Jamama Design

Blaine Simpson

Jamama Design

Blaine Simpson

Published $Date: 2004/04/14 05:46:13 $

Table of Contents

List of Tables

Introduction

If you notice any mistakes in this document, please email me at blaine.simpson@admc.com [mailto:blaine.simpson@admc.com?subject=design%20Guide] so that I can correct them. I've just converted this from Texinfo to Docbook, so there are most likely many mistakes in this revision. You can also email me if you have problems with the procedures explained herein, or if you have questions, comments, suggestions or complaints.

Available formats for this document

This document is available in several formats.

You may be reading this document right now at http://jamama.sourceforge.net/design, or in a distribution somewhere else. I hereby call the document distribution from which you are reading this, your *current distro*.

http://jamama.sourceforge.net/design hosts the latest versions of all available formats. If you want a different format of the same *version* of the document you are reading now, then you should try your current distro. If you want the latest version, you should try http://jamama.sourceforge.net/design.

Sometimes, distributions other than http://jamama.sourceforge.net/design do not host all available formats. So, if you can't access the format that you want in your current distro, you have no choice but to use the newest version at http://jamama.sourceforge.net/design.

Table 1. Alternate formats of this document

format	**your distro**	**at http://jamama.sourceforge.net/design**
Chunked HTML	index.html	http://jamama.sourceforge.net/design/index.html
All-in-one HTML	design.html	http://jamama.sourceforge.net/design/design.html
PDF	design.pdf	http://jamama.sourceforge.net/design/design.pdf

Concise Summary of Jamama

Jamama is a Java Managed Mail Server.

The functionality is inspired by the *excellent* mail server Exim [http://www.exim.org], but takes advantage of features of the Java language to simplify the regular expression, extension, and pluggability mechanisms; and takes advantage of Java technologies such as JMX and JAXB to provide real time remote management and dynamic XML configuration, correspondingly.

You configure Jamama by editing the XML configuration file according to the schema, and/or by using a JMX client (including using a web browser to access the HTML JMX Adaptor). When Jamama starts

up, it loads all of the objects that are specified in the configuration file. The configuration file is automatically updated with configuration changes that you make using a JMX client.

Jamama is being implemented according to the Extreme Programming [http://www.extremeprogramming.org] (aka *XP*) methodology. The goals of Jamama are purposefully broad and long-term, but each design decision is nonetheless made according to XP criteria: does the anticipated action/feature/technology directly lead to satisfaction of one (or more) of these goals, without adding too much work, complexity or size?

Primary goals of Jamama

- Full ESMTP compliance
- JMX compliance
- Schema-validated XML configuration (and automatic saving to same XML
- Full pluggability for instances for
  - Mail Routing Director
  - Repositories
  - Routers (mail filtering, routing and consumption abilities)
  - Servers (besides SMTP), such as IMAP, POP, Anti-Spam, Mail-lists

Current status

Jamama is *not* a functionaly mail server yet. At this time, Jamama has full dynamic XML configuration and you can administer these objects with any JMX client. JMX constructors have not yet been tied into the Jamama code, so currently you have to add and remove objects with the XML configuration file (using all default settings if you wish), and after that you can use a JMX client if you wish. Until we implement notification (or implement the feature some other way), changes are not automatically persisted. Changes made through JMX will not be persisted to your XML config file until you use save through the Jamama JMX objec or give the save command through the Jamama console (invoke Jamama with the -i switch to get a console).

Comparison to Apache's James

There is a large overlap of functionality between this project and Apache's James. The developers of Jamama are dedicating hundreds of hours, however, because they are unsatisfied with James. In our opinion, the main goal of James is to showcase the Avalon framework and, in typical Apache project fashion, to implement Java methodologies and Design Patterns for the sake of it. As a particular example, the James project wants to have a servlet-like development model for mail routing, so in order to accommodate that, they have to separate filtering from message modification and processing-- the problem is, these tasks must be performed together if they are to be done right.

One direct consequence of the James projects' infatuation for "Mailets" is, they "accept" all Spam and only filter/route completely accepted messages. They somehow twist this extreme limitation to sound like a benefit by calling James a "black hole for Spam", because James servers have to use their processing and RAM power to consume all Spam sent to them. Users of Exim have understood for years

that the best strategy is to handle routing at the earliest point possible. The Jamama "Router" interface is much more powerful than the Mailet interface, but the lifecycle was not designed to emulate the Servlet lifecycle. Another symptom of the inability to implement useful features because they are too busy with Design Patterns, is the lack of adoption of Avalon JMX capabilities. To summarize: Jamama is about using Extreme Programming to accommodate the email problem domain. James is about using the email problem domain to accommodate Avalon and a score of Design patterns.

Chapter 1. Architecture

Essential Objects

Jamama Object/Instance (aka a *Configurable*)	Objects that are directly configurable through JMX are *Configurable*. This includes Directors (normally only one), Routers, Repositories and ThirdPartyConfiguroros. These are instantiated by virtue of having a block under <instances> in the Jamama configuration file. (When we implement it, you will be able to create new instances with JMX also).
Repository	A generic data getter/fetcher/remover interface to a storage receptacle backend like a DB or a FS branch. A Repository must be able to handle the type of object that you need to persist, and there may be multiple repositories that can persist a particular object type. Each Repository may implement the ability to manage multiple storage objects, so that one Repository could store the same object to different files or to different mailboxes.
item Mit = Message-In-Transit.	Corresponds to elements in a Sendmail "mailq". Message in mail system which has not reached its final destination (i.e., not delivered to a local mailbox or a remote system or dropped).
Director (Initiates all routing)	This is the only object that "initiates" non-ping routing for a Mit. (A Router may reroute a Mit, but that is nonetheless a Mit in-process; to route ANOTHER Mit, a Router must persist a new Mit (and Server will thereafter pick it up). Periodically runs routingRun(null, Mit.STATE_RUNNOW, false) and routingRun(null, Mit.STATE_DEFERRED, false). This attempts to route all Mits that are ready to go.
Pipeline	Sequenced list of Routers. If every configured Router in were to return STATUS_CONTINUE, they Routers would each be accessed in this order and finally the default routing method in Director, which will abort the Mit.
Jamama	The main Jamama process, *Jamama* is a JMX Agent. The Director thread (above) is the main thread, and may have subthreads. Jamama also starts up other Servers, each in its own thread, (and they may also have subthreads).
Router	Filtering AND Mit consumption abilities. A Mit is "routed" through a chain of Routers.
Server	Servers must expose the "run" attribute, i.e., setRun(bool) and getRun(). This will typically be accomplished by implenting Runnable and by invoking (thread.start(this)) for setRun(true). These threads typically generate mits like `Director.createMit().persist(false)` It may have sub-threads, of course.

A Server may also be a Router. (E.g., SMTPServer is primarily a Mit Generator that listens on smtp port in order to create Mits, but it's route() method is also called by Mits running in the same thread-- and only succeeds if the current thread is connected to that host in ETRN mode).

Startup

All of the instances in the main configuration XML file are instantiated.

- The Director and its pipeline.
- Repositories
- Routers
- Servers
- Utility objects used or shared by any of the objects above
-

These objects can thereafter be remotely managed with JMX. For any config change, routers are re-validated and the XML config file is re-written.

Chapter 2. Package and Class Layout

Top Level Jamama Architecture

> **Note**
>
> Need to make a diagram image of this

The basic package strategy is that the main com.admc.jamama.Repository class holds only objects shared among objects in different packages, plus the main Interfaces and the Jamama class itself. External data sources like databases and file systems.

- mamaproxy (interface: com.admc.jamama.DirectorProxy; DirectorProxyAdapter; MamaProxy)
- smtpclient (class: com.admc.jamama.smtp.SMTPClient); [wrapper scripts]
- Jamama JMX Agent JVM (class: com.admc.jamama.Jamama)
  - Repository. (interface: com.admc.jamama.Repository; RepositoryAdapter; classes: com.admc.jamama.repository.X)
  - Server. (interface: com.admc.jamama.Server; ServerAdapter; classes: com.admc.jamama.server.X)
  - Director (interface: com.admc.jamama.Director; classes: com.admc.jamama.director.JamamaDirector)
    - Mit (class: com.admc.jamama.Mit)
    - Router (virtual interface: com.admc.jamama.Router; RouterAdapter; classes: com.admc.jamama.router.X)
    - Proxy (com.admc.jamama.DirectorProxyAdapter; e.g. MamaProxyAdapter)

Some More Sample Implementation Classes

```
com.admc.jamama.router.MailboxRouter extends RouterAdapter

com.admc.jamama.repository.MailboxRepository extends Repository

com.admc.jamama.router.SMTPRouter extends RouterAdapter

com.admc.jamama.server.smtp.Server
```

Chapter 3. Accommodating local Mail User Agents

mamaproxy

A shell script that uses the DirectorProxy interface to make an Mit. It "emulates" a local connection to Sendmail, as used by MUAs. DirectorProxy uses MBean proxy to permit other JVMs to invoke Director.createMit().persist(false). All Mits are created with Director.createMit() (because the Director must identify the Repository) and then be persisted with mit.persist(false). Traditional "mailx" MUA with Sendmail:
mail -> (/usr/sbin/sendmail -i recipient -> ROUTE) ->...

IF local mailbox, then no IPC is needed because the file is just written. If remote, then cfg files are consulted and mail is sent out. With Jamama, the transmitter program always just "relays" to Jamama.

Justifications

- Since there are a bunch of dynamic configuration structures, it will take a considerable amount of work to load all of those structures. We just keep a JVM running with those structures and connect to that. (The alternative would be to keep restarting our main JVM to check Mit's instead of running as a daemon-- that would kill remote Admin ability, etc.).
- We use a management/director process to interface to mailbox persistence (normally in a database). For efficiency, we run these inside our main JVM instead of restarting this management/director process all the time or of running another JVM. Need IPC to these processes to write to local mailboxes.

```
mail -> (/usr/sbin/mamaproxy args) -> Director.createMit().persist(false)
     (by remote JMX).
```

Note

(postnote: and need to set mit.state = STATE_ROUTENOW).

smtpclient

A shell script that makes a TCP/IP SMTP xmit to any SMTP server. By default it goes to the smtp port on localhost, so it can be used to "emulate" a local connection to Sendmail, as used by MUAs.

Chapter 4. App Management Use Cases

(to determine what needs to be an MBean object).

Management of Mits

- Drop (destroy)
- Thaw (decrease retry time)
- Freeze (increase retry time)
- Modify spam status
- Modify virus status

Management of Router settings

- What email domains to handle
- What relays to permit in
- What email domains to handle
- What relays to permit in

Management of Server

- start
- stop

Management of Pipeline

- add
- remove
- modify-sequence

Chapter 5. Some Hairy Design Details

FAILURE/RETRY info

The best place to store failure information is in the object CAUSING the problem. Like in the UnavailableHostCache, not in the message itself. The next time routingRun() is run, all Mits waiting for that host will go.

SMTPCaching

- Keeps per-host status info. It needs to be shared among all Internet Routers.
- Operation: update(domain,ip) Update SMTPCache records for host and/or domain. Call

SMTP RECEPTION

```
MAIL FROM:
    mit = Director.createMit();
    mit.state = STATE_INCIPIENT;
    try {
        Director.headPipe.route(mit, true, null);
    } catch (Exception e) {
        Send error status to client.
        mit = null;
        return;
    }
for each (RCPT TO:) {
    try {
        Director.headPipe.route(mit, true, null);
    } catch (Exception e) {
        Tell client that this recipient not allowed.
    }
}
DATA: (? skip this ?)
    try {
        Director.headPipe.route(mit, true, null);
    } catch (Exception e) {
        Send error status to client.
        mit = null;
        return;
    }
mit.state = Mit.STATE_ROUTENOW
mit.persist(false);
// MIT will send it as soon as it can.  Don't want to ma
// client wait for us after we have persisted it.
```

SMTP

- BULK TRANSMITS done by 1-level recursion, not by multithreading. (I think no need to check for recurse since there is no way to recurse other than 1 level in our recursion one case.
- This incurs no multi-threading, although we have to deal with the existing multithreading in SMTPSession).

OUTGOING SMTPConsumer

- Always SMTPCaching.update(domain,ip) after every successful or failed conn.
- Don't use the Incoming server for outgoing unless client ran ETRN.
- It would be nice to use JavaMail, but I don't know if that supports bulk xmits. If not, could still use JavaMail for prototyping the SMTPSharedConsumer.
- After first successful xmit to this domain, run

```
Director.routingRun("nextPipe == " + mypipe
 + "destdomain == '" + destdomain + "'");
If there are more Mits for this target ipaddr, then t
     router() method will be called recursively.
 ++recursecount;
 try {
    if (recursecount > 1) {
        if (state != STATE.GREETED) throw... // asser
        piggyback(mit);
        return;
    }
} finally { --recursecount; }
piggyback(mit) {
    out.println(mit.this...
    flush();
    return;
}
```

INCOMING SMTPServer

-
  - Always SMTPCaching.update(domain,ip) after every successful or failed conn.
  - Need to maintain a ETRN mode HashMap of currentThread X session.
  - When a SMTPSession gets an ETRN, it can run

```
        Director.routingRun("nextPipe == " + myPipe
         + ip == '" + ipaddr + "'");
   SMTPServer.route(Mit mit, boolean ping) {
        if (ping == true) return;
        session = (SMTPSession) etrnSessions.get(Thread
        session.piggyback(mit);  // Same as in SMTPSess
        return;
   }
```