Middleware and Web Services for the Collaborative Information Portal of NASA's Mars Exploration Rovers Mission

Elias Sinderson[1], Vish Magapu[2], and Ronald Mak[3]

[1] Computer Sciences Corporation
NASA Ames Research Center, M/S 269-3
Moffett Field, California 94035
esinderson@mail.arc.nasa.gov
[2] Science Applications International Corporation
NASA Ames Research Center, M/S 269-3
Moffett Field, California 94035
vmagapu@mail.arc.nasa.gov
[3] Research Institute for Advanced Computer Science
NASA Ames Research Center, M/S 269-3
Moffett Field, California 94035
rmak@mail.arc.nasa.gov

Abstract. We describe the design and deployment of the middleware for the Collaborative Information Portal (CIP), a mission critical J2EE application developed for NASA's 2003 Mars Exploration Rover mission. CIP enabled mission personnel to access data and images sent back from Mars, staff and event schedules, broadcast messages and clocks displaying various Earth and Mars time zones. We developed the CIP middleware in less than two years time using cutting-edge technologies, including EJBs, servlets, JDBC, JNDI and JMS. The middleware was designed and implemented as a collection of independent, hot-deployable web services, providing secure access to back end file systems and databases. This service-oriented approach to developing an integrated system is an example of cutting edge middleware design. Throughout the middleware we enabled crosscutting capabilities such as runtime service configuration, security, logging and remote monitoring. This paper presents our approach to mitigating the challenges we faced, concluding with a short review of the lessons we learned from this project and noting some of the things we would do differently and why.

1 Introduction

The 2003 Mars Exploration Rover (MER) mission was the latest in a series of science missions to the planet Mars. The primary goal was to search for irrefutable evidence of liquid water existing on the surface in the Martian past. The mission was composed of two redundant (dual launch, dual lander) mobile science platforms, or rovers, outfitted with a variety of instruments. NASA

H.-A. Jacobsen (Ed.): Middleware 2004, LNCS 3231, pp. 1–17, 2004.

Fig. 1. Artists rendition of a MER mobile science platform (rover)

launched two spacecraft carrying these rovers in June 2003, and in January 2004 they landed safely on Mars. After landing the rovers at two separate locations, they proceeded to take numerous images of their surroundings and collect data from the Martian rocks and soil. Intermittently throughout each Martian day, or sol, the rovers transmitted their data back to Earth where it was collaboratively analyzed and activities were planned for the next sol. This analysis and planning cycle had to be completed in time for the next set of command sequences to be uplinked as the sun is rising on Mars.

The MER Collaborative Information Portal (CIP) was conceived of to provide mission personnel a sense of situational awareness, allowing the MER scientists, engineers and managers to accomplish their daily tasks by facilitating access to various mission data products, schedules, broadcast announcements, and the like. This paper focuses on the middleware and web services that were developed for CIP. The rest of this section presents a brief overview of the day-to-day mission operations, and describes previous work at Ames that both led to and influenced the CIP project. Sec. 2 outlines the overall approach to systems development that was employed. Sec. 3 provides an initial, high-level overview of the client application functionality and back end data stores before proceeding on to the details of the CIP middleware. Sec. 4 looks at the crosscutting functionality that was implemented across all of the CIP middleware services. We conclude the paper by identifying several of the lessons learned on the project.

1.1 MER Operations

The MER operations (Ops) environment is stressful, to say the least. The turnaround time from downlink to uplink is approximately 14 hours, barely enough time for the teams of scientists and engineers to analyze the recent data, plan the next sols activities, and then generate and verify the command sequences before uplink radiation. To make matters worse, a sol is approximately forty minutes longer than a day is on Earth, so each day's activities, meetings, etc. occur a little later than they did the previous day. What this means, practically, is that the time of telemetry downlink, and all associated activities occurring afterwards, moved forward by approximately an eight-hour shift every other week, making it extremely difficult for mission personnel to maintain a sense of situational awareness about the mission. Operations proceeded in this fashion 24x7 for the duration of each rover's life. The cumulative stress on Ops personnel makes it exceedingly difficult for them to remember when activities are to occur or where data products are located.

The total amount of data associated with the MER mission is quite large. The 1996 Mars Pathfinder mission, for reference, produced 2.3 gigabits of data in over 17,000 data products in the nineteen days the rover was active. With two rovers in operation during MER, several GB of data was being produced every day, including Experimental Data Records (EDRs) from the rovers themselves, Reduced Data Records (RDRs) as a result of EDR analysis, and the planning documents and command sequences produced within Ops. This volume of data not only needs to be archived, but also catalogued and tagged with metadata for ease of indexing, location and retrieval. Furthermore, the interdependencies of the various teams make the challenges of cooperative and collaborative work more difficult than they would be otherwise. For example, if modifications are made to a given resource then others who depend on that information must be notified as quickly as possible. Subsequent modifications to data products ripple outwards through other resources and people as the awareness of the changes spreads.

1.2 Background

In the mid 1990s, the DARWIN project focused on the problem of providing real-time access to aeronautical data captured during wind tunnel testing over the internet [6]. DARWIN provided secure access to wind tunnel data, allowing researchers at remote facilities to collaboratively analyze experimental results using a standard web browser [11]. This system was implemented using Common Gateway Interface (CGI) scripting and Java applet technologies to access underlying data repositories [12]. One of the aspects of this work that the CIP project inherited was the D3 metadatabase schema [10].

The initial prototypes of the CIP tools were developed as applets and delivered within a commercial portal package, however we quickly identified several drawbacks to this approach. The reloading of applets on hyperlink traversal and web page refresh was problematic from a user experience perspective as well as

making it difficult to maintain state in the tools. Further, testing of this environment, with multiple browsers, browser versions and Java plugin versions was extremely resource intensive. Lastly, applets did not offer the sort of rich and flexible client UI that was desired for CIP. For the above reasons, it was decided that the CIP client should be developed as a full-fledged Java application.

2 CIP Development

As mentioned previously, CIP provides an enhanced sense of situational awareness to mission personnel. Of primary importance was to allow secure local and remote access to mission data products and scheduling information. The system was specified to be available 24x7, with better than a 99% availability over the course of the mission, while handling a load of 150 or more concurrent users. Furthermore, addressing the needs of a diverse user community including scientists, engineers, and managers posed a challenge in and of itself.

The basic principles we followed during the CIP development were to (a) apply applicable industry standards, (b) utilize existing COTS software and (c) use available commercial development tools. Essentially, we wanted to avoid reinventing the wheel, as we were working against hard deadlines (i.e. launch dates) that quite simply could not be pushed back.

CIP needed to be platform-independent, so we naturally chose to use Java throughout the system, with J2EE [9] being used for the middleware and SOAP-based web services [2] providing the client interface to the middleware EJBs [3]. A major benefit in the tools we selected was that the IDE we used was tightly integrated with the application server, making it possible to deploy and test our code from within the IDE. Further, both the client and server web service interfaces were automatically generated simply by invoking build utilities provided with the application server. This saved us much effort in the long run in not having to code these by hand whenever we had to change an interface.

The decision to pursue a Service Oriented Architecture (SOA) built on SOAP-based web services was, at the time, felt to contain some risk, especially given the relative immaturity of the associated technologies and the nature of the system under development. However, the benefits offered in terms of firewall negotiation, stateless connections and security made our early adoption of these technologies a reasonable gambit. This approach has since been validated in several ways, not the least of which by other MER applications utilizing the CIP web services for authentication, authorization and data access in ways that we had not originally anticipated.

3 CIP Architecture

In this section we provide an expose of the CIP architecture. The first two subsections outline the client application functionality and the back end data stores before presenting the CIP middleware and web services in detail. We conclude

Fig. 2. Screenshot of the CIP client application

this section by examining how crosscutting functionality such as security and logging was implemented across all of the web services.

3.1 Client Application

CIP employs a three-tier client-server model in which the client is a desktop application and the server is a J2EE compliant middleware platform consisting of a number of web services. The server maintains user preferences and profile data [1] in order to provide a consistent interface and behavior from one session to the next. The suite of tools within CIP facilitated increased communication, improved situational awareness and provided a greater exposure to mission information than would otherwise have been possible.

The CIP client application is divided into different information panels as shown in Fig. 2, with broadcast messages and clocks displayed in the upper left, the event horizon in the upper right, and lower tabbed pane for doing time conversions, displaying event and personnel schedules, browsing data files, searching for files, monitoring the arrival of new files, and displaying observations. In this screen shot, part of the tactical timeline schedule is visible with the time scale showing both Earth and Mars times. The auxiliary window is displaying an image of the martian horizon that has been selected in the data file browser.

In essence, mission management, operations and science team members used CIP to retrieve the various data products, staffing and event schedules and main-

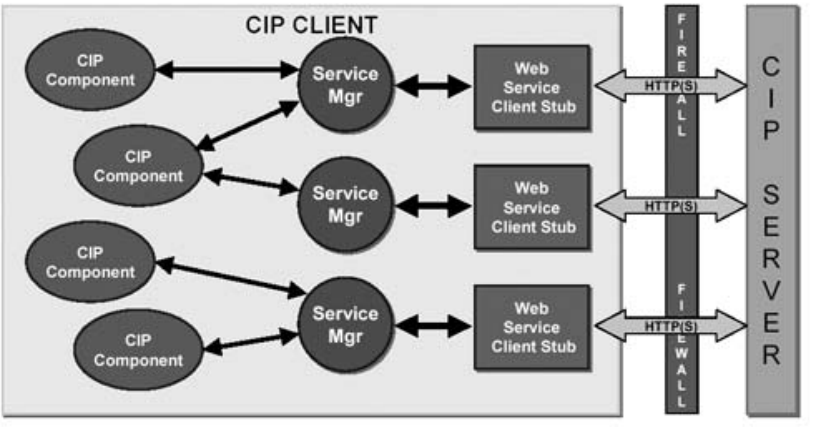

Fig. 3. CIP client architecture showing the relationship between the tools, their service managers and the CIP middleware

tain a sense of what was going on in the mission. In addition to integrating the various mission data repositories into a common interface, CIP provided custom tools for the following: Time keeping and conversion between Mars Local Solar Time for each rover (LST-A and LST-B) and time zones on Earth. Tracking of strategic and tactical events with an event horizon and schedule viewer that is linked to information about science teams and data products. Access to mission broadcast announcements and lastly, the ability to manage subscriptions to different types of notifications about new resources.

The CIP Client was implemented as a layered architecture, with each tool panel relying on one or more of the middleware web services. Each service had a *service manager* which ran in its own thread and served to manage the connection to the service. Each web service connection utilized a service adapter which translated between data formats and mediated the exception handling for the service manager. A number of different design patterns [4] were used throughout the CIP client architecture, including singletons, factories, proxies, decorators and iterators, among others.

CIP Client Tools. The CIP clocks maintain the correct time to within minute accuracy (due to network latency) and convert between different time zones on Earth as well as two Mars time zones LST-A and LST-B. An Event Horizon panel is also provided to display upcoming, ongoing and recent events, including communication windows, shifts and handovers, activity planning and approval meetings, communications windows, press conferences, etc.

The schedule viewer is an interactive tactical event timeline. The time zones supported by the scheduler are the same as those supported by the CIP Clocks. Tactical events are displayed in the schedule viewer, which can also display staffing information for mission personnel, press conferences, etc. The schedule

is hyperlinked such that when a user clicks on an event, information about the associated teams, processes, and data products is displayed in a pop-up window.

The Browse Files and Find Files tabs provide tools for searching and browsing mission data products and resources by name, type, date, etc. In addition, the New Files tab allows users to subscribe to notification about active data products that they are interested in. The mission resources of interest include generated summary reports, rover health information, and strategic and activity plan summaries among other things. The data products reside on mission specific file systems, while annotations and other resource metadata are maintained in a metadata database. These repositories are described in more detail in the next subsection.

The broadcast announcements panel allows CIP users to communicate about mission announcements, news, events, and plans. Past messages are archived to a database and accessible through a message browser utility located in a pull down menu. Using the message browser, users can delete messages from their view or make important announcements 'sticky' so that they don't scroll off the screen as new messages appear.

3.2 Data Repositories

A critical aspect of the CIP application is being notified when new resources become available or when existing resources are modified. This allows the metadata cache to be kept up to date, in turn ensuring that mission personnel are using the most recent versions of resources to make important decisions. Maintaining a sufficient level of situational, group, and workspace awareness in a distributed system required a reliable event notification infrastructure.

For simplicity and responsiveness under extreme user loads, MER utilized several flat NFS file systems to store mission data products, hence the amount of control over the repository was limited when compared to other, more robust content management systems. To provide CIP with information about active resources, the Solaris NFS logging daemon, nfslogd, was used to trap file operations. Information is written to a log file that is monitored by a data acquisition daemon running on the CIP server.

When a given resource is active, a notification is sent to the CIP middleware in the form of a JMS message. The notifications contain enough information to update the CIP metadata repository appropriately. Subsequently, a notification was sent to any concerned clients about the update. The CIP metadata is maintained in an Oracle database on a dedicated server. This database also maintains schemas for scheduling information and archival storage of broadcast messages.

3.3 Middleware

CIP is a three-tier enterprise system based on the Java language. The client tier consists of a desktop application developed with Java Swing components, while the J2EE middleware tier consists of Enterprise Java Beans (EJB) and servlets. Web services acted as the interface between the client applications and

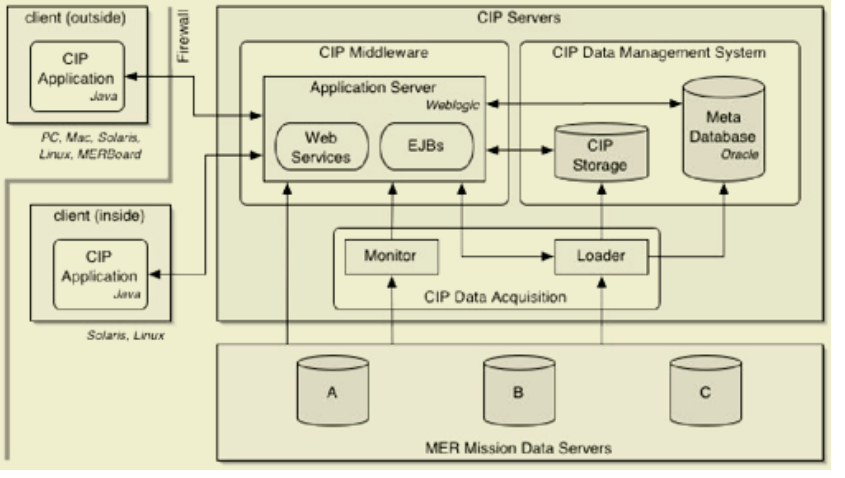

Fig. 4. CIP middleware architecture

the EJBs. The data repository tier included the mission file system, the Oracle databases, and data monitor and data loader utilities. The middleware EJBs used JDBC calls to access the databases. The Java Message Service (JMS) provides asynchronous messaging among the three tiers.

The CIP middleware architecture and its' relationship with the client application and back end data repositories is shown in Fig. 4. In the final deployed configuration, the CIP middleware, data acquisition and data management system ran on separate machines, with the MER mission data servers were NFS mounted as shown.

The CIP middleware consisted of a number of independent services, forming an example of a service oriented architecture. Each service had a public interface implemented by a stateless session EJB, acting as a session facade [7], that was the service provider. Each service provider had a SOAP processor to handle incoming web services requests and their responses. CIP used the HTTPS protocol to ensure secure communications between the client applications and the middleware. A general representation of the CIP web service architecture is presented in Fig. 5.

The following web services were exposed to the client application: User management, Time, Database query, File streaming, and JMS messaging. These services were chosen based on logical groupings of functionality present in the system requirements. The primary goals of the middleware included scalability, reliability, extensibility, and security. For the most part, these goals were fulfilled by our use of J2EE and web services over HTTPS. Our other goal for the middleware was that it remained invisible to the end client, giving each user the illusion of having direct and exclusive access to the data. It was evident that we had achieved this latter goal when users asked, "What server?", not realizing that they were connected to one in the first place.

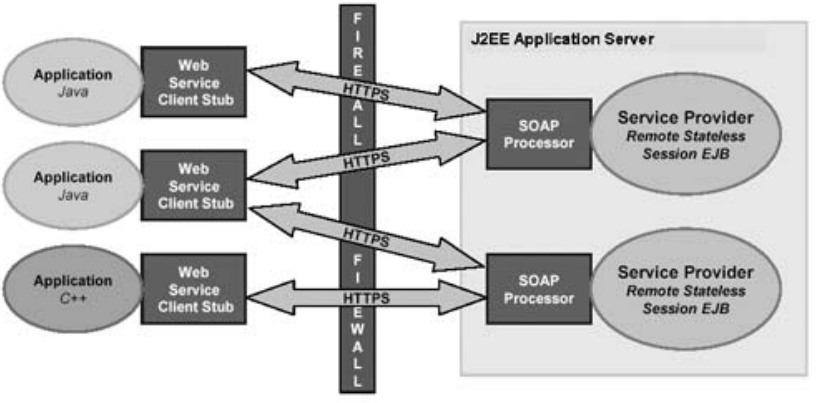

Fig. 5. CIP Web Services

User Management Service. The CIP User Management Service provides authentication and authorization services for CIP clients and is designed to be an independent vertical service that can be used for either purpose across several MER subsystems apart from CIP. The requirements for other MER subsystems to use the CIP User Management Service was inclusion of the user management service client stubs by way of a jar file.

Time Service. It was important for everyone working on the MER mission to be able to answer the question "What time is it?". The mission ran on Mars time, and since a Sol is about 40 minutes different than an Earth day, regularly scheduled events drifted later from day to day relative to Earth time. Moreover, two Martian time zones, one per rover were used extensively throughout the mission.

The CIP application displayed clocks that showed Mars and Earth times in various time zones chosen by the user. The CIP middleware supplied accurate times, which went over the Internet to the CIP applications. Due to network latencies, the times displayed by the CIP applications could be a few seconds off; we only guaranteed accuracy to the minute. The middleware obtained accurate Earth time via a Network Time Protocol server and computed the Mars times from the Earth time.

Query Service. The middleware query service accessed schedules and metadata stored in the Oracle databases of the data repository tier. CIP client applications displayed staff and event schedules, which they requested from the middleware. The applications also allowed users to browse data and images sent by the rovers. This information was categorized according to metadata fields, such as which rover, which sol, or which instrument. Users could also do searches based on metadata field values. The metadata was generated and loaded into the database by the data loader utility in the data repository tier.

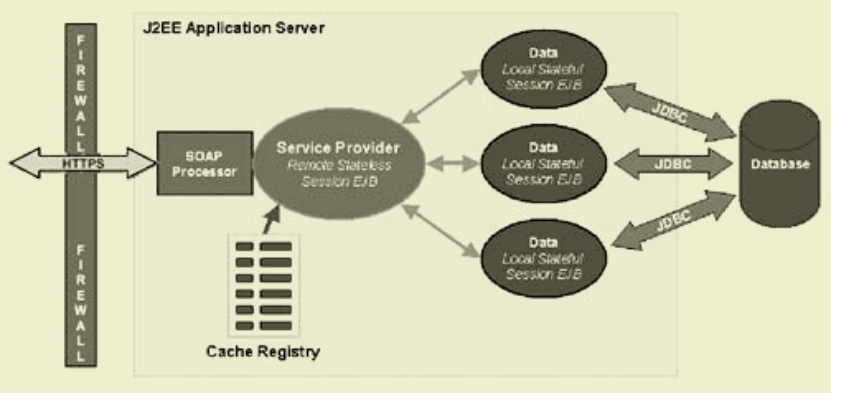

Fig. 6. The CIP query service

As shown in Fig. 6, the query service used stateful session EJBs to perform the queries via JDBC calls. Each session EJB stored the query results. By keeping track of these stateful session EJBs in a registry – a hash table keyed by the query strings – the middleware was able to cache query results in memory. As of this writing, the middleware cache has achieved a cumulative hit rate of 65% over the longest recorded server uptime.

Streamer Service. A user of the CIP client application often requested downloads of data and images. The middleware responded to such a request by accessing the desired file from the mission file servers in the data repository tier.

The middleware transmitted each file in small blocks, with the web services interface converting each block of binary data into ASCII – two characters per original byte – using base 64 notation. The block was then encrypted and sent over HTTPS to the requesting client application. The web services client stub decrypted the block and converted it back to binary. The application received all the blocks sequentially, and so it was simple for it to reassemble the data or image.

Despite all the data conversions and the encryption and decryption, files downloaded at the rate of around 100 MB per hour. Most of the data or image files were smaller than a few megabytes, although some data products were quite large – panoramic mosaics, for example, that were composed of many smaller images being stitched together.

Message Service. The CIP Message Service provided JMS messaging capabilities between the CIP client and the middleware. The clearest example of this functionality is the broadcast announcements tool, which allowed managers to send messages out to the mission. Many of the CIP components, however, were able to take advantage of this framework so that the information they displayed was the freshest possible.

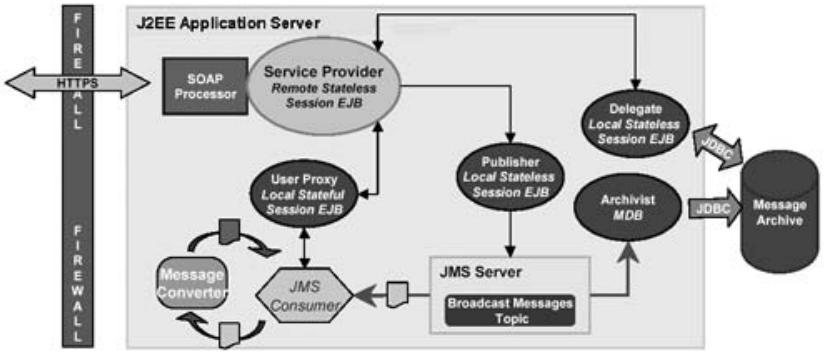

Fig. 7. CIP Message Service

The Schedule Viewer subscribed to messages about modifications to the mission schedules. When a new schedule is uploaded to the server, or an existing schedule is modified, a message is published to 'schedules' topic. When the client receives the message, the display is updated to reflect that new scheduling information is available. Similarly, the New Files tab allows users to subscribe to active resources by their type and when they were last modified. When resources are modified or added to one of the mission repositories, an active resource notification message is published to a 'resources' topic and if a client has subscribed to notifications of that type, it is notified.

On the data acquisition side of things, the database loader subscribes to a 'monitor' topic, which carries messages about NFS operations on mission data products. This information is used to keep the metadata about the mission resources accurate and up to date. Thus, when the loader receives a notification that new data products are available it populates the metadata database with information about them.

The decoupling of event producers and event consumers within the overall system is perhaps one of the greatest benefits of using messaging oriented architectures. This approach redeemed itself when it became necessary to migrate the data acquisition components to another machine in order to lighten the load on the primary machine that clients connected to. Without the decoupling of system components provided by JMS, this would have required a fair amount of effort.

One of the difficult challenges in designing the CIP Message Service was in allowing multiple clients to connect using the same user ID. This situation would arise, for example, whenever a user left an instance of the CIP client application running in their office or at home and then subsequently attempted to log in at a terminal in one of the science areas. This posed a problem because the application used durable subscriptions for several of the system services. As the JMS specification states, durable subscriptions require a unique client ID to be used consistently from one session to the next and only a single client can use this client ID at any given time [5]. The solution was to use a client proxy on

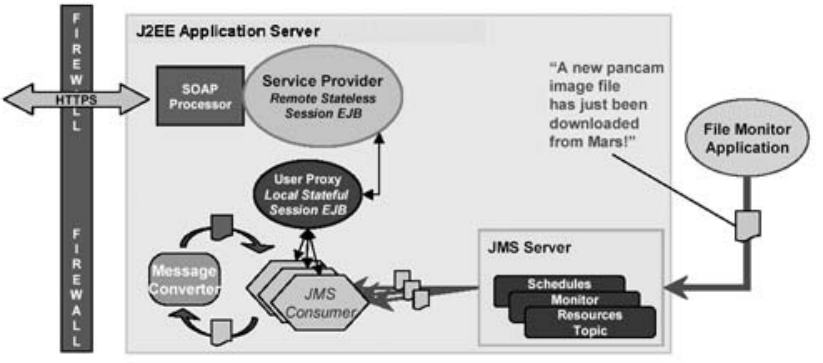

Fig. 8. CIP file notification process

the server to connect to the JMS server in the J2EE container of the application server. In this way, when a user logged in more than once, we could detect that this was the case and manage the JMS subscriptions appropriately.

4 Crosscutting Functionality

In this section we take a look at the crosscutting functionality that was implemented across all of the CIP middleware services. At the time CIP was being developed, the current approaches to SOA did not address overarching issues such as logging, system monitoring, runtime configuration and security. Whereas more recent activities focus on specifying additional application tiers for these activities, [8], our solutions to these issues were considerably less formal, although sufficient for their intended purpose. The monitoring and logging methods described below all rely upon the use of shared data structures, referenced by the application server classpath setting.

4.1 Security and Authorization

According to Sun Micro Systems J2EE specification there are two ways to control access to application resources using the J2EE architecture, declarative authorization and programmatic authorization. In the declarative authorization the application permission model is defined in a deployment descriptor. In the programmatic authorization the container makes access control decisions before dispatching method calls to a component. The CIP User Management Service uses programmatic authorization. In this architecture, the J2EE container serves as an authorization boundary between the components it hosts and their callers

The CIP authentication process is as follows. The CIP client sends a login request by sending username and password on HTTPS (or HTTP if the client is inside the Ops firewall). The User Management Service authenticates the credentials by interacting with the J2EE container. If the authentication is successful, the service generates an access token and returns the token to the client. If the request fails the service returns a null token.

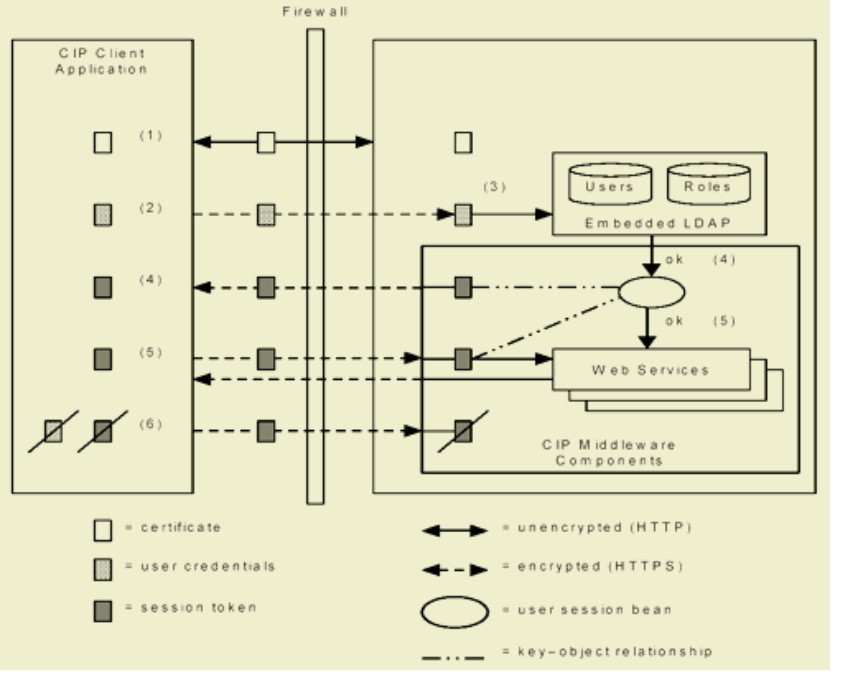

Fig. 9. CIP Authentication and authorization scheme in which (1) SSL certificate is exchanged, (2) user credentials are submitted over a secure channel, (3) user credentials are verified against the embedded LDAP store, (4) a user session EJB is created along with a corresponding session token, which is passed back to the client, (5) subsequent requests for middleware web services include the user session token, which is used by the middleware to look up the user session EJB and authorize the request against the associated roles and privileges, and (6) the expiration of the session token and removal of the user session EJB upon client logout. The actions in (6) may also occur if a CIP client is idle for too long and their session expires

The CIP user authorization procedure is accomplished through the definition and enforcement of user roles. The authorization process for a CIP client is as follows. When a CIP client requests a CIP middleware service, it supplies the login token obtained in a prior authentication request. The token is validated and then user is checked against the roles for granting permission to access the service requested. If the check is successful, the service request will be allowed to proceed, otherwise an authorization error will be returned to the client. The complete life cycle of CIP authentication and authorization described above is depicted in Fig. 9.

4.2 Logging and System Monitoring

Extensive run time logging and real time monitoring enhanced the middleware's reliability. The middleware web services logged every user request and it's subsequent processing within the system. As shown in Fig. 10, the log entry contains

```
2004-04-01 12:09:32,225 INFO : rmak: 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 : rmak: 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 : rmak: 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
```

Fig. 10. Sample log entries for the CIP metadata query service

a time stamp, the user's name, the named of the called method, details of the request, and key information about the results. Data mining these logs, using standard Unix utilities such as *grep* and *sort*, allowed us to deduce usage patterns and tune the system configuration accordingly.

A separate utility monitored the status of the middleware and graphically reported statistics such as memory usage and response times. Knowing the health of server at all times enables system administrators to correct any problems before they became serious. This monitoring was made possible by writing status information for each of the web services to a static data structure that was made available via inclusion on the application servers' main classpath. Fig. 11 shows a screenshot of the graphical monitoring utility. In addition to the statistics tab shown, the Users tab displayed the users currently logged in, the file tab showed which files had recently been accessed by users, and the cache tab showed the metadata that was currently cached from users interaction with the system. In addition to the GUI monitoring utility, a cron job was set up to report the same information to an administrative email address as well.

4.3 Runtime Configuration

An important measurement of software reliability is how long a system 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 meet some change in operational usage or deployed configuration.

A key feature that has allowed the CIP server to stay up and running for extended periods (over 70 days at a time) was dynamic reconfiguration. CIP's middleware design allowed individual services to be hot redeployable. In other words, it was possible to reconfigure and restart a given service while the rest

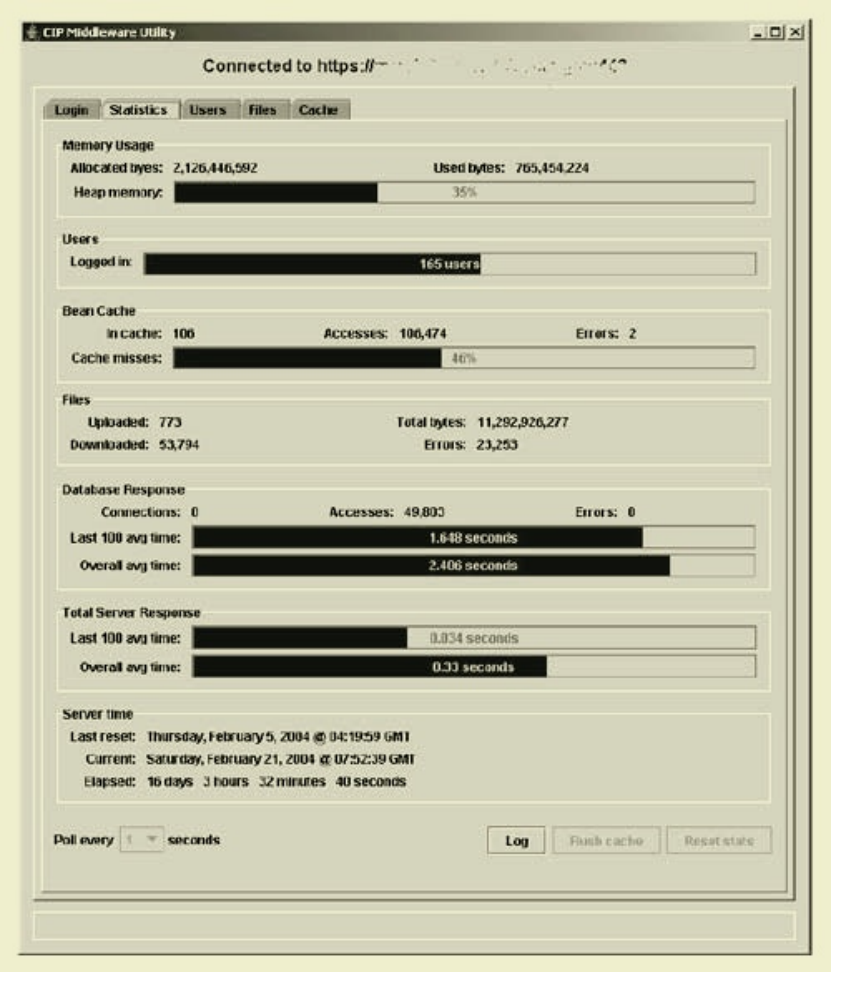

Fig. 11. The CIP Middleware monitoring utility, displaying the primary statistics tab, showing the current memory profile, number of users logged in, cache data, file upload / download statistics, database and server response times, and server uptime. The other tabs displayed the currently logged in users, recently downloaded files and cached metadata

of the middleware web services (and CIP as a whole) continued to run without interruption. To reconfigure a service, a system administrator first edited the service's configuration file and then redeployed the service, causing it to read in the new configuration. Redeploying a service typically took only a few seconds, and often users did not notice any service interruptions even though they had active clients up and running. When the CIP client was unable to connect to a service, the request was queued until the service was reachable.

5 Conclusions

All things considered, CIP performed admirably throughout the MER mission and has received a significant amount of praise for the broad functionality it provided. The responsiveness of the system, its' robustness in the face of heavy user load and the overall availability of the system throughout the mission all met the original requirements and specifications. Further, the flexibility and ease of runtime configuration made it relatively straightforward to make last minute changes when necessary. Nevertheless, there are, of course, aspects of the design that we would do differently if we had to do it all over again.

When we first designed the middleware, a very complex data model [10] was already in place that precluded the use of entity EJBs. Therefore, as described above, we implemented our own data caching algorithm using stateful session EJBs. This became problematic in that we had to solve many thread synchronization problems. In retrospect, we should have simplified the data model and used entity EJBs. The obvious lesson for future projects is to (re)design the data model at the same time as the middleware is being developed to ensure that entity EJBs can be used.

After proving its worth during a series of Operational Readiness Tests at JPL, the mission managers deemed CIP to be mission critical. By then, it was too late for us to increase its reliability by clustering the middleware servers. While CIP turned out to be extremely reliable (with better than 99.9% uptime) despite the middleware running on a single server, admins were obliged to monitor the system extensively over the course of the mission. The lesson for future projects is to always design the middleware to run on clustered servers if possible, since one can always disable clustering if it isn't needed.

Despite the recognized shortcomings, overall we feel the CIP middleware is a good example of a service oriented architecture that was able to overcome several difficult challenges. The use of shared data structures to enable session management across multiple web services and monitoring of server statistics is a useful technique that may be applied in other situations. Further, using JMS to integrate the data acquisition components proved to be a wise design choice, providing the necessary flexibility to migrate these components to another system for performance reasons. Lastly, we were able to effectively unify multiple web services into a single client application, giving many of the users the distinct impression that the CIP client was a unified application with direct and immediate access to the mission data repositories. CIP continues to be one of the primary tools used by MER mission personnel, both remotely and within mission control.

References

1. Joshua Bloch and Mark Pozefsky. Jsr 10: Preferences api specification. Technical report, Sun Microsystems, Inc. and International Business Machines Corporation (IBM), May 2002. http://www.jcp.org/en/jsr/detail?id=10.

2. Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Herik Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1. Technical report, World Wide Web Consortium, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, May 2000.
3. Linda DeMichiel. Jsr 19: Enterprise javabeans 2.0. Technical report, Sun Microsystems, Inc., June 1999.
http://www.jcp.org/en/jsr/detail?id=19.
4. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
5. Mark Hapner, Rich Burridge, Rahul Sharma, Joseph Fialli, and Kate Stout. *Java Message Service*. Sun Microsystems,
http://java.sun.com/products/jms/docs.html, April 2002.
6. D. J. Koga, D. J. Korsmeyer, and J. A. Schreiner. Darwin information system of nasa – an introduction. In *19th AIAA Advanced Measurement and Ground Testing Technology Conference*, New Orleans, LA, June 1996. AIAA.
7. Floyd Marinescu. *EJB Design Patterns*. Wiley Computer Publishing, 2002.
8. Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*. IEEE Computer Society, 2003.
9. Bill Shannon. *Java2 Platform Enterprise Edition Specification, v1.3*. Sun Microsystems, Inc., July 2001.
10. Joan Walton, Robert E. Filman, Chris Knight, David J. Korsmeyer, and Diana D. Lee. D3: A collaborative infrastructure for aerospace design. In *Workshop on Advanced Collaborative Environments*, San Francisco, CA, August 2001.
11. Joan Walton, Robert E. Filman, and David J. Korsmeyer. The evolution of the darwin system. In *Symposium on Applied Computing*, pages 971–977, Como, Italy, March 2000. ACM.
12. Joan Walton, D. Korsmeyer, R. Batra, and Y. Levy. The darwin workspace environment for remote access to aeronautics data. In *35th Aerospace Sciences Meeting*, Reno, NV, January 1997.