Workflow-gestützte Bereitstellung von Grid Middleware-Diensten

bei der Fakultät für Elektrotechnik und Informationstechnik

der TU Dortmund eingereichte

Dissertation

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

vorgelegt von

Dipl.-Inf. Stefan Freitag

Referent: Prof. Dr.-Ing. Uwe Schwiegelshohn, TU Dortmund
Korreferent: Prof. Dr. Achim Streit, Karlsruhe Institute of Technology
Tag der mündlichen Prüfung: 20.12.2012

Für Birgit.

Danksagung

Was lange währt, wird endlich gut! Ganz in diesem Sinne wäre die Dissertation ohne eine tiefgründige Einarbeitung in die behandelten Themen des Grid und Cloud Computing sowie der oftmals darunterliegenden Virtualisierung nicht möglich gewesen. Neben den theoretischen Aspekten bildet die Administration der zeitweise am Institut für Roboterforschung beheimateten D-Grid Referenzinstallation sowie des D-Grid Ressourcen Zentrums Ruhr eine weitere, praktische Grundlage für die erzielten Ergebnisse.

Mein Dank gilt in erster Linie Prof. Dr. -Ing. Uwe Schwiegelshohn, der mir die Bearbeitung mehrerer D-Grid-relevanter Themen, wie die „Entwicklung von Scheduling-Strategien zur effizienten Nutzung von Grid-Ressourcen“ als Teil des High Energy Physics Community Grid-Projekts und die „Expansion of the D-Grid Foundation for Commercial Use“, ermöglichte und somit den thematischen Grundstein für diese Dissertation legte.
Weiterhin möchte ich mich herzlich bei allen Mitarbeiterinnen und Mitarbeitern des Instituts für Roboterforschung bedanken – es war schön mit euch! Besonders hervorzuheben sind meine Kollegen Alexander Fölling, Christian Grimme, Matthias Hofmann, Markus Kemmerling und Alexander Papaspyrou. Für die sorgfältige Durchsicht und Korrektur des Manuskripts danke ich Karin Vehr.

Abstract

Thanks to successful national and international initiatives Grid Computing begins to leave its shadowy existence. Years ago it started with users and developers mainly coming from the field of high energy and particle physics, and now also attracts increasing interest in other disciplines like climatology and biomedical research.
Within the last five years with Cloud Computing another enabling technology emerged. This one utilizes resource and platform virtualization in its back-end and can be understood as complementary to Grid Computing.
A possible and by European Grid initiatives pursued combination of both technologies is the on-demand operation of Grid middleware services in a Compute Cloud. For this purpose the middleware services and the underlying operating systems are encapsulated in virtual appliances.
In this context, an important task is to establish comprehensible and replicable processes for the creation of these virtual appliances. This work evaluates the feasibility of workflows to describe these processes and shows workflows created for three different Grid middlewares. In the long run they allow a highly automated installation and configuration not only of Grid middleware services, but also supplemental services.

BPMN, Cloud Computing, Grid Computing, Virtualization, Workflows

Zusammenfassung

Dank nationaler und internationaler Initiativen tritt das Grid Computing langsam aus seinem Schattendasein hervor. Ausgehend von den ursprünglichen Hauptanwendern und auch -entwicklern aus der Hochenergie- und Teilchenphysik, weitet sich die Anwendung dieser Technologie auf andere Disziplinen, wie etwa die Klimaforschung und die Bio-Medizin, aus. Als zum Grid Computing komplementäre Technologie etablierte sich in den letzten fünf Jahren das Cloud Computing, welches im Hintergrund oft auf die Virtualisierung von Ressourcen und Plattformen zurückgreift.

Eine mögliche und von europäischen Grid-Initiativen forcierte Kombination beider Technologien liegt in dem bedarfsorientierten Betrieb von Grid Middleware-Diensten in einer Cloud. Dabei sind die Dienste mitsamt dem unterliegenden Betriebssystem in eine virtuelle Appliance zu kapseln.

Bei der Nutzung solcher Appliances stellen Abläufe für die kontrollierte, nachvollziehbare und wiederholbare Erzeugung einen wichtigen Punkt dar. Mit der Beschreibung dieser Abläufe als Workflows ist in dieser Arbeit eine erfolgversprechende Möglichkeit untersucht, die langfristig eine hochgradig automatisierte Installation und Konfiguration nicht nur von Grid Middleware-Diensten erlaubt.

BPMN, Cloud Computing, Grid Computing, Virtualisierung, Workflows

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Die Betreiber von Rechenzentren sind unabhängig davon, ob es sich um akademische oder kommerzielle handelt, an einem effizienten Betrieb und einer möglichst hohen Auslastung der vorhandenen Rechen- und Speicherkapazitäten interessiert. Insbesondere für die kommerziellen Betreiber scheint die Maximierung der Betriebseffizienz, etwa durch Optimierungen in den Bereichen der Kühlung, des Platzbedarfs und des Stromverbrauchs, lohnenswert. Die dort erzielten Einsparungen senken die Fixkosten des Betreibers und steigern so unter der Annahme konstanter Einnahmen den Gewinn.

Blickt man wenige Jahrzehnte zurück in die Vergangenheit, so wird eine weitere Ursache für die Bestrebung hin zu einer effizienten Nutzung deutlich: die damalige preisbedingte Knappheit an verfügbaren Rechen- und Speicherressourcen. Zur Ressourcenknappheit kam früher ein weiteres Problem hinzu: die verfügbaren Ressourcen waren nur für lokale Anwender nutzbar. Das heutzutage gängige Anmelden auf entfernten Ressourcen war vor dem Jahr 1983[1] kaum vorstellbar. Nach und nach rückte somit die Schaffung von möglichst transparenten Zugängen zu entfernten Ressourcen in den Mittelpunkt. Mittlerweile ist die Bereitstellung lokaler Ressourcen für entfernte Anwender ein fester Bestandteil im Angebot vieler Rechenzentren.

Weitere Bestrebungen zur Effizienzsteigerung, z. B. die Einführung des CPU-Schedulings, zielten auf die quasi zeitgleiche Nutzung von Hardware durch verschiedene Anwender ab und führten über Multi-User-Betriebssysteme letztendlich zur Möglichkeit der gemeinsamen Nutzung ganzer Ressourcenfarmen. Über die Zeit entwickelten sich verschiedene Formen des Zugriffs auf solche entfernt lokalisierten Farmen.

[1] In diesem Jahr wurde die hierzu verwendete Software Remote Shell (RSH) als Teil der Berkeley Software Distribution 4.2 veröffentlicht. Mittlerweile ist sie durch Secure Shell (SSH) abgelöst.

Eine frühe und heute noch gängige Vorgehensweise ist die Einrichtung lokaler Nutzerkonten. Ein entfernt lokalisierter Anwender baut hierbei eine optional verschlüsselte Verbindung über ein Wide Area Network (WAN) zur Ressource auf und kann anschließend so agieren, als befände er sich vor Ort. Dieses Vorgehen besitzt nicht nur Vorteile:

- Vor der Nutzung ist eine Kommunikation zwischen dem Anwender und dem Betreiber der Ressource notwendig. Die spontane Inanspruchnahme zuvor ungenutzter Ressourcen ist somit nicht möglich.

- Für jeden entfernten Anwender muss der Ressourcenbetreiber ein Nutzerkonto vorhalten, obwohl die Nutzung der Ressource oftmals dem Prinzip der zeitlichen Lokalität unterliegt. Hieraus folgt ein im Laufe der Zeit immer größer werdender Anteil an brachliegenden bzw. ungenutzten lokalen Nutzerkonten.

- Damit der Anwender sich mit der Ressource verbinden kann, muss der Betreiber Einstellungen in den um die Ressource herum liegenden Firewalls vornehmen, wobei jedes weitere „Loch“ die Gefahr eines möglichen Einbruchs vergrößert.

Mit dem Grid Computing entwickelte man ein Konzept für den sicheren Zugriff auf entfernte Ressourcen, welches zumindest die ersten beiden der zuvor erwähnten Schwächen vermeidet. Ein Teil dieses Konzeptes beschreibt hierzu etwa wiederverwendbare lokale Konten, die dem Prinzip der zeitlichen Lokalität gerecht werden[2].
Aus dem kommerziellen Umfeld trat im Laufe des Jahres 2007, gut 10 Jahre nach der Beschreibung von Computational Grids in [FK98], erstmals das Cloud Computing als weitere Möglichkeit zur Nutzung entfernter Ressourcen in Erscheinung. Eine der Kernkomponenten des Cloud Computing ist die Virtualisierung. Der Einzug dieser Technologie auf der Netzwerk-, Speicher- und Plattformebene ermöglichte weitere Effizienzsteigerungen bei dem Betrieb von Ressourcen.

Sowohl die Kunden als auch die Betreiber profitieren von der Virtualisierung durch eine größere Flexibilität. So können Kunden beispielsweise ihre Anwendungen inklusive Betriebssystem in so genannte virtuelle Appliances kapseln, wodurch die ansonsten notwendigen Anpassungen an die Software-Umgebung des jeweiligen Ressourcenbetreibers entfallen. Auf der anderen Seite versetzt die Virtualisierung die Betreiber in die Lage, ihren Kunden maßgeschneiderte Lösungen in Form von virtuellen Appliances anzubieten. Beide Parteien haben daher ein Interesse an Methoden und Werkzeugen, die sich mit der dynamischen und flexiblen Erzeugung solcher Appliances auseinandersetzen.

[2] Beim ersten Zugriff des Nutzers auf die Ressource erfolgt eine feste Abbildung zwischen Nutzer und lokalem Konto. Greift der Nutzer innerhalb eines definierten Zeitraumes nicht erneut auf die Ressource zu, wird das zugehörige lokale Konto bereinigt und die feste Zuordnung aufgehoben.

1.2 Ziel

Wie später genauer erläutert wird, erfolgt derzeit die Erzeugung virtueller Appliances auf unterschiedlichste Arten und Weisen und unter Zuhilfenahme verschiedenster Werkzeuge. Diese Werkzeuge realisieren vereinfacht betrachtet denselben Prozess, der u. a. die Bereitstellung einer virtuellen Maschine und die Installation und Konfiguration eines Betriebssystems sowie das Einspielen von Anwender-Software umfasst. Miteinander verglichen variieren die Werkzeuge allerdings sehr stark hinsichtlich ihrer Implementierung, etwa in der Granularität der Konfigurationsmöglichkeiten, so dass die Überlegung hin zu einer allgemeinen Beschreibung des Prozesses und von dort ausgehend zur Realisierung spezifischer Workflows geht.

Die Beschreibung von Prozessen und die semi- bzw. vollautomatisierte Ausführung einzelner oder aller Prozessschritte sind in Unternehmen ein gängiges Mittel, um sowohl die verfügbaren menschlichen als auch IT-Ressourcen bestmöglich einzusetzen und fügen sich daher nahtlos in das Konzept einer effizienten Ressourcennutzung ein.

Das Ziel dieser Arbeit liegt daher in der Beschreibung von Prozessen und der Erstellung daraus ableitbarer Workflows für eine weitestgehend vollautomatische Erstellung virtueller Appliances. Der Schwerpunkt soll hierbei auf Appliances für den Bereich des Grid Computing liegen. Nachfolgend sind die zu erreichenden Einzelziele erläutert.

- Innerhalb virtueller Appliances kommen üblicherweise verschiedenste Betriebssysteme zum Einsatz, wodurch die zu entwerfenden Workflows eine *Unterstützung multipler Betriebssysteme* vorsehen sollen. Diese Unterstützung wird zunächst auf Linux-Distributionen eingeschränkt, soll jedoch später auch auf Microsoft Windows-Betriebssysteme ausdehnbar sein. Dies stellt aufgrund der Fokussierung auf den Bereich des Grid Computing keine tatsächliche Einschränkung dar, da alle gängigen Grid Middlewares wie Globus Toolkit, gLite und UNICORE für ebensolche Linux-Distributionen verfügbar sind. Zwischen den einzelnen Linux-Distributionen gibt es allerdings Unterschiede, beispielsweise in Bezug auf die Standardwerkzeuge für die Software-Verwaltung, welche entsprechend zu berücksichtigen sind.

- Es sollen Workflows für *verschiedene, derzeit gängige Grid Middlewares* erstellt werden. Aktuell sind in Deutschland durch die D-Grid-Initiative die Middlewares gLite 3.2 (verwendet, z. B. in der Hochenergiephysik), Globus Toolkit 4 bzw. 5 (verwendet, z. B. in der Astroteilchenphysik) und UNICORE 6 (verwendet, z. B. in der Luftfahrtforschung) stark vertreten. Genau für diese drei Grid Middlewares soll der Entwurf und die Umsetzung von Workflows zur Installation und Konfiguration einzelner Middleware-Dienste erfolgen.

- Der verfolgte Ansatz darf sich nicht nur auf den Bereich der Grid Middlewares beschränken, sondern soll auch auf andere Software ausgeweitet werden können. Dies kann im Kontext eines Rechenzentrums beispielsweise die Dienste eines NFS-, eines DHCP- oder eines LDAP-Servers betreffen.

- Die Erstellung der Workflows für die einzelnen Grid Middleware-Dienste soll sofern möglich unter Verwendung von Standards erfolgen. Gerade in der Wirtschaft findet man verschiedene Modelle und Modellierungssprachen für die Workflows bzw. Prozesse, wobei drei von ihnen nach [FG08] weit verbreitet sind: Ereignisgesteuerte Prozessketten, Aktivitätsdiagramme aus dem Umfeld der Unified Modeling Language (UML) sowie die Business Process Modeling Notation.

Das Erreichen der Einzelziele und damit des Gesamtziels stellt keine triviale Aufgabe dar. Im Bereich der Betriebssysteme existiert zwar mit der ausschließlichen Betrachtung von Linux-Distributionen eine Einschränkung, doch können selbst diese Distributionen in Abhängigkeit ihres Ursprungs (z. B. Red Hat- im Vergleich zu Debian-basierten) in wesentlichen Punkten stark voneinander abweichen. Aus diesem Grund sind die jeweiligen Besonderheiten der in dieser Arbeit betrachteten Distributionen zu analysieren und zu berücksichtigen. Ein Fakt, der diese Aufgabe erleichtert, ist die in [Int06] beschriebene Linux Standard Base (LSB), an welche sich viele Distributionen anlehnen. Die LSB enthält Richtlinien, die neben dem Software-Umfang einer rudimentären Betriebssysteminstallation auch eine Verzeichnisstruktur gemäß des Filesystem Hierarchy Standard (FHS) (vgl. [Fil04]) beschreiben. In diesem Zusammenhang zwingt die forcierte Integration von Grid Middlewares in die offiziellen Software-Repositories der Distributionen auch den Middlewares die Einhaltung der FHS-Richtlinien auf.

Eine weitere Herausforderung stellt die Integration der mit den Grid Middlewares ausgelieferten Installations- und Konfigurationswerkzeuge in die zu erstellenden Workflows dar. Liegen für die Middlewares vorkonfigurierte Software-Pakete vor, so fallen durch den Einsatz distributionsspezifischer Paketverwaltungen wie YUM oder YaST zumindest bei der Installation Vereinfachungen an. Mit solchen Vereinfachungen kann bei den Konfigurationswerkzeugen nicht gerechnet werden. Allerdings lohnt sich die Betrachtung der Frage, ob ebendiese Werkzeuge zur Konfiguration der Middleware langfristig auf Workflows abgebildet werden können. Auf dem Weg zur Beantwortung dieser Frage ist eine Festlegung zu treffen, wie tief man in die Werkzeuge eingreifen will, um einzelne Teile – oder das gesamte Werkzeug – durch Workflows zu ersetzen. Nicht nur in diesem Fall, sondern bei allen zu erstellenden Workflows, ist eine möglichst geeignete Aufteilung der durchzuführenden Arbeiten in Einzelschritte zu finden. Die so geschaffene Modularität ermöglicht die Wiederverwendbarkeit einmal erstellter Workflows oder auch einzelner Aktivitäten.

1.3 Aufbau

Im Anschluss an die Einleitung sind in Kapitel 2 existierende Ansätze für die semi- bzw. vollautomatische Installation von Linux-Distributionen und Grid Middlewares vorgestellt. Es werden ebenso Ansätze für die Erzeugung virtueller Appliances beleuchtet. Das Kapitel 3 gibt einen Überblick über zwei Möglichkeiten für einen effizienteren Umgang mit verfügbaren Ressourcen. Beide kommen in dieser Arbeit zum Einsatz. Zum einen wird die Virtualisierung mit ihren Ausprägungen der Speicher-, Netzwerk- und Plattform-Virtualisierung vorgestellt und zum anderen ein Einblick in Prozesse, Workflows und deren Beschreibungssprachen gegeben.

Neben den Grid Middleware-Diensten werden auch andere Dienste betrachtet. Darunter fallen beispielsweise die Nagios und Ganglia Monitoring-Dienste sowie ein Stapelverarbeitungssystem. Die während der Betrachtung entstandenen Workflows für die Installation und Konfiguration sind in Kapitel 4 aufgeführt.

Verschiedene Szenarien, in denen eine Workflow-gestützte Bereitstellung virtueller Appliances sinnvoll erscheint, sind in Kapitel 5 dargestellt. Aufgrund von Workflow-Eigenschaften, wie etwa der Wiederholbarkeit und der Reproduzierbarkeit, ist diese Form der Bereitstellung insbesondere bei der Erzeugung wissenschaftlicher Daten interessant. Bei Szenarien im Bereich der Grid Middlewares liegt der Fokus – wie in Abschnitt 1.2 bereits angedeutet – auf gLite 3, Globus Toolkit in den Versionen 4 und 5 sowie auf der UNICORE Middleware 5 bzw. 6. Diese bestehen jeweils aus einer Menge an Diensten, die teilweise Grid-zentral vorliegen oder lokal auf einer Grid-Ressource ausgeführt werden. Aufgrund der Vielzahl der Dienste erfolgt eine Beschränkung auf Workflows für das Aufsetzen und Konfigurieren lokaler Dienste.

Des Weiteren ist in Kapitel 5 mit dem D-Grid eine deutsche Grid-Initiative beschrieben, die diese drei Middlewares einsetzt und teilweise nochmals eine spezifische Konfiguration einzelner Dienste erfordert. Innerhalb des D-Grid kann die Workflow-Unterstützung etwa bei der so genannten D-Grid Referenzinstallation Einsatz finden. Das Konzept und die Komponenten der Referenzinstallation sind in Abschnitt 5.4 vorgestellt.

Die Arbeit schließt mit einer Evaluation der entstandenen Workflows in Kapitel 6. Hierbei werden Vorteile der Workflow-Nutzung, wie etwa eine gestiegene Wiederverwendbarkeit und die leichte Austauschbarkeit einzelner Komponenten, demonstriert. Ebenso erfolgt eine Betrachtungen möglicher Erweiterungen und Optimierungen. Eine Zusammenfassung der erzielten Ergebnisse sowie ein Ausblick auf weitere Arbeiten ist in Kapitel 7 wiedergegeben.

1.4 Grundlagen

Neben den während der mehrjährigen Administration von Grid-Ressourcen gesammelten Erfahrungen, die insbesondere in das Kapitel 4 sowie die Abschnitte 3.1 und 5.3 flossen, eröffnete die Tätigkeit in verschiedenen Grid- und Cloud-bezogenen Projekten Möglichkeiten zur themenspezifischen Forschung. Aufgrund dieser Forschung entstanden mehrere wissenschaftliche Veröffentlichungen, die als Grundlage dieser Arbeit zu betrachten sind. So erfolgten hinsichtlich der in Abschnitt 5.4 beschriebenen D-Grid-Initiative mit [Fre10b] und [FW11] zwei Veröffentlichungen, die die Initiative und deren Perspektiven näher beleuchten. Einen weiteren, für diese Arbeit sehr wichtigen, Aspekt stellt die technische Realisation des D-Grid dar. Hierzu zählen die im D-Grid eingesetzten Grid Middlewares genauso wie die zentralen Dienste, etwa für die Ressourcenregistrierung und das Accounting. In diesem Bereich wurde aktiv an der Erstellung des D-Grid Betriebskonzepts (vgl. [BDE+07]) mitgewirkt.

Mit dem Aufkommen des Trends zur Ressourcenvirtualisierung erschien es vielversprechend, diese Technologie im Kontext von Grid-Rechenzentren zu betrachten. Dabei sind u. a. die Speicher-, Netzwerk- und Plattform-Virtualisierung sowie die Erstellung von virtuellen Appliances in den Veröffentlichungen [Fre09] und [Fre10a] detailliert beschrieben. Eine Zusammenfassung der Ergebnisse dieser beiden Veröffentlichungen ist in Abschnitt 3.1 eingeflossen.

Verglichen mit der Virtualisierung fand die Cloud einen noch rasanteren Einzug in das Fach-Jargon der IT-Welt. Viele der derzeit angebotenen Compute Clouds nutzen im Hintergrund die Virtualisierung, um die vorhandenen physischen Ressourcen möglichst effizient und granular Kunden über definierte Schnittstellen wie die EC2-API zur Verfügung zu stellen. Wie man eine Compute Cloud-Ressource in ein Grid einfügt und welche Unterschiede zwischen Grids und Clouds existieren ist in den Veröffentlichungen [FR10], [Fre11] und [FFEA12] dargelegt. In der hier präsentierten Arbeit finden sich die zugehörigen Ideen im Abschnitt 5.2 wieder.

Kapitel 2

Existierende Ansätze und Lösungen

Für die automatische Installation und Konfiguration von Betriebssystemen, von Anwendungssoftware und auch virtueller Appliances existieren bereits kommerzielle und Open Source-Lösungen. Dieses Kapitel zeigt im Abschnitt 2.1 zwei dieser Lösungen für Betriebssysteme und stellt im Anschluss daran die bisherigen Verfahren bei den betrachteten Grid Middlewares vor. Schließlich geht Abschnitt 2.3 auf die Möglichkeiten zur Erzeugung virtueller Appliances ein und stellt mit KIWI ein Werkzeug, welches bei openSUSE Einsatz findet, genauer vor.

2.1 Installation & Konfiguration von Betriebssystemen

Im privaten als auch im gewerblichen Bereich wünscht man sich eine Nutzung von Software ohne weitere Gedanken oder auch Zeit an die zuvor notwendige Installation und Konfiguration zu verschwenden. Diese beiden Schritte sollten daher idealerweise im Vorfeld durch jemand anderen ausgeführt worden sein. Alternativ wäre eine geführte semi- oder noch besser eine vollautomatische Durchführung der Schritte durch einen Dienst wünschenswert.

Für Betriebssysteme entstanden im Laufe der Zeit verschiedene Verfahren, die diesem Wunsch Rechnung trugen. Zunächst entwarfen einzelne Systemadministratoren beispielsweise Skript-basierte Verfahren zur automatisierten Durchführung der Installations- und Konfigurationsvorgänge. Neben einer Vereinfachung führten diese Skripte ebenso zu einer Beschleunigung der Vorgänge, wodurch z. B. kompromittierte Systeme schneller wieder hergestellt werden konnten. Mit fortschreitender Zeit begannen viele Anbieter von Linux-Distributionen eigene Verfahren (vgl. Tabelle 2.1), wie etwa Fully Automated Installation (FAI) zu entwickeln, auf die Administratoren heutzutage entsprechend zurückgreifen können. Einige der Verfahren führen nach abgeschlossener Installation des Betriebssystems ebenso eine Dienstkonfiguration durch und richten beispielsweise lokale Nutzerkonten und

Linux-Distribution	Verfahren
Debian Linux	Fully Automated Installation
Red Hat Linux	Kickstart
Scientific Linux	Kickstart
SUSE Linux	AutoYaST

Tabelle 2.1: Linux-Distributionen und ihre Verfahren zur automatisierten Installation

die Firewall ein.

In Anbetracht der Vielzahl der existierenden Verfahren ergibt sich zwangsläufig die Frage nach herstellerübergreifenden Standardisierungsbestrebungen hin zu einer einheitlichen Installationsroutine oder zumindest einem gemeinsam verwendeten Beschreibungsformat für die Installations- und Konfigurationsinformationen. Manche Verfahren setzen bei letzterem auf Mengen von Schlüssel-Wert-Paaren in unstrukturierten Textdateien, andere hingegen auf eine durchweg strukturierte Darstellung der Informationen unter Verwendung der Extensible Mark-Up Language (XML) und der XML Schema Definition (XSD).

Die Abschnitte 2.1.2 und 2.1.3 stellen mit Kickstart und AutoYaST zwei Installationsroutinen gängiger Linux-Distributionen detailliert vor. Die Betrachtung des Kickstart-Verfahrens steht in direktem Zusammenhang mit dem Anwendungsszenario der D-Grid Referenzinstallation (vgl. Abschnitt 5.4), da dort auf die Kickstart-unterstützende Distribution Scientific Linux (SL) gesetzt wird. Zuvor sind in Abschnitt 2.1.1 allgemeine Anforderungen an Verfahren zur automatisierten Installation und Konfiguration eines Betriebssystems festgehalten.

2.1.1 Allgemeine Anforderungen

Abhängig von der eingesetzten Linux-Distribution existieren unterschiedliche Verfahren, die eine automatisierte Installation und Konfiguration des Betriebssystems unterstützen. Für die in Umsetzung und Umfang variierenden Verfahren beschrieben Cons et al. in [CCD+01] folgende allgemeinen Anforderungen:

- Ein solches Verfahren sollte keine oder nur eine geringfügige Interaktion mit dem Nutzer erfordern. Idealerweise stößt dieser die Installation nur an und alle weiteren Schritte laufen automatisch ab. Eine Umsetzung dieser Anforderung findet man in der Möglichkeit zur so genannten unbeaufsichtigten Installation (engl.: unattended installation) wieder. Der Nutzer führt hierbei der Installationsroutine die benötigten Informationen, z. B. per Konfigurationsdatei im Vorfeld, zu und wird nach dem Ende der Routine über deren Ausgang in Kenntnis gesetzt.

Diese Form der Installation entlastet beispielsweise Systemadministratoren, die ansonsten laut [Deu00] bedingt durch sich stetig wiederholende Tätigkeiten bei einer manuellen Installation Monotonieerscheinungen wie Leistungsabnahme, Verminderung der Belastungsfähigkeit und Zunahme von Herzschlagarrythmie aufweisen können.

- Das Verfahren sollte sich sowohl für einzelne Systeme, wie etwa heimische Personal Computer (PC), als auch für den Einsatz in großen Rechenzentren eignen. Gerade in Rechenzentren müssen oftmals ganze Cluster von Rechnern installiert werden, wobei eine zeitgleiche Installation mehrerer Rechner die Reduktion der total benötigten Durchführungszeit ermöglicht.

  Bei der Installation mehrerer Rechner wäre es zudem wünschenswert, eine Reihenfolge festlegen zu können. Dadurch ließe sich z. B. die durch eine Netzwerk-Installation entstehende Last innerhalb eines Rechencluster besser balancieren und Engpässe im lokalen Netzwerk könnten vermieden werden.

- Das Verfahren sollte einfach auf weitere Betriebssysteme ausgeweitet werden können. Dieser Punkt birgt ein enormes Potential, z. B. hinsichtlich eines hohen Wiederverwendbarkeitsgrades einmal erstellter Vorlagen. Ebenso sollte die Integration eigener, ergänzender Funktionalität möglich sein.

- Vom Anwender spezifizierte Eingaben sollten auf Validität prüfbar sein, um etwaige Fehler bei der Installation bzw. Konfiguration frühzeitig zu erkennen. Die Prüfung könnte etwa durch das Abgleichen der Eingaben mit regulären Ausdrücken oder unter Anwendung eines XML-Schemas erfolgen.

2.1.2 Fallbeispiel Scientific Linux

Das Betriebssystem Scientific Linux basiert auf einem Zusammenschluss von technischen und wissenschaftlichen Mitarbeitern mehrerer Hochenergiephysik-Einrichtungen, unter ihnen auch das Fermi National Accelerator Laboratory (Fermilab). Das Ziel der Kooperation war die Erstellung einer speziell auf die Anwendungsschwerpunkte der Einrichtungen zugeschnittene Linux-Distribution. Das High Energy Physics Linux (HEPL) bildete den ersten Prototypen dieser Bestrebungen und wurde laut [Daw10] später aufgrund eines Namenskonflikts in Scientific Linux umbenannt. Die Veröffentlichung der ersten offiziellen Scientific Linux-Version fand im Mai des Jahres 2004 statt, zuletzt erschien im Februar 2012 die Version 6.2.

Als Ausgangsbasis für das damalige High Energy Physics Linux diente ein Red Hat Enterprise Linux 3, wodurch implizit das dort bei automatisierten Installationen zur Geltung

kommende Kickstart-Verfahren auch in Scientific Linux übernommen wurde.

Das Kickstart-Verfahren Bei diesem Verfahren sind alle während einer Installation erforderlichen Informationen in einer Datei, bestehend aus einer Menge von Schlüssel-Wert-Paaren, spezifizierbar. Beim Aufruf der Routine zur Installation des Betriebssystems wird hierzu die Option `ks` mit dem Namen der Kickstart-Datei als Argument verwendet. Das Beispiel in Listing 2.1 verwendet für die Installation die Datei `anaconda-ks.cfg`, welche auf einem Webserver mit der IP-Adresse 192.168.1.1 hinterlegt ist.

```
1 boot: linux ks=http://192.168.1.1/kickstart/anaconda-ks.cfg
```

Listing 2.1: Automatisierte Installation mittels einer präparierten Kickstart-Datei

Ein Ausschnitt aus einer solchen Kickstart-Datei ist in Listing 2.2 gezeigt, wobei sich die Gliederung in die drei Abschnitte `command`, `packages` und `post` erkennen lässt.

```
1  lang en_US
   langsupport --default=en_US
3  keyboard de-latin1-nodeadkeys
   mouse generic3ps/2
5  timezone --utc Europe/Berlin
   reboot
7  text
   install
9  bootloader --location=mbr --append vga=791
   zerombr
11 clearpart --all --drives=xvda,xvdb
   part swap --size 1 --grow --ondisk xvdb --fstype ext3
13 part / --fstype ext3 --size 100 --grow --ondisk xvda
   auth --useshadow --enablemd5
15 network --bootproto=dhcp --device=eth0
   firewall --disabled
17
   %packages --resolvedeps
19 @X Window System
   openssl-devel
21 syslinux

23 %post --nochroot
   mkdir -p /mnt/data
```

Listing 2.2: Auszug aus einer Kickstart-Datei

Die Bedeutung der einzelnen in Listing 2.2 aufgeführten Einträge ist nachfolgend erläutert.

- Der Abschnitt `command` (Zeile 1 bis 16) enthält u. a. Informationen zu den Installationsquellen, der Festplattenpartitionierung, der Firewall- und der Netzwerkkonfiguration.

- Im Abschnitt `packages` (Zeile 18 bis 21) enthält jede der Zeilen den Namen eines zu installierenden Pakets bzw. einer zu installierenden Paketgruppe. Letztere sind am vorangestellten `@` zu erkennen. Ein dem Paketnamen vorangestelltes Minuszeichen wählt das Paket für die Installation ab.
  Etwaige Paketabhängigkeiten werden durch das Hinzufügen von `resolvedeps` an das Schlüsselwort `%packages` soweit möglich automatisch aufgelöst.

- Der Abschnitt `post` (Zeile 23 und 24) beginnt mit dem Schlüsselwort `%post` und wird nach der Installation aller Pakete automatisch ausgeführt. Hauptsächlich finden sich hier Skriptaufrufe zur Konfiguration der zuvor installierten Pakete wieder.

Neben den hier vorgestellten Abschnitten einer Kickstart-Datei existieren noch weitere, wie etwa `pre`, der ein zu `post` analoges Verhalten aufweist. Die Inhalte des `pre`-Abschnitts werden also vor der Paketinstallation abgearbeitet.

2.1.3 Fallbeispiel SUSE Linux Enterprise Server

Die im Jahr 1992 gegründete Software und Systementwicklung GmbH (SUSE) veröffentlichte 1996 ihre erste Linux-Distribution unter dem Namen SUSE Linux. Ein wesentlicher Bestandteil dieser Distribution war das Werkzeug Yet another Setup Tool (YaST), über welches die Installation des Betriebssystems sowie die weitere Konfigurationen von z. B. Hardware-Komponenten und Netzwerkeinstellungen ablief. Seit dem ersten Release hat sich YaST kontinuierlich weiterentwickelt und wurde mit AutoYaST um ein Verfahren zur Automatisierung der Installation ergänzt.

Das AutoYaST -Verfahren Verglichen mit dem von Red Hat entwickelten Kickstart-Verfahren setzt AutoYaST auf eine strukturierte Darstellung und Speicherung der Installations- und Konfigurationsinformationen im XML-Format. Die Gültigkeit der im XML-Dokument gespeicherten Informationen wird mittels der XML-Schemasprache RelaxNG überprüft. RelaxNG wurde im Jahr 2001 von der Organization for the Advancement of Structured Information Standards (OASIS) verabschiedet und fokussiert sich analog zur XML Schema Definition auf die Beschreibung der XML-Dokumentstruktur. Im Jahr 2003 wurde für die Sprache mit RelaxNG Compact eine kompaktere Syntax eingeführt, die eine Ähnlichkeit zu regulären Ausdrücken aufweist. Dies führt verglichen mit den Document

Type Definitions (DTDs) und XML Schema Definitions zu einer insgesamt sehr kompakten Schemadefinition. Zudem bleibt bei der Verwendung von RelaxNG Compact durch die Möglichkeit der Transformation in eine XML Schema Definition eine weitgehende Kompatibilität erhalten.

Das Vorgehen bei AutoYaST zur Einbringung der notwendigen Informationen in die Installationsroutine ähnelt dem des Kickstart-Verfahrens und ist in Listing 2.3 gezeigt. Das Argument `autoyast` enthält hierbei einen Verweis auf die zu verwendende XML-Datei, die lokal oder auch entfernt auf NFS- oder Web-Servern liegen kann.

```
boot: linux autoyast=http://192.168.1.1/autoyast/client01_autoyast.xml
```

Listing 2.3: Automatische Installation mittels einer präparierten AutoYaST-Datei

Ein Ausschnitt aus einer von AutoYaST verwendeten und für SUSE Linux 9 geeigneten XML-Datei ist in Listing 2.4 dargestellt. Gut erkennbar ist die induzierte, hierarchische Untergliederung der Informationen.

```
<?xml version="1.0"?>
<profile>
  ...
  <bootloader>
    <activate config:type="boolean">false</activate>
    <loader_device>/dev/hda</loader_device>
    <loader_type>grub</loader_type>
    <location>mbr</location>
    ...
  </bootloader>
  <software>
    <packages config:type="list">
      <package>kernel-source</package>
      ...
    </packages>
    <patterns config:type="list">
      <pattern>base</pattern>
    </patterns>
  </software>
  ...
</profile>
```

Listing 2.4: Auszug aus einer AutoYaST-Kontrolldatei

In dem Listing entspricht das XML-Tag `profile` der Wurzel des Dokuments und enthält ein oder mehrere `resource`-Elemente. Zwei weitere hier nicht dargestellte XML-Tags sind `installation` und `configuration`. Das Tag `installation` kapselt u. a. Einstellungen für den Bootloader, die Festplattenpartitionierung und eine Liste der zu

installierenden Softwarepakete bzw. -gruppen. Das Tag `configuration` umschließt u. a. Informationen zur Netzwerkkonfiguration und zu den lokalen Nutzerkonten. Ebenso ist in diesem Abschnitt die Ausführung Nutzer-definierter Skripte nach Ende des Installationsprozesses beschreibbar. Das Listing 2.4 beschreibt somit die Installation des Bootloaders `grub` in den Master Boot Record (MBR) der Partition `dev/hda` sowie die Installation des Paketes `kernel-source` und aller zur Gruppe `base` gehörenden Pakete.

2.1.4 Bewertung der Verfahren

Mit dem Kickstart- und dem AutoYaST -Verfahren wurden zwei aktuelle Verfahren für die automatisierte Installation eines Betriebssystems vorgestellt. Neben diesen existieren, wie zu Beginn erwähnt, weitere Verfahren, wie etwa FAI oder auch solche für Microsoft Windows-basierte Betriebssysteme.

Ein wesentlicher Unterschied zwischen den vorgestellten Verfahren besteht in der Menge und der Granularität der Informationen, die den Installationsroutinen bei deren Ausführung mitgegeben werden können. Während das Kickstart-Verfahren nur recht rudimentäre Einstellungen erlaubt, bietet AutoYaST die Möglichkeit feingranularer Einstellungen an. So fehlt die in Listing 2.5 gezeigte Möglichkeit, im Vorfeld der Installation lokale Nutzer zu spezifizieren, bei dem Kickstart-Verfahren völlig.

```
<users config:type="list">
  <user>
    <uid>12</uid>
    <gid>100</gid>
    <home>/root</home>
    <username>root</username>
    <user_password>password</user_password>
    <shell>/bin/bash</shell>
    <encrypted config:type="boolean">true</encrypted>
    ...
  </user>
</users>
```

Listing 2.5: User-Abschnitt einer AutoYaST-Kontrolldatei

Der unterschiedliche Grad der Granularität spiegelt sich unmittelbar in der Größe der Kickstart- bzw. AutoYaST -Dateien wider. In manchen Fällen sind AutoYaST -Dateien etwa um den Faktor 10 größer[1].

Ein Vorteil von AutoYaST gegenüber dem Kickstart-Verfahren ist die Validierung aller Angaben in der vom Nutzer spezifizierten Datei auf Einhaltung des erlaubten Wer-

[1] Der genannte Faktor basiert auf den gemittelten Dateigrößen für Kickstart- und AutoYaST -Dateien, wie sie am D-Grid Ressourcen Zentrum Ruhr im Einsatz waren.

tebereichs und Datentypübereinstimmung mittels RelaxNG. Ungültige Angaben, für z. B. MAC-Adressen oder UID, werden so im Vorfeld der Installation erkannt und dem Nutzer gemeldet. Solch eine Validierung existiert beim Kickstart-Verfahren nicht, hier muss der Nutzer die Überprüfung manuell durchführen. Ein geübter Nutzer kann dafür allerdings auch eine Kickstart-Datei sehr einfach manuell erstellen, dies ist bei AutoYaST -Dateien wesentlich aufwendiger.

Betrachtet man die Installation von Appliances, die mehrere verteilte Komponenten umfassen (z. B. Multi-Tier-Architekturen mit ihrer Auftrennung in Nutzerschnittstelle, Geschäftslogik und Datenbank), bieten die beiden untersuchten Verfahren nur suboptimale Lösungen an: Für jede der Appliance-Komponenten ist eine separate Datei mit Installations- und Konfigurationsinformationen zu pflegen. Dies führt zu der Idee einer Erweiterung der im Kickstart- bzw. AutoYaST -Verfahren verwendeten Formate, so dass eine solche Datei Informationen zu mehr als einer Installation enthält. Eine gute Grundlage für diese Art der Erweiterung bietet der strukturierte und hierarchische Aufbau des AutoYaST -Formats.

Zusammenfassend bleibt festzuhalten, dass es zur automatisierten Installation und Konfiguration eines Betriebssystems distributionsspezifische Verfahren gibt. Diese Verfahren unterscheiden sich im verwendeten Dateiformat und der Granularität der Informationen, wodurch die Austauschbarkeit der Dateien zwischen bzw. die Wiederverwendbarkeit mit anderen Verfahren nahezu unmöglich ist. Des Weiteren waren keine Bestrebungen hin zu einem herstellerübergreifenden Verfahren erkennbar.

2.2 Installation & Konfiguration von Anwendungen

Die Einrichtung des Betriebssystems, egal ob automatisiert oder manuell, stellt einen wesentlichen Schritt auf dem Weg hin zu einer Software-Appliance dar. Hierfür sind neben dem Betriebssystem aber noch weitere Dienste oder auch Anwendungen notwendig, die ebenso installiert und konfiguriert werden müssen. Die Anzahl der Dienste variiert je nach Ziel der Appliance, wobei sich einfache Appliances auf die Bereitstellung genau eines Dienstes beschränken.

Für die Installation und die Konfiguration eines solchen Dienstes gibt es selbst bei einer Einschränkung der Betrachtung auf ausschließlich Linux-Distributionen kein einheitliches Schema. Eine Ursache hierfür sind unterschiedliche Vorgaben durch die einzelnen Betriebssysteme, beispielsweise für die Ablageorte von Konfigurationsdateien oder die Verfahren zum Starten und Stoppen von Diensten. Ein weiterer Grund ist die manchmal sehr unterschiedliche Ausgangsposition für die Software-Installation. Diese kann etwa bereitgestellte Binärpakete verwenden oder die vollständige Übersetzung des Quellcodes erfordern. Letz-

teres ist bei der Linux-Distribution Gentoo[2] der Fall.

Nachfolgend ist das Vorgehen bei der Installation und Konfiguration von Grid Middleware-Diensten anhand der drei Beispiele gLite, Globus Toolkit und UNICORE beschrieben. Die Installation und Konfiguration solcher Dienste stellen oftmals schwierige, zeitaufwendige und vor allem fehleranfällige Vorgänge dar. Für die in Abschnitt 5.3.2 vorgestellte Middleware gLite besteht die Schwierigkeit einerseits in der Vielzahl der auf einer Grid-Ressource zu installierenden Dienste und deren spärlicher bzw. veralteter Dokumentation. Andererseits verändern sich fortwährend, in Folge der Weiterentwicklung der Middleware, die Schnittstellen. Mit den in [BBB+05] beschriebenen LCG Compute Element (CE) und gLite Compute Element[3] sowie dem CREAM Compute Element (vgl. [ASZ+10]) gab es beispielsweise gleich drei unterschiedliche Middleware-Dienste für den Zugriff auf Rechenressourcen innerhalb der letzten fünf Jahre.

Ein weiterer wichtiger Faktor in diesem Zusammenhang ist die Verträglichkeit mit bereits installierter Software, wie etwa Bibliotheken oder anderen Grid Middlewares. Aus Sicht der jeweiligen Middleware-Entwickler mag die Vorstellung des Betriebs mehrerer Grid Middlewares auf derselben Ressource undenkbar sein, doch gerade dies wird im D-Grid praktiziert (vgl. Abschnitt 5.4).

2.2.1 gLite 3.2 Middleware

Die einzelnen Dienste der gLite Middleware sind einheitlich über die Paketverwaltungssoftware Yellowdog Updater Modified (YUM) (vgl. Abschnitt 4.2) installier- und über das Werkzeug YAIM ain't an Installation Manager (YAIM) (vgl. [LAPS+11]) konfigurierbar. YAIM selbst entsprang der Abteilung Specific Service Activities 3 (SA3) des Projekts Enabling Grids for E-science (EGEE) mit dem Ziel, den Deployment-Prozess der gLite Middleware-Dienste zu vereinheitlichen und zu vereinfachen. Das Werkzeug nutzt hierzu die Informationen in den vom Administrator bereitzustellenden Dateien `site-info.def`, `users.conf` und `groups.conf`. Die Datei `site-info.def` (vgl. Listing 2.6) enthält alle notwendigen Parameter zur Konfiguration der Middleware-Dienste in Form von Schlüssel-Wert-Paaren, z. B. den Namen der Grid-Ressource, eine Liste der unterstützten virtuellen Organisationen oder etwa die Eigenschaften des Batchsystems.

```
SITE_NAME="UNI-DORTMUND"
SITE_EMAIL="dgrid-admin@grid.tu-dortmund.de"
CE_CPU_MODEL="Xeon"
CE_CPU_VENDOR="Intel"
```

[2] http://www.gentoo.org/

[3] Das gLite Compute Element war als Nachfolger des LHC Computing Grid (LCG) Compute Element geplant, doch ging es nie in den Produktivstatus über und ist mittlerweile obsolet.

```
CE_CPU_SPEED="2000"
CE_OS_ARCH="x86_64"
CE_PHYSCPU="86"
CE_LOGCPU="129"
VOS="astrogrid atlas c3grid dech ... progrid textgrid wisent"
QUEUES="dgiseq dgipar dgiexp""
```

Listing 2.6: Auszug aus einer site-info.def-Datei

Die verschiedenen Gruppen von VO-Mitgliedern, die ein Middleware-Dienst später akzeptieren soll, sind in der Datei `groups.conf` in Form von Attributen anzugeben. Die Datei enthält zudem Informationen darüber, wie VO-Mitglieder auf die lokalen Nutzerkonten abgebildet werden sollen. Für die Existenz der bei der Abbildung verwendeten Nutzerkonten sorgt YAIM durch Auswertung des Inhalts der Datei `users.conf`. Hierin sind die anzulegenden Nutzerkonten inklusive UID und GID beschrieben.

Das YAIM-Werkzeug wurde in der früheren gLite Version 3.0 üblicherweise zweimal aufgerufen. Der erste Aufruf diente der Installation der notwendigen Softwarepakete und wurde in den aktuelleren Versionen durch einen Aufruf des Paketmanagers YUM ersetzt. In Listing 2.7 sind das alte und das aktuelle Vorgehen für die Installation gegenübergestellt. Als zu installierendes Paket wurde in dem Listing stellvertretend das Metapaket `glite-CREAM` gewählt.

```
# altes Vorgehen
yaim -i -s /path_to/site-info.def  -m glite-CREAM

# neues Vorgehen
yum -y install glite-CREAM
```

Listing 2.7: Installation von Metapaketen der gLite Middleware

Die Konfiguration der Middleware erfolgt mittels YAIM unter der Angabe des zu konfigurierenden Dienstes als Argument für die `-n`-Option und der Angabe des vollständigen Pfades zur Datei `site-info.def` (vgl. Listing 2.8). Die Datei `site-info.def` wiederum enthält Verweise auf die Speicherorte der Dateien `groups.conf` und `users.conf`.

```
yaim -c -s /path_to/site-info.def  -n Metapaket
```

Listing 2.8: Konfiguration installierter gLite Middleware-Metapakete

2.2.2 Globus Toolkit 4.0.8 Middleware

Die Globus Toolkit Middleware ist von den Webseiten der Globus Alliance[4] beziehbar. Neben vorkonfigurierten, distributionsspezifischen Paketen steht auch der Quellcode der

[4] `http://www.globus.org/toolkit/downloads/latest-stable/`

Middleware zum Download zur Verfügung.

Vor der Installation der Middleware aus einem vorkonfigurierten Paket heraus muss zunächst das Grid Packaging Tool (GPT) installiert werden. Dieses übernimmt die Extraktion des heruntergeladenen Globus Middleware-Pakets und alle weiteren notwendigen Schritte, wie beispielsweise die Konfiguration der einzelnen Job-Manager. Die Verwendung des Grid Packaging Tools mit dem Globus Toolkit 4.0.8-Paket ist in Listing 2.9 gezeigt.

```
1 # Extraktion des Pakets
  gpt-install gt4.0.8_binary-x86_64-unknown-linux-gnu-bin.tar.gz
3
  # Ausführen aller post-install-Skripte des Pakets
5 gpt-postinstall
```

Listing 2.9: Installation der Globus Toolkit Middleware

Für die Konfiguration der einzelnen Dienste, wie den Grid Resource Allocation Manager (GRAM) Server, den GSISSH Server, den Reliable File Transfer (RFT) Server und den Web Service (WS)-basierten GRAM Server, bringt die Globus Toolkit Middleware keine Werkzeuge mit, weswegen alle notwendigen Änderungen vom Administrator manuell durchzuführen sind.

Das beschriebene Vorgehen für die Installation der Middleware kann sich während der Laufzeit des Projekts Initiative for Globus in Europe (IGE)[5] ändern. Ein Teilziel des Projekts besteht in der Einbringung der Globus Toolkit Middleware in die Software-Repositories verschiedener Linux-Distributionen, beispielsweise Debian Linux und Ubuntu Linux. Dadurch wären die Dienste des Globus Toolkit analog zur gLite Middleware einfach über den Aufruf des Paketmanagers installierbar.

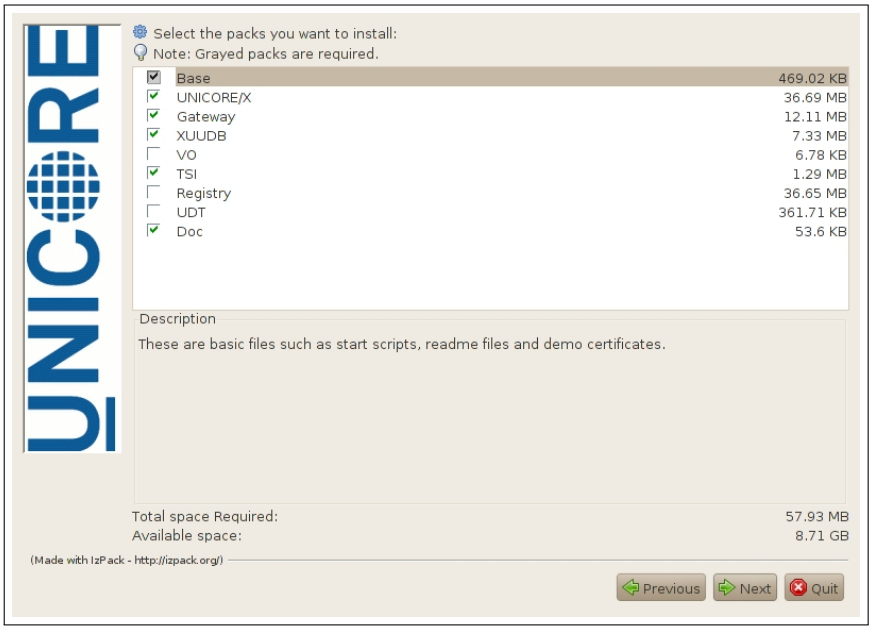
2.2.3 UNICORE 6.1 Middleware

Die UNICORE Middleware wird mitsamt graphischer Installationsroutine als `jar`-Archiv und als `tar.gz`-Archiv zum Herunterladen angeboten. Verwendet man das `jar`-Archiv, so erfolgt die Installation und Konfiguration anhand von Dialogen (vgl. Abbildung 2.1) weitestgehend geleitet. Im anderen Fall sind die Archivinhalte von Hand zu extrahieren und – ähnlich zur Globus Toolkit Middleware – die einzelnen Dienste händisch zu konfigurieren.

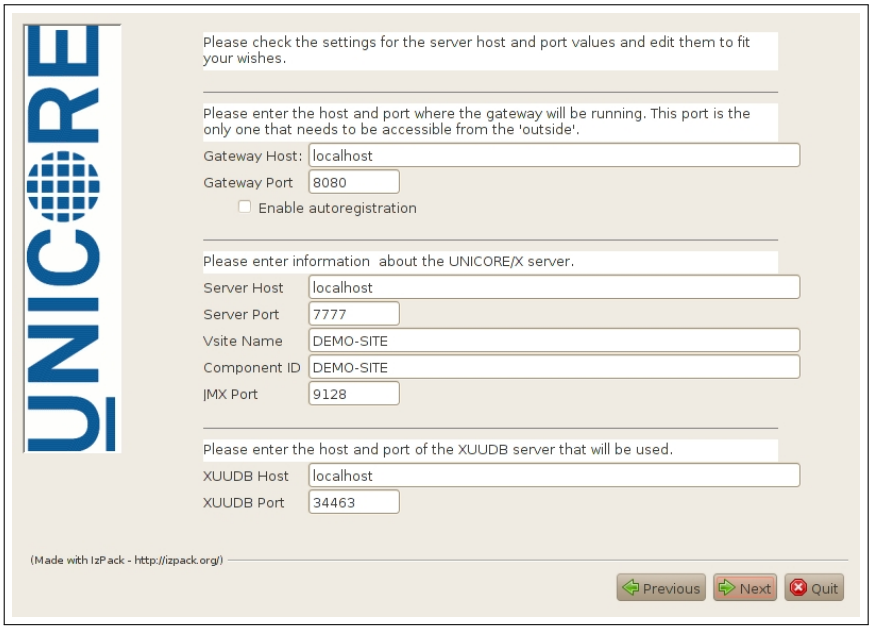
2.2.4 Evaluation der Verfahren

In den drei vorgehenden Abschnitten wurde das aktuelle Vorgehen für die Installation und Konfiguration von drei in Europa sehr weit verbreiteten Grid Middlewares beschrieben. Die

[5] `http://www.ige-project.eu/`



(a) Auswahl der zur Installation verfügbaren Dienste



(b) Konfiguration von Dienstparametern

Abbildung 2.1: GUI-basierte Installation von UNICORE 6.1 (Screenshots)

Vorgehensweisen für die Installation sind so verschieden wie die Projekte aus denen die Middlewares stammen.

Für die gLite Middleware existiert ein Software-Repository aus dem die Dienste über die Paketverwaltungssoftware YUM installierbar sind. Im Gegensatz hierzu verfügen das Globus Toolkit und UNICORE über kein solches Repository und setzen auf das Herunterladen von Software-Archiven von einer Webseite. Diese Archive entsprechen bei UNICORE einer Binär-Distribution und müssen nach dem Entpacken nur noch konfiguriert werden. Für das Globus Toolkit existiert neben den diversen Binär-Distributionen auch ein Quellcode-Archiv anhand dessen Systemadministratoren die Software eigenhändig übersetzen können.

Ähnlich uneinheitlich erfolgt die Konfiguration der Dienste bei den betrachteten Middlewares. Für die gLite Middleware existiert mit YAIM ein Werkzeug, welches die Konfigurationen basierend auf den in der Datei `site-info.def` bereitgestellten Informationen vollständig durchführt. Im Gegensatz hierzu erfolgt nach der Installation des Globus Toolkit lediglich eine Teilkonfiguration: die Einstellungen für beispielsweise den Reliable File Transfer-Dienst oder die Upstream-Links des MDS sind manuell durch den Systemadministrator nachzutragen. Bei der Installation der UNICORE Middleware aus einem `tar.gz`-Archiv heraus, erfolgt ebenso nur eine Teilkonfiguration, wobei die erforderlichen Parameter in der Datei `install.properties` spezifiziert werden können.

Die aktuellen Arbeiten innerhalb der European Middleware Initiative (EMI) führen zumindest zu einer Angleichung bei der Installation der Dienste der hier untersuchten Grid Middlewares. Diese sollen zukünftig über ein durch die European Middleware Initiative angebotenes Software-Repository installierbar sein. Offen bleibt die Frage nach einer einheitlichen Vorgehensweise für die Konfiguration.

2.3 Erzeugung virtueller Appliances

Erfolgt die Installation einer Anwendungssoftware in eine virtuelle Maschine, so kann man aus diesen beiden Komponenten mitsamt dem in der VM laufenden Betriebssystem eine virtuelle Appliance (vgl. Abschnitt 3.1.4) bündeln. Für die Erzeugung solcher virtueller Appliances existieren bereits verschiedene kommerzielle und auch Open Source-Lösungen. Nachfolgend sind stellvertretend mit SUSE Studio, rBuilder und der Image Creation Station (ICS) drei von ihnen vorgestellt.

SUSE Studio Das SUSE Studio[6] wird durch die Firma Novell bereitgestellt und soll laut Hersteller eine unkomplizierte und schnelle Erzeugung virtueller Appliances ermöglichen.

[6] `http://susestudio.com/`

Die Erstellung einer Appliance erfolgt über eine Web-Oberfläche und im Wesentlichen in vier Schritten:

1. Hinzufügen von Software-Paketen und -Repositories (vgl. Abbildung 2.2(a)),
2. Konfiguration lokaler Einstellungen wie Sprache und Uhrzeit (vgl. Abbildung 2.2(b)) und Festlegung der Größe des Haupt- und Festplattenspeichers,
3. Hinzufügen von Overlay-Dateien[7] und
4. Erzeugung der Appliance unter Angabe des Ausgabeformats (z. B. als VMware- oder Xen-spezifische Appliance).

Die Umsetzung des vierten Schritts erfolgt mittels KIWI, welches zur Vollständigkeit am Ende dieses Kapitels kurz vorgestellt wird. Ist die Erzeugung der Appliance abgeschlossen, so kann diese im SUSE Testdrive auf ihre Funktionalität hin getestet werden.

Aktuell werden von SUSE Studio nur SUSE-basierte Linux-Distributionen wie openSUSE, SUSE Linux Enterprise Server und SUSE Linux Enterprise Desktop (SLED) in den jeweiligen 32- und 64-Bit-Varianten unterstützt. Es ist allerdings bei den über SUSE Studio erzeugbaren Appliances nur die geführte Betriebssysteminstallation und -konfiguration vorgesehen, die Installation und Einrichtung von Anwendungssoftware wird nicht unterstützt.

rBuilder Die Software rBuilder[8] des Unternehmens rPath ist in den zwei kostenfreien Varianten rBuilder Online und rBuilder sowie der kostenpflichtigen Variante rBuilder unlimited erhältlich. Die freien Varianten unterscheiden bei der Installation, welche etwa bei rBuilder Online vollständig entfällt. Stattdessen greift man hier als Kunde auf ein Platform as a Service (PaaS)-Angebot (vgl. Abschnitt 5.2.1) von rPath zu. Alternativ und bei Sicherheitsbedenken bezüglich eines Zugriffs auf virtuelle Appliances, die außerhalb der eigenen Firewall liegen, kann rBuilder ebenso lokal installiert werden.

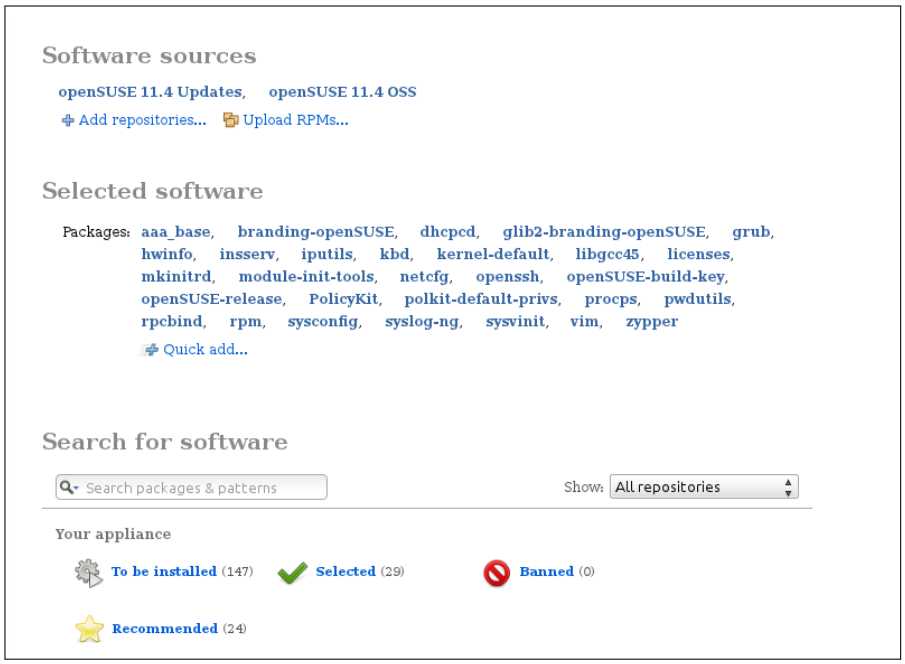
Alle drei rBuilder-Varianten unterstützen rPath Linux[9], CentOS und Ubuntu als Betriebssystem in den Appliances. rBuilder unlimited unterstützt weiterhin SUSE Linux. Das einzige Ausgabeformat von rBuilder Online ist das so genannte Amazon Machine Image (AMI)-Format für die Amazon Elastic Compute Cloud. Die beiden anderen rBuilder-Varianten ermöglichen zusätzlich die Erzeugung von Appliances für Compute Clouds wie die Globus Cloud[10] und verschiedene Hypervisoren, beispielsweise von VMware und Xen.

[7] Diese Dateien finden sich später an vordefinierten Stellen im Dateisystem der virtuellen Appliance wieder.

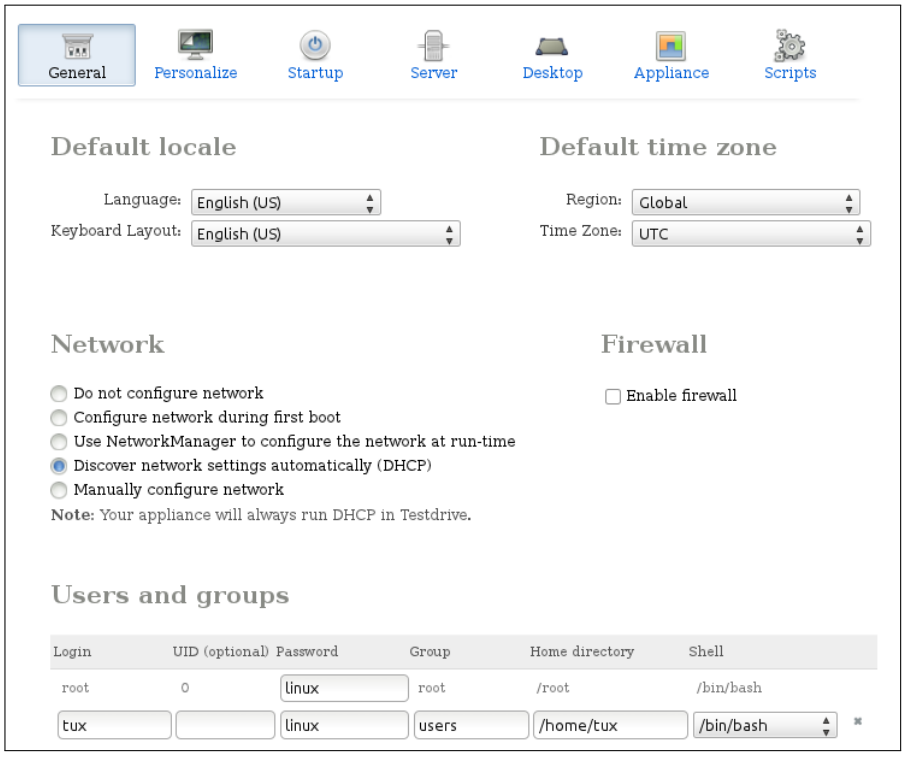
[8] `http://www.rpath.com/corp/products/rbuilder`

[9] rPath Linux ist eine Distribution, die von rPath selbst gepflegt und weiterentwickelt wird.

[10] `http://www.nimbusproject.org/`



(a) Auswahl der Software-Quellen



(b) Konfigurationsmöglichkeiten

Abbildung 2.2: SUSE Studio (Screenshots)

Eine Verwendung findet rBuilder im CernVM-Projekt. Buncic et al. beschreiben hierzu in [BASB+10] die Idee der Erstellung einer portablen, virtualisierten Analyseumgebung für die am Conseil Européen pour la Recherche Nucléaire (CERN) beherbergten LHC-Experimente.

Image Creation Station Mit der in [SSF+09] von Smith et al. vorgestellten Image Creation Station wurde an der Universität Marburg ein Werkzeug zum bedarfsorientierten Starten virtueller Appliances auf Xen-Basis entwickelt. Die Image Creation Station ist bezüglich ihrer Architektur in ein Frontend und ein Backend unterteilt, wobei letzteres die Verwaltung der virtuellen Appliances übernimmt.

Das Frontend besteht, wie in Abbildung 2.3 dargestellt, aus einem Grid-Dienst sowie einer Web-Oberfläche und dient der Interaktion mit dem Nutzer. Über die Web-Oberfläche erlangt der Nutzer einen Überblick über dessen hinterlegte, virtuelle Appliances und deren Status. Ebenso ist über die Oberfläche ein SSH-basierter Zugang zu den Appliances möglich. Die Erzeugung und Zerstörung der Appliances kann sowohl über die Web-Oberfläche als auch den Grid-Dienst erfolgen.

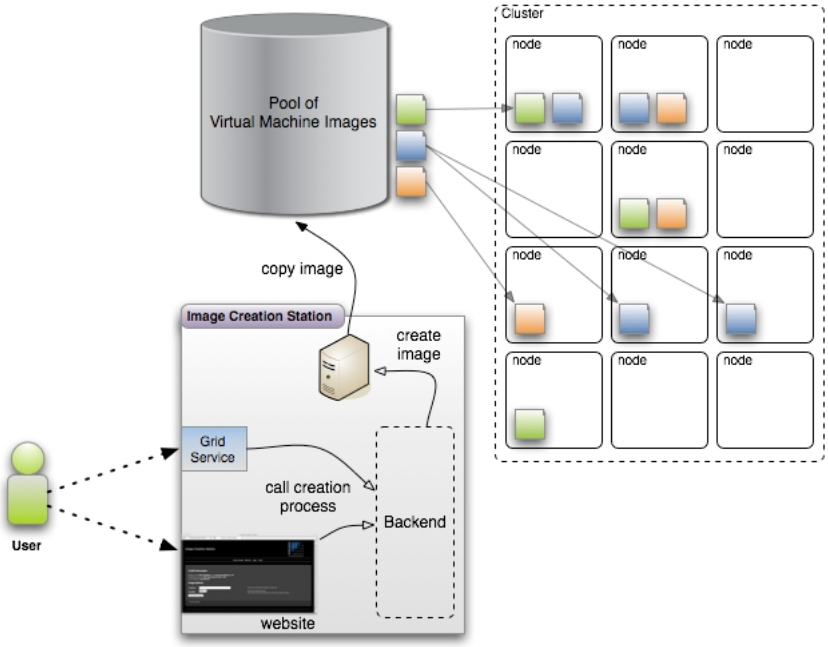


Abbildung 2.3: Architektur der Image Creation Station, Quelle: [Fal08]

Derzeit arbeitet die Image Creation Station im Bereich des Deployments virtueller Appliances nur mit der in [FPSF06] vorgestellten Xen Grid Engine (XGE) zusammen. Die Xen Grid Engine ist ein Virtual Machine Manager und in der Lage für einen durch einen Nutzer eingereichten Job eine zuvor für diesen Job als geeignet markierte, virtuelle Appliance zu

starten. Dieser Punkt enthält implizit die Einschränkung, dass die Xen Grid Engine nicht selbständig die passende Appliance ausfindig machen und starten kann: liegen für einen Nutzer, z. B. zwei oder mehr virtuelle Appliances im Virtual Machine Image Pool vor, verwendet die Xen Grid Engine die in der Image Creation Station als „default“ gewählte Appliance. Benötigt der Nutzer für einen Job eine andere virtuelle Appliance, so muss er erst die Standardauswahl in der Image Creation Station ändern.

Evaluation Die drei zuvor vorgestellten Lösungen erzeugen bzw. starten anhand von Nutzereingaben virtuelle Appliances, wobei dieser Begriff hier nur abgeschwächt gilt. Im Wesentlichen beschränken sich die Image Creation Station und das SUSE Studio auf die Bereitstellung einer virtuellen Maschine mit vorinstalliertem und konfiguriertem Betriebssystem.

Eine Installation von Anwendungssoftware ist bei SUSE Studio über das Einbinden externer Paketquellen möglich, jedoch bietet das Studio bei der Software-Konfiguration keine Unterstützung der vorgestellten Anwendungsszenarien. Es bleibt festzustellen, dass weder rBuilder, noch die Image Creation Station, noch das SUSE Studio die einfache Erzeugung virtueller Appliances für beliebige Dienste ermöglichen. Unklar bleibt zudem bei allen untersuchten Lösungen die interne Realisierung der Abläufe zur Erstellung einer virtuellen Appliance. Diese Abläufe könnten als Workflows realisiert sein oder z. B. aus einer einfachen Aneinanderreihung auszuführender Skripte bestehen.

Virtuelle Appliances für SUSE Linux-Distributionen

Mittels des von Schäfer und Schraitle in [SS11] beschriebenen KIWI Image Systems sind Appliances für die Linux-Distributionen openSUSE, SUSE Linux Enterprise Server und SUSE Linux Enterprise Desktop erzeugbar. KIWI zerlegt die Erzeugung einer Appliance in zwei voneinander entkoppelte Schritte (vgl. Abbildung 2.4). Im ersten Schritt werden in einer `chroot`-basierten Umgebung das Betriebssystem sowie die Anwendungssoftware installiert und konfiguriert. Das so gefüllte Wurzelverzeichnis ist in KIWI als *physical extend* bezeichnet. Während des zweiten Schritts transformiert KIWI dieses Wurzelverzeichnis in

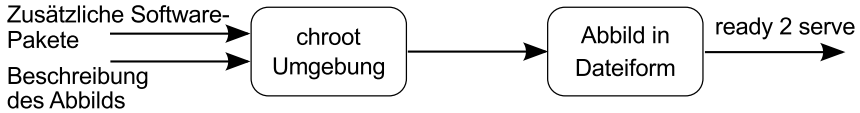


Abbildung 2.4: Zweistufiger Prozess der Abbilderzeugung mittels KIWI

einen vom Nutzer spezifizierten Typ von Festplattenabbild und erzeugt die Beschreibung der zugehörigen virtuellen Maschine. Das Festplattenabbild und die Beschreibung bilden

zusammen das Gegenstück zum *physical extend*, den *logical extend*. Unabhängig von dem gewählten Abbildtyp verwendet KIWI stets dasselbe Wurzelverzeichnis. Das Betriebssystem und die Anwendersoftware müssen somit nur einmal in der `chroot`-Umgebung konfiguriert werden. Eine so vorkonfigurierte Umgebung kann dann in alle von KIWI unterstützten Appliance-Formate gewandelt werden.

Das interne Design und die Veröffentlichung des Quellcodes von KIWI unter der GNU's Not Unix (GNU) General Public License (GPL) ermöglichen die einfache Ausweitung der Unterstützung auf andere, nicht-SUSE-basierte Linux-Distributionen. Es muss lediglich die vorgegebene Namenskonventionen für neu hinzuzufügende Funktionen beachten werden: diese Funktionen sollen mit dem Namen der entsprechende Linux-Distribution beginnen.

Kapitel 3

Eingesetzte Technologien

Auf der technischen Ebene sind zwei Themen von besonderer Relevanz für die Workflow-gestützte Bereitstellung von Grid Middleware-Diensten. Da die Dienste zeitgemäß in virtuellen Maschinen betrieben werden sollen, werden in Abschnitt 3.1 die verschiedenen Ausprägungen der Virtualisierung beschrieben. Der Fokus liegt dabei auf dem Bereich der Plattform-Virtualisierung. Zum anderen führt Abschnitt 3.2 in das Gebiet der Prozesse und Workflows ein. Letztere dienen u. a. zur Strukturierung und Automatisierung der bei der Installation und Konfiguration der Grid Middleware-Dienste anfallenden Tätigkeiten.

3.1 Virtualisierung

Das Konzept der Virtualisierung trat in den späten 1960er Jahren erstmals in Erscheinung. Zu dieser Zeit präsentierte Strachey in [Str59] einen Ansatz zum Time-Sharing auf Großrechnern. Dieser Ansatz erlaubt die Unterteilung von CPU-Zeit in Scheiben und deren Zuweisung an unterschiedliche Nutzer bzw. deren Programme. Ein Selektionsprozess, der den Vorläufer heutiger CPU-Scheduler bildet, bestimmt dabei die als nächstes auszuführende Zeitscheibe. Der Schritt hin zum Multitasking und virtuellen CPUs war nicht weit. Stracheys Ansatz fand u. a. bei ATLAS, dem damals leistungsfähigsten Rechner der Welt, Berücksichtigung. ATLAS wurde im Jahr 1962 an der University of Manchester in Betrieb genommen und unterstützte neben dem Time-Sharing auch die Konzepte des virtuellen Speichers und des Paging (vgl. [Lav78]). Die Idee zur Einführung von virtuellem Speicher entsprang hauptsächlich wirtschaftlichen Gründen. Der begrenzte physische Hauptspeicher wurde durch den wesentlich günstigeren, dafür langsameren Trommelspeicher ergänzt und in einen großen, von außen uniform ansprechbaren Speicher zusammengefasst.

Bis heute blieb der Kerngedanke dieser Form der Virtualisierung unverändert: eine Abstraktionsschicht oberhalb einer physischen Ressource – in den vorherigen Beispielen war

dies der Prozessor bzw. der Hauptspeicher – verbirgt deren Eigenschaften vor darauf zugreifenden Anwendungen. Stattdessen gewährt die Schicht die Sicht auf eine aggregierte oder dekomponierte Form dieser physischen Ressource. Bevor diese Sicht jedoch verfügbar ist, finden innerhalb der Abstraktionsschicht verschiedene Schritte statt:

1. Die Phase der Disaggregation schafft eine Trennung zweier bis dato fest verbundener Komponenten. Dabei kann es sich u. a. um die Auftrennung der festen Verbindung zwischen einem Programm und einer physischen CPU oder etwa die Auflösung der Bindung zwischen einem Dienst und der ausführenden physischen Plattform handeln.

2. Ein weiterer Schritt betrifft die ggf. auch räumliche Konsolidierung der verfügbaren physischen Ressourcen. Ein Beispiel hierfür ist das Verschieben von E/A-Ressourcen aus physischen Servern in separate Hardware-Module, wie es bei Blade-Systemen üblich ist. Die Konnektivität zwischen den ausgelagerten Ressourcen und den physischen Servern erfolgt dann über geeignete Bussysteme.

3. Es erfolgt die Bereitstellung einer Schnittstelle für den Zugriff auf die Ressource. Diese Schnittstelle verbindet die Virtualisierungsschicht mit oberhalb davon angesiedelten Anwendungen und verbirgt gleichzeitig die hinter der Disaggregation und Konsolidierung liegende Komplexität.

In Bezug auf die unterschiedlichen Virtualisierungsformen ergibt sich die in Abbildung 3.1 dargestellte Unterteilung. Im Wesentlichen findet die Virtualisierung Verwendung bei Ressourcen, Applikationen und Plattformen. Die Netzwerk-Virtualisierung und ihre Unterarten

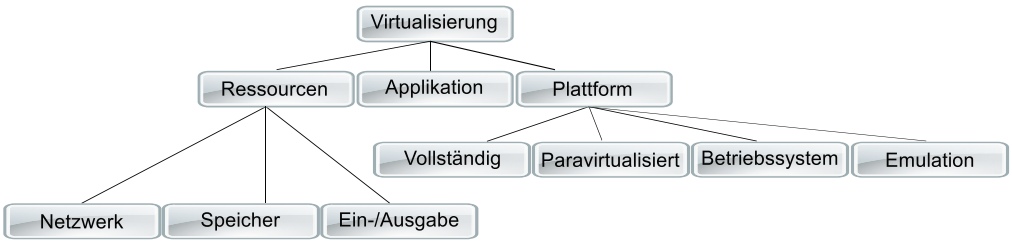


Abbildung 3.1: Übersicht über die Virtualisierungsarten

sind in Abschnitt 3.1.1 beschrieben. Im Anschluss daran gehen die Abschnitte 3.1.2 und 3.1.3 auf die Speicher- und E/A-Virtualisierung ein. Abschließend stellt Abschnitt 3.1.4 die Plattform-Virtualisierung vor.

3.1.1 Netzwerk-Virtualisierung

Die Disaggregation separiert in Netzwerken zunächst die logische von der darunterliegenden physischen Topologie. Ausgehend von der logischen Netzwerktopologie ermöglicht die

so genannte externe Virtualisierung die Zusammenfassung disjunkter Netzwerke zu einem großen, virtuellen Netzwerk und ebenso die Zerlegung eines Netzwerks in diverse voneinander isolierte, virtuelle Netzwerke.

Ein Beispiel für die Aggregation disjunkter Netze zu einem logischen ist das Virtual Private Network (VPN). In jeder an dem Virtual Private Network teilnehmenden Netzwerkpartition ist auf mindestens einem Rechner mit Außenkonnektivität ein Virtual Private Network Gateway installiert (vgl. Abbildung 3.2). Teilnehmer aus unterschiedlichen Partitionen kommunizieren über so genannte Tunnel, die zwischen den Gateways existieren. Erreicht ein ausgehendes Paket den Gateway einer Partition kapselt es dieser und sendet das gekapselte Paket an den Gateway der Zielpartition weiter. Dieser extrahiert das ursprüngliche Paket und leitet es an den ursprünglichen Empfänger weiter. Die disjunkten Netzwerkpartitionen sind somit im Virtual Private Network aus Nutzersicht zu einem großen, zusammenhängenden Netzwerk verschmolzen, wobei die einzelnen Partitionen, wie in Abbildung 3.2 gezeigt, durch separate Firewalls geschützt sein können.

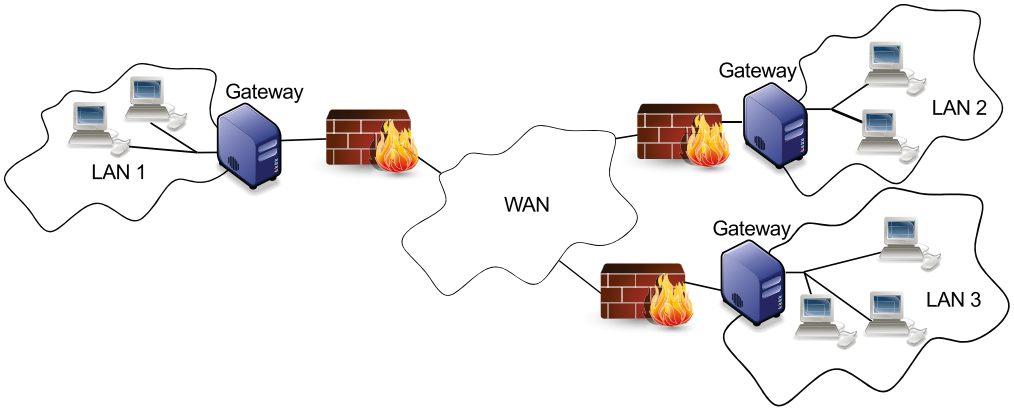


Abbildung 3.2: Beispiel für ein Virtual Private Network

Das Virtual Local Area Network (VLAN) stellt eine weitere praktische Anwendung der externen Virtualisierung dar. Beliebige Kombinationen aus Geräten bzw. Gerätegruppen in physisch disjunkten Netzwerken sind mittels dieser Technologie zu einer logischen Einheit, einer VLAN-Domäne, zusammenschaltbar. Innerhalb dieser Domäne kommunizieren die Teilnehmer, als würden sie sich im selben logischen Netzwerk befinden. Diese vollständige Loslösung von der Netzwerktopologie ist eines der herausragenden Merkmale von VLANs. Dabei stellen Mechanismen im VLAN-Protokoll die Separation verschiedener VLAN-Domänen auf demselben Übertragungsmedium sicher. Ohne die Mechanismen könnten alle am Medium angeschlossenen Teilnehmer, insbesondere solche anderer VLAN-Domänen, die übertragenen Daten abgreifen.

Im Gegensatz zur externen Virtualisierung stellt die interne Virtualisierung einzelnen

Software-Containern eine Netzwerk-ähnliche Funktionalität zur Verfügung. Die von Barham et al. in [BDF+03] beschriebene Virtualisierungssoftware Xen nutzt diese Form der Virtualisierung, um Gastsystemen (virtuellen Appliances) über das Wirtssystem (physischer Host) Netzwerkschnittstellen und damit gleichbedeutend einen Netzzugang anzubieten. Hierzu erzeugt Xen im Wirtssystem zusätzlich zu den physischen vorhandenen noch weitere, virtuelle Netzwerkkarten. Diese virtuellen Karten sind an virtuelle Schnittstellen (Virtual Interfaces (VIFs)) gekoppelt, deren Endpunkte in den Gastsystemen liegen. Abhängig von der Konfigurationen der virtuellen Schnittstellen ist ein Netzwerkzugang für Gastsysteme in den Modi Bridged, Routed und Network Address Translation (NAT) möglich. Der Bridged-Modus mit einer Außenanbindung über die Schnittstelle `peth0` ist in Abbildung 3.3 dargestellt. Die Schnittstelle `vif0.0` ist Bestandteil der Bridge `br0` und

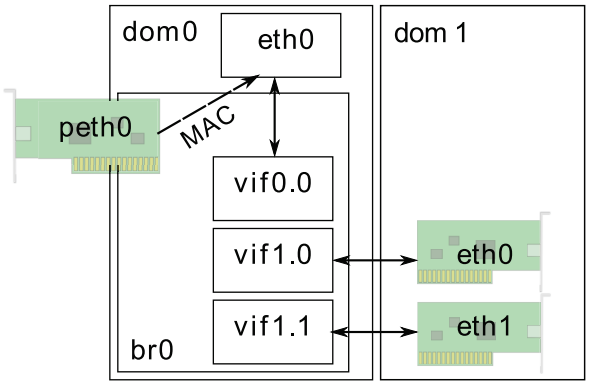


Abbildung 3.3: Beispiel des Bridged-Modus bei Xen, Quelle: [NGP+10]

dient der bidirektionalen Kommunikation mit dem Wirtssystem, welches bei Xen in der so genannten `dom0` läuft. Über die weiteren in der Bridge existierenden, virtuellen Schnittstellen `vifx.y` (`x`: Xen Domain Identificator, `y`: Schnittstellennummer innerhalb der Domain) läuft der Netzwerkverkehr von und zur entsprechenden Xen Domain `x`, in diesem Beispiel der Domain 1.

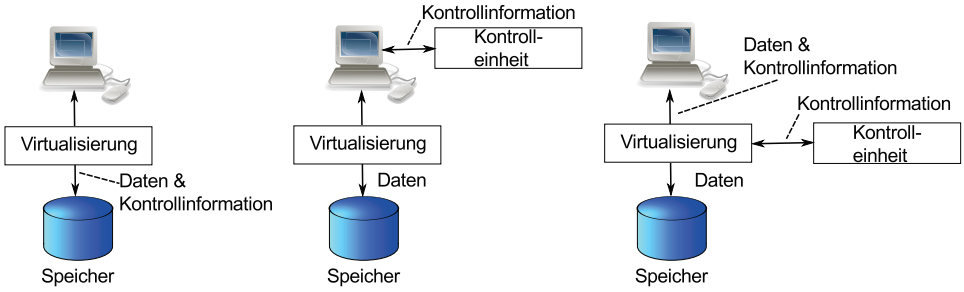
3.1.2 Speicher-Virtualisierung

Petabyte-Speichersysteme bestehen oft aus einer Kombination von Festplatten- und Bandspeichereinheiten. Die Speicherkapazitäten dieser einzelnen Einheiten werden innerhalb des Systems auf verschiedenen Ebenen neu gruppiert, um nach außen einen großen, zusammenhängenden Speicher anzubieten.

Auf der untersten Schicht des in Abbildung 3.4 gezeigten Schichtenmodells existieren Mengen von Festplatten als physischer Speicher. Diese sind zu Redundant Array of Independent Disks (RAID)-Gruppen zusammengeschlossen und typischerweise an eine Storage

Abbildung 3.4: Schichtenmodell bei der Speicher-Virtualisierung

Area Network (SAN) Fabric angebunden. Die Fabric liefert der darüber liegenden Schicht virtuelle Logical Unit Numbers (LUNs) aus, wobei die Methoden des Zoning und LUN Masking den Zugriff auf die zu logischen Gruppen zusammengefassten Logical Unit Numbers regeln. Über einzelne Host Bus Adapter (HBA) oder einen HBA RAID-Verbund sind die im Storage Area Network existierenden Logical Unit Numbers auf einzelnen File-Servern einbind- und mit einem Dateisystem formatierbar. Anschließend steht der Speicher zur Aufnahme von Datenobjekten bereit.



(a) in-band (b) out-of-band (c) split-path

Abbildung 3.5: Die drei Klassen von Speichernetzwerken

Innerhalb eines Storage Area Networks, so beschreibt Tate in [Tat03], tritt die Virtualisierung in den Untertypen in-band, out-of-band und split-path auf. Diese unterscheiden sich anhand der verwendeten Pfade für die Nutzdaten und die zugehörigen Kontrollinformationen[1]. Bei in-band-Netzwerken (vgl. Abbildung 3.5(a)) fließen die Nutzdaten sowie die Kontrollinformationen über den selben Pfad, was die Entstehung von Engpässen begünstigt. Im Vergleich zu den anderen vorgestellten Typen sind sie jedoch einfach und aufgrund nicht notwendiger weiterer Hardware kostengünstig zu realisieren. Die Auftrennung des

[1] Die Kontrollinformationen enthalten u. a. Informationen über den physischen Speicherort der Nutzdaten.

einen Pfades für Nutzdaten und Kontrollinformationen in separate Pfade macht aus einem in-band- ein out-of-band-Netzwerk (vgl. Abbildung 3.5(b)). Die Separation ermöglicht es, die gesamte Bandbreite auf dem Pfad zum Speicher für die Nutzdatenübertragung einzusetzen und zwei disjunkte Speicherorte für die Nutzdaten und die Kontrollinformationen zu verwenden. Out-of-band-Speichernetzwerke erfordern allerdings zusätzliche Hardware außerhalb des Nutzdatenpfads. Der Fokus verschiebt sich mittlerweile hin zu den so genannten split-path-Netzwerken (vgl. Abbildung 3.5(c)), einem Hybrid der zuvor beschriebenen Ansätze. Die Nutzdaten und Kontrollinformationen laufen bei diesem Untertyp bis zu einem gewissen Punkt über denselben Pfad. Dort werden die Kontrollinformationen aus dem Pfad entfernt und einer Kontrolleinheit zugeführt.

3.1.3 E/A-Virtualisierung

Verglichen mit dem Fortschritt in den Bereichen der Speicher- und Netzwerk-Virtualisierung steckt die E/A-Virtualisierung noch in ihren Kinderschuhen.

Typischerweise kommunizieren physische Server untereinander und mit der Außenwelt über verschiedene E/A-Schnittstellen für z. B. das Daten- und das Speichernetzwerk (vgl. Abbildung 3.6). Für eine E/A-Virtualisierung trennt man zunächst die E/A-Schnittstellen,

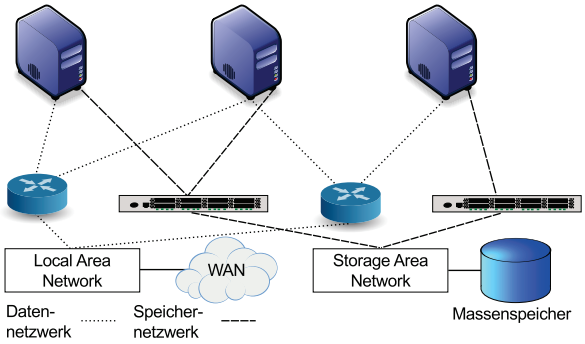


Abbildung 3.6: Traditionelle Netzwerkanbindung eines Servers, Quelle: [Sha08a]

wie Network Interface Cards (NICs) und Host Bus Adapter, von den physischen Servern und lagert sie aus. Am Auslagerungsort werden die einzelnen E/A-Ressourcen zu einem Pool konsolidiert und Teile hieraus wieder über die Virtualisierungsschicht den einzelnen physischen Servern zugewiesen. Der Zugriff der physischen Server auf die beispielsweise in einen E/A-virtualisierten Switch ausgelagerten Ressourcen (vgl. Abbildung 3.7) erfolgt über Bussysteme wie PCI Express (PCIe). Neben der hohen Bandbreite von bis zu 64 Gbps bei der PCIe Version 3.0 und geringen Latenzzeiten sprechen auch die geringen Kosten pro Port für diesen Bus.

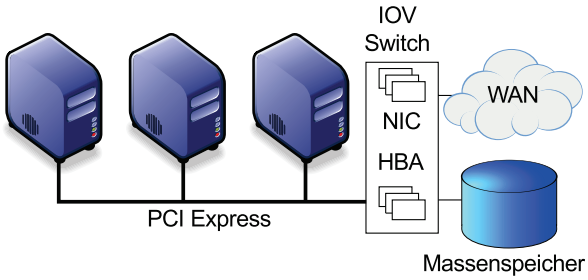


Abbildung 3.7: Server-Anbindung an einen E/A-virtualisierten Switch

3.1.4 Plattform-Virtualisierung

Bedingt durch die Disaggregation, verschwindet bei dieser Virtualisierungsform die feste Bindung zwischen einer Plattform[2] und dem darauf ausgeführten Betriebssystem. Die disaggregierten, physischen Ressourcen sind über die Virtualisierungsschicht so genannten virtuellen Maschinen, die man als Software-Container auffassen kann, zuweisbar. Durch die Verwendung von virtuellen Maschinen lassen sich beispielsweise mehrere Betriebssysteme – je eines pro Container – auf einer Plattform parallel betreiben. Analog zu Anwendungen, die um die vorhandenen Ressourcen konkurrieren, konkurrieren virtuelle Maschinen (engl.: Virtual Machines (VMs)) um den Zugriff auf die unter der Virtualisierungsschicht befindliche physische Hardware. Ein Teil der Virtualisierungsschicht ist daher für die Partitionierung dieser Hardware und das Scheduling verantwortlich.

Bereits 1974 formulierten Popek und Goldberg in [PG74] Anforderungen an virtuelle Maschinen:

- Die in einer virtuellen Maschine vorhandene Hardware-Umgebung soll sich für Programme möglichst „echt“ anfühlen.
- Die Ausführung eines Programms in einer virtuellen Maschine soll nur minimale Performance-Einbußen im Vergleich zur Ausführung auf der physischen Hardware haben.
- Die Gesamtkontrolle über alle Systemressourcen soll an zentraler Stelle innerhalb der Virtualisierungsschicht liegen.

Mittlerweile haben virtuelle Maschinen Einzug in die IT-Infrastruktur vieler Unternehmen gefunden, wobei die vor mehr als dreißig Jahren formulierten Anforderungen weiterhin Bestand haben. Nachfolgend sind einige der Vorteile virtueller Maschinen gegenüber ihren nicht-virtualisierten Gegenstücken skizziert.

[2] Unter dem Begriff der Plattform versteht man in diesem Zusammenhang im weitesten Sinn die Hardware eines Servers bzw. Personal Computers.

- *Dienstisolation*: Durch unterschiedlichste Anforderungen von Diensten an das darunterliegende Betriebssystem ist eine gemeinsame Installation mehrerer Dienste innerhalb derselben Betriebssysteminstanz meist nur aufwendig realisierbar. Teilweise wird die Realisierbarkeit durch den Einsatz, der von Furlani und Osel in [FO96] beschriebenen, Software Modules begünstigt. Jedoch führt das hinter Modules liegende Konzept langfristig zu einer großen Anzahl installierter Software, was einerseits einen erhöhten Wartungsaufwand und andererseits mehr Fläche für potentielle Angreifer bedeutet.

  Die Platzierung der einzelnen Dienste in voneinander isolierte, virtuelle Maschinen entzerrt die Situation. Hieraus folgt auch eine größere Stabilität der einzelnen Dienste, da z. B. durch andere Dienste hervorgerufene Seiteneffekte nicht länger auftreten. Des Weiteren ermöglicht der Betrieb in voneinander abgeschotteten Umgebungen die individuelle Festlegung von Sicherheitsstufen für jeden der Dienste.

- *Konsolidierung*: Heutige physische Server sind so leistungsfähig, dass sie mühelos mehr als einen Dienst bereitstellen können. Aus Gründen der Energieeffizienz und finanziellen Aspekten sind Dienstanbieter daher daran interessiert, möglichst viele Dienste unter Beachtung von Rahmenbedingungen (z. B. Einhaltung der Dienstgüte und Redundanz) auf möglichst wenige physische Systeme zu bündeln. Die Plattform-Virtualisierung mit der zuvor vorgestellten Dienstisolation stellt hierfür eine ideale Technologie dar.

- *Reduktion der Anschaffungs- und Betriebskosten*: Die Kosten einer virtualisierten Infrastruktur teilen sich in i) einmalige Kosten für die notwendige Hard- und Software sowie die damit verbundenen Dienste, ii) fortlaufende Kosten für den Betrieb und die Verwaltung (z. B. Beschaffungsprozesse) und iii) sporadisch auftretende Kosten auf.

  Das Ziel einer Dienstkonsolidierung ist oft die Reduktion der Anzahl der benötigten physischen Systeme. Diese müssen u. U. leistungsfähiger sein, jedoch wiegen andere Aspekte, wie etwa der reduzierte Stellplatz und damit auch die maximal mögliche Rechenleistung pro Quadratmeter, mehr. Des Weiteren ist durch die bessere Auslastung der einzelnen physischen Server ein Einsparpotenzial im Bereich der Kühlung und des Leistungsverbrauchs gegeben. Im Bereich der Speicher- und Datennetzwerke sind ebenso Kosteneinsparungen möglich, sofern mehrere virtuelle Maschinen dieselben Schnittstellen eines physischen Systems verwenden können.

  Für jeweils ein Beispiel aus den Branchen Gesundheitswesen, Logistik und Versicherungen ist das mögliche Einsparpotential in Tabelle 3.1 aufgezeigt. Hierbei ist zu berücksichtigen, dass die Daten aus einem Bericht des Unternehmens VMware stam-

men, welches Virtualisierungssoftware vertreibt.

	Gesundheitswesen		Logistik		Versicherungen	
Plattform-Virtualisierung	ohne	mit	ohne	mit	ohne	mit
Erforderl. physische Server	62	6	58	8	92	8
Ø CPU Auslastung (%)	5	80	<10	60-65	<10	60-70

Tabelle 3.1: Einsparpotentiale durch Plattform-Virtualisierung, Quelle: [VMw06]

- *Hochverfügbarkeit*: Für den Begriff der Hochverfügbarkeit existieren verschiedene Definitionen. Für den Bereich des Cluster Computing ist in [IEE04] eine Definition durch die Institute of Electrical and Electronics Engineers (IEEE) Task Force on Cluster Computing gegeben durch:

  > High Availability [...] refers to the availability of resources in a computer system, in the wake of component failures in the system.

  In Szenarien ohne Virtualisierung ist ein Dienst fest an das ausführende, physische System gebunden. Fällt dieses System planmäßig (z. B. aufgrund einer Wartung) oder außerplanmäßig aus, ist davon auch der Dienst betroffen - er steht für die Zeit des Systemausfalls nicht zur Verfügung. Durch die mit der Virtualisierung einhergehende Disaggregation von Dienst und System kann der Dienst bei Ausfall des Systems semi- oder vollautomatisch auf ein anderes verfügbares System verschoben werden. Unter bestimmten Voraussetzungen kann diese Migration auch „live“ erfolgen, wodurch die Verfügbarkeit des Dienstes erhalten bleibt.

Im Laufe der Zeit etablierten sich innerhalb der Plattform-Virtualisierung verschiedene Unterarten, wie z. B. die Emulation oder die Paravirtualisierung. Der folgende Abschnitt stellt die wesentlichen, heute im Einsatz befindlichen Arten vor.

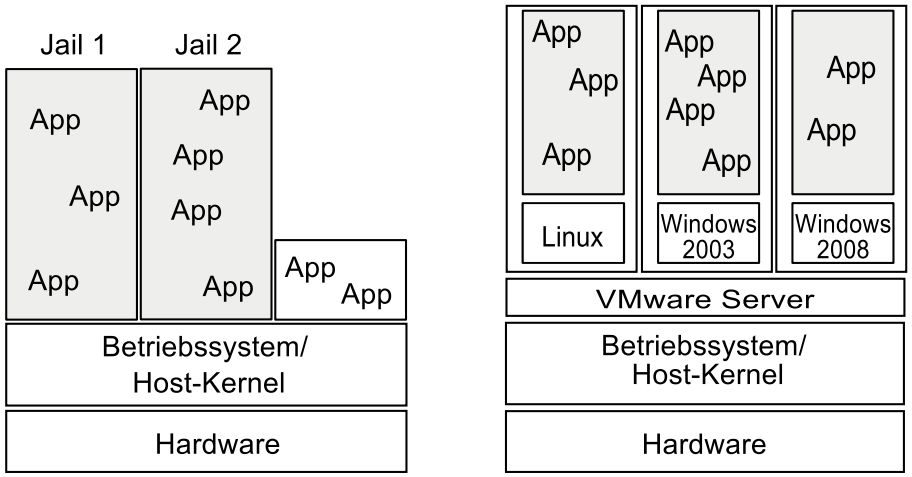
Ausprägungen der Plattform-Virtualisierung Analog zu der Netzwerk-Virtualisierung existieren auch bei der Plattform-Virtualisierung verschiedene Unterarten. Mit der Emulation, der Betriebssystem-Virtualisierung, der vollständigen Virtualisierung und der Paravirtualisierung sind nachfolgend vier von ihnen vorgestellt.

- Bei der *Emulation* werden die Funktionen eines Systems vollständig nachempfunden, wobei das emulierende System die emulierten Komponenten nicht besitzen muss. Neben Prozessoren können so Peripheriegeräte, wie Grafikkarten, serielle Schnittstellen und Universal Serial Bus (USB) Controller, wie auch vollständige Plattformen

emuliert werden. Ihren Einsatz findet die Emulation etwa im Rahmen der Softwareentwicklung für noch nicht verfügbare Prozessoren. So bietet beispielsweise der von Lawton in [Law96] vorgestellte Bochs-Emulator die Möglichkeit der Emulation von i386-, i486- und Pentium-Prozessoren mit den jeweiligen Erweiterungen, wie die Multimedia Extension (MMX) und die Streaming SIMD Extensions (SSE).

- Die *Virtualisierung auf Betriebssystemebene* setzt im Kernel des Host-Systems an. Dieser stellt für Anwendungen virtuelle, voneinander isolierte Laufzeitumgebungen, auch Container oder Jails genannt, bereit und ist ebenso für deren Verwaltung zuständig (vgl. Abbildung 3.8(a)). Die Container verfügen über keinen separaten Kernel, sondern nutzen den des Host-Systems. Eine direkte Konsequenz hieraus ist die fehlende Unterstützung für vom Host-System abweichende Betriebssysteme in den Jails. Die Bereitstellung eines modifizierten Kernels oder bestimmter Treiber an nur einem der Jails ist ebenso problematisch.

  Zwei Beispiele für Software, die eine Virtualisierung auf Betriebssystemebene ermöglicht, sind Virtuozzo und das in [Dik06] vorgestellte User Mode Linux. Letzteres kann nicht nur Anwendungen, sondern auch den Linux-Kernel selbst als unprivilegierten Prozess ausführen. Hierdurch vereinfacht sich u. a. das Debugging und Profiling neuer Kernel-Versionen erheblich.



(a) Betriebssystem-Virtualisierung (Jails)

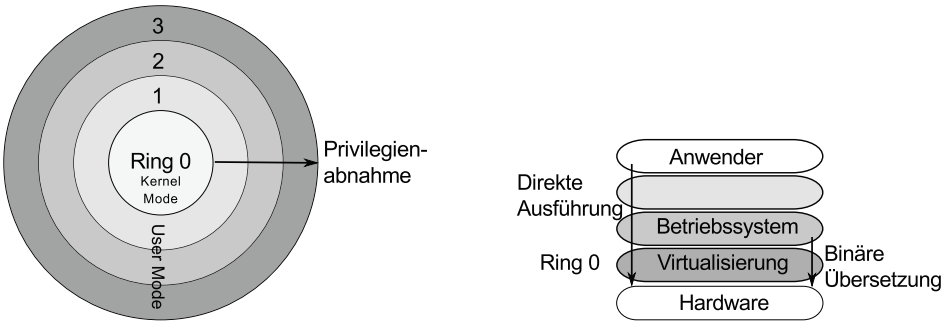
(b) Vollständige Virtualisierung (VMware Server)

Abbildung 3.8: Betriebssystem- und vollständige Virtualisierung

- Die *vollständige Virtualisierung* erlaubt die Ausführung von unmodifizierten Betriebssystemen in den virtuellen Maschinen und unterstützt daher insbesondere proprietäre, Closed Source-Betriebssysteme. Verglichen mit der Emulation liegt ein we-

sentlicher Unterschied in der notwendigen Übereinstimmung zwischen dem CPU-Typen im Host-System und in der darauf ausgeführten virtuellen Maschine.

Normalerweise erlaubt ein Betriebssystem Anwendungen einen nur eingeschränkten Zugriff auf die vorhandenen Hardware-Ressourcen. Die einzelnen Zugriffsebenen sind dabei durch Hardware bzw. verschiedene CPU-Modi realisiert und in graphischen Darstellungen oftmals ringförmig angeordnet (vgl. Abbildung 3.9(a)). Der innerste Ring 0 entspricht der untersten Zugriffsebene und damit der privilegiertesten Zugriffsart. Anwendungen auf dieser Ebene, wie etwa das Betriebssystem selbst, können direkt mit der Hardware interagieren. Da die Virtualisierungsschicht unter-



(a) Ring-Schema bei x86-Systemen

(b) Ring-Querschnitt bei vollständiger Virtualisierung, Quelle: [VMw07b]

Abbildung 3.9: Ring-Ansicht bei nicht-virtualisierten und virtualisierten Systemen

halb des Gast-Betriebssystems und oberhalb der Hardware angesiedelt sein muss, wird das Betriebssystem von Ring 0 auf Ring 1 angehoben. Privilegierte Instruktionen des Gast-Betriebssystems werden durch den Virtual Machine Monitor (VMM) abgefangen und verlustbehaftet emuliert bzw. in nicht-privilegierte übersetzt (vgl. Abbildung 3.9(b)). Für andere, z. B. durch einen Anwender ausgeführte Instruktionen in Ring 3 gilt, dass diese keinerlei Modifikationen benötigen und daher mit nativer Geschwindigkeit ausgeführt werden.

- Im Gegensatz zur vollständigen Virtualisierung kommt die *Paravirtualisierung* ohne die binäre Übersetzung privilegierter Instruktionen des Gast-Betriebssystems aus. Die Gast-Betriebssysteme greifen über Hypercalls und damit über eine durch die Virtualisierungssoftware bereitgestellte Schnittstelle auf die physische Hardware zu (vgl. Abbildung 3.10). Dies erklärt die notwendigen Modifikationen am Gast-Betriebssystem, denn hier müssen alle Aufrufe privilegierter Instruktionen durch Hypercall-Aufrufe ersetzt werden. Proprietäre bzw. Closed Source-Betriebssysteme wie Microsoft Windows sind aufgrund dessen nur in einer emulierten oder vollständig virtualisierten Umgebung ausführbar. Vergleicht man die Paravirtualisierung mit der vollständigen

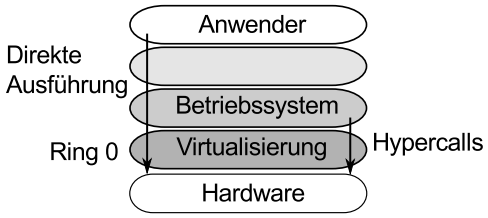


Abbildung 3.10: Ring-Architektur bei Paravirtualisierung, Quelle: [VMw07b]

Virtualisierung, so ist die Performance bei der ersteren höher, da die virtuellen Maschinen gleichsam direkt mit der Hardware kommunizieren können.

Xen Xen ist ein sehr bekannter Vertreter der Plattform-Virtualisierungssoftware und wurde anfänglich von der Systems Research Group der University of Cambridge entwickelt. Im Jahr 2003 wurde die Architektur von Xen 1.x (vgl. Abbildung 3.11) auf dem Symposium of Operating System Principles (SOSP) vorgestellt. Eines der initial avisierten Ziele war die Bereitstellung einer allgemeinen und homogenen Infrastruktur für das Verteilte Rechnen, die durch den Einsatz virtueller Maschinen erreicht werden sollte. Gegen Ende August 2010 erschien das Release 4.0.1 von Xen, wobei dessen Entwicklung mittlerweile durch kommerzielle Anbieter wie Citrix vorangetrieben wird.

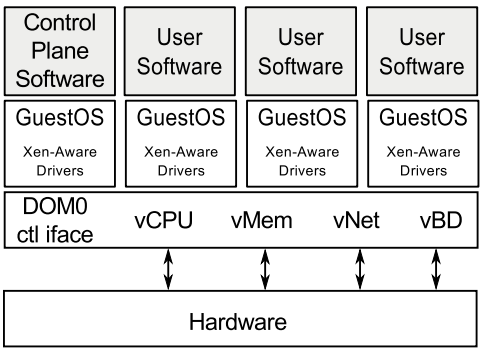


Abbildung 3.11: Architektur von Xen Version 1.x, Quelle: [BDF+03]

Grundlegendes Konzept hinter Xen ist die Erzeugung der privilegierten Domäne 0, kurz `dom0`, während des Startvorgangs des Host-Systems. Diese Domäne darf die in Abbildung 3.11 gezeigte Schnittstelle `ctl iface` nutzen, benötigt hierfür aber spezielle Werkzeuge, die in das in der Domäne 0 laufende Betriebssystem integriert werden müssen. Über den Zugriff auf diese Schnittstelle können beispielsweise weitere Domänen, die `domUs`, gestartet werden.

Entwicklungen im Bereich Plattform-Virtualisierung Nachfolgend sind weitere Entwicklungen der letzten Jahre kurz vorgestellt. Neben den Begriffen der Virtual Appliance (VA) und des Just Enough Operating System (JeOS) wird auch das Open Virtual Machine Format (OVF), welches sich u. a. mit einer Hersteller-neutralen Beschreibung von Virtual Appliances auseinandersetzt, beschrieben.

Die vielen Entwicklungen im Bereich der Plattform-Virtualisierung zeigten schnell den Bedarf an offenen, allseits akzeptierten Standards. Ein möglicher Fokus eines solchen Standards liegt in der Formulierung eines übergreifend verwendbaren Formats für die Beschreibung und Speicherung von Festplattenabbildern. VMware veröffentlichte in diesem Zusammenhang in [VMw07a] die Beschreibung des Formats Virtual Machine Disk (VMDK). Ebenso gab Microsoft im Jahre 2006 die Spezifikation des Formats Virtual Hard Disk (VHD) in [Mic06] bekannt.
Eine Konsequenz offener Standards bzgl. der Festplattenabbilder ist der höhere Wiederverwendungsgrad der Abbilder, da sich die Abbilder dann mit Virtualisierungssoftware verschiedener Hersteller nutzen ließen. Die bisherigen proprietären Lösungen führen die Anwender langfristig in den Status eines vendor lock-in. Um diesem lock-in entgegenzuwirken und Fremdkunden den Umstieg auf die unternehmenseigene Virtualisierungslösung zu vereinfachen, sind viele kommerzielle Anbieter dazu übergegangen, Migrationswerkzeuge zur Verfügung zu stellen. Zwei Beispiele hierfür sind VMware vCenter Converter (vgl. [VMw09]) und XenConvert (vgl. [Cit09]) der Firma Citrix.

Virtuelle Appliances Im engeren Sinn entspricht eine virtuelle Maschine nach Popek und Goldberg (vgl. [PG74]) einem vollständig in Software implementierten Duplikat eines physischen Servers. Wie ihr physisches Gegenstück ist eine virtuelle Maschine einzig und allein durch ihre Ressourcenkonfiguration bestehend aus, z. B. CPU, Hauptspeicher, Festplattenkapazität und Netzwerkschnittstellen beschrieben. Eine Virtual Appliance stellt hierzu eine Erweiterung dar und umfasst, wie in [VMw07c] beschrieben, einerseits analog zur virtuellen Maschine eine Ressourcenkonfiguration und andererseits eine Software-Konfiguration in Form von Betriebssystem und Anwendungssoftware, die in Festplattenabbildern vorinstalliert und -konfiguriert sind.

Diese Zusammenführung von Hard- und Software-Konfiguration in eine Virtual Appliance bietet sowohl für den Ressourcenbetreiber als auch den Anwender Vorteile. Der vom Anwender normalerweise auf einer Ressource benötigte Software-Stack kann in eine Virtual Appliance ausgelagert werden und ist somit vollständig gekapselt und von der physischen Umgebung auf der Ressource entkoppelt. Dem Ressourcenbetreiber obliegen beim Einsatz von Virtual Appliances daher lediglich die Aufgaben der Bereitstellung und des Managements der physischen Ressourcen. Neben der Einarbeitung sowie der Installation

und Konfiguration der Anwendungssoftware entfällt für den Betreiber ebenso die Unterstützung des Anwenders bei der Behebung von Problemen seiner Software mit der physischen Umgebung. Andererseits entlastet der Einsatz von Virtual Appliances den Anwender von der Notwendigkeit, den für ihn erforderlichen Software-Stack an die unterschiedlichen physischen Umgebungen bei den Ressourcenbetreibern anzupassen. Gerade in Verbindung mit Festplattenabbildern, die für verschiedene Virtualisierungsplattformen geeignet sind, ist dies ein immenser Vorteil.

Der Erfolg einer Virtual Appliance hängt u. a. von einer möglichst breiten Nutzerbasis und damit implizit verbunden der Unterstützung unterschiedlicher Virtualisierungslösungen ab. Mit Letzterem sind z. B. Xen und VMware und somit auch die unterschiedlichen Typen von Festplattenabbildern gemeint. Grundsätzlich existieren zwei Möglichkeiten die Unterstützung einer Appliance für mehrere solcher Lösungen zu realisieren. Die erste Möglichkeit beschäftigt sich mit der Überführung einer einmal erzeugten Appliance, vorliegend in Form der Festplattenabbilder und der Beschreibung der virtuellen Maschine, in ein anderes Ausgabeformat. Dieses Vorgehen erfordert die Umwandlung des alten in das neue Beschreibungsformat sowie die Transformation der vorliegenden Festplattenabbilder. Realisiert ist dieses Vorgehen z. B. im VMware Converter. Die zweite Möglichkeit setzt bereits vor der Zusammenstellung einer virtuellen Appliance an. Die später in der Appliance vorhandene Software wird nicht direkt in die virtuelle Maschine, sondern in eine `chroot`-Umgebung installiert (vgl. Abschnitt 2.3). Nach Abschluss der Installation und Konfiguration der Software übernimmt ein Prozess die Transformation dieser Umgebung in ein Festplattenabbild und erzeugt die Beschreibung der virtuellen Maschine im geforderten Format. Ein solches Vorgehen erweist sich besonders dann als wertvoll, wenn eine hinsichtlich der Virtualisierungssoftware heterogene Server-Landschaft vorhanden ist. Ohne weiteren großen Aufwand können in diesem Fall für die existierenden Varianten der Virtualisierungssoftware Appliances bereitgestellt werden.

Just Enough Operating System Im Laufe der Zeit nahmen Komplexität und Umfang marktüblicher Betriebssysteme, wie Linux oder Microsoft Windows, bedeutend zu. Dies ist beispielsweise anhand der Betrachtung des so genannten Footprints[3] (vgl. Tabelle 3.2) über mehrere Microsoft Windows-Versionen hinweg nachvollziehbar.

Die rapide Vergrößerung des Footprints, z. B. zwischen den Jahren 2001 und 2007 um den Faktor 10, ist zum Großteil durch die starke Zunahme an neu verfügbarer Hardware bedingt. Damit ein Betriebssystem diese neue Hardware von Hause aus unterstützt, werden die hierfür notwendigen Treiber und Bibliotheken in das Betriebssystem integriert. In der

[3] Bei einer Standardinstallation benötigter Festplattenspeicherplatz.

Jahr	Version des Betriebssystems	Footprint
1985	Windows 1.0	2 Disketten
1995	Windows 95	50 MByte
2000	Windows 2000 Professionell	1 GByte
2001	Windows XP	1.5 GByte
2007	Windows Vista Premium	15 GByte
2009	Windows 7	20 GByte

Tabelle 3.2: Footprints des Betriebssystems Microsoft Windows, Quellen: [Mic11, Mic07a, Mic10c, Mic10a, Mic10b, Mic07b]

Praxis ist in einem physischen Server jedoch nur ein Bruchteil der unterstützten Hardware vorhanden, d. h. der überwiegende Anteil der vom Betriebssystem bereitgestellten Treiber ist überflüssig.

Bei der Plattform-Virtualisierung ergibt sich ein ähnliches Szenario: die Virtualisierungsschicht präsentiert dem in einer virtuellen Maschine ausgeführten Betriebssystem eine sehr eingeschränkte und im Vorfeld bekannte Menge an „virtueller“ Hardware. Unter dieser Prämisse verfolgt man das Ziel, die Größe der von einer virtuellen Maschine verwendeten Festplattenabbilder zu minimieren. Dieses Ziel ist für Ressourcenanbieter und Anwender von Interesse, da beide davon profitieren: ein kleineres Festplattenabbild lässt sich etwa schneller zu einem Ressourcenanbieter transferieren und dort ebenso zügiger zwischen physischen Systemen migrieren.

Eine Vorgehensweise bei der Größenoptimierung ist das Löschen aller nicht in der späteren Virtual Appliance benötigten Treiber, Bibliotheken und Dienste. Das erste Betriebssystem, auf das dieses Vorgehen angewandt wurde, war Ubuntu Linux im September 2007[4]. Aus dieser Zeit stammt der Begriff des Just Enough Operating System (JeOS), welches einem Festplattenabbild mit minimalem Betriebssystemkern und unverzichtbaren Treibern für die Ausführung als Teil einer Virtual Appliance entspricht. Ergänzt wird es durch Werkzeuge zur Wartung und der Möglichkeit zur Nachinstallation weiterer Softwarepakete. Die kompakte Form des JeOS bildet somit einen guten Ausgangspunkt für die Erstellung von Virtual Appliances, da diese weitestgehend nur auf die Bereitstellung einer einzelnen Anwendung ausgelegt sind und der Appliance-Ersteller alle möglichen Abhängigkeiten zwischen Betriebssystem und Anwendung bereits im Vorfeld kennt und so auflösen kann. In einer so erstellten Appliance befinden sich somit nur solche Treiber und Bibliotheken, die für die ordnungsgemäße Ausführung der bereitzustellenden Anwendung unerlässlich sind.

[4] http://news.cnet.com/8301-13580_3-9776585-39.html

Einige kommerzielle Anbieter, wie Red Hat, Novell und Oracle, bieten mittlerweile selber JeOS-Varianten ihrer Software-Distributionen zum freien Download an. Beispiele aus dem kommerziellen, als auch dem Open Source-Bereich, sind in Tabelle 3.3 aufgeführt, wobei es für das in Abschnitt 2.1.2 vorgestellte Betriebssystem Scientific Linux noch kein solches JeOS gibt.

Linux-Distribution	Größe der JeOS-Datei
Oracle Linux	ca. 130 MByte
Ubuntu Linux	ca. 100 MByte
openSUSE	ca. 130 MByte
CentOS	nicht verfügbar

Tabelle 3.3: Beispiele verfügbarer JeOS

Open Virtual Machine Format Nach der Erzeugung einer Virtual Appliance ergibt sich die Frage nach deren Deployment auf den vorhandenen physischen Ressourcen. Die Vielfalt der am Markt verfügbaren Software für Plattform-Virtualisierung und deren unterschiedliche Konfigurationen zeigten die Notwendigkeit für ein Metadaten-Modell für Software-spezifische Hinweise zum Deployment. Als Antwort auf diesen erkannten Bedarf wurde das Open Virtual Machine Format (OVF) unter Mitwirkung von Hardware- (z. B. Dell, IBM, Intel und NEC), Betriebssystem- (z. B. Microsoft und Sun), Virtualisierungssoftware- (z. B. VMware und XenSource) und Virtual Appliance-Herstellern (z. B. Symantec) realisiert.

Das Open Virtual Machine Format stellt eine Hersteller-neutrale, erweiterbare Spezifikation für die Zusammenstellung und die Verteilung von so genannten virtuellen Systemen, die aus einer oder mehreren Virtual Appliances bestehen, dar (vgl. [VX07b, VX07a]). Die Kapselung mehrerer Virtual Appliances zu einem virtuellen System findet u. a. gerade im Bereich der Multi Tier-Architekturen[5] Anwendung. Das Open Virtual Machine Format ermöglicht die Beschreibung und das spätere Deployment dieser Tier-Komponenten als zusammenhängende Einheit.

[5] Multi Tier-Architekturen verfügen über verteilt installierte, interagierende Komponenten mit disjunkten Aufgaben. Im Geschäftsbereich findet man oftmals die drei Tiers Präsentation, Geschäftslogik und Datenhaltung.

3.2 Workflows

Für die Automation eines Gesamtablaufs oder einzelner Teile gilt es, die einzelnen durchzuführenden Aktivitäten – hier die durchgeführten Schritte bei der Installation und Konfiguration von Software – zu identifizieren und zu strukturieren. Zum Erreichen einer ausreichend hohen und damit allgemein verständlichen Abstraktionsebene, bietet sich der Einsatz einer graphischen Darstellung an. Eine Nebenbedingung an die Wahl der graphischen Darstellung ist deren einfache Umwandlung in eine (semi-)automatisierte Ausführung.

Zunächst erfolgt eine kurze Einführung in das Thema Workflows und eine Erläuterung der verwendeten Begriffe. In Abschnitt 3.2.1 sind Ereignisgesteuerte Prozessketten als eine Möglichkeit zur graphischen Modellierung von Prozessen beschrieben. Die beiden darauf folgenden Abschnitte 3.2.2 und 3.2.3 stellen mit der XML Process Definition Language (XPDL) und der Business Process Modeling Notation (BPMN) zwei Beschreibungssprachen für Workflows vor und zeigen deren Verwendung beispielhaft an zwei Aufgaben. Abschließend sind in einer Evaluation beide Sprachen gegenübergestellt.

Georgakopoulos et al. beschreiben in [GHS95] einen Workflow als Ansammlung untereinander angeordneter Aufgaben, die zur Durchführung eines Geschäftsprozesses notwendig sind, verweisen jedoch auch auf weitere existierende Definitionen, wie etwa in [Kim95]. Dort wird ein Workflow als eine Aktivität beschrieben, zu deren Gelingen die koordinierte Ausführung multipler Aufgaben durch unterschiedliche Entitäten erforderlich ist. Die Workflow Management Coalition (WfMC) liefert in [Wor99] eine weitere Definition mit

> The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.

Neben den vielfältigen Definitionen existieren abhängig vom Betrachter mindestens ebenso viele Möglichkeiten zur Kategorisierung der Workflows. Für Arbeitsabläufe innerhalb eines Unternehmens ist beispielsweise eine Einteilung in administrative Workflows, Produktionsworkflows und ad hoc-Workflows möglich, wie sie in [GHS95] vorgeschlagen wird.

- Administrative Workflows entsprechen strukturierten, wiederkehrenden und vorhersagbaren Prozessen (z. B. Routineabläufen), die nur selten zeitkritisch und für ein Unternehmen von geringem Geldwert sind. Durch die Strukturierung der einzelnen Aktivitäten ist deren koordinierte Ausführung gut automatisierbar.

- Produktionsworkflows entstehen analog zu den administrativen Workflows aus sich wiederholenden, vorhersagbaren Prozessen. Der Schwerpunkt liegt hier jedoch auf Geschäftsprozessen, die meist zeitkritisch und strategisch bedeutend sind. Zudem

sind diese im direkten Vergleich mit administrativen Workflows von höherer Komplexität und involvieren multiple Systeme (z. B. solche anderer Unternehmen). Dennoch lässt sich auch hier durch den vorhersagbaren Charakter die Koordination und Ausführung der einzelnen Aufgaben gut automatisieren.

- Ad hoc-Workflows treten dort auf, wo a priori bekannte Prozessstrukturen fehlen. Meist findet man diese Form von Workflows in Arbeitsbereichen, die über einen hohen Anteil an menschlicher Interaktion verfügen, z. B. bei einer Angebotserstellung für Unternehmenskunden. Der Mensch trifft hier „zur Laufzeit“ Entscheidungen hinsichtlich der Anordnung und Koordination einzelner Aktivitäten sowie der Koordination mit anderen Systemen.

Anhand dieser Kategorisierung wird deutlich, dass innerhalb eines Unternehmens nur die Produktionsworkflows und administrativen Workflows für eine computergestützte Automatisierung geeignet sind. Letztere besitzen allerdings gegenüber den Produktionsworkflows eine geringere Priorität, da Produktionsworkflows einem größeren Geldwert gleichkommen.

Neben der gerade vorgestellten Einteilung sind Workflows z. B. auch aufgrund des Grades der Mensch- bzw. Systemorientierung einteilbar. Eine der weit verbreiteten Untergliederungen ist jedoch die Einteilung in Datenfluss- und Kontrollfluss-zentrische Workflows. Allerdings verwendet man unabhängig von der Kategorisierung bei Workflows ein einheitliches Vokabular, welches im folgenden Abschnitt beschrieben ist.

Begrifflichkeiten Die Ausgangsbasis für einen Workflow ist der Prozess. Aus diesem Prozess wird mittels einer Spezifikationssprache und einem Workflow-Modell die Workflow-Spezifikation erstellt (vgl. Abbildung 3.12). In den Abschnitten 3.2.2 und 3.2.3 sind die

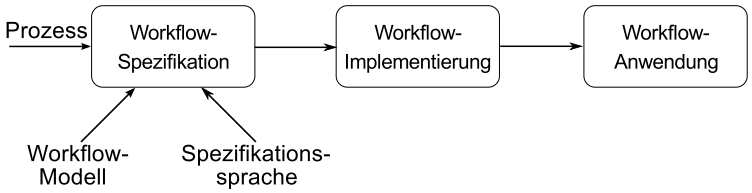


Abbildung 3.12: Ablauf einer Workflow-Erstellung, Quelle: [GHS95]

XML Process Definition Language und die Business Process Modeling Notation als zwei solche Spezifikationssprachen vorgestellt. Das zu der XML Process Definition Language gehörende Workflow-Modell ist in Abbildung 3.17 dargestellt, das entsprechende Modell für die Business Process Modeling Notation beschreibt Barkmeyer in [Bar08a]. Einzelne Bestandteile der Workflow-Spezifikation sind z. B. der Kontrollfluss, die Behandlung etwaiger Ausnahmen und Fehler sowie weitere Attribute für Aufgaben (z. B. Dauer und Priorität).

Nach dem Abschluss der Spezifikationsphase beginnt die technische Umsetzung durch die Implementierung der Logik, so dass letztendlich die fertige Workflow-Anwendung vorliegt.

Der Zusammenhang zwischen den wesentlichen Spezifikationselementen ist in Abbildung 3.13 aufgezeigt und die Bedeutung der einzelnen Elemente nachfolgend beschrieben.

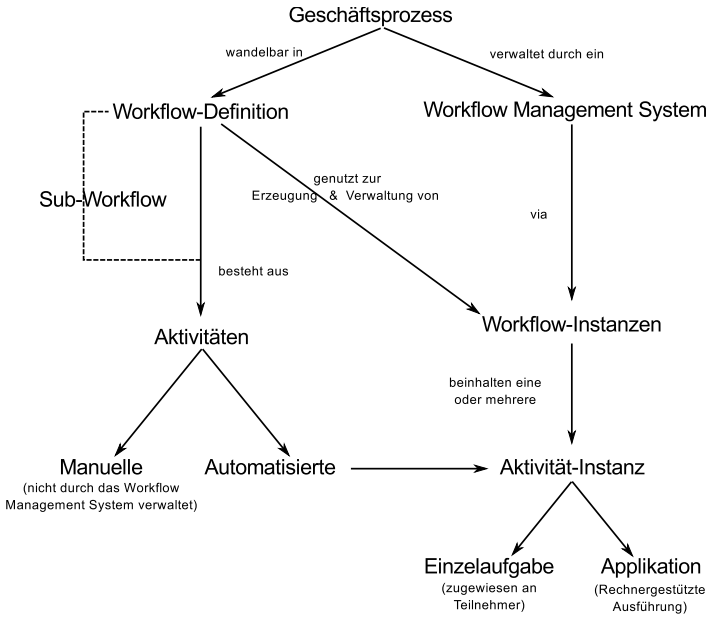


Abbildung 3.13: Beziehungen zwischen einzelnen Begriffen der Workflow-Terminologie, Quelle: [Wor99]

- Ein *Prozess* beschreibt die Transformation von Eingaben zu Ausgaben durch die Anwendung einzelner Arbeitsschritte, die durch Entitäten ausgeführt werden. Die einzelnen Arbeitsschritte können wiederum aus einem oder mehreren Prozessen bestehen. Neben den reinen Nutzinformationen verfügt ein Prozess über administrative Informationen (z. B. den Prozesserzeuger) und zur Laufzeit benötigte Informationen wie die Ausführungspriorität. Johansson et al. definierten in [JMPW94] u. a. folgende Eigenschaften, die ein Prozess erfüllen sollte:

  - Definierte Wertebereiche für die als Eingabe übergebenen Argumente und den Inhalt der Ausgabe.
  - Direkter Zusammenhang zwischen den Positionen der Aktivitäten im Prozess und ihrer zeitlichen Anordnung.
  - Die Prozessausführung erzeugt einen Mehrwert für den Kunden und/ oder den Prozesseigentümer.

  - Der Prozess existiert nicht allein und ist daher in eine Umgebung zu integrieren.

- In einem Workflow beschreibt eine *Aktivität* einen einzelnen, in sich abgeschlossenen und aus einem Prozess extrahierten Schritt. Dieser Schritt ist nur aus dem Workflow selbst heraus aufrufbar und wird entweder manuell oder automatisiert abgearbeitet, wofür Ressourcen in menschlicher Form bzw. in Form von Hardware aufgewendet werden müssen. Eine Aktivität selbst kann wieder aus einer oder mehreren Aktivitäten bestehen und verwaltet intern – analog zu Prozessen – Meta-Informationen, etwa über die eigene Priorität.

- Der *Teilnehmer* ist an der Ausführung des Workflows bzw. einzelner Aktivitäten beteiligt. Die Menge an Teilnehmern muss nicht zwingend nur aus Menschen bestehen, sondern kann auch organisatorische Einheiten, Rollen und maschinenbasierte Agenten beinhalten.
  Die Ermittlung der notwendigen Workflow-Teilnehmer erfolgt üblicherweise während der Workflow-Spezifikation, wobei zeitgleich eine Abbildung der vorhandenen Ressourcen auf die verschiedenen Teilnehmer erzeugt wird. Aus dieser kann das Workflow Management System (WfMS) zur Laufzeit eine geeignete Ressource zur Abarbeitung einer Aktivitätsinstanz auswählen.

- Die Übergänge zwischen den einzelnen Aktivitäten sind durch *Transitionen* beschrieben, wobei jede Transition über drei Komponenten verfügt: i) die Quellaktivität, ii) die Zielaktivität und iii) die Übergangsbedingung. Letztere wird zur Laufzeit ausgewertet und kann leer sein (unconditional transition) oder boolesche Ausdrücke enthalten (conditional transition). Abhängig von der Anordnung der Transitionen ergibt sich eine sequentielle oder parallele Abarbeitung der Aktivitäten.

- Eine *Applikation* entspricht einer Anwendung, die zur Abarbeitung einer Aktivität aufgerufen wird, wobei allerdings einer Aktivität mehrere Applikationen zugeordnet sein können. In der Applikationsbeschreibung lassen sich u. a. die zu übergebenden Argumente und Rückgabewerte festhalten. Neben lokal implementierten Funktionen des Workflow Management Systems sind ebenso externe Anwendungen als Applikationen möglich.

- *Workflow-relevante Daten (WfrD)* werden während der Ausführung des Workflows erzeugt bzw. verarbeitet und können bei Verwendung von conditional transitions auf die Übergänge zwischen den einzelnen Aktivitäten Einfluss haben. Da diese Daten allen Aktivitäten des Workflows zur Verfügung stehen, lassen sich so beispielsweise Zwischenergebnisse zwischen den Aktivitäten propagieren. Ebenso ist eine Mani-

pulation der Daten durch Applikationen als auch durch die Workflow Engine selbst möglich.

Üblicherweise sind Workflow-relevante Daten typisiert. Die in Abschnitt 3.2.2 vorgestellte XML Process Definition Language unterstützt neben den einfachen Typen wie `String`, `Float`, `Integer` und `Reference` mit `RecordType`, `UnionType`, `EnumerationType`, `ArrayType` und `ListType` auch komplexe.

- Für den Begriff des *Workflows* wurden zu Beginn dieses Abschnitts verschiedene Definition vorgestellt. Im Folgenden wird die Definition nach [Wor99] zugrunde gelegt. Ist eine erste Definition des Prozesses erstellt, so muss diese für eine teilweise oder auch vollständig automatisierte Abarbeitung in einen Workflow umgewandelt werden. Der Zusammenhang zwischen Prozess- und Workflow-Modell sowie der iterative Verfeinerungsprozess sind in Abbildung 3.14 skizziert. Die Erzeugung des

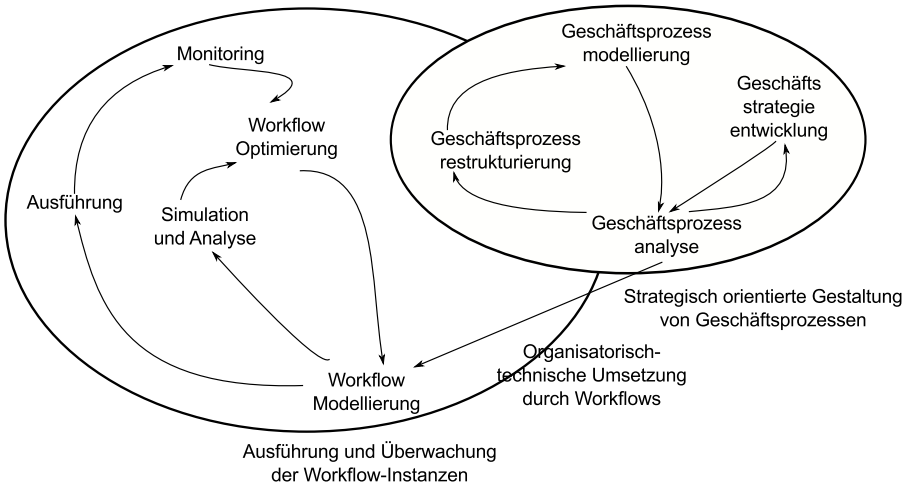


Abbildung 3.14: Prozess- und Workflow-Modell, Quelle: [Gad10]

Workflows beschreibt Gadatsch in [Gad10] als organisatorisch-technische Umsetzung der erstellten Prozessdefinition. Nach der Erstellung des Workflows stehen dem Workflow-Erzeuger für die Ausführung des Workflows zwei Möglichkeiten offen: die Ausführung als Teil einer Simulation und die Ausführung auf dem realen System. In letzterem Fall ist die Überwachung der Workflow-Ausführung von besonderer Bedeutung. Sowohl anhand der Simulation und Analyse als auch anhand der Ausführung auf dem realen System ist eine Optimierung des Workflows durchführbar. Die Ergebnisse der Optimierung können insbesondere eine Rückwirkung auf die Modellierung des Workflows haben.

- Bei einer *Workflow Engine* handelt es sich um einen Dienst, der eine Laufzeitum-

gebung für die Ausführung zuvor definierter Workflows bereitstellt. Im Rahmen der Ausführung ist die Engine z. B. in der Lage einzelne Workflow-Beschreibungen zu interpretieren und entsprechende Instanzen zu verwalten.

- Ein *Workflow Management System* überwacht die Ausführung zuvor definierter Workflows. Hierzu kann das Workflow Management System eine oder mehrere Workflow Engines einsetzen, um die jeweils für einen Workflow geeignete Laufzeitumgebung zur Verfügung zu stellen. Des Weiteren überwacht das System die Interaktion der Workflow-Instanzen mit den Teilnehmern und Applikationen.

3.2.1 Ereignisgesteuerte Prozessketten

Keller et al. beschreiben in [KNS01] die Möglichkeit der Modellierung von Geschäftsprozessen durch Ereignisgesteuerte Prozessketten (EPK). Die Ereignis-, Funktions- und Konnektorknoten (vgl. Abbildung 3.15) stellen einen kleinen Ausschnitt der verschiedenen bei dieser Form der Prozessmodellierung vorhandenen Komponenten dar und werden durch Kanten zur Darstellung des Kontrollflusses ergänzt. Ein Ereignis beschreibt in einer EPK

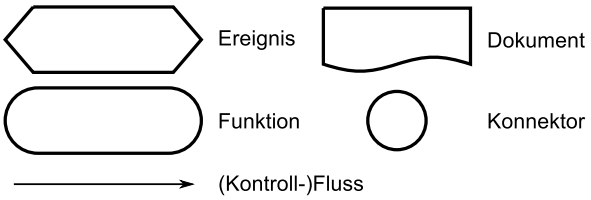


Abbildung 3.15: Komponenten der Prozessmodellierung, Quelle: [BES01]

das Eintreten eines Zustandes, der entweder eine Funktion (Handlung) auslöst oder das Resultat einer solchen ist. Die Funktion selbst enthält die Beschreibung der beim Eintreten des Ereignisses auszuführenden Aktionen. Eine ihrer wesentlichen Eigenschaften ist daher der Verbrauch von Ressourcen wie Personal und Zeit. Mittels der Konnektoren lassen sich Ereignisse und Funktionen logisch miteinander verknüpfen, wobei als Operationen z. B. `AND`, `OR` oder `XOR` unterstützt werden. Üblicherweise existieren in Unternehmen auch Dokumente, die zwischen einzelnen Abteilungen oder mit der Außenwelt ausgetauscht werden müssen. Aus diesem Grund sind Dokumente ebenso als Bestandteil einer Prozesskette modellierbar.

Ein Ziel bei der Erstellung von Prozessketten ist die vereinfachte Darstellung der unternehmensinternen Prozesse, wobei der Detailgrad des Modells vom Ersteller selbst abhängt. Jedoch haben alle Modelle - unabhängig vom Detailgrad - die Einhaltung der chronologischen bzw. sachlogischen Abfolge der Tätigkeiten als gemeinsames Merkmal.

Einen Ausschnitt aus einer Prozesskette für den Bücherversand und die anschließende Rechnungserstellung zeigt Abbildung 3.16. Der Ausschnitt beginnt mit der Funktion des Verpackens und Versendens der bestellten Bücher eines Kunden. Der Kontrollfluss spaltet sich anschließend abhängig vom gewählten Paketdienst durch einen XOR-Konnektor in drei Zweige mit jeweils einem Ereignis auf. Nach der Abarbeitung des entsprechenden Ereignisses fallen die drei Zweige wieder über einen XOR-Konnektor zusammen und gehen in die Funktion der Rechnungserstellung und des Rechnungsversands über.

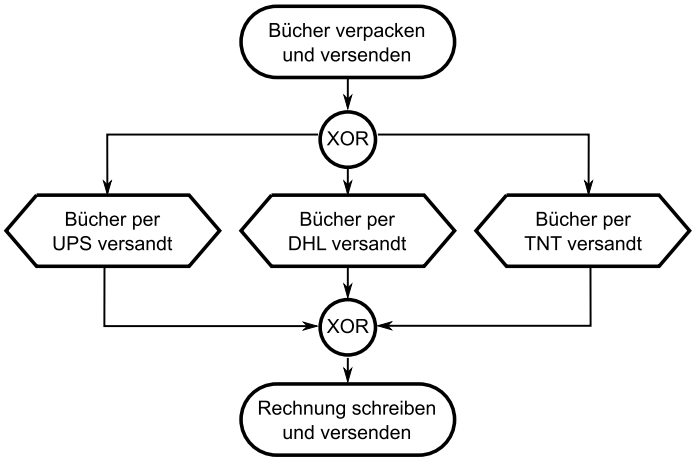


Abbildung 3.16: Ausschnitt aus einer Prozesskette, Quelle: [BES01]

Im Gegensatz zu beispielsweise der später vorgestellten Business Process Modeling Notation hinterliegt den Ereignisgesteuerten Prozessketten keine durch eine Organisation durchgeführte Standardisierung. Dies führt zu einer nur mäßigen Akzeptanz und Verbreitung außerhalb Deutschlands sowie einem nur schwach ausgebauten Ecosystem, dem u. a. Workflow Engines fehlen. Daher werden EPKs als mögliche Realisierung nicht weiter verfolgt.

3.2.2 XML Process Definition Language

Die in [Sha08b] vorgestellte XML Process Definition Language bietet eine weitere Möglichkeit zur Modellierung von Prozessen und wurde im Jahr 2008 in der Version 2.1a durch die Workflow Management Coalition, eine Allianz aus Anwendern, Herstellern und Beratern, mit dem Ziel der Standardisierung von Begriffen und Schnittstellen bei technisch unterstützten Geschäftsprozessen verabschiedet. Neben der weitaus bekannteren Business Process Modeling Notation (vgl. Abschnitt 3.2.3) konnte sich die XML Process Definition Language als Standard durchsetzen und findet aktuell beispielsweise Anwendung in den Bereichen Enterprise Resource Planning (ERP), Business Activity Monitoring (BAM)

und Customer Relationship Management (CRM). Im Bereich des Grid Computing wird die XML Process Definition Language, wie in [BRSW09] beschrieben, innerhalb der UNICORE Middleware verwendet.

Einige der im XPDL-Metamodell (vgl. Abbildung 3.17) verwendeten Begriffe, wie Application, Participant und Transition, wurden bereits vorgestellt, andere Begriffe, wie Pool und Lane, sind BPMN-spezifische Erweiterungen und entsprechend in Abschnitt 3.2.3 vorgestellt. Deutlich im Metamodell zu erkennen ist die Aggregationsbeziehung zwischen dem Prozess und den Teilnehmern, Applikationen, Aktivitäten und Daten. Externe Informationen sind dem Prozess über die Formulierung als Workflow-relevante Daten zuführbar.

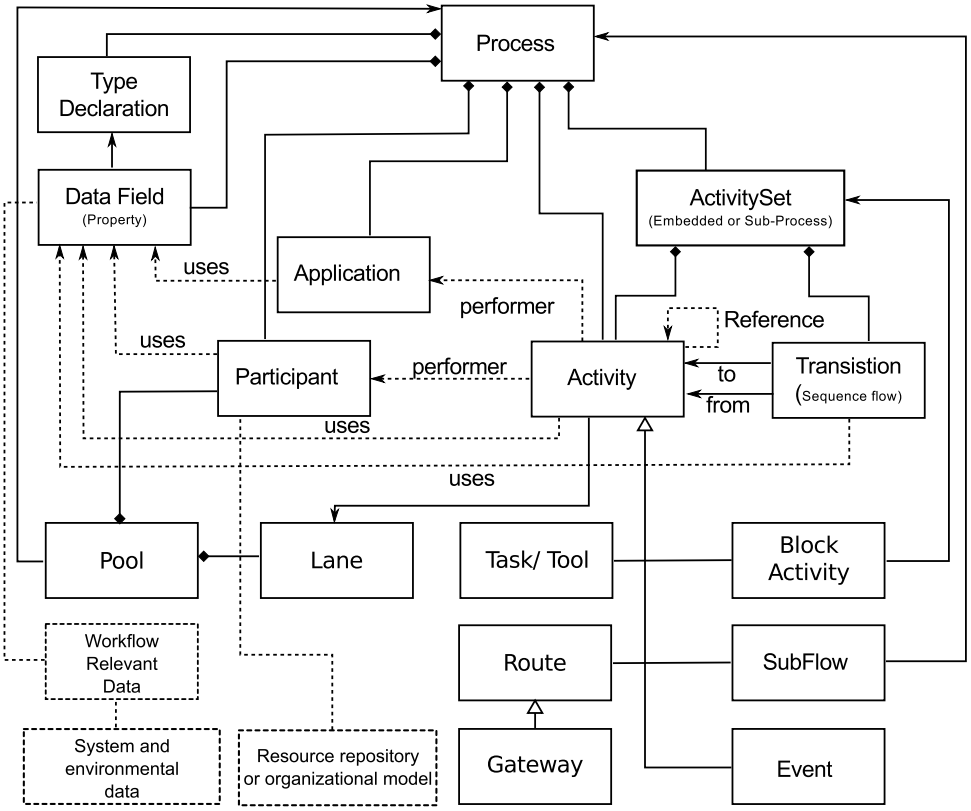


Abbildung 3.17: XPDL 2.0-Metamodell, Quelle: [Sha08b]

Neben einfachen Aktivitäten unterstützt XPDL weitere, komplexe Aktivitäten.

- Die *Dummy-Aktivität* spielt bei komplexen Routing-Entscheidungen eine Rolle und dient etwa der Zusammenführung bzw. Aufspaltung des Kontrollflusses. Da in dieser Aktivität selbst kein Prozessierungsschritt erfolgt, sind ihr keine Applikationen zugeordnet.

- Die Aktivität *Schleife* erzwingt eine sich wiederholende Ausführung der im Schleifenrumpf enthaltenen Aktivitäten bis ein Abbruchkriterium erfüllt ist.

- Ein *Subflow* ist ein Workflow, der aus einem anderen Workflow heraus aufgerufen wird. Dabei können dem Subflow Argumente übergeben und am Ende des Subflows Rückgabewerte zurückgeliefert werden. Eine Anwendung finden die Subflows bei der Definition wiederverwendbarer Abschnitte einzelner Prozesse.

3.2.3 Business Process Modeling Notation

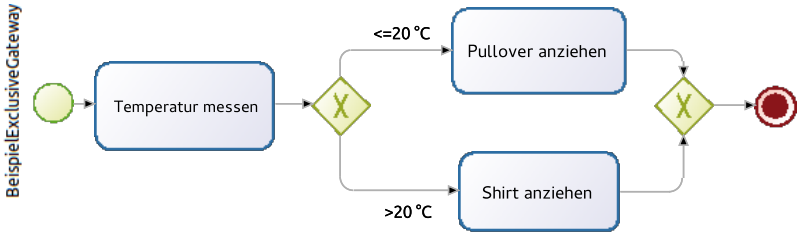
Im Laufe der vergangenen Jahre etablierte sich die Business Process Modeling Notation (BPMN) zu einem weit akzeptierten Standard für die visuelle Darstellung von Prozessen. Die graphische Darstellung ermöglicht ein besseres Prozessverständnis und vereinfacht die Interaktion mit anderen Organisationen auf Prozessebene.

Im Jahr 2006 wurde die Version 1.0 der Business Process Modeling Notation als Standard der Object Management Group (OMG) aufgenommen. Der ursprüngliche Zweck der Notation war die Realisierung der graphischen Darstellung von Prozessen, die in der Business Process Modeling Language (BPML) verfasst wurden. Die derzeit aktuelle BPMN Version 1.2 wurde im Jahre 2009 von der Business Process Management Initiative (BPMI) verabschiedet (vgl. [AAA+09]) und für das Ende des Jahres 2010 rechneten viele Unternehmen mit dem Erscheinen des Nachfolgers BPMN Version 2.0. Daher existieren bereits viele kompatible Modellierungswerkzeuge.

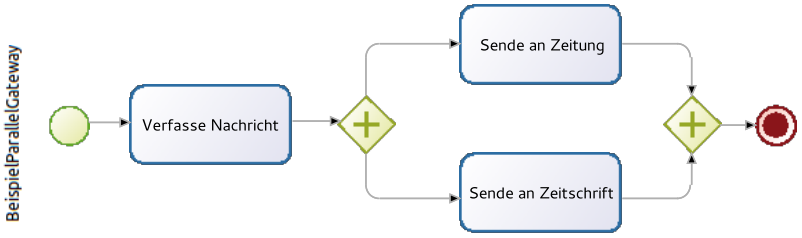
Analog zur XML Process Definition Language existieren in der Business Process Modeling Notation beispielsweise Konstrukte zur Realisierung von Aktivitäten und Subflows. Die Aufspaltung und Zusammenführung des Kontrollflusses ist an den so genannten Gateways möglich, welche sich anhand ihrer Funktionsweise in exklusive, inklusive und parallele Gateways einteilen lassen (vgl. Abbildung 3.18). Während beim parallelen Gateway alle ausgehenden Pfade durchlaufen bzw. ausgeführt werden, ist es beim exklusiven Gateway genau einer.

Die Business Process Modeling Notation verfügt zur visuellen Darstellung der Zugehörigkeiten von Elementen zu organisatorischen Entitäten (z. B. Abteilungen) über die beiden Konstrukte *Pool* und *Swim-Lane*, kurz Lane. Üblicherweise ist ein Pool einem Teilnehmer des Workflows zugeordnet und kapselt dessen Aktivitäten visuell gegenüber denen anderer Teilnehmer ab. Die Interaktion zwischen einzelnen Pools erfolgt über den Austausch von Nachrichten. Eine Lane bildet eine Partition innerhalb eines Pools und hilft dabei, die Aktivitäten innerhalb des Pools zu strukturieren.

Ein Beispiel für einen einfachen mittels der Business Process Modeling Notation modellierten Geschäftsprozess ist in Abbildung 3.19 dargestellt. Die Abgeschlossenheit des Prozesses „Stelle ausschreiben“ wird durch die Fassung in einen Pool ausgedrückt. Innerhalb des Pools existieren mit einer Fachabteilung und einer Personalabteilung zwei unterschied-



(a) Exklusiver Gateway



(b) Paralleler Gateway

Abbildung 3.18: Beispiele für BPMN-Gateways

liche organisatorische Einheiten, die jeweils über eine separate Lane verfügen. Die Rechtecke mit den abgerundeten Ecken stellen die eigentlichen Aktivitäten in den Einheiten dar. Die Abarbeitung des Prozesses beginnt beim Startpunkt, der durch einen einfachen, in der

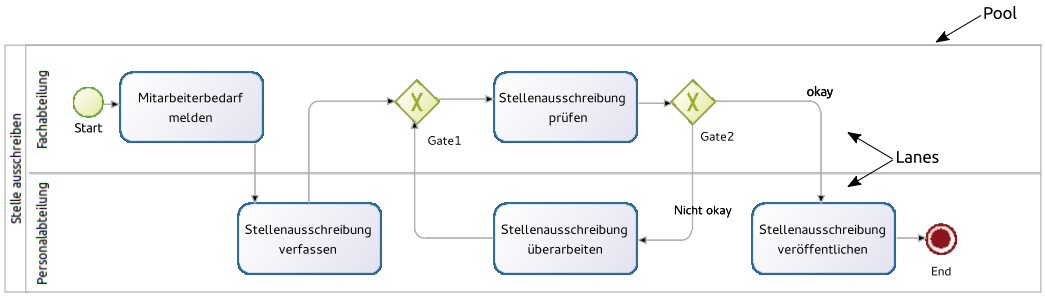


Abbildung 3.19: Beispiel für einen einfachen BPMN-Workflow, Quelle: [All09]

Abbildung grün gekennzeichneten Kreis dargestellt ist. Der Endpunkt der Abarbeitung ist durch einen roten Kreis mit dickem Rand symbolisiert.

3.2.4 Evaluation der Beschreibungssprachen

Dieser Abschnitt zeigt an zwei Beispielen die Verwendung der XML Process Definition Language und der Business Process Modeling Notation zur Definition von Workflows und ebenso die Implementierung der zugehörigen Aktivitäten. Im Fall der XML Process Definition Language erfolgt die Implementierung basierend auf der ZOPE 3 Workflow Management Coalition Engine, die in Abschnitt 3.2.4.1 beschrieben ist. Die Open Source-Software

Bonita[6] und Groovy finden bei der BPMN-Implementierung Verwendung.

Das Ziel des ersten Workflows ist es, nicht mehr benötigte Dienste aus der Liste der automatisch beim Hochfahren des Systems gestarteten Dienste zu entfernen. Der zweite Workflow installiert das Software-Paket `lcg-CA` auf einem System und konfiguriert zuvor die hierzu benötigten Software-Repository-Informationen.
Die erstellten Implementierungen der Workflows nutzen auf Applikationsebene teilweise betriebssystemspezifische Konfigurationen, wobei nachfolgend von Scientific Linux als Betriebssystem ausgegangen wird.

3.2.4.1 XML Process Definition Language

Der Workflow zur Deaktivierung nicht mehr benötigter Dienste (vgl. Abbildung 3.20) besteht aus den Aktivitäten `existsService` und `disableService` sowie den Dummy-Aktivitäten `ErrorHandling` und `Join`. Als Eingabe erhält der Workflow die Liste der zu deaktivierenden Dienste. Zudem kann nach Beendigung des Workflows über das Auslesen der Variablen `exitStatus` der Abarbeitungsstatus (erfolgreich/ nicht erfolgreich) festgestellt werden. Workflow-intern gibt auch jede der Nicht-Dummy-Aktivitäten nach ihrer Ausführung einen `exitStatus` zurück.

Die Übergänge `existsService` → `disableService` und `disableService` → `Join` erfolgen bedingt und nur genau dann, wenn der `exitStatus` der jeweils ersten Aktivität eine erfolgreiche Ausführung anzeigt. Wird beispielsweise die Existenz des zu deaktivierenden Dienstes im System festgestellt (`exitStatus` von `existsService` gleich 0), dann erfolgt der Übergang zur Aktivität `disableService`. Andernfalls wird in die Aktivität `ErrorHandling` gewechselt, die in diesem Beispiel nicht implementiert und daher durch eine Dummy-Aktivität dargestellt ist. In der Dummy-Aktivität `Join` sind beide Ausführungszweige, normaler Verlauf und Ausnahmebehandlung, wieder zusammengeführt.

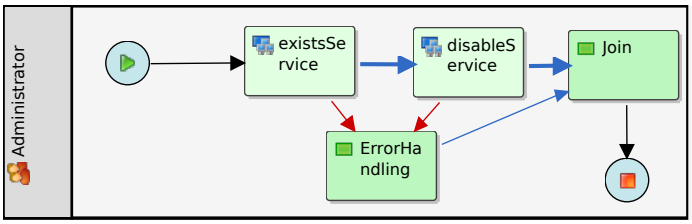


Abbildung 3.20: XPDL-Workflow zur Deaktivierung von Diensten

Eine mögliche Implementierung für die Aktivität `existsService` ist in Listing 3.1 gezeigt, wobei für Einzelheiten der ZOPE 3 Workflow Management Coalition Engine auf

[6] `http://www.bonitasoft.com/`

den nachfolgenden Abschnitt verwiesen wird. Der Funktion `start` der Klasse `ExistsService` wird die Liste der zu deaktivierenden Dienste in Form des `String`-Felds `listOfServices` übergeben. Während einer Iteration über dieses Feld wird per `checkFileExists`-Funktion geprüft, ob im Verzeichnis `/etc/init.d/` eine Datei mit entsprechendem Namen für den Dienst existiert. Ist dies nicht der Fall, gibt die Funktion den Wert `-1` zurück, woraufhin in der `start`-Funktion eine Ausnahme geworfen wird. Im ausnahmefreien Fall bleibt der Wert `0` in der Variablen `exitStatus` erhalten und wird über den Aufruf `self.activity.workItemFinished` als Rückgabewert dieser Applikation an den Aufrufenden zurückgegeben.

```
1 class ExistsService(ApplicationBase):
      def start(self, listOfServices):
3         exitStatus = 0
          for service in listOfServices:
5             exitStatus = checkFileExists("/etc/init.d/" + service)
              if exitStatus == -1:
7                 raise Exception('Service not existing.')
          self.activity.workItemFinished(self, exitStatus)
```

Listing 3.1: Beispielhafte Implementierung der XPDL-Applikation `existsService`

Wie sich eine XPDL-Applikation mit einer Aktivität verknüpfen lässt, ist in Listing 3.2 gezeigt. Die Datei, welche die Beschreibung des Workflows in der XML Process Definition Language enthält, wird durch die Instruktion `xpdl.read` geöffnet. Der Befehl in Zeile 2 selektiert den auszuführenden Workflow `disableService`. Letzteres ist notwendig, da eine Datei mehrere Workflow-Beschreibungen enthalten kann.
Die explizite Abbildung von Aktivitäten auf Applikationen erfolgt in den Zeilen 7 und 8 und erfordert ein Objekt des Typs `AttributeIntegration`. Den einzelnen aus der Workflow-Beschreibung abgeleiteten `WorkItems` werden Python-Klassen zugeordnet, die allesamt von `ApplicationBase` abgeleitet sein müssen (vgl. Listing 3.1).
Gestartet wird der Workflow, welcher mit `["apcid", "yum"]` die Liste der zu deaktivierenden Dienste als Eingabe erhält, über den Aufruf der `start`-Funktion in Zeile 11.

```
  PACKAGE = xpdl.read(open(os.path.join('./', 'disableServices.xpdl')))
2 PD = PACKAGE[u'disableService']
  INTEGRATION = AttributeIntegration()
4 PD.integration = INTEGRATION
  INTEGRATION.AdministratorParticipant = Administrator
6 zope.component.provideUtility(PD, name=PD.id)
  INTEGRATION.existsServiceWorkItem = ExistsService
8 INTEGRATION.disableServicesWorkItem = DisableServices
```

```
CONTEXT = PublicationContext()
PROC = PD(CONTEXT)
PROC.start(["apcid", "yum"])
```

Listing 3.2: Zusammenführung von XPDL-Aktivität und -Applikation

Das Ziel des zweiten Beispiel-Workflows ist die Installation des Software-Pakets `lcg-CA`. Zunächst werden im dazugehörigen Workflow (vgl. Abbildung 3.21) während der Aktivität `configureRepository` Informationen über das zu verwendende Software-Repository im Verzeichnis `/etc/yum.repos.d/` hinterlegt, bevor die Folgeaktivität `installPackages` das Paket `lcg-CA` installiert. Analog zum vorherigen Beispiel besitzt i) der Workflow die Dummy-Aktivitäten `ErrorHandling` und `Join` sowie ii) jede der Nicht-Dummy-Aktivitäten einen Rückgabewert `exitStatus`.

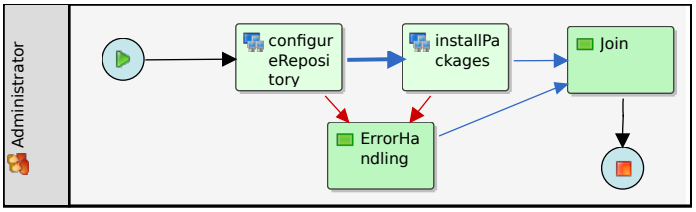


Abbildung 3.21: XPDL-Workflow zur Installation des Pakets `lcg-CA`

Im ersten Schritt wird die Workflow-Beschreibung aus der Datei `example.xpdl` geladen (vgl. Listing 3.3). und der auszuführende Workflow festgelegt, hier per `PACKAGE[U'installLCGCA']`.

```
from zope.wfmc import xpdl
import os
PACKAGE = xpdl.read(open(os.path.join('./', 'example.xpdl')))
PD = PACKAGE[U'installLCGCA']
```

Listing 3.3: Laden einer XPDL-Beschreibung

Der Folgeschritt (vgl. Listing 3.4) bildet die Grundlage für die Verbindung zwischen der eingelesenen Workflow-Beschreibung und der Logik über ein Objekt des Typs `AttributeIntegration`. Dieses wird dem Attribut `integration` des in Listing 3.3 enthaltenen `PD`-Objekts (`PD` = Prozessdefinition) zugewiesen.

```
from zope.wfmc.attributeintegration import AttributeIntegration
Integration = AttributeIntegration()
PD.integration = Integration
```

Listing 3.4: Einsatz der `AttributeIntegration`

Im Anschluss daran erfolgt die Verknüpfung zwischen den in der Workflow-Beschreibung existierenden Teilnehmern und den Teilnehmern auf Applikationsebene (vgl. Listing 3.5). Die ausführende Rolle des Administrators ist auf Applikationsebene von der Klasse `Participant`, welche die Basisklasse für alle Teilnehmer darstellt, abgeleitet.

```
1  Integration.Administrator = Administrator
   Integration.Participant = Participant
```

Listing 3.5: Einbindung von Teilnehmern

Zuletzt wird die Prozessdefinition `PD` mittels des `Provideutility`-Aufrufs als Zope-Komponente registriert und mit `Proc` eine Workflow-Instanz generiert. Dieser sind beim Aufruf der Methode `start` die notwendigen Argumente zuzuführen (vgl. Listing 3.6). In diesem Fall entsprechen die Argumente den erforderlichen Informationen über das zu verwendende Software-Repository.

```
   Zope.Component.Provideutility(PD, Name=PD.Id)
2  Context = Publicationcontext()
   Proc = PD(Context)
4  Proc.start(["lcg-Ca.Repo", "lcg-CA", "lcg-CA", "http://linuxsoft.cern.
       ch/lcg-cas/current"])
```

Listing 3.6: Erzeugen und Starten einer Workflow-Instanz

Wie zuvor erwähnt erfolgt die Implementierung der beiden Beispiel-Workflows mittels der ZOPE 3 Workflow Management Coalition Engine, welche nachfolgend vorgestellt ist.

ZOPE 3 und die Workflow Management Coalition Engine Auf den Webseiten[7] der Workflow Management Coalition wird auf eine Vielzahl an unterschiedlichen XPDL-Implementierungen verwiesen. Die durchgeführte Evaluation nutzt eine frei verfügbare, Python-basierte Implementierung, die sich in das Z Object Publishing Environment (ZOPE) 3 integriert. Dies ist ein Applikationsserver, der durch Richter in [Ric05] als

> application and backend server framework that allows developers to quickly implement protocols, build applications (usually Web-based) and function as glue among other net-enabled services

beschrieben ist und u. a. Möglichkeiten zur Nutzerauthentifikation, zur Sitzungsverwaltung und zur automatischen Generierung und Validierung von Web-Seiten in der Hypertext Mark-Up Language (HTML) bietet.

[7] `http://www.wfmc.org/xpdl-implementations.html`

Intern besitzt ZOPE 3 einen modularen Aufbau und ermöglicht so die Einbindung von Erweiterungen (Products), über die beispielsweise Anbindungen an relationale Datenbanken realisiert sind. Zudem verfügt es mit der ZOPE Object Database (ZODB) bereits über eine persistente Datenbank zur Speicherung von Python-Objekten. Ähnlich zu Django[8] unterstützt ZOPE 3 zur Erzeugung generischer Vorlagen für HTML-Seiten Mark-Up-Sprachen, wie die Document Template Mark-Up Language (DTML) und die ZOPE Page Templates (ZPT).
Einer der Vorteile von ZOPE 3 ist die weitreichende Verfügbarkeit: fast jede Linux-Distribution enthält bereits vorkompilierte Binärpakete und selbst für Microsoft Windows-Betriebssysteme sind Binärpakete verfügbar. Weiterhin sind die Abhängigkeiten zu anderer Software gering, lediglich Python 2.4, der GNU Compiler und das Paket `zlib` werden vorausgesetzt.

Gegen Ende des Jahres 2004 wurde das erste Release von ZOPE 3 veröffentlicht. Die Evaluation nutzt Version 3.4.0, deren Installation beispielhaft in Listing 3.7 dargestellt ist.

```
# Installation der Abhängigkeiten
yum -y install python-devel gcc zlib

# Download von ZOPE 3.4.0
wget http://download.zope.org/zope3.4/3.4.0/Zope-3.4.0.tgz -P /tmp

# Kompilieren und Installieren von ZOPE 3.4.0
tar xzf /tmp/Zope-3.4.0.tgz -C /tmp
cd /tmp/Zope-3.4.0
./configure --prefix=/opt/Zope-3.4.0
make
make check
make install
```

Listing 3.7: Installation von ZOPE 3 auf Scientific Linux 5

In Zeile 2 werden durch die Installation der Pakete `python-devel`, `gcc` und `zlib` die Software-Abhängigkeiten aufgelöst. Anschließend erfolgt das Herunterladen und Entpacken des ZOPE 3.4.0 Software-Archivs in den Zeilen 5 und 8. In Zeile 10 wird über den Aufruf des `configure`-Skripts mittels der `-prefix`-Option das Zielverzeichnis für die ZOPE 3-Binärpakete festgelegt, bevor in den Zeilen 11 bis 13 die Kompilation und Installation durch den Aufruf von drei `make`-Befehlen erfolgt.

Im Anschluss an die ZOPE 3-Installation wird die Workflow Engine, die im separat beziehbaren Paket `zope.wfmc`[9] enthalten ist, installiert. Mit ihr können in Python oder

[8] `http://www.djangoproject.com/`
[9] `http://pypi.python.org/pypi/zope.wfmc/3.5.0`

der XML Process Definition Language beschriebene Workflows ausgeführt werden. Intern verwendet das Paket hierfür eine Untermenge der in Abbildung 3.17 dargestellten Komponenten des XPDL-Metamodells.

Die Installation des `zope.wfmc`-Pakets setzt die Existenz des Python-Pakets `setuptools` sowie des GNU C-Compilers voraus und ist in Listing 3.8 dargestellt. Durch das Kommando in Zeile 1 wird lediglich das Paket `python-setuptools` installiert, da der GNU C-Compiler bereits bei der ZOPE 3-Installation (vgl. Listing 3.7) auf dem System eingerichtet wurde. In Zeile 2 überprüft das Werkzeug `easy_install` die Aktualität der eben installierten `python-setuptools` und führt gegebenenfalls ein Upgrade unter Verwendung eines zentralen Python-Repositories durch. Durch die Zeilen 3 und 4 wird das Software-Archiv `zope.wfmc-3.5.0.tar.gz` heruntergeladen und in das lokale Verzeichnis `/tmp/` entpackt. Die eigentliche Installation des Pakets erfolgt in den Zeilen 6 und 7 durch den Aufruf von `setup.py` mit den Optionen `build` bzw. `install`.

```
1 yum -y install python-setuptools.noarch
  easy_install -U setuptools
3 wget http://pypi.python.org/packages/source/z/zope.wfmc/zope.wfmc-3.5.0.
      tar.gz -P /tmp
  tar xzf /tmp/zope.wfmc-3.5.0.tar.gz -C /tmp
5 cd /tmp/zope.wfmc-3.5.0
  python setup.py build
7 python setup.py install
```

Listing 3.8: Installation des Pakets `zope.wfmc`

Während der Evaluationsphase der XML Process Definition Language werden verschiedene Workflows entworfen und (teil-)implementiert. Diese Workflows decken unter anderem die Installation eines Ganglia Servers, eines Nagios Servers, eines TORQUE Servers und auch einer gLite VOBox ab.

Im Laufe der Applikationsimplementierung für die einzelnen Workflows zeigte sich die Notwendigkeit für ein strukturiertes Vorgehen bei der Ablage der Applikationen enthaltenden Dateien im Dateisystem. Da die Installations- und Konfigurationsschritte bei den zuvor erwähnten Diensten in Abhängigkeit vom verwendeten Betriebssystem variieren können, bildet das Kriterium „Betriebssystem“ in der hierarchisch angelegten Ablagestruktur (vgl. Abbildung 3.22) die oberste Ebene. Der Punkt `operating system` gliedert sich in weitere Unterkategorien auf, die die Kürzel der entsprechenden Linux-Distributionen tragen, etwa `sl` für Scientific Linux und `sles` für SUSE Linux Enterprise Server. Für jede der Unterkategorien existiert eine weitere Unterteilung bzgl. der Versionsnummer und ebenso der Architektur (32 Bit bzw. 64 Bit). Unterhalb der Architekturzweige sind die Applikationen für die Installation bzw. Konfiguration nach Anwendung getrennt in Unterverzeichnissen

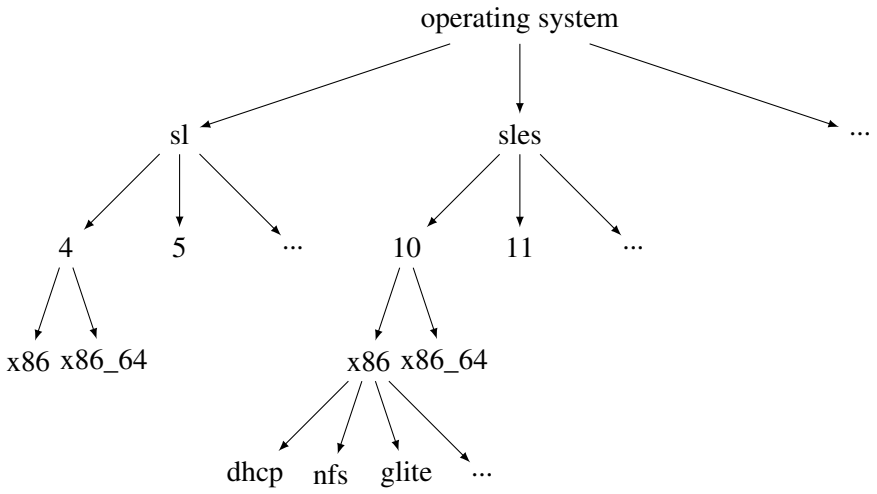


Abbildung 3.22: Strukturierte Ablage der XPDL-Applikationen im Dateisystem

hinterlegt. Die Applikationen für die Konfiguration von Betriebssystemdiensten wie NFS oder DHCP können hier ebenso abgelegt werden.

3.2.4.2 Business Process Modeling Notation

Analog zu den in Abschnitt 3.2.4.1 beschriebenen XPDL-Workflows ist in diesem Abschnitt je ein BPMN-Workflow für i) das Löschen von Einträgen aus der Liste der beim Systemstart geladenen Dienste und ii) für die Installation des `lcg-CA`-Pakets vorgestellt.

Das Entfernen eines Dienstes ist hier in die zwei Aktivitäten `Exists Service?` und `Disable Service` zerlegt (vgl. Abbildung 3.23). Als Eingabe erhält der Workflow den Namen des Dienstes in Form eines Strings und prüft im ersten Schritt `Exists Service?`, ob dieser Dienst im System existiert. Tritt während des Schritts ein Fehler auf, so wird der Workflow an dieser Stelle über das Durchlaufen der Aktivität `Error Handling` abgebrochen und verlassen. Andernfalls erfolgt ein Übergang in die Aktivität `Disable Service`, in welcher die eigentliche Entfernung des Dienstes erfolgt.

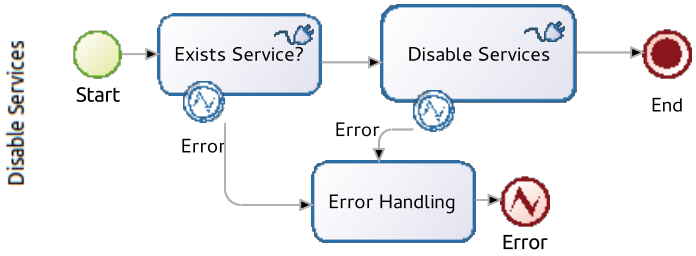


Abbildung 3.23: BPMN-Workflow zur Deaktivierung eines Dienstes

Die Implementierung der zu den Aktivitäten gehörenden Applikationen erfolgt unter

Verwendung der Programmiersprache Groovy. Das Listing 3.9 zeigt die für das Entfernen des Dienstes `service` ausgeführten Groovy-Befehle. In Zeile 1 wird in dem String `command` das auf Betriebssystemebene auszuführende Kommando zusammengesetzt. Mittels des `chkconfig`-Befehls und dem Argument `off` lässt sich der automatische Dienststart beim Hochfahren des Systems unterbinden. Die Ausführung des Kommandos erfolgt in Zeile 2, wobei die `execute()`-Methode ein Prozess-Objekt zurückliefert. Der Aufruf von `waitFor()` in Zeile 3 unterbricht die Ausführung der Groovy-Befehle solange, bis der Prozess abgeschlossen ist. Analog erfolgt in den Zeilen 5 bis 7 das Anhalten des Dienstes für die aktuelle Sitzung mittels des `service`-Befehls.

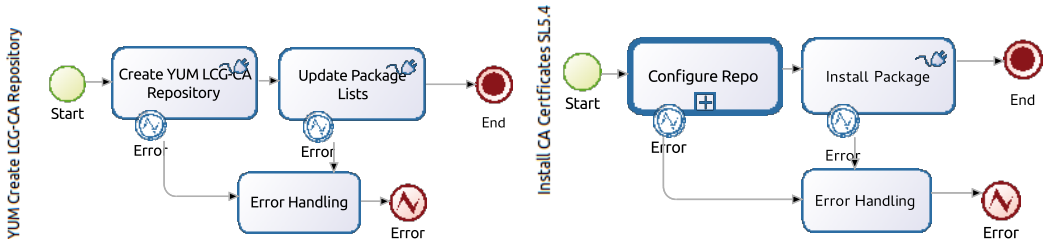
```
1 def String command = "sudo chkconfig " + ${service} + " off"
  def Process proc = command.execute()
3 proc.waitFor()

5 command = "sudo service " + ${service} + " stop"
  proc = command.execute()
7 proc.waitFor()
```

Listing 3.9: Abschalten eines Dienstes (Groovy-Applikation)

Die Verknüpfung zwischen den einzelnen Workflow-Aktivitäten und den Applikationen erfolgt direkt in der verwendeten Software Bonita Studio[10] durch Konnektoren, wobei eine Aktivität mehrere Konnektoren besitzen kann.

Der Zweck des nun vorgestellten Workflows ist die Installation des `lcg-CA`-Pakets. Im Vergleich zum vorherigen Workflow setzt dieser einen Subflow für die Konfiguration der notwendigen Repository-Informationen ein. Das Hinterlegen der Repository-Informa-



(a) Konfiguration eines YUM Repositories (b) Installation des Pakets lcg-CA

Abbildung 3.24: Zwei Beispiele für BPMN-Workflows

tionen (vgl. Abbildung 3.24(a)) erfordert beim exemplarisch verwendeten Betriebssystem Scientific Linux das Anlegen einer Datei im Verzeichnis `/etc/yum.repos.d/`. Das Listing 3.10 zeigt den Groovy-Code für die zugehörige Aktivität `Create YUM LCG-CA Repository`. In Zeile 2 wird mit den erforderlichen Informationen das String-Feld

[10] `http://www.bonitasoft.com/overview/bonita-studio`

`lines` erstellt, dessen Elemente in Zeile 3 in die Datei `/etc/yum.repos.d/lcg-CA.repo` geschrieben werden.

```
1 def File outputFile = new File ("/etc/yum.repos.d/lcg-CA.repo")
  def lines = [ '[lcg-CA]', 'name=lcg-ca', 'baseurl=http://linuxsoft.cern.
    ch/LCG-CAs/current', 'enabled=1', 'protect=1' ]
3 outputFile.write(lines[0..4].join("\n"))
```

Listing 3.10: Hinterlegen der Repository-Information (Groovy-Applikation)

Im Anschluss an die Erzeugung dieser Datei erfolgt die Aktualisierung der Liste der über YUM verfügbaren Software-Pakete in der Aktivität `Update Package Lists` (vgl. Listing 3.11).

```
1 def String command = "sudo yum makecache"
  def Process proc = command.execute()
3 proc.waitFor()
```

Listing 3.11: Aktualisierung der verfügbaren YUM-Paketliste (Groovy-Applikation)

Für die Installation des `lcg-CA`-Pakets ist die in Abbildung 3.24(b) enthaltene Aktivität `Install Package` zuständig. Die ihr hinterlegte Applikation nutzt die Paketverwaltung YUM des Betriebssystems (vgl. Listing 3.12), jedoch wird vor deren Aufruf über die Ausführung des Subflows `Configure Repo` die notwendige Repository-Information wie zuvor beschrieben im System hinterlegt.

```
1 def String command = "sudo yum -y install lcg-CA"
  def Process proc = command.execute()
3 proc.waitFor()
```

Listing 3.12: Installation des Pakets `lcg-CA` (Groovy-Applikation)

Jede der in den Abbildungen gezeigten Workflow-Aktivitäten (bis auf die `ErrorHandling`-Aktivitäten) verfügt über ein durch einen Kreis mit innenliegendem Blitz symbolisiertes Catch Error Event. Über dieses werden die Aktivitäten im Fehlerfall verlassen. Die normale Abarbeitung des Workflows wird entsprechend beendet und die jeweilige Aktivität `ErrorHandling` gestartet. Den Endpunkt einer fehlerhaften Abarbeitung eines Workflows bildet in allen Fällen letztendlich das Error End Event.

Im Gegensatz zu den XPDL-Beispielen ist der Groovy-Code der einzelnen Applikationen nicht im Dateisystem abgelegt, sondern wird intern, in einem Workspace, von der hier verwendeten Software Bonita Studio gespeichert. Die Software bietet jedoch auch eine Funktion, über die sich bei Bedarf einzelne Workflows inklusive Applikationen als komprimiertes Software-Archiv aus diesem Workspace exportieren lassen. Weitere Details dazu sind in Abschnitt 4.9 vorgestellt.

3.2.5 Zusammenfassung

Neben den Ereignisgesteuerten Prozessketten sind mit der XML Process Definition Language und der Business Process Modeling Notation insgesamt drei Möglichkeiten zur graphischen Modellierung von Prozessen bzw. Workflows vorgestellt. Eine Untersuchung dieser drei Optionen auf die Unterstützung einer automatisierten Ausführung reduzierte die Auswahl um die Ereignisgesteuerten Prozessketten. Für die beiden übriggebliebenen Kandidaten sind exemplarisch zwei Workflows umgesetzt, wobei der eine die Installation eines Software-Pakets vornimmt und der andere das Abschalten nicht-benötigter Systemdienste durchführt.

Während der Implementierung der Workflow-Applikationen hat sich bei der gewählten Python-basierten XPDL-Umsetzung gezeigt, dass diese keine Subflows unterstützt. Gerade in Bezug auf die Aspekte der Modularität und der Wiederverwendbarkeit stellt dies einen immensen Nachteil dar. Ob sich dieser Nachteil auch auf andere Implementierungen der XML Process Definition Language erstreckt, wurde nicht weiter untersucht. Zudem kann die XPDL in ihrer aktuellen Version zwar als ausgereift angesehen werden und findet in Produkten, wie beispielsweise UNICORE Verwendung, aber sie ist oftmals proprietär erweitert, so dass sich zwangsläufig Probleme beim Workflow-Austausch zwischen verschiedenen XPDL-Implementierungen ergeben. Insgesamt spricht dies gegen eine Verwendung der XML Process Definition Language im Rahmen dieser Arbeit.

Die Business Process Modeling Notation gehört zu den Standards für die graphische Modellierung von Workflows und hat eine weite Verbreitung im unternehmerischen Umfeld erreicht. Im direkten Vergleich mit den in Abschnitt 3.2.1 beschriebenen Ereignisgesteuerten Prozessketten spricht dieser Punkt zusammen mit dem gut ausgebauten Ecosystem und den während der Evaluationsphase gesammelten Erfahrungen für eine Modellierung der Workflows mittels der Business Process Modeling Notation.

Kapitel 4

Basisdienste

Nachdem in Kapitel 2 die existierenden Lösungen für die Installation und Konfiguration von Betriebssystemen und Software betrachtet wurden, wird in diesem Kapitel ein selbst entwickeltes Verfahren vorgestellt, welches Workflows nutzt. Da die Beschreibung von Workflows auf unterschiedliche Arten und Weisen erfolgt, sind drei erfolgversprechende Kandidaten in Abschnitt 3.2 evaluiert. Aus dieser Evaluation geht die Business Process Modeling Notation als Sieger hervor, womit sie die Notation für alle nachfolgend vorgestellten Workflows festlegt.

Es sind vornehmlich Dienste aus dem Bereich des Grid Computing, genauer von einer Menge ausgewählter Grid Middlewares, für die in der vorliegenden Arbeit Workflows präsentiert sind. Für die im Detail in Abschnitt 5.3 vorgestellten Grid Middlewares lässt sich feststellen, dass sie fast ausschließlich in akademischen Einrichtungen, wie Universitäten und Forschungszentren, betrieben werden. Dort erfolgt der Betrieb üblicherweise entweder in einem Rechenzentrum oder innerhalb eines Fachbereichs bzw. Lehrstuhls. Unabhängig hiervon benötigt man eine gewisse Menge an begleitenden Diensten, wie beispielsweise einen Nagios Server für die Systemüberwachung. Unter dem Begriff der Basisdienste ist genau diese Menge an Diensten zu verstehen.
So sind in Abschnitt 4.3 etwa Workflows für den Basisdienst eines Network File System (NFS) Servers sowie dessen Clients im Detail vorgestellt. Die für Dynamic Host Configuration Protocol (DHCP) bzw. Lightweight Directory Access Protocol (LDAP) Server relevanten Workflows sind in den Abschnitten 4.4 und 4.5 vorgestellt. Es sind in Abschnitt 4.6 Workflows für die Installation eines Nagios Servers als auch des Nagios Clients skizziert. Des Weiteren sind in Abschnitt 4.7 Workflows für Ganglia, eine weitere Überwachungssoftware, und in Abschnitt 4.8 Workflows für die Installation und Konfiguration des TORQUE Batchsystems vorgestellt. Auf letzterem erfolgt die Ausführung der über die Grid Middleware oder auch lokal eingereichten Jobs.

Um den Entwicklungen bei optimierten Ressourcennutzung Rechnung zu tragen, ist die Workflow-gestützte Installation der Dienste in virtuelle Maschinen vorgesehen. Die Grundlagen hierfür, insbesondere die Plattform-Virtualisierung, sind in Abschnitt 3.1 beschrieben.

4.1 Bereitstellung einer virtuellen Maschine

Aufgrund der in Abschnitt 3.1.4 vorgestellten Vorteile der Plattform-Virtualisierung gehen viele Ressourcenanbieter und auch Nutzer dazu über, virtuelle Maschinen bzw. Appliances einzusetzen. Die Bereitstellung einer virtuellen Maschine umfasst neben der Spezifikation der ihr zugewiesenen physischen Hardware (z. B. Anzahl der Kerne der Central Processing Unit (CPU) und Hauptspeicher) in fast allen Fällen auch die Einbindung virtueller Festplatten. Eine solche Festplatte liegt als Abbilddatei vor und enthält meist ein bereits vorkonfiguriertes Betriebssystem oder wird dazu genutzt, dieses zu installieren.

Wie in Abschnitt 2.1, beschrieben gibt es für die Installation eines Betriebssystems unterschiedliche Verfahren. In Abbildung 4.1 ist für Scientific Linux 5.4 ein Workflow gezeigt, der das Kickstart-Verfahren nutzt. Initial wird dabei in der Aktivität `Create Disk Image` eine virtuelle Festplatte erzeugt, deren Speicherort im Dateisystem und Größe in GByte vom Nutzer spezifizierbar sind. Die Erzeugung läuft in der der Aktivität hinterlegten Applikation unter Verwendung des `qemu-img`-Kommandos (vgl. Listing 4.1) ab.

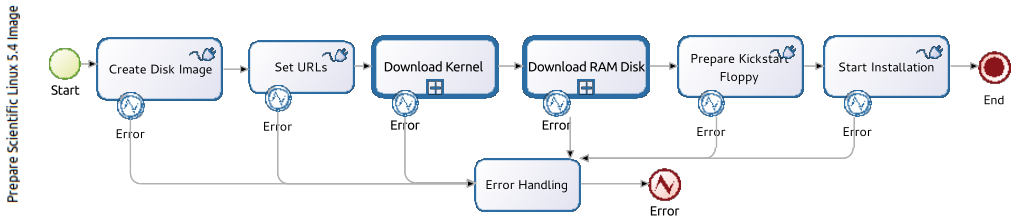


Abbildung 4.1: Vorbereitung einer Scientific Linux 5.4-basierten virtuellen Maschine

```
1 def String command = "qemu-img create ${filename} ${disk_Image_Size}G"
  def Process proc = command.execute()
3 proc.waitFor()
```

Listing 4.1: Erzeugung einer virtuellen Festplatte (Groovy-Applikation)

Die später dem Nutzer präsentierte Web-Oberfläche zur Angabe der Workflow-Argumente ist in Abbildung 4.2 gezeigt. Wie zuvor erwähnt, sind die Größe der Abbilddatei (Standardwert: 8 GByte), der Speicherort im Dateisystem (Standardwert: leer) sowie zusätzlich die zu verwendende Kickstart-Datei (Standardwert: `/tmp/ks.cfg`) und die Architektur (Standardwert: `i386`) einstellbar.

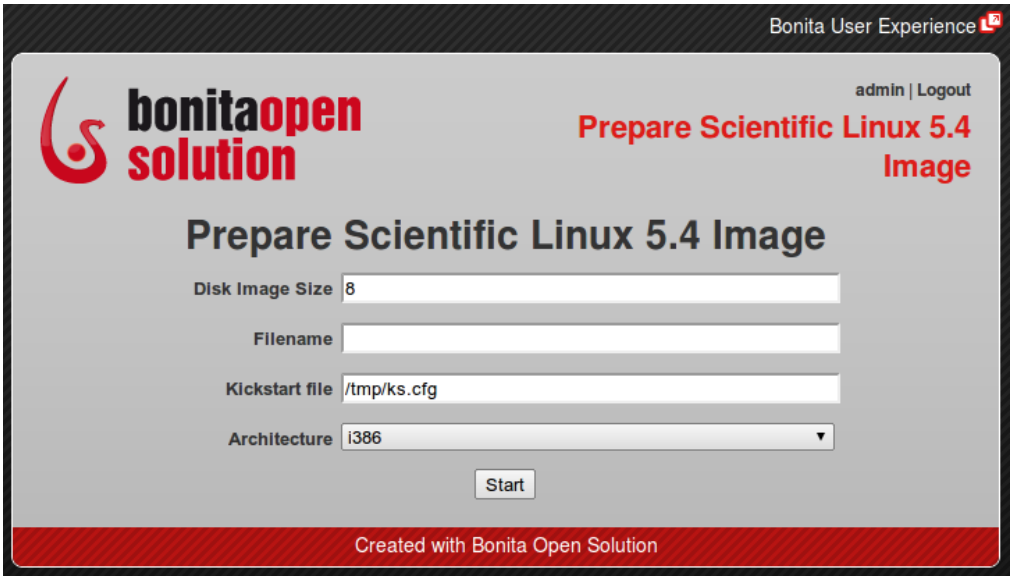


Abbildung 4.2: Vorbereitung einer Scientific Linux 5.4-basierten virtuellen Maschine (Web-Oberfläche)

Der für den Start der Installationsroutine notwendige Linux-Kernel und die passende Random Access Memory (RAM) Disk werden in den Aktivitäten `Download Kernel` und `Download RAM Disk` unter Verwendung von Subflows in das Verzeichnis `/tmp/` heruntergeladen.

Ist der Nutzer im Besitz einer Kickstart-Datei, die er zum Bespielen der virtuellen Festplatte verwenden möchte, so kann er dies tun. Das Listing 4.2 zeigt, wie die Kickstart-Datei des Nutzers in der Aktivität `Prepare Kickstart Floppy` des Workflows in ein Floppydisk-Abbild kopiert wird.

```
1  def String command = "dd if=/dev/zero of=/tmp/ks-floppy.img bs=1440K
       count=1"
   def Process proc = command.execute()
3  proc.waitFor()

5  command =  "/sbin/mkfs -F -t ext2 /tmp/ks-floppy.img"
   proc = command.execute()
7  proc.waitFor()

9  command = "mkdir -p /tmp/ks-floppy/"
   proc = command.execute()
11 proc.waitFor()

13 command = "sudo mount -o loop /tmp/ks-floppy.img /tmp/ks-floppy"
   proc = command.execute()
```

```
15 proc.waitFor()

17 command = "cp -p ${path_to_kickstart} /tmp/ks-floppy/ks.cfg"
   proc = command.execute()
19 proc.waitFor()

21 command = "sudo umount /tmp/ks-floppy/"
   proc = command.execute()
23 proc.waitFor()
```

Listing 4.2: Erstellung eines Kickstart Disk-Abbilds (Groovy-Applikation)

In den Zeilen 1 bis 3 wird analog zu einer Abbilddatei für eine Festplatte eine Abbilddatei für eine Floppydisk angelegt. Nach dem Bespielen der Datei mit einem `ext2`-Dateisystem in den Zeilen 5 bis 7 wird die vom Nutzer spezifizierte Kickstart-Datei in die Abbilddatei kopiert. Hierzu sind u. a. das Mounten und Unmounten der Abbilddatei notwendig, welches in den Zeilen 13 bis 15 bzw. 21 bis 23 erfolgt.

Das Floppydisk-Abbild wird in der hinter der Aktivität `Start Installation` liegenden Applikation mit weiteren Argumenten (z. B. der Hauptspeichergröße, dem Pfad zum Linux-Kernel und der RAM Disk sowie dem Pfad zum Festplattenabbild) an den Quick Emulator (QEMU) übergeben (vgl. Listing 4.3). QEMU selbst leitet daraufhin mit diesen Informationen den Start einer virtuellen Maschine ein.

```
1 def String command = "qemu -hda ${filename} -fda /tmp/ks-floppy.img -m
      512 -localtime -kernel /tmp/vmlinuz -initrd /tmp/initrd.img -append
      ks=floppy"
  def Process proc = command.execute()
3 proc.waitFor()
```

Listing 4.3: Start einer virtuellen Maschine durch QEMU (Groovy-Applikation)

Der vorgestellte Workflow lässt sich ohne Aufwand um die Unterstützung anderer, aktueller Versionen von Scientific Linux erweitern, in dem der verwendete Linux-Kernel und die RAM Disk austauschbar gemacht werden.

Nach der erfolgreichen Installation des Basisbetriebssystems kann über die Paketverwaltung weitere Software installiert werden. Nachfolgend sind hierzu entworfene Workflows vorgestellt.

4.2 Paketmanager

Die Installation von Software auf einem System kann auf unterschiedlichste Weisen erfolgen, wobei das Herunterladen und Kompilieren von Quellcode sowie das Einspielen bereits vorkonfigurierter Binärpakete zwei dieser Möglichkeiten darstellen. In letzterem Fall greift man zumeist auf einen Paketmanager zurück, der sich z. B. um das Auflösen von Abhängigkeiten zwischen einzelnen Software-Paketen kümmert und mindestens die Operationen des Installierens und Entfernens einzelner Pakete unterstützt.

Im Folgenden sind für die beiden Paketmanager Yellowdog Updater Modified (YUM) und Yet another Setup Tool (YaST) jeweils Workflows für das Hinterlegen von Repository-Informationen in einem System sowie für das Hinzufügen und Entfernen von Software beschrieben.

YUM YUM[1] findet bei vielen Red Hat-basierten Linux-Distributionen Verwendung. Die einzelnen Pakete liegen in entfernt lokalisierten Repositories vor und sind bei Bedarf unter Angabe des jeweiligen Paketnamens installierbar. Existieren zwischen einzelnen Paketen Abhängigkeiten, so ist YUM in der Lage, diese eigenständig zu erkennen und aufzulösen.

Üblicherweise sind die Informationen zu den verwendeten Software-Repositories in separaten Dateien im Verzeichnis `/etc/yum.repos.d/` abgelegt und unterscheiden sich durch folgende Merkmale

- Name der Konfigurationsdatei
  Von YUM werden nur solche Dateien im Verzeichnis `/etc/yum.repos.d/` berücksichtigt, die auf `.repo` enden.
- Logischer Bezeichner des Repositories
  Der Bezeichner ist – wie alle weiteren Punkte auch – in der Konfigurationsdatei aufgeführt und wird von eckigen Klammern umschlossen. Des Weiteren darf er nur aus zusammenhängendem Text bestehen und muss unter allen in `/etc/yum.repos.d/` vorliegenden logischen Bezeichnern eindeutig sein.
- Adresse des Repositories (`baseurl`)
  Die `baseurl` besteht aus einem oder einer Menge von Verweisen auf Verzeichnisse, in denen sich die `repodata`-Datei des Software-Repositories befindet. YUM bietet die Möglichkeit, lokale Repositories (per `file://`) und auch entfernte Repositories (z. B. per `http://` oder `ftp://`) einzubinden.
- Textuelle Beschreibung
  Neben dem logischen Bezeichner ist auch die Angabe einer Menschen-lesbaren

[1] `http://yum.baseurl.org/`

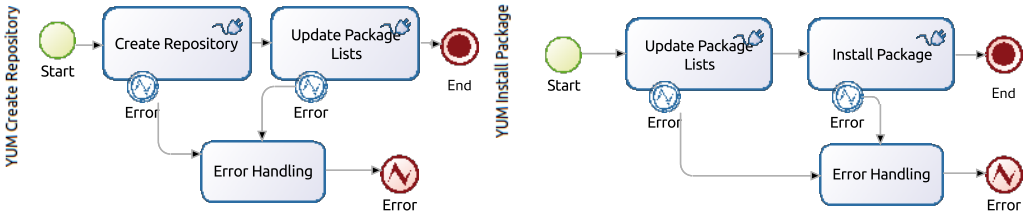
Beschreibung des Repositories möglich. Diese wird innerhalb der Konfigurationsdatei in der Variablen `name` gespeichert.

Ein Beispiel für den Inhalt einer Repository-Konfigurationsdatei zeigt Listing 4.4. In diesem Listing ist ebenso die Verwendung der Variable `enabled` verdeutlicht. Der Wert dieser Variable (`0` = inaktiv/ `1` = aktiv) legt fest, ob das Repository von YUM berücksichtigt werden soll. Analog ist die Durchführung einer Gültigkeitsüberprüfung der GNU Privacy Guard (GPG)-Signatur des Software-Repositories über die Variable `gpgcheck` aktivierbar. Die Überprüfung dient dem Aufbau einer Vertrauensbeziehung zum Repository-Anbieter und sollte üblicherweise vor der Installation von Paketen aus dieser Quelle stattfinden.

```
1 [sl-base]
name=Scientific Linux 5.4
3 baseurl=http://ftp.scientificlinux.org/linux/scientific/54/$basearch
    /SL
enabled=1
5 gpgcheck=0
```

Listing 4.4: Standard-Informationen zu einem YUM Repository

Der für YUM implementierte Workflow zur Hinterlegung der Repository-Informationen im Verzeichnis `/etc/yum.repos.d/` besteht, wie in Abbildung 4.3(a) dargestellt, aus den zwei Aktivitäten `Create Repository` und `Update Package Lists` sowie der Fehlerbehandlung.



(a) Hinzufügen eines Repositories (b) Software-Installation aus einem Repository

Abbildung 4.3: Zwei Workflows für den YUM Paketmanager

In der ersten Aktivität gibt der Anwender über eine Web-Oberfläche (vgl. Abbildung 4.4) die notwendigen Informationen ein. Nach Auswertung dieser wird über die Ausführung der in Listing 4.5 gezeigten Applikation die entsprechende Datei im Verzeichnis `/etc/yum.repos.d/` erzeugt. Hierbei werden die in `${ ...}` angegeben Variablen im Listing durch die Nutzereingaben ersetzt.

```
def File outputFile = new File ("${file_Name}")
def lines = [ '[${repository_Name}]', 'name=${repository_Description
    }', 'baseurl=${base_URL}', 'enabled=1', 'protect=1', 'gpgcheck=0
    ' ]
outputFile.write(lines[0..5].join("\n"))
```

Listing 4.5: Hinterlegen der YUM Repository-Informationen (Groovy-Applikation)

Die zweite Aktivität namens `Update Package Lists` aktualisiert den lokal im System vorliegenden Paket-Cache.

Für die Installation von Software aus einem Repository ist ein zweiter Workflow (vgl. Abbildung 4.3(b)) nutzbar. Bevor die Installation der Software in der Aktivität `Install Package` jedoch erfolgt, findet in der Aktivität `Update Package Lists` zunächst eine Aktualisierung der Informationen über alle verfügbaren Pakete statt. Das Listing 4.6 zeigt die bei der Installation ausgeführte Groovy-Applikation, deren Parameter dem Namen des zu installieren Pakets entspricht. Bei mehreren zu installierenden Paketen sind diese untereinander durch Leerzeichen zu trennen.

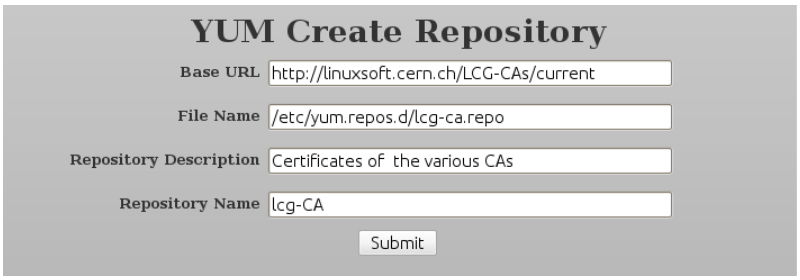


Abbildung 4.4: Web-Oberfläche zum Workflow YUM Create Repository

```
def String command = "sudo yum -y install ${yum_Package}"
def Process proc = command.execute()
proc.waitFor()
```

Listing 4.6: Installation von Software aus einem Repository (Groovy-Applikation)

Analog zu dem hier vorgestellten Vorgehen sind Workflows für die Installation ganzer Paketgruppen oder auch lokal vorliegender RPM-Pakete denkbar.

YaST Das Hinterlegen von Informationen zu einzelnen Repositories erfolgt bei dem Paketmanager YaST u. a. mittels des Werkzeugs `zypper`[2]. Dieses Werkzeug ermöglicht beispielsweise das Hinzufügen eines Repositories durch die Angabe der Option

[2] `http://en.opensuse.org/Portal:Zypper`

`add repository` (kurz: `ar`) in Verbindung mit der Repository-URL und einem beschreibenden Namen für das Repository. So ist nach der erfolgreichen Ausführung des Befehls in Listing 4.7 das hinter der URL liegende Repository im System unter dem Namen `vlc_repo` ansprechbar.

```
$ zypper ar http://download.videolan.org/pub/vlc/SuSE/11.1 vlc_repo

Adding repository 'vlc_repo' [done]
Repository 'vlc_repo' successfully added
Enabled: Yes
Autorefresh: No
URI: http://download.videolan.org/pub/vlc/SuSE/11.1
```

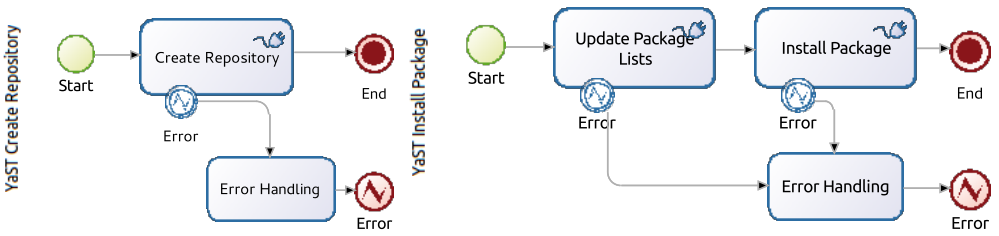
Listing 4.7: Hinzufügen eines YaST Repositories mittels `zypper`

Ein Workflow, der die in Listing 4.7 gezeigte Aktion kapselt, ist in Abbildung 4.5(a) dargestellt. Neben der Aktivität `Create Repository`, deren zugehörige Applikation in Listing 4.8 angegeben ist, gibt es mit `Error Handling` nur noch eine einzige weitere Aktivität innerhalb dieses Workflows.

```
def String command = "sudo zypper ar ${repositoryURL} ${name}"
def Process proc = command.execute()
proc.waitFor()
```

Listing 4.8: Hinzufügen eines YaST Repositories (Groovy-Applikation)

Der Workflow für die Software-Installation heraus aus einem YaST-Repository (vgl. Abbildung 4.5(b)) ist fast identisch mit dem für YUM präsentierten Workflow. Es unterscheiden sich lediglich die hinter den Aktivitäten `Update Package Lists` und `Install Package` liegenden Applikationen.



(a) Hinzufügen eines Repositories (b) Software-Installation aus einem Repository

Abbildung 4.5: Zwei Workflows für den YaST Paketmanager

Neben einer graphischen Oberfläche verfügt YaST ebenfalls über einen Kommandozeilen-Client `yast`, welcher hier für die Installation der Software zum Einsatz

kommt. In Verbindung mit der Option `-i` sowie der Liste der zu installierenden Pakete findet er in der in Listing 4.9 dargestellten Groovy-Applikation Verwendung.

```
def String command = "sudo yast -i ${yast_Package}"
def Process proc = command.execute()
proc.waitFor()
```

Listing 4.9: Installation von Software via YaST (Groovy-Applikation)

4.3 Network File System Service

Die Hauptaufgabe des Network File Systems ist die Schaffung eines Zugangs zu zentral auf einem Server gelagerten Verzeichnissen und Dateien für andere Rechner (Klienten). Dies ist beispielsweise dann von Interesse, wenn man auf mehreren lokalen oder auch entfernten Klienten auf dieselben Informationen zugreifen möchte. Statt diese auf jedem der Klienten lokal vorzuhalten und Mechanismen für eine etwaige Synchronisation der Informationen zwischen der Klienten zu konzipieren, sind die Informationen beim Network File System an einer einzigen Stelle, auf welche die Klienten zugreifen können, vorgehalten.

Die auf dem Server für Klienten zugreifbaren Verzeichnisse sind als Exports bezeichnet und zusammen mit Autorisierungsinformationen[3] in der Datei `/etc/exports` gespeichert. Eine Export-Definition selbst erstreckt sich über eine Zeile und besitzt einen Aufbau analog zur Darstellung in Listing 4.10.

```
Verzeichnis  Host_1(Option[, ...]) Host_2(Option[, ...])
```

Listing 4.10: Schematischer Aufbau einer Export-Definition in der Datei `/etc/exports`

In dem Listing beschreibt `Verzeichnis` das Verzeichnis auf dem Server, welches anderen Rechnern zugänglich gemacht werden soll. `Host_1` und `Host_2` entsprechen Klienten, denen der Zugriff auf das Verzeichnis erlaubt sein soll. Diese Klienten sind entweder durch deren IP-Adresse oder den Fully Qualified Hostname (FQHN) zu spezifizieren.
Die am häufigsten verwendeten Optionen in diesem Zusammenhang sind `ro` (Nur-Lese-Freigabe), `rw` (Lese- und Schreib-Freigabe) und `no_root_squash` (der Client-seitige `root`-Nutzer erhält auf dem Server ebenso als `root`-Nutzer Zugriff auf die exportierten Dateien).

In dem in Listing 4.11 gezeigten Beispiel für eine gültige `exports`-Datei wird über die erste Zeile das Verzeichnis `home` des NFS Servers den Rechnern mit den IP-Adressen `192.168.0.1` und `192.168.0.2` lesend und schreibend verfügbar gemacht. Die zweite

[3] Eine Internet Protocol (IP)-basierte Autorisation regelt den Zugriff auf die Informationen.

Zeile erlaubt dem Rechner mit der IP-Adresse `192.168.0.1` das Verzeichnis `/usr/local` des NFS Servers als nur-lesend einzubinden.

```
/home          192.168.0.1(rw) 192.168.0.2(rw)
/usr/local     192.168.0.1/255.255.255.255(ro)
```

Listing 4.11: Beispielhafter Inhalt der Datei `/etc/exports`

Die wesentlichen Aktionen auf der Datei `/etc/exports` sind das Einfügen und Entfernen einzelner Exporte. In dem entwickelten Workflow (vgl. Abbildung 4.6) für das Einfügen eines neuen Exports wird zunächst in der Aktivität `Create Backup` eine Sicherheitskopie der aktuellen `exports`-Datei angelegt. Die Folgeaktivität `Add Line To Exports` fügt für den neuen Export eine weitere Zeile in die Datei `/etc/exports` ein. Im Anschluss daran erfolgt die Ausführung der Aktivität `Update Exported File Systems`. Der Inhalt der zu dieser Aktivität gehörenden Applikation ist in Listing 4.12 gezeigt. Im Wesentlichen wird hier der Befehl `exportfs` genutzt, um die Liste der Exports im Betriebssystem aufzufrischen.

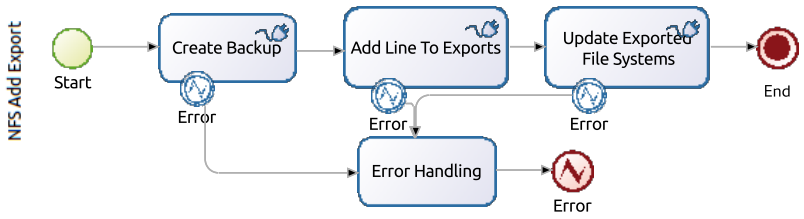


Abbildung 4.6: Workflow für das Hinzufügen eines Eintrags in die Datei `/etc/exports`

```
def String command = "sudo exportfs -avr"
def Process proc = command.execute()
proc.waitFor()
```

Listing 4.12: Aktualisierung der Dateisystem-Exporte (Groovy-Applikation)

Bevor allerdings Einträge in die `exports`-Datei geschrieben werden können, muss die NFS-Software auf dem Server eingespielt sein. Für Letzteres ist die erste Aktivität des Workflows zur Installation eines NFS Servers (vgl. Abbildung 4.7) verantwortlich. Sie ruft – statt einer Applikation – den Workflow zur Installation eines Software-Pakets über YUM (vgl. Abschnitt 4.2) auf und übergibt als Argument den erforderlichen Paketnamen `nfs-utils`. Die Festlegung, auf welchen Runlevels der NFS Server zukünftig automatisch starten soll, erfolgt in der Aktivität `Set Runlevels`. Abschließend übernimmt die Aktivität `Start Service` den unmittelbaren Start des NFS-Dienstes, wobei aufgrund der initial nicht-existierenden Einträge in der Datei `/etc/exports` keine Verzeichnisse exportiert werden.

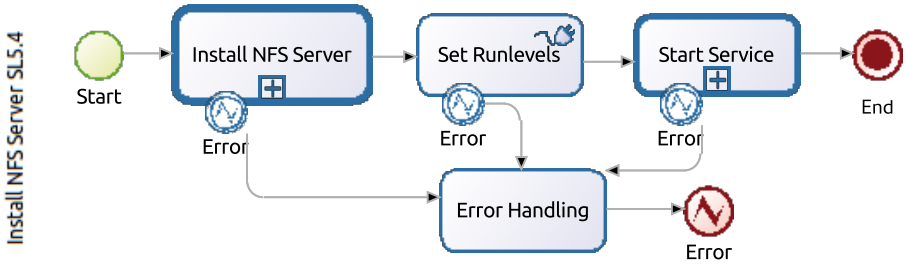
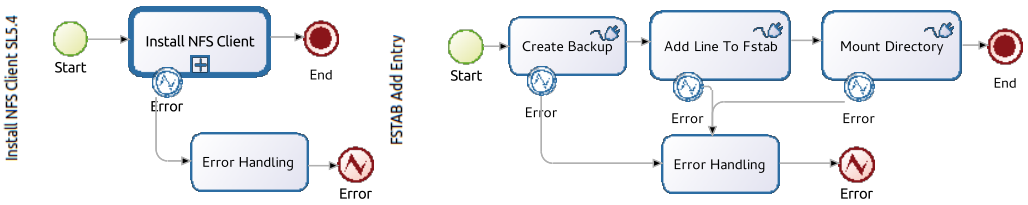


Abbildung 4.7: Workflow zur Installation eines NFS Servers auf SL 5.4

Network File System Client Auf der Client-Seite müssen analog zum Server ebenfalls Software-Pakete für NFS eingespielt werden. Bei dem hier als Beispiel verwendeten Betriebssystem Scientific Linux ist die Client-Software ebenfalls Bestandteil des Pakets `nfs-utils`, so dass ein Teil des in Abbildung 4.7 gezeigten Workflow wiederverwendbar ist. Der resultierende Workflow für den Client ist in Abbildung 4.8(a) dargestellt.



(a) Installation eines NFS Client (b) Hinzufügen eines Eintrags in die Datei `/etc/fstab`

Abbildung 4.8: Workflows für den NFS Client

Die Einbindung eines vom NFS Server exportierten Verzeichnisses erfolgt beim Client über einen Eintrag in die Datei `/etc/fstab` (vgl. Listing 4.13). Von außen erhält der Workflow (vgl. Abbildung 4.8(b)) hierfür Werte, die als Argumente für die Aktivität `Add Line to Fstab` dienen, beispielsweise `fs_spec` (NFS Server sowie einzubindendes Verzeichnis), `fs_file` (lokales Zielverzeichnis) und `fs_vfstype` (Dateisystemtyp, hier `nfs`). Vor dem Einfügen des neuen Eintrags wird zunächst in der Aktivität `Create Backup` eine Sicherheitskopie der Datei `/etc/fstab` angelegt. Als letzten Schritt hängt die Aktivität `Mount Directory` den neu hinzugefügten NFS-Eintrag in das Dateisystem ein.

```
1 def String output = "${fs_spec} ${fs_file} ${fs_vfstype} ${fs_mntops} ${
      fs_freq} ${fs_passno}"
  File f = new File ("/etc/fstab")
3 f.append(output)
```

Listing 4.13: Aktualisierung der Datei `/etc/fstab` (Groovy-Applikation)

4.4 Dynamic Host Configuration Protocol Service

Das Dynamic Host Configuration Protocol (DHCP) ermöglicht die Übertragung von Konfigurationsparametern an Netzwerkteilnehmer (Rechner, Drucker, ...) bei deren Start. Daher spielt DHCP in Verbindung mit weiteren Diensten beispielsweise bei der Netzwerkbasierten Rechnerinstallation eine wesentliche Rolle.
Der DHCP-Dienst selbst besteht aus drei Komponenten: dem DHCP Server, den DHCP Clients und dem DHCP-Protokoll, über welches die Informationen vom Server an die Klienten ausgeliefert werden.

Betriebssystemabhängig befindet sich die DHCP-Konfigurationsdatei an unterschiedlichen Stellen unterhalb des Verzeichnisses `/etc`. Bei den in den Abschnitten 2.1.2 und 2.1.3 vorgestellten Betriebssystemen Scientific Linux und SUSE Linux Enterprise Server liegt sie als Datei `/etc/dhcpd.conf` vor. In dieser Datei sind neben zu verwendenden Network Time Protocol (NTP) Servern und Gateways vor allem statische als auch dynamische Bindungen von Media Access Control (MAC)-Adressen an IP-Adressen spezifiziert. Bei der dynamischen Bindung erhält ein anfragender Rechner eine Adresse aus der Menge frei verfügbarer IP-Adressen, dem so genannten Pool. Im Gegensatz hierzu verknüpft die statische Bindung eine MAC-Adresse eindeutig mit einer IP-Adresse (vgl. Listing 4.14).

```
1 host <hostname> {
    hardware ethernet XX:XX:XX:XX:XX:XX; # MAC-Adresse
3   fixed-address YYY.YYY.YYY.YYY;       # IP-Adresse
  }
```

Listing 4.14: Kopplung von MAC- und IP-Adressen in der Datei `/etc/dhcpd.conf`

Für die Installation des DHCP Servers existiert ein analog zur Installation des Network File System Servers entworfener Workflow (vgl. Abbildung 4.9). Einzig die Aktivität `Install NFS Server` ist durch die Aktivität `Install DHCP Server` ausgetauscht. Ebenso lehnt sich der Abbildung 4.10 gezeigte Workflow für das Hinzufügen einer stati-

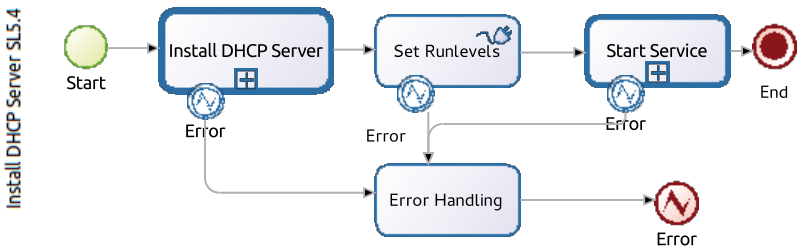


Abbildung 4.9: Workflow zur Installation eines DHCP Servers auf SL 5.4

schen Abbildung von MAC- auf IP-Adresse stark an den in Abbildung 4.6 visualisierten Workflow für das Hinzufügen einer Zeile zur Datei `/etc/fstab` an.

Diese Anlehnungen bzw. Analogien weisen bereits auf mögliche Verbesserungen bei der

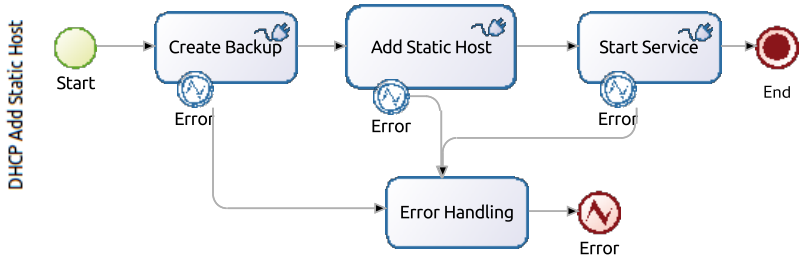


Abbildung 4.10: Workflow für das Hinzufügen eines Hosts in die DHCP-Konfiguration

Wiederverwendbarkeit einmal erstellter Workflows hin. Wie genau solche Verbesserungen aussehen können, ist in Abschnitt 6.3 beschrieben.

4.5 Lightweight Directory Access Protocol Service

Der LDAP-Dienst hat sich, neben z. B. dem Network Information Service (NIS)[4], als eine Möglichkeit der Nutzerverwaltung etabliert und besteht in seiner einfachsten Konfiguration aus einem zentralen Server nebst vielen Clients. In Szenarien mit mehr als einem LDAP-Server kann man für die automatische Datenreplikation zwischen den Servern auf Dienste wie `slurp` zurückgreifen.

Entstanden ist LDAP aus dem Wunsch heraus, einen leichtgewichtigen Verzeichnisdienst für den Zugriff auf entfernte, X.500-konforme Verzeichnisse zu implementieren. Im Vergleich mit seinem nicht-leichtgewichtigen Gegenstück, dem Directory Access Protocol (DAP), existieren bei dem Lightweight Directory Access Protocol einige Vereinfachungen, beispielsweise die Repräsentation viele Attributwerte durch einfache Zeichenfolgen. Des Weiteren verfügt LDAP über eine einfachere interne Protokollstruktur und als Folge der beobachteten Nutzungsmuster eine Optimierung für schnelle Lesezugriffe.

Viele Linux Distributionen, so auch Scientific Linux und SUSE Linux Enterprise Server, enthalten bereits vorgefertigte Binärpakete für den LDAP Server und den LDAP Client. Der für die Installation der Server-Komponenten eingesetzte Workflow (vgl. Abbildung 4.11) nutzt diese Tatsache und ähnelt daher den bereits zuvor beschrieben Workflows zur Installation eines NFS bzw. DHCP Servers.

Analog erfolgt auch die Installation des LDAP Client durch den in Abbildung 4.12(a) gezeigten Workflow. Die Aktivität `Install Client` ruft hierbei den Subflow `YUM In-`

[4]NIS war früher unter Yellow Pages (YP) bekannt. Das Akronym YP ist aber in Großbritannien ein eingetragenes Warenzeichen der British Telecom Public Limited Company (PLC), worauf hin YP in NIS umbenannt wurde.

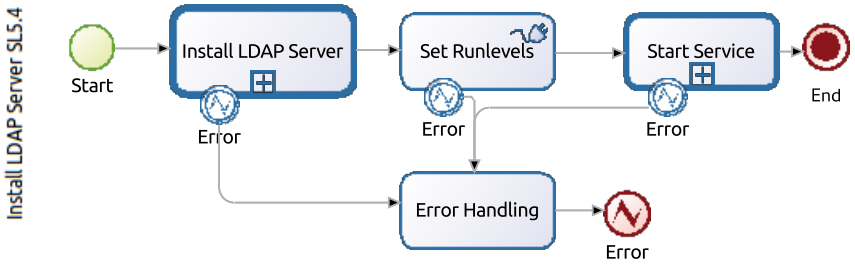
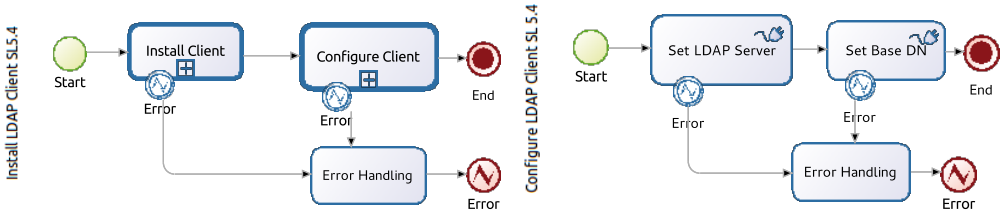


Abbildung 4.11: Workflows zur Installation eines LDAP Server auf SL 5.4

`stall Package` (vgl. Abbildung 4.3(b)) zur Installation der notwendigen Software-Pakete auf. Im Anschluss daran übernimmt der über die Aktivität `Configure Client` eingebundene Subflow (vgl. Abbildung 4.12(b)) die Konfiguration des Client.



(a) Installation (b) Konfiguration

Abbildung 4.12: Workflows für den LDAP Client

Innerhalb dieses Subflows wird durch die Aktivität `Set LDAP Server` bzw. die dahinter liegende Applikation (vgl. Listing 4.15) der Uniform Resource Identifier (URI) des LDAP Servers in die Datei `/etc/openldap/ldap.conf` geschrieben.

```
def String output = "URI ${uri}\n"
def File dstFile = new File ("/etc/openldap/ldap.conf")
dstFile << output
```

Listing 4.15: Setzen des LDAP Server URI (Groovy-Applikation)

Ergänzend schreibt die hinter der Aktivität `Set BaseDN` liegende Applikation (vgl. Listing 4.16) den so genannten BaseDN in die Datei.

```
def String output = "BASE ${base_dn}"
def File dstFile = new File ("/etc/openldap/ldap.conf")
dstFile.append(output)
```

Listing 4.16: Setzen des BaseDN im LDAP Client (Groovy-Applikation)

Die Kombination beider Angaben ermöglicht es dem Client Anfragen an den LDAP Server bzgl. des Verzeichnisbaumes unterhalb des BaseDN zu stellen.

Einfügen eines LDAP-Eintrags Initial enthält ein LDAP-Verzeichnis keine Einträge und ist über Werkzeuge wie `ldapadd` befüllbar. Bei `ldapadd` sind neben dem hinzuzufügenden Objekt der hinzufügende Nutzer (durch Angabe des `binddn` und ggf. Passworts) sowie Kontaktinformationen für den LDAP Servers anzugeben. Zur Verdeutlichung ist in Listing 4.17 das Hinzufügen eines LDAP-Objekts aus einer Datei mittels `ldapadd` dargestellt.

```
1 ldapadd -x -H ldap://<LDAP Server>:<Port>/ -D <binddn> -w <password> -f <
    file>
```

Listing 4.17: Hinzufügen eines LDAP-Objekts aus einer Datei

Die Kombination von `<LDAP Server>` und `<Port>` bildet hierbei die Kontaktinformation und der Wert für `<binddn>` identifiziert den Nutzer.

Realisiert ist das Einfügen durch den in Abbildung 4.14(a) dargestellten Workflow. Die dem Anwender präsentierte Eingabemaske ist in Abbildung 4.13 zu sehen, wobei für die Portnummer der Standardwert 389 bereits in der Maske vorgegeben ist.

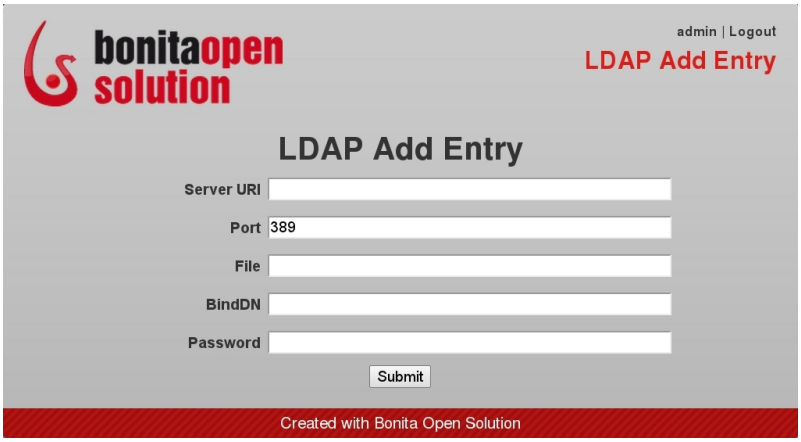


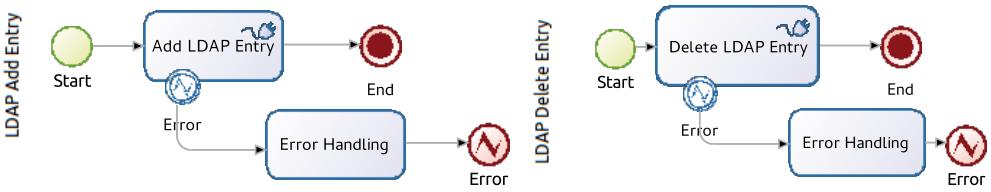
Abbildung 4.13: Web-Oberfläche für das Hinzufügen eines LDAP-Eintrags

Entfernen eines LDAP-Eintrags Innerhalb des LDAP-Verzeichnisbaums sind Objekte eindeutig durch ihren Distinguished Name (DN) identifiziert. Beim Entfernen eines Objekts wird diese Eigenschaft ausgenutzt: das zu löschende Objekt ist explizit per Distinguished Name zu spezifizieren. Zusätzlich zum Distinguished Name sind bei Verwendung des Werkzeugs `ldapdelete` noch der Kontaktstring des LDAP Servers sowie der ausführende Nutzer (anhand von `binddn` und Passwort) als Eingaben notwendig (vgl. Listing 4.18).

```
1 ldapdelete -x -H ldap://<LDAP Server>:<Port> -D <binddn> -w <password>
    DN_2_Remove
```

Listing 4.18: Entfernen eines Objekts aus dem LDAP-Verzeichnis

Der resultierende, für das Entfernen eines LDAP-Eintrags verantwortliche Workflow ist in in Abbildung 4.14(b) gezeigt.



(a) Einfügen eines Objekts (b) Entfernen eines Objekts

Abbildung 4.14: Zwei Workflows für Operationen auf einem LDAP-Verzeichnis

4.6 Nagios Monitoring Service

Nagios[5] ist eine Open Source Software, die Ressourcenbetreiber bei der Überwachung von Rechnern und Diensten unterstützt. Der Begriff der Überwachung bezieht sich in diesem Kontext auf das Prüfen der Verfügbarkeit und der Einhaltung vordefinierter Grenzwerte. Nagios selbst ist modular aufgebaut und besteht aus folgenden Komponenten:

- Ein zentraler Daemon ist für die zeitliche Anordnung und Ausführung der konfigurierten Tests verantwortlich. Des Weiteren übernimmt der Daemon die Auswertung der Tests und leitet ggf. Maßnahmen, wie z. B. die Benachrichtigung der zuständigen Systemadministratoren, ein.

- Die einzelnen Tests sind in Nagios als Plug-ins einbindbar. Von den Nagios Plug-in-Webseiten[6] ist ein Software-Archiv mit einem Grundstock von Plug-ins beziehbar, welches durch die offene Architektur von Nagios sehr gut durch eigene Entwicklungen erweiterbar ist.

- Durch den Nagios Remote Plugin Executor (NRPE) kann der Daemon Plug-ins auf entfernten Ressourcen ausführen. Das Gegenstück zum NRPE bildet der Nagios Service Check Acceptor (NSCA), welcher zum Einsatz kommt, wenn die entfernten Ressourcen nicht direkt durch den Nagios Remote Plugin Executor erreichbar sind. In diesen Fällen werden auf den Ressourcen eigenständig die Tests durchgeführt und die Ergebnisse per Nagios Service Check Acceptor an den zentralen Daemon weitergereicht.

[5] `http://www.nagios.org`
[6] `http://nagiosplugins.org/`

Nachfolgend sind mehrere Nagios-betreffende Workflows vorgestellt. Diese umfassen beispielsweise die Installation eines Nagios Servers, eines Nagios Remote Plugin Executors und der Plug-ins.

Nagios Server Die Installation des Nagios Servers erfolgt nicht mittels der Paketverwaltungssoftware YUM, sondern über das Herunterladen, Entpacken und Kompilieren des Nagios Quellcodes. Die dabei ausgeführten Aktivitäten und deren Anordnung untereinander sind in Abbildung 4.15 gezeigt. Nach der Installation der vorausgesetzten Software

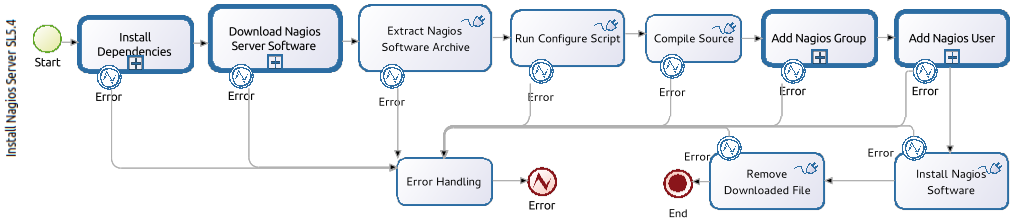


Abbildung 4.15: Workflow zur Installation des Nagios Servers

(z. B. des Compilers `gcc` und des Werkzeugs `make`) mittels eines Subflow-Aufrufs in der Aktivität `Install Dependencies` erfolgt in den nachgelagerten Aktivitäten `Download Nagios Server Software` und `Extract Nagios Software Archive` das Herunterladen und Entpacken des Nagios Quellcodes in ein temporäres Verzeichnis.

Die Aktivität `Download Nagios Server Software` bindet hierzu über einen Subflow-Aufruf den Workflow `Download Files` ein. Die in diesem Workflows gekapselte Groovy-Applikation ist in Listing 4.19 zu sehen. Dabei wird für das Herunterladen von Dateien aus dem Internet das Werkzeug `wget` verwendet, welchem von außen die herunterzuladende Datei sowie das Zielverzeichnis auf dem lokalen System mitgeteilt werden müssen. Der Fortschritt des Downloads wird in der Datei `/tmp/wget.log` protokolliert.

```
1 def String command = "wget --append-output=/tmp/wget.log -N --no-proxy ${
    filename} -P ${directory}"
  def Process proc = command.execute()
3 proc.waitFor()
```

Listing 4.19: Herunterladen einer Datei aus dem Internet (Groovy-Applikation)

Nach dem erfolgreichen Download erzeugt die Aktivität `Run Configure Script` die erforderlichen Makefiles durch den Aufruf des im Software-Archiv mitgelieferten `configure`-Skripts (vgl. Listing 4.20). Von außen können der in Listing dargestellten Applikation Argumente, beispielsweise das spätere Installationsverzeichnis, in Form der `configureOptions` zugeführt werden. Diese werden 1:1 an das `configure`-Skript weitergereicht.

```
1 def String command = "./configure ${configureOptions}"
  def Process proc = command.execute(null, new File("${temporaryDirectory}/
      nagios-3.2.3/"))
3 proc.waitFor()
```

Listing 4.20: Aufruf des configure-Skripts für den Nagios Server (Groovy-Applikation)

Der Start des Kompiliervorgangs wird in der Aktivität `Compile Source` eingeleitet. Nachdem der Vorgang abgeschlossen ist, werden in den Aktivitäten `Add Nagios Group` und `Add Nagios User` der Nagios-Nutzer sowie eine Nagios-Gruppe angelegt, um später die Ausführung des Nagios-Dienstes unter Verwendung der User Identification Number (UID) bzw. der Group Identification Number (GID) dieses Nutzers zu erlauben. Die letzten beiden Aktivitäten des Workflows, `Install Nagios Server` und `Remove Downloaded File` widmen sich der Installation des Kompilats und dem Entfernen des zu Beginn heruntergeladenen Software-Archivs.

Nagios Client Die Installation der Nagios Client Software auf einem Rechner ist in die Aktivitäten `Install Nagios NRPE` und `Install Nagios Plug-Ins`, wie im zugehörigen Workflow in Abbildung 4.16 gezeigt, zerlegbar. Jede dieser Aktivitäten ruft einen der nachfolgend vorgestellten Workflows per Subflow-Aufruf auf.

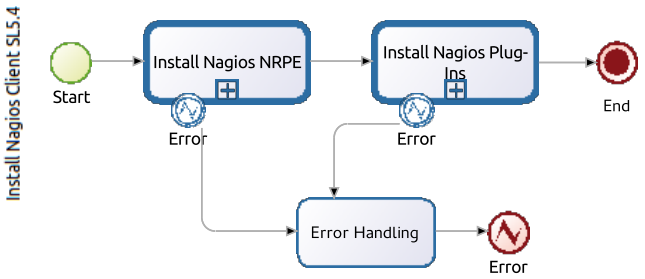


Abbildung 4.16: Workflow zur Installation der Nagios Client Software

So zeigt Abbildung 4.17 den Workflow für die Installation des Nagios Remote Plugin Executors. Da die Installation des Nagios Remote Plugin Executors ebenfalls über das Herunterladen, Entpacken und Kompilieren des entsprechenden Quellcodes erfolgt, besitzen die im Workflow verwendeten Aktivitäten verglichen mit denen des Workflows für den Nagios Server (vgl. Abbildung 4.15) analoge Funktionalität. Aufgrund dessen bietet sich hier eine Untersuchung hinsichtlich der Wiederverwendbarkeit an, die in Abschnitt 6.3 aufgegriffen wird.

Wie bereits angedeutet, erfolgt die Auflösung der Software-Abhängigkeiten in der Aktivität `Install Dependencies` und das Herunterladen und Entpacken des Nagios Re-

mote Plugin Executor-Quellcodes in den Aktivitäten `Download NRPE Software` und `Extract NRPE Software`. Der Inhalt der bei der Extraktion zum Einsatz kommenden Groovy-Applikation ist in Listing 4.21 zu sehen. Als Argumente können der Applikation die zu extrahierende Datei und das Zielverzeichnis übergeben werden.

```
1 def String command = "tar xzf ${archiveName} -C ${targetDirectory}"
  def Process proc = command.execute()
3 proc.waitFor()
```

Listing 4.21: Extrahieren eines `tar.gz`-Archivs (Groovy-Applikation)

Dem Kompilieren des Quellcodes in der Aktivität `Compile Source` schließen sich – analog zum Vorgehen beim Nagios Server – die Erzeugung einer Nagios-Gruppe und eines Nagios-Nutzers an. Die Installation des Nagios Remote Plugin Executor-Kompilats und das Löschen des zu Beginn heruntergeladenen Quellcode-Archivs bilden die beiden abschließenden Aktivitäten des Workflows.

Nachdem die Installation des Nagios Remote Plugin Executor durchlaufen ist, beginnt der in Abbildung 4.16 gezeigte Workflow mit der Installation der Nagios Plug-ins. Der hierzu entworfene Workflow ist in Abbildung 4.18 dargestellt. Aus der Darstellung ersichtlich

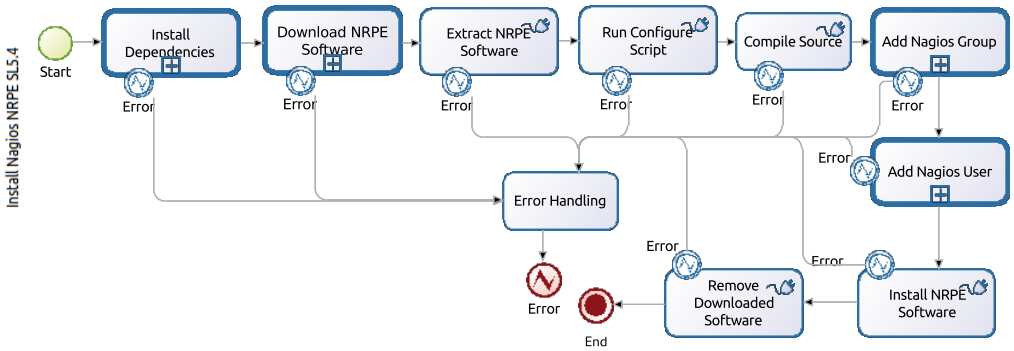


Abbildung 4.17: Workflow zur Installation des Nagios NRPE

ist die Analogie zu den Abläufen bei der Installation des Nagios Servers bzw. des Nagios Remote Plugin Executors. Lediglich die Schritte des Anlegens einer Nagios-Gruppe und eines Nagios-Nutzers entfallen. Daher wird auf den Workflow an dieser Stelle nicht weiter eingegangen.

4.7 Ganglia Monitoring System

Neben Nagios existiert mit Ganglia[7] ein weiteres Werkzeug für die kontinuierliche Überwachung von Ressourcen. Ganglia selbst ist eine Open Source Software und unter der Berkeley

[7] `http://ganglia.sourceforge.net/`

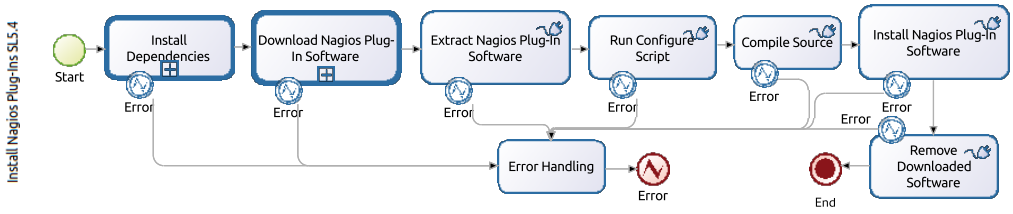


Abbildung 4.18: Workflow zur Installation der Nagios Plug-ins

Software Distribution (BSD)-Lizenz veröffentlicht.
Im Gegensatz zu Nagios fragt Ganglia nicht aktiv den Zustand von Ressourcen ab, sondern Daemons auf den Ressourcen propagieren diesen zu einer zentralen Instanz. Die zentral aggregierten Informationen ermöglichen einen Blick auf aktuelle als auch historische Daten für beispielsweise die CPU-Last, den Netzwerkdurchsatz und den freier Festplattenspeicher einzelner Ressourcen. Intern setzt Ganglia zur Informationspropagation zwischen den Ressourcen auf ein Multicast-basiertes Protokoll und erlaubt den Aufbau hierarchischer Strukturen. Des Weiteren nutzt es etablierte Standards, z. B. XML für die Repräsentation von Daten und die External Data Representation (XDR) (vgl. [Eis06]) für den kompakten Datentransport.

Aus Sicht eines Systemadministrators gliedert sich Ganglia in drei Komponenten auf: i) den Ganglia Monitoring Daemon (`gmond`), ii) den Ganglia Meta Daemon (`gmetad`) und iii) die Personal Home Page (PHP)-basierte Web-Oberfläche. Nachfolgend sind Workflows für die Installation der einzelnen Komponenten dargestellt, wobei vereinfachend angenommen wird, dass `gmond` und die Web-Oberfläche auf demselben Rechner installiert werden.

Ganglia Meta Daemon und Web-Oberfläche Der Ganglia Meta Daemon erhält periodisch von Datenquellen Informationen und bereitet diese für eine spätere Darstellung über die Web-Oberfläche auf. Als Datenquellen sind sowohl `gmond` Daemons als auch andere `gmetad` Daemons möglich. Letzteres ist z. B. ein gängiges Vorgehen bei der Überwachung mehrerer disjunkter Rechen-Cluster. Die zweite Komponente, die Web-Oberfläche (vgl. Abbildung 4.19 für ein Beispiel), bietet eine Echtzeit-Übersicht über die aggregierten Informationen der einzelnen überwachten Datenquellen.

Da für die in dieser Arbeit verwendeten Betriebssysteme Scientific Linux und SUSE Linux Enterprise Server keine Binärpakete für die Ganglia Software vorlagen, wird diese analog zum Vorgehen bei Nagios aus dem Quellcode übersetzt. Der entwickelte Workflow zur Installation des Daemons `gmetad` und der Web-Oberfläche ist in Abbildung 4.20 dargestellt. Die im Workflow vorhandenen Aktivitäten sind sowohl hinsichtlich Anordnung als auch Funktionalität ähnlich zu denen des in Abschnitt 4.6 vorgestellten Workflows zur

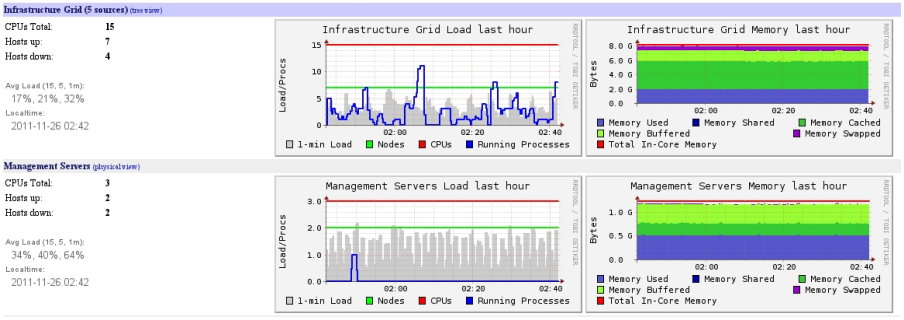


Abbildung 4.19: Beispielhafte Web-Oberfläche der Ganglia Software, Quelle: `http://ganglia.millennium.berkeley.edu/`

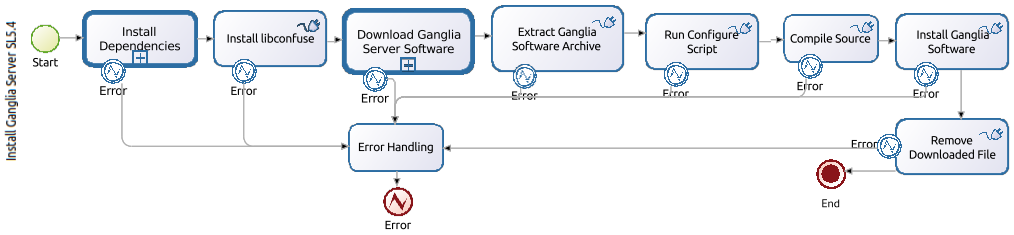


Abbildung 4.20: Workflow zur Installation des Ganglia gmetad

Installation des Nagios Servers, daher wird hier nicht weiter darauf eingegangen.

Abbildung 4.21: Web-Oberfläche des Workflows zur Installation des Ganglia Servers

Wie aus Abbildung 4.21 ersichtlich, erhält der Workflow als Eingabe ein Tupel bestehend aus einem Link zu einem Software-Archiv, welches den Ganglia-Quellcode enthält, ii) einem Pfad zur temporären Speicherung des Archivs und iii) Optionen, welche dem in der Aktivität `Run Configure Script` aufgerufenen `configure`-Skript übergeben werden.

Ganglia Monitoring Daemon Der Ganglia Monitoring Daemon `gmond` wird auf den zu überwachenden Ressourcen ausgeführt und propagiert periodisch deren Zustand sowie weitere relevante Veränderungen an einen oder mehrere `gmeta` Daemons. Des Weiteren lauscht der `gmond` als Bestandteil einer Multicast-Gruppe nach Informationen weiterer überwachter Ressourcen.

Die Installation des `gmond` (vgl. Abbildung 4.22) verläuft weitestgehend analog zur Installation des `gmetad`. Es wird dasselbe Quellcode-Archiv verwendet, allerdings sind etwa die in der Aktivität `Run Configure Script` an das `configure`-Skript übergebenen Optionen und die so genannten Targets beim Aufruf des `make`-Werkzeugs in der Aktivität `Compile Source` anders. Anhand dieser Erkenntnis bietet es sich an, die Zusammenführung der beiden Workflows zu einem einzigen zu untersuchen, wobei über den Einbau weiterer Logik die Wahl einer `gmeta`- oder `gmond`-Installation ermöglicht werden kann.

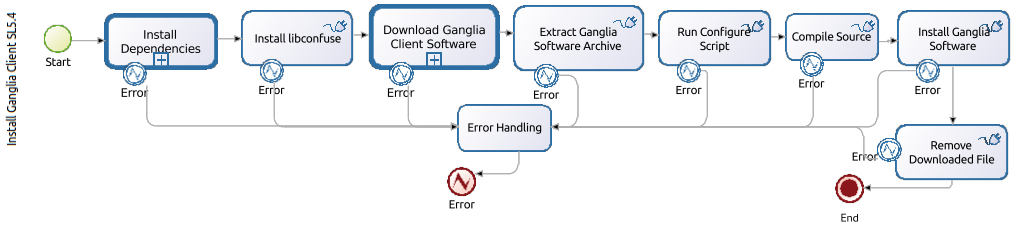


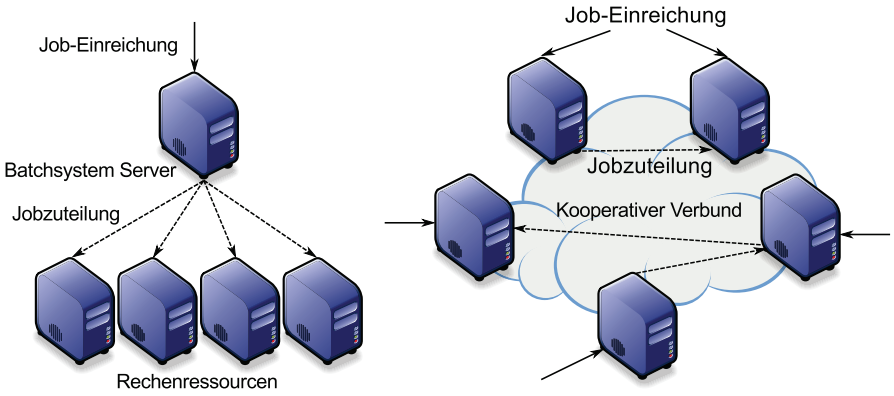
Abbildung 4.22: Workflow zur Installation des Ganglia gmond

4.8 Batchsystem

In einem Batchsystem (dt.: Stapelverarbeitungssystem) sind Mengen von Rechen-, Speicher- und Netzwerkressourcen sowie Software-Lizenzen zusammengefasst und stehen Nutzern über wohldefinierte Schnittstellen zur Verfügung. Im Fall einer Rechenressource haben Nutzer über eine solche Schnittstelle beispielsweise die Möglichkeit, Rechenaufträge (Jobs) einzureichen und deren Ergebnisse abzuholen.
Die Motivation für ein Batchsystem liegt in der zentralisierten Verwaltung einzelner Rechenressourcen und damit verbunden, dem reduzierten administrativen Aufwand für den Betreiber. Des Weiteren erfordert die limitierte Anzahl bestimmter Ressourcentypen, wie Software-Lizenzen, eine Ressourcen-übergreifende Verwaltung zur Sicherstellung eines hohen Nutzungsgrads.

Über die Zeit etablierten sich zwei unterschiedliche Konzepte für den Aufbau eines Batchsystems (vgl. Abbildung 4.23). Beim Master/ Slave-Konzept gibt es einen dedizierten Server, der die Zuteilung eingereichter Jobs auf eine oder mehrere Rechenressourcen vornimmt. Auf den Rechenressourcen sind zur lokalen Jobverwaltung Agenten installiert,

welche einerseits die Kommandos des Servers entgegennehmen und andererseits den Server über Änderungen einzelner Jobzustände informieren. Im Gegensatz zum Master/ Slave-Konzept greift das Kooperative Maschinen-Konzept die Idee der Dezentralisierung auf. Jede Rechenressource des kooperativen Verbundes kann lokal vorliegende Jobs selber ausführen oder diese an andere Verbundressourcen zur Ausführung abgeben. Insgesamt wird damit das Ziel einer gleichmäßigen Arbeitslastverteilung im Verbund verfolgt.



(a) Master/ Slave-Konzept (b) Kooperative Maschinen-Konzept

Abbildung 4.23: Verschiedene Konzepte für den Aufbau von Batchsystemen

Nachfolgend ist die schrittweise Installation und Konfiguration eines solchen Batchsystems anhand von Workflows beschrieben. Für die exemplarische Umsetzung wurde das Batchsystem Terascale Open-Source Resource and Queue Manager (TORQUE)[8] ausgewählt, welches frei verfügbar und zudem Bestandteil der D-Grid Referenzinstallation (vgl. Abschnitt 5.4) ist. In Bezug auf die zuvor vorgestellten Konzepte ist TORQUE ein Repräsentant des Master/ Slave-Konzepts: es existiert zentral der `pbs_server`-Prozess zu dem sich die auf den angeschlossenen Rechenressourcen ausgeführten `pbs_mom`-Prozesse verbinden. Letztere dienen der lokalen Jobverwaltung.

TORQUE selbst verfügt über einen Plug-in-Mechanismus, der die Austauschbarkeit des Zuweisungsalgorithmus von Jobs zu den einzelnen Ressourcen sicherstellt. Im Originalzustand setzt TORQUE einen First-come, first-served (FCFS)-Algorithmus ein, der oftmals gegen den Scheduler Maui[9] oder dessen kommerziellen Nachfolger Moab[10] ausgetauscht wird.

[8] `http://www.clusterresources.com/pages/products/torque-resource-manager.php`

[9] `http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php`

[10] `http://www.clusterresources.com/pages/products/moab-cluster-suite/workload-manager.php`

Die Installation der einzelnen Maui- und TORQUE-Komponenten (z. B. Server, Client zur Job-Einreichung und Agent zur lokalen Job-Verwaltung) ist sowohl über das Kompilieren des Quellcodes als auch über das Einspielen von RPM-Paketen möglich. Für beide Vorgehen sind in den vorangegangenen Abschnitten bereits Workflows vorgestellt. Nachfolgend wird auf die Installation anhand vorhandener RPM-Pakete zurückgegriffen.

4.8.1 TORQUE Server

Die notwendigen Schritte für die Installation und Konfiguration des TORQUE Batchsystem Servers sind im Workflow `Install Torque Server SL5.4` (vgl. Abbildung 4.24) zusammengefasst und nachfolgend erläutert.

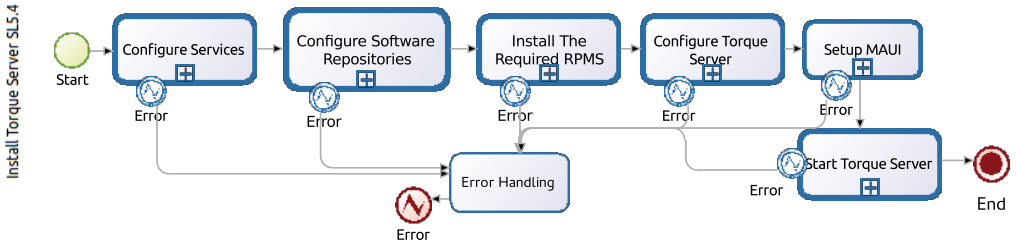


Abbildung 4.24: Workflow für die Installation des TORQUE Servers

Die erste Aktivität `Configure Services` stoppt nicht-benötigte Systemdienste, wozu sie den in Abbildung 3.23 dargestellten Workflow über einen Subflow-Aufruf verwendet. Das Herunterladen und Installieren der TORQUE Server Software erfolgt direkt nach der Einrichtung der zugehörigen YUM-Repositories. Für den TORQUE Server werden hierzu durch den Aufruf der Aktivität `Configure Software Repositories` bzw. des dahinterliegenden Workflows `Configure Torque Server Repositories` (vgl. Abbildung 4.25) Informationen über die drei Repositories `base`, `updates` und `externals` auf dem System hinterlegt.

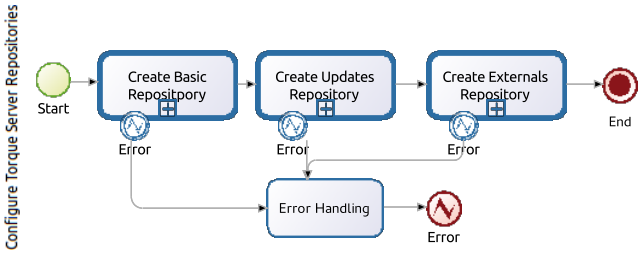


Abbildung 4.25: Workflow für die Repository-Konfiguration des TORQUE Servers

Die Konfiguration des Servers übernimmt ein separater Workflow (vgl. Abbildung 4.26), der über den Aufruf der Aktivität `Configure Torque Server` eingebunden ist. Wäh-

rend dessen Ausführung wird über eine Groovy-Applikation (vgl. Listing 4.22) in der Aktivität `Set Server Name` der Rechnername, auf dem der `pbs_server`-Prozess läuft, in die Datei `/var/torque/server_name` eingetragen. Sollte das Verzeichnis `/var/torque/` bis dato nicht existieren, so wird es in Zeile 3 der Applikation angelegt.

```
1 def d= new File('/var/torque/')
  if (!d.exists()){
3   d.mkdirs()
  }
5 def f = new File('/var/torque/server_name')
  f.write("${serverName}")
```

Listing 4.22: Konfiguration von `/var/torque/server_name` (Groovy-Applikation)

Bevor in dem Server einzelne Queues eingerichtet werden können, muss dieser gestartet werden. Für diese Aufgabe ist die Aktivität `Start Service` verantwortlich. In der Aktivität `Add Queues` erfolgt das Hinzufügen von Queues unter Verwendung von Standardwerten für etwa `queue_type` und `Priority`. Dabei kommt die in Listing 4.23 dargestellte Groovy-Applikation zum Einsatz, welche von außen im Argument `queues` eine Liste mit den Namen der einzurichtenden Queues erhält. Für jedes Element dieser Liste werden in der Datei `qmgr_commands` die auszuführenden Befehle zur Einrichtung festgehalten und abschließend ausgeführt.

```
  def String output = ""
2 def String[] queueArray= "${queues}".split()
  queueArray.each(){ item ->
4   output =  "create queue " +  item + "\n"
    output += "set queue " + item + " queue_type = Execution\n"
6   output += "set queue " + item + " Priority = 100\n"
    output += "set queue " + item + " max_queuable = 100\n"
8   output += "set queue " + item + " max_running = 100\n"
    output += "set queue " + item + " resources_max.nodect = 1\n"
10  output += "set queue " + item + " resources_default.nodes = 1:ppn=1\n"
    output += "set queue " + item + " resources_max.walltime = 72:00:00\n"
12  output += "set queue " + item + " acl_group_enable = False\n"
    output += "set queue " + item + " enabled = True\n"
14  def File f = new File ("/tmp/qmgr_commands")
    f.write(output)
16  def String command = "qmgr < /tmp/qmgr_commands"
    def Process proc = command.execute()
18  proc.waitFor()
  }
```

Listing 4.23: Erzeugung von Queues im Batchsystem TORQUE (Groovy-Applikation)

Der letzte Konfigurationsschritt besteht im Hinzufügen der Namen aller Workernodes in die Datei `/var/torque/server_priv/nodes`. Die zugehörige Applikation ist in Listing 4.24 gezeigt und erhält von außen die Liste der Workernodes.

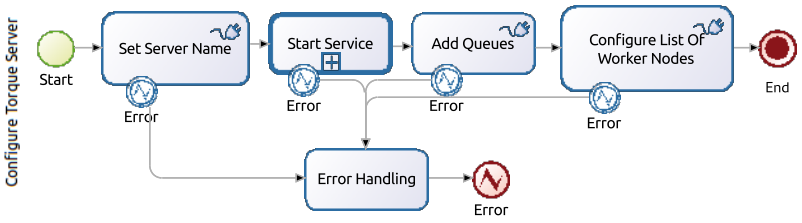


Abbildung 4.26: Workflow für die Konfiguration des TORQUE Servers

```
def File file = new File ("/var/torque/server_priv/nodes")
def String output = ""

"${workernodes}".tokenize().each(){ item ->
  output += item + "\n"
}
file.write(output)
```

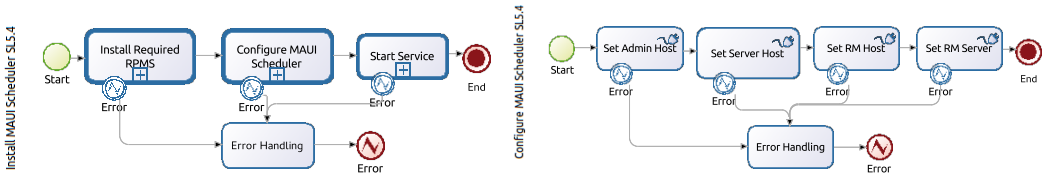
Listing 4.24: Hinzufügen von Workernodes (Groovy-Applikation)

Nach Abarbeitung des Workflows ist der TORQUE Server einsatzbereit, nutzt allerdings noch einen FCFS-Algorithmus für das Scheduling. Daher wird im Folgenden die Installation des Maui Cluster Schedulers, der alternative Algorithmen anbietet, beschrieben.

Maui Cluster Scheduler Wie zuvor erwähnt, wird der Maui Cluster Scheduler oftmals anstelle des mit TORQUE mitgelieferten FCFS-Schedulers verwendet. Maui selbst greift auf die von TORQUE angebotenen Kommunikationsschnittstellen für Plug-ins zurück und setzt daher dessen Existenz voraus. Diese Abhängigkeit ist im Workflow für die Installation des TORQUE Servers (vgl. Abbildung 4.24) durch die Anordnung der Aktivitäten berücksichtigt.

Die Installation des Maui Schedulers (vgl. Abbildung 4.27(a)) erfolgt analog zum Vorgehen beim TORQUE Server und somit anhand einer Menge von vorgefertigter Binärpakete aus einem Software-Repository. Da die Maui Software im selben Repository wie die TORQUE Software liegt, entfällt hier allerdings das Hinterlegen von Repository-Informationen im System.

Nach der erfolgreichen Installation der Binärpakete geht man in der Aktivität `Configure MAUI Scheduler` (vgl. Abbildung 4.27(a)) per Subflow-Aufruf zur Konfiguration des Schedulers über. Die einzelnen Konfigurationsschritte sind in dem in Abbildung 4.27(b) gezeigten Workflow zusammengefasst, wobei die Anpassungen alle in der



(a) Installation (b) Konfiguration

Abbildung 4.27: Workflows für den Maui Cluster Scheduler

zentralen Konfigurationsdatei `maui.cfg` vorgenommen werden. In dieser lassen sich beispielsweise Standing Reservations und Nutzer- bzw. Gruppen-spezifische Begrenzungen einrichten.

Über den entworfenen Workflow zur Konfiguration von Maui sind die Größen `ADMINHOST`, `SERVERHOST`, `RMSERVER` und `RMHOST` einstellbar. Beispielsweise legt der Wert der Variablen `ADMINHOST` fest, von welchen Rechnern aus der Maui Scheduler administrative Kommandos entgegennimmt. Die in der Aktivität `Set Admin Host` zum Einsatz kommende Groovy-Applikation (vgl. Listing 4.25) nutzt zum Ersetzen des voreingestellten Wertes `null` das Werkzeug `sed`.

```
1 def String[] command = ["sh", "-c", "sed -i \"s/ADMINHOST.*null/ADMINHOST
      ${adminHost}/g\" /var/spool/maui/maui.cfg"]
  def Process proc = command.execute()
3 proc.waitFor()
```

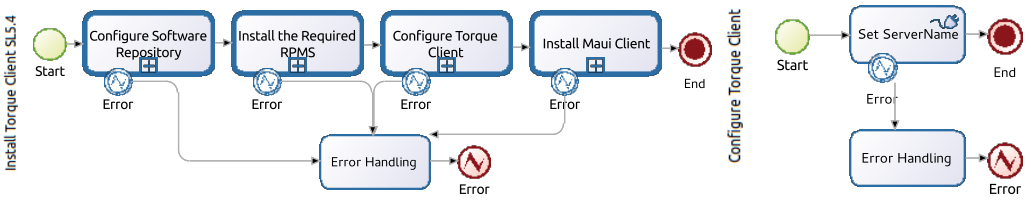
Listing 4.25: Setzen des Wertes für ADMINHOST (Groovy-Applikation)

Der Wert für die Variable `SERVERHOST` entspricht dem Namen des Rechners auf dem der Maui Cluster Scheduler läuft und hinter dem `RMSERVER` verbirgt sich der Name des Rechners, auf dem der TORQUE Server ausgeführt wird. Der Inhalt der Variablen `RMSERVER` wird von Maui u. a. zur Etablierung eines Kommunikationskanal zum TORQUE Server verwendet.

4.8.2 TORQUE Client

Die Einreichung von Jobs an den Batchsystem Server erfolgt über eine separate Client Software, die z. B. auf einem Arbeitsplatzrechner installierbar ist. Die Workflows für die Installation und Konfiguration der TORQUE Client Software (vgl. Abbildung 4.28) sind nachfolgend erläutert.

Der Installation der TORQUE Software-Pakete in Form von RPMs folgt in der zweiten Aktivität des in Abbildung 4.28(a) gezeigten Workflows. Hierbei wird die erfolgreiche Hinterlegung der notwendigen Informationen zu den Software-Repositories während der Akti-



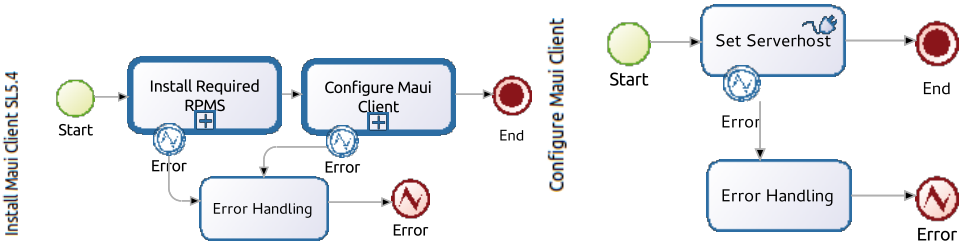
(a) Installation (b) Konfiguration

Abbildung 4.28: Workflows für die TORQUE Client Software

vität `Configure Software Repository` vorausgesetzt. Die nachfolgende Aktivität `Configure Torque Client` leitet über einen Subflow-Aufruf die Konfiguration des TORQUE Client ein. Der zugehörige Workflow (vgl. Abbildung 4.28(b)) besteht aus nur einer Aktivität, in der der Rechnername des zu kontaktierenden Batchsystem Server in die Datei `/var/torque/server_name` geschrieben wird. Nach dem Rücksprung aus dem Subflow beginnt mit dem Übergang zur Aktivität `Install Maui Client` die Installation der Maui Client Software.

Maui Cluster Scheduler Client Zusammen mit der TORQUE Client Software wird oftmals auch die Maui Client Software installiert. Diese wird zwar nicht zur Job-Einreichung benötigt, ermöglicht aber die Anzeige der aktuellen Anordnung der aktiven, wartenden und angehaltenen Jobs im Batchsystem. Weitere Befehle, wie beispielsweise `diagnose`, geben zudem eine Übersicht über Reservierungen und eventuell existierende Beschränkungen für Nutzer oder Gruppen.

Die Workflows für die Installation und Konfiguration der Client Software (vgl. Abbildung 4.29) sind recht einfach gehalten. Nach dem Download und der Installation der not-



(a) Installation (b) Konfiguration

Abbildung 4.29: Workflows für die Maui Cluster Scheduler Client Software

wendigen zwei RPMs für den Maui Client in der Aktivität `Install Required RPMS` des Installationsworkflows (vgl. Abbildung 4.29(a)) wird in der Aktivität `Set Serverhost` des Konfigurationsworkflows die Datei `maui.cfg` durch das Setzen des Wertes für

die Variable `SERVERHOST` modifiziert. Letzteres übernimmt die in Listing 4.26 gezeigte Groovy-Applikation.

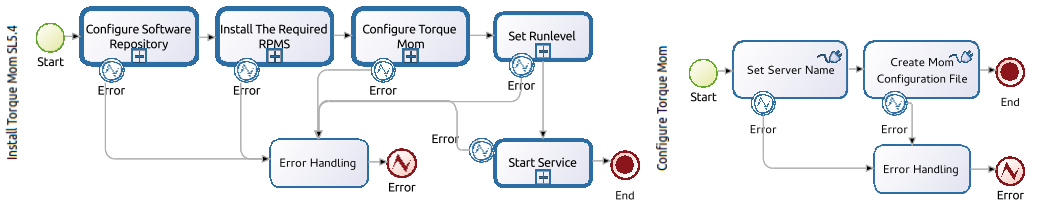
```
def command = ["sh", "-c", "sed -i \"s/SERVERHOST.*null/SERVERHOST ${
    serverHost}/g\"  ${configLocation}"]
def Process proc = command.execute()
proc.waitFor()
```

Listing 4.26: Setzen des Wertes für SERVERHOST (Groovy-Applikation)

Die in der Applikation verwendete Variable `${configLocation}` beschreibt den Ort der Maui-Konfigurationsdatei im Verzeichnisbaum und ist standardmäßig mit dem Wert `/var/spool/maui/maui.cfg` belegt. Nach Abschluss der Workflows kann der Maui Client bereits mit dem Maui Server kommunizieren.

4.8.3 TORQUE Rechenknoten

Die Workflows für die Installation und Konfiguration eines Rechenknotens des TORQUE Batchsystems sind in Abbildung 4.30 dargestellt. Die notwendigen Pakete zur Installation



(a) Installation (b) Konfiguration

Abbildung 4.30: Workflow für den TORQUE Mom Dienst

der Batchsystem Software werden im zweiten Schritt des in Abbildung 4.30(a) dargestellten Workflows installiert; die zuvor ausgeführte Aktivität `Configure Software Repository` hinterlegt entsprechende Repository-Informationen im System. Die Konfiguration der Software geschieht über einen Subflow-Aufruf in der Aktivität `Configure Torque Mom`, welche den in Abbildung 4.30(b) gezeigten Workflow einbindet. Dieser besteht aus nur zwei Aktivitäten. Während der Ausführung der Aktivität `Set Server Name` wird in die Datei `/var/spool/pbs/server_name` der Name des Batchsystem Servers eingetragen. Die eigentliche TORQUE-Konfigurationsdatei, in der u. a. die ideale und maximal Last des Rechenknotens sowie die Verwendung eines Scratch-Verzeichnisses einstellbar ist, wird in der Aktivität `Create Mom Configuration File` erzeugt. Das Listing 4.27 zeigt die hinter dieser Aktivität implementierte Applikation.

```
def String output = "$pbsserver ${serverName}\n"
output += "$loglevel 3\n"
```

```
3 output += "$logevent 255\n"
  def File f = new File ("/var/torque/mom_priv/config")
5 f.write(output)
```

Listing 4.27: Erzeugen der Datei /var/torque/mom_priv/config (Groovy-Applikation)

Die erzeugte Datei `config` enthält mit der Angabe des Batchsystem Servers (`pbsserver`), des Log Level (`loglevel`) und der zu loggenden Ereignisse (`logevent`) eine Minimalkonfiguration.

4.9 Verknüpfung von Workflows und virtuellen Maschinen

In den bisherigen Abschnitten dieses Kapitels sind einerseits Workflows für eine Menge von Basisdiensten, die auf einem Betriebssystem aufsetzen, beschrieben und andererseits Workflows für die Bereitstellung virtueller Maschinen und die Installation von Betriebssystem in solche. Im Folgenden wird die Verbindung der beiden Komponenten erläutert und somit die Installation von Dienst und Betriebssystem in eine zuvor erzeugte, virtuelle Maschine dargestellt. Hierbei sind drei Schritte im Detail vorgestellt, nach deren Ausführung eine virtuelle Appliance vorliegt:

1. Die Extraktion der Workflows aus der Entwicklungsumgebung und die Integration in das Workflow-Archive der Workflow Engine.

2. Die Erzeugung einer virtuellen Maschine sowie die Installation des Betriebssystems über die Workflow Engine.

3. Die Installation eines Dienstes innerhalb der virtuellen Maschine.

Integration der Workflows in das Archiv der Workflow Engine Die in dieser Arbeit vorgestellten, in der Business Process Modeling Notation spezifizierten Workflows sind mit Bonita Studio erstellt. Dieses ist als reines Entwicklungswerkzeug gedacht und bietet daher beispielsweise Möglichkeiten zur testweisen Ausführung der erstellten Workflows an. Allerdings ist Bonita Studio nicht zur Verwendung durch den Endanwender (hier: der Appliance-Erzeuger) geeignet, da dieser nicht an der Modellierung, sondern an der Ausführung einzelner Workflows interessiert ist. Er benötigt in diesem Zusammenhang lediglich eine graphische Oberfläche über die einzelne Workflows gestartet und notwendige Parameter bereitgestellt werden können.
Eine Realisierung dieser Anforderung ist durch die Bonita Workflow Engine gegeben. Diese ist auf die Ausführung von Workflows spezialisiert, bietet eine Web-Oberfläche zur Parametereingabe und enthält keinerlei Modellierungskomponenten. Um mit dem Bonita Studio

erstellte Workflows mit der Workflow Engine zu nutzen, sind die Workflows zunächst aus Bonita Studio zu exportieren, wobei bezogen auf die zeitgleich exportierten und zur Ausführung notwendigen Software-Bibliotheken die drei Varianten *Light*, *Embedded* und *Client WAR* zur Auswahl stehen.

- Light: Erzeugung eines Web Service Archives (WARs) sowie eines separaten Verzeichnisses `lib`, welches die erforderlichen Bibliotheken enthält.
- Embedded: Erzeugung eines Web Service Archives, in dem die erforderlichen Bibliotheken enthalten sind.
- Client WAR: Erzeugung eines Web Service Archives, das direkt in einem Java Enterprise Edition (JEE)-Dienst verwendbar ist.

Des Weiteren erzeugt der Export-Vorgang ein Prozessdiagramm sowie die Prozessbeschreibung in einem Business Archive (BAR).

Einrichtung der Workflow Engine Nachdem zuvor der Export von Workflows aus dem Bonita Studio beschrieben wurde, ist nachfolgend die Installation und die Verwendung der Workflow Engine dargestellt. Diese setzt im Wesentlichen die Existenz eines Application Servers, wie JBoss[11] oder Apache Tomcat[12], voraus. In dieser Arbeit kommt bei der Integration und späteren Ausführung der Workflows die Kombination von JBoss und Bonita Workflow Engine zum Einsatz.

Auf den Webseiten[13] von BonitaSoft, dem Anbieter der Workflow Engine, steht für diese Kombination ein vorgefertigtes Software-Archiv zum kostenlosen Download bereit. In Listing 4.28 sind das Herunterladen mittels des Werkzeugs `wget` sowie die Extraktion des Software-Archivs in ein lokales Verzeichnis dargestellt.

```
1 $ wget http://download.forge.objectweb.org/bonita/BOS-5.5.1-JBoss-5.1.0.
    GA.zip
  $ unzip BOS-5.5.1-JBoss-5.1.0.GA.zip
```

Listing 4.28: Installation der Bonita Workflow Engine

In diesem Verzeichnis liegt nach dem Entpacken der Ordner `BOS-5.5.1-JBoss-5.1.-0.GA` vor, aus welchem durch den Aufruf des Skripts `run.sh` der mitgelieferte JBoss Server gestartet wird (vgl. Listing 4.29).

```
  $  cd./ BOS-5.5.1-JBoss-5.1.0.GA
2 $ ./bin/run.sh
```

[11] `http://www.jboss.org/jbossas`
[12] `http://tomcat.apache.org/`
[13] `http://download.forge.objectweb.org/bonita/`

```
4 ===============================================================
  JBoss Bootstrap Environment
6 JBOSS_HOME: /home/stefan/Downloads/BOS-5.5.1-JBoss-5.1.0.GA
  [...]
8 ===============================================================
  [...]
10 14:27:32,815 INFO [Http11Protocol] Starting Coyote HTTP/1.1 on http
     -127.0.0.1-8080
   14:27:32,845 INFO [AjpProtocol] Starting Coyote AJP/1.3 on ajp
     -127.0.0.1-8009
12 14:27:32,859 INFO [ServerImpl] JBoss (Microcontainer) [5.1.0.GA (build:
     SVNTag=JBoss_5_1_0_GA date=200905221053)] Started in 1m:26s:759ms
   [...]
```

Listing 4.29: Starten des JBoss Servers über die Kommandozeile

Nach dem Start des JBoss Servers ist die graphische Oberfläche der Workflow Engine über einen Webbrowser unter der URL `http://localhost:8080/bonita/console/-login.jsp` erreichbar (vgl. Abbildung 4.31). Der initial für die Anmeldung an der Oberfläche zu verwendende Nutzername ist `admin` und das Passwort `bpm`.

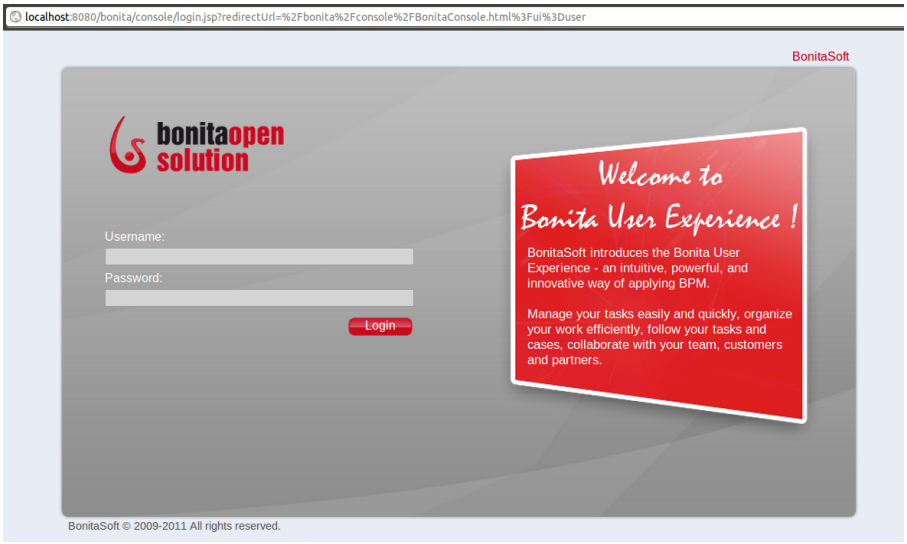


Abbildung 4.31: Anmeldemaske der Bonita Workflow Engine

Nach erfolgreicher Anmeldung wird man zur Administrationsansicht weitergeleitet, welche in Abbildung 4.32 gezeigt ist. In dieser Ansicht können beispielsweise weitere Workflows installiert oder bereits vorhandene temporär deaktiviert werden.

Für die Installation bzw. Integration neuer Workflows ist im linken Teil der Ansicht das Menü `Processes` auszuklappen, der Unterpunkt `Processes` zu wählen und abschlie-

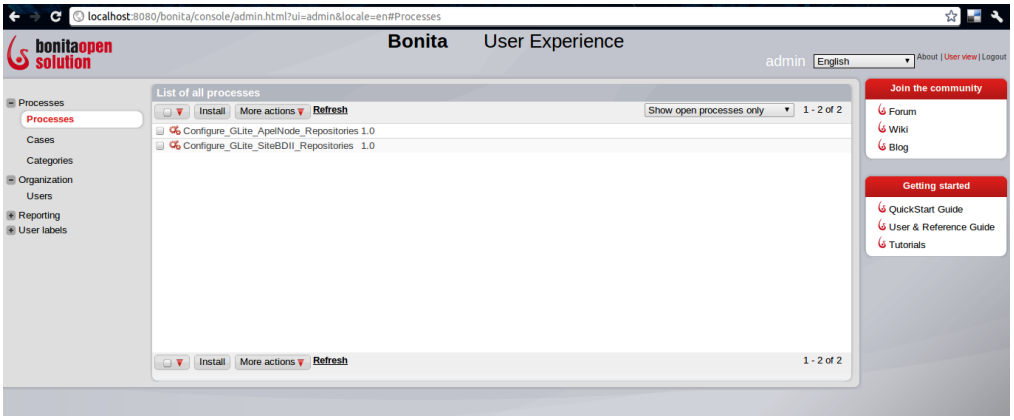


Abbildung 4.32: Administratoransicht der Bonita Workflow Engine

ßend der `Install`-Knopf zu drücken. In dem darauf folgenden Dialog ist der zu installierende und als Business Archive vorliegende Workflow anzugeben. Das Business Archive erhält man wie zuvor beschrieben durch den Export des Workflows über das Bonita Studio.

Weitere Vorarbeiten Bevor der erste importierte Workflow ausgeführt werden kann, müssen weitere Vorarbeiten durchgeführt werden. Nachfolgend sind diese im Detail erläutert.

- Einige Workflow-Aktivitäten bzw. -Applikationen erfordern die privilegierte Ausführung bestimmter Befehle. Zu dieser Menge von Befehlen zählen u. a. `mkdir`, `mount`, `umount` und `yum`.
  Durch die Erzeugung von Einträgen in der Datei `sudoers` im Verzeichnis `/etc` ist diese privilegierte Ausführung möglich. Das Listing 4.30 zeigt hierzu beispielhaft den Inhalt einer solchen `sudoers`-Datei. Gehört in diesem Beispiel der Workflow-Ausführende der Gruppe `adm` an, so darf dieser ohne die Eingabe eines Passworts die Befehle `mount`, `umount`, `debootstrap`, `apt-get`, `mkdir` privilegiert ausführen.

```
1 # Angabe der Nutzer-Privilegien
  root     ALL=(ALL:ALL) ALL
3
  # Mitglieder der Gruppe admin können root-Privilegien erhalten
5 %admin ALL=(ALL) ALL

7 # Mitglieder der Gruppe sudo können jegliche Kommandos ausführen
  %sudo    ALL=(ALL:ALL) ALL
9
  # Manuell hinzugefügt für die Bonita Workflow Engine
```

```
11 %adm ALL = NOPASSWD: /bin/mount,/bin/umount,/usr/sbin/debootstrap,/
       usr/bin/apt-get,/bin/mkdir
```

Listing 4.30: Inhalt der Datei `/etc/sudoers`

Die Konfiguration der `sudoers`-Datei ist besonders dann notwendig, wenn Workflow-basiert virtuelle Maschinen installiert werden, da die Installation üblicherweise unter Verwendung eines nicht-privilegierten Nutzerkontos erfolgt. Im Vergleich hierzu sind die Workflows innerhalb der erstellten virtuellen Maschine als `root`-Nutzer ausführbar, wodurch die Konfiguration der `sudoers`-Datei überflüssig wird.

- Die Workflow Engine überprüft bei dem Import eines Workflows nicht, ob alle dafür notwendigen Subflows bereits vorhanden sind. Dies kann dazu führen, dass dem Endanwender über die Web-Oberfläche nicht fehlerfrei ausführbare Workflows zur Ausführung angeboten werden.
  Daher ist bei einem Workflow-Import zu prüfen, ob alle aufgerufenen Subflows bereits im Archiv der Workflow Engine vorhanden sind.

Erzeugung einer virtuellen Maschine Die Erstellung einer virtuellen Maschine ist durch den Aufruf eines Workflows möglich. Nach dem Start der Workflow Engine durch den Systemadministrator ist deren Web-Oberfläche über einen Browser unter der URL `http://-localhost:8080/bonita/console/login.jsp` erreichbar. An dieser Oberfläche (vgl. Abbildung 4.31) muss sich der Nutzer anmelden. Ist die Anmeldung erfolgreich verlaufen, so kann aus der Menge der importierten Workflows einer zur Installation einer virtuellen Maschine gewählt werden.

In Abbildung 4.33 ist die Eingabemaske für die Erzeugung einer Debian Linux 6.0-basierten virtuellen Maschine gezeigt. Diese Maske wird dem Nutzer vor der Abarbeitung des Workflows präsentiert und liest alle von außen erforderlichen Argumente ein. Neben dem zu verwendenden Debian Linux-Mirror sind in der Maske beispielsweise noch der Name und die Größe des zu erzeugenden Festplattenabbilds vom Nutzer zu spezifizieren. Die abschließende Aktion des Nutzers besteht im Drücken des `Submit`-Knopfes. Am Ende des Workflows liegt in der Abbild-Datei das entsprechende Debian Linux vor.

Workflow-basierte Installation eines Dienstes in die virtuelle Maschine Um einen Dienst in die zuvor kreierte, virtuelle Maschine zu installieren, ist diese zunächst zu starten. Die Workflow Engine und deren Workflow-Archiv sind anschließend über das Network File System in die virtuelle Maschine einzubinden. Ist dies geschehen, so sind wie zuvor beschrieben einzelne Workflows innerhalb der virtuellen Maschine nach Anmeldung an der Web-Oberfläche ausführbar.

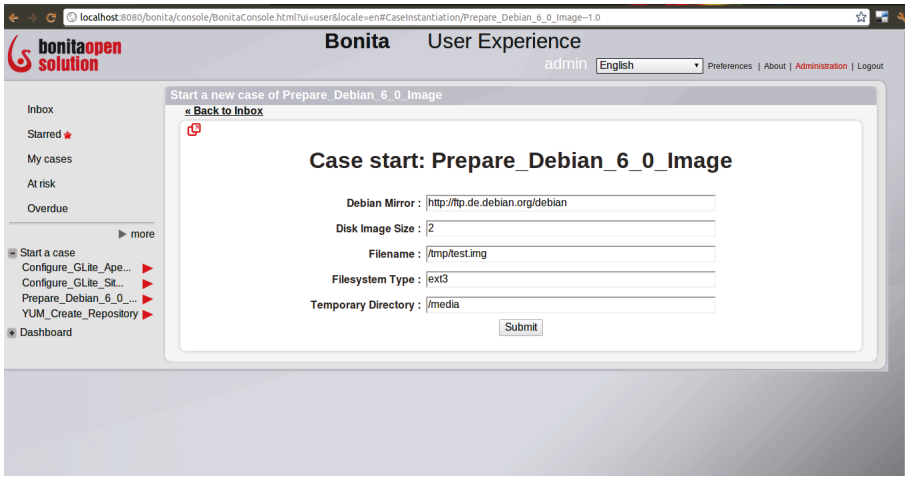


Abbildung 4.33: Aufruf eines Workflows über die Bonita Web-Oberfläche

4.10 Zusammenfassung

In diesem Kapitel sind für verschiedene Basisdienste, die man üblicherweise in einem Rechenzentrum als Voraussetzung für den Betrieb von Grid Middleware-Diensten benötigt, Workflows zur Installation und Konfiguration vorgestellt.

Zu diesen Basisdiensten zählen einfache Network File System Server, die eine kontrollierte, IP-basierte Freigabe einzelner Verzeichnisse ermöglichen, oder auch DHCP Server, die eine statische bzw. dynamische Zuweisung von IP-Adressen zu MAC-Adressen vornehmen (vgl. Abbildung 4.34). Weiterhin beschrieben ist der LDAP-Verzeichnisdienst mitsamt Workflows für das Einfügen und Entfernen von Objekten.

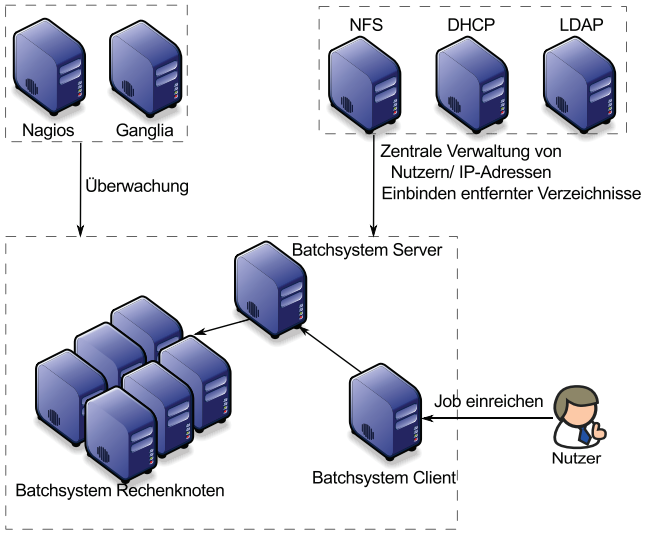


Abbildung 4.34: Zusammenhang zwischen den vorgestellten Basisdiensten

Oft werden diese Dienste durch zwei weitere Open Source-Lösungen, die der Überwachung der Infrastruktur dienen, ergänzt. Hierbei handelt es sich um das Ganglia Monitoring System und Nagios. Für beide Lösungen sind die erstellten Workflows zur Installation und Konfiguration – jeweils für die Server- als auch die Client-Komponente – dargestellt und erläutert.

Zudem existieren in vielen Rechenzentren für das High Performance Computing (HPC) und Grid Computing Stapelverarbeitungssysteme, in denen die Ausführung der von Nutzern eingereichten Jobs erfolgt. In diesem Kontext ist in Abschnitt 4.8 der Begriff des Batchsystems eingeführt und der Terascale Open-Source Resource and Queue Manager sowie der oft zeitgleich verwendete Scheduler Maui vorgestellt. Für beide sind ebenso Workflows zur Installation und Konfiguration präsentiert.

Der abschließende Punkt des Kapitels beschreibt die Verbindung der automatisierten Erzeugung virtueller Maschinen mit der automatisierten Installation von Diensten. Es sind dabei u. a. das Vorgehen zur Extraktion der Workflows aus der Entwicklungsumgebung und die Integration in das Workflow-Archive der Workflow Engine erläutert.

Kapitel 5

Szenarien

Der Einsatz von Workflows zur automatisierten Installation und Konfiguration von – im Vergleich zu Kapitel 4 – komplexeren Diensten ist nachfolgend anhand verschiedener Beispielszenarien beschrieben. Dazu stellt Abschnitt 5.1 einige Vorteile des Einsatzes von Workflows und Appliances im Bereich des wissenschaftlichen Arbeitens vor. Hier spielt gerade die Reproduzierbarkeit der Experimentumgebung bzw. der daraus resultierenden Daten eine wesentliche Rolle.

Ein weiteres betrachtetes Szenario (vgl. Abschnitt 5.2) ist die dynamische Kapazitätserweiterung verfügbarer Ressourcen, wie sie im Projekt D-Grid Scheduler Interoperability (DGSI)[1] angedacht ist. Unter dem Stichpunkt der Ressourcendelegation kann dort ein Scheduler eine in seiner Domäne verfügbare Ressource für ein zuvor ausgehandeltes Zeitfenster an einen anderen Scheduler ausleihen. In dem beschriebenen Szenario erfolgt die Erweiterung eines Clusters über multiple physische Ressourcen.

Die Erweiterung einer Ressource ist jedoch auch über die Hinzunahme fremder Kapazitäten aus einer Compute Cloud heraus möglich. Der Begriff des Cloud Computing und einzelne Teilaspekte sind in Abschnitt 5.2.1 beschrieben. Der darauf folgende Abschnitt beschreibt kurz die Ausgangssituation für die Ausdehnung oder auch initiale Erzeugung einer Grid-Ressource innerhalb einer solchen Compute Cloud, bevor in Abschnitt 5.3 Workflows für die Installation und Konfiguration der drei Grid Middleware gLite, Globus Toolkit und UNICORE vorgestellt werden.

Abschließend sind in Abschnitt 5.4 speziell für das D-Grid entworfene Workflows vorgestellt.

[1] http://www.d-grid-ggmbh.de/index.php?id=98

5.1 Reproduzierbarkeit wissenschaftlicher Daten

Sowohl die Arbeit als auch die Zusammenarbeit im wissenschaftlichen Umfeld unterliegt allgemein anerkannten Prinzipien (vgl. [BDE+98]), welche sich u. a. auf das Vorgehen zur Erzielung neuer Erkenntnisse und Ergebnisse, die Dokumentation von erzielten Resultaten und die Sicherung sowie Aufbewahrung von Daten beziehen.

Allgemein kann man in den experimentellen Wissenschaften ein wissenschaftliches Endergebnis als Summe einzelner Experimente und/ oder Beobachtungen sowie deren Auswertung und Interpretation beschreiben. Eine Voraussetzung für die Nachvollziehbarkeit des Endergebnisses ist die Reproduzierbarkeit der bei den Experimenten bzw. Beobachtungen entstandenen Daten.

Werden die Daten[2] im Rahmen einer Veröffentlichung verwendet, empfiehlt die Deutsche Forschungsgemeinschaft eine 10-jährige Aufbewahrung am Ort der Datenerzeugung. Die Aufbewahrung ermöglicht, u. a. den erneuten Zugriff auf die Daten zu einem späteren Zeitpunkt, ohne dass diese neu generiert werden müssen. Im Falle einer Anzweiflung der veröffentlichten Ergebnisse oder der genutzten Daten erweist sich die Speicherung ebenso als sinnvoll. Durch den Zugriff auf die Daten können Vorwürfe entkräftet oder belegt werden. Als Beispiele für solche Vorwürfe sind in [BDE+98] etwa die Erfindung von Ergebnissen (fabrication of data), das selektive Ausblenden unerwünschter Daten, die Substitution von Ergebnissen durch erfundene Ergebnisse und die missbräuchliche Anwendung statistischer Verfahren in der Absicht, Daten in ungerechtfertigter Weise zu interpretieren, angeführt.

Unabhängig davon, ob Anzweiflungen be- oder entkräftigt werden, muss ein Gutachter in der Lage sein, die Daten auf Manipulation bzw. Fälschung zu überprüfen. Da dem Gutachter aber zumeist weder Originaldaten noch Zeit, um das Experiment selbst zu wiederholen, zur Verfügung stehen, kann nur eine oberflächliche Überprüfung stattfinden.

Der Einsatz von Workflows bzw. daraus resultierenden virtuellen Appliances kann gerade bei der Sicherung von Primärdaten und bei der erneuten Durchführung des Experiments durch Dritte hilfreich sein. Die Durchführung eines Experiments unterliegt üblicherweise der Abarbeitung einer festen Folgen von Schritten über die sich der Durchführende im Vorfeld klar sein sollte. Dabei ist es unerheblich, ob die Schritte linear oder quasi parallel ausgeführt werden. In beiden Fällen kann die Abarbeitungsreihenfolge in einen Workflow überführt werden, wobei die einzelnen Aktivitäten Computer-gestützt oder manuell ausführbar sein können und untereinander entsprechend der Abarbeitungsreihenfolge mit Transitionen versehen sind.

Anhand des so konstruierten Workflows können Dritte den grundsätzlichen Ablauf des

[2] Bei den hier erwähnten Daten handelt es sich stets um Primärdaten.

Experiments nachvollziehen und dabei eventuell bereits erste Fehler identifizieren. Eine weitere Detailkenntnis ergibt sich über die Betrachtung der einzelnen Aktivitäten bzw. der dahinter liegenden Applikationen. Diese gibt Einblicke in z. B. die für das Experiment verwendete Software(-version) und die zwischen Einzelschritten übergebenen Daten. Letzteres ermöglicht die Erkennung von Fehlern bei der Datenübergabe bzw. -weiterverarbeitung innerhalb des Experiments.

In einigen Fällen nutzen Experimente auch Rechenressourcen. Der nächste Abschnitt beschreibt, wie man die vorhandenen Ressourcen dynamisch erweitern kann.

5.2 Dynamische Erweiterung eines Compute Clusters über multiple physische Ressourcen

Viele Anwender stehen oftmals vor dem Problem, eine für ihre Bedürfnisse geeignete Rechenressource zu finden. Die Suche nach einer solchen erfolgt meist manuell oder aber unter Zuhilfenahme eines Ressource-Brokers. Übersteigen die Anforderungen des Anwenders die Kapazitäten eines einzelnen Ressourcen-Anbieters, so scheiden dessen Ressourcen üblicherweise als ungeeignet aus. Es besteht jedoch die Möglichkeit, dass die zusammengefassten Kapazitäten mehrerer Ressourcen-Anbieter die Anforderungen erfüllen. In diesem Fall bietet der zeitweise Zusammenschluss der freien Kapazitäten der verschiedenen Betreiber eine Lösung. Der Initiator eines solches Zusammenschlusses könnte sowohl einer der Ressourcen-Betreiber als auch der Anwender selbst sein.

Im Gegensatz zur statischen Kapazitätsvergrößerung, etwa durch den Zukauf von Ressourcen, birgt die dynamische Variante wesentliche Vorteile für den Betreiber. Eine statische Vergrößerung der vorhandenen Ressourcen ist z. B. nur sinnvoll, wenn der Betreiber in absehbarer Zukunft mit einer steigenden Grundlast rechnen kann. Ein Kapazitätsausbau nur zum Abfangen von Spitzenlasten führt letztendlich zu einer Unterauslastung in Zeiten, in denen nur die Grundlast bewältigt werden muss.

Im Gegensatz hierzu ermöglicht die nur temporäre Hinzunahme von Ressourcen, i) Anfragen nach Ressourcen ausreichend befriedigen zu können und ii) die Auslastung der eigenen Ressourcen weiterhin auf einem hohen Niveau zu halten; erst wenn die eigenen Ressource nicht mehr ausreichen, werden zusätzliche ausgeliehen. Im kommerziellen Umfeld erfolgt das temporäre Hinzufügen von Kapazitäten zum Abfangen von Lastspitzen beispielsweise bei Online-Shops kurz vor Weihnachtszeit für das Jahresendgeschäft oder in Unternehmen für die Erstellung von Rechnungen auf monatlicher oder Quartalsbasis.

Bevor ein Betreiber seine Kapazitäten dynamisch durch die Hinzunahme von Ressourcen externer Anbieter erweitern kann, müssen Rahmenbedingungen für eine reibungslose

Integration geschaffen werden. Einige dieser Rahmenbedingungen sind nachfolgend erläutert.

- Die externen Ressourcen sind während der Leihzeit in das eigene Ressourcenmanagement zu integrieren, so dass der Betreiber die alleinige und vollständige Kontrolle über die Ressourcen besitzt.

- Der Betreiber muss auf die Informationssysteme der externen Anbieter frei zugreifen können, um z. B. nach verfügbaren physischen/ virtuellen Ressourcen sowie deren Preis pro Zeiteinheit suchen zu können. Die publizierten Attribute könnten sich etwa an dem Standard Grid Laboratory Uniform Environment (GLUE) (vgl. [ABE+09]) orientieren, welcher in den Grid Middlewares gLite 3, Globus Toolkit 4 und UNICORE 6 Verwendung findet.

- Die temporär hinzugenommenen Ressourcen müssen vor der Freigabe zur Nutzung vom Betreiber in dessen System integriert werden. Dies kann z. B. durch das Übertragen und Starten von speziell präparierten virtuellen Appliances auf die ausgeliehenen Ressourcen geschehen.

- Des Weiteren sind eventuell noch Anpassungen an der Netzwerkkonfiguration sowie möglichen Firewalls notwendig. Ersteres kann z. B. durch die dynamische Erzeugung eines Virtual Private Networks oder Virtual Local Area Networks (vgl. Abschnitt 3.1.1) zwischen den verschiedenen Ressourcenstandorten erfolgen.

Im folgenden Abschnitt ist mit dem Cloud Computing bzw. dessen IaaS-Angebot eine Möglichkeit für eine solche dynamische Erweiterung eigener Kapazitäten durch Fremdanbieter beschrieben.

5.2.1 Cloud Computing

Das Cloud Computing ist eine der sich derzeit schnell ausbreitenden Technologien in der IT. Der vor circa drei Jahren einsetzende Hype um Cloud Computing ist anhand der kontinuierlich steigenden Anfragenanzahl zu diesem Thema an die Suchmaschine Google belegbar (vgl. Abbildung 5.1). Das erste große Aufkommen von Suchanfragen zu diesem Thema fand zwischen August und September 2007 statt, seitdem ist ein kontinuierlicher Anstieg des erzeugten Suchverkehrs zu verzeichnen. Ähnlich zu dem in Abschnitt 5.3 vorgestellten Grid Computing mangelt es an einer allgemein akzeptierten Definition für Cloud Computing. Vaquero et. al. verglichen in [CVRML09] viele der existierenden Definitionen, um den kleinsten gemeinsamen Nenner zu identifizieren. Es trat jedoch kein einziges Merkmal in

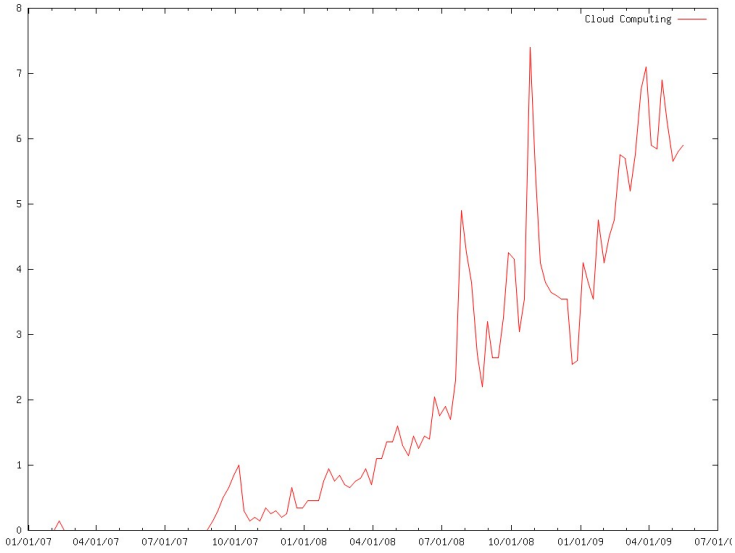


Abbildung 5.1: Google Trends-Ergebnisse Cloud Computing (Stand: 25.05.2009). Die Werte auf der y-Achse reflektieren den Quotienten aus dem Suchverkehr für den Suchbegriff zum durchschnittlichen Suchverkehr.

allen untersuchten Definitionen auf, allerdings wurden Skalierbarkeit, Virtualisierung und das pay per use-Geschäftsmodell häufig genannt.

Mit dem Erscheinen des Cloud Computing etablierten sich weitere Begriffe, die sich an den angebotenen Dienstleistungen *Infrastruktur*, *Entwicklungs- und Betriebsplattform* und *Software* orientieren. Der Einsatz von Virtualisierung ermöglicht es Anbietern, ihre freien Kapazitäten dynamisch an Kunden auszuleihen. Dies trägt aus Betreibersicht einerseits zu einer verbesserten Auslastung der vorhandenen Ressourcen und andererseits, bedingt durch das pay per use-Geschäftsmodell, zu Mehreinnahmen bei. Diese Form der Dienstleistung wird als Infrastructure as a Service (IaaS) bezeichnet und ist für Kunden über Schnittstellen wie das Application Programming Interface (API) der Amazon Web Services Elastic Compute Cloud (EC2) (vgl. [Ama11]) erreichbar. Eine IaaS Cloud formt üblicherweise einen abgeschlossenen Bereich, in den Kunden virtuelle Appliances hochladen und verwalten. Um die zeitweise Erweiterung eines Kundennetzwerkes durch in der Cloud befindliche Infrastruktur zu erlauben, gibt es bereits auf VPN-basierende Lösungen, wie Amazons Virtual Private Cloud (VPC). In einer Virtual Private Cloud gestartete virtuelle Appliances fügen sich hierbei nahtlos in das Kundennetzwerk ein. Die problemlose Integration der extern betriebenen, virtuellen Appliances in die kundeneigene Sicherheitsinfrastruktur stellt daher eine wesentliche Eigenschaft einer VPC dar. Betrachtet man die Nutzungskosten einer VPC, so setzen sich diese einerseits aus den Kosten für jede VPN-Verbindungsstunde[3] und

[3]Zeitspanne während der die VPN-Verbindung zur Nutzung bereit steht.

andererseits aus den Kosten für die über die VPN-Verbindung transferierte Datenmenge zusammen.

Neben der Bereitstellung reiner Infrastruktur können Kunden – über eine oberhalb des IaaS-Angebots liegende Abstraktionsebene – auf Plattformen wie die Google Apps Engine[4] oder Microsofts Azure[5] zurückgreifen, um auf eben diesen eigene Dienste zu implementieren. Die Implementierung auf dieser so genannten Platform as a Service-Schicht, erlaubt ein dynamisches Wachsen bzw. Schrumpfen des Dienstes basierend auf der aktuell vorliegenden Last.

Software, wie eine Textverarbeitung oder eine Tabellenkalkulation, wird bei einem Software as a Service-Angebot nicht länger auf lokalen Rechnern installiert und ausgeführt. Stattdessen greift der Nutzer auf die Software, welche sich in der Cloud befindet, bei Bedarf zu. Sind Lizenzkosten für eine Software zu entrichten, so scheinen gerade Software as a Service (SaaS)-Angebote aufgrund des pay per use-Geschäftsmodells und einer oftmals sekundengenauen Abrechnung günstiger als der Kauf einer Software-Lizenz.

5.2.2 Dynamische Erzeugung von Grid Middleware-Diensten

Mit dem IaaS-Konzept und der damit einhergehenden Möglichkeit zur bedarfsorientierten Bereitstellung von Infrastruktur ergibt sich die Frage, ob nicht auch einzelne Grid Middleware-Dienste i) in einer Cloud betreibbar sind und ii) diese bei Bedarf ebenso wachsen bzw. schrumpfen können. In diesem Zusammenhang ergibt sich unmittelbar die Frage nach Möglichkeiten zur dynamischen Bereitstellung solcher Grid-Dienste. Hierbei helfen die Betrachtungen aus den Abschnitten 3.1 und 3.2 zu den Themen Virtualisierung und Workflows.

Für den ersten Fall, den Betrieb von Grid-Diensten in einer Compute Cloud, könnten die einzelnen Grid Middleware-Dienste beispielsweise in Virtual Appliances gekapselt werden und so bei unterschiedlichen Ressourcenanbietern genutzt werden. Andersherum wären Ressourcenanbieter ebenso in der Lage, zeitgleich mehrere Grid Middlewares anzubieten, ohne dass diese sich gegenseitig beeinflussen. Dies folgt aus der Isolationseigenschaft virtuellen Maschinen. Die im zweiten Fall notwendige Dynamik könnte über die in dieser Arbeit betrachtete Workflow-gestützte Bereitstellung von Grid Middleware-Diensten erfolgen. Über diese wäre eine bedarfsorientierte Installation einzelner Dienste möglich. Der Einsatz von Workflows ermöglicht zudem einen hohen Automationsgrad und sichert die zuverlässige Reproduzierbarkeit der Ergebnisse, hier der virtuellen Appliances.

Im Grid Computing-Umfeld spielen Workflows seit längerer Zeit bereits eine Rolle. Beispielsweise stellten Buyya und Yu im Jahr 2005 in [YB05] eine Klassifikation wissen-

[4] `http://code.google.com/appengine/`

[5] `http://www.microsoft.com/windowsazure/`

schaftlicher, im Grid einsetzbarer Workflow-Systeme vor, wobei das Workflow-Design, das Workflow-Scheduling, die Fehlertoleranz und die Migration von Daten als Kriterien zur Unterscheidung dienten.

Auf der Seite der Grid-Anwender findet man meist dort Workflows, wo große, dezentral gelagerte Datenmengen verarbeitet werden müssen, wie etwa in der Hochenergiephysik und der Klimaforschung. Die Workflows dienen in diesem Kontext oft dem vereinfachten Zugriff und dem Verbergen der komplexen Transaktionen zwischen verschiedenen Diensten im Hintergrund.

Es fehlen bisher jedoch Workflows für den Bereich der dynamischen Bereitstellung von Grid Middleware-Diensten, worunter auch die Installation und Konfiguration der Dienste fallen. In Abschnitt 5.3 sind daher für die drei Grid Middlewares gLite, Globus Toolkit und UNICORE entsprechende Workflows präsentiert und somit die technische Realisierbarkeit (Proof-of-Concept) gezeigt. Unabhängig von der einfachen Bereitstellung gibt es in Grid-Initiativen jedoch auch organisatorische Anforderungen an die Dienstanbieter, die einer spontanen, dynamischen Erzeugung von Diensten entgegenstehen und eher auf die permanente Bereitstellung von Grid-Diensten bzw. -Ressourcen abzielen. Nachfolgend sind einige dieser Anforderungen am Beispiel der European Grid Initiative (EGI) bzw. der National Grid Initiative - Deutschland (NGI-DE) erläutert.

Bevor eine Grid-Ressource als vollständiges und im Produktivstatus befindliches Mitglied der European Grid Initiative gelistet wird, muss sie eine Zertifizierungsphase durchlaufen, die sich stark an der des EGEE-Projekts anlehnt. Letztere ist in [LK08] durch Liabotis beschrieben. Zuvor wird in einer national zentralen Datenbank ein Eintrag für die Grid-Ressource erzeugt, unter dem die Ressourcenadministratoren Informationen über die bereitgestellten Dienste sowie Kontaktinformationen hinterlegen. Mit dem Eintragen der Dienste werden diese automatisch in die im NGI-DE existierende und in EGI integrierte Monitoring-Infrastruktur aufgenommen, so dass in periodischen, meist stündlichen Abständen die Funktionalität der Ressource überprüft wird. Treten innerhalb einer vordefinierten Zeitspanne keine Probleme auf der Grid-Ressource auf, so wird die Zertifizierungsphase beendet und die Ressource geht in den Produktivstatus über.

Da die organisatorischen Vorgänge das Verstreichen gewisser Zeitspannen oder auch manuelle Eingriffe erfordern, kann nicht von einer unmittelbaren Verfügbarkeit der Grid-Ressourcen gesprochen werden. Um dem langlebigen Charakter einer Grid-Ressource gerecht zu werden, könnten Betreiber die minimal notwendige Menge an Grid Middleware-Diensten permanent in einer Compute Cloud betreiben und bei Bedarf weitere Ressourcen, zumeist in Form von Workernodes, dynamisch hinzufügen.

5.3 Grid Middleware

Das Konzept des Grid Computing kam in einer Zeit auf, als es das Verteilte Rechnen bereits gab. Daher gleichen sich beide Konzepte in mancher Hinsicht, beispielsweise gehen sie davon aus, dass an vielen räumlich verteilten Standorten die dort vorhandenen Ressourcen nur begrenzt durch Eigennutzung ausgelastet sind. Der Zugriff auf die freien Kapazitäten durch externe, also nicht am Standort ansässige Nutzer ist sowohl für die Ressourcenbetreiber als auch die entfernten Nutzer erstrebenswert und kommt einer Win-Win-Situation gleich: einerseits steigt die durchschnittliche Ressourcenauslastung, was im Sinne des Betreibers ist, und andererseits kann der externe Nutzer den Zugriff auf die Ressource zur Unterstützung seiner Arbeit bzw. Forschung einsetzen.

Das Aufkommen kommerzieller Rechenzentren und deren Inanspruchnahme durch das Outsourcing von IT-Infrastruktur und -Diensten (z. B. Web und E-Mail Server) führte in den letzten Jahren zum Konzept des Cloud Computing, wie es in Abschnitt 5.2.1 skizziert ist. Für die nachfolgend durchgeführte Betrachtung des Grid Computing werden allerdings ausschließlich akademische Einrichtungen als Ressourcenbetreiber angenommen. Diese Annahme spiegelt die Situation während der Entstehungsphase der ersten Grid Middlewares wider.

Bereits vor und auch während dieser Entstehungsphase gab es bei den Anforderungen, die wissenschaftliche Anwendungen an die Hardware in Form von Rechenleistung und Speicherbedarf haben, einen deutlichen Anstieg. So benötigt beispielsweise die aktuell durchgeführte Suche nach dem Higgs-Boson in der Teilchenphysik, genauer die Erzeugung von ausreichend vielen Ereignissen bei Monte Carlo-Simulationen, bis zu 200 Stunden Ausführungszeit auf 10.000 CPU-Kernen. Bis zu einem gewissen Punkt befriedigte man in der Vergangenheit die gestiegenen Anforderungen durch wenige, zentrale Rechenzentren, welche spezielle Hardware, wie etwa Supercomputer betrieben. Im Jahr 1964 nahm man beispielsweise den CDC6600 mit einer Peak Performance von ca. $3 \cdot 10^6$ Floating Point Operations per Second (FLOP) in Betrieb (vgl. [Tho65]). Zum Vergleich hierzu besitzt das im Jahr 2008 eingeweihte IBM Roadrunner-System[6] eine Peak Performance von mehr als 10^{15} FLOP. Allerdings sind solche Supercomputer in der Anschaffung als auch im Betrieb kaum für Forschungseinrichtungen finanzierbar, so beträgt der reine Anschaffungspreis für das Roadrunner-System ca. 133 Millionen US Dollar (USD) und die Leistungsaufnahme liegt bei ca. 2,35 Megawatt (vgl. [Mar08, TOP08]).

Daher suchte man nach einem anderen Weg. Statt Rechenleistung aus einzelnen, zentralen Supercomputern zu beziehen, setzt man beim Grid Computing darauf, dass sich dezentral viele Ressourcen zu einem Netz zusammenschalten und Nutzer aus diesem Netz ihren

[6] `http://www.lanl.gov/roadrunner/`

Bedarf decken können – ganz so wie es im Stromnetz praktiziert wird. Mit dieser Idee im Hinterkopf prägten Kesselman und Foster in [FK98] den Begriff eines Computational Grids wie folgt

> A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

Diese Definition wurde um eine in [Fos02] veröffentliche Prüfliste ergänzt. Die Liste führt im Wesentlichen drei Punkte auf, durch die ein Grid charakterisiert ist: 1) die Bereitstellung nicht-trivialer Dienstgüte, 2) den Einsatz von offenen und standardisierten Protokollen und Schnittstellen und 3) die Koordination von Ressourcen ohne zentrale Kontrolle.

Letztendlich führten die Überlegungen zu der in Abbildung 5.2 gezeigten Schichtenarchitektur, die sich noch heute in vielen Grid Middlewares wiederfindet. Die unterste Ebene

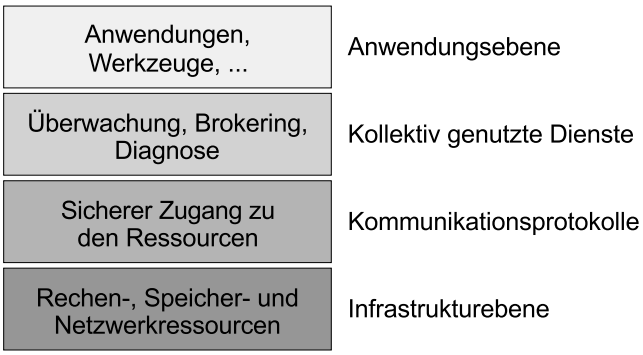


Abbildung 5.2: Schichten einer Grid Middleware, Quelle: [FK03]

enthält Mengen von Ressourcen, wie etwa Rechen- und Speicherressourcen, welche entfernten Nutzern über wohldefinierte Schnittstellen und Protokolle zur Verfügung stehen. Abhängig vom Ressourcentyp stellen die Schnittstellen unterschiedliche Funktionalität bereit: bei Rechenressourcen stehen Mechanismen zur Jobkontrolle (z. B. Starten, Anhalten und Fortsetzen) und -überwachung im Vordergrund, während bei Speicherressourcen der Fokus auf dem Einlagern und Auslesen von Dateien sowie Mechanismen für die Speicherverwaltung und -reservierung liegt.

Der Zugang zu und die Kommunikation mit den auf der Infrastrukturebene befindlichen Ressourcen erfolgt gesichert, z. B. über den weit verbreiteten Grid Security Infrastructure (GSI)-Standard (vgl. [FKTT98]). Oberhalb der Sicherungsschicht sind kollektiv genutzte Dienste anzutreffen, welche beispielsweise die Funktionalität der Ressourcen überwachen.

Die oberste Schicht, der Bereich der Grid-Anwendungen, umfasst ein breites Spektrum. Eine Übersicht über die in der European Grid Initiative registrierten Anwendungen gibt

die EGI Applications Database[7]. Diese Datenbank umfasst u. a. Anwendungen aus den Bereichen der Physik, Chemie, Mathematik und Medizin. Die Forschungseinrichtungen der zuvor erwähnten Disziplinen gruppieren sich für die Grid-Nutzung in so genannte Virtuelle Organisationen (VO), deren Konzept im folgenden Abschnitt beschrieben ist.

5.3.1 Virtuelle Organisationen

Den Begriff der virtuellen Organisation (engl.: virtual organization) führten Foster et al. im Jahr 2001 in das Grid Computing-Umfeld ein und fassten hierunter Personen und Ressourcen zusammen, die kooperativ an der Lösung einer oder mehrerer wohldefinierter Problemstellungen beteiligt sind (vgl. [FKT01]). Einige Beispiele für solche virtuellen Organisationen aus der Hochenergiephysik sind ATLAS[8], LHCb[9], CMS[10] und ALICE[11], die sich allesamt mit verschiedenen am Large Hadron Collider (LHC) durchgeführten Experimenten auseinandersetzen.

Auf die Rechen- sowie Speicherressourcen innerhalb einer virtuellen Organisation haben die Mitglieder einen durch Nutzer- und/ oder Gruppenrichtlinien (engl.: policies) einschränkbaren Zugriff. Das Anlegen solcher Gruppen sowie die weitere Verwaltung der Mitglieder erfolgt meist unter Zuhilfenahme von Diensten, wie dem Virtual Organization Membership Service (VOMS) oder dem Virtual Organization Management Registration Service (VOMRS).

Der Betrieb des VOMS-Servers erfolgt durch einen oder mehrere VO-Administratoren. Eine zweite Gruppe, die VO-Repräsentanten, ist zuständig für die VOMS-Inhalte und kann beispielsweise Rollen und Gruppen gezielt einzelnen Mitgliedern der virtuellen Organisation zuweisen. Die Nutzung des VOMS-Servers birgt neben der zentralisierten Mitgliederverwaltung einen weiteren Vorteil: der VOMS-Servers verfügt über Schnittstellen, die z. B. im Fall der Attribut-basierten Autorisierung von Bedeutung sind. Im normalen Betrieb wird der VOMS-Servers daher sowohl von VO-Mitgliedern als auch von Grid-Ressourcen kontaktiert. Bei der Erzeugung eines Stellvertreter-Zertifikats mit VOMS-Erweiterung (vgl. Listing 5.1) verbindet sich beispielsweise ein VO-Mitglied mit dem VOMS-Server. Letzterer überprüft, ob das Mitglied die angeforderten Gruppen bzw. Rollen annehmen darf und liefert im positiven Fall das Zertifikat mit entsprechender Erweiterung zurück.
In dem gezeigten Listing ist durch den Eintrag `VO: dgops` zunächst die Zugehörigkeit des Zertifikatbesitzers zur VO `dgops` bestätigt. Des Weiteren enthält die VOMS-Erweiterung

[7] `http://appdb.egi.eu/`
[8] `http://grid.racf.bnl.gov/siteAAA/VOservices/USATLASVO.html`
[9] `http://lhcb.web.cern.ch/lhcb/`
[10] `http://cms.cern.ch/iCMS/`
[11] `http://aliceinfo.cern.ch/Collaboration/index.html`

drei Attribute. Keines dieser Attribute räumt dem VO-Mitglied weitere Rechte einräumen, da es in jeder der Gruppen die Rolle `NULL` zugewiesen bekommen hat.

```
1 === VO dgops extension information ===
  VO         : dgops
3 subject    : /C=DE/O=GermanGrid/OU=TU-Dortmund/CN=Stefan Freitag
  issuer     : /O=GermanGrid/OU=FZK/CN=host/dgrid-voms.fzk.de
5 attribute  : /dgops/Role=NULL/Capability=NULL
  attribute  : /dgops/admin/Role=NULL/Capability=NULL
7 attribute  : /dgops/member/Role=NULL/Capability=NULL
  timeleft   : 11:59:53
9 uri        : dgrid-voms.fzk.de:15027
```

Listing 5.1: VOMS-Erweiterung als Teil eines Stellvertreter-Zertifikats

Aktuell findet sich der VOMS-Server noch bei vielen virtuellen Organisationen des LHC Computing Grid wieder, allerdings deutet sich dort bereits ein Übergang hin zu Verwendung des VOMRS-Servers zur Mitgliederverwaltung an.

Bei der Konzeption des Virtual Organization Management Registration Service berücksichtigte man einige Schwächen des Virtual Organization Membership Service. So erhöhte man z. B. bei der Nutzerregistrierung und -zulassung den Automationsgrad. Des Weiteren ist der Umfang der persistent gespeicherten Informationen für VO-Mitglieder erweiterbar worden und kann nun u. a. die postalische Anschrift sowie Informationen zur telefonischen Erreichbarkeit umfassen. Im Gegensatz zu VOMS ist ebenso die Speicherung von mehr als einem X.509-Zertifikat je VO-Mitglied möglich. Die Abbildung 5.3 zeigt hierzu beispielhaft die im VOMRS der D-Grid VO `dgops` erfassten Attribute der Mitglieder. Diese umfassen neben dem Vor- und Nachnamen des Mitglieds, ebenso dessen Institut inklusive der vollständigen Anschrift sowie die Nationalität.

Nach dieser kurzen Einführung in das Konzept der virtuellen Organisationen sind in den folgenden Abschnitten die Grid Middlewares gLite, Globus Toolkit und UNICORE sowie die dafür entwickelten Workflows vorgestellt.

5.3.2 gLite Middleware

Die anfängliche Version 1.0 der gLite Middleware wurde im April des Jahres 2005 veröffentlicht. Vor dem Einsatz dieser Middleware im EGEE-Projekt setzte man dort auf andere Middlewares, wie die des LHC Computing Grid (LCG) und des European DataGrid Project (EDG). Mit dem Ziel eines reibungslosen Übergangs von der zuletzt verwendeten LCG zur gLite Middleware führte man im EGEE-Projekt die Komponenten beider Middlewares zum gLite Release 3.0 zusammen.

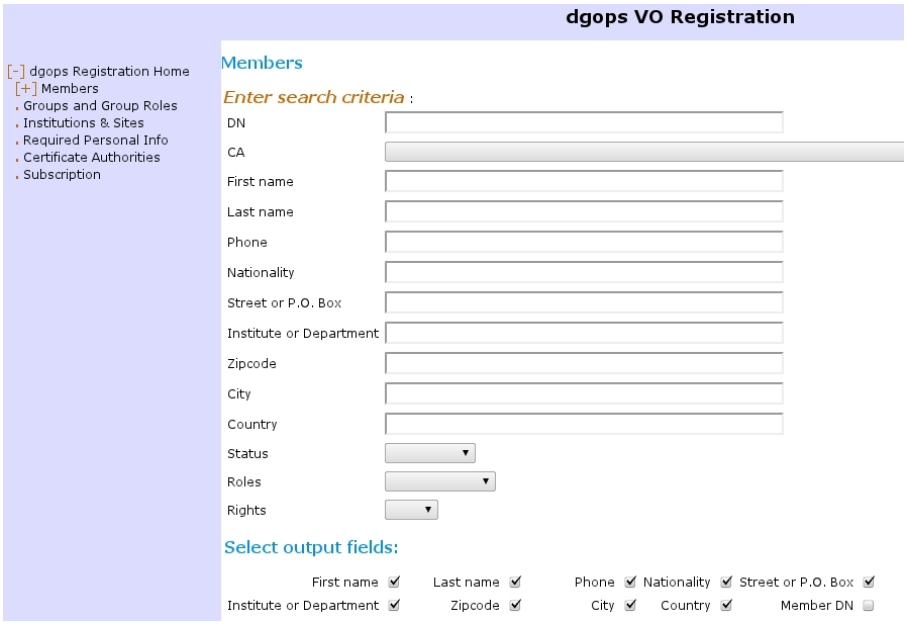


Abbildung 5.3: Suchmaske des Virtual Organization Management Registration Service

Die in der ersten Version der gLite Middleware enthaltenen Komponenten und ihre Einordnung in das auf Seite 105 vorgestellte Schichtenmodell zeigt Abbildung 5.4. Auffällig bei der Betrachtung der Komponenten ist das Fehlen eines Speicherdienstes bzw. -elements. Dieses kam erst in einer späteren Version der Middleware hinzu.

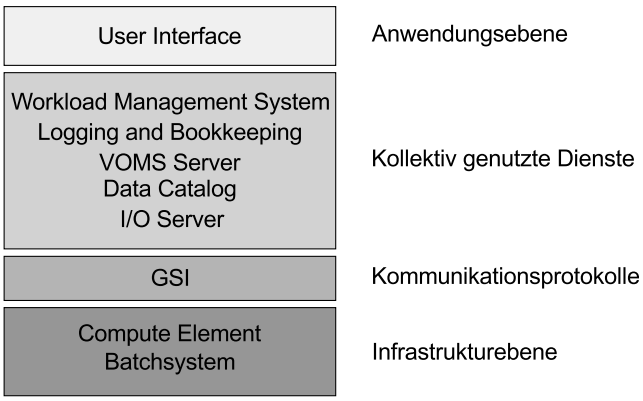


Abbildung 5.4: Komponenten der gLite Middleware Version 1.0

Derzeit läuft auf vielen Grid-Ressourcen die gLite Middleware in Version 3.2. Diese baut im Gegensatz zum Vorgänger nicht mehr auf dem Betriebssystem Scientific Linux 4, sondern auf Scientific Linux 5 bzw. Debian Linux auf. Außerdem wurden der Middleware nach und nach weitere Komponenten hinzugefügt bzw. obsolete Komponenten aus ihr

entfernt. So wurde etwa der I/O Server entfernt und das CREAM CE als neue Compute Element-Variante hinzugefügt.

Eine minimale Konfiguration einer gLite Grid-Ressource umfasst genau einen lokalen Berkeley Database Information Index (BDII)[12], einen Accounting-Client sowie ein Compute Element inklusive Batchsystem und/ oder ein Storage Element (SE) (vgl. Abbildung 5.5). Weiterhin werden durch Liabotis in [LK08] mit 8 CPU-Kernen bzw. 1 TByte Massenspei-

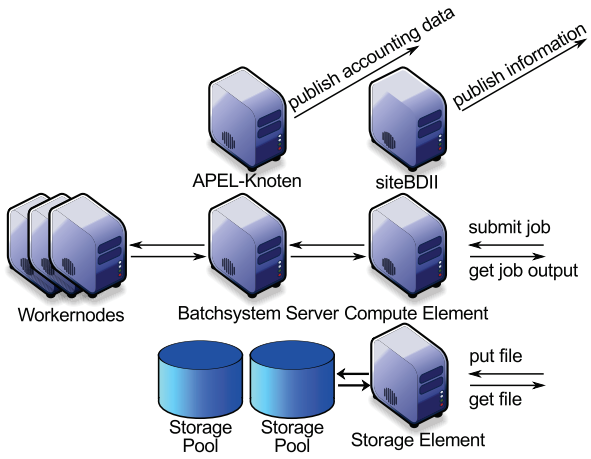


Abbildung 5.5: Komponenten einer minimalen gLite Grid-Ressource

cher die Mindestgrößen des jeweiligen Ressourcentyps definiert. Ein Workflow für die Installation dieser minimal erforderlichen Dienste ist in Abbildung 5.6 gezeigt. Bevor die Installation der einzelnen Dienste beginnt, sind in der Aktivität `Define Resource-specific Information` die Inhalte der drei Dateien `site-info.def`, `users.conf` und `groups.conf` festzulegen. Diese Dateien werden in den Folgeaktivitäten wie in Abschnitt 2.2.1 beschrieben prozessiert. Eine Untermenge der vier Dienste siteBDII, CREAM CE, APEL-Knoten und Batchsystem ist parallel installierbar. Lediglich das Batchsystem muss vor dem Compute Element installiert werden. Diese Parallelität spiegelt sich in der UND-Verknüpfung an Gate 1 des Workflows wieder. Des Weiteren laufen die einzelnen Aktivitäten nach deren Ausführung an Gate 2 wieder zusammen und erst nachdem alle Aktivitäten abgeschlossen sind, wird die manuelle Registrierung der Dienste in der Aktivität `Registration` angestoßen.

Die folgenden Abschnitte beschreiben detailliert die Workflow-basierte Installation und die Konfiguration verschiedener gLite Middleware-Komponenten, u. a. auch der der in Abbildung 5.5 gezeigten.

Vielen dieser Komponenten gemein ist die Hinterlegung von Informationen zu den erfor-

[12] Der lokale BDII ist nachfolgend als siteBDII bezeichnet.

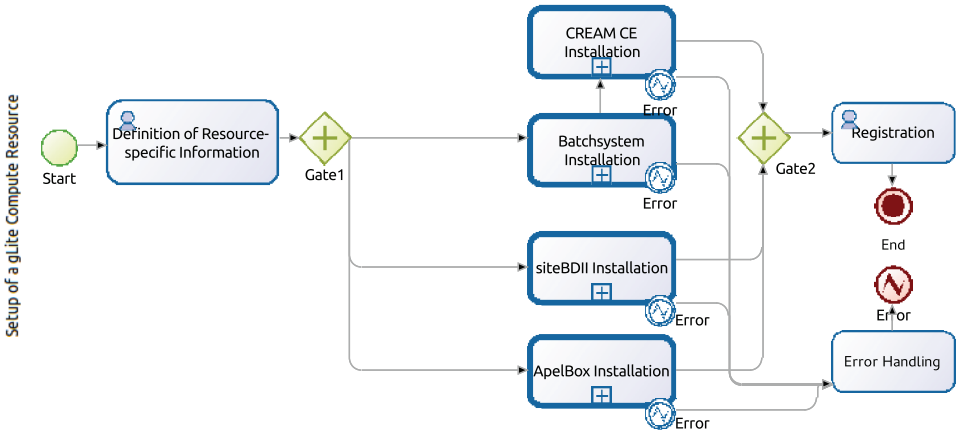


Abbildung 5.6: Workflow für die Einrichtung einer minimalen gLite Grid-Ressource

derlichen Software-Repositories für die Middleware und die Zertifikate der Certification Authorities. Diesbezügliche Workflows sind direkt im Anschluss an diesen Abschnitt vorgestellt. Weiterhin sind Workflows für die Installation und Konfiguration verschiedener gLite Middleware Komponenten, wie siteBDII (vgl. Abschnitt 5.3.2.2), CREAM Compute Element (vgl. Abschnitt 5.3.2.3), Workernodes (vgl. Abschnitt 5.3.2.5) und VOBox (vgl. Abschnitt 5.3.2.7), vorgestellt.

5.3.2.1 Konfiguration der Software-Repositories

Alle Komponenten der gLite Middleware sind aus Software-Repositories installierbar. Neben der manuellen Einrichtung der Repository-Konfiguration im Verzeichnis `/etc/yum.repos.d/` kann im Fall von gLite Version 3.2 diese Konfiguration direkt als Datei von einer Webseite[13] heruntergeladen werden. Die Datei enthält die Konfigurationsinformationen für drei separate Repositories: i) für die Pakete des offiziellen Releases, ii) für Aktualisierungen und iii) für Pakete aus externen Quellen. Das Listing 5.2 zeigt beispielhaft den Inhalt dieser Datei für einen gLite siteBDII.

```
1 [glite-BDII]
  name=gLite 3.2 BDII
3 baseurl=http://glitesoft.cern.ch/EGEE/gLite/R3.2/glite-BDII/sl5/x86_64/
      RPMS.release/
  gpgkey=http://glite.web.cern.ch/glite/glite_key_gd.asc
5 gpgcheck=1
  enabled=1
7
  [glite-BDII_updates]
9 name=gLite 3.2 BDII updates
```

[13] `http://grid-deployment.web.cern.ch/grid-deployment/glite/repos/3.2/`

```
baseurl=http://glitesoft.cern.ch/EGEE/gLite/R3.2/glite-BDII/sl5/x86_64/
    RPMS.updates/
gpgkey=http://glite.web.cern.ch/glite/glite_key_gd.asc
gpgcheck=1
enabled=1

[glite-BDII_ext]
name=gLite 3.2 BDII external
baseurl=http://glitesoft.cern.ch/EGEE/gLite/R3.2/glite-BDII/sl5/x86_64/
    RPMS.externals/
gpgcheck=0
enabled=1
```

Listing 5.2: Repository-Konfiguration für einen gLite siteBDII

Für die gLite Middleware Version 3.1 fehlt die Möglichkeit zum Download der Repository-Konfiguration, daher ist sie von Hand bzw. über den in Abbildung 5.7 gezeigten Workflow zu erzeugen. Übrigens ist dieser Workflow auch mit der Version 3.2 verwendbar.

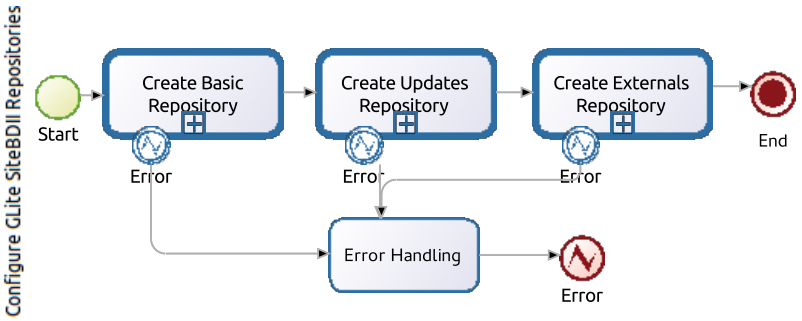


Abbildung 5.7: Workflow zur Konfiguration der gLite siteBDII Repositories

Die drei Aktivitäten `Create Basic Repository`, `Create Updates Repository` und `Create Externals Repository` des Workflows rufen den selben Subflow `YUM Create Repository` (vgl. Abbildung 4.3(a)) auf, jedoch mit unterschiedlichen Argumenten. Der Subflow erzeugt während der Ausführung der Aktivität `Create Repository` (vgl. Listing 4.5) aus den übergebenen Argumenten `$file_Name`, `$repository_Name`, `$repository_Description` und `$base_URL` die Repository-Informationen im Verzeichnis `/etc/yum.repos.d/`.

Ein weiteres Software-Repository beinhaltet die Zertifikate der durch die European-Policy Management Authority for Grid Authentication (EUGridPMA) akkreditierten Certification Authorities. Die Installation dieser Zertifikate kann je nach Middleware-Dienst notwendig sein und lässt sich in drei Schritte zerlegen: i) das Hinterlegen von Informationen zu dem YUM-Repository, aus dem die Zertifikate eingespielt werden, ii) die Aktualisie-

rung der Repository-Informationen und iii) die Installation des Metapakets `lcg-CA` bzw. nach der Bildung der European Grid Initiative des Pakets `ca-policy-egi-core`. Diese Schritte wurden zu Aktivitäten des in Abbildung 5.8 dargestellten Workflows, wobei die Schritte i) und ii) in die Aktivität `Configure Software Repository` zusammengefasst sind. Der dritte Schritt, die Installation des Metapakets, erfolgt wie in Abschnitt 4.2

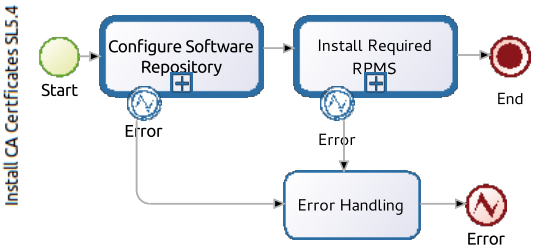


Abbildung 5.8: Workflow zur Installation der Certification Authority-Zertifikate

beschrieben durch den Aufruf des Paketmanagers YUM.

5.3.2.2 siteBDII

Der siteBDII bildet die Grid Site-zentrale Komponente des auf dem Lightweight Directory Access Protocol basierenden Informationsdienstes der gLite Middleware. Die einzelnen gLite-Komponenten einer Grid Site verfügen über einen Information Provider, der Informationen über die Komponente einsammelt, aggregiert und in das LDAP Data Interchange Format (LDIF) überführt. Diese Informationen beziehen sich bei einem Compute Element u. a. auf die Liste der unterstützten, virtuellen Organisationen und die Füllstände der einzelnen Queues im Batchsystem. In periodischen Abständen kontaktiert der siteBDII die Information Provider der einzelnen Komponenten und nimmt seinerseits ebenso eine Aggregation vor. Somit verfügt der siteBDII über alle relevanten Informationen zur Grid Site. In dem hierarchisch aufgebauten Informationssystem fragt ein TopBDII[14] die ihm bekannten siteBDII ab, wodurch er über die Informationen aller angeschlossenen Grid Sites verfügt. Ein Nutznießer dieses Vorgehens ist das Workload Management System (WMS), welches aufgrund der Informationen im TopBDII die Zuweisung von Grid-Jobs zu den einzelnen Sites bzw. Ressourcen vornimmt. Weiterhin haben auch Grid-Anwender die Möglichkeit, direkt per LDAP-Anfrage den TopBDII auszulesen und die Information zu verwenden.

Ein Workflow für die Installation und Konfiguration eines siteBDII ist in Abbildung 5.10 dargestellt. Mit der Ausführung der ersten Aktivität des Workflows wird der automatische Start von Betriebssystem-nahen Diensten konfiguriert. Hierzu ruft die Aktivität `Configure Services` den Subflow `Configure SL 5.4 VM Services` (vgl. Abbil-

[14] Manchmal auf TopLevel BDII genannt.

dung 5.9) auf. Während des Subflows wird beispielsweise der Network Time Protocol-Dienst aktiviert und die eigenständige Aktualisierung von Software-Paketen abgeschaltet. Dieser Vorgang wiederholt sich ebenso wie der Folgeschritt `Configure GLite Software-Repository`, der, wie in Abschnitt 5.3.2.1 beschrieben, die Repository-Informationen im System hinterlegt, bei der Installation der anderen, später vorgestellten gLite Middleware-Komponenten entsprechend.

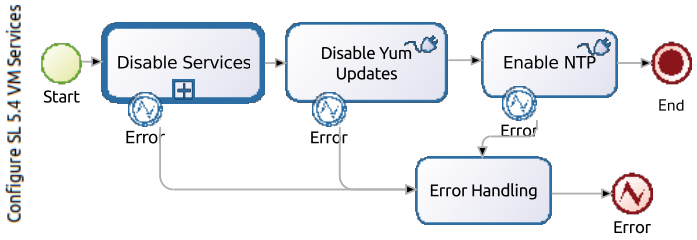


Abbildung 5.9: Workflow zur Konfiguration Betriebssystem-naher Dienste

Im Anschluss daran erfolgt in den Schritten `Install Yaim for SBDII` und `Install The Required GLite RPMS` erst die Installation der erforderlichen YAIM-Pakete und dann aller weiteren für den siteBDII erforderlichen Software-Pakete. Beide Aktivitäten nutzen hierzu den Subflow `YUM Install Package` (vgl. Abbildung 4.3(b)) unter Verwendung unterschiedlicher Argumente. Die Aktivitäten `Prepare users.conf`

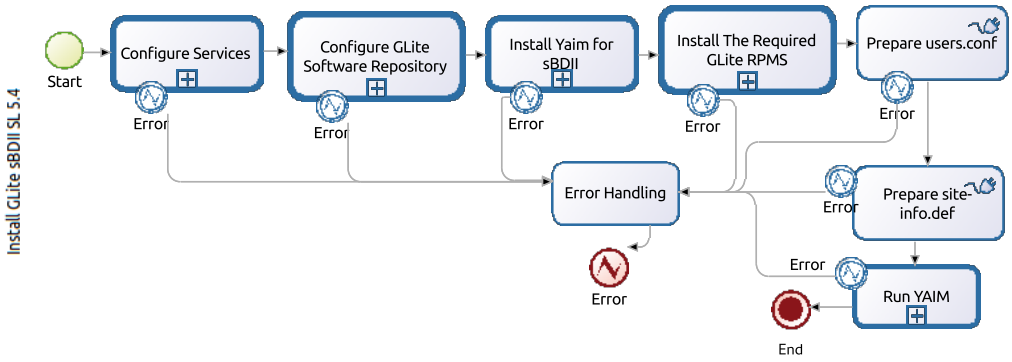


Abbildung 5.10: Workflow zur Installation und Konfiguration des gLite siteBDII

und `Prepare site-info.def` kopieren die von YAIM benötigten Konfigurationsdateien `site-info.def` und `users.conf` in das Verzeichnis `/opt/glite/yaim/-etc/`, wobei der Nutzer die Lage dieser beiden Dateien im Dateisystem initial dem Workflows zuführen muss. In Listing 5.3 ist beispielhaft die für das Kopieren der Datei `users.-conf` vorhandene Applikation abgebildet.

```
1 def String command = "cp ${path2users_conf} /opt/glite/yaim/etc/users.
    conf"
def Process proc = command.execute()
```

```
3 proc.waitFor()
```

Listing 5.3: Kopieren der Datei `users.conf` (Groovy-Applikation)

Abschließend erfolgt in der Aktivität `Run YAIM` die Ausführung des YAIM-Werkzeugs, welches die Konfiguration der siteBDII-Dienstes durchführt.

5.3.2.3 CREAM Compute Element

Ein Compute Element bildet für entfernte Nutzer bei der Inanspruchnahme von Rechenleistung den Zugangspunkt zur Grid-Ressource und ist daher u. a. auch für die Nutzerauthentifikation zuständig.
Das CREAM Compute Element löst mit der Veröffentlichung der gLite Version 3.2 das bis dahin produktiv genutzte LCG Compute Element ab und bietet die Job-Einreichung über die so genannte Computing Resource Execution and Management (CREAM)-Schnittstelle an. Diese erlaubt im Vergleich zu dem LCG Compute Element neben der Einreichung über das Workload Management System auch ein direktes Einreichen der Jobs am Compute Element.

Intern transformiert das Compute Element den erhaltenen Job in eine vom Batchsystem unterstützte Sprache und nimmt zudem eine Abbildung des Grid-Nutzers auf ein lokal existierendes Nutzerkonto vor. Letzteres ist erforderlich, da das Batchsystem nur die Jobs lokal bekannter Nutzer prozessieren kann. Nach der erfolgreichen Transformation und Abbildung übergibt das Compute Element den Job an den Batchsystem Server zur eigentlichen Abarbeitung.

Einen Workflow zur Installation und Konfiguration eines CREAM CEs zeigt Abbildung 5.11. Die ersten vier Aktivitäten des Workflows (`Configure Services`, `Configure GLite Software Repository`, `Install Yaim for CREAM CE` und `Install The Required GLite RPMS`) verlaufen wie in Abschnitt 5.3.2.2 für den

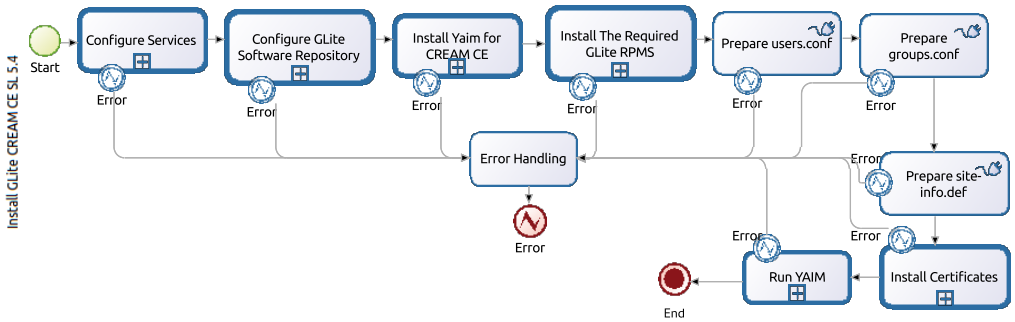


Abbildung 5.11: Workflow zur Installation und Konfiguration eines gLite CREAM Compute Elements

sBDII beschrieben: nach der Konfiguration der Betriebssystemdienste werden Repository-Informationen im System hinterlegt und die notwendigen Software-Pakete über die Installation des Metapakets `glite-CREAM` installiert.
Für die Konfiguration des Compute Elements wird neben den Dateien `users.conf` und `site-info.def` auch die Datei `groups.conf` benötigt. Diese enthält Informationen zur Abbildung der Grid-Nutzer auf die lokalen Nutzerkonten und wird in der Aktivität `Prepare groups.conf` von einer vom Workflow-Ausführenden zu spezifizierenden Stelle im Dateisystem ebenfalls in das Verzeichnis `/opt/glite/yaim/etc/` kopiert. Bevor das YAIM-Werkzeug durch die letzte Aktivität des Workflows aufgerufen wird (vgl. Listing 5.4), erfolgt die Installation der European Grid Initiative Trustanchor, sowie des Host-Zertifikats und -Schlüssels durch die Aktivität `Install Certificates` bzw. den dahinterliegenden Subflow.

```
1 def String command = "./yaim -c -s ${path_to_site_info_def} -n ${
      node_type}"
  def proc = command.execute(null, new File("/opt/glite/yaim/bin"))
3 proc.waitFor()
```

Listing 5.4: Run YAIM Script (Groovy-Applikation)

Der Aktivität `Run YAIM` werden von außen die Argumente `path_to_site_info_def` und `node_type` zugeführt, wobei der Pfad zur Datei `site-info.def` den Standardwert `/opt/glite/yaim/etc/site-info.def` besitzt. Anhand des Wertes von `node_type` führt YAIM die dienstspezifische Konfiguration durch.

5.3.2.4 Batchsystem Server

Die gLite Middleware-Distribution bringt selbst eine Batchsystem-Software mit. Derzeit besteht die Software aus der Kombination von TORQUE und Maui, wie sie in Abschnitt 4.8 vorgestellt ist. In Abbildung 5.12 ist ein Workflow für die Installation und Konfiguration des gLite Batchsystem Servers gezeigt. Die Bedeutung der einzelnen Aktivitäten ist unverändert zu den bereits in den beiden vorherigen Abschnitten betrachteten.
Im Vergleich mit dem in Abschnitt 4.8 vorgestellten Workflow für den TORQUE Server (vgl. Abbildung 4.24) ist festzustellen, dass hier die Aktivitäten zur Konfiguration von TORQUE sowie Maui völlig fehlen. Die Einrichtung der einzelnen Queues des Batchsystems, der Datei `nodes`, der Datei `server_name` sowie von Maui erfolgt durch YAIM während der Ausführung der Aktivität `Run YAIM`. Hierzu wird die Datei `site-info.def` benötigt, da ebendiese beispielsweise die anzulegenden Queues sowie die zugehörigen Access Control Lists (ACLs) enthält.

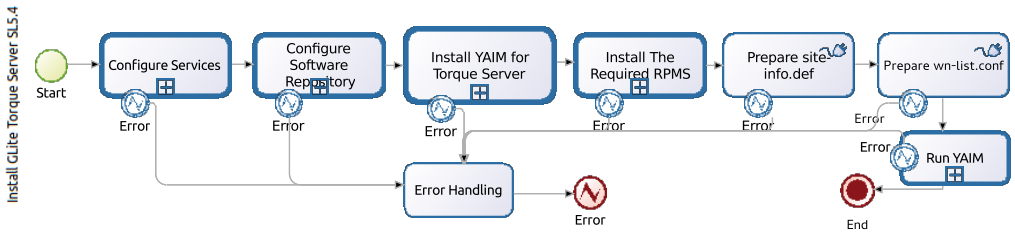


Abbildung 5.12: Workflow zur Installation und Konfiguration des gLite TORQUE Servers

5.3.2.5 Rechenknoten

Neben dem Vorhandensein der Batchsystem-Software sind auf einem Rechenknoten zur Unterstützung der gLite Middleware noch weitere Anpassungen erforderlich. Diese Anpassungen umfassen u. a. die Installation von Software für den gesicherten Datentransfer oder den Zugriff auf Storage Elements sowie das Setzen von VO-spezifischen Umgebungsvariablen. Letztere sind in der Datei `site-info.def` definiert und mit Werten zu versehen.
Wie zuvor angedeutet, müssen auf den Rechenknoten für jede virtuelle Organisation lokale Nutzerkonten eingerichtet werden, auf die das Compute Element später die Grid-Anwender abbilden kann. Die hierfür erforderlichen Informationen sind in den beiden Dateien `users.conf` und `groups.conf` abgelegt. Das Ablegen dieser Dateien in das Verzeichnis `/opt/glite/yaim/etc/` erfolgt mittels der Aktivitäten `Prepare users.conf` und `Prepare groups.conf` des in Abbildung 5.13 gezeigten Workflows.
Des Weiteren sind vom Anwender die Speicherorte der bereits ausgefüllten Dateien `wn-list.conf` und `site-info.def` zu benennen. Diese Dateien werden in separaten Aktivitäten ebenso in das Verzeichnis `/opt/glite/yaim/etc/` kopiert. Nach der Instal-

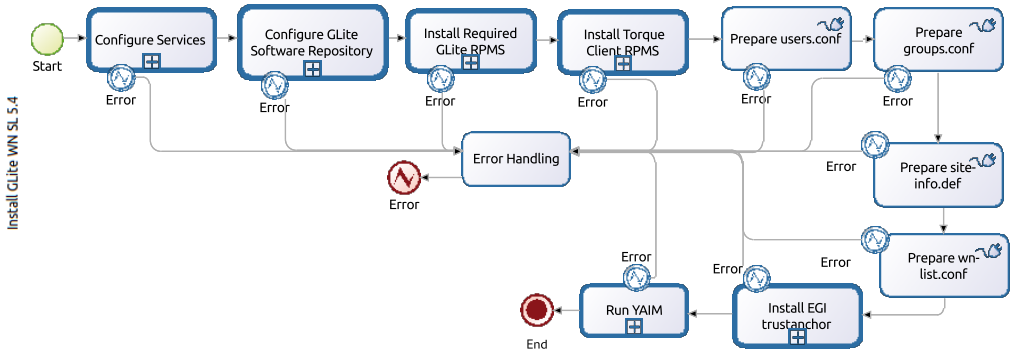


Abbildung 5.13: Workflow zur Installation und Konfiguration eines gLite Worker Node

lation der European Grid Initiative Trustanchor ruft die letzte Aktivität über einen Subflow das YAIM-Werkzeug auf, welches die Konfiguration der gLite Middleware auf dem Rechenknoten durchführt.

5.3.2.6 APEL-Knoten

Der APEL-Knoten ist erst seit Version 3.2 ein Bestandteil der gLite Middleware. In Version 3.1 realisierte die MonBox (Monitoring Box) die Aufgaben des APEL-Knotens. Auf der MonBox liefen hierzu mit dem Relational Grid Monitoring Architecture (R-GMA) Server und Accounting Processor for Event Logs (APEL) Server zwei Applikationen, wobei R-GMA für die Publikation der durch APEL gesammelten Accounting-Daten verantwortlich war. Auf dem APEL-Knoten übernimmt nun Apache ActiveMQ[15] die Aufgabe des R-GMA-Dienstes.

Das Einsammeln der Accounting-Daten ermöglicht insbesondere die Auskunft über die Nutzung der Batchsysteme einer Grid Site bzw. eines ganzen Grids durch die verschiedenen virtuellen Organisationen. Die Visualisierung der aggregierten Daten erfolgt üblicherweise über ein Portal, etwa das European Grid Initiative Accounting-Portal[16]. Die Abbildung 5.14 zeigt mit der Aufteilung der in der European Grid Initiative verfügbaren CPU-Stunden auf die vier am CERN beheimateten Experimente (ATLAS, ALICE, CMS und LHCb) einen Ausschnitt des Portals.

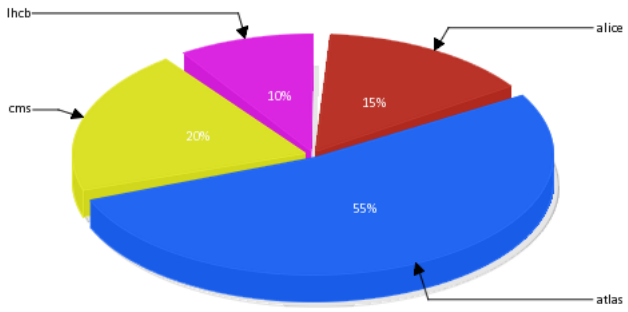


Abbildung 5.14: Verteilung der EGI CPU-Stunden auf die vier Experimente ATLAS, ALICE, CMS und LHCb (Stand: 28.12.2011)

Die Installation der Dienste des APEL-Knotens erfolgt gLite-typisch unter Verwendung des YAIM-Werkzeugs, wobei der in Abbildung 5.15 gezeigte Workflow genutzt werden kann. Dieser enthält nur Aktivitäten bzw. Subflows, die alle bereits in vorherigen Abschnitten vorgestellt wurden, was auf einen hohen Wiederverwendungsgrad einmal erstellter Subflows bei der gLite Middleware hindeutet.
Vom Workflow-Ausführenden ist lediglich die Datei `site-info.def` bereitzustellen. Aus ihr gewinnt YAIM Informationen über die vorhandenen Batchsysteme, für die entsprechende Accounting-Daten zu erzeugen bzw. zu publizieren sind.

[15] `http://activemq.apache.org/`
[16] `http://www3.egee.cesga.es/gridsite/accounting/CESGA/egee_view.php`

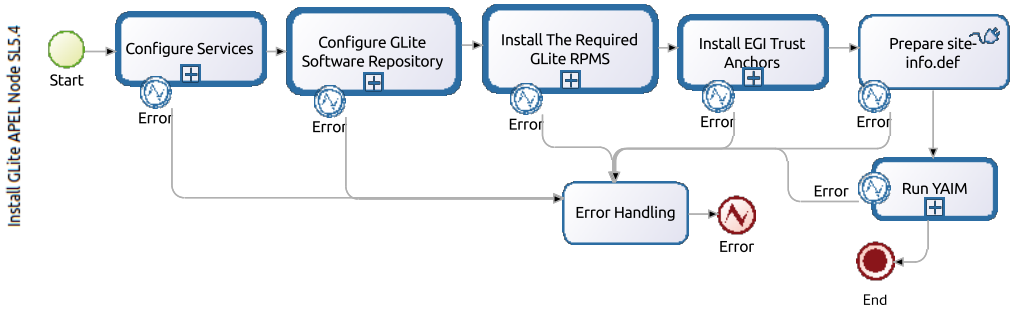


Abbildung 5.15: Workflow zur Installation und Konfiguration eines APEL-Knotens

5.3.2.7 Virtual Organization Box

Neben der Möglichkeit Rechen-Jobs über die Middleware-Schnittstellen auf entfernte Grid-Ressourcen zu submittieren, erfordern einige virtuelle Organisationen die kontinuierliche Ausführung von VO-eigenen Diensten bzw. Agenten auf ebendiesen Grid-Ressourcen. Beispielsweise erfolgt so die Ressourcenüberwachung der virtuellen Organisation ALICE mittels der Software Monitoring Agents using a Large Integrated Services Architecture (MonALISA) (vgl. [LVC+09]).

Zur besseren Isolation dieser Dienste und Agenten von anderen Komponenten der Grid-Ressource, ist der Betrieb auf separaten Knoten, den Virtual Organization Boxes, empfohlen. Der Zugang zu einer Virtual Organization Box erfolgt über eine GSI-basierte SSH-Verbindung und sollte zudem stark beschränkt sein, z. B. auf die Software Manager der virtuellen Organisation.

Ein Workflow zur Installation einer Virtual Organization Box ist in Abbildung 5.16 gezeigt. Im Gegensatz zum Workflow für den siteBDII (vgl. Abbildung 5.10) entfällt hier die separate Installation der YAIM-Pakete. Diese werden über Abhängigkeiten, die im Metapaket `glite-VOBOX` definiert sind, während der Ausführung der Aktivität `Install The Required GLite RPMS` mitinstalliert.

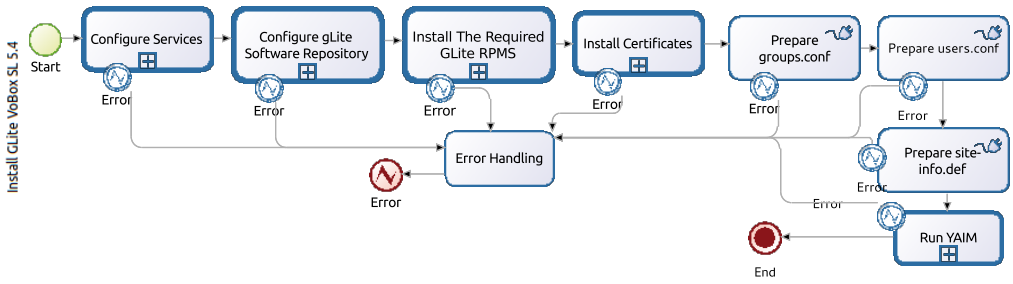


Abbildung 5.16: Workflow zur Installation und Konfiguration einer gLite VOBox

Nachdem die gLite Software über die Paketverwaltung eingespielt ist, übernimmt die darauf folgende Aktivität `Install Certificates` über einen Subflow (vgl. Abbildung 5.17) die Installation des Host-Zertifikats und -Schlüssels sowie die Installation der Zertifikate der in der European Grid Initiative akzeptierten Certificate Authorities.

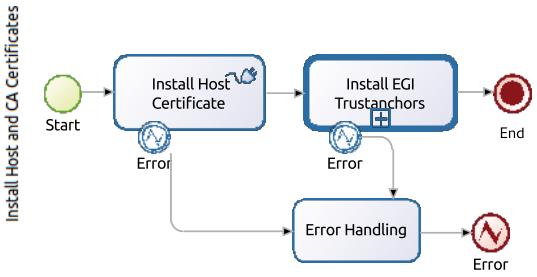


Abbildung 5.17: Workflow zur Installation des Host-Zertifikats bzw. -Schlüssels und der EGI Trustanchor

Vor der Konfiguration der gLite Software im Schritt `Run YAIM` werden noch die Dateien `groups.conf` und `site-info.def` mittels zweier Aktivitäten an entsprechender Stelle im Dateisystem abgelegt.

In den vorherigen Abschnitten sind Workflows für Dienste der gLite Middleware beschrieben, die mit der Bereitstellung von Rechenleistung zusammenhängen. Wie in Abbildung 5.5 zu sehen, gibt es ebenso Middleware-Dienste, über die Grid-Anwender auf Massenspeicher zugreifen können. Da für diese Dienste eine analoge Vorgehensweise anwendbar ist, ist auf eine Erstellung von Workflows verzichtet worden. Stattdessen werden nachfolgend die zentralen Komponenten der gLite Middleware dargestellt.

5.3.2.8 Zentrale Komponenten

Neben den Komponenten einer gLite Grid-Ressource beinhaltet die Middleware weitere zentrale Komponenten (vgl. Abbildung 5.18), die mindestens einmal in einem Grid vorkommen müssen. Einige dieser Komponenten sind, bis auf die bereits in Abschnitt 5.3.1 vorgestellten VOMS- bzw. VOMRS, nachfolgend beschrieben.

- Der TopBDII bildet in dem hierarchisch aufgebauten Informationssystem der gLite Middleware das Element am oberen Ende der Hierarchie. Er kontaktiert in periodischen Abständen eine vordefinierte Menge von siteBDIIs der verschiedenen Grid Sites und liest die dort bereitgestellten Informationen aus. Diese Informationen stehen nach einer Aggregation u. a. dem Workload Management System für den nachfolgend erläuterten Match making-Prozess zur Verfügung.

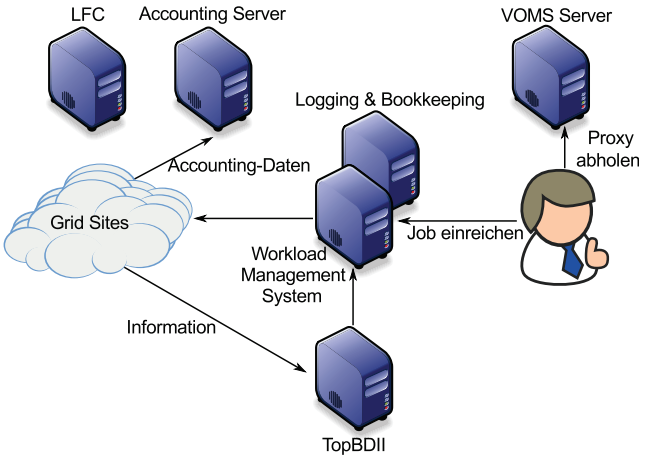


Abbildung 5.18: Zentrale Komponenten eines gLite-basierten Grids

- Eine der Hauptaufgaben des Workload Management Systems besteht in der Durchführung des Match making zwischen eingehenden Rechen-Jobs und den dem Workload Management System bekannten Grid-Ressourcen. Sofern eine nicht-leere Menge an geeigneten Ressourcen für einen Job ermittelt ist, leitet das Workload Management System den Job zur Ausführung an das entsprechende Compute Element weiter. Bei der Wahl des Compute Elements spielen die Auslastung als auch die im Job kodierten Datenabhängigkeiten eine wesentliche Rolle.
  Zudem können Nutzer über das Workload Management System Einfluss auf die Jobausführung (Abbrechen des Jobs, Abholen der Ergebnisse, . . .) nehmen (vgl. [PP08]).

- Der Logging and Bookkeeping (LB)-Dienst speichert kurzfristige Informationen über Jobs, die durch das Workload Management System an die Grid-Ressourcen geleitet wurden (vgl. [ŠSD+10]). Diese Informationen helfen den Grid-Nutzern bei der Verfolgung des Job-Status und finden u. a. im Grid Real-Time Monitor[17] Verwendung.

- Zur Herstellung einer gewissen Redundanz ist es möglich eine Datei auf mehrere Storage Element zu replizieren. Da jedes Storage Element der Replik im lokalen Dateisystem einen eigenen Namen zuweist, wird der LCG File Catalog (LFC) als übergeordnete Instanz benötigt, die alle Repliken für den Grid-Nutzer wieder unter einem einzigen logischen Namen verfügbar macht.

- Eine weitere zentrale Komponente ist der Accounting-Dienst. Die von den einzelnen Grid-Nutzern in Anspruch genommene Rechenleistung wird auf jeder Grid-Ressource durch den APEL-Dienst (vgl. Abbildung 5.3.2.6) ermittelt und an eine zentrale Accounting Server-Instanz übertragen.

[17] `http://rtm.hep.ph.ic.ac.uk/`

Für diese zentralen Komponenten der gLite Middleware können analog zu den Komponenten einer Grid Site ebenso Workflows für die automatische Installation und Konfiguration erstellt werden. Dieser Punkt stellt einen Aspekt für zukünftige Arbeiten bzw. Erweiterungen dar.

5.3.3 Globus Toolkit Middleware

Die Globus Toolkit-Middleware ist, wie auch die gLite Middleware, ein Open Source-Projekt. Der Hauptanteil der Weiterentwicklung erfolgt durch die Globus Alliance[18], zu deren Mitgliedern u. a. das Argonne National Laboratory (ANL), die University of Chicago, die University of Edinburgh und das National Center for Supercomputing Applications zählen. Nach der Veröffentlichung des ersten Releases der Middleware im Jahr 1998 entwickelte sich das Toolkit schnell weiter und liegt Ende des Jahres 2011 in Version 5.0 vor.

Aufgrund eines veränderten Dienstangebots – Version 5.0 fehlt im Vergleich zur Vorgängerversion z. B. eine Webservice-Schnittstelle zur Job-Einreichung – nutzen viele Anwender in Deutschland noch Version 4.0.x der Middleware. Daher sind nachfolgend Workflows für die Installation und Konfiguration von sowohl Globus Toolkit 4.0.8 als auch Globus Toolkit 5.0 vorgestellt.

5.3.3.1 Globus Toolkit 4

Die letzte Version, Version 4.0.8, des Globus Toolkit 4.0 wurde im Dezember 2008 veröffentlicht und erreichte das End-Of-Life bereits Ende des Jahres 2009. Damit endete nach vier Jahren die Unterstützung der Entwickler (inklusive der Bereitstellung von Security Updates) für das Globus Toolkit 4.0.

Das Globus Toolkit 4.0 besitzt wie sein Nachfolger einen modularen Aufbau, der aus den fünf Säulen Sicherheit, Datenmanagement, Ausführungsmanagement, Informationssystem und Laufzeitumgebung besteht (vgl. Abbildung 5.19).

Im Bereich des Ausführungsmanagements sind die WS GRAM- und die GRAM-Schnittstelle von besonderem Interesse, da hierüber die Einreichung und Überwachung von Rechen-Jobs erfolgt. Ein möglicher Datentransfer zu einer bzw. von einer Globus Toolkit-Ressource ist auf unterer Ebene mittels des GridFTP (vgl. [ABB+03]) möglich. Dieser Dienst basiert auf dem gängigen File Transfer Protocol (FTP), welches um Grid-spezifische Mechanismen zur Sicherung des Kontrollkanals und der Datenkanäle erweitert ist. Oberhalb der GridFTP-Schicht ist der Reliable File Transfer-Dienst angesiedelt, welcher über den Austausch von Nachrichten im Simple Object Access Protocol (SOAP)-Format etwa 3rd Party GridFTP-Datentransfers einleitet und verwaltet. Das Monitoring and Discovery

[18] http://www.globus.org/alliance

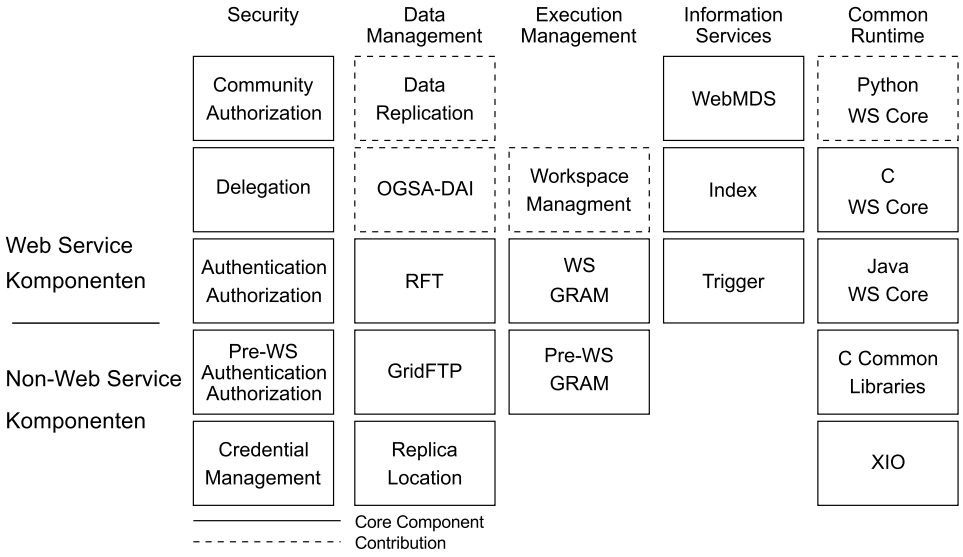


Abbildung 5.19: Komponenten der Globus Toolkit Middleware 4.0, Quelle: [Fos05]

System (MDS) bildet einen Teil des Informationssystems im Globus Toolkit und entspricht einer Menge von Diensten zur Ressourcenentdeckung und -überwachung.

Das Globus Toolkit in Version 4.0.x ist auf verschiedene Weisen installierbar. Es stehen einerseits Binärpakete für verschiedene Linux-Distributionen, wie Red Hat, Debian und Fedora, zur Verfügung. Andererseits ist aber auch eine Installation mittels der Übersetzung des Quellcodes auf dem Zielsystem möglich.
Da für die beiden, in Abschnitt 2.1 vorgestellten, Betriebssysteme keine passenden Binärpakete verfügbar sind, erfolgt in den nachfolgend vorgestellten Workflows die Installation der Globus Toolkit Middleware durch das Übersetzen des Quellcodes.

Voraussetzungen Bevor die Installation der Globus Toolkit Middleware beginnen kann, müssen alle Voraussetzungen hinsichtlich der Software-Anforderungen erfüllt sein. Die detaillierte Liste der erforderlichen Software ist auf den Webseiten des Globus Toolkit[19] zu finden. Im Wesentlichen müssen eine Java-Entwicklungsumgebung, Apache Ant, ein C-Compiler, das Werkzeug `make`, PostgreSQL, Entwicklerpakete von OpenSSL und das Grid Packaging Tool (GPT) installiert werden. Ein Workflow, der die Installation dieser Software-Komponenten durchführt, ist in Abbildung 5.20 gezeigt. Dabei wird davon ausgegangen, dass der C-Compiler und das `make`-Werkzeug standardmäßig auf dem System vorhanden sind.
Während der ersten Aktivität des Workflows erfolgt über den in Abbildung 5.21 dargestellten Subflow die Installation von Apache Ant. Die Software wird hierbei durch die Aktivi-

[19] `http://www.globus.org/toolkit/docs/4.0/admin/docbook/ch03.html`

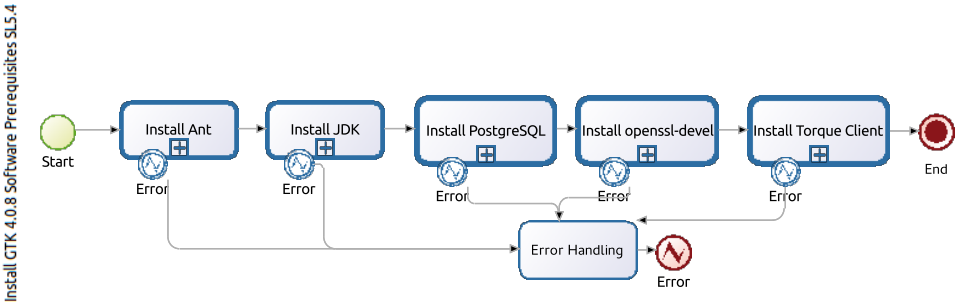


Abbildung 5.20: Workflow zur Installation der Globus Toolkit 4.0 Software Prerequisites

tät `Download Ant Software` in ein lokales Verzeichnis heruntergeladen. Bevor sie in der Aktivität `Extract Ant Software` in das Zielverzeichnis extrahiert wird, ermittelt der Schritt `Generate FQN` den vollständigen Pfad zum Software-Archiv im System. Die

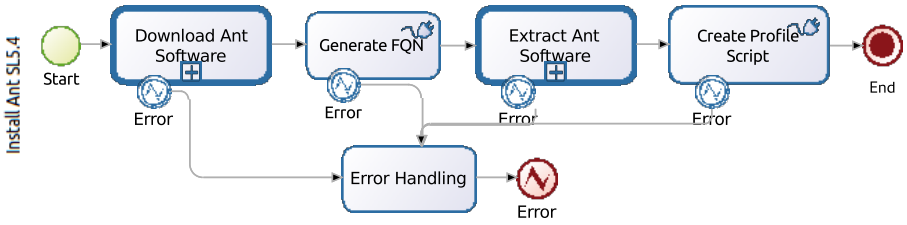


Abbildung 5.21: Workflow zur Installation von Apache Ant

letzte Aktivität des Subflows erzeugt – wie in Listing 5.5 zeigt – eine Datei im Verzeichnis `/etc/profile.d`. In ihr sind die Variablen `ANT_HOME` und `PATH` entsprechend gesetzt, um die Ant-Software im System nutzbar zu machen. Standardmäßig erfolgt die Installation von Apache Ant Version 1.8.2, weswegen dies in der Groovy-Applikation fest-kodiert ist.

```
def File file = new File ("/etc/profile.d/ant.sh")
def String output = "ANT_HOME=${extractionDirectory}/apache-ant-1.8.2\n"
output = output +  "PATH=\$ANT_HOME/bin:\$PATH"
file.write(output)
```

Listing 5.5: Create Profile Script (Groovy-Applikation)

Die auf die Apache Ant-Installation folgenden Aktivitäten des Workflows zur Installation der Software-Voraussetzungen, also `Install JDK`, `Install PostgreSQL` und `Install openssl-dev` verwenden den bereits vorgestellten Subflow `YUM Install Package`.

Damit sind die Voraussetzungen für die Installation der Globus Toolkit Middleware erfüllt. Will man jedoch die Middleware an ein lokal vorhandenes Batchsystem anbinden, so ist noch die hierfür erforderliche Batchsystem-Software zu installieren. Dies erfolgt im Workflow durch die Aktivität `Install Torque Client`, welche den in Abbildung 4.28(a)

dargestellten Workflow als Subflow einbindet.

Grid Packaging Tool Das Grid Packaging Tool dient in verschiedenen Projekten als Werkzeug für das Deployment von Grid Middlewares. Für die Installation des Grid Packaging Tools ist es zunächst notwendig, das Quellcode-Archiv herunterzuladen. Dies geschieht in der Aktivität `Download GPT` des zugehörigen Workflows (vgl. Abbildung 5.22) über den Aufruf des Subflows `Download Files`. Anschließend extrahiert die Aktivität `Extract GPT` den Inhalt des Archivs in das Verzeichnis `/tmp`. Das Kompilieren des Quellcodes und die Installation des Kompilats in das, durch die Variable `GPT_INSTALL_LOCATION` spezifizierte, Verzeichnis übernimmt die Workflow-Aktivität `Build GPT`. Die dieser Aktivität zugeordnete Groovy-Applikation ist in Listing 5.6 angegeben. Da-

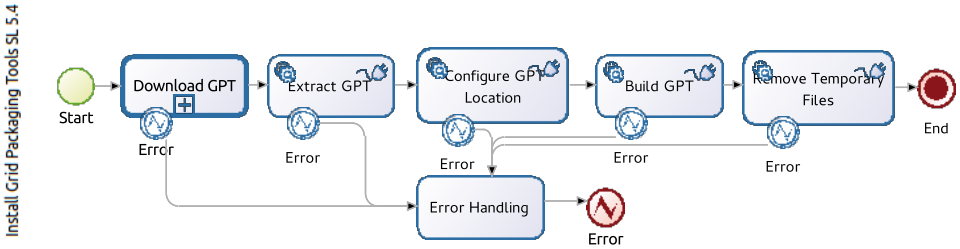


Abbildung 5.22: Workflow zur Installation und Konfiguration des GPT

bei legt das Argument `new File("/tmp/gpt-3.2/")` für die Methode `execute` das Arbeitsverzeichnis für den Befehl auf den Extraktionsort des Grid Packaging Tool-Archivs fest.

```
def String command = "./build_gpt --prefix=${gptLocation}"
def Process proc = command.execute(null, new File("/tmp/gpt-3.2/"))
proc.waitFor()
```

Listing 5.6: Kompilieren des Grid Packaging Tools (Groovy-Applikation)

Abschließend werden das zuvor heruntergeladene Grid Packaging Tool-Archiv und das temporäre Verzeichnis, in das das Archiv extrahiert wurde, gelöscht. Die Aktivität `Remove Temporary Files` verwendet hierzu die in Listing 5.7 gezeigte Groovy-Applikation.

```
def String command = "rm -rf /tmp/gpt-3.2-src.tar.gz"
def Process proc = command.execute()
proc.waitFor()

command = "rm -rf /tmp/gpt-3.2/"
proc = command.execute()
proc.waitFor()
```

Listing 5.7: Entfernen des Grid Packaging Tool-Archivs (Groovy-Applikation)

Nachdem die Installation des Grid Packaging Tools abgeschlossen ist, kann die Installation der Globus Toolkit Middleware beginnen.

Installation und Konfiguration der Globus Toolkit Middleware Die Installation und vor allem die Konfiguration der Middleware bestehen aus vielen Einzelschritten, deren übergreifendes Zusammenspiel in einen Workflow (vgl. Abbildung 5.23) abbildbar ist. In den

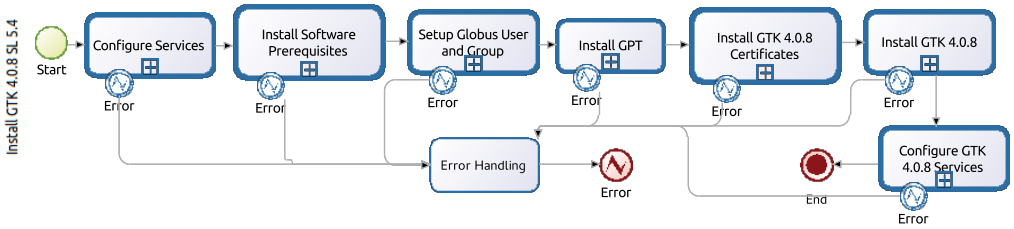


Abbildung 5.23: Workflow zur Installation und Konfiguration des Globus Toolkit

folgenden Abschnitten sind die wesentlichen Schritte – die Installation der Middleware im lokalen Dateisystem bis hin zur Konfiguration einzelner Dienste – detailliert beschrieben. Bereits beschrieben wurden die Installation der Software-Voraussetzungen bzw. der Ablauf der Aktivität `Install Software Prerequisites` sowie die Installation des Grid Packaging Tools bzw. die Aktivität `Install GPT`.

Anlegen des Nutzers `globus` Die Globus Toolkit Middleware läuft innerhalb eines Web Service-Containers unter Verwendung eines lokalen Nutzerkontos. Somit sind als Teil der Installation der Middleware ein solcher lokaler Nutzer sowie eine Gruppe, zu der der Nutzer gehören wird, zu erzeugen. Beides erfolgt während der Aktivität `Setup Globus User and Group` des Workflows aus Abbildung 5.23. Die Aktivität greift intern auf den in Abbildung 5.24 dargestellten Workflow zurück, der aus den zwei Aktivitäten `Add Group` und `Add User` besteht. Beispielsweise besitzt letztere die Parameter `home directory` und `login name`, deren Werte später an das Kommando `useradd` übergeben werden.

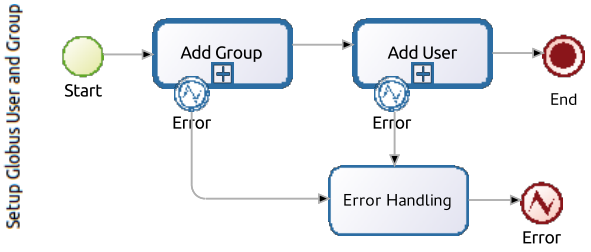


Abbildung 5.24: Workflow für das Einrichten des Nutzers `globus`

Installation der Globus Toolkit Software Wie bereits angedeutet, kann die Installation der Globus Toolkit Software aus Binärpaketen oder durch die Übersetzung des Quellcodes erfolgen. In dem hier vorgestellten Workflow (vgl. Abbildung 5.25) erfolgt die Installation unter Verwendung des Quellcodes. Dieser wird während der Ausführung der ersten Aktivität des Workflows unter Verwendung des Subflows `Download_Files` als Archiv von den Webseiten der Globus Alliance heruntergeladen. Darauf folgt in der Aktivität `Extract Archive` die Extraktion des Software-Archivs in das Verzeichnis `/tmp`. Nachdem der Quellcode auf dem System vorliegt, wird dieser über den Aufruf eines `configure`-Skripts in der Aktivität `Configure GTK 4.0.8` auf das Kompilieren vorbereitet (vgl. Listing 5.8). Hierbei wird das Installationsverzeichnis für die Middleware mittels der Angabe der `-prefix`-Option auf `/opt/globus-4.0.8/` festgelegt. Eine Folgeversion des Workflows kann diese Restriktion durch Parametriesierung aufheben.

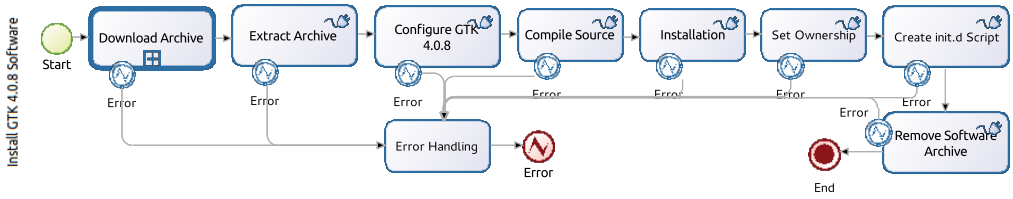


Abbildung 5.25: Workflow zur Installation und Konfiguration der Globus Toolkit Software

```
def String command = "./configure --prefix=/opt/globus-4.0.8/ --enable-
    prewsmds --enable-wsgram-pbs --enable-i18n"
def Process proc = command.execute(null, new File("/tmp/gt4.0.8-all-
    source-installer/"))
proc.waitFor()
```

Listing 5.8: Konfiguration des Globus Toolkit 4.0.8 Quellcodes (Groovy-Applikation)

Das Kompilieren und die nachgeschaltete Installation des Kompilats erfolgen in der Aktivität `Installation`. Der Inhalt der zugehörigen Groovy-Applikation ist in Listing 5.9 dargestellt, wobei im Wesentlichen das Werkzeug `make` zum Einsatz kommt.

```
def String command = "make"
def Process proc = command.execute(null, new File("/tmp/gt4.0.8-all-
    source-installer/"))
proc.waitFor()
command = "make install"
proc = command.execute(null, new File("/tmp/gt4.0.8-all-source-installer/
    "))
proc.waitFor()
```

Listing 5.9: Kompilieren des Globus Toolkit 4.0.8 Quellcodes (Groovy-Applikation)

Während der Aktivität `Set Ownership` werden die Besitzrechte (Nutzer- sowie Gruppenrechte) der zuvor installierten Dateien der Globus Middleware mittels des `chmod`-Befehls angepasst. Nachfolgend wird durch die Aktivität `Create init.d Script` im Verzeichnis `/etc/init.d` ein Skript zum Starten und Stoppen des Globus Toolkit Web Service-Containers hinterlegt. Des Weiteren sorgen zwei Aufrufe des Befehls `chkconfig` (vgl. Listing 5.10) dafür, dass der Web Service-Container bei jedem Neustart des Systems mitgestartet wird.

```
...
def String command = "chkconfig --add globus-container"
def Process proc = command.execute()
proc.waitFor()

command = "chkconfig globus-container on"
proc = command.execute()
proc.waitFor()
```

Listing 5.10: Erzeugen des init.d-Skripts (Groovy-Applikation)

Die letzte Aktivität des Workflows löscht einerseits das heruntergeladene Software-Archiv und andererseits auch das temporär benötigte Extraktionsverzeichnis.

Installation der erforderlichen X.509-Zertifikate Dieser Abschnitt beschreibt die Umsetzung der in Abbildung 5.23 zu findenden Aktivität `Install GT 4.0.8 Certificates` bzw. des entsprechenden Subflows.

Zur Identifizierung gegenüber Grid-Nutzern und anderen Grid Middleware-Diensten sowie zur Sicherung der Kommunikation setzt die Globus Toolkit Middleware auf X.509-Zertifikate. Auf einer Globus Toolkit-Resource sind daher ein Host-Zertifikat sowie der zugehörige Schlüssel zu hinterlegen, wobei sich als Standardverzeichnis `/etc/grid-security` etabliert hat. In dieses Verzeichnis, genauer in das Unterverzeichnis `certificates`, sind ebenso die Zertifikate der von der Ressource unterstützten Certification Authorities zu hinterlegen. Weiterhin kann der Web Service-Container der Middleware über ein separates X.509-Zertifikat abgesichert sein.
Der Ablauf der Hinterlegung bzw. Installation der einzelnen Zertifikate ist durch den Workflow in Abbildung 5.26 beschrieben.

Die erste Aktivität des Workflows ruft den gleichnamigen Subflow `Install Host and CA Certificates` (vgl. Abbildung 5.17) auf. Nach dem Abschluss dieser Aktivität sind sowohl Host-Zertifikat und -Schlüssel als auch die Zertifikate der CAs an ihrem Platz. Zudem wurden die Zugriffsberechtigungen auf Host-Zertifikat bzw. -Schlüssel eingeschränkt. Die zweite Aktivität des Workflows, `Configure Container Certifi-`

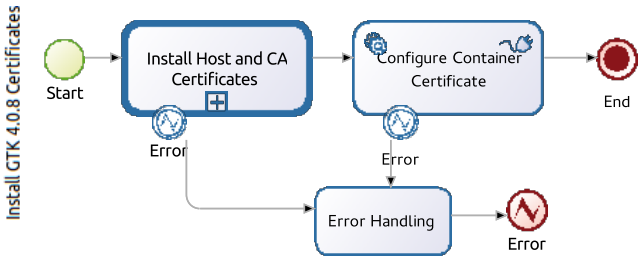


Abbildung 5.26: Workflow zur Installation des Host- und Container-Zertifikats

cate, richtet u. a. die Verwendung des Host-Zertifikats als Zertifikat des Globus Toolkit Web Service-Containers ein. Hierzu erfolgen, wie in Listing 5.11 gezeigt, Kopieroperationen sowie i) die Einschränkung der Zugriffsrechte und ii) die Änderung der Besitzrechte an den entsprechenden Dateien.

```
def String command = "cp /etc/grid-security/hostkey.pem /etc/grid-
    security/containerkey.pem"
def Process proc = command.execute()
proc.waitFor()

command = "cp /etc/grid-security/hostcert.pem /etc/grid-security/
    containercert.pem"
proc = command.execute()
proc.waitFor()

command = "chmod 400/etc/grid-security/containerkey.pem"
proc = command.execute()
proc.waitFor()

command = "chmod 644 /etc/grid-security/containercert.pem"
proc = command.execute()
proc.waitFor()

command = "chown globus.globus /etc/grid-security/container*.pem"
proc = command.execute()
proc.waitFor()
```

Listing 5.11: Anlegen des Zertifikats für den WS-Container (Groovy-Applikation)

Konfiguration der Globus Middleware-Dienste Wie Abbildung 5.19 zeigt, besteht das Globus Toolkit aus verschiedenen Modulen bzw. Diensten, wobei die Verfügbarkeit dieser Dienste auf dem System von den Angaben beim Aufruf des `configure`-Skripts (vgl. Listing 5.8) abhängt.

Stellvertretend für diese Dienste sind nachfolgend für GridFTP, GRAM, RFT und MDS

Workflows zur Konfiguration vorgestellt. Die Einzelkonfiguration der Dienste wiederum ist zu dem in Abbildung 5.27 gezeigten Workflow zusammengefasst. Jede Dienstkonfiguration entspricht in diesem Workflow einer separaten Aktivität.

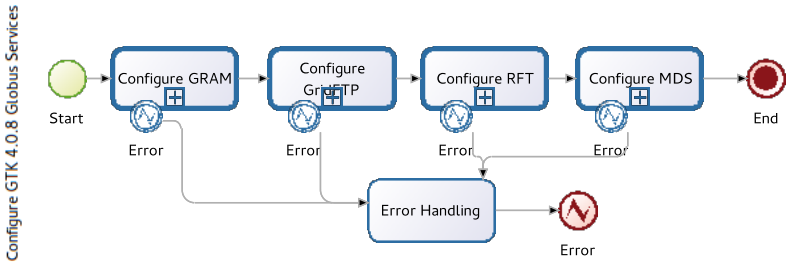


Abbildung 5.27: Workflow zur Konfiguration der Globus Toolkit-Dienste

GridFTP Die Bereitstellung des GridFTP-Dienstes erfolgt über eine Einbindung in den Extended Internet Daemon (XINETD). Hierbei ist die Datei `/etc/xinetd.d/gsi-ftp` zu erzeugen und mit Inhalt zu füllen. Ein Beispiel hierfür ist in Listing 5.12 zu sehen, wobei der Platzhalter `<$GLOBUS_LOCATION>` während der Ausführung der Aktivität `Create Xinetd Configuration` des Workflows (vgl. Abbildung 5.28) durch den Pfad zur Globus Toolkit-Installation ersetzt wird.

```
service gsiftp
{
 instances   = 100
 socket_type   = stream
 wait   = no
 user   = root
 env   += LD_LIBRARY_PATH=<$GLOBUS_LOCATION>/lib
 env   += GLOBUS_TCP_PORT_RANGE=20000,25000
 server   = <$GLOBUS_LOCATION>/sbin/globus-gridftp-server
 server_args   = -i
 nice = 10
 disable = no
}
```

Listing 5.12: Inhalt der Datei `/etc/xinetd.d/gsiftp`

Weiterhin erfolgt in der Aktivität `Add GridFTP Service` für den durch GridFTP genutzten Port und Protokoll-Typ (Transmission Control Protocol (TCP) bzw. User Datagram Protocol (UDP)) ein Eintrag in die Datei `/etc/services`[20]. Die Aktivität `Modify hosts.allow` richtet durch Einträge in der Datei `/etc/hosts.allow` die Nutzung

[20]Die Datei enthält eine Abbildung zwischen beschreibenden Namen für Internet-Dienste und den unterliegenden Port-Nummern und Protokolltypen.

des GridFTP-Dienstes durch entfernte Rechner ein.

Abschließend startet die Aktivität `Reload Xinetd Configuration` über einen Sub-

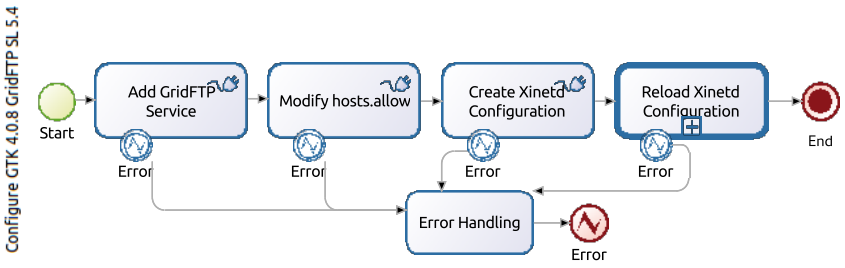


Abbildung 5.28: Workflow zur Konfiguration von GridFTP

flow den Extended Internet Daemon neu, so dass diesem die zuvor erzeugte Konfigurationsdatei bekannt wird.

GRAM Der GRAM-Dienst bietet auf einer Globus Toolkit-Ressource nach außen eine Schnittstelle zur Einreichung und Überwachung von Rechen-Jobs an. Intern kommuniziert der Dienst hierzu ggf. mit dem vorhandenen Batchsystem Server.

Analog zum GridFTP-Dienst erfolgt auch die Einbindung dieses Dienstes über den Extended Internet Daemon. Das Listing 5.13 zeigt den Inhalt der während der Aktivität `Create Xinetd Configuration` des Workflows (vgl. Abbildung 5.29) angelegten Datei `/etc/xinetd.d/gsigatekeeper`, wobei die Variable `<$GLOBUS_LOCATION>` noch durch den Pfad zur Globus Toolkit-Installation ersetzt wird.

```
service gsigatekeeper
{
  socket_type = stream
  protocol    = tcp
  wait        = no
  user        = root
  env        += LD_LIBRARY_PATH=<$GLOBUS_LOCATION>/lib
  env        += GLOBUS_TCP_PORT_RANGE=20000,25000
  server      = <$GLOBUS_LOCATION>/sbin/globus-gatekeeper
  server_args = -conf <$GLOBUS_LOCATION>/etc/globus-gatekeeper.conf
  disable     = no
}
```

Listing 5.13: Inhalt der Datei `/etc/xinetd.d/gsigatekeeper`

Die zwei Aktivitäten `Add GRAM Service` und `Modify hosts.allow` des Workflows erfüllen eine Funktionalität wie die Aktivitäten des GridFTP-Workflows. Hinzugekommen ist die Aktivität `Modify Sudoers File`. Die Applikation hinter dieser Akti-

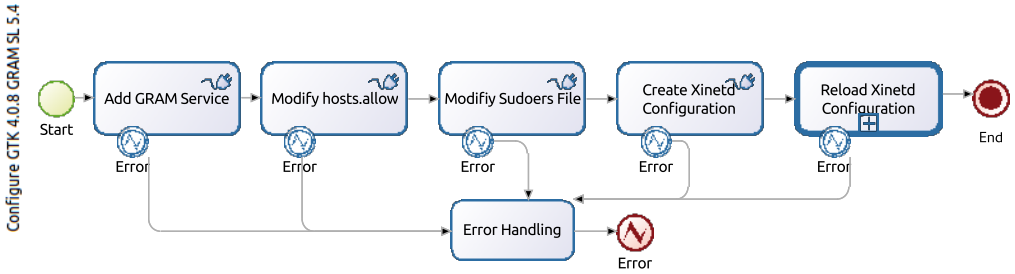


Abbildung 5.29: Workflow zur GRAM-Konfiguration

vität (vgl. Listing 5.14) fügt zwei Zeilen in die Datei `/etc/sudoers` ein, die eine privilegierte Ausführung der Dateien `globus-job-manager-script.pl` und `globus-gram-local-proxy-tool` durch den Nutzer `globus` erlauben. Dabei wird die Variable `${globusLocation}` durch das Installationsverzeichnis der Middleware ersetzt.

```
def File f = new File("/etc/sudoers")
def String output = "\nglobus  ALL=(ALL) NOPASSWD:  ${globusLocation}/
    libexec/globus-gridmap-and-execute -g /etc/grid-security/grid-mapfile
    ${globusLocation}/libexec/globus-job-manager-script.pl *"
f << output
output  = "\nglobus  ALL=(ALL) NOPASSWD: ${globusLocation}/libexec/globus
    -gridmap-and-execute -g /etc/grid-security/grid-mapfile  ${
    globusLocation}/libexec/globus-gram-local-proxy-tool *"
f << output
```

Listing 5.14: Modifikation der Datei `/etc/sudoers` (Groovy-Applikation)

RFT Zur Nutzung des Reliable File Transfer-Dienstes benötigt die Globus Toolkit Middleware im Hintergrund eine relationale Datenbank. In dem hier vorgestellten Workflow (vgl. Abbildung 5.30) übernimmt PostgreSQL diese Rolle. Der zugehörige Server wurde bereits durch einen zuvor aufgerufenen Workflow (vgl. Abbildung 5.20) installiert.

Die Aktivität `Enable PostgreSQL TCP/IP Connections` des Workflows verändert den Inhalt der Datei `/etc/sysconfig/postgresql`, so dass der Datenbank-Server explizit Verbindungen über TCP/IP zulässt. Die zugehörige Groovy-Applikation zeigt Listing 5.15.

```
def String output_string = """\nPGOPTS=\"-i\""""
def File f = new File ("/etc/sysconfig/postgresql")

f << output_string
```

Listing 5.15: Modifikation der PostgreSQL-Konfiguration (Groovy-Applikation)

Für die Initialisierung der PostgreSQL-Datenbank ist die Aktivität `Initialize PostgreSQL Database` verantwortlich. Wie in Listing 5.16 gezeigt, wird die neue Datenbank durch den Nutzer `postgres` angelegt und im Verzeichnis `/var/lib/pgsql/data` liegen.

```
def String command "sudo -u postgres initdb -D /var/lib/pgsql/data/"
def Process proc = command.execute()
proc.waitFor()
```

Listing 5.16: Initialisierung der PostgreSQL-Datenbank (Groovy-Applikation)

Im letzten Schritt des Workflows werden dem Nutzer `globus` Zugriffsrechte auf die Datenbank eingeräumt. Dies geschieht durch Anpassungen in der Datei `pg_hba.conf`.

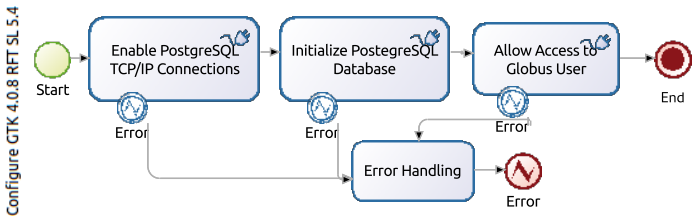


Abbildung 5.30: Workflow zur RFT-Konfiguration

MDS Das Monitoring and Discovery System ist ein Teil des Informationssystems der Globus Toolkit Middleware. Pro Grid-Ressource sollte ein nach außen publizierender MDS-Dienst vorhanden sein. Ähnlich zur gLite Middleware liegt auch hier eine hierarchische Architektur vor, wobei in der jeweils unteren Ebene über die Definition von Upstreams festgelegt ist, zu welchen übergeordneten Instanzen die Informationen geleitet werden.

Der für die Einrichtung des MDS-Dienstes erstellte Workflow (vgl. Abbildung 5.31) besteht aus zwei Einzelaktivitäten. Während der Ausführung der Aktivität `Configure Upstream` wird eine neue Datei `hierarchy.xml` erzeugt, welche die Liste der Upstreams enthält. Der bisherige Inhalt dieser Datei wird in einer Sicherheitskopie gespeichert

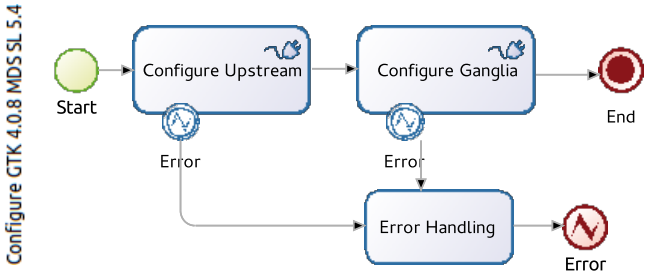


Abbildung 5.31: Workflow zur MDS-Konfiguration

(vgl. Listing 5.17).

```
def String command = "cp ${globusLocation}/etc/globus_wsrf_mds_index/
    hierarchy.xml ${globusLocation}/etc/globus_wsrf_mds_index/hierarchy.
    xml.old"
def Process proc = command.execute()
proc.waitFor()

def String content = "<config>\n"
for ( element in "${upstreams}".split() ) {
  content = content + "<upstream>" + element + "</upstream>\n"
}
content = content + "</config>\n"

def File f = new File("${globusLocation}/etc/globus_wsrf_mds_index/
    hierarchy.xml")
f << content

command = "chown globus: ${globusLocation}/etc/globus_wsrf_mds_index/
    hierarchy.xml"
proc = command.execute()
proc.waitFor()
```

Listing 5.17: Eintrag der MDS Upstreams (Groovy-Applikation)

Eines der Argumente, die beim Aufruf des Workflows angegeben werden müssen, ist eine durch Leerzeichen separierte Auflistung der einzutragenden Upstream-Endpunkte. Workflow-intern liegt diese Auflistung als Variable `${upstreams}` vor.
Die zweite Aktivität des Workflows, `Configure Ganglia`, richtet die Publikation der über die Ganglia Software (vgl. Abschnitt 4.7) gesammelten Informationen über den MDS-Dienst ein.

5.3.3.2 Globus Toolkit 5

Das Globus Toolkit weist bereits beim Übergang von Version 3 zu Version 4 einen starken Bruch bei den eingesetzten Technologien auf. Dies wiederholt sich beim Übergang von Version 4 zu Version 5. Letztere wurde im Januar 2010 veröffentlicht und enthielt beispielsweise nicht mehr die Komponenten WS-GRAM und Reliable File Transfer, wie sie in Version 4 vorkamen (vgl. Abbildung 5.19). Die in Globus Toolkit 5 enthaltenen Komponenten sind in Abbildung 5.32 gezeigt. Ein Grund für den Übergang von WS-GRAM zu einem Nicht-WS-orientierten GRAM ist ein reduzierter (Kommunikations-)Overhead während beispielsweise der Job-Einreichung. Gerade bei kurzlebigen Jobs nahm dieser Overhead bei WS-GRAM einen überproportional großen Anteil an der Gesamtdauer eines Jobs

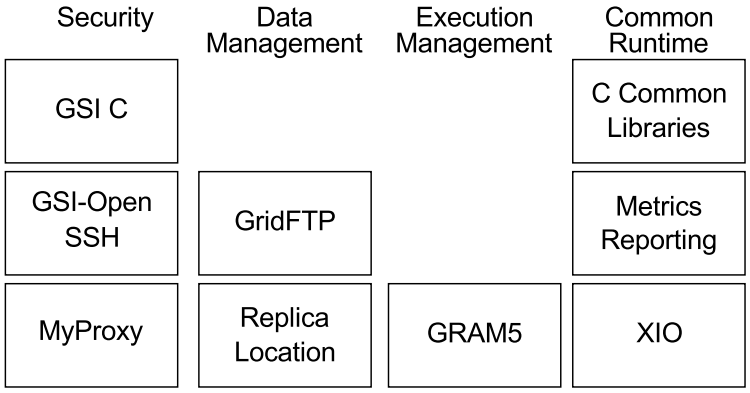


Abbildung 5.32: Komponenten des Globus Toolkit Middleware 5, Quelle: `http://www.globus.org/toolkit/docs/5.0/`

ein.

Analog zu Abschnitt 5.3.3.1 sind hier die einzelnen für die Installation und Konfiguration einer Globus Toolkit 5.0.3-Ressource notwendigen Schritte zu einem Workflow (vgl. Abbildung 5.33) zusammengefasst. Die Aktivitäten `Configure Services` und `Install Software Prerequisites` sind in ihren Aufgaben analog zu den gleichnamigen Aktivitäten des Workflows zur Installation der Middleware in Version 4 konzipiert. Im Anschluss an die beiden Aktivitäten erfolgt im Schritt `Install GT 5.0.3 Certificates` das Einspielen des Host-Zertifikats und der CA-Zertifikate über den Aufruf des Subflows `Install Host and CA Certificates`.

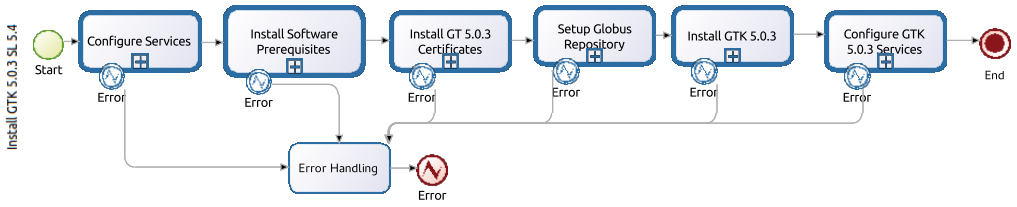


Abbildung 5.33: Workflow zur Installation und Konfiguration der Globus Toolkit 5 Software

Um die Globus Toolkit-Software aus YUM-Repositories herunterladen zu können, hinterlegt die später im Detail vorgestellte Aktivität `Setup Globus Repositories` die notwendigen Informationen im Systemverzeichnis `/etc/yum.repos.d`. Die abschließenden Aktivitäten des Workflows setzen sich mit der Installation und der Konfiguration der zuvor installierten Dienste des Globus Toolkit auseinander und nutzen hierzu Subflows.

Voraussetzungen Im Vergleich zu Version 4 des Globus Toolkit entfällt bei Version 5 die Installation von z. B. Apache Ant und PostgreSQL, lediglich die Software zur Anbindung an

das lokale Batchsystem ist einzuspielen. Dies führt zu dem in Abbildung 5.34 dargestellten Workflow, der nur aus der Aktivität `Install Torque Client` besteht. Die Funktion dieser Aktivität ist in Abschnitt 4.8.2 beschrieben.

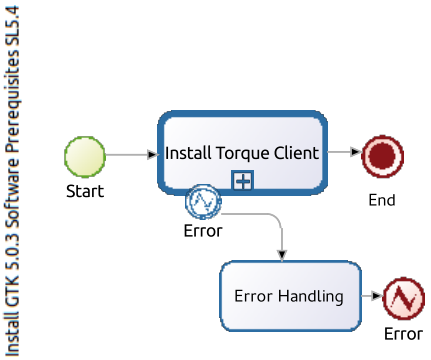


Abbildung 5.34: Workflow zur Installation der Globus Toolkit 5 Software Requirements

Installation der erforderlichen X.509-Zertifikate Analog zur Middleware Version 4.x des Globus Toolkit benötigt man auch bei Version 5 ein Host-Zertifikat, den zugehörigen Schlüssel sowie die Zertifikate der erlaubten Certification Authorities.
Die Installation dieser drei Komponenten erfolgt über die Aktivität `Install GT 5.0.3 Certificates` des Workflows aus Abbildung 5.33 bzw. über den Aufruf des bereits bekannten Subflows `Install Host and CA Certificates`.
Aufgrund des Fehlens des WS-Containers, wie er in Version 4 existiert, entfallen hier weitere Schritte zum Anlegen des Container-Zertifikats und -Schlüssels (vgl. Abbildung 5.26).

Installation der Globus Toolkit Software Wie zu Beginn des Abschnitts erwähnt, ist für Version 5 des Globus Toolkit eine Installation unter Verwendung von Software-Repositories möglich. Für die Hinterlegung der notwendigen Repository-Informationen ist der in Abbildung 5.35 dargestellte Workflow zuständig. Dieser lehnt sich stark an den in Ab-

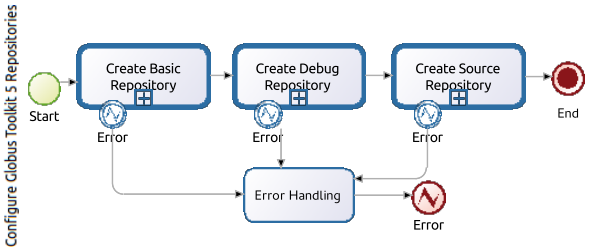


Abbildung 5.35: Workflow zur Konfiguration der Globus Toolkit 5 Software-Repositories

schnitt 5.3.2.1 vorgestellten Workflow für den gLite siteBDII an und besitzt die drei Akti-

vitäten `Create Basic Repository`, `Create Debug Repository` und `Create Source Repository`. Die erste Aktivität erzeugt eine Datei im Verzeichnis `/etc/yum.repos.d` mit dem in Listing 5.18 gezeigten Inhalt.

```
[IGE]
name=IGE - x86_64
baseurl=http://repo-rpm.ige-project.eu/redhat/el5/x86_64
enabled=1
gpgcheck=1
gpgkey=http://repo-rpm.ige-project.eu/RPM-GPG-KEY-IGE
```

Listing 5.18: IGE Basic Software Repository-Konfiguration

Die zweite und dritte Aktivität erzeugen entsprechend Dateien für die Debug- und Quellcode-Repositories der Software im gleichen Verzeichnis.

Nach Einrichtung der drei Repositories erfolgt die Installation der Pakete für die Globus Toolkit-Dienste während der Aktivität `Install GTK 5.0.3` (vgl. Abbildung 5.33). Diese Aktivität nutzt den Subflow `YUM Install Packages` mit den Argumenten `ige-meta-globus-default-security`, `ige-meta-globus-gsissh`, `ige-meta-globus-gridftp` und `ige-meta-globus-gram5`, so dass am Ende neben GSISSH und GridFTP auch der GRAM-Dienst installiert ist.

Konfiguration der Globus Toolkit-Dienste Mit dem Erscheinen von Globus Toolkit 5 nahm im Vergleich zur Vorgängerversion die Anzahl der angebotenen Dienste ab. Übrig blieben mit GRAM, GridFTP und GSISSH drei Dienste, deren Konfiguration der in Abbildung 5.36 gezeigte Workflow durchführt. Die in den Aktivitäten `Configure GRAM`,

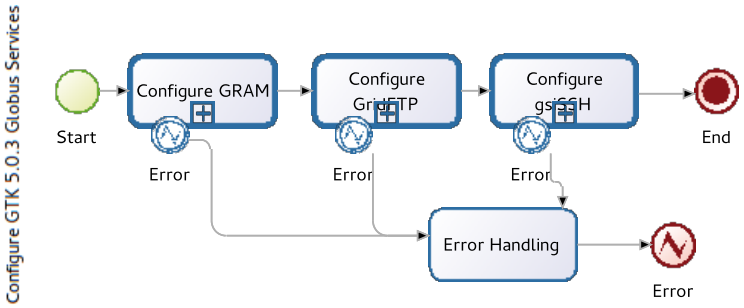


Abbildung 5.36: Workflow zur Konfiguration der Globus Toolkit 5-Dienste

`Configure GridFTP` und `Configure gsiSSH` ausgeführten Schritte entsprechen den bei der Globus Toolkit Version 4 (vgl. Abschnitt 5.3.3.1) angewandten und werden daher hier nicht mehr explizit vorgestellt.

5.3.4 UNICORE Middleware

Die Entwicklung der Middleware Uniform Interface to Computing Resources (UNICORE) begann im Jahr 1997 als ein vom Bundesministerium für Bildung und Forschung (BMBF) gefördertes Projekt. Ein Schwerpunkt des Projekts lag in der Entwicklung einer Software-Infrastruktur für den sicheren, intuitiven und nahtlosen Zugang zu High Performance Computing-Ressourcen (vgl. [Erw02]).
Im August 2007 erfolgte mit der Veröffentlichung von UNICORE 6 der Umstieg auf eine zum Web Service Resource Framework (WSRF)-konforme Architektur. Eine weitere Änderung im direkten Vergleich mit dem Vorgänger UNICORE 5 ist die Unterstützung der vom Open Grid Forum (OGF) spezifizierten Job Submission Description Language (JSDL) (vgl. [Rom09]).

Neben der Abarbeitung einfacher Rechen-Jobs erlaubt die UNICORE Middleware ebenso die Ausführung komplexer Workflows. In beiden Fällen überlässt die Middleware dem Grid-Nutzer die Wahl der zu verwendenden Ressourcen, so sind etwa multiple UNICORE-Ressourcen innerhalb einer einzelnen Grid Site oder multiple Grid Sites als ausführende Stellen möglich.

Um den Übergang von UNICORE 5 auf UNICORE 6 für die Ressourcenbetreiber möglichst einfach zu gestalten, sind die Namen der UNICORE 6-Dienste trotz Umstellung auf das Web Service Resource Framework an die Namen in der Vorgängerversion angelehnt (vgl. Tabelle 5.1). In Abschnitt 5.3.4.1 sind die einzelnen Dienste der UNICORE 5 Middleware sowie deren Installation und Konfiguration genauer beschrieben. Der darauf folgende Abschnitt stellt UNICORE 6 und die dafür entworfenen Workflows vor.

	UNICORE 5	UNICORE 6
Batchsystem-Adapter	TSI	TSI
Job-Management	NJS	UNICOREX
Nutzerverwaltung	UUDB	XUUDB
Kontaktpunkt für Nutzer	Gateway	Registry

Tabelle 5.1: Dienste in UNICORE 5 und UNICORE 6

5.3.4.1 UNICORE 5

Die UNICORE 5 Middleware verwendet eine modulare Architektur (vgl. Abbildung 5.37). Die Gateways bilden die zentrale Komponenten in einem UNICORE 5-Grid und sind u. a. für die Authentifizierung der Grid-Nutzer verantwortlich. Im Rahmen der Einreichung eines Rechen-Jobs kontaktiert der Grid-Nutzer den Gateway mittels der UNICORE 5 Client

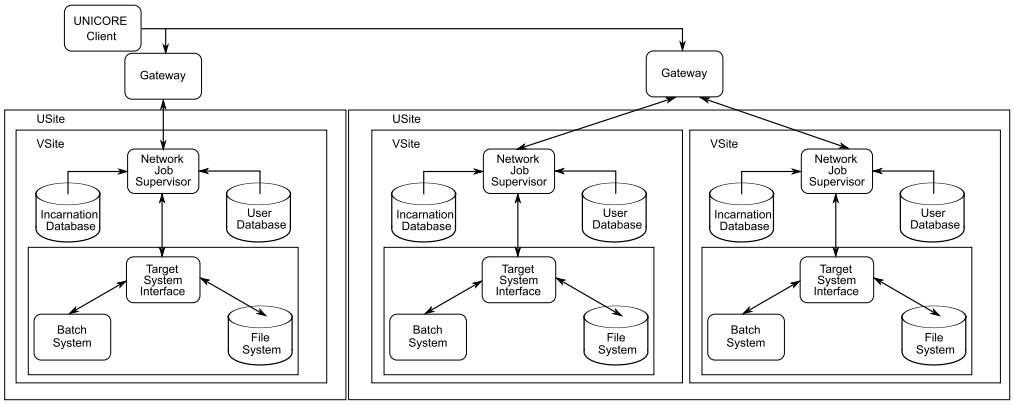


Abbildung 5.37: Architektur von UNICORE 5, Quelle: [RM06]

Software. Der Gateway selbst kommuniziert, abgesichert via Secure Socket Layer (SSL), mit den einzelnen Network Job Supervisors der verschiedenen USites[21], um beispielsweise Rechen-Jobs dorthin weiterzuleiten oder deren Zustand im Auftrag eines Grid-Nutzers abzufragen.

Nachfolgend ist ein Workflow (vgl. Abbildung 5.38) für die Installation und Konfiguration der Dienste einer VSite beschrieben. Die Aktivitäten `Configure Services` sowie

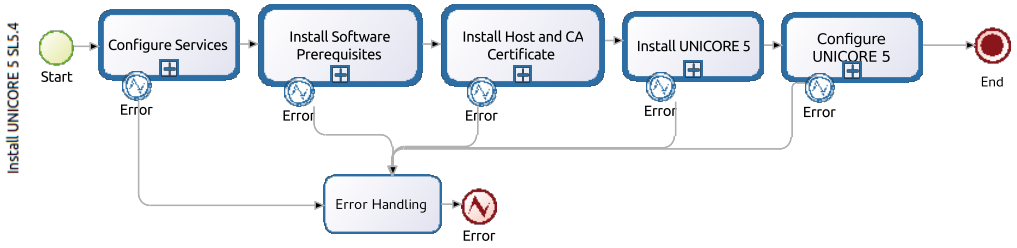


Abbildung 5.38: Workflow zur Installation von UNICORE 5

`Install Software Prerequisites` dienen – wie bei den bereits beschriebenen Middlewares – der Konfiguration der systemnahen Dienste sowie der Auflösung der existierenden Software-Abhängigkeiten. Die Installation von Host-Zertifikat, Host-Schlüssel und der Zertifikate der Certification Authorities erfolgt während der Aktivität `Install Host and CA Certificates` über den Aufruf des gleichnamigen Subflows (vgl. Abbildung 5.17). Die letzten beiden Aktivitäten des Workflows, `Install UNICORE 5` und `Configure UNICORE 5`, beschäftigen sich mit der Installation und Konfiguration der Middleware-Dienste und sind später im Detail vorgestellt.

[21] Eine USite bzw. UNICORE Site besteht aus einer oder mehreren VSites. Jede dieser VSites enthält wiederum einen Network Job Supervisor.

Installation der Software-Abhängigkeiten Die Liste der Software-Abhängigkeiten besteht aus der Software Perl, welche als installiert vorausgesetzt wird, sowie dem Java Development Kit (JDK) bzw. einer Java Runtime Environment (JRE) und der TORQUE Client Software. Die Installation des Java Development Kits und des TORQUE Client übernehmen die beiden Aktivitäten des in Abbildung 5.39 dargestellten Workflows. Dabei greift

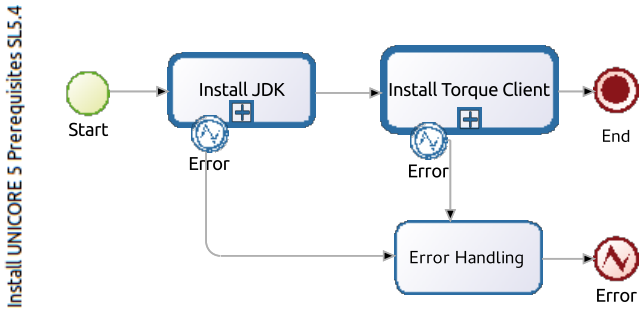


Abbildung 5.39: Workflow zur Installation der UNICORE 5 Software-Abhängigkeiten

die Aktivität `Install JDK` auf den Subflow `YUM Install Package` (vgl. Abbildung 4.3(b)) zurück und `Install Torque Client` nutzt einen der in Abschnitt 4.8.2 beschriebenen Subflows.

Installation der UNICORE 5 Software Die für die Installation der UNICORE 5-Dienste benötigte Software ist entweder für jeden Dienst separat über SourceForge-Webseiten[22] oder als ein Gesamtarchiv über Webseiten des D-Grid Projekts beziehbar, wobei nachfolgend die zweite Variante zum Einsatz kommt.
Die Aktivität `Download UNICORE 5 Software` des entworfenen Workflows (vgl. Abbildung 5.40) lädt über einen Subflow-Aufruf das Software-Archiv in ein vom Workflow-Ausführenden spezifiziertes Verzeichnis herunter. Im Anschluss daran erfolgt in der Aktivität `Extract UNICORE 5 Archive` die Extraktion des Archivinhalts in das Verzeichnis `/opt`. Ein weiterer Schritt im Workflow besteht aus dem Anlegen zweier lokaler Nut-

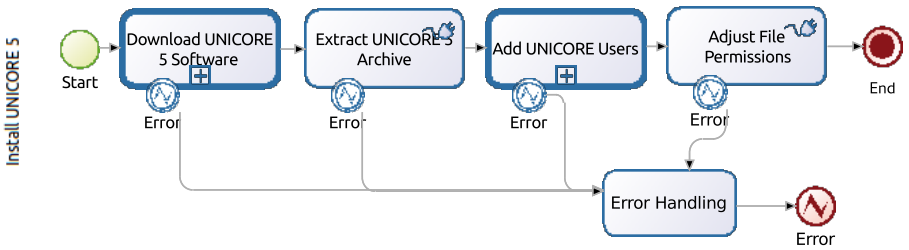


Abbildung 5.40: Workflow zur Installation des UNICORE 5 Archivs

[22] `http://sourceforge.net/projects/unicore/`

zer, welches durch den in Abbildung 5.41 gezeigten Subworkflow erfolgt. Unter der User Identification Number des einen Nutzers wird später der Network Job Supervisor-Dienst ausgeführt. Unter Verwendung des anderen Nutzers erfolgt die Abfrage von Informationen zu einzelnen Rechen-Jobs und Queues. Nach dem Erzeugen der beiden Nutzer passt die

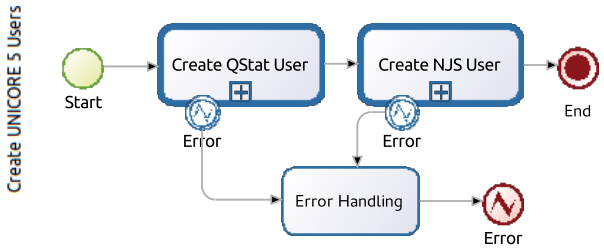


Abbildung 5.41: Workflow zur Einrichtung der Nutzer `njs` und `qstat`

Aktivität `Adjust File Permissions` die Besitzrechte für das extrahierte Verzeichnis `njs_4.6.2_build_1` sowie die Dateien `start_njs.sh` und `stop_njs.sh` an. Damit ist die Installation der UNICORE 5-Software abgeschlossen und die Konfiguration kann beginnen.

Konfiguration der UNICORE 5-Dienste Die wesentlichen Dienste einer VSite umfassen den Network Job Supervisor (NJS), das Target System Interface (TSI) und auch die UNICORE User Database (UUDB). Die Konfiguration dieser drei Dienste erfolgt über den in Abbildung 5.42 dargestellten Workflow, der für jeden von ihnen einen separaten Subflow verwendet.

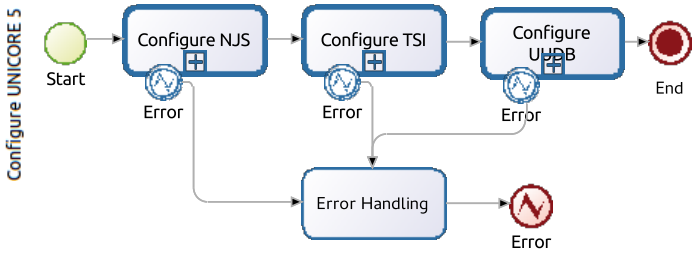


Abbildung 5.42: Workflow zur Konfiguration der UNICORE 5-Dienste

Network Job Supervisor Der Network Job Supervisor registriert sich bei dessen Start am UNICORE Gateway und steht anschließend Grid-Nutzern zur Job-Einreichung zur Verfügung. Ein Rechen-Job, eingereicht am UNICORE Client, erreicht den NJS über den Gateway in Form eines Abstract Job Objects (AJOs). Ist das Abstract Job Object zur lokalen Ausführung bestimmt, so wird es vom Target System Interface aufbereitet und an den Batchsystem Server zur Ausführung weitergeleitet.

Der Network Job Supervisor greift selbst auf mehrere Konfigurationsdateien zurück, die nach der Installation der UNICORE 5-Software anzupassen sind. Hierzu gehören beispielsweise die Dateien `njs.properties` und `njs_admin`. Der in Abbildung 5.43 gezeig-

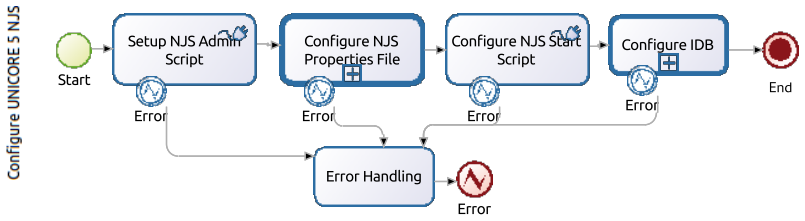


Abbildung 5.43: Workflow zur Konfiguration des UNICORE 5 NJS

te Workflow nimmt die notwendigen Modifikationen in diesen Dateien vor. Während der Ausführung der Aktivität `Setup NJS Admin Script` wird in der Datei `njs_admin` der fest-kodierte Eintrag `zam285.zam.kfa-juelich.de` durch den Hostnamen des lokalen Systems ersetzt. Für die Anpassungen in der Datei `njs.properties`, die aus Schlüssel-Wert-Paaren besteht, ist ein separater Subflow zuständig (vgl. Abbildung 5.44). Jede der Aktivitäten dieses Subflows ersetzt einen Wert für die Schlüssel `njs.my_address`, `njs.vsite_name`, `njs.gateway`, `njs.ssl_password`, `njs.njs_cert_loc` und `njs.unicore_ca_loc`.

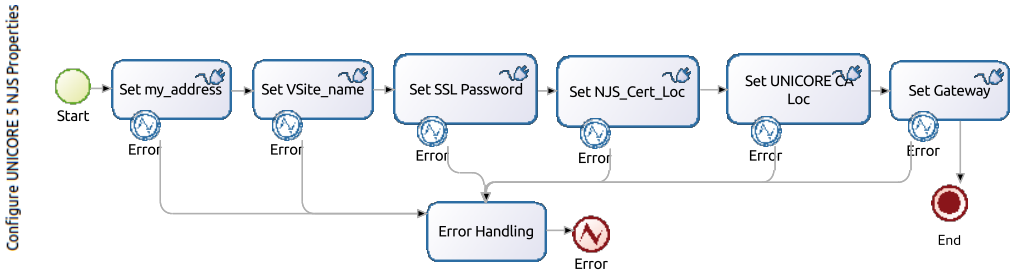


Abbildung 5.44: Workflow zur Konfiguration des Datei njs.properties

Für die Modifikationen innerhalb der Incarnation Database (IDB) existiert ebenso ein separater Subflow (vgl. Abbildung 5.45). Die Datenbank enthält ressourcenspezifische Informationen über die Anzahl der vorhandenen CPUs, den Hauptspeicher pro Rechenknoten und die auf den Knoten verfügbare Software. Diese Informationen finden während des Matchmaking-Prozesses, welcher im Gegensatz zur in Abschnitt 5.3.2 vorgestellten gLite Middleware Client-seitig erfolgt, Berücksichtigung.
Die Aktivität `Set QStat User` fügt in die Incarnation Database den Namen des lokalen Nutzers ein, der für den Kontakt mit dem Batchsystem Server genutzt werden soll. Die drei darauf folgenden Aktivitäten `Set USpace Directory`, `Set Outcome Directory` und `Set Spool Directory` legen lokale UNICORE-spezifische Pfade fest. Ab-

schließend hinterlegen die beiden Aktivitäten `Set TSI Name` und `Set TSI Source` die erforderliche Informationen zur Kontaktaufnahme mit dem Target System Interface in der Datei.

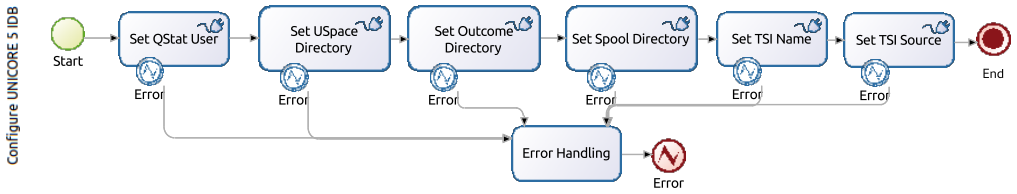


Abbildung 5.45: Workflow zur Konfiguration der UNICORE 5 IDB

Target System Interface Das Target System Interface ist ein in der Programmiersprache Perl geschriebener Dienst, der zwischen dem Network Job Supervisor und dem Batchsystem Server sitzt. Der Dienst konstruiert aus den vom Network Job Supervisor erhaltenen Abstract Job Object eine Jobbeschreibung, die von dem Batchsystem Server interpretierbar ist. Neben dem Einreichen von Rechen-Jobs erfolgt ebenso die Abfrage der Job-Stati über das Target System Interface.

In der Aktivität `Configure NJS Machine` des Workflows zur Konfiguration des Tar-

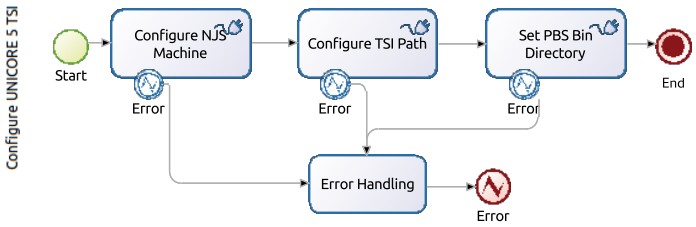


Abbildung 5.46: Workflow zur Konfiguration der UNICORE 5 TSI

get System Interface (vgl. Abbildung 5.46) erfolgt die Einrichtung der Anbindung an den Network Job Supervisor. In den beiden weiteren Aktivitäten `Configure TSI Path` und `Set PBS Bin Directory` werden die Pfade zu i) dem Installationsverzeichnis der TSI-Software und ii) den Binärdateien des Batchsystems gesetzt.

UNICORE User Database Die UNICORE User Database liegt in zwei Varianten vor, wobei Variante 1 die X.509-Zertifikate der Grid-Nutzer und Variante 2 lediglich die in den Zertifikaten enthaltenen Distinguished Names für die Authentifizierung enthält. Zudem sind in beiden Varianten der UNICORE User Database die notwendigen Informationen zur Abbildung der Grid-Nutzer auf ein oder mehrere lokale Nutzerkonten enthalten.

Im später genauer vorgestellten D-Grid erfolgt das Füllen der Datenbank über das Skript `dgridmap` (vgl. Abschnitt 5.4.2). Der in Abbildung 5.47 gezeigte Workflow greift auf

dieses Skript zurück und installiert es in der Aktivität `Download Dgridmap Script` in das Verzeichnis `/opt/d-grid/bin`. Die Ausführung des zuvor heruntergeladenen Skripts erfolgt während der Abarbeitung der Aktivität `Execute Dgridmap Script`, die Ausgabe des Skripts wird dabei in die Datei `/tmp/tmp_uudb` geschrieben. Die abschließende Aktivität des Workflows dient der eigentlichen Füllung der UUDB. Dazu bedient sich die hinter der Aktivität liegende Applikation des UNICORE-spezifischen Kommandos `uudb_admin` und der in der Datei `/tmp/tmp_uudb` enthaltenen Informationen.

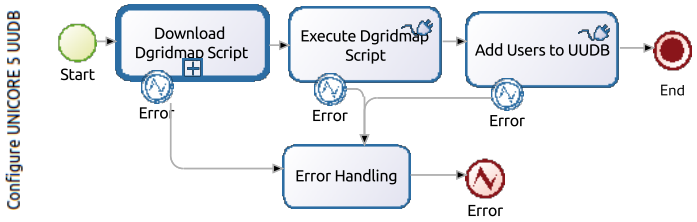


Abbildung 5.47: Workflow zur Konfiguration der UNICORE 5 UUDB

Insgesamt sind damit die Installation und die Konfiguration der UNICORE 5-Dienste einer VSite abgeschlossen.

5.3.4.2 UNICORE 6

Seit dem August des Jahres 2007 ist die UNICORE Middleware in der Version 6 verfügbar. Einhergehend mit der neuen Version änderte sich die Benennung einiger Komponenten der Middleware. Insgesamt ergibt sich für UNICORE 6 die in Abbildung 5.48 dargestellte Architektur. Der Gateway stellt im Vergleich zu UNICORE 5 bei UNICORE 6 keine zentrale

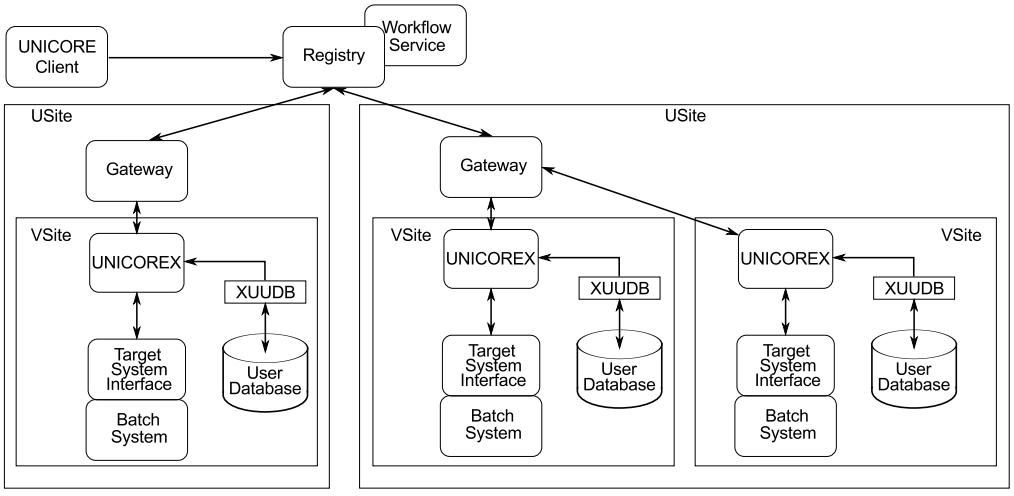


Abbildung 5.48: Architektur von UNICORE 6, Quelle: [Rom09]

Komponente dar, sondern den einzigen Eintrittspunkt zu einer USite bzw. den darin enthaltenen VSites. Als neue zentrale Komponenten treten die UNICORE 6 Service Registry und der Workflow Service (Orchestrator) auf. Innerhalb der Registry werden die angemeldeten Dienste verwaltet und können durch Grid-Nutzer über die Kontaktaufnahme per UNICORE Client gefunden werden. Soll ein Workflow ausgeführt werden, übernimmt der Workflow Service Orchestrator die Ausführungskoordination der einzelnen Workflow-Aktivitäten. In Abbildung 5.48 nicht gezeigt ist die Komponente Common Information Service (CIS), welche ähnlich zu einem gLite TopBDII (vgl. Abschnitt 5.3.2.8) Informationen über alle registrierten Dienste sammelt bzw. nach außen verfügbar macht.

Ähnlich zum Abschnitt über UNICORE 5 sind hier Workflows für die Installation der Komponenten einer USite beschrieben. So ist etwa der Gesamt-Workflow für die Installation und Konfiguration von Gateway, UNICORE/X, XUUDB und Target System Interface in Abbildung 5.49 dargestellt. Vereinfachend erfolgt dabei die Annahme, dass all diese Komponenten auf demselben System zu installieren sind.

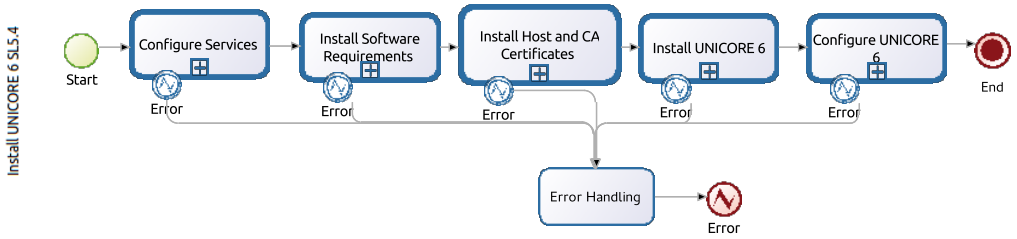


Abbildung 5.49: Workflow zur Installation und Konfiguration von UNICORE 6

Die Bedeutung der Aktivitäten `Configure Services` und `Install Host and CA Certificates` ist in den vorherigen Abschnitten ausreichend dargestellt. Die für UNICORE 6 vorhandenen Software-Abhängigkeiten löst die Aktivität `Install Software Requirements` durch die Einbindung eines Subflows (vgl. Abbildung 5.50) auf. Die RPM-basierte Installation und die anschließende Konfiguration der einzelnen UNICORE-Dienste erfolgt in den beiden Aktivitäten `Install UNICORE 6` und `Configure UNICORE 6` des Workflows.

Auflösen der Software-Abhängigkeiten Bevor die Installation der UNICORE 6-Software startet, löst ein weiterer Workflow (vgl. Abbildung 5.50) die existierenden Software-Abhängigkeiten auf. Hierzu werden in den ersten beiden Aktivitäten `Install Python` und `Install Perl` die erforderlichen Perl- und Python-Pakete über einen Subflow-Aufruf installiert. Während der Ausführung der Aktivität `Install JDK` erfolgt die Einrichtung einer Java 6-Laufzeitumgebung und als Abschluss installiert die Aktivität `Install Torque Client` den TORQUE Client. Letzterer stellt die erforderlichen Befehle für die

Job-Einreichung in das lokale Batchsystem zur Verfügung.

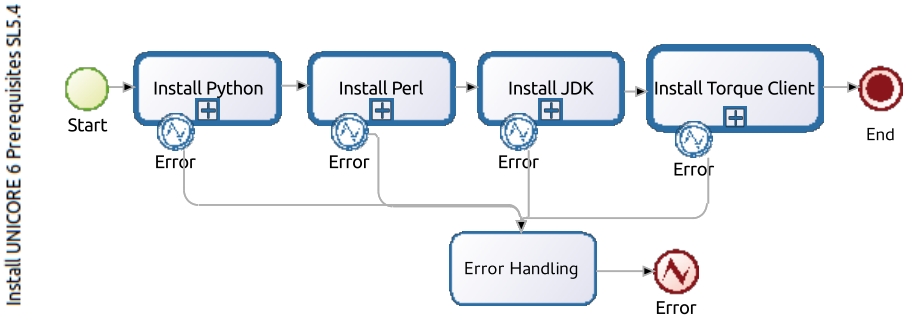


Abbildung 5.50: Workflow zur Installation der UNICORE 6-Abhängigkeiten

Installation der UNICORE 6-Komponenten Die Installation der einzelnen UNICORE 6-Komponenten erfolgt durch das Herunterladen und anschließende Einspielen von RPM-Archiven, die u. a. über die Webseiten zur D-Grid Referenzinstallation bezogen werden können. Daher beginnen die nachfolgenden Installationsworkflows für den Gateway, UNICORE/X, die XUUDB und das Target System Interface mit den beiden Aktivitäten `Download RPM` und `Install RPM`.

Zunächst ist aber in Abbildung 5.51 ein Workflow für die Installation der zuvor erwähnten UNICORE-Komponenten gezeigt, wobei die lineare Anordnung der Aktivitäten den zeitgleich schreibenden Zugriff auf die Datenbank der Paketverwaltung umgeht.

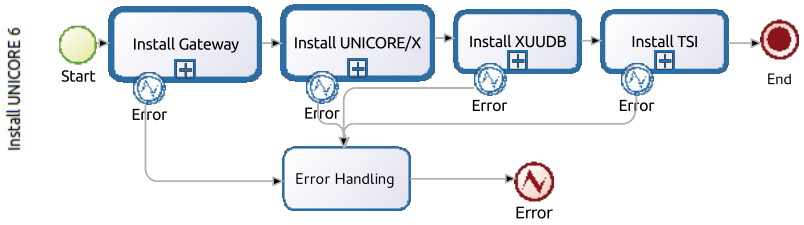


Abbildung 5.51: Workflow zur Installation des UNICORE 6-Komponenten

Die Konfiguration der einzelnen Komponenten erfolgt über die Ausführung eines zweiten, ebenso linearen Workflows, wie er in Abbildung 5.52 gezeigt ist. Jede der darin aufgeführten Aktivitäten startet die Konfiguration der zugehörigen Komponente über den Aufruf eines Subflows.
Da im Gegensatz zur Installation die Konfigurationsschritte der Komponenten voneinander unabhängig sind, lassen sich diese parallel ausführen. Ein Workflows, der dies ausnutzt, ist in Abschnitt 6.5 vorgestellt.

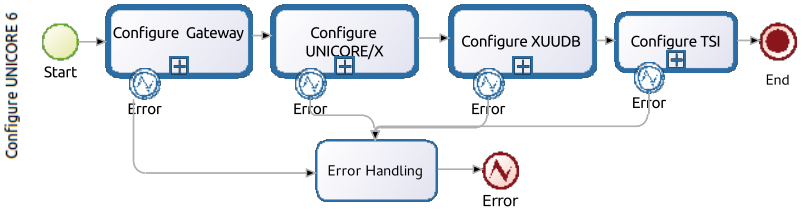


Abbildung 5.52: Workflow zur Konfiguration des UNICORE 6-Komponenten

Installation und Konfiguration des Gateways Nachdem das Herunterladen und Installieren der Gateway-Software in den ersten beiden Aktivitäten des Workflows (vgl. Abbildung 5.53) abgeschlossen ist, werden während der Ausführung der Aktivität `Change File Ownership` die Besitzrechte für die neu im System vorhandenen Dateien gesetzt. Als neuer Besitzer wird standardmäßig der Nutzer `njsadmin` verwendet. Von außen kann jedoch durch die Übergabe eines Arguments an die Aktivität ein anderer Nutzer angegeben werden. Abschließend löscht die in der Aktivität `Delete RPM` ausgeführte Applikation die zuvor heruntergeladene Gateway-Software wieder.

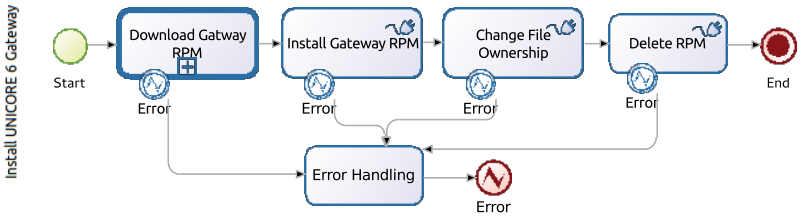


Abbildung 5.53: Workflow zur Installation des UNICORE 6 Gateways

Die Einrichtung des Gateways erfordert Anpassungen in einigen Konfigurationsdateien sowie die Erzeugung eines Truststores. Dieser enthält nach der Ausführung der Aktivität `Create Truststore` des zugehörigen Workflows (vgl. Abbildung 5.54) die Zertifikate aller von der European Policy Management Authority for Grid Authentication akkreditierten Certification Authorities. In einer weiteren Komponente des Workflows, der Aktivität

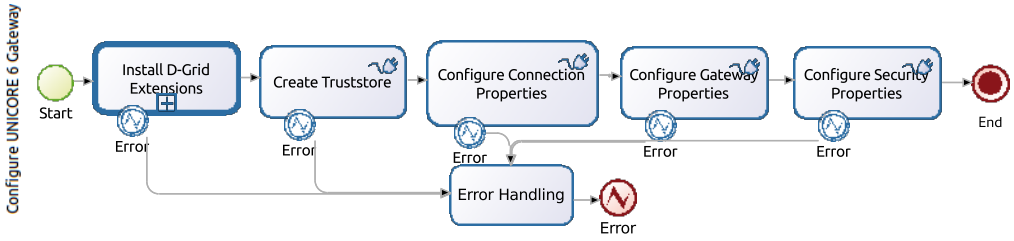


Abbildung 5.54: Workflow zur Konfiguration des UNICORE 6 Gateways

`Install D-Grid Extensions`, werden über die Einbindung eines Subflows Erwei-

terungen installiert, die für den Betrieb des Gateways innerhalb des D-Grid notwendig sind.

Auf dem Gateway sind in der Datei `connections.properties` Verbindungsinformationen, für z. B. die lokalen VSites, in Form von Schlüssel-Wert-Paaren abzulegen. Analog ist in der Datei `gateway.properties` der Rechnername und der Port zu spezifizieren, über den der Gateway-Dienst später von außen erreichbar sein soll. Die Änderungen an diesen beiden Dateien übernehmen einzelne Applikationen in den Aktivitäten `Configure Connection Properties` und `Configure Gateway Properties`. Die letzte Aktivität des Workflows erzeugt die Datei `security.properties`. In dieser sind die Pfade sowie Passwörter zu den vom Gateway verwendeten Keystore und Truststore festgehalten. Das Listing 5.19 zeigt die hierbei zum Einsatz kommende Groovy-Applikation, wobei die Argumente für beispielsweise `${securityPropertiesLocations}` und `${keystoreLocation}` von außen der Applikation bzw. Aktivität zugeführt werden.

```
def File f = new File ("${securityPropertiesLocations}")
def String output = ""
output = output + "keystore=${keystoreLocation}\n"
output = output + "keystorepassword=${keystorePassword}\n"
output = output + "truststore=${trustStoreLocation}\n"
output = output + "truststorepassword=${truststorePassword}\n"
f.write(output)
```

Listing 5.19: Erzeugen der Datei `security.properties` (Groovy-Applikation)

Die Workflows für die Installation der UNICORE/X- (vgl. Abbildung 5.55), der XUUDB- (vgl. Abbildung 5.58) und der Target System Interface-Komponente (vgl. Abbildung 5.60) verlaufen analog zur Installation des Gateways ab, weshalb im Folgenden nur die Workflows für die Konfiguration dieser Komponenten beschrieben sind.

Konfiguration des UNICORE/X Während der Konfiguration der UNICORE/X-Komponente werden in den ersten zwei Aktivitäten des Workflows (vgl. Abbildung 5.56) die später verwendeten Key- bzw. Truststores angelegt. Weiterhin werden die notwendigen Passwörter zum Öffnen der beiden Stores in der Datei `wsrflite.xml` hinterlegt.

Nach dem Abschluss der beiden Aktivitäten erfolgen über einen Subflow-Aufruf in der Aktivität `Modify IDB` Anpassungen an der Incarnation Database. Diese besteht u. a. aus den Abschnitten `Applications`, `Scripts` und `Resources`, in denen – abhängig von dem über das Target System Interface angesprochenen Batchsystem – Einstellungen vorgenommen werden müssen. Im Abschnitt `Resources` sind beispielsweise die Architektur, die Anzahl an CPUs und der Hauptspeicher der Rechenknoten des Batchsystems anzuge-

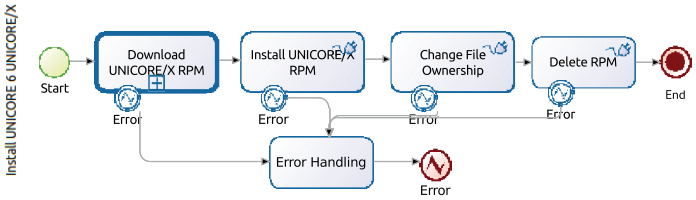


Abbildung 5.55: Workflow zur Installation des UNICORE 6 UNICORE/X

ben. All diese Einstellungen an der Incarnation Database realisiert der in Abbildung 5.57 gezeigte Workflow, der auf menschliche Interaktion setzt. Neben diesen Informationen aus der

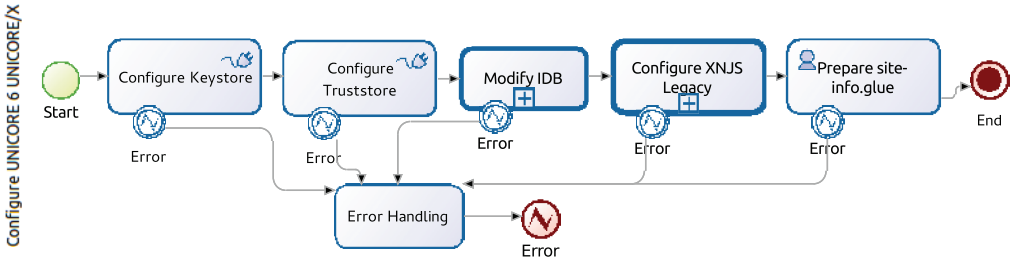


Abbildung 5.56: Workflow zur Konfiguration des UNICORE 6 UNICORE/X

Incarnation Database werden weitere GLUE-spezifische Attribute, die in der Datei `site-info.glue` gespeichert sind, an den Common Information Service der UNICORE Middleware übermittelt. Der Common Information Service selbst ist der Informationsdienst der Middleware und entspricht dem siteBDII bei gLite. Während der Ausführung der Aktivität `Modify site-info.glue` des Workflows zur Konfiguration des UNICORE/X werden diese Ressourcen-spezifischen Werte, z. B. der Eigentümer/ Betreiber, in die Datei `site-info.glue` eingetragen.

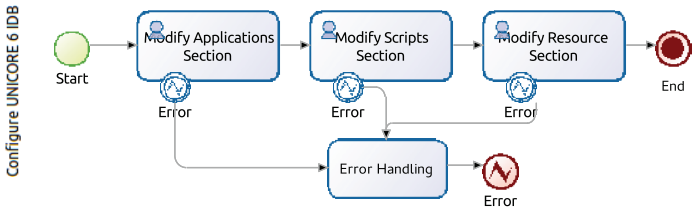


Abbildung 5.57: Workflow zur Konfiguration der UNICORE 6 IDB

Konfiguration der XUUDB Der XUUDB-Dienst der UNICORE Middleware besteht aus einer Client- und einer Server-Komponente, die separat voneinander durch den in Abbildung 5.59 gezeigten Workflow konfiguriert werden.

Die beiden Aktivitäten `Configure Client` und `Configure Server` überneh-

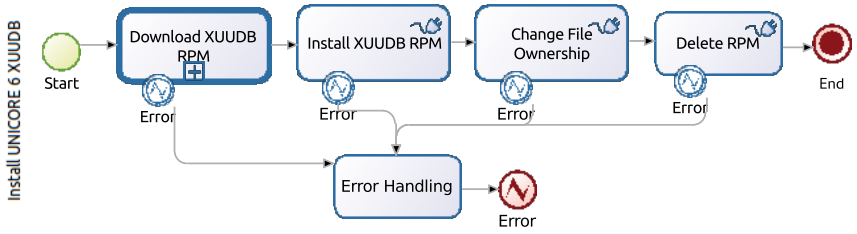


Abbildung 5.58: Workflow zur Installation der UNICORE 6 XUUDB

men diese Aufgabe und erzeugen die Dateien `xuudb_client.conf` respektive `xuudb_server.conf` in den entsprechenden Verzeichnissen. Das Listing 5.20 zeigt die für die Erzeugung der Datei `xuudb_server.conf` zuständige Groovy-Applikation.

```
def String output = ""
output += "xuudb_http_host=https://${xuudb_host}\n"
output += "xuudb_http_port=34463\n"
output +=  "xuudb_use_ssl=true\n"
output +=  "xuudb_type=dn\n"
output += "xuudb_data_file=/var/lib/unicore-xuudb/data/UnicoreUserDB.data
    \n"
output += "xuudb_keystore_file=${server_keystore}\n"
output += "xuudb_keystore_password=${server_keystore_password}\n"
output += "xuudb_keystore_type=PKCS12\n"
output += "xuudb_truststore_file=${server_truststore}\n"
output += "xuudb_truststore_password=${server_truststore_password}\n"
output += "xuudb_truststore_type=JKS\n"

def File f = new File ("${path2ServerConfig}")
f.write(output)
```

Listing 5.20: Erzeugen der Datei `xuudb_server.conf` (Groovy-Applikation)

Die Werte für alle Parameter, wie `${server_truststore}` und `${xuudb_host}`, werden von außen der Applikation bzw. Aktivität zugeführt.

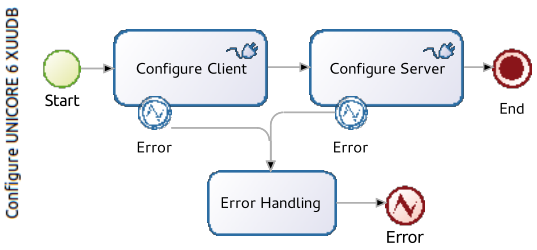


Abbildung 5.59: Workflow zur Konfiguration des UNICORE 6 XUUDB

Konfiguration des TSI Die Konfiguration des UNICORE 6 Target System Interface setzt sich aus zwei Aktivitäten zusammen, wie der zugehörige Workflow in Abbildung 5.61 zeigt. Während der ersten Aktivität, `Configure TSI Properties`, wird in der Datei

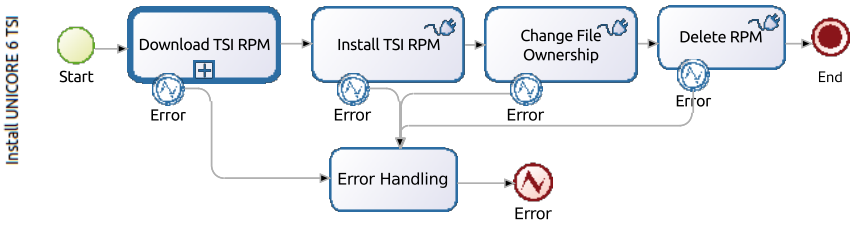


Abbildung 5.60: Workflow zur Installation des UNICORE 6 TSI

`tsi.properties` u. a. der Pfad zum Installationsverzeichnis des Target System Interface gesetzt. In der Aktivität `Modify TSI Script` erfolgt das Setzen des Pfades zu den ausführbaren Dateien der Batchsystem Client-Software (vgl. Abschnitt 4.8.2) in der Datei `tsi/tsi`. Das Target System Interface verwendet diese Client-Software für die Kommunikation mit dem Batchsystem Server.

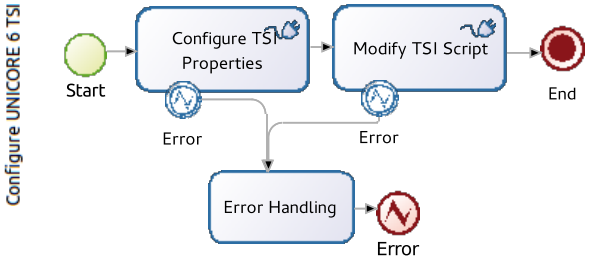


Abbildung 5.61: Workflow zur Konfiguration des UNICORE 6 TSI

Die vorherigen Abschnitte stellen Workflows für die Installation und auch Konfiguration einzelner Dienste der Grid Middlewares gLite, Globus Toolkit und UNICORE vor. Genau diese drei Middleware spielen in der deutschen Grid Initiative D-Grid, die nachfolgend als weiteres Anwendungsszenario dient, eine wesentliche Rolle.

5.4 D-Grid Referenzinstallation

Die deutsche Grid-Initiative D-Grid[23] entstand im Jahr 2004 durch eine gemeinsame Anstrengung von Wissenschaft und Industrie. Zu Beginn lagen die Ziele dieser Initiative in der Schaffung und dem nachhaltigen Betrieb einer nationalen e-Science-Infrastruktur sowie

[23] `http://www.d-grid.de/`

deren spätere Integration in das europäische Umfeld.
Andere Länder, wie beispielsweise die USA und Großbritannien, konnten zum Zeitpunkt der Gründung des D-Grid ähnlich ausgerichtete Initiativen vorweisen, die jedoch bereits seit langem aktiv waren. Dort ist also – im Vergleich zu Deutschland – der Bedarf für eine e-Science-Infrastruktur auf nationaler Ebene durch Wissenschaftler als auch Förderer frühzeitig erkannt worden.

Zeitlich betrachtet untergliedert sich die Initiative hinter dem D-Grid in drei teils überlappende Phasen (vgl. Abbildung 5.62), wobei die erste im dritten Quartal des Jahres 2005 startete und im Jahr 2008 endete. Während dieser Phase entwickelte man im D-Grid u. a. Basisdienste zur Unterstützung des Betriebs, wozu beispielsweise auch der Grid Resource Registration Service (GRRS), ein Registrierungsdienst für Grid-Ressourcen, zählt. Des Weiteren wurde mit [BDE+07] die erste Version eines Betriebskonzepts für die Rechen- und Speicherressourcen des D-Grid veröffentlicht. Auf organisatorischer Ebene schlossen sich in dieser ersten Phase deutsche Wissenschaftler zu virtuellen Organisationen, wie etwa High Energy Physicists Community Grid (HEPCG) und Collaborative Climate Community Data and Processing Grid (C3Grid), zusammen.
Zur Stützung der Nachhaltigkeit verschob sich in der zweiten Phase des D-Grid der Fokus

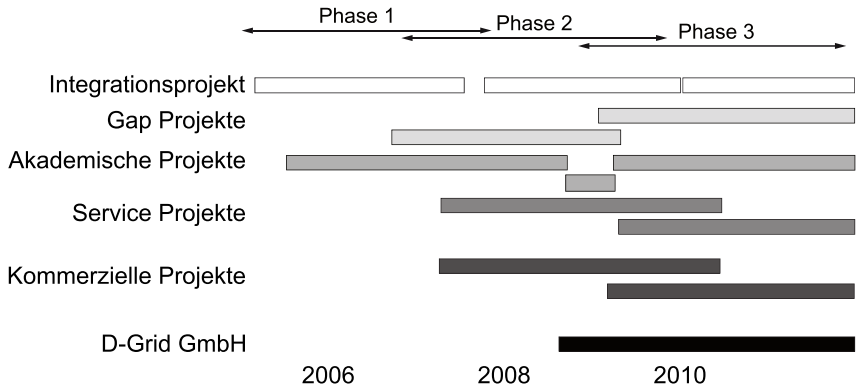


Abbildung 5.62: Zeitliche Gliederung des D-Grid in Phasen, Quelle: [Sch10]

von der rein akademischen Nutzung der Infrastruktur hin zu der langfristig angestrebten Mischnutzung durch Anwender aus dem akademischen und dem kommerziellen Umfeld. Ein wesentlicher Bestandteil der zweiten Phase war daher die Entwicklung von Werkzeugen zur Buchführung über verbrauchte Ressourcen. Zusätzlich wurden in Zusammenarbeit mit Independent Software Vendors (ISVs) Lösungen für die Problematik der Nutzung von Software-Lizenzen in Grid-Umgebungen gesucht.
Im April des Jahres 2008 gab es einen weiteren Aufruf des Bundesministerium für Bildung und Forschung zu Projektvorschlägen für die dritte Phase des D-Grid. Neben der Fortsetzung gemeinsamer Projekte von Wirtschaft und Wissenschaft stand bei diesem Aufruf die

Erzeugung professioneller, nachhaltiger und nutzerfreundlicher Dienste im Vordergrund. Die nachfolgend beschriebene D-Grid Referenzinstallation entstand in einer sehr frühen Phase des D-Grid und sollte u. a. den Betreibern der Grid Middleware-Dienste eine Installations- und Konfigurationshilfe sein. Zwei der wesentlichen Komponenten der Referenzinstallation sind i) ein detaillierter Leitfaden für die Installation und Konfiguration der aktuell empfohlenen Middleware-Komponenten sowie ii) die Bereitstellung der zur Umsetzung des Leitfadens notwendigen Software-Pakete.
Der Leitfaden, der in Form eines Wikis[24] vorliegt, bildet eine gute Ausgangsposition für die Umsetzung in einen Workflow. Mit Hilfe dieses Workflows ließe sich perspektivisch betrachtet die Installation und Konfiguration einer D-Grid Ressource, die eine oder mehr Middlewares unterstützt, automatisieren.

Zur besseren Übersicht beschreiben die folgenden Abschnitte sowohl die Struktur der Referenzinstallation als auch einige ausgewählte Komponenten näher.

5.4.1 Struktur

Nach Gründung der D-Grid-Initiative, bestehend aus dem D-Grid Integrationsprojekt (DGI) und mehreren Community-Projekten, suchte man nach einer durch möglichst viele Communities nutzbaren Konfiguration der Rechen- und Speicherressourcen des D-Grid. Da in den einzelnen Communities die Entscheidung für bzw. gegen eine der Grid Middlewares gLite, Globus Toolkit und UNICORE noch nicht absehbar war, entstand die Idee einen Ressourcen-Prototypen zu entwickeln, der alle drei Middlewares unterstützt. Dieser Prototyp bildet den Vorläufer für die erste Version der Referenzinstallation, deren Komponenten in Abbildung 5.63 dargestellt sind. Das Globus Toolkit Frontend, das gLite CREAM Compute Element, der UNICORE Network Job Supervisor und der Batchsystem Server inklusive der Rechenknoten sind bzgl. deren Installation und Konfiguration in den Abschnitten 5.3.2, 5.3.3, 5.3.4 und 4.8 vorgestellt. Des Weiteren zählen neben den drei Compute Middlewares mit dCache (vgl. [FG06]) und Open Grid Services Architecture (OGSA)-Data Access and Integration (DAI) (vgl. [KAA+05]) auch zwei Storage Middlewares zur Referenzinstallation. Für diese beiden besteht ebenso die Möglichkeit, Workflows für die Installation und Konfiguration zu entwerfen, was hier aber nicht weiter verfolgt wird. Der im Betriebskonzept des D-Grid empfohlene interaktive Knoten bildet eine weitere Komponente der Referenzinstallation, die Workflows für dessen Installation und Konfiguration sind in Abschnitt 5.4.3 beschrieben.

Eine Übersicht über die für die erste Version der Referenzinstallation verwendeten Kombinationen aus Betriebssystem und Grid Middleware ist in Tabelle 5.2 dargestellt. Dabei

[24] `http://dgiref.d-grid.de/wiki/Introduction`

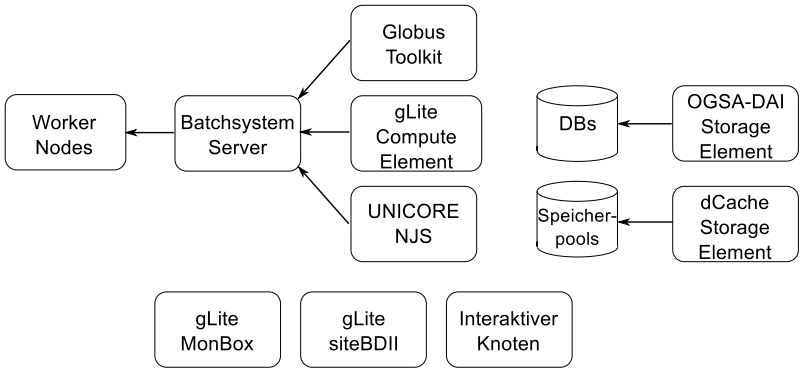


Abbildung 5.63: Komponenten der ersten Version der D-Grid Referenzinstallation

ist hervorzuheben, dass SUSE Linux Enterprise Server (SLES) im Gegensatz zu Scientific Linux lizenz- und damit kostenpflichtig ist. Dies führte in den weiteren Versionen der Referenzinstallation nach und nach zur vollständigen Verdrängung von SUSE Linux Enterprise Server.

Grid Middleware	Betriebssystem	
	Scientific Linux 4	SLES 10
gLite Compute Element	X	
gLite siteBDII	X	
gLite MonBox	X	
Globus Toolkit Frontend		X
UNICORE Network Job Supervisor		X
dCache Storage Element	X	
OGSA-DAI Storage Element		X

Tabelle 5.2: Kombinationen aus Grid Middleware-Komponente und Betriebssystem

Nachfolgend sind in Ergänzung zu den bereits in Abschnitt 5.3 beschriebenen Workflows für die automatische Installation und Konfiguration der Middlewares Globus Toolkit, gLite und UNICORE weitere, D-Grid-spezifische Workflows vorgestellt. Der in Abbildung 5.64 dargestellte Workflow zeigt allerdings zunächst einen Weg zur (teil-)automatisierten Erzeugung einer D-Grid Ressource, welche alle drei vorgestellten Grid Middlewares unterstützt. Die einzelnen Aktivitäten `Setup gLite Services`, `Setup Globus Toolkit Services` und `Setup Unicore Services` leiten über Subflow-Aufrufe die Installation bzw. Konfiguration der verschiedenen Komponenten der Grid Middlewares ein. Dabei ist zu beachten, dass dieser Workflow – als auch der Workflow zur Installation der

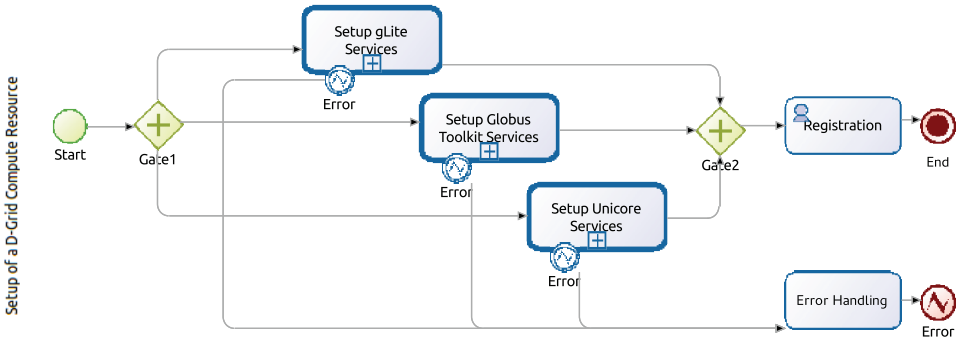


Abbildung 5.64: Workflow für das Setup der D-Grid Referenzinstallation

Komponenten einer gLite Grid Site (vgl. Abbildung 5.6) – in dieser Arbeit nicht implementiert ist. Beide Workflows müssen zur korrekten Abarbeitung in einzelnen Aktivitäten auf eine separate virtuelle Maschine zugreifen, d. h. beispielsweise, dass die Installation jeder der gLite Middleware-Komponenten in einer anderen virtuellen Maschine zu erfolgen hat. Die Umsetzung dieses Punktes stellt eine der zukünftigen Arbeiten bzw. Erweiterungen dar.

5.4.2 dgridmap-Skript

Verglichen mit anderen Grid-Initiativen zeichnet das D-Grid die von Beginn an vorhandene Unterstützung mehrerer Grid Middlewares aus. Viele der D-Grid-Ressourcen unterstützen zwei oder sogar alle drei in der Referenzinstallation vorgeschlagenen Middlewares und bieten so breitgefächerte Zugangsmöglichkeiten an. Eine im D-Grid Betriebskonzept verankerte Forderung, die beim Einsatz mehrerer Middlewares auf einer Ressource einzuhalten ist, ist die Abbildung eines VO-Mitglieds auf dasselbe lokale Nutzerkonto, unabhängig davon über welche Grid Middleware dieses auf die Ressource zugreift.

Diese vorgeschriebene feste Zuordnung können Ressourcenbetreiber in Eigenregie herstellen oder auf ein durch die D-Grid-Initiative bereitgestelltes Skript namens `dgridmap` (vgl. [FG10]) zurückgreifen. Laut [BDE+07] soll dieses Skript mindestens einmal täglich ausgeführt werden, so dass die kontinuierliche Aktualität der Autorisierungsinformationen für die einzelnen Middlewares gesichert ist. Intern baut das Skript unter Verwendung des Host-Zertifikats eine Verbindung zu einem am Forschungszentrum Jülich (FZJ) betriebenen Dienst auf, der in einer Datenbank nach den von der Ressource unterstützten virtuellen Organisationen sucht, die zugelassenen Mitglieder ermittelt, diese über Abbildungsvorschriften in das Middleware-spezifische Format wandelt und letztendlich an die aufrufende Ressource ausliefert.

Nachfolgend sind Beispiele für solche ausgelieferten Informationen bezogen auf die Middlewares Globus Toolkit, gLite und UNICORE vorgestellt. Neben diesen Middlewares

unterstützt das Skript `dgridmap` auch die Storage Middlewares dCache und OGSA-DAI.

- Für die Globus Toolkit Middleware liefert das Skript `dgridmap` eine Datei zurück, die direkt als `grid-mapfile` nutzbar ist und dem in Listing 5.21 dargestellten Format genügt.

```
"/C=DE/O=Organization/OU=OrgUnit/CN=Vorname_1 Nachname_1"
    localAccount_1
"/C=DE/O=Organization/OU=OrgUnit/CN=Vorname_2 Nachname_2"
    localAccount_2
"/C=DE/O=Organization/OU=OrgUnit/CN=Vorname_3 Nachname_3"
    localAccount_3
```

Listing 5.21: Beispiel für ein grid-mapfile

Hierbei wird der Grid-Nutzer anhand des Distinguished Name, welcher in dessen X.509-Zertifikat enthalten ist, auf ein lokales Nutzerkonto, abgebildet.

- Die gLite Middleware basiert zu einem gewissen Anteil auf dem Globus Toolkit und verwendet aus diesem Grund u. a. auch ein `grid-mapfile`. Des Weiteren unterstützen einzelne Komponente der Middleware die Attribut-basierte Autorisation. Dabei sind die im so genannten VO extension-Abschnitt des Stellvertreter-Zertifikats festgehaltenen Rollen- und Gruppenzugehörigkeiten von essentieller Bedeutung.

```
=== VO dech extension information ===
VO         : dech
subject    : /C=DE/O=GermanGrid/OU=TU-Dortmund/CN=Stefan Freitag
issuer     : /C=DE/O=GermanGrid/OU=Fraunhofer SCAI/CN=host/glite-io.
    scai.fraunhofer.de
attribute  : /dech/Role=NULL/Capability=NULL
timeleft   : 11:51:45
uri        : glite-io.scai.fraunhofer.de:15000
```

Listing 5.22: Beispiel für einen VO extension-Abschnitt

Das Attribut `/dech/Role=NULL/Capability=NULL` aus dem Beispiel in Listing 5.22 wird während des Autorisierungsvorgangs mit den im `voms-grid-mapfile` enthaltenen Einträgen (vgl. Listing 5.23) abgeglichen. Jede Zeile dieser Datei besteht aus einem Attribut und einem zugeordneten lokalen Nutzerkonto bzw. einer Menge solcher Konten[25] auf die VO-Mitglieder abgebildet werden können.

```
"/hepcg/admin/Role=softwareadmin/Capability=NULL" hpsgm
```

[25]Eine Gruppe von lokalen Konten ist durch einen vorangestellten Punkt zu erkennen, in diesem Beispiel etwa die Gruppe `hp`.

```
"/hepcg/admin/Role=softwareadmin" hpsgm
"/hepcg/Role=NULL/Capability=NULL" .hp
"/hepcg" .hp
```

Listing 5.23: Beispiel für ein voms-grid-mapfile

Ist die Abbildung des Nutzers über die im VO extension-Abschnitt mitgelieferten Attribute nicht möglich, so kann dieser noch über die Auswertung des Distinguished Names unter Zuhilfenahme des `grid-mapfiles` abgebildet werden.

- Im Gegensatz zur gLite und Globus Toolkit Middleware liefert das Skript `dgrid-map` für UNICORE 6 keine Datei, die direkt vom Network Job Supervisor bzw. der XUUDB verwendet werden kann. Die zurückgelieferte Datei enthält pro Zeile eine fortlaufende Nummer, ein Kürzel für die Ressource, ein oder mehrere Namen lokaler Nutzerkonten, das Schlüsselwort `user`, die Namen der zu den Nutzerkonten gehörenden VOs und den Distinguished Name des Nutzer-Zertifikats (vgl. Listing 5.24).

```
412;udo-grid.uni-dortmund.de;cm0005:ed0075:hp0007:op0004;user;dgcms:
    education:hepcg:dgops;CN=Stefan Freitag,OU=TU-Dortmund,O=
    GermanGrid,C=DE
```

Listing 5.24: Ausgabe des dgridmap-Skripts für UNICORE 6

Der Aufruf des Kommandos `unicore-xuudb-admin` mit der durch das `dgrid-map`-Skript zurückgelieferten Datei als Argument sorgt für die Übernahme der Nutzer in die XUUDB.

Ein Workflow (vgl. Abbildung 5.65) übernimmt in Abhängigkeit von der Grid Middleware neben dem Herunterladen und Installieren des Skripts `dgridmap` auch die Einrichtung eines `cron`-Jobs, der die tägliche Aktualisierung der Autorisierungsinformationen gewährleistet. Die Aktivität `Download dgridmap Software` kopiert das aus dem Internet heruntergeladene Skript standardmäßig in das Verzeichnis `/opt/d-grid/bin`. Im Anschluss daran modifiziert die Aktivität die Ausführungs- und Besitzrechte an der Datei. Abhängig vom Wert der Workflow-Variable `middleware_type` verzweigt die Kontrolle über das Gate 1 in eine der Aktivitäten für die Globus Toolkit, gLite bzw. UNICORE Middleware. Jede der Aktivitäten erzeugt entsprechend der gewählten Middleware einen Job im Verzeichnis `/etc/cron.d` und läuft am Gate 2 mit den anderen Aktivitäten zusammen.

Das Listing 5.25 zeigt die hinter der Aktivität `GTK Cron Job` liegende Groovy-Applikation. Sie legt einen cron-Job an, der das `dgridmap`-Skript zur Erzeugung der temporären Datei `/tmp/tmp_gridmap.txt` einsetzt. Weiterhin ersetzt der Job das bisherige `grid-mapfile` durch diese Datei und setzt die Besitz- und Ausführungsrechte.

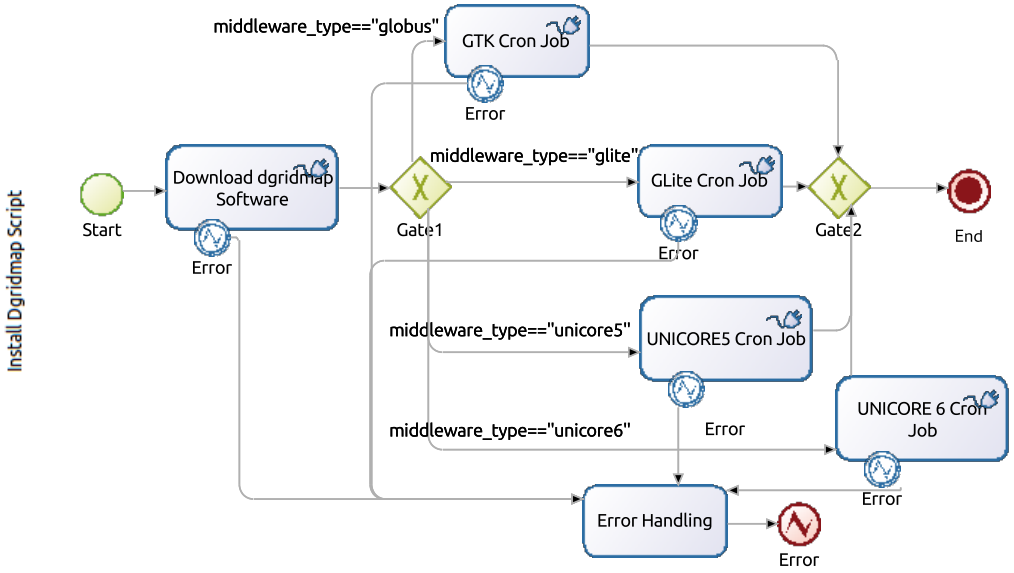


Abbildung 5.65: Workflow zur Installation und Konfiguration des dgridmap-Skripts

```
def outputFile = new File ('${pathCronFile}')

fileContent = "#!/bin/sh\n"
fileContent += "${path2dgridmap}/dgridmap --output-g /tmp/tmp_gridmap.txt\n"
fileContent += "rm -f /etc/grid-security/grid-mapfile\n"
fileContent += "cp /tmp/tmp_gridmap.txt /etc/grid-security/grid-mapfile\n"
fileContent += "rm -f /tmp/tmp_gridmap.txt\n"
outputFile << fileContent

command = "chown root:root ${pathCronFile}"
proc = command.execute()
proc.waitFor()

command = "chmod a+x ${pathCronFile}"
proc = command.execute()
proc.waitFor()
```

Listing 5.25: Erzeugen des cron-Jobs für das Globus Toolkit (Groovy-Applikation)

5.4.3 Interaktiver Knoten

Der interaktive Knoten ist keine unbedingt notwendige Komponente der Referenzinstallation, jedoch wird Ressourcenbetreibern dessen Bereitstellung durch das Betriebskonzept

nahegelegt. Laut dem Betriebskonzept soll dieser Knoten weiterhin über GSI-SSH und/oder UNICORE-SSH erreichbar sein und Nutzern vornehmlich zu Testzwecken, wie dem Übersetzen und Debuggen ihrer Software-Entwicklungen dienen. Ebenso soll über diesen Knoten auch die Installation von Software in das VO-eigene Softwareverzeichnis auf der Grid-Ressource möglich sein.

Ein Workflow für die Einrichtung des später über GSI-SSH erreichbaren, interaktiven Knotens ist in Abbildung 5.66 gezeigt. Nach dem (De-)Aktivieren von Diensten auf Be-

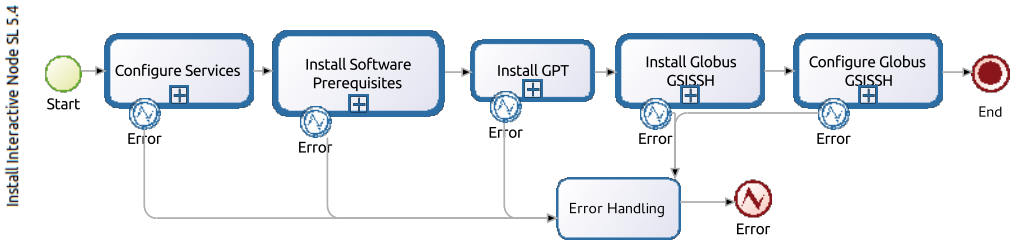


Abbildung 5.66: Workflow zur Installation des interaktiven Knotens

triebssystemebene während der Aktivität `Configure Services` und dem Einspielen aller Software-Abhängigkeiten durch die Aktivität `Install Software Prerequisites`, erfolgt die Installation des Grid Packaging Tools durch die Ausführung der Aktivität `Install GPT` (vgl. Abbildung 5.22). Die gerade erwähnten Aktivitäten kommen bereits bei zuvor beschriebenen Workflows zum Einsatz (vgl. z. B. Abbildung 5.22), weswegen an dieser Stelle auf eine detaillierte Beschreibung verzichtet wird.

Die hier angewandte Vorgehensweise zur Installation der GSI-SSH-Software weicht übrigens von der in Abschnitt 5.3.3.2 für die Globus Toolkit Middleware Version 5 vorgestellten ab, da die dort verwendeten Software-Repositories nicht die notwendigen Pakete enthalten. Stattdessen wird in der Aktivität `Install Globus GSISSH` ein Subflow (vgl. Abbildung 5.67) aufgerufen, der u. a. das Software-Archiv, welches den Globus Toolkit 5-Quellcode enthält, herunterlädt und extrahiert. Analog zur Installation des Globus Toolkit 4 (vgl. Abschnitt 5.3.3.1) kompiliert die Aktivität `Compile Source` des Subflows den Quellcode, bevor das Kompilat durch die Aktivität `Installation` in ein zuvor spezifiziertes Verzeichnis kopiert wird. Der letzte Schritt des Subflows löscht das Software-Archiv sowie temporär extrahierte Dateien.

Der Installation der notwendigen Komponenten des Globus Toolkit schließt sich deren Konfiguration an. Der hierzu eingesetzte Workflow ist in Abbildung 5.68 gezeigt. Die Applikation hinter der Aktivität `Copy init.d Script` des Workflows (vgl. Listing 5.26) kopiert das für den Start des Dienstes aufzurufende Skript in das Verzeichnis `/etc/init.d/`. Zeitgleich erfolgt zudem die Ersetzung der Zeichenkette `Provides: ssh` durch `Provides: gsissh` in dem Skript.

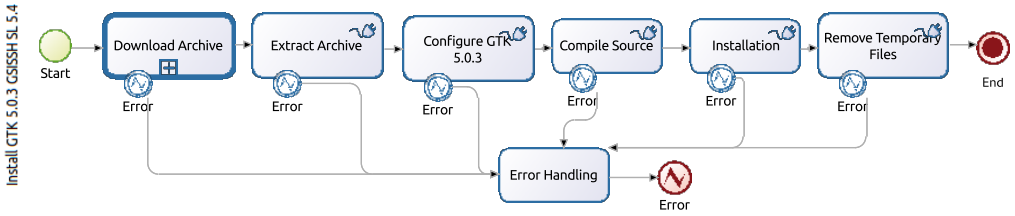


Abbildung 5.67: Workflow zur Installation der GSISSH-Komponente

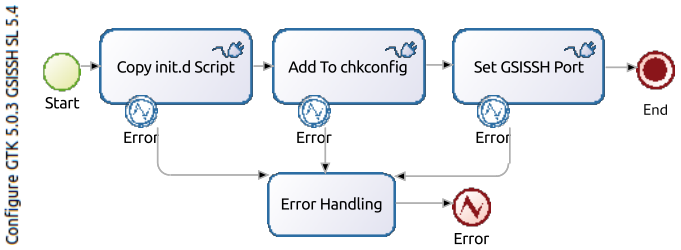


Abbildung 5.68: Workflow zur Konfiguration der GSISSH-Komponente

Nachfolgend trägt die Aktivität `Add To chkconfig` den Dienst `gsissh` in die Liste der beim Hochfahren des Systems zu startenden Dienste ein. Die letzte Aktivität des Workflows `Set GSISSH Port` modifiziert in einer Konfigurationsdatei die Port-Nummer, auf der der GSI-SSH-Server später eingehende Verbindung annimmt, auf den Wert 2222.

```
def String command = "cp /usr/local/globus/sbin/SXXsshd /etc/init.d/
    gsissh"
def Process proc = command.execute()
proc.waitFor()

command = "sed -i 's/Provides: ssh/Provides: gsissh/g' /etc/init.d/gsissh
    "
proc = command.execute()
proc.waitFor()
```

Listing 5.26: Kopieren des init.d-Skripts für GSISSH (Groovy-Applikation)

5.4.4 gLite User Interface

Für die Einreichung von Rechen-Jobs an die verschiedenen Ressourcen des D-Grid ist eine Middleware-spezifische Client-Software notwendig. Um Grid-Nutzern bzw. VO-Mitgliedern die Installation dieser Software auf ihren eigenen Rechner zu ersparen, bietet die D-Grid-Initiative für die gLite Middleware ein zentrales User Interface (UI) am Forschungs-

zentrum Karlsruhe (FZK) an[26]. Die Installation eines solchen User Interface ist üblicherweise auf zwei Weisen möglich:

- Automatisch per Paketverwaltungssoftware
  Für diese Variante sind – analog zur Installation anderer gLite Middleware Dienste – die Informationen zu mehreren Software-Repositories im System zu hinterlegen. Im Anschluss daran ist die Installation des Metapakets `glite-UI` möglich. Dieses beinhaltet eine Liste von Abhängigkeiten zu den für das User Interface erforderlichen Software-Paketen, welche daher zusammen mit dem Metapaket installiert werden.

- Manuell über komprimierte Archivdateien
  Die User Interface Software liegt in zwei separaten Archiven vor. Zum einen handelt es sich dabei um ein Archiv, welches die gLite Software-Komponenten enthält und zum anderen um ein Archiv, über dessen Einspielen die Abhängigkeiten der Komponenten zu anderer Software aufgelöst werden.

Die Abbildung 5.69 zeigt einen Workflow für letzteren Fall, wobei die Bedeutung der Aktivitäten `Configure Services` und `Install EGI Trustanchors` bereits in vorherigen Abschnitten beschrieben ist. Diese werden durch Subflow-Aufrufe in den beiden Aktivitäten `Install GLite User Interface Software` und `Configure GLite User Interface Software`, in denen die gLite User Interface Software installiert bzw. konfiguriert wird, ergänzt.

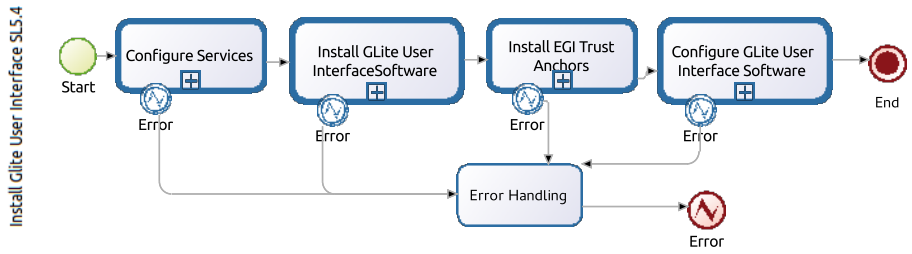


Abbildung 5.69: Workflow zur Installation eines gLite User Interface

Die ersten vier Aktivitäten des Workflows für die Installation der Software (vgl. Abbildung 5.70) laden die beiden erforderlichen Archive aus dem Internet herunter und extrahieren diese in das spätere Zielverzeichnis. Nach dem Abschluss der Extraktion werden die zwei heruntergeladenen Archive nicht länger benötigt und durch die Aktivitäten `Delete UI Archive` und `Delete External Dependencies Archive` vom System entfernt.

[26] Für die beiden anderen, im D-Grid unterstützten Middlewares Globus Toolkit und UNICORE existiert kein äquivalentes Angebot.

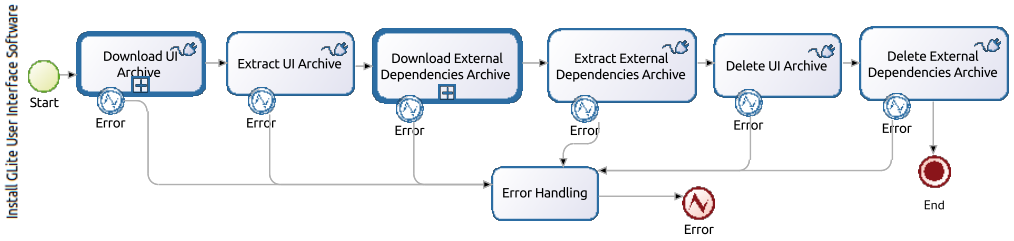


Abbildung 5.70: Workflow zur Installation der gLite User Interface Software

Während der Konfiguration des gLite User Interface durch einen Workflow (vgl. Abbildung 5.71) hinterlegen einzelne Aktivitäten die erforderlichen Dateien `users.conf`, `groups.conf` sowie VO-spezifische Informationen an entsprechende Stellen im Dateisystem. In der letzten Aktivität des Workflows erfolgt über die Einbindung eines Subflows der Aufruf des Werkzeugs YAIM.

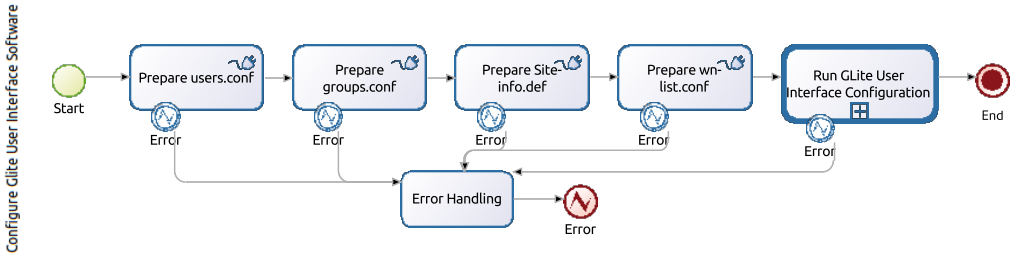


Abbildung 5.71: Workflow zur Konfiguration eines gLite User Interface

Kapitel 6

Evaluation

Dieses Kapitel wendet sich einerseits dem Verfahren zur Erstellung der in den vorherigen Kapiteln präsentierten Workflows zu, andererseits zeigt es Erweiterungen bzw. mögliche Verbesserungen des Verfahrens auf.

Die Ausgangssituation in puncto Workflows ist in Abschnitt 6.1 zusammengefasst. Eines der herausragendes Ergebnisse dieses Punktes ist die Zerlegung der Installations- und Konfigurationsvorgänge für verschiedene Grid Middleware-Dienste in modulare Aktivitäten und deren Zusammenfügung zu Workflows. Die Austauschbarkeit dieser modularen Aktivitäten verdeutlicht Abschnitt 6.2 anhand mehrerer Beispiele. Diese beziehen sich auf den Austausch von mit den Aktivitäten verknüpften Applikationen als auch Subflows.

Neben der Austauschbarkeit spielt der Aspekt der Wiederverwendbarkeit einmal erstellter Aktivitäten, Applikationen und Subflows eine nicht minder wichtige Rolle. Viele der in Kapitel 5 vorgestellten Workflows für die Installation der Grid Middleware-Komponenten greifen die Wiederverwendbarkeit bereits auf, etwa bei der Installation der European Grid Initiative Trustanchor. In dem Abschnitt 6.3 sind mit einem Subflow zur Hinterlegung der YUM Repository-Information und einem Workflow für das `make`-basierte Kompilieren und Installieren von Software zwei weitere Workflows vorgestellt, die diesem Aspekt Rechnung tragen.

Durch die kontinuierliche Verfeinerung von Workflows steigt die Anzahl der darin eingebundenen Aktivitäten und somit unvermeidlich die Komplexität. Unter diesem Gesichtspunkt rückt die Behandlung von auftretenden Ausnahmen bzw. Fehlern mehr und mehr in den Vordergrund. Eine solche Behandlung ist nicht Bestandteil der in dieser Arbeit präsentierten Workflows und stellt somit eine zukünftige Erweiterung dar. Aufgrund des modularen Aufbaus gestaltet sich die Integration einer Fehlerbehandlung einfach. In Abschnitt 6.4 sind hierzu Möglichkeiten vorgestellt und an einem Beispiel verdeutlicht.
Des Weiteren ist in Abschnitt 6.5 die Parallelisierbarkeit von Aktivitäten bzw. Subflows

aufgezeigt, wodurch eine reduzierte Ausführungszeit für Workflows erzielbar ist.

Die einmal erstellten Workflows können als Vorlage dienen, wenn es darum geht, für weitere Betriebssysteme oder Software-Komponenten Workflows zur automatischen Installation und Konfiguration zu erzeugen. Am Beispiel der Unterstützung eines weiteren Betriebssystems, hier Debian Linux[1], ist in Abschnitt 6.6 die Adaption existierender Workflows beschrieben.

Nach der Untersuchung verschiedener Workflow-Eigenschaften, wie der Austauschbarkeit, Wiederverwendbarkeit und der Erweiterung, ist in Abschnitt 6.7 mit der Prozess- bzw. Workflow-Optimierung ein weiterer bislang unzureichend behandelter Punkt aufgegriffen. Die Evaluation schließt mit der Betrachtung der Nutzung entworfener Workflows über eine Web-Schnittstelle ab.

6.1 Workflow-Erstellung

Für den Bereich der Grid Middlewares existieren bislang weitestgehend nur textuelle Beschreibungen der Installation und Konfiguration einzelner Dienste. In der gLite Middleware unterstützt neben diesen Beschreibungen beispielsweise das Werkzeug YAIM die Konfiguration. Es fehlt jedoch an Workflows, die i) auf den Beschreibungen basieren und ii) die automatische Installation und die Konfiguration einzelner Middleware-Dienste bis hin zur Bereitstellung einer LaaS-Umgebung[2] ermöglichen.

Am Institut für Roboterforschung (IRF) wurde in der ersten Phase der D-Grid-Initiative (vgl. Abschnitt 5.4) eine Machbarkeitsstudie durchgeführt, die als Ergebnis die gleichzeitige Nutzbarkeit eines lokalen Batchsystems durch die Grid Middlewares gLite, Globus Toolkit und UNICORE zeigt (vgl. [AFF+09]). Diese Studie bildet die Grundlage für die später aus der Taufe gehobene D-Grid Referenzinstallation.
Ein weiteres Ergebnis der Studie sind Shell-Skripte, in denen die durchgeführten Installations- und Konfigurationsschritte zur besseren Reproduzierbarkeit fixiert sind. Ebenso ist die Abarbeitungsreihenfolge der einzelnen Skripte untereinander festgehalten. Beides zusammen stellt den einen Teil der Grundlagen für die Workflow-Erstellung dar. Der andere Teil entspringt der kontinuierlichen Erweiterung und Verbesserung der Skripte, basierend auf der periodisch durchgeführten Validierung der Installations- und Konfigurationsanleitungen der Referenzinstallation.

Diese Kombination von Anleitungen und Shell-Skripten für die einzelnen Grid Middle-

[1] `http://www.debian.org/`

[2] Unter einer Landscape as a Service (LaaS)-Umgebung ist eine Menge von verteilten Diensten zu verstehen, die zusammen installiert und konfiguriert werden sollen. Beispielsweise sind alle für eine gLite-basierte Grid-Ressource notwendigen Dienste als LaaS kapselbar.

ware-Dienste bildet somit den Ausgangspunkt für die Erstellung der in dieser Arbeit präsentierten Workflows. Ein Vorteil, der sich direkt aus der Nutzung der Anleitungen ergibt, ist die Existenz einer linearen Anordnung der einzelnen Schritte. Diese lineare Anordnung ist implizit bedingt durch die üblicherweise manuelle Ausführung der Einzelschritte durch einen Systemadministrator. Somit ist aus einer Anleitung mit wenig Aufwand ein linearer Workflow, der für jeden Schritt der Anleitung eine Workflow-Aktivität enthält, konstruierbar.
Hinter den Aktivitäten des Workflows verbergen sich initial keine Applikationen (vgl. Abbildung 6.1, links), die beim Aufruf der jeweiligen Aktivität ausgeführt werden. Allerdings bietet sich eine Verbindung der vorhandenen Skripte – umgesetzt in Applikationen – mit den Aktivitäten an (vgl. Abbildung 6.1, rechts), wobei die Workflow-Beschreibungssprache dies unterstützen muss. In diesem Kontext ist in Abschnitt 3.2.4 die Evaluation zwei sol-

Abbildung 6.1: BPMN-Aktivität und um eine Applikation erweitere Aktivität

cher Sprachen beschreiben. Aufgrund der Vorteile der Business Process Modeling Notation gegenüber der XML Process Definition Language ist diese als Realisierungssprache für die Workflows gewählt.

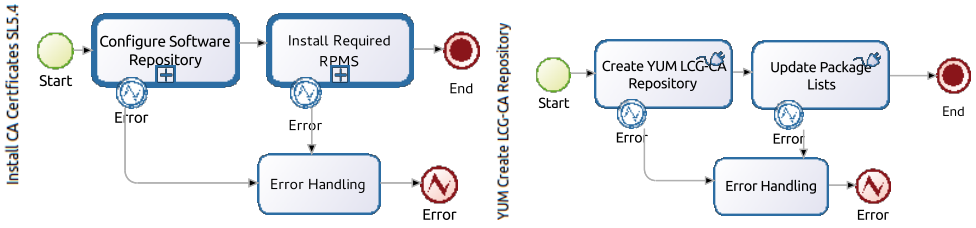
6.2 Austauschbarkeit von Applikationen und Subflows

Die Überführung von Anleitungen und Shell-Skripten zur Installation von Grid Middleware-Diensten in modular aufgebaute und strukturierte Workflows bringt viele Vorteile mit sich. Einer dieser Vorteile ist die einfache Austauschbarkeit einzelner Applikationen oder auch ganzer Subflows durch alternative Varianten. Letztere können beispielsweise dieselbe Funktionalität in einer verbesserten bzw. optimierten Variante bereitstellen. Insgesamt ergibt sich so ein hoher Grad an Wiederverwendbarkeit für einen einmal erstellten Workflow und dessen Komponenten.
Nachfolgend sind zwei Workflows vorgestellt, an denen dieser Vorteil verdeutlicht wird. Beide Workflows entstanden im Rahmen dieser Arbeit und aufgrund von Weiterentwicklungen innerhalb der Grid Middlewares mussten Teile der Workflows ausgetauscht werden.

Einspielen der LCG-CA- und EGI Trustanchor-Metapakete In der gLite Middleware erfolgt die Authentifizierung der Grid-Nutzer an den verschiedenen Diensten wie dem Compute Element dem Workload Management System anhand von X.509-Zertifikaten bzw.

deren kurzlebigen Stellvertreter-Zertifikaten. So verarbeitet der VOMRS-Dienst den Inhalt des X.509-Nutzerzertifikats, insbesondere welche Certification Authority (CA) das Zertifikat ausgestellt hat. Diese Information hilft bei dem Abgleich des Zertifikats mit dem Inhalt der Certificate Revocation List (CRL)[3] der Certification Authority.
Damit dieser Mechanismus greifen kann, müssen die Middleware-Dienste Zugriff auf die Zertifikate der Certification Authorities und die Certificate Revocation Lists haben. Des Weiteren ist eine periodische, üblicherweise tägliche Aktualisierung der Certificate Revocation Lists notwendig.

In den Versionen 3.1 und 3.2 der gLite Middleware wurde dies lange Zeit durch die Installation des Metapakets `lcg-CA` realisiert. Vor der Installation des Metapakets sind Informationen über das zugehörige Software-Repository im System zu hinterlegen. Ein Workflow, der diese beiden Aktivitäten durchführt, ist in Abbildung 6.2(a) dargestellt. Durch die



(a) Installation des LCG-CA-Metapakets (b) Erzeugen der Repository-Konfiguration

Abbildung 6.2: Workflows für die Installation des LCG-CA-Metapakets

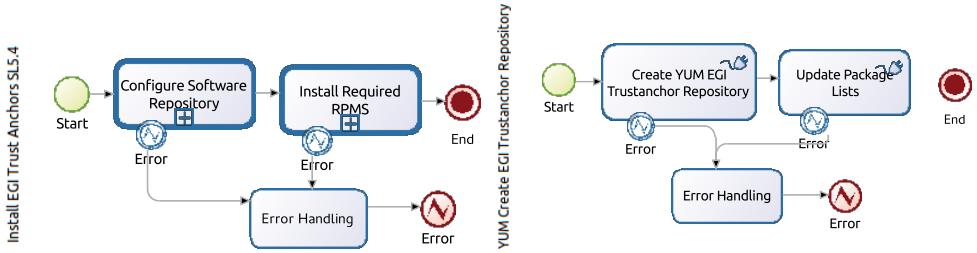
Aktivität `Configure Software-Repository` erfolgt unter Verwendung eines Subflows (vgl. Abbildung 6.2(b)) das Hinterlegen der Repository-Konfiguration im Verzeichnis `/etc/yum.repos.d`. Das Einspielen des Metapakets geschieht im Anschluss hieran durch die Ausführung der Aktivität `Install Required RPMS`.

Mit dem Übergang von der Enabling Grids for E-science-Initiative zur European Grid Initiative gibt es gegen Ende des Jahres 2010 diesbezüglich eine Umstellung. Es wurde entschieden, die Verwendung des Metapakets `lcg-CA` in der European Grid Initiative einzustellen. An der Initiative partizipierende Ressourcen-Betreiber sind stattdessen dazu angehalten, das Metapaket `ca-policy-egi-core` zu installieren, welches sich in einem anderen Software-Repository befindet. Die zwei Metapakete enthalten initial die Zertifikate derselben Certification Authorities, dies kann sich zukünftig jedoch ändern und so eine Auftrennung zwischen den durch die European Grid Initiative und das LHC Computing Grid unterstützen Certification Authorities darstellen.

Die durchzuführenden Workflows zur Einrichtung der EGI Trustanchor sind in den Ab-

[3] Eine Liste, die die Seriennummern ungültiger Zertifikate enthält. Ist die Seriennummer des Nutzerzertifikats Teil dieser Liste, wird der Nutzer vom Dienst abgewiesen.

bildungen 6.3(a) und 6.3(b) gezeigt. Analog zur Installation des Metapakets `lcg-CA` wird zunächst in der Aktivität `Create YUM EGI Trustanchor Repository` die notwendige Information über das zu verwendende Software-Repository im Verzeichnis `/etc/yum.repos.d` hinterlegt (vgl. Abbildung 6.3(b)). Das Einspielen des Metapakets `ca-policy-egi-core` erfolgt während der Ausführung der Aktivität `Install Required RPMS` des Workflows in Abbildung 6.3(a).



(a) Installation der EGI Trustanchor (b) Konfiguration des EGI Trustanchor-Repositories

Abbildung 6.3: Workflows zur Installation der EGI Trustanchor

Die Kapselung der Installations- und Konfigurationsschritte für die Grid Middleware-Dienste in separate Aktivitäten sowie Subflows (vgl. Abschnitt 5.3.2) und die Trennung zwischen Logik und Implementierung ermöglicht eine einfache Umstellung auf die Verwendung des neuen EGI Metapakets: die bisherige Aktivität verbleibt weiterhin im Workflow, allerdings wird sie durch eine andere Applikation ausgefüllt. Am Beispiel des Workflows für die gLite VoBox (vgl. Abbildung 5.16) wird beim Aufruf der Aktivität `Install Certificates` nicht länger der Subflow `Install_CA_Certificates_SL5_4`, sondern der Subflow `Install_EGI_Trustanchors_SL5_4` ausgeführt.

Aktualisierung der Globus Toolkit Middleware Das folgende Beispiel beschreibt den Austausch nicht nur einer einzelnen Applikation, sondern eines ganzen Subflows. Der Original- als auch der Austausch-Subflow sind für die Installation der Globus Toolkit-Software verantwortlich, jedoch für unterschiedliche Versionen.

In Abschnitt 5.3.3.1 ist für die Version 4 des Globus Toolkit ein Workflow zur Installation unter Verwendung des Grid Packaging Tools und des Globus Toolkit-Quellcodes vorgestellt. Ein hierzu alternativer Workflow kann die Installation basierend auf einem bereits präparierten `tar.gz`-Archiv für das Globus Toolkit vornehmen und so den Kompiliervorgang umgehen. Beide Varianten sind passende Kandidaten zur Ausfüllung der Aktivität `Install GTK 4.0.8` des in Abbildung 5.23 gezeigten Workflows.
Es bietet sich aber nicht nur an, unterschiedliche Installationsarten für einen Dienst auf alternative Subflows abzubilden. Ebenso können alternative Implementierungen unterschied-

liche Versionen einer Software bereitstellen. Dies ist bei Software Updates, insbesondere Minor Releases[4], von Relevanz und ermöglicht ein sinnvolles Vorgehen für die Aktualisierung des Gesamt-Workflows.
Eine praktische Anwendung fand das Vorgehen bei der Aktualisierung des Globus Toolkit Version 5 auf der Grid-Ressource `udo.gt01.grid.tu-dortmund.de` der Technischen Universität Dortmund. In der Ausgangssituation ist auf der Ressource die im Rahmen der D-Grid Referenzinstallation bereitgestellte Globus Toolkit Middleware in Version 5.0.1 installiert. Die Installation der Software erfolgt gemäß Anleitung über die Hinterlegung der D-Grid Software Repository-Konfiguration auf dem System und das Einspielen mehrerer Pakete. Beides ist in den Aktivitäten `Setup Globus Repository` und `Install Globus` des in Abbildung 6.4 gezeigten Workflows kodiert.

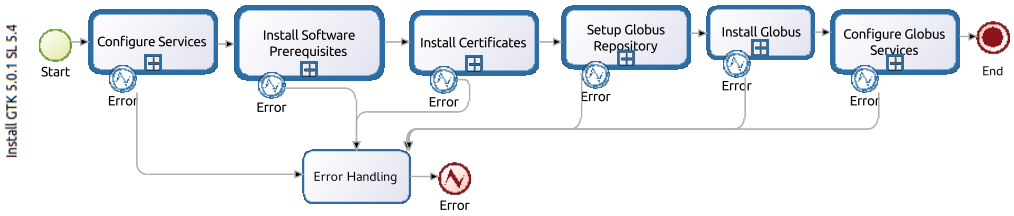


Abbildung 6.4: Workflow zur Installation von GT 5.0.1 auf SL 5.4

Aufgrund eines neuen Releases der Middleware sollte der Umstieg auf die aktuellere Version 5.0.3 des Globus Toolkits erfolgen, für welche die D-Grid-Initiative jedoch keine Software-Pakete bereitstellte. Daher erfolgt die Installation der Software durch die Übersetzung des Quellcodes. Der resultierende Workflow, der neben der Software-Aktualisierung auch eine geänderte Installationsart für das Toolkit realisiert, ist in Abbildung 6.5 dargestellt. Die einfache Installation von RPM-Paketen über die Ausführung des Subflows `YUM`

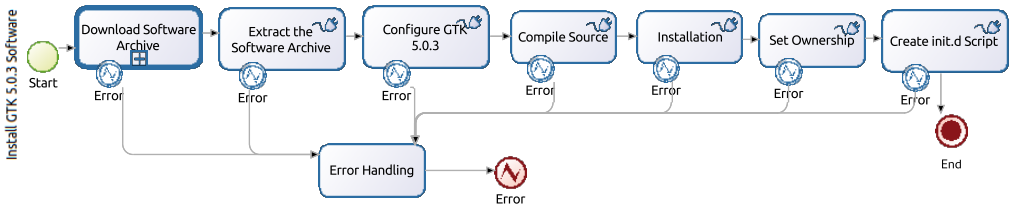


Abbildung 6.5: Workflow zur Installation der GT 5.0.3 Software auf SL 5.4

`Install Package` während der Aktivität `Install Globus` (vgl. Abbildung 6.4) ist somit dem in Abbildung 6.5 gezeigten Subflow gewichen.

[4]Zumeist dient diese Release-Form der Behebung von Fehlern. Wesentliche Änderungen bei der Installation und Konfiguration der Software treten kaum auf.

6.3 Wiederverwendbarkeit von Applikationen und Subflows

Ein weiterer Vorteil der Zerlegung der durchzuführenden Arbeiten in einzelne Aktivitäten bzw. in die dahinter liegenden Applikationen und Subflows liegt in der Möglichkeit, diese wiederzuverwenden. Dieser Aspekt ist an verschiedenen Punkten bei der Erstellung der in Kapitel 4 und 5 beschrieben Workflows eingeflossen und nachfolgend an Beispielen wie dem Workflow `YUM Create Repository` erläutert.

YUM Create Repository Bevor die Installation von Software aus einem YUM Repository erfolgen kann, müssen auf dem System grundlegende Informationen über dieses Repository eingespielt werden. Der Workflow `YUM Create Repository` (vgl. Abbildung 4.3(a)) erzeugt hierzu im Verzeichnis `/etc/yum.repos.d` eine Datei, welche nach Abschluss des Workflows die Konfigurationsinformationen für das Repository enthält.

In einer frühen Variante des Workflows nahm weder die Aktivität `Create YUM Repository` noch der Workflow selbst Argumente entgegen. Aufgrund der in der zugehörigen Applikation fest-kodierten Repository-Informationen (vgl. Listing 6.1) eignete sich der Workflow für die Erzeugung von genau diesem einen Software-Repository. Ohne die Berücksichtigung des Aspekts der Wiederverwendbarkeit ist für jedes verwendete Software-Repository ein solcher, separater Workflow notwendig.

```
def outputFile = new File ("/etc/yum.repos.d/lcg-ca.repo")
def lines = [
      '[lcg-CA]',
      'name=lcg-CA',
      'baseurl=http://linuxsoft.cern.ch/LCG-CAs/current',
      'enabled=1',
      'protect=1']
outputFile.write(lines[0..4].join("\n"))
```

Listing 6.1: Hinterlegung von Repository-Information (Groovy-Applikation)

Um dieser Entwicklung entgegenzuwirken, sind die festen Werte in der hinter der Aktivität `Create YUM Repository` liegenden Applikation durch Variablen ersetzt. Dies betrifft i) den Dateinamen innerhalb des Verzeichnisses `/etc/yum.repos.d`, ii) den als Beschreibung des Repositories dienenden Freitext, iii) den logischen Bezeichner sowie iv) den Uniform Resource Locator (URL) unter dem das Repository erreichbar ist (vgl. Listing 6.2).

```
def outputFile = new File ("${file_Name}")
def lines = [
    '[${repository_Name}]',
    'name=${repository_Description}',
    'baseurl=${base_URL}',
```

```
6       'enabled=1',
        'protect=1']
8   outputFile.write(lines[0..4].join("\n"))
```

Listing 6.2: Verbesserte Hinterlegung der Repository-Information (Groovy-Applikation)

Damit ist nur noch ein einziger Workflow notwendig, über den sich alle Repository-Konfigurationen im System hinterlegen lassen. Die Übergabe der erforderlichen Informationen erfolgt beim Aufruf des Workflows, so dass die in Abbildung 6.6 gezeigten Aktivitäten `Create Basic Repository`, `Create Updates Repository` und `Create Externals Repository` auf denselben Subflow zurückgreifen.

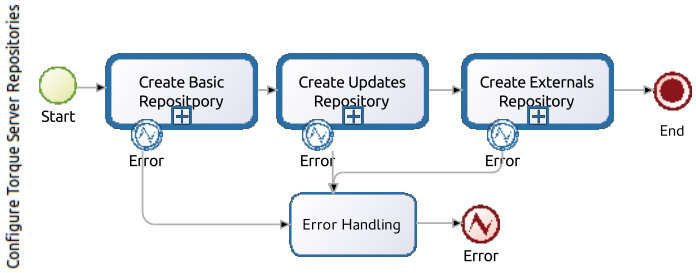


Abbildung 6.6: Workflow zur Konfiguration der TORQUE Server Repositories

Mit diesem Vorgehen ist einerseits die Flexibilität eines initial erstellten Workflows und andererseits auch dessen Wiederverwendbarkeit gezeigt.

Configure, Make und Make install Eine zum vorhergehenden Beispiel ähnliche Situation existiert bei der Betrachtung der in Abschnitt 4.6 und 4.7 vorgestellten Workflows für die Installation des Nagios und des Ganglia Servers.

Für Software besteht neben der Installation aus vorkonfigurierten Paketen oftmals die Möglichkeit, sie über das Kompilieren des Quellcodes im System verfügbar zu machen. Die hierbei durchzuführenden Schritten betreffen i) das Herunterladen des Quellcodes, ii) die Auflösung etwaiger Software-Abhängigkeiten, iii) das Entpacken des Quellcodes, iv) die Festlegung der Kompilieroptionen, v) den Kompiliervorgang selbst, vi) die Installation des Kompilats und vii) das Entfernen des heruntergeladenen Quellcode-Archivs und eventuell temporär verwendeter Verzeichnisse.

Die einzelnen Aktivitäten i) bis vii) sind zu einem Workflow (vgl. Abbildung 6.7) zusammenfassbar, welcher sich – wie am Beispiel des `YUM Create Repository`-Workflows gezeigt – mittels der Einbringung von Variablen flexibilisieren lässt. Während der Ausführung der Aktivität `Install Dependencies` werden die Software-Abhängigkeiten aufgelöst, was idealerweise hinsichtlich der besseren bzw. einfacheren Austauschbarkeit in

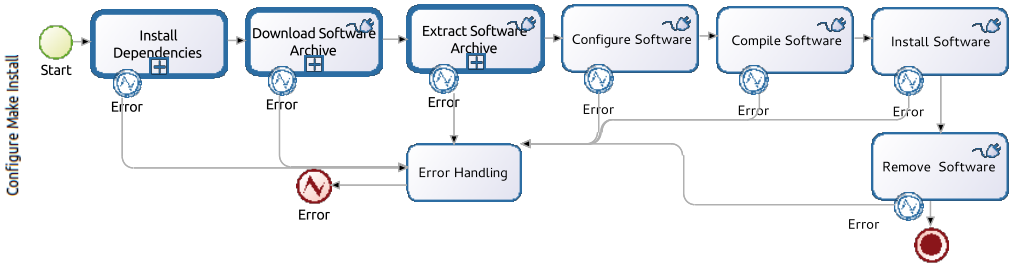


Abbildung 6.7: Generischer Configure, Make, Install-Workflow

einem Subflow und nicht über eine Applikation geschehen sollte. Im Anschluss an diesen Schritt, erfolgt das Herunterladen und Extrahieren der Software in den beiden Aktivitäten `Download Software Archive` und `Extract Software Archive`. Der über die Aktivität `Download Software Archive` aufgerufene Subflow besteht im Wesentlichen aus der Aktivität `Download Files`. Der hinter dieser Aktivität liegende Groovy-Code ist in Listing 6.3 gezeigt. Dabei sind der Verweis auf das herunterzuladende Software-Archiv und dessen Ablageort im lokalen Dateisystem über die Belegung der Variablen `filenames` und `directory` vom Aufrufenden zuvor festgelegt worden.

```
def String command = "wget --append-output=/tmp/wget.log -N --no-proxy ${filenames} -P ${directory}"
def Process proc = command.execute()
proc.waitFor()
```

Listing 6.3: Download von Software (Groovy-Applikation)

Nachdem der Quellcode der Software in extrahierter Form vorliegt, wird dieser während der Aktivität `Configure Software` durch den Aufruf des mit der Software mitgelieferten `configure`-Skripts präpariert, wobei u. a. `Makefiles` erzeugt werden können. Der Groovy-Code in den Zeilen 1 bis 8 des Listings 6.4 sorgt für die Ausführung des `configure`-Skripts im richtigen Verzeichnis und wird ebenso bei dem Kompilier- und Installationsvorgang verwendet.

```
1 def String tempString = "${downloadURL}"
  def String filename = "${downloadDirectory}"  + "/" + tempString.substring( tempString.lastIndexOf('/')+1);
3
  def command = [ "sh", "-c", "tar --no-recursion --exclude '*/*' " + filename ]
5 def Process proc = command.execute()
  proc.waitFor()
7 def String userDir = proc.in.text
  System.setProperty("user.dir", userDir)
9
```

```
command = "./configure " + ${configureOptions}
proc = command.execute()
proc.waitFor()
```

Listing 6.4: Extraktion und Konfiguration von Software (Groovy-Applikation)

Die Aktivität `Compile Software` des Workflows leitet die Kompilation des Quellcodes ein. Der Groovy-Code der zugehörigen Applikation ist auszugsweise in Listing 6.5 dargestellt.

```
...
def String userDir = proc.in.text
System.setProperty("user.dir", userDir)

command = "make"
proc = command.execute()
proc.waitFor()
```

Listing 6.5: Kompilation von Software (Groovy-Applikation)

In einer erweiterten Version der Aktivität könnten – analog zum Aufruf des `configure`-Skripts in Listing 6.4 – auch dem Werkzeug `make` Optionen mitgegeben werden. Hierdurch ließe sich der in Listing 6.6 gezeigte Groovy-Code bzw. die zugehörige separate Applikation in der Aktivität `Install Software`, die für die Installation der kompilierten Software eingesetzt wird, vermeiden.

```
...
def String userDir = proc.in.text
System.setProperty("user.dir", userDir)

command = "make install"
proc = command.execute()
```

Listing 6.6: Installation von Software (Groovy-Applikation)

Die letzte Aktivität des Workflows entfernt das anfänglich heruntergeladene Software-Archiv, da dies nach der Installation der Software nicht länger benötigt wird. Der hierzu verwendete Groovy-Code ist in Listing 6.7 gezeigt.

```
def String tempString = "${downloadURL}"
def String filename = "${downloadDirectory}"   + "/" + tempString.
    substring( tempString.lastIndexOf('/')+1);
def String command = "rm -f " +  filename
def Process proc = command.execute()
proc.waitFor()
```

Listing 6.7: Löschen eines Software-Archivs (Groovy-Applikation)

Über die Zeilen 1 und 2 wird aus der Download URL und dem Download-Verzeichnis der vollständige Pfad zu der zu löschenden Datei gewonnen und in Zeile 3 mit dem `rm`-Befehl verknüpft.

6.4 Fehlerbehandlung

Während der Ausführung eines Workflows können Situationen eintreten, die eine korrekte Fortsetzung verhindern und so eine Ausnahmebehandlung erfordern. Ein häufiger Auslöser für eine solche Situation ist der Eintritt eines unvorhergesehenen Ereignisses, weswegen dieses oft als Zwischenereignis bezeichnet wird.
In der Business Process Modeling Notation lassen sich Zwischenereignisse in zwei disjunkte Klassen einteilen. Die Ereignisse der ersten Klasse führen zum Abbruch der aktuell ausgeführten Aktivität, wogegen die Ereignisse der zweiten Klasse die Fortsetzung der Abarbeitung erlauben und lediglich einen neuen, aktiven Ausführungszweig hinzufügen.

In der graphischen Darstellung eines Workflows sind die Ausstiegspunkte für Zwischenereignisse an den Rand der Aktivitäten, in denen sie auftreten können, geheftet (vgl. Abbildung 5.8). Die Ausstiegspunkte für nicht-unterbrechende Zwischenereignisse sind durch einen Kreis mit doppelter, gestrichelter Randlinie zu erkennen. Entsprechend repräsentieren Kreise mit einer durchgehenden, doppelten Randlinie die Ausstiegspunkte für unterbrechende Zwischenereignisse.

Neben einfachen Ausnahmen bei der korrekten Abarbeitung bilden Fehler einen weiteren, schwerwiegenderen Zwischenereignis-Typ. Er ist durch das in Abbildung 5.8 zu erkennende Blitzsymbol im Ausstiegspunkt charakterisiert. Im Gegensatz zu anderen Zwischenereignis-Typen können Fehler nicht alleinstehend in Workflows erscheinen, sondern müssen stets an Aktivitäten geheftet sein.
Genauer betrachtet sind Fehler-betreffende Zwischenereignisse in empfangende und sendende unterteilbar. Die sendenden Zwischenereignisse stellen Endereignisse dar, die beispielsweise einen betroffenen Subflow vollständig abbrechen und die Information hierüber nach außen mitteilen. Außen übernehmen die empfangenden Zwischenereignisse, die sich ausschließlich an den Rändern der Aktivitäten befinden, diese Information.

Als Beispiel für die Fehlerbehandlung dient hier ein Workflow zur Hinterlegung der Repository-Informationen für die European Grid Initiative Trustanchor im System (vgl. Abbildung 6.8). Die während der Ausführung des Workflows ansgestoßenen Applikationen in den Aktivitäten `Create YUM EGI Trustanchor Repository` und `Update Package Lists` führen Groovy-Code aus, der unter bestimmten Bedingungen nicht die erwünschte Wirkung haben kann. Ein solcher Fall tritt ein, wenn z. B. der Workflow-

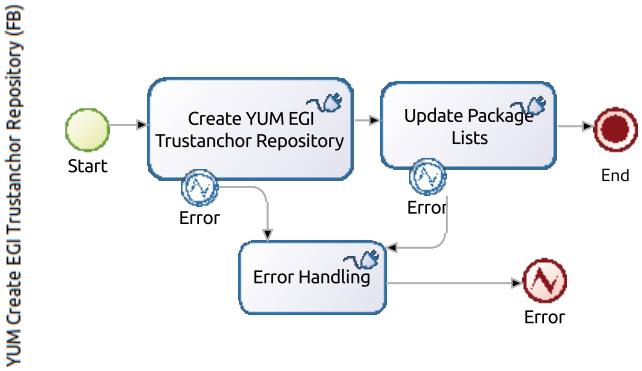


Abbildung 6.8: Beispiel-Workflow zur erweiterten Fehlerbehandlung

Ausführende nicht über ausreichende Rechte im Dateisystem verfügt, um in dem Zielverzeichnis `/etc/yum.repos.d` Dateien anlegen zu können (vgl. Listing 6.8).

```
def outputFile = new File ("/etc/yum.repos.d/egi-trustanchor.repo")
def lines = [ '[EGI-trustanchors]',
  'name=EGI-trustanchors',
  'baseurl=http://repository.egi.eu/sw/production/cas/1/current/',
  'enabled=1',
  'gpgcheck=1',
  'gpgkey=http://repository.egi.eu/sw/production/cas/1/GPG-KEY-EUGridPMA-
      RPM-3'
  ]
outputFile.write(lines[0..5].join("\n"))
```

Listing 6.8: Hinterlegen von Repository-Informationen (Groovy-Applikation)

Eine weitere Quelle für Fehler liegt in der Ausführung von Befehlen, die Modifikationen an nicht zugreifbaren bzw. schreibbaren Dateien vornehmen wollen. Im Listing 6.9 bezieht sich dies etwa auf die hinter YUM liegenden Datenbanken.

```
def String command = "yum makecache"
def Process proc = command.execute()
proc.waitFor()
```

Listing 6.9: Aktualisierung des YUM Caches (Groovy-Applikation)

Ohne die Behandlung der möglicherweise auftretenden Fehler entstehen in diesen Situationen nicht-behandelte Ausnahmen, die zum vollständigen Abbruch des Workflows führen[5]. Daher ist bei der Erstellung der in den vorherigen Kapiteln vorgestellten Workflows jeder Aktivität und damit gleichzeitig den darunter liegenden Applikationen ein Ausstiegspunkt

[5]Es bestünde auch die Möglichkeit zur Fortsetzung des Workflows, doch ist aufgrund der aufgetretenen Ausnahme das Ziel des Workflows nicht mehr zuverlässig erreichbar.

für den Fall eines Fehlers zugeordnet. Tritt ein Fehler auf, so wird die gerade ausgeführte Aktivität über diesen Punkt verlassen und die Aktivität des `Error Handling`, in der die Fehlerbehandlung stattfinden kann, eingeleitet.
Perspektivisch sind so die in dieser Arbeit vorgestellten Workflows bzgl. der Fehlerbehandlung über etwa eine Differenzierung in Fehlerarten bzw. -klassen und die Festlegung von Abarbeitungsregeln für den Fehlerfall erweiterbar.

6.5 Parallelisierung von Applikationen und Subflows

Die im vorherigen Abschnitt beschriebene Fehlerbehandlung widmet sich der Abarbeitung von Ausnahmen und trägt somit zur kontrollierten Ausführung eines Workflows bei. Die Parallelisierung einzelner Aktivitäten und Subflows hingegen verfolgt mit einer möglichst zeit-effizienten Ausführung des Workflows ein anderes Ziel.

Wie in Abschnitt 6.1 erwähnt, entspringen die betrachteten Workflows einer Menge von Skripten und Anweisungen, die zur manuellen Ausführung gedacht sind. Dementsprechend bestehen die ersten Versionen der Workflows aus linearen Verkettungen einzelner Aktivitäten. Im Zuge einer Workflow-Optimierung (vgl. Abschnitt 6.7) ist die Parallelisierung[6] voneinander unabhängiger Aktivitäten auf einem System durchaus denkbar. Als Voraussetzung hierfür müssen jedoch die notwendigen Ressourcen in einem ausreichenden Maß vorhanden sein.

Beispielhaft soll die Parallelisierung an einem bestehenden Workflow, hier dem zur Installation eines gLite CREAM Compute Elements (vgl. Abschnitt 5.3.2.3), gezeigt werden. Der Workflow ist zunächst auf etwaige Abhängigkeiten einzelner Aktivitäten untereinander zu untersuchen. Bei dieser Analyse sind folgende Punkte feststellbar:

- Die Installation der gLite Software sowie von YAIM setzen die Hinterlegung der notwendigen Repository-Information im System voraus.
- Die Konfiguration der gLite Software setzt die Installation der Software und die Existenz der Dateien `users.conf`, `groups.conf` und `site-info.def` voraus.
- Die zwei Aktivitäten `Configure Services` und `Configure GLite Software Repository` sind voneinander unabhängig und können daher parallel ausgeführt werden.
- Ebenso voneinander unabhängig sind die drei Aktivitäten zur Aufbereitung der Dateien `users.conf`, `groups.conf` und `site-info.def` sind

[6] In diesem Kontext steht die Parallelisierung für die zeitlich parallele Ausführung mehrerer Aktivitäten des Workflows.

Eine mögliche Variante des Workflows, in dem nun Aktivitäten parallel ausgeführt werden, ist in Abbildung 6.9 dargestellt. Die Konfiguration der Betriebssystem-nahen Dienste findet

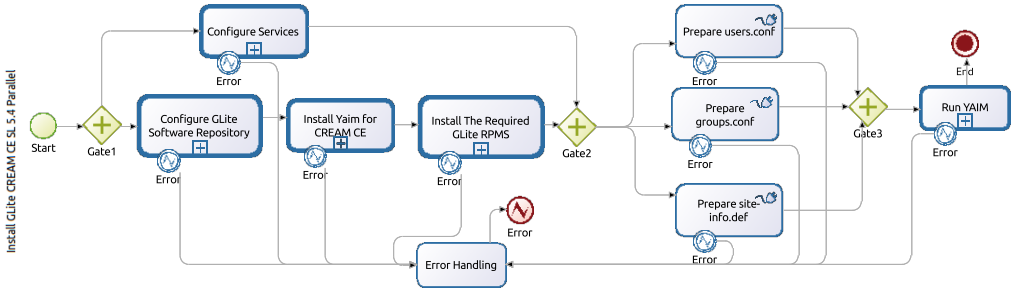


Abbildung 6.9: Beispiel-Workflow zur Parallelisierung

zwischen `Gate 1` und `Gate 2` parallel zur Einrichtung des Software-Repositories und der Installation der gLite Middleware statt. Erst wenn die Abarbeitung in beiden Zweigen `Gate 2` erreicht, wird mit der Einrichtung der Dateien `users.conf`, `groups.conf` und `site-info.def` im System begonnen. Da die drei hierbei verwendeten Aktivitäten untereinander keine Abhängigkeiten besitzen, können sie parallel zwischen `Gate 2` und `Gate 3` ausgeführt werden. Am `Gate 3` laufen die Abarbeitungszweige zusammen und gehen in die Ausführung der letzten Aktivität `Run YAIM` über.
Analog sind für die Optimierung der anderen vorgestellten Workflows deren Aktivitäten auf Abhängigkeiten zu untersuchen und entsprechend umzuordnen.

Ein Beispiel für die Parallelisierung von Subflows ist nachfolgend an der Konfiguration der Dienste eines UNICORE 6-Knotens verdeutlicht. Die einzelnen Subflows zur Konfiguration des Gateways, des UNICORE/X, der XUUDB und des Target System Interface sind in der initialen Version des Workflows (vgl. Abbildung 5.52) linear angeordnet. Da zwischen den einzelnen Subflows keine Abhängigkeiten existieren, sind diese ebenso parallel ausführbar. Eine mögliche Workflow-Variante, die dieser Feststellung Rechnung trägt, ist in Abbildung 6.10 dargestellt. Die Verzweigung in die parallele Abarbeitung der Kompo-

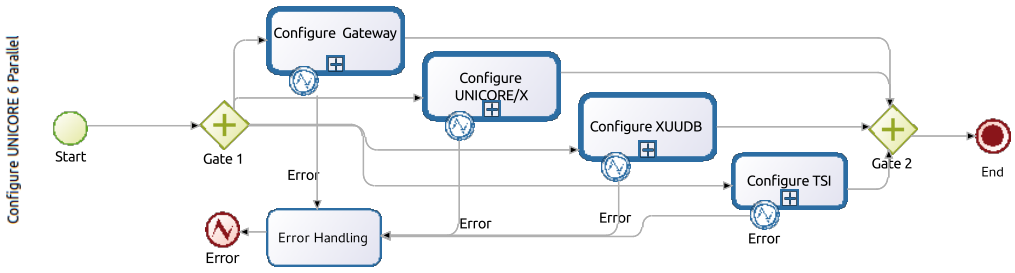


Abbildung 6.10: Workflow zur Konfiguration des UNICORE 6-Komponenten

nenten-Konfiguration findet mit dem Durchschreiten des `Gate 1` statt und endet sobald in

allen Zweigen der entsprechende Subflow abgeschlossen und somit das `Gate 2` erreicht ist.

6.6 Unterstützung weiterer Betriebssysteme und Software

Ein wichtiges Unterziel des angewandten Vorgehens bei der Workflow-Erstellung ist die einfache Erweiterbarkeit auf andere als die in dieser Arbeit betrachteten Betriebssysteme Scientific Linux und SUSE Linux Enterprise Server.

Als Teil der Evaluation sind für weitere Linux-basierte Betriebssysteme die Installationsmethoden untersucht. Die Betriebssystemauswahl fiel auf Debian Linux[7] und auf Ubuntu Linux[8]. Debian Linux ist ein interessanter Kandidat, da es sich seit kurzem wie Scientific Linux als Basisbetriebssystem für alle im D-Grid angebotenen Grid Middlewares eignet.

Debian Linux Die Workflow-basierte Installation des Betriebssystems erfolgt hier – hinsichtlich der später möglichen Einbindung in eine virtuelle Maschine – in eine lokal erzeugte Abbilddatei. Die Erzeugung der Abbilddatei erfolgt über den Aufruf des Kommandos `qemu-img` in Verbindung mit der Option `create` und der Angabe der Abbildgröße in GByte (vgl. Listing 6.10, Zeile 1). In dieser Datei wird mittels des Kommandos `mkfs` standardmäßig ein `ext3`-Dateisystem angelegt. Spätere Versionen des Workflows sollten das Dateisystem-Format von dieser festen Vorgabe lösen und auswählbar machen. Die so aufbereitete Abbilddatei ist über den `mount`-Befehl als Loopback-Gerät in das lokal vorliegende Dateisystem einbindbar (vgl. Zeile 6 des Listing 6.10). Im Anschluss daran kann mittels des Kommandos `debootstrap` eine Debian-basierte Linux Distribution installiert werden. Lediglich die Angabe der Debian Linux-Version (hier: `lenny`) und des Software-Repositories sind notwendig.

```
1 qemu-img create <path>/debian_domU.img 8G

3 mkfs.ext3 <path>/debian_domU.img

5 mkdir <path>/debian_domU/
  mount -o loop <path>/debian_domU.img <path>/debian_domU/
7
  debootstrap lenny <path>/debian_domU http://ftp.us.debian.org/debian
```

Listing 6.10: Installationsschritte für Debian Linux

[7] http://www.debian.org/
[8] http://www.ubuntu.com/

Die gerade beschriebenen Schritte lassen sich ohne weiteres in einzelne Aktivitäten bzw. Applikationen kapseln und zu einem Workflow (vgl. Abbildung 6.11) anordnen. Die ers-

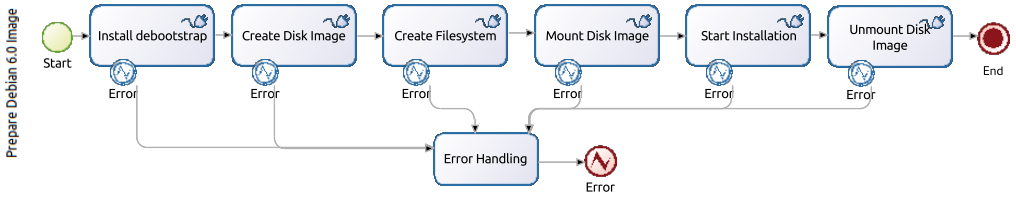


Abbildung 6.11: Workflow zur Erzeugung einer Abbilddatei mit Debian Linux

te Aktivität des Workflows installiert die nachfolgend benötigte Software `debootstrap` über den Aufruf des Paketmanager. Im Anschluss daran legt die Aktivität `Create Disk Image` durch den Aufruf von `qemu-img` das leeres Festplattenabbild an. Die zwei darauf folgenden Aktivitäten des Workflows, das Formatieren des Festplattenabbilds in `Create Filesystem` und das Einhängen des Abbilds in das aktuelle Dateisystem in `Mount Disk Image`, enthalten Applikationen deren Funktionalität analog zu der in Listing 6.10 dargestellten ist. Der Workflow schließt mit dem Aufruf des Kommandos `debootstrap` während der Aktivität `Start Installation` und dem Aushängen des Festplattenabbilds aus dem lokalen Dateisystem in der letzten Aktivität `Unmount Disk Image` ab. Nach der erfolgreichen Abarbeitung des Workflows liegt somit eine frische Installation des Betriebssystems Debian Linux in dem Festplattenabbild vor.

Ubuntu Linux Analog zu dem in Abschnitt 2.3 beschriebenen Vorgehen bei der KIWI-Software kann das Kommando `debootstrap` das Betriebssystem auch in eine `chroot`-Umgebung installieren. Hierfür benötigt es als Eingabe zwei Argumente, zum einen das Verzeichnis, das als `chroot`-Umgebung dienen soll, und zum anderen einen Verweis auf das zu verwendende Software-Repository für Ubuntu Linux. Die einzelnen, bei einer solchen Installation ausgeführten Befehle sowie Teile der Ausgabe sind in Listing 6.11 dargestellt.

```
# sudo mkdir <path>/virtual_machines/debian_chroot
# sudo debootstrap --arch i386 lucid /mnt http://archive.ubuntu.com/
    ubuntuI: Retrieving Release
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Checking component main on http://archive.ubuntu.com/ubuntu...
...
I: Retrieving base-files
I: Validating base-files
```

```
...
I: Unpacking required packages...
I: Unpacking adduser...
...
I: Configuring the base system..
...
I: Configuring initramfs-tools...
I: Base system installed successfully
```

Listing 6.11: Befehle zur Installation von Ubuntu Linux

Die beiden in dem Listing privilegiert ausgeführten Befehle sind auf einen Workflow, wie er in Abbildung 6.12 gezeigt ist, abbildbar. Die einzelnen Aktivitäten dieses Workflows ähneln den bereits für die Installation von Debian Linux vorgestellten. Da das Betriebssystem in diesem Beispiel in einer `chroot`-Umgebung erzeugt wird, entfallen alle das Festplattenabbild betreffenden Aktivitäten, wie etwa `Create Disk Image` und `Unmount Disk Image`.

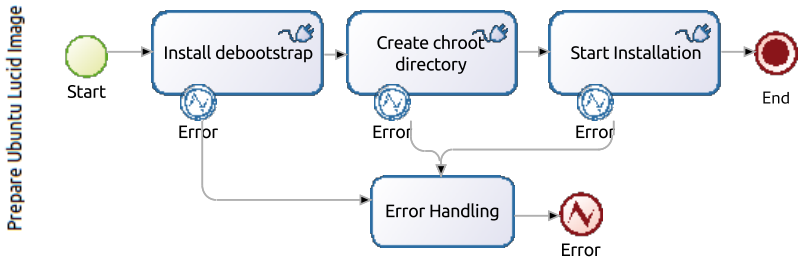


Abbildung 6.12: Workflow zur Installation von Ubuntu Linux

6.7 Prozess- und Workflow-Optimierung

Workflows und die dahinter liegenden Prozesse unterliegen – wie in Abbildung 3.14 angedeutet – üblicherweise einer kontinuierlichen Optimierung, bei der nach Freund und Götzer (vgl. [FG08]) unterschiedliche Ziele verfolgt werden können.

- Durch die Aufschlüsselung von Vorgängen in einzelne Aktivitäten und einer Analyse hinsichtlich deren Notwendigkeit, können überflüssige Aktivitäten identifiziert und eingespart werden. Da jede Aktivität mit einem zeitlichen Aufwand und Ressourcenverbrauch assoziiert ist, führt das Streichen überflüssiger Aktivitäten zu Einsparungen.

- Die Durchlaufzeit eines Workflows entspricht der Summe aller anfallenden Bearbei-

tungszeiten, Warte- und Transportzeiten entlang des kritischen Pfades[9]. Oftmals sind Bearbeitungs- und Transportzeit gut vorhersagbar, jedoch lassen externe Einflüsse, wie eine notwendige menschliche Interaktion (z. B. Nutzereingaben), die Wartezeit zu einer schwer kalkulierbaren Größe werden. Die Senkung der durchschnittlichen als auch der maximalen Durchlaufzeit kommt einer verbesserten Abarbeitung des Workflows gleich und stellt somit ein wünschenswertes Ziel dar.
In Abschnitt 6.5 ist mit der Parallelisierung einzelner Workflow-Aktivitäten eine Möglichkeit zur Senkung der Durchlaufzeiten dargestellt.

- Die Schaffung bzw. Steigerung der Prozesstransparenz führt zu einem besseren Verständnis des Prozesses und hilft so, z. B. etwaige Engpässe schnell zu identifizieren und zu beheben.

- Die kontinuierliche Verbesserung der Prozessqualität[10] ist eines der wichtigeren Ziele. Zur Messung der Prozessqualität können verschiedene Metriken herangezogen werden. Die Struktur des für diese Arbeit herangezogenen Capability Maturity Model Integration (CMMI)-Reifegradmodells ist in Tabelle 6.1 dargestellt. Basierend auf der Qualität eines Workflows wird dieser einer der sechs Stufen des Modells zugeordnet.
  Bei Workflows, die der untersten Stufe des Modells zugeordnet werden, typischer-

Stufe	Bezeichnung	Bedeutung
0	Incomplete	Initialer Zustand
1	Performed	Erreichen des spezifischen Ziele des Workflows
2	Managed	Verwaltung des Workflows
3	Defined	Verwaltung und Verbesserung des Workflows auf Basis von Standards
4	Quantitatively Managed	Einsatz einer statistischen Workflow-Kontrolle
5	Optimizing	Verbesserung des Workflows anhand der Daten aus der statistischen Workflow-Kontrolle

Tabelle 6.1: Das CMMI-Reifegradmodell, Quelle: [FBS09]

weise ad-hoc-Workflows, fehlt es an einer Umgebung für die kontrollierte Ausfüh-

[9] Ein bzw. der Pfad im Prozess, dessen Abarbeitung am längsten dauert.

[10] Der Qualitätsbegriff bezieht sich in diesem Kontext auf die fehlerfreie Ausführung des Prozesses unter Beachtung wohl-definierter Randbedingungen.

rung. Dies führt in vielen Fällen zu einer niedrigen Quote erfolgreich ausgeführter Workflows, da der Erfolg hier stark von der Kompetenz der involvierten Personen abhängt und sich nicht reproduzieren lässt. Eine wesentliche Steigerung der Erfolgsquote und damit einhergehend das gesicherte Erreichen der Workflow-Ziele wertet den Workflow zu einem der Stufe 1 auf.

Auf Stufe 1 treten wie auf Stufe 0 noch Fehlschläge auf, allerdings sind diese dem Überschreiten von vorher festgelegten Grenzwerten zuzuschreiben. Die Grenzwerte beziehen sich dabei auf den erlaubten, maximalen Ressourcen- und/ oder Zeitverbrauch des Gesamt-Workflows oder einzelner Aktivitäten. Aufgrund der limitierten Ressourcen ist auf dieser Stufe eine Überzeichnung der verfügbaren Kapazitäten möglich, die bei auftretenden Engpässen zum Abbruch von einigen, gerade aktiven Workflows führt.

Fügt man den Workflows eine Verwaltung hinzu, so migrieren sie auf Stufe 2 des Modells. Als Teil der Verwaltung sind Richtlinien zu definieren, die eine geplante Ausführung erlauben. Ein weiteres Merkmal dieser Stufe ist die Überwachung und Kontrolle des Workflows, zuvor war nur die erfolgreiche bzw. nicht-erfolgreiche Ausführung feststellbar. Das Erreichen der Stufe 3 des Reifegradmodells erfordert die Festlegung einer Menge von Standard-Workflows und deren kontinuierliche Verbesserung. Über diese Form von Workflows ist beispielsweise innerhalb einer Organisation die Herstellung eines gewissen Konsistenzniveaus möglich. Bezogen auf die in dieser Arbeit vorgestellten Workflows, könnten die Workflows für das Herunterladen oder Extrahieren von Software-Archiven in den Bereich der Standard-Workflows fallen. Eine weitere, typische Folge der kontinuierlichen Verbesserung sind sehr detaillierte Beschreibungen der einzelnen Workflows. Neben den beteiligten Rollen sind auch die erwarteten Parameter inklusive Gültigkeitsbereich sowie Möglichkeiten zur Verifikation der Ausgabe festgehalten.

Für Workflows der Stufe 4 werden statistische Daten hinsichtlich des Erreichens vorgegebener Performance- und Qualitätsziele erhoben, wobei die Definition der Ziele beispielsweise auf die Endnutzer ausgerichtet ist.

Die fortlaufende Optimierung eines Workflows aufgrund der in Stufe 4 gesammelten Daten hebt diesen auf Stufe 5 an. Die Optimierungen umfassen beispielsweise Modifikationen am Workflow selbst oder Veränderungen auf technischer Ebene, z. B. die Bereitstellung zusätzlicher Hardware-Ressourcen.

Die in den Kapiteln 4 und 5 vorgestellten Workflows lassen sich problemlos in das in Tabelle 6.1 gezeigte Reifegradmodell einordnen. Da keinerlei statistische Informationen über die Workflows bzw. deren Ausführung festgehalten werden, entfallen sowohl die vierte als auch

die fünfte Stufe des Modells. Ebenso wenig sind Richtlinien hinsichtlich einer kontrollierten Ausführung definiert, wodurch die Stufen 2 und 3 als mögliche Kandidaten entfallen. Mit einer entsprechenden Fehlerbehandlung, die die erfolgreiche Ausführung der Workflows begünstigt, sind die Workflows auf Stufe 1 anzusiedeln, ansonsten jedoch nur auf Stufe 0. Dieses Ergebnis zeigt mögliche zukünftige Arbeiten auf, wobei als Ziel dieser die kontinuierliche Anhebung der Workflows auf Stufe 2 oder höher festgehalten werden kann.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Der Fokus dieser Arbeit liegt in der Erarbeitung und Bereitstellung von Workflows für eine hochgradig, bestenfalls vollständig automatisierte Installation und Konfiguration von Grid Middleware-Diensten. Zeitgemäß sind als Zielplattform für diese Dienste keine physischen Systeme, sondern virtuelle Maschinen vorgesehen. Dieser Punkt erweitert die Aufgabenstellung um die automatisierte Erzeugung der virtuellen Maschinen sowie die Installation und Konfiguration des Basisbetriebssystems als Teil einer solchen Maschine.

Ausgehend von dieser Zieldefinition sind in Kapitel 2 verschiedene bereits existierende Ansätze für die automatisierte Installation von Linux-basierten Betriebssystemen am Beispiel von Scientific Linux und SUSE Linux Enterprise Server untersucht und evaluiert. Zudem sind in diesem Kapitel die Vorgehen für die Installation der drei Grid Middlewares gLite, Globus Toolkit und UNICORE vorgestellt. Zusammenfassend zeichnet sich für diese ein uneinheitliches Bild ab: während beispielsweise die Pakete der gLite Middleware aus einem Software-Repository heraus installierbar sind, müssen für das Globus Toolkit und die UNICORE Middleware Software-Archive von einer Webseite heruntergeladen und prozessiert werden. Ähnlich verhält es sich bei der späteren Konfiguration der Middleware-Dienste, da jede der Middlewares ein eigenes Verfahren anwendet.
Abschließend sind in Kapitel 2 mit rBuilder und der Image Creation Station Möglichkeiten zur Erzeugung virtueller Appliances aufgezeigt und am Beispiel der Software KIWI im Detail beschrieben.

Um die Chance auf die Nachhaltigkeit der in dieser Arbeit erzielten Ergebnisse zu wahren, setzt die genutzte Workflow-Modellierung auf etablierte Standards auf. In diesem Kontext sind in Kapitel 3 mit Ereignisgesteuerten Prozessketten, der XML Process Definition Language und der Business Process Modeling Notation drei Beschreibungssprachen erläu-

tert. Weiterhin ist anhand zweier Beispiele die Notation von Workflows und deren Komponenten in zwei der drei Beschreibungssprachen dargestellt. Diese Beispiele bilden eine der Grundlagen für die durchgeführte Evaluation der Sprachen, wobei sich letztendlich die Business Process Modeling Notation aufgrund ihres hohen Verbreitungsgrades und der Unterstützung durch viele Workflow Engines durchsetzen konnte.
Das Kapitel 3 enthält ebenso eine ausführliche Darstellung der Virtualisierungstechnologie. Bringt man eine Struktur in dieses Themengebiet, so sieht man es in die Virtualisierung einzelner Ressourcen, von Applikationen und von ganzen Plattformen zerfallen. Gerade letzteres erlaubt, u. a. in Kombination mit der Virtualisierung von Netzwerkressourcen, die Erstellung virtueller Maschinen und somit virtueller Appliances.

Für das Teilziel der Unterstützung der Workflow-basierten Installation von Linux-basierten Betriebssystemen sind grundlegende Ergebnisse in Abschnitt 4.1 gezeigt. Die erstellten Workflows greifen im Falle des Betriebssystems Scientific Linux auf ein Kickstart-basiertes Verfahren zurück, wobei die hierbei notwendige Kickstart-Datei in wenigen Schritten von Hand erstellt werden kann. Im Rahmen der durchgeführten Evaluation sind diese Ergebnisse in Abschnitt 6.6 auf die Betriebssysteme Debian Linux und Ubuntu Linux angewandt. Für diese beiden ist zudem die Installation in ein Festplattenabbild bzw. in `chroot`-Umgebung beschrieben.
In Kapitel 4 sind zudem Workflows für Installation und Konfiguration verschiedener so genannter Basisdienste gezeigt. Viele dieser Dienste, zu denen etwa ein Network File System-Server, ein Dynamic Host Configuration Protocol-Server und ein Lightweight Directory Access Protocol-Server zählen, bilden üblicherweise eine Voraussetzung für die aufzusetzenden Grid Middleware-Dienste. Weiterhin sind in diesem Kapitel Workflows für die Installation und Konfiguration der Client- und Server-Komponenten der Ganglia- und Nagios-Software festgehalten. Diese Software ist beispielsweise in der Lage die Erreichbarkeit von Rechnern und Diensten sowie deren Auslastung graphisch darzustellen.
Einer der zentralen Bestandteile einer Grid Ressource ist das Batchsystem, bestehend aus Rechenknoten, einer Verwaltungskomponente sowie Clients zur Einreichung von Jobs. Für die existierenden Batchsysteme ist stellvertretend die frei zugängliche TORQUE-Software, welche national und international einen hohen Verbreitungsgrad hat, gewählt und analysiert. Das Ergebnis dieser Analyse sind Workflows, die eine automatisierte Installation und Konfiguration für jede der drei TORQUE-Komponenten erlauben.

Im Rahmen des Kapitels 5 sind zunächst mit der Reproduzierbarkeit wissenschaftlicher Daten und der dynamischen Erzeugung von Grid Middleware-Diensten auf Compute Cloud-Infrastrukturen zwei mögliche Anwendungsszenarien angedeutet.
Der Schwerpunkt des Kapitels liegt allerdings auf der Beschreibung der für die einzelnen

Grid Middleware-Dienste erstellten Workflows, wobei sowohl die gLite, die Globus Toolkit als auch die UNICORE Middleware Berücksichtigung finden. So sind etwa Workflows für das gLite Compute Element und den siteBDII sowie für die UNICORE 6-Komponenten Gateway und Target System Interface angegeben. Insgesamt enthält das Kapitel für mehr als zehn Dienste der drei derzeit in Deutschland meistgenutzten Grid Middlewares Workflows zur deren Installation und Konfiguration. Zur weiteren Berücksichtigung des nationalen Aspekts sind in Abschnitt 5.4 die Besonderheiten der Grid-Initiative D-Grid dargestellt. Dies inkludiert auch die Beschreibung von Workflows zur Installation des so genannten Interaktiven Knotens und die Verwendung des `dgridmap`-Skripts.

Die detaillierte Evaluation der in den Kapiteln 4 und 5 präsentierten Workflows findet in Kapitel 6 statt. Die einfache Austauschbarkeit und auch die Wiederverwendbarkeit einmal entworfener Applikationen und Subflows ist anhand von Beispielen belegt. Diese Beispiele, etwa die Umstellung auf den Einsatz der European Grid Initiative Trustanchor, entspringen realen Szenarien.
Des Weiteren sind Möglichkeiten für die bei der initialen Erstellung der Workflows bewusst vernachlässigte Behandlung von Ausnahmen und Fehlern vorgestellt. Eine dieser Möglichkeiten ist in spätere Versionen der einzelnen Workflows integrierbar, indem die leere Aktivitätshülse `Error Handling` gefüllt wird.

Als weiterer Punkt der Evaluation ist die Optimierung der Workflows untersucht. In dem ersten Teil der Untersuchung ist mit der parallelen Ausführung von voneinander unabhängigen Aktivitäten eine Verbesserungsvorschlag gegenüber der rein sequentiellen Ausführung vorgestellt. Mit einem modifizierten Workflow zur Installation und Konfiguration eines gLite CREAM Compute Element sowie einem weiteren Workflow zur parallelisierten Konfiguration der UNICORE 6-Komponenten sind hierzu zwei Beispiele angeführt. Der zweite Teil der Untersuchung bezieht sich auf die qualitative Einordnung der erstellten Workflows anhand eines Reifegradmodells. Diese Einordnung bildet die Grundlage für zukünftige Verbesserungen hinsichtlich der Workflow-Qualität und dient so letztendlich der Optimierung.

Insgesamt ist festzustellen, dass die formulierten Ziele dieser Arbeit erreicht sind. Es sind in den vorhergehenden Kapiteln Workflows für die Installation und Konfiguration von Diensten der Grid Middlewares gLite, Globus Toolkit und UNICORE gezeigt. Zur Vereinfachung ist dabei von Scientific Linux als unterliegendes Betriebssystem ausgegangen worden, jedoch zeigt die Evaluation die problemlose Erweiterbarkeit des Vorgehens auf andere Betriebssysteme wie Debian Linux. Zudem kommt mit der Business Process Modeling Notation eine weitverbreitete und standardisierte Beschreibungssprache zum Einsatz.

7.2 Ausblick

Gerade die in Kapitel 6 durchgeführte qualitative Einordnung der präsentierten Workflows in das CMMI Reifegradmodell zeigt den Bedarf für weitere Arbeiten. Aufgrund der Beschreibung der einzelnen Stufen des Modells sind die hierfür durchzuführenden Tätigkeiten umrissen und inkludieren beispielsweise die verbesserte Fehlerbehandlung sowie die Spezifikation von Grenzen für den maximalen Ressourcenverbrauch einzelner Aktivitäten bzw. des Gesamt-Workflows.
Diese Qualitätssteigerung ist von essentieller Bedeutung und kann einen Beitrag zu den aktuellen Entwicklungen im Bereich des Grid und Cloud Computing leisten. Im Kontext verschiedener Projekte des 7. Forschungsrahmenprogramms der Europäischen Union, etwa StratusLab[1] und VENUS-C[2], nimmt die Bereitstellung und damit einhergehend die Erzeugung virtueller Appliances eine besondere Stellung ein. Dies trifft ebenso auf viele Grid Computing-Initiativen wie beispielsweise die European Grid Initiative und die Initiative for Globus in Europe zu, die ihrerseits die Bereitstellung virtueller Appliances zur vereinfachten Installation der Grid Middleware-Dienste auf den Ressourcen vorantreiben.
Die in dieser Arbeit präsentierten Workflows beziehen sich u. a. auf die Installation und Konfiguration von Middleware-Diensten beider Initiativen und stellen somit eine gute Ausgangsbasis für ein reproduzierbares und transparentes Vorgehen für die Erstellung solcher Appliances dar. Zudem begünstigt die Modellierung der Workflows in der standardisierten und in der Unternehmenswelt weit verbreiteten Business Process Modeling Notation die zukünftige Verfügbarkeit der notwendigen Workflow Engines und Entwicklungsumgebungen.

Im Sinne der Transparenz und der Vermeidung von Mehraufwand ist die Bereitstellung einmal entwickelter Workflows in einem öffentlichen Software-Repository möglich. Interessierte Entwickler können so, z. B. die hinter den Aktivitäten liegenden Applikationen ihren eigenen Bedürfnissen anpassen und bereits existierende Workflows als Subflows einbinden. Da Ressourcen-Anbieter ebenfalls einen Einblick in das Workflow-Repository haben können, ist für sie ein Überblick über die in einer Appliance installierte Software möglich.
Das für die Ablage der Workflows genutzte Software-Repository muss mindestens den Mehrbenutzer-Betrieb und die Versionierung unterstützen. Letzteres macht insbesondere in Verbindung mit der in Abschnitt 6.2 beschriebenen Austauschbarkeit von Applikationen und Subflows Sinn und erlaubt – falls notwendig – den Rückschritt auf eine vorherige, funktionierende Version eines Workflows.
Bevor allerdings eine neue Version eines Workflows in das Software-Repository eingestellt

[1] `http://stratuslab.eu/`
[2] `http://www.venus-c.eu/Pages/Home.aspx`

wird, ist diese auf ihre Funktionalität zu prüfen. Erfahrungsgemäß finden solche Tests aber nur vereinzelt statt oder decken lediglich Teilaspekte der Funktionalität ab. Aus diesem Grund setzt man bei größeren Software-Projekten auf Hilfswerkzeuge, die periodisch Build-Prozesse für die Software anstoßen und im Anschluss daran automatisiert Tests durchführen und auswerten. Ein solches Werkzeug ist die eInfrastructure for Testing, Integration and Configuration of Software (ETICS) (vgl. [MBC+08]), welche im Zusammenhang mit der gLite Middleware eingesetzt wird. Dieses Werkzeug greift im Rahmen des kontinuierlichen Build-Prozesses auf den im Software-Repository befindlichen Quell-Code der Middleware zu, kompiliert diesen und erstellt anschließend die einzelnen Software-Pakete für die Middleware-Dienste.
Bezogen auf den Themenkomplex der Workflows sind statt der Übersetzung des Quell-Codes und dem Bau von Software-Paketen einzelne Workflows ggf. überwacht auszuführen und deren Ergebnisse auszuwerten. Da die Workflows hauptsächlich die Installation und Konfiguration von Diensten innerhalb einer Appliance durchführen, kann die Überprüfung der rudimentären Funktion dieser als Testgrundlage dienen.

Die in Abschnitt 5.3 vorgestellten Dienste stellen nur einen Ausschnitt aus der Menge der verfügbaren Dienste der Middlewares gLite, Globus Toolkit und UNICORE dar. Eine zukünftige Erweiterung der hier dargestellten Ergebnisse besteht in der Erstellung von Workflows für weitere Komponenten dieser Middlewares. Hierzu zählen insbesondere die Grid-zentralen Dienste wir das Workload Management System der gLite Middleware und die UNICORE 6 Registry.
In diesem Zusammenhang bietet auch die Gründung der European Middleware Initiative[3] Möglichkeiten für Erweiterungen. Zum einen unterstützt die European Middleware Initiative neben gLite und UNICORE auch die Middleware Advanced Resource Connector (ARC)[4] und zum anderen gibt es dort beispielsweise nicht länger ein separates User Interface für jede Middleware, sondern ein gemeinsames EMI User Interface, welches die Client-Software der Middlewares ARC, dCache, gLite und UNICORE enthält. Diese beiden Fakten können zu angepassten bzw. neuen Workflows führen.
Eine Ausweitung der Workflows in eine andere Richtung besteht in der Hinzunahme der Unterstützung für die Middlewares dCache und OGSA-DAI. Beide Middlewares bieten verschiedene Schnittstellen zu Speicherressourcen an und sind in dieser Arbeit, die sich allein auf Rechenressourcen und deren Installation sowie Konfiguration fokussiert, vernachlässigt.

Den wohl spannendsten Punkt für zukünftige Arbeiten bildet die Erstellung ganzer Landschaften und damit die bedarfsorientierte, kombinierte Erzeugung mehrerer virtueller Appliances innerhalb eines einzigen Workflows.

[3] http://www.eu-emi.eu/
[4] http://www.nordugrid.org/arc/

Abkürzungsverzeichnis

ACL	Access Control List 115
AJO	Abstract Job Object 140
AMI	Amazon Machine Image 20
ANL	Argonne National Laboratory 121
API	Application Programming Interface 101
APEL	Accounting Processor for Event Logs 117
ARC	Advanced Resource Connector 187
BAM	Business Activity Monitoring 47
BAR	Business Archive 91
BDII	Berkeley Database Information Index 109
BMBF	Bundesministerium für Bildung und Forschung 137
BPMI	Business Process Management Initiative 49
BPML	Business Process Modeling Language 49
BPMN	Business Process Modeling Notation 41
BSD	Berkeley Software Distribution 79
C3Grid	Collaborative Climate Community Data and Processing Grid 151
CA	Certification Authority 166
CE	Compute Element 15
CERN	Conseil Européen pour la Recherche Nucléaire 22
CIS	Common Information Service 144
CMMI	Capability Maturity Model Integration 180
CPU	Central Processing Unit 62
CRL	Certificate Revocation List 166
CRM	Customer Relationship Management 48
CREAM	Computing Resource Execution and Management 114
DAI	Data Access and Integration 152
DAP	Directory Access Protocol 73
DHCP	Dynamic Host Configuration Protocol 61
DGI	D-Grid Integrationsprojekt 152
DGSI	D-Grid Scheduler Interoperability 97
DN	Distinguished Name 75
DTD	Document Type Definition 11
DTML	Document Template Mark-Up Language 55
EC2	Elastic Compute Cloud 101
EDG	European DataGrid Project 107
EGI	European Grid Initiative 103
EMI	European Middleware Initiative 19
EPK	Ereignisgesteuerte Prozessketten 46

ERP Enterprise Resource Planning . . . 47
EGEE Enabling Grids for E-science . . . 15
EUGridPMA European Policy Management Authority for Grid Authentication . . . 111
ETICS eInfrastructure for Testing, Integration and Configuration of Software 187
FAI Fully Automated Installation . . . 7
Fermilab Fermi National Accelerator Laboratory . . . 9
FCFS First-come, first-served . . . 83
FHS Filesystem Hierarchy Standard . . . 4
FLOP Floating Point Operations per Second . . . 104
FQHN Fully Qualified Hostname . . . 69
FTP File Transfer Protocol . . . 121
FZJ Forschungszentrum Jülich . . . 154
FZK Forschungszentrum Karlsruhe . . . 159
GID Group Identification Number . . . 78
GLUE Grid Laboratory Uniform Environment . . . 100
GNU GNU's Not Unix . . . 24
GPG GNU Privacy Guard . . . 66
GPL General Public License . . . 24
GPT Grid Packaging Tool . . . 17
GRAM Grid Resource Allocation Manager . . . 17
GRRS Grid Resource Registration Service . . . 151
GSI Grid Security Infrastructure . . . 105
HBA Host Bus Adapter . . . 29
HEPCG High Energy Physicists Community Grid . . . 151
HEPL High Energy Physics Linux . . . 9
HPC High Performance Computing . . . 96
HTML Hypertext Mark-Up Language . . . 54
IaaS Infrastructure as a Service . . . 101
ICS Image Creation Station . . . 19
IDB Incarnation Database . . . 141
IEEE Institute of Electrical and Electronics Engineers . . . 33
IGE Initiative for Globus in Europe . . . 17
IP Internet Protocol . . . 69
IRF Institut für Roboterforschung . . . 164
ISV Independent Software Vendor . . . 151
JDK Java Development Kit . . . 139
JEE Java Enterprise Edition . . . 91
JRE Java Runtime Environment . . . 139
JSDL Job Submission Description Language . . . 137
JeOS Just Enough Operating System . . . 37
LaaS Landscape as a Service . . . 164
LB Logging and Bookkeeping . . . 120
LCG LHC Computing Grid . . . 15
LDAP Lightweight Directory Access Protocol . . . 61
LDIF LDAP Data Interchange Format . . . 112
LFC LCG File Catalog . . . 120
LHC Large Hadron Collider . . . 106
LSB Linux Standard Base . . . 4
LUN Logical Unit Number . . . 29

MAC Media Access Control 72
MBR Master Boot Record 13
MDS Monitoring and Discovery System 121
MMX Multimedia Extension 34
MonALISA Monitoring Agents using a Large Integrated Services Architecture . . 118
NAT Network Address Translation 28
NSCA Nagios Service Check Acceptor 76
NFS Network File System 61
NGI-DE National Grid Initiative - Deutschland 103
NIC Network Interface Card 30
NIS Network Information Service 73
NJS Network Job Supervisor 140
NRPE Nagios Remote Plugin Executor 76
NTP Network Time Protocol 72
OASIS Organization for the Advancement of Structured Information Standards 11
OGF Open Grid Forum 137
OGSA Open Grid Services Architecture 152
OMG Object Management Group 49
OVF Open Virtual Machine Format 37
PaaS Platform as a Service 20
PC Personal Computer 9
PCIe PCI Express 30
PHP Personal Home Page 80
PLC Public Limited Company 73
QEMU Quick Emulator 64
RAID Redundant Array of Independent Disks 28
RAM Random Access Memory 63
RFT Reliable File Transfer 17
R-GMA Relational Grid Monitoring Architecture 117
RPM RPM Package Manager
RSH Remote Shell 1
SaaS Software as a Service 102
SA3 Specific Service Activities 3 15
SAN Storage Area Network 28
SE Storage Element 109
SL Scientific Linux 8
SLES SUSE Linux Enterprise Server 153
SLED SUSE Linux Enterprise Desktop 20
SOAP Simple Object Access Protocol 121
SOSP Symposium of Operating System Principles 36
SSE Streaming SIMD Extensions 34
SSH Secure Shell 1
SSL Secure Socket Layer 138
SUSE Software und Systementwicklung GmbH 11
TCP Transmission Control Protocol 129
TORQUE Terascale Open-Source Resource and Queue Manager 83
TSI Target System Interface 140

Literaturverzeichnis

[AAA+09] Michael Anthony, Assaf Arkin, Sylvan Astier, Rob Bartel, Ed Barkmeyer, Conrad Bock, Donna Burbank, Steinar Carlsen, Petko Chobantonov, Ugo Corda, Fred Cummins, Bob Daniel, Tony Fletcher, Steven Forgey, Karl Frank, Jean-Luc Giraud, Brian James, George Keeling, Markus Klink, Antoine Lonjon, Monica Martin, Lee Mason, Frank McCabe, Dale Moberg, Martin Owen, Pete Rivett, Suzette Samoojh, Jesus Sanchez, Robert Shapiro, Bob Smith, Manfred Sturm, Balasubramanian Suryanarayanan, Michelle Vanchu-Orozco, David Williams und Paul Wuethrich. Business Process Model and Notation (BPMN): Version 1.2: OMG Document Number: formal/2009-01-03, 20.10.2009.

[ABB+03] William E. Allcock, Joe Bester, John Bresnahan, Sam Meder, Pawel Plaszczak und Steve Tuecke. GridFTP: Protocol Extensions to FTP for the Grid, April 2003.

[ABE+09] Sergio Andreozzi, Stephen Burke, Felix Ehm, Laurence Field, Gerson Galang, Balazs Konya, Maarten Litmaath, Paul Millar und John-Paul Navarro. GLUE Specification v. 2.0, 03.03.2009.

[AFF+09] Manfred Alef, Thomas Fieseler, Stefan Freitag, Ariel Garcia, Christian Grimm, Wolfgang Gürich, Hamza Mehammed, Lars Schley, Olaf Schneider und Gian Luca Volpato. Integration of Multiple Middlewares on a Single Computing Resource. *Future Generation Computer Systems*, 25(3):268–274, 2009.

[All09] Thomas Allweyer. *BPMN 2.0 - Business Process Model and Notation: Einführung in den Standard für die Geschäftsprozessmodellierung*. Books on Demand, Norderstedt, 2. Auflage, 2009.

[Ama11] Amazon Web Services. Amazon Elastic Compute Cloud API Reference: API Version 2011-02-28, 06.05.2011.

[ASZ+10] Cristina Aiftimiei, Massimo Sgaravatto, Luigi Zangrando, Sergio Traldi, Paolo Andreetto, Sara Bertocco, Simone Dalla Fina, Alvise Dorigo, Eric Frizziero,

Alessio Gianelle, Moreno Marzolla und Mirco Mazzucato. Design and Implementation of the gLite CREAM Job Management Service. *Future Generation Computer Systems*, 26(4):654–667, 2010.

[Bar08a] Ed Barkmeyer. Business Process Definition MetaModel: Volume I: Common Infrastructure, 2008.

[Bar08b] Wolfgang Barth. *Nagios: System- und Netzwerk-Monitoring*. Open Source Press, München, 2. Auflage, 2008.

[BASB+10] Predrag Buncic, Carlos Aguado Sanchez, Jakob Blomer, Leandro Franco, Artem Harutyunian, Pere Mato und Yushu Yao. CernVM – A Virtual Software Appliance for LHC Applications. *Journal of Physics: Conference Series*, 219(4), 2010.

[BBB+05] Ian Bird, Kors Bos, Nick Brook, Dirk Duellmann, Chris Eck, Ian Fisk, David Foster, Bruce Gibbard, Claudio Grandi, Francois Grey, John Harvey, Andreas Heiss, Frederic Hemmer, Sverre Jarp, Roger Jones, David Kelsey, Jürgen Knobloch, Massimo Lamanna, Holger Marten, Pere Mato Vila, Farid Ould-Saada, Bernd Panzer-Steindel, Laura Perini, Les Robertson, Yves Schutz, Ulrich Schwickerath, Jamie Shiers und Torre Wenaus. LHC Computing Grid: Technical Design Report: Version 1.04, 20.06.2005.

[BDE+98] Ulrike Beisiegel, Johannes Dichgans, Gerhard Ertl, Siegfried Großmann, Bernhard Hirt, Claude Kordon, Lennart Philipson, Eberhard Schmidt-Aßmann, Wolf Singer, Cornelius Weiss, Sabine Werner und Björn H. Wiik. *Vorschläge zur Sicherung guter wissenschaftlicher Praxis: Empfehlungen der Kommission Selbstkontrolle in der Wissenschaft*. WILEY-VCH Verlag GmbH, Weinheim, 1998.

[BDE+07] Otto Büchner, Christa Dohmen, Thomas Eifert, Harry Enke, Thomas Fieseler, Anton Frank, Stefan Freitag, Ariel Garcia, Christian Grimm, Wolfgang Gürich, Helmut Heller, Thomas Jejkal, Holger Nitsche, Olaf Schneider und Wolfgang Ziegler. Betriebskonzept für die D-Grid Infrastruktur, 2007.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt und Andrew Warfield. Xen and the Art of Virtualization. In Michael L. Scott und Larry Peterson, Hrsg., *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Seiten 164–177, New York, 2003. Association for Computing Machinery.

[Ber05a] Sven van de Berghe. Using The Incarnation Database (V4.6), 2005.

[Ber05b] Sven van de Berghe. Using The Unicore Gateway 4.1.0, 2005.

[Ber06] Sven van de Berghe. Using The NJS and TSI (V4), 2006.

[BES01] Heinz Baumgartner, Klaus Ebert und Karsten Schleier. Regeln zur Modellierung von ereignisgesteuerten Prozessketten: Beilage zur kaufmännischen ZPG - Mitteilung Nr. 24, 2001.

[BLVM02] Tavis Barr, Nicolai Langfeldt, Seth Vidal und Tom McNeal. Linux NFS-HOWTO, 2002.

[BRSW09] Daniel Becker, Morris Riedel, Achim Streit und Felix Wolf. Grid-Based Workflow Management for Automatic Performance Analysis of Massively Parallel Applications. In Norbert Meyer, Domenico Talia und Ramin Yahyapour, Hrsg., *Grid and Services Evolution*, Seiten 103–118. Springer-Verlag US, Boston, 2009.

[CCD+01] Lionel Cons, Germán Cancio, Philippe Defert, Mark Olive, Ignacio Reguero und Cedric Rossi. Automating Linux Installations at CERN. In Mirco Mazzucato und Michele Michelotto, Hrsg., *Proceedings of Computing In High-Energy And Nuclear Physics*, Seiten 601–605, 2001.

[Cit09] Citrix Systems, Inc. Citrix XenConvert Guide: XenConvert 2.0.1: Release 1, 2009.

[CVRML09] Juan Caceres, Luis M. Vaquero, Luis Rodero-Merino und Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2009.

[Daw10] Troy Dawson. Scientific Linux - Indepth History, 04.10.2010.

[Deu00] Deutsches Institut für Normung e. V. Ergonomische Grundlagen bezüglich psychischer Arbeitsbelastung - Teil 1: Allgemeines und Begriffe, 2000.

[Dik06] Jeff Dike. *User Mode Linux*. Bruce Perens' Open Source Series. Prentice Hall, Upper Saddle River, 1. Auflage, 2006.

[DL99] Ralph E. Droms und Ted Lemon. *The DHCP Handbook: Understanding Deploying and Managing Automated configuration Services*. Network Architecture and Development Series. MacMillan Technical Publishing, Indianapolis, 1999.

[Eis06] Mike Eisler. RFC 4506: XDR - External Data Representation Standard, 2006.

[EK08] Johannes Ernesti und Peter Kaiser. *Python: Das umfassende Handbuch*. Galileo Computing. Galileo Press, Bonn, 1. Auflage, 2008.

[Erw02] Dietmar W. Erwin. UNICORE - A Grid Computing Environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410, 2002.

[Fal08] Niels Fallenbeck. Design of the Image Creation Station, 2008.

[FBS09] Eileen C. Forrester, Brandon L. Buteau und Sandy Shrum. *CMMI for Services: Guidelines for Superior Service*. The SEI Series in Software Engineering. Addison-Wesley, Amsterdam, 2009.

[FFEA12] Florian Feldhaus, Stefan Freitag und Chaker El Amrani. State-of-the-Art Technologies for Large-Scale Computing. In Werner Dubitzky, Krzysztof Kurowski und Bernhard Schott, Hrsg., *Large-scale computing techniques for complex system simulations*, Jgg. 80, Seiten 1–16. Wiley, Hoboken, 2012.

[FG06] Patrick Fuhrmann und Volker Gülzow. dCache, Storage System for the Future. In Wolfgang E. Nagel, Wolfgang Lehner und Wolfgang V. Walter, Hrsg., *Euro-Par 2006 Parallel Processing*, Jgg. 4128 of *Lecture Notes in Computer Science*, Seiten 1106–1113, Berlin, 2006. Springer.

[FG08] Jakob Freund und Klaus Götzer. *Vom Geschäftsprozess zum Workflow: Ein Leitfaden für die Praxis*. Hanser, München, 2008.

[FG10] Thomas Fieseler und Wolfgang Gürich. Development and Operation of the D-Grid Infrastructure. In Simon C. Lin und Eric Yen, Hrsg., *Production Grids in Asia*, Seiten 199–211, Boston, 2010. Springer-Verlag US.

[Fil04] Filesystem Hierarchy Standard Group. Filesystem Hierarchy Standard, 2004.

[FK98] Ian Foster und Carl Kesselman, Hrsg. *Computational Grids: State of the Art and Future Directions in High–Performance Distributed Computing: Reprinted by permission of Morgan Kaufmann Publishers from The Grid: Blueprint for a Future Computing Infrastructure, I. Foster and C. Kesselman (Eds), 1998.* Morgan Kaufman Publishers, Massachusetts, 1998.

[FK03] Ian Foster und Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufman Publishers, San Francisco, 2. Auflage, 2003.

[FKT01] Ian Foster, Carl Kesselman und Steve Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[FKTT98] Ian Foster, Carl Kesselman, Gene Tsudik und Steven Tuecke. A Security Architecture for Computational Grids. In Li Gong und Michael Reiter, Hrsg., *Proceedings of the 5th ACM Conference on Computer and Communications Security*, Seiten 83–92, New York, 1998. ACM Press.

[FO96] John L. Furlani und Peter W. Osel. Abstract Yourself With Modules. In Helen E. Harrison und Amy K. Kreiling, Hrsg., *LISA '96: Proceedings of the 10th USENIX Conference on System Administration*, Seiten 193–204, Berkeley, 1996. USENIX Association.

[Fos02] Ian Foster. What is the Grid? - A Three Point Checklist. *GRIDtoday*, 1(6):22–25, 2002.

[Fos05] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In Hai Jin, Daniel Reed und Wenbin Jiang, Hrsg., *IFIP International Conference on Network and Parallel Computing*, Jgg. 3779 of *Lecture Notes in Computer Science*, Seiten 2–13, Berlin, 2005. Springer.

[FPSF06] Niels Fallenbeck, Hans-Joachim Picht, Matthew Smith und Bernd Freisleben. Xen and the Art of Cluster Scheduling. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, Virtualization Technology in Distributed Computing, Seite 4, Washington, 2006. IEEE Computer Society.

[FR10] Stefan Freitag und Marcel Risch. Erweiterung einer D-Grid-Ressource um eine Compute-Cloud-Schnittstelle. In Paul Müller und Bernhard Neimair, Hrsg., *Verteilte Systeme im Wissenschaftsbereich*, Jgg. 166 of *GI Proceedings*, Seiten 13–22, Bonn, 2010. Gesellschaft für Informatik e.V.

[Fre09] Stefan Freitag. Virtualisierungstechnologien in Grid Rechenzentren. In Paul Müller und Bernhard Neumair, Hrsg., *Verteilte Systeme im Wissenschaftsbereich*, Jgg. 149 of *GI Proceedings*, Seiten 137–146, Bonn, 2009. Gesellschaft für Informatik e.V.

[Fre10a] Stefan Freitag. Impact of Advanced Virtualization Technologies on Grid Computing Centres. In Simon C. Lin und Eric Yen, Hrsg., *Managed Grids*

and Cloud Systems in the Asia-Pacific Research Community, Seiten 251–263. Springer, Boston, 2010.

[Fre10b] Stefan Freitag. The D-Grid Infrastructure. In Ladislav Hluchý, Peter Kurdel und Jolana Sebestyénová, Hrsg., *Proceedings of the 6th International Workshop on Grid Computing for Complex Problems*, Seiten 14–19. Librix, 2010.

[Fre11] Stefan Freitag. Rechnen im Netz: Grid versus Cloud. *iX - Magazin für professionelle Informationstechnik*, 2011(08):94–97, 2011.

[FW11] Stefan Freitag und Philipp Wieder. The German Grid Initiative D-Grid: Current State and Future Perspectives. In Xiaoyu Yang, Lizhe Wang und Wei Jie, Hrsg., *Guide to e-Science*, Computer Communications and Networks, Seiten 29–52. Springer, London, 2011.

[Gad10] Andreas Gadatsch. *Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. Vieweg+Teubner Verlag, Wiesbaden, 6. Auflage, 2010.

[GHS95] Dimitrios Georgakopoulos, Mark Hornick und Amit Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.

[IEE04] IEEE Computer Society. IEEE Task Force on Cluster Computing - High Availability, 2004.

[Int06] International Organization for Standardization. *Linux Standard Base (LSB) Core Specification 3.1*. International Organization for Standardization, Genf, 1. Auflage, 2006.

[JMPW94] Henry J. Johansson, Patrick McHugh, John A. Pendlebury und William A. Wheeler. *Business Process Reengineering: Breakpoint Strategies for Market Dominance*. Wiley, 1994.

[KAA+05] Konstantinos Karasavvas, Mario Antonioletti, Malcolm Atkinson, Neil Chue Hong, Tom Sugden, Alastair Hume, Mike Jackson, Amrey Krause und Charaka Palansuriya. Introduction to OGSA-DAI Services. In Pilar Herrero, María S. Pérez und Victor Robles, Hrsg., *Scientific Applications of Grid Computing*, Jgg. 3458, Seiten 1–12, Berlin, 2005. Springer.

[KGK+07] Dierk König, Andrew Glover, Paul King, Guillaume Laforge und Jon Skeet. *Groovy in action*. Manning, Greenwich, 2007.

[Kim95] Won Kim, Hrsg. *Modern Database Systems: The Object Model, Interoperability, and Beyond*. ACM Press, New York, 1995.

[KNS01] Gerhard Keller, Markus Nüttgens und August-Wilhelm Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK), 11.06.2001.

[LAPS+11] Maarten Litmaath, Maria Alandes Pradillo, David Smith, Andrew Elwell, Cristina Aiftimiei, Aleš Křenek, Zdeněk Šustr, Massimo Sgaravatto, Asterios Katsofodimos, Owen Synge, Gergely Debreczeni und Felix Nikolaus Ehm. YAIM 4 Guide for SysAdmins, 2011.

[Lav78] Simon Hugh Lavington. The Manchester Mark I and Atlas: A Historical Perspective. *Communications of the ACM*, 21(1):4–12, 1978.

[Law96] Kevin P. Lawton. Bochs: A Portable PC Emulator for Unix/X. *Linux Journal*, 1996(29es), 1996.

[LK08] Ioannis Liabotis und Konstantinos Koukopoulos. EGEE-III Deployment Guide: EU Milestone: MSA3.5.1, 2008.

[LVC+09] Iosif Legrand, Ramiro Voicu, Catalin Cirstoiu, Costin Grigoras, Latchezar Betev und Alexandru Costan. Monitoring and Control of Large Systems with MonALISA. *Communications of the ACM*, 52:49–55, 2009.

[Mar08] John Markoff. Military Supercomputer Sets Record, 2008.

[MBC+08] Alberto Di Meglio, Marc-Elian Bégin, Peter Couvares, Elisabetta Ronchieri und Eva Takacs. ETICS: The International Software Engineering Service for the Grid. *Journal of Physics: Conference Series*, 119(4), 2008.

[Mic06] Microsoft Corporation. Virtual Hard Disk Image Format Specification, 11.10.2006.

[Mic07a] Microsoft Corporation. Windows XP Installation Requirements: Revision: 4.9, 2007.

[Mic07b] Microsoft Corporation. Windows 95 Installation Requirements: Revision: 1.4, 23.04.2007.

[Mic10a] Microsoft Corporation. Windows 7 Installation Requirements, 15.06.2010.

[Mic10b] Microsoft Corporation. A history of Windows - Microsoft Windows, 18.07.2010.

[Mic10c] Microsoft Corporation. Windows Vista Installation Requirements: Version 5.20, 2010.

[Mic11] Microsoft Corporation. Windows 2000 Installation Requirements, 2011.

[ML07] Nilo Mitra und Yves Lafon. SOAP Version 1.2 Part 0: Primer (Second Edition): W3C Recommendation, 27. April 2007.

[NGP+10] Mordechai Nunberg, David Greaves, Oriana Palivan, Wang Zhigang und Philip Garett. Xen Wiki: XenNetWorking, 24.10.2010.

[Par10] Parallels Holdings. Parallels Virtuozzo Containers 4.6 for Linux: User's Guide, 2010.

[PG74] Gerald J. Popek und Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17:412–421, 1974.

[PP08] Fabio Pacini und Luca Petronzio. EGEE User's Guide: WMS Service, 2008.

[Ric05] Stephan Richter. *Zope 3 Developers Handbook: A practical guide to developing and administering content management systems with Zope*. Sams, Indianapolis, 1. Auflage, 2005.

[RM06] Morris Riedel und Daniel Mallmann. Standardization Processes of the UNICORE Grid System. In Jens Volkert, Thomas Farhringer, Dieter Kranzlmüller und Wolfgang Schreiner, Hrsg., *Proceedings of 1st Austrian Grid Symposium*, Jgg. 210, Seiten 191–203, Wien, 2006. Österreichische Computer-Gesellschaft.

[Rom09] Mathilde Romberg. Migration von UNICORE 5 zu UNICORE 6, 2009.

[Sch10] Uwe Schwiegelshohn. D-Grid: A National Grid Infrastructure in Germany. *Annals of Telecommunications*, 65(11-12):763–769, 2010.

[Sha08a] Shreyas Shah. I/O Virtualization in ATCA/UTCA Platforms, 24.10.2008.

[Sha08b] Robert Shapiro. Workflow Management Coalition Workflow Standard: Process Definition Interface – XML Process Definition Language: Document Number WFMC-TC-1025, 2008.

[SS11] Marcus Schäfer und Thomas Schraitle. openSUSE - KIWI Image System Cookbook: Version 4.87, 2011.

[ŠSD+10] Zdeněk Šustr, Jiří Sitera, František Dvořák, Jiří Filipovič, Daniel Kouřil, Aleš Křenek, Luděk Matyska, Miloš Mulač, Jan Pospíšil, Miroslav Ruda, Zdeněk Salvet und Michal Voců. Something you may have wanted to know about L&B. *Journal of Physics: Conference Series*, 219(6), 2010.

[SSF+09] Matthew Smith, Matthias Schmidt, Niels Fallenbeck, Tim Dörnemann, Christian Schridde und Bernd Freisleben. Secure On-Demand Grid Computing. *Future Generation Computer Systems*, 25(3):315–325, 2009.

[Str59] Christopher Strachey. Time Sharing in Large Fast Computers. In *Proceedings of the International Conference on Information Processing*, Seiten 336 – 341, 1959.

[Tat03] Jon Tate. Virtualization in a SAN, 2003.

[Tho65] James E. Thornton. Parallel operation in the control data 6600. In *AFIPS 1964 Fall Joint Computer Conference*, Seiten 33–41. Spartan Books, 1965.

[TOP08] TOP500.org. Roadrunner | TOP500 Supercomputing Sites, 2008.

[VMw06] VMware, Inc. Reducing Server Total Cost of Ownership with VMware Virtualization Software, 2006.

[VMw07a] VMware, Inc. VMware Virtual Disks: Virtual Disk Format 1.1: VMware Technical Note, 13.11.2007.

[VMw07b] VMware, Inc. Understanding Full Virtualization, Paravirtualization, and Hardware Assist: White Paper, 18.10.2007.

[VMw07c] VMware, Inc. Best Practices for Building Virtual Appliances - Whitepaper, 30.07.2007.

[VMw09] VMware, Inc. User's Guide: vCenter Converter Standalone 4.0.1: EN-000158-02, 2009.

[VX07a] VMware, Inc. und XenSource. OVF - Open Virtual Machine Format Specification, 2007.

[VX07b] VMware, Inc. und XenSource. The Open Virtual Machine Format - Whitepaper for OVF Specification, 2007.

[Wor99] Workflow Management Coalition. Workflow Management Coalition Terminology & Glossary: Document Number WFMC-TC-1011, 1999.

[YB05] Jia Yu und Rajkumar Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. *Special Interest Group on Management of Data Record*, 34(3):44–49, 2005.