ASTRONOMICAL DATA ANALYSIS SOFTWARE AND SYSTEMS XIV
ASP Conference Series, Vol. 347, 2005
P. L. Shopbell, M. C. Britton, and R. Ebert, eds.

ROME (Request Object Management Environment)

Mihseh Kong, John C. Good, and G. Bruce Berriman

Infrared Processing and Analysis Center, California Institute of Technology, Mail Code 100-22, CA 91125

Abstract. Most current astronomical archive services are based on an HTML/CGI architecture where users submit HTML forms via a browser and CGI programs operating under a web server process the requests. Most services return an HTML result page with URL links to the result files or, for longer jobs, return a message indicating that email will be sent when the job is done.

This paradigm has a few serious shortcomings. First, it is all too common for something to go wrong and for the user to never hear about the job again. Second, for long and complicated jobs there is often important intermediate information that would allow the user to adjust the processing. Finally, unless some sort of custom queueing mechanism is used, background jobs are started immediately upon receiving the CGI request. When there are many such requests the server machine can easily be overloaded and either slow to a crawl or crash.

Request Object Management Environment (ROME) is a collection of middleware components being developed under the National Virtual Observatory Project to provide mechanism for managing long jobs such as computationally intensive statistical analysis requests or the generation of large scale mosaic images. Written as EJB objects within the open-source JBoss applications server, ROME receives processing requests via a servelet interface, stores them in a DBMS using JDBC, distributes the processing (via queuing mechanisms) across multiple machines and environments (including Grid resources), manages real-time messages from the processing modules, and ensures proper user notification.

The request processing modules are identical in structure to standard CGI-programs – though they can optionally implement status messaging – and can be written in any language. ROME will persist these jobs across failures of processing modules, network outages, and even downtime of ROME and the DBMS, restarting them as necessary.

1. Introduction

All of us are familiar with retrieving data via a Web browser when the system we are accessing is busy, the request can time-out or take forever, leaving us with nothing to do but wait. Some data retrieval systems handle such situations by accepting a request as a background batch job, returning an acknowledgement of the request immediately and emailing the user when the job is done. Neither approach provides the user with a means of monitoring the progress or interacting with the processing of his request.

From the data provider's point of view, a choice has to be made as to whether to build the service as a blocking CGI job or a batch job. Furthermore, when hundreds, thousands, or even tens of thousands of requests pour into an ap-

plication system (some requiring intense computation or complicated DBMS searching), the system will inevitably reach its limit, and grind to a halt.

ROME is a set of software components that sit between client applications and data retrieval/computational services and manage client requests. From a user's point of view, ROME provides them with the ability to submit, monitor and to some extent interact with their processing requests. For the service provider, ROME stages pools of requests in a secure, fault tolerant system so that the server machine that performs the computation and the DBMS search will not become overloaded.

In this paper, we will describe the high level design of ROME and present request submission scenarios from both a user's and a data retrieval processor's point of view.

2. Design

ROME consists of "Request Manager" and "Request Processor" functionalities. The Request Manager (RM) is a collection of servlets and Enterprise Java Bean (EJB)-based services whose function is to manage information relating to user requests. The Request Processor is a lightweight engine for overseeing the operation of a set of processing applications and conveying status information back to the Request Manager (and ultimately the user).

The ROME request submission mechanism operates through standard client interfaces (Web forms and JAVA GUIs) which collect user parameters with a standard HTTP Web interface. ROME takes responsibility for scheduling and overseeing the execution of request although it contains no processing code itself for efficiency reasons. The request processing is done by CGI programs on a collection of servers.

From a user's point of view, all of the jobs submitted are "batch" and all they see in response to their submission is an acknowledgement (one job ID for each request). There is a Web form for polling and a JAVA GUI client interface to view the job status with immediate and continuous feedback. These job monitor mechanisms allow some user interaction with job executions. For example, a user may submit a set of requests simultaneously, then monitor those jobs' status using ROME's request monitor JAVA GUI. Based on the results for a few job threads, the user may decide that he does not need to complete some or all of the other jobs and can send interrupts to abort them. Similarly, after monitoring some long-running requests for a while he may decide to add email notification and disconnect.

In order to accomplish the above senario for possibly thousands of simultaneous requests, we need a system that not only maintains the database of requests and messages efficiently but is also capable of persisting data in case of the system failure. ROME achieves this by employing the Enterprise Java Bean (EJB) technology (*e.g.*, the open source EJB system JBoss), coupled with a robust DBMS infrastructure (in our case INFORMIX). The EJB framework is an industry standard for business component development and provides an abstraction between component transaction monitors (CTMs) and distributed object services. EJBs reside in a specialized JAVA VM (EJB server) which is respon-

sible for providing the infrastructure for managing such things as transactions, persistence, concurrency, and security.

In one respect our problem is quite different from the standard business transaction. Unlike those systems, our concern is mainly for long-lived queries. In their case, the actual processing of information (which often involves rapid database interaction) can perfectly well be handled within the EJB code itself. In ours, we wish to decouple the actual processing, which could take hours or even days, from the EJB server so the EJB container can do what it suppose to do best and not become a processing bottleneck.

We have done this by separating the components that processes the requests (Request Processor JVM) from the system that manages them (Request Manager JVM) so that the latter can remain lightweight and responsive to request and message handling. With this separation, the Request Management (RM) is responsible for accepting requests and messages, synchronizing them to the database, and retrieving them when requested by a Request Processor (RP). A Request Processor is responsible for managing the worker threads that actually process the requests (usually by means of an external "CGI" process).

Our design (see Figure 1: ROME Architecture Diagram) persists the requests by writing their states to the database so they can be recovered after a system failure. It balances load by having several request processors (RP) external to the RM, each of them having multiple worker threads to run the application programs that actually process the query. Figure 1 shows at least one instance of all the components that comprise or interact with ROME. The steps involved in a typical processing scenario are:

1. A user submits a query,
2. A worker thread in the RP picks up the request,
3. The worker thread starts a copy of a processing application,
4. The worker thread sends information about itself to the RM,
5. The worker thread sends input parameters to the application,
6. The application sends messages to the worker thread,
7. The worker thread sends message to the RM,
8. The RM sends message to the user,
9. The user queries for information pertaining to a certain request.

3. Current Status

ROME has been fully implemented using the open source EJB container JBoss and Informix DBMS and tested within the IRSA testbed. We are currently conducting intense load testing with 20000 requests to ensure its robustness. We are planing to implement a version that interfaces with MYSQL DBMS so that ROME does not rely on any vendor specific software.

The current implementation schedules the requests with first in/ first serve policy plus a priority flag for slow jobs, but it is up to the user and ROME operator to set job priorities. A more sophisticated scheduling algorithm is desirable but not developed.

ROME is designed for long-lasting jobs; if a user has submitted hundreds of such jobs, he can monitor them all at once using ROME's job status monitor

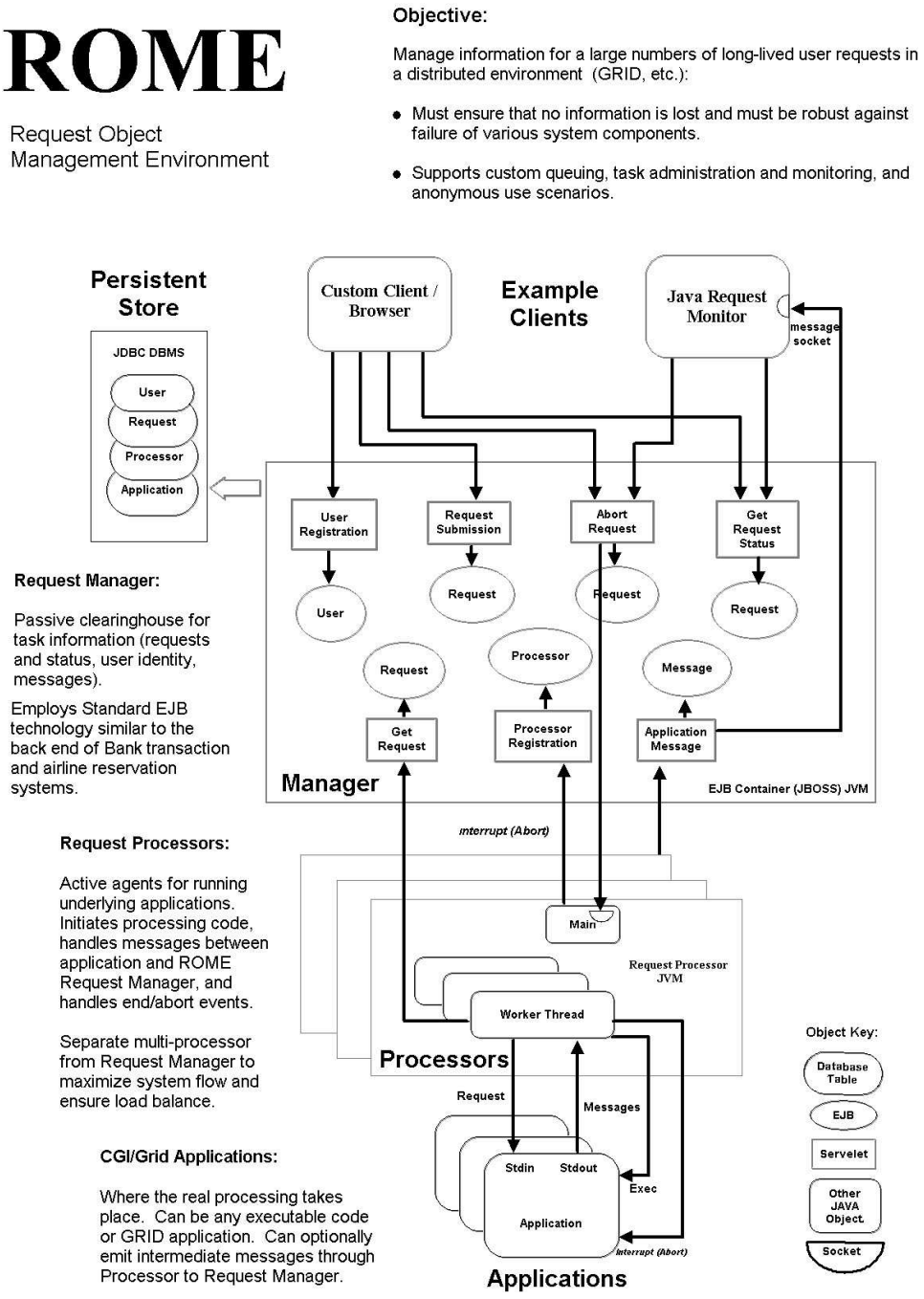
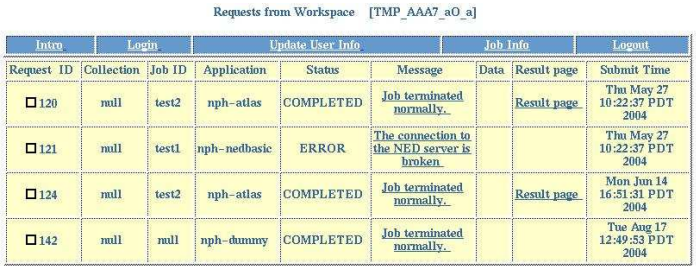


Figure 1. Architecture Diagram

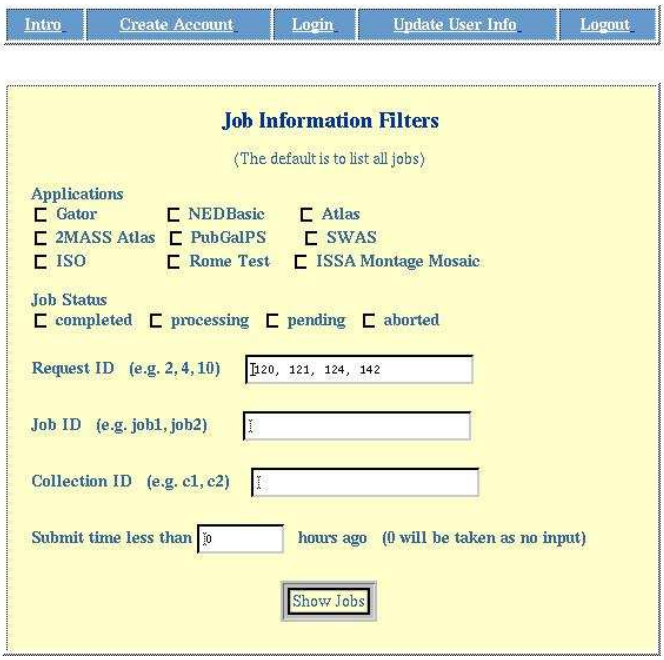


Requests from Workspace [TMP_AAA7_aO_a]

Intro	Login	Update User Info	Job Info	Logout

Request ID	Collection	Job ID	Application	Status	Message	Data	Result page	Submit Time
☐ 120	null	test2	nph-atlas	COMPLETED	Job terminated normally.		Result page	Thu May 27 10:22:37 PDT 2004
☐ 121	null	test1	nph-nedbasic	ERROR	The connection to the NED server is broken			Thu May 27 10:22:37 PDT 2004
☐ 124	null	test2	nph-atlas	COMPLETED	Job terminated normally.		Result page	Mon Jun 14 16:51:31 PDT 2004
☐ 142	null	null	nph-dummy	COMPLETED	Job terminated normally.			Tue Aug 17 12:49:53 PDT 2004

Figure 2. Job status



Intro | Create Account | Login | Update User Info | Logout

Job Information Filters

(The default is to list all jobs)

Applications

☐ Gator ☐ NEDBasic ☐ Atlas
☐ 2MASS Atlas ☐ PubGalPS ☐ SWAS
☐ ISO ☐ Rome Test ☐ ISSA Montage Mosaic

Job Status

☐ completed ☐ processing ☐ pending ☐ aborted

Request ID (e.g. 2, 4, 10) 120, 121, 124, 142

Job ID (e.g. job1, job2)

Collection ID (e.g. c1, c2)

Submit time less than 0 hours ago (0 will be taken as no input)

Show Jobs

Figure 3. Job filter

page. Figure 2 shows a job status page and Figure 3 a job filter pages for locating the jobs currently of interest to the user.

Acknowledgments. ROME is funded by the National Virtual Observatory, under NSF Cooperative Agreement AST01122449.