Document Classification: **PUBLIC DOMAIN**

SATNET PROJECT

R1 SOFTWARE ARCHITECTURE

REFERENCE: SATNET-3-R1

DATE: NOVEMBER 4, 2013

ISSUE: DRAFT

CUBESAT

CALIFORNIA POLYTECHNIC STATE UNIVERSITY

1 Document Control Data

1.1 Distribution License

1.2 Contact

Cal Poly - San Luis Obispo	Cal Poly - San Luis Obispo	Cal Poly - San Luis Obispo
Dr. Ricardo Tubio-Pardavila	Prof. Jordi Puig-Suari	Prof. John Bellardo
Aerospace Eng. Dept.	Aerospace Eng. Dept.	Computer Science Dept.
(805) 709 1080	(805) 756-5087	-
-	FAX:(805) 756-2376	-
rtubiopa@calpoly.edu	jpuigsua@calpoly.edu	bellardo@calpoly.edu

1.3 Change History Log

Date	Revision	Author	Description of Changes
2013.10.28	DRAFT	rtubiopa@calpoly.edu	Initial edition
2013.11.04	DRAFT	rtubiopa@calpoly.edu	Minor changes and typos correction.

1.4 List of Acronyms

AD	Applicable Document
API	Application Programming Interface
ASR	Availability and Scheduling Rules Set of rules that define the availability and scheduling for sharing a given ground station.
G-Client	Ground Station Client Software component of the SATNet network for ground station operators to share their facilities.
G-Client-IF	Ground Station Client Interface Communications interface provided by the SATNet network to the G-Clients.
G-Operator-UI	Ground Station Operator User Interface User interface for ground station operators to access to the services of the SATNet network.
ISP	Internet Service Provider
M-Client	Mission Operation Client Software component of the SATNet network for satellite operators to utilize remote ground station facilities.
M-Client-IF	Ground Station Client Communications interface provided by the SATNet network to the M-Clients.
N-System	Network Communications System Cloud-computing based component of the SATNet network for interconnecting G-Client(s) and M-Client(s).
Non-SCS	Non-Scheduled Communications Service Communications service provided by the SATNet network for permitting ground stations to store data received without previous request by spacecraft operators.

Pre-SCS	Pre-Scheduled Communications Service Communications service provided by the SATNet network for enabling the data messages exchange between spacecraft operators and ground stations.
RD	Reference Document
S-Operator-UI	Spacecraft Operator User Interface User interface for spacecraft operators to access to the services of the SATNet network.
SATNet	SATellite Network
SOR	Spacecraft Operation Request Request placed on the N-System by satellite operators for requesting the utilization of certain ground stations.
TBC	To Be Confirmed
TBD	To Be Determined
TBW	To Be Written
UHF	Ultra High Frequency

1.5 Table of Contents

Contents

1.6 Applicable Documents

ID	Title	Reference	Author	Issue
AD-0	Project Management Plan	satnet-0	CalPoly - rtubiopa@calpoly.edu	TBD
AD-1	User Specification	satnet-1	CalPoly - rtubiopa@calpoly.edu	DRAFT
AD-2	R1 Release Specification	satnet-2-R1	CalPoly - rtubiopa@calpoly.edu	DRAFT
AD-4	R1 Testing	satnet-4-R1	CalPoly - rtubiopa@calpoly.edu	DRAFT

1.7 Reference Documents

ID	Title	Reference	Author	Issue
RD-0	Space Engineering - System Engineering General Requirements	ECSS-E-ST-10C	ECSS - www.ecss.nl	C
RD-1	Space Engineering - Technical Requirements Specification	ECSS-E-ST-10-06C	ECSS - www.ecss.nl	C
RD-2	Space Engineering - Software	ECSS-E-ST-40C	ECSS - www.ecss.nl	C
RD-3	Space Engineering - Ground Systems and Operations	ECSS-E-ST-70C	ECSS - www.ecss.nl	C
RD-4	UML Specification	OMG web	www.omg.org	2.4.1
RD-5	Style Guide for Python Code	PEP-08	www.python.org	2013-08-01

1.8 Object & Scope

The object of this document is to provide a software architecture for meeting the objectives of the release 1, as per document AD-2. Therefore, this document must include a clear definition of the software components that the software for the release 1 must be composed of, together with the main requirements for the development methodology to be used. All implementation decisions made along this document must be dully justified. In addition to this, the technologies for this implementation must also be selected.

The contents of this document apply during the development phase for the release 1 of the software. However, they may vary from the initial definition as the implementation of the release goes on.

This document is divided in the following main sections:

1. A first section in where the components-based architecture for the software is depicted. In this section, the functionalities that each component must carry-out, the associated activity diagram and the interaction with the rest of the system, are described.
2. A second section that contains the previous definitions in terms of requirements for each of the software components to be implemented. This will be the input for the Integration document ([AD-4]), that will focus its contents on the definition of the verification of the requirements from this section.

2 System Architecture

2.1 Network Elements

For the release 1, the network elements considered are the following:

1. The cloud N-System will be composed of a single central server known as the **Network Server** element, or **N-Server** for short.
2. Mission operators (M-Operators) will use the **M-Client** element to access to the services provided by the central N-Server.
3. Ground Station operators (G-Operators) will use the **G-Client** element to access to the services provided by the central N-Server.

Since the objective for this project (in accordance with requirements from AD-1) is to develop only the network definition and implement the required network infrastructure, none of the client elements will be implemented. Only testing applications will be developed for performing tests on the final N-Server element. These testing clients will be distributed, but only for testing issues. Therefore, final users of the network must develop their own clients, even though they may use these testing applications as a starting point.

Figure below depicts this topology for the release 1.

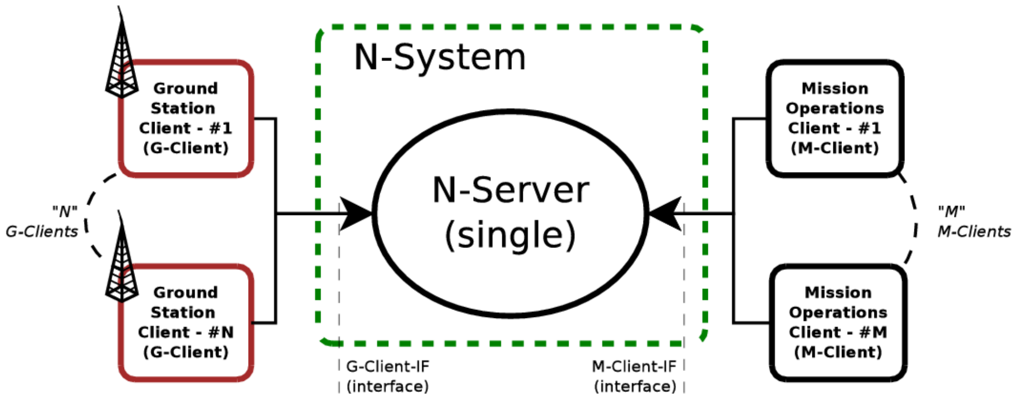
N-System
Ground Station Client - #1 (G-Client)
Ground Station Client - #N (G-Client)
"N" G-Clients
N-Server (single)
Mission Operations Client - #1 (M-Client)
Mission Operations Client - #M (M-Client)
"M" M-Clients
G-Client-IF (interface)
M-Client-IF (interface)

Figure 1: Network topology

2.2 N-Server Element

The software architecture for this element of the network is composed of the following software components:

- The **WebServices** software component, a web application developed under the Django Framework. This web services provide a Graphical User Interface (GUI) to all the users of the N-Server: M-Operators, G-Operators and N-Operator.
- The **Protocol** software component that is a Python application daemon running on the Debian OS. This is the software component responsible for exchanging data in between G- and M- software clients.

The utilization of a separated software component for implementing the services offered to software clients, is required in order to reduce the overhead of utilizing an over-HTTP protocol. This way, the protocol daemon can reduce the server response time by using a lightweight TCP based remote protocol and improve the network performance. This protocol will be based in the usage of *AMP RPC protocol* **(TBC)**, provided by the Twisted Network Framework (see *Annex A* of this document for further details on the protocol selection).

The utilization of a Web Development Framework like Django will ease the implementation of the user interfaces and the information storage in a database like MySQL. Native database access libraries included in Python will permit an easy-access also to the information of the database by re-using the SATNet Model DB also in the Protocol component.

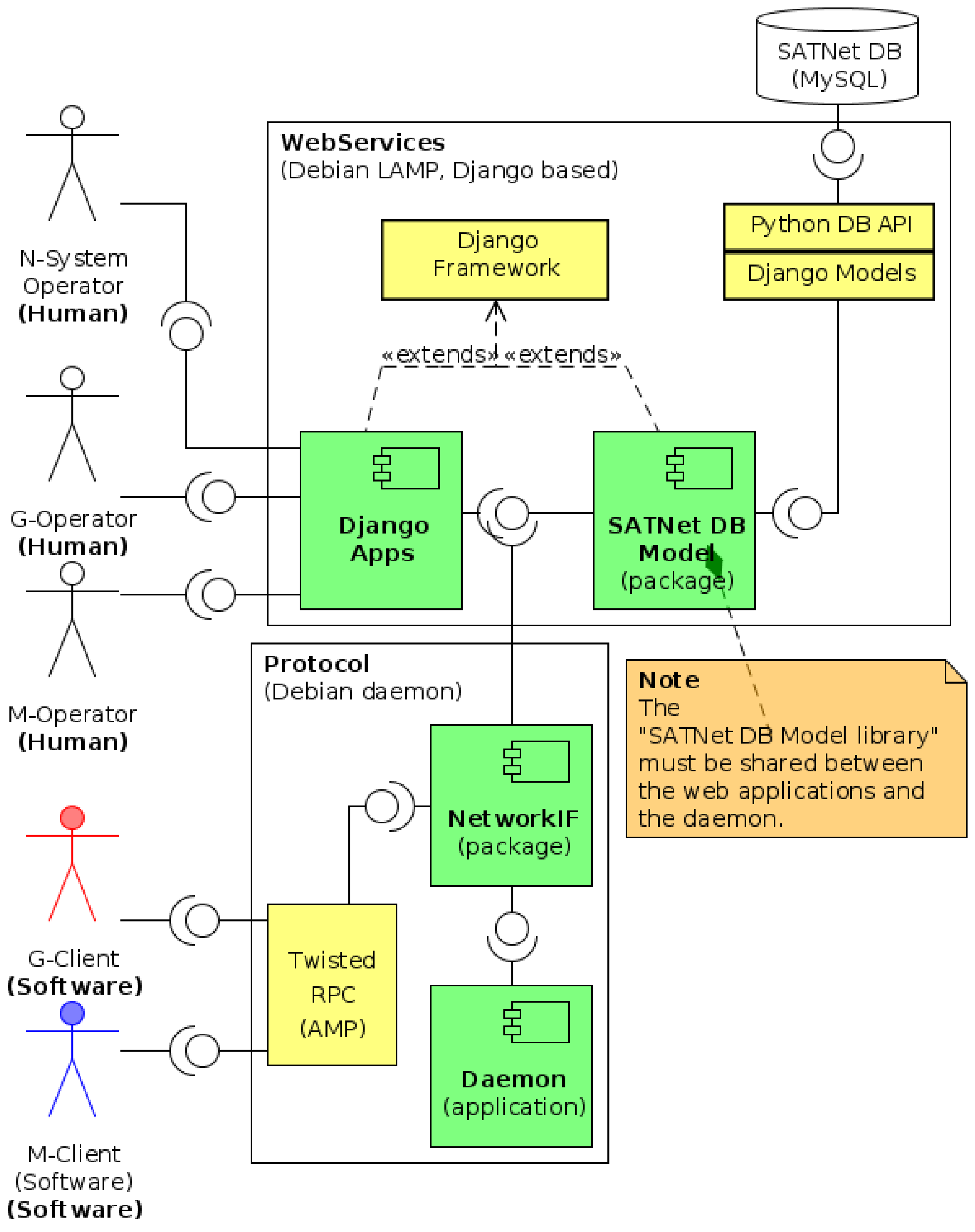

Figure 2: N-Server architecture

The WebService component is composed of the following libraries, since it must follow the MVC model for the Django Framework:

- The **SATNet DB Model** Python package based on the Models of the Django Framework for Python objects direct mapping to a database.
- A **WebServices** set of Django applications, with the views and controllers required.

The Protocol component is composed of the following libraries and applications:

- A **NetworkIF** Python package with the callback functions for processing the remote calls.
- A **Protocol** component that includes the main execution loop for this daemon, together with the link for the usage of the SATNet DB model library. It also holds the runtime data of the application and is responsible for its startup, which involves reading configuration, processing command line arguments, operating system integration and setting up the required logging system.

All the interfaces among these components will be defined in the requirements section.

2.3 Protocol Messages Sequence

This section contains the definition of the remote procedure calls to be invoked by either by client elements or by the server, for a bidirectional communication across the network. The remote procedure calls to be implemented by servers are the following:

- startRemote(), synchronous: whenever a client (either G- or M-) wants to connect to the central server for starting a remote spacecraft operation, with the remote client whose identifier is “remote-id”. ** Returns a code indicating whether the slot has ended or not, and whether the other client required for the remote operation is still connected or not.
- sendMsg(), asynchronous: invoked for a client to send a message to the remote entity.
- endRemote(), asynchronous: invoked by a client whenever this one wants to finalize the remote operation.

The remote procedure calls to be implemented by clients are the following:

- notifyMsg(), asynchronous: used by the N-Server for sending the message to the remote client that must receive it.
- notifyConnection(), asynchronous: used by the N-Server for notifying to a given client the connection of the additional remote client.
- notifyError(), asynchronous: used by the N-Server for notifying clients about an error in the network.
- notifySlotEnd(), asynchronous: notifies to a client the end of the operations slot.

A more detailed and precise information of these remote procedures and the data types involved is provided in the requirements section.

The protocol defined for the adequate sequence of calls to these remote procedures is defined in the figure below, in which a typical remote operations example is shown.

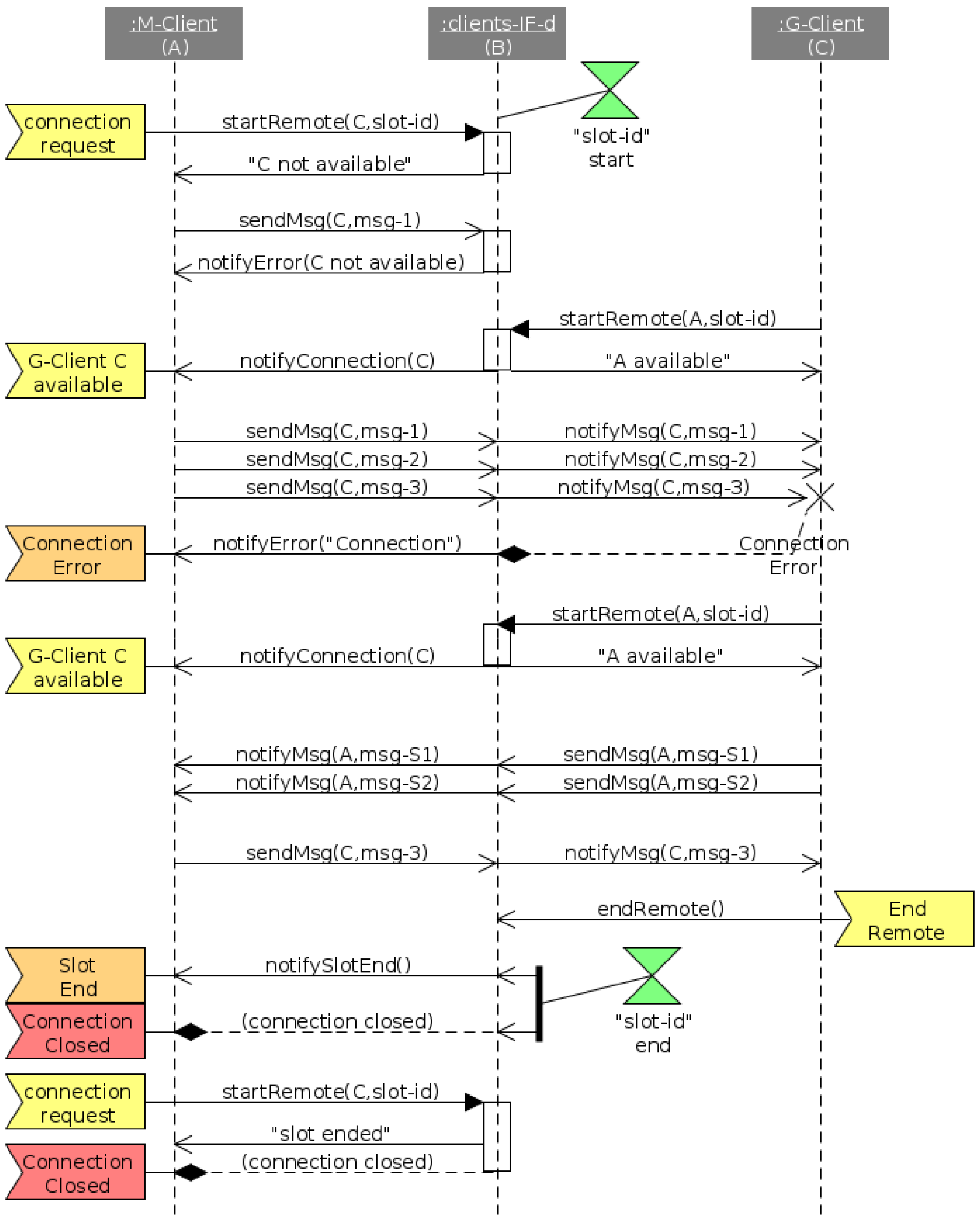

Figure 3: Remote Procedure Calls

2.4 Activity Diagrams

The following flowcharts represent the implementation of the activities that the software components must carry out for implementing the required functionalities, as per [AD-2]. The diagrams here included are the following:

- Protocol Flowcharts:
  - *Software Clients Connection*: activities for when software clients connect to the N-Server.
  - *Data Messages Exchange*: activities to execute for messages transmission in between clients.
- WebServices Flowcharts:
  - *Users Registration*: a flowchart depicting what tasks are performed for registering new system users.
  - *Configuration*: defines the tasks to be performed for users to configure spacecraft and ground stations through the WebServices component.
  - *Slot Booking*: defines the tasks for M- and G- clients to arrange the remote operation slots through the N-Server.
  - *View Slots*: operations for a given client to view the current state of the operation slots in which it is involved.

2.4.1 Protocol Flowcharts

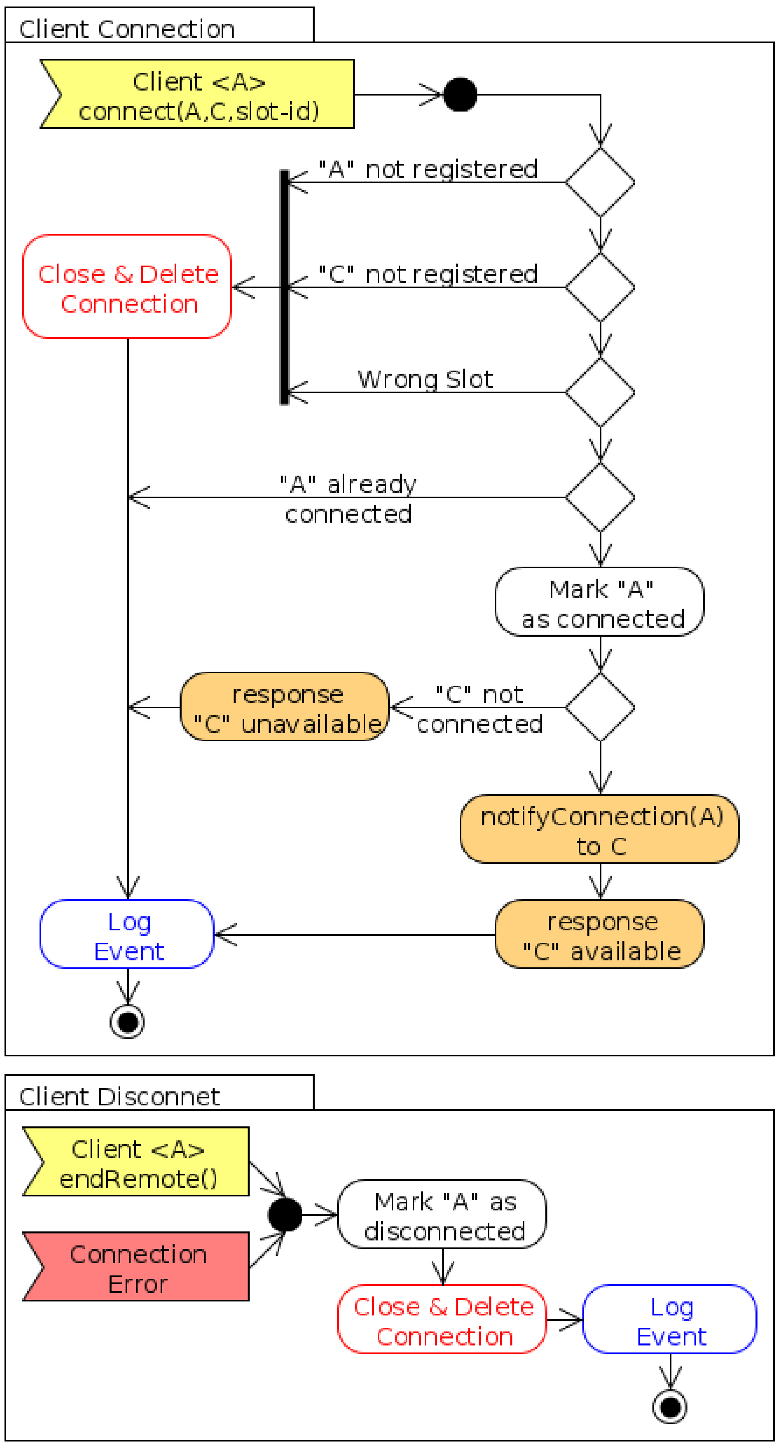

Figure 4: Software Clients Connection

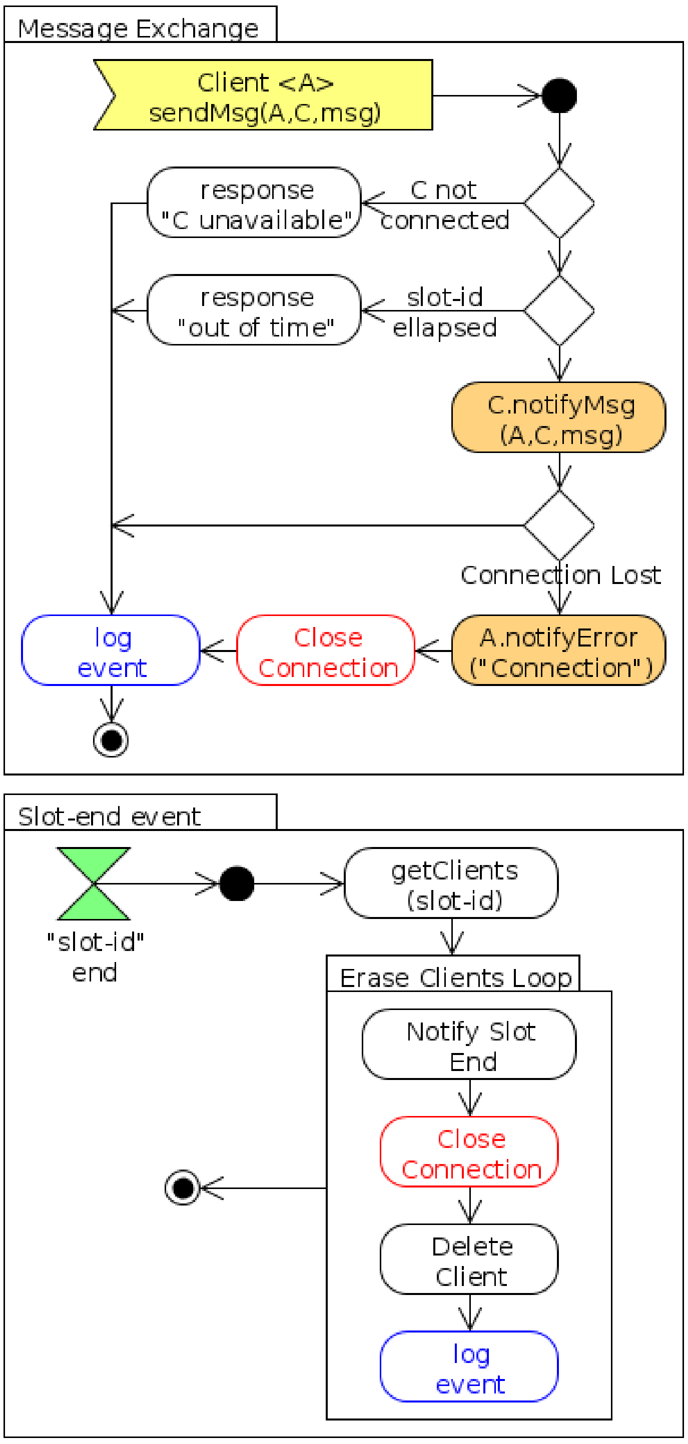

Figure 5: Data Messages Exchange

2.4.2 WebServices Flowcharts

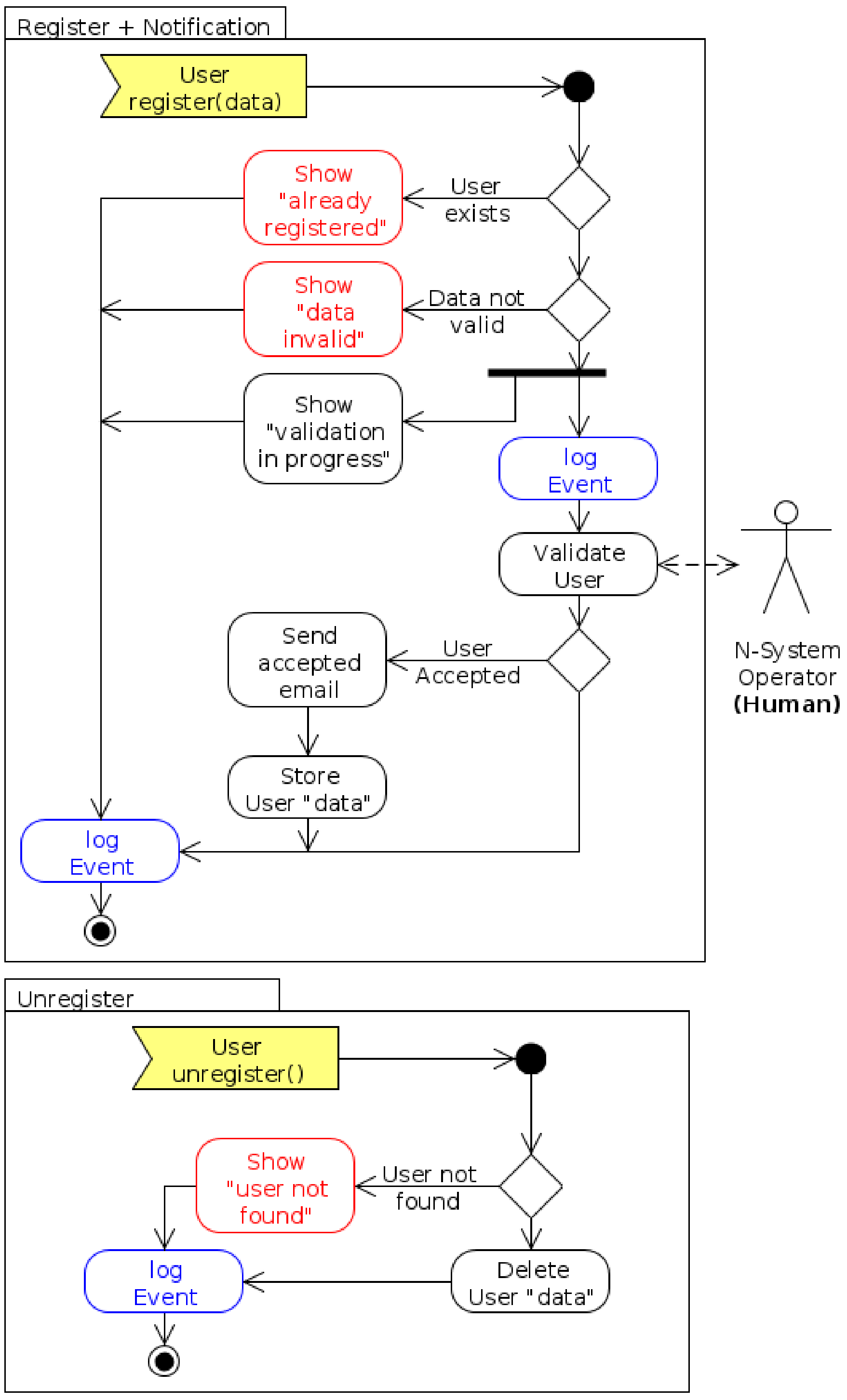

Figure 6: Users Registration

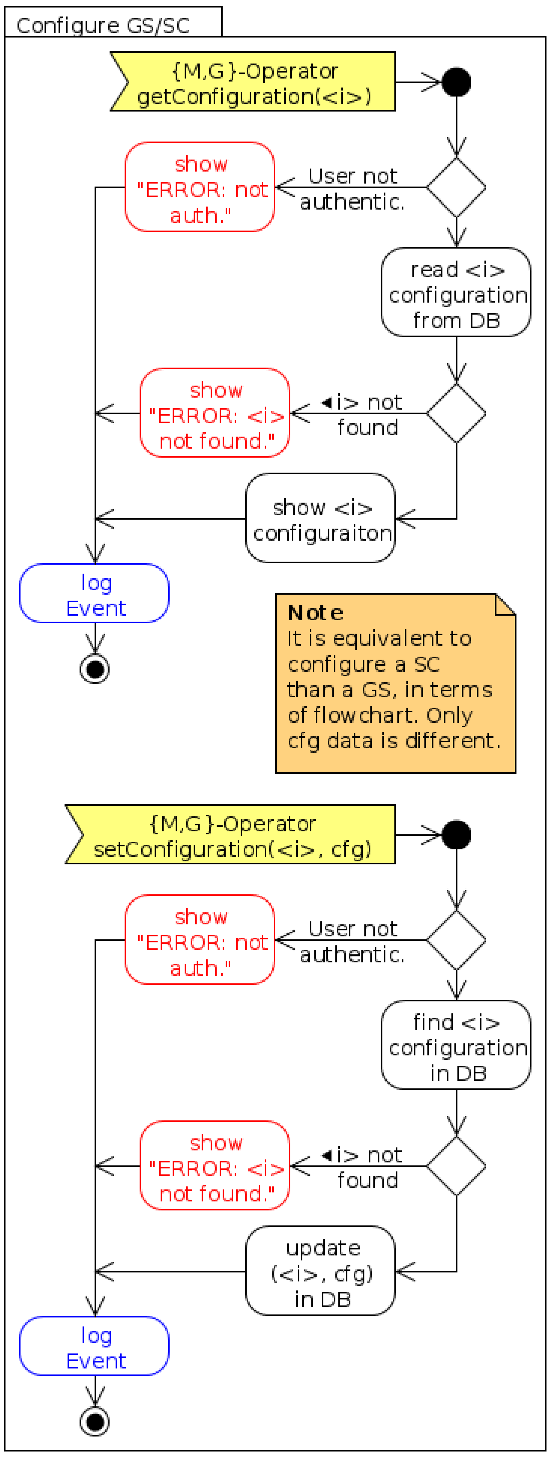
Configure GS/SC
{M,G}-Operator getConfiguration(<i>)
show "ERROR: not auth."
User not authentic.
read <i> configuration from DB
show "ERROR: <i> not found."
<i> not found
show <i> configuraiton
log Event
Note
It is equivalent to configure a SC than a GS, in terms of flowchart. Only cfg data is different.
{M,G}-Operator setConfiguration(<i>, cfg)
show "ERROR: not auth."
User not authentic.
find <i> configuration in DB
show "ERROR: <i> not found."
<i> not found
update (<i>, cfg) in DB
log Event

Figure 7: Ground Stations and Spacecraft Configuration

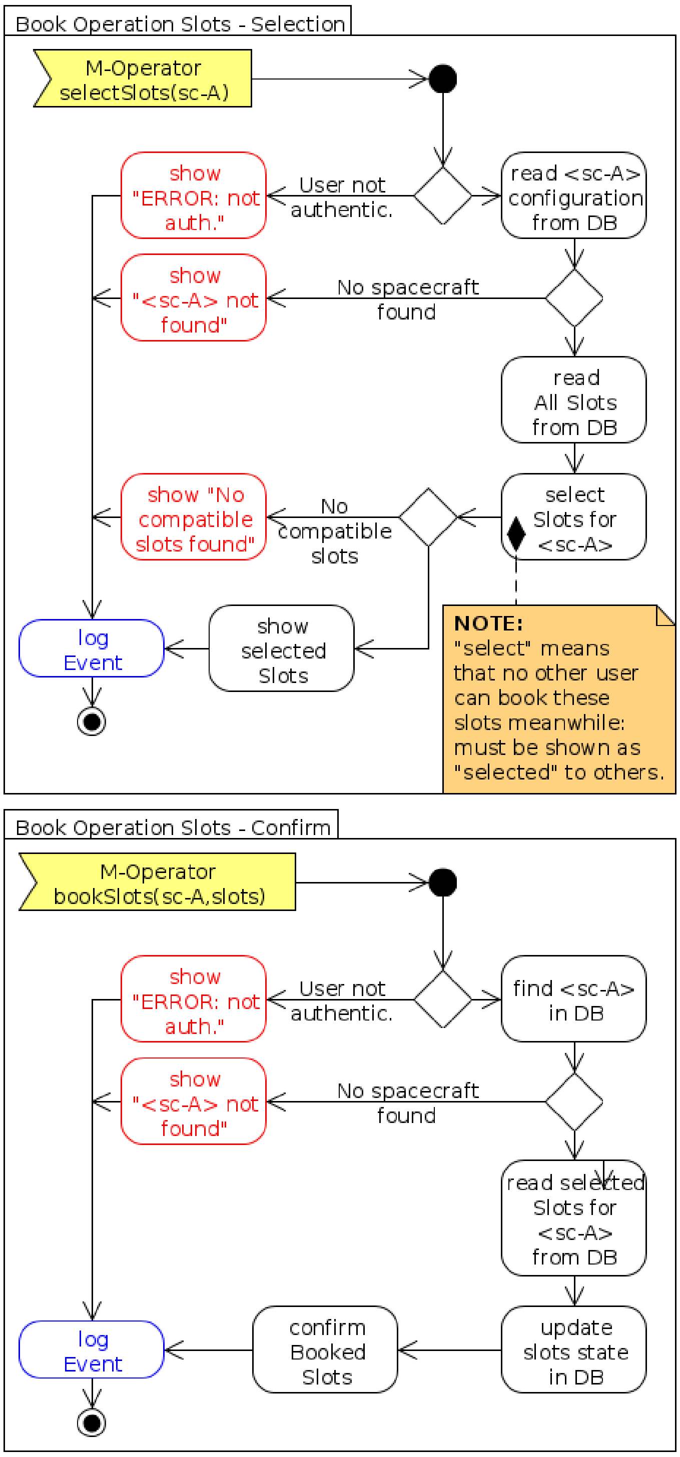
Book Operation Slots - Selection
M-Operator selectSlots(sc-A)
show "ERROR: not auth."
User not authentic.
read <sc-A> configuration from DB
show "<sc-A> not found"
No spacecraft found
read All Slots from DB
show "No compatible slots found"
No compatible slots
select Slots for <sc-A>
log Event
show selected Slots
NOTE: "select" means that no other user can book these slots meanwhile: must be shown as "selected" to others.
Book Operation Slots - Confirm
M-Operator bookSlots(sc-A,slots)
show "ERROR: not auth."
User not authentic.
find <sc-A> in DB
show "<sc-A> not found"
No spacecraft found
read selected Slots for <sc-A> from DB
log Event
confirm Booked Slots
update slots state in DB

Figure 8: Slot Booking

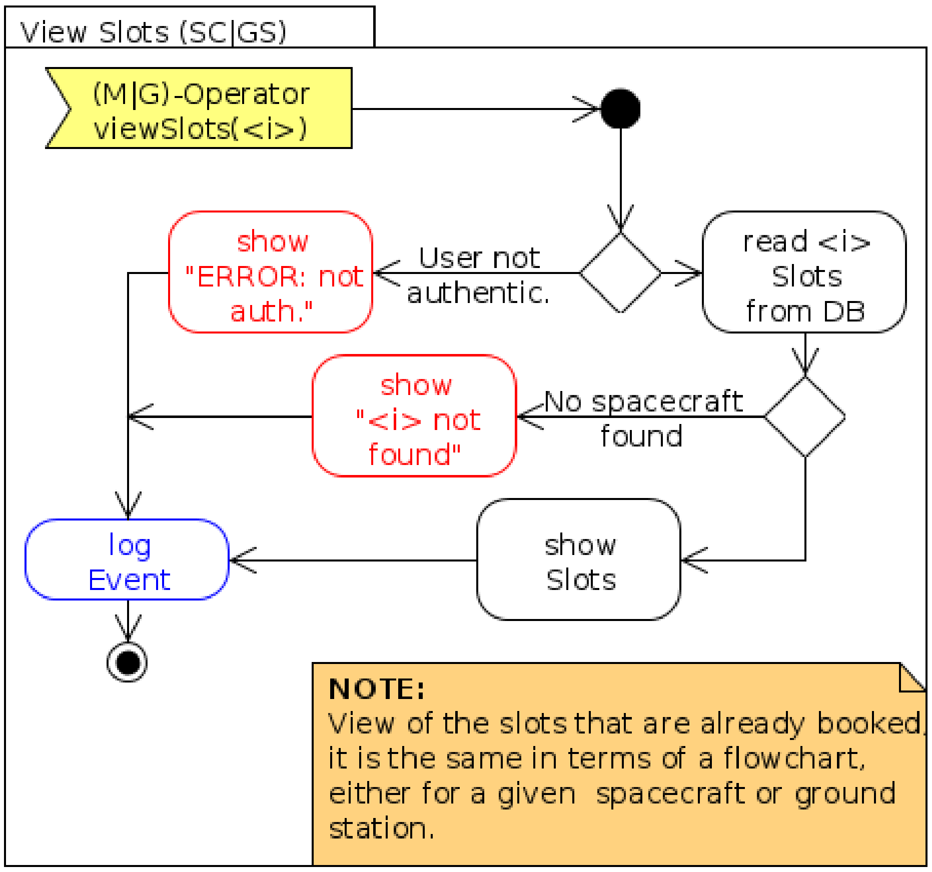
View Slots (SC|GS)
(M|G)-Operator viewSlots(<i>)
show "ERROR: not auth."
User not authentic.
read <i> Slots from DB
show "<i> not found"
No spacecraft found
log Event
show Slots
NOTE:
View of the slots that are already booked, it is the same in terms of a flowchart, either for a given spacecraft or ground station.

Figure 9: View Slots

3 Design Requirements

3.1 Target Platform

GEN-OST-010	**Target Platform (1)** The protocol shall be executed on a Debian 7.2 “Wheezy” stable operating system.
GEN-OST-020	**Target Platform (2)** The Python environment utilized shall be the one distributed with Debian. *The version of the Python environment included in that distribution is 2.7.3.*
GEN-OST-030	**Target Platform (3)** The Django framework utilized shall be the one distributed with Debian. *The version of the Django package included in that distribution is 1.4.5.*
GEN-OST-040	**Target Platform (4)** The Twisted framework utilized shall be the one distributed with Debian. *The version of the Python package included in that distribution is 12.0.0.*
GEN-OST-050	**Target Platform (5)** The MySQL server utilized shall be the one distributed with Debian. *The version of the MySQL server included in that distribution is 5.5.31.*
GEN-OST-060	**Target Platform (6)** The Apache server utilized shall provide an Apache web server.

The version of the Apache server included in that distribution is 2.2.22.

3.2 Development Requirements

GEN-DEV-010	**Development Environment**
	All configuration files specific of any development environment shall not be tracked by GIT.
	Therefore, developers are free to use any IDE they want as long as they maintain the configuration files for that IDE untracked by GIT. For this, they must include those files/directories within the required .gitignore file in the repository.
GEN-DEV-020	**Testing (1)**
	All developed Python modules shall be fully tested individually first.
GEN-DEV-030	**Testing (2)**
	Tests shall be provided in a directory named “test” within the main directory for a given Python package.
	Test directories for the main packages have already been created. However, as long as new packages are added to this initial repository structure, its associated test directories must also be added.
GEN-DEV-040	**Testing (3)**
	All unit tests shall be performed through the “unittest” library of the selected Python environment.
GEN-DEV-050	**Repository Access (1)**
	The repository will be hosted at GitHub and shall be accessed by using the GIT tools through HTTPS/SSH.
GEN-DEV-060	**Repository Access (2)**
	Developers must register at GitHub and access to the repositories must be granted.
GEN-DEV-070	**Libraries**
	Unless otherwise so stated, no libraries or binaries must be uploaded to the repository.

	This will permit avoiding an excessive amount of files in the repository.
GEN-DEV-080	**Documentation**
	Unless otherwise so stated, no documentation that can be extracted from the source code must be uploaded to the repository.
	This documentation will be generated outside the repository and uploaded to the website.
GEN-DEV-090	**Structure**
	The main structure for the repository shall use the directories and file hierarchy defined in “Annex B: Repository Organization”.
GEN-DEV-100	**Coding Style**
	Python code shall follow the [RD-5] official Python coding style guide.
GEN-DEV-110	**Code License**
	All developed source files shall include the required headers for the Apache 2 software license.

3.3 Protocol Component Requirements

3.3.1 Functional Requirements

PRT-FUN-010	**Flowcharts** The Protocol component shall perform the tasks defined in flowcharts from sections *Protocol Messages Sequence* and *Protocol Flowcharts*.
PRT-FUN-020	**Connection (1)** The protocol shall permit software clients to start a remote connection through the remote procedure call to "connect()".
PRT-FUN-030	**Connection (2)** The protocol shall permit the connection to the system only to software entities from registered users.
PRT-FUN-040	**Connection (3)** The protocol shall permit the connection to the system only to software entities that require a remote operation with a registered user.
PRT-FUN-050	**Connection (4)** The protocol shall permit the connection to the system only to software entities that request an operation slot that is about to start within the following 5 minutes **(TBD)**.
PRT-FUN-060	**Connection (3)** The protocol shall notify the availability of the remote entity requested by a given client, as a response to the "connect()" remote procedure call.
PRT-FUN-070	**Connection (5)** The protocol shall permit software clients to end their remote connection through the remote procedure call of method "endRemote()".
PRT-FUN-080	**Connection (4)**

	The protocol shall notify the availability of a remote entity to a given client, whenever the former was not present when the client requested a connection.
PRT-FUN-090	**Message Exchange (1)**
	The protocol shall receive messages from a remote client that invokes the “sendMsg()” remote procedure call.
PRT-FUN-100	**Message Exchange (2)**
	The protocol shall send the messages received from a given client to the entity that the client requested for remote operations, through the “notifyMsg()” remote procedure call.
PRT-FUN-110	**Error Notification (1)**
	The protocol shall notify the loss of a established connection to the other remote client, through the “notifyError()” remote procedure call.
PRT-FUN-120	**Error Notification (4)**
	The protocol shall notify that the additional remote client required for the communication is not available yet, through the “notifyError()” remote procedure call.
PRT-FUN-130	**Operation Slot End (1)**
	The protocol shall notify the end of an ongoing remote operation through the “notifySlotEnd()” remote procedure call, when the operation slot end is reached.
PRT-FUN-140	**Operation Slot End (2)**
	The protocol shall close all connections to the entities involved in a remote operation when the booked slot for that operation takes to an end.

3.3.2 Network Interface Requirements

PRT-NIF-010	**Protocol Remote Procedure Calls, N-Server Side** The protocol shall implement the following methods of the Client-IF interface defined as per *Annex E*: “startRemote()”, “endRemote()”, “sendMsg()”.
PRT-NIF-020	**Protocol Remote Procedure Calls, Client Side** Protocol clients shall implement the methods of the Client-IF interface defined as per *Annex E*: “notifyError()”, “notifyConnection()”, “notifyMsg()”, “notifySlotEnd()”.

3.3.3 OS Integration Requirements

PRT-OSI-050	**Daemonization** The protocol shall be executed as a daemon within the System-V booting system.
PRT-OSI-060	**Logging System** The protocol shall output its log to the “syslog” system.
PRT-OSI-070	**Command Line Interface (1)** The protocol shall not provide an interactive command line interface.
PRT-OSI-080	**Command Line Interface (2)** The protocol shall read the following arguments from the command line arguments: **(TBD)**
PRT-OSI-090	**Protocol Configuration File** The protocol shall read a configuration from an external JSON file.

3.4 WebServices Component Requirements

3.4.1 Functional Requirements

WEB-FUN-010	**Flowcharts**	The WebServices component shall perform the tasks defined in flowcharts from section *WebServices Flowcharts*.
WEB-FUN-020	**Authentication**	The WebServices component shall only permit the usage of the web services to authenticated users, with the single exception of the registration service.
WEB-FUN-030	**Logging**	The WebServices component shall log all events in the logging system, as required per the flowcharts from section *WebServices Flowcharts*.
WEB-FUN-040	**Registration (1)**	The WebServices component shall permit the registration of new users in the system.
WEB-FUN-050	**Registration (2)**	The WebServices component shall permit the registration only of those users who provide valid data for the process.
WEB-FUN-060	**Registration (3)**	The WebServices component shall permit the registration only of those users who have not been registered yet.
WEB-FUN-070	**Registration (4)**	The N-System operator shall approve or deny the registration of new users before their registry in the system is effective.
WEB-FUN-080	**Registration (5)**	The WebServices component shall send an email to the users whose registration in the system had been approved.
WEB-FUN-090	**Registration (6)**	

The WebServices component shall store the data of new users in the SATNet DB database.

WEB-FUN-100	**Registration (7)**
	The WebServices component shall log this event in the logging system.
WEB-FUN-110	**Authentication**
	The WebServices component shall permit the access to the system only to registered users that have successfully undergo a per-session authentication process.
WEB-FUN-120	**Unregister (1)**
	The WebServices component shall permit the unregister of already registered users.
WEB-FUN-130	**Unregister (2)**
	The WebServices component shall delete all user information from the SATNet DB database after undergoing the registration process.
WEB-FUN-140	**Configuration (1)**
	Authenticated G-Operators shall provide a *Ground Station Configuration Data Type* parameter as defined in *Annex C* for configuring the usage of their facilities.
WEB-FUN-150	**Configuration (2)**
	Each G-Operator shall have a single ground station configured in the system.
WEB-FUN-160	**Configuration (3)**
	Authenticated M-Operators shall provide a *Spacecraft Configuration Data Type* parameter as defined in *Annex C* for configuring their spacecraft.
WEB-FUN-170	**Configuration (4)**
	Each M-Operator shall have a single ground station configured in the system.
WEB-FUN-180	**Configuration (5)**

	M-Operators shall retrieve the configuration for their spacecraft.
WEB-FUN-190	**Configuration (6)**
	The WebServices component shall permit only the usage of this service to authenticated M-Operators and G-Operators and only in their ground stations or spacecraft.
WEB-FUN-200	**Slot Booking (1)**
	The WebServices component shall show to the M-Operators all the available and compatible operation slots over any registered ground stations that will permit the remote operation of their spacecraft (flowchart *Book Operation Slots* - *Selection*).
WEB-FUN-210	**Slot Booking (3)**
	M-Operators shall book the required operation slots through the WebServices component.
WEB-FUN-220	**Slot Booking (4)**
	Once available and compatible operation slots are shown to M-Operators, they will remain marked as “selected” in the database of the system and, thus, will not be shown for selection to other users.
WEB-FUN-230	**Slot Booking (5)**
	The maximum time that an operation slot can remain in “selected” mode is 5 minutes **(TBD)**.
WEB-FUN-240	**Slot Booking (6)**
	The operation slots booked by a given M-Operator shall be stored like that in the SATNet DB database.
WEB-FUN-250	**Slot Booking (7)**
	The operation slots booked by a given M-Operator shall not be shown to other spacecraft oeprators.
WEB-FUN-260	**View Booked Slot (1)**
	G-Operators shall view the slots booked by a remote M-Operator.
WEB-FUN-270	**View Booked Slot (1)**

Each operation slot shown to a G-Operator over the related ground station shall show both the identifier of the remote M-Operator and the channel requested.

3.4.2 Interface Requirements

WEB-API-010 **WebServices API**

The controllers that process the requests made through the WebServices component pages, shall utilize the methods of the API, as per definition in *Annex D*.

NOTE 1: Even though for release 1 this API is not going to be accessible through the Internet, defining it in this way will permit expose it to the public through JSON-RPC over HTTPS in future software releases.

NOTE 2: _SpacecraftConfiguration, *GroundStationCofiguration* and *OperationSlot* data types are defined in Annex C._

3.4.3 OS Integration Requirements

PRT-OSI-050 **Apache Server and Django Framework**

The WebServices component shall be executed within the Django framework running on an Apache 2 server.

PRT-OSI-060 **Logging System**

The protocol shall output its log to the logging system of the Django framework.

4 Annex A: RPC technologies comparison

The object of this annex is to provide a comparison of the main RPC technologies available nowadays that meet the following requirements:

1. Support for C and Python programming languages is mandatory.
2. Support authentication and ciphering suites for establishing a secure communications channel.
3. Provide a bidirectional communications channel for RPC commands to be invoked, through the same TCP connection.
   - Server is not required to establish a new connection to the client for invoking a remote procedure that is hosted at client's machine.
4. The communications protocol is based on a simple TCP transport layer.
   - This ensures a small overhead that usually occurs when utilizing much more complex communication stacks like HTTP/TCP or similar.
5. Serialization must be fast and efficient.

As per the requirements of the architecture proposed for this first software release, the RPC framework required must permit the Protocol component to directly invoke the methods of *SATNet DB Models Library*.

4.1 Comparison Table

In the following table, a comparison of some of the RPC/messaging systems available is shown. The following is a list with the definition of some terms that require of a previous definition:

- Latest: the date for the latest *stable* available release.
  - (dev) means under development.
  - Py3x means that is part of the library distributed together with Python 3.x.
  - Django means that is part of the Django Web Framework.
  - Xy means that the latest available release was made more than X years ago.
- Support: this term refers to the support in terms of documentation and other resources available for carrying out a development with this technology.
- Complexity: estimation of how complex would it be to develop the required system with this technology.
- Symmetry: evaluates whether the initial connection made by a client can be utilized for exchanging messages in both directions without the need of creating a new connection.

(*1) TLS support for RPyC is based in a no-longer supported Python library.

4.2 Result: Twisted AMP

In accordance with the table above, the only protocol stack that meets the requirements imposed is **Twisted AMP**.

The optimal solution would be a RPC protocol that is transport agnostic (or, at least, that it could use TLS/SSL TCP sockets), serialization agnostic (or, at least, that it supported the usage of light and fast serialization libraries like MessagePack) and capable of sending requests to/from clients through a single connection established by the remote client (NAT/firewall issues).

ZeroMQ is a very fast protocol for data messaging in between nodes of a distributed system. It supports sending messages through a socket-like programming interface and is capable of exchanging this information across multiple different topologies. However, it lacks of a method for Remote Procedure Calls and, therefore, it would involve the implementation of a custom messaging system with any of the available serialization libraries. Besides, it does not support encryption by October 2013.

gevents is neither a RPC library, nor a message exchanging system: is a very fast and light libevent adaptation to Python that permits the execution of concurrent Greenlets, as a response to the events detected on an underneath socket connection. This way, the usage of such a technology would involve either to impose the usage of Python as network clients (for re-using the same gevents based library both on clients and servers), or to carry out an independent development on the client-side.

Twisted with AMP permits a fast events based development and has support for multiple languages. It has a good performance (see link Twisted benchmarks), good support and documentation. It only lacks of some flexibility when it comes to select the serialization and RPC protocol, and it might be hard to use in embedded systems with few hardware resources.

Protocol	Latest	Support	Transport	Marshalling	Security	Complexity	Symmetry	Languages
xmlrpclib	Py3x	Py3x	HTTP	XML	TLS	Low	No	?
json-symm	>2y	Poor	TCP	JSON-RPC	TLS	Low	Yes	?
json-rpclib	(dev)	Poor	HTTP	JSON-RPC	TLS	Low	No	?
django-jrpc	Django	Django	HTTP	JSON-RPC	TLS	Low	No	?
gevents	2012/09	Fair	TCP	Agnostic	TLS	High	Yes	Python
RPyC	2012/12	Fair	TCP	Own	TLS(*1)	Med.	Yes	Python
Pythomnic3k	2012/08	Fair	TCP	Many	TLS	Med.	Yes	Python
Twisted AMP	2013/06	Good	TCP	AMP	TLS	Low	Yes	Many
txjsonrpc	>1y	Poor	Twisted	JSON-RPC	TLS	Low	Yes	Many
ZeroMQ	2013/10	Good	TCP	Agnostic	No	High	Yes	Many
ZeroRPC	(dev)	Fair	ZeroMQ	Message-Pack	No	Low	No	Python
Spyne	2013/07	Fair	ZeroMQ	Many	No	Low	No	Python
Thrift	2013/08	Good	TPC	Many	TLS	Low	No	Many

5 Annex B: Repository Organization

This annex contains the definition of the files and directories structure for the GIT repository of the release 1. This definition contains the main files and directories; however, additional files are permitted to be included whenever they are added to the directories of the packages that they belong.

In order to define additional packages and modules, Python names convention as per [RD-5] apply. However, new directories and files must only be placed under the current directory hierarchy.

Figures below show the current definition for the repository.

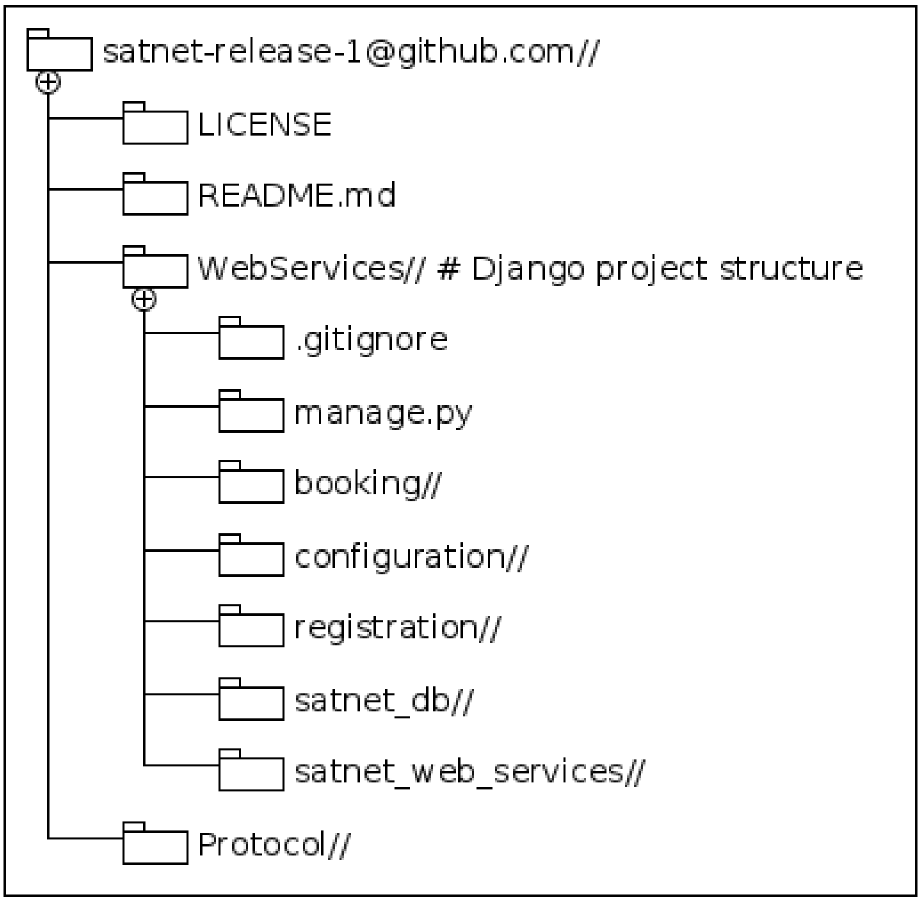

Figure 10: WebServices repository

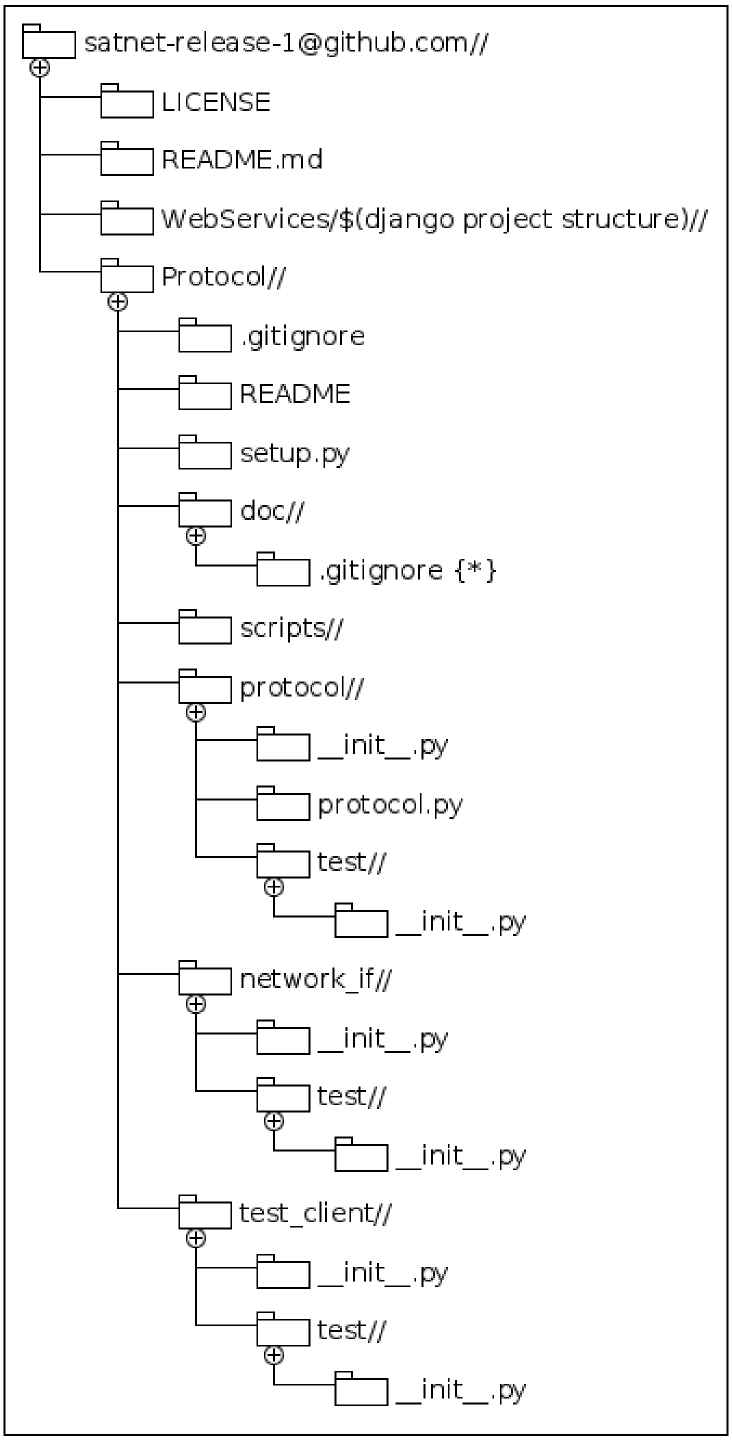

Figure 11: Protocol repository

6 Annex C: Data type definitions

The data type definitions included in this section utilize the JSON notation for defining those types in an abstract way.

- Along the following data type definitions, all identifiers are supposed to be **unique**.

6.1 Ground Station Configuration Data Type

- Ground Station concept: facility located at a given place, that has a set of communications channels to be exploited during the time slots on which operators are available.

```
{
    "identifier": "My Ground Station",
    "contactElevation": 10.4,
    "location" :
    {
        "latitude": 25.5000,
        "longitude": 0.0000,
    },
    "channels":
    [
        {
            "channelId": "channel-01",
            "modulation": "FM",
            "frequencyRange":
            {
                "minimum": 450.54,
                "maximum": 800.23,
            },
            "bandwidth": [12.5, 25, 50],
            "bitrate": [600, 1200, 2400],
            "polarization": ["LHCP", "RHCP", "LINEAR"],
        }
    ],
    "availability":
    [
        {
            "date": "07/17/2014",
            "slotStart": "10:00",
            "slotEnd": "11:00",
            "channelId": "channel-01",
        }
    ]
}
```

6.2 Spacecraft Configuration Data Type

- Spacecraft concept: mobile device that has a set of communications channels to permit its remote operation through a ground station.

```
{
    "identifier": "My Spacecraft",
    "callsign": "KXSC02",
    "tracking":
    {
        "url": "www.celestrak.com/NORAD/elements/f.txt",
        "spacecraftId": "XATCOBEO",
    },
    "channels":
    [
        {
            "channelId": "slowChannel",
            "modulation": "FM",
            "frequency": "457.356",
            "bandwidth": "25",
            "bitrate": "600",
            "polarization": "LHCP",
        },
        {
            "channelId": "fastChannel",
            "modulation": "FM",
            "frequency": "457.356",
            "bandwidth": "50",
            "bitrate": "1200",
            "polarization": "RHCP",
        }
    ]
}
```

6.3 Operation Slot Data Type

- Operation Slot concept: time slot during which a given ground station can operate a given spacecraft through one its channels. This channel matches the communications requirements of, at least, one of the communication channels of the spacecraft.

```
{
    "identifier": "slot-20140717100011 00-MyGroundStation",
    "slotDate": "07/17/2014",
    "slotStart": "10:00",
    "slotEnd": "11:00",
    "gsIdentifier": "My Ground Station",
    "scIdentifier": "My Spacecraft",
    "scChannel": "fastChannel",
}
```

7 Annex D: WebServices API

- The following API is defined utilizing the conventions for the JSON-RPC procedures.
- For not defining twice in the same document the same data structure, *SpacecraftConfiguration*, *GroundStationConfiguration* and *OperationSlot* data types have not been re-defined, as a proper usage of the JSON-RPC normative would required. Instead, the value for those variables has been left filled with $TYPE$ for indicating that the fields to be included in that variable are defined in *Annex C*.

```
>>>
{
    "jsonrpc": "2.0",
    "method": "getSCConfiguration",
    "params":{"identifier": "XATCOBEO"},
    "id": 1
}
<<<
{
    "jsonrpc": "2.0",
    "result": {"SpacecraftConfiguration": \
                "$SCConfiguration$"},
    "id":1
}
```

```
>>>
{
    "jsonrpc": "2.0",
    "method": "getGSConfiguration",
    "params": {"identifier": "VigoGS"},
    "id": 2
}
<<<
{
    "jsonrpc": "2.0",
    "result": {"GroundStationConfiguration": \
                "$GSConfiguration$"},
    "id":2
}
```

```
>>>
{
    "jsonrpc": "2.0",
    "method": "selectSlots",
    "params": {"identifier": "XATCOBEO"},
    "id": 3
}
<<<
{
    "jsonrpc": "2.0",
    "result": {["OperationSlot": "$OPSlot$"]},
    "id":3
}
```

```
>>>
{
    "jsonrpc": "2.0",
    "method": "bookSlots",
    "params": {"identifier": "XATCOBEO", \
                ["OperationSlot": "$OPSlot$"]},
    "id": 4
}
<<<
{
    "jsonrpc": "2.0",
    "result": "OK",
    "id":4
}
```

```
>>>
{
    "jsonrpc": "2.0",
    "method": "viewSlots",
    "params": {"identifier": "VigoGS"},
    "id": 5
}
<<<
{
    "jsonrpc": "2.0",
    "result": {["OperationSlot": "$OPSlot$"]},
    "id":5
}
```

8 Annex E: Network Interface

8.1 Network-IF, N-Server Remote Calls

- In the following interface definition, the constants represent the possible return values for the remote method *startRemote()*.

```
#define CODE_OK 0
#define CODE_CLIENT_OK CODE_OK
#define CODE_NO_CLIENT -1
#define CODE_NO_SLOT -2
```

```
int startRemote(int clientId, int slotId);
void endRemote();
void sendMsg(byte[] data, int len);
```

8.2 Network-IF, Client Remote Calls

```
void notifyError(String description);
void notifyConnection(int clientId);
void notifyMsg(byte[] data, int len);
void notifySlotEnd(int slotId);
```