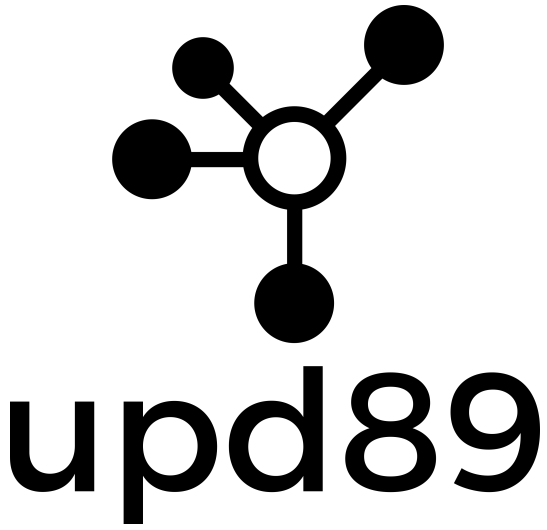
HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Orchestrierung von Security-Updates für Linux-Serversysteme

Bachelorarbeit im Frühjahrssemester 2016



Autoren
Ueli Bosshard
Philipp Christen

Betreuer
Prof. Dr. Farhad Mehta

Experte
Dr. Hermann Lehner

Gegenleser
Mirko Stocker

Rapperswil, 16. Juni 2016

Abstract

Das Unternehmen Nine Internet Solutions AG betreibt Linux-Server für Unternehmen. Da die Server direkt am Internet angeschlossen sind, muss die eingesetzte Software fortlaufend auf dem neuesten Stand gehalten werden. Dies wird durch einen Mitarbeiter erledigt, welcher im Durchschnitt einen Tag pro Woche damit verbringt. Aufgrund der grossen Anzahl von Servern und Paketen sowie Abhängigkeiten zwischen den verschiedenen Arten von Servern ist es schwierig, den Überblick über die installierten Versionen und anstehenden Updates zu behalten.
Im Rahmen dieser Arbeit wurde eine verteilte Software-Lösung entwickelt, mit welcher Systemadministratoren Updates auf Systemen in Auftrag geben können. Diese Updates werden anschliessend automatisch auf den Servern installiert. Dadurch lässt sich der zeitliche Aufwand deutlich reduzieren. Zusätzlich gewinnen sie einen optimalen Überblick über den Zustand der zu verwaltenden Systeme bezüglich der Aktualisierungen.
Das zentrale Control-Center ist ein Webinterface und wurde mit Ruby on Rails umgesetzt. Die Agents auf den Servern wurden in Python implementiert. Die Agenten kommunizieren mit dem Kontrollcenter asynchron über eine API. Für das Kontrollcenter kommt PostgreSQL als Datenbank zum Einsatz, in der alle Informationen über die Systeme verwaltet werden. Besondere Beachtung wurde auf die sichere Kommunikation zwischen den Komponenten gelegt, sowie auf die intuitive Bedienung des Kontrollcenters. Die Veröffentlichung als Open-Source-Projekt ermöglicht die Erweiterung der Benutzer- und Entwicklerbasis auf weitere interessierte Personen.

Webseite: upd89.org

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Entscheidungen

Management Summary

Ausgangslage

nine.ch betreibt über 1500 Linux-Serversysteme mit unterschiedlichen Konfigurationen. Die Software-Komponenten werden über herkömmliche Debian-Pakete installiert. Bei der Veröffentlichung von Sicherheitsupdates müssen diese zeitnah auf sämtlichen betroffenen Serversystemen installiert werden, was bisher manuell mit verschiedenen Bash-Scripts erreicht wurde.

Dies hat mehrere Nachteile: Es gibt keine zuverlässigen Feedback-Mechanismen, verursacht jede Woche mehrere Stunden Aufwand und resultiert im schlimmsten Fall in einer Verzögerung von mehreren Tagen bis zur Installation.

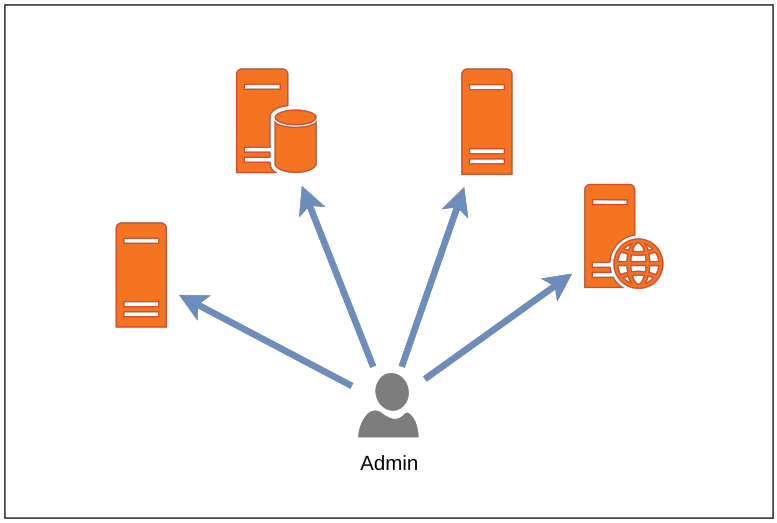


Abbildung 0.1.: Ausgangslage: Ein Mitarbeiter aktualisiert jedes System einzeln

Vorgehen und Technologien

Auf jedem Server soll ein Agent laufen, der ausstehende Updates an ein Kontrollcenter meldet, welches mit Ruby on Rails und PostgreSQL umgesetzt wird. So kann das System leicht in die bestehende Umgebung eingebunden und weiterentwickelt werden.

Auf dem Agent wird eine Python-Bibliothek zur Anbindung an die Debian-Paketverwaltung verwendet. Die Kommunikation zwischen Agents und Control Center erfolgt per HTTPS mit Clientzertifikaten, um eine sichere Kommunikation auch über unsichere Netze zu gewährleisten.

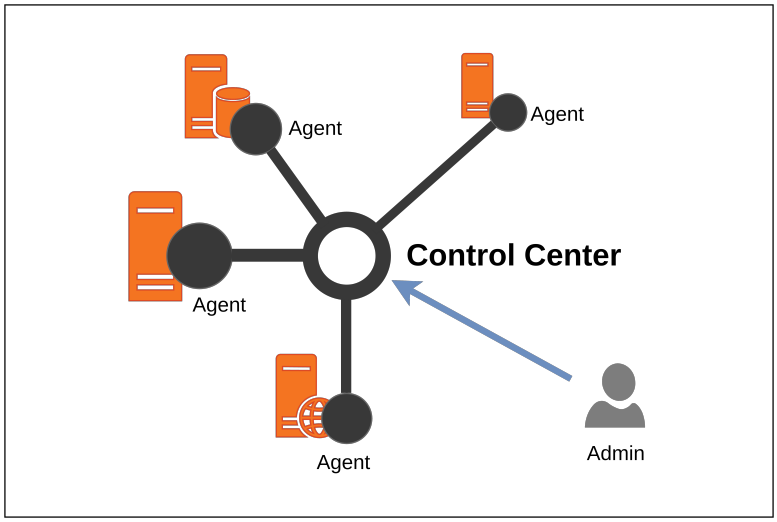


Abbildung 0.2.: Die Lösung ermöglicht das zentrale Updatemanagement einer grossen Anzahl von Servern.

Ergebnisse

Im Rahmen dieser Bachelorarbeit wurde eine Open-Source-Lösung entwickelt, welche das Einspielen von Updates auf Serversystemen vereinfacht. Der Benutzer kann über die Software bestimmen, auf welchen Systemen welche Updates durchgeführt werden sollen. Diese werden an die entsprechenden Systeme gesendet, dort automatisch ausgeführt und zurück an das Kontrollcenter gemeldet.

Der Benutzer kann zudem Systeme und Pakete gruppieren, um Masseninstallationen von Updates auf mehreren Systemen gleichzeitig durchzuführen. Durch verschiedene Auswertungen kann er sich eine bessere Übersicht über die Systeme und Pakete verschaffen.

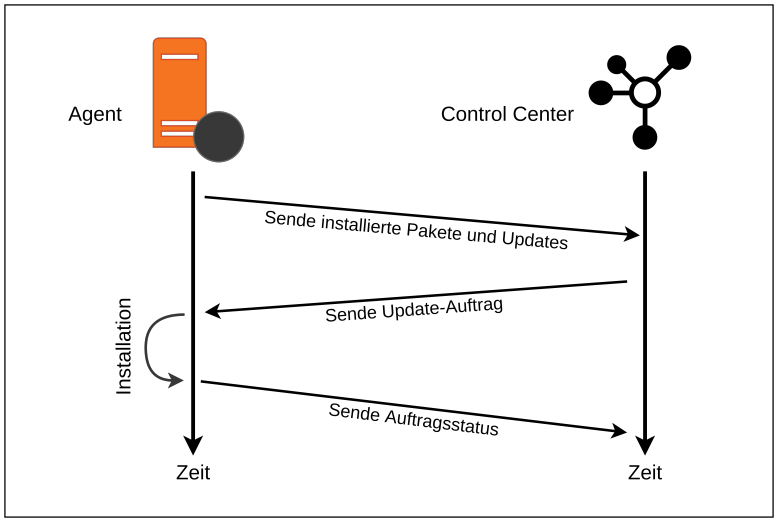


Abbildung 0.3.: Die asynchrone Kommunikation erfolgt ereignisgesteuert.

Ausblick

Die Software wurde mit dem Ziel entwickelt, als Open-Source-Lösung veröffentlicht zu werden. Deshalb ist der Quellcode öffentlich erhältlich und das Benutzen oder Erweitern der Applikation ist erwünscht. nine.ch wird die Software für ihre Zwecke noch erweitern und in ihre bestehende IT-Landschaft einbetten.
Eine Anbindung an andere Paketverwaltungssysteme ist denkbar, da die Agents nur über eine API mit dem Kontrollcenter kommunizieren und die Implementation der Agents davon unabhängig ist.

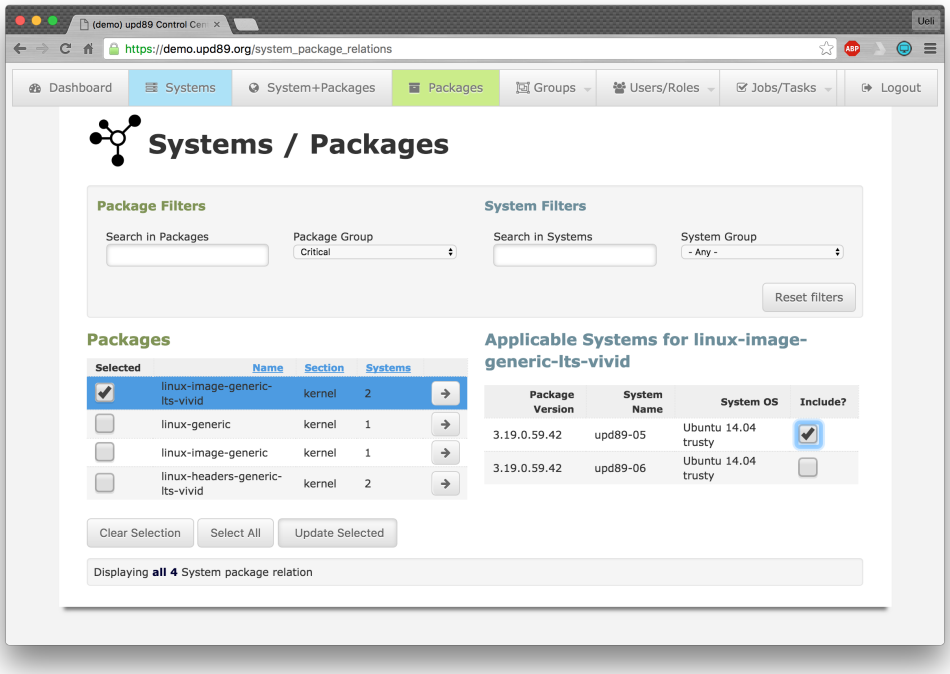


Abbildung 0.4.: Die Weboberfläche ermöglicht die gezielte Auswahl von Updates.

Technischer Bericht

1. Einleitung und Übersicht

Im Auftrag des Industriepartners wurde eine moderne Lösung für das zentrale Updatemanagement erarbeitet. Was zuvor per Shellscript erledigt wurde ist nun übersichtlich über ein Browser-basiertes Webinterface möglich. Der Administrator profitiert dabei von den aktuellen Statusinformationen, die ihn bei seinen Überlegungen unterstützen.
Dieser Bericht widerspiegelt das Vorgehen während der Bachelorarbeit und geht auf die einzelnen Phasen der Arbeit ein.
Die nachfolgenden Kapitel beschäftigen sich mit der Feststellung der **Anforderungen** und der **Analyse** aus welcher die Formulierung der Use Cases, Abuse Cases und des Domänenmodells resultierte. Im Abschluss wird die Analyse mit den Umsetzungen der Konkurrenz verglichen, um zu zeigen, wie sich diese Lösung von anderen Lösungen unterscheidet.
Anschliessend wird in der **Umsetzung** auf die Architektur eingegangen. Besonders wichtig bei der Umsetzung waren Überlegungen zur sicheren Kommunikation zwischen den einzelnen Komponenten. Ein weiterer Schwerpunkt liegt auf der Gestaltung der Benutzeroberfläche, welche der Hauptgrund für den Wechsel vom Shellscript zum Webinterface darstellt.
Das **Ergebnis** wird in einem eigenen Kapitel gezeigt und bewertet. Schliesslich zeigt der **Ausblick** mögliche Erweiterungsschritte.
Im letzten Kapitel wird auf das **Projektmanagement** eingegangen.

2. Anforderungen

2.1. Aufgabenstellung

Auf den nachfolgenden Seiten ist die unterschriebene Aufgabenstellung abgebildet.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Aufgabenstellung Bachelorarbeit
„Orchestrierung von Security-Updates für Linux-Serversysteme“
FS 2016

1. Auftraggeber und Betreuer

- *Auftraggeber:* Nine Internet Solutions AG, Albisriederstrasse 243a, 8047 Zürich
- *Ansprechpartner Auftraggeber:* Hr. Samuel Sieg, samuel.sieg@nine.ch, +41 44 637 40 18
- *Betreuer:* Prof. Dr. Farhad Mehta

2. Studierende

Diese Arbeit wird als Bachelorarbeit an der Abteilung Informatik durchgeführt von

- Hr. Philipp Christen
- Hr. Ueli Bosshard

3. Ausgangslage

Das Unternehmen nine.ch betreibt über 1500 Linux-Serversysteme mit unterschiedlichen Funktionen sowie Hardware- und Software-Ausstattungen. Die Installation der Software-Komponenten geschieht über herkömmliche Debian-Pakete. Für diese Pakete werden regelmässig Sicherheitsupdates veröffentlicht, welche zeitnah auf sämtlichen betroffenen Serversystemen installiert werden müssten.

Aktuell wird dies mit einer Sammlung von Bash-Scripts erreicht, welche manuell auf einzelnen Systemen aktiviert werden. Automatisches Auflösen von Abhängigkeiten oder zuverlässige Feedback-Mechanismen existieren nicht. Jede Woche verursacht dies mehrere Stunden Aufwand für einen Mitarbeiter. Zudem beträgt die Zeitspanne zwischen Veröffentlichung des Updates und der Installation auf einem System im schlimmsten Fall mehrere Tage.

Prof. Dr. Farhad Mehta ■ farhad.mehta@hsr.ch

HSR Hochschule für Technik Rapperswil ■ Oberseestrasse 10 ■ CH-8640 Rapperswil

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

4. Beschreibung der Aufgabe

Im Rahmen der Bachelorarbeit wird eine Software-Lösung entwickelt, welche das Einspielen der Updates auf diesen Serversystemen vereinfacht. Die Software bietet dem Systemadministrator die Möglichkeit, in Auftrag zu geben, auf welchen Systemen welche Updates durchgeführt werden sollen. Sie stellt zudem sicher, dass gegebene Regeln und Abhängigkeiten nicht verletzt werden. Solche Regeln können beispielsweise Abhängigkeiten von Systemen sein, etwa dass High-Availability-Cluster nicht alle gleichzeitig das gleiche Update einspielen und so gewisse Services nicht verfügbar sind oder dass bestimmte Updates zuerst auf Test-Umgebungen statt auf den produktiven Systemen installiert werden sollen.

Die Software zeigt darüber hinaus den aktuellen Zustand der Systeme an, d.h. sie zeigt, welche Updates wo ausstehend sind und welche Updates erfolgreich beziehungsweise fehlerhaft ausgeführt wurden.

Dabei müssen folgende Punkte beachtet werden:

1. Die Applikation kann über den Web-Browser bedient werden.
2. Die Anwendung soll es ermöglichen, anstehende Updates auf allen verbundenen Systemen einzusehen.
3. Der verfügbare Verlauf über installierte Updates soll ebenfalls ersichtlich sein.
4. Auf den Linux-Systemen, welche verwaltet werden soll, läuft ein Agent, welcher mit dem zentralen Managementsystem kommuniziert.
5. Die Kommunikation zwischen den Agents und dem Managementsystem soll verschlüsselt ablaufen.
6. Fehlermeldungen und benötigte Interaktionen für ein Update (z.B. Neustart eines Services oder des Systems) sollen dem Benutzer angezeigt und es sollen dafür Vorschläge angeboten werden.
7. Es wird ein detailliertes Logging darüber geführt, wer wann auf welchem System welche Updates freigegeben hat, bei Bedarf auch mit Kommentar/Grund.

Folgende nicht-funktionale Anforderungen sind besonders wichtig:

8. Zuverlässigkeit: Die Software hat jederzeit einen korrekten und konsistenten Datenstand. Die verschiedenen Szenarien sind mit Software-Tests geprüft.
9. Bedienbarkeit: Die Software ist für den Benutzer einfach zu bedienen und verständlich. Die Vorgänge sind nachvollziehbar.

Es gelten folgende Vorgaben und Rahmenbedingungen:

10. Die Software soll Open Source sein; es wird die MIT-Lizenz verwendet.
11. Es wird ausschliesslich Ubuntu (Versionen 12.04 und 14.04) als Server-Betriebssystem unterstützt.
12. Als Programmiersprache muss, wenn möglich, Ruby verwendet werden. Für die systemnahe Komponente (Agent) kann alternativ auch Python oder Go eingesetzt werden.
13. Als Datenbankserver wird PostgreSQL verwendet.

Die folgenden Punkte sind optional:

Prof. Dr. Farhad Mehta ■ farhad.mehta@hsr.ch

HSR Hochschule für Technik Rapperswil ■ Oberseestrasse 10 ■ CH-8640 Rapperswil

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

14. Die Agents sollen paketiert und über Puppet auf den Systemen verteilt werden können.
15. Die Applikation soll für die Gruppenbildung der Systeme eigene Vorschläge anbieten, z.B. aufgrund von Systemeigenschaften wie OS-Version, installierte Software, etc.

Die folgenden Punkte befinden sich nicht im Scope des Projekts und sind somit nicht Ziele der Bachelorarbeit:

16. Abhängigkeiten unter den Paketen auflösen - dies geschieht durch den Paketmanager des Betriebsystems.
17. Automatisches Durchführen von Updates aufgrund von Regeln - der Auslöser eines Updatevorganges ist immer ein menschlicher Benutzer.
18. Bereits installierte Updates können über die Applikation nicht entfernt werden, es wird kein Rollback angeboten.
19. Verfügbarkeit des Services - das Hosting findet bei nine.ch statt.

Die Deliverables umfassen zwei Komponenten:

20. Der Agent, welcher auf jedem einzelnen angeschlossenen System läuft und via apt die anstehenden Updates ausliest und ausführt.
21. Das Control-Center, welches auf einem zentralen Server läuft und wo Benutzer Informationen auslesen sowie Updates starten können.

5. Zur Durchführung

Mit dem Betreuer finden wöchentliche Besprechungen statt. Besprechungen mit dem Auftraggeber sind von den Studierenden nach Bedarf zu initialisieren.

Alle Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten, die Besprechung ist durch die Studierenden zu leiten und die Ergebnisse sind in einem Protokoll festzuhalten, das den Betreuern und dem Auftraggeber per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Dokumentation.

6. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html). Die zu erstellenden

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Dokumente sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Alle Resultate sind vollständig auf CD/DVD in 3 Exemplaren abzugeben. Der Bericht ist ausgedruckt in doppelter Ausführung abzugeben.

7. Termine

Siehe Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html.

8. Arbeitsumfang

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von 30 Stunden budgetiert. Die verwendete Arbeitszeit muss erfasst und dokumentiert werden.

9. Beurteilung

Für die Beurteilung ist der HSR-Betreuer verantwortlich. Die Benotung erfolgt gemäss folgender Tabelle.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/6
2. Berichte	1/6
3. Inhalt	3/6
4. Mündliche Prüfung zur Bachelorarbeit	1/6

Im Übrigen gelten die Bestimmungen der Abteilung Informatik für Bachelorarbeiten (siehe https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html).

Prof. Dr. Farhad Mehta
Rapperswil, den ~~14.09.2015~~ 21.03.2016

Prof. Dr. Farhad Mehta ■ farhad.mehta@hsr.ch

HSR Hochschule für Technik Rapperswil ■ Oberseestrasse 10 ■ CH-8640 Rapperswil

2.2. Ausgangslage

Die Aufgabenstellung fordert die Entwicklung einer spezifischen Software, womit sich diese Bachelorarbeit auf die Lösungserstellung fokussiert. Im Folgenden sind die Anforderungen als User Stories und in den Nicht-Funktionalen Anforderungen beschrieben und werden im darauffolgenden Kapitel analysiert.

2.3. User Stories

1. Als Benutzer möchte ich zu einem spezifischen System alle anstehenden Updates einsehen können, um mir schnell einen Überblick zu diesem System zu verschaffen.
2. Als Benutzer möchte ich Systeme gruppieren können, um sie später einfacher zu finden.
3. Als Benutzer möchte ich Pakete mehreren Paketgruppen zuordnen können, um sie später einfacher zu finden.
4. Als Administrator möchte ich weitere Benutzer anlegen können, so dass bei meiner Abwesenheit trotzdem Updates ausgeführt werden können.
5. Als Administrator möchte ich für Benutzer Rollen festlegen können, damit nicht jeder Benutzer einzeln konfiguriert werden muss.
6. Als Administrator möchte ich die Rechte von Benutzergruppen auf System- und Paketgruppen-Ebene definieren können, damit ich nicht für jeden Benutzer Rechte pro System und Paket festlegen muss.
7. Als Benutzer möchte ich auf einen Blick die wichtigsten Informationen zu all meinen Systemen sehen, um Problem sofort zu erkennen.
8. Als Benutzer möchte ich Pakete auf bestimmten Systemen gleichzeitig mit wenig Aufwand installieren lassen können, damit ich mich nicht bei jedem System einzeln anmelden muss.
9. Als Benutzer möchte ich sehen, ob ein Agent sich seit längerer Zeit nicht mehr gemeldet hat, um zu verhindern dass der betroffene Server während dieser Zeit ohne Aktualisierungen auskommen muss.
10. Als Benutzer möchte ich nach einer Auswahl von Systemen und Paketen, welche ich installieren lassen will, eine Übersicht der auszuführenden Aufgaben erhalten, damit ich Fehler verhindern kann.
11. Als Betreiber von Ubuntu-Servern ausserhalb der Firma nine.ch möchte ich, dass die Software Update Nine.ch (UPD89) Open Source ist, damit ich Verbesserungen oder neue Features selbst implementieren kann.
12. Als Benutzer möchte ich, dass die Kommunikation zwischen Agent und Control Center verschlüsselt abläuft, damit keine Man in the middle (MITM)-Attacken ausgeführt werden und so Angriffe gestartet werden können.

13. Als Benutzer möchte ich sehen, wer wann auf welchen Systemen welche Updates installierte, um bei etwaigen Problemen die Ursache schnell finden zu können.

14. Als Benutzer möchte ich im Control Center sehen, was apt bei einem Updatevorgang zurückgab, damit ich mich nicht bei jedem System einzeln anmelden und die Logfiles auslesen muss.

15. Als Administrator möchte ich neu registrierte Systeme einsehen können und diese gleich einer Systemgruppe zuweisen können, um immer über neue Registrationen informiert zu sein.

16. Als Benutzer möchte ich informiert werden, wenn ein Auftrag auf einem System fehlgeschlagen ist oder nicht ausgeliefert werden konnte, damit ich zeitnah prüfen kann, was mit diesem System nicht in Ordnung ist.

17. Als Benutzer möchte ich Systeme und Pakete nach verschiedenen Kriterien filtern und sortieren, damit ich mein gesuchtes Ziel schnell erreiche.

2.4. Nicht-funktionale Anforderungen

Technologien

Anforderungen an die Technologie gehen primär aus der Aufgabenstellung hervor:

- Bedienung über den Web-Browser
- Ruby als Haupt-Programmiersprache
- Rails als Web-Applikations-Framework
- Ubuntu als unterstütztes Betriebssystem zur Verwaltung mit den Versionen 12.04 und 14.04[1]

Weitere Frameworks, z.B. auf Javascript-Ebene, oder Bibliotheken werden dem Entwicklerteam überlassen.

Sicherheit und Nachvollziehbarkeit

Anforderungen an die Sicherheit und Nachvollziebarkeit gehen ebenfalls aus der Aufgabenstellung hervor:

- Kommunikation zwischen Agents und Managementsystem erfolgt verschlüsselt
- Es wird ein Log darüber geführt, wer welches Update freigegeben hat

Qualität

Zusammen mit dem Industriepartner wurden folgende Nicht-funktionale Anforderungen in Bezug auf die Qualität notiert:

- Zuverlässigkeit: nine.ch betreibt über 1500 Server. Diese müssen alle vom Control Center gesteuert werden können.
- Bedienbarkeit: Ein Benutzer mit durchschnittlichen Domänenkentnissen sollte in der Lage sein, die Applikation zu bedienen.

Open-Source

Die umgesetzte Softwarelösung soll mit einer Open-Source-Lizenz veröffentlicht werden, so dass neue Features oder Verbesserungen beliebig implementiert werden können.

[1] Ubuntu 16.04 war zu Beginn des Projektes noch nicht erschienen

3. Analyse

3.1. Lösungskonzept

Aus der Aufgabenstellung und den Sitzungen mit den Auftraggebern wurden zunächst Use- und Abuse-Cases entwickelt, welche die Arbeitsabläufe mit der zu erstellenden Lösung aufzeigen.
Ebenfalls als sehr nützlich herausgestellt haben sich die User-Interface-Mockups um Abläufe durchzudenken, ohne einen klickbaren Prototypen erstellen zu müssen.

3.2. Use Cases

In diesem Kapitel werden die Anforderungen an die Software in Form von Use-Cases (Anwendungsfällen) dokumentiert. Die Use-Cases wurden aufgrund des Auftrages und weiteren Besprechungen mit dem Industriepartner definiert. Abbildung 3.1 zeigt eine Übersicht über die Use-Cases für den Agent sowie das Control Center.

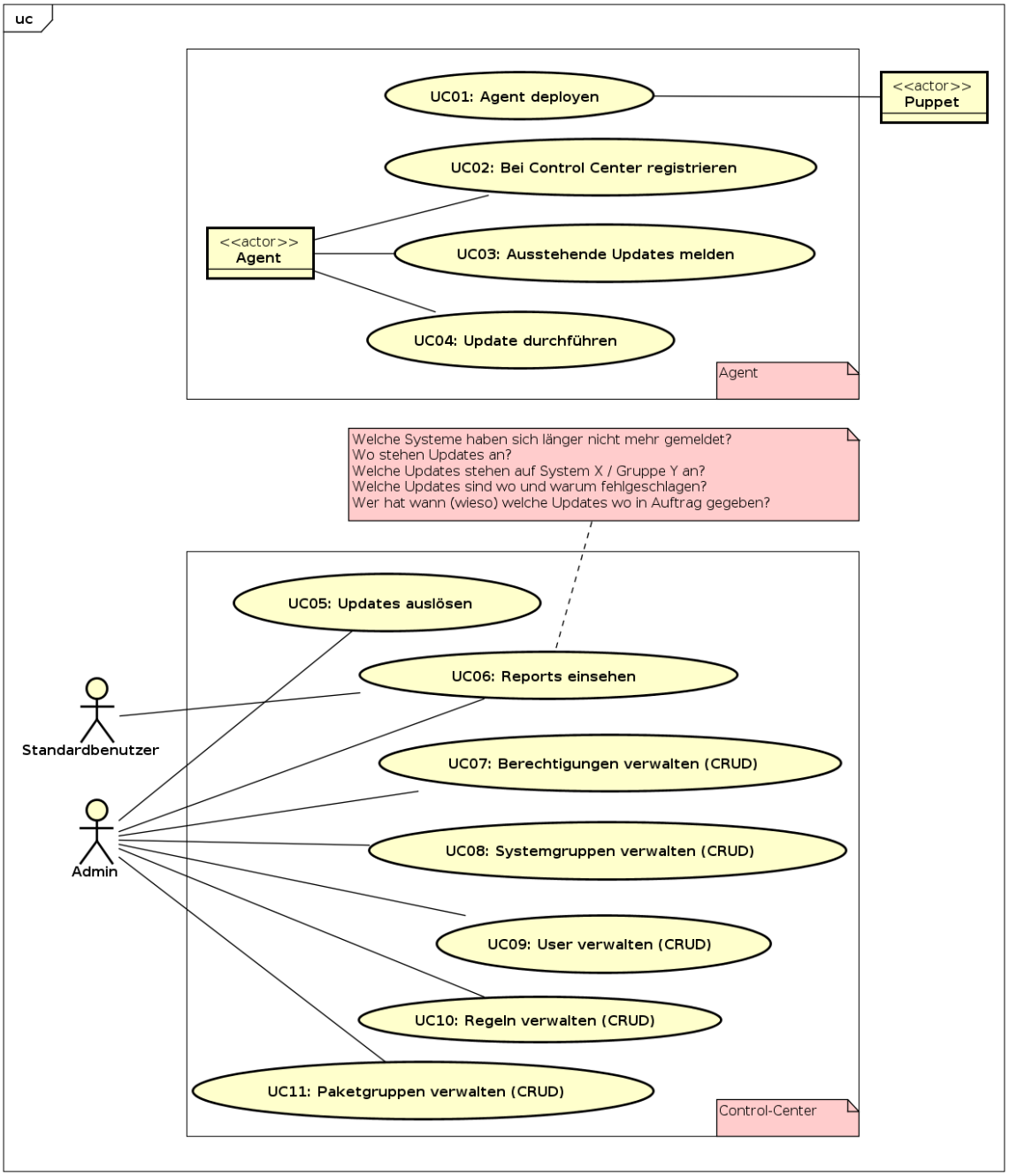


Abbildung 3.1.: Übersicht der Use Cases

UC01: Agent deployen

Der Agent wird auf einem System ausgeliefert. Dabei werden Parameter zur Konfiguration des Agents bei der Installation mitgegeben.
Technische Details:

- Das Paket wird als .deb via Puppet ausgeliefert
- Konfiguration des Agents enthält: Adresse des Control-Centers und das CA-Zertifikat

UC02: Registration von Agent an Control Server

Ein Agent registriert sich beim Control-Center.
Technische Details:

- Registration geschieht über die beim Deployment erhaltene Adresse und dem generierten Zertifikat
- Der Agent übermittelt seinen Fully Qualified Domain Name

UC03: Ausstehende Updates melden

Ein Agent aktualisiert die Paketliste. Er prüft, ob Updates anstehen und meldet diese Liste dem Control-Center. Gleichzeitig sendet er auch noch Hostinformationen, damit das Control-Center seine Informationen ggf. aktualisieren kann.
Technische Details:

- Paketliste wird via apt-Schnittstelle aktualisiert
- Hostinformationen enthalten seine OS-Version sowie Hostnamen

UC04: Update durchführen

Ein Agent erhält vom Control-Center einen Task mit einer Liste von vorzunehmenden System-Updates. Er prüft, ob diese bei sich anstehen und führt diese durch. Das Empfangen der Updates wird dem Control-Center bestätigt, wo der Status dieses Tasks auf 'Queued' gesetzt wird.

UC05: Update auslösen

Das Control-Center hat eine Liste von ausstehenden Updates von einem oder mehreren Systemen erhalten. Diese zeigt es in einer Übersicht an. Ein User mit entsprechender Berechtigungs-Stufe gibt eines oder mehrere dieser Updates frei. Das Control-Center erstellt daraus Tasks, welche der User freigeben kann. Falls die Tasks freigegeben werden, schickt das Control-Center diese an die jeweiligen Systeme.

UC06: Statusreport anzeigen

Ein Benutzer lässt sich vom Control-Center eine Liste von Systemen und Updates anzeigen. Der User sieht den Zustand jedes Systems und Updates.

UC07: Berechtigungen verwalten

Ein Administrator kann für Systemgruppen und Paketgruppen Berechtigungs-Stufen setzen, so dass nur User mit dieser Stufe Aktionen auf diesen Gruppen vornehmen können.

UC08: Systemgruppen verwalten

Ein Administrator kann eine System-Gruppe erstellen, bearbeiten oder löschen. Dies beinhaltet auch das Entfernen oder Hinzufügen von Systemen sowie das Setzen von Berechtigungs-Stufen.

UC09: User verwalten

Ein Administrator kann neue User erstellen, Berechtigungs-Stufen ändern oder bestehende User löschen.

UC10: Regeln verwalten

Ein Administrator kann Regeln für Systemgruppen erstellen, ändern und entfernen. Diese Regeln greifen bei der Auswahl der Systeme und Pakete.

UC11: Paketgruppen verwalten

Ein Administrator kann neue Gruppen für Pakete erstellen, bearbeiten oder löschen. Dies beinhaltet auch das Entfernen oder Hinzufügen von Paketen sowie das Setzen von Berechtigungs-Stufen.

3.3. Abuse Cases

Bei der Lösung handelt es sich um ein verteiltes System, das über unsichere Netze kommuniziert. Daraus ergeben sich Angriffspunkte, auf die bei der Umsetzung besonders geachtet werden muss.

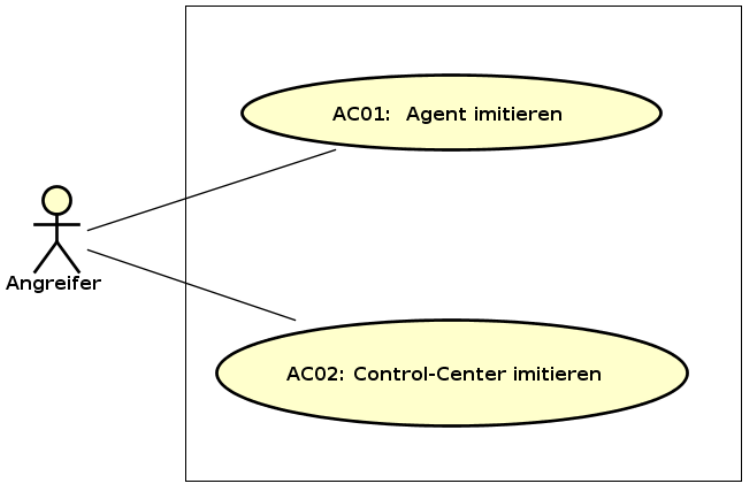


Abbildung 3.2.: Übersicht der Abuse Cases

AC01: Agent imitieren

Szenario: Ein Angreifer gibt sich als Agent aus und meldet sich beim Control Center an. Damit ist es dem Angreifer möglich, verfälschte Daten an das System zu schicken und dem Systemadministrator gezielt falsche Informationen zu vermitteln. Im schlimmsten Fall trifft der Systemadministrator falsche Entscheidungen.

AC02: Control-Center imitieren

Szenario: Ein Angreifer gibt sich als falsches Control Center aus und meldet sich beim Agent. Der Angreifer kann Aufträge für Software-Updates erteilen und so direkt auf das System Einfluss nehmen.

3.4. Domäne

Das nachfolgende Domänen-Modell (Abbildung 3.3) zeigt alle nötigen Attribute für die Umsetzung der Applikation. Es beinhaltet die von den Agents erhaltenen Daten (grün), die konfigurierbaren Daten, wie beispielsweise Benutzer, Rollen und Gruppenzugehörigkeiten (rot) und ausserdem die Informationen, welche für die Auftragsverwaltung benötigt werden (gelb).

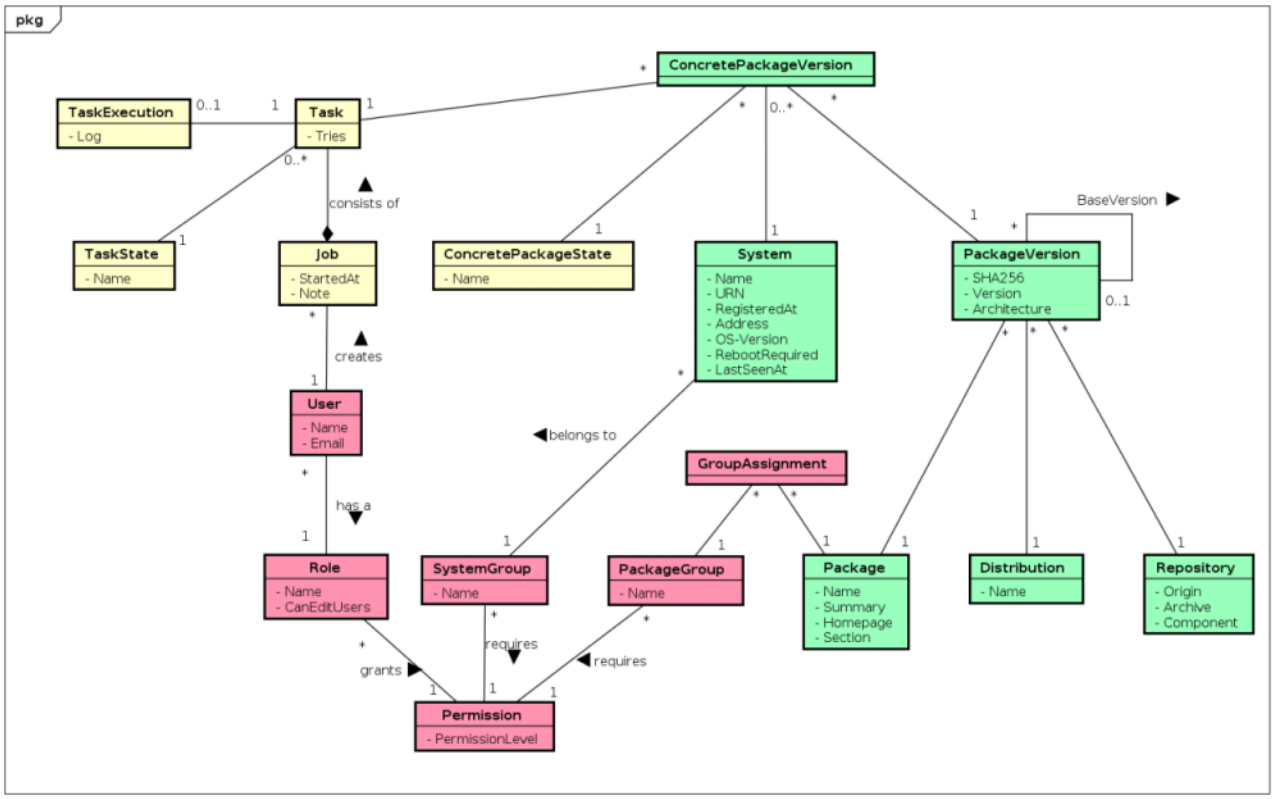


Abbildung 3.3.: Domänen-Modell

Erläuterungen

System

Ein System repräsentiert einen einzelnen Server, auf welchem die Updates verwaltet werden sollen. Relevante Daten über die Systeme werden erfasst oder beim Registrieren durch den Agent mitgeteilt.

Package

Ein Package ist ein einzelnes Applikationspaket, welches vom Paketmanager (hier: apt) verwaltet wird. Beispiele dafür sind vim, openssl oder sudo. Das Paket selbst ist unabhängig von Versionen oder Distributionen.

PackageVersion

Eine Paket-Version ist eine verfügbare Version eines bestimmten Paketes auf einem spezifischen Repository/einer Distribution. Eine spezifische Paket-Version kann eine Base-Version sein. Das bedeutet, dass diese Paket-Version zusammen mit der Distribution ausgeliefert wurde. Wenn eine Paket-Version keine Base-Version ist, kann sie aber einen optionalen Verweis auf eine frühere Version besitzen.

ConcretePackageVersion

Eine Paket-Version kann mit einem System in Verbindung gebracht werden, zum Beispiel indem der Agent eine bestimmte Paket-Version als ausstehendes Update meldet. Diese Verbindung ist eine konkrete Paket-Version und zeigt, welchen Zustand diese Version auf diesem Paket hat (definiert durch einen **ConcretePackageState**):

- Installiert
- Veraltet
- Ausstehend
- Fehlgeschlagen
- In Auftrag gegeben

Distribution

Eine Distribution, oft auch 'Distro' genannt, ist ein Betriebssystem basierend auf dem Linux-Kernel. Beispiele: Debian, Suse oder Ubuntu. Diese werden wiederum in spezifische Distros unterteilt, etwa bei Ubuntu auf Ubuntu 12.04 precise, 14.04 trusty oder 16.04 xenial.

Repository

Ein Repository ist eine Sammlung von Paketen für eine bestimmte Distribution.

Job

Ein Job wird dann erstellt, wenn ein Benutzer einen Auftrag erteilt. Pro betroffenem System enthält dieser Job einen Task, welcher dann an die Agenten geschickt wird.

Task

Ein Task enthält alle konkreten Aufträge für ein spezifisches System, welche von einem User in einem Durchgang erfasst wurden. Ein Task kann sich in den folgenden Zuständen (**TaskState**) befinden:

- Erledigt
- Fehlgeschlagen
- Ausstehend
- In Bearbeitung

- Nicht zugestellt

Ein Task kann eine **TaskExecution** beinhalten, welche bei der Rückmeldung vom Agent mitgeliefert wird. Sie enthält lediglich die Ausgabe von apt, welches vom Agent direkt weitergeschickt wird.

User

Ein User ist ein Benutzer, welcher sich über seine Email-Adresse und sein Passwort identifiziert und einen Namen und eine Rolle (**Role**) besitzt.

Role

Eine Rolle definiert, zu welcher Gruppe ein Benutzer gehört und welche Rechte dieser besitzt. Einerseits wird definiert, ob andere Benutzer bearbeitet werden können, andererseits wird ein **PermissionLevel** zugewiesen.

PermissionLevel

Paketgruppen, Systemgruppen sowie Rollen besitzen alle ein PermissionLevel. Diese Zahl bestimmt, ob ein Benutzer mit einer bestimmten Rolle das Recht besitzt, Systeme oder Pakete in spezifischen Gruppen zu bearbeiten und zu aktualisieren. Dazu muss das PermissionLevel der Rolle des Benutzers höher sein als das benötigte PermissionLevel der Gruppe des Systems.
Bei Paketen, welche in mehreren Gruppen sein können (über ein **GroupAssignment**), gilt das höchste PermissionLevel.

3.5. Konkurrenz

Hier werden bereits existierende Lösungen für ähnliche Problemstellungen vorgestellt und analysiert. Folgende Lösungen wurden untersucht:

- Landscape (Canonical)
- Ansible
- Spacewalk (Red Hat)
- WSUS (Microsoft)
- Puppet

Landscape

Produkt	Landscape
Hersteller	Canonical
Beschreibung	Landscape von Canonical ist ein proprietärer Web-Service für das Verwalten von Ubuntu-Systemen an einem zentralen Ort. Es kann Software verwaltet und installiert werden, inklusive Security-Updates. Updates können auch automatisch installiert oder wieder entfernt werden. Landscape unterstützt neben Servern auch Desktops.
Features	• Deployment • Software und Update Management inkl. Repository-Verwaltung • Monitoring, inkl. Prozess-Zugriff sowie div. Hardware-Abfragen • RBAC (Role-Based Access Control)
Lizenz	Proprietär
Kosten	Benötigt eine Advantage-Lizenz, welche im Minimum 320 USD pro Server pro Jahr kostet.[1]
URL	landscape.canonical.com/
Eignung	Abgesehen von den hohen Kosten ist Landscape nicht Open Source. Die Lösung ist auch zu umfangreich und bietet viele Funktionen, welche bei nine.ch bereits durch andere Tools abgedeckt werden (z.B. Deployment, Monitoring).

[1] Quelle: www.ubuntu.com/management/ubuntu-advantage

Ansible

Produkt	Ansible
Hersteller	AnsibleWorks, Inc.
Beschreibung	Ansible ist eine Plattform zur Automatisierung von Tasks und kombiniert Softwareverteilung, Kommando-Ausführung und Konfigurationsmanagement. Server werden via SSH angesprochen, es ist kein Agent erforderlich. Ansible Tower ist die zentrale Web-basierte Kommandozentrale und ist Open-Source.[2]
Features	• Applikations-Deployment • Konfigurations-Management • User-definierbare Playbooks für Tasks • REST-Schnittstelle für Einbinden in andere Systeme
Lizenz	GNU General Public License v3
Kosten	Im Enterprise-Paket kostet Ansible Tower 50 USD pro Server pro Jahr für $>$ 1000 Server.[3]
URL	www.ansible.com/
Eignung	Hier beläuft sich der totale Preis bei 1500 Systemen auf 75'000 USD pro Jahr, nur um die Updates zu verwalten. Ansible, vor allem auch mit Ansible-Galaxy kombiniert, bietet viel mehr Funktionalität, als nine.ch benötigt.

Spacewalk

Produkt	Spacewalk
Hersteller	Red Hat
Beschreibung	Spacewalk ist eine Systemmanagement-Lösung für Linux-Systeme und ist Open Source. Es basiert auf einer Community, welche auch den Support übernimmt. Es bietet unter anderem auch ein Update-Management über Systeme und Systemgruppen hinweg. Das Ganze funktioniert über eine zentrale Web-Applikation. Spacewalk unterstützt Fedora, CentOS, SLE und Debian.
Features	• Inventar von Hardware und installierter Software • Installation von Software und Updates • Konfigurationen verteilen und verwalten • Content-Verteilung
Lizenz	GNU General Public License v2
Kosten	Kostenlos[4]
URL	spacewalk.redhat.com/
Eignung	"Support für Debian/Ubuntu experimentell, Eierlegende Wollmillchsau"[5]

[2] Quelle: www.ansible.com/subscription-agreement
[3] Quelle: www.ansible.com/pricing
[4] Quelle: spacewalk.redhat.com/faq.html
[5] Quelle: interne Evaluation von nine.ch

WSUS

Produkt	Windows Server Update Services
Hersteller	Microsoft
Beschreibung	Windows Server Update Services ist eine proprietäre Update-Software von Microsoft für zentralisiertes Verwalten von Patches für Microsoft-Server.
Features	• Verteilen der Updates, einmaliges Herunterladen • Client-Komponente welche dem Administrator Status-Details liefert • Gruppieren von Systemen
Lizenz	Proprietär
Kosten	Kostenlos[6]
URL	technet.microsoft.com/de-de/windowsserver/bb332157.aspx
Eignung	WSUS eignet sich nur für Microsoft-Produkte. Da nine.ch ausschliesslich Linux-basierende Server verwendet, fällt WSUS aus technologischen Gründen weg.

Puppet

Produkt	Puppet
Hersteller	Puppet Labs
Beschreibung	Puppet ist ein Tool um Systeme zu konfigurieren, Software zu installieren, Programme auszuführen sowie Daten zu synchronisieren. Puppet ist als Open-Source oder als kostenpflichtige Enterprise-Version erhältlich.
Features	• Unterstützt verschiedene Linux- und Windows-Systeme • in Ruby geschrieben • deklarative Sprache zur Ressourcenbeschreibung für Systemkonfigurationen • Monitoring (Puppet Dashboard) • Client-Server-Architektur mit einem Daemon auf jedem Node
Lizenz	Apache 2.0 (seit 2.7.0, früher GPL)
Kosten	Kostenlos als Open-Source[7] oder 120 USD pro Jahr pro Server mit Standard-Support als Enterprise-Verion[8]
URL	puppetlabs.com/
Eignung	Puppet ist bereits bei nine.ch im Einsatz. Leider eignet sich Puppet nicht für die Updateverwaltung[9]

[6] Quelle: technet.microsoft.com/en-us/windowsserver/bb332157
[7] Quelle: info.puppetlabs.com/open-source-puppet-download.html
[8] Quelle: puppetlabs.com/puppet/how-to-buy
[9] Quelle: ask.puppetlabs.com/question/6576/linux-patch-management-via-puppet/

4. Umsetzung

Dieses Kapitel beschreibt die Umsetzung anhand der Architektur und legt einen Schwerpunkt auf die Themen Sicherheit und Design.

4.1. Architektur

Control Center

Das Control Center wurde mit Ruby on Rails umgesetzt. Rails ist ein Model-View-Controller (MVC)-Framework, wodurch grundsätzliche Entscheidungen zur Architektur bereits vorweggenommen wurden.

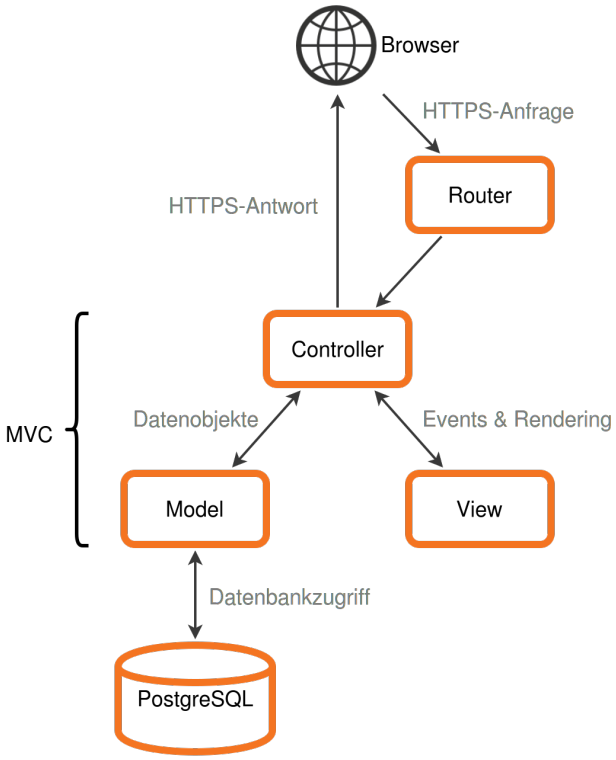


Abbildung 4.1.: MVC-Pattern in Rails-Umgebung

Ruby on Rails setzt standardmässig das Active-Record-Pattern ein. Ein 'Active Record' nimmt in Rails die Rolle des Models ein ('das M in MVC'[1]) und repräsentiert die Schicht im System, welche für die Business Logic zuständig ist.

[1] guides.rubyonrails.org/active_record_basics.html

Das MVC-Pattern teilt Applikationen in drei Teile, welche jeweils verschiedene Aufgaben und Verantwortungen haben. Ziel ist, die Wiederverwendbarkeit und Trennbarkeit der Komponenten zu gewährleisten.
Ruby on Rails bildet das Pattern bereits in der Dokumentenstruktur ab. Im Verzeichnis /app befinden sich Ordner für Controllers, Models und Views.

Entscheid 4.1: Rails-Version

Als Rails-Version wurde 4.2 gewählt, obwohl Rails 5 im Mai 2016 als Release Candidate erschienen wäre[a] und Features wie z.B. WebSockets beinhaltet hätte. Es wurde aber gegen den Gebrauch der neuen Version entschieden, da sie zum Zeitpunkt der Evaluation noch nicht aus dem Beta-Stadium heraus war und die Kompatibilität mit verschiedenen Ruby-Gems nicht immer gewährleistet ist.

[a] Quelle: weblog.rubyonrails.org/2016/5/6/this-week-in-rails-railsconf-recap-rails-5-0-rc-1-is-out/

Gems

Rails ist ein Gem für Ruby. Gems sind Pakete für den Paketmanager von Ruby (RubyGems), womit für eine Ruby-Applikation vorgefertigte Bibliotheken und weitere Applikationen eingebunden werden können.
Unter anderem wurden bei UPD89 folgende Gems verwendet:

sass-rails	Kompiliert Sass- und Scss-Files nach CSS
coffee-rails	Ermöglicht das Benutzen von Coffeescript[2].
jquery-rails	jQuery wird für DOM-Manipulationen und AJAX benutzt.
rspec-rails	Testing-Framework für Rails.
factory_girl_rails	Stellt Fixtures für Unit-Testing zur Verfügung.
rubocop	Prüft den Stil des Codes gemäss den Ruby Code-Guidelines.
faker	Ermöglicht einfaches 'Faken' von Daten für Unit-Testing.
better_errors	Hilft beim Debugging im Gegenteil zu den standardmässig eher unspektakulären Fehlermeldungen.
rake	'Software-Management Task Tool', hilft beim Ausführen von Aufgaben wie z.B. dem Erstellen von Default-Daten für die Datenbank.
font_awesome_rails	Bindet die Icons von FontAwesome ein.
will_paginate	Ermöglicht Pagination von ActiveRecord-Einträgen.

[2] coffeescript.org/

sucker_punch	Asynchrone Hintergrund-Aufträge, wird für das Senden von Tasks an die Agents benötigt.
faraday	Bibliothek für HTTP-Zugriffe mit HTTPS-Support.
sorcery	Benutzer-Authentifizierung und Login-Funktionalität.
filterrific	Unterstützung für Filter und Sortiermöglichkeiten.
cancancan	Berechtigungen auf Aktions-Ebene (CRUD).

Entscheid 4.2: Javascript-Framework

Nach anfänglicher Überlegung wurde gegen ein grosses Javascript-Framework wie AngularJS entschieden. Obwohl bereits Erfahrungen mit AngularJS[a] im Team vorhanden sind, wurde es als zu umfangreich eingestuft. Rails sollte nicht nur als Backend benutzt werden, sondern auch gleich für die Views zuständig sein. Die Industriepartner teilten diese Meinung.

[a] angularjs.org/

BackgroundTask

Das Senden der Tasks an die einzelnen Agents wird asynchron ausgeführt. Dazu wird das Gem 'sucker_punch' verwendet. Nachdem der Benutzer den Job mit den zu versendenden Tasks bestätigt, wird pro betroffenem System ein Hintergrund-Task ('BackgroundTask') gestartet. Dieser versucht eine HTTPS-Verbindung mit dem Webserver auf dem Agent aufzubauen und als Paylod die Task-Daten mitzuschicken. Falls die Verbindung nicht aufgebaut werden konnte, ruft sich der Hintergrund-Task mit einer Verzögerung erneut auf, bis die maximale Anzahl Versuche erreicht wurde.
Die Verzögerung zwischen den Versuchen sowie die maximale Anzahl Versuche kann vom Benutzer konfiguriert werden.
Der Hintergrund-Task enthält eine Referenz auf den Task über dessen ID, so dass der Task nach Fehlschlagen oder erfolgreichem Übermitteln auf den entsprechenden Zustand gesetzt werden kann.

DataMutationService

Der API-Controller, welcher von den Agents bei Benutzung der API aufgerufen wird, leitet die Anfragen an den DatenMutationService weiter. Dieser Service ist verantwortlich für das Anlegen und Mutieren von bestehenden Objekten. Operationen, bei welchen nur ein neuer Eintrag angelegt werden muss, wenn noch keiner mit diesen Parametern existiert, werden von den entsprechenden Models selbst behandelt.

Berechtigungen

Um zu prüfen, ob ein Benutzer eine Aktion ausführen darf oder nicht, wird 'cancancan' als Hilfs-Bibliothek verwendet. In der Klasse 'Ability' wird für jede Aktion von jedem gewünschten Controller definiert, welche Voraussetzungen erfüllt werden müssen damit sie ausgeführt werden kann. Auf

diese Art ist das Verantwortung über das Verwalten der Berechtigungen an einem einzelnen Ort konzentriert und nicht in jedem einzelnen Controller verteilt.
Ob ein Benutzer beispielsweise einen User bearbeiten darf, hängt davon ab ob die Rolle des ausführenden Benutzers das Flag 'is_user_manager' gesetzt hat:

Programmcode 4.1: Klasse Ability - Nur UserManager dürfen User erstellen/ändern

```
can :read, User
can [ :create, :update ], User do
  user.role.is_user_manager
end
```

Das Prüfen der Berechtigung erfolgt nach dem folgenden Schema:

Programmcode 4.2: Kontrolle ob Aktion ausgeführt werden darf

```
if can? :update, user
    # is allowed
else
    # is not allowed
end
```

Agent

Der upd89-agent wurde in Python umgesetzt. Um die Übersichtlichkeit und Wartbarkeit zu gewährleisten, wurde der Code zu einem grossen Teil in Klassen und Libraries unterteilt.

Klassen

Die Klassen orientieren sich an dem Klassenmodell (Abbildung 4.2) des Control Centers. Sie dienen hauptsächlich dazu, Daten zu erfassen und serialisiert an das Control Center zu schicken.

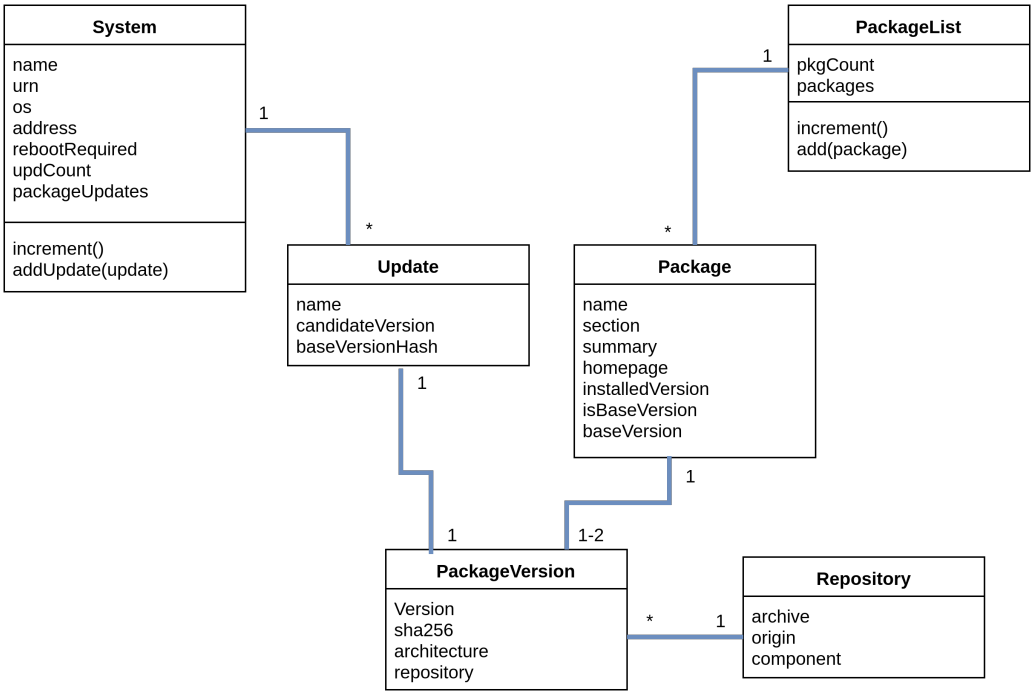


Abbildung 4.2.: Klassenübersicht für den Agent

Eigene Libraries

bottletlsdaemon.py	Lässt einen Webserver als Daemon im Hintergrund laufen und stellt die Anforderungen an die HTTPS-Verbindung sicher. Wurde inspiriert durch das Python-Modul BottleDaemon[3].
configloader.py	Liest Werte aus einer angegebenen Konfigurationsdatei.
httpsclientauthconnection.py	Stellt sicher, dass Verbindungen mit einem Clientzertifikat abgesichert sund und dass die Gegenstelle mit einem spezifischen Certificate Authority-Zertifikat unterschrieben wurde.
log.py	Loggt bei Bedarf auf den Bildschirm oder in eine Datei.
mission.py	Koordiniert die Abläufe für eine spezifische Aufgabe.

[3] pypi.python.org/pypi/BottleDaemon/0.1.2

persist.py	Abstrahiert die persistente Speicherung von Daten. Im Hintergrund wird shelve benutzt.
pkg.py	Abstrahiert die Zugriffe auf den Paketmanager.
sysinfo.py	Liefert Informationen zum System, zum Beispiel Hostname, IP-Adresse oder Distribution.
upstream.py	Konvertiert Informationen in ein JSON-Objekt und ermittelt die Ziel-URL.

Abhängigkeiten

Folgende Abhängigkeiten auf externe Python-Module existieren:

apt[4]	Für den Zugriff auf Debian-Packetmanager "apt".
daemonize[5]	Um den Agent als Daemon zu starten.
configparser[6]	Liest Werte aus einer Konfigurationsdatei.
bottle[7]	Minimalistisches Web-Framework, um vom Control Center angesprochen zu werden.
shelve[8]	Für persistente Datenspeicherung.

Paketierung

Es wurde ein Python-Paket[9] erstellt, welches über pip installiert werden kann.

[4] apt.alioth.debian.org/python-apt-doc/
[5] pypi.python.org/pypi/daemonize
[6] pypi.python.org/pypi/configparser
[7] bottlepy.org/docs/dev/index.html
[8] docs.python.org/2/library/shelve.html
[9] Verfügbar auf pypi.python.org/pypi/upd89

API

Agent und Control Center kommunizieren miteinander direkt über eine Application Programming Interface (API). Dies wurde aus folgenden Gründen beschlossen:

- einfache Erweiterbarkeit: Momentan werden nur Ubuntu-Server (mit apt) unterstützt. Die API ist agnostisch gegenüber dem Betriebssystem und kann theoretisch auch andere Paketmanager unterstützen.
- lose Kopplung: Die Komponenten Agent und Control Center funktionieren als eigenständige und unabhängige Applikationen.
- bereit für Middleware: durch die Trennung der beiden Haupt-Komponenten kann eine Middleware, etwa eine Message Queue (siehe auch Kapitel 6.3) ohne grossen Aufwand eingeführt werden.

Entscheid 4.3: Message Queue

Es wurde beschlossen, dass die Komponenten direkt miteinander kommunizieren anstatt eine Message Queue zu benutzen. Eine Message Queue würde bedeuten, dass jeder Agent eine eigene Queue bräuchte. Das Thema wird im Ausblick (Kapitel 6.3) näher behandelt.

Die API selbst ist unkompliziert und zielt darauf ab, dem Entwickler eines Agents Arbeit abzunehmen sowie die Logik im Control Center zu behalten.
Um den Netzwerkverkehr in Grenzen zu halten, wurde darauf verzichtet, bei jedem Aufruf alle Informationen zu einem Paket mitzuschicken, sondern es wird hauptsächlich mit dem Hash des Paketes gearbeitet. Sollte das Control Center einen Hash nicht kennen, wird dies dem Agent mitgeteilt und die vollen Paketinformationen werden nachgereicht.
Zudem wird mit Deltas gearbeitet: Der Agent sendet jeweils nur neue Informationen wie etwa neue anstehende Updates. Das Control Center prüft jeweils, ob die Anzahl der gesamthaft vorhandenen Updates noch mit dem Kontrollwert übereinstimmt, welcher ebenfalls durch den Agent übermittelt wird. Ist eine Diskrepanz vorhanden, muss der Agent die gesamten Informationen erneut übermitteln, da eventuell ein Update auf einem anderen Weg installiert wurde. Allerdings kann hier wieder mit der Liste der Hashes gearbeitet werden, womit sich die Last auf dem Netzwerk in Grenzen hält.
Die API ist Bestandteil des Control Centers, da die meisten Aufrufe vom Agent zum Control Center hin verlaufen. Aber auch in der Gegenrichtung existiert eine (kleine) API. Somit kann auch das Control Center dem Agent Tasks ausliefern. Weitere API-Endpunkte auf Agent-Seite sind durchaus denkbar, aber nicht Bestandteil dieser Arbeit (Siehe auch Kapitel 6.6).

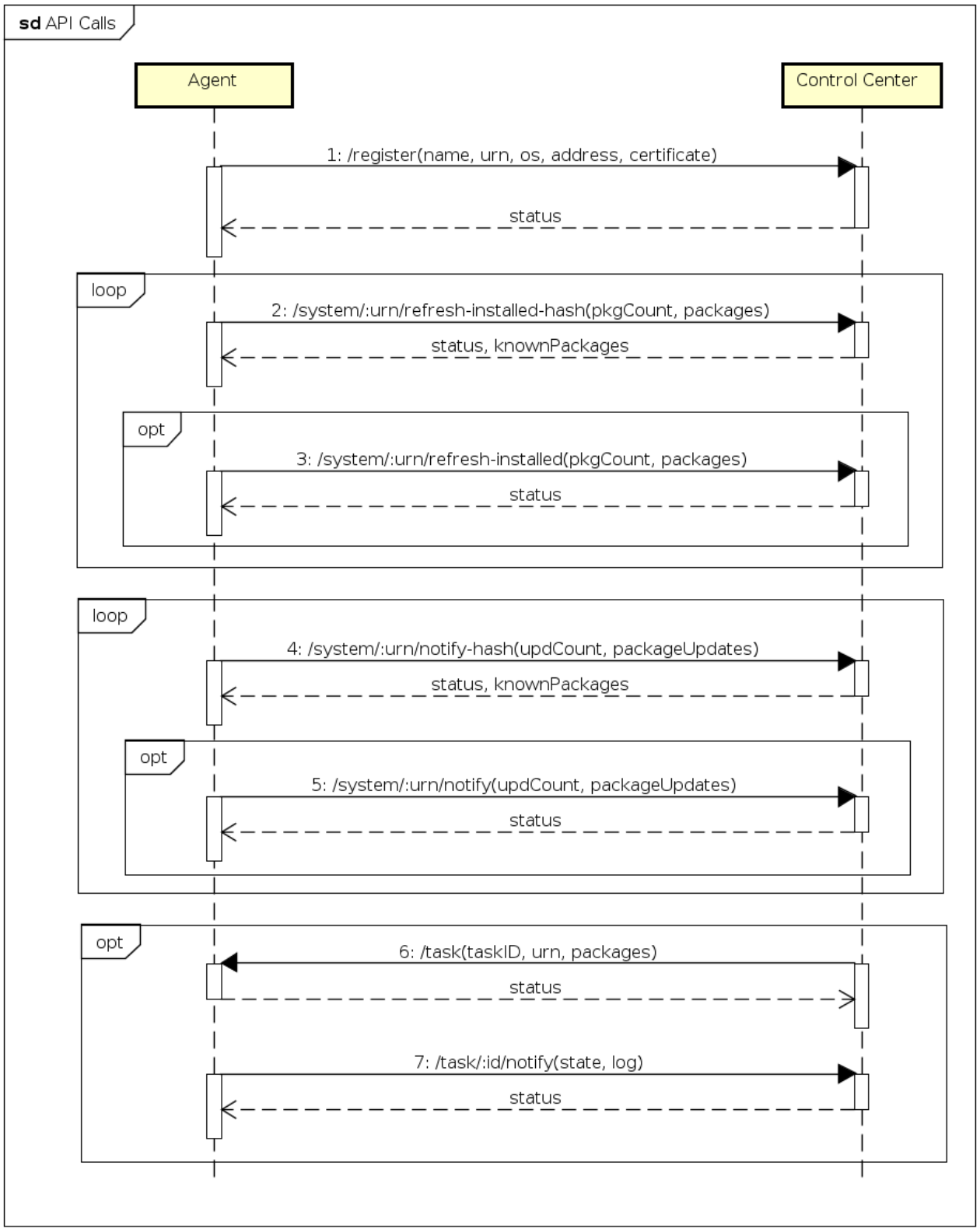


Abbildung 4.3.: Sequenzdiagramm bezüglich API-Aufrufe

Ablauf

Der typische Ablauf einer 'Unterhaltung' zwischen Agent und Control Center läuft folgendermassen ab (Grafisch dargestellt in Bild 4.3):

1. Registrierung: Nach dem Deployment und Start des Agents nimmt dieser Kontakt mit dem Control Center auf und registriert sich. Dazu übermittelt er seine Eckdaten wie Name, Betriebssystem und Adresse.

2. Installierte Pakete (Hashes): Damit das Inventar beim Control Center stimmt, schickt der Agent all seine momentan installierten Pakete. Dies umfasst nicht die anstehenden Updates. Zuerst werden nur die Hash-Werte der einzelnen Paketversionen geschickt.

3. Installierte Pakete (Komplett): Sollte das Control Center nicht alle gesendeten Hashes kennen, schickt der Agent die vollen Informationen.

4. Updates (Hashes): In regelmässigen Abständen prüft der Agent über apt, ob Updates anstehen. Ist das der Fall, übermittelt er die Hashes an das Control Center.

5. Updates (Komplett): Wenn Hashes unbekannt sind, meldet das Control Center dies und der Agent schickt erneut die kompletten Angaben zu den Updates.

6. Auftrag: Ein Benutzer löst auf dem Control Center ein Update auf diesem System aus, worauf ein Task an den entsprechenden Agent geschickt wird. Dieser enthält Versionsnummer und Namen der Pakete, welche aktualisiert werden sollen sowie eine Task-Nummer um den Task auf dem Control Center zu referenzieren.

7. Feedback: Der Agent versucht, das Update über apt auszuführen. Die Ausgaben von apt werden gesammelt und zusammen mit dem erreichten Ergebnis (Erfolgreich oder Fehlgeschlagen) zurück an das Control Center geschickt mit der vorhin erhaltenen Task-Nummer.

Punkte 2 oder 3 werden alle 2 Stunden ausgeführt, Punkte 4 und 5 alle 10 Minuten.

4.2. Sicherheit

In der Analyse der Anforderungen wurde festgestellt, dass auf die Sicherheit ein besonderes Augenmerk gelegt werden muss. Insbesondere die festgestellten Abuse Cases (Siehe 3.3) mussten berücksichtigt werden.

Übersicht

Im Fokus stand eine vertrauenswürdige Verbindung zwischen allen Teilnehmern des Systems. Namentlich sind das die Agents, das Control Center, sowie der Administrator, der per Browser auf die Administrationsoberfläche zugreift.

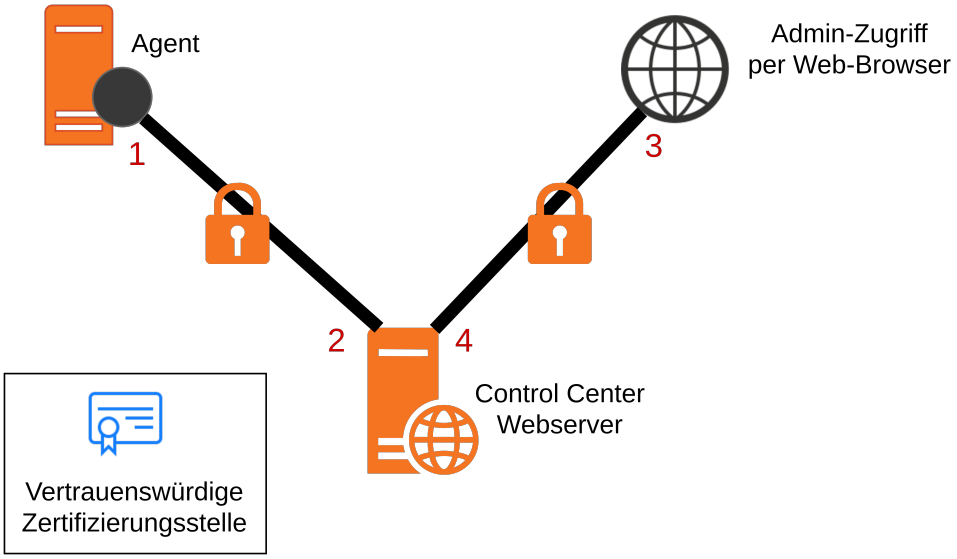


Abbildung 4.4.: Übersicht Verbindungssicherheit

Jeder Teilnehmer muss sich sicher sein, dass er mit dem korrekten Gegenüber kommuniziert.

1 Der Agent überprüft das Zertifikat des Control Centers, damit er weiss, dass er den erhaltenen Auftrag ausführen darf. Ausserdem stellt er sicher, dass er Statusinformationen an das korrekte Gegenüber abliefert.

2 Das Control Center seinerseits stellt ebenfalls mit dem Zertifikat des Agents fest, ob er dem Gegenüber vertrauen kann und die gelieferten Informationen für ihn relevant und korrekt sind.

3 Der Administrator kann anhand des Zertifikats des Servers feststellen, ob er mit dem korrekten Control Center verbunden ist.

4 Das Control Center vertraut dem Administrator anhand des eingegebenen Passworts.

API Security

Um die Kommunikation zwischen Agent und Control Center abzusichern, wird eine eigene Zertifizierungsstelle (CA) eingesetzt. Im Grunde kann die CA mit Openssl [10] erstellt werden. Als Unterstützung wird easy-rsa [11] verwendet, um die benötigten Zertifikate mit wenig Aufwand zu generien.

Erzeugen der Zertifikate

Das Konzept einer Zertifizierungsstelle (CA) sieht wie folgt aus: Die CA unterschreibt den eigenen öffentlichen Schlüssel (pub) und generiert daraus das öffenliche CA-Zertifikat (CA-crt). Anschliessend unterschreibt die CA die öffentlichen Schlüssel der Teilnehmer, in unserem Fall das Control Center und die Agents.

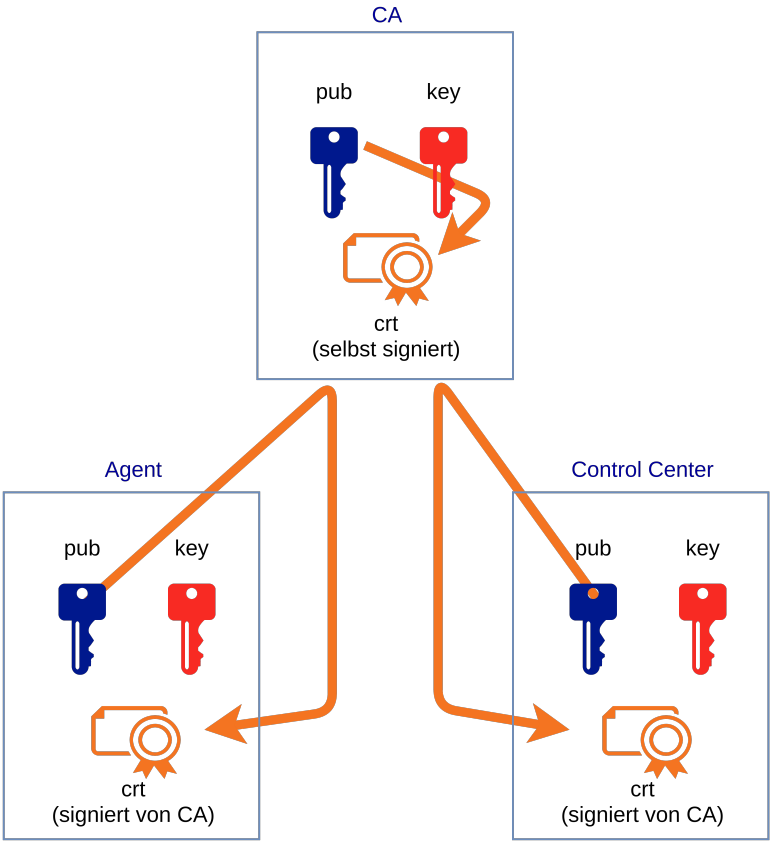


Abbildung 4.5.: Erzeugen von Zertifikaten mittels Signieren

[10] Openssl-Webseite: www.openssl.org/
[11] easy-rsa-Webseite: openvpn.net/easyrsa.html

Sichere Verbindung

Die Teilnehmer sind im Besitz des CA-Zertifikats, welches als Vertrauensanker dient und können so die Zertifikate ihrer Kommunikationspartner überprüfen um eine sichere Verbindung gewährleisten.

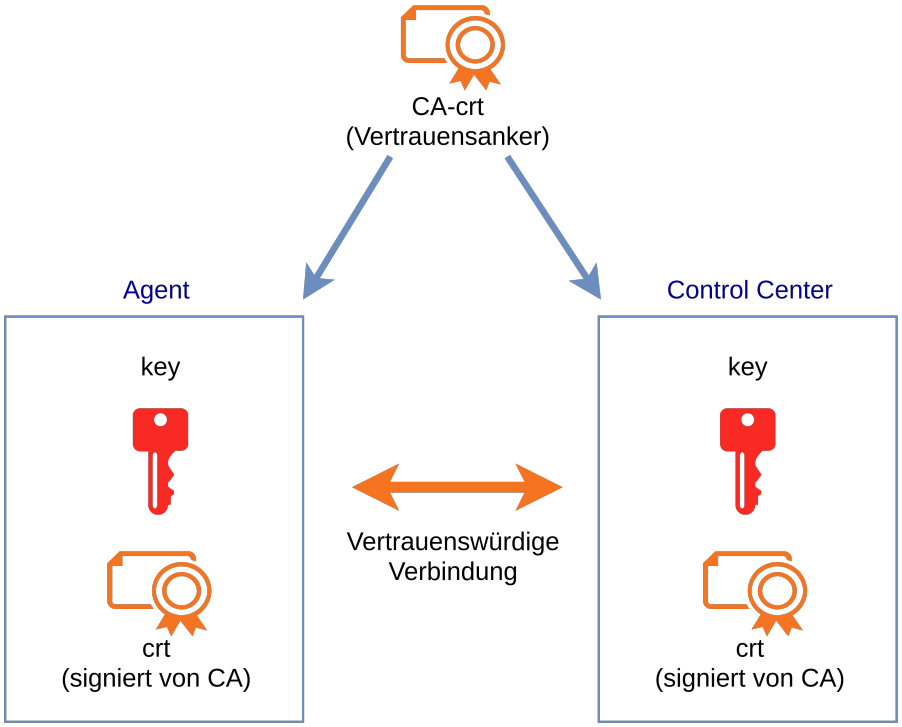


Abbildung 4.6.: Vertrauenswürdige Verbindung aufgrund Vertrauensanker

4.3. Design und Implementation

Vor dem Umsetzen dieser Arbeit war der Vorgang zum Aktualisieren der Systeme sehr konsolenlastig. Daher war ein Ziel der Arbeit, die Applikation benutzerfreundlich und intuitiv zu gestalten. Dies wurde mit einer engen Zusammenarbeit mit den Industriepartnern sowie mit eigenen Tests und Versuchen erzielt.

Mockups

Als Diskussionsgrundlage und Vorlage für die Umsetzung wurden etwa 30 verschiedene Mockups mit Balsamiq[12] umgesetzt. Die Wichtigsten werden hier kurz porträtiert.

Gruppierungen

Gemäss Use-Cases 08 und 11 (siehe 3.2 und 3.2) können Systeme und Pakete einer (oder mehreren für Pakete) beliebigen Gruppe zugeordnet werden. In den Entwürfen zu den entsprechenden Ansichten wurde zusätzlich zu den umgesetzten Funktionalitäten noch eine automatische Zuweisungsregel konzeptioniert. Auf Bild 4.7 ist die Gruppe 'Test-Group' zu sehen, welcher alle neu registrierten Systeme mit 'test' im Namen automatisch zugewiesen werden. Dieses Feature wurde aber aufgrund von anderen Prioritäten nicht umgesetzt.

[12] balsamiq.com/products/mockups/

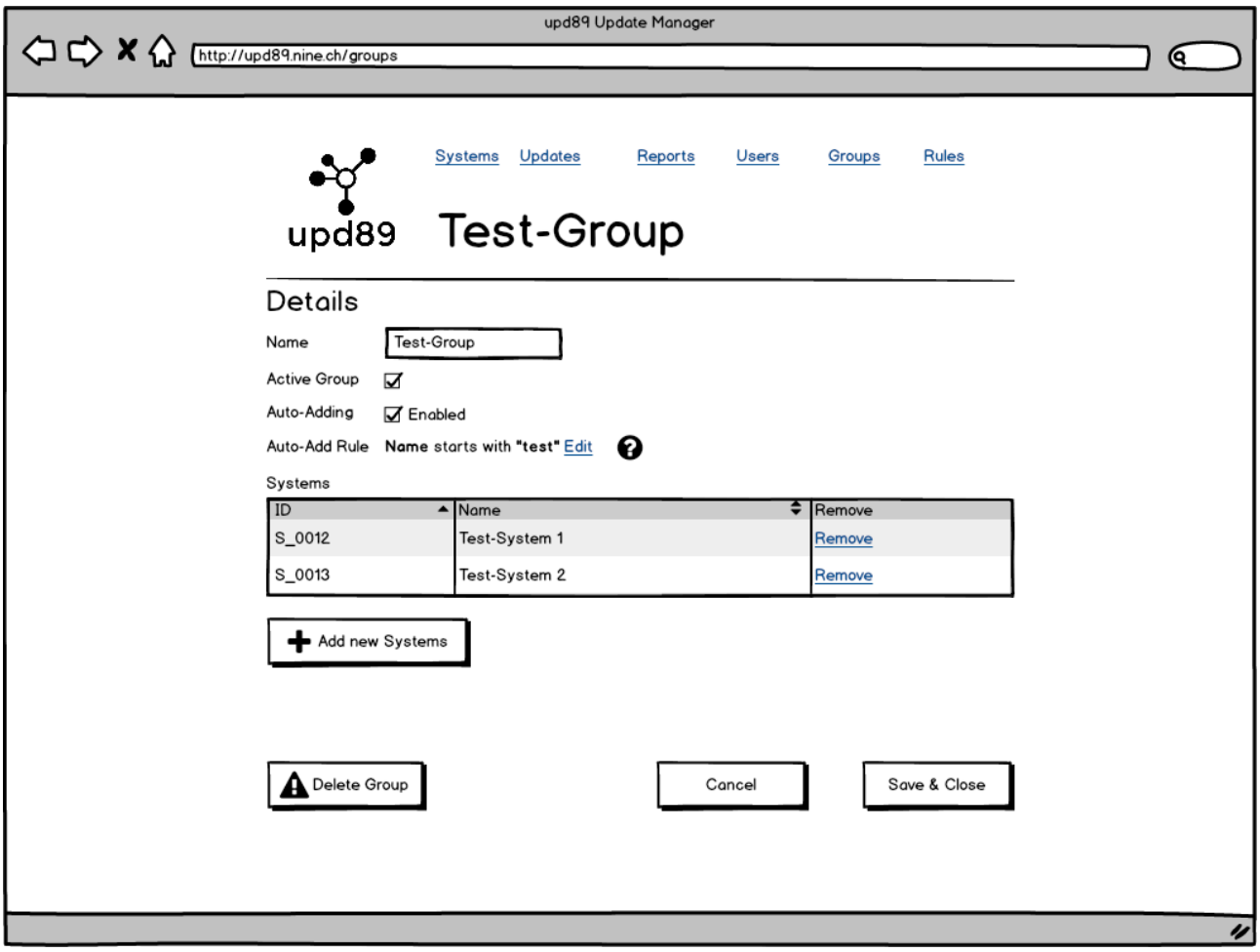


Abbildung 4.7.: Detail-Ansicht der Systemgruppe 'test-group'

Berechtigungen

Für den Use-Case 07 (siehe 3.2) wurden drei verschiedene Vorgehensweisen konzipiert. Die erste (Bild 4.8) sah vor, dass pro Benutzer die Einstellungen für jedes System, jede Art von Anwendungen und sogar unterscheidend zwischen grossen oder kleinen Versionssprüngen vorgenommen werden kann. Dies wurde aber verworfen, da der Administrationsaufwand bei Änderungen zu gross gewesen wäre. Zudem kann nicht garantiert werden, dass 'grosse' Versionssprünge korrekt erkannt werden.
Eine Alternative sah weiterhin die Verwaltung der Rechte pro Benutzer vor, aber in einer tabellarischen Form mit simplen Ja/Nein-Berechtigungen für verschiedene Aktionen (Bild 4.9). Hier wurde auch zwischen 'normalen' und 'kritischen' Updates unterschieden, also nur zwischen zwei Paketgruppen. Der Vorteil war die einfache Bedienung und die klare Darstellung der Berechtigungen. Aufgrund der gewünschten beliebigen Anzahl Paketgruppen wurde diese Alternative aber ebenfalls verworfen.
Schlussendlich hat sich eine Rollen- und Stufen-basierte Rechteverwaltung durchgesetzt. Jeder Benutzer wird einer Rolle zugeteilt, welche eine bestimmte Berechtigungsstufe besitzt. Ob ein Benutzer nun ein Paket auf einem System aktualisieren darf, hängt von der Gruppenzugehörigkeit des Paketes und des Systems ab.
Auf diese Weise ist es sehr einfach, ganzen Benutzergruppen die Rechte für mehrere Systeme oder

Pakete auf einmal zu entziehen oder zu erteilen, was bei den alternativen Vorschlägen mindestens eine Einstellung pro User erfordert hätte.

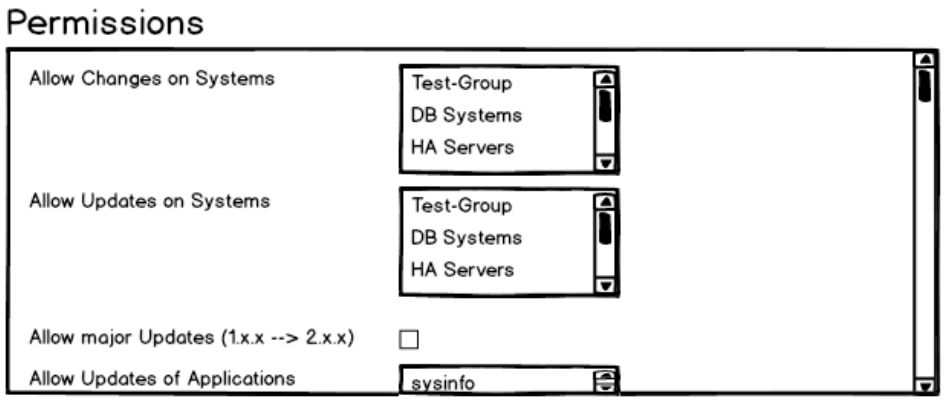


Abbildung 4.8.: Verworfenes Rechtesystem mit verschiedenen Einstellungen

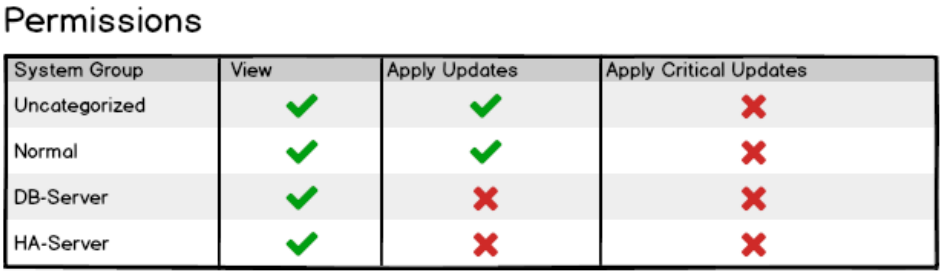


Abbildung 4.9.: Verworfenes Rechtesystem als Tabelle mit Icons

Dashboard

Eine Art Übersichtsseite war von Beginn an geplant und auch durch den Industriepartner begrüsst. Bei einer grösseren Anzahl von Systemen ist es oft einfacher, eine vereinfachte Zusammenfassung als eine detaillierte Auflistung aller einzelnen Systeme zu lesen. Dies, sowie relevante Aktionen bei Auftreten von bestimmten Situationen anzubieten, war das Ziel des Dashboards (Abbildung 4.10).

Einerseits sollte sichtbar sein, wieviele Systeme verfügbar sind, auf welchen Systemen Updates anstehen oder wo gerade Installationen durchgeführt werden. Andererseits sollten auch die letzten Aktivitäten und deren Ergebnis sichtbar sein, um Probleme und deren potentiellen Ursachen schnell erkennen zu können.

Da die Funktionalität zur automatischen Zuweisung von neu registrierten Systemen und noch unbekannten Paketen in Gruppen verworfen wurde, musste dies nun manuell erledigt werden. Dazu wurde eine Auflistung dieser Systeme und Pakete auf dem Dashboard konzipiert, wo sie auch gleich in Gruppen eingeteilt werden können.

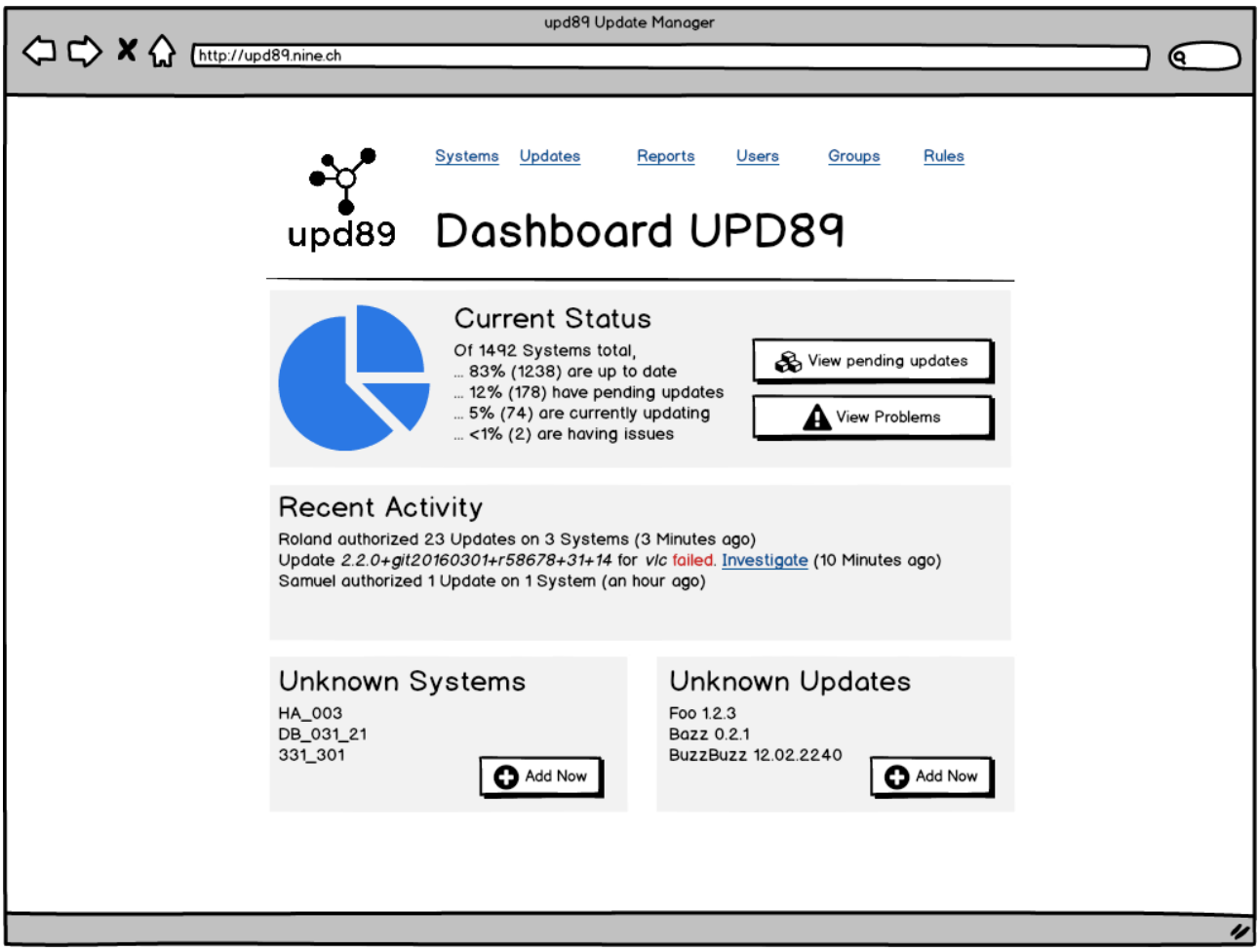


Abbildung 4.10.: Konzept des Dashboards

System- und Paket-Ansicht

Neben den spezifischen Ansichten für alle Pakete oder Systeme mit entsprechenden Filtern, von denen aus Updatevorgänge gestartet werden können, war es ein Wunsch des Industriepartners, auch eine kombinierte Ansicht mit Filterung und Suche nach Paketen und Systemen zu haben. Nach mehreren Entwurfsiterationen wurde schliesslich Bild 4.11 als passendes Design gefunden. Hier ist es möglich, mehrere Pakete gleichzeitig auf mehreren, einzeln auswählbaren Systemen in Auftrag zu geben.

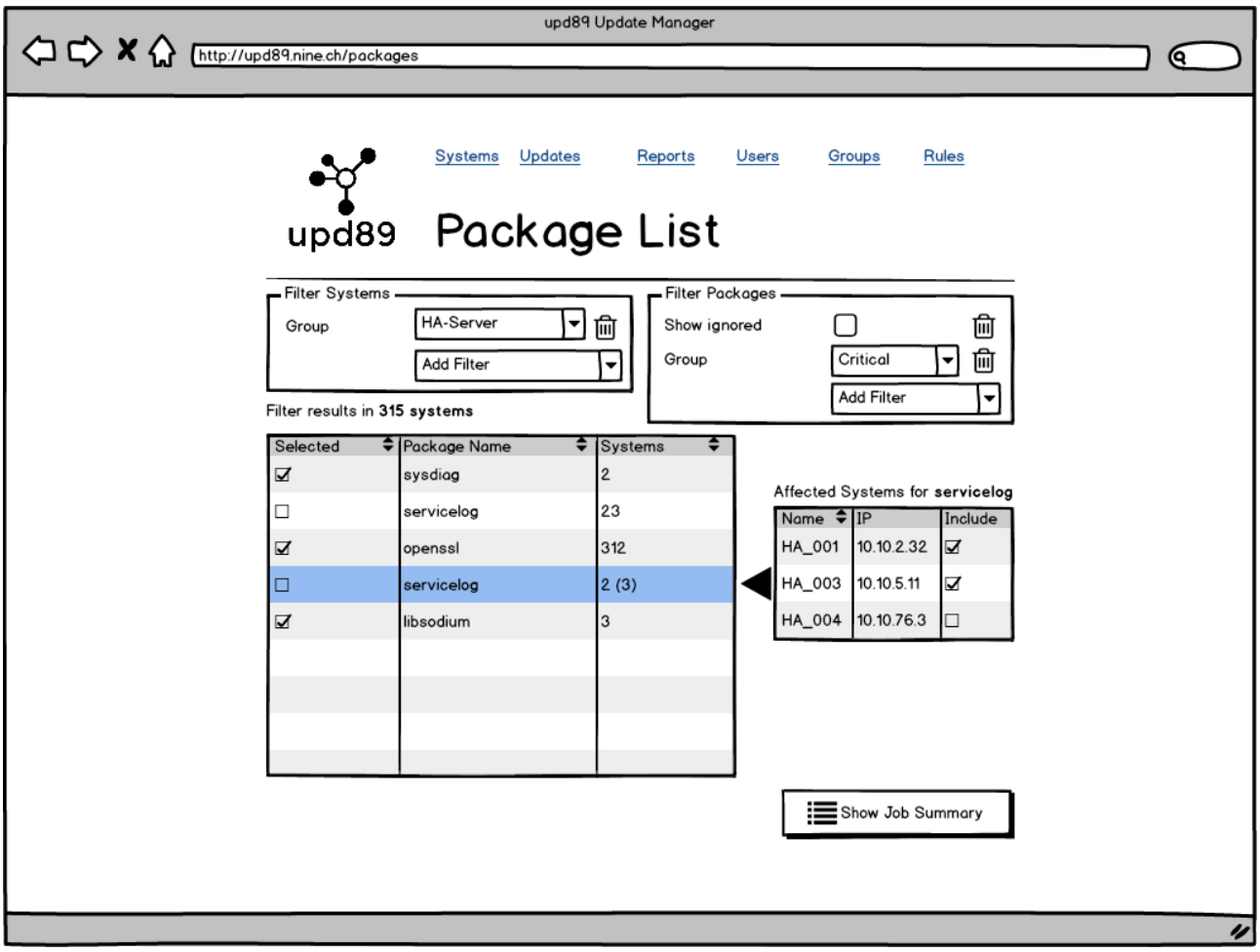


Abbildung 4.11.: Haupt-Ansicht mit Paket- und System-Filter

User Interface

Die Oberflächen halten sich grösstenteils an die Mockups aus der Planung, einige Details wurden aber aufgrund von Input durch den Industriepartner oder durch Hallway-Testing und Usability-Tests abgeändert. Siehe dazu auch das Kapitel Testing (4.4).

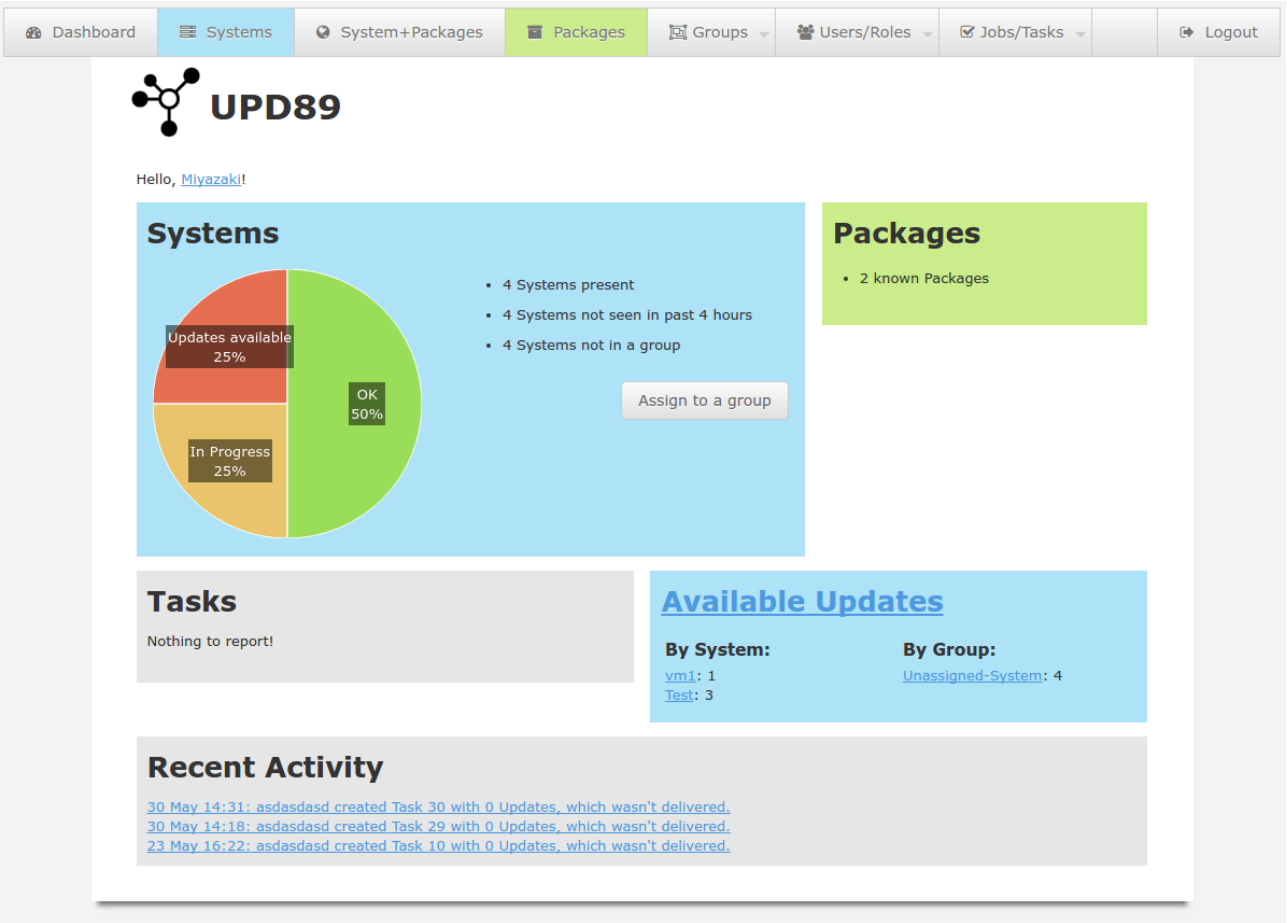


Abbildung 4.12.: Umgesetztes Dashboard

Javascript

Obwohl das Control Center in Ruby on Rails umgesetzt wurde, spielte Javascript eine wichtige Rolle. Ohne Javascript verursacht fast jede Aktion durch den Benutzer und jede Änderung am Inhalt ein Nachladen der Seite, was je nach Inhalt und Verbindungsqualität langsam und ablenkend sein kann.

Mit Javascript können Interaktionen des Benutzers bereits im Browser abgefangen werden und es kann darauf reagiert werden ohne den Umweg über den Server zu machen. Bei Bedarf kann der Javascript-Code aber auch Teilinhalte nachladen, ohne dass die momentane Seite verlassen werden muss. In der Paket- und System-Ansicht (Bild 4.13) beispielsweise wird über AJAX beim Filtern von Paketen und Systemen oder auf die entsprechenden Gruppen oder beim Sortieren der Tabellen jeweils nur der Inhalt der Tabelle neu geladen. Ein Klick auf den Pfeil neben einem Paket lädt ebenfalls via AJAX die betroffenen Systeme für die System-Tabelle auf der rechten Seite nach. Zudem wird die Schaltfläche zum Ausführen der Updates dynamisch aktiviert oder deaktiviert, je nach Auswahl von Paketen und Systemen.

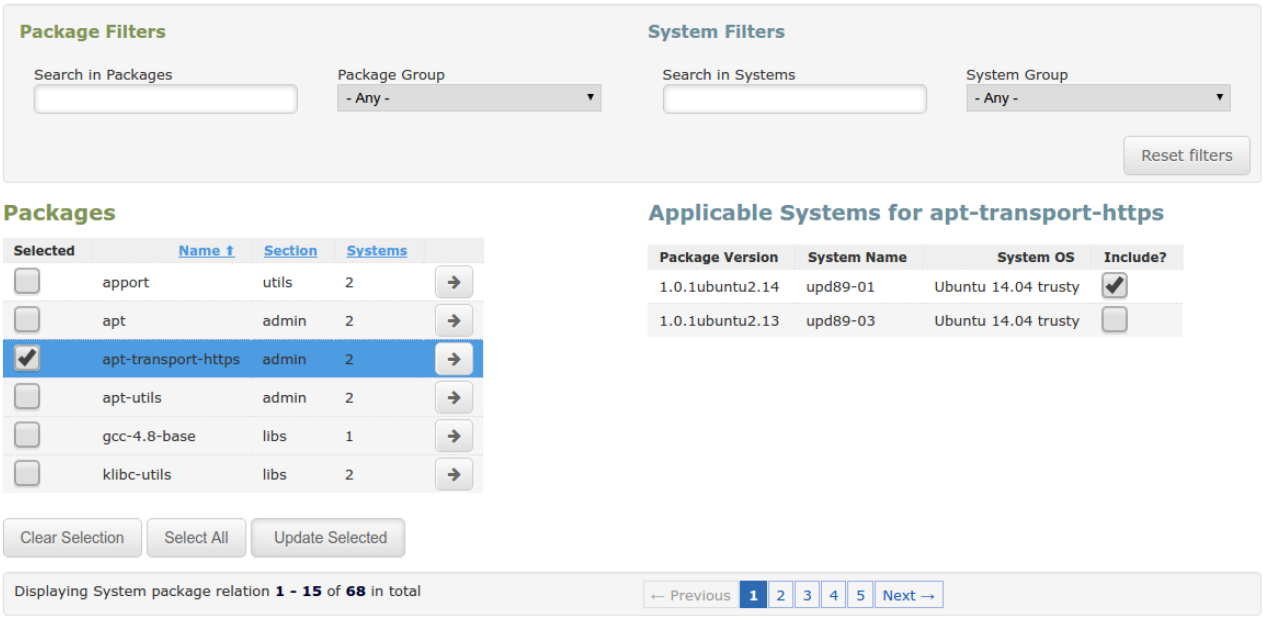


Abbildung 4.13.: Beispiel für Javascript-Erweiterungen des Rails-Frontends

Logo, Icon, Schriftart

Das Logo wurde im Squarespace Logo Generator[13] erstellt mit dem Icon 'Nodes'[14] von Gregor Črešnar. Die Schriftart für 'UPD89' ist Montserrat[15] von Julieta Ulanovsky.
Das Icon wurde als Symbol für die zentrale Verwaltung und die Kommunikation mit den einzelnen Servern gewählt. Es kann bei Bedarf durch ein beliebiges Logo ersetzt werden.

Farben

Generell wurde das Interface eher schlicht gehalten. Grautöne sowie Rot, Gelb und Grün als Signalfarben (4.14) lassen die Seiten übersichtlich und nicht überladen erscheinen.

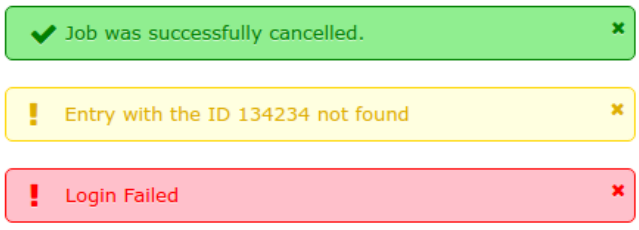


Abbildung 4.14.: Beispiele der drei Hinweise 'Erfolg', 'Warnung', 'Error'

Die selbst definierten Farben wurden in einer separaten SCSS-Datei definiert, wodurch es für

[13] www.squarespace.com/logo/
[14] thenounproject.com/search/?q=nodes+by+gregor&i=138240
[15] www.fontsquirrel.com/fonts/montserrat

anwendungs- oder benutzerspezifische Anpassungen einfach abzuändern ist. Eine Übersicht der wichtigsten Farben ist in Bild 4.15 zu sehen.

Abbildung 4.15.: Verwendete Farben mit Hex-Codes

Ein besonderes Augenmerk wurde auf die farbliche Kennzeichnung der Systeme und Pakete gelegt. Besonders in der kombinierten Ansicht, wo es Filtermöglichkeiten zu Systemen und Paketen gibt, ist es mit einer farblichen Unterscheidung einfacher zu sehen, was welche Entitätsgruppe betrifft. Diese Farben wurden auch ins Menu und ins Dashboard weitergezogen; in der ganzen Applikation sind Infos und Einstellungen, welche für Systeme relevant sind, generell im 'System-Blau' und solche für Pakete im 'Paket-Grün' (Beispiel im Bild 4.16).

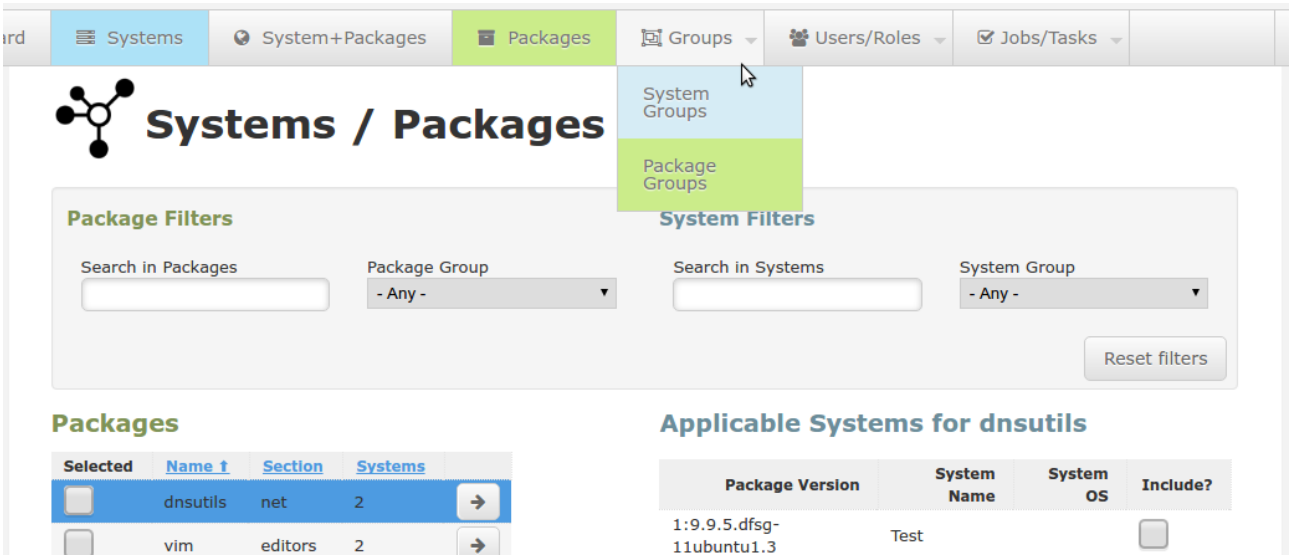


Abbildung 4.16.: Farb-Hinweise: Blau für Systeme, Grün für Pakete

Building Blocks

Um verschiedene grafische Elemente konsistent gleich aussehen zu lassen, wurde das HTML Kickstart-Framework von Joshua Gatcke/99lime.com[16] verwendet. Dadurch waren gewisse Funktionalitäten wie etwa das Menu oder die Hinweise bereits vorhanden.
Für die Icons wurde FontAwesome[17] verwendet.

Regelwerk

Entscheid 4.4: Regelwerk

Die Anforderung bezüglich einem Regelwerk aus der Aufgabenstellung wurde während des Projekts in Absprache mit den Projektpartnern gestrichen. Die Funktionalität wurde als zu komplex erachtet und andere Features wurden höher priorisiert. Jedoch wurden Vorschläge und Mockups erstellt, wie in Kapitel 6.7 ersichtlich ist.

[16] www.99lime.com/elements/
[17] fontawesome.io/

4.4. Testing

Unit-Tests

Für Unit-Tests wurde im Control Center RSpec[18] benutzt. Die Tests wurden lokal manuell sowie durch Travis-CI (siehe auch Kapitel 7.4) automatisch nach einem Commit ausgeführt.
Die Unit-Tests decken vor allem die API, aber auch Modelle und Controllers ab. Dazu wurden folgende Hilfs-Bibliotheken verwendet:

rspec-rails	Erweitert das bei Rails enthaltene Testing-Framework, so dass RSpec-Tests ausgeführt werden können[19]
factory_girl_rails	FactoryGirl vereinfacht das Erstellen von Fixtures[20]
faker	Liefert bei Bedarf realistische Test-Daten wie z.B. Namen, Email-Adressen, IP-Adressen, etc.[21]
guard-rspec	Führt Unit-Tests automatisch wieder aus, wenn eine Änderung daran festgestellt wurde[22]

Die Benutzerauthentisierung bereitete häufig Probleme, da die Dokumentation des Gems nicht auf dem neuesten Stand war.

System-Tests

Gemäss Projektplan (siehe Anhang D) wurden verschiedene Meilensteine definiert. Die Meilensteine ab der Construction-Phase umfassten auch das Erfüllen von Use-Cases. Nach dem Erreichen dieser Meilensteine wurde jeweils ein manueller System-Test durchgeführt und die Use-Cases getestet (Resultate in Tabelle 4.1).

[18] rspec.info/
[19] github.com/rspec/rspec-rails
[20] github.com/thoughtbot/factory_girl_rails
[21] github.com/stympy/faker
[22] github.com/guard/guard-rspec

Tabelle 4.1.: System-Tests

MS4	MS5	MS6	MS7	MS8	MS9	Use-Case
					✓	UC01: Agent Deployen
			✓	✓	✓	UC02: Bei CC registrieren
✓	✓	✓	✓	✓	✓	UC03: Ausstehende Updates melden
		✓	✓	✓	✓	UC04: Update durchführen
		✓	✓	✓	✓	UC05: Updates auslösen
				✓	✓	UC06: Reports einsehen
			✓	✓	✓	UC07: Berechtigungen verwalten
				✓	✓	UC08: Systemgruppen verwalten
			✓	✓	✓	UC09: User verwalten
				✓	✓	UC10: Regeln verwalten
		✓	✓	✓	✓	UC11: Paketgruppen verwalten

Usability-Tests

In unregelmässigen Abständen wurden Hallway-Tests durchgeführt. Dazu werden verfügbare, zufällige Personen in Reichweite dazu benutzt, neue Features und Interfaces zu testen. Es reichen bereits wenige Testpersonen, um den Grossteil der Usability-Probleme zu finden [1].
Im der Construction-Phase wurden zusätzlich noch Usability-Tests vorgenommen, wobei Personen mit mindestens durchschnittlichem Domänenwissen (über Systemadministration und Linux-Systeme) ausgewählt wurden.
Die Ergebnisse der Tests befinden sich im Anhang (Kapitel I).
Zusätzlich wurde nach jedem Sprint eine Demonstration für die Projektpartner durchgeführt. Hier wurden ebenfalls wertvolle Rückmeldungen über die Bedienbarkeit der Applikation gegeben.

4.5. Dokumentation

Entscheid 4.5: Dokumentationssprache

Es wurde entschieden, dass die Projektdokumentation - etwa dieses Dokument - auf Deutsch verfasst wird, die Software selbst, sowie Anleitungen dazu jedoch auf Englisch veröffentlich werden. Dies aus dem Grund, da Englisch die meistbenutzte Sprache im Internet ist [2] und aus eigener Erfahrung die meisten Open-Source-Projekte mindestens auf Englisch dokumentiert sind.

Entscheid 4.6: Dokumentations-Technologie

Die Dokumentation wurde in Latex verfasst. Dies wurde primär aufgrund der vorherigen Erfahrung und der besseren Möglichkeiten zur Gestaltung entschieden. Beide Teammitglieder hatten bereits Erfahrungen mit Latex gesammelt und vorherige Arbeiten damit erstellt.

5. Ergebnis

Im Rahmen der Bachelorarbeit wurde eine Softwarelösung mit zentraler Web-Komponente und verteilten Agents entwickelt. Die Web-Applikation kann auf den gängigen Browsern gut dargestellt werden[1] und kann unter normalen Bedingungen mit den Agent kommunizieren.
Die Anforderungen aus dem Projektauftrag (siehe Kapitel 2.1 bezüglich Nummerierung) wurden umgesetzt:

1. Die Applikation kann über den Web-Browser erreicht werden.
2. Anstehende Updates können auf den verbundenen Systemen eingesehen werden.
3. Es kann pro System eine Übersicht der ausgeführten Tasks eingesehen werden, worin die durchgeführten Updates enthalten sind.
4. Pro System läuft ein Agent, welcher mit dem Control Center kommuniziert.
5. Die Kommunikation mit dem Control Center verläuft verschlüsselt bei richtiger Konfiguration.
6. Meldungen von apt werden weitergereicht und bei Fehlschlagen eines Updates wird dieses besonders gekennzeichnet. Wird ein Neustart benötigt, wird das dem Control Center ebenfalls mitgeteilt.
7. Die kürzlich durchgeführten Aktivitäten aller Benutzer ist einfach einsehbar und es kann pro Job ein Kommentar vom Auslöser hinzugefügt werden.
9. Die Bedienbarkeit wurde durch Usability-Tests bestätigt.
10. Die Software wurde als Open-Source-Lösung mit der MIT-Lizenz (siehe auch Anhang B) entwickelt.
11. Sie unterstützt mindestens die Ubuntu-Versionen 12.04, 14.04 und 16.04.
12. Als Programmiersprachen wurden hauptsächlich Ruby und Python verwendet.
13. Als Datenbank kommt PostgreSQL zum Einsatz.
14. Es wurde ein Python-Paket[2] erstellt, welches auch mit Puppet verteilt werden kann.

Punkt 8 konnte nicht im vorgesehenen Detailgrad erarbeitet werden. Die durchgeführten Tests zeigten keine Probleme bezüglich der Zuverlässigkeit. Insbesondere das Verhalten bei vielen parallelen Zugriffen wurde nicht spezifisch getestet. Auf der Seite des Control Centers kann mit der maximaler Anzahl Threads in Passenger das Problem entschärft werden.
Der optionale Punkt 15 wurde nicht erreicht. Systeme und Pakete müssen manuell an Gruppen zugewiesen werden. Es wurden jedoch Vorschläge und Mockups dazu erstellt, siehe Kapitel Ausblick (6.4).

[1] Getestet auf Firefox 47 und Chrome 51
[2] Siehe Kapitel 4.1

5.1. Schlussfolgerung

UPD89 ist grafisch nicht so detailreich ausgearbeitet wie kommerzielle Produkte wie etwa Landscape und besitzt nicht so viele Features wie grössere Open-Source-Produkte wie Spacewalk. Im Vergleich mit anderen verfügbaren Produkten konzentriert sich die Applikation auf die Kernfunktion, das Aktualisieren von Ubuntu-Systemen. Dies wurde erreicht und funktioniert stabil.
Die Vorgabe, dass Rails benutzt werden sollte, war einerseits ein kleiner Nachteil, da praktisch noch kein Know-How im Team vorhanden war; andererseits war es auch ein Vorteil, da Rails bei der Entwicklung viel Arbeit abnahm. Bei einer Durchführung ohne Rails hätte ein anderes Web-Applikations-Framework verwendet werden müssen, wo das Know-How ebenfalls nicht vorhanden gewesen wäre.
Dass einige Features nicht umgesetzt werden konnten (beispielsweise das Regelwerk und weitere Features im Kapitel Ausblick) ist schade, trotzdem konnten alle primären Anforderungen erfüllt werden. Auch dass die Projektpartner interessiert am Projektverlauf waren und eigene Ideen einbrachten, zeugt von einer prinzipiell guten Idee. Ob das Projekt von der Open-Source-Gemeinde angenommen und benutzt wird, bleibt abzuwarten.

5.2. Softwaredokumentation

In diesem Kapitel wird kurz beschrieben, wie die Software benutzt werden soll. Im Anhang (Kapitel G) ist die komplette Anleitung zu finden, so wie sie auf Github hinterlegt ist. Da die öffentlichen Dokumente in Englisch verfasst wurden (Siehe auch 4.5), ist dort auch die Softwaredokumentation auf Englisch.

CA

Für die Absicherung der Kommunikation zwischen Control Center und Agent werden Zertifikate verwendet, die von einer eigenen CA signiert sind (vgl. Kapitel 4.2). Die Generierung dieser Zertifikate ist mit Hilfe von Easy-Rsa [3] in wenigen Minuten erledigt. Folgende Schritte sind nötig dazu:

- Installation von Easy-RSA
- Anpassen der Konfiguration
- Generieren der Zertifikate

Folgende Befehle installieren Easy-RSA, erstellen ein Verzeichnis für die CA und ändern die Einstellungen, um Zertifikate mit einer Länge von 4096 Bit zu generieren.

Programmcode 5.1: Setup Easy-RSA

```
sudo apt install easy-rsa
make-cadir ca ; cd ca
sed -i "s/extendedKeyUsage.*/extendedKeyUsage=serverAuth,clientAuth/g" openssl*.cnf
sed -i "s/export KEY_SIZE=.*/export KEY_SIZE=4096/g" vars
nano vars
```

[3] openvpn.net/easyrsa.html

Die Anpassung der folgenden Werte sorgt dafür, dass die Zertifikate mit sinnvollen Informationen versehen werden.

Programmcode 5.2: CA Konfiguration

```
export KEY_COUNTRY="US"
export KEY_PROVINCE="CA"
export KEY_CITY="SanFrancisco"
export KEY_ORG="Fort-Funston"
export KEY_EMAIL="me@myhost.mydomain"
export KEY_OU="MyOrganizationalUnit"
```

Ist die Konfiguration abgeschlossen, werden die Zertifikate mit folgenden Befehlen erstellt. Dabei ist darauf zu achten, dass die korrekten Hostnamen für das Control Center und für die Agents eingesetzt werden.

Programmcode 5.3: Generieren der Zertifikate

```
./clean-all
./pkitool --initca
./pkitool --server controlcenter.your-domain.org
./pkitool agent1.your-domain.org
./pkitool agent2.your-domain.org
./pkitool agent3.your-domaon.org
```

Control Center

Vorbedingungen

Es gelten folgende Voraussetzungen, damit ein reibungsloser Betrieb der Software sichergestellt werden kann:

- Git muss installiert sein
- Eine CA muss aufgesetzt sein
- Passenger 5.0.27 / Apache 2.4.7 (Ubuntu 14.04) müssen installiert und konfiguriert sein
- Ruby 2.2.3 und Rails 4.2.4 müssen installiert sein
- Postgresql-9.3 muss installiert und konfiguriert sein

Installation

Sind alle Voraussetzungen erfüllt, kann das Control Center mit folgenden Befehlen installiert werden:

Programmcode 5.4: Installation

```
git clone https://github.com/upd89/controlcenter.git
cd controlcenter
bundle install
rake db:create
```

```
rake db:migrate
rake db:base_data
```

Zu Testzwecken kann der Server lokal gestartet werden:

Programmcode 5.5: Startet einen lokalen Server

```
rails server
```

Agent

Der Agent kann auf verschiedene Arten installiert werden. Einerseits wie das Control Center durch das Klonen von GitHub:

Programmcode 5.6: Installation

```
apt install python-apt python-daemonize python-configparser
git clone https://github.com/upd89/agent.git
python setup.py build
python setup.py install
update-rc.d upd89 defaults
```

Andererseits aber auch durch pip:

Programmcode 5.7: Installation via pip

```
pip install upd89
update-rc.d upd89 defaults
```

Anpassungen können in `/etc/upd89/config.ini` vorgenommen werden.
Hinweis: Es wird eine direkte Verbindung vom Agent zum Control Center benötigt. Dies bedeutet, dass jeder Agent via HTTPS das Control Center erreichen können muss und umgekehrt.

6. Ausblick

Sowohl durch den Industriepartner als auch durch das Team selbst kamen verschiedene Ideen für weitergehende Features auf. Aus Zeitgründen konnten leider nicht alle Ideen umgesetzt werden.

6.1. 'Workflow'

Ein Wunsch der Industriepartner war es, den momentanen Arbeitsablauf abbilden zu können. Ein Beispiel ist, dass zuerst die 'unkritischen' Pakete auf den 'unkritischen' Systemen installiert werden müssen, bevor dort 'kritische' Pakete installiert werden können.

Dies wurde im 'Workflow'-Feature als Vorschlag präsentiert (Bild 6.1). Hier gibt es eine definierbare Reihenfolge von Arbeitsschritten, welche nacheinander durchgeführt werden müssen. Diese Arbeitsschritte könnten manuell festgelegt und bearbeitet sowie sortiert werden (Siehe Bild 6.2 für den Edit-Modus).

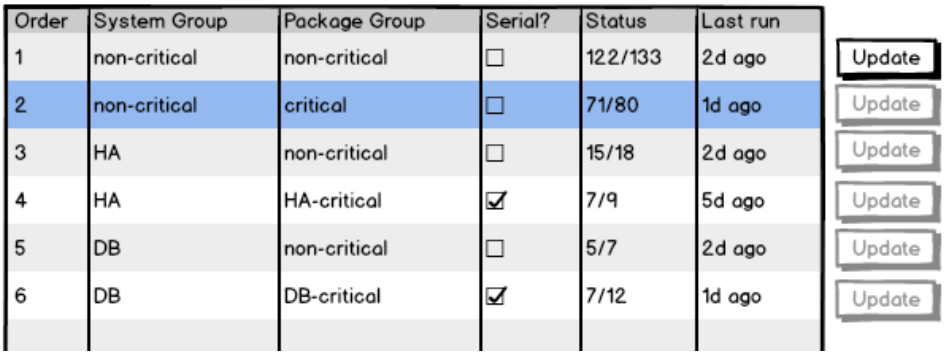


Abbildung 6.1.: Konzept Workflow-Feature

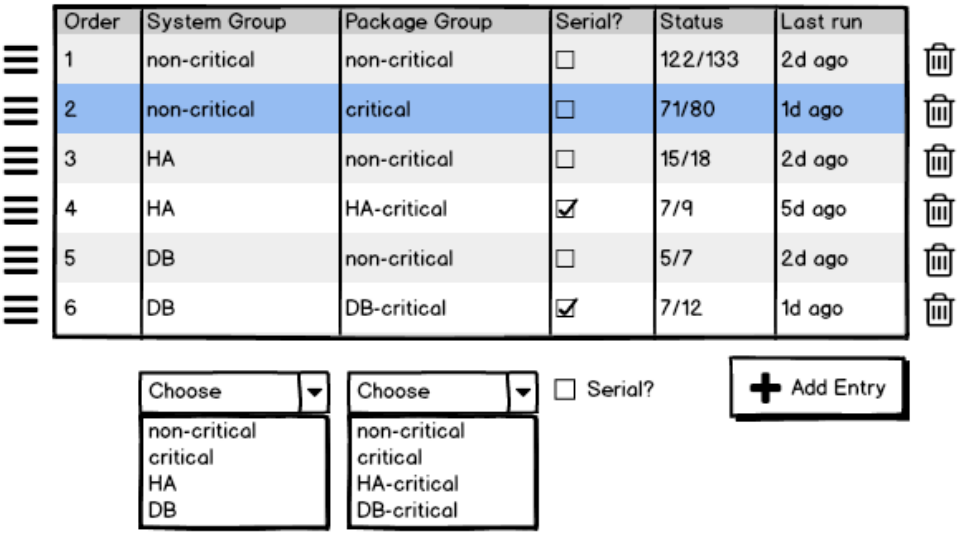


Abbildung 6.2.: Konzept Workflow-Feature bearbeiten

6.2. Dry-Run

In apt existiert das Feature 'dry-run', womit ein Updatevorgang simuliert[1] werden kann. Dies könnte auch als Feature im Control Center implementiert werden, indem vor dem Versenden eines Tasks zuerst eine Simulation durchgeführt werden könnte, um zu sehen, welche Pakete Abhängigkeiten nach sich ziehen oder fehlschlagen würden.

6.3. Message-Queue

Mehrmals kam das Thema einer Message Queue auf. Die umgesetzt Lösung lässt das Control Center mit den Agents einzeln kommunizieren. Ein Nachteil dieses Vorgehens ist, dass jeder Agent für sich dafür sorgen muss, dass die Informationen beim Control Center ankommen. Auf der anderen Seite muss das Control Center dafür sorgen, dass genügend Threads vorhanden sind, die eingehende Verbindungen von den Agents verarbeiten.
Mit einer Message Queue könnte das Control Center die Informationen nacheinander aus der Message Queue abholen, wodurch nur ein Thread nötig wäre. Bei einem Ausfall des Control Centers würden die Nachrichten zwischengespeichert, bis das Control Center wieder online ist.
Die Nachteile einer Message Queue sollten aber nicht unterschätzt werden. Jeder Agent würde eine eigene Queue benötigen, welche die Aufträge für ihn beinhaltet. Jeder Agent wäre ohne Unterbruch mit der Queue verbunden, im Fall von nine.ch also 1500 Stück. Ausserdem bringt dieses Messaging-System eine neue Abhängigkeit mit sich.

[1] Ohne Updates zu installieren, siehe auch linux.die.net/man/8/apt-get

6.4. Automatische Gruppen-Zuweisung

Ein optionaler Punkt aus der Aufgabenstellung (Kapitel 2.1) waren Vorschläge der Applikation für System oder Pakate bezüglich der Gruppenzuteilung. Aufgrund von einfachen Regeln wie z.B. dem Namen oder einer IP-Range bei Systemen oder der 'Section'[2] bei Paketen könnten Vorschläge gemacht werden, zu welchen Gruppen etwas gehört.
In den Mockups wurde ein solches Feature bereits in groben Zügen angedacht (siehe Abbildung 6.3). Pro System könnte die Auto-Zuweisung auch separat aktiviert und konfiguriert werden.

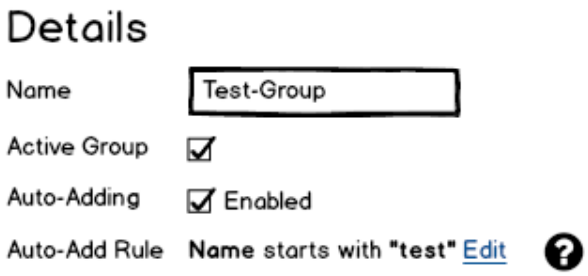


Abbildung 6.3.: Automatische Gruppenzuweisung für Systeme (Detail)

6.5. Einfachere Registrierung

Bei der aktuellen Lösung muss eine eigene CA eingerichtet werden und das signierte Zertifikat beim Deployment vom Agents mit ausgeliefert werden (siehe auch Kapitel 4.2). Einfacher wäre es, wenn der Agent beim ersten Kontakt mit dem Control Center sich sein Zertifikat signieren lässt. Inspiriert ist der Ansatz durch Puppet, bei welchem sich die Agents beim puppet-master ein Zertifikat ausstellen lassen, siehe [3], [4] und [5].

6.6. Geplante Tasks

Ein zu Beginn vorgesehenes Feature war die Möglichkeit, Tasks auf einen gewissen Zeitpunkt hin zu planen. Dieses wurde aber aufgrund von anderen Prioritäten verworfen. Der Benutzer kann ein Datum und eine Uhrzeit eingeben, welche dann mit dem Task an den Agent übermittelt wird. Dieser plant die Ausführung selbst ein und meldet sich nach Abschluss des Tasks wieder beim Control Center.
Dies hätte auch die Möglichkeit ergeben, einen geplanten Task wieder abbrechen zu können, solange er noch nicht ausgeführt worden ist, womit die API auf Agent-Seite hätte erweitert werden müssen. Ein Grund gegen dieses Feature war der Wunsch des Industriepartners, dass alle Tasks manuell durch einen Menschen ausgelöst werden müssen. Dies reduziert auch potentielle Probleme mit der Applikation, da Fehler zeitnah und nicht erst bei der Ausführung des Tasks auftauchen würden.

[2] Datenfeld bei apt-Paketen
[3] www.masterzen.fr/2010/11/14/puppet-ssl-explained/
[4] docs.puppet.com/background/ssl/
[5] docs.puppet.com/puppet/4.5/reference/ssl_autosign.html

6.7. Regelwerk

Ein Feature, welches oft diskutiert aber schlussendlich nicht umgesetzt wurde, ist das Regelwerk. Dahinter steckt ein Mechanismus, welcher Abhängigkeiten innerhalb von verschiedenen System- und Paketgruppen generiert und kontrolliert. Ein Beispiel für eine solche Abhängigkeit ist etwa, dass nicht alle 'High-Availability'-Systeme auf einmal aktualisiert werden dürfen, da so möglicherweise kurze Ausfälle entstehen könnten.
Dieses Feature wurde aber aufgrund von zu hoher Komplexität und fehlender Zeit verworfen. Ein Mockup ist in der Abbildung 6.4 zu sehen.

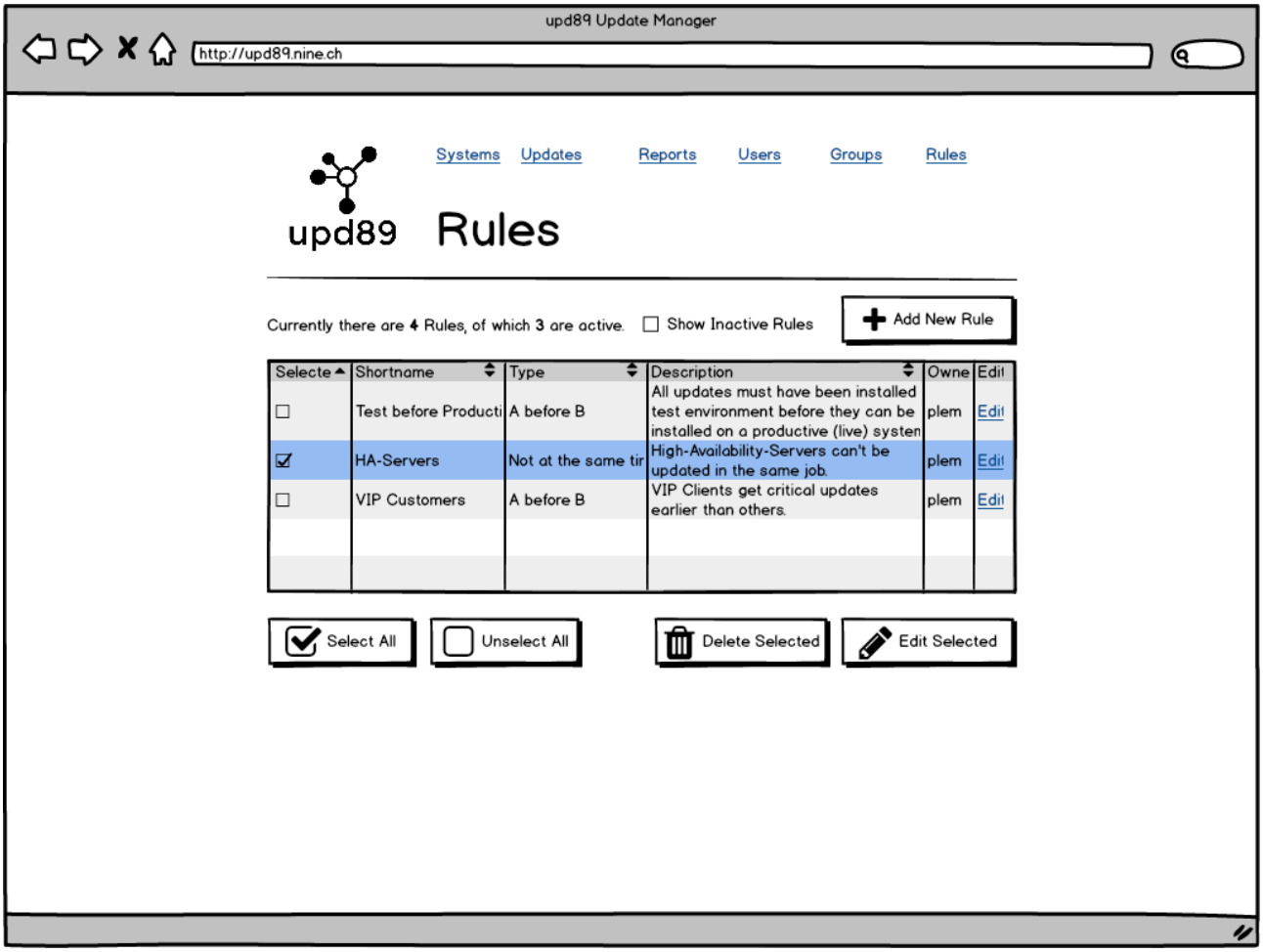


Abbildung 6.4.: Verworfenes Regelwerk-Feature

7. Projektmanagement

7.1. Vorgehensmodell

Das gesamte Projekt wurde im Stil von Rational Unified Process geführt und dementsprechend in die vier Phasen Inception, Elaboration, Construction und Transition aufgeteilt. Eine Übersicht über den Projektverlauf ist in der Abbildung 7.1 zu sehen.

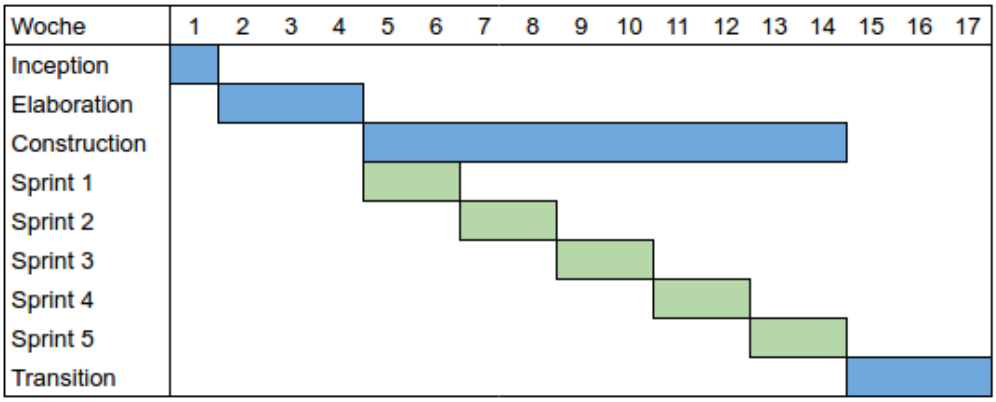


Abbildung 7.1.: Übersicht Projektverlauf

Die Inception-Phase diente dazu, den Auftrag zu definieren und sich gemeinsam auf klare Ziele und ein grobes Vorgehen zu einigen. Dazu kam die Installation der Infrastruktur.
In der Elaborations-Phase wurden weitere Informationen gesammelt. Ein Grossteil der Use-Cases und Arbeitspakete wurden definiert sowie das Domänenmodell erstellt. Zudem wurden die wichtigsten Risiken (Siehe auch 7.3) definiert. Hauptziel war es, die Construction-Phase zu planen.
In der Construction-Phase wurde nach agiler Vorgehensweise gearbeitet, so dass der Auftraggeber die Schwerpunkte setzen und priorisieren konnte (dazu mehr in Kapitel 7.1). Hier wurde die fertige Applikation entwickelt.
Die Transition-Phase diente der Übergabe und Auslieferung sowie der Fertigstellung der Dokumentation.

Sprints

Es wurden in der Construction-Phase in zweiwöchigen Sprints gearbeitet. Zu Beginn fand jeweils ein Sprintplanungsmeeting mit nine.ch statt, wo der Plan für den kommenden Sprint diskutiert wurde und gleichzeitig eine Retrospektive auf den vergangenen Sprint abgehalten wurde.
Im Anhang (Kapitel E) befinden sich die Auswertungen der einzelnen Sprints.

7.2. Rollen und Verantwortlichkeiten

An diesem Projekt arbeiten die beiden unten aufgeführten Informatik-Studenten. Aufgrund der entsprechenden Stärken und beruflichen Orientierungen übernimmt Philipp Christen die Verantwortung über das Frontend, Ueli Bosshard übernimmt die Verantwortung über das Backend und die Kommunikation mit apt.

	
Ueli Bosshard Stärken: Systemtechnik, OS, Netzwerk, DB	Philipp Christen Stärken: Web-Entwicklung, Frontend, UX, Mockups

7.3. Risiken

Um die potentiellen Risiken während des Projekts zu sammeln, wurde eine Risiko-Checkliste verwendet. Für jedes Risiko wurde ein zusätzlicher Aufwand in Stunden geschätzt, welcher anfallen würde wenn das Risiko eintritt. Multipliziert mit der geschätzten Eintrittswahrscheinlichkeit ergab dies einen gewichteten Mehraufwand. Dieser wurde in fünf Kategorien (sehr gering bis sehr schwer) eingestuft. Ziel war es, alle Risiken bis Ende Elaborations-Phase entweder komplett auszuschliessen oder soweit zu entschärfen, dass beim Eintreten Sicherheitsmassnahmen greifen und keine grosse Verzögerung oder anderer Schaden verursacht werden.
Es wurden nur für das Projekt relevante Risiken analysiert. Risiken wie 'Krankheit' wurden absichtlich nicht notiert, da es hier keine sinnvollen Präventionsmassnahmen gibt.

Tabelle 7.2.: Alle berücksichtigten Risiken

ID	Risiko	SP[1]	EW[2]	SG[3]	Gewichtung	Präventionsmassnahmen	Massnahme bei Eintritt
R01	Regelwerk zwischen Updates und Systemen zu komplex	40	25%	10	Sehr Schwer	Klare Anforderungen definieren, Erwartungen des Auftraggebers früh abholen.	Einschränken der Anforderungen.
R02	Komplexes Permission-System für Benutzer	24	25%	6	Schwer	In Elaborations-Phase auf Rechtesystem einigen, so dass Machbarkeit gewährleistet ist.	Anforderungen einschränken oder komplett weglassen.
R03	Ruby für Agent-Umsetzung oder Anschluss an apt nicht geeignet	16	25%	4	Mittel	So früh wie möglich Eignung von Ruby für Agent überprüfen.	Python benutzen, wo es ähnliche Projekte gibt.
R04	apt-Schnittstelle weniger umfangreich als erwartet	4	50%	2	Gering	Frühzeitiges Abklären des Umfangs der apt-Schnittstelle.	Selbst erstellte Schnittstelle einsetzen oder Aufgabe anpassen, falls nicht anders lösbar.
R05	Zertifikat-Handling ist komplexer als erwartet	8	10%	0.8	Sehr Gering	Machbarkeit früh abklären.	Spezialisten[4] anfragen.
R06	Control Center durch gleichzeitige Anfragen überfordert	16	25%	4	Gering	Auslastung einplanen, mit erstem stabilem Prototypen Last-Tests fahren.	Spezialisten[4] anfragen.
R07	Auftraggeber mit Umsetzung unzufrieden	24	25%	6	Schwer	Use-Cases, Scope und NFAs früh bestätigen lassen. Regelmässige Meetings mit gemeinsamer Planung.	Meeting mit Betreuer und Auftraggeber einberufen.

[1] Schadenspotenzial (h)
[2] Eintrittswahrscheinlichkeit (%)
[3] Schaden gewichtet (h)
[4] nine.ch oder Prof. Steffen (HSR)

Kritische Risiken

Die besonders kritischen Risiken werden hier kurz erläutert und genauer beschrieben.

Risiko	R01
Titel	Regelwerk zwischen Updates und Systemen zu komplex
Beschreibung	Es war von Beginn an nicht klar, wie das Regelwerk genau aussehen soll, welches die Kombinationen von Systemen und Updates einschränkt. Im schlimmsten Fall können es beliebig tiefe Abhängigkeiten sein, welche über mehrere Gruppen hinweg geprüft werden müssen.
Prävention/Massnahme	In der Elaborations-Phase sollte so gut wie möglich geklärt werden, in welcher Form das Regelwerk entstehen soll. Sobald sich herausstellt, dass die gewählte Tiefe der Umsetzung zu viel Aufwand verursacht, wird mit dem Auftraggeber entschieden, ob die Anforderung gegebenenfalls eingegrenzt werden kann.

Risiko	R02
Titel	Komplexes Permission-System für User
Beschreibung	Wie beim Regelwerk zwischen Updates und Systemen können die Berechtigungen der Benutzer beliebig fein konzeptioniert werden. Dadurch müssen nicht nur die Berechtigungen selbst geprüft werden, sondern auch Abhängigkeiten unter den Berechtigungen - es ist sinnlos, wenn man einem Benutzer das Updaten einer spezifischen Gruppe verbietet, aber das Erstellen und Bearbeiten einer Gruppe erlaubt ist.
Prävention/Massnahme	Wiederum soll in der Elaboration-Phase mit dem Auftraggeber das Rechtesystem vereinbart werden um die Machbarkeit zu gewährleisten. Falls dies nicht gelingt oder sich als schwerer als geplant entpuppt, sollen wenn möglich die Anforderungen eingeschränkt werden. Falls es keine brauchbare Lösung gibt, muss auf das Rechtesystem verzichtet werden.

Risiko	R07
Titel	Auftraggeber mit Umsetzung nicht zufrieden
Beschreibung	Der Auftraggeber kann trotz korrekter Umsetzung mit dem Endresultat nicht zufrieden sein, etwa indem implizite und nicht dokumentierte Annahmen von Kundenseite her nicht oder nur teils erfüllt wurden.

Prävention/Massnahme	Es soll früh abgegrenzt werden, was genau zum Projektumfang gehört, welche Use-Cases und NFAs gelten und welche Features implementiert werden. Dies soll vom Auftraggeber bestätigt werden. Regelmässige Meetings mit Statusbesprechung und Planung der nächsten Wochen sollen unterschiedliche Vorstellungen verhindern und alle Parteien am Entstehungsprozess beteiligen lassen. Falls trotzdem Unzufriedenheit auftreten sollte, muss dies zusammen mit dem Betreuer genauer angeschaut werden.

Risikoüberwachung

Die kritischen Risiken R01 und R02 konnten entschärft werden. Das Regelwerk (R01) wurde zusammen mit dem Auftraggeber als zu komplex angesehen und hätte den Rahmen des Projekts gesprengt. Die Abklärungen und Vorschläge wurden im Ausblick (Kapitel 6.7) erwähnt.
Das Permission-System für die Benutzer (R02) wurde vereinfacht und als Rechte-Skala (PermissionLevel, siehe auch Kapitel 3.4) definiert. Dies erfüllt die Anforderungen an das Feature, aber ist weniger komplex umzusetzen als alternative Lösungen (siehe dazu die Mockups in Kapitel 4.3).
R07 wurde ebenfalls im Laufe der Arbeit minimiert. nine.ch wurde mindestens alle zwei Wochen über neue Erkenntnisse und Entwicklungen informiert und Feedback wurde jeweils direkt diskutiert und in den nächsten Sprint übernommen.

7.4. Infrastruktur

Da es sich um ein Open-Source-Projekt handelt, wurde der Quellcode komplett veröffentlicht. Bereits während der Entwicklung war er jederzeit auf Github[1] einsehbar.
Entwickelt wurde primär mit VIM[2], Atom[3] oder RubyMine[4]. Bei jedem Commit nach GitHub checkte Travis-CI[5] den neuesten Stand aus und führte alle Unit-Tests durch. Das Resultat wurde einerseits via Email an die Entwickler geschickt, aber auch auf GitHub sowie auf RedMine[6] angezeigt. Eine Grafische Darstellung der Abläufe befindet sich in Abbildung 7.2.
Diagramme wurden mit Astah Professional[7] und Draw.io[8] erstellt. Bildbearbeitungen wurden mit Gimp[9] oder Pinta[10] vorgenommen. Für SVGs, wie zum Beispiel dem Poster, wurde Inkscape[11] verwendet.
Das Erstellen der Dokumentation fand in ShareLatex[12] statt. So konnte gleichzeitig am Dokument

[1] github.com/upd89/
[2] www.vim.org/
[3] atom.io
[4] www.jetbrains.com/ruby
[5] travis-ci.org/
[6] www.redmine.org/
[7] astah.net/editions/professional
[8] www.draw.io/
[9] www.gimp.org/
[10] pinta-project.com/pintaproject/pinta/
[11] inkscape.org/en/
[12] www.sharelatex.com/

gearbeitet werden und es benötigte keine eigene lokale Latex-Umgebung. In regelmässigen Abständen wurden die Tex-Files auf das Dokumentations-Repo auf Github[13] committet und per Webhook wurde ein PDF-File für die Ablage automatisch generiert.

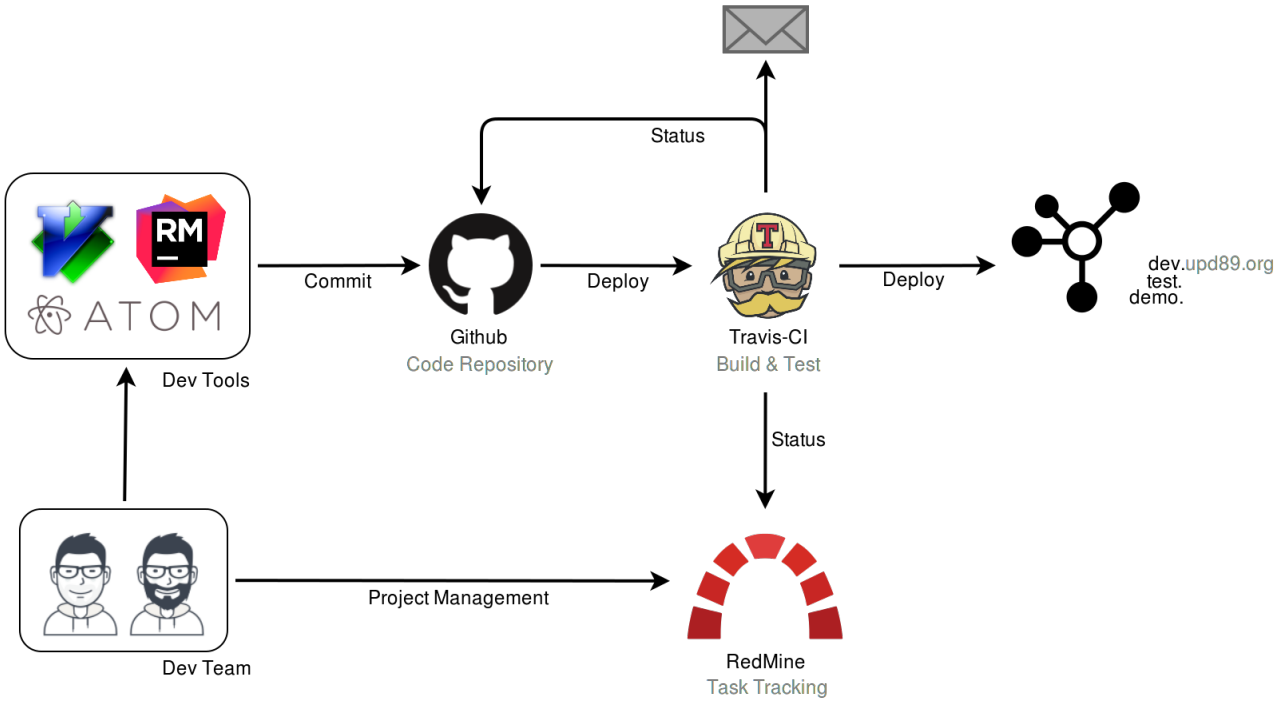


Abbildung 7.2.: Entwicklungsumgebung

Zu Testzwecken wurden von nine.ch 10 Server gesponsert, welche in 3 Gruppen eingeteilt wurden: Development, Test, Demo. Jede Gruppe enthält ein Control-Center sowie 2-3 Agent-Systeme. Die Dev-Systeme dienten dem aktiven Entwickeln des Frontends sowie zum Testen von neuen Features. Auf den Test-Systemen wurden nur stabile Versionen des Control-Centers installiert und dienten mehrheitlich dem Testen der Kommunikation mit den Agents. Auf den Systemen der Gruppe Demo wurden nur getestete Versionen aufgespielt. Diese dienten als Demonstrationszweck für die Industriepartner und Betreuer.
Es wurden sowohl die eigenen Notebooks als auch die von der HSR zur Verfügung gestellten Arbeitsplätze benutzt. Daneben waren zwei VMs der HSR im Einsatz; eine für Redmine und eine für Entwicklungen und Testing der Software. Der Source Code wurde über Github verwaltet. Für Ruby und Rails wurde angestrebt, eine möglichst aktuelle Version (Ruby 2.2 respektive Rails 4.2) zu unterstützen. Als Datenbank-System wurde PostgreSQL eingesetzt. Javascript wurde primär für kleinere Optimierungen und Erleichterungen benutzt.

[13] github.com/upd89/doc

7.5. Arbeitsablauf

Um die Qualität der abgeschlossenen Tasks zu erhöhen und sicherzustellen wurde das 'Vier-Augen-Prinzip' angewendet. Jeder Task musste, bevor dessen Status auf 'Erledigt' gesetzt wurde, vom jeweils anderen Teammitglied kontrolliert und als akzeptabel eingestuft hat. Dazu existierten unterschiedliche Bedingungen: Bei Unterstützungstasks (z.B. Dokumentation o. Ä.) wurde, wo sinnvoll, eine Liste von Akzeptanzkriterien im Task selbst definiert; bei Umsetzungstasks musste der Code über Unit-Tests getestet und gelintet[14] sein.
Der normale Workflow eines Tasks in Redmine sah deshalb folgendermassen aus:

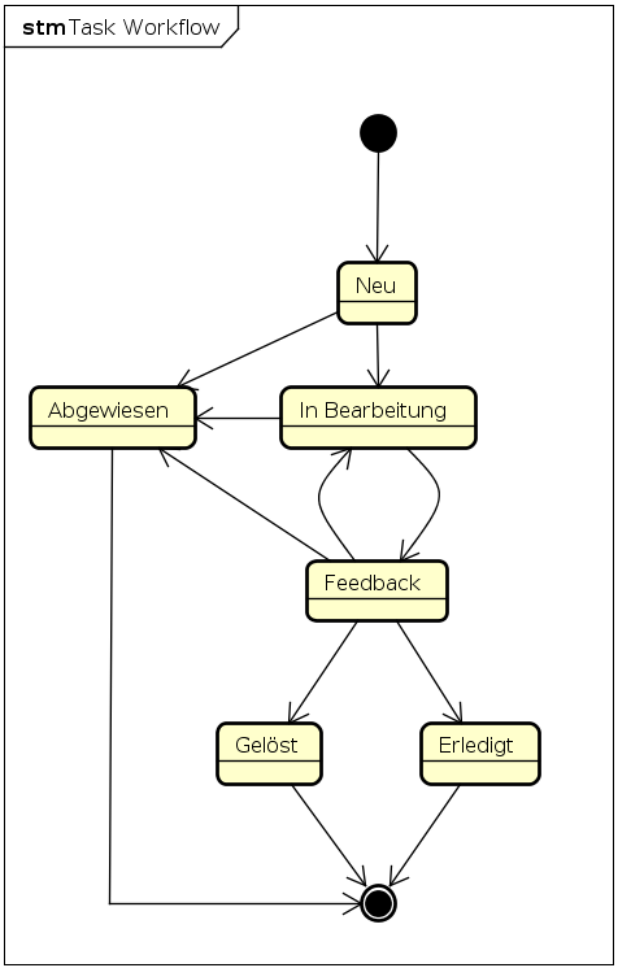


Abbildung 7.3.: Workflow eines Tasks

Wenn der Reviewer mit dem Resultat des Tasks nicht zufrieden war oder noch offene Punkte oder Fragen hatte, wurde der entsprechende Task wieder an den ursprünglichen Bearbeiter zurückgewiesen und in den Status 'In Bearbeitung' gesetzt. Das Feedback dazu konnte mündlich oder via Kommentar-Funktion erfolgen.

[14] Linting ist ein Prozess zur statischen Code-Analyse um potenzielle Fehler zu finden

7.6. Qualitätsmanagement

Stil

Um den Stil des Source-Codes einheitlich zu halten, wurde der 'Ruby On Rails Style Guide'[15] angewendet. Um diesen zu prüfen und sicherzustellen wurde Rubocop[16] eingesetzt.

Testing

Wo immer sinnvoll wurden Unit-Tests für das Testen der Funktionalität verwendet. Es wurde eine Sammlung von Unit-Tests erstellt, welche die Applikation automatisch testeten.
Dabei wurde aber nicht stur der Ansatz des Test-Driven Developments verfolgt, es sollten insbesondere kritische Komponenten ausgiebig getestet werden, um die Nachhaltigkeit des Projekts sicherzustellen. Die Umsetzung wurde im Kapitel 4.4 behandelt.

Usability

Weil auch einfache Bedienbarkeit im Fokus lag, wurden Usability-Überlegungen zusammen mit dem Kunden besprochen und entsprechend umgesetzt. Es wurden auch einige Usability-Tests, primär in Form von Hallway-Testing, geplant und umgesetzt. Dies wird in Kapitel 4.4 genauer beschrieben.

[15] github.com/bbatsov/rails-style-guide
[16] batsov.com/rubocop/

Literatur

[1] Jakob Nielsen. *Why You Only Need to Test with 5 Users*. 2000. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/ (besucht am 14. 06. 2016).

[2] W3Techs. *Usage of content languages for websites*. 2016. URL: https://w3techs.com/technologies/overview/content_language/all (besucht am 14. 06. 2016).

Glossar

Agent Komponente, die dem Control Center meldet, welche Updates ausstehen sind und die Updates auf Befehl des CC durchführt. vii, 8, 24, 28, 31, 45

AJAX 'Asynchronous JavaScript and XML', eine Web-Technologie wo Inhalte asynchron nach dem initialen Laden der Seite über Javascript dynamisch nachgeladen wird. 23, 39

API Application Programming Interface. 28, 51, 83

apt Kurz für 'Advanced Packaging Tool', ein Paket-Manager für Debian- und Ubuntu-Systeme. 9, 16, 18, 27, 28, 30, 45, 50

https://manpages.debian.org/cgi-bin/man.cgi?query=apt&sektion=8

CA Zertifizierungsstelle. 32, 33, 46, 47, 51

Certificate Authority unterschreibt Zertifikate, damit festgestellt werden kann, welche Zertifikate bzw. Systeme einander vertrauen können. 26

Control Center Komponente, die Reports anzeigt und die Steuerung der Updates ermöglicht. vii, 8, 9, 15, 22, 28, 31, 39, 50

CRUD 'Create, Read, Update, Delete', grundlegende Operationen an einem Datensatz. 24

Daemon Ein im Hintergrund ablaufender Prozess. 27

HSR Hochschule für Technik Rapperswil. 95, 97, 99–108, 111, 112, 114–116, 118

http://hsr.ch/

MITM Man in the middle. 8

Mockup Vereinfachter Entwurf eines komplexeren Produkts (z.B. einer Webseite), welches annähernd so aussieht wie das Endprodukt, aber noch keine Funktionalität bietet. So können wichtige Design-Entscheidungen bereits vor dem ersten Prototypen gemacht werden.. 34

MVC Model-View-Controller. 22, 23

nine.ch Das Unternehmen Nine Internet Solutions AG. vii, ix, 8, 10, 19–21, 50, 53, 55, 57, 58, 73, 76, 96–100, 102, 103, 106, 108–116, 118

https://nine.ch

Pagination Das Aufteilen von einer langen Liste in mehrere Seiten, um sie übersichtlicher zu machen und die Ladedauer zu minimieren.. 23

pip Paket-Manager für Python. 27

https://pypi.python.org/pypi/pip

Rational Unified Process Ein Prozess um Software zu entwickeln. 53

https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

SCSS 'Sassy CSS', erweitertes CSS mit Unterstützung von Variabeln, Funktionen und weiteren Annehmlichkeiten. Scss-Dateien werden durch den SASS-Compiler in CSS-Dateien umgewandelt.. 40

http://sass-lang.com/

UPD89 Update Nine.ch. 8, 23, 46