A Reliable Service–Oriented Architecture for NASA's Mars Exploration Rover Mission

Ronald Mak
University Affiliated Research Center (UARC)
University of California at Santa Cruz
NASA Ames Research Center
Mail Stop 269–3
Moffett Field, CA 94035
650–604–0727
rmak@mail.arc.nasa.gov

Joan Walton
NASA Ames Research Center
Mail Stop 269–3
Moffett Field, CA 94035
650–604–2005
jdwalton@mail.arc.nasa.gov

Leslie Keely
NASA Ames Research Center
Mail Stop 269–3
Moffett Field, CA 94035
650–604–0570
leslie@ptolemy.arc.nasa.gov

Dennis Heher
SAIC
NASA Ames Research Center
Mail Stop 269–3
Moffett Field, CA 94035
650–604–4514
heher@ptolemy.arc.nasa.gov

Louise Chan
SAIC
NASA Ames Research Center
Mail Stop 269–3
Moffett Field, CA 94035
650–604–3377
lchan@mail.arc.nasa.gov

Abstract—The Collaborative Information Portal (CIP) was enterprise software developed jointly by the NASA Ames Research Center and the Jet Propulsion Laboratory (JPL) for NASA's highly successful Mars Exploration Rover (MER) mission. Both MER and CIP have performed far beyond their original expectations.[1, 2]

Mission managers and engineers ran CIP inside the mission control room at JPL, and the scientists ran CIP in their laboratories, homes, and offices. All the users connected securely over the Internet. Since the mission ran on Mars time, CIP displayed the current time in various Mars and Earth time zones, and it presented staffing and event schedules with Martian time scales. Users could send and receive broadcast messages, and they could view and download data and image files generated by the rovers' instruments.

CIP had a three–tiered, service–oriented architecture (SOA) based on industry standards, including J2EE and web services, and it integrated commercial off–the–shelf software. A user's interactions with the graphical interface of the CIP client application generated web services requests to the CIP middleware. The middleware accessed the back–end data repositories if necessary and returned results for these requests. The client application could make multiple service requests for a single user action and then present a composition of the results. This happened transparently, and many users did not even realize that they were connecting to a server. CIP performed well and was extremely reliable; it attained better than 99% uptime during the course of the mission.

In this paper, we present overviews of the MER mission and of CIP. We show how CIP helped to fulfill some of the mission needs and how people used it. We discuss the criteria for choosing its architecture, and we describe how the developers made the software so reliable. CIP's reliability did not come about by chance, but was the result of several key design decisions. We conclude with some of the important lessons we learned from developing, deploying, and supporting the software.

TABLE OF CONTENTS

1. MISSION OVERVIEW

The two rovers of the Mars Exploration Rover (MER) mission, Spirit and Opportunity, arrived at Mars in January 2004 after seven–month journeys from Earth. NASA scientists designed these twin robotic geologists to search for evidence of liquid water in the past on the

[1] 0-7803-8155-6/04/$17.00© 2005 IEEE
[2] IEEEAC paper #1075, Version 1, Updated September 13, 2004

Figure 1 – Mars Exploration Rover
(Photo courtesy of NASA and JPL.)

Martian surface. The rovers landed on opposite sides of the planet: Spirit inside Gusev Crater on January 3, and Opportunity on Meridiani Planum on January 24. Mission control was at NASA's Jet Propulsion Laboratory (JPL) in Pasadena, CA.

Each rover carried an impressive array of cameras and scientific instruments. See Figure 1. The cameras included a panoramic camera (pancam), a navigation camera (navcam), and front and rear hazard–avoidance cameras (hazcams). The scientific instruments deployed on a movable arm included the Miniature Thermal Emission Spectrometer (mini–TES) that identified minerals, the Mössbauer Spectrometer that identified iron–bearing minerals, the Alpha Particle X–Ray Spectrometer (APXS) that determined the composition of rocks, the Microscopic Imager that looked at fine–scale features, and the Rock Abrasion Tool (RAT) that ground away the outer surfaces of rocks to expose their interiors for examination. [1]

NASA designed the rovers for nominal 90–sol missions. A "sol" is a Martian day, which is nearly 40 minutes longer than an Earth day. Other than a few software and mechanical problems that the engineers were able to overcome, the rovers performed better and far longer than initial expectations and entered into extended missions. By early September 2004, each rover had operated over 200 sols and had explored more territory than originally planned.

The science generated by the rovers was even more impressive than their longevity. NASA's international Deep Space Network (DSN) antennas received the data and images sent by the rovers, which JPL then processed and stored in its data servers. After analyzing these data and images, NASA scientists concluded that liquid water did

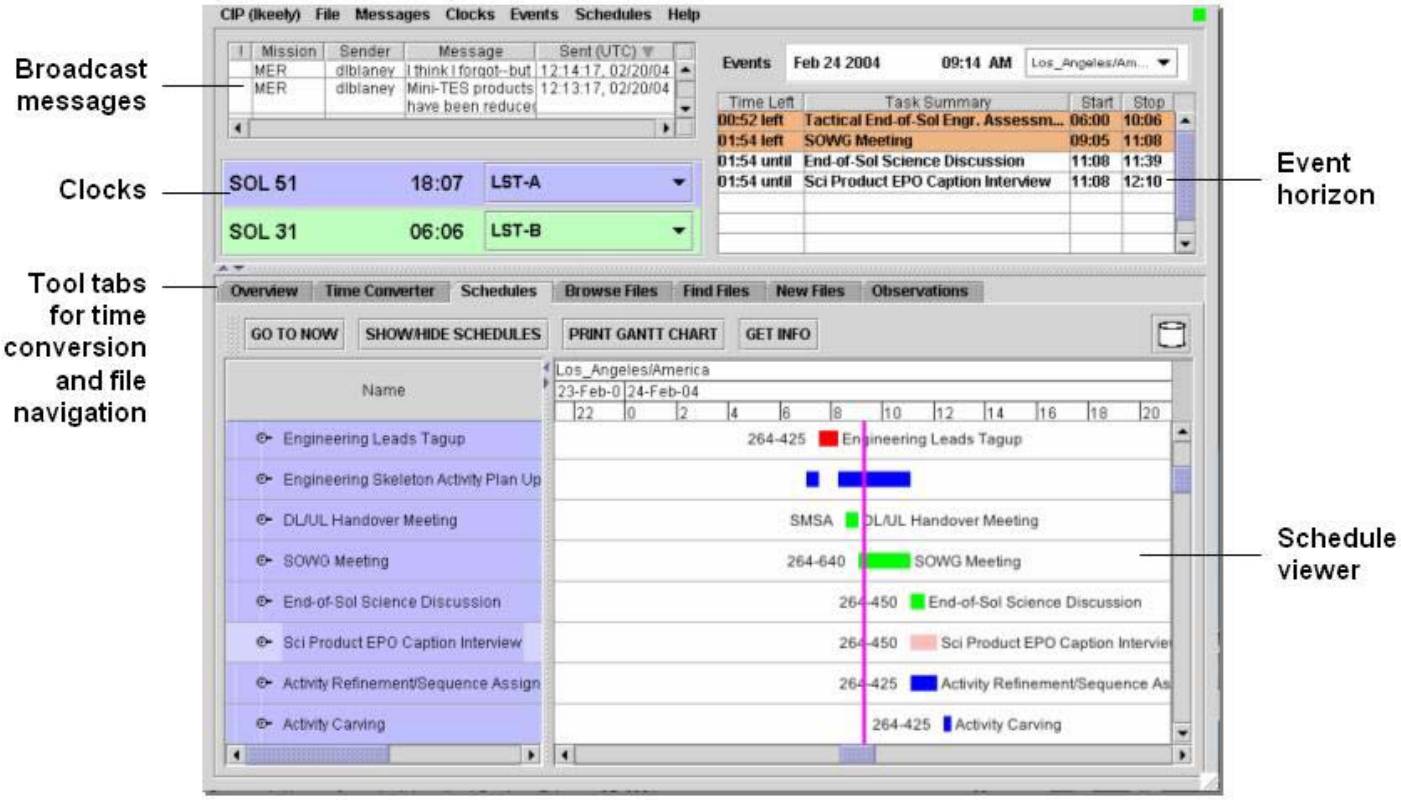
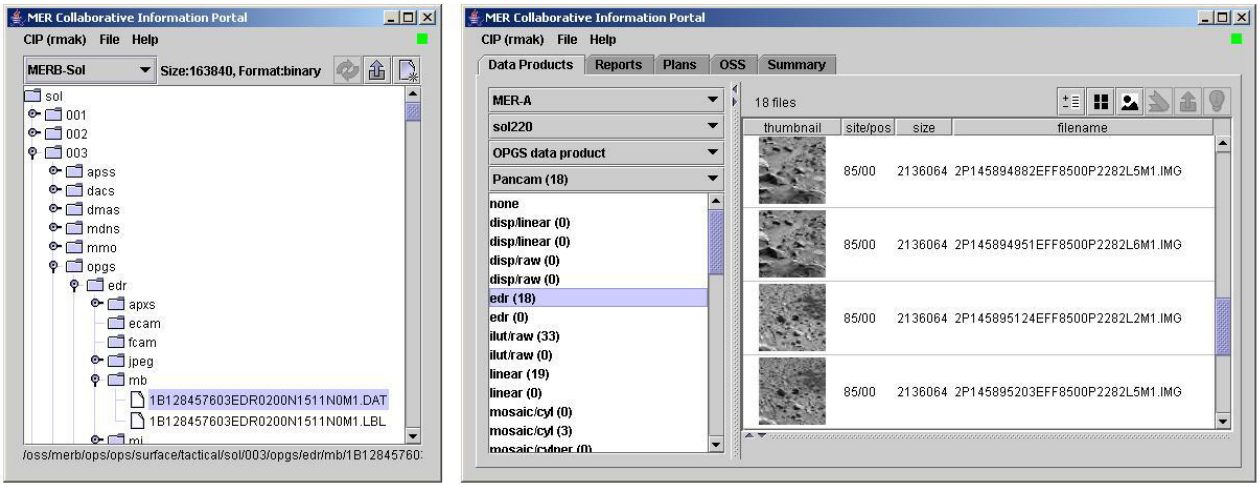


Figure 2 – The CIP Client Application



By Unix directories **By data products**

Figure 3 – Multiple Ways to Navigate the Data and Image Files

indeed exist on the surface of Mars in the distant past. [2, 3, 4, 5]

However, the MER mission was more than just the rovers. Two Earth–bound teams of NASA scientists, engineers, and mission managers, one team per rover, worked around the Mars clock to direct the rovers and analyze their results.

Tasks for Each Sol

Simply stated in order, each rover team's tasks for each sol were:

(1) Receive a downlink of data and images from the rover.
(2) Process and analyze these results.
(3) Plan the next sol's activities.
(4) Construct the rover command sequence.
(5) Send an uplink of the command sequence to the rover.

Mission Management

To coordinate all these activities on the ground, time management, data management, and personnel management were important.

Time management—During the initial nominal 90–sol mission and partway into the extended missions, mission personnel worked on Mars time. Therefore, meetings and other mission events scheduled on Mars time would drift nearly 40 minutes later relative to each Earth day. There were two Mars time zones, one per rover, and there were

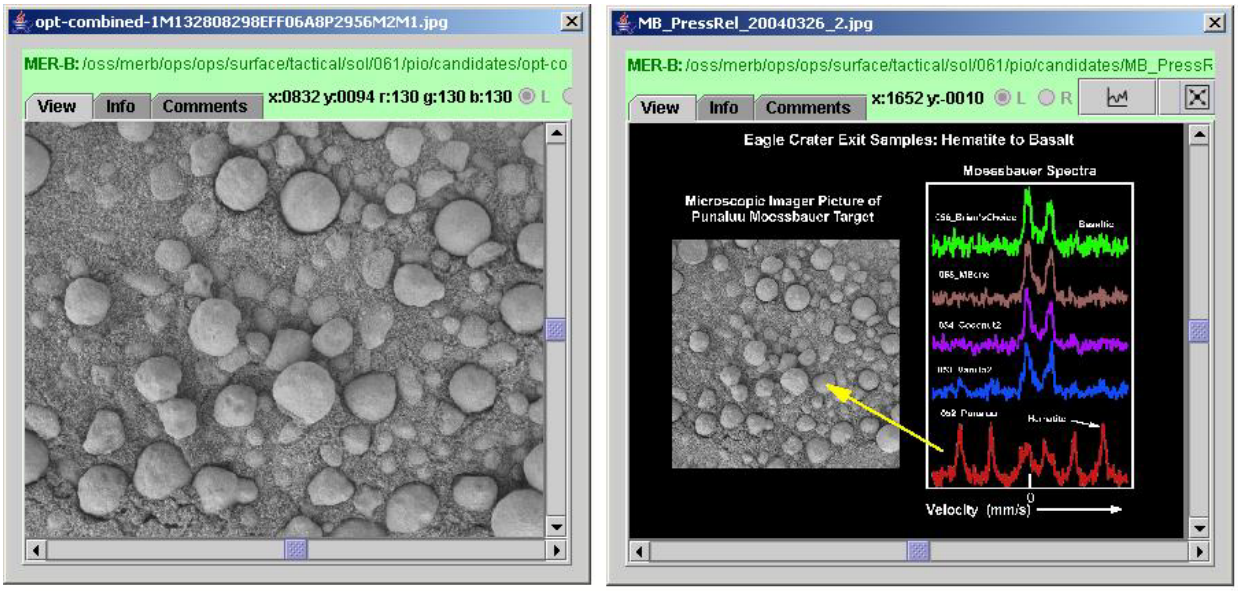


Figure 4 – Data Viewers
(Images courtesy of NASA and JPL.)

several important Earth time zones.

Knowing "What time is it now?" and "When is my next meeting?" was critical for many members of the rover teams.

Data management—During each sol, there was a handoff of data and images between the engineers of a rover team and its scientists. The engineers commanded the rover and then received and processed the results. The scientists analyzed the results and worked with the engineers to plan the next sol's activities for the rover. They had to correlate what they had planned for the rover and what actually happened.

JPL kept the processed data and images in data servers managed as a Unix file system. This repository contained both structured and unstructured heterogeneous data, and the scientists used specialized and general analysis tools. Some of the data had security restrictions that prevented access by foreign nationals.

Personnel management—The mission personnel on the two rover teams worked under various roles. Different roles had different information needs, which management needed to communicate. Some individuals varied their roles during different times of a sol, and others moved from rover team to another, perhaps assuming different roles for each rover.

Staff management was complex during the mission. Not only did each person need to know what he or she was supposed to be doing, but also with whom. It was necessary to know who else was working, where, and when.

2. THE COLLABORATIVE INFORMATION PORTAL

The NASA Ames Research Center and JPL jointly developed the Collaborative Information Portal (CIP) for the MER mission. Its crosscutting features and functionality served the mission managers and the mission scientists and engineers. Many found it to be useful during each sol throughout the mission. It assisted the rover teams with their daily tasks, and it helped provide time, data, schedules, and messages.

The Client Application

Figure 2 is a screen shot of the CIP client application, which ran under Microsoft Windows, MacOS X, Sun Solaris, and the Linux operating systems on PCs, laptops, workstations, and 50–inch touch–screen displays. The client application consolidated several useful tools into a single consistent and intuitive user interface.

Schedule Viewer—CIP assisted with time and personnel management by displaying staff and event schedules. People could use the Schedule Viewer Tool to see when events occurred, who was working when and where, and what roles they needed to fill that day. The schedules helped them adjust to Mars time, since regularly scheduled events drifted later from day to day relative to Earth time.

Event Horizon—Users could place scheduled events into the Event Horizon Tool. This tool then displayed a running countdown of the time left until the start of the event. The displayed events changed color to indicate nearness of the start times.

Data Navigation—CIP's Data Navigator Tools assisted with data management. NASA scientists and engineers

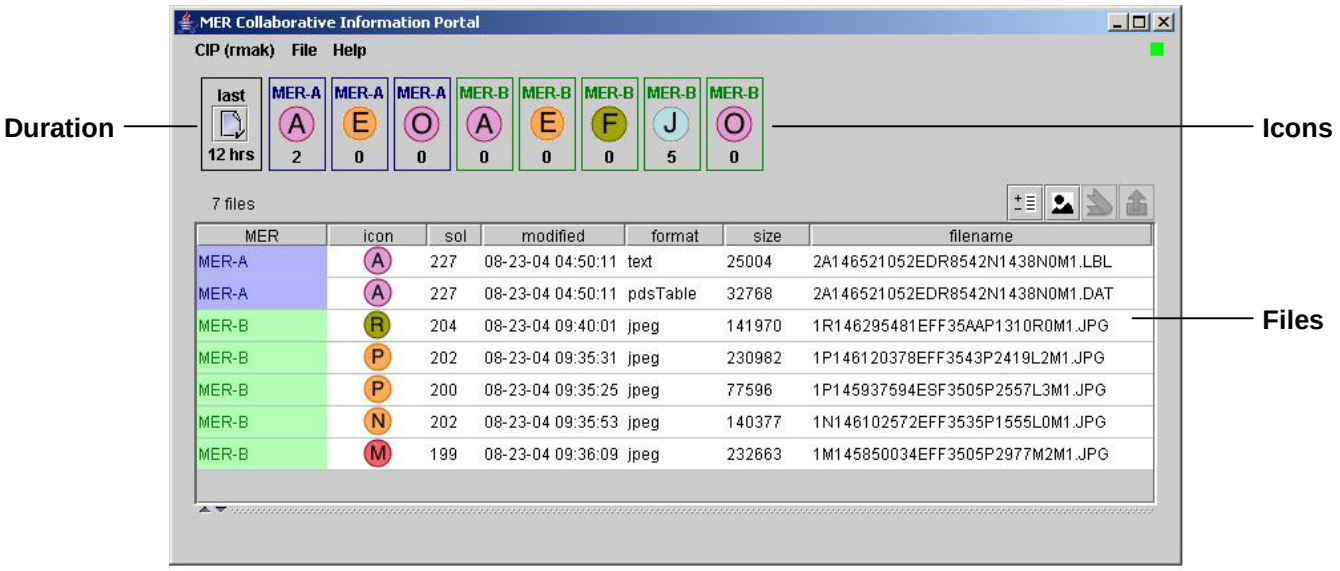


Figure 5 – New Files

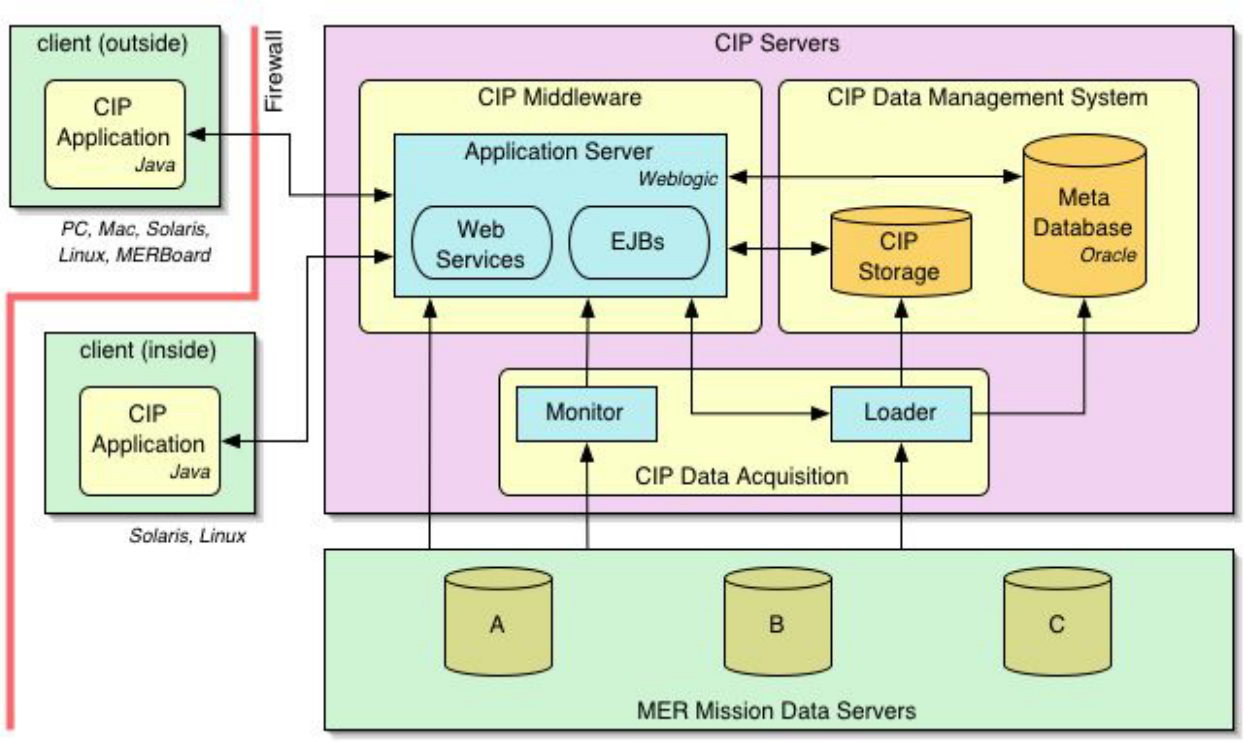


Figure 6 – A Three–Tiered Enterprise System with a Service–Oriented Architecture

could use the tools to access and display the data and images files residing in the JPL data servers. CIP transported this information securely over the Internet through the JPL mission firewalls.

CIP users had two ways to navigate the data and image files. They could go directly to the files via hierarchical Unix directories. Or, they could browse the files as "data products", which were organized hierarchically by the rover, sol, and instrument or camera that generated the original raw data. See Figure 3.

CIP's data repository tier generated metadata for the downloaded data and images stored in the mission data servers. Based on this metadata, the Data Navigator tools automatically classified and organize the data and images into the data product hierarchy. The tools used this classification to determine which viewer to use to display a file. See Figure 4. Users could also search for files based on the metadata fields.

Clocks—During the mission, it was not always sufficient to say something like, "It will happen at 14:30." Was that Mars time or Earth time, and in which time zone? The Clock Tool displayed clocks that showed Mars and Earth times in multiple time zones chosen by the user.

Time Conversion—The Time Converter Tool enabled users to convert times between various Earth and Mars time zones.

Broadcast Announcements—The Broadcast Announcements Tool enabled mission personnel to send messages to other CIP users. Typical messages were new data product announcements. Users could browse archived messages.

New Files—Users who were interested in specific data products could use the New Files Tool to register their interest in those products. See Figure 5. Each user could set the duration, and various icons represented his or her interests. The file information list displayed the products that became available during the selected duration. Whenever a new product became available in the mission data servers, the product's file information automatically appeared in the list.

3. A Service–Oriented Architecture

CIP was a three–tiered enterprise system. CIP users ran copies of the client application that used the Internet to access shared data. On the server side, software known as "middleware" handled simultaneous data requests from the client applications, and it securely accessed the "backend" data repositories, which included the mission data servers and the CIP Oracle databases containing metadata, schedules, and the message archive. [6] See Figure 6.

Given the mission requirements and the nature of the CIP client application, we designed CIP to have a service–

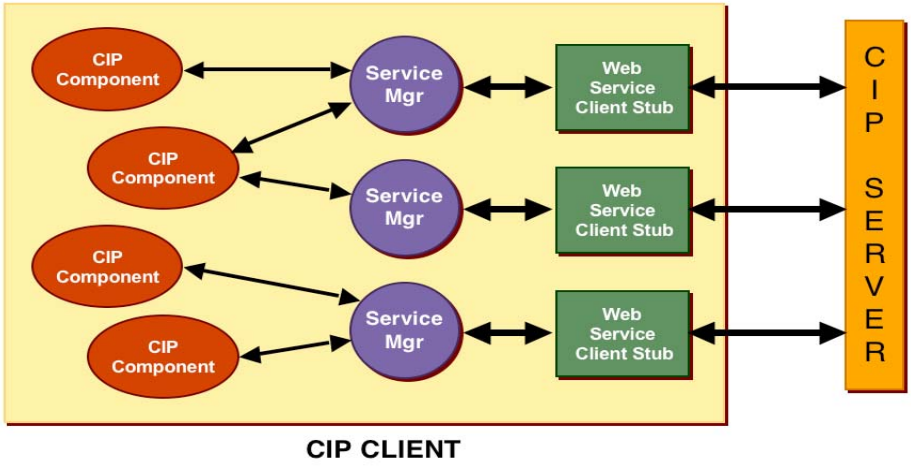


Figure 7 – The Component–Based Client Application Architecture with Web Services

oriented architecture (SOA). An SOA consists of a loosely coupled collection of services, where each service is a well–defined, self–contained function that is independent of other services. The services communicate with each other and with the client applications through a set of protocols known as web services. [7]

Why SOA

Users ran copies of the CIP client application to obtain information such as times and schedules and to access data and images. The client application satisfied the user by making requests to the CIP middleware for service, such as "Tell me what time it is on Mars in Spirit's time zone" or "Download the image file with this file path."

Web services—Using web services for communication between the client applications and the middleware offered several key advantages.

Web services communicate using a textual XML–based industry–standard protocol known as SOAP[3]. Service requests and responses are actually small XML documents passed between the client and server. CIP transmitted these documents securely using HTTPS.

Web services do not require persistent connections. A client connects to the middleware server, makes a request, receives the response, and disconnects. The CIP middleware kept track of an individual user's requests during a session with an access token. A client application received a unique token from the middleware whenever a user logged in, and the client passed this token back to the middleware as part of each subsequent request.

Web services are language independent, and the web services standard defines a finite set of XML–based data types. [8] Therefore, any programming language that has library routines to communicate via SOAP and to convert between native data types and the XML data types can use web services. We wrote the CIP client application in Java, and during the mission, the CIP middleware (also written in Java) responded to requests from the CIP client application and from Java and C++ applications developed by other projects.

Industry standards—The web services standard was but one example of our following industry standards to develop CIP. We had limited time and resources[4], and not the luxury to re–invent the wheel. Following industry standards allowed us to use commercial off–the–shelf (COTS) software wherever possible.

The Client Tier

We designed the CIP application to be a "thick client" desktop application, as opposed to a "thin client" application that ran within a web browser. A thick client makes better use of the user's local computer and provides better interactivity and responsiveness. We implemented the client application using the widely available Java platform and graphical user interface components from its Java Foundation Classes ("Swing").

Figure 7 shows our component–based approach for the client tier. Each client tool was a CIP Component object, and a Service Manager object supported one or more CIP Component objects. Each Service Manager object managed the connections to a particular remote middleware service

[3] SOAP originally stood for Simple Object Access Protocol. Now the acronym supposedly doesn't stand for anything, although some claim it ought to stand for Service–Oriented Architecture Protocol.

[4] For example, the initial version of the middleware had to be ready in four months to be available for an Operational Readiness Test at JPL. Three software engineers completed the middleware in a year and a half. There were twelve CIP developers overall, and the entire project lasted about three years.

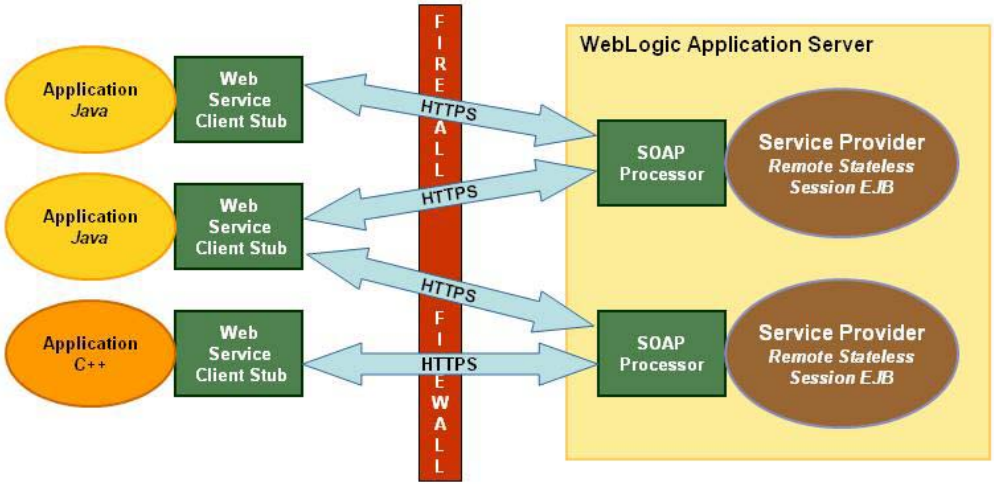


Figure 8 – Web Services and Service Provider Beans in the Middleware

by using a Web Services Client Stub. For example, the clock components used the Time Service Manager object, which managed the connections to the middleware's time service. The Web Services Client Stubs did the conversions between the clients' native data types and the XML data types.

Since it used web services, the client application connected to the middleware whenever a user action triggered a service request, and then promptly disconnected as soon as it got the response. The client automatically polled the middleware periodically via service requests for the current time and for any new broadcast messages.

The Middleware Tier

The CIP middleware communicated using web services with all the actively running copies of the CIP client application. It consisted of a Java-based commercial off-the-shelf application server and the Java components that we developed. We based our components on the Java 2 Enterprise Edition (J2EE) industry standard. [9] These components ("beans") were Enterprise JavaBeans (EJB) that operated at run time under the control of the WebLogic application server from BEA Systems, Inc. [10]

Middleware Services—The services provided by the CIP middleware to the client applications were:

- *User management service* to process user logins and logouts and to maintain user sessions.
- *Time service* to provide Mars and Earth times in various time zones.
- *Metadata query service* to fetch metadata from the CIP database.
- *Schedule query service* to fetch schedules from the CIP database.

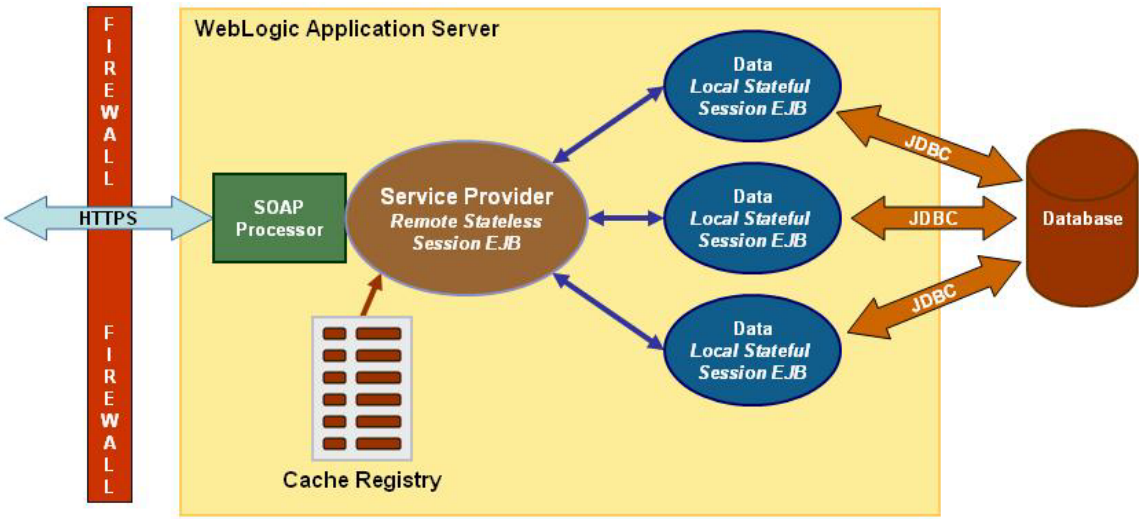


Figure 9 – Data Beans in the Memory Cache

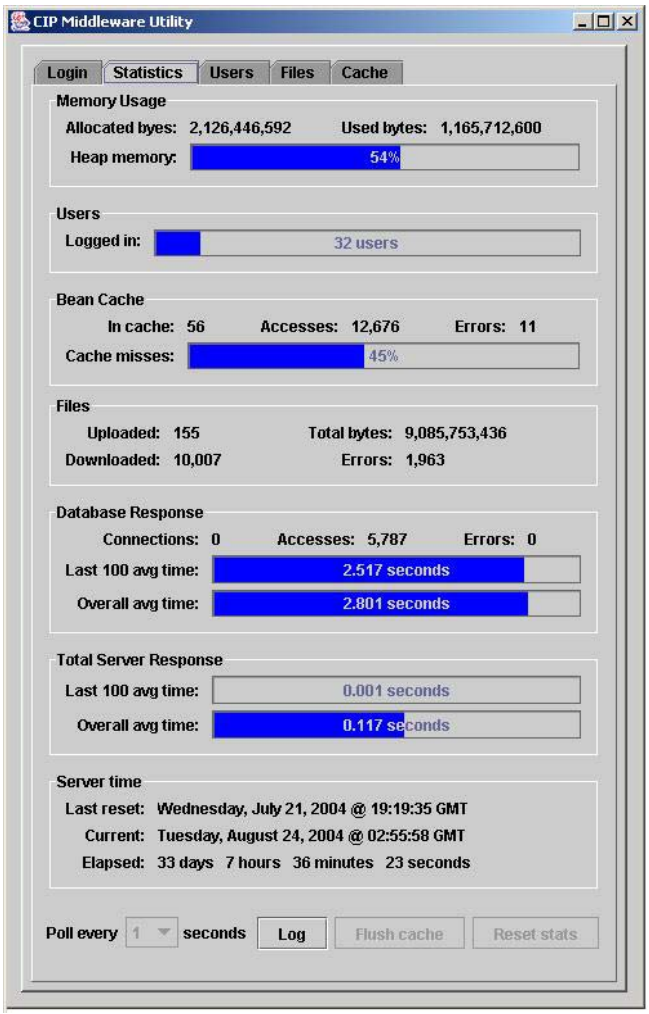


Figure 10 – The Middleware Utility

- *File streamer service* to download and upload files.
- *Message service* for asynchronous notifications, to broadcast and receive messages, and to fetch archived messages from the CIP database.

The middleware also provided basic security and a number of "ilities", including:[5]

- Accessibility
- Scalability
- Extensibility
- Reliability
- Adjustability
- Adaptability
- Maintainability

CIP security was a combination of user management and data encryption. The CIP middleware required each user to log in with a user name and password. Each user had pre-assigned privileges that allowed or disallowed access to certain data or images. Digital certificates from Verisign, Inc. enabled the CIP middleware to encrypt all data traffic between it and the client applications. [11]

A stateless session EJB represented each middleware service. Each such Service Provider bean had public methods and a SOAP Processor to enable the client applications to request services by invoking the methods remotely via web services. The SOAP Processor did the conversions between the Java data types and the XML data types. See Figure 8.

The application server maintained an instance pool of the stateless session beans, and it created or destroyed these instances in response to the request load. This made CIP scalable: as more requests arrived from the users, the application server automatically replicated more Service Providers to handle them.

Several of the middleware services created data beans, which were stateful session EJBs. These beans maintained state information, and the application server cached them in memory. For example, the metadata and schedule query services created data beans that used Java Database

```
2004-04-01 12:09:32,225 INFO : jdoe: Metadata.query()
2004-04-01 12:09:32,230 DEBUG: SELECT file_view.* FROM MER_B.file_view WHERE
((file_view.modified >= 1080806949117) AND (file_view.category = 'dataFile') AND
(file_view.filename LIKE '/%/merb/ops/ops/surface%/%/rcam/%' ESCAPE '\'))
2004-04-01 12:09:33,126 DEBUG: Records fetched: 0, skipped: 0
2004-04-01 13:50:06,816 INFO : mjane: Metadata.query()
2004-04-01 13:50:06,820 DEBUG: SELECT file_view.* FROM MER_B.file_view WHERE
((file_view.seqnum = 66) AND (file_view.category = 'dataProduct') AND
(file_view.owner = 'opgs') AND (file_view.type LIKE '%/jpeg/MER-B' ESCAPE '\'))
2004-04-01 13:50:10,073 DEBUG: Records fetched: 1, skipped: 0
2004-04-01 13:50:11,546 INFO : jdoe: Metadata.getObjectsByParent()
2004-04-01 13:50:11,550 DEBUG: SELECT * FROM MER_B.file_view WHERE (parent_pk =
16127) AND (category = 'dataFile')
2004-04-01 13:50:12,108 DEBUG: Records fetched: 5, skipped: 0
```

Figure 11 – Sample Middleware Log Entries

[5] Also known as Motherhood and Apple Pie.

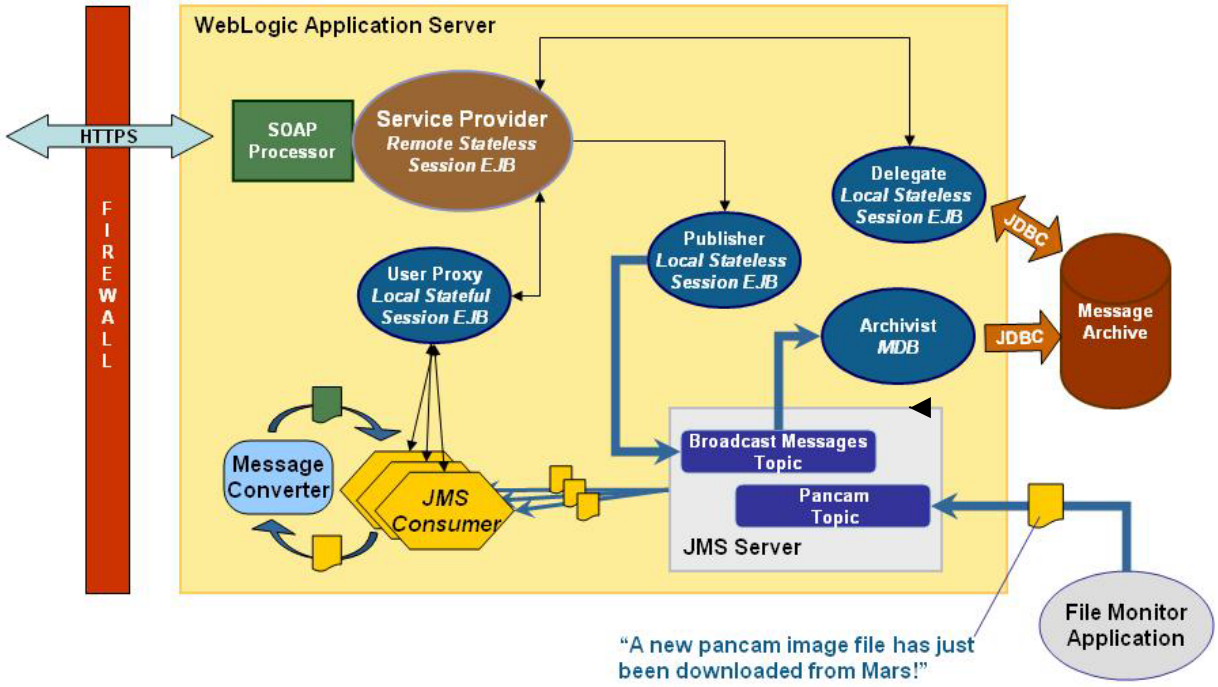


Figure 12 – New File Notification and Broadcast Messages

Connectivity (JDBC) calls to query the CIP databases. [12] Each data object kept a reference to the returned query results. This memory cache of data beans greatly improved the performance of repeated query service requests for the same data. If the data beans were already in the cache, the service did not need to make the much more time–consuming database queries. See Figure 9.

Web services made CIP very extensible. The "plug and play" services were easy to add, remove, or replace in the middleware; the application server handled these operations "hot" — *i.e.*, while continuing to run.

Monitoring and logging—We built a number of sensors into the middleware. We then developed a Middleware Utility program to monitor the middleware's status constantly, and to report graphically such statistics as memory usage and response times. Knowing the health of server at all times enabled the system operators to correct problems before they became serious. See Figure 10.

The middleware logged every activity, such as a user request. For each user request, the log entry contained a timestamp, the user's name, the name of the called method, details of the request, and key information about the results. See Figure 11. We did data mining in these logs to compute various statistics, such as how frequently users accessed certain types of schedules, or to deduce usage patterns, such as what methods users employed to locate data products. This enabled us to fine–tune the middleware's operations.

Asynchronous Messaging—CIP had two types of asynchronous messages:

- *Notification messages* that informed the CIP middleware or CIP users that new data and image files are available.
- *Broadcast messages* that CIP users could send to all the other users.

To implement asynchronous messaging, the CIP middleware used the Java Message Service (JMS), which was a part of the application server. [13]

JMS uses a *publish–subscribe* model. The middleware had a number of topics that represented different types of messages. A message consumer (such as a CIP client application) subscribed to one or more topics. Then whenever a message producer (a CIP client application or another CIP component) published (sent) a message to that topic, JMS delivered the message to all the message consumers who had subscribed in that topic. CIP messaging was asynchronous: message queuing and delivery occurred in parallel with all other operations. As mentioned earlier, each client application automatically polled the middleware periodically for its messages.

Figure 12 shows how the File Monitor in the data repository tier notified users who were interested in the availability of new panoramic camera images. As soon as the File Monitor detected a new panoramic camera image, it published a message to the Pancam Topic.

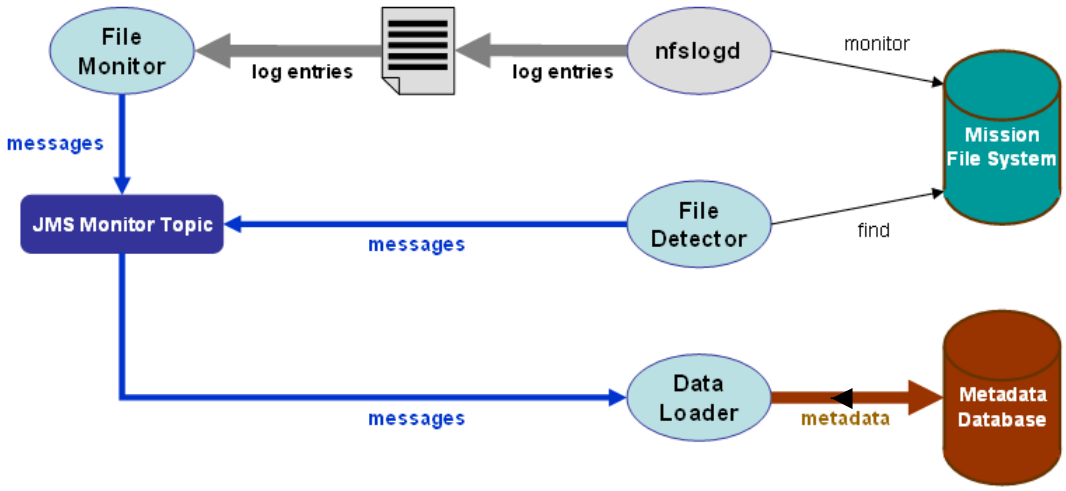


Figure 13 – The Data Repository Tier

CIP applications received their messages via web services. The middleware maintained a JMS Consumer object for each user to receive messages. A Message Converter reformatted each message so that the middleware can later return it as a web services response. Whenever a user's client application polled the middleware for messages via a service request, the User Proxy bean (a stateful session EJB) checked the user's JMS Consumer object, and it retrieved any delivered messages to return in response.

Figure 12 also shows how the Broadcast Messages topic, to which all CIP client applications subscribed, enabled a user to send messages to all the other users. Whenever a user sent a broadcast message via a service request, the Publisher bean (a stateless session EJB) published the message to the topic. The Message Archivist, a message–driven EJB that also subscribed to the topic, received and archived all broadcast messages into the Message Archive database. Each user received broadcast messages via polling.

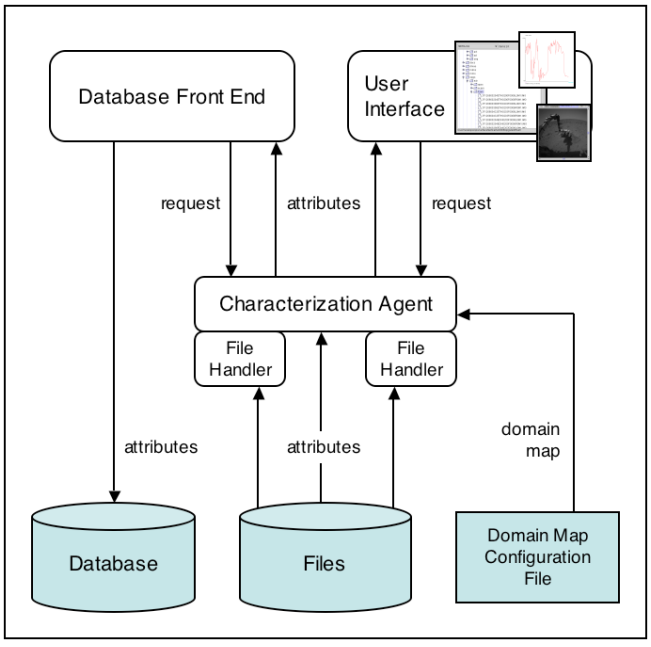


Figure 14 – Metadata Generation

If a user wanted to browse the archived messages, the client application made a service request, and the Delegate bean (a stateless session EJB) made the JDBC query into the Message Archive. The middleware returned all the archived messages in response.

The Data Repository Tier

As shown earlier in Figure 6, the data repository tier encompassed the CIP databases and the mission data servers.

The File Monitor constantly watched the logs generated by the Unix utility program *nfslogd*, which wrote a log entry every time it detected a file creation, read, move, or update. [14] See Figure 13. The utility used a configuration file that contained regular expressions representing the file paths that were relevant to CIP. It filtered out any files whose paths did not match any of the expressions.

Unlike the File Monitor, the File Detector used the Unix utility program *find* to "walk" the directory tree of the mission file system and find any relevant newly created or updated files. [15] It also used a configuration file that contained regular expressions for file paths. The File Detector walked the directories once during each run. It was a backup for the File Monitor whenever *nfslogd* was not running.

As soon as the File Monitor or the File Detector encountered a newly created or updated file that was relevant, it sent a message to the appropriate JMS topic, as

was shown in Figure 12. Data Loader subscribed to the topic.

Upon receiving a message, the Data Loader generated metadata for the file. Using regular expressions from its configuration file, the loader derived metadata field values from the file path itself. The loader also obtained some information from the Unix file system, and for some types of files, it read the file header to get more metadata field values. See Figure 14. Example metadata fields included the file name, the creation date and time, to which rover the file belonged, the rover location, which rover instrument generated the file data, during which sol, *etc*. The loader inserted, deleted, or updated the metadata in the database.

Data Modeling—The early phase of high-level architectural design includes defining how a system will provide its services. This involves data modeling, process modeling, interface design, and partitioning the system into components that the developers can build efficiently. The specifics of this partitioning are dependent on each system.

An important responsibility of the data repository tier was to maintain the data model that it shared with the client and middleware tiers. The CIP data model consisted of a logical model (how applications viewed the data) and a physical model (how the data was stored). The client applications worked with only the logical model. The middleware worked with both the logical and physical data models in order to convert each client request into the proper SQL statements and to return the results in a form suitable for the client.

4. RELIABILITY

CIP was extremely reliable. During the first seven months of the rovers' nominal and extended missions, its middleware stayed up over 99.9% of the time, and it ran nonstop for as long as 77 days at a time. [16] Several key factors contributed to this reliability.

We followed industry standards, and we used COTS software. For our production middleware server, we ran the WebLogic application server from BEA Systems on the Solaris operating system from Sun Microsystems [17]. In the data acquisition tier, we used the Oracle Enterprise Server 9i.

The application server further contributed to reliability by constantly monitoring the behavior of the EJBs, and it did automatic retries or error recovery whenever necessary.

On our development servers, we did extensive stress testing of the middleware before we deployed CIP and even during the mission. CIP usage patterns had sharp spikes, as many users became very active shortly after the rovers downloaded new data and images. Our stress testing showed us how the middleware would behave during such spikes and pointed out performance bottlenecks. We were able to adjust the system parameters accordingly to enable the middleware to handle heavy loads better. We developed a standalone, interactive utility to perform the stress testing by simulating any number of users performing various client functions, such as accessing schedules or downloading files. See Figure 15.

An important measurement of software reliability is how long it stays up and running. An application can unexpectedly crash, or system administrators can bring it down for maintenance. A common maintenance operation for CIP was to reconfigure a service to accommodate a change in an operational parameter, such as the time it took for a signal to travel from Earth to Mars (one-way light time).

Therefore, dynamic reconfiguration was a key feature that allowed CIP to stay up and running for long periods without scheduled server maintenance downtimes. CIP's middleware design and the application server allowed individual services to be "hot redeployable": we could add, remove, replace, or restart a service while the rest of the middleware (and CIP as a whole) continued to run. To reconfigure a service, a system administrator first edited the service's configuration file and then redeployed the service. When the service restarted, it read in its new configuration. Redeploying a service typically took only a few seconds, and often users did not notice any interruptions.

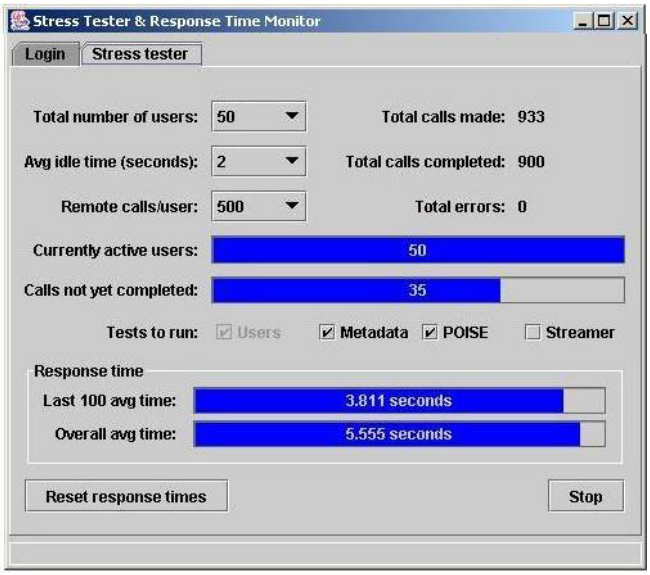


Figure 15 – The Middleware Stress Tester

5. LESSONS LEARNED

We learned several important lessons during the design, development, and deployment of CIP. [16]

By following industry standards and using proven COTS software for the infrastructure (operating system, application server, and database server), you can be reasonably assured that the underlying "plumbing" will work. Then the real challenges of enterprise development are not in the *coding*, but in the *integration* of the various components.

Make judicious use of vendor–supplied technical support. Whenever there were problems that we couldn't easily resolve ourselves, especially during the many crunch times, it was often useful to call for support and, in effect, add temporary but knowledgeable members to our development team. Nevertheless, it was important to test and evaluate their advice and suggestions before applying them.

Shared coding practices, source control, and system configuration management are critical for successful development. Make sure there is strong buy–in from all the developers from day one.

Ever-changing requirements before deployment and ever–changing operational parameters after deployment make it crucial to develop services that are plug–and–play, mutually independent, and dynamically reconfigurable.

Do lots and lots of *user* testing. Before the actual rovers landed on Mars, JPL ran a series of Operational Readiness Tests where teams of mission managers, engineers, and scientists worked together with simulated rovers. They tested software systems such as CIP under realistic conditions. We found and fixed many bugs during these tests and gained invaluable user feedback.

Do lots and lots of *stress* testing. If you don't know what the limits of your system are, your users will surely find out — at the worst possible times.

Having a good data modeling process is essential. While creating the data model, be sure to include all consumers and producers, *i.e.*, the stakeholders, to ensure reaching a consensus and meeting all usage requirements. Application developers often lack a deep understanding of data modeling and view databases as a simple lookup tables, thus missing opportunities to leverage fully the database capabilities.

At the beginning of the development of the CIP middleware, the data usage requirements were not yet firm, and the middleware didn't use the logical data model properly. The result was that we decided to cache data using stateful session EJBs instead of using entity EJBs. [18]. We subsequently spent much time dealing with threading and concurrency issues that entity beans would have taken care of automatically.[6]

Real–time server monitoring and logging helped the system operators keep track of what's going on and head off any potential problems. The Middleware Utility program and the Performance Monitoring tab of WebLogic's web browser–based console program together gave the operators a quick way to assure themselves that all was well.[7] The middleware logs provided ways to analyze usage patterns and fine–tune CIP's middleware.

If the enterprise system needs to respond to client requests in near real time, then make sure to capture this requirement during the early design stages, as it will greatly influence the system architecture.

We were concerned initially that web services would cause performance problems, since using XML documents for service requests and responses involved much data conversions, encryptions, and decryptions. CIP was able to achieve a data throughput rate of 100 MB per hour between a client application and the middleware, which was usually sufficient.

Respect for the tier boundaries of an *n*–tier enterprise system requires open lines of communication for collaboration during development. Communication of requirements must always be a two-way channel.

Developing enterprise software is inherently difficult. Don't make it any harder. Use common sense. Keep things simple.

6. CONCLUSION

At the time this paper was written (early September 2004), both rovers had lasted well over 200 sols, far beyond their original nominal 90–sol missions. This was testament to the excellent work and dedication of the mission managers, scientists, and engineers at JPL and its university and industry collaborators around the world. Fortunately, MER software systems such as CIP have also performed well throughout the original and the extended missions.

---

[6] Later on, we got better data usage requirements and understood the data model better. But by then, we decided we didn't have time to convert our code to use entity beans. In hindsight, we should have taken the hit to our development schedule and converted.

[7] Once CIP became operational shortly before the rovers landed on Mars, the system operators mostly were the CIP developers in a new role. We monitored CIP locally at JPL, and with proper VPN access, we could do it remotely from NASA Ames and from our homes.

Service–oriented architecture, or SOA, has been making the rounds as the latest industry buzzword. Not all of its concepts are new. What's new is the widespread acceptance by industry of the standards that SOA encompasses and the availability of much SOA infrastructure software and components.

CIP has been a highly visible SOA success story. It validated the architectural tenets of developing a collection of mutually independent services that respond to client requests and of using web services for communications between clients and the server.

However, SOA by itself does not guarantee a reliable system. CIP was reliable because of the conscious decisions that its designers and developers made, as described in this paper. Reliable software is good architecture plus good software engineering.

ACKNOWLEDGEMENTS

Funding for the development of CIP came from NASA's Computing, Information, and Communications Technology (CICT) Program and the Computing, Networking, and Information Systems (CNIS) Project.

Besides the authors of this paper, other CIP project members included Roy Britten (QSS), Sanjay Desai (SAIC), Matt D'Ortenzio (NASA), Glen Elliot (JPL), Robert Filman (RIACS), Kim Hubbard (NASA), Sandra Johan (NASA), Carson Little (Asani), Quit Nguyen (SAIC), Tarang Patel (SAIC), John Schreiner (NASA), Jeff Shapiro (QSS), Elias Sinderson (CSC), and Robert Wing (JPL).

This project would not have been possible without the support, assistance, and collaboration of JPL and the MER team.

REFERENCES

[1] Jet Propulsion Laboratory, NASA fact sheet, "Mars Exploration Rover", http://www.jpl.nasa.gov/news/fact_sheets/mars03rovers.pdf.

[2] NASA press release, March 2, 2004, "Opportunity Rover Finds Strong Evidence Meridiani Planum Was Wet", http://marsrovers.jpl.nasa.gov/newsroom/pressreleases/20040302a.html.

[3] NASA press release, March 5, 2004, "Volcanic Rock in Mars' Gusev Crater Hints at Past Water", http://marsrovers.jpl.nasa.gov/newsroom/pressreleases/20040305a.html.

[4] NASA press release, March 23, 2004, "Standing Body of Water Left Its Mark in Mars Rocks", http://marsrovers.jpl.nasa.gov/newsroom/pressreleases/20040323a.html.

[5] NASA press release, April 1, 2004, "Spirit Finds Multi-Layer Hints of Past Water at Mars' Gusev Site", http://marsrovers.jpl.nasa.gov/newsroom/pressreleases/20040401a.html.

[6] http://www.oracle.com/index.html

[7] Douglas K. Berry, *Web Services and Service–Oriented Architectures: The Savvy Manager's Guide*, San Francisco: Morgan–Kaufmann Publishers, 2003.

[8] http://www.w3.org/2002/ws/

[9] http://java.sun.com/j2ee/

[10] http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/server

[11] http://www.verisign.com/

[12] http://java.sun.com/products/jdbc/

[13] http://java.sun.com/products/jms/

[14] http://mirrors.ccs.neu.edu/cgi-bin/unixhelp/man-cgi?nfslogd+1

[15] http://mirrors.ccs.neu.edu/cgi-bin/unixhelp/man-cgi?find+1

[16] Ronald Mak, "Enterprise Development for Mars and other Alien Places", keynote address presented at BEA eWorld 2004 Conference, San Francisco, CA, May 26, 2004. Updated, expanded, and re–presented as a talk to the SDForum Software Architecture and Modeling SIG, Palo Alto, CA, August 11, 2004.

[17] http://wwws.sun.com/software/solaris/sparc/index.html

[18] Richard Monson–Haefel, *Enterprise JavaBeans, 3rd edition*, Sebastapol, CA: O'Reilly, 2001.

BIOGRAPHIES

Ronald Mak *worked on the CIP development team as the architect and lead developer of its middleware. After the rovers landed on Mars, he provided mission support both at NASA Ames and at JPL. He is a Project Scientist in the University Affiliated Research Center (UARC), which is a partnership between the University of California at Santa Cruz and the NASA Ames Research Center in Moffett Field, CA. Prior to working at NASA, Ron had over 15 years of industry experience developing enterprise software systems. He has taught graduate courses in computer science, and he is the author of books on numerical computing and on compiler writing. He has a B.S. in the mathematical sciences and an M.S. in computer science from Stanford University.*

Joan Walton *is the Group Lead of the Information Design Group in the Computational Sciences Division at the NASA Ames Research Center. In her decade at Ames, she led several multi-year projects to produce distributed information management systems, including the DARWIN system for the Ames wind tunnels and the Mars Exploration Rovers Collaborative Information Portal. In her role as a Project Manager of CIP, Joan led a team of twelve software developers. She was responsible for guiding the technical direction of the project, tracking the schedule, meeting milestones, and interacting with MER mission management to develop requirements and to meet JPL deployment criteria. She holds a Bachelor of Arts Degree in Physics from Swarthmore College and a Masters Degree in Medical Information Sciences from Stanford University.*

Leslie Keely *is a Computer Scientist at the NASA Ames Research Center. She designed and led the development of the CIP client applications, and she also does research in the area of data visualization. Leslie has a B.S. in Computer Science and a B.S. in Botany from the University of Oklahoma.*

Dennis Heher *worked on CIP as the lead developer of the data acquisition module. He is a Computer Scientist with Science Applications International Corporation (SAIC) at the NASA Ames Research Center. He has a B.A. in Computer and Information Sciences from the University of California at Santa Cruz and an M.S. in Computer Engineering from Santa Clara University.*

Louise Chan *is a Computer Scientist with Science Applications International Corporation (SAIC) at the NASA Ames Research Center. She designed and led the implementation of the data repository tier. She also does research in the areas of data modeling and management architecture. She holds a B.S. degree in Computer Science from the University of Maryland*