Student Research Project

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

Academic Advisor of the Project:
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 Introduction

1.1 Task Description

The goal of the project is the creation of a webbased (HTML5) administration tool for SANs. This includes the backend, which is responsible for the connection to the SAN via CLI (SSH or Telnet), as well as the frontend which is the interface for the user.

The frontend is supposed to be user friendly in order to allow even unskilled users to manage the basic functions of a SAN.

The backend is supposed to be extendable via plugins to make sure that even switches can be supported that are not currently existing during the time of the development or those which could not be taken into consideration due to the lack of resources.

1.2 Introduction to the theme

In the development of big computer systems and networks it is necessary to centralize certain resources to make them better usable. Part of this is the creation of a Storage Area Network for the centralized administration of the storage. A SAN is a special data network which is working independent of the existing LAN and is used to take load of it regarding the harddrive access. It also increases performance in the harddrive access due to protocols specialized for this kind of data with less overhead.

1.3 Status at the beginning

The SAN Management Appliance (called SMA in the following chapter) will be developed from scratch. For the development a small SAN with 2 hp Switches is supposed to be set up in the DHBW Stuttgart. (Due to some organizational problems it never came to that)

1.4 Business Use

The SMA is supposed to be a free alternative to all the costly administration tools sold by the hardware developer. Furthermore the user interface was developed trying to be user friendly to make it easy to use without much expert knowledge.

1.5 Struktur des Dokuments

In the following chapters there will be some basic information regarding SANs given to increase comprehension for users with no or less knowledge about them. Furthermore there will sections regarding the design and development, as well as the setup of the system.

2 Storage Area Network

This chapter is giving a short introduction into Storage Area Networks.

2.1 Definition

"Storage Area Networks are dedicated storage networks, which connect server and storage systems using broadband networks like Fibre Channel with each other."[11]

2.2 General

Storage networks are used to centralize storage and to use it more efficiently. Instead of direct attached hard disks to the server, the storage is administered at a central location. In addition to hard disk storage also other types like tape drives can be used and connected to a SAN.

The whole communication between server and storage is done over an extra network, which is independent of the Local Area Network, to prevent overload of the "normal" network.

The network is usually not operating on standard TCP/IP but rather using the Fibre Channel Protocol, which allows for better throughput on file data.

The administration of the storage networks is done using so called SAN Fabrics. These Fabrics are single or a combination of multiple SAN Switches, which handle the connection between storage and server - meaning either grant or deny connection.

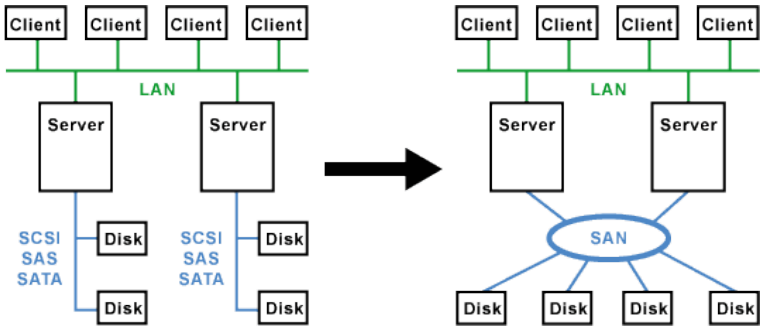

Figure 2.1: SAN[12]

3 Design

This chapter is giving some information about the ideas for the SMA as well as the planned implementation.

3.1 General Structure

The SMA consists in multiple components.

For one there is the Web Frontend, which is the user interface. Another part is the driver component, which is responsible for the communication with the SAN Switch, the parsing of the returned information as well as the integration of new data into the switch or the local database in the backend of the SMA.

3.1.1 Frontend

The frontend is supposed to be easily accessible via any Browser. This is the main reason for the SMA being a Web Application. The Frontend is realized using HTML5, JavaScript and CSS (with help of the grid system `http://960.gs/`).

All this allows easy access via Browser.

3.1.2 Rails App

For the implementation of the logic the decision was made to use `Ruby on Rails`.

The Rails App is responsible for rendering templates and therefore the creation of the frontend. It is also responsible for executing some additional Ruby scripts which handle the communication to the SAN Switch (the driver component).

The logic in the Rails App is responsible for access control as well as the data in the data model.

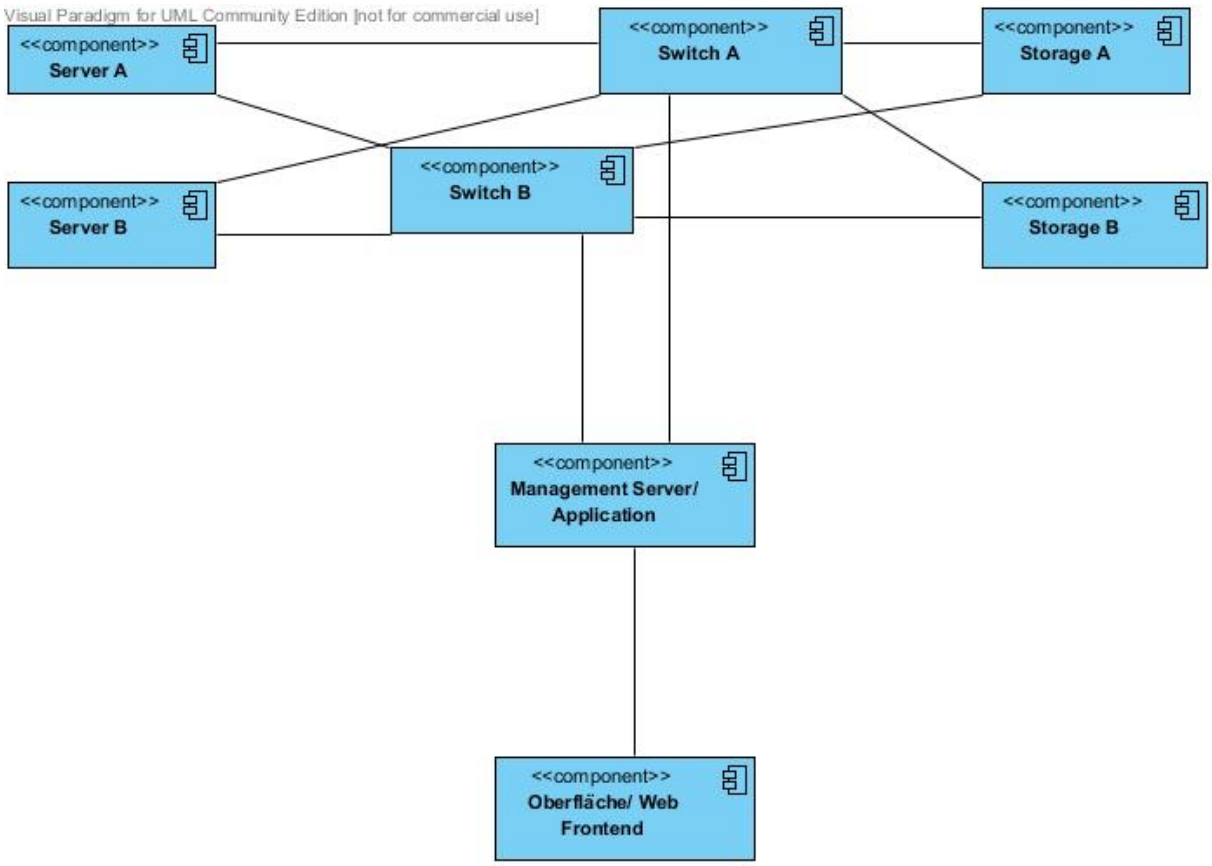

Figure 3.1: SMA Structure

3.1.3 Switch Driver

The Ruby scripts for the communication with the SAN switch can also be called a driver since they translate the commands between the Rails App and the SSH commands for the switch.

The Rails App then can call the command on the driver e.g. to list all the connected servers, which causes the execution of the commands via SSH to list all the connected devices. The returned data is then being parsed and returned to the Rails App in an understandble form.

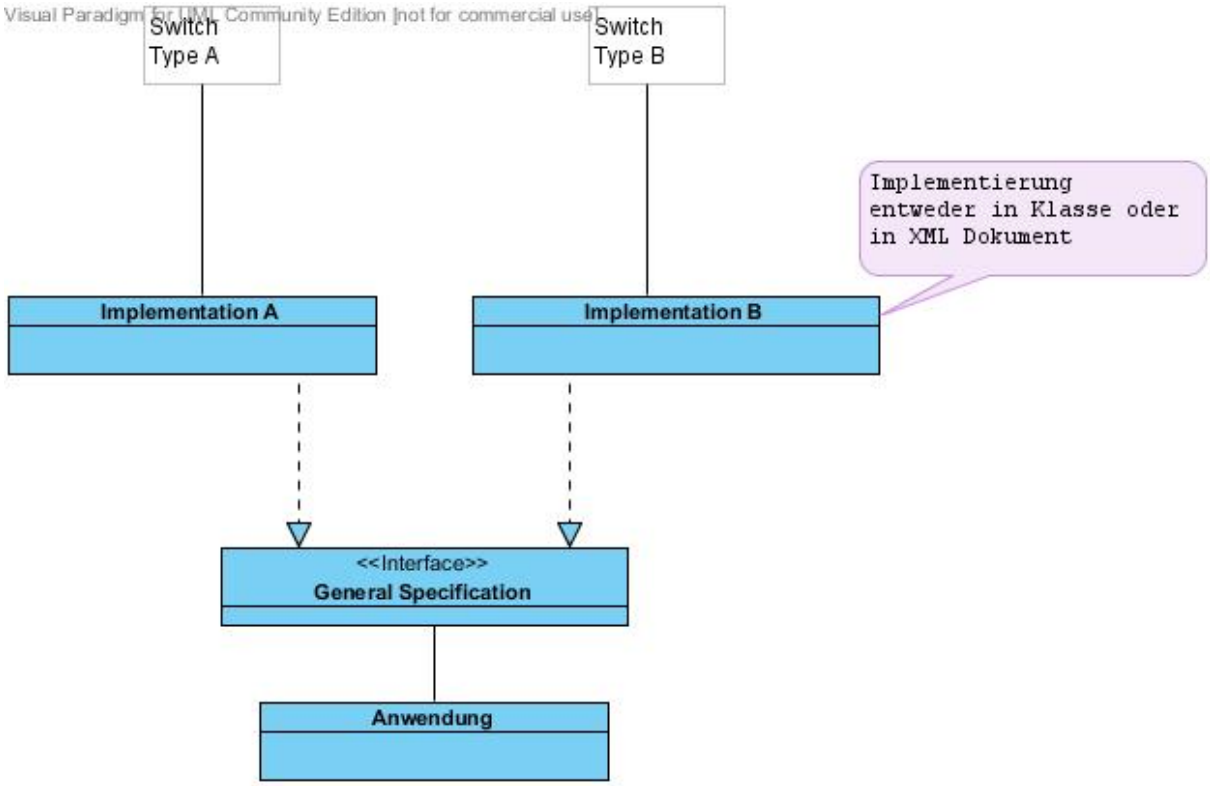

Figure 3.2: Plugin System

4 Implementation

This chapter is all about the implementation of the SMA and can be used to gain insight and understanding for the further development. Starting here some Ruby on Rails knowledge is necessary.

Information regarding Ruby can be found using the following links:

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

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

For Ruby on Rails the following link has proven to be useful:

`http://rubyonrails.org/`

If sources in form of a book are prefered, the following can be recommended:

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

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

`The Rails 3 Way` by Obie Fernandez

Furthermore there is some knowledge in jQuery and Processing.js required in the following sections for better code understanding.

`http://processingjs.org`

`http://jquery.org`

4.1 Data model

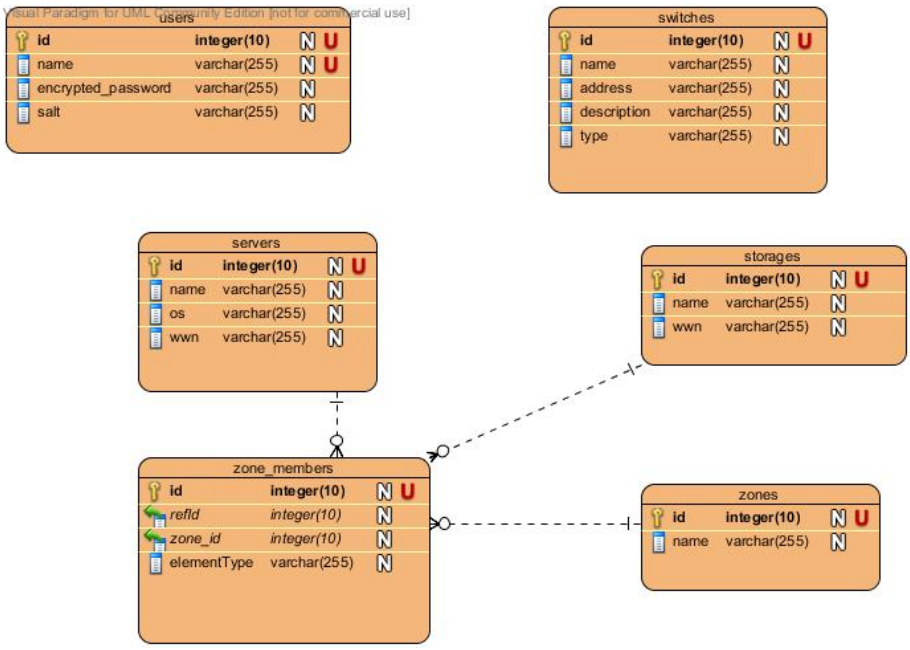

Figure 4.1: Data model

4.2 Access Control

There exists a used which is created on execution on the Seed Script (`db/seeds.rb`). (`rake db:seed RAILS_ENV=production`)[1]

The name of the user is `sanmanager` and the default password is `application`, which can be changed after login.

No functionality has been included to add or remove users, although it would be possible by inserting them into the database directly or via rails console.

4.2.1 Filter

The access control was implemented in the file `app/controller/application_controller.rb` using the *before_filter* `require_login`. The *before_filter* in the ApplicationController causes the filter to be used in all the other controllers, since all controllers inherite from ApplicationController.

```
1 class ApplicationController < ActionController::Base
2   protect_from_forgery
```

[1] More detailed information can be found in the next chapter

```
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
```

This filter checks whether or not a session for a user exists and which user it is. Since there only is one user in the SMA under normal circumstances, the second part is not that important.

The login is done using the `app/controller/login_controller.rb`. In this controller the filter is deactivated, in order to allow the access to actions without prior login.

```
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
```

The login itself is rather simple: using the `authenticate`-methode from the User-Model, the username and password are checked for true or false (if true the reference to the user is returned, else null). In case of sucess the reference to the user is saved in the session which then in turn can be used for the filter.

The exact functionality can be understoof easiest by having a look at `app/controller/application_con` `app/controller/login_controller.rb` and `app/model/user.rb`.

4.3 Configuration and Administration

To show and administer the configuration data multiple controllers have been implemented for Server, Storage, Switch and Zone.

A switch can be added or removed manually including the login data for the SSH connection as well as the IP-address or the DNS-name.

Furthermore the driver is specified which is used for communication with the switch.

The data for storage and server are supposed to be added by reading data from the switch. The removal can be done manually.

Zones are supposed to be added manually and then submitted to the switch, although already existing zones are also supposed to be added automatically on parsing the switch data.

The implementation of this is rather simple and it is mostly just some actions for showing, adding, removing and updating data.

In the following listing a shortened example can be seen to get an understanding of the implemented 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
```

More specific information can be gained by checking the single Controllers and the associated models.

4.4 Canvas Element

For making the SAN elements viewable in a user-friendly fashion a user interface using process.js[2] has been implemented in an HTML5 Canvas Element-

This Canvas Element allows for easy creation of connections/zones between server and storage using a specific switch.

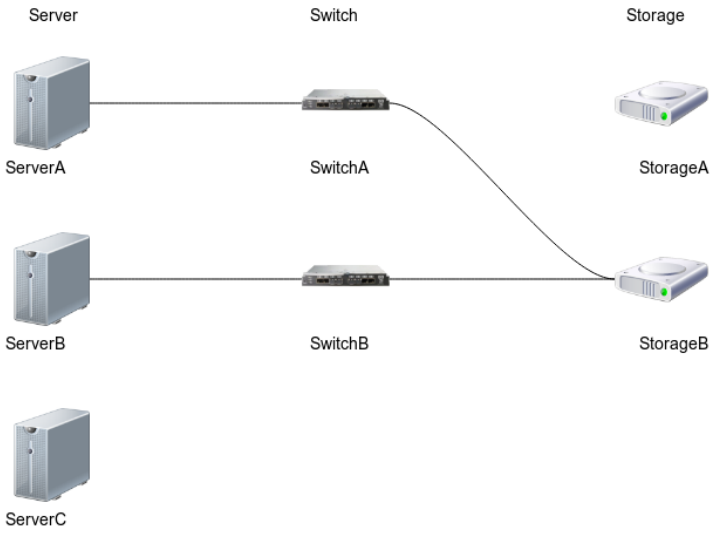

Figure 4.2: Canvas Element

More specific information on how processing.js works can be found at `http://processingjs.org`. There you can find multiple tutorials and examples to better understand how the code works.

4.4.1 Functional principle

The following paragraphs are about the functional principle.

The data about server, storages, switches and connection are being retrived via jQuery[3] using an Ajax-call.

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

The data transfered data is in XML which is being generated by the Rails App using a normal View template from the configured data. The frontend then parses this data and shows it in the canvas element.

An example for the XML data looks like this:

```
<model>
  <serverList>
    <server id="1">ServerA</server>
    <server id="2">ServerB</server>
    <server id="3">ServerC</server>
  </serverList>
  <storageList>
    <storage id="1">StorageA</storage>
    <storage id="2">StorageB</storage>
  </storageList>
  <switchList>
    <switch id="1">SwitchA</switch>
    <switch id="2">SwitchB</switch>
  </switchList>
  <connectionList>
    <connection>
      <from>1</from>
      <over>1</over>
      <to>2</to>
    </connection>
    <connection>
      <from>2</from>
      <over>2</over>
      <to>2</to>
    </connection>
  </connectionList>
</model>
```

The code for this element can be found in the file `public/js/sanoverview.pde.js`.

4.4.2 Create connection

New connections can be created by clicking on either server or storage, then the switch and at the end of the last type left (if storage has been selected before => server and vice versa).

The position of the mouse is used to determine which element was clicked as well as to draw a line from the clicked element to the current mouse location.

After clicking on the switch the first part of the connection is drawn in the canvas and another line is drawn from the switch to the current mouse position.

On clicking on the last element for the connection, the connection is actually created in the canvas element (not yet on the server).

In order to abort this process just click on any other part of the canvas. This removes this temporary connection again.

4.4.3 Remove connection

In order to remove a connection from the canvas just select the connection from the Connectionlist below the canvas and then click on `Delete selected`.

4.4.4 Send changes

As soon as all changes have been made you can click on `Commit changes`. This causes an Ajax-Call to be made sending a list of all the current connections as XML to the server.

In the backend the received list of connection is compared to the connections in the database and updated accordingly (new connections created, missing connections removed etc).

4.5 Driver

The driver scripts have to be put to `lib/driver`.

These scripts create a connection to the server, execute some commands and close the connection again.

An example looks like this:

```
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
```

Via the SANCommunicationsController the fitting driver for the switch (configured on creation of the switch) is being chosen, the methods executed and the fitting result processed.

`IMPORTANT:` The classname and the filename have to be identical (case-sensitive), since the listing of the drivers on switch creation is done using the file name and the connection to the switch by using the same string to pick the appropriate class.

5 System setup

The SAN Management Appliance has been implemented using Ruby on Rails and the communication with a switch using normal Ruby.

Due to constant development in Ruby and Ruby on Rails it is important to pick compatible versions of the components

The installation is not totally trivial and is therefore done in the following sections using a Debian 6 as example. It is assumed that the system is clean and no older version of Ruby or Rubygems exists on the system. Furthermore it is assumed that a location for the Rails App has been chosen and the Rails App has been extracted there.

5.1 Apache 2

As a web server an Apache 2 is being used.

Bash-Script for the installation:

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

5.2 SQLite3

For storing the configuration information of the SAN they are being stored in an SQLite database.

Bash-Script for the installation:

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

5.3 Ruby Installation

By now Debian has a proper package of Ruby1.9 in the Repository which can be used.

The Ruby Version Manager is intentionally not used since the PATH variable is not necessarily consistent for all users and it can create errors in execution.

Bash-Script for the 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
```

In the Bash-Script the command 'update-alternatives' is being used to create proper links to allow Ruby to be executed on the CLI by typing ruby and not just ruby1.9.1.

5.4 RubyGems

Ruby on Rails is using RubyGems - the official packetsystem for Ruby - in order to make sure that all Ruby based components are available.

In a Ruby on Rails projects the specification of the required Gems can be found in the Gemfile in the Rails Root directory.

In order to make sure that the version of Rubygems is compatible, it may be best to use a manual installation and not the Repository (Since otherwise Debian might want to also install Ruby1.8).

Bash-Script for RubyGems-Installation:

```
#!/bin/bash
cd /tmp
wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.15.tgz
tar -xvzf rubygems-1.8.15.tgz
cd rubygems-1.8.15
ruby setup.rb
update-alternatives --install /usr/bin/gem gem /usr/bin/gem1.9.2 500
gem update
```

5.4.1 Gems

After having installed RubyGems, some Gems should be installed explicitely.

Bash-Script for 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
```

Now change directory to the root of the Rails App and execute the following command:

```
1 bundle install
```

If that worked and no errors where returned, the installation can continue.

5.5 Passenger/modrails

Passenger is the module for Apache2, which makes it possible to execute Rails App.

Bash-Script for the installation:

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

All further steps necessary for the passenger installation will be handled by passenger. This may be some still missing packages or adding some lines to apache.conf or sites-available

After that Apache2 can be restarted using the following command:

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

5.6 Database Initialization

In order to make the Rails App actually work, the database has to be created and filled with default values.

This can be done using the following commands:

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

After having done this the installation is finished and the SMA can be used as far as it has been developed by now.

6 Conclusion

Due to some organisatorial difficulties the whole functionality could not be done within the duration of the project.

Only some basic functionality has been implemented.

This contains the frontend (including Layout etc) as well as the data model to store the data of the connected switches and the connections between the servers and storages.

Furthermore a dummyswitch has been implemented in order to simulate the connection between SMA and a switch despite the lack of an actual SAN.

For this dummyswitch a DefaultDriver has been implemented in parts, which handles the communication with the server via SSH.

The parsing of the returned data, as well as adding the data to the database has not yet been implemented.

As soon as this project has reached a proper level it can certainly be used as a free alternative to the existing products for SAN administration. With the current status however it is not yet usable and require more work.

7 Glossary

Appliance	Preconfigured and tested solution
Fabric	Combination of multiple SAN-Switches in order to improve performance and increase reliability.
RubyGems	Official package system for the programming language Ruby
Switch	In this document SAN Switches are refered to as Switches
Telnet	Technology for Remote Access to a Server via character-wise transfer of the Input.

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 Attachment A - Code Documentation

8.1 Important Directories

Table 8.1 contains an overview of all the important directories that are being used in this app. All the directories that are not mentioned here, have not been touched and only contain the default Rails-generated Code.

Table 8.1: Important Directories

Directory	**Description**
app	Contains the main files of the app, including the Models, the Views and the Controller
app/controllers	Contains all the Controller in which the different Actions are specified.
app/models	Contains all the Model-files in which attribute accessors, relations and valid conditions for the fields are defined.
app/views	Contains all Views for the different Actions as well as the Layout which is used for the different Views.
config	Contains important configuration files - the only files changed in the SMA is the routes.rb
db	Contains the files for initialising the database and to communicate with the database using the ObjectMapper. In case of using sqlite3 the database files are also stored here.
db/migrate	Contains the Migration-files used to build the database as well as to include changes.
lib/driver	Contains the drivers for the communication with the SAN Switches
log	Contains the logs, in which all the requests to the server and potential errors are logged.
public	Contains all the files that do not need to be interpreted, e.g. JavaScript, Stylesheet and HTML-Error files.
public/css	Contains all the Stylesheets
public/img	Contains images for 960gs and the icons for the canvas element.
public/js	Contains all the JavaScripts
ROOT	Contains the Gemfile, in which the required Gems are specified..

8.2 Important files

Table 8.3 contains a list of a few important files. The Controller and the Views are not listed individually, but they are also important.

Table 8.3: Important files

File	**Description**
Gemfile	Contains a list of all the important Gems, which are necessary for the app.
config/routes.rb	Contains the routes to the actions on the different Controllers
db/schema.rb	Database schema for the ObjectMapper
db/seeds.rb	Bootstrapping-Code to create basic inital database entries such as the default user
lib/driver/DefaultDriver.rb	unfinished default driver
public/css/layout.css	css-code for the layout (positioning)
public/css/style.css	css-code for things like color and border
public/css/960.css	960gs-Gridsystem
public/js/jquery.js	Contains jQuery
public/js/processing.js	Contains Processing.js
public/js/rails.js	Contains Rails specific jQuery-Code
public/js/sanoverview.pde.js	Contains the Code for the Canvas element

8.2.1 sanoverview.pde.js

This section is supposed to give a better insight into how the canvas element works, since the code might a big hard to understand.

Initialisation code

At the top of the script a couple of things are being initialized, such as icons, the position, lists/maps for the elements like server, storage, switch and connections. Furthermore there is a jQuery-Snippet that loads the configuration from the backend.

setup()

Initialization of the canvas element with the stroke level, text size and the lists from the XML data

draw()

Code which re-draws the Canvas element.

initializeElementList()

Creates the objects from the XML configuration and adds them to the list `sanelements` and the hashmap `connectionelements`

setHeight()

Calculates the size of the Canvas element from the amount of storage, server and switch elements

drawElements()

Runs through `sanelements` and `connectionelements` and calls the method *drawElement()* on the elements. It also writes the “column headers" for the icons.

drawTempConn()

Draws a temporary connection during the connection creation process for better visualization.

sendConnectionUpdate()

Generates XML from the connections and sends it via Ajax call back to the backend.

Class SANElement

Class for objects of the type SANElement - SANElements are servers, storages and switches. The objects primary function is to take care of properly drawing the elements in the canvas and to group the information required for that.

Class Connection

Class for Connection. Contains the data which specifies a connection (from, over, to), a compareTo-method to compare it to other connections and code to draw the connection.

mouseClicked()

Handles the mouseClicked-event and is called then. In this case on clicking the mouse position is used to identify the clicked elements and to create a new connection upon finishing the creation process successfully.

Further jQuery-Code

At the bottom of the script there are 3 jQuery-snippets, which are used to handle the list and the buttons below the canvas element. The actions created by this code are the deletion of a connection from the canvas, the selection of a connection from the list as well as the call of the sendConnectionUpdate-function.