Multi Account Resident Mail Exchanger Environment (MARMEE) Based On X822-MSP (Mail Submission Pipeline) — A BISOS Feature-Area And A ByStar Capability

Document #PLPC-180051
Version 0.6
December 28, 2018

This Document is Available on-line at:
http://www.by-star.net/PLPC/180051

Neda Communications, Inc.
Email: http://www.by-star.net/contact

Contents

List of Figures

Chapter 1

About MARMEE

1.1 About Multi Account Resident Mail Exchanger Environment (MARMEE)

Multi Account Resident Mail Exchanger (MARME) is a collection of python based Linux facilities that provide for rich sending and receiving of email.

MARME is then augmented by a resident MTA (Mail Transfer Agent) and MUAs (Mail User Agents) such as the Emacs Mail User Environment (Blee-Msg) producing Multi Account Resident Mail Exchanger Environment (MARMEE).

Uses of MARMEE include:

1. Reliable Application Integrated Tracked Mail Sending For Confirmed Communications
2. Email Marketing (Similar to Constant Contact capabilities)
3. Device resident complete mail sending, mail receiving and mail processing environments

MARME tracks delivery of email and allows for detection and processing of delivery reports, receipt notifications and bounced messages.

MARME integrates the following major software packages:

offlineimap (Python Commands Package): Used for retrieving and synchronizing email from remote servers

notmuch (Python Commands Package): Used for searching and reading messages

flufl.bounce (Python Library Package): Identifies bounced messages and isolates original message within bounced messages

email (Python Library Package): A library for managing email messages, including MIME and other RFC 2822-based [2] message documents.

smtplib (Python Library Package): Defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

msgOut from bxMsg (ByStar Python Library Package): Builds on email and smtplib for outgoing email.

msgIn from bxMsg (ByStar Python Library Package): Builds on email for incoming email.

msgLib from bxMsg (ByStar Python Library Package): Builds on email for common aspects of incoming and outgoing email.

Based on the orchestration of these major software packages MARME manages sending and tracking of sent messages.

1.2 About Marmee Software

The entirety of Marmee software is Libre-Halaal (FOSS).

Marmee is part of ByStar. Marmee software is a BISOS Feature Area and Marmee services are ByStar capabilities.

Marmee software is in python and has been packaged as pip in unisos.marme – available at `https://pypi.org/project/unisos.marme`.

The entire Marmee software is available at: `https://github.com/bisos-pip/marme`.

1.3 Outline Of This Document

The outline of this document is illustrated in Figure 1.1.

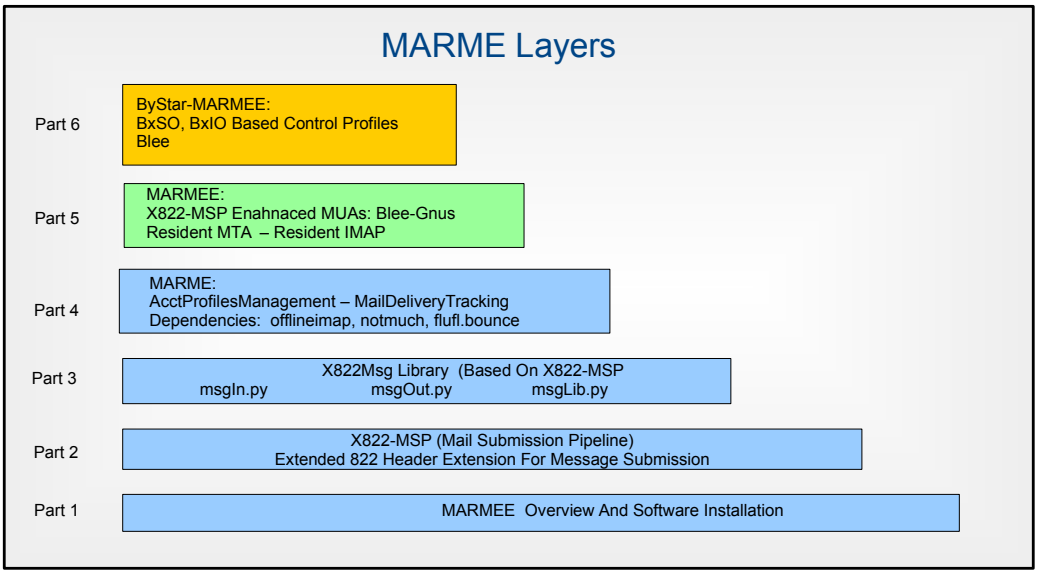

Figure 1.1: MARME Layerings

Part I

MARMEE Software Installation, Configuration, Control And Operation

Chapter 2

MARMEE Software Installation And Configuration

2.1 Overview Of Marmee Configuration And Installation Process

Marmee can be installed on any linux distro with python support.

Marmee is a BISOS-Feature-Area. Marmee usage is abstracted as BxOs (ByStar Objects) and can be used as a "Foreign BxO" when the full BISOS is not adopted.

The underlying requirements are:

- A Linux Distro
- Python 2.7 or higher
- A user account with sudo privileges

2.2 Installing This Software Package In BISOS Or Independently

There are multiple ways to obtain and installing and configuring this software package.

This software package is part of BISOS. For information about BISOS refer to:

> **The Universal BISOS: ByStar Internet Services Operating System**
> **Model, Terminology, Implementation And Usage**
> **A Framework For Cohesive Creation, Deployment and Management Of Internet Services**
> http://www.by-star.net/PLPC/180047 — [1]

You can install this software package in /bisos based on BISOS policies or you can install it as an independent standalone software package.

The scripts mentioned below permit installation of Marmee at any base location.

This documentation primarily focuses on installation and configuration in /bisos based on BISOS policies (/bisos, /bxo, /de/run).

2.2.1 The Foreign-BxO Model Of Marmee Installation And Configuration

When the full BISOS is not adopted, the following setps allows for use of Marmee with Foreign-BxOs.

Foreign-BxOs are limited respresentation of BxO (ByStar Objects) that permit use of Marmee without full adoption of BISOS.

Performing the following setps will result in installation, configuration and verification of Marme software.

1. Install the bisos.bootstrap package:

   ```
   sudo -H pip install --no-cache-dir --upgrade --force-reinstall bisos.bootstrap
   ```

2. Create /bisos bases. Run bisosBaseDirSetup.sh:

   ```
   bisosBaseDirSetup.sh
   ```

3. Install Marme package and dependencies. Run bisosMarmeInstall.sh:

   ```
   bisosMarmeInstall.sh
   ```

4. Create the foreignBxo base directory. System's bx-platformInfoManage.py are copied into the active virtenv. Run foreignBxoBaseSetup.sh:

   ```
   foreignBxoBaseSetup.sh
   ```

5. Obtain And Install Your fbxo:

   ```
   Example:
   cd ~/foreignBxo/
   scp -r yourPriv:~/exampleBxo .
   ```

6. Configure Marme with your fbxo. fbxoMarme-exampleBxo.sh in turn runs fbxoMarmeSetup.sh with /foreignBxo/exampleBxo/fbxoMarme-exampleBxo.config:

   ```
   Example:
   cd ~/foreignBxo/exampleBxo
   ./fbxoMarme-exampleBxo.sh
   ```

7. Verify That Marme Is Properly Configured:

   ```
   Example:
   cd ~/foreignBxo/exampleBxo
   ./fbxoMarme-exampleBxo.sh verify
   ```

2.3 MARMEE Installation And Configuration As A BISOS Software Package

With /bisos bases in place, we can now install unisos.marme and configure it for usage in the /bisos environment.

2.3.1 MARMEE Software Package Installations In BISOS

We will next install unisos.marme in /bisos/venv/py2-bisos-3:

```
source /bisos/venv/py2-bisos-3/bin/activate
```

```
pip install --no-cache-dir --upgrade unisos.marme
```

2.3.2 MARMEE Software Package Configuration In BISOS

unisos.marme software package receives its run base configuration parameters from a series of FileParameters (fp) in:

```
/bisos/venv/py2-bisos-3/lib/python2.7/site-packages/unisos/marme-base/pkgInfo/fp
```

Necessary tools are provided to configure and inspect these configurations as Parameters.

Running:

```
pkgMarmeManage.py
```

provides you a menu of commands that you can run to configure the unisos.marme package.

You can inspect the current values by running:

```
pkgMarmeManage.py -v 20
   --icmsPkgInfoBaseDir=
    "/bisos/venv/py2-bisos-3/local/lib/python2.7/site-packages/unisos/marme-base"
  -i pkgInfoParsGet
```

2.3.2.1 MARMEE Control Base Specification

In the BISOS environment most configuration parameters can be determined based on defaults, but the MARMEE Control Base needs to be explicitly specified.

Assuming that MARME control files reside in $HOME/marmeControlBase

Default configuration within BISOS can be set with:

```
pkgMarmeManage.py -v 20
 --icmsPkgName="marme.dev" --icmsPkgControlBaseDir="${HOME}/marmeControlBase"
 --icmsPkgInfoBaseDir=
   "/bisos/venv/py2-bisos-3/local/lib/python2.7/site-packages/unisos/marme-base"
  -i pkgInfoParsDefaultsSet bisosPolicy
```

2.3.3 MARME Software Preparations

In addition to the MARME Software Package itself, MARME uses various common packages and libraries. These are obtained and managed by running:

```
pkgMarmeManage.py -v 20 -i binsPreps
```

2.4 MARMEE Installation And Configuration As An Idependent Software Package

You can also install MARMEE independent of BISOS.

Such an independent and configuration involves creating the necessary control bases similar to how they are done within BISOS.

Chapter 3

MARMEE Software Control And Operation

3.1 Marme As A Collection Of Interactive Command Modules (ICMs)

Marme is implemented as a collection of collaborative Interactive Command Modules (ICMs).

The concept of ICMs is described in:

> **Unified Python Interactive Command Modules (ICM) and ICM-Players**
> **A Framework For Development Of Expectations-Complete Commands**
> **A Model For GUI-Line User Experience**
> http://www.by-star.net/PLPC/180050 — [?]

Marme ICMs use FileParameters for configuration of Marme parameters and features.

3.2 MARME Control FileParameters – marmeAcctsManage.py

MARME is controlled through a set of FileParameters residing at the path specified by the –icmsPkgControlBaseDir option of pkgMarmeManage.py.

These FileParameters are usually configured and inspected by marmeAcctsManage.py

Running:

```
marmeAcctsManage.py
```

provides a menu for setting the parameters.

These options and parameters are described below.

Prior to the execution of MARAME facilities, you should verify that the needed parameters are properly set.

3.2.1 MARME Identifiers and Control Parameters

MARME operates based on a set of parameters and identifiers which we enumerate below.

```
 /AbstractionTerminology/:: mailAcctDefault, inMailAcct, outMailAcct
mailAcctName         :: Name for a mailAcct which can be inMailAcct or outMailAcct or both
mailAcctCur          :: Currently Slected mailAcct (drives inMailAcct and outMailAcct)

 outMailAcct              :: Name of outgoing mail account (smtpServer)
outMailAcctControlerPars :: Control FPs for outgoing mail account owner (firstName, lastName)
 outMailAcctAccessPars   :: Control FPs for outgoing mail account (smtpServer)

 inMailAcct               :: Name of incoming mail account (imapServer)

 inMailAcctAccessPars    :: Control FPs for incoming mail account (imapServer)
 inMailAcctControlerPars  :: Control FPs for incoming mail account owner (firstName, lastName)
 inMailAcctRetrievePars  :: Control FPs for incoming mail account -- What folders to bring and where t
  inMailAcctMboxesPath     :: Base directory of all inMailAcct Mailboxes
  inMailAcctInbox          :: (maildir) Base directory of inMailAcct Inbox
 inMailAcctMboxCur     :: (maildir) Base directory of currently selected inMailAcct Mbox
```

3.2.2 Control And Config Structures And Usage

```
/File Bases/
mailAcctsBaseDir         :: ../
controlBaseDir       :: ../control/ -- common,inMail/mailAcctName,outMail/mailAcctName
../control/inMail/sa-20000/fp/access/
../control/inMail/sa-20000/fp/access
configBaseDir            :: ../conf/     -- ../conf/mailAcctName/_configName
../conf/sa-20000/_offlineimaprc
varBaseDir          :: ../var     -- ../var/inMail/mailAcctName/maildir, ../var/outMail/mailAcctNam
tmpBaseDir               :: ../tmp/.

inMailAcctAccessBase     :: join(controlBaseDir, "inMail", inMailAcct)
inMailAcctMboxesBase     :: join(varBaseDir, "inMail", inMailAcct, "maildir")
inMailAcctInbox          :: join(inMailAcctMboxesBase, "Inbox")
```

The default profile is chosen by:

```
echo "mailDom" > ${marmeBase}/control/common/defaultMailDom/value
```

3.3 MARME Software Package Operation

All MARME scripts, programs and tools are Interactive Command Modules (ICM). See http://www.by-star.net/PLPC/180050 for details.

All MARME scripts, programs and tools are written in the COMEEGA (Collaborative Org-Mode Enhanced Emacs Generalized Authorship) style. Emacs and org-mode are needed for these enhancements.

The following control and informational tools are in the marmeBase/bin.

3.3.1 Control and Informational Tools – Control Base

```
pkgMarmeManage.py           == Install ICMs-Pkg dependencies
marmeAcctsManage.py         == Configure and Manage Mail Svcs
inMailRetrieve.py           == offlineimap based on inMailBase
inMailUserAgent.py          == notmuch on inMail-Maildirs
inMailDsnProc.py            == Act on DSNs
outMailSend.py              == Based on msgOut
```

3.4 MARME Software Interfaces and Usage

3.4.1 Mail Submission/Injection Rules/Logic

Four sets of improvements are shown in this figure:

1. A dedicated mailbox “envelope@example.com” – separate from admin@.
2. Improved Mail Injection Rules – top right oval.
3. “DSN /NDR (bounce) processor” – top left oval.
4. A Message Tracking database

Two sets of interfaces are also shown in this figure:

1. A msgOut.py interface
2. A msgStateTransition.py interface

We expand on these below.

Part II

X822-MSP (Mail Submission Pipeline)

Chapter 4

X822-MSP (Mail Submission Pipeline)

4.1 Overview Of X822-MSP (Mail Submit Pipeline)

Internet Email format and protocols are based on a set of specifications that date back to RFC-822. Over the years RFC-822 has been updated (enhanced and modified). We refer to the current set of these specifications as X822.

In the context of submitting/sending email (mail origination, mail composition, mail injection), we have extended th X822 fields. We call these extensions X822-MSP (Mail Submit Pipeline).

The scope of some (most) of these extensions are purely local. In other words, some of these extensions are not seen by external mail systems.

The scope of some of these extensions are global. In other words, some of these extensions remain visible to external mail systems.

In this part we focus on the specification of the extention fields for X822-MSP and we do not deal with implementations. The implemention in the form of python libraries is described separately in the next part.

4.2 X822-MSP Local Extensions vs Global Extensions – BX-Tags vs X-B-Tags

The distinction betweem BX-Tags vs X-B-Tags is that BX-Tags are consumed and stripped at MailSending phase but X-B-Tags are kept and may be visible to the recipient and when bounced.

The X-B- header lines are public (global) and should not be stripped. The BX- header lines are private (local) and will be stripped.

4.2.1 X822-MSP: Mail Gui To Mail Submit Client Software Pipeline

"Mail Gui To Mail Submit Client Software Pipeline" is a model for extending X822 header lines for the purpose of communication between independent components of a mail sending system.

During mail-composition, a number of mail-headers are added to the email header.

When the email is to be sent, all the necessary information for the mail submission client may be available within the email headers.

Standard capabilities of X822 Mail Submit Pipeline (X822-MSP) are:

- - Envelope-Addr specification
- - Deleivery-Status-Notifications Request (bounce addresses and delivery reports)
- - Disposition-Notifications (read-receipts)
- - Flexible Parameterized Message Submission Information (host, ssl, user, passwd)

4.2.1.1 Sending Model

Sending is the act of delivering the message to another process for the purpose of transfer. Sending can be one of:

1. Injection – using the command line and pipes
2. Submission – using a protocol (smtp, etc.)

4.2.1.2 Composition Model: (Mail User Agent)

The msg itself is used as a container to gether and carry all parameters and all requests for the message submission. The following local header fields are recognized:

```
  BX-Non-Delivery-Notification
 BX-Non-Delivery-Notification-Req-PerRecipient:
 BX-Non-Delivery-Notification-Req-For:
 BX-Non-Delivery-Notification-Req-To:
 X-B-Non-Delivery-Notification-Actions:
BX-Delivery-Notification
 BX-Delivery-Notification-Req-PerRecipient:
 BX-Delivery-Notification-Req-For:
 BX-Delivery-Notification-Req-To:
BX-Disposition-Notification
 BX-Disposition-Notification-Req-PerRecipient:
 BX-Disposition-Notification-Req-For:
 BX-Disposition-Notification-Req-To:
X-B-Envelope-Addr:
```

```
X-B-CrossRef:
BX-Sending
 BX-Sending-Method:          # inject, submit
 BX-Sending-Run-Control:   # dryrun, debug
BX-MTA-Injection -- Obsoleted
 BX-MTA-Injection-Plugins:   # for composite injection profile
 BX-MTA-Injection-Method:    # inject, submit
 BX-MTA-Injection-Control:   # dryrun, debug
BX-MTA-Rem
 BX-MTA-Rem-Protocol:        # smtp
 BX-MTA-Rem-Host:
 BX-MTA-Rem-Port:
 BX-MTA-Rem-User:
 BX-MTA-Rem-Passwd:
 BX-MTA-Rem-LinkConfidentiality:  ssl/tls
 BX-MTA-Rem-CertFile:
 BX-MTA-Rem-KeyFile:
 BX-MTA-Submission-Pre-Plugins:    # executed before send
 BX-MTA-Submission-Post-Plugins:   # executed after send for error reporting
```

4.2.1.3 Submission Model: (Mail Transfer Agent – Mail Sending Agent)

When the Message Sending Agent (MSA) receives a message with the X822-MSP enhancements, it goes through the following steps:

- BX-MTA-Submission-Pre-Plugins are executued in order specified.
- All the BX- headers are recognized and converted to params.
- Where appropriate BX- headers are converted to standard headers.
- Some BX- headers are stripped.
- Complete SMTP Submit Protocol based on the email.smtp python library is executed.
- BX-MTA-Submission-Post-Plugins are executued in order specified.

Part III

X822 Msg Library

Chapter 5

X822 Msg Library

5.1 Overview Of The x822Msg Library

x822Msg library implements X822-MSP (Mail Submission Pipeline) through which Mail User Agents (MUA) communicate submission methods and DSNs (Delivery Status Notifications) to mail sending agents (MSA) through header lines.

The unisos.822Msg library is available at PyPi as:
https://pypi.org/project/unisos.x822Msg

The complete source code for the unisos.822Msg library is available at:
https://github.com/unisos-pip/x822Msg

The x822Msg consists of three primary modules:

- msgLib.py – Facilities that are common for sending and receiving
- msgOut.py – Mail Sending (outgoing) Facilities
- msgIn.py – Mail Receiving (incoming) Facilities

These are described below.

5.2 msgLib.py

```
from  unisos.x822Msg import msgLib
```

This module includes facilities that are relevant for both sending and receiving mail. For example, returning the complete list of all recipients.

5.3 msgOut.py Library

```
from  unisos.x822Msg import msgOut
```

5.3.1 msgOut.py – Interface And Implementation

5.3.1.1 msgOut.py Interface

A complete interface for email-sending that supports delivery-notification-requests, non-delivery-notification-requests, and receipt-notification-requests has been defined and implemented.

5.3.1.2 msgOut.py Implementation

msgOut.py implements delivery-notification-requests, non-delivery-notification-requests, and receipt-notification-requests.

In the header fields of out going messages we can include:

```
Disposition-Notification-To: envelope@example.com
Return-Receipt-To: envelope@example.com
Notice-Requested-Upon-Delivery-To: <enroleeAddress>
```

Additionally with smtplib at the time of:

```
SMTP.sendmail(from_addr, to_addrs, msg[, mail_options, rcpt_options])
```

The rcpt_options can be used.

Based on the above, most of what can reasonably be done to receive a positive delivery report has been accomplished.

5.3.1.3 msgOut.py Usage

An example of usage of msgOut.py is included in msgOutExample1.py.

5.4 msgIn.py Library

```
from  unisos.x822Msg import msgIn
```

Part IV

MARME – Overview

Chapter 6

MARME Overview

6.1 MARME Model, Abstractions And Terminology

The MARME model fully separates incoming mail retrieval (inMail) from mail sending functionality (outMail). After an email is sent, its delivery and final receipt is tracked through automated retrieval of Delivery Status Notifications (DSN).

6.1.1 Tracked Mail Sending Applications Framework Overview

The general framework for sending and tracking is shown in Figure 6.1.

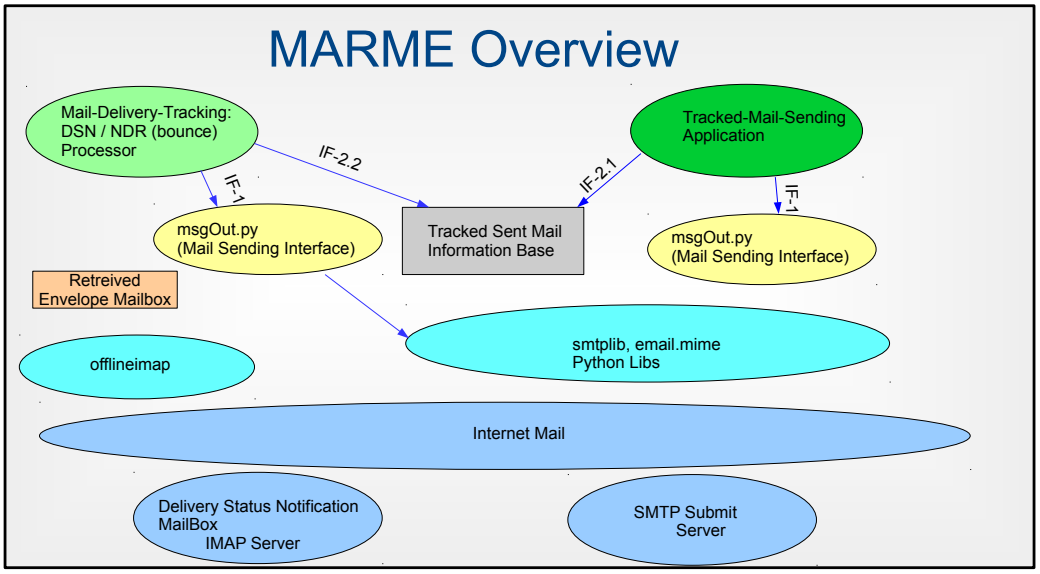

Figure 6.1: Multi Account Resident Mail Exchanger (MARME) Architecture Overview

The organization of components of MARME is described below.

6.1.1.1 Mail Sending Applications

The mail sending application uses X822-MSP (Mail Submit Pipeline) tags for sending of emails and then delivers it to msgOut.

6.1.1.2 Mail Submission/Injection Interface (msgOut from bxMsg)

Mail Submission/Injection Interface is msgOut.py of the bxMsg library.

msgOut accepts emails that are augmented by X822-MSP (Mail Submit Pipeline) tags and submits/injects them based on that information.

6.1.1.3 Mail Sending Agent (SMTP Client)

msgOut then processes X822-MSP tags and uses smtplib to submit the message.

smtplib includes a complete SMTP implementation. Shown in Figure 6.1 labeled as “Mail Sending Agent (SMTP Client)”.

6.1.1.4 SMTP Submit Server

The address of SMTP Submit Server and the needed credentials are specified in X822-MSP tags which then point to the selected SMTP Submit Server.

6.1.1.5 A Dedicated DSN (Delivery Status Notification) Address and Mailbox

At submission time, Delivery Status Notification requests are directed to be sent to a dedicated DSN address and mailbox such as: envelope@example.com.

6.1.1.6 Bounces Mailbox

Bounce messages (both due To: and Cc: bad addresses) will then end up at mailbox of enevlope@example.com.

This is shown on the left side of Figure 6.1.

Chapter 7

MARME - Delivery Tracking

7.1 Message Transition Tracking – Interface And Implementation

As information about delivery/non-delivery of outgoing emails is observed, they will be logged in a database.

The state transition table below captures all recognized states and transitions.

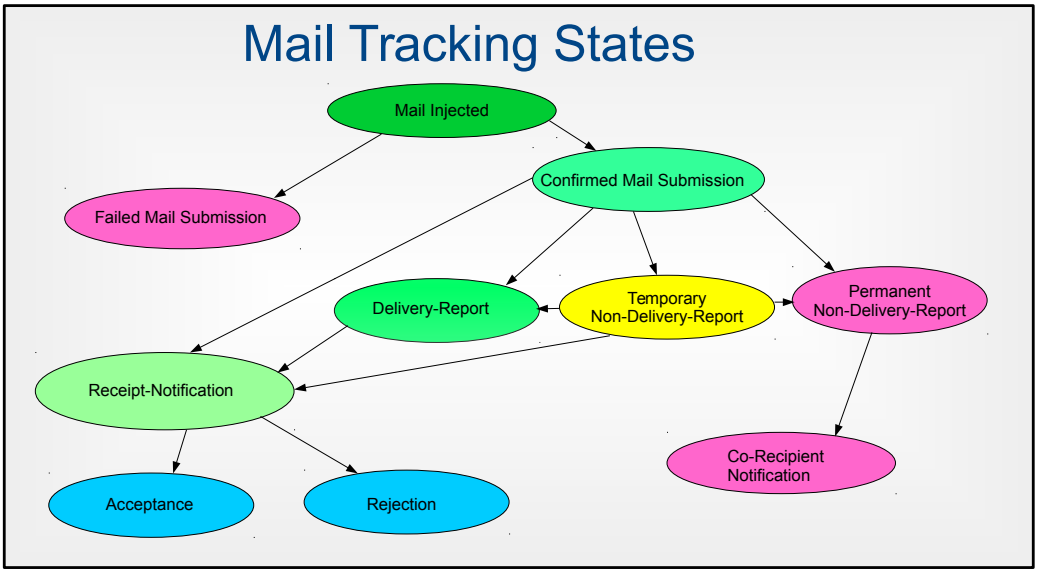

Figure 7.1: Mail Tracking States

7.1.1 Message Transition Tracking Python Interface

Based on the above state definitions, a simple interface – shown below – can be used to capture tranistions.

```
def msgStateTransitionLog(
        msg,
        currentState,
        nextState,
        transitionInfo=None,
):
    """Log a significant state change with regard to delivery of a message."""
```

7.1.2 Message Transition Tracking Python Implementation – Application Database Integration

Based on the above interface, the mail tracking state transitions need to be logged in appropraite databases.

Chapter 8

MARME DSN Processor

8.0.1 DSN /NDR (bounce) Processor

Based on the above all delivery related messages will be sent to the dedicated envelope@example.com.

These include:

- Failed Delivery (NDR) – Non-Delivery Reports (bounces)
- Successful Delivery – In response to DSN Requests
- Delayed Delivery due to temporary failures – soft failures
- Receipt Notifications – Read receipts

These can then be auto processed.

The contours of how this is being done is outlined below:

- A daemon will imap poll envelope@example.com account/mbox for new messages.
- For each positive delivery, a log is created
- For each hard failure (bounce), the failed address is recognized and a copy of the failure is sent to other recipients of the same message.
- For each soft failure (temporary failure) a log is made.

8.0.2 Automated NDR Notification Of Co-Recipients

Consider the situation where John sends and email to Alice and to (or CCs) Bob. Alice's email address results in a non-delivery and John receives a non-delivery-report (NDR). But, Bob has no knowledge of this NDR and may behave based on the assumption that Alice has received John's email.

MARME provides for optional NDR notification of co-recipients. Upon receipt of the NDR by John, Bob can be automatically notified that Alice did not receive John's email.

8.0.3 MARME DSN Overview

The general framework for sending and tracking is shown in Figure 8.1.

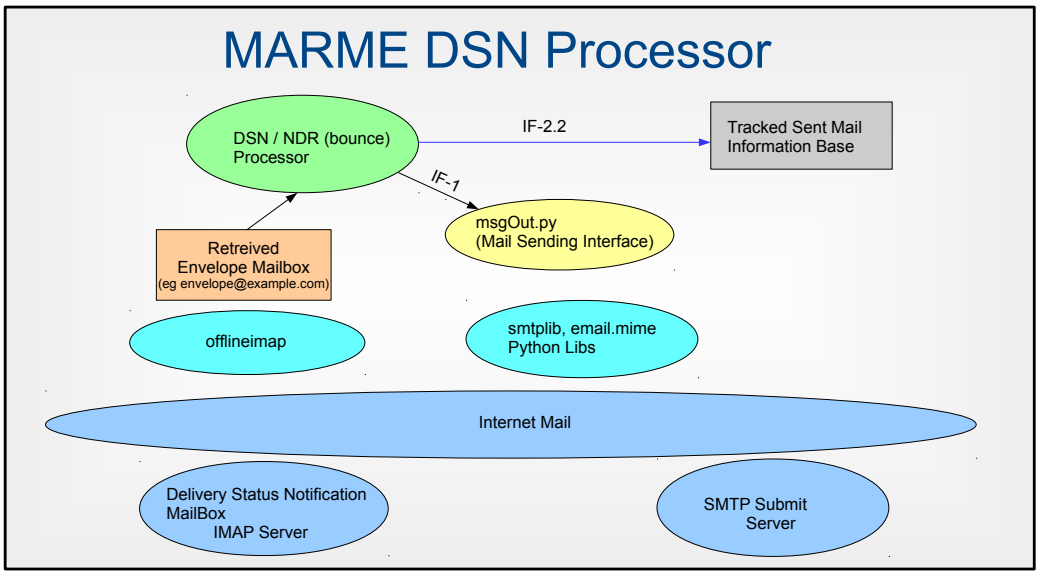

Figure 8.1: MARME Delivery Status Notification (DSN) Processor

Chapter 9

Mail Sending Application Examples

9.1 Confirmed Notifications Email Delivery Applications

A common class of applications that can be built with MARME are “Confirmed Notifications Email Delivery Applications”.

An example could be email delivery of offer letters in a human resources applications.

9.2 Email Marketting Applications

A common class of applications that can be built with MARME are “Email Marketting Applications”.

MARME is capable of providing most of the capabilities that the likes of Constant Contact offer.

Part V

MARME User Environment – MARMEE

Chapter 10

MARME User Environment

10.1 Multi Account Resident Mail Exchanger Environment (MARMEE)

10.1.1 Device software integration

We use an architecture based on the concept of a **Device-Resident End-MTA** middleware module, acting as intermediary between the protocol software and the MUA.

Figure **??** shows the software architecture for integration of EMSD-UA with qmail to create a Device-Resident End-MTA. On its external interface (shown in grey and yellow at the bottom of the figure), the Device-Resident End-MTA interacts with the Internet at large using SMTP, and IMAP. On its internal interface (local loop-back interface; address 127.0.0.1) the Device-Resident End-MTA interacts with the MUA based on SMTP and IMAP. Thus the MUA need have no awareness of EMSD at all. This architecture is quite general and can be used on almost all platforms. In this model, the MUA is always configured for the 127.0.0.1 interface for the SMTP gateway, and the IMAP server. The Device-Resident End-MTA is then configured with the real external server information.

offlineimap is used to optionally synchronize the device's mailstore/Maildir (shown in grey) so that the user's inbox is locally available, even when there is no network connectivity.

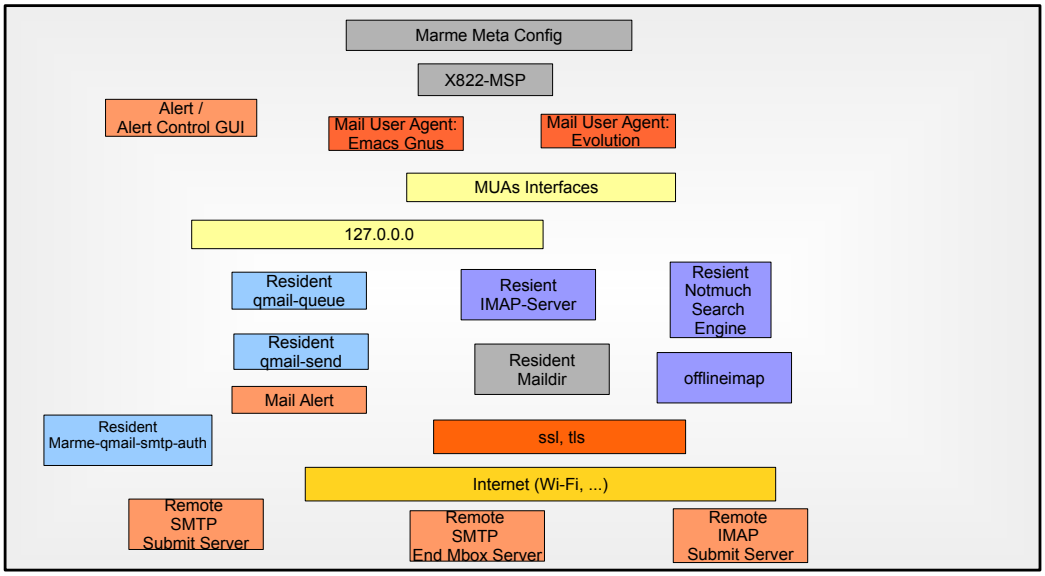

Figure 10.1: General Mail User Environment (Marmee)

Though this architecture is based on qmail, the resulting Device-Resident End-MTA package is quite general, and can be installed in all Linux PDA platforms, and very likely other platforms too.

Note that because all software shown in Figure ?? is free/Libre software, the Device-Resident End-MTA can be made available on any Linux-based device without any restrictions.

Part VI

ByStar MARMEE

Chapter 11

ByStar MARMEE

11.1 ByStar MARMEE

In what was presented above the remote smtp server and the remote imap server were

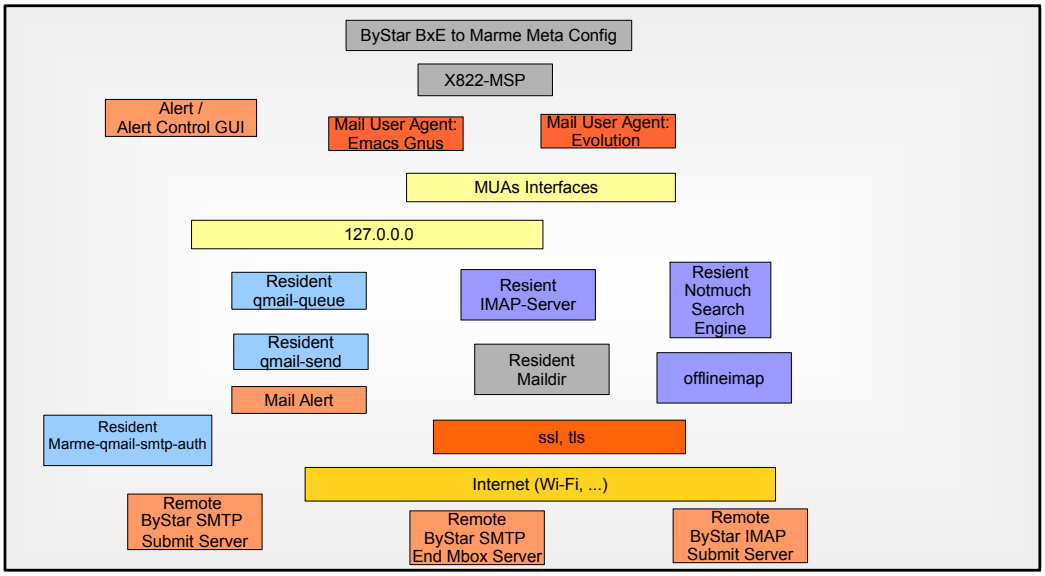

Figure 11.1: ByStar Mail User Environment (BxMarmee)

Bibliography

[1] Inc. ” ” Neda Communications. ” the libre-halaal bystar reference model terminology, architecture and design ”. Permanent Libre Published Content ”180047”, Autonomously Self-Published, ”December” 2014. http://www.by-star.net/PLPC/180047.

[2] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001. Obsoleted by RFC 5322, updated by RFCs 5335, 5336.