

MaROS Strategic Relay Planning and Coordination Interfaces

Daniel A. Allard
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109
818-354-4344
Daniel.allard@jpl.nasa.gov

Abstract—The Mars Relay Operations Service (MaROS) is designed to provide planning and analysis tools in support of ongoing Mars Network relay operations.^{1 2} Strategic relay planning requires coordination between lander and orbiter mission ground data system (GDS) teams to schedule and execute relay communications passes. MaROS centralizes this process, correlating all data relevant to relay coordination to provide a cohesive picture of the relay state.

Service users interact with the system through thin-layer command line and web user interface client applications. Users provide and utilize data such as lander view periods of orbiters, Deep Space Network (DSN) antenna tracks, and reports of relay pass performance.

Users upload and download relevant relay data via formally defined and documented file structures including some described in Extensible Markup Language (XML). Clients interface with the system via an http-based Representational State Transfer (ReST) pattern using Javascript Object Notation (JSON) formats.

This paper will provide a general overview of the service architecture and detail the software interfaces and considerations for interface design.

TABLE OF CONTENTS

1. INTRODUCTION.....	1
2. INFORMATION SPACE AND ARCHITECTURE OVERVIEW	2
3. DATA INTERFACES	4
4. CONCLUSIONS	10
REFERENCES	11
BIOGRAPHY	11

1. INTRODUCTION

Modern Mars surface missions rely primarily upon relay orbiters to provide delivery of uplink commands and sequences as well as downlink science and engineering data products. The primary driver for the use of orbital relay by surface missions is to conserve lander energy. It takes a great deal more energy for a lander to communicate with a

Deep Space Network (DSN) antenna than a nearby orbiter, energy that can be used for greater movement and science activities. Orbiters do not have the energy constraints of a typical lander and furthermore have longer view periods with DSN antennas because of the height of their orbits. This has led to relay becoming the standard approach for communications between earth systems and remote landed assets. The Mars Exploration Rovers (MER) mission in particular has made extensive use of orbital relay [1], and the Phoenix mission relied entirely upon relay for earth-to-lander communications [2].

Strategic and tactical relay planning and execution involve not only activities on board lander and orbiter spacecraft, but require a significant amount of coordination between lander and orbiter mission teams. Relay mission teams must manage a number of conflicting wants and needs, including the fact that all missions currently or potentially supporting relay have their own individual set of science objectives.

Since the start of Mars mission relay, a relay process has been put in place to standardize the way relay is managed. However, this process involves a number of different steps across missions and there is a general lack of data accountability across projects. One primary concern is that it can be very difficult to determine the exactly what is going on at any given time due to the scattered data across various files and systems.

The short-lived Phoenix mission introduced a number of elements into the relay picture. For one, it was the first time the Mars Reconnaissance Orbiter (MRO) was used in a primary relay role [2]. Additionally, with Phoenix in a polar location it had a large number of relay opportunities per day with the desired utilization of relay passes changing from week to week and even day to day.

In an attempt to better account for this dynamic relay picture over the Phoenix mission timeframe a new system called Relay Data Engineering (RDE) was introduced [3]. This system provided auto-tracking of the total data volume received on a pass-by-pass basis and enabled trending of the life-of-mission data volume. However, limitations in the overall relay process prevented key capabilities such as accurate identification of predicted verses actual downlink data volumes.

¹978-1-4244-3888-4/10/\$25.00 ©2010 IEEE.

² IEEEAC paper #1509, Version 1, Updated November 1, 2010

In 2008 work was begun to overhaul the end-to-end relay process including the implementation of a new system, the Mars Relay Operations Service (MaROS). The goal of this system is to standardize the relay approach across all relay assets and provide a centralized data store enabling visualization of the current, accurate relay picture.

The first phase of implementation addresses the strategic process. During this phase, all user data management transactions are provided via upload and download of relay data files to the service. A web user interface provides timelines and correlated charts of overflight data.

2. INFORMATION SPACE AND ARCHITECTURE OVERVIEW

Strategic Relay Coordination

The phase 1 strategic coordination process involves the upload and download of a set of files to and from MaROS. These data required for predictions, calculations and decisions concerning the relay process. The following table summarizes the input and output data utilized in the strategic coordination process:

File Type	Acronym	Content
Light Time File	LTF	Light times to and from Earth and Mars
Orbiter Sequence of Events File	OSOE	DSN Antenna tracks, transmission data rates, system configuration parameters
Lander Orbital Propagation Timing Geometry	LOPTG	Geometric pass data
Overflight Summary File	OSF	Correlated report summarizing information published to MaROS (Output)
Ace Schedule	None	Staffing schedule for flight mission controllers (“aces”)
Orbiter Request File	ORF	Lander desiresments for relay utilization
Overflight Acknowledgement File	OAF	Orbiter team agreements.
Scorecard	None	Post-pass statistics
Overflight Performance Assessment File	OPAF	Pass predict and actual data

Table 1: Relay Coordination Data Files

Each of these files was used in or had an equivalent type in the legacy system. All legacy files were implemented in

proprietary formats or in some cases only available as Excel spreadsheets. Many of these files were utilized in point-to-point transactions and in some cases extracted and exposed in other downstream files, and in each case required special parsers to interpret.

Certain files such as the OSOE and Scorecard were provided in structurally “loose” formats that could not be reliably parsed. As part of the overhaul of the relay coordination process, most of these legacy files were re-designed to the Extensible Markup Language (XML) format to ensure machine compatibility and enhance overall readability and adaptability.

The legacy coordination process involved point-to-point file deliveries and email-based notifications over the course of a monthly planning cycle. With the new process, lander and orbiter teams provide data files to MaROS, which extracts and persists file data and forwards notifications to subscribed end-users. The following diagram describes the lifecycle of coordination data from long-range calculations through the execution of the overflight and subsequent provision of post-pass analysis information:

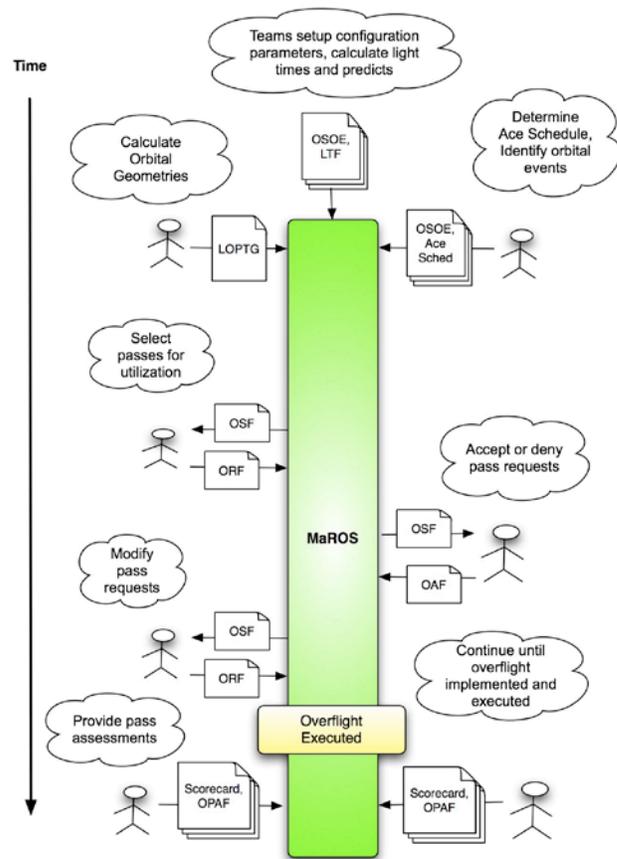


Figure 1: Overflight Lifecycle

Each data type provides value to the overall process.

- Lander geometries are predictions of relay pass behavior and are used in the formulation of requests.
- Requests and Acknowledgements represent decisions made by lander and orbiter teams.
- Light time data, orbiter data and ace schedules are all referenced when identifying pass latencies. Scorecard and post-pass assessments (OPAF) provided lessons learned for use in future decision making.

Details of each data type and transaction formats will be described as part of the information management discussion.

System Architecture

As a full description of the system architecture is beyond the scope of this paper, this section will attempt to summarize key relevant architectural components.

The system is presented as a “service” to end users and end user systems. Users perform a set of transactions with the service, either to provide data to or to retrieve data from the service. It is the role of this service to ingest, persist and provide access to data, and also to calculate several types of derived data and further correlate everything together.

The architecture style uses a centralized database, where external clients publish to and fetch data from a central service. Data from publish transactions is persisted by the service for use by follow-up transactions. Clients interface with a “service layer” which in turn utilizes a relational DBMS for persistence. Note that external clients do not have direct access to the DBMS, they interact with data only through the service layer. This prevents client dependence on specific database structures and naming conventions and thus enables the database structures to evolve without unduly impacting external systems.

MaROS server components are deployed to institutionally maintained hardware visible to the Jet Propulsion Laboratory (JPL) flight operations network. This hardware includes a set of standard software packages and support for MySQL databases. The system provides load balancing and backup functions enabling MaROS to meet key operational performance requirements such as a high level of accessibility and minimal down times.

The following diagram depicts the core components of the architecture:

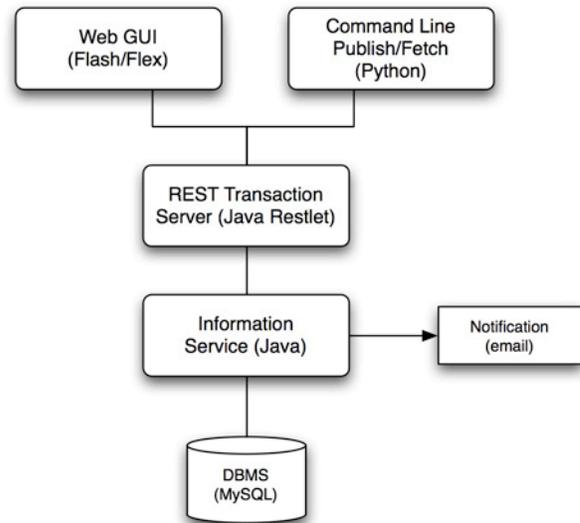


Figure 2: Application Architecture

At the heart of the architecture is the Information Service layer that performs all data management business logic functions. These include upload of data for long-term persistence and access by clients, download of data for client correlation and visualization, calculation of derived data types such as latencies and conflicts, and notifications to external users.

Long-term persistence is handled by a relational database system. MaROS utilizes a MySQL as this is fully supported in the deployment environment.

A transaction layer sits above the service layer, intercepting client requests and delivering resulting information.

Finally, clients interface with this transaction layer. Two types of clients are provided, a rich web user interface built with Flash and FLEX, and a thin command line client built with Python.

Much of the data published to the system is provided via files. When a file is presented to the system it is parsed and the data extracted, reformatted and published to the database. The system maintains a history of file transactions in separate tables from the source data.

File data may be provided any number of times over the same time range, and these represent changes to the data set rather than just additions. For example, if a set of requests is provided over a time range where requests are already present, the new data represents “updates” to the data set as opposed to “inserts”.

Most input files are generated via other software tools (e.g. geometric view period calculations), however this is not exclusive, as at least the schedule of orbiter “Ace” staffing

availability is generated by an operator “by hand” and this is expected to be the case for the near future. All input files adhere to a format as specified in a related System Interface Specification (SIS).

Once data is uploaded and extracted from a file it is transformed into a set of objects within the server virtual machine. These objects are transformed into database records for persistence and to JSON structures for delivery to client applications such as the web user interface. The following diagram shows the relations between the different data structures:

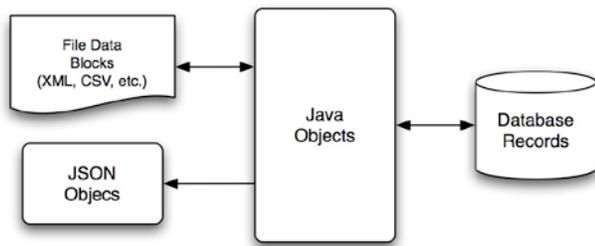


Figure 3: Data Structure Relations

The intermediate Object structures are utilized by all business logic components dealing with upload, download and computational logic.

Service Interface

The chosen interface approach to the service is via http using a ReST (Representational State Transfer) style. With this style, a set of “resources” are defined that, in the case of MaROS, represent types of transactions (publish or fetch) each with an associated set of parameters driving the transaction. Behind the scenes, service layer applications written in Java execute transactions and perform computations.

Service Clients

A web graphical user interface (GUI) and command line scripts are both provided for end-user interaction with the system. For the first phase of software development the web user interface provides timeline based visualization of system data, summary pages correlating planned versus actual results and charts of predicted and actual pass performance. During this phase, all modification of data is restricted to command line file update transactions.

Both types of clients perform service transactions via ReST. Command-line clients perform publish transactions and query a correlated summary file from the service. The web GUI client performs fetch calls for the “raw” object data such as sets of orbital view periods, orbital events and lander-orbiter request conflicts.

Data Interface Formats

MaROS supports a range of formats for data interfaces. Input interfaces include XML, CSV and some legacy text formats, while output interfaces include JSON, XML, and CSV. The following provides an overview of some formats supported by the system.

Extensible Markup Language (XML)

XML is a flexible, extensible format for creating structured documents [4]. It sees growing use by mission data systems to represent a wide range of information types. Most of the new interfaces generated as part of the MaROS system engineering development are in XML. XML is very human-readable from the structuring of the data elements. The primary value of the XML is the extensibility of the structure. New elements can be added to existing structures with very little impact on downstream parsers, as most parsers will simply ignore unrecognized elements.

Comma Separated Value (CSV)

Comma separated lists of data are a common way to share information between software systems, largely due to the relative ease of parsing the structure. By their nature CSV interface files must be rigidly formatted, as the addition or removal of a column will always impact producers and consumers of the file and would certainly require a change to the SIS.

The CSV format is useful when dealing with tabular data exported from tools such as Excel. The Ace Schedule is a specific example, where the actual schedule is managed as an Excel spreadsheet. It is also desired as an output format, to import into other spreadsheets and also for quick visual comparison of data record values, however for that purpose additional formatting is usually required to forcibly justify columns and visually align records.

Javascript Object Notation (JSON)

The JSON is a lightweight, structured data format based off a sub-set of the JavaScript programming language [5]. The structure brings with it less structural overhead than the equivalent XML and is useful for interactions between interfacing software (e.g. GUI) clients and providing information services. It is similar to the XML in terms of the ease of extension of the format. JSON parsers and encoders are available in JSON and Flex.

3. DATA INTERFACES

To understand the drivers for use of different interface formats we must discuss the different types of information managed by the system.

The fundamental role of the system is to coordinate utilization of a pass “overflight”. An overflight occurs during a range of time when an orbiter is in view from a lander on the ground, representing a *potential* communications opportunity. MaROS information management is centered on the allocation of these overflights for use by landers for relay. The actual determination of what overflights to use depends upon a number of factors including the total data transmission capability of the overflight, the amount of time to transmit or receive data from the overflight (the “latency”) and whether or not the overflight is in conflict in some way. Nearly all of the information managed by the system is related to overflight management.

Information is published and modified at different phases in the operations cycle. The first phase of system development supports the *Strategic Planning* cycle as well as post-pass performance assessment. The strategic planning cycle runs from the initial identification of view period opportunities until the final acknowledgement of orbital requests for use.

Strategic planning data can be provided far in advance of the actual relay event. Such data may be published weeks, months, or potentially years in advance, however the physical nature of the relay environment makes it difficult to predict certain information with great accuracy. Typically, strategic planning data is provided as part of a monthly relay cycle, with most data provided one or more weeks in advance of the overflight pass.

Lander View Periods

Overview

Lander view periods represent the of time span from which a “celestial entity” is in view in the sky from horizon to horizon. Celestial entities include objects such as Orbiters, the Martian Moons and the Earth. Lander navigation teams typically calculate this information and include additional geometric data such as the maximum elevation for the pass, orbital inclination, etc. Lander view period data is typically generated for two week to one-month time spans and provided to mission systems via a Lander Orbital Propagation Timing Geometry (LOPTG) file. Note that, while lander view period files could potentially be calculated out for years of orbits, the accuracy of the calculations decrease over time due to small or in some cases large changes in the orbital geometry. The Mars Reconnaissance Orbiter, for example, sits in a low orbit which is affected by atmospheric drag and is very difficult to predict over long times. This lander view period data forms the basis for eventual requests.

Input File Format

The format of an LOPTG file is a legacy CSV-separated text format with no extra structural framework. The result is

a file of minimal size but which requires special parsers to interpret, and it is not especially human readable unless the reader is already highly familiar with the file format. An update of the file format to XML was considered during the early engineering of MaROS. However, any updates to this particular file would impact a number of other software subsystems including providers of the file and other systems ingesting the file. It was decided that the cost of these impacts was outside the scope of the current phase of work, and so MaROS was implemented with a specialty parser to support the legacy format.

The following are two example lines of the LOPTG file content (with newlines added for some legibility):

```
ORBRISE, 53, 2009-238T21:18:13.733, 2.4550703E+06,  
66.183,1.97329E-09, 1.841340932E+02,  
1.676537E+03, 3.79607537E+03,  
8.85315041E-03, 9.2936663E+01,  
5.7220487E+01, -1.02267752E+02,  
6.13504877E+01,  
ORBMAXELV, 74, 2009-238T21:21:36.224,  
2.4550703E+06, 66.183, 6.9127242060482,  
2.73586681E+02, 1.019251655732523E+03, ...
```

The above information represents time and elevation data concerning the moment of an orbiter rising at the horizon and the moment that an orbiter sits at its highest maximum elevation in the sky during a pass. To understand the specific meaning of each data parameter, an ingester of the file must consult the relevant interface specification document (the SIS).

Upon publication of this file data to MaROS, the contents are parsed into Java view period objects and persisted into a set of tables in the database. The publication of these data triggers calculations including the identification of planning warnings and estimation of uplink and downlink latencies.

Web UI Output Format

Once data has been persisted to the database it is available for download for various purposes. This includes the basic visualization of data on the web user interface. Data is provided to the web UI by the MaROS service via ReST invocations in the JSON notational format. As described earlier, the JSON is a relatively lightweight format with enough structure to represent complex object types.

Not all of the information extracted from the LOPTG file is necessary for the visualization of data on the web GUI. For example, the LOPTG includes lander positional data that is not required for viewing upon the web timeline display. As responsiveness of the web GUI is highly desired, the bare minimum of data is delivered with any ReST call.

The following is a sample JSON structure representing a Lander View Period:

```
[
  {
    "LanderViewPeriod": {
      "ViewPeriodType": "ORBITER",
      "StartTime": "2009-299T20:31:54",
      "EndTime": "2009-299T20:48:22",
      "MaxElTime": "2009-299T23:40:00",
      "MaxEl": "30.1951",
      "OverflightId": "ODY_MRB_2009_299_04",
      "LanderId": "MRB",
      "OrbiterId": "ODY"
    }
  }
]
```

Note that the “name-value” pair structuring of this format makes for a highly readable format, while at the same time the format incurs minimal additional structure to enable complex object structures.

Both the Java service and Flash/FLEX client utilize libraries supporting encoding and decoding of the JSON format.

Light Time Data

Overview

Light time data is a series of data calculations of the time it takes light to travel to Mars and back at any given moment. These times change as the distance between Earth and Mars changes over the course of their orbits. For example, in 2006 when the Earth was closest to Mars the light time was ~300 seconds, while late in 2007 the light time was ~1300 seconds. As with the LOPTG, light time data is provided in a legacy file format. Light time data is utilized by the web GUI and is an important parameter in forward link and return link latency calculations.

Input Format

The following snippet shows the format of the data of the light time file:

Applicable Time	DOWN-LEG	UP-LEG
06-069/00:00:00	713.904	713.763
06-069/06:00:00	715.142	715.000
06-069/12:00:00	716.379	716.237
06-069/18:00:00	717.616	717.474
06-070/00:00:00	718.852	718.710

The “Up-Leg” time is the light time from Earth to Mars, and the “Down-Leg” is the time from Mars to Earth. As with the LOPTG, a specialty parser was implemented to interpret the data.

Unlike lander view periods, light time data calculations rarely change over time and so a typical light time file may contain years’ worth of data.

Ace Schedule

Overview

The ace schedule is a table of the work shifts of the staffing of relay aces. An ace must be on duty to provide relay services for all nominal relay uplink operations. Ace schedule data is used by the web GUI as a part of the overall visualization of data by time, but is also incorporated into latency calculations as the “nominal” forward link latency (i.e. the time before a pass that an uplink product must be provided) includes the identification of a time that an Ace staffer is on duty.

Input Format

These schedules are managed via Excel spreadsheets. For this reason it is most convenient for these files to be provided in comma separated value (CSV) format instead of XML, as Excel can be readily exported into CSV without requiring any further processing.

The following example shows the format of the input file:

```
CCSD3ZF0000100000001NJPL3KSOL015$$MARK$$
MISSION_NAME = MARS_RECONNAISSANCE_ORBITER;
SPACECRAFT_NAME = MARS_RECONNAISSANCE_ORBITER;
DATA_SET_ID = ACE_SCHEDULE;
FILE_NAME = Ace_schedule_Mar08_12.ace;
APPLICABLE_START_TIME = 2009-060T00:00:00;
APPLICABLE_STOP_TIME = 2009-074T00:00:00;
PRODUCT_CREATION_TIME = 2009-037T22:55:57;
CCSD3RE000000$$MARK$$NJPL3IFOM01300000001
$$EOH
2009-060T17:00:00; 0T02:00:00; normal;Ace On-
Console ; 303-971-xxxx or 303-971-xxxx
2009-060T21:00:00; 02:00:00 ; normal ;
unassigned;303-971-xxxx or 303-971-xxxx
2009-061T17:00:00 ; 0T03:00:00; normal; Bubba
Smythe or Bertha Smith; 818-354-xxxx
2009-062T17:00:00; 14T00:00:00; on-call; John Doe;
818-354-xxxx
$$EOF
```

While this format is relatively easy to produce and ingest, it is not as extensible and lacks the “declarative clarity” of the XML.

Orbiter Events

Overview

The Orbiter team provides the Orbiter Events file to the MaROS system. This file consists of a set of different types of “events” used for a variety of purposes by the MaROS system and end users of the system. These events include:

- Time windows of Deep Space Network (DSN) uplink and downlink antenna to orbiter track,
- Data rates and efficiency of data transfer at any point in time,
- Orbit number changes over time,
- “Non-relay” periods where, for one reason or another, the orbiter will not be capable of performing relay for the lander regardless of the view period opportunity.

Orbiter event data is used for a variety of purposes by MaROS, including latency and conflict calculations.

Legacy Orbiter Events files were of a “loose”, unstructured format that were difficult to parse and interpret. Furthermore, new types of events are envisioned so an extensible format was desired. For these reasons XML was chosen to replace the legacy format.

Input Format

The following example shows some of the different data in the file:

```
<OrbitNumber StartTime = "2009-260T04:20:15.161">
  <Orbit>34417</Orbit>
</OrbitNumber>
```

```
<DSNDownlink StartTime = "2009-260T04:18:47.025">
  <Duration>0T00:23:49.447</Duration>
</DSNDownlink>
```

```
<DataRate StartTime = "2009-260T09:52:23.327">
  <Rate>39816</Rate>
  <Efficiency>0.129825</Efficiency>
</DataRate>
```

As shown, each of the different event types listed in the file has different data elements and different numbers of elements. Only one parameter is common across all event types, the start time of the event. Because this value is common (and required) for all entries, it is included as an attribute of the event rather than an element, with all of the other supporting data provided as elements.

Late in the first phase, additional relay system parameters were identified for inclusion in the Orbiter Event File supporting latency and conflict calculations. These new data types were added to the file format with minimal impact upon the file ingestion logic, largely due to the extensibility of the XML, where updating other formats such as CSV would have been more problematic.

As with View Period data, the Orbital Events are provided to the web GUI via a JSON structure similar to the Lander

View Period structure, as is all of the rest of the ingested system data.

Overflight Summary File – Geometric Data

Overview

The Overflight Summary File (OSF) is introduced with the MaROS system to replace certain legacy relay planning products. The contents of the file represent the “state” of the relay process over a requested period of time. The file includes data correlated from multiple input sources as well as calculated values. A variety of filters can be applied upon generation of this file including filters on specific landers and orbiters and filtering out passes that do not meet a minimum duration or minimum maximum elevation.

Both the content and use of the OSF change over the course of the planning lifecycle. Typically, the OSF is first requested after the LOPTG, Orbiter Events, Light Time and Ace Schedule files have been published. At this point the OSF will contain basic geometric view period data and timing data such as the orbit number at the start time of the pass and any pass latencies calculated from view period and orbiter event data.

Output Format

The following shows the output of one summary element within the OSF at this point:

```
<OverflightSummary
  OverflightID = "ODY_MRA_2009_274_04"
  SecondaryID = "ODY"
  OrbiterRiseTime = "2009-274T20:49:05.495"
  OverflightDuration = "0T00:15:28.076"
  MaxElevation = "22.2326757448093"
  ConflictType = "none">
  <OverflightTiming
    OverflightID = "ODY_MRA_2009_274_04"
    RequestType = "geometry">
    <Orbit>34595</Orbit>
    <LMST>2043T00:00:00</LMST>
    <FirstBitTime>
      2009-275T00:13:39
    </FirstBitTime>
  </OverflightTiming>
</OverflightSummary>
```

From this set of core geometric data, the lander team identifies a set of passes desired for relay utilization and generates an “Orbiter Request File (ORF) for submission to the system.

The OSF format will be revisited at later points in this discussion.

Orbiter Requests

Overview

When a lander team decides that an overflight should be used as a communications pass it is presented as a “request”. Typically these requests are derived from view period geometries, adjusted by a small amount of time from the start and end time of the pass. Requests include additional information such as positional data (roll, yaw and pitch), “pass-through” relay parameters and priority of the pass. Three types of requests can be provided to the system: tentative, proposed and “formal” requests. These different types of requests allow for different types of negotiation processes between lander and orbiter teams; for example, orbiter A may require a proposal-and-acceptance process, while orbiter B may simply accept formal requests up front.

In the first phase of deployment, orbiter requests are produced as part of scripted ground data system processes as opposed to direct selection via a user interface or exported from another file type (e.g. Excel).

Input Format

The following is an example of a simple request, as it would appear within an ORF:

```
<OrbiterRequest
  OverflightID = "ODY_MRA_2009_274_04"
  RequestType = "request">
<RequestCategory>contingency</RequestCategory>
<HailStartTime>2008-274T20:49:05</HailStartTime>
<HailDuration>00:10:00</HailDuration>
<LinkType>return</LinkType>
<ForwardRate>8</ForwardRate>
<ReturnRate>128</ReturnRate>
</OrbiterRequest>
```

The content of the request may vary in terms of what elements are provided. A valid request may only contain basic timing parameters such the start time for the orbiter to hail the lander and the type of the request. Another valid request may contain any of up to nineteen total parameters including data rates, response times, pass through parameters, etc. Again the extensibility of the XML is quite valuable in the handling of the variable number of parameters.

Eventually the lander team decides upon a set of passes for request and submits the ORF to the system. The submission of this file triggers the calculation of new latencies and conflicts associated with the requests in the file and triggers notifications (in the first phase via email) to the other user teams that a new set of requests is available.

In response to this notification, the orbiter team requests a version of the OSF with the requests included to further process into an eventual acknowledgement.

Overflight Summary File – Requests Submitted

Overview

Once requests have been submitted, the content of a downloaded OSF will now include this new request information as well as any latencies or conflicts that were generated in response to the ORF submission.

Output Format

The following example shows the OSF updated with the new data present:

```
<OverflightSummary
  OverflightID = "ODY_MRA_2009_274_02"
  ...
  ConflictType = "request">
  ...
<OrbiterRequest
  OverflightID = "ODY_MRA_2009_274_04"
  RequestType = "request">
  <RequestCategory>contingency</RequestCategory>
  <HailStartTime>2008-274T20:49:05</HailStartTime>
  <HailDuration>00:10:00</HailDuration>
  <LinkType>return</LinkType>
  <ForwardRate>8</ForwardRate>
  <ReturnRate>128</ReturnRate>
</OrbiterRequest>
  <OverflightTiming
    OverflightID = "ODY_MRA_2009_274_02"
    RequestType = "request">
    <Orbit>34588</Orbit>
    <LMST>2042</LMST>
    <FirstBitTime>
      2009-274T10:23:48
    </FirstBitTime>
    <LastBitTime>
      2009-274T10:23:48
    </LastBitTime>
  </OverflightTiming>
  <OverflightConflict
    OverflightID = "ODY_MRA_2009_274_02">
  <ConflictDescription>
    Overflight Conflict UNACKNOWLEDGED_REQUEST:
    lander MRA request for orbiter ODY has not been
    acknowledged from 2009-274T08:10:13 to 2009-
    274T08:24:49</ConflictDescription>
  </OverflightConflict>
</OverflightSummary>
```

Note that the format of the elements of the request data is essentially the same as the format of the corresponding request file and eventual acknowledgement file. This

enables external scripting processes to generate ORF and acknowledgement files in a relatively straightforward manner from the original OSF.

As mentioned, this new OSF contains additional data that was not present in any of the input files, specifically the calculated latencies (the First Bit and Last Bit downlink times) and the identified conflicts. In this case, the conflict identified is that the request has not yet been acknowledged by the orbiter team, and therefore is not actually scheduled for relay.

The orbiter team processes this OSF file containing the lander requests. In turn, the orbiter team generates a new file of acknowledgements to the orbiter requests.

Overflight Acknowledgements

Overview

Overflight acknowledgements represent the “ok or not ok” from the orbiter team as to whether or not a set of requests is going to be implemented. Acknowledgements are published via an Overflight Acknowledgement File (OAF). The format is fundamentally the same as that for the Overflight Requests, with a small amount of difference in the total set of elements that can be provided.

Input Format

The following is an example of a record from within an acknowledgement file produced in response to a request in the OSF:

```
<OverflightAcknowledgement
  OverflightID = "ODY_MRA_2009_274_04"
  ackType = "implemented">
<HailStartTime>2008-274T20:49:05</HailStartTime>
<HailDuration>00:10:00</HailDuration>
<LinkType>return</LinkType>
<ForwardRate>8</ForwardRate>
<ReturnRate>128</ReturnRate>
</ OverflightAcknowledgement >
```

In this case, the acknowledged values are the same as the requested values, meaning that the orbiter team has accepted the request “as is” for implementation as a relay pass.

When the acknowledgement file is published to the system conflicts are re-calculated and end-user systems are notified of the new acknowledgements as well as any new conflicts that might be generated in response to the requests and acks data not matching.

Overflight Summary File - Acknowledgements Submitted

Overview

Once the OAF has been published with passes listed as “implemented” the long-term relay process is essentially complete.

Output Format

A summary of the process state can be downloaded as an OSF as the following shows:

```
<OverflightSummary
  OverflightID = "ODY_MRA_2009_274_02"
  ...
  ConflictType = "none">
  ...
<OrbiterRequest
  OverflightID = "ODY_MRA_2009_274_04"
  RequestType = "request">
  <RequestCategory>contingency</RequestCategory>
  <HailStartTime>2008-274T20:49:05</HailStartTime>
  <HailDuration>00:10:00</HailDuration>
  <LinkType>return</LinkType>
  <ForwardRate>8</ForwardRate>
  <ReturnRate>128</ReturnRate>
</OrbiterRequest>
<OverflightAcknowledgement
  OverflightID = "ODY_MRA_2009_274_04"
  ackType = "implemented">
  <HailStartTime>2008-274T20:49:05</HailStartTime>
  <HailDuration>00:10:00</HailDuration>
  <LinkType>return</LinkType>
  <ForwardRate>8</ForwardRate>
  <ReturnRate>128</ReturnRate>
</ OverflightAcknowledgement >
</OverflightSummary>
```

Now that an acknowledgement has been submitted in response to the request and the parameter values match, there is no longer a related conflict in the system and so there is no longer a conflict included as part of the OSF data.

Post Pass Assessment

After an overflight occurs, lander and orbiter teams generate assessments of the pass performance for provision to the MaROS system. There are two types of files that are generated: a “scorecard” snapshot of key pass performance values and “overflight performance assessment files” containing time-ordered data of a variety of information types.

Scorecard

Overview

The scorecard is essentially a snapshot of key meta-data information concerning the behavior of the pass. These include the total volume of data uplinked or downlinked,

minimum, maximum and average transmitter power levels and total number of frame and packets sent.

Input Format

The following is a small sample of the contents of a scorecard:

```
<ScorecardEntry OverflightID =
  "MRO_MRB_2008_071_02">
  <SessionAttributes>
    <SessionStartTime>
      2009-071T09:27:55.208
    </SessionStartTime>
    <SessionEndTime>
      2009-071T09:32:55.208
    </SessionEndTime>
    <LinkType>both</LinkType>
    <ForwardRate>8</ForwardRate>
    <ReturnRate>128</ReturnRate>
  </SessionAttributes>
  <LinkConditions>
<AntennaType>helix</AntennaType>
    <LanderPitch>0.25</LanderPitch>
    <LanderYaw>137.6</LanderYaw>
  ....
```

With the information from the scorecard, the actual state of the pass may be compared with the planned or predicted states. It is typically the first place that mission teams look when issues are found with the performance of a pass.

It is envisioned that new data types might be added to the scorecard in the future and so again the XML is a useful format.

Overflight Performance Assessment File

Overview

Other forms of pass assessment may be provided via an Overflight Performance Assessment File (OPAF). This file contains sets of data representing both predictions and actual calculations and measurements ("profiles") of a variety of different pass parameters. These include information such as the overflight elevation and transmitter power as a function of time. The primary use of this information is plotting these curves together to analyze trends and identify issues.

Input File Format

The file format is designed such that new types of profiles could be added at any point, with the only restriction that key fields are included such as the overflight ID, profile name, the type, units and the time-ordered values.

The following is an example snippet from an OPAF:

```
<Profile
  OverflightID = "MRO_MRB_2008_067_01"
  ProfileName = "Bytes Received During Overflight"
  ProfileType = "reported"
  ProfileUnits = "Bytes" >
  <ProfileEntry
    Time = "2008-067T12:32:21">0</ProfileEntry>
  <ProfileEntry
    Time = "2008-067T12:33:21">100</ProfileEntry>
  ...
```

With the overflight ID and profile definition parameters being both common and required for each profile, they are included as attributes of the block.

4. CONCLUSIONS

MaROS Interfaces

The MaROS system supports a variety of file formats for the ingestion and retrieval of relay planning and post-pass assessment data. In the first phase of development, utilized file formats include:

- XML was chosen for most new data file formats including Orbiter Events, Orbiter Requests and Overflight Acknowledgements. These interfaces were updated several times over the course of the first phase of development with relatively minimal impact upon existing software components.
- CSV was implemented for a single input (the Ace Schedule) due to the "hands-on" spreadsheet driven nature of the data management. The Ace Schedule relatively inexpensive to implement and did not change significantly beyond the initial implementation.
- JSON was used for description of all data delivered to the web user interface. The lightweight structure enabled a high level of data throughput from the server to the GUI client and good responsiveness from the point of view of the web user interface.
- Legacy formats were maintained where the impact of change on providing systems was considered too costly, i.e. the LOPTG and Light Time files.

Lessons Learned

The extensibility of the XML is the primary driver for its broad use with external customers, though the widespread availability of supporting tools and libraries is an important factor. Ease of adaptation was certainly a factor in the first phase of development during early prototyping against rapidly changing SISs as well as some from later requirement changes. In one case late in phase in the phase a

set of additional data types were added with minimal no impact to existing file structure (the Orbital Events File) and inflicted minimal impact on development, mostly from the addition of new representational Java types.

Future Work

The MaROS system manages a range of XML schema defined information that is likely to provide value to other external users besides the immediate set of mission customers, and is also likely to see evolutionary changes through the course of the Mars Science Laboratory and other future missions involved in relay. For these reasons it may be worthwhile to migrate the adaptation of XML system interfaces to an XML registry built for that purpose.

Also it would be valuable to re-consider upgrading legacy file formats such as the LOPTG and Light Time File to the XML format, so that they could more readily evolve in the manner of the Orbiter Events file.

Final Words

The extensibility of system software interfaces will continue to be important in the face of the continuous evolutions of the Mars Network relay coordination information space.

REFERENCES

- [1] OrbitalHub, <http://orbitalhub.com/?cat=113>
- [2] Phoenix Mars Mission, <http://phoenix.lpl.arizona.edu/faq.php>
- [3] Daniel Allard, Charles Edwards, "Development of a Relay Performance Tool for the Mars Network," 2009 IEEE Aerospace Conference, March 7-14, 2009.
- [4] <http://www.w3.org/XML/>
- [5] <http://json.org>

BIOGRAPHY

Dan Allard has worked as a software engineer at the Jet Propulsion Laboratory for the past 19 years. He is the software architect and a lead developer of the Mars Operations Relay Service (MaROS) under development in support of ongoing and next-generation Mars Network missions. Prior to this he was the technical lead on the development of the Relay Data Engineering (RDE) system provided for accountability of relay performance over the Phoenix era. Other recent work includes the development of a message-based ground data system for the Mars Science Laboratory as well as research and development of ontology-based distributed communications in the space link and battlefield environments.

Acknowledgements

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

