The Collaborative Information Portal and NASA's Mars Exploration Rover Mission

Ronald Mak
University Affiliated Research Center (UARC)
University of California at Santa Cruz
NASA Ames Research Center

Joan Walton
NASA Ames Research Center

The Collaborative Information Portal was enterprise software developed jointly by the NASA Ames Research Center and the Jet Propulsion Laboratory for NASA's Mars Exploration Rover mission. Mission managers, engineers, scientists, and researchers used this Internet application to view current staffing and event schedules, download data and image files generated by the rovers, receive broadcast messages, and get accurate times in various Mars and Earth time zones. This article describes the features, architecture, and implementation of this software, and concludes with lessons we learned from its deployment and a look towards future missions.

The Mars Exploration Rover (MER) mission was phenomenally successful. The rovers, Spirit and Opportunity, were robotic geologists that had impressive arrays of cameras and scientific instruments. [1] See Figure 1. The rovers sent data and images back to earth, which ground–based systems processed and stored in the mission file servers. After analyzing the data and images, NASA scientists concluded that liquid water did exist on the surface of Mars in the distant past. [2, 3]

The mission was not just about the rovers and data. On earth, people managed the mission, sent commands to the rovers, and analyzed the downloaded data and images. They used the Collaborative Information Portal (CIP) to help them perform their daily tasks, whether they were working inside mission control or the science areas at the Jet Propulsion Laboratory (JPL), or in their homes, schools, or offices. CIP had a three–tiered, service–oriented architecture that comprised a client tier, a middleware tier, and a data repository tier.

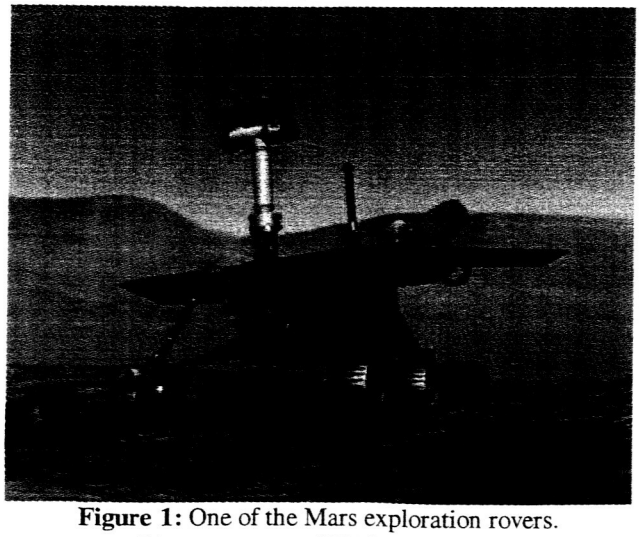

Figure 1: One of the Mars exploration rovers. (Photo courtesy of NASA and JPL.)

THE CIP CLIENT APPLICATION

Mission managers and engineers working inside mission control at JPL controlled and communicated with the rovers. Mission scientists and researchers working at JPL and elsewhere planned the rovers' operations and analyzed the downloaded data and images. There were two teams, one per rover, although some people moved from one team to the other. Most worked on Mars time, and each person could have different roles at different times of the day.

Figure 2 shows how the CIP client application assisted the rover teams with their daily tasks by consolidating several useful tools into a single consistent and intuitive user interface.

Schedule viewer

Mission personnel referred to the staff and event schedules that CIP displayed, especially if they worked on Mars time. The Schedule Viewer Tool let them know when events would occur, who was working when and where, and what roles they needed to fill that day. The schedules helped them adjust to Mars time—since a Martian day, or "sol", is about 40 minutes longer than an Earth day, 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, which then displayed a running countdown of the time

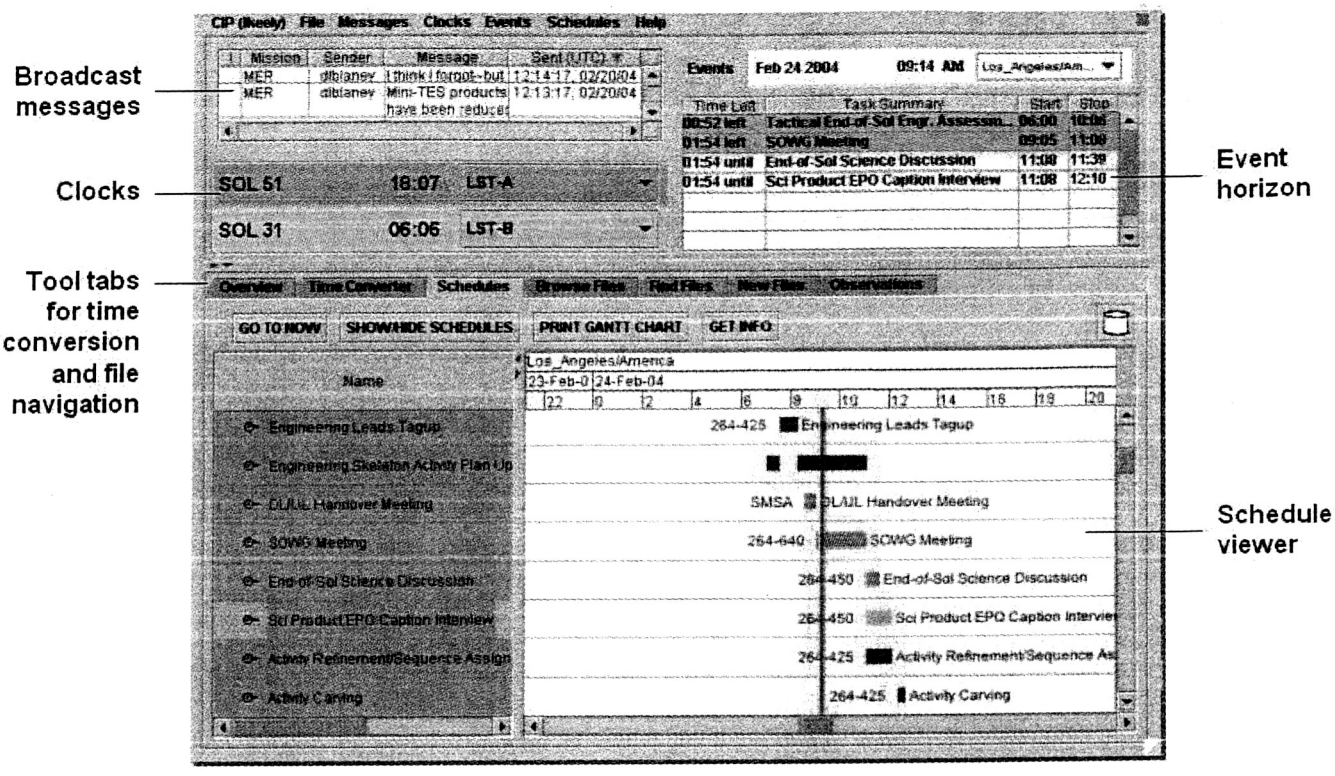

Figure 2: A screen shot of the CIP client application.

left until the start of the event. Event table rows changed colors to warn of impending start times.

Data navigation

NASA scientists and researchers used CIP's Data Navigator Tools to access the data and images stored in the mission file servers. CIP transported this information securely over the Internet. Users could navigate the data and images organized hierarchically in directories, or as "data products" organized by rover, sol, and instrument. See Figure 3.

CIP's data repository tier generated metadata for the data and images. Based on this metadata, the Data Navigator Tools automatically classified and organized the data and images into hierarchies. The tools used this classification to determine which viewer to use when displaying a file. Users could also do searches based on the metadata fields.

Clocks

"What time is it?" was important to know for everyone working on the MER mission. The mission ran on Mars time, and there were two Martian time zones, one per rover.

The CIP client application displayed clocks that showed Mars and Earth times in various time zones chosen by the user. The CIP middleware server provided accurate times to the client applications.

Broadcast announcements

The Broadcast Announcements Tool allowed CIP users to send messages to other users. Typical messages were announcements about new data products. Users could browse archived messages.

THE CIP ARCHITECTURE

Figure 4 shows CIP's service–oriented architecture (SOA). Partitioning the software into a client tier, a middleware tier, and a data repository tier balanced and distributed the computational resources, and enabled CIP to meet the goals of scalability, reliability, extensibility, and security.

The client tier

Users ran the Java-based CIP client application on their desktop or laptop computers. Therefore, there could be many copies of this application running simultaneously.

We designed the client application to be a "thick client" application that ran by itself, as opposed to a "thin client" application that ran within a web browser. A thick client application made better use of the user's computer and provided better interactivity and responsiveness. It contacted the middleware over the Internet only whenever it needed to request a service, such as when a user clicked a button. It polled the middleware periodically for the current time and

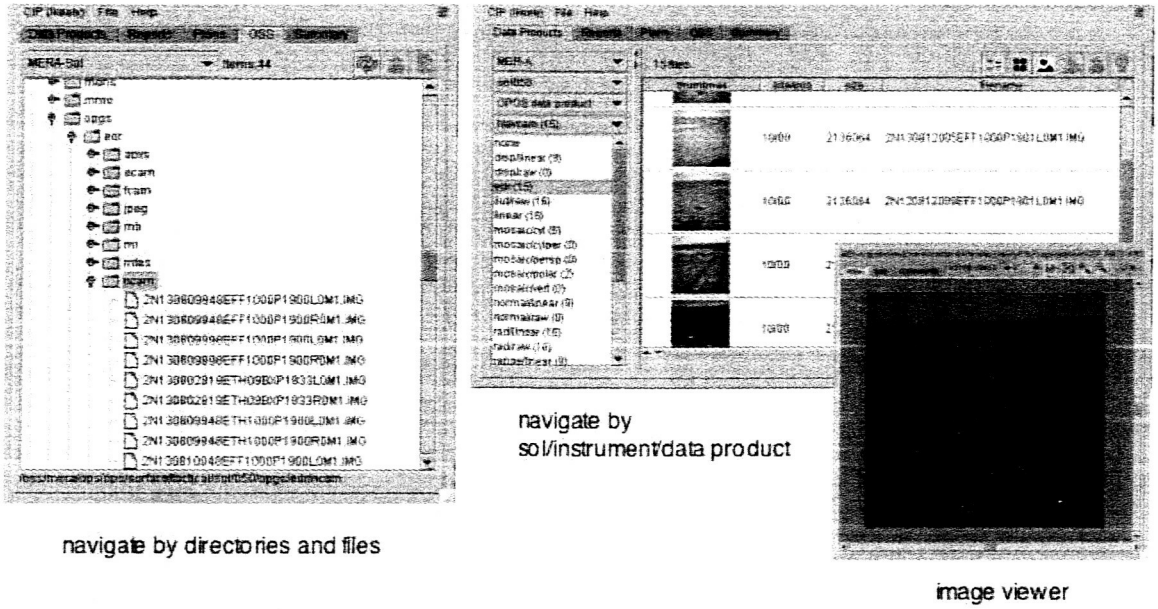

Figure 3: Multiple viewing modes of the Data Navigation Tools.
(Images courtesy of NASA and JPL.)

for any new broadcast messages. We implemented the client application using the widely available Java platform and graphical user interface components from its Java Foundation Classes ("Swing").

Figure 5 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. For example, the clock components used the Time Service Manager object, which managed the connections to the middleware's time service.

The middleware tier

The CIP middleware communicated over the Internet with all the actively running copies of the CIP client application. The middleware included a Java-based commercial off-the-shelf application server and the Java components that we developed. We designed it using SOA principles and two industry standards, Java 2 Enterprise Edition (J2EE) and web services. [4, 5] The J2EE-based components ("beans") that we wrote were Enterprise JavaBeans (EJB) that operated at run time under the control of the WebLogic application server from BEA Systems, Inc. [6]

The middleware provided services to the various tools of the client applications. Client applications requested services from the middleware, which returned a response to each request. The CIP middleware services included:

- *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 database.
- *Schedule query service* to fetch schedules from the database.
- *File streaming service* to download and upload files.
- *Message service* for asynchronous notification and broadcast messages.

Figure 6 shows how one or more Service Provider EJBs, which were stateless session beans, represented each service. Each bean had public methods that client applications invoked remotely over the Internet to request services. The application server maintained an instance pool of these stateless beans—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. Because these beans maintained state information, the application server cached them in memory. For example, the metadata and schedule query services created data beans that used Java Database Connectivity (JDBC) calls to query the database repositories. [7] Each data bean kept a reference to the returned query results. By taking advantage of the application server's memory cache of data objects, the query services greatly improved the performance of repeated 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.

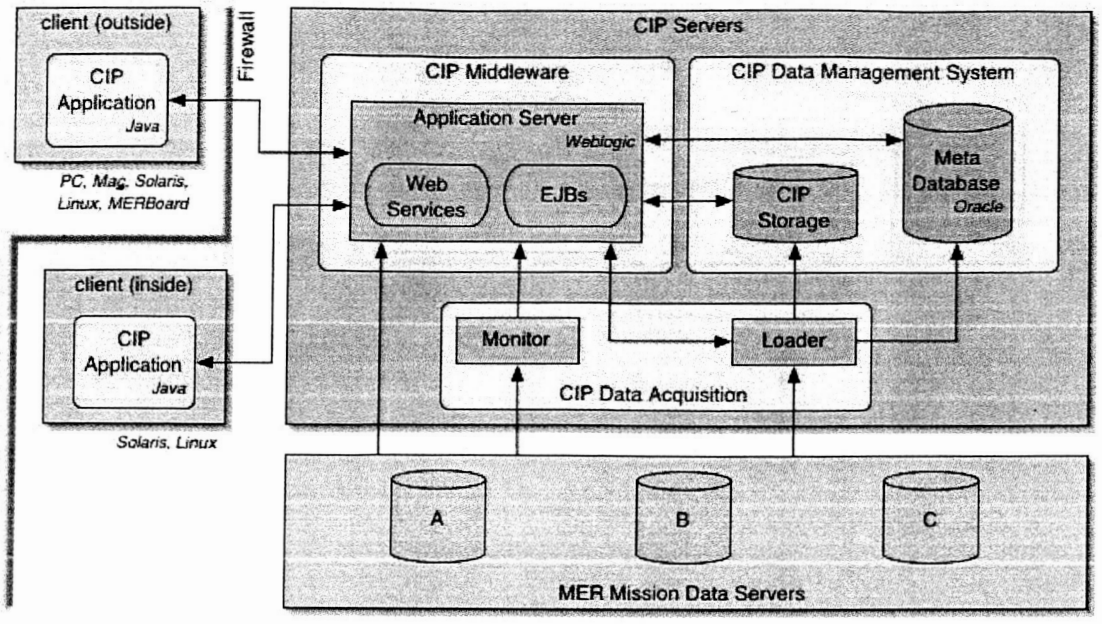

Figure 4: The CIP architecture.

A key middleware innovation was the use of web services. As shown in Figures 5 and 6, client applications used web services to communicate with the remote Service Provider EJBs. Each client Service Manager object had a web services client stub that was the proxy for the remote Service Provider bean.

Whenever a client application needed to request a service, it made a local call to a method with the same name in the client stub. The stub converted the call to a service request in the form of a text document encoded in the XML-based SOAP protocol, as defined by the web services standard. [5] The stub sent this document to the middleware server using the encrypted HTTPS protocol. The SOAP processor of the target Service Provider bean decrypted the request and invoked the appropriate public method of the bean. The generated response returned similarly across the Internet to the requesting client application as an encrypted SOAP document. The client stub decrypted the response and converted it to Java objects for the client application.

A client application always made local calls to the client stub and got local results back from the stub. Web services handled all the details of connecting to the remote Service Provider EJB, encryption and decryption, and sending requests and responses across the Internet.

Organizing the middleware as a collection of loosely coupled services made CIP very extensible, since it was very easy to "plug and play" new services and to replace or remove obsolete ones. The application server enhanced reliability by monitoring the operation of the services and automatically doing any necessary retries and error recoveries.

The middleware logged every activity, such as user requests. For each 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. We did data mining in these logs afterwards 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.

We developed a separate clientside 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 the server at all times enabled the system operators to correct problems before they became serious.

We put the CIP middleware under intensive stress testing before we deployed it into operation. This testing pointed out performance bottlenecks and helped ensure that CIP would be able to handle heavy loads. 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.

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 is to reconfigure an application to accommodate a change in an operational parameter.

A key feature that allowed CIP to stay up and running for long periods (over 77 days at a time before scheduled server

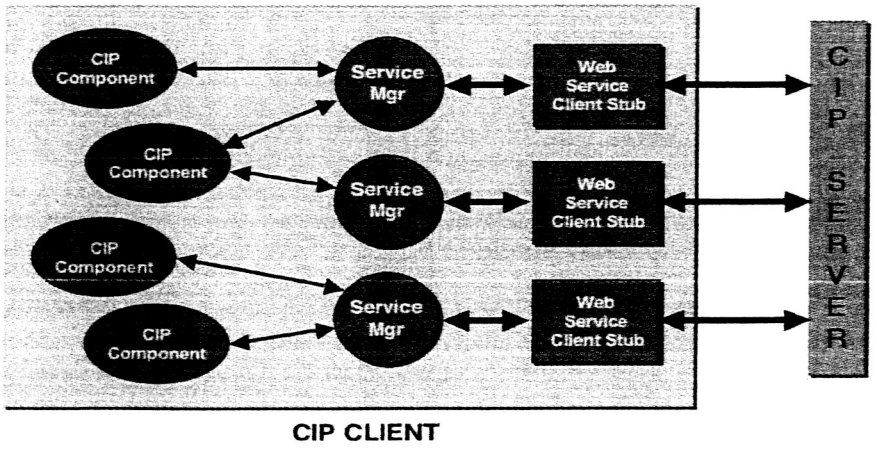

Figure 5: The component–based architecture of the CIP client application.

maintenance shutdowns) was dynamic reconfiguration. We designed the individual middleware services to be hot redeployable. In other words, we could 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 (for example, to change the one-way light time, which was the time it took a signal to travel from Earth to Mars) 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.

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 a 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 users' client applications. [8]

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 were available.
- *Broadcast messages* that CIP users could send to all the other users.

To implement asynchronous messages, the CIP middleware used the Java Message Service (JMS). [9] JMS employs 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

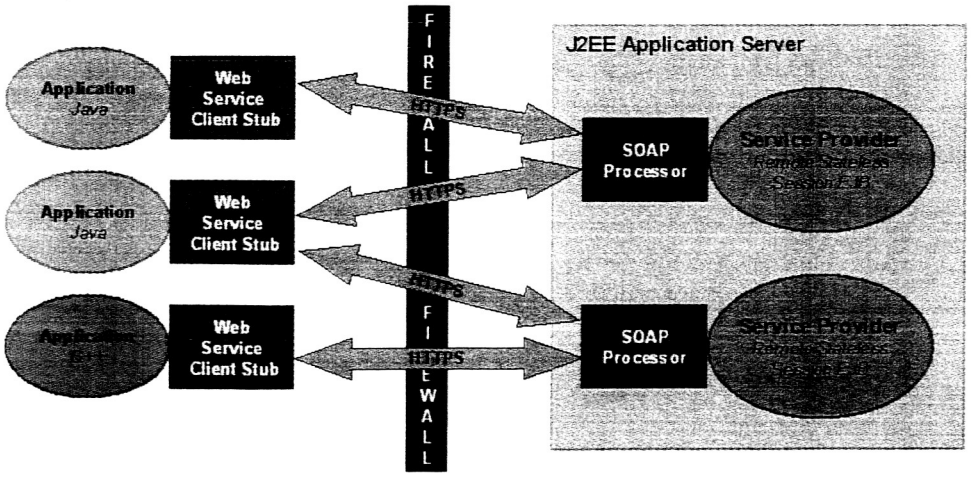

Figure 6: Web services and the Service Provider EJBs.

delivered the message to all the message consumers who subscribed to that topic. CIP messaging was asynchronous—message queuing and delivery occurred in parallel with all other operations.

For example, users who were interested in panoramic camera images subscribed to the pancam images topic. Whenever the data repository tier detected a new panoramic camera image, it published a message to the topic. The interested users received the message via web services the next time their client applications polled the middleware for messages.

There was also a broadcast messages topic, to which all CIP users subscribed and which enabled one user to send messages to all the other users. In the middleware, a message–driven EJB also subscribed to the topic. It received and archived all the broadcast messages in the database. CIP users could browse the archived messages.

The data repository tier

The data repository tier encompassed the Oracle databases [10] and the MER mission file system, which ran on separate servers. See Figure 7.

The File Monitor constantly watched the logs generated by the Unix utility program *nfslogd*, which wrote a log entry every time a file was created, read, moved, or updated. [11] The utility had a configuration file that contained regular expressions for 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. [12] It also used a configuration file that contained regular expressions of 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 Data Loader. The Data Loader generated metadata for that file. Using regular expressions from a 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. 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 or updated the metadata in the database.

LESSONS LEARNED

We learned several important lessons during the design, development, and deployment of CIP. [13]

Following industry standards and using proven commercial software for the infrastructure reasonably assured us that the underlying "plumbing" would work. The real challenges of enterprise development were not in the *coding*, but in the *integration* of the various components.

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

Both user and stress testing were critical. JPL ran a series of Operational Readiness Tests where teams of mission managers, engineers, and scientists tested software systems such as CIP under realistic conditions. We found and fixed many bugs during these tests and gained invaluable user feedback. Our stress testing allowed us to discover the limits

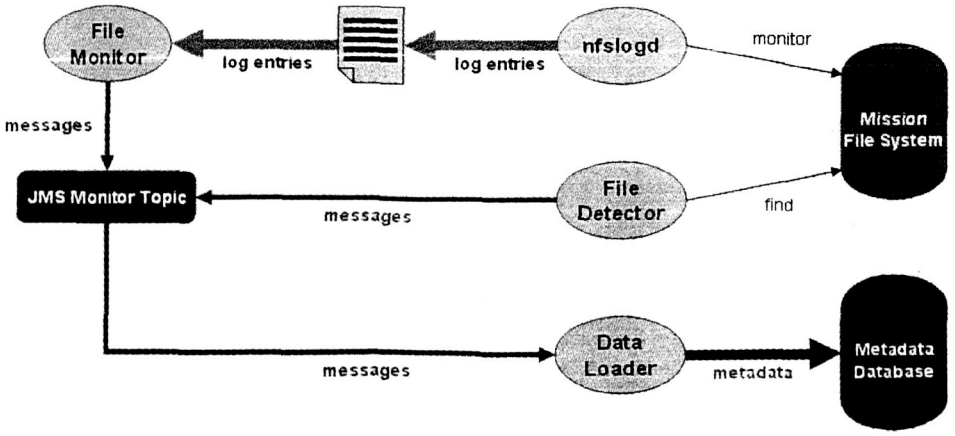

Figure 7: The data repository tier.

of our software before the users did.

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 logs provided ways to analyze usage patterns and fine-tune CIP's middleware.

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.

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

CIP AND FUTURE MISSIONS

The success of CIP on the MER mission encourages us to make improvements to the software and to add new features for user collaboration and inter-application data exchange. Ames and JPL are currently designing a new version of CIP for the upcoming Phoenix Mars 2007 Scout mission. [14]

Efforts are underway to define common enterprise software that multiple NASA centers can use for their future missions. SOA and other industry standards will play important roles in creating a multi-center, multi-mission infrastructure that will allow reusable software components from different missions to communicate with each other and to exchange data.

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 article, other CIP project members included Roy Britten (QSS), Louise Chan (SAIC), Sanjay Desai (SAIC), Matt D'Ortenzio (NASA), Glen Elliot (JPL), Robert Filman (RIACS), Dennis Heher (SAIC), Kim Hubbard (NASA), Sandra Johan (NASA), Leslie Keely (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, April 1, 2004, "Spirit Finds Multi-Layer Hints of Past Water at Mars' Gusev Site", http://marsrovers.jpl.nasa.gov/newsroom/pressreleases/20040401a.html.

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

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

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

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

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

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

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

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

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

[13] 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.

[14] http://mars.jpl.nasa.gov/missions/future/phoenix.html

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. Contact him at ron@apropos-logic.com or at rmak@mail.arc.nasa.gov.

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. Contact her at jdwalton@mail.arc.nasa.gov.