf das git-Repository gesetzt. Für das Auschecken des Quellcodes wird daher ein SSH-Kennwort benötigt. Um das Speichern dieses sensitiven Kennwortes zu vermeiden, muss das SSH-Kennwort bei jedem Start von RTK abgefragt werden.

Im Zuge der Arbeit stellte sich eine weitere organisatorische Eigenheit beim Einsatz des Repositories heraus, die eine Anpassung des Arbeitsprozesses mit dem Repository erforderlich macht. Die bisherige Arbeitsweise führt dazu, dass ein Entwickler beim Rollout seiner Software manuell die letzte stabile Version auscheckt. RTK liegt bei diesem Vorgehen jedoch keine Informationen darüber vor, welche Version als stabil gilt. Daher wird für RTK der Einsatz von Versionstags[6] nötig, welche der Entwickler einem Versionsstand zuweisen kann. Getaggte Versionen kennzeichnen eine stabile, einsatztaugliche Version. Je nach Parameter beim Aufruf von RTK kann die letzte getaggte Version ausgerollt werden oder ein bestimmter Versionstag angegeben werden, welcher ausgerollt werden soll. Alternativ kann auch die letzte Entwicklerversion ohne Versionstag (“Master“) genutzt werden. Letzteres ist sinnvoll für das automatische Kompilieren einer Testversion auf dem Entwicklerrechner.

3.3 Verwendete Programmiersprache

Folgende Eigenschaften weist das Programm auf:

- Es besteht aus mehreren Teilprogrammen, welche verschiedene Aufgabenteilbereiche erfüllen[7]
- Der Programmablauf ist überwiegend linear
- Es werden nur marginale Ansprüche an Rechenleistung gestellt
- Das Programm muss auf unterschiedlichen Rechnerarchitekturen und Betriebssystemderivaten ausgeführt werden
- Die überwiegende Aufgabe besteht im Vergleichen von Variablen und dem Ausführen von Shell-Befehlen

Die oben beschriebenen Anforderungen lassen sich am besten mit einer Interpretersprache erfüllen. Sie bietet gegenüber Skriptsprachen den Vorteil, dass sich auch komplexe Datenstrukturen und Funk-

[6] von engl. “Marker“, “Abgrenzer“, nicht der dt. “Tag“
[7] Hauptprogramm, E-Mail-Versand, Kompilierungsvorgang

tionen gut implementieren lassen. Diese Funktionen werden benötigt, um beispielsweise Passwortdatenbanken anzulegen oder Konfigurationsdateien einzulesen. Gegenüber einer Compilersprache bietet sie den Vorteil, dass der Zwischenschritt der Compilierung des eigenen Programmcodes entfällt und viele Funktionen wie das Parsen von Kommandozeilen oder der Aufbau einer SSH-Verbindung als fertiges Modul eingebunden werden können. Die Länge des Quellcodes wird dadurch gegenüber einer Compilersprache deutlich reduziert. Dies reduziert auch den späteren Wartungsaufwand. Da bereits für andere Projekte Python als Interpretersprache zum Einsatz kommt und somit Erfahrung im Hinblick auf die Verfügbarkeit von Python auf den verschiedenen Rechnerarchitekturen besteht, wurde das Programm in Python entwickelt und andere Interpretersprachen (z.B. Perl) nicht in Betracht gezogen.

3.4 Benachrichtigung an den Benutzer

Um den Benutzer des Rollout-Toolkit über den korrekten Ablauf oder aufgetretene Fehler zu informieren, ist es erwünscht, diesen nach dem Durchlauf des Programmes zu benachrichtigen. Die bevorzugte Methode zur Benachrichtigung war nach Rücksprache mit der Abteilung einstimmig die E-Mail, da das Abrufen und Verarbeiten der eigenen E-Mails einen bewährten Prozess darstellt. Die Benachrichtigung erfolgt nicht nur im Fehlerfall, sondern auch im Erfolgsfall. Dies verhindert, dass ein Fehler im E-Mail-Versand zur fälschlichen Annahme eines erfolgreich verlaufenen Kompilierungsvorganges führt. Durch die regelmäßige Kontrolle der E-Mails seitens des Benutzers ist gewährleistet, dass er die Benachrichtigung zur Kenntnis nimmt. Dabei beinhaltet der Betreff die Stichwörter “Success“ bzw. “Failure“, um bereits am Betreff der E-Mail den Erfolg oder Misserfolg des Rollouts erkennen zu können.

Interne Nachfragen im Rechenzentrum ergaben, dass das Zustellen von E-Mails an interne Postfächer über den lokalen E-Mailserver auch ohne Logindaten möglich ist. Da alle betroffenen Rechner an das Internet sowie an das Intranet angeschlossen sind, stellt das Versenden von E-Mails - im Gegensatz zu vorherigen Befürchtungen - technisch kein Problem dar.

3.5 Jobsystem

Der High Performance Computer JuDGE sowie der Rechenknoten ime263 nutzen zum Ausführen länger andauernder Prozesse ein Jobsystem. Ein neuer Job wird über ein Shell-Skript angelegt, in welchem neben dem auszuführenden Prozess zusätzliche Informationen über die maximale Laufzeit sowie die benötigten Ressourcen[8] angegeben werden. In diesem Skript werden weiterhin Log-Dateien und E-Mail Adressen des Benutzers zur Benachrichtigung über die Abarbeitung des Jobs festgelegt. Der Job wird vom Jobsystem in Abhängigkeit vom Ressourcenverbrauch in eine Warteschlange eingereiht und zu gegebener Zeit auf einer Arbeits-Node ausgeführt. Das Kompilieren der größeren Programme bewegt sich dabei in etwa an der Grenze des Ressourcenverbrauches, der für das Ausführen auf einer Login-Node geduldet wird. Die Nutzung des Jobsystems ist daher auf einigen Maschinen obligatorisch. Dabei kann in den Konfigurationsdateien von RTK festgelegt werden, ob für eine Maschine das Jobsystem verwendet wird, oder ob die Kompilierung direkt ausgeführt wird. Weiterhin sind zwei verschiedene Jobsysteme auf unterschiedlichen Maschinen im Einsatz, die sich marginal unterscheiden. Daher ist in der Konfigurationsdatei anzugeben, welches Jobsystem Verwendung findet. Über das Jobsystems werden dann das Kompilieren mittels `make` sowie der optionale Installationsbefehl

[8] Anzahl der Nodes, Anzahl der CPU sowie die Menge an Arbeitsspeicher

`make install` ausgeführt. Nach Beenden des Prozesses verschickt das Jobsystem eine E-Mail an den Benutzer.[11]

3.6 Anwendungsfälle

1. Das Programm EPA (Programmiersprache C) soll auf allen Rechnern ausgerollt werden, für die es entwickelt wurde. Dabei soll der letzte, stabile Versionstag verwendet werden. Als Benutzer soll `ckrause` genutzt werden, da der lokale Benutzername nicht mit dem Benutzernamen auf den Remote-Rechnern übereinstimmt.

2. Ein in Python geschriebenes Programm (ICA) soll auf einer spezifischen Maschine (JuDGE) in einer älteren, stabilen Version (v1.1) ausgerollt werden, weil die aktuelle Version mit der Maschine nicht kompatibel ist.

3. Nachdem ein Python-Rollout (ICA) der letzten stabilen Version mit dem Rollout-Toolkit fehlgeschlagen ist, soll er für den problematischen Rechner (ime263) wiederholt werden. Hierfür sollen Debug-Informationen per E-Mail verschickt werden.

4. Ein Entwickler möchte ein in C geschriebenes Programm (EPA) auf dem eigenen Rechner kompilieren und in ein bestimmtes Testverzeichnis installieren. Dabei soll die aktuelle Version aus dem Repository verwendet werden. Es sollen nur Warnmeldungen in den E-Mails auftauchen.

5. Der Entwickler des letzten Beispieles entscheidet sich dafür, nach jeder Mittagspause einen Testlauf seines Programmes zu starten. Hierfür soll das C Programm jeden Tag um 11:30 Uhr kompiliert werden. Da seine Festplatte verschlüsselt ist, soll ein Cron-Job angelegt werden, der die Passwörter im Klartext beinhaltet und an das Programm übergibt.

4 Konzeption

4.1 Module

Das Programm wurde in drei Module unterteilt.

rtk.py ist das Hauptmodul. Es wird initial vom Benutzer aufgerufen und läuft auf dem lokalen Rechner. Mittels Kommandozeilenparameter werden Projekt, Zielmaschine und einige andere Einstellungen festgelegt, die Konfiguration wird über die Konfigurationsdateien eingelesen.

rtkCompile.py ist ein Unterprogramm, welches für alle C Programme auf der Zielmaschine ausgeführt wird. Es übernimmt die Kompilierung des Projektes mittels `make` und kopiert via `make install` die entstandenen, ausführbaren Dateien in die Zielordner der jeweiligen Maschine.

rtkFeedback.py versendet Statusmails über die erfolgreiche oder nicht erfolgreiche Ausführung des Programmes. Eine Statusmail wird vom Hauptmodul gesendet. Weiterhin sendet rtkCompile.py für jede Zielmaschine jeweils eine Statusmail. Wird der Kompiliervorgang über das Jobsystem eingestellt, so erfolgt weiterhin eine Benachrichtigung des Jobsystems nach Beendigung des Prozesses. Letztere E-Mail wird vom Jobsystem aus versendet und nicht vom rtkFeedback-Modul.

4.2 Konfigurationsdateien

Eines der grundlegenden Entwicklungsparadigmen war die Entwicklung eines generischen Programmes mit der Möglichkeit, Erweiterungen und Änderungen über Konfigurationsdateien abzubilden, sodass auch tiefgreifende Änderungen ohne Quellcodeanpassung ermöglicht werden. Die Programmeinstellungen werden über drei verschiedene Konfigurationsdateien abgebildet.

rtk.conf In dieser Datei werden allgemeine Informationen abgelegt, welche unabhängig von einem konkreten Projekt oder einer konkreten Maschine sind. Hierzu gehören:

- der Ort des Git-Archives
- der Ort der lokalen Log-Datei
- der E-Mail-Server zum Versenden von E-Mails

Im Folgenden findet sich eine beispielhafte rtk.conf

```
[git]
gitMachine=ime263
gitArchive=ime263.ime.kfa-juelich.de/data/PLI-Group/repositories/rtk.git

# A unique date string will additionally be attached to the rtkLogFile filename
[local]
rtkLogFile=/tmp/rtkLogFile

[mail]
eMailServer=mail.fz-juelich.de
```

machine.conf In dieser Datei werden spezifische Einstellungen für die Zielmaschine getätigt. Hierzu gehören:

- der FQDN zur Herstellung der SSH-Verbindung
- die Auswahl des Jobsystems (keines, MSUB oder PBS)
- Angabe eines temporären Arbeitsverzeichnis (optional)
- Systemvariablen importieren (optional)

Der optionale Parameter zur Änderung des temporären Arbeitsverzeichnisses resultiert aus dem Einsatz von Multinode-Systemen in Kombination mit dem Jobsystem. Da die standardisierten temporären Verzeichnisse nicht Knoten-übergreifend verfügbar sind, kann ein Prozess, der über das Jobsystem auf der Arbeitsnode gestartet wird nicht auf die temporären Verzeichnisse der Einstiegs-Node zugreifen. Sie müssen daher in einem übergreifend verfügbaren Dateipfad abgelegt werden.

Die Funktion zum Import von Systemvariablen wurde eingeführt um Pfadangaben für Bibliotheken an den Compiler zu übermitteln.

Im Folgenden findet sich eine beispielhafte machine.conf

```
[judge]
hostUrl=judge.fz-juelich.de
useJobSystem=MSUB
tempDir=$WORK
importCommand=source /usr/local/jsc-inm/registration/bashrc
```

project.conf Zu den projektspezifischen Parametern gehören:

- das git-Repository
- das Git Übertragungsprotokoll (SSH oder git)
- der E-Mail Empfänger für Statusmails
- der E-Mail Sender
- erlaubte Maschinen, auf denen das Projekt ausgerollt werden darf
- die Programmiersprache (Python oder C),
  sowie für C die Entscheidung über die Ausführung von `make install`
- nur Python: Pfadangaben für die Installation
  (Angabe für jede Maschine separat, obligatorisch)
- nur C: Umgebungsvariablen für CMake, welche u.a. Installationspfade für `make install` festlegen (Angabe für jede Maschine separat, optional)

Im Folgenden findet sich eine beispielhafte project.conf für C:

```
[EPA]
gitRepo=ime263.ime.kfa-juelich.de/data/PLI-Group/repositories/epa.git
gitProto=ssh
allowMachines=ime263,local,judge,imedv18
eMailSender=ch.krause@fz-juelich.de
eMailReceiver=ch.krause@fz-juelich.de
install=yes
cMakeEnv.ime263=-DITK_DIR=/opt/PLISoft/toolkits/InsightToolkit-4.5.2/
cMakeEnv.judge=-DITK_DIR=/usr/local/jsc-inm/ITK/InsightToolkit-4.6.0/build/
```

Im Folgenden findet sich eine beispielhafte project.conf für Python:

```
[RTK]
gitRepo=ime263.ime.kfa-juelich.de/data/PLI-Group/repositories/rtk.git
gitProto=ssh
allowMachines=ime263,local,judge
eMailSender=ch.krause@fz-juelich.de
eMailReceiver=ch.krause@fz-juelich.de
install=copy
pythonDest.ime263=/tmp/rtktemp
pythonDest.judge=$WORK/temptemp
```

Detaillierte Erläuterungen über die Konfiguration findet sich in der Beispiel-Konfigurationsdatei im Quellcode.

4.3 Befehlszeilenoptionen

Die Einstellungen für den Aufruf des Programmes geschehen über Kommandozeilenparameter. Dieses Verfahren bietet im Gegensatz zur Eingabe innerhalb des Programmes die Möglichkeit, das Programm über ein Shell-Skript oder via Cron-Job zu starten. Es ist mindestens das Projekt mit `-p` anzugeben. Andere Parameter sind optional, sie besitzen einen Standardwert.

Eine Ausnahme bildet die Eingabe von Passwörtern. Diese werden während des Programmablaufes abgefragt. Eine Übergabe via Kommandozeile ist optional möglich, birgt jedoch das Risiko, dass das Kennwort unbemerkt auf der Festplatte gespeichert wird, z.B. in der Historie der eingegebenen Kommandozeilenbefehle. Das abgefragte Passwort ist entweder das SSH-Kennwort oder die Passphrase für eine Public-Key Authentifizierung. Es wird die Standardeinstellung der lokalen SSH Konfiguration genutzt.

Folgende Kommandozeilenparameter sind möglich:

- `-p projekt`: Angabe des Projektes, z.B. `-p EPA`. Das Projekt beachtet Groß- und Kleinschreibung.

- `-m zielmaschine`: Angabe der Zielmaschine. Die Maschine beachtet Groß- und Kleinschreibung.

  - `-m ime263,judge`: Angabe mehrerer Maschinen möglich
  - `-m all` (Standardwert): das Projekt wird auf alle erlaubten Maschinen ausgerollt

- `-u user`: Angabe des SSH Benutzernamen:

  - `-u vnachname` Benutzer wird auf `vnachname` für alle Zielsysteme gesetzt
  - `-u ask` individuellen Benutzer für jedes System erfragen
  - Standard: Wird kein Benutzername angegeben, wird der aktive Benutzername des lokalen Systems verwendet.

- `-t tag`: Dient zur Auswahl der ausgecheckten Programmversion:

  - `-t TRUE`: checkt die letzte, getaggte Version aus (Standard)
  - `-t FALSE`: checkt die letzte, ungetaggte "Master" Version aus
  - `-t v1.08`: checkt die spezifische, angegebene Version aus

  Existiert im git-Repository bisher keine getaggte Version, wird der letzte "Master" unter Ausgabe einer Hinweismeldung genutzt.

- `-l level`: Setzt das Logginglevel

  - `-l DEBUG`: Ausgabe von gesprächigen, aber unübersichtlichen Debug-Meldungen
  - `-l INFO`: Standardwert
  - `-l WARNING`: Unterdrückung von Info-Meldungen. Nur Warnungen anzeigen.

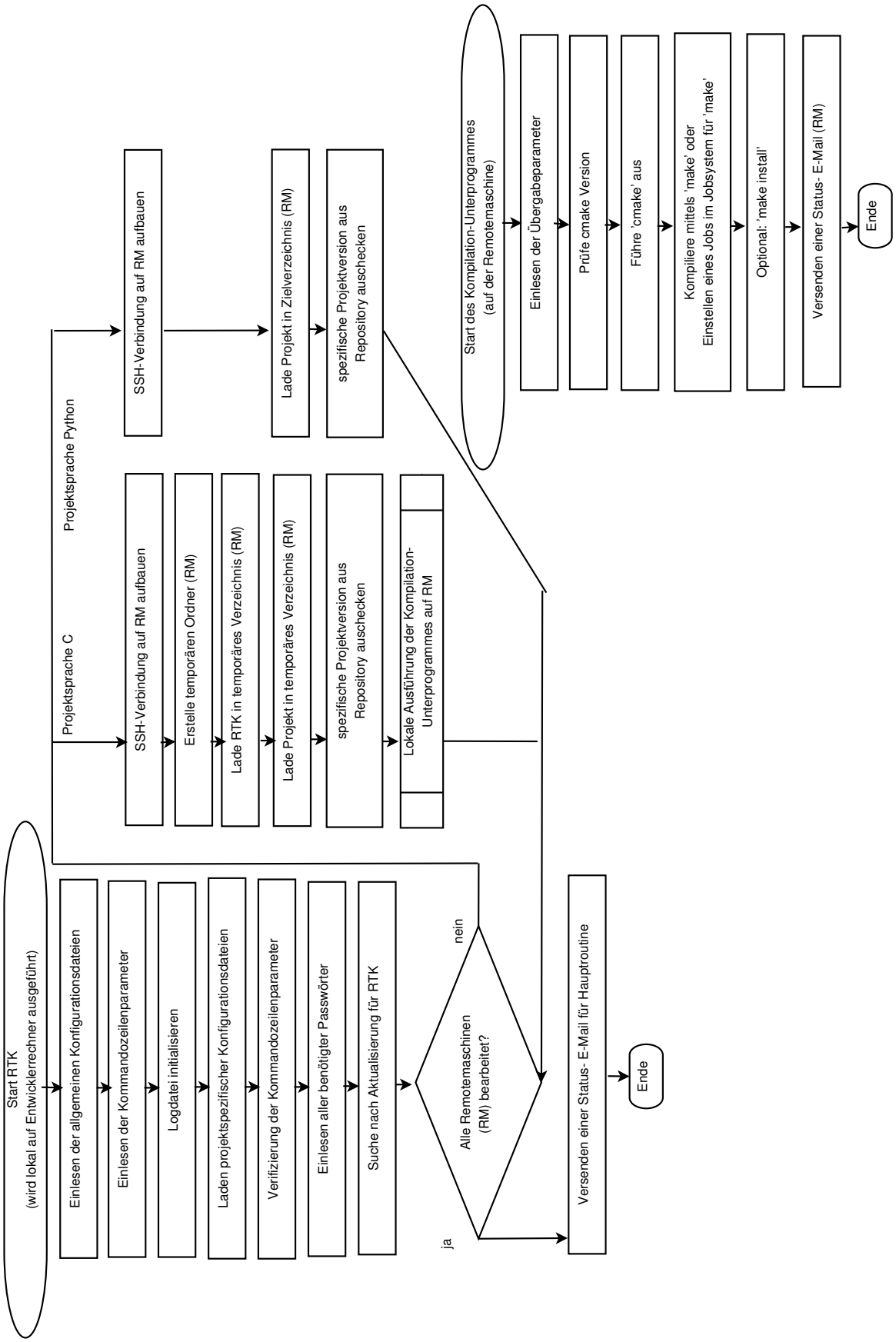
- `-g TRUE`: “Generic Password“:
  Wird dieser Parameter gesetzt, wird nur an einer Stelle ein Passwort abgefragt und für alle Maschinen verwendet. Hilfreich, wenn das Passwort für alle Maschinen identisch ist.

- `-d TRUE`: Verhindert die Aktualisierung des lokalen RTK-Verzeichnisses mit jedem Programmstart. Ein veraltetes, lokal befindliches RTK nutzt gegebenenfalls falsche Bibliotheken- oder Installationspfade durch obsolete Konfigurationsdateien.

- `-f passlib`: Übergibt die Passwörter im Klartext über die Kommandozeile. Wird im Format `-f 'machine1,user1,password1;machine2,user2,password2'` angegeben, wobei ein “;“ verschiedene Maschinen von einander trennt und “,“ die Trennung zwischen Hostname, Benutzer und Passwort angibt. Aus Sicherheitsgründen ist diese Funktion nicht zu empfehlen. Sie ist für automatische Starts via git-Hook oder Cron-Job jedoch erforderlich. Das Passwort wird hierbei im Klartext auf der Festplatte gespeichert. Nur für verschlüsselte Festplatten empfohlen!

Einige Beispiele für angewandte Befehlszeilenoptionen finden sich in den Anwendungsfällen in 5.1

4.4 Programmablaufplan



5 Ergebnisse

5.1 Anwendungsfälle

Im folgenden werden die in Kapitel 3.6 beschriebenen Anwendungsszenarien erörtert.

1. `./rtk.py -p EPA -u ckrause`
   Startet einen Rollout für das Projekt EPA. Der Benutzername für alle Maschinen ist `ckrause`. Folgende Einstellungen müssen nicht explizit über die Kommandozeile festgelegt werden, da es sich um Standardeinstellungen handelt:

   - es wird auf allen erlaubten Maschinen ausgerollt
   - es wird die letzte, getaggte Version ausgecheckt
   - die Kennwörter werden einzeln für jede Maschine abgefragt
   - Logging findet auf `INFO`-Ebene statt

2. `./rtk.py -p ICA -m judge -t v1.1`
   Startet einen Rollout für das Projekt ICA auf der Maschine JuDGE. Es wird Version v1.1 ausgecheckt. Standardeinstellungen sind:

   - Logging findet auf `INFO`-Ebene statt
   - der verwendete Benutzername entspricht dem lokalen Benutzernamen des Computers

3. `./rtk.py -p ICA -m ime263 -l DEBUG`
   Startet einen Rollout für das Projekt ICA auf der Maschine ime263 mit DEBUG-Logging. Da nur auf einer Maschine ausgerollt wird und das git-Archiv auf eben dieser Maschine liegt, wird nur einmal ein Kennwort abgefragt. Standardeinstellungen sind:

   - es wird die letzte, getaggte Version ausgecheckt
   - der verwendete Benutzername entspricht dem lokalen Benutzernamen des Computers

4. `./rtk.py -p EPA -m local -l WARNING -t FALSE`
   Startet einen Rollout für das Projekt EPA auf der lokalen Maschine mit WARNING-Logging. Es wird die aktuelle "Master"-Version verwendet und kein Tag ausgecheckt. Standardeinstellungen sind:

   - der verwendete Benutzername entspricht dem lokalen Benutzernamen des Computers
   - die Kennwörter für "Lokal" und das git-Archiv werden einzeln abgefragt

5. `./rtk.py -p EPA -m local -l WARNING -t FALSE -f 'local,u1,p1;ime263,u2,p2'`
   Startet den Rollout aus Beispiel 4, übergibt jedoch die Zugangsdaten über den Kommandozeilen-Aufruf. Die Installation auf dem lokalen Rechner erfolgt nicht über eine lokale Shell, sondern grundsätzlich ebenfalls über SSH, sodass auch für den lokalen Rechner ein SSH Kennwort erforderlich ist. Ebenfalls benötigt wird das Kennwort für das git-Archiv (ime263).

5.2 Logmeldungen

Logmeldungen des Hauptprogrammes erreichen im Erfolgsfall den Anwender mit folgendem Betreff: “Success: RTK: EPA Main Thread“. Eine Mail enthält in etwa folgenden Inhalt:

```
##########/tmp/rtkLogFile2015-01-01T19:59:18.276259############
INFO:- Logging activated!
INFO:- CLI parameter parsed, checked config files successfully for your given
       project and machines!
INFO:- Your given projects are in allowed Machines! That’s good!
INFO:- Using external CLI passlib!
INFO:- Passlib successfully imported!
INFO:- I now have also the credentials for the git Archive!
INFO:- Check for local RTK update skipped due to users choice!
INFO:- I’ll try now to log in to machine judge
INFO:- SSH session login successful on judge
INFO:- Project Language is C
INFO:- Download rtk-git archive to Tempfolder
       /work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259 on judge completed
INFO:- Also completed EPA download to tempfolder
WARNING:- Should check out latest tagged version, but there is no tagged version!
INFO:- Compiling locally on judge! Everything finished here!
INFO:- Logging out from judge
```

Logmeldungen des rtkCompile.py-Modul enthalten das erstellte Jobscript, sofern der Jobprozess genutzt wird, als auch die Kommandozeilen-Ausgabe des ausgeführten `cmake` Kommandos. Kommandozeilen-Ausgaben, die aus der Ausführung des Jobscriptes beruhen, können nicht angefügt werden, da das Jobscript erst zu einem späteren Zeitpunkt auf der Maschine ausgeführt wird. Aufgrund der Status E-Mail des Jobscriptes kann jedoch der Erfolg oder Misserfolg des Jobscriptes und somit der Kompilierung ersehen werden. Im Fehlerfall kann der Ort der erstellten Logdatei dem Jobscript entnommen werden, um diese manuell zu überprüfen.

```
##########/work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259/rtkJob.sh############
#!/bin/bash -x

#MSUB -S /bin/bash
#MSUB -N RTK
#MSUB -l nodes=1:ppn=2,mem=2gb,walltime=02:00:00
#MSUB -j oe
#MSUB -m ae
#MSUB -o /work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259/rtkJobLog
#MSUB -e /work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259/rtkJobLog
#MSUB -M ch.krause@fz-juelich.de
cd /work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259/EPA/build
source /usr/local/jsc-inm/registration/bashrc
make
make install
```

```
##########/work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259/rtkCommandOutput####
-- The C compiler identification is GNU 4.3.4
-- The CXX compiler identification is GNU 4.3.4
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Found MPI_C: /opt/parastation/mpi2-intel/lib/libmpich.so;
   /opt/parastation/mpi2-intel/lib/libopa.a;/usr/lib64/libpthread.so;
   /opt/parastation/lib64/libpscom.so;/usr/lib64/librt.so
-- Found MPI_CXX: /opt/parastation/mpi2-intel/lib/libmpichcxx.so;
   /opt/parastation/mpi2-intel/lib/libmpich.so;/opt/parastation/mpi2-intel/
    lib/libopa.a;/usr/lib64/libpthread.so;/opt/parastation/lib64/libpscom.so;
   /usr/lib64/librt.so
-- Boost version: 1.49.0
-- Found the following Boost libraries:
--   mpi
--   program_options
--   serialization
-- Configuring done
-- Generating done
-- Build files have been written to:
   /work/inm1/ckrause/rtk.2015-01-01T19:59:18.276259/EPA/build
```

Die Status E-Mails, welche vom Jobsystem versendet werden, sehen wie folgt aus:

```
PBS Job Id: 1110.localhost
Job Name:   RTK
Exec host:  ime263/1+ime263/0
Execution terminated
Exit_status=0
```

5.3 Fehlermeldungen

Im Folgenden sind einige beispielhafte Fehlermeldungen ersichtlich, die bei der Nutzung von RTK auftreten können.

Bei geändertem SSH Schlüssel auf der Zielmaschine erfolgt eine Fehlermeldung seitens SSH, die auf eine eventuelle Man-in-the-Middle-Attacke hindeutet. Diesen Fehler fängt das verwendete pxssh-Modul aus Sicherheitsgründen nicht ab. Die Fehlerursache geht nicht aus der Fehlermeldung hervor, wird aber bei einem manuellen Login-Versuch auf der entsprechenden Maschine ersichtlich.

```
INFO:- I'll try now to log in to machine local
WARNING:- SSH session failed on login.
WARNING:End Of File (EOF). Exception style platform.
<pexpect.pxssh.pxssh object at 0x7f3441d4a650>
version: 3.1
command: /usr/bin/ssh
args: ['/usr/bin/ssh', '-q', '-l', 'christian', 'localhost']
searcher: <pexpect.searcher_re object at 0x7f3441d4a590>
buffer (last 100 chars): ''
before (last 100 chars): ''
after: <class 'pexpect.EOF'>
```

Eine folgende Meldung erhält man bei einem fehlerhaft eingegebenen Passwort. Neben der unten stehenden "password refused" Meldung existiert eine ähnliche Meldung "password denied". Erstere erscheint, wenn der Passwortdialog von der Zielmaschine beendet wird, letztere erscheint, wenn erneut nach einem Passwort gefragt wird. In der Praxis ist die Unterscheidung nicht relevant.[6]

```
INFO:- I'll try now to log in to machine local
WARNING:- SSH session failed on login.
WARNING:password refused
```

Zeitweise verhindert die Remote-Shell einen Logout mit der `logout()`-Prozedur, weil ein aktiver Hintergrundprozess läuft. Dieser Hintergrundprozess ist für die Arbeit von RTK nicht relevant, das Auftreten lässt sich über das verwendete Modul `pxssh` allerdings nicht abschalten. Daher wird ein entsprechender Fehler von Seiten RTK ignoriert; RTK setzt seine Arbeit mit der Verbindung auf die nächste Maschine fort. Die Verbindung bleibt in diesem Fall auf Betriebssytemebene bestehen und wird nach einer Wartezeit von etwa einer Stunde von der Remote-Maschine selbstständig getrennt.

```
INFO:- Project Language is Python
INFO:- Checked out version v0.5
WARNING:- Logout from judge failed, continue anyway!
```

Sporadisch tritt beim Login auf JuDGE eine fehlerhafte Erkennung des Kommandozeilen-Prompt auf. Der Fehler tritt in der Regel nur beim ersten Verbindungsaufbau auf, sodass RTK in diesem Falle einen zweiten Versuch startet. Dennoch wird eine entsprechende Warnung in die Logdatei geschrieben.

```
INFO:- Check for local RTK update skipped due to users choice!
INFO:- I'll try now to log in to machine judge
WARNING:- SSH session failed on login.
WARNING:could not synchronize with original prompt
INFO:- Retrying to connect
```

In diesem Fall fehlt ein Eintrag in der Konfigurationsdatei project.conf.

```
INFO:- Project Language is Python
WARNING:- Error in project.conf with machine ime263: (No option 'pythondest.ime263'
          in section: 'RTK') I'll go on with the next machine!
```

Hier gab es eine falsche Versionsangabe der auszucheckenden Projekt-Software mit Abbruch des Skriptes:

```
INFO:- Also completed EPA download to tempfolder
WARNING:GitTag version does not exist: v0.5
```

Angabe einer Versionsnummer, obwohl bisher keine getaggte Version existiert führt zu einer Fehlermeldung. Es wird stattdessen die aktuelle "Master" Version genutzt.

```
INFO:- Also completed EPA download to tempfolder
WARNING:- Should check out latest tagged version,
          but there is no tagged version! Continue anyway!
INFO:- Compiling locally on judge! Everything finished here!
```

5.4 Ausführen mittels Cron-Job

Cron ist ein Unix-Dienst zum zeitgesteuerten Ausführen von Programmen. Cron wird typischerweise für Systemaufräumarbeiten oder Datensicherungen genutzt, welche regelmäßig zu wiederkehrenden Zeitpunkten ausgeführt werden sollen. Wird RTK mittels Cron in der unten stehenden Konfiguration ausgeführt, wird die aktuellste, getaggte Version aus dem Repository an jedem ersten eines Monats um 10 Uhr ausgerollt. Die Konfigurationzeile wird via `crontab -e` in Cron hinterlegt.

```
00 10 1 * * /path/to/rtk.py -p projekt -f 'm1,u1,p1;m2,u2,p2'
```

Ein kritischer Nachteil der Nutzung von Cron ist, dass die Kennwörter im Klartext an RTK übergeben werden müssen und somit im Klartext auf der Festplatte gespeichert werden. Da die Cron-Konfigurationsdatei bei physikalischem Zugriff auf einen Rechner auch ohne Benutzerkennwort ausgelesen werden kann, ist dieser Nachteil üblicherweise nur dann vertretbar, wenn der physikalische Zugriff auf den Rechner für unbefugte ausgeschlossen werden kann. Dies ist zum Beispiel bei Computern mit Festplattenverschlüsselung oder Festplattenkennwort gegeben.[12]

6 Ausblick

6.1 Erweiterung des Programmes um Testläufe

Eine mögliche Erweiterung des Programmes stellt das automatische Ausführen von Testläufen der kompilierten Anwendung dar. Es lässt sich für jedes Programm eine Serie von Testdaten generieren, welche nach Abschluss der Kompilierung zur Durchführung eines Tests genutzt werden. Eine einfache Fehlererkennung stellt die Prüfung dar, ob das Programm im Anschluss ordnungsgemäß beendet wird[9] oder ob es mit einem Fehler abbricht. Weitergehende Fehlererkennungen könnten die erzeugten Daten mit Referenzwerten vergleichen, um beispielsweise Vorzeichenfehler oder große Abweichungen (Überläufe, vollkommen weiße/schwarze Ausgabebilder) zu erkennen. Darüber hinaus gehende Fehlererkennungen sind komplexer zu implementieren. Die Ergebnisdaten unterscheiden sich nach Programmänderung unter Umständen bewusst. Erkennungsroutinen sind daher nicht mehr oder nur schwer programmtechnisch abbildbar, sofern nicht ausschließlich Geschwindigkeitsoptimierungen implementiert wurden.

6.2 Ausführen mittels git-Hook

Ein git-Hook bezeichnet eine Funktion des Versionierungsprogrammes Git, beim Einchecken einer neuen Version ein Kommando auszuführen. So kann das Ausrollen der Programmversionen beim Einchecken automatisch gestartet werden. Der git-Hook stellt eine Alternative zum Cron-Job dar. Die Implementierung über einen git-Hook ist analog zur Ausführung über den Cron-Job problematisch in Hinsicht auf die im Klartext gespeicherten Passwörter.

6.3 Portierung nach Python 3

Python ist momentan in zwei parallel entwickelten Versionen verfügbar: Python 2 und Python 3. Aktuelle Haupt-Versionen sind Python 2.7 sowie Python 3.3. Die Spaltung der unterstützen Hauptversionen ist der Ursache geschuldet, dass beide Versionen in einigen Funktionen nicht kompatibel zueinander sind. RTK ist in Python 2.7 geschrieben. Die Ursache hierfür ist die fehlende Verfügbarkeit von Python 3 auf JuDGE. Es ist nicht zu erwarten, dass in nächster Zeit ein Update für JuDGE auf Python 3 zur Verfügung steht.

Einige von RTK verwendeten Module sind nicht für Python 3 verfügbar, hierzu gehört der "ConfigParser". Die entsprechenden Python 3 Module verwenden eine geänderte Syntax, welche im Falle einer Migration auf Python 3 angepasst werden muss. Da das Hauptprogramm selbst nicht unter JuDGE gestartet wird, ist grundsätzlich der Einsatz von Python 3 für rtk.py denkbar, solange rkyCompile.py weiterhin in einer Version für Python 2 zur Verfügung steht. Aufgrund der bisher noch breiten Unterstützung für Python 2 wurde davon abgesehen, verschiedene Teile von RTK in verschiedenen Python-Versionen zu entwickeln.[13]

[9] d.h. es liefert beim Beenden `exit 0` zurück

6.4 Fazit

Schon während des Erstgespräches mit dem Betreuer wurde offensichtlich, dass es sich bei dem Thema dieser Bachelorarbeit um ein sehr IT-nahes Thema handelt, welches in der Arbeitsgruppe Faserbahnarchitektur aufgrund der Dringlichkeit einen hohen Stellenwert besaß. Das fachübergreifende Forschungsgebiet mit physikalischen, medizintechnischen und informationstechnischen Anteilen passte gut zu meinem Studiengang. Der hohe informationstechnische Anteil innerhalb der Bachelorarbeit war im Rahmen meiner bisherigen IT-Ausbildung nicht unbedingt von Nachteil, dennoch hatte ich aufgrund meiner geringen Erfahrung in der Anwendungsprogrammierung die Möglichkeit, mich in neue Arbeitsbereiche einzuarbeiten. Das Lösen von technischen Problemen erledigte ich überwiegend selbstständig mit Hilfe von Dokumentationen, dennoch hatte man für Fragen, technisch als auch organisatorisch, stets ein offenes Ohr. In regelmäßigen Gesprächen wurde der aktuelle Status sowie aktuelle Probleme besprochen und gemeinsam nach Lösungen gesucht, sodass ich den Eindruck gewann, Teil des Teams zu sein.

Das von mir entwickelte Programm erfüllt die gestellten Anforderungen. Darüber hinaus wurde durch die Realisierung über leicht veränderliche Konfigurationsdateien die Handhabung sehr einfach gehalten. Der Lerneffekt auf meiner Seite war groß, daher halte ich die Bachelorarbeit für einen persönlichen Erfolg.

A Abkürzungsverzeichnis

ASCII American Standard Code for Information Interchange: Eine der ersten bedeutensten Zeichenkodierungen für englischen Text, heute noch teilweise im Einsatz.

CPU Central Processing Unit: Zentralprozessor eines Computers

Cron Zeitgesteuerte Ausführung von Programmen auf Linux/Unix Betriebssystemen

FOM Fiber Orientation Map: 3D Karte der Nervenfaserbahnen des Gehirns.

FQDN Full Qualified Domain Name: eindeutiger Gerätename für die Erreichbarkeit über das Netzwerk

FTP File Transfer Protokoll: Ein Netzwerkprotokoll zur Datenübertragung.

GPU Graphic Processing Unit: Prozessor der im Computer verbauten Graphikkarte

Man-in-the-Middle-Attacke Angriffsszenario für das Abhören und/oder Manipulieren von verschlüsselten Verbindungen: Der Angreifer schaltet sich zwischen Sender und Empfänger und gibt sich beim Sender als ordnungsgemäßer Empfänger aus, sowie beim Empfänger als ordnungsgemäßer Sender. Nachrichten werden zwischen beiden Teilnehmern weitergeleitet. Da der Angreifer nun zu beiden Seiten der Endpunkt einer verschlüsselten Verbindung ist, kann er die Nachrichten im Klartext lesen. Dieser Angriff wird durch das konsequente Überprüfen der öffentlichen Schlüssel unterbunden, da eine Veränderung des Gegenüber in diesem Fall augenscheinlich wird.

MIME Multipurpose Internet Mail Extensions: Standard für die Zeichen- und Anhangs-Kodierung von E-Mails

MPI Message Passing Interface: Standard für Nachrichtenaustausch bei parallelem Rechnen auf HPC-Clustern.

MSUB Konfigurationdatei-Format für das Moab Job-Verwaltungssystem

PBS Portable Batch System: Computer Software zur Job-Verwaltung

PLI Polarized Light Imaging: Ein in vitro Verfahren zur Bestimmung der Raumrichtung der Nervenfaserbahnen.

RAM Read-Only Memory: flüchtiger Arbeitsspeicher eines Computers; dient zur Speicherung aktiver Programme

Samba Sprachliche Umschreibung des Protokolls SMB (Server Message Block): Ein Netzwerkprotokoll der Firma Microsoft, dient im Betriebssystem Windows als Netzwerkprotokoll für Dateiübertragungen. Portierungen für andere Betriebssysteme sind verfügbar. Aktuelle Version ist 3.0

SCP Secure Copy: Ein Datenübertragungsverfahren im Netzwerk, welches auf das Netzwerkprotokoll SSH zurückgreift. Es bietet durch SSH eine Datenverschlüsselung bei der Übertragung, ist aber langsam. Es eignet sich vorwiegend zur Übertragung einzelner Dateien geringer Größe.

SMTP Simple Mail Transfer Protocol: Netzwerkprotokoll zum Versenden von E-Mails

SSH Secure Shell: Ein verschlüsseltes Netzwerkprotokoll für die Steuerung von entfernten Rechnern über eine Kommandozeile.

Webdav Netzwerkprotokoll für Dateiübertragungen. Basiert auf dem Webseiten-Übertragungsprotokoll http (Hypertext Transfer Protocol)

Literatur

[1] Webseite des INM-1 des Forschungszentrum Jülich
http://www.fz-juelich.de/inm/inm-1/DE/Home/home_node.html

[2] Masterarbeit Marcel Huysegoms: Simultane Registrierung histologischer Bilder des Gehirns unter Anwendung von Markov Random Fields

[4] Amunts, K., C. Lepage, L. Borgeat, H. Mohlberg, T. Dickscheid, M.-E. Rousseau, S. Bludau, P.-L. Bazin, L. B. Lewis, A.-M. Oros-Peusquens, N. J. Shah, T. Lippert, K. Zilles, and A. C. Evans (2013). BigBrain: An Ultrahigh-Resolution 3D Human Brain Model. Science 340 (6139), 1472 - 1475.

[5] Webseite des Forschungszentrum Jülich über Judge http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUDGE/Configuration/Configuration_node.html

[6] Quellcodedokumentation für pxssh
http://pexpect.readthedocs.org/en/latest/_modules/pexpect/pxssh.html

[7] Mehrprozessorsysteme und Parallelverarbeitung
http://www.controllersandpcs.de/lehrarchiv/pdfs/hs/MP_1.pdf

[8] Basics of SIMD Programming auf Kernel.org
https://www.kernel.org/pub/linux/kernel/people/geoff/cell/ps3-linux-docs/CellProgrammingTutorial/BasicsOfSIMDProgramming.html

[9] Alexander aus der Fünten: "Versionsverwaltung: Git & SVN" der RWTH Aachen,
http://dme.rwth-aachen.de/en/system/files/file_upload/course/12/proseminar-methoden-und-werkzeuge/versionsverwaltung.pdf

[10] Elektronik-Kompendium: Unterschied zwischen Compiler- und Interpretersprache
http://www.elektronik-kompendium.de/sites/com/1705231.htm

[11] HowTo des FZ Jülich: Jobsystem auf Judge http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUDGE/Userinfo/Quick_Introduction.html

[12] Crontab Manpage http://unixhelp.ed.ac.uk/CGI/man-cgi?crontab

[13] Mehrprozessorsysteme und Parallelverarbeitung
https://wiki.python.org/moin/Python2orPython3

Alle Internetquellen wurden zuletzt aufgerufen am 11.01.2015

B Quellcode

B.1 rtk.py

```
#!/usr/bin/python2
################################## Main Module ###################################
import os                           # Shell-util
import shutil                       # Shell-util to remove folders
import tempfile                     # Create an unique tempfile
import pexpect                      # Run Shell-command interactively
import ConfigParser                 # Read external .conf files
import datetime                     # timestamp
import re                           # Search with regular expressions
import logging                      # Logging-Function
import pxssh                        # Establish SSH-connection automatically
import getpass                      # Get username/password readline input
import rtkFeedback                  # Sub-Function for sending e-mails
from optparse import OptionParser # Parse command-line parameter

##### Exit on error
def badExit(quitmessage):
   logging.warning(quitmessage)
   rtkFeedback.feedback("Failure", subject, rtkLogFile, eMailSender, eMailReceiver,
       eMailServer)
   logging.shutdown()
   exit(1)

##### Parse Configuration-Files
configProject = ConfigParser.ConfigParser()
configProject.read("project.conf")

configMachine = ConfigParser.ConfigParser()
configMachine.read("machine.conf")

configRTK = ConfigParser.ConfigParser()
configRTK.read("rtk.conf")

timeStamp = datetime.datetime.now().isoformat()

try:
   gitMachine=configRTK.get("git", "gitMachine")
   gitRtkRepo=configRTK.get("git", "gitArchive")

   eMailServer=configRTK.get("mail", "eMailServer")
   rtkLogFile=configRTK.get("local", "rtkLogFile") + timeStamp
except ConfigParser.NoOptionError as err:
   print("- Error in rtk.conf: " + str(err))
   exit(1)

##### Parse CLI-Arguments and print help
optionParser = OptionParser("rtk-main [-m] machine [-p] project")
optionParser.add_option("-m", "--machine", dest="machine", default="all", help="target"
                        " machine(s), example: local,ime263; default: all (allowed machines)")
optionParser.add_option("-p", "--project", dest="project", help="target project, chose only"
                        " one")
optionParser.add_option("-u", "--user", dest="username", default=getpass.getuser(), help="ssh"
                        " username, type '-u ask' to ask for each machine, default:"
                        " use system username")
optionParser.add_option("-g", "--generic-password", dest="genericPass", help="Set this to '-g"
                        " True' to use the same, generic password for each login-machine and"
                        " enter it only once, this is NOT for entering your password")
optionParser.add_option("-d", "--disable-rtkupdate", dest="disableRtkUpdate", help="Set this"
                        " to '-d True' to disable the update of your local rtk version")
optionParser.add_option("-l", "--log", dest="loglevel", default="INFO", help="set loglevel:"
                        " debug, info, warning default: INFO")
optionParser.add_option("-t", "--tag", dest="gitTag", default="True", help="Can set to True"
                        " (use last tagged version), False (ignore tags, instead use trunk"
                        " version / branch master) or a specific tag-name, default: True")
optionParser.add_option("-f", "--feed-passlib", dest="passlib", default="False", help="Feed"
```

```
65                          " Userdata directly into passlib. DONT USE THIS FOR SECURITY REASON!"
66                          " Don't forget the gitArchive! Syntax:"
67                          " -f 'judge,judgeUser,judgePass;ime263,ime263User,ime263Pass'")
68
69 (options, args) = optionParser.parse_args()
70
71 ##### Get Loglevel, set local logfile and add additional streamlogger (std.out)
72 logging.basicConfig(level=options.loglevel.upper(), filemode='w',
73                     filename=rtkLogFile, format='%(levelname)s:%(message)s')
74 logger = logging.getLogger()
75 logger.addHandler(logging.StreamHandler())
76 logging.info("- Logging activated!")
77
78 ##### Check Project and load Project specific variables from configFile
79 if not options.project in configProject.sections():
80    logging.warning("- Project '" + options.project + "' not found in project.conf."
81                    " Project is case sensitive.")
82    logging.shutdown()
83    exit(1)
84 try:
85    subject="RTK: " + options.project + " Main Thread"
86    eMailReceiver=configProject.get(options.project, "eMailReceiver")
87    eMailSender=configProject.get(options.project, "eMailSender")
88    install=configProject.get(options.project, "install")
89    gitProjectRepo=configProject.get(options.project, "gitRepo")
90    gitProto=configProject.get(options.project, "gitProto")
91 except ConfigParser.NoOptionError as err:
92    badExit("- Error in project.conf: " + str(err))
93
94 ##### Check for programming-language
95 if not install.upper() == "YES" and not install.upper() == "NO" and not install.upper() == "
       COPY":
96    badExit("- To recognize programming-language of the project, set in project.conf"
97            " install=yes or install=no (both C) or install=copy (Python). Currently install="
98            + install)
99
100 ##### Parse CLI Parameter and join with config-files
101 if options.machine == "all":
102     options.machine = set(configProject.get(options.project, "allowMachines").split(','))
103 else:
104     options.machine = set(options.machine.split(','))
105     if options.machine.difference(set(configProject.get(options.project, "allowMachines").
          split(','))):
106        logging.warning("- Machine below not in allowMachine in project.conf, exit")
107        badExit(str(options.machine.difference(set(configProject.get(options.project,
108                    "allowMachines").split(',')))))
109
110 logging.info("- CLI parameter parsed, checked config files successfully for your given"
111              " project and machines!")
112
113 ##### Check for non-existent machine in allowedMachine
114 if set(configProject.get(options.project, "allowMachines").split(',')).difference(set(
       configMachine.sections())):
115    logging.warning("- Error in project.conf: non-existent machine in allowMachines in"
116                    " project " + options.project)
117    badExit(str(set(configProject.get(options.project, "allowMachines").split(',')).difference(
          set(configMachine.sections()))))
118
119 logging.info("- Your given projects are in allowed Machines! That's good!")
120
121 ##### Get ssh credentials for each remote machine and - if neccessary - for the gitMachine
122 passlib={}
123 if options.passlib.upper() == "FALSE":
124
125    #### Read password via CLI
126    if options.genericPass:
127      options.genericPass = getpass.getpass("You've said you have one master password for each"
128                                            " machine. Please enter it now:")
129      logging.info("- You have entered a masterpassword for each machine! I will use that in"
130                   " future!")
131
132    for machine in options.machine:
```

```
133       if not options.username == "ask" and options.genericPass:
134          passlib[machine] = [options.username, options.genericPass]
135          continue
136       print ("Please enter SSH credentials for machine " + machine)
137       if options.username == "ask" and not options.genericPass:
138          passlib[machine] = [raw_input("Please enter username: "),
139                              getpass.getpass("Please enter password: ")]
140       if not options.username == "ask" and not options.genericPass:
141          passlib[machine] = [options.username, getpass.getpass("Please enter password for "
142                                                                + options.username + ":")]
143       if options.username == "ask" and options.genericPass:
144          passlib[machine] = [raw_input("Please enter username: "), options.genericPass]
145    logging.info("- I got all your credentials for your given machines!"
146                 " Thank you for helping me doing my job!")
147 else:
148
149   #### Use given password database through CLI command "-f" ...
150    logging.info("- Using external CLI passlib!")
151    for tempString in options.passlib.split(';'):
152       passlib[tempString.split(',')[0]] = [tempString.split(',')[1], tempString.split(',')[2]]
153
154    #### ... and check if a machine is missing!
155    for machine in options.machine:
156       if not machine in passlib.keys():
157          badExit("- You feed me with an external passlib but forget to tell me the password"
158                  " of " + machine)
159    logging.info("- Passlib successfully imported!")
160
161 ##### Check and ask for the gitMachine Credentials
162 if gitMachine in options.machine:
163    gitCredentials = [passlib[gitMachine][0], passlib[gitMachine][1]]
164 elif not options.passlib.upper() == "FALSE":
165    if not gitMachine in passlib.keys():
166       badExit("- You feed me with an external passlib but forget to tell me the password of"
167               " the gitMachine " + gitMachine)
168    gitCredentials = [passlib[gitMachine][0], passlib[gitMachine][1]]
169 elif not options.username == "ask" and options.genericPass:
170    gitCredentials = [options.username, options.genericPass]
171 else:
172    print ("Please enter SSH credentials for machine " + gitMachine + ":")
173    if options.username == "ask" and not options.genericPass:
174        gitCredentials = [raw_input("Please enter username: "),
175                          getpass.getpass("Please enter password: ")]
176    if not options.username == "ask" and not options.genericPass:
177        gitCredentials = [options.username, getpass.getpass("Please enter password for "
178                          + options.username + ":")]
179    if options.username == "ask" and options.genericPass:
180        gitCredentials = [raw_input("Please enter username: "), options.genericPass]
181
182 print ("- Got your passwords successfully!")
183 logging.info("- I now have also the credentials for the git Archive!")
184
185 ##### Check, if RTK is up to date
186 if not options.disableRtkUpdate:
187   currentRtkTemp = tempfile.mkdtemp()
188   child = pexpect.spawn('git clone ssh://' + gitCredentials[0] + "@" + gitRtkRepo + " "
189                         + currentRtkTemp)
190   child.expect(["[pP]assword:","passphrase.*rsa'","passphrase.*pub'"])
191   child.sendline(gitCredentials[1])
192   child.expect(pexpect.EOF)
193   if not os.popen("cd " + currentRtkTemp +"; git log -1").readline() == os.popen("git log -1")
          .readline():
194      logging.warning("- RTK is not up-to-date! I'll try to load the current version!"
195                      "If this is your second time you get this message,"
196                      " try a 'git pull' manually!")
197      child = pexpect.spawn('git pull')
198      if child.expect (':')==0:
199        child.sendline(gitCredentials[1])
200        child.expect(pexpect.EOF)
201      badExit("- I have found an update of RTK and tried to download it."
202              " I will exit now! Please restart with same parameters!")
203   shutil.rmtree(currentRtkTemp, ignore_errors=True)
```

```
204   logging.info("- I have checked for a local RTK update and will go on now!")
205 else:
206   logging.info("- Check for local RTK update skipped due to users choice!")
207
208 ##### Iterate over machines and establish ssh connection
209 for machine in options.machine:
210
211    #### Both: Get hostUrl
212    try:
213       hostUrl=configMachine.get(machine, "hostUrl")
214    except ConfigParser.NoOptionError as err:
215       logging.warning("- Error in machine.conf with machine " + machine + ": No hostURL found"
216                       " (" + str(err) + ") I'll go on with the next machine!")
217       rtkFeedback.feedback("Failure", subject, rtkLogFile, eMailSender, eMailReceiver,
218                            eMailServer)
219       continue
220    logging.info("- I'll try now to log in to machine " + machine)
221
222    #### Both: Establish SSH connection
223    hostConnect = pxssh.pxssh()
224    try:
225       hostConnect.login(hostUrl, passlib[machine][0], passlib[machine][1])
226    except pexpect.pxssh.ExceptionPxssh as err:
227       logging.warning("- SSH session failed on login.")
228       logging.warning(str(err))
229       logging.debug(str(hostConnect))
230       ### Retry to connect if "could not synchronize with original prompt" a second time
231       if str(err) == "could not synchronize with original prompt":
232          try:
233             logging.info("- Retrying to connect")
234             hostConnect = pxssh.pxssh()
235             hostConnect.login(hostUrl, passlib[machine][0], passlib[machine][1])
236          except pexpect.pxssh.ExceptionPxssh:
237             logging.warning("- SSH session session initiation failed two times. Aborting.")
238             logging.warning(str(err))
239             logging.debug(str(hostConnect))
240             rtkFeedback.feedback("Failure", "RTK:" + options.project + " on " + machine,
241                                  rtkLogFile, eMailSender, eMailReceiver, eMailServer)
242             continue
243       else:
244          rtkFeedback.feedback("Failure", "RTK:" + options.project + " on " + machine,
245                               rtkLogFile, eMailSender, eMailReceiver, eMailServer)
246          continue
247    logging.info("- SSH session login successful on " + machine)
248
249    #### Python: If language is Python
250    if install.upper() == "COPY":
251       logging.info("- Project Language is Python")
252
253       ### Python: Get variables
254       try:
255          projectRemoteFolder=configProject.get(options.project, "pythonDest." + machine)
256       except ConfigParser.NoOptionError as err:
257          logging.warning("- Error in project.conf with machine " + machine + ": (" + str(err)
258                          + ") I'll go on with the next machine!")
259          rtkFeedback.feedback("Failure", subject, rtkLogFile, eMailSender, eMailReceiver,
260                               eMailServer)
261          continue
262
263       ### Python: Download project-git archive to projectRemoteFolder
264       if gitProto == "ssh":
265          hostConnect.sendline('git clone ssh://' + gitCredentials[0] + '@' + gitProjectRepo
266                               + ' ' + projectRemoteFolder)
267       else:
268          hostConnect.sendline('git clone git://' + gitProjectRepo + ' ' + projectRemoteFolder)
269       sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'","Are you sure you want"
270                                    " to continue connecting", "fatal:"])
271       if sshMsg == 3:
272         hostConnect.sendline()
273       if sshMsg == 2:
274         hostConnect.sendline('yes')
275         sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'"])
```

```
276       if sshMsg == 1:
277         hostConnect.sendline(gitCredentials[1])
278       if sshMsg == 0:
279         hostConnect.sendline(gitCredentials[1])
280       hostConnect.prompt()
281       logging.debug(hostConnect.before)
282
283       hostConnect.sendline('cd ' + projectRemoteFolder + '; git checkout master; git reset'
284                            ' --hard HEAD; git pull')
285       sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'","Are you sure you want"
286                                    " to continue connecting", "fatal:"])
287       if sshMsg == 3:
288         logging.warning("- Could not 'git pull' current version!")
289         rtkFeedback.feedback("Failure", subject, rtkLogFile, eMailSender, eMailReceiver,
290                              eMailServer)
291         continue
292       if sshMsg == 2:
293         hostConnect.sendline('yes')
294         sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'"])
295       if sshMsg == 1:
296         hostConnect.sendline(gitCredentials[1])
297       if sshMsg == 0:
298         hostConnect.sendline(gitCredentials[1])
299
300       hostConnect.prompt()
301       logging.debug(hostConnect.before)
302       logging.info("- Project " + options.project + "downloaded to " + projectRemoteFolder)
303
304   #### C: If language is C
305   if not install.upper() == "COPY":
306       logging.info("- Project Language is C")
307
308       ### C: Get variables
309       if "usejobsystem" in configMachine.options(machine):
310          useJobSystem=configMachine.get(machine, "useJobSystem")
311       else:
312          useJobSystem="no"
313          logging.warning("- Variable 'useJobSystem' not set in machine.conf for " + machine
314                          + ", i won't use the Job System")
315       if "importcommand" in configMachine.options(machine):
316          importCommand=configMachine.get(machine, "importCommand")
317       else:
318          importCommand=" "
319          logging.info("- Variable 'importCommand' not set in machine.conf for " + machine)
320       cMakeVar="cmakeenv." + machine
321       if cMakeVar in configProject.options(options.project):
322          cMakeEnv=configProject.get(options.project, cMakeVar)
323       else:
324          cMakeEnv=" "
325
326       ### C: Create Tempfolder on Remote site
327       if "tempdir" in configMachine.options(machine):
328          hostConnect.sendline("echo " + configMachine.get(machine, "tempDir"))
329          hostConnect.prompt()
330          rtkRemoteFolder=re.search('\n(/[a-zA-z0-9.-:]+)', hostConnect.before).group(1) + "/
               rtk." + timeStamp
331          logging.debug(hostConnect.before)
332       else:
333          hostConnect.sendline("mktemp -d")
334          hostConnect.prompt()
335          logging.debug(hostConnect.before)
336          try:
337             rtkRemoteFolder = re.search('(/tmp/[a-zA-z0-9.]+)', hostConnect.before).group(1)
338          except AttributeError as err:
339             logging.warning("- Unable to create Tempfolder on " + machine)
340             rtkFeedback.feedback("Failure", "RolloutToolkit on " + machine, rtkLogFile,
341                                  eMailSender, eMailReceiver, eMailServer)
342             continue
343          logging.info("- I have created a temporary folder called " + rtkRemoteFolder)
344       projectRemoteFolder=rtkRemoteFolder + '/' + options.project
345
346       ### C: Download rtk-git archive to rtkRemoteFolder
```

```
347       hostConnect.sendline('git clone ssh://' + gitCredentials[0] + '@' + gitRtkRepo + ' '
348                            + rtkRemoteFolder)
349       sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'","passphrase.*pub'",
350                                    "Are you sure you want to continue connecting"])
351       if sshMsg == 3:
352         hostConnect.sendline('yes')
353         sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'","passphrase.*pub'"])
354       if sshMsg == 2:
355         hostConnect.sendline()
356         sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'"])
357       if sshMsg == 1:
358         hostConnect.sendline(gitCredentials[1])
359       if sshMsg == 0:
360         hostConnect.sendline(gitCredentials[1])
361       hostConnect.prompt()
362       logging.debug(hostConnect.before)
363       logging.info("- Download rtk-git archive to Tempfolder " + rtkRemoteFolder + " on "
364                    + machine + " completed")
365
366       ### C: Download project-git archive to projectRemoteFolder
367       if gitProto == "ssh":
368          hostConnect.sendline('git clone ssh://' + gitCredentials[0] + '@' + gitProjectRepo
369                               + ' ' + projectRemoteFolder)
370          sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'","passphrase.*pub'"])
371          if sshMsg == 2:
372            hostConnect.sendline()
373            sshMsg = hostConnect.expect(["[pP]assword:","passphrase.*rsa'"])
374          if sshMsg == 1:
375            hostConnect.sendline(gitCredentials[1])
376          if sshMsg == 0:
377            hostConnect.sendline(gitCredentials[1])
378       else:
379          hostConnect.sendline('git clone git://' + gitProjectRepo + ' ' + projectRemoteFolder)
380       hostConnect.prompt()
381       logging.debug(hostConnect.before)
382       logging.info("- Also completed " + options.project + " download to tempfolder")
383
384    #### Both: Checkout of a tagged version, depends on gitTag variable
385    if options.gitTag.upper() == "TRUE":
386       hostConnect.sendline('cd ' + projectRemoteFolder + '; git checkout `git tag | tail -1`')
387       hostConnect.prompt()
388       logging.debug(hostConnect.before)
389       try:
390          gitTag = re.search("'([a-zA-z0-9.]+)'", hostConnect.before).group(1)
391          logging.info("- Checked out latest tagged version: " + gitTag)
392       except AttributeError as err:
393         gitTag = "No tagged version found!"
394         logging.warning("- Should check out latest tagged version, but there is no tagged"
395                         " version! Continue anyway!")
396    elif options.gitTag.upper() == "FALSE":
397       logging.info("- Option '-t' set to False, so i did not check out the latest tagged"
398                    " version")
399    else:
400       try:
401          gitTag = re.search('(^[a-zA-z0-9.]+)', options.gitTag).group(1)
402       except AttributeError as err:
403          badExit("Bad characters in gitTag " + options.gitTag + ". Please don't try to hack"
404                  " me!")
405       if gitTag == options.gitTag:
406          hostConnect.sendline('cd ' + projectRemoteFolder + '; git checkout ' + gitTag)
407          hostConnect.prompt()
408          logging.debug(hostConnect.before)
409          try:
410             if re.search('([Ee]rror)', hostConnect.before).group(1):
411                badExit("GitTag version does not exist: " + gitTag)
412          except AttributeError as err:
413             logging.info("- Checked out version " + gitTag)
414       else:
415          badExit("Bad characters in gitTag " + options.gitTag + ". Please don't try to hack"
416                  " me!")
417
418    #### C: Start sub-program for compiling locally on remote machine
```

```
419    if not install.upper() == "COPY":
420       hostConnect.sendline("nohup " + rtkRemoteFolder +"/rtkCompile.py '" + useJobSystem
421                            + "' '" + install + "' '" + machine + "' '" + options.project
422                            + "' '" + rtkRemoteFolder + "' '" + cMakeEnv + "' '"
423                            + importCommand + "' '" + eMailServer + "' '" + eMailSender + "' '"
424                            + eMailReceiver + "' > " + rtkRemoteFolder + "/rtkCommandOutput &")
425       hostConnect.prompt()
426       logging.debug(hostConnect.before)
427       logging.info("- Compiling locally on " + machine + "! Everything finished here!")
428
429    #### Both: ssh-logout
430    try:
431       hostConnect.logout()
432       logging.info("- Logging out from " + machine)
433    except pexpect.TIMEOUT as err:
434       logging.warning("- Logout from " + machine + " failed, continue anyway!")
435       logging.debug(str(err))
436
437 ##### Send feedback via mail
438 logging.shutdown()
439 rtkFeedback.feedback("Success", subject, rtkLogFile, eMailSender, eMailReceiver, eMailServer)
```

B.2 rtkCompile.py

```
1  #!/usr/bin/python2
2  ########################## Compile Module for compiling project ##########################
3  import os                          # Shell-util
4  import rtkFeedback                 # Sub-Function for sending e-mails
5  import re                          # Search with regular expressions
6  import sys                         # Import CLI Parameter
7
8  def badExit(quitmessage):
9     logger.write(quitmessage)
10    logger.close()
11    rtkFeedback.feedback("Failure", subject, rtkLogList , eMailSender, eMailReceiver,
          eMailServer)
12    exit(1)
13
14 ##### Read CLI Parameters given by rtk.py
15 useJobSystem=sys.argv[1]
16 install=sys.argv[2]
17 machine=sys.argv[3]
18 project=sys.argv[4]
19 rtkLocalDir=sys.argv[5]
20 cMakeEnv=sys.argv[6]
21 importCommand=sys.argv[7]
22 eMailServer=sys.argv[8]
23 eMailSender=sys.argv[9]
24 eMailReceiver=sys.argv[10]
25
26 ##### Create Variables
27 subject="RTK: " + project + " on " + machine
28 buildPath=rtkLocalDir + "/" + project + "/build"
29 rtkLogFile=rtkLocalDir + "/rtkLogFile"
30 rtkJobLog=rtkLocalDir + "/rtkJobLog"
31 rtkJob=rtkLocalDir + "/rtkJob.sh"
32 if not useJobSystem.upper()=="NO":
33    rtkLogList=rtkLogFile + ";" + rtkJob + ";" + rtkLocalDir + "/rtkCommandOutput"
34 else:
35    rtkLogList=rtkLogFile + ";" + rtkLocalDir + "/rtkCommandOutput"
36
37 ##### Activate Logging
38 logger = open(rtkLogFile, 'w')
39
40 ##### Check installed CMake version
41 try:
42    cmakeInst = os.popen('cmake --version').read()
43    cmakeInstVersion = re.search('(\d+).(\d+).(\d+).?(\d*)', cmakeInst)
```

```
44    if cmakeInstVersion:
45       logger.write("- Installed CMake-Version found:" + cmakeInstVersion.group(1) + "."
46                    + cmakeInstVersion.group(2) + "." + cmakeInstVersion.group(3) + "."
47                    + cmakeInstVersion.group(4) + "\n")
48    else:
49       if os.popen('which cmake').read():
50          logger.write("- CMake found, but CMake version could not identified,"
51                       " trying anyway...\n")
52       else:
53          badExit("- CMake not found! Aborting...")
54 except OSError as err:
55     logger.write("- Error: %s - %s." (err.filename,err.strerror) + "\n")
56     logger.write("- An error occured while getting the installed CMake Version!"
57                  " Ignore Check and go on...\n")
58
59 ##### Check required CMake version
60 try:
61    cmakeNeed = open(rtkLocalDir + "/" + project + "/CMakeLists.txt").readlines()
62    for line in cmakeNeed:
63       cmakeNeedVersion = re.search('CMAKE_MINIMUM_REQUIRED.*(\d+).(\d+).(\d+).?(\d*)', line)
64       if cmakeNeedVersion:
65          logger.write("- Required CMake-Version " + cmakeNeedVersion.group(1) + "."
66                       + cmakeNeedVersion.group(2) + "." + cmakeNeedVersion.group(3)
67                       + "." + cmakeNeedVersion.group(4)+ "\n")
68          break
69    else:
70       logger.write("- Required CMake version could not identified, trying anyway...\n")
71 except OSError as err:
72     logger.write("- Error: %s - %s." (err.filename,err.strerror)+ "\n")
73     logger.write("- An error occured while getting the required CMake Version!"
74                  " Ignore Check and go on...\n")
75
76 ##### Check if installed CMake is not less than required version
77 if cmakeNeedVersion and cmakeInstVersion:
78     if (int(cmakeInstVersion.group(3)) + int(cmakeInstVersion.group(2)) * 100
79         + int(cmakeInstVersion.group(1)) * 10000) < (int(cmakeNeedVersion.group(3))
80         + int(cmakeNeedVersion.group(2)) * 100 + int(cmakeNeedVersion.group(1)) * 10000):
81            badExit("- CMake version installed is less than CMake version required."
82                    " Please update CMake version, or try to change CMAKE_MINIMUM_REQUIRED"
83                    " in CMakeLists.txt")
84
85 ##### Run CMake
86 exitCode = os.system("mkdir -p " + buildPath + "; cd " + buildPath + "; cmake "
87                      + cMakeEnv + " ..")
88 if not exitCode == 0:
89    badExit("- CMake terminated with following Exit-Code:" + str(exitCode))
90 logger.write("- CMake successfully done!\n")
91
92 ##### Check useJobSystem and run 'make' and optionally run make install
93 if not useJobSystem.upper()=="NO":
94    with open(rtkJob, "a") as myRtkJob:
95       myRtkJob.write("#" + useJobSystem.upper() + " -S /bin/bash\n")
96       myRtkJob.write("#" + useJobSystem.upper() + " -N RTK\n")
97       myRtkJob.write("#" + useJobSystem.upper()
98                      + " -l nodes=1:ppn=2,mem=2gb,walltime=02:00:00\n")
99       myRtkJob.write("#" + useJobSystem.upper() + " -j oe\n")
100      myRtkJob.write("#" + useJobSystem.upper() + " -m ae\n")
101      myRtkJob.write("#" + useJobSystem.upper() + " -o " + rtkJobLog + "\n")
102      myRtkJob.write("#" + useJobSystem.upper() + " -e " + rtkJobLog + "\n")
103      myRtkJob.write("#" + useJobSystem.upper() + " -M " + eMailReceiver + "\n")
104      myRtkJob.write("cd " + buildPath + "\n")
105      myRtkJob.write(importCommand + "\n")
106      myRtkJob.write("make\n")
107      if install.upper()=="YES": myRtkJob.write("make install\n")
108
109   if useJobSystem.upper() == "MSUB": JobID = os.popen("msub " + rtkJob).read()
110   elif useJobSystem.upper() == "PBS": JobID = os.popen("qsub " + rtkJob).read()
111   else: badExit("- Job System in config file is neither PBS nor MSUB")
112   logger.write ("- Use job system for make JobID is:" + JobID + ", FULL LOGFILE AT: "
113                 + rtkJobLog + "\n")
114
115 else:
```

```
116    if importCommand == " ":
117       exitCode = os.system("cd " + buildPath + "; make")
118    else:
119       exitCode = os.system("cd " + buildPath + ";" + importCommand + "; make")
120    if not exitCode == 0:
121       badExit("- Make terminated with following Exit-Code:" + str(exitCode))
122    logger.write("- Run make without job system, make successfully done!\n")
123    if install.upper()=="YES":
124       exitCode = os.system("cd " + buildPath + "; make install")
125       if not exitCode == 0:
126          badExit("- Make install terminated with following Exit-Code:" + str(exitCode))
127       logger.write("- Completed: make install!\n")
128    else:
129       logger.write("- I should not run make install, so i didn't!\n")
130
131 ##### Send Status Message
132 rtkFeedback.feedback("Success", subject, rtkLogList, eMailSender, eMailReceiver, eMailServer)
```

B.3 rtkFeedback.py

```
1  ######################### Feedback Module for sending status mails ########################
2  import smtplib                        # Mail sending library
3  from email.mime.text import MIMEText # Convert text into mime format
4
5  def feedback(success, subject, logfile, eMailSender, eMailReceiver, eMailServer):
6     # Read Logfile
7     logfileList = logfile.split(';')
8     readIn = ""
9     for log in logfileList:
10       fp = open(log, 'rb')
11       readIn += "\n############" + log + "#############\n" + fp.read()
12       fp.close()
13    msg = MIMEText(readIn)
14    eMailReceiverList = eMailReceiver.split(',')
15    # Create Mail
16    msg['Subject'] = success + ": " + subject
17    msg['From'] = eMailSender
18    msg['To'] = str(eMailReceiverList)
19
20    # Send the message via SMTP server
21    smtpObj = smtplib.SMTP(eMailServer)
22    smtpObj.ehlo()
23    smtpObj.sendmail(eMailSender, eMailReceiverList, msg.as_string())
24    smtpObj.quit()
```

B.4 rtk.conf

```
1  # The server/archive are used only for RTK itself, but the credentials in the program are used
       for each ssh-based git repository
2  [git]
3  gitMachine=ime263
4  gitArchive=ime263.ime.kfa-juelich.de/data/PLI-Group/repositories/rtk.git
5
6  # A unique date string will additionally be attached to the rtkLogFile filename
7  [local]
8  rtkLogFile=/tmp/rtkLogFile
9
10 # Mailserver
11 [mail]
12 eMailServer=mail.fz-juelich.de
```

B.5 project.conf

```
#### projects ####
# [EPA]                                     # Case sensitive
# gitRepo=server.de/path/to/epa.git         # FQDN and path of the projects git repository
# gitProto=ssh                              # gitProto=ssh or gitProto=git
# allowMachines=ime263,local,judge          # Comma-separate all allowed machines
# install=yes [no] or install=copy          # Run 'make install' (C-only) or copy to specific
                                            # file (Python-only)
# cMakeEnv.judge=ITK_DIR=/path/to/ITK/      # cmake environment variables for the specific
                                            # system, only for C
# pythonDest.judge=/path/to/python          # Enter the python directory for specific machine
# eMailSender=ch.krause@fz-juelich.de       # chose one fz-juelich.de adress
# eMailReceiver=ch.krause@fz-juelich.de     # Comma-separate multiple adresses if you want

[EPA]
gitRepo=ime263.ime.kfa-juelich.de/data/PLI-Group/repositories/epa.git
gitProto=ssh
allowMachines=ime263,local,judge,imedv18
eMailSender=ch.krause@fz-juelich.de
eMailReceiver=ch.krause@fz-juelich.de
install=yes
cMakeEnv.ime263=-DITK_DIR=/opt/PLISoft/InsightToolkit-4.5.2/build-Release-With_Reviews/
cMakeEnv.judge=-DITK_DIR=/usr/local/registration/software/ITK/InsightToolkit-4.6.0/build/

[ITK]
gitRepo=itk.org/ITK.git
gitProto=git
allowMachines=ime263,local,judge
install=no
eMailSender=ch.krause@fz-juelich.de
eMailReceiver=ch.krause@fz-juelich.de

[RTK]
gitRepo=ime263.ime.kfa-juelich.de/data/PLI-Group/repositories/rtk.git
gitProto=ssh
allowMachines=ime263,local,judge
eMailSender=ch.krause@fz-juelich.de
eMailReceiver=ch.krause@fz-juelich.de
install=copy
pythonDest.ime263=/tmp/rtktemp
pythonDest.judge=$WORK/temptemp
```

B.6 machine.conf

```
#### machines ####
#[ime263]                            # Case sensitive
#hostUrl=ime263.ime.kfa-juelich.de   # FQDN of the machine to connect via SSH
#useJobSystem=no                     # If set to MSUB or PBS, the job system
                                     # will be used (only for C, not for Python)
#tempDir=/work  or tempDIR=$WORK     # Optionally use different tempDir, if not
                                     # set use 'mktemp -d' (only for C, not for Python)
#importCommand=source /path/bla      # Run this command to import variables etc.
                                     # before 'make' (only for C, not for Python)

[ime263]
hostUrl=ime263.ime.kfa-juelich.de
useJobSystem=PBS

[judge]
hostUrl=judge.fz-juelich.de
useJobSystem=MSUB
tempDir=$WORK
importCommand=source /usr/local/jsc-inm/registration/bashrc

[local]
hostUrl=localhost
useJobSystem=no

[imedv18]
hostUrl=imedv18.ime.kfa-juelich.de
useJobSystem=no
```

Erklärung

Hiermit versichere ich, dass ich die Arbeit eigenständig verfasst und keine Quellen außer den angegebenen verwendet habe. Alle Ausführungen aus fremden Quellen wurden kenntlich gemacht. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Dormagen, den 12. Januar 2015

Christian Krause