

Mars Mission Computing Environment

System Design

Version: 0.2

Author: Mark Smith

Document History

<i>Version</i>	<i>Date</i>	<i>Author</i>	<i>Changes</i>
0.1	27/06/03	Mark Smith	Initial Draft
0.2	10/02/04	Mark Smith	After coming back to this, I decided to document the design independantly of the time log.

Table of Contents

Overview.....	3
JMF Overview.....	7
Audio Processing.....	8
Discovery and Networking Solution.....	8
JXTA.....	9
JINI/JavaSpaces.....	10
Mission Planning and Monitoring.....	10
Mission Discovery.....	10
Analysis Model.....	11
Suit System.....	13
Rover System:	14
Power Up Use Case.....	17
Suit User Interface.....	18
General Windows.....	18
Main Screen.....	18
Communications Window.....	18
Configuration Window.....	19
Rover User Interface	19
Login Screen.....	19
Main Screen.....	20
Communications Window.....	20
Mission Planning Interface.....	21
Linkage of UI to Main Systems.....	21
Voice Communications Networks.....	23
Server Construction and startup.....	24

Overview

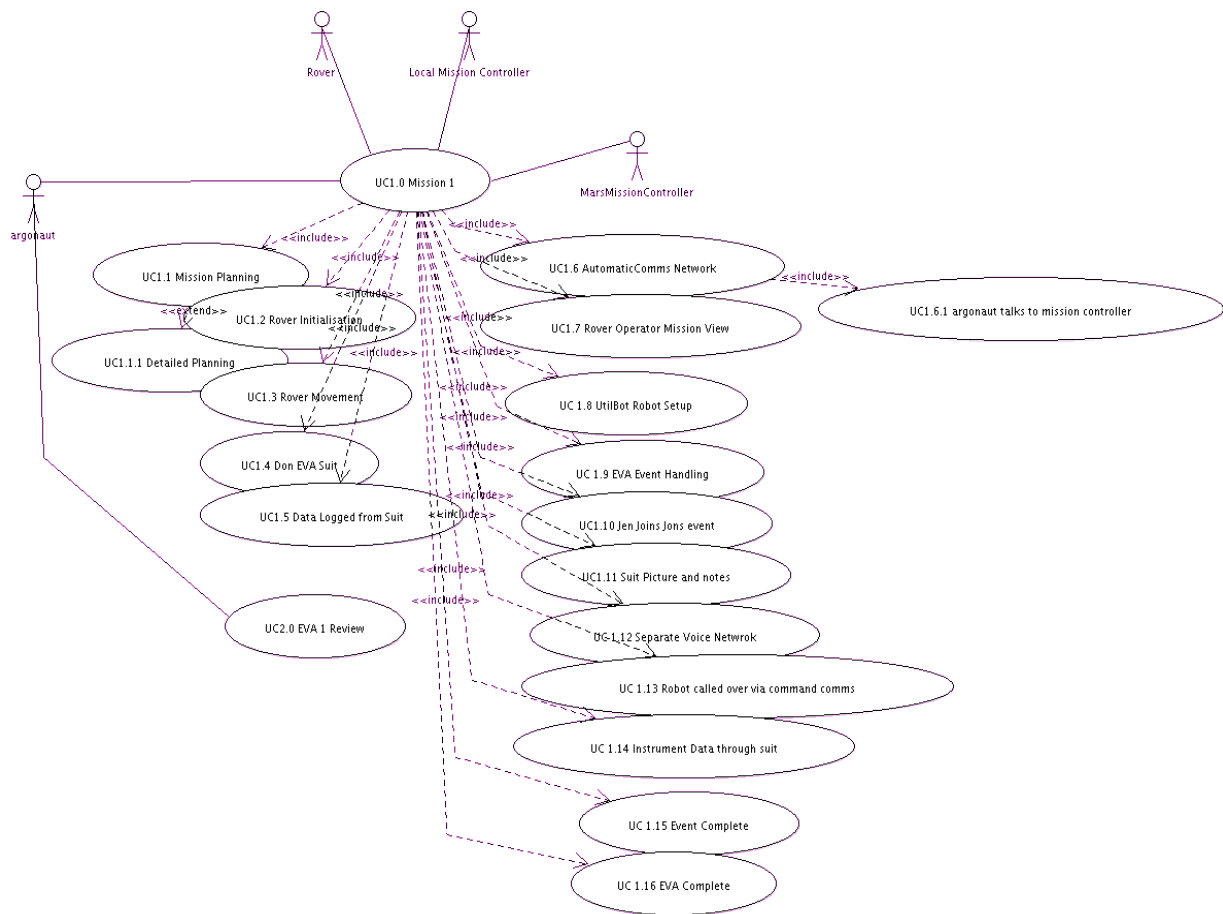
This document describes the design of the mars mission computing environment.

What problem are we trying to solve here? The basic case is – suit powers up & discovers what mission its on. It then needs to set up a voice communications network with it's peers. The voice network is a set of data streams sent over an IP network (voice over IP).

The suit powers up, discovers it's part of EVA-2 by talking to a local authority called MissionController. It asks mission controller – I am suit 62 what mission/eva am I on. This responds with the correct values. The suit searches for a JXTA group named EVA-2. If none is available, it starts the group, and gives an RTP Media Resource Locator for it's audio. As each peer is added to the group, hopefully each group member gets an announcement, and can then pick up the audio stream.

The audio and data streaming is solved using JMF. The networking and discovery is solved using JINI. The main problem domain – missions, plans, eva's etc are the business objects for this project. That is, they are the payload – the data we are really interested in seeing. As a person is using the eva environment they have a number of roles: eva participant, rover operator, or remote mission monitor. They connect using the technology described here to the actual rover and astronaut through the network and receive realtime updates of the situation.

The complete use case diagram is shown below:



The use cases are:

<i>Use Case Number</i>	<i>Use Case Name</i>	<i>Description</i>
UC 1.0	Mission 1	<p>Jon - Geologist Steve - Psychologist Robert - Physicist Jen - Geologist</p> <p>Steve remains inside the rover, while the rest of the group go outside.</p> <p>They walk for a while, talking on the voice comm.. The mission is to explore the valley they are in. So they split up. Jon discovers an interesting feature.</p>
UC1.1	Mission Planning	The team plan a mission to go from the hab out to Nili Patera in Syrtis Major. They call the mission MIS-1.

<i>Use Case Number</i>	<i>Use Case Name</i>	<i>Description</i>
UC1.1.1	Detailed Planning	Jon wants to go out to ili Patera in Syrtis Major. He brings up the locations database, and selects the appropriate maps. The destination is known in GPS co-ordinates. The distance is then calculated to the destination and any geographical input helps the program estimate fuel/resources required for the trip. Jon adds other personnel (Jen, Steve & Robert) to the roster for the mission. He brings up the mission timeline, and, with the program estimating travel time, identifies which times/days he will be where in the mission and plans out approximate EVA's. When the mission is saved, and approved, it is lodged with the mission register as an active mission, scheduled to go.
UC1.2	Rover Initialisation	After planning (UC1.1) the team transfers to the rover via airlocks and start up. They tell the rover they are MIS-1. This transfers all the planning materials.
UC1.3	Rover Movement	The team then drive to their destination. The rover guides their travel using the planned route.
UC1.4	Don EVA Suit	Jon,Rob, Jen don suits and power up. The suit discovers the rover & determines it is part of MIS-1. The suit asks the wearer to confirm they are MIS-1. (could be a visitor from another rover) & confirm their identity.
UC1.5	Data logged from suit	From power up, all data available (vital signs, temperature, suit pressure, oxygen, video, GPS etc are captured and stored & streamed back to the rover.
UC1.6	Automatic Comms Network	Upon confirmation of MIS-1 an immediate voice communications network is set up. This allows for communications between the group of people in MIS-1. The comms network includes the rover operator steve. Jon, Jen, Robert & Steve talk to each other to check the comms network is operational

<i>Use Case Number</i>	<i>Use Case Name</i>	<i>Description</i>
UC1.7	Rover Operator Mission View	The rover operator goes to the computer console and logs in. MIS-1 standard view is shown. This is a map of the area (loaded from mission data) with each suit indicated as an icon with the wearer name on the outside. The voice network is played through speakers. UtilBot 1 also appears on the monitor with a different icon type as it detaches from outside the rover.
UC1.8	UtilBot Robot Setup	For this EVA, they have chosen to take UtilBot-1 A general purpose tool carrier. It's default behaviour is to tag along with the group. Jon & Jen put geological instruments in UtilBot 1
UC1.9	EVA Event Handling	Jon says to suit "SUIT: EVENT Geological formation 21, OK". The suit responds "New Event". This flashes on Steve's screen by blinking Jon's icon. The event name 'Geological formation 21' appears on Steve's screen. Steve clicks on the icon and a view of the event comes up in a separate window. This is video from Jon. The event is sent as notification to Mars Hab who are logging missions. Mina, an operator there sees the event and clicks on it. At present she sees only the name of the event. If she clicks, she will see what mission/EVA, and be able to see who is there etc. She can choose to get vid feed/audio feed if bandwidth allows. All data on Jon's suit is now associated with the event. The rover is recording all data against the event. Jen and Rob are also notified of Jon's event via the PDA. The notification may come over the voice network as well.
UC1.10	Jen Joins Jon's Event	Jon calls over Jen. She clicks on Jon's event in her display and her data is also associated with Jon's event. Her attendance to the event is noted at rover & hab. At rover her vid is also displayed.
UC1.11	Suit Picture and Notes	Jon looks at the format and directs suit "SUIT, PICTURE Big Rock, OK". This picture is named and noted to all parties monitoring. Jen then directs "SUIT, NOTES to PICTURE Big Rock. This rock is blah blah blah, OK".

<i>Use Case Number</i>	<i>Use Case Name</i>	<i>Description</i>
UC1.12	Separate Voice Network	Jon shows Jen, and they begin talking on voice comm. Net. Rober says "Can you go separate?" They apologise. Jen says "SUIT, EVENT COMM, OK". All personnel associated with the current event for Jen have a little voice net set up for them. The mission comms net, and eva comms net are now options in the PDA. By clicking on one they can rejoin either conversation. They can also opt to keep listening to mission comms, but only transmit to the eva comms. All comms are associated with the event and streamed/logged.
UC1.13	Robot called over via voice network	Jon calls over Util Bot 1 by saying "SUIT, COMMS WITH UTILBOT-1 CALLTYPE COMMAND". He gets a third comms session on the PDA. He then says "UTILBOT1, MOVE TO ME". The bot comes over. The bot & data is automatically added to the event and this is notified.
UC1.14	Instrument Data through suit	Jon takes out the instruments and plugs it into his suit. They take data readings and these are logged into the Event.
UC1.15	Event Complete	They have now finished the event. Jon says "SUIT, EVENT COMPLETE", and all are removed from the event and keep logging against the mission. The comms network is also closed.
UC1.16	EVA Complete	They all return to the rover. When they enter the rover, EVA-1 is completed automatically.
UC1.6.1	Argonaut talks to Mission Controller	Jon selects the comms network for EVA-1 in his display. He then talks and his voice is streamed to all other participants in the communication. They reply, and their replies are transmitted back.
UC2.0	EVA Review	Afterwards in the rover the entire data set can be reviewed and tagged iwth further results or detail. Reports may be written and this can hyperlink to events from the mission eva data.

JMF Overview

5/7/03: Have just been reading the JMF docs. The basic scenario is that you:

- Capture data (this makes a data source)

- create a processor (or player if you just want to render it)
- set the data from the capture to the processor to a data sink which will save the data.
- Set a player onto the processor so you can see it as well.

So, we can repeat this for each device and sensor we have in our system. Each will be a new capture device, and will therefore provide a data source that can be rendered. We will have to provide our own render components, and probably mux/demux as well. But the basic scenario is a good way of transmitting data around (using RTP)

So, that solves the technology choice for capture playback, and storage of data. Java Media framework gives us an extensible framework that we can adapt for any data capture device we use (ie ocean controls labjack, or their lower priced parallel port version) We can write a java based controller of these and provide this to JMF for processing. Ditto we can write a data source for the GPS units based on GPS Java. The audio & video are already done for us. Then, we just need to mux up all the data and then transport it.

Audio Processing

JMF will be used for media capture & streaming. At each station where a human is using a microphone, we need to capture data using a jmf 'capturedevice'. This will provide a data source. We will take the stream from the data source and create two processors. The first will log the data to a file. The second will produce an RTP data source. This is then transmitted to the receiver (the first receiver being the rover mission controller. Events will tell us others to include in the RTP session. (ie there is an RTP event for new participant.)

At the receiver, the incoming RTP data source is decoded and presented to a player. The player will put the data out to a speaker.

Discovery and Networking Solution

The next major tech push is for the discovery of surrounding people/equipment/devices that the suit can use. A simple solution would be just give each an IP address, configure an LDAP server with the addresses, and then have a configuration item in the LDAP that tells the suit what mission it's on etc. But, we also want to have some messaging between the various items. A management stream as opposed to the data stream provided by JMF. JXTA has peers, pipes and messages. This could help us. Jini is another candidate. How will all this work for us?

What problem are we trying to solve here? The basic case is – suit powers up & discovers what mission its on. Whichever entity first starts up then creates the 'comms network'. A comms network is simply RTP streams of audio (and later, vid) shared amongst peers. That is – the suit powers, discovers it's part of EVA-2. The discovery is done by find a JXTA peer called MissionController. It asks mission controller – I am suit 62 what mission/eva am I on. This responds with the correct values. The suit searches for a JXTA group named EVA-2. If none is available, it starts the group, and gives an RTP Media Resource Locator for it's audio. As each peer is added to the group, hopefully each group member gets an announcement, and can then pick up the audio stream.

JXTA

JXTA means 'juxtapose' or side by side. The idea is that you have:

1. Peer – a peer is an entity that implements JXTA protocols. Peers have end points to which pipes can be connected. Peers are equal members and there are various types of peer:
 1. Edge Peer- send/receive message, but not cache advertisements, nor route
 2. Full features edge peer - tx/rx msg, cache advertisements, not forward discovery.
 3. Rendezvous peer – cache advertisements, forward discovery requests to help other peers discover resources. If looking for a rendezvous, and none found, will become rendezvous
2. Peer group – organisation of peers. Secure and unsecure membership. Services defined for group. Can download implementations of services (via modules)
3. Service – peers cooperate and communicate to publish, discover, and invoke network services.
4. Pipe – peers use pipes to send messages to one another. Pipes are asynch, unidirectional. Hence tx/rx pipe set up between endpoints. Hence, point to point, or propagate (one out to multiple in).
5. Advertisements – peers, peer groups, pipes, services are represented by advertisements. These are language neutral meta data.

The security model uses certificates to identify you, so quite sophisticated.

Messages are multiple name/value pairs. Each element can be parsed out and acted upon. Hence, we can send a message down a pipe to a peer and extract the 'name' element or something.

It means that we have to implement the whole method set via messages. I don't like that. I want to use a naming service to discover an EJB and invoke methods on it. That is much easier – defined classes to talk to. Or even just RMI. If I know what the class and server are, I'll get an instance and talk to it.

I just want a straight distributed system with services available for people in the network to use. We could use the name/value pair messages to get the RMI names to use, and then connect & use an RMI class. It seems that there is a way to advertise a method, and the parameters it takes. You then have to find the method in the advertisement for the service, and do the method, passing the right parameters. It's extremely ugly just to do a method call across the network. That's what EJB, RMI, CORBA are all about.

So, the basic idea in JXTA is:

1. start up and tell jxta who you are (they have config tool for this. Presumably, we either run this, or do it ourselves programmatically.)
2. create a peer group, or try to connect to the marseva env group.
3. Locate the mission store service.
4. Connect using pipe. Comm identity and determine mission.
5. For the mission, find eva we should be a part of.
6. Find peer group for eva.
7. Connect to eva peer group.
8. Publish audio comms service. Find other instances of this service. Connect and exchange RTP MRL. Set up JMF streams, and establish audio comms.

How would it work for new audio sessions? We would create a new peer group for a smaller audio session. Then, publish audio services to each in the peer group, connect & set up.

How would we do events? We would publish an event source? Perhaps the default 'service' that all peers in an eva group must support is the eventexchange service. Any peer can raise an event, and then join it, or notate it etc. I need to model the services, so I know what they look like, and can then use them between the peers.

Similarly, there could be a peer group created for mission monitoring. The rovers would join and publish a mission summary service. The mission monitor would then connect to each summary service, and receive summary messages from that service.

JINI/JavaSpaces

JINI is a technology to allow for the lookup and provision of a services to network entities . The services are java objects as opposed to the JXTA object.

The idea of JINI is that there is a lookup service. The lookup service contains entries. Each entry is a service object. The service object comes local to the client and then executes remotely to the service provider. There is provision for an event model whereby events can be propogated in a distributed manner. Also, entries in a jini lookup are leased. This give time expiry to reduce staleness.

In this manner, our problem is solved by declaring a multicast group called suit-mission. We discover this group's lookup service, and find the mission controller controller service. See below for what happens next.

Mission Planning and Monitoring.

Mission Discovery

Discover mission controller. If no mission controller available, confirm with wearer, then become mission controller. Retry discover if directed. If mission controller and no database, create one, with default mission and default If no identity set, ask mission controller for next available suit number. Download configuration data. Ask controller which mission suit is on, and identity of person associated. Start up all data sources, and begin a) logging data, then streaming data over RTP to mission controller.

Mission controller will create new communications session for mission, then eva. The most specific comms session is the default used if no other is chosen by the user. The session of choice dictates the audio streams mixed together for presentation.

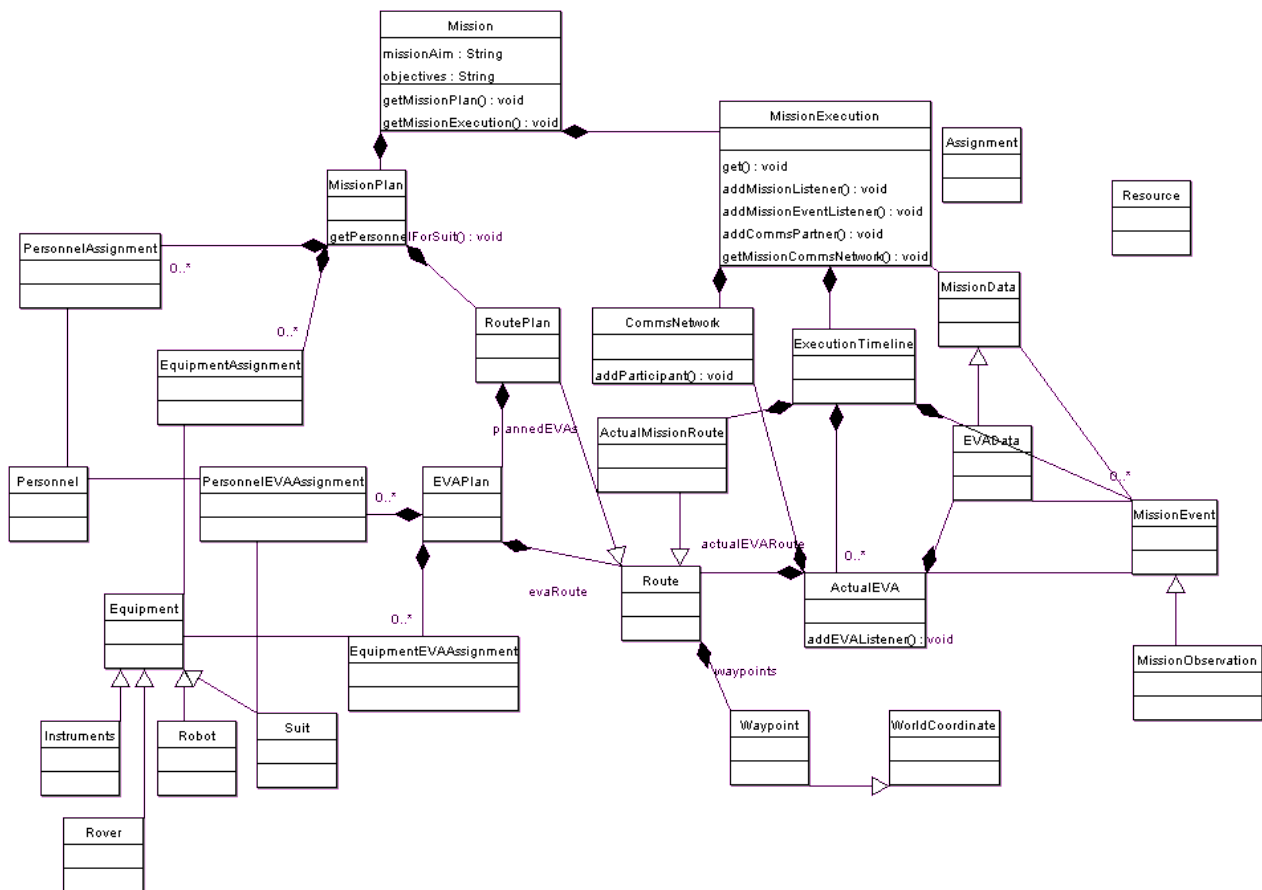
Processing events:

- new participants/eva members: get their audio stream and add

Analysis Model

16/7/03

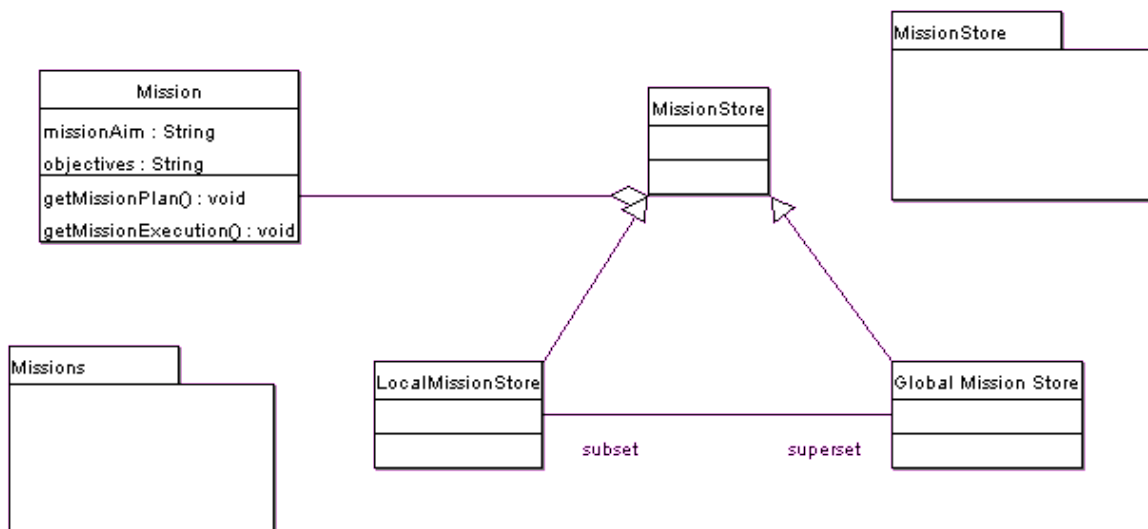
I have been working on a model of the basis for the system. I'm using ArgoUML – a free modelling tool. Unfortunately, it's a little buggy, so, it's not so easy to think in it. Still, here's the current mission decomposition:



A Mission is defined to have:

- A plan. A plan represents the booking of resources, the design of a route, and any eva's along the way. This is used to check schedules, book equipment & people.
- An execution. When the mission starts, a mission execution is created. This then tracks the mission. It's like a log, except it also represents 'now'. So, you can get the current attributes of the mission from this object.

Each plan is then a construction of planned routes, equipment and personnel. The execution entity records the eva as it occurs through data provided by the various participants.



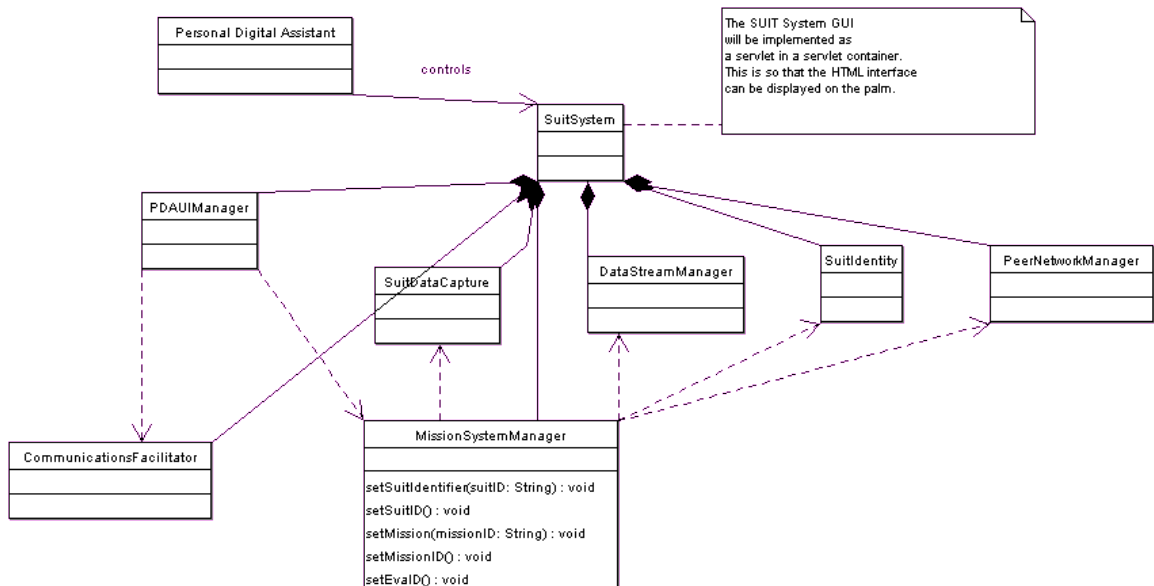
In terms of tracking, there is a mission store that holds all the current missions. There is a 'local' mission store for holding a version of the mission store that is 'nearer' in bandwidth terms, or connectivity terms than the global mission store. This is for rovers. They will carry some local version of the mission store, then resync with the global.

There are three main applications in use for the system:

- Suit System
- Rover System
- Mission Monitor System

Suit System

A conceptual model of the Suit System is shown below:

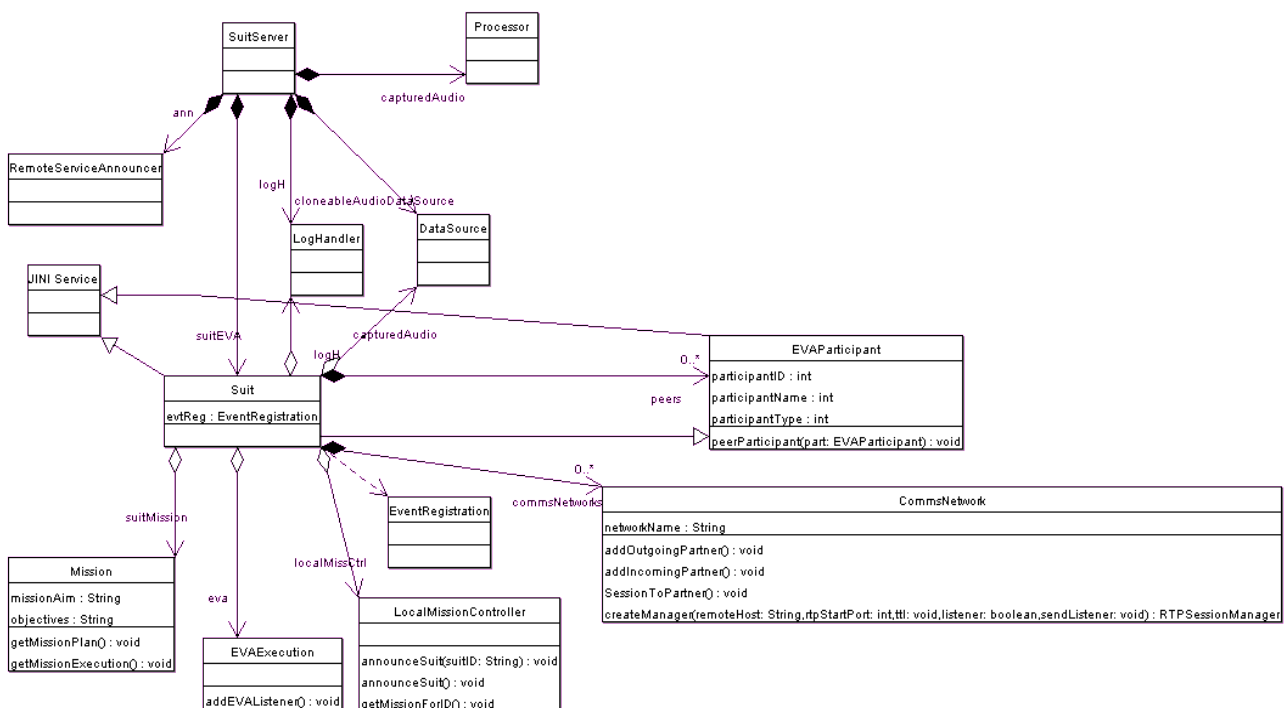


The Suit System has a PDA for it's primary interface. The PDA talks to the Suit System. The Suit System has:

- **PDA UI Manager.** This looks after the User interface for the suit (ie manages the PDA)
- **Suit Data Capture.** The SUIT Data Capture component of the suit system has the responsibility of capturing data from:
 - microphone
 - video
 - sensors
 - GPS
 - 'plugged in' portable devices
- **Data Stream Manager.** The DataStreamManager is responsible for:
 - encoding capture data and streaming to other interested parties
 - decoding incoming data and outputing to relevant local output devices.
- **Suit Identity:** Every SuitSystem has an identity. This needs to be stored and communicated when necessary
- **Peer network Manager:** The PeerNetworkManager is responsible for communicating over the network with other resources on the network
- **Mission System Manager:** The mission system Manager is responsible for:
 - determining the mission/EVAs available in the local network

- confirming with the argonaut the mission/EVA to use
- handling communications with other argonauts and utilbots
- handling events/media notation.
- **Communication Faciliator:** The communications facilitator manages communication between the various parties:
 - argonaut suits
 - rover operator
 - remote mission monitors

The implementation of the suit system is shown below:



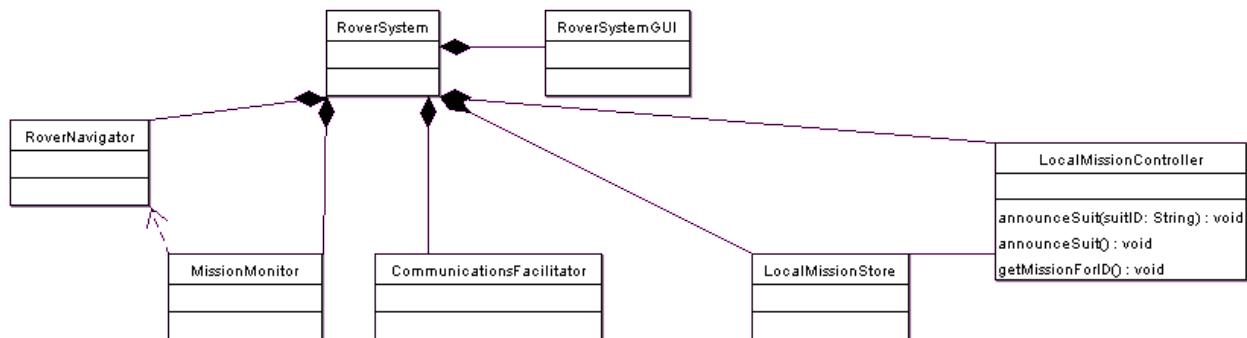
A SuitServer is the 'executable' that instantiates a suit class. The suit implements eva participant, VoiceCommsNetworkParticipant, and RemoteServiceAnnouncerListener

```
public class Suit implements
    EVAParticipant,
    RemoteServiceAnnouncerListener,
    VoiceCommsNetworkParticipant {
```

It has a mission that is retrieved from the local mission controller. The mission contains the details of the eva the suit is a part of. For each communications network the suit is a part of, there is a comms network entity.

Rover System:

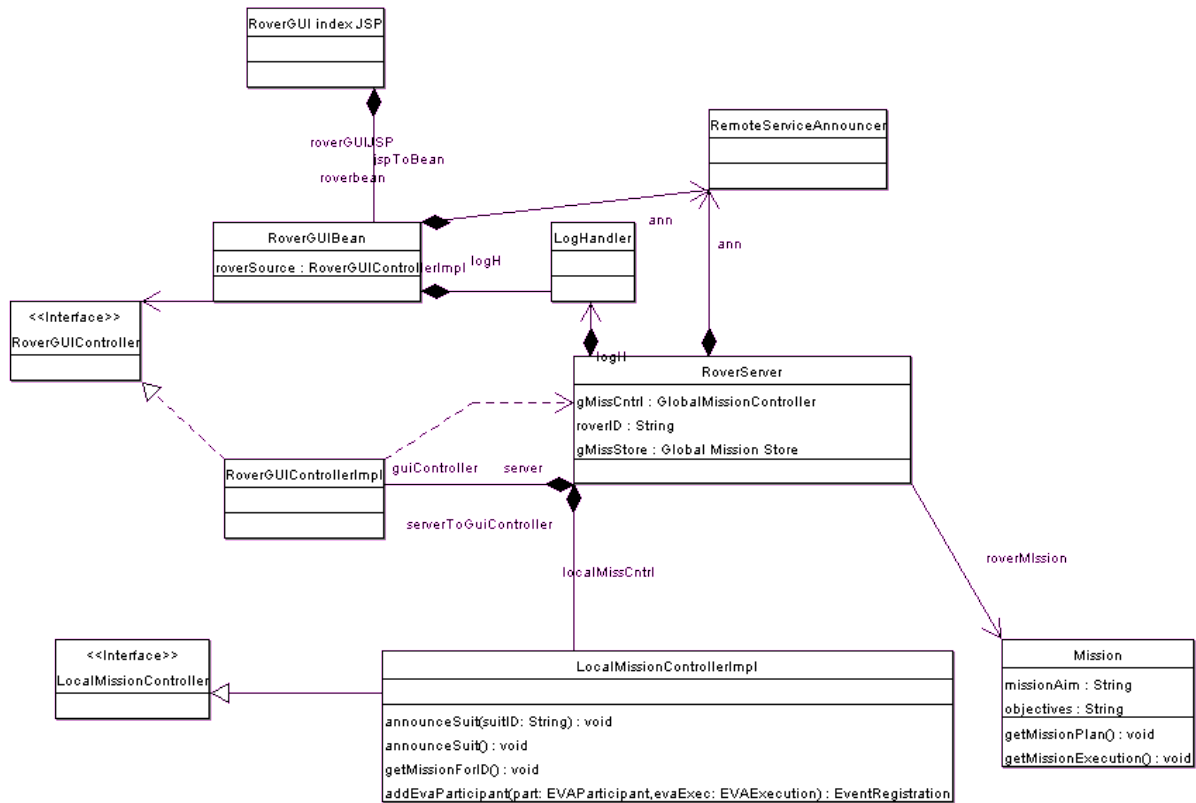
A Conceptual model of the rover system is shown below:



The rover has the following entities:

- **RoverNavigator:** Rover Navigation communicates with the rover GPS and tracks progress. This will also allow for autopilot type features. It also produces position reports for logging and other components.
- **Mission Monitoring:** Mission Monitoring is the software used to track the mission as it progresses through the plan.
- **Communications Facilitator:** The communications facilitator manages communication between the various parties:
 - - argonaut suits
 - - rover operator
 - - remote mission monitors
- **Local Mission Store:** A mission Store holds all details of the mission. It is the local mission environment. That is, stored on the rover.

The implementation of the rover is shown below:

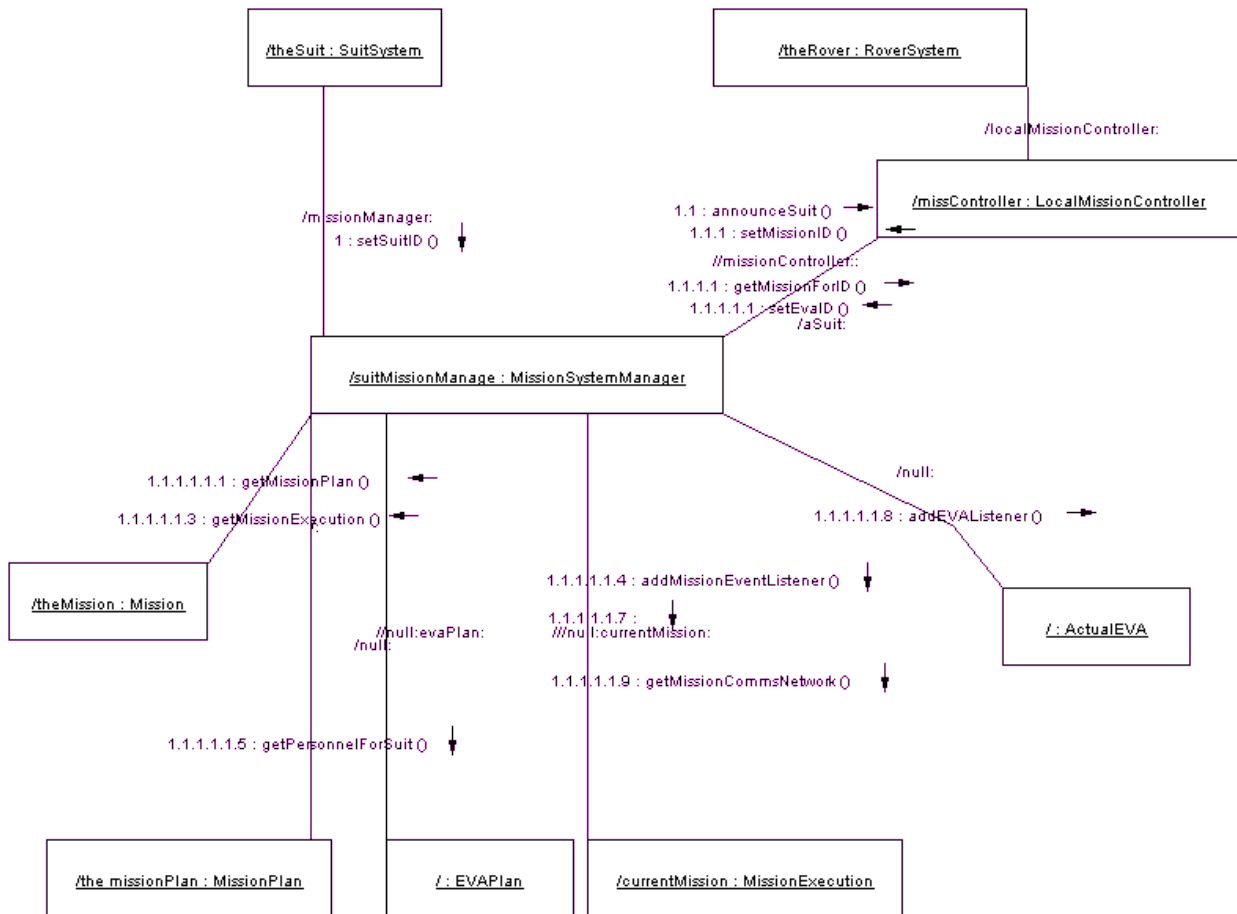


The rover server creates a local mission controller implementation. This provides the local mission controller interface, which is a JINI service. The rover server will discover the global mission controller, and hence the global mission store. From these it will obtain the mission for the rover. The rover then creates a rover gui controller. This is a JINI service that allows a gui to use the rover facilities.

Power Up Use Case

Of particular interest for this first prototype is the comms networks attached to the mission execution and then an eva. You can ask for the comms network. This gives you the list of current participants which can then be added to a mixed audio session in JMF.

We use JINI, and find a look up service. We ask JINI for the local mission controller service. I assume that there is only one mission controller per local network. From the mission controller we can work out where this suit, and hence, it's wearer fits in.



Basically, the suit fires up, looks for the mission controller, sends the the suit id. The mission controller asks each mission what suit(equipment) allocations it has for the suit, in todays date. This tells it a mission to give to the suit. Now that it knows the suit is available, it gives the suit a mission, then, the eva. The suit uses these to find out the missionexecution, and actual eva, from which it gets two comms networks, and adds the suit & personnel to each comms network.

Thus, the suit is added, and the wearer can begin transmitting on the network.

Suit User Interface

The SUIT User interface is a JSP application, run from the palm pilot browser.

General Windows

The SUIT user interface will show the following information on any screen:

- Viewing window – section for whatever the focus of the current screen is.
- Buttons for available actions in bottom row.
- status bar, consisting of sections for:
 - suit operational status
 - mission name, eva name
 - personnel name
 - suit id

Main Screen

The main screen will have the following actions:

- startup suit
- shutdown suit
- communications
- suit configuration

Initially, the main screen will show a status of 'shutdown'. The suit id will be displayed, but no personnel details.

When the user clicks on startup, the status will change to starting. Once the suit has 'started up' (see above for initial power up sequence), the status will change to operational. The current mission and eva will be shown in the status bar.

Any errors will be shown in the viewer window.

The shutdown button will set the status to shutting down. Once the suit has finished 'shutting down', the status will change to 'shutdown'. The shutdown function is the reverse of the startup sequence, and will be described in *tbd*.

The suit configuration button will lead to the configuration window.

The communications button will lead to the communications window.

Communications Window

The communications window will display the current communications networks available, and allow the user to create new comms, switch between current comms, and close comms. The mission, and eva comms default networks should not be able to be closed.

Viewer:

Hyperlink line for each comms network, stating name of network, and number of participants.

Default is that the top two networks are mission, then eva.

checkbox next to each comms session.

Actions:

New – create a new comms network by going to the new comms network screen (TBD)

close selected. Closes the selected comms network.

switch to selected. This will make the selected (one) network the current comms network.

Back to main. Goes back to the main screen.

Status:

as per main screen.

Configuration Window

This window will show the current personnel name and allow it to be changed.

Viewer:

text box and combo showing current personnel name, with the personnel names available for the mission in the dropdown box. If no mission is available, then this will be greyed out.

Actions:

save – saves the settings in the viewer window.

Main window – goes back to the main window.

Rover User Interface

The rover user interface will allow for monitoring the current mission. It is a webapp, but will open up a swing based applet for JMF viewing.

The same general screen guidelines as per the suit apply.

Login Screen

The login screen will allow the user to select the user from a drop down box, and then provide a password. The correct password for the user will allow the user through to the main screen. If the rover does not know it's mission, then the admin login is the only login available.

Viewer:

- user dropdown
- password

Actions:

Login

Status status showing:

- Rover operational status
- mission name
- personnel name
- rover id

Main Screen

The main screen will have the following actions:

- Startup: This will cause the rover software to begin. This will start up the rover server.
- Shutdown. This will cause the rover software to stop.
- Mission. This will show a window allowing the user to define what mission the rover is on, and to monitor the current mission. This allows for seeing each EVA & the data feeds, and reviewing information as well as seeing the current information.
- Communications. This allows the user to communicate with others.
- Navigation. This allows the user to see the planned route and how we are tracking.

Any errors will be shown in the viewer window.

Communications Window

The communications window will display the current communications networks available, and allow the user to create new comms, switch between current comms, and close comms. The mission default network should not be able to be closed.

If a mission monitor is remote, there may be a default mission monitor network. If an EVA is underway, then an eva network will be available.

Viewer:

Hyperlink line for each comms network, stating name of network, and number of participants.

Default is that the top networks is mission.

checkbox next to each comms session.

Actions:

New – create a new comms network by going to the new comms network screen (TBD)

close selected. Closes the selected comms network.

switch to selected. This will make the selected (one) network the current comms network.

Back to main. Goes back to the main screen.

Status:

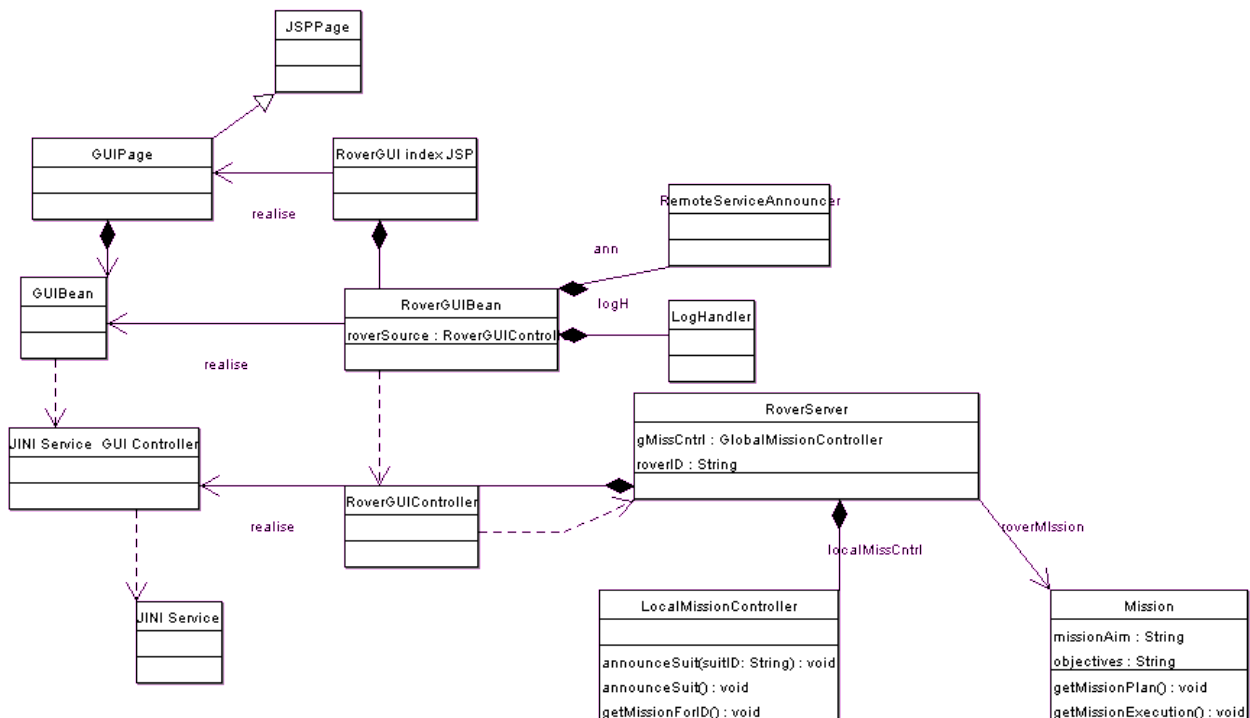
as per main screen.

Mission Planning Interface.

Mission Planning is for making up a plan and transferring it to the rover.

Linkage of UI to Main Systems

The User Interfaces will be implemented in JSP. They will talk via java to the main 'servers' for the suit, rover and monitor. The JSP page will have a Java Bean that will talk using JINI to the Rover service, or Suit service as the case may be.



The diagram shows that a JSP page has a bean. The bean has access to a gui controller. The gui controller in turn has access to the JINI services defined for the given entity. In the case above, the Rover index.jsp has access to the rover gui controller. The gui controller is a JINI service of the rover server. The rover server also provides the local mission controller and the mission for the rover server. The gui controller can access these objects for the gui, and enact changes to the rover. The gui bean in turn displays the data with the jsp screen.

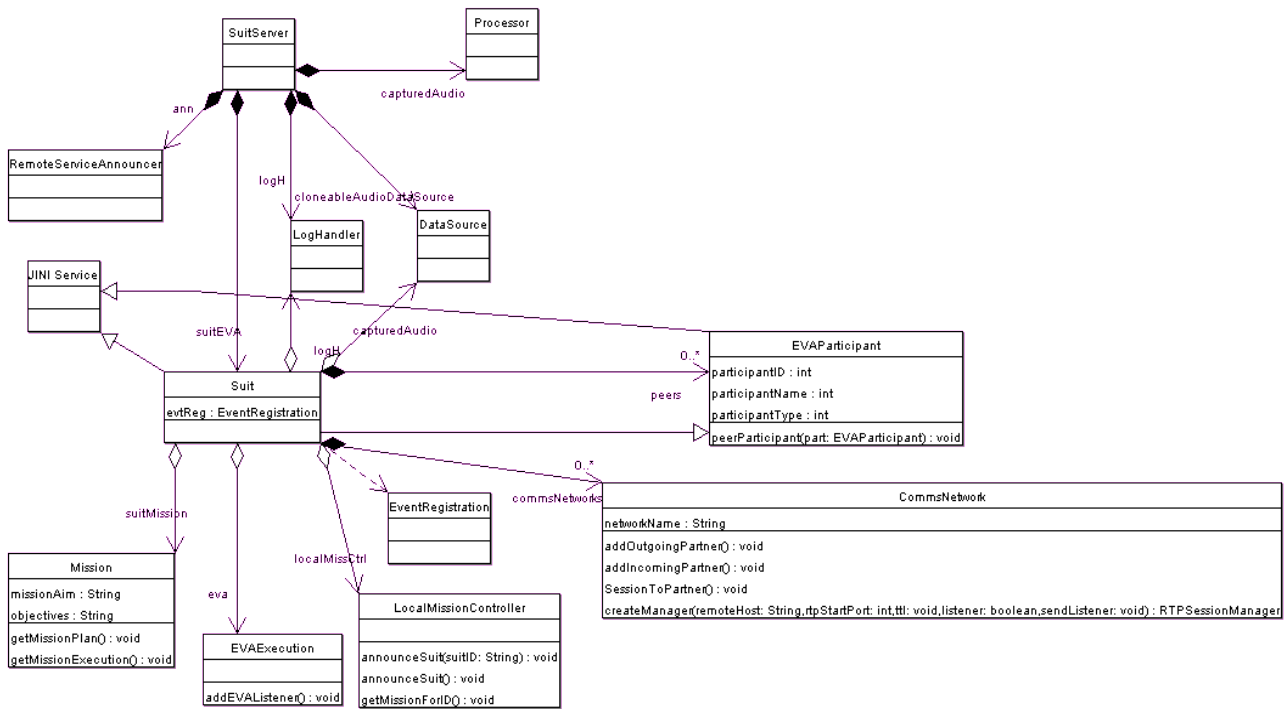
RemoteServiceAnnouncer is a class that records a requested variable, and service name (class name) and then calls back when the service is available. This allows for asynchronous handling of service availability. RSA has the following functions:

1. ServiceAvailability: The client tells RSA what service it is waiting on, and the variable to set, and it will give you an event when it's available. This way you can continue to process and it will just announce when the remote service you want is there.
2. ServiceOutages: We could also implement an outage event, then a restoration event. These would be where the service is pinged, and if the service does not respond, its state changes to 'out'. Once the ping works again, then it is changed to 'restored'.
3. ServiceUnavailable: We should also have a timeout set on our remoteseviceannouncer class so that it can give you a timeout, rather than available event.
4. Registrar Availability management. It should behind the scenes determine if a registrar is available, and retry after a time out if not.

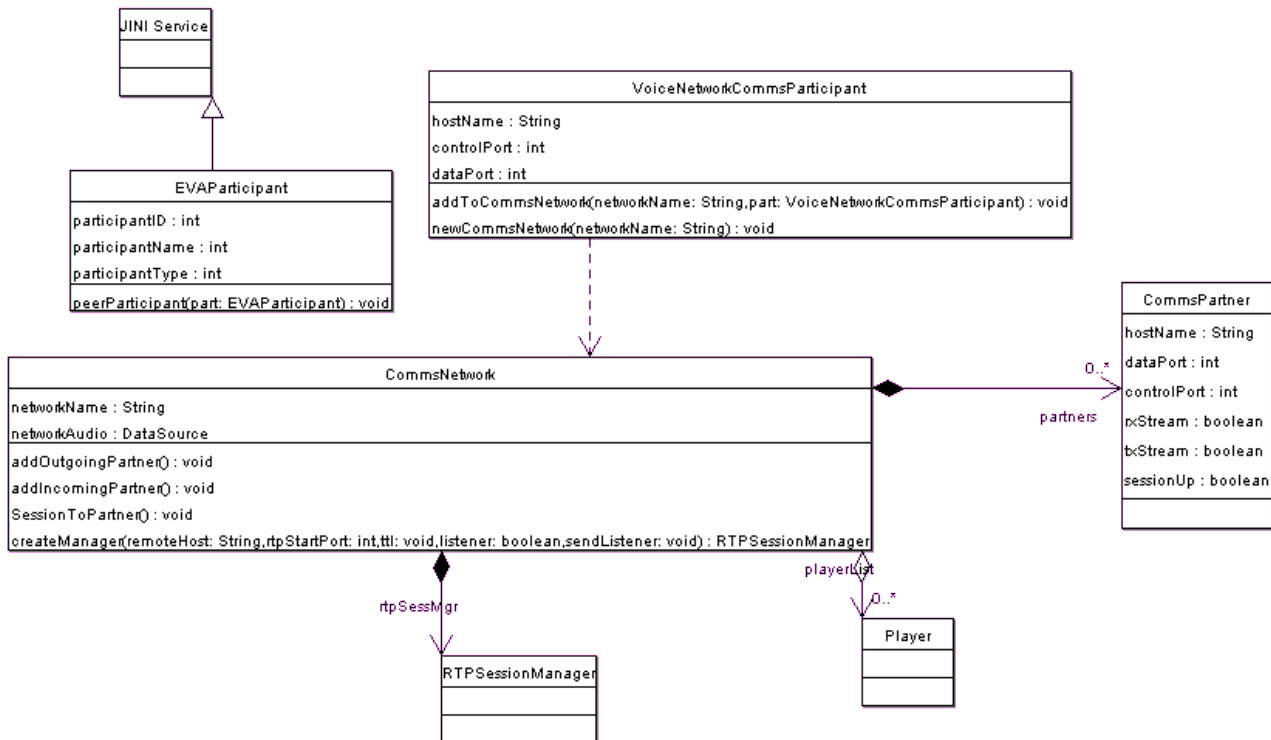
LogHandler provides a standard logging mechanism for logging messages to a nominated file and/or standard out.

Voice Communications Networks

The setup of voice communications networks is a complex affair. The idea is to have any entity (suit, rover, robot) add themselves to the eva as a participant. From there, they can be notified when ever new participants are added to the eva. A participant can then provide a voice communications network participant entity from which they construct the local RTP streams required to establish data streaming over the network.



As shown above, each Suit has a series of communications networks (CommsNetwork). A Comms Network is the class the controls the JMF data streaming, and enables the voice communications to occur.



Parties to the communications network are called VoiceNetworkCommsParticipants. JMF RTP streams require a host and require the data and control ports used to access the streams. These values are used to establish the communications between two machines. So, the Suit class implements VoiceNetworkCommsParticipant:

```

public class Suit implements
    EVAParticipant,
    RemoteServiceAnnouncerListener,
    VoiceCommsNetworkParticipant {
  
```

When a Voice comms network participant is added to a communications network, they are either an outgoing partner or incoming. An outgoing is a data stream that is to be transmitted to all partners, whereas an incoming is one to be received. The outgoing streams will be mixed, then sent off as one stream. The incoming streams will be mixed, then realised through a player.

The CommsNetwork uses a CommsPartner to hold the state of each partner to the communication.

Server Construction and startup.

The Rover Server starts with the roverID on the command line:

```
java com.msa.marseva.servers.RoverServer 3
```

this tells the rover server to start and the command line argument of 3 is the number of the rover. The rover server is a rover that starts, and tries to find the global mission controller. This in turn

provides the global mission store from which the mission for the rover can be discovered. The mission is then loaded into the rover and is available for suits to download using the local mission controller service (see diagram in previous section).

For startup:

1. web server with reggie loaded
2. reggie
 1. If running a test,
 1. start global mission GlobalLevelServer
 2. Run rover test case – this initialises the mission
3. rover gui web app using internet explorer or mozilla
4. click start, and this will start the rover server.
5. The rover server will load up.
6. Reload the web browser, the state should change.
7. Go into missions, start it
8. go into eva 1 and start it.

Project decomposition

The project has a series of sub projects:

Project Name	Objectives	Reference Use Cases	Effort Estimate
GPS Data Stream	<ul style="list-style-type: none"> GPS JMF data capture JMF logging & streaming. Integration of GPS Streams into EVAParticipant class. 	UC 1.3 Rover Movement UC 1.5 Data Logged from Suit	2 Person Months
Audio Data Stream	<ul style="list-style-type: none"> JMF audio data capture JMF logging & streaming. Integration of audio Streams into EVAParticipant class. 	UC 1.6 Automatic Comms Network.	2 Person Months
Video Data Stream	<ul style="list-style-type: none"> Video JMF data capture JMF logging & streaming Integration into EVA Participant 	UC 1.5 Data Logged from Suit	2 Person Months
Voice Communications Network	<ul style="list-style-type: none"> GUI control of JMF streams to implement network switching & mixing 	UC 1.6 Automatic Comms Network. UC 1.12 Separate Voice Network UC 1.6.1	1 Person Months (dependant on basic audio streams working)
GPS Mapping	<ul style="list-style-type: none"> Real time tracking of GPS stream over maps (suit, rover) 	UC 1.7 Rover Operator Mission View	3 Person Months (dependant on GPS Streams, co dependant on mission planning)
EVA Event Management	<ul style="list-style-type: none"> EVA event creation, notation & streaming 	UC 1.9 EVA Event Handling UC 1.15 Event Complete UC 1.10 Jen Joins Johns Event	1 Person Months
Remote Monitoring	<ul style="list-style-type: none"> Remote Monitoring of Mission (eg Hab, Earth Mission Control) & event interjection. 	UC 1.9 EVA Event Handling	1 Person Months

Project Name	Objectives	Reference Use Cases	Effort Estimate
Mission Review & Reporting	<ul style="list-style-type: none"> • Attach summaries & conclusions to mission data • Replay sections of mission • 	UC 2.0 EVA Review	3 Person Months
Mission Planning	<ul style="list-style-type: none"> • Design a mission. • Approve a mission • start a mission 	UC1.1.1 Detailed Planning	2 Person Months
Local Global Mission Store Sync	<ul style="list-style-type: none"> • When a rover returns from out of LAN range, the data needs to be synced 	None	1 Person Months
Local Mission alteration.	<ul style="list-style-type: none"> • Ability to cchange the mission route and other data while mission is underway 	None	0.5 person months (dependant on mission planning)
Capture, Streaming of other data sources from Suit	<ul style="list-style-type: none"> • integration of generic data sources as JMF streams 	UC 1.5 UC 1.14 Instrument data through suit	2 Person Months
Rover Vitals Display	<ul style="list-style-type: none"> • Rover monitoring of vitals, and display of JMF streams. 	UC 1.5, UC 1.7	3 Person Months. Difficult GUI & data presentation. Dependant on data streaming projects
Robot Control	<ul style="list-style-type: none"> • robot control using voice comms • Vocab for shared goals/movements • Conceptual basis for describing robot operations with AI 	UC 1.8 UtilBot Setup UC 1.13 Robot called over voice network.	6 Person Months. Big project. Need to design vocab, and do AI for shared goals/movements etc.
HAB Software	<ul style="list-style-type: none"> • Mission Planning done here + Monitoring of multiple missions • Resource/ Personnel management 	None	2 Person Months