BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Biomedical Engineering

BOINSO – Open source development of a distributed satellite monitoring network and its implications on telemedicine applications of the future

By: Gregor Beyerle
Student Number: 1210227035

Supervisor: FH-Prof. Dipl. Ing. Dr. Lars Mehnen

Vienna, May 11, 2015

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (for example see §§21, 46 and 57 UrhG (Austrian copyright law) as amended as well as §11 of the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

In particular I declare that I have made use of third-party content correctly, regardless what form it may have, and I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see §11 para. 1 Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Place, Date Signature

Kurzfassung

Nach dem Start des CubeSat Programmes 1999 begannen Micro- und Pico-Satelliten ihren Platz in Raumfahrtprogrammen einzunehmen. Heutzutage werden Flugkörper dieser Größenordnung von Forschungseinrichtungen verwendet um Ingenieure auszubilden und selbst kostengünstig Forschung zu betreiben. Selbst kommerzielle Anbieter entwickeln derzeit in diesem Bereich, da größere und schwerere Satelliten, welche höher gelegene Orbits einnehmen, weitaus teurer sind. Auf Grund von Kostenersparnissen und der kurzen Lebensdauer von Kleinsatelliten werden diese in Umlaufbahnen gebracht die nur 120–1200 km über dem Meeresspiegel liegen. Objekte in solchen Höhen besitzen keinen geostationären Orbit, was dazu führt, dass von einer Bodenstation kein durchgängiger Funkkontakt hergestellt werden kann. Ein Kontrollzentrum hat daher nur zu etwa zwölf Terminen pro Tag Kommunikationsfenster von ungefähr fünfzehn Minuten. Dieser Umstand ist nur schwer hinnehmbar.

Weltweit gibt es hunderte Einrichtungen, welche in der Lage wären Verbindung zu erdnahen Satelliten aufzunehmen. Betreiber dieser Stationen arbeiten schon jetzt daran Satelliten manuell zu verfolgen, Daten zu empfangen und diese den zugehörigen Kontrollzentren weiterzuleiten. Dieser Prozess findet für gewöhnlich ohne Koordination der Kontrollzentren statt und erfolgt meist bruchstückhaft und unzuverlässig. Stationen, welche auf einen Satelliten eingestellt wurden verbleiben die meiste Zeit in Warteposition. Projekte wie das "Global Educational Network for Satellite Operations" versuchen die schwache Funknetzabdeckung sowie die Nutzungseffizienz der Sende- und Empfangsstationen zu verbessern. Für Missionen, die sich nicht von einem stark zentralisierten Netzwerk abhängig sein wollen gibt es allerdings noch keine veritablen Alternativen.

Diese Arbeit beschäftigt sich mit der Implementation eines verteilten Netzwerkes von Bodenstationen unter Anwendung moderner Web-Technologien und Architekturparadigmen.

Schlagworte: BOINSO, Open Source, CubeSat

Abstract

After the start of the CubeSat program in 1999 pico and micro satellites began to make an appearance in educational space flight. Today spacecrafts in this size category are used by research institutions to train engineers and collect large amounts of data. As bigger satellites in higher orbits are more expensive even commercial satellite missions start to venture in this area. Due to the lower costs and the short life period of those devices they orbit in an altitude of 120–1200 km. Objects in these heights do not have an geostationary orbit which leads to difficulties while establishing regular radio contact. A mission control center tracking a satellite has about 15 minutes per pass which occurs up to twelve times per day in optimal conditions for radio communication. This is insufficient for most endeavors.

Currently there are hundreds of educational and amateur radio stations equipped to establish satellite communication. The operators of those stations track satellites and transmit down-link data to mission control independently, and are mostly uncoordinated. The original mission control center and ground stations configured to track a single satellite have to rest in an idle state for the majority of the time. Projects such as Global Educational Network for Satellite Operations tackle the issue of the poor communication coverage and the ineffective use of ground station resources. However it has yet to offer a viable option for projects which require an individual communication network approach without connecting to a centralized system.

In this paper it is the goal to tackle this issue by implementing a mission control and ground station network using a set of modern web application tools and architectural paradigms.

Keywords: BOINSO, Open Source, CubeSat

Acknowledgements

I want to thank my colleague Marcel Cimander for working with me on this project for the last year and his great work on GPredict which will hopefully soon be a part of the official release cycle. I would also like to thank Alexandru Csete for his wonderful work on GPredict, the fact that he also still maintains the Win32 build of the software and for his kind consideration.

I also want to thank Mr Dipl.-Ing. Mag. Christian Siehs for "pushing" me into web development as otherwise I would most likely not have been able to pursue this project the way I did. Additionally I would like thank my girlfriend who had to cope with me for the time I was working on this project and proved an extreme resilience to my grumpiness when facing a major bug.

Finally I would like to thank Mr FH-Prof. Dipl. Ing. Dr. Mehnen, who approached and trusted us with this very demanding but also very rewarding project. I would never have dreamed of actually working on a project of such a scale within my undergraduate program.

Contents

1 Introduction

The use of pico and micro satellites in the context of educational aerospace programs has introduced opportunities for both private enthusiasts and students in small scale cost efficient satellite endeavors. Mostly lacking central oversight and costly infrastructure around the globe the efficiency of those programs is depending on the collaborative efforts of a growing community sharing both hardware, computational infrastructure and time. Due to the characteristics of orbiters deployed in a Low Earth Orbit (LEO) a stationary observer can establish a direct line of sight only up to twelve times a day – as stated in [1] – which can be maintained for about fifteen minutes per pass.

The challenges posed by the difficult monitoring of small satellites orbiting in LEO and possible ways to overcome them were outlined in [2]. This bachelor paper focuses on the implementation of an accessible and easily modifiable solution to connect Mission Control Centers (MCCs) and give Ground Control Centers (GCCs) the opportunity to deliver viable contributions to the monitoring process.

In the following chapters the reader is going to be presented with an outline of the tools and materials used to achieve this goal, the structure of the different applications written for this task as well as details about their implementation. Finally there will be a recapitulation of challenges which arose during the work on the project and an outlook on future developments.

1.1 BOINSO Core Web Application

The BOINSO Core Web Application is the main entry point for MCC administrators for maintaining the data related to their managed satellites. Besides the administrative interface the Core Web Application offers an Application Programming Interface (API) accessible via Representational State Transfer (REST) calls. An additional part of the Core Application is an OAuth2 provider which is used to authenticate users with a MCC.

1.2 BOINSO MCC Web Client

The BOINSO MCC Web Client is a HTML5/JavaScript sample implementation of a custom BOINSO client. It offers GCCs the possibility to register with a MCC, manually download Two Line Elements (TLE) files, administer their profile data and their tracking data.

1.3 BOINSO GPredict Bridge

The BOINSO GPredict Bridge is a piece of middle-ware distributed as a python module. It is used to pull updates of satellite data from the MCCs to their affiliated GCCs and in return automatically upload tracking data to the BOINSO Core Web Application of the satellites owner using the client API.

2 Material

The following sections list the different tools used while working on the components of the BOINSO network applications.

2.1 General

This section includes tools which allow a group of developers to work on complex projects in a decentralized manner. As this project is intended to be used in an academic context and monetary resources are scarce in this field tools that are either open source or free of charge for educational programs.

2.1.1 GIT-SCM

The Git Source Code Mirror (git-scm) is a "[...] fast, scalable, distributed revison control system with a [...] rich command set that provides both high-level operations and full access to internals" as declared in [3]. It follows the same "branch -> develop -> merge" work-flow as Subversion (SVN) and is freely distributed under the GNU General Public Licence Version 2 (GPLv2). It was originally developed by Linus Torvalds to offer Linux Kernel developers a free way to collaboratively work on a distributed code-base efficiently.

2.1.2 GitHub

GitHub is a provider of cloud hosted git-scm repositories. As GitHub was originally introduced by members of the open source community it still maintains a very generous relationship to open source contributors. GitHub users who publish their work to public repositories may use virtually all services bound to the GitHub infrastructure free of charge with only minimal limitations. GitHub also offers premium enterprise accounts which include a certain amount of private repositories and premium services if they are needed.

2.1.3 Travis-CI

Travis-CI is a web hosted continuous integration server. Continuous integration is the automated process of building and testing a project with every introduction or modification of a software component. Its goal is to assure and improve the quality of the project while alerting the development team if a change would lead to a breaking application.

In contrast of other continuous integration solutions like Jenkins – an open source continuous integration server implemented in the Java programming language – the configuration of a Travis-CI process is done by adding a simple configuration file to root of your git-scm repository as seen in listing 1.

```
language: python

python:
    - "2.7"
    - "3.4"

services: postgresql

before_script:
    - psql -c 'create database travisci;' -U postgres

install:
    - pip install -r requirements.txt
    - pip install coveralls

script:
    - coverage run --rcfile=.coverage.rc manage.py test

after_success:
    coveralls --rcfile=.coverage.rc
```

Code 1: BOINSO Travis-CI configuration file. For this file type the YAML syntax is used. The built process runs once for every Python interpreter version included in the configuration. Services like a data base connection can be included and configured prior to the built process. It is possible to include command line expressions and additional hooks which react to Travis-CI events.

2.1.4 Vagrant

Vagrant is a tool which is used to virtualize development environments. It offers a command line interface to download, start up, pause, halt and provision images of virtual machines which come configured with all the dependencies needed to develop, run and test an application. Base images can be built to closely model a production system as closely as possible being configured by a trained system administrator masking the complexity of this system from developers and designers. Depending on the base image type Vagrant users have to have access to a certain virtualization provider – also known as hypervisor.

The initialization process of a Vagrant environment depends on the presence of a vagrant configuration file – simply known as the Vagrantfile. Listing 2 depicts the Vagrantfile of the BOINSO Core Web Application.

```
Vagrant.configure(2) do |config|

  config.vm.box = "WalternativE/django_base_box"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.network :forwarded_port, host: 8000, guest: 8000

end
```

Code 2: BOINSO Core Web Application Vagrantfile including expressions to set the virtual base box, a simple shell provisioner executing a bash script which installs variable project dependencies, and the automated port forwarding from the virtual box to the host development machine

Provisioning can be done by DevOps tools like Puppet or Chef or any other program used in this context. As the scope and the resources of this project are limited a simple shell provisioner was used. An example for a provisioning script can be seen in listing 3.

```
#!/usr/bin/env bash

cd /vagrant
sudo rm /var/lib/apt/lists/* -R
sudo apt-get update
sudo apt-get upgrade -y

# whatever you want to provision comes here
# right now only shell provisioning works in this box
# chef/puppet will be added (eventually)

sudo pip3 install -r requirements.txt

echo "provisioner is done"
exit 0
```

Code 3: Vagrant shell provisioning bash script

2.1.5 Open Source Licensing

Open source software is easy to extend and distribute as portability issues can be solved directly in the source code. Unfortunately there are numerous developers and companies which sole purpose seems to be the destruction of open source projects through the medium of patenting and licensing schemes. To protect a project and its collaborators a well established software license should be used. The Free Software Foundation (FSS) provides a set of licenses and guidelines for their own licensing products and affiliated licenses. A starter guide for choosing the right free software license can be found at [4]. Project initiator should be aware that licenses recommended by the FSS are most likely going to be rather strict free software licenses which makes later commercial use harder in many cases. That is the reason for the BOINSO project favoring the Apache License 2.0.

2.2 BOINSO Core Web Application

This section focuses on the tools and frameworks used to implement the core of the BOINSO Core Web Application.

2.2.1 pip

In a modern Python environment pip is used to manage Python modules. Pip itself is a python module which accesses the Python Package Index (PyPI), downloads a module and its dependencies, starts the compilation process for Python extensions and either adds the module to the global Python interpreters Python path or the path of a local virtual Python environment. Besides installing single packages on demand for console execution pip can also be used as a dependency management tool. Rather than manually managing all dependencies of a projects developers normally include a requirements.txt file to their project roots which is used by pip to install project dependencies in the right version. Listing 4 shows the requirements.txt file of the BOINSO Core Web Application.

```
# if you change your database backend you don't need this driver
psycopg2==2.6

# for the safest timezone support install this package
pytz==2014.10

Django==1.7.5
djangorestframework==3.0.5
django-oauth-toolkit==0.7.2


# for enabling CORS-Headers (used when AJAX calls are fired from another domain)
django-cors-headers==1.0.0

# this package is exclusively for testing ssl
# never use the django devel server or the django ssl server for production!
django-sslserver==0.14

# for building docu only
sphinx
sphinx_rtd_theme
```

Code 4: BOINSO Core Web Application requirements.txt file including project dependencies and their version. Version notation follows the "major.minor.patch" pattern. Omitting versioning information means that the latest available version of a module will be installed – in this case this is not likely to cause problems as sphinx is only used to generate documentation.

2.2.2 Virtualenv

Virtualenv is a Python command line tool to create local light weight Python environments with encapsulated tools and an isolated Python module path. Virtual environments can be activated on a per shell basis and replace the system's standard Python installation in this scope. Virtualenv is used to separate the dependencies of a distinct application from other project as different dependency versions are prone to errors when being installed to the same context.

2.2.3 Django

Django is one of the major Python web frameworks. It is backed by a very committed developer community, easy to use, secure and scalable. Without further configuration it includes powerful management tools, database independent Object Relational Mapping (ORM) facilities, an intuitive template engine and an automatically generated administration interface. The project maintainers offer these tools but are very keen on making sure that all components are as modular as possible. This means that almost all components can be substituted by other established projects (e.g.: Jinja template engine instead of Django template engine or SQLAlchemy ORM instead of Django ORM). Besides substituting core components the Django app system also allows developers to write framework extensions which add additional functionality.

Django can be used to its full potential when working on data driven dynamic web applications. To give developers granular control of application functionality and to enforce a maintainable code base the framework makes extensive use of the Model View Controller (MVC) pattern. Figure 1 shows the typical collaborative processes between the different MVC components.

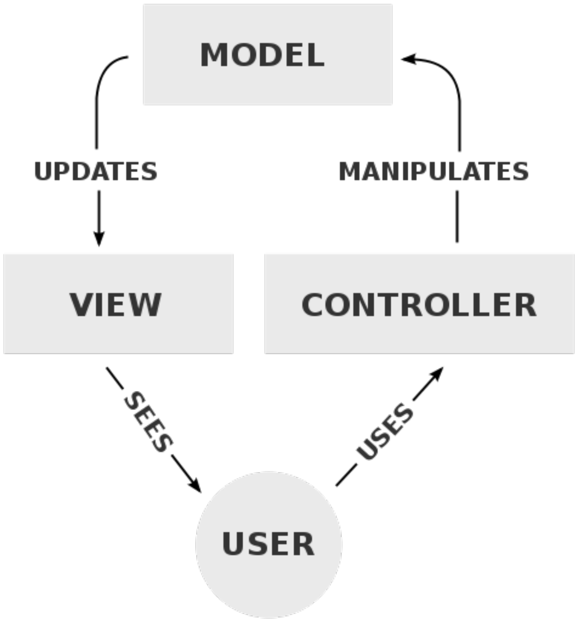

Figure 1: Typcial collaboration of MVC components http://commons.wikimedia.org/wiki/File:MVC-Process.svg Last acces: 09.05.2015

```python
class Satellite(models.Model):

    """
    Satellite model represents an earht orbiter.
    Closely modelled after GPredict satellite representation.
    """

    catalogue_number = models.IntegerField(blank=False)
    version = models.IntegerField(default=1)
    name = models.CharField(max_length=140)
    nickname = models.CharField(max_length=140)
    tle = models.FileField(null=True, blank=True)

    # definitions for operation choices
    OP_STAT_UNKNOWN = 0          # unknown status
    OP_STAT_OPERATIONAL = 1      # operational
    OP_STAT_NONOP = 2            # non operational
    OP_STAT_PARTIAL = 3          # partially operational
    OP_STAT_STDBY = 4            # standby
    OP_STAT_SPARE = 5            # spare
    OP_STAT_EXTENDED = 6         # extended mission

    STAT_CHOICES = (
        (OP_STAT_UNKNOWN, 'operation status unknown'),
        (OP_STAT_OPERATIONAL, 'operational'),
        (OP_STAT_NONOP, 'non operational'),
```

```
        (OP_STAT_PARTIAL, 'partially operational'),
        (OP_STAT_STDBY, 'on standby'),
        (OP_STAT_SPARE, 'spare'),
        (OP_STAT_EXTENDED, 'extended mission')
    )

    status = models.IntegerField(max_length=2,
                                 choices=STAT_CHOICES,
                                 default=OP_STAT_UNKNOWN)

    def __str__(self):
        return "{} Version {}".format(self.name, self.version)
```

Code 5: BOINSO Core Web Application satellite model

Model

The model is the programmatic representation of an entity. The ORM implementation maps the different data members of the model class to a table in the databases and casts the fields to the related database vendor type implementations. An example for a simple model implementation can be seen in listing 5. Besides the correct mapping to the data base the field types also allow the usage of validation chains which throw errors if input values violate input constraints.

View

The view is the layer in which data is presented to clients who can interact with it without having knowledge of the underlying logic. In classical Django applications a view is called a template and rendered to a Hypertext Markup Language (HTML) document presented to a web browser. The BOINSO Core Web Application makes use of the Django Representational State Transfer (REST) Framework where a serializer is used instead of a template. Listing 6 shows the serializer related to the satellite model shown in listing 5 extending a model serializer provided by the Django REST Framework which provides resource identification and resource location by Uniform Resource Locator (URL) fields catering to the Hypermedia as the Engine of Application State (HATEOAS) principle introduced in [5].

```
class SatelliteSerializer(serializers.HyperlinkedModelSerializer):

    """
    Serialzes Satellite objects. Clients should see data
    regardles of their login status. Read only.
    """

    transponders = serializers.HyperlinkedRelatedField(
        many=True,
        read_only=True,
        view_name='transponder-detail'
    )
```

```

    class Meta:
        model = Satellite
        fields = ('url', 'catalogue_number', 'version', 'name', 'nickname',
                  'tle', 'status', 'transponders')
```

Code 6: BOINSO Core Web Application satellite serializer

Controller

The controller reacts to signals set by the user, manipulates the state of a model and updates views. Business logic is incorporated in this component. In the Django vernacular a controller is – to the confusion of many users – called a view. A sample of a more complex controller is depicted in listing 7 where a client request containing an OAuth2 access token is used to filter for the related user object. The view establishes the connection to the serializer class and checks requests for both authentication (client is registered user) and permission (registered user has the rights to view/modify requested data).

```
class UserProfileProxy(generics.ListAPIView):

    """
    Narrows down the search for a user via his authentication.
    Authenticated user sees his own profile and a link to his user endpoint.
    Is used to get userdata via oauth token authentication.
    """

    serializer_class = UserProfileSerializer
    authentication_classes = (OAuth2Authentication,)
    permission_classes = (IsAuthenticated, TokenHasReadWriteScope)

    def get_queryset(self):
        """
        This view should only get the profile of the authenticated user.
        """

        user = self.request.user
        return UserProfile.objects.filter(user=user)
```

Code 7: BOINSO Core Web Application UserProfileProxy

2.2.4 Nginx

Nginx is an open source reverse proxy for protocols for web and mail protocols, as well as a load balancer, cache and a web server. It is available for all major operating systems, easy to deploy and easy to configure. In the context of a deployed Django application Nginx is used to tunnel requests to the application's Web Server Gateway Interface (WSGI) server, to serve static content (images, style sheets, etc.) and to cache non dynamic content. An example for a production configuration file for a Django application using Nginx can be seen in listing 8.

```
server {
        listen 8000 default_server;
        listen [::]:8000 default_server ipv6only=on;

        server_name localhost my_domain.com;

        client_max_body_size 20M;

        location / {
                proxy_set_header Host $http_host;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_pass http://127.0.0.1:8001;
        }

        location /static/ {
                autoindex on;
                alias /PATH_TO_PROJECT/staticfiles/;
        }
}
```

Code 8: Nginx configuration file example. Virtual host is instructed to listen to port 8000 while limiting the maximum body size to 20 megabyte. Requests directed at the virtual host are forwarded to the local application server including the original headers. A static file request is handled by Nginx.

2.2.5 Gunicorn

Gunicorn is a Python WSGI server implemented adhering to the WSGI definition postulated in [6]. As the integrated Django development server is well suited for testing and developing but it is not suited for a production environment due to stability and security implications deployment the usage of a well established solution such as Gunicorn or uWSGI. In cases where the production environment cannot sustain aforementioned solutions (as on Windows server machines) or whenever it is simply not desired to use an additional third party solution there are also modules for the popular Apache Web Server (Berkeley Software Distribution (BSD) httpd) or the Microsoft Internet Information Services (IIS).

Gunicorn is normally installed as a Python module in a virtual environment. This means that there is normally no native operating system hook which starts, restarts and monitors the server processes. In a Python environment a service which can be used to keep the server alive is called supervisord. With a simple configuration file like the one in listing 9 supervisord can handle the server life-cycle, restarting failed and zombie processes. The configuration execution itself can be handed to the operating system via the system initiator daemon (on Unix systems mostly initd, systemd or upstartd).

```
[supervisord]
logfile=/tmp/supervisord.log ; (main log file;default $CWD/supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4 rotation;default 50MB)
logfile_backups=10           ; (num of main logfile rotation backups;default 10)
loglevel=info                ; (log level;default info; others: debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default supervisord.pid)
nodaemon=false               ; (start in foreground if true;default false)
minfds=1024                  ; (min. avail startup file descriptors;default 1024)
minprocs=200                 ; (min. avail process descriptors;default 200)

[program:gunicorn]
command=/VIRTUAL_ENV_PATH/bin/gunicorn APPLICATION_NAME.wsgi:application -w 4
    --bind 127.0.0.1:8001 --pid /tmp/gunicorn.pid ;
stdout_events_enabled = true
stderr_events_enabled = true
directory=/PATH_TO_APPLICATION/ ;
```

Code 9: Supervisord example configuration file

2.3 BOINSO MCC Web Client

This section focuses on the tools and frameworks used to implement the client side web client for the BOINSO application.

2.3.1 AngularJS

AngularJS is a client side JavaScript MVC framework for constructing single page applications. A single page application – written using HTML, Cascading Style Sheets (CSS) and JavaScript – offers rich client experience incorporating a modular and testable design. In contrast to classic web applications where every navigation results in a new request and the download of all web resources a single page application loads all the static content on the first request loading dynamic asynchronously on demand.

A concept heavily used in AngularJS is the Inversion of Control (or dependency injection) paradigm. Functionality shared by different objects is encapsulated in services which are listed as object dependencies. Injected dependencies (or instances thereof) become part of the object's state and can be called as if they were original members. In test situations those services can be easily exchanged by mock objects improving component isolation.

To make templates in AngularJS semantically more expressive it is possible to implement custom directives. This process loosely resembles the working draft for custom elements (also known as web components) as stipulated in [7]. In many cases this fact can be used to distribute compact application modules as directives which are easy to use and configure.

2.3.2 Stylus

Stylus is a CSS precompiler language which is used to write styling information in a more efficient and readable way. **CCS!** (**CCS!**) precompilers offer programming language features like arithmetic expressions, variable declarations, function declarations and selector nesting which are not part of the official CSS3 implementation. With the addition of stylus modules mundane and repetitive tasks like adding vendor prefixes, or working with basic typographic definitions can be solved automatically.

2.3.3 Node Package Manager

The Node Package Manager (npm) is used to manage NodeJS packages. Many JavaScript projects are already deployed to the node package index and can be used as a managed dependency. For using npm as a local dependency management and build tool a package configuration file as seen in listing 10 can be added to the project's root directory.

```
{
  "name": "BOINSO_MCC_WEB_CLIENT",
  "version": "1.0.0",
  "description": "A web based client for mission control communication and
      administration",
  "repository": {
    "type": "git",
    "url": "https://github.com/WalternativE/BOINSO-MCC-Web-Client.git"
  },
  "keywords": [
    "BOINSO"
  ],
  "author": "Gregor Beyerle",
  "license": "Apache 2.0",
  "bugs": {
    "url": "https://github.com/WalternativE/BOINSO-MCC-Web-Client/issues"
  },
  "homepage": "https://github.com/WalternativE/BOINSO-MCC-Web-Client",
  "dependencies": {
      "bootstrap-styl": "latest",
      "angular": "latest"
  },
  "devDependencies": {
      "stylus": "latest",
      "typographic": "latest",
      "nib": "latest",
      "uglify-js": "latest"
  },
  "scripts": {
      "build:css": "stylus -u nib -u typographic app/styles/main.styl",
      "watch:css": "stylus -u nib -u typographic -w app/styles/main.styl",
```

```
        "build:js": "uglifyjs --compress --mangle -- app/js/main.js >
            app/js/main.min.js",
        "build": "npm run build:css"
    }
}
```

Code 10: BOINSO MCC Web Client package configuration. Used to include project specific meta data, deployment and development dependencies, and script definitions.

2.3.4 Bower

Bower is a package manager similar to npm. Traditionally it is used in situation in which pure front-end components are installed without the need of a deep dependency tree. Bower can be configured at a per project basis with a configuration file as displayed in listing 11.

```
{
  "name": "BOINSO_MCC_WEB_CLIENT",
  "version": "0.0.1",
  "homepage": "https://github.com/WalternativE/BOINSO_MCC_WEB_CLIENT",
  "authors": [
    "WalternativE <gregor@beyerle.at>"
  ],
  "description": "BOINSO MCC Management WebApp",
  "keywords": [
    "BOINSO MCC Client"
  ],
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "test",
    "tests",
    "app/components/"
  ],
  "dependencies": {
    "a0-angular-storage": "~0.0.10",
    "angular-bootstrap": "~0.12.1",
    "angular-ui-router": "~0.2.13"
  },
  "license": "Apache 2.0"
}
```

Code 11: BOINSO MCC Web Client bower configuration. Used to include project specific meta data in case the deployment to the bower package index is a project goal as well as front-end dependencies.

2.3.5 Bootstrap

Bootstrap is is a HTML, CSS, and JavaScript framework for developing responsive, mobile first web applications. It is distributed and maintained by Twitter, available under the Massachusetts Institute of Technology (MIT) license and easy to adjust in pure CSS, Sass or less (both being CSS precompiler languages). It offers an easy to use grid system and a tool set of isolated responsive components as well as commonly used utility classes.

2.4 Boinso GPredict Bridge

This section focuses on the tools used in the conception of the piece of middleware that connects Ground Control Centers (GCCs) using GPredict to their related Mission Control Centers (MCCs).

2.4.1 GPredict

GPredict is an application which allows users to track orbiters communicating elevation and azimuth values to the rotator control daemon (a Ham Radio Control Libraries (HamLib) utility) while also transferring Doppler shift parameters – alternation of radio frequencies due to high velocity movement of orbiting objects – to the rig control daemon (a HamLib utility controlling the radio). Independent from the way in which the client is finally implemented many parts of the aforementioned projects should be incorporated into the project because they have an extensive community and decrease the projects complexity immensely.

The BOINSO project relies on a modified version of GPredict which automatically switches between a list of orbiters which is stored in a module definition. Those orbiters are represented by their file representations in the GPredict data directories and might additionally include transponder definitions which can be used to fine tune the Doppler shift calculations.

3 Methods

The following sections include the steps taken to create the BOINSO network applications and the way the distinct components exchange information.

3.1 BOINSO Core Web Application

3.1.1 Core Data Model

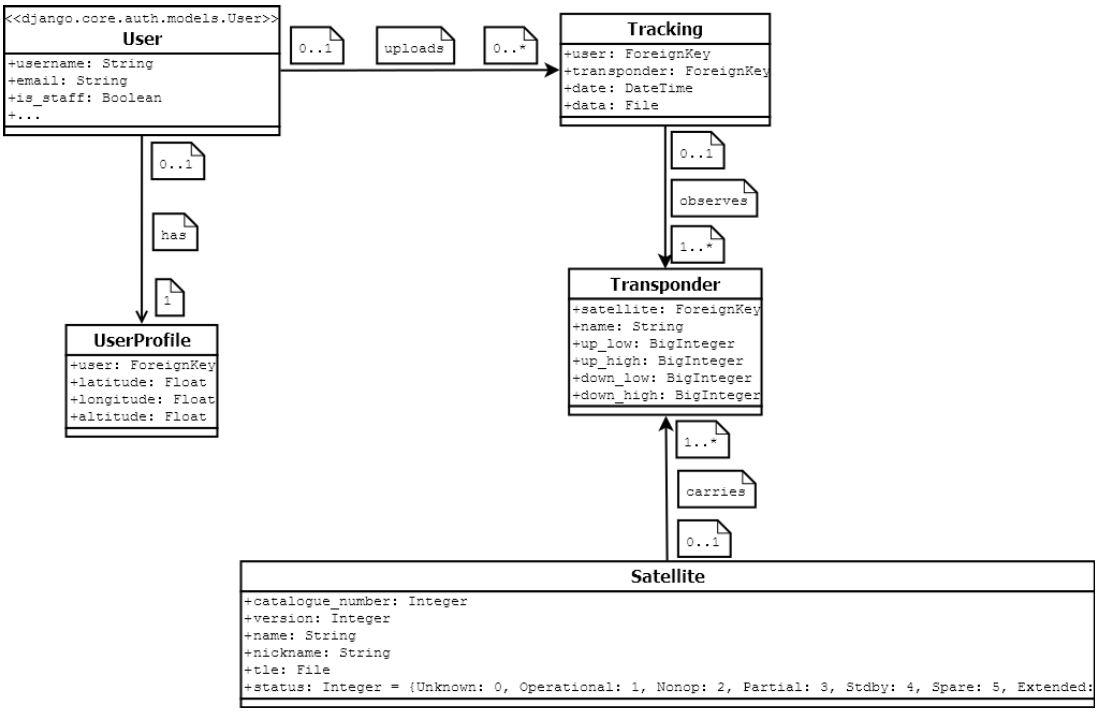

Figure 2: BOINSO Core Web Application model graph

As seen in figure 2 the model graph was modeled with only a view important entities as the initial functionality only includes passive satellite passes – meaning the tracking of a reoccurring transponder transmission. The Django core implementation of the user model was incorporated into the graph as it already included extensive integration in the authentication and permission system. The extension of the user profile was used to add Ground Control Center (GCC) related information which offers researches means for statistical segmentation.

3.1.2 Data model migrations

In order to implement changes to the data model (during initial development as well as the future) Django's migration system was used. The migration system created script files which allowed to make substantial changes to the underlying data source in a portable and retractable manner. A typical migration file as seen in listing 12 provides semantic naming, chronological enumeration and the possibility to transport model and data changes to all collaborating developers using the common source code repositories.

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models, migrations


class Migration(migrations.Migration):

    dependencies = [
        ('core', '0003_satellite'),
    ]

    operations = [
        migrations.AddField(
            model_name='satellite',
            name='catalogue_number',
            field=models.IntegerField(default=1),
            preserve_default=False,
        ),
    ]
```

Code 12: BOINSO Core Web Application model migration file including the addition of an integer typed catalogue number field to the satellite model.

3.1.3 Web API

Table 1 shows the different Application Programming Interface (API) endpoints which were implemented to interact with a Mission Control Center (MCC) in a programmatic way using Hypertext Transfer Protocol (HTTP) verbs. As almost every programming language has a certain amount of networking capabilities being able to make HTTP calls it was seen as the most reasonable approach to make the client development as open as possible. In order to make the API usable for open scripted clients as well as closed compiled clients an implementation of the OAuth2 authorization framework as stipulated in [8].

In order to keep users relatively independent a user profile (once registered at the MCC web application) poses as an application (using the OAuth2 vernacular) making the access to

Method	Endpoint	Usage	Returns	Auth
POST	/api/sign_up/	Sign up new GCC	OAuth2 credentials	None
GET	/api/login/	Retrieve registered GCC	OAuth2 credentials	HTTP Basic
GET	/api/satellites/	Retrieves list of available satellites	Array of satellites	None
GET	/api/satellites/:id/	Retrieves satellite with specific ID	Satellite	None
GET	/api/transponders/:id/	Retrieves transponder with specific ID	Transponder	None
GET	/api/user-profiles/	Retrieves user related to auth token	User profile	OAuth2
GET	/api/user-profiles/:id/	Retrieves user profile with specific ID	User profile	OAuth2
PUT/PATCH	/api/user-profiles/:id/	Updates user profile with ID	User profile	OAuth2
DELETE	/api/user-profiles/:id/	Deletes user profile with specific ID	Deleted Notification	OAuth2

Table 1: Web API specifications listing method with HTTP-verbs relative endpoint URL a short usage note a short description of the returned values and the used authentication type

OAuth2 protected endpoints uniform on all client implementation. The OAuth2 provider used in the Boinso Core Web Application was added using the extensive Django OAuth Toolkit distributed under the Berkeley Software Distribution (BSD) license. Utilizing the capabilities of the provided tools it was possible to both implement the Boinso MCC Web Client as well as the BOINSO GPredict Bridge with a web native callback driven authentication work-flow even though the initial configuration of the BOINSO Core Web Application was considerably more complex.

To guarantee a secure transmission of authentication data (login calls require HTTP Basic authentication which is relatively easy to decipher while OAuth2 access tokens are not required to be encrypted) a Secure Socket Layer (SSL)/Transport Level Security (TLS) encrypted connection should be used. MCCs can create their own certificates but should also consider investing in signed certificates by accredited institutions.

Besides the functionality of the web API and its versatile authentication system another feature of the used components include an automatically generated browsable API. It was not part of this project to change the design or branding which leads to the API root in figure 3 stills showing the Django Rest Framework base style.

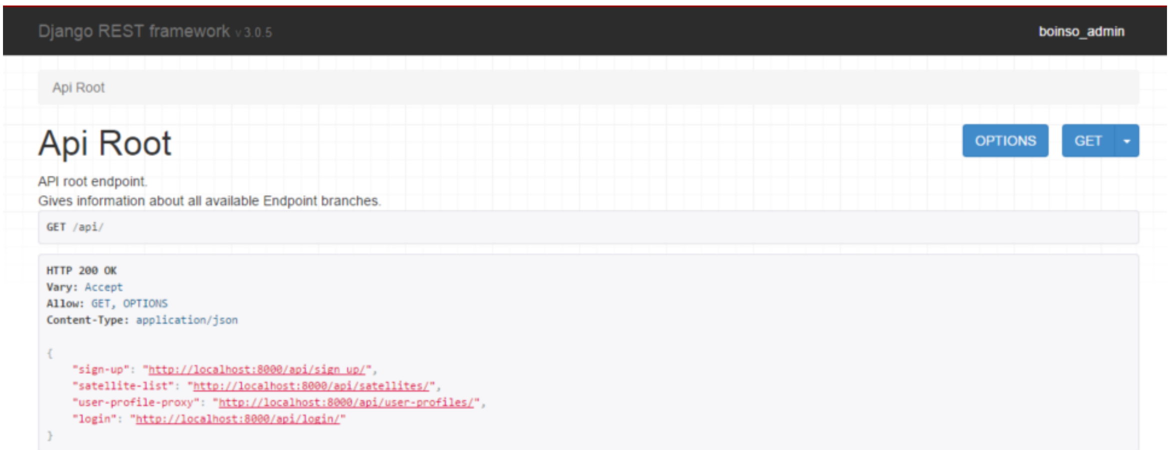

Figure 3: Browsable API root view. API components are interconnected by URLs allowing users both manually as well as programatically discovering API components.

3.2 BOINSO MCC Web Client

The BOINSO MCC Web Client was constructed as a sample client implementation, an easy to use and extend web application template for MCCs and the first interaction point for GCCs. The main focus was to keep it as simple as possible while giving users a intuitive experience making it possible for them to register with a MCC, update their profile information and get informed about the MCC's satellites.

Using AngularJS, it's ngResource module and the capabilities of the Django Cross-Origin Resource Sharing (CORS) Headers module (in the BOINSO Core Web Application) the application was build to request and load user data – if the authentication process resulted in positive application feedback – asynchronously and rather than interrupting user commands by alerts in cases of failure redirecting him or her to a state where the problem can be countered.

In AngularJS this behavior could be implemented by using HTTP interceptor chains like seen in listing 13 making extensive use of lazy dependency injection to avoid the risk of circular constructor dependencies.

```
$httpProvider.interceptors.push(['$injector', function($injector) {
    return {
        responseError: function(rejection) {

            // I inject these services via implicitly calling the $injector service
            // if I don't do it like this I get a circular dependency error
            // I wish that some day I fully understand my doings
            var authService = $injector.get('authService');
            var $state = $injector.get('$state');
            var store = $injector.get('store');
            var $q = $injector.get('$q');

            if (rejection.status === 401) {
                $state.go('home');
                store.remove('auth_token');
                authService.login();
            }

            return $q.reject(rejection);
        }
    };
}]);
```

Code 13: Example for a HTTP interceptor reacting to HTTP 401 messages redirecting the user to a log-in overlay.

3.3 BOINSO GPredict Bridge

The BOINSO GPredict Bridge was build as a small functional piece of middleware connecting the BOINSO Core Web Application with the user's GCC client machine. Private radio enthusiasts tend to invest a lot of time in their – quite expensive – observation set-ups. The machine controlling the hardware (radio rig, rotator controller and eventually additional peripherals) are optimized for maximal stability which makes their maintainers very skeptical of new components. Considering the module had to be easy to install, update and use as well as being very portable (supporting even Windows XP machines had to be considered) the application was written in pure Python with only few and rather common dependencies (like the formidable requests module).

As the idea behind the module was to introduce satellite tracking automation it was constructed that – if configured correctly – can be run autonomously via Unix cron jobs or Windows services. The configuration is generated in the users home directory (as for instance a vim.rc file would be) and contains the user information, the root URLs of the affiliated MCCs as well as related health and skip flags. The configuration are written as valid JavaScript Object Notation (JSON) files otherwise the configuration parsing fails.

It was the projects goal to produce a python package that is not only a collection of scripts but rather a piece of software that can be distributed to the Python Package Index (PyPI) allowing users to use management tools like the in subsection 2.2.1 mentioned pip. To reach this goal a setup file as seen in listing 14 was included to the project root.

```python
# Always prefer setuptools over distutils
from setuptools import setup, find_packages
# To use a consistent encoding
from codecs import open
from os import path

here = path.abspath(path.dirname(__file__))

# Get the long description from the relevant file
with open(path.join(here, 'DESCRIPTION.rst'), encoding='utf-8') as f:
    long_description = f.read()

setup(
    name='boinsogpredictbridge',
    version='0.1.0',
    description='Utility bridge between BOINSO and GPredict',
    long_description=long_description,

    # The project's main homepage.
    url='https://github.com/WalternativE/boinso-gpredict-bridge/',

    # Author details
    author='Gregor Beyerle, UAS Technikum Wien',
    author_email='gregor@beyerle.at',

    # Choose your license
```

```
    license='Apache Software License',

    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',

        # Indicate who your project is intended for
        'Intended Audience :: Developers',
        'Intended Audience :: Science/Research',

        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: Apache Software License',

        # Specify the Python versions you support here. In particular, ensure
        # that you indicate whether you support Python 2, Python 3 or both.
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.2',
        'Programming Language :: Python :: 3.3',
        'Programming Language :: Python :: 3.4',
    ],

    # What does your project relate to?
    keywords='boinso gpredict satellite observation',

    # You can just specify the packages manually here if your project is
    # simple. Or you can use find_packages().
    packages=find_packages(exclude=['contrib', 'docs', 'tests*']),

    # List run-time dependencies here.  These will be installed by pip when
    # your project is installed. For an analysis of "install_requires" vs pip's
    # requirements files see:
    # https://packaging.python.org/en/latest/requirements.html
    install_requires=['requests', 'enum34'],
)
```

Code 14: Setup script for Python package

4 Results

As the results of a software engineering project – in the most favorable cases – are functioning software the following paragraphs include display the different project components working together. In the following paragraphs it will be shown how a Mission Control Center (MCC) can add a satellite with a single transponder in the administrative interface as well as a Ground Control Center (GCC) registering with the MCC, and changing his user profile both via the BOINSO MCC Web Client or – modeling an arbitrary client implementation – directly via the web Application Programming Interface (API).

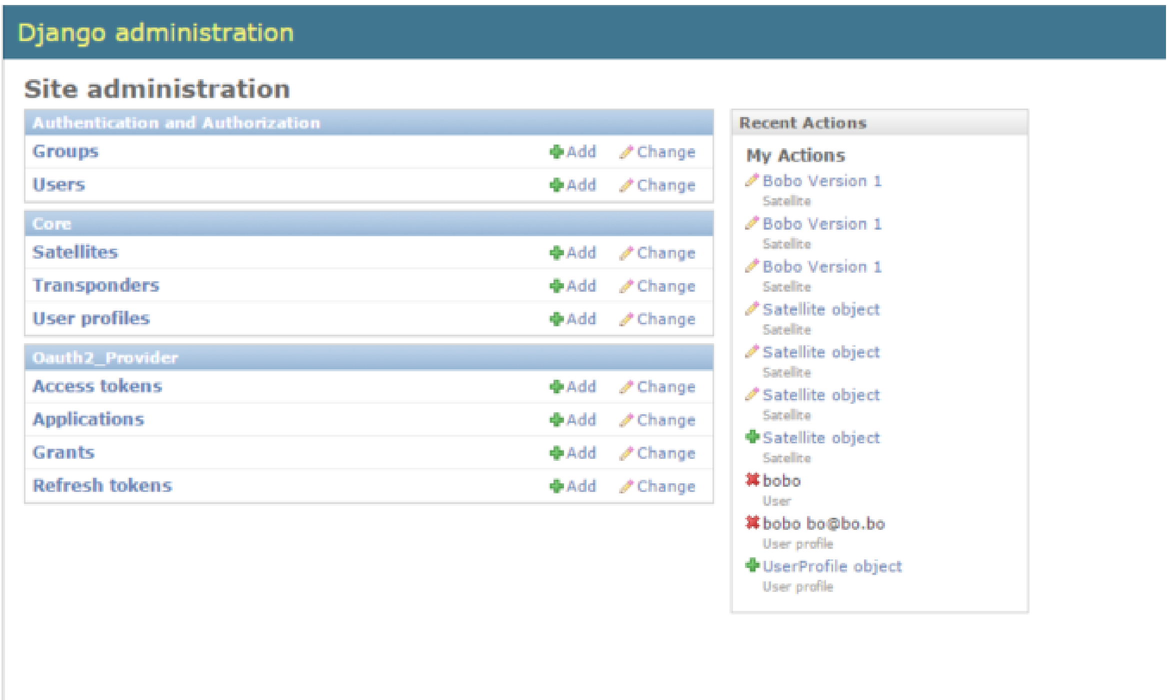

Figure 4: BOINSO Core Web Application main administrative page listing the different modules and tasks

Figure 4 shows the administrative page of a MCC administrator. The different groups separate the installed Django modules. A developer can decide which modules are presented and how they are displayed. The Django administrator pages are extremely extendable and can be configured for the most administrative tasks – like in this case adding a new satellite.

In figure 5 the form which is used to ad a satellite and its related transponders. The administrative forms are automatically generated by the Django internals after the underlying data model. Inline forms – like in this case the transponder form – are not standard Django behavior

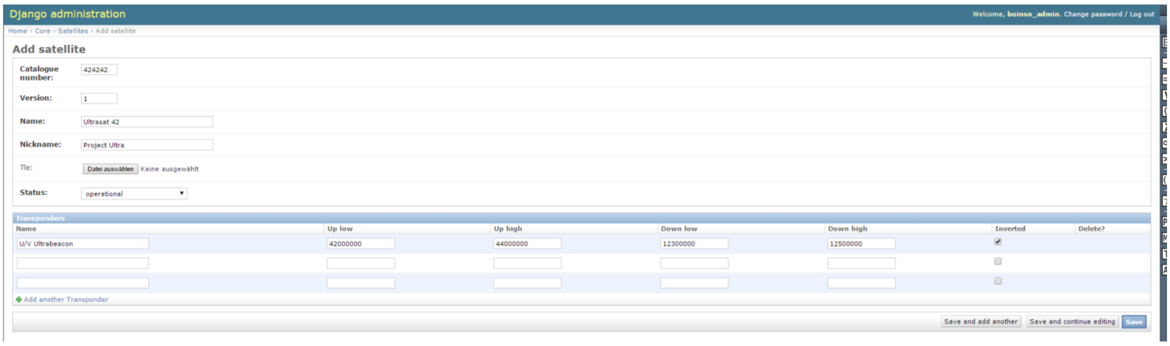

Figure 5: Django admin satellite management form with inline transponder form.

but an be explicitly set as shown in listing 15 which also shows how other custom modules are registered with the administrative interface.

```
from django.contrib import admin
from core.models import UserProfile, Satellite, Transponder


class TransponderInline(admin.TabularInline):
    model = Transponder


class SatelliteAdmin(admin.ModelAdmin):
    inlines = [
        TransponderInline,
    ]

admin.site.register(UserProfile)
admin.site.register(Satellite, SatelliteAdmin)

# there is really no need to work on an Transponder individually as
# they are managed inline with the satellite objects - but hey - choices!
admin.site.register(Transponder)
```

Code 15: Python script explicitly configuring the Django administrative interface

As seen in figure 6 the newly created satellite is listed and can be modified or deleted. Sorting options as well as search fields can also be added to the Django administrative interface. In this case it is not very likely for a MCC to have many satellites so the added complexity is not needed.

From a client perspective the main entry point would be the MCC's web page – in this case modeled by the BOINSO MCC Web Client. Figure 7 shows the homepage where every user is informed about the available satellites as well as motivated to participate.

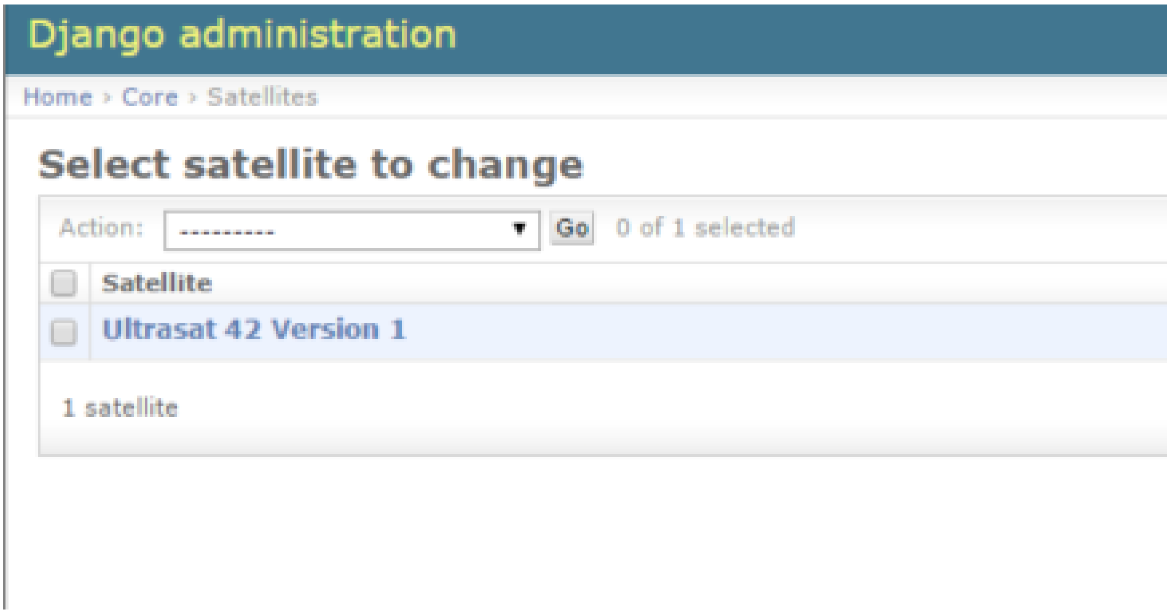

Figure 6: List of existing satellite objects listed by their model representation

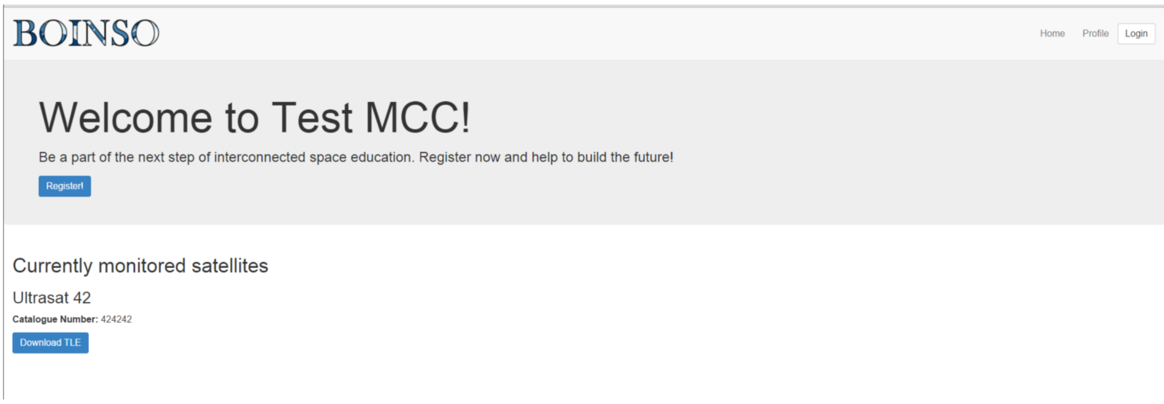

Figure 7: BOINSO MCC Web Client homepage

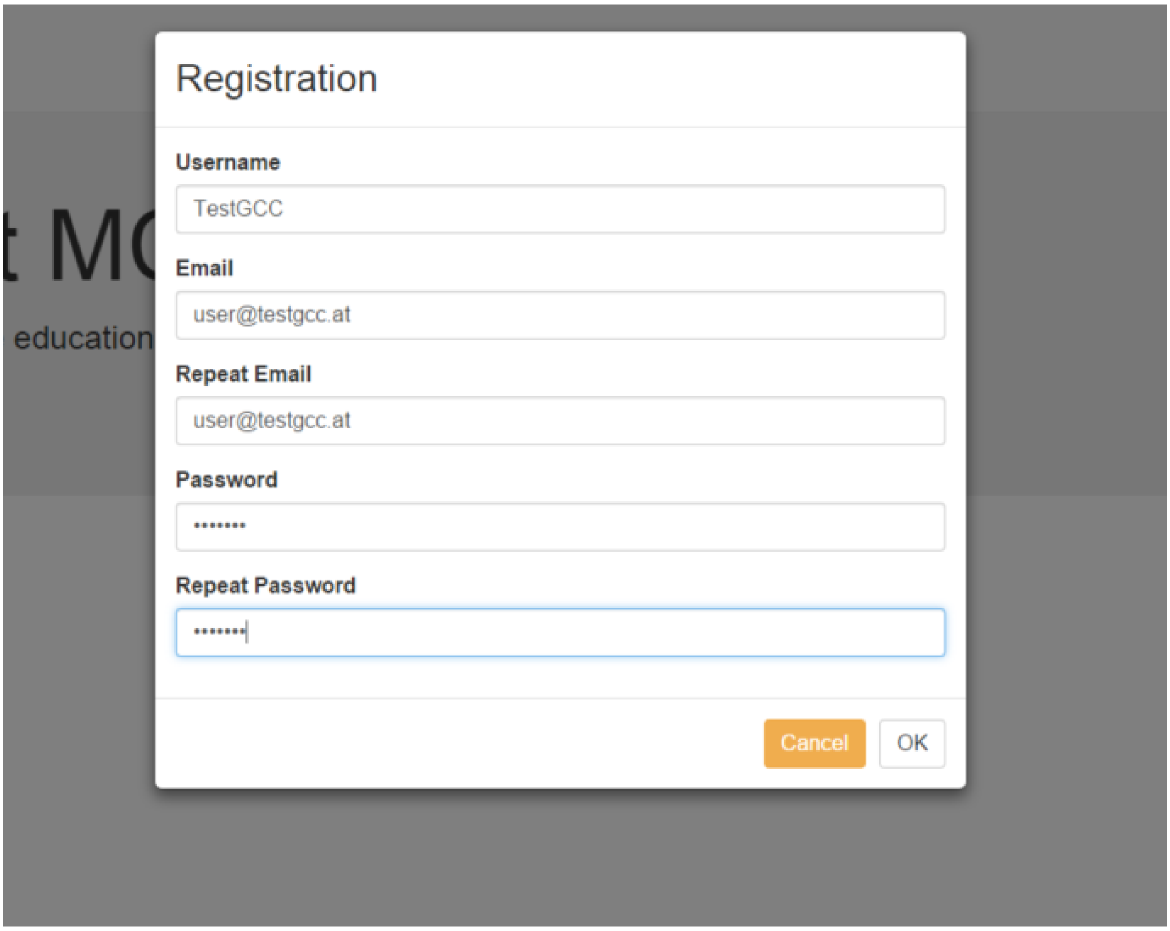

Figure 8: BOINSO MCC Web Client registration form

The registration form as seen in figure 8 collects the most vital information about new users. A successful user creation automatically creates a connected OAuth2 application which provides the client id and the client secret. In the web application those more complex communication steps are hidden from the user.

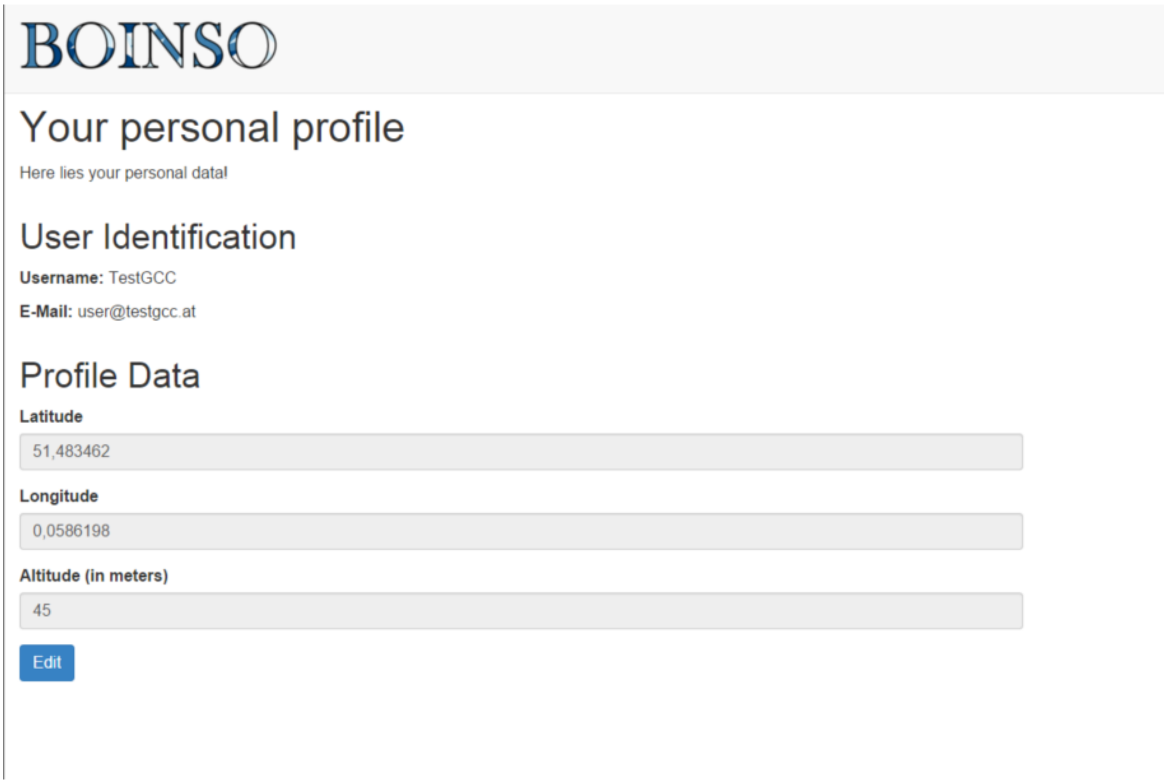

Figure 9: BOINSO MCC Web Client profile page

A logged-in user may access his or her profile page as seen in figure 10. A user without proper authentication – this includes registered users with a timed out access token – are redirected to the home page where the log-in form is opened.

As the initial user profile always includes the geo-location of Greenwich – just by convention to make the initial registration easier for new users – users have the possibility to change their profile data as seen in figure 10.

One of the projects main goals is to give users the possibility to write their own clients – or fill the gaps between legacy software and the BOINSO network with middleware as we did with the BOINSO GPredict Bridge. To simulate an arbitrary client the Representational State Transfer (REST) API testing tool Postman was used. The registration process is a straight forward POST call to the related API endpoint as seen in figure 11.

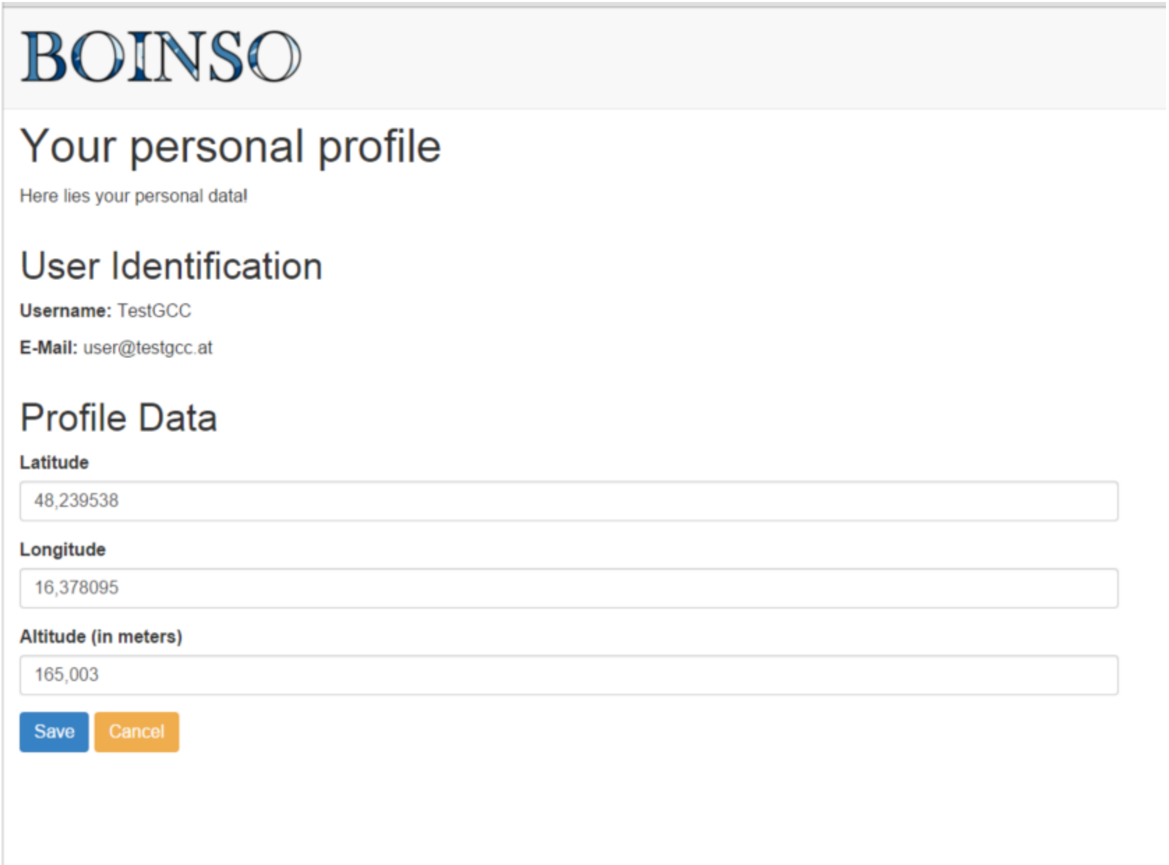

Figure 10: BOINSO MCC Web Client profile data management form

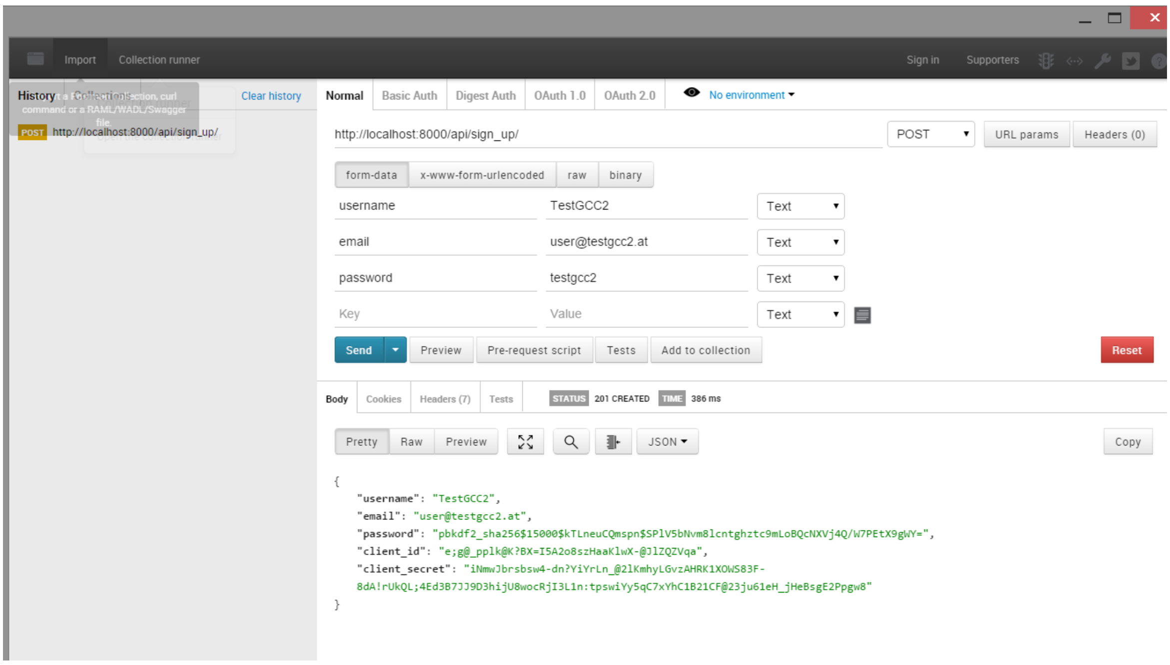

Figure 11: The registration of a new user via Postman

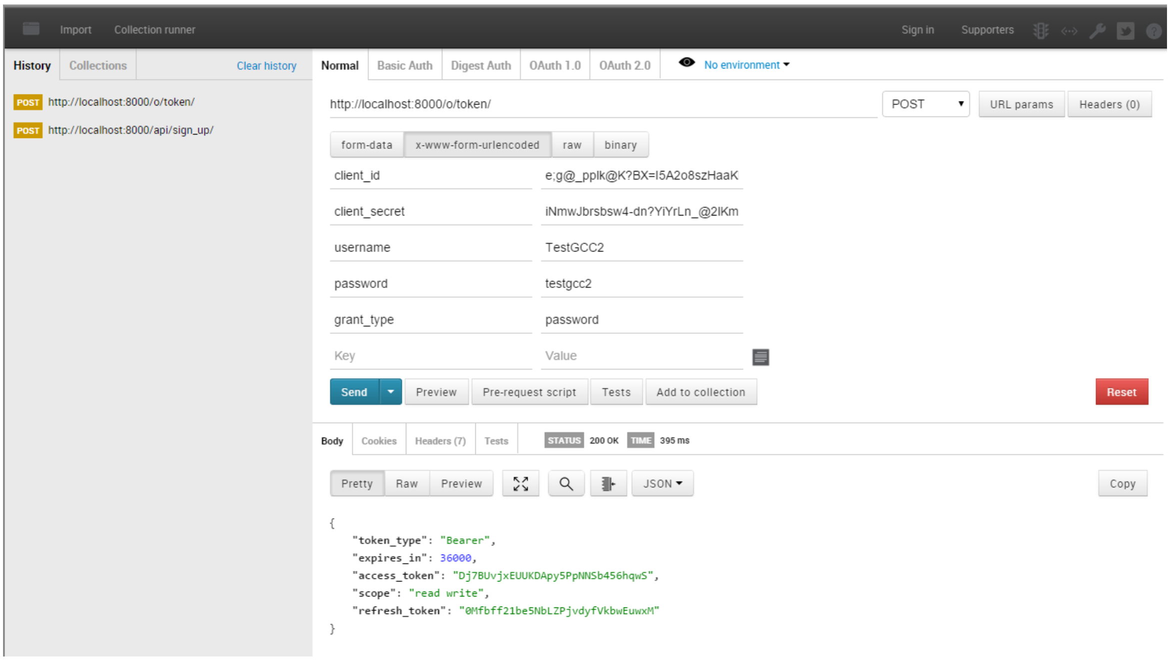

Figure 12: Retrieval of the user OAuth2 access token via Postman

As seen in figure 11 the response to the registration includes both the client id as well as the client secret. In combination with the user name and user password a call to the Uniform Resource Locator (URL) of the OAuth2 provider endpoint – implemented by the Django OAuth Toolkit – results in the retrieval of a distinct authentication token (with a limited lifetime) as seen in figure 12.

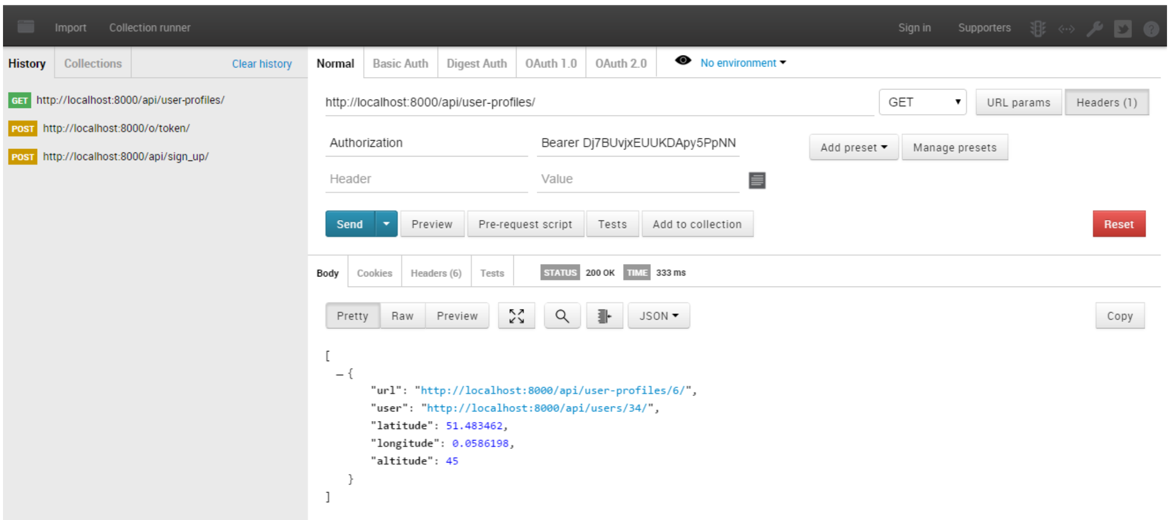

Figure 13: Retrieval of the user profile belonging to the OAuth2 authentication token

As user credentials tend not to change often client designers are not encouraged to safe them after accessing a user token as a lost token can be rather retrieved rather easily whereas lost credentials impose a major security threat. Figure 13 proves that the OAuth2 token – assumed it is neither timed out nor lacking bearer rights – is enough to retrieve a user's profile.

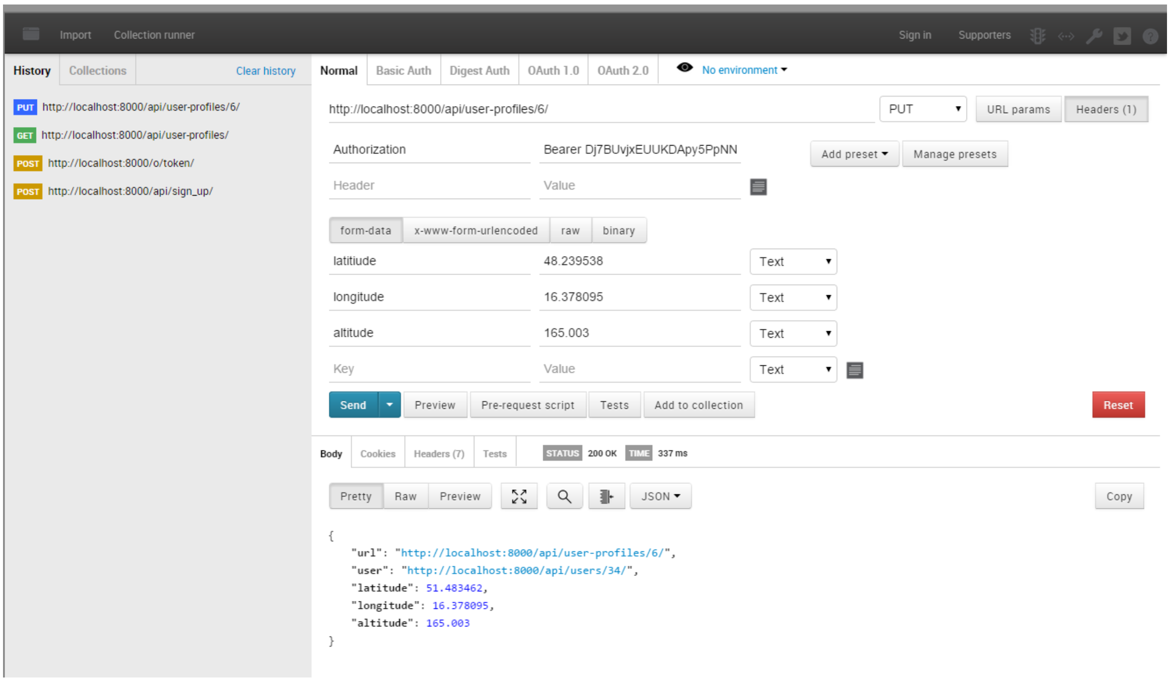

Figure 14: Update of a distinct user profile via Postman

Finally the information about the retrieved user profile – the user profile url field – can be used as the next endpoint, proving the conceptual benefits of Hypermedia as the Engine of Application State (HATEOAS) in action. Using the Hypertext Transfer Protocol (HTTP) verb PUT new information – as long as they are in a valid format – can be directed at the user profile endpoint resulting in updated user profile values as seen in figure 14.

5 Discussion

During the development of this application I faced situations in which I found myself lost in the complexity of modern web-centric software engineering. Especially the complex asynchronous nature of web communication and the problematic handling of information state and client authentication often blocked my advances but after finishing the first major step – with the implementation of the first two main components the BOINSO Core Web Application and the BOINSO MCC Web Client – I feel confident that this project can be a useful base for future development.

The majority of tools (and many more not used in this project) where new to me and it took some time to get accustomed to the different programming paradigms but I think that structured as they are they are a relatively future proof foundation for both the project and the way Mission Control Centers (MCCs) and Ground Control Centers (GCCs) are going to collaborate in the future.

The most problematic step is still the full implementation of the BOINSO GPredict Bridge package as it is a crucial part of the collaborative process. It still does not have enough functionality to be convenient for GCC administrators who may be technical enthusiast but still favor the most comfortable solution. Also still not optimally solved is the upload of tracking data as this component is still too dependent on the output of the radio (or software defined radio solution). Right now there does not exist enough data about GCCs who use other solutions than FlDigi or similar products and the actual monitoring work-flow is still to obscure.

The next steps include the founding of partnerships with interested universities in Portugal, Denmark and Japan which have to be introduced to the general concept of the project and the further development. If this step can be taken and my old mistakes – the code base is still by far not tested enough and the core model graph definitely needs a second opinion – can be re-factored I see a good and prosperous future for the BOINSO project and maybe even new affiliated programs.

As a part of the biomedical engineering community I am especially interested in the future medical applications that can be build on our technology. As far as I can see – if we succeed – the efficiency of educational satellite projects should increase dramatically which leads to an increased amount of available satellite time and a bigger incentive for new satellite projects being launched. The additional satellite time can be used to consolidate programs and reserve time for testing transmission of medical data over satellites in Low Earth Orbit (LEO). Colleagues have already proven that medical images – like X-Rays of fractured bones – can be transported via small radio devices in a timely manner. Regions without any other means of connection could upload and receive information using the nearest GCC with actual internet connection as a proxy.

Bibliography

[1] C. Kief, R. Buffington, N. Purushotham, R. S. Erwin, J. Androlewicz, J. Lyke, and J. Jackson, "GENSO, SPA, SDR and GNU radio: The pathway ahead for space dial tone," *Infotech@Aerospace*, vol. 2011, 2011. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2011-1596

[2] G. Beyerle, "An open source solution for distributed ground and mission control communication for low earth orbit satellites – a feasibility study," Wien: FH Technikum Wien, Bachelorstudiengang Biomedical Engineering, 2014. [Online]. Available: https://bitbucket.org/WalternativE/bachelor1

[3] git-scm user group. (2014) Git - the stupid content tracker. [Online]. Available: https://github.com/git/git

[4] Free Software Foundation. (2014) How to choose a license for your own work. [Online]. Available: http://www.gnu.org/licenses/license-recommendations.html

[5] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[6] P. J. Eby. (2003) Pep 0333 – python web server gateway interface v1.0. Python Foundation. [Online]. Available: https://www.python.org/dev/peps/pep-0333/

[7] E. Glazkov. (2014) Custom elements. World Wide Web Consortium. [Online]. Available: http://www.w3.org/TR/custom-elements/

[8] E. D. Hardt. (2012) The oauth 2.0 authorization framework. Internet Engineering Task Force (IETF). [Online]. Available: http://tools.ietf.org/html/rfc6749

List of Figures

List of Tables

List of Code

List of Abbreviations

API Application Programming Interface

GENSO Global Educational Network for Satellite Operations

BOINC Berkeley Open Infrastructure for Network Computing

LEO Low Earth Orbit

GEO Geostationary Orbit

GPL GNU General Public License

MCC Mission Control Center

GCC Ground Control Center

COTS Commercial off-the-shelf

ISEB International Space Education Board

CSA Canadian Space Agency

ESA European Space Agency

JAXA Japan Aerospace Exploration Agency

NASA National Aeronautics and Space Administration

CPU Central Processing Unit

GPU Graphics Processing Unit

PRC Public Resource Computing

TNC Terminal Node Controller

LTS Long Term Support

CGI Common Gateway Interface

MPL-2.0 Mozilla Public License Version 2.0

NORAD North American Aerospace Defense Command

HamLib Ham Radio Control Libraries

TLE Two Line Elements

git-scm Git Source Code Mirror

SVN Subversion

GPLv2 GNU General Public Licence Version 2

REST Representational State Transfer

PyPI Python Package Index

FSS Free Software Foundation

ORM Object Relational Mapping

MVC Model View Controller

HTML Hypertext Markup Language

URL Uniform Resource Locator

HATEOAS Hypermedia as the Engine of Application State

HTTP Hypertext Transfer Protocol

WSGI Web Server Gateway Interface

BSD Berkeley Software Distribution

IIS Microsoft Internet Information Services

CSS Cascading Style Sheet

npm Node Package Manager

SSL Secure Socket Layer

TLS Transport Level Security

MIT Massachusetts Institute of Technology

CORS Cross-Origin Resource Sharing

JSON JavaScript Object Notation