Studienarbeit

SAN Management Appliance

Tobias Schoknecht

May 31, 2012

Kurzreferat

Dieses Dokument befasst sich mit Entwicklung und Dokumentation einer SAN[1] Management Appliance im Zuge einer Studienarbeit. Dies beinhaltet die Konzeption und Entwurfsentscheidungen bzgl. der Architektur und der verwendeten Technologien.

Die Intention der Arbeit ist es eine Administrationsoberfläche für SANs zu erstellen, mit denen sich diese benutzerfreundlich und ohne viel Fachwissen verwalten lassen - unabhängig von den kostenpflichtigen Tools der Hardwarehersteller.

Abstract

This document is about the development and documentation of a SAN Management Appliance in relation with a mandatory project at the DHBW Stuttgart. It contains the conception and design regarding the architecture and the technologies used.

The intention of this project is to create an end-user friendly administration interface for SANs with which it is easy to administer a SAN without much know how - independent of the tools with costs of the hardware manufacturers.

[1] Storage Area Network

Tobias Schoknecht
Matrikel Nr. 0655334; Kurs TIT09AIB
Jägerstraße 55
70174 Stuttgart
Mail: tobias.schoknecht@gmail.com

Betreuer des Projektes:
Mario Lombardo
tocario GmbH
Heilbronner Straße 7
70174 Stuttgart
Mail: ml@tocario.com

Contents

List of Tables

List of Figures

List of abbreviations

CLI	Commandline Interface
GUI	Graphical User Interface
HTML	Hypertext Markup Language
RVM	Ruby Version Manager
SAN	Storage Area Network
SMA	SAN Management Appliance
SSH	Secure Shell
XML	Extensible Markup Language

1 Einleitung

1.1 Aufgabenbeschreibung

Das Ziel des Projektes ist die Erstellung eines webbasierten (HTML5) Administrationstools für SANs. Dies beinhaltet zum einen das Backend, welches die Anbindung an das SAN via Kommandozeile (SSH oder Telnet) realisiert, sowie das Frontend welches die Schnittstelle zum Benutzer darstellt.

Das Frontend ist hierbei möglichst benutzerfreundlich zu halten, um auch Nutzern mit wenig Fachwissen bezüglich SANs die Verwendung zu ermöglichen.

Das Backend soll durch Plugins erweiterbar gehalten werden, um die Oberfläche anpassbar für zur Zeit der Studienarbeit noch nicht existente oder einfach in Ermangelung an Resourcen nicht unterstützte Switches zu halten.

1.2 Einführung zum Thema

Im Zuge der Entwicklung großer Computersysteme und Netze ist die Konsolidierung und Harmonisierung von Hardware und Softwaresystemen unerlässlich. Ein Teil davon ist der Aufbau eines Storage Area Networks zur zentralen Verwaltung des Speichers, um diesen besser ausnutzen zu können, sowie zur einfacheren Nutzung von Daten durch verschiedene Server. Das extra Datennetz, welches unabhängig vom normalen LAN ist, dient hierbei der Entlastung des LANs bzgl. der Festplattendaten, sowie der Performance-Steigerung durch Protokolle mit weniger Overhead.

1.3 Anfänglicher Status

Die SAN Management Appliance (im Folgenden auch als SMA bezeichnet) wird von Grund auf neu entwickelt. Vorgesehen war für die Entwicklung, dass ein kleines SAN mit 2 hp Switches in der DHBW Stuttgart aufgestellt wird. In Folge einiger organisatorischer Schwierigkeiten ist dies jedoch nicht geschehen.

1.4 Geschäftlicher Nutzen

Die SMA stellt eine kostenlose Alternative zu den kostenpflichtigen Administrationstools der Hardwarehersteller dar. Weiterhin ist die Oberfläche bewusst benutzerfreundlich gehalten worden, um auch bei geringem Fachwissen die Administration zu erlauben.

1.5 Struktur des Dokuments

Im Folgenden werden in diesem Dokument Grundlagen zu SANs vermittelt, um ein Verständnis der Aufgabe sowie des Resultats zu ermöglichen. Weiterhin wird auf Design, Entwicklung und bewusst getroffene Entwurfsentscheidungen eingegangen.

2 Storage Area Network

Dieses Kapitel befasst sich kurz mit dem Nutzen und der Funktionsweise von Storage Area Networks, um einen kleinen Überblick zu geben.

2.1 Begriffsdefinition

"Storage Area Networks sind dedizierte Speichernetze, die Server und Speichersysteme über Breitbandnetze wie Fibre Channel miteinander verbinden und gegenseitig entkoppeln."[11]

2.2 Allgemein

Speichernetze dienen dazu den Speicher zu zentralisieren und diesen effizienter auszunutzen. Statt einiger direkt am Server angeschlossener Festplatten, wird der Speicher mehrerer Server zentral an einer Stelle verwaltet. Es kann jedoch nicht nur Festplattenspeicher genutzt werden, auch andere Speichertypen wie Bandspeicher können an das SAN angeschlossen werden.

Sämtliche Kommunikation zwischen Server und Speicher erfolgt dann über ein extra Netzwerk, welches unabhängig vom Local Area Network existiert, um die normale Netzwerkkommunikation nicht zu überlasten.

Dieses Netzwerk operiert im allgemeinen nicht auf TCP/IP sondern über das Fibre Channel Protokoll, welches besseren Durchsatz bei Blocksatzdaten erlaubt.

Die Verwaltung der Speichernetze erfolgt über sogenannte SAN Fabrics. Diese Fabrics sind einzelne oder mehrere verbundene SAN Switches, welche die Verbindungen zwischen Speicher und Server verwalten, d.h. den Zugriff entweder erlauben oder verweigern.

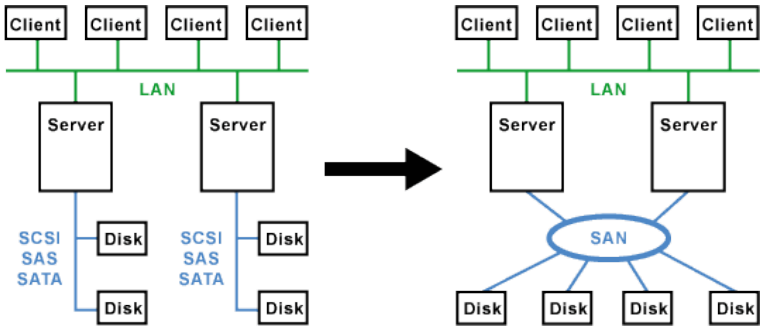

Figure 2.1: SAN[12]

3 Design

Dieses Kapitel befasst sich mit den Ideen für die SMA und deren angedachte Umsetzung.

3.1 Allgemeine Struktur

Die SMA ist in mehrere Komponenten gegliedert.

Zum Einen gibt es das Web Frontend, welches die Benutzerschnittstelle darstellt. Zum Anderen gibt es die Treiberkomponente, die sich um die Kommunikation mit dem SAN Switch kümmert und die Daten aus dem Switch aus liest, sowie neue Daten wieder einpflegt.

3.1.1 Frontend

Das Frontend soll einfach über den Browser ansteuerbar sein, weshalb sich für eine Lösung via Web Application entschieden wurde, bei der das eigentliche Frontend mit HTML5, JavaScript und CSS (unter zur Hilfennahme von `http://960.gs/`) umgesetzt ist. Dies ermöglicht dann ein einfaches Zugreifen via Browser.

3.1.2 Rails App

Für die Umsetzung der Logik wurde das Web Application Framework `Ruby on Rails` ausgewählt.

Die Rails App kümmert sich um das Rendern der Templates und damit der Erzeugung der Oberfläche. Weiterhin regeln weitere Ruby Scripts die Kommunikation mit dem SAN Switch und lesen Daten in die interne Konfigurationsdatenbank ein, sowie setzt Befehle an den Switch ab, um neue Konfiguration zu erzeugen.

Die Logik in der Rails App kümmert sich auch um die Zugriffsberechtigung zur SMA.

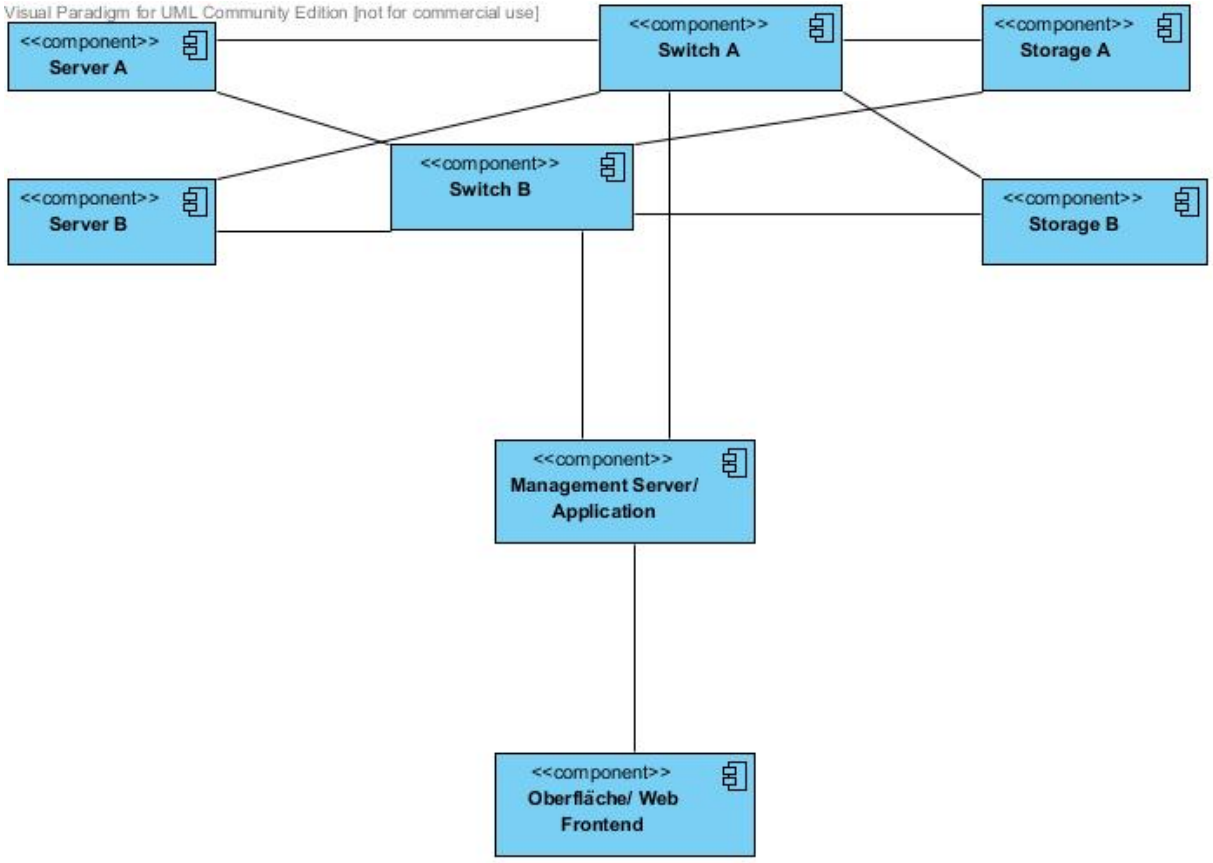

Figure 3.1: SMA Struktur

3.1.3 Switch Treiber

Die Ruby Scripts zur Kommunikation mit dem SAN Switch können auch als Treiber bezeichnet werden, da sie die Befehle zwischen Rails App und SSH Befehlen für den Switch übersetzen.

Die Rails App kann dann auf dem Treiber z.B. den Befehl zum Auflisten aller angeschlossener Server aufrufen, was dafür sorgt, dass via SSH die Befehle zum Auflisten aller Geräte abgesetzt werden, die Rückgabe geparst und in eine für die Rails App verständliche Form gebracht wird.

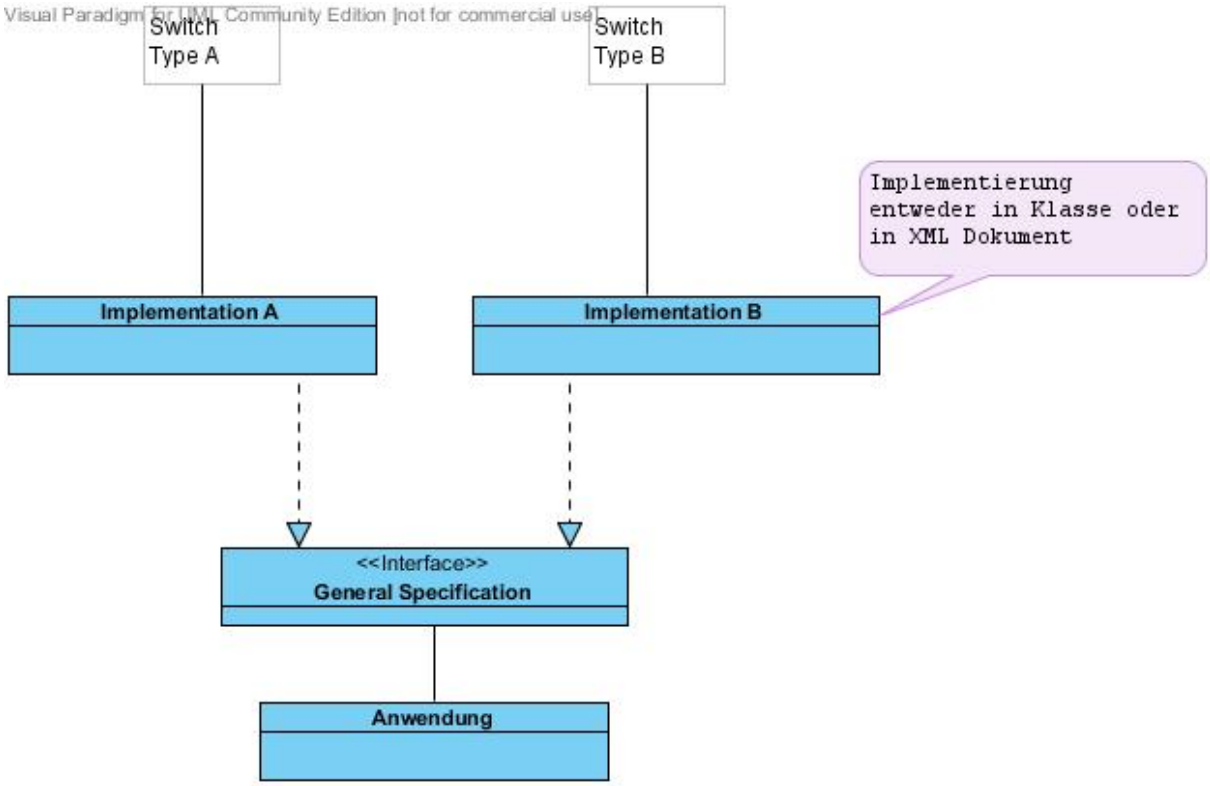

Figure 3.2: Plugin System

4 Implementierung

Dieses Kapitel befasst sich mit der Implementierung der SMA und kann zum Verständnis für die Weiterentwicklung genutzt werden. Ab hier wird ein grundlegendes Verständnis von Ruby on Rails vorausgesetzt.

Informationen zu Ruby können unter folgenden Links gefunden werden:

`http://www.ruby-lang.org/de/`

`http://www.ruby-doc.org/core-1.9.3/`

Für Ruby on Rails bietet sich der nachfolgenden Link an:

`http://rubyonrails.org/`

Für Quellen in Buchform wären die folgenden Bücher zu empfehlen:

`Programming Ruby 1.9: The Pragmatic Programmers' Guide` von Dave Thomas, Chad Fowler & Andy Hunt

`Ruby on Rails 3 Tutorial: Learn Rails by Example` von Michael Hartl

`The Rails 3 Way` von Obie Fernandez

Weiterhin werden für die nächsten Abschnitte fürs Codeverständnis Kenntnisse in jQuery und Processing.js benötigt:

`http://processingjs.org`

`http://jquery.org`

4.1 Datenmodell

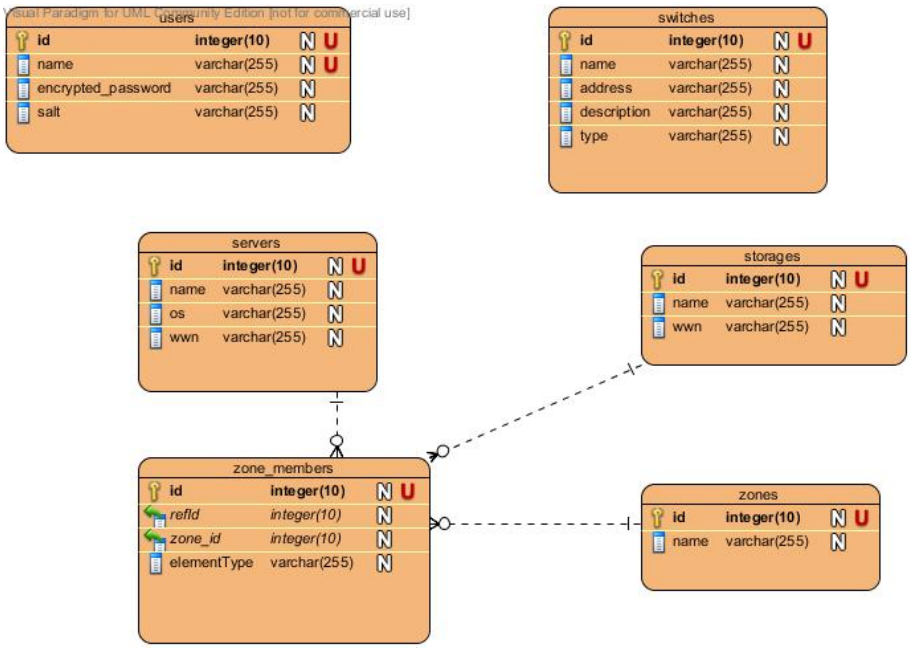

Figure 4.1: Datenmodell

4.2 Zugriffskontrolle

Es existiert ein Benutzer, der beim Ausführen des Seed Scripts (`db/seeds.rb`) angelegt wird. (`rake db:seed RAILS_ENV=production`)

Dieser Benutzer heißt `sanmanager` und das Standardpasswort lautet `application`, welches nach dem Login geändert werden kann.

Es wurde keine Funktionalität in die Oberfläche implementiert, um weitere Benutzer anzulegen oder zu entfernen, wobei es über einfügen in die Datenbank möglich wäre.

4.2.1 Filter

In der Datei `app/controller/application_controller.rb` wurde über den *before_filter* `require_login` der Zugriffsschutz realisiert. Der *before_filter* im Application Controller sorgt dafür, dass dieser auf alle anderen Controller angewendet wird, da es der Meta Controller ist, von dem alle anderen erben.

```
1 class ApplicationController < ActionController::Base
2   protect_from_forgery
3
4   before_filter :require_login
5
```

```
6   private
7
8     def logged_in?
9       !!current_user
10    end
11
12    def require_login #filter to check whether a user is logged in
13      unless logged_in?
14        flash[:error] = "The requested site requires login."
15        redirect_to root_url
16      end
17    end
18
19    def current_user #checks if the current user exists and is set in the session
20      @_current_user ||= session[:current_user] && User.find(session[:current_user])
21    end
22  end
```

Dieser Filter überprüft ob eine Session für einen Benutzer existiert und um welchen Benutzer es sich handelt. Da bei der SMA unter normalen Umständen nur ein Benutzer vorhanden ist, ist der zweite Teil eher weniger wichtig.

Der Login wird über `app/controller/login_controller.rb` realisiert. Hier wird der Filter deaktiviert, um einen Zugriff auf die einzelnen Actions ohne Login zu ermöglichen.

```
1   class LoginController < ApplicationController
2
3     skip_before_filter :require_login, :only => [:create, :destroy] #deactivates controller for the ↩
        ⟶actions create and destroy
4
5     def create
6       #...
7     end
8
9     def destroy
10      #...
11    end
12
13  end
```

Der Login selbst ist relativ simpel: es wird über die `authenticate`-Methode aus dem User-Model der Nutzername mit dem Passwort auf wahr oder falsch geprüft. Im Erfolgsfall wird der Benutzer in der Session gespeichert, was darauf folgend den Filter mit true beantwortet.

Die genaue Funktionsweise im einzelnen kann über `app/controller/application_controller.rb`, `app/controller/login_controller.rb` und `app/model/user.rb` verstanden werden.

4.3 Konfigurationsdaten und Verwaltung

Zum Anzeigen und Verwalten der Konfigurationsdaten wurden mehrere Controller implementiert für Server, Storage, Switch und Zone.

Ein Switch kann manuell hinzugefügt oder entfernt werden mit entsprechenden Anmeldedaten für die SSH-Verbindung, sowie die IP-Adresse oder den DNS-Namen. Weiterhin wird hier der Treiber festgelegt über den dann kommuniziert wird.

Die Daten für Storage und Server sollen über das Auslesen der Daten aus dem Switch hinzugefügt werden, können aber manuell entfernt werden.

Zonen sollen manuell angelegt werden können und dann an den Switch übermittelt werden.

Die Umsetzung selbst ist ziemlich simpel und umfasst bisher nur das Hinzufügen von Daten, das eventuelle Entfernen und die Anzeige dieser Informationen.

Hier ein gekürztes Beispiel zur Veranschaulichung der Actions:

```
1  class SwitchController < ApplicationController #inherites from ApplicationController and therefore ↩
       ⟶gets the before filter
2    def add
3          #action to add a new switch to the database
4    end
5
6    def remove
7          #action to remove a switch from the database
8    end
9
10   def edit
11         #action to show the view for editing a switch
12   end
13
14   def update
15         #action to send the changes to the server/database
16   end
17
18   def new
19         #action to show the template for a new switch
20   end
21
22   def show
23     #action to list all switches currently known to the system
24   end
25
26 end
```

Genaueres kann in den einzelnen Controllern und den dazugehörigen Models nachgeschaut werden.

4.4 Canvas Oberfläche

Zur benutzerfreundlichen Anzeige des SANs (also der Server, Switches, Storages und Verbindungen dazwischen) wurde via processing.js[1] eine Oberfläche in einem HTML5

[1] `http://processingjs.org`

Canvas-Element implementiert. Diese Oberfläche erlaubt die einfache Erzeugung von Verbindungen zwischen Server und Storage über einen bestimmten Switch.

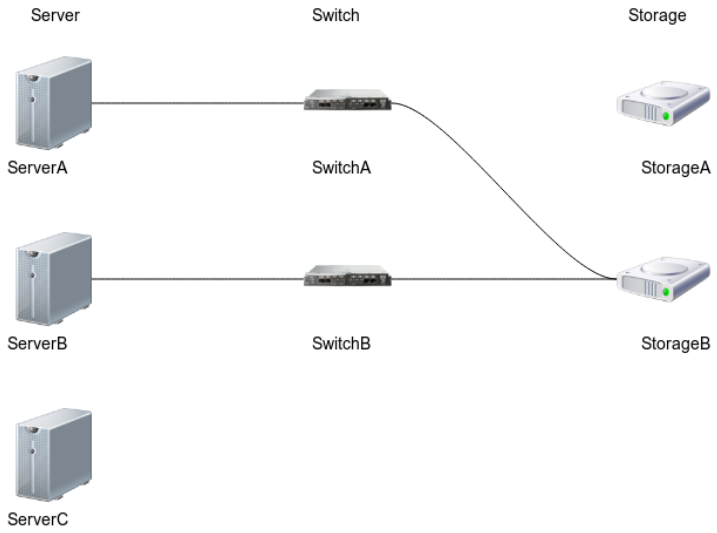

Figure 4.2: Canvas Oberfläche

Genauere Informationen zur Funktionsweise von Processing.js können unter `http://processingjs.org` gefunden werden. Dort sind hinreichend Tutorials und Beispiele vorhanden, um sich schnell in die Funktionsweise einzufinden.

4.4.1 Funktionsweise

Im Folgenden wird auf die Funktionsweise eingegangen.

Die Daten über Server, Storages, Switches und Verbindungen werden mit Hilfe von jQuery[2] über einen Ajax-Call abgerufen.

Die Daten sind im XML-Format gehalten und werden auf der Backend-Seite aus den Config-Daten mittels einer normalen View generiert und dann auf der Frontend-Seite geparst und angezeigt.

Ein Beispiel für die XML-Daten sieht wie folgt aus:

```
1 <model>
2   <serverList>
3     <server id="1">ServerA</server>
4     <server id="2">ServerB</server>
```

[2] `http://jquery.org`

```
5       <server id="3">ServerC</server>
6     </serverList>
7     <storageList>
8       <storage id="1">StorageA</storage>
9       <storage id="2">StorageB</storage>
10    </storageList>
11    <switchList>
12      <switch id="1">SwitchA</switch>
13      <switch id="2">SwitchB</switch>
14    </switchList>
15    <connectionList>
16      <connection>
17        <from>1</from>
18        <over>1</over>
19        <to>2</to>
20      </connection>
21      <connection>
22        <from>2</from>
23        <over>2</over>
24        <to>2</to>
25      </connection>
26    </connectionList>
27  </model>
```

Der Code für dieses Element ist unter `public/js/sanoverview.pde.js` zu finden.

4.4.2 Verbindung anlegen

Neue Verbindungen können durch Klicken auf Server oder Storage, dann auf einen Switch und zuletzt das entsprechende Pendant (Server => Storage, Storage => Server) angelegt werden.

Über die Mausposition wird bestimmt welches Element angeklickt wurde und dann eine temporäre Verbindung zur aktuellen Mausposition gezeichnet. Beim Klick auf den Switch wird die erste Teilverbindung eingezeichnet und vom Switch aus eine Verbindung zur aktuellen Mausposition erstellt.

Wenn man nun auf das letzte Element für die Verbindung klickt, wird die Verbindung in der Oberfläche hinzugefügt. Sollte man irgendwann während dieses Vorgangs nicht auf ein Element klicken, so wird die temporäre Verbindung verworfen.

4.4.3 Verbindung entfernen

Um eine angelegte Verbindung zu entfernen, wählt man diese in der Verbindungsliste unter dem Canvas-Element aus und klickt dann auf `Delete selected`.

4.4.4 Änderungen schicken

Sobald man alle Änderungen durchgeführt hat, so kann man auf `Commit changes` klicken. Dies löst einen Ajax-Call aus, der eine Liste der aktuellen Verbindungen als XML ans Backend sendet. Außerdem wird eine Liste mit entfernten Verbindungen gesendet, um diese aus der Datenbank zu entfernen.

4.5 Treiber

Die Treiberscripts werden in `lib/driver` abgelegt.

Die Scripts stellen eine Verbindung zum Server her, führen dann die Befehle aus und beenden die Verbindung.

Ein Beispiel sieht wie folgt aus:

```
1  require 'net/ssh'
2
3  class DefaultDriver
4          def initialize(username, password, host, port)
5                  @session = Net::SSH.start(host, username, :port => port, :password => password)
6          end
7
8          def getservers
9            result = @session.exec!("nsshow") #example command
10           #other ssh command...
11           #code to parse the result and return it to the server
12         end
13
14         def getstorages
15           #...
16         end
17
18           #...
19
20         def finalize
21                 @session.close
22         end
23 end
```

Über den SAN-Communications-Controller wird der für den Switch passende Treiber ausgewählt (beim Anlegen des Switches konfiguriert), die Methode ausgeführt und dann das passende Resultat verarbeitet.

`WICHTIG:` Der Klassenname und der Dateiname müssen übereinstimmen (case-sensitive), da die Auflistung der Treiber beim Anlegen eines Switches über die Dateinamen und beim Zugriff auf den Switch die Auswahl der Klasse über den gleichen String erfolgt.

5 System setup

Die SAN Management Appliance ist in Ruby on Rails und die Kommunikation mit dem SAN Switch in normalem Ruby realisiert.

Durch die ständige Weiterentwicklung von Ruby sowie Ruby on Rails ist darauf zu achten, dass hierbei die richtigen Versionen der Komponenten zum Einsatz kommen.

Die Installation der SAN Management Appliance ist nicht ganz trivial und wird deshalb im folgenden am Beispiel eines Debian 6 Systems erläutert. Es wird davon ausgegangen, dass sich auf dem System keine ältere Ruby oder Rubygems Version befindet. Weiterhin wird angenommen, dass man sich bereits ein Verzeichnis für die Rails App ausgesucht hat und diese dorthin entpackt hat. Eine Empfehlung wäre `/srv`.

5.1 Apache 2

Für die Auslieferung des Contents kommt hier ein Apache 2 Webserver zum Einsatz. Alternativ wäre der Einsatz von Nginx möglich, der hier nicht näher behandelt wird.

Bash-Script zur Installation:

```
1 #!/bin/bash
2 apt-get install apache2
```

5.2 SQLite3

Zur Speicherung von Konfigurationsinformationen des SANs werden diese in einer SQLite Datenbank abgelegt.

Bash-Script zur Installation:

```
#!/bin/bash
apt-get install libsqlite3-ruby libsqlite-dev libsqlite3-dev
```

5.3 Ruby Installation

Da Debian mittlerweile auch ein Package für Ruby1.9 im Repository hat, kann dieses direkt installiert werden.

Der Ruby Version Manager (RVM) kommt hierbei bewusst nicht zum Einsatz, da die PATH Variable nicht konsistent für alle Benutzer des Systems angepasst wird und es dadurch zu Fehlern in der Ausführung kommen kann.

Bash-Script zur Installation[1]:

```
#!/bin/bash
apt-get install ruby1.9.1 ruby1.9.1-dev
update-alternatives --install /usr/bin/ruby ruby /usr/bin/ruby1.9.1 500\
                        --slave   /usr/bin/ri   ri   /usr/bin/ri1.9.1\
                        --slave   /usr/bin/irb  irb  /usr/bin/irb1.9.1\
                        --slave   /usr/bin/erb  erb  /usr/bin/erb1.9.1\
                        --slave   /usr/bin/rdoc rdoc /usr/bin/rdoc1.9.1
```

Das obige Bash-Script enthält ebenfalls mit update-alternatives den Teil um Verlinkungen zum Aufruf von Ruby richtig zu setzen (also aufrufen von ruby statt nur ruby1.9.1 im CLI).

5.4 RubyGems

Ruby on Rails arbeitet mit Rubygems - dem offiziellen Paketsystem für Ruby - um sicher zu stellen, dass alle Ruby basierten Komponenten vorhanden sind.

In einem Ruby on Rails Projekt können diese Spezifikationen in der Gemfile im Rails Root Verzeichnis gefunden werden.

Um sicher zu gehen, dass es sich um eine kompatible Version handelt, empfiehlt sich hier die manuelle Installation (da Debian ansonsten eventuell noch Ruby1.8 mit installieren will).

Bash-Script zur RubyGems-Installation:

```
#!/bin/bash
cd /tmp
wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.15.tgz
```

```
4 tar -xvzf rubygems-1.8.15.tgz
5 cd rubygems-1.8.15
6 ruby setup.rb
7 update-alternatives --install /usr/bin/gem gem /usr/bin/gem1.9.2 500
8 gem update
```

5.4.1 Gems

Nachdem nun RubyGems installiert ist, sollten noch ein paar Gems und dazugehörige Packages explizit installiert werden.

Bash-Script zur Gems-Installation:

```
1 #!/bin/bash
2 gem install rails --version 3.1.1
3 gem install fastthread
4 gem install bundler
5 gem install rake --version 0.9.2.2
6 gem install net-ssh
```

Als nächstes wechselt man in das root-Verzeichnis der Rails App (abhängig davon, wo diese platziert wurde) und führt dort den folgenden Befehl aus:

```
1 bundle install
```

Sofern das alles geklappt hat, kann man weiter machen.

5.5 Passenger/modrails

Passenger ist das Modul für Apache2, dass die Ausführung von Rails Apps ermöglicht.

Bash-Script zur Installation:

```
1 #!/bin/bash
2 gem install passenger
3 passenger-install-apache2-module
```

Alle weiteren Schritte die zur Installation nötig sind, werden von Passenger abgehandelt (einzufügende Zeilen in apache.conf und sites-available sowie eventuell noch fehlende Packages).

Im Anschluss started man den Apache2 neu:

```
1 /etc/init.d/apache2 restart
```

5.6 Datenbankinitialisierung

Damit die Rails App dann auch tatsächlich funktioniert, müssen noch die nötigen Tabellen und default Werte in die Datenbank eingetragen werden. Dies erfolgt mittels der folgenden Befehle:

```
1 #!/bin/bash
2 rake db:migrate RAILS_ENV="production"
3 rake db:seed RAILS_ENV="production"
```

Damit ist nun die Installation abgeschlossen und die SMA kann genutzt werden, soweit sie denn bereits benutzbar ist.

6 Fazit

Aufgrund von anfänglicher organisatorischer Schwierigkeiten konnte innerhalb des Zeitraums der Studienarbeit nicht der volle Funktionsumfang umgesetzt werden.

Es wurden lediglich grundlegende Funktionen implementiert.

Dies beinhaltet die Oberfläche (mit Layout etc) sowie ein Datenmodell zum Verwalten der verbundenen Switches, der Verbindungen zwischen den daran angeschlossenen Servern und Speichern.

Weiterhin wurde ein Dummyswitch implementiert, um in Ermangelung eines tatsächlichen SANs trotzdem die Kommunikation zwischen SMA und einem Switch zu simulieren.

Für diesen Dummyswitch wurde ebenfalls teilweise ein Standardtreiber implementiert, der sich um die Kommunikation mit dem Server über SSH kümmert. Das explizite Parsen der Rückgabe, sowie das Einfügen der Daten in die Datenbank wurde noch nicht implementiert.

Sobald dieses Projekt eine hinreichende Reife erreicht hat, so ist es sicherlich in der Lage als eine kostenlose, freie Alternative zu den bestehenden Produkten zur SAN Verwaltung zu fungieren. Im aktuellen Status jedoch ist es noch nicht nutzbar und bedarf noch einiger weiterer Arbeit.

7 Glossary

Appliance	Vorkonfigurierte und getestete Lösung.
Fabric	Zusammenschluss mehrerer SAN-Switches zur Verbesserung der Performance und zur Erhöhung der Ausfallsicherheit.
RubyGems	Offizielles Paketsystem für die Programmiersprache Ruby
Switch	In diesem Dokument werden mit Switch SAN-Switches bezeichnet, die die Verbindung von Server zu Speicher ermöglichen.
Telnet	Technologie zur Remote-Arbeit auf einem Server durch Zeichenweise Übertragung der Eingabe.

Bibliography

[1] Brendan Ribera. http://threebrothers.org/brendan/blog/ruby-1-9-2-on-ubuntu-11-04/. 19.01.2012.

[2] http://www.modrails.com/install.html. 19.01.2012.

[3] http://api.rubyonrails.org/.

[4] Kalid Azad. http://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/. 02.02.2012.

[5] http://railscasts.com/episodes/35-custom-rest-actions. 02.02.2012.

[6] http://www.engineyard.com/blog/2010/the-lowdown-on-routes-in-rails-3/. 02.02.2012.

[7] http://guides.rubyonrails.org/action_controller_overview.html 02.02.2012.

[8] Nathan Smith. http://960.gs/.

[9] http://www.accessibleculture.org/articles/2011/10/jaws-ie-and-headings-in-html5/. 02.02.2012.

[10] http://www.colorzilla.com/gradient-editor/.

[11] http://www.itwissen.info/definition/lexikon/Speichernetz-SAN-storage-area-networ
html. 24.05.2012.

[12] http://www.elektronik-kompendium.de/sites/net/0906071.htm. 24.04.2012.

8 Anhang A - Codedokumentation

8.1 Wichtige Verzeichnisse

Tabelle 8.1 enthält eine Übersicht der wichtigen Verzeichnisse die in dieser App verwendet werden. Alle weiteren Dateien und Verzeichnis wurden nicht extra angepasst.

Table 8.1: Wichtige Verzeichnisse

Verzeichnis	**Beschreibung**
app	Enthält die Hauptdateien der App inklusive Controller, Model und Views
app/controllers	Enthält alle Controller in denen die Actions festgelegt werden.
app/models	Enthält die Model-Dateien, in denen Attributszugriffe, Relationen und weitere Filter für die Felder festgelegt werden.
app/views	Enthält alle Views für die verschiedenen Actions und das Layout, dass auf die verschiedenen Views angewendet wird
config	Enthält wichtige Konfigurationsdateien - die einzige in der SMA veränderte Datei ist die routes.rb
db	Enthält die Dateien zur Initialisierung der Datenbank und um diese via ObjectMapper anzusprechen, sowie bei Benutzung von sqlite3 die Datenbankdateien
db/migrate	Enthält Migration-Dateien zum Aufbau der Datenbank und zum Einspielen von Änderungen.
lib/driver	Enthält die Treiber zur Kommunikation mit den SAN Switches
log	Enthält das Log, welches alle Requests an den Server mit loggt sowie Fehler anzeigt, sofern welche aufgetreten sind.
public	Enthält alle Dateien, die nicht interpretiert werden müssen, wie z.B. JavaScript, Stylesheet und HTML-Fehler.
public/css	Enthält alle Stylesheets
public/img	Enthält Bilder für 960gs und die Icons für das Canvas-Element.
public/js	Enthält alle JavaScripts
ROOT	Enthält die Gemfile, in der die nötigen Gems festgelegt werden.

8.2 Wichtige Dateien

Tabelle 8.3 enthält einer Auflistung einzelner wichtiger Dateien. Die Controller und Views werden nicht alle einzeln aufgelistet, sind aber auch dazu zu zählen.

Table 8.3: Wichtige Dateien

Datei	**Beschreibung**
Gemfile	Enthält die Liste aller wichtigen Gems, die für die App benötigt werden.
config/routes.rb	Enthält die Routen zu den Actions auf den verschiedenen Controllern
db/schema.rb	Datenbank-Schema für den ObjectMapper
db/seeds.rb	Bootstrapping-Code zum Anlegen von grundlegenden Datensätzen, wie z.B. dem Default-User
lib/driver/DefaultDriver.rb	Unfertiger Standardtreiber
public/css/layout.css	Enthält css-Befehle bzgl. Layout
public/css/style.css	Enthält css-Befehle für Sachen wie Farbe und Rahmen
public/css/960.css	Enthält das 960gs-Gridsystem
public/js/jquery.js	Enthält jQuery
public/js/processing.js	Enthält Processing.js
public/js/rails.js	Enthält Rails spezifischen jQuery-Code
public/js/sanoverview.pde.js	Enthält den Code für das Canvas-Element

8.2.1 sanoverview.pde.js

Im Folgenden soll die Funktionsweise des Canvas-Elements etwas besser beleuchtet werden, da der Code unter Umständen nicht so leicht verständlich ist. Für alle nicht erwähnten Punkte ist trotzdem ein Blick in den Code zu empfehlen.

Initialisierungscode

Am Anfang des Scripts werden einige Sachen initialisiert, wie die Icons, deren Positionen, die Listen bzw. Maps für die Elemente wie Server, Storage, Switch und Connections. Weiterhin ist hier über ein jQuery-Snippet der Code zum Laden der Konfiguration zu finden.

setup()

Initialisiert das Canvas-Element mit Liniendicke, Textgrößte und die Listen aus den XML-Daten.

draw()

Code mit dem das Canvas-Element regelmäßig neu gezeichnet wird.

initializeElementList()

Legt Objekte aus der XML-Konfiguration an und fügt diese in die Liste `sanelements` sowie die HashMap `connectionelements` ein.

setHeight()

Berechnet die Größe des Canvas-Elements anhand der Anzahl an Elementen für Storage, Server und Switch

drawElements()

Läuft durch `sanelements` und `connectionelements` und ruft auf den Elementen die *drawElement()*-Methode auf. Außerdem schreibt diese Funktion die "Spaltenköpfe" über die Icons.

drawTempConn()

Zeichnet eine neue temporäre Verbindung während des Anlegeprozesses Visualisierung ein.

sendConnectionUpdate()

Generiert XML aus den Connections und sendet diese via Ajax-Call an das Backend.

Class SANElement

Klasse für Objekte des Typs SANElement - damit sind Server, Storages und Switches gemeint. Das Objekt ist primär dafür da, damit die Elemente vernünftig im Canvas-Element gezeichnet werden können und erlaubt eine Gruppierung der Informationen, die dafür notwendig sind.

Class Connection

Klasse für Verbindungen. Hält die Daten, die die Verbindung spezifizieren (von, über, bis), eine Vergleichsfunktion zum Vergleich mit anderen Verbindungen, sowie den Code, um die Verbindung einzuzeichnen.

mouseClicked()

Händelt das Mausklick-Event, was in diesem Zusammenhang mit der Mausposition zur Identifikation der geklickten Elemente genutzt wird und legt bei erfolgreichem Verlaufen des Anlegeprozesses eine neue Verbindung an.

Weiterer jQuery-Code

Weiterhin gibt es noch 3 jQuery-Snippets, die sich mit der Liste und den Buttons unter dem Canvas-Element befassen, wie dem Entfernen einer Verbindung, der Auswahl einer Verbindung aus der Liste und dem Anstoßen des Sendens des Verbindungsupdates.