Fachhochschule Aachen
Campus Jülich

Fachbereich: Medizintechnik und Technomathematik
Studiengang: Scientific Programming

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Konzeption eines teilautonomen Auftragssystems für das JuNet-IP-Management am Forschungszentrum Jülich

Bachelorarbeit von

Gitte Kremling

Jülich, August 2017

Eidesstattliche Erklärung

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

Ort und Datum	Unterschrift

Die Arbeit wurde betreut von:

Erstprüfer: Prof. Dr. rer. nat. Volker Sander
Zweitprüfer: Dipl.-Inf. Ralph Niederberger

Diese Arbeit wurde erstellt am:

Jülich Supercomputing Centre
Forschungszentrum Jülich GmbH

Zusammenfassung

Um die Institute des Forschungszentrums Jülich untereinander und mit der Außenwelt datentechnisch zu verbinden, wurde das Local Area Network "JuNet" geschaffen. Zur Verwaltung und Kontrolle dieses Netzes werden die daran angeschlossenen Geräte in einer Datenbank erfasst. Aufgrund des großen Umfangs der Daten ist deren Administration nur mit einem geeigneten Management-Tool zu bewerkstelligen. Dieses muss Funktionen zur Eintragung von Änderungen an den Datenbeständen und zum Auslesen bestimmter Informationen bereitstellen.

Da das derzeit verwendete Management-Tool lediglich unterstützend bei diesen Aufgaben eingesetzt wird, wird im Rahmen dieser Arbeit ein neues Auftragssystem konzipiert, das die Aufträge teilautonom bearbeitet. Eine Anforderung ist dabei die Einführung einer automatischen regelbasierten Autorisierung. Hierzu wird eine rollenbasierte oder attributbasierte Zugriffskontrolle in Betracht gezogen. Außerdem wird ein Konzept für eine Ablaufsteuerung zur automatischen Bearbeitung von Aufträgen entwickelt, wobei eine Implementierung als regelbasiertes System oder Zustandsautomat diskutiert wird. Eine zusätzliche gewünschte Erweiterung für das neue System stellt die Erstellung einer Auftragshistorie dar. In der Arbeit werden verschiedene relationale und dokumentenorientierte Datenbankmodelle zum Speichern einer solchen analysiert.

Inhaltsverzeichnis

1 Einleitung **1**

2 Auftragssystem für das JuNet-IP-Management **3**
- 2.1 Zugrunde liegende Datenbanken . . . 3
- 2.2 Bisheriges System . . . 8
- 2.3 Zukünftiges System . . . 9

3 Autorisierung **13**
- 3.1 Art der Zugriffskontrolle . . . 13
  - 3.1.1 Rollenbasierte Zugriffskontrolle . . . 13
  - 3.1.2 Attributbasierte Zugriffskontrolle . . . 14
  - 3.1.3 Anwendung auf das JuNet-IP-Management . . . 14
- 3.2 Extensible Access Control Markup Language . . . 15
- 3.3 Diskussion und Umsetzung . . . 18

4 Ablaufsteuerung **21**
- 4.1 Regelbasiertes System . . . 21
- 4.2 Zustandsautomat . . . 23
- 4.3 Diskussion und Umsetzung . . . 23
  - 4.3.1 Globaler Workflow . . . 24
  - 4.3.2 Überprüfungen . . . 28

5 Auftragshistorie **31**
- 5.1 Anforderungen . . . 31
- 5.2 Mögliche Datenbankstrukturen . . . 31
  - 5.2.1 Relationale Datenbanken . . . 32
  - 5.2.2 Dokumentenorientierte Datenbanken . . . 35
- 5.3 Diskussion und Umsetzung . . . 40

6 Zusammenfassung und Ausblick **45**

A. Abbildungsverzeichnis **I**

B. Tabellenverzeichnis **II**

C. Listings **III**

1 Einleitung

Die Forschungszentrum Jülich GmbH, eine Einrichtung des Bundes und des Landes Nordrhein-Westfalen, umfasst zehn Institute, die in den Bereichen Energie, Umwelt, Information und Gehirn forschen [1]. Um diese untereinander und mit der Außenwelt datentechnisch zu verbinden, wurde das Local Area Network JuNet eingerichtet. Zum Betrieb dieses Netzes gehört unter anderem auch die Erfassung aller angeschlossenen Geräte, um einen Überblick über das Netzwerk zu bekommen.

Momentan wird zur Verwaltung der Netzobjekte ein im Forschungszentrum Jülich entwickeltes Management-Tool eingesetzt, welches bei dieser Aufgabe unterstützt und Funktionen zum Speichern, Laden und Bearbeiten der Daten bereitstellt. Historisch bedingt wird jedoch die Übernahme der Daten aus den Rechneran-, -um- und -abmeldungen noch manuell erledigt. Aus diesem Grund soll ein neues Auftragssystem entwickelt werden, welches die Arbeitsschritte weitgehend automatisiert und nur in kritischen Fällen einen manuellen Eingriff notwendig macht. Gegenstand dieser Arbeit ist die Konzeption eines solchen teilautonomen Auftragssystems.

In Kapitel 2 wird die Datenstruktur zum Speichern der einzelnen Netzobjekte vorgestellt, der Arbeitsablauf und die Funktionsweise des bisherigen Systems beschrieben und die Anforderungen und Randbedingungen an das zukünftige System erläutert. In den folgenden Kapiteln werden dann Konzepte zur Umsetzung der hier aufgeführten wesentlichen drei Anforderungen Autorisierung, Ablaufsteuerung und Auftragshistorie entwickelt.

Verschiedene Möglichkeiten zur Einführung einer Autorisierung werden in Kapitel 3 diskutiert. Dabei wird u.a. auf die Extensible Access Control Markup Language (XACML) als Standard für eine attributbasierte Zugriffskontrolle eingegangen.

Kapitel 4 beschäftigt sich mit der Implementierung einer Ablaufsteuerung zur automatischen Bearbeitung von Aufträgen. Hierfür werden sowohl regelbasierte Systeme als auch Zustandsautomaten in Betracht gezogen und die letztendliche Umsetzung erläutert.

Mögliche Datenbankstrukturen zum Speichern einer Auftragshistorie werden in Kapitel 5 analysiert, wobei relationale und dokumentenorientierte Datenbanken zur Diskussion stehen.

In Kapitel 6 wird schließlich ein Fazit gezogen und Erweiterungsmöglichkeiten für das Auftragssystem als Ausblick aufgezeigt.

2 Auftragssystem für das JuNet-IP-Management

Unter dem Begriff "JuNet" werden alle Subnetze des Forschungszentrums Jülich zusammengefasst. Das Campusnetz bildet die Grundlage der gesamten elektronischen Kommunikation innerhalb der Organisation. Für die effektive Verwaltung dieses Netzwerks ist es erforderlich, Daten zu den verschiedenen angeschlossenen Geräten zu speichern. Zum einen ermöglicht eine solche Datenhaltung einen Überblick über die einzelnen Netzwerkkomponenten zu er- und behalten. Zum anderen werden diese Daten bei der Generierung von verschiedenen zur Verwaltung des Netzes notwendigen Konfigurations-, Initialisierungs- und Managementdateien benötigt. Aufgrund des großen Umfangs der Daten ist deren Verwaltung nicht per Hand, sondern nur mit einem geeigneten Management-Tool, zu bewerkstelligen. Mit einem solchen Tool müssen sowohl Änderungen an den Datenbeständen eingetragen als auch bestimmte Informationen ausgelesen werden können.

Im Folgenden wird zunächst die Datenstruktur der zu speichernden Informationen vorgestellt. Darauf folgt eine Beschreibung des Arbeitsablaufs und der Funktionsweise des bisherigen Systems und die Erläuterung der Randbedingungen und Anforderungen an das zu konzipierende zukünftige System.

2.1 Zugrunde liegende Datenbanken

Die zu speichernden Daten werden in die fünf Entitäten Person, Gerät, Interface, Netz und Alias unterteilt. Das Entity-Relationship-Diagramm in Abbildung 2.1 zeigt die Beziehungen zwischen diesen. Aus Gründen der Übersichtlichkeit wurden in diesem Diagramm die Attribute der Entitäten nicht eingezeichnet. Sie können den Spalten der nachfolgenden Tabellenstrukturen entnommen werden.

Die eingezeichneten Beziehungen lassen sich folgendermaßen in Worte fassen. Jedes Gerät wird von genau einer Person administriert und kann von beliebig vielen weiteren Personen benutzt werden. Es besitzt ein Interface pro Subnetz, an das es angeschlossen ist. Jedes Interface kann verschiedene Aliasse haben, wobei jeder Alias, bestehend aus Aliasname und Domain, nur einmal im gesamten Netz verwendet werden darf, sodass er genau einem Interface zugeordnet werden kann.

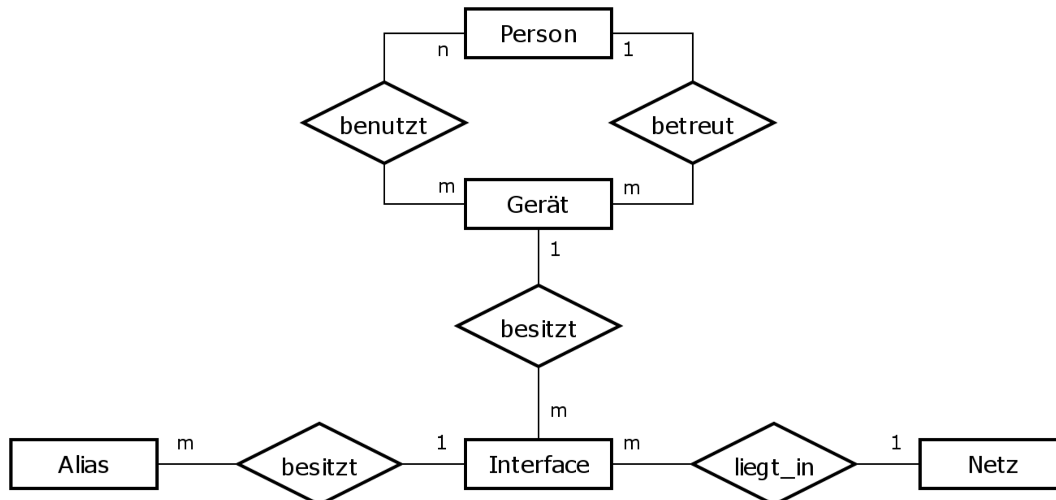

Abbildung 2.1: ER-Diagramm der zugrunde liegenden Datenstrukturen

Die Daten werden in einer Oracle-Datenbank mit verschiedenen Tabellen gespeichert, welche den genannten Entitäten entsprechen. Die Beziehungen zwischen den Entitäten werden in der Datenbank entweder durch Fremdschlüssel in einer Entitäten-Tabelle oder durch eine zusätzliche Tabelle, welche die Zuordnungen enthält, umgesetzt. Im Folgenden wird die genaue Struktur der Datenbank beschrieben.

d_pers (Person)

Die Tabelle *d_pers* stellt alle personenbezogenen Informationen bereit. Sie wird nicht vom IP-Management selbst, sondern vom JSC Dispatch[1] verwaltet. Die relevanten Spalten sind in Tabelle 2.1 beschrieben. Es werden Informationen, wie der Name, die Telefonnummer und die Email-Adresse einer Person, gespeichert.

Tabelle 2.1: Beschreibung der Tabelle *d_pers*

Spaltenname	**Beschreibung**
PERS_KEY	Identifikationsnummer der Person (Primärschlüssel)
NAME	Nachname
VORNAME	Vorname
SEX	Geschlecht (Herr | Frau)
TITEL	Titel
ORG	Organisationseinheit
EMAIL	E-Mail-Adresse
VORWAHL	Telefonvorwahl
TEL	interne Telefonnummer
...	...

[1] Verwaltungsinstanz des Jülich Supercomputing Centres (JSC)

x_device (Gerät)

In der Tabelle *x_device* werden allgemeine Informationen zu den einzelnen Geräten gespeichert. Dazu gehören beispielsweise der Name, der Standort, die Klasse und der Typ. Die wichtigsten Tabellenspalten sind in Tabelle 2.2 zu sehen.

Tabelle 2.2: Beschreibung der Tabelle *x_device*

Spaltenname	**Beschreibung**
DV_OBJ_ID	Identifikationsnummer des Gerätes (Primärschlüssel)
DV_LABEL	Rechnername
SERIAL	Seriennummer
DV_CLASS	Klasse des Gerätes
TYPE	Typ des Gerätes
DV_BUILDING	Standort des Gerätes (Gebäude)
DV_ROOM	Standort des Gerätes (Raum)
DV_LOC_ORG	Standort des Gerätes (Organisationseinheit)
PERSDATA	Speicherung personenbezogener Daten (Y | N)
ARRIVAL	Datum des Eintragens
...	...

x_port (Interface)

In der Tabelle *x_port* werden Daten zu den einzelnen Interfaces gespeichert. In Tabelle 2.3 sind die wichtigsten Spalten incl. Beschreibung aufgelistet. Über die Spalte NET_OBJ_ID wird das Subnetz referenziert, in der sich das Interface befindet. Die Zuordnung zu den Geräten aus der *x_device* Tabelle erfolgt durch Einträge in einer zusätzlichen Tabelle (*x_link*).

Die Spalte PT_DEV_ADDR beinhaltet die IPv4-Adresse des Interfaces. Die Spalten LLA und GUA dienen dem Speichern von IPv6-Adressen. Diese können manuell gesetzt oder je nach Konfiguration des Netzes, in der sich das Interface befindet, automatisch generiert werden. Dabei wird das entsprechende Netzpräfix verwendet und entweder die mittels Hardware-Adresse berechnete EUI64-Adresse[1] oder die umgewandelte IPv4-Adresse (4to6) angehangen. Der gewählte Adresstyp wird in der Spalte INET6_ADDR_TYPE festgehalten.

[1] Vom IEEE standardisiertes MAC-Adressformat zur Identifikation von Netzwerkgeräten

Tabelle 2.3: Beschreibung der Tabelle *x_port*

Spaltenname	**Beschreibung**
PT_OBJ_ID	Identifikationsnummer des Interfaces (Primärschlüssel)
PT_DEV_ADDR	IP-Adresse
NET_OBJ_ID	Identifikationsnummer des zugehörigen Netzes
HARDW_ADDR	Hardware-Adresse
SSH	SSH-Unlock (Y | N)
INET6_ADDR_TYPE	Typ der IPv6-Adresse (EUI64 | 4to6)
LLA	IPv6 Link Local Address
GUA	IPv6 Global Unicast Address
...	...

x_link

Die Tabelle *x_link* verknüpft die Tabelle *x_device* mit den Tabellen *d_pers* und *x_port*, d.h. sie bildet die Beziehungen zwischen einem Gerät und seinem Administrator, seinen Usern und seinen Interfaces ab. In Tabelle 2.4 sind die Spaltennamen mit einer kurzen Beschreibung gelistet.

Die Spalte CLASS_L gibt an, um welche Art von Verknüpfung es sich handelt, die Spalte OBJ_ID_L enthält dann die Identifikationsnummer aus der entsprechenden Tabelle. Analog zu CLASS_L beschreibt CLASS_H, worauf sich der Eintrag in der Spalte OBJ_ID_H bezieht. Zur Zeit enthält der Datensatz, der über die Identifikationsnummer referenziert wird, stets Informationen über das Gerät selbst. Aus diesem Grund wird in CLASS_H die Klasse des Gerätes eingetragen.

In der Tabelle muss zu jedem Gerät mindestens ein Eintrag mit einem Interface und genau ein Eintrag mit einem Administrator existieren. Die Anzahl der Einträge zu Benutzern ist variabel.

Tabelle 2.4: Beschreibung der Tabelle *x_link*

Spaltenname	**Beschreibung**
CLASS_H	Klasse des Gerätes
OBJ_ID_H	Identifikationsnummer des Gerätes
CLASS_L	Art der Verknüpfung (PORT | ADMIN | USER)
OBJ_ID_L	Identifikationsnummer des Interfaces bzw. der Person

n_devname (Alias)

In der Tabelle *n_devname* sind die Informationen zu den Aliasnamen der Interfaces gespeichert. Die wichtigsten Spalten werden in Tabelle 2.5 beschrieben. Die

Zuordnung zum Interface wird durch einen Fremdschlüssel, die Identifikationsnummer des Ports, gewährleistet. Über diese Spalte kann die Tabelle mit den in *x_port* gespeicherten Daten zu den Interfaces korreliert werden.

Den Primärschlüssel der Tabelle bilden die Spalten DN_NAME und DOMAIN zusammen, sodass jeder Aliasname nur genau einmal in einer Domain vorkommen darf und damit eindeutig einem Interface zugeordnet werden kann. Andersherum können zu einem Port mehrere Einträge in der Tabelle existieren, d.h. ein Interface kann verschiedene Aliasnamen haben. Jedes Interface hat jedoch nur genau einen kanonischen Namen. Dieser wird durch CANONICAL="Y" gekennzeichnet.

Tabelle 2.5: Beschreibung der Tabelle *n_devname*

Spaltenname	**Beschreibung**
DN_NAME	Aliasname
DOMAIN	Domain
PT_OBJ_ID	Identifikationsnummer des zugehörigen Interfaces
CANONICAL	kanonischer Aliasname (Y | N)
...	...

n_netname (Netz)

Die Tabelle *n_netname* enthält die Informationen zu den einzelnen (Sub-)Netzen des JuNets. Neben der Definition des Adressbereichs werden hier auch verschiedene Netz-Bezeichnungen und Informationen zum IPv6-Betrieb gespeichert. In Tabelle 2.6 werden die wichtigsten Spalten aufgezeigt und beschrieben.

Tabelle 2.6: Beschreibung der Tabelle *n_netname*

Spaltenname	**Beschreibung**
NN_OBJ_ID	Identifikationsnummer des Netzes (Primärschlüssel)
NN_NET_ADDR	Netzadresse
SUBNET_MASK	Subnetzmaske
NN_VLAN_NUM	VLAN ID
NN_NAME	Name des Netzes
MAIN_NAME	kanonischer Name (Y | N)
NN_S_SCOPE	zugehöriges physikalisches Netz
NN_DESC	Beschreibung des Netzes
NN_V4_STATE	IPv4 aktiv (Y | N)
NN_V6_STATE	IPv6 aktiv (Y | N)
NN_V6_NETPREF	Netzpräfix für IPv6-Adressen
NN_V6_AUTOCONF	Stateless Address Autoconfiguration aktiv (Y | N)
...	...

2.2 Bisheriges System

Über das JuNet-Portal, ein Web-Portal zur Verwaltung des Campus-Netzwerks, können Mitarbeiter des Forschungszentrums neue Netzgeräte anmelden, Änderungen zu bereits eingetragenen Geräten melden und Geräte aus der Datenbank löschen lassen. Das Portal wird auch für andere Verwaltungsaspekte des JuNets, wie das Melden von SSH-Keys oder Registrieren von X.509-Zertifikaten[1], genutzt.

Um auf das JuNet-Portal zugreifen zu können, muss sich ein entsprechender Benutzer zunächst per persönlichem X.509-Zertifikat oder mit seinem FZJ-E-Mail-Account authentifizieren. Zur Beantragung einer Änderung oder Löschung eines Gerätes sind nur solche Mitarbeiter autorisiert, die in der JuNet-Datenbank für mindestens ein Gerät als Administrator eingetragen sind.

In das Anmeldungs-, Änderungs- bzw. Löschungs-Formular können dann verschiedene Daten, entsprechend der zugrunde liegenden Datenbankstruktur, eingetragen werden. Bei der Anmeldung eines neuen Gerätes im Netz müssen beispielsweise unter anderem Host-ID, -Klasse, -Typ und Standort angegeben werden.

Nachdem der Auftrag abgesendet wurde, werden die Daten erstmals auf grobe Korrektheit, im Wesentlichen die Gültigkeit der Zeichen, geprüft. Sofern dabei keine Fehler auftreten, generiert der Server eine E-Mail mit den Auftragsdaten und sendet diese an den Auftraggeber und die Verantwortlichen des JuNet-IP-Managements. Sollte der Auftraggeber nicht der Administrator sein oder wurde im Feld "zusätzlich informieren" ein Eintrag gemacht, wird die E-Mail auch an diese Personen gesendet.

Ein für das JuNet-IP-Management zuständiger Mitarbeiter liest dann diese E-Mail und kann den Auftrag bearbeiten. Zur Erleichterung dieser Aufgabe wurde am Forschungszentrum Jülich ein Management-Tool entwickelt [2]. Dieses ist in der Programmiersprache Perl geschrieben und besteht aus einem Web Service und einem zugehörigen Client mit grafischer Oberfläche, genannt JuMAN-Tool. Zusammen übernehmen sie die Aufgabe der erneuten, feineren Überprüfung der Daten sowie die letztendliche Modifikation der Datenbank.

Der zuständige Mitarbeiter öffnet lokal den Client und kopiert die erhaltenen Auftragsdaten in die entsprechenden Felder der grafischen Oberfläche. Daraufhin werden die Einträge zunächst clientseitig auf Korrektheit geprüft. Hier werden erneut die Zeichen geprüft und auf sonstige Fehler getestet, welche keine Informationen aus den Datenbanken benötigen. Treten bei diesen Überprüfungen keine Fehler auf,

[1] ITU-T-Standard zum Erstellen digitaler Zertifikate

sendet der Client die Auftragsdaten per SOAP[1] an den Web Service. Die Kommunikation basiert dabei auf HTTPS und erfordert eine Benutzername-Passwort-Authentifikation.

Der Web Service kann über einen DBI-Proxy[2] Informationen aus den Datenbanken abrufen und somit genauere Überprüfungen machen. Beispielsweise wird an dieser Stelle der Fehler abgefangen, dass bei einer Anmeldung ein Aliasname angegeben wurde, der bereits einem anderen Gerät zugeordnet ist. Sind die Daten korrekt, wird die Datenbank über den DBI-Proxy entsprechend angepasst.

Eine Besonderheit des bisherigen Systems ist, dass die Überprüfungen, die auf verschiedenen Stufen der Erstellung und Abarbeitung eines Auftrags gemacht werden, unterschiedliche Resultate liefern können. Beispielsweise kann ein Eintrag, der im JuNet-Portal einen Fehler wirft, im Web Service lediglich eine Warnung auslösen. Diese unterschiedlichen Implementierungen ermöglichen es, eventuell kritische Einträge dennoch vornehmen und somit Spezialfälle behandeln zu können. Ein Beispiel hierfür ist die Anmeldung eines Gerätes mit einer bereits verwendeten Hardware-Adresse. Normalerweise ist eine solche Eintragung nicht erlaubt und führt daher auf Benutzerseite, d.h. im JuNet-Portal, zu einem Fehler. Um dasselbe Gerät an verschiedenen Außenstellen des Forschungszentrums einsetzen zu können, ist ein solcher Eintrag jedoch notwendig und wird aus diesem Grund vom Web Service akzeptiert.

Eine weitere Besonderheit ist die Art der Autorisierung. Eine Berechtigung, ein Gerät zu ändern oder zu löschen, hat jeder Mitarbeiter, der in der JuNet-Datenbank für mindestens ein Gerät als Administrator eingetragen ist. Das bedeutet aber nicht, dass er der Administrator eben dieses Gerätes sein muss. Mitarbeiter können also jegliche Änderungen an fremden Geräten melden, auch wenn sie dazu in der Realität nicht autorisiert sind. Es liegt dann in der Hand der Mitarbeiter des JuNet-IP-Managements zu entscheiden, ob der Auftrag ausgeführt oder abgelehnt wird.

2.3 Zukünftiges System

Da das bisher verwendete Management-Tool keine automatisierten Abläufe vorsieht und an einigen Stellen erweitert werden soll, wird in dieser Arbeit ein Konzept für ein neues System entwickelt. Dabei müssen verschiedene Anforderungen erfüllt und Randbedingungen eingehalten werden. Im Folgenden werden verschiedene Aspekte genannt, die bei der Konzeption zu beachten sind.

[1] Netzwerkprotokoll zum Austausch von Nachrichten und Dokumenten

[2] Treiberunabhängige Schnittstelle zur verwendeten Oracle-Datenbank

Programmiersprache

Als Programmiersprache für das neue System ist Perl vorgegeben. Dies liegt darin begründet, dass alle Mitarbeiter des IP-Managements diese Sprache beherrschen und das Programm dadurch später von ihnen erweitert und gewartet werden kann. Außerdem ist das bisherige System ebenfalls in dieser Sprache geschrieben, sodass eine Übernahme einzelner Codeausschnitte, wie beispielsweise der Methoden zur Überprüfung der Auftragsdaten, möglich ist.

Beibehaltung der Datenbankstruktur und des JuNet-Portals

Für das zukünftige System soll die aktuelle Datenbankstruktur beibehalten werden. Der Grund hierfür ist, dass andere Programme der Netzwerk-Verwaltung sie ebenfalls nutzen und die Kompatibilität zu diesen bestehen bleiben muss.

Außerdem soll das JuNet-Portal die Schnittstelle zum Melden von Änderungen seitens der Mitarbeiter bleiben. Da hier bei langjährigem Einsatz keine gravierenden Probleme aufgetreten sind und kleine Erweiterungen leicht einzupflegen sind, ist eine Neuimplementierung nicht notwendig.

Autorisierung

Wie in Abschnitt 2.2 beschrieben, kann zur Zeit jeder Mitarbeiter des Forschungszentrums, der Administrator mindestens eines Gerätes im JuNet ist, Änderungen an jedem beliebigen Gerät vornehmen oder dieses gar löschen lassen. Die einzige Instanz, die eine unberechtigte Anfrage ablehnen kann, sind dabei die für das JuNet-IP-Management zuständigen Personen.

Sicherer und arbeitserleichternd wäre hier eine automatische Autorisierung, die keine manuelle Entscheidung mehr benötigt. Dabei sind verschiedene Rollen zu beachten. Zum einen gibt es den Administrator, welcher dazu berechtigt ist, lediglich seine eigenen Geräte zu ändern oder zu löschen. Zum anderen gibt es die Mitarbeiter, die für das Management des JuNets zuständig sind. Sie dürfen Änderungen und Löschungen an jedem Gerät vornehmen.

Jede Organisationseinheit des Forschungszentrums benennt einen IT-Beauftragten (ITB), welcher als erster Ansprechpartner des Instituts zu allen Fragen der IT fungiert. Beim JuNet-IP-Management stellt der ITB eine weitere Rolle dar. Er wird autorisiert sein, Geräte seiner Organisationseinheit zu ändern oder zu löschen.

Außerdem soll es beim neuen System möglich sein, weitere Autorisierungen hinzuzufügen. So soll beispielsweise ein Administrator seine Rechte auf einen Kollegen übertragen können, damit dieser vertretungsweise seine Aufgaben übernehmen kann. Dabei ist sowohl die Autorisierung für einzelne als auch für alle von ihm administrierten Geräte denkbar.

Ablaufsteuerung

Eine weitere wichtige Anforderung an das zukünftige System ist die teilautonome Auftragsbearbeitung. Die Daten eines abgeschickten Auftrags sollen nicht mehr wie in Abschnitt 2.2 beschrieben von den Mitarbeitern des JuNet-Managements per Hand aus einer E-Mail in ein anderes Formular eingetragen werden, sondern automatisch an ein anderes Programm übertragen und dort verarbeitet werden.

Dieses prüft dann die übergebenen Daten auf Korrektheit. Sollten dabei keine Fehler auftreten, kann der Auftrag automatisch verarbeitet und die Datenbank entsprechend angepasst werden. Tritt ein gravierender Fehler auf, welcher die Verarbeitung des Auftrags verhindert, soll diese abgebrochen und eine entsprechende Benachrichtigung an den Antragsteller versendet werden. Entsteht bei den Überprüfungen eine Warnung, d.h. besteht zu einem Benutzereintrag Klärungsbedarf, ist ein manueller Eingriff erforderlich und der Auftrag soll solange in einen wartenden Zustand übergehen.

Die Mitarbeiter des JuNet-IP-Managements sollen die Möglichkeit haben auf einer eigenen Weboberfläche innerhalb des für die Formulare genutzten Web-Portals die wartenden Aufträge zu akzeptieren, zu bearbeiten oder abzulehnen. Je nach Entscheidung wird der Auftrag dann vom Auftragssystem erneut geprüft und bearbeitet oder verworfen.

Damit die Antragsteller immer über den Zustand ihres Auftrags informiert bleiben, soll bei jeder Status-Änderung eine E-Mail an sie versendet werden. Der Auftrag kann dabei folgende Status annehmen: hinzugefügt, abgelehnt, wartend oder erledigt. Die Benachrichtigungen sollen auch an die Mitarbeiter des Managements gesendet werden, sodass sie unter anderem informiert werden, wenn ein Auftrag einen manuellen Eingriff erfordert.

Auftragshistorie

Eine zusätzliche Funktion, die das neue System enthalten soll, ist die Erstellung einer Auftragshistorie und die Möglichkeit des Suchens in einer solchen. Die Historie soll dabei die Daten zu jeder Anmeldung, Änderung und Löschung eines Gerätes

speichern. Sie dient dazu, den Administratoren und Mitarbeitern des JuNet-IP-Management einen Überblick über den Verlauf der Gerätedaten zu verschaffen. Des Weiteren soll sie dazu genutzt werden, die Daten zu den wartenden Aufträgen zu speichern, die wegen eines notwendigen manuellen Eingriffs nicht direkt bearbeitet werden können.

Erweiterbarkeit und Wartbarkeit

Erweiterbarkeit und Wartbarkeit sind wichtige Aspekte, die bei der Konzeption des Systems zu beachten sind. Die Erweiterbarkeit schließt dabei sowohl das Hinzufügen weiterer Überprüfungen als auch die Integration zusätzlicher Datenfelder im Auftragsformular ein. Auch die Erweiterung auf andere Aufgaben des Netzwerk-Managements ist hier denkbar. Beispielsweise könnte ein ähnlicher teilautonomer Ablauf für die Registrierung von Zertifikaten oder Beantragung von Firewall-Freischaltungen genutzt werden.

3 Autorisierung

Eine Anforderung an das zukünftige System ist die Einführung einer Autorisierung, damit die Entscheidung über die Berechtigung einer Anfrage nicht mehr manuell getroffen werden muss. Dazu müssen zunächst Regeln definiert werden, wann die entsprechende Berechtigung eines Antragstellers vorliegt. Hierfür werden die Rollen Mitarbeiter, Administrator, Vertreter, IT-Beauftragter und Mitarbeiter des JuNet-IP-Managements eingeführt. Für die Autorisierung gelten dann folgende Regeln:

1. Jeder Mitarbeiter kann ein neues Gerät anmelden.
2. Ein Administrator kann seine Geräte ändern oder löschen.
3. Ein Administrator kann Vertreter für seine Geräte benennen.
4. Ein Vertreter kann die ihm zugewiesenen Geräte ändern oder löschen.
5. Ein IT-Beauftragter kann Geräte seines Instituts ändern oder löschen.
6. Ein IT-Beauftragter kann Vertreter für sein Institut benennen.
7. Ein Mitarbeiter des JuNet-IP-Managements kann alle Geräte ändern oder löschen.

Diese Regeln sollen in einer erweiterbaren Form gespeichert und effizient auf Anfragen angewandt werden können. Im Folgenden werden dazu verschiedene Implementierungsmöglichkeiten diskutiert.

3.1 Art der Zugriffskontrolle

Für die Implementierung der Autorisierung im Rahmen des JuNet-IP-Managements ist eine rollenbasierte oder attributbasierte Zugriffskontrolle denkbar. Im Folgenden werden die beiden Modelle allgemein beschrieben und anschließend diskutiert, welches für diesen Anwendungsfall besser geeignet ist.

3.1.1 Rollenbasierte Zugriffskontrolle

Bei der rollenbasierten Zugriffskontrolle (Role-based Access Control, kurz RBAC) werden den Benutzern eines Systems Rollen mit verschiedenen Zugriffsrechten zugeordnet [3]. Bei einer Anfrage wird dann zunächst ermittelt, in welcher Rolle der Benutzer ist und anschließend geprüft, ob diese Rolle berechtigt ist, die geforderte

Operation auszuführen. Es können auch Rollenhierarchien definiert werden, welche Erbrelationen zwischen den Rollen darstellen. Ein Beispiel hierfür sind die Rollen "Mitarbeiter" und "Manager". Manager erben die Rechte von Mitarbeitern, sind aber zusätzlich für weitere Operationen berechtigt.

Da die Berechtigungen nicht an einzelne Benutzer sondern an ganze Gruppen vergeben werden, ist ein eigenes Benutzer-Management nicht notwendig. Jedem Benutzer müssen lediglich seine Rollen zugewiesen werden. An einer anderen Stelle wird dann definiert, welche Rollen zu welchen Operationen berechtigt sind.

3.1.2 Attributbasierte Zugriffskontrolle

Die attributbasierte Zugriffskontrolle (Attribute-based Access Control, kurz ABAC) ist eine Weiterentwicklung von RBAC [3]. Wie der Name bereits erahnen lässt, basieren hierbei die Zugriffsregeln nicht auf Benutzerrollen sondern auf sog. Attributen. Die Attribute können in die vier Kategorien Benutzer, Operation, Ressource und Umgebung zusammengefasst werden. Mithilfe von Benutzerattributen können Informationen zum anfragenden Benutzer gespeichert werden. Beispiele hierfür sind Alter, Institut und Beruf. Operationsattribute beschreiben die Aktion, die ausgeführt werden soll, wie beispielsweise "Lesen" oder "Löschen". Ressourcenattribute enthalten Informationen zum Objekt, auf welches zugegriffen werden soll. Zu Attributen, die aus der Umgebung übernommen werden können, zählen z.B. Uhrzeit und Ort.

Die Zugriffsregeln werden bei ABAC in sogenannten Policies definiert. Diese beinhalten Regeln in Form von "if-then"-Anweisungen, in denen verschiedene Attributwerte abgefragt und mit erwarteten Werten verglichen werden können. Definierte Regeln können sowohl Zugriff gewähren als auch verbieten. Eine Policy könnte beispielsweise folgendermaßen aussehen: Ein Benutzer kann ein Dokument lesen, falls das Dokument aus dem Institut des Benutzers stammt und die Anfrage zwischen 9 und 17 Uhr gestellt wird. In diesem Fall müssten folgende Attribute abgefragt werden: das Institut des Benutzers, das Institut des Dokumentes, die angefragte Aktion und die Uhrzeit der Anfrage.

3.1.3 Anwendung auf das JuNet-IP-Management

Mithilfe einer rollenbasierten Zugriffskontrolle können den Benutzern die oben genannten verschiedenen Rollen zugeordnet werden. Die Definition der geltenden Regeln gestaltet sich allerdings teilweise schwierig, da sie sich nicht nur auf die jeweilige Rolle beziehen, sondern auch weitere Aspekte beachtet werden müssen. Regel 2 zum

Beispiel besagt, dass Administratoren ihre Geräte ändern und löschen können. Beim RBAC-Modell ist es jedoch nicht möglich das Besitzkriterium in die Entscheidung einzubeziehen, d.h. die Autorisierung davon abhängig zu machen, welche Geräte von dem Antragsteller administriert werden.

Aus diesem Grund muss eine attributbasierte Zugriffskontrolle verwendet werden. Hier können einem Benutzer seine administrierten Geräte über ein Attribut zugeordnet und in einer Policy darauf zugegriffen werden. Da ABAC eine Erweiterung des RBAC-Modells ist, können damit dennoch Rollen dargestellt werden. Diese werden in Form eines Benutzerattributes gespeichert. Auch die anderen Informationen zu den Benutzern, wie die zugewiesenen Geräte eines Vertreters oder das Institut von IT-Beauftragten, können mithilfe von Attributen gespeichert werden.

3.2 Extensible Access Control Markup Language

Der Standard zur Implementierung des ABAC-Modells ist die e**X**tensible **A**ccess **C**ontrol **M**arkup **L**anguage (XACML) [4]. Sie definiert eine Architektur, eine Policy Sprache und ein Request/Response Schema, die im Folgenden beschrieben werden.

Architektur

Die von XACML definierte Architektur besteht aus vier Komponenten: Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Administration Point (PAP) und Policy Information Point (PIP). In Abbildung 3.1 sind die Relationen dieser Komponenten sowie ihre Rollen bei der Verarbeitung einer Autorisierungsanfrage dargestellt.

Sendet ein Benutzer eine Anfrage, wird diese zunächst vom PEP unterbrochen, um zu prüfen, ob der Benutzer berechtigt ist, diese Anfrage auszuführen. Der PEP wandelt die Anfrage in einen XACML Request um und sendet diesen an den PDP. Hier wird die Zugriffsentscheidung getroffen, indem die geltenden Policies in Bezug auf die Anfrage ausgewertet werden. Werden in einer Policy Attribute benötigt, deren Werte nicht im XACML-Request enthalten sind, können diese bei PIPs abgefragt werden. Ein PIP hat Zugriff auf eine Datenbank oder Cloud, denen er die benötigten Informationen entnehmen kann. Am PDP wird dann mithilfe der Policy-Auswertungen entschieden, ob der Zugriff zugelassen oder abgelehnt wird und ein entsprechender XACML Response an den PEP gesendet. Bei einem positiven Ergebnis gewährt dieser dann den Zugriff auf die angefragte Ressource; andernfalls verwehrt er ihn. Über den PAP können die verschiedenen Policies gemanagt werden, d.h. neue Policies hinzugefügt und existierende geändert oder gelöscht werden.

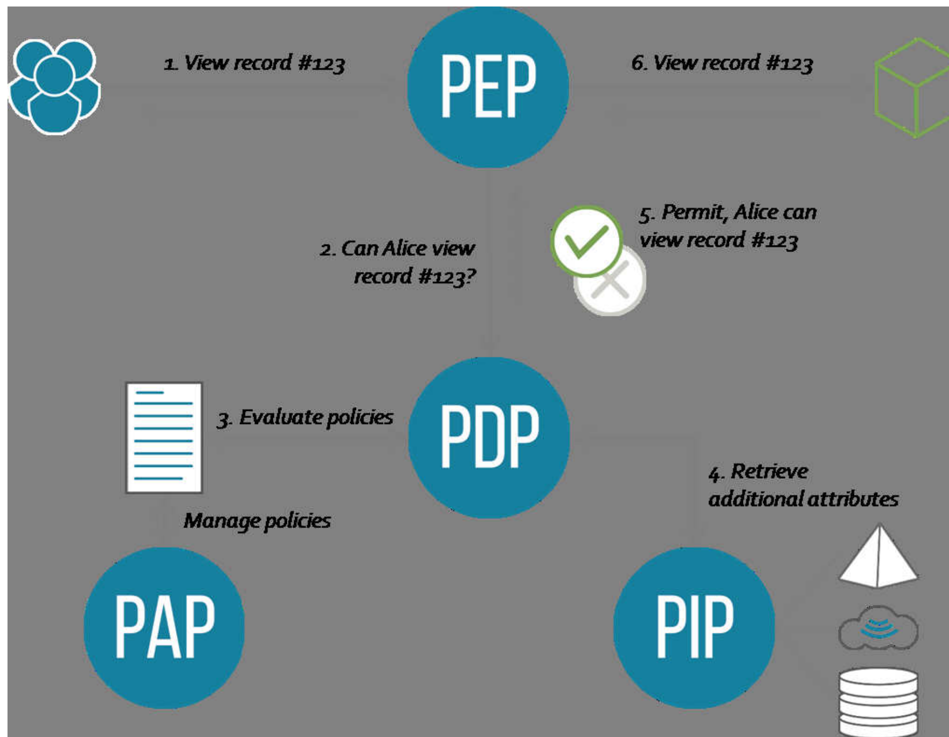

Abbildung 3.1: XACML Architektur und Verlauf einer Autorisierungsanfrage [5]

Policy Sprache

Policies beschreiben die Zugriffsregeln und werden bei XACML als XML-Dateien gespeichert. Das Wurzelelement einer solchen XML-Datei ist entweder *Policy* oder *PolicySet*, wobei eine Policy mehrere Regeln und ein PolicySet wiederum mehrere Policies enthalten kann. Da die einzelnen Regeln und Policies jeweils verschiedene Zugriffsentscheidungen treffen können, muss zu jeder Policy und jedem PolicySet ein *Combining Algorithm* angegeben werden, welcher besagt wie die verschiedenen Ergebnisse zu einer Entscheidung zusammengefasst werden. Ein Beispiel hierfür ist der Algorithmus *Deny Overrides*, der besagt, dass sofern eine der Regeln oder Policies den Zugriff verwehrt, das Gesamtergebnis ebenfalls ein "Deny" ist, auch wenn andere das Ergebnis "Permit" lieferten. Zusätzlich zu sieben vordefinierten Algorithmen können auch eigene hinzugefügt werden.

Jede Policy enthält ein *Target*-Element und beliebig viele *Rule*-Elemente. Das Target legt eine Reihe von Bedingungen für das Subjekt, d.h. den Benutzer, die Ressource und die Aktion einer Anfrage fest, die gegeben sein müssen, damit die entsprechende Policy angewandt werden kann. Sind diese erfüllt, werden die in dieser Policy definierten Regeln ausgewertet. Eine Regel ist hauptsächlich über ein

Condition-Element, das Bedingungen enthält, und ein *Effect*-Attribut mit dem Wert "Permit" oder "Deny" definiert.

Die Bedingungen in *Target*- und *Rule*-Elementen werden über boolsche Funktionen auf verschiedenen Attributen des Subjekts, der Ressource, der Aktion oder der Umgebung definiert. Die Attribute können dabei, wie bei der Architektur beschrieben, der Anfrage entnommen oder an anderen Stellen abgefragt werden. In den Bedingungen werden die Attribute dann mithilfe der Funktionen mit erwarteten Werten verglichen, um eine Zugriffsentscheidung zu treffen. Funktionen können dabei auch selbst definiert und verschachtelt werden, wodurch eine Bedingung beliebig komplex gestaltet werden kann.

Ein Beispiel für eine Regel ist in Listing 3.1 zu sehen. Die Regel besagt, dass jede Anfrage abgelehnt werden soll, die von einem Benutzer stammt, der sich vor mehr als 30 Tagen das letzte Mal eingeloggt hat.

Listing 3.1: Beispiel einer XACML Regel [6]

```
<xacml3:Rule RuleId="f6637b3f-3690-4cce-989c-2ce9c053d6fa" Effect="Deny">
    <xacml3:Description>Use it or lose it: this policy denies access if ↩
                    ↪lastLogin is more than 30 days away from today&apos↩
                    ↪;s date</xacml3:Description>
    <xacml3:Target/>
    <xacml3:Condition >
        <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-↩
                    ↪of">
            <xacml3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:↩
                    ↪function:dateTime-less-than"/>
            <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:↩
                    ↪dateTime-add-dayTimeDuration">
                <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:↩
                    ↪function:dateTime-one-and-only">
                    <xacml3:AttributeDesignator Category="urn:oasis:names:tc↩
                    ↪:xacml:1.0:subject-category:access-subject" ↩
                    ↪AttributeId="com.acme.user.lastLogin" DataType="↩
                    ↪http://www.w3.org/2001/XMLSchema#dateTime" ↩
                    ↪MustBePresent="false"/>
                </xacml3:Apply>
                <xacml3:AttributeValue DataType="http://www.w3.org/2001/↩
                    ↪XMLSchema#dayTimeDuration">P30D</xacml3:↩
                    ↪AttributeValue>
            </xacml3:Apply>
            <xacml3:AttributeDesignator Category="urn:oasis:names:tc:xacml↩
                    ↪:3.0:attribute-category:environment" AttributeId="↩
                    ↪urn:oasis:names:tc:xacml:1.0:environment:current-↩
                    ↪dateTime" DataType="http://www.w3.org/2001/↩
                    ↪XMLSchema#dateTime" MustBePresent="false"/>
        </xacml3:Apply>
    </xacml3:Condition>
</xacml3:Rule>
```

Request/Response Schema

Auch XACML Requests und Responses werden in XML geschrieben. Eine Anfrage besteht dabei im Wesentlichen aus verschiedenen Attributen zu dem Subjekt, der Ressource und der Aktion, die ausgeführt werden soll. Diese Informationen werden an den PDP gesendet und von ihm in Bezug auf die Policies ausgewertet, um zu entscheiden, ob der Zugriff erlaubt wird oder nicht. Die jeweilige Entscheidung wird dann über ein XACML Response zurückgegeben, wobei einer der vier Werte "Permit", "Deny", "Indeterminate" und "Not Applicable" angenommen werden kann. Die Antwort "Indeterminate" wird gesendet, falls bei der Auswertung ein Fehler aufgetreten ist oder ein benötigtes Attribut gefehlt hat, sodass keine Entscheidung getroffen werden konnte. "Not Applicable" bedeutet, dass das Target keiner Policy auf die gestellte Anfrage passt, d.h. die Anfrage nicht von diesem Service beantwortet werden kann.

3.3 Diskussion und Umsetzung

XACML eignet sich gut für die Autorisierung im Auftragssystem. Die Policy Sprache ist dazu geeignet, die oben beschriebenen Zugriffsregeln abzubilden und das Request/Response Schema ist ebenfalls passend. Über den PAP können neue Policies leicht hinzugefügt werden und durch die Erweiterbarkeit von XACML sind den Regeln dabei kaum Grenzen gesetzt. Die Möglichkeit der Abfrage weiterer Attribute über PIPs ist ebenfalls ein wichtiger Aspekt, da Informationen aus externen Datenbanken, wie der Administrator eines Gerätes oder die Vertreter, die ein Administrator bestimmt hat, für die Autorisierungsentscheidung hinzugezogen werden müssen.

Allerdings gibt es keine offizielle Implementierung von XACML in Perl, sodass diese selbst geschrieben werden müsste. Dazu gehört unter anderem die Implementierung des PEP, d.h. die Konvertierung der Anfragen in XACML Requests, und des PDP, d.h. die Auswertung der XACML Requests mithilfe der XACML Policies und das Formulieren der Antwort als XACML Response. Außerdem müssten entweder auch die Policies von einem benutzerfreundlichen Format in die oben beschriebene Sprache konvertiert oder direkt von den Benutzern in dieser geschrieben werden. Durch die Verwendung der Policy Sprache und des Request/Response Schema von XACML wird aufgrund der beschriebenen Aspekte ein Programmier-Overhead bedingt. Daher ist die Implementierung mithilfe eines selbst definierten einfacheren Schemas, das an die speziellen Bedürfnisse des Auftragssystems angepasst ist, an dieser Stelle sinnvoller. Die von XACML definierte Architektur kann dabei jedoch übernommen werden.

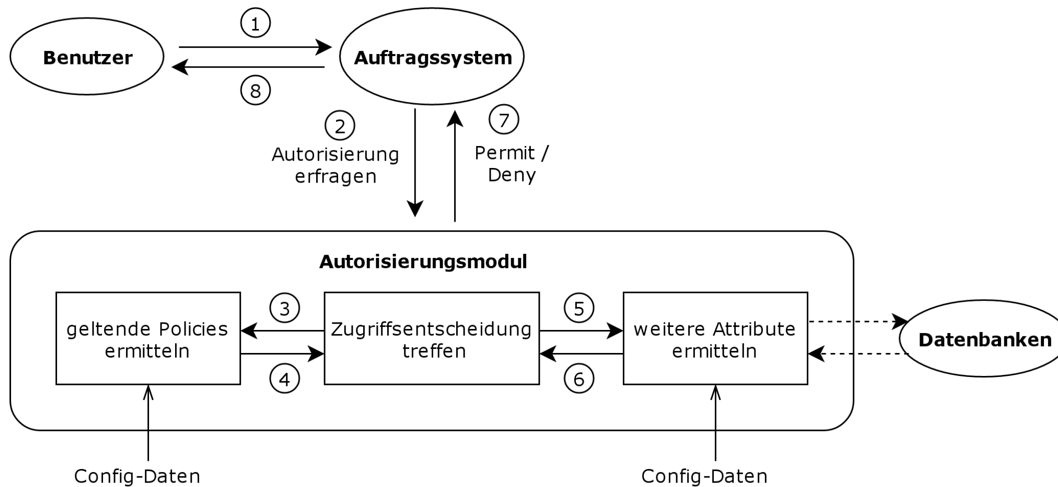

Abbildung 3.2: Aufbau des Autorisierungsmodul und Kommunikationsablauf mit anderen Komponenten

Die Implementierung der Autorisierung sollte in einem eigenen Perl-Modul gekapselt werden. Der Aufbau dieses, die Kommunikationsschnittstellen zu den anderen Komponenten sowie das Kommunikationsablaufschema sind in Abbildung 3.2 dargestellt. Erstellt ein Benutzer einen neuen Auftrag im Web-Portal, wird dieser an das Auftragssystem übermittelt (1). Dieses fungiert als PEP und leitet die Anfrage zunächst an das Autorisierungsmodul weiter, um zu erfragen, ob eine entsprechende Berechtigung für diese Anfrage vorliegt (2). Das Autorisierungsmodul übernimmt dann die Aufgaben des PDP. Es ermittelt die geltenden Policies (3 und 4) und wertet die erhaltene Anfrage in Bezug auf diese aus. Werden dabei weitere Attribute benötigt, werden diese von externen Datenbanken abgefragt (5 und 6). Die dazu verwendeten PIPs sind als interne Methoden im Autorisierungsmodul realisiert. Nachdem eine Entscheidung getroffen wurde, wird diese als Antwort an das Auftragssystem gesendet (7) und dem Benutzer eine entsprechende Meldung ausgegeben (8).

Die Policies werden als Konfigurationsdaten, zum Beispiel in Form einer XML-Datei, gespeichert. Diese Daten enthalten, wie bei XACML, Regeln in Form von boolschen Funktionen zu verschiedenen Attributen und Verknüpfungsalgorithmen für die einzelnen Regeln. Im Autorisierungsmodul werden die Konfigurationsdaten dann eingelesen und verarbeitet. Ein zusätzlicher PAP existiert hierbei nicht. Zum Hinzufügen, Ändern und Löschen von Policies müssen die Konfigurationsdaten entsprechend angepasst werden.

Auch für die PIPs können Konfigurationsdaten angegeben werden. Diese enthalten Definitionen von Subjekt- und Ressourcen-Attributen inklusive der Angabe, an welcher Stelle ihr Wert abgefragt werden kann. Das Autorisierungsmodul verarbeitet dann diese Daten und kommuniziert bei Bedarf mit den entsprechenden Datenbanken.

Das so aufgebaute Autorisierungsmodul ist durch die Auslagerung der Policy- und Attribut-Definitionen sehr gut erweiterbar. Neue Regeln und Attribute müssen lediglich den Konfigurationsdaten hinzugefügt werden; es müssen keine weiteren Änderungen am Code oder System vorgenommen werden. Die Form der Konfigurationsdaten sollte dabei möglichst einfach gehalten werden, um eine benutzerfreundliche Erweiterung zu ermöglichen. Gleichzeitig sollte sie möglichst variabel sein, damit auch komplexere Regeln und Attribute hinzugefügt werden können.

4 Ablaufsteuerung

Durch eine eingebaute Ablaufsteuerung soll das neue System Aufträge weitgehend automatisch bearbeiten. Dabei müssen die in Abschnitt 2.3 beschriebenen Anforderungen erfüllt und Randbedingungen eingehalten werden. Diese besagen, dass die über das JuNet-Portal gestellten Anfragen zur Anmeldung, Änderung oder Löschung eines Netzobjektes zunächst auf Korrektheit geprüft werden müssen. Treten dabei keine Fehler auf, kann der Auftrag ausgeführt und damit die Datenbanken entsprechend aktualisiert werden. Sollte ein Fehler entdeckt werden, welcher die Verarbeitung des Auftrags verhindert, muss diese abgebrochen und eine entsprechende Benachrichtigung versendet werden. Einen weiteren Fall stellt eine bei den Überprüfungen entstandene Warnung aufgrund einer Unklarheit bei einem Benutzereintrag dar. In diesem Fall ist ein manueller Eingriff erforderlich und der Auftrag soll solange in einen wartenden Zustand übergehen.

Im Folgenden wird diskutiert, welche Art der Ablaufsteuerung sich für dieses Szenario am besten eignet. Hierzu wird eine Implementierung als regelbasiertes System oder Zustandsautomat in Betracht gezogen und auf die letztendliche Umsetzung eingegangen.

4.1 Regelbasiertes System

Regelbasierte Systeme dienen der Abbildung logischen Wissens in die Programmierung [7]. Das logische Wissen besteht dabei aus einer Sammlung von Regeln in Form von “if-then”-Anweisungen. Der “if”-Teil wird als Prämisse bezeichnet und beinhaltet verschiedene Fakten, die beobachtet werden können. Die Konklusion, der “then”-Teil, beschreibt die Aktion, die ausgeführt werden soll, wenn die Prämisse eintrifft. In einem regelbasierten System werden diese Regeln in einer Reihe von Kreisläufen durchlaufen. In jedem Durchlauf wird die zu den vorliegenden Fakten passende Regel ermittelt und angewandt. Da die Ausführung der zugehörigen Aktion die Fakten wiederum verändern kann, ist es möglich, dass im nächsten Durchlauf eine andere Regel angewandt wird und der Verarbeitungsprozess somit voranschreitet.

Während eine “if-then”-Anweisung in konventionellen Programmiersprachen eine bedingte Verzweigung darstellt, d.h. einen Punkt im Code, an dem die Ausführung verschiedene Wege nehmen kann, sind die Regeln in einem regelbasierten System

an einer Stelle gesammelt und es wird je nach Sachlage eine davon ausgewählt und ausgeführt. Das bedeutet, dass die Regeln nicht geordnet sind und jede Regel zu jeder Zeit ausgeführt werden kann.

Ein Beispiel für ein regelbasiertes System ist in Listing 4.1 zu sehen. Es beschreibt den Ablauf zur Zubereitung einer Mahlzeit. Die Regeln hierzu lauten wie folgt. Gibt es dreckiges Geschirr, soll dieses abgewaschen werden. Stehen sauberes Geschirr und rohes Essen zur Verfügung, soll das Essen zubereitet werden. Liegen gekochtes Essen und sauberes Geschirr vor, soll das Essen serviert werden. In dem Listing ist auch ein Beispiel für die Ausführung des Systems mit den initialen Fakten "dreckiges Geschirr" und "rohes Fleisch" aufgezeigt.

Listing 4.1: Beispiel eines regelbasierten Systems

```
Rule 1 - Clean-Up:
  Given
    1: ( dirty-dishes )
  Do
    Remove fact 1
    add fact ( clean-dishes )

Rule 2 - Prepare-Food:
  Given
    1: ( clean-dishes )
    2: ( raw-food ?foodtype )
  Do
    Remove fact 1, fact 2
    add fact ( cooked-food ?foodtype )
    add fact ( dirty-dishes )

Rule 3 - Serve-Food
  Given
    1: ( clean-dishes )
    2: ( cooked-food ?foodtype )
  Do
    Remove fact 1, fact 2
    add fact ( dirty-dishes )

Start                  ( dirty-dishes ) ( raw-food beef )
Clean-Up               ( clean-dishes ) ( raw-food beef )
Prepare-Food           ( dirty-dishes ) ( cooked-food beef )
Clean-Up               ( clean-dishes ) ( cooked-food beef )
Serve-Food             ( dirty-dishes )
Clean-Up               ( clean-dishes )
```

4.2 Zustandsautomat

Ein Zustandsautomat wird durch eine Sammlung von Zuständen und Übergängen zwischen diesen, sog. Transitionen, definiert. Zu jedem Zeitpunkt befindet sich das System in genau einem der Zustände. Tritt ein Ereignis ein, das in diesem Zustand definiert ist, geht der Automat in einen neuen Zustand über, wobei Aktionen ausgeführt werden können. Während des Aufenthalts in einem Zustand können ebenfalls Aktionen ausgeführt werden. Enden diese, führt dies automatisch zu einem Ereignis und eine Transition in einen neuen Zustand wird ausgeführt.

Da in einem Zustandsautomaten die nächste Aktion bei Eintritt eines Ereignisses vom aktuellen Zustand abhängt, ist der Verlauf im Vergleich zu regelbasierten Systemen zeitabhängig.

Ein Beispiel für einen Zustandsautomaten ist in Abbildung 4.1 zu sehen. Er beschreibt den Ablauf beim Reservieren und anschließenden Buchen eines Fluges. Zunächst befindet sich der Automat im initialen Zustand. Wird ein Flug reserviert, geht er in den Zustand “flight reserved” über. Nun kann entweder die Reservierung storniert oder der Flug gebucht werden. Wird die Reservierung storniert oder ist die Buchung des Fluges nicht möglich, wird eine Transition in den Zustand “reservation canceled” ausgeführt. Soll der Flug gebucht werden und ist dies möglich, gelangt der Automat in den Zustand “flight booked”.

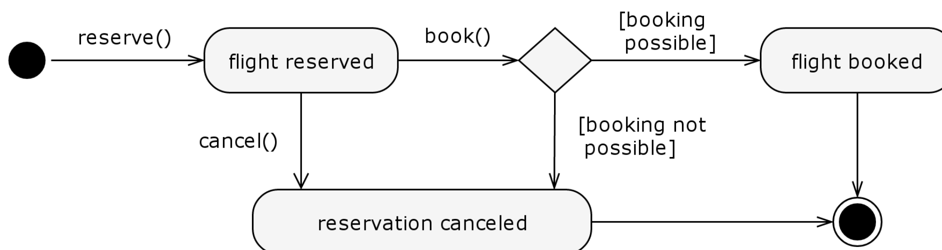

Abbildung 4.1: Beispiel eines Zustandsautomaten

4.3 Diskussion und Umsetzung

Im Folgenden wird diskutiert, welches der beiden oben beschriebenen Modelle sich besser zur Implementierung der automatischen Ablaufsteuerung beim Auftragssystem eignet. Dabei muss sowohl die Steuerung des globalen Workflows, d.h. der verschiedenen Schritte der Verarbeitung eines Auftrags wie oben beschrieben, als auch die Auswahl der auszuführenden Überprüfungen anhand der ausgefüllten Datenfelder betrachtet werden.

4.3.1 Globaler Workflow

Der globale Workflow steuert die gesamte Bearbeitung des Auftrags, insbesondere die Reaktion auf das Ergebnis der Überprüfungen. Da hierbei das weitere Vorgehen vom aktuellen Zustand eines Auftrags abhängt und nicht zu jeder Zeit dieselben Regeln angewandt bzw. Aktionen ausgeführt werden können, bietet sich hierfür ein Zustandsautomat an. Der Ablauf entspricht also eher einem Entscheidungsbaum als einer Sammlung losgelöster Regeln, die alle zu jeder Zeit angewandt werden können.

In Abbildung 4.2 ist ein entsprechender Zustandsautomat zu sehen. Für jeden Auftrag wird eine eigene Instanz dieses Automaten verwendet. Zu Beginn befindet sich dieser im Zustand `INITIAL`. Die erste Aktion, die dann ausgeführt werden kann, ist das Überprüfen des Auftrags auf Korrektheit. Abhängig von dem Ergebnis dieser, befindet sich der Auftrag anschließend im Zustand `CHECKS_OK`, `CHECKS_WARN` oder `CHECKS_ERR`.

In dem Fall, dass die Überprüfungen ohne Fehler durchgelaufen sind, wird der Auftrag anschließend bearbeitet und eine Transition in den Zustand `COMPLETED` ausgeführt. Traten bei den Überprüfungen gravierende Fehler auf, werden diese behandelt, d.h. eine entsprechende Meldung an den Antragsteller geschickt, und der Auftrag in den Zustand `ABORTED` überführt. Wurde bei den Überprüfungen eine Warnung ausgegeben, soll diese auch an den Auftraggeber und die Mitarbeiter des JuNet-IP-Managements gemeldet werden. Der Auftrag befindet sich anschließend im Zustand `WAITING`. In diesem können verschiedene Ereignisse eintreten.

Wird der Auftrag vom Antragsteller zurückgezogen oder von einem Mitarbeiter des JuNet-IP-Managements aufgrund eines Fehlers verwehrt, gelangt er in den Zustand `ABORTED` und wird nicht weiter bearbeitet. Die Mitarbeiter des JuNet-IP-Managements haben zwei weitere Möglichkeiten auf die Warnung zu reagieren. Sie können die Werte von Datenfeldern ändern und damit den Auftrag der Warnung entsprechend anpassen oder die Warnung ignorieren und den Auftrag so akzeptieren. In beiden Fällen gelangt der Auftrag in den Zustand `INITIAL` und wird erneut überprüft. Bei dieser zweiten Überprüfung werden die bereits ignorierten Warnungen nicht mehr beachtet. In der Zeit zwischen den beiden Prüfungsdurchläufen können andere Aufträge ausgeführt und somit Einträge in der JuNet-Datenbank geändert worden sein. Dadurch können bei den späteren Überprüfungen neue Fehler und Warnungen hinzukommen. Es ist beispielsweise möglich, dass eine angegebene IP-Adresse im ersten Durchlauf nicht belegt ist und somit akzeptiert wird, in einem späteren Durchgang jedoch zu einem Fehler führt, da sie in der Zwischenzeit vergeben wurde.

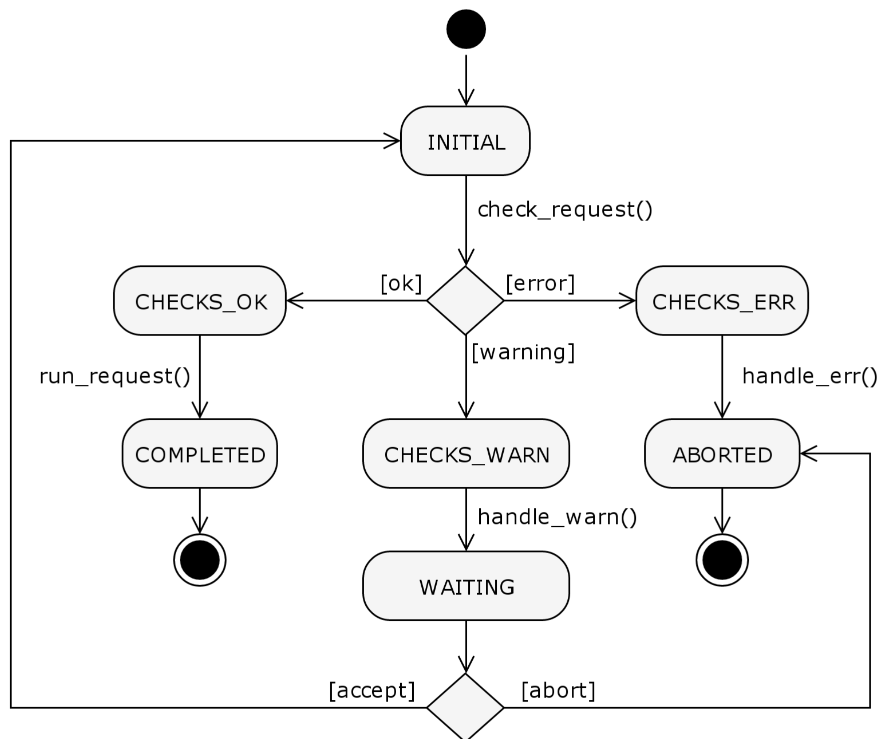

Abbildung 4.2: Zustandsautomat des globalen Workflows zur Bearbeitung eines Auftrags

Zustandsautomaten können in Perl über das Modul `workflow` implementiert werden [8]. Dabei können die verschiedenen Zustände, Zustandsübergange und Aktionen in XML-Dateien eines bestimmten Schemas angegeben werden. Zu jedem Zustand können die darin ausführbaren Aktionen angegeben werden. Jede Aktion wird in einer weiteren XML-Datei einer Funktion zugeordnet und besagt, welcher Zustand nach Ausführung der Aktion angenommen wird. Außerdem können zu einer Aktion Bedingungen festgelegt werden, die vor Ausführung dieser überprüft werden müssen. Ein Zustand kann auch als "autorun" definiert werden, sodass die einzelnen Aktionen nicht explizit aufgerufen werden müssen, sondern automatisch ausgeführt werden. Sind dabei in einem Zustand mehrere Aktionen definiert, wird anhand der Bedingungen entschieden, welche ausgeführt wird. Jeder Workflow besitzt außerdem einen eigenen Kontext, in dem Variablen definiert werden können. Auf diese kann u.a. in den Aktionen und beim Prüfen der Bedingungen zugegriffen werden. Die zu den Aktionen gehörenden Funktionen können keine Parameter übergeben bekommen, sondern müssen auf den Kontextvariablen arbeiten. Damit sichergestellt werden kann, dass die entsprechenden benötigten Variablen vorhanden sind, können diese in der XML-Datei, in der die Aktionen definiert werden, angegeben werden.

Listing 4.2 zeigt die XML-Datei zur Definition des Zustandsautomaten für die Ablaufsteuerung des Auftragssystems. Zusätzlich zu den in Abbildung 4.2 dargestellten Zuständen wurde ein Hilfszustand `CHECKS_DONE` definiert. In der Aktion "check_request" werden die Kontextvariablen "err" und "warn" im Fall von auftretenden Fehlern oder Warnungen gefüllt. Der Hilfszustand `CHECKS_DONE` behandelt dann das Ergebnis der Überprüfungen und entscheidet mithilfe dieser Kontextvariablen, welcher Zustand als nächstes angenommen wird. Kontextvariablen werden auch im Zusammenhang mit der erneuten Überprüfung von wartenden Aufträgen genutzt. In der Methode "accept" werden die ignorierten Warnungen in einer Kontextvariablen gespeichert. Bei der anschließenden Überprüfung wird diese dann genutzt, um eine erneute Ausgabe zu verhindern.

Alle Zustände außer `INITIAL` und `WAITING` laufen automatisch durch. Im initialen Zustand wird auf das Starten der Überprüfungen und im wartenden Zustand auf die Aktion der Mitarbeiter des JuNet-IP-Managements gewartet.

Listing 4.2: Implementierung des Zustandsautomaten mithilfe des workflow-Moduls

```
<workflow>
  <type>auftragssystem</type>

  <state name="INITIAL">
    <!-- neuen Auftrag pruefen und Ergebnis in Kontext speichern -->
    <action name="check_request" resulting_state="CHECKS_DONE"/>
  </state>

  <state name="CHECKS_DONE" autorun="yes">
    <!-- Fehler aufgetreten -->
    <action name="null" resulting_state="CHECKS_ERR">
      <condition test="exists $context->{err}"/>
    </action>

    <!-- Warnung aufgetreten -->
    <action name="null" resulting_state="CHECKS_WARN">
      <condition test="exists $context->{warn}"/>
    </action>

    <!-- Auftrag korrekt -->
    <action name="null" resulting_state="CHECKS_OK">
      <condition test="!(exists $context->{warn} or exists $context
                   ->{err})"/>
    </action>
  </state>
```

```
  <state name="CHECKS_OK" autorun="yes">
    <!-- Auftrag bearbeiten und anschliessend Mail versenden -->
    <action name="run_request" resulting_state="COMPLETED"/>
  </state>

  <state name="COMPLETED"/>

  <state name="CHECKS_ERR" autorun="yes">
    <!-- Fehler an Auftragsteller melden -->
    <action name="handle_err" resulting_state="ABORTED"/>
  </state>

  <state name="ABORTED"/>

  <state name="CHECKS_WARN" autorun="yes">
    <!-- Auftraggeber und Mitarbeiter des JuNet-IP-Managements ueber
                  Warnung informieren -->
    <action name="handle_warn" resulting_state="WAITING"/>
  </state>

  <state name="WAITING">
    <!-- Auftrag abbrechen und Antragsteller informieren -->
    <action name="abort" resulting_state="ABORTED"/>

    <!-- Datenfeld aendern und Auftrag erneut ueberpruefen -->
    <action name="change_field" resulting_state="INITIAL"/>

    <!-- Auftrag trotz Warnung akzeptieren und erneut pruefen -->
    <action name="accept" resulting_state="INITIAL"/>
  </state>
</workflow>
```

Listing 4.3 zeigt beispielhaft die Anwendung des Workflows. Es wird ein neuer Workflow entsprechend den Konfigurationsdateien erstellt, eine Aktion ausgeführt und der resultierende Zustand abgefragt.

Listing 4.3: Anwendung des workflow-Moduls

```
# Konfigurationsdateien angeben
$self->_factory()->add_config_from_file(
     workflow   => "workflow.xml",
     action     => "action.xml"
);

# neuen Workflow erzeugen
my $workflow = $self->_factory()->create_workflow('auftragssystem');
```

```
# Auftrag zu Kontext hinzufuegen und Ueberpruefung dieses starten
$workflow->context->param( request => $request);
$workflow->execute_action( 'check_request' );

# neuen Zustand abfragen (WAITING/ABORTED/COMPLETED)
print $workflow->state;
```

4.3.2 Überprüfungen

Da einige Datenfelder eines Auftrags optional sind, muss für die Überprüfung dieser zunächst ermittelt werden, welche Datenfelder ausgefüllt wurden und somit geprüft werden müssen. Da hierbei keine Reihenfolge der Überprüfungen festgelegt ist und diese somit nicht zeitabhängig sind, kann ein Zustandsautomat ausgeschlossen und die Implementierung als regelbasiertes System in Betracht gezogen werden. Die Prämissen der Regeln wären dann jeweils die verschiedenen Datenfelder und die Konklusionen die entsprechenden darauf anzuwendenden Prüffunktionen. Da die Regeln somit aber recht einfach sind und nicht zueinander in Beziehung stehen, d.h. kein richtiger Verarbeitungsprozess dargestellt wird, ist die Implementierung als regelbasiertes System unverhältnismäßig komplex. Aus diesem Grund werden sie in Form von einfachen "if-then"-Anweisungen implementiert.

Um dabei die Regeln erweiterbar zu halten, sollen sie in einer Config-Datei, wie in Listing 4.4 zu sehen, angegeben werden können. Die Überprüfungen werden dabei in globale Checks, welche immer durchgeführt werden sollen, und spezifische Datenfeld-Checks aufgeteilt.

Die verwendeten Namen der Überprüfungen entsprechen den Namen von Perl-Funktionen, welche diese durchführen. Sie sind in einem eigenen Modul definiert und dadurch gut erweiterbar. Jede dieser Perl-Funktionen bekommt die kompletten Auftragsdaten und jeweils ein Array zum Speichern evtl. auftretender Fehler und Warnungen übergeben. Nachdem alle angegebenen Datenfelder auf Korrektheit geprüft wurden, kann wie oben beschrieben je nach Inhalt der beiden Arrays entschieden werden, welchen Zustand der Auftrag als nächstes annimmt.

Listing 4.4: Beispiel einer Config-Datei zur Angabe der Prüffunktionen

```
<config>
    <global>
        <check>check_authorization</check>
    </global>

    <datenfeld name="geb">
        <check>check_geb_syntax</check>
    </datenfeld>
```

```
    <datenfeld name="hwa">
        <check>check_hwa_syntax</check>
        <check>check_hwa_unique</check>
    </datenfeld>
</config>
```

Zur Verarbeitung dieser Config-Datei und zur Ermittlung der auf einen Auftrag anzuwendenden Regeln wird ein eigenes Modul `Checks` geschrieben, dessen Verwendung in Listing 4.5 aufgezeigt ist.

Listing 4.5: Ermitteln der anzuwendenden Regeln mithilfe des Moduls `Checks`

```
use strict;
use Checks;

# XML-Datei einlesen
Checks->read_config( 'config.txt' );

# fuer den vorliegenden Auftrag durchzufuehrende Checks ermitteln
$checks_to_make = Checks->determine_checks( $auftrag );

# uebergebene Datenstruktur ($auftrag)
{geb => "16.3"}

# resultierende Datenstruktur ($checks_to_make)
[ 'check_authorization', 'check_geb_syntax' ]
```

5 Auftragshistorie

Eine weitere Anforderung an das zukünftige System ist die Erstellung einer Auftragshistorie, sodass Administratoren, autorisierte Benutzer und Mitarbeiter des JuNet-IP-Managements den Datenverlauf der Geräte im JuNet nachvollziehen können. Außerdem soll die Historie dazu genutzt werden, die Daten zu wartenden Aufträgen zu speichern, die wegen eines benötigten manuellen Eingriffs nicht direkt bearbeitet werden können. Im Folgenden werden verschiedene mögliche Datenstrukturen zum Speichern einer solchen Historie diskutiert, wobei zunächst die Anforderungen an diese beschrieben werden.

5.1 Anforderungen

Die Datenstruktur zum Speichern der Auftragshistorie muss verschiedene Anforderungen erfüllen. Dazu gehört zum einen eine effiziente Speicherung, damit so wenig Speicherplatz wie möglich benötigt wird. Das bedeutet unter anderem, dass keine Daten redundant gespeichert werden sollten.

Zum anderen sollen Suchanfragen schnell bearbeitet werden können. Die Performance der Anfragen ist wichtig, um die Wartezeit auch bei ggf. komplexeren Suchen oder Suchen, die viele Ergebnisse liefern, möglichst gering zu halten.

Des Weiteren sollen in der Auftragshistorie Erweiterungen leicht integriert werden können. Wenn beispielsweise bei Änderungsmeldungen ein weiterer Eintrag gemacht werden kann, soll dieses Datenfeld ohne großen Aufwand auch in die Auftragsdaten in der Historie übernommen werden können.

Diese drei Anforderungen werden als Bewertungskriterien für die im nächsten Kapitel vorgestellten verschiedenen Datenbankstrukturen verwendet.

5.2 Mögliche Datenbankstrukturen

Zum Speichern der Auftragshistorie sind verschiedene Datenbankstrukturen denkbar. Diese müssen die drei Auftragsarten Anmeldung, Änderung und Löschung unterstützen. Da die zu speichernden Daten jeweils unterschiedlich sind, sollte für jede Art eine eigene Instanz in der Datenbank genutzt werden.

Im Vergleich zu Anmeldung und Löschung, gestaltet sich das Speichern einer Änderungsmeldung komplexer, da viele Datenfelder dabei optional sind und verschiedene Möglichkeiten zur Auswahl stehen, die entsprechenden Daten zu speichern. Eine Möglichkeit besteht darin, nach jeder Änderung einen Snapshot des Eintrags in der JuNet-Datenbank zu dem entsprechenden Gerät zu speichern. Da in einer Änderungsmeldung jedoch meist nur wenige Datenfelder aktualisiert werden und die anderen dementsprechend vom vorigen Snapshot übernommen werden, würden dabei viele Daten redundant gespeichert und der Speicherplatz somit unnötig erhöht werden. Aus diesem Grund werden statt ganzer Snapshots lediglich die Werte zu den Datenfeldern, die geändert wurden, gespeichert. Um jedoch die ganze Änderung als Auftrag zu speichern, soll dabei zu jedem Datenfeld sowohl der neue als auch der vorige Wert gespeichert werden.

Da Anmeldungen und Löschungen somit geringere Anforderungen an eine Datenbankstruktur stellen, ist für die Entscheidung für oder gegen eine solche die Bewertung in Bezug auf das Speichern der Änderungsmeldungen ausschlaggebend. Anmeldungen und Löschungen können dann in jedem Fall auch in dieser Struktur gespeichert werden.

Im Folgenden werden die verschiedenen möglichen Datenbankstrukturen am Beispiel der Änderungsmeldungen erläutert und ihre Vor- und Nachteile beleuchtet. Dabei werden sowohl relationale als auch dokumentenorientierte Ansätze betrachtet.

5.2.1 Relationale Datenbanken

Relationale Datenbanken bestehen aus einer Sammlung von Tabellen mit festgelegten Spaltennamen und -typen. Jede Tabellenzeile entspricht dabei einem Datensatz. Beziehungen zwischen den Tabellen können über Fremdschlüssel dargestellt werden. Im Folgenden werden zwei verschiedene mögliche Tabellenstrukturen zum Speichern der Auftragshistorie in einer relationalen Datenbank vorgestellt und bewertet.

Zusammengefasst in einer Tabelle

Eine Möglichkeit besteht darin, eine große Tabelle anzulegen, welche pro Zeile alle Informationen zu einem Auftrag enthält. In Tabelle 5.1 ist eine entsprechende Tabellenstruktur zum Speichern der Änderungsmeldungen aufgezeigt. Sowohl die Metadaten zu den Aufträgen, wie Antragsteller, Erstellungsdatum und Host-ID, als auch die Änderungsdaten sind hier gespeichert. Dabei gibt es für jedes Datenfeld zwei Spalten, jeweils eine für den alten und eine für den neuen Wert.

Tabelle 5.1: Tabellenstruktur einer großen Tabelle mit allen Auftragsdaten

Spaltenname	**Datentyp**	**Eigenschaft**
Auftrags_ID	integer	Primärschlüssel
Auftraggeber	varchar	Pflicht
Host_ID	varchar	Pflicht
Datum	date	Pflicht
Abgeschlossen	bool	Pflicht
HWA_alt	varchar	optional
HWA_neu	varchar	optional
Geb_alt	varchar	optional
Geb_neu	varchar	optional
...	...	...

Der Vorteil dieser Struktur besteht darin, dass alle Informationen zu einem Auftrag in einer Tabelle gespeichert sind und somit für entsprechende Suchanfragen nicht zuvor mehrere Tabellen miteinander verknüpft werden müssen. Aus diesem Grund wären bei diesem Ansatz die Suchanfragen recht performant. Ein Nachteil ist allerdings, dass die Tabelle viele NULL-Werte enthalten wird, da bei den meisten Änderungsaufträgen nur wenige Daten geändert und somit wenige Felder ausgefüllt werden. Dies führt zu einem höheren Speicherplatzbedarf als notwendig wäre. Negativ anzumerken ist außerdem, dass Erweiterungen um neue Datenfelder die gesamte Tabelle und damit auch bereits eingetragene Datensätze beeinflussen. Es müssen zwei neue Spalten, jeweils eine für den alten und neuen Wert, eingefügt und die Werte in diesen bei allen bisher existierenden Zeilen auf NULL gesetzt werden.

Aufgeteilt auf mehrere Tabellen

Eine andere Möglichkeit ist die Aufteilung der Informationen zu einem Auftrag auf mehrere Tabellen. Sinnvoll wäre dabei die Gliederung in eine Metadaten-Tabelle und je eine Tabelle pro änderbarem Datenfeld. Entsprechende Tabellenstrukturen sind in Tabelle 5.2 zu sehen. Als Beispiel für die Tabellen zum Speichern der Änderungsdaten wurde hier das Datenfeld der Hardware-Adresse verwendet. Über den Primär- bzw. Fremdschlüssel Auftrags-ID können die Tabellen dann miteinander verknüpft werden.

Tabelle 5.2: Struktur der Metadaten-Tabelle und einer Datenfeld-Tabelle am Beispiel der Hardware-Adresse

Metadaten		
Spaltenname	**Datentyp**	**Eigenschaft**
Auftrags_ID	integer	Primärschlüssel
Auftraggeber	varchar	Pflicht
Host_ID	varchar	Pflicht
Datum	date	Pflicht
Abgeschlossen	bool	Pflicht
Hardware-Adresse		
Spaltenname	**Datentyp**	**Eigenschaft**
Auftrags_ID	integer	Primärschlüssel, Fremdschlüssel
HWA_alt	varchar	Pflicht
HWA_neu	varchar	Pflicht

Diese Struktur behebt die Kritikpunkte der vorigen: Es wird weniger Speicherplatz benötigt, da keine NULL-Werte gespeichert werden und Erweiterungen sind leichter einzupflegen. Soll ein neues Datenfeld hinzugefügt werden, muss eine neue Tabelle erstellt werden. Die bereits gespeicherten Aufträge brauchen allerdings keinen Eintrag in dieser und bleiben damit von der Änderung unberührt. Nachteil dieser aufgeteilten Struktur ist jedoch, dass für die meisten Suchanfragen zunächst ein Join der Einzeltabellen ausgeführt werden muss, wodurch die Performance der Anfragen schlechter ist.

Tabelle für Zustandsänderungen

Unabhängig von der Wahl einer der oben beschriebenen Tabellenstrukturen, ist es sinnvoll zusätzlich zu Meta- und Auftragsdaten den Zustandsverlauf eines Auftrags zu speichern. Darin sollte zu jeder Zustandsänderung der resultierende Zustand, der Eintrittszeitpunkt und gegebenenfalls ein Kommentar gespeichert werden können. Die Kommentarspalte wird dabei primär beim Zustand “WAITING” zum Eintragen des Grundes für die Notwendigkeit eines manuellen Eingriffs verwendet. Da für jeden Auftrag mehrere Zustandsänderungen gespeichert werden müssen, ist es sinnvoll diese Daten in eine eigene Tabelle auszulagern. Eine entsprechende Tabellenstruktur ist in Tabelle 5.3 aufgezeigt. Über den Fremdschlüssel Auftrags-ID können die Zustandsänderungen dem zugehörigen Auftrag zugeordnet werden.

Tabelle 5.3: Struktur der Tabelle zum Speichern der Zustandsänderungen eines Auftrags

Spaltenname	**Datentyp**	**Eigenschaft**
Auftrags_ID	integer	Fremdschlüssel
Zeitpunkt	date	Pflicht
Zustand	varchar	Pflicht
Kommentar	varchar	optional

Mithilfe dieser Tabelle können auch die Bearbeitungszeiten der Aufträge als Differenz zwischen den Eintrittszeitpunkten in die Zustände "INSERTED" und "COMPLETED" ermittelt werden. Diese Daten könnten später gegebenenfalls genutzt werden, um Statistiken über die Wartezeiten bis zum Abschluss eines Auftrags zu erstellen.

5.2.2 Dokumentenorientierte Datenbanken

Dokumentenorientierte Datenbanken eignen sich zum Speichern sogenannter semistrukturierter Daten in einzelnen Dokumenten. Während bei relationalen Datenbanken die Datensätze in zuvor definierten Tabellen gespeichert werden und die Informationen zu einem Objekt meist über mehrere Tabellen verteilt sind, sind bei dokumentenorientierten Datenbanken alle Daten eines Objektes in einem Dokument zusammengefasst. Das Schema der Dokumente kann dabei variieren, sodass nebeneinander verschiedene Typen von Objekten gespeichert werden können. Jedes einzelne Dokument hat einen eindeutigen Identifikator und kann darüber oder über den Inhalt des Dokumentes referenziert werden.

MongoDB als Implementierung

Ein weitverbreitetes dokumentenorientiertes Datenbankmanagementsystem ist MongoDB [9, 10]. Im Vergleich zu anderen dokumentenorientierten Datenbankmanagementsystemen, wie RavenDB oder CouchDB, ist die Abfragesprache bei MongoDB ähnlich zu SQL, wodurch der Einarbeitungsaufwand reduziert wird. Außerdem ist MongoDB aufgrund der hohen Verbreitung gut dokumentiert und es stehen viele Beispiele zur Verfügung.

Bei MongoDB werden die Dokumente als BSON-Dateien abgespeichert. Das BSON-Format basiert auf JSON, speichert aber im Gegensatz dazu die Dateien binär ab und stellt zusätzliche Datentypen zur Verfügung. Ein BSON-Dokument enthält

eine Sammlung von Datenfeldern in Form von Schlüssel-Wert-Paaren. Während die Schlüssel immer vom Typ String sind, können die Werte verschiedene Datentypen annehmen. Neben Standard-Datentypen wie Integer, String und Double können auch komplexere Strukturen wie Arrays und verschachtelte Dokumente genutzt werden. Jedes Dokument hat neben den benutzerdefinierten Datenfeldern ein "_id"-Feld, welches als eindeutiger Identifikator fungiert. Wird beim Hinzufügen eines Dokumentes dieses Feld nicht angegeben, wird automatisch ein Wert dafür gesetzt.

In Listing 5.1 ist ein Beispiel-Dokument zu sehen. Das Objekt besteht neben der ID aus der Item-Bezeichnung, einer Auflistung zugehöriger Kleidungskategorien und einer Verfügbarkeitsliste. Diese Liste ist selbst wieder ein Dokument mit den Kleidergrößen als Schlüssel und der verfügbaren Stückzahl als Wert.

Listing 5.1: Beispiel-Dokument bei MongoDB

```
{ _id: 1234
  item: "shorts",
  category: ["informal", "casual"],
  quantity: {S: 19, M: 15, L: 23}
}
```

Mehrere Dokumente werden in MongoDB zu einer Collection zusammengefasst. Sie stellt das Analogon zu Tabellen bei relationalen Datenbanken dar. Die Dokumente einer Collection müssen allerdings nicht dasselbe Schema haben, d.h. jedes einzelne kann unterschiedliche Datenfelder enthalten.

Die Abfragesprache bei MongoDB basiert ebenfalls auf der Dokumentenstruktur, die zur Definition der Datensätze verwendet wird. Die Filter zum Suchen nach bestimmten Dokumenten in einer Collection werden im BSON-Format geschrieben. Um damit auch verschiedene Operationen, wie beispielsweise Größenvergleiche darstellen zu können, gibt es entsprechende vordefinierte Operatoren. Diese sind mit einem einleitenden "$" gekennzeichnet. In einem zweiten Parameter der Abfragefunktion können die auszugebenden Datenfelder eingeschränkt werden.

In Listing 5.2 ist eine Beispiel-Anfrage einer MongoDB und die entsprechende Abfrage in SQL zu sehen. Gesucht werden hier Items der Kategorie "casual", die mindestens viermal in Größe S verfügbar sind. Ausgegeben werden die Item-Bezeichnungen in aufsteigender Reihenfolge.

Listing 5.2: Beispielabfrage einer MongoDB und entsprechende SQL-Anfrage

```
db.items.find(
  {category: "casual", quantity.S: {$gt: 4}},
  {item: 1})
```

```
SELECT item
FROM items
WHERE category = "casual" AND quantityS > 4
ORDER BY item ASC
```

Indizierung

Normalerweise müssen zur Bearbeitung einer Anfrage alle Dokumente einer Collection geöffnet und darauf überprüft werden, ob sie dem Anfragefilter entsprechen. Um die Abfrage-Performance zu verbessern, gibt es bei vielen dokumentenorientierten ebenso wie bei relationalen Datenbanken, die Möglichkeit Indizes anzulegen. Mit deren Hilfe kann die Anzahl der zu untersuchenden Dokumente reduziert werden.

Bei MongoDB wird ein Index als B-Baum gespeichert. Dieser enthält die verschiedenen Werte eines Feldes oder mehrerer Felder zusammen mit Referenzen auf die zugehörigen Dokumente in geordneter Form. Durch die Sortierung der Werte können Vergleiche auf Gleichheit oder Bereichsfilter effizient unterstützt werden. Statt alle Dokumente zu durchlaufen und zu prüfen, ob sie der Anfrage entsprechen, muss bei Existenz eines passenden Index lediglich der B-Baum traversiert und die passenden Dokumente dereferenziert werden. Zudem wird eine performante Sortierung der Ergebnisse durch Indizes unterstützt.

Für das ID-Feld existiert standardmäßig ein Index. Zusätzlich dazu können pro Collection bis zu 64 weitere erstellt werden, um die Performance bestimmter Anfragen zu steigern. Die Indizes müssen dabei nicht beim Erstellen der Collection angegeben werden, sondern können auch hinzugefügt werden, wenn sie bereits Dokumente enthält. Außerdem ist es möglich, Indizes jederzeit zu löschen, wenn sie nicht mehr benötigt werden.

Dokumentenvalidierung

Eine Besonderheit dokumentenorientierter Datenbanken ist die Schemafreiheit der Dokumente. Da in der Praxis die Dokumente einer Collection jedoch meist ein ähnliches Schema haben, bietet MongoDB die Möglichkeit der Dokumentenvalidierung. Damit können bestimmte Schemavorgaben, wie Feldernamen und Wertetypen, für die Dokumente einer Collection erstellt werden. Beim Hinzufügen von neuen oder Aktualisieren bestehender Dokumente wird dann geprüft, ob diese den Vorgaben entsprechen. Bereits existierende Dokumente bleiben hingegen von der Validierung unberührt. Außerdem kann eingestellt werden, ob bei einem Schemawiderspruch lediglich eine Warnung ausgegeben werden soll oder es zu einem Fehler kommt und die Aktion so nicht ausgeführt werden kann.

Skalierung

Eine weitere Eigenschaft von MongoDB ist seine hohe horizontale Skalierbarkeit. Mithilfe von “Sharding” können die Datensätze einer Collection gleichmäßig auf mehrere Server, sog. Shards, verteilt werden. Dadurch kann eine Lastverteilung und damit eine höhere Performance erreicht werden.

Die Zuordnung der Datensätze zu einem Shard funktioniert dabei über einen sog. Shard Schlüssel, der aus einem oder mehreren Feldern der Datensätze besteht. Die Dokumente einer Collection werden entsprechend verschiedener Wertebereiche des Schlüssels in ungefähr gleich große Blöcke gegliedert. Diese Blöcke werden dann auf die Shard-Instanzen verteilt. Wird ein Block durch Hinzufügen einiger Dokumente zu groß, sorgt die automatische Ausbalancierung von MongoDB für eine Fragmentierung und eventuelle Verschiebung eines der Fragmente auf einen anderen Shard.

Zusätzlich zu den Shard-Instanzen benötigt MongoDB eine zentrale Instanz mit den zugehörigen Konfigurations- und Verwaltungsdaten. Auf dieser wird u.a. gespeichert, welche Datensätze auf den jeweiligen Shards abgelegt sind. Anfragen eines Benutzers werden dann mithilfe der Konfigurationsinstanz an die betroffenen Shards weitergeleitet und die Antworten ggf. zusammengeführt.

Nebenläufige Anfragen

Ein großer Unterschied zwischen relationalen und dokumentenorientierten Datenbanken ist ihr Umgang mit nebenläufigen Anfragen. Während die relationalen dazu das Transaktionskonzept mit den ACID-Eigenschaften[1] einsetzen, wird bei den dokumentenorientierten meist nur eine schwächere Form hiervon implementiert.

Bei MongoDB sind Schreib- und Leseoperationen lediglich auf Dokumenten-Ebene atomar und isoliert. Das bedeutet, dass ein einzelnes Dokument niemals in einem inkonsistenten Zustand vorliegen kann, in dem einige Datenfelder geändert wurden, andere aber noch nicht. Andersherum bedeutet es aber auch, dass Operationen, die auf mehreren Dokumenten arbeiten, eben diese Atomarität und Isolation nicht aufweisen. Dadurch kann es vorkommen, dass die Datenbank einen Zustand annimmt, in dem eine Schreiboperation auf einen Teil der Dokumente bereits angewandt wurde, auf einen anderen Teil jedoch noch nicht. Auch Leseoperationen sind in diesem Zustand dann zugelassen. Da Schreiboperationen jedoch meist nur auf einem Objekt arbeiten und die Daten zu einem Objekt bei dokumentenorientierten Datenbanken idealerweise in einem Dokument gespeichert werden, reicht diese Form der Isolation in den meisten Fällen aus.

[1] **A**tomicity, **C**onsistency, **I**solation, **D**urability [11]

Auftragshistorie in MongoDB

Wenn die dokumentenorientierte Datenbank MongoDB zum Speichern der Auftragshistorie eingesetzt würde, könnten die entsprechenden Dokumente zum Speichern einer Änderungsmeldung wie in Listing 5.3 aufgebaut sein. Während die Metadaten des Auftrags dabei in je einem Feld gespeichert werden, werden die Änderungsdaten und der Zustandsverlauf jeweils in untergeordnete Dokumente ausgelagert. Dabei hat jedes Änderungsdatum den Namen des Datenfeldes als Schlüssel und ein Array bestehend aus altem und neuem Wert als Wert. Die Zustandsänderungen werden in einem Array aus Dokumenten mit jeweils einem Feld für den erreichten Zustand, das Eintrittsdatum und ggf. einen Kommentar gespeichert.

Listing 5.3: Beispiel-Dokument zum Speichern eines Auftrags

```
{ _id: 1234,
  host_id: "zam2212.zam.kfa-juelich.de",
  auftraggeber_id: 234,
  abgeschlossen: true,
  daten: {geb: ["16.3", "16.4"],
          raum: ["203", "333"]}
  zustaende: [{zustand: "INSERTED", zeitpunkt: ISODate
               ("2017-06-01T12:44:45.560Z")},
             {zustand: "WAITING", zeitpunkt: ISODate
               ("2017-06-01T12:45:23.627Z"), kommentar:
               "Nicht Administrator"},
             {zustand: "COMPLETED", zeitpunkt: ISODate
               ("2017-06-01T15:12:38.201")}]
}
```

Da der grobe Aufbau, d.h. die Datenfelder auf der ersten Ebene, bei allen Dokumenten gleich sein sollen, ist die Nutzung einer Dokumentenvalidierung sinnvoll. Durch eine solche kann garantiert werden, dass jedes Dokument die benötigen Metadaten, Änderungsdaten und Informationen zu Zustandsänderungen enthält und die gleichen Feldnamen und Datentypen zur Angabe dieser verwendet.

Die Verwendung der Sharding-Funktion, d.h. die Verteilung der Daten auf mehrere Server, ist in diesem Anwendungsfall hingegen zunächst nicht notwendig. Zum einen ist der Gesamtspeicherbedarf der Dokumente nicht extraordinär groß und zum anderen wird es voraussichtlich wenige gleichzeitige Anfragen geben, sodass eine Lastverteilung nicht benötigt wird.

Nebenläufige Anfragen entstehen, wenn gleichzeitig verschiedene Historiensuchen gestartet oder mehrere Aufträge bearbeitet werden. Besonders zu beachten sind

hierbei die Schreiboperationen nach der Bearbeitung von Aufträgen, die den Auftrag der Historie hinzufügen bzw. eine weitere Zustandsänderung zu diesem vermerken. Da diese Schreiboperationen allerdings nur auf einem Dokument entsprechend der Auftrags-ID arbeiten, reicht hier die Isolation von MongoDB aus, um einen konsistenten Zustand der Datenbank versichern zu können.

Ein Vorteil dieser Implementierung ist, dass die Anfrageperformance hoch ist, da alle Informationen zu einem Auftrag in einem Dokument gespeichert und somit keine Verknüpfungen solcher nötig sind. Außerdem kann die Performance typischer Anfragen durch passende Indizes noch gesteigert werden. Der wohl größte Vorteil des Speicherns der Auftragshistorie in einer dokumentenorientierten Datenbank ist, dass Erweiterungen leicht integriert werden können. Da für die Dokumente kein Schema vorher festgelegt werden muss, können neue Datenfelder einfach beim Hinzufügen eines Auftrags eingefügt werden. Dennoch kann durch die Dokumentenvalidierung ein starkes Abweichen vom grundsätzlichen Aufbau eines Objektes verhindert werden. Da diese Validierung die bereits existierenden Dokumente jedoch nicht betrifft, werden damit die Erweiterungsmöglichkeiten nicht eingeschränkt.

Im Vergleich zur Implementierung der Auftragshistorie mithilfe einer relationalen Datenbank hat dieser Ansatz jedoch auch Nachteile. Während bei der Nutzung einer relationalen Datenbank auf die bereits verwendete zentrale Oracle Datenbank zurückgegriffen werden kann, muss für die dokumentenorientierte Datenbank eine neue MongoDB aufgesetzt werden, wodurch ein erhöhter Konfigurations- und Verwaltungsaufwand entsteht. Außerdem ergibt sich durch die Denormalisierung der Daten und den daraus entstehenden Redundanzen ein höherer Speicherbedarf. Dieser ist zwar zur Steigerung der Anfrageperformance beabsichtigt, muss hier aber dennoch als Nachteil aufgeführt werden.

5.3 Diskussion und Umsetzung

In Tabelle 5.4 sind die Vor- und Nachteile der verschiedenen vorgestellten Datenbankstrukturen im Hinblick auf die in Abschnitt 5.1 genannten Bewertungskriterien noch einmal zusammengefasst dargestellt.

Es lässt sich schnell erkennen, dass bei dem Ansatz einer relationalen Datenbank mit einer großen Tabelle die Nachteile überwiegen. Die anderen beiden Ansätze sind hingegen relativ gleichgestellt. Bei beiden Ansätzen sind Erweiterungen leicht zu integrieren. Während bei der Implementierung mit einer relationalen Datenbank mit mehreren Tabellen die Anfrageperformance unter den notwendigen Tabellen-

Tabelle 5.4: Zusammengefasste Bewertung der verschiedenen Datenbankstrukturen

DB-Struktur	**Speicherbedarf**	**Anfrage-performance**	**Erweiterbarkeit**
Relational, eine Tabelle	-	+	-
Relational, mehrere Tabellen	+	-	+
Dokumenten-orientiert	-	+	+

verknüpfungen leidet, bedingt die Denormalisierung der Daten bei der dokumentenorientierten Datenbank einen erhöhten Speicherbedarf.

Da die Anfrageperformance für die Auftragshistorie eine wichtigere Rolle spielt und der Speicherbedarf bei Nutzung einer dokumentenorientierten Datenbank lediglich geringfügig höher ist, ist dieser Ansatz zu bevorzugen, sofern die Bereitschaft zum Konfigurieren und Verwalten einer neuen Datenbank gegeben ist.

Für die Implementierung kann der Perl Driver, der von MongoDB zur Verfügung gestellt wird, genutzt werden [12]. Über verschiedene Funktionen können damit unter anderem Collections erstellt, Dokumente hinzugefügt, geändert oder gelöscht und Indizes und Dokumentenvalidierungen definiert werden. Die einzelnen Dokumente werden dabei als Hash-Objekte dargestellt.

In Listing 5.4 ist das Hinzufügen eines Auftrags in die Collection und einer Zustandsänderung zum Auftrag mithilfe des Perl Drivers aufgezeigt.

Listing 5.4: Hinzufügen eines Auftrags und einer Zustandsänderung

```
$collection->insert_one(
{ _id => 1234,
  host_id => "zam2212.zam.kfa-juelich.de",
  auftraggeber_id => 234,
  abgeschlossen => false,
  daten => {geb => ["16.3", "16.4"],
            raum => ["203", "333"]}
  zustaende => [{zustand => "INSERTED", zeitpunkt => DateTime
                  ->now}]
})
```

```
$collection->update_one(
  {_id => 1234},
  {'$push' => {zustaende => {zustand => "COMPLETED", ↩
                    ↪zeitpunkt => DateTime->now}}},
  {'$set' => {abgeschlossen => true}}
)

# resultierende Datenstruktur:
{ _id => 1234,
  [...]
  abgeschlossen => true,
  zustaende => [{zustand => "INSERTED", zeitpunkt => ISODate↩
                ↪("2017-06-01T12:44:45.560Z")},
               {zustand => "COMPLETED", zeitpunkt => ISODate↩
                ↪("2017-06-01T12:46:15.698Z")}]
}
```

Wie oben beschrieben ist es sinnvoll eine Dokumentenvalidierung für die einzelnen Aufträge zu definieren. Ein entsprechender Funktionsaufruf inklusive Schemadefinition ist in Listing 5.5 zu sehen.

Listing 5.5: Definition einer Dokumentenvalidierung

```
$db->run_command({
  collMod => "auftraege",
  validator => {host_id => {$type => "string"},
               auftraggeber_id => {$type => "int"},
               abgeschlossen => {$type => "bool"},
               daten => {$type => "doc"},
               zustaende => {$type => "array"}},
  validationAction => "err"
})
```

Des Weiteren sollte die Möglichkeit genutzt werden, die Anfrageperformance über Indizes zu verbessern. Dabei müssen allerdings auch ihre Nachteile beachtet werden. Zum einen erhöhen Indizes den Speicherbedarf der Datenbank. Zum anderen wird zwar die Performance der Anfragen durch sie gesteigert, die Performance beim Einfügen neuer bzw. beim Ändern bestehender Datensätze hingegen verringert. Dies liegt darin begründet, dass neue Dokumente gegebenenfalls einem Index hinzugefügt und aktualisierte Dokumente eventuell im Index verschoben werden müssen. Aus diesen Gründen sollte vor der Erstellung von Indizes analysiert werden, welche Suchanfragen typisch für die Anwendung sind und nur solche Indizes erstellt werden, die eben diese deutlich beschleunigen.

Bei der Auftragshistorie ist ein Index für das Feld “auftraggeber_id“ sinnvoll, da Benutzer sich vermutlich meist die von ihnen aufgegebenen Aufträge anzeigen lassen und solche Anfragen damit beschleunigt werden können. Zu erwarten sind außerdem häufige Suchen nach den Aufträgen zu einem bestimmten Gerät. Um diese schnell bearbeiten zu können, bietet sich ein Index für das Feld ”host_id“ an.

Aufgrund der oben beschriebenen Nachteile von Indizes sollte es zunächst bei diesen beiden belassen werden. Wenn das System längere Zeit in Betrieb ist, können Statistiken zu typischen Anfragen erstellt und auf deren Grundlage ggf. neue Indizes hinzugefügt werden.

6 Zusammenfassung und Ausblick

Das in dieser Arbeit entwickelte Konzept für ein teilautonomes Auftragssystem zur Verwaltung und Bearbeitung von Gerätean-, -um- und -abmeldungen im JuNet ist in der Lage das in Kapitel 2 beschriebene bisherige System zu ersetzen und erfüllt dabei die dort aufgeführten Anforderungen.

Die Autorisierung wurde als attributbasierte Zugriffskontrolle eingeführt (siehe Kapitel 3). Verschiedene Zugriffsregeln, Benutzerrollen und -attribute können in Form von erweiterbaren Konfigurationsdaten definiert werden. Ein selbstentwickeltes Perl-Modul nutzt diese Daten dann, um eine Entscheidung über die Berechtigung einer Anfrage zu treffen. Die Architektur des Autorisierungsmodul ist dabei dem XACML Standard nachempfunden.

Zur automatischen Bearbeitung von Aufträgen wurde eine Ablaufsteuerung in Form eines Zustandsautomaten konzipiert (siehe Kapitel 4). Ein neuer Auftrag muss dabei zunächst auf Korrektheit überprüft werden, wobei die geltenden Regeln hierzu in einer Config-Datei angegeben werden können. Abhängig vom Ergebnis der Überprüfungen wird der Auftrag dann umgesetzt, d.h. die Gerätedatenbank entsprechend angepasst, aufgrund eines Fehlers abgebrochen oder zwecks manuellen Eingriffs in einen wartenden Zustand versetzt. Ist ein manueller Eingriff erforderlich, haben die Mitarbeiter des JuNet-IP-Managements verschiedene Möglichkeiten darauf zu reagieren. Sie können den Auftrag akzeptieren, bearbeiten oder ablehnen. Während des Bearbeitungsprozesses eines Auftrags bleiben Auftraggeber und Mitarbeiter des JuNet-IP-Managements durch entsprechende Benachrichtigungen stets über seinen aktuellen Zustand informiert.

Als Datenbankstruktur für die neue Auftragshistorie wurde ein dokumentenorientierter Ansatz mit MongoDB gewählt (siehe Kapitel 5). Die Daten zu jeder Anmeldung, Änderung und Löschung eines Gerätes werden dabei in einzelnen BSON-Dokumenten grob definierter Form gespeichert. Die Vorteile dieses Konzepts sind die hohe Anfrageperformance und gute Erweiterbarkeit.

Den nächsten Schritt bildet nun die Implementierung der entwickelten Konzepte. Dabei sollte zunächst nur die Bearbeitung einer einfachen Auftragsart, wie beispielsweise die Änderung einer Hardware-Adresse, automatisiert werden und in allen anderen Fällen ein manueller Eingriff erforderlich sein. Nach ausgiebigen internen Tests

kann das neue System dann das bisherige ablösen und schrittweise um weitere automatisierte Vorgänge erweitert werden. Dazu müssen dann lediglich die Konfigurationsdaten der einzelnen Module angepasst werden; Änderungen am Programmcode sind nicht mehr notwendig.

Ausblickend sind verschiedene Erweiterungen für das System denkbar. Um die Anpassung der Konfigurationsdaten für die Autorisierung und Ablaufsteuerung benutzerfreundlicher zu gestalten, könnte hierfür jeweils eine grafische Oberfläche entwickelt werden. Über diese könnten dann die aktuellen Einstellungen eingesehen und Änderungen an diesen vorgenommen werden.

Eine weitere mögliche zukünftige Funktion stellt die Durchführung eines Rollbacks von Aufträgen dar. Da zu den Änderungsmeldungen in der Auftragshistorie auch die ehemaligen Werte der geänderten Datenfelder gespeichert werden, könnten einzelne Aufträge leicht rückgängig gemacht werden. Eine Schwierigkeit hierbei würden jedoch Abhängigkeiten zwischen verschiedenen Aufträgen darstellen. Es ist beispielsweise möglich, dass die ehemalige IP-Adresse eines Gerätes, die dann wieder angenommen werden soll, in der Zwischenzeit bereits neu vergeben wurde. Vor der Implementierung einer Rollback-Funktion müsste also geklärt werden, wie auf solche Fälle reagiert werden soll.

Mithilfe der Auftragshistorie ist es möglich, unter Beachtung der gesetzlichen Vorgaben, Statistiken zur Bearbeitungszeit der Aufträge und zur Häufigkeit verschiedener Fehler und Warnungen zu erstellen. Ihre Ergebnisse könnten dabei helfen das System benutzerfreundlicher zu gestalten und die Mitarbeiter des JuNet-IP-Managements durch weitere Automatisierungen zu entlasten.

Insgesamt können Erweiterungen und Anpassungen durch das modulare Design und die Auslagerung verschiedener Konfigurationsdaten relativ leicht in das Auftragssystem integriert werden, sodass es für zukünftige Änderungen und neue Anforderungen gerüstet ist.

A. Abbildungsverzeichnis

2.1 ER-Diagramm der zugrunde liegenden Datenstrukturen 4

3.1 XACML Architektur und Verlauf einer Autorisierungsanfrage [5] . . 16
3.2 Aufbau des Autorisierungsmodul und Kommunikationsablauf mit anderen Komponenten 19

4.1 Beispiel eines Zustandsautomaten 23
4.2 Zustandsautomat des globalen Workflows zur Bearbeitung eines Auftrags 25

B. Tabellenverzeichnis

2.1 Beschreibung der Tabelle *d_pers* 4
2.2 Beschreibung der Tabelle *x_device* 5
2.3 Beschreibung der Tabelle *x_port* 6
2.4 Beschreibung der Tabelle *x_link* 6
2.5 Beschreibung der Tabelle *n_devname* 7
2.6 Beschreibung der Tabelle *n_netname* 7

5.1 Tabellenstruktur einer großen Tabelle mit allen Auftragsdaten 33
5.2 Struktur der Metadaten-Tabelle und einer Datenfeld-Tabelle am Beispiel der Hardware-Adresse 34
5.3 Struktur der Tabelle zum Speichern der Zustandsänderungen eines Auftrags 35
5.4 Zusammengefasste Bewertung der verschiedenen Datenbankstrukturen 41

C. Listings

3.1 Beispiel einer XACML Regel [6] 17

4.1 Beispiel eines regelbasierten Systems 22
4.2 Implementierung des Zustandsautomaten mithilfe des workflow-Moduls 26
4.3 Anwendung des workflow-Moduls 27
4.4 Beispiel einer Config-Datei zur Angabe der Prüffunktionen 28
4.5 Ermitteln der anzuwendenden Regeln mithilfe des Moduls `Checks` . 29

5.1 Beispiel-Dokument bei MongoDB 36
5.2 Beispielabfrage einer MongoDB und entsprechende SQL-Anfrage . . 36
5.3 Beispiel-Dokument zum Speichern eines Auftrags 39
5.4 Hinzufügen eines Auftrags und einer Zustandsänderung 41
5.5 Definition einer Dokumentenvalidierung 42

Literatur

[1] *Forschungszentrum Jülich - Über uns.* URL: http://www.fz-juelich.de/portal/DE/UeberUns/_node.html (besucht am 07/2017).

[2] Ellen Hentschel. „Ein Verwaltungssystem für Netzobjekte im JuNet auf Basis von Web Services“. Diplomarbeit. Fachhochschule Aachen, 25. Feb. 2008.

[3] Vincent C. Hu et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations.* National Institute of Standards and Technology (NIST). Jan. 2014. URL: http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-162.pdf (besucht am 07/2017).

[4] *eXtensible Access Control Markup Language (XACML) Version 3.0.* OASIS Standard. 22. Jan. 2013. URL: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (besucht am 07/2017).

[5] *XACML Reference Architecture.* Axiomatics. URL: https://www.axiomatics.com/blog/xacml-reference-architecture/ (besucht am 07/2017).

[6] *XACML.* Wikipedia, The Free Encyclopedia. URL: https://en.wikipedia.org/w/index.php?title=XACML&oldid=793014393 (besucht am 07/2017).

[7] Kostas Stergiou. *Knowledge-Engineering & Knowledge-Based Systems.* Lecture 6: Rule-based Systems by John Platt. University of the Aegean. URL: https://www.icsd.aegean.gr/lecturers/konsterg/teaching/KE/KE.html (besucht am 07/2017).

[8] Chris Winters; Jonas B. Nielsen et al. *Workflow - Simple, flexible system to implement workflows.* Comprehensive Perl Archive Network (CPAN), 2004-2017. URL: https://metacpan.org/pod/Workflow (besucht am 07/2017).

[9] *The MongoDB 3.4 Manual.* MongoDB, Inc., 2008 - 2017. URL: https://docs.mongodb.com/manual/ (besucht am 07/2017).

[10] *DB-Engines Ranking von Document Stores.* 2017. URL: https://db-engines.com/de/ranking/document+store (besucht am 07/2017).

[11] Shefali Naik. *Concepts of Database Managment System.* Pearson India, 1. Apr. 2013. Kap. 8. URL: http://proquest.techbus.safaribooksonline.de/book/databases/9789332537422 (besucht am 07/2017).

[12] David Golden et al. *MongoDB - Official MongoDB Driver for Perl*. Comprehensive Perl Archive Network (CPAN), 2017. URL: https://metacpan.org/pod/MongoDB (besucht am 07/2017).