Linux groupware server natively serves Outlook clients

EXCHANGE ALTERNATIVE

Zarafa replaces Microsoft Exchange on a Linux server and collaborates with Outlook thanks to native MAPI support.

BY SEBASTIAN KUMMER AND MANFRED KUTAS

Many Linux servers work in heterogeneous environments, serving Windows clients that use MS Outlook for mail and calendar functions. Even if a new groupware system is introduced, the widespread use of Windows clients sometimes makes Outlook an inescapable alternative. Unfortunately, Outlook uses the native Windows Messaging Application Programming Interface (MAPI) for communication with other applications, and the Outlook client is designed to talk to a Windows Exchange server. This preference for Exchange makes the integration of Linux-based groupware systems difficult.

Alternative groupware applications such as Scalix, Kolab, and Open Exchange use an Outlook connector to integrate with Windows clients running Outlook. These tools convert Outlook's MAPI queries to other, partly proprietary protocols. Apart from this, processing relies on services such as WebDav. In many cases, each function has to be configured separately on the client side, and this leads to inflationary administrative overhead. Additionally, this kind of integration only covers a small proportion of Outlook's functionality.

The Zarafa groupware server [1] takes a different approach. Instead of converting Outlook requests, Zarafa offers a comprehensive Microsoft-compatible MAPI interface for Linux environments. Complex conversion of requests is no longer needed because the Zarafa server talks MAPI, specifically MAPI4Linux.

Zarafa is a commercial groupware server that runs on Linux. Because the Zarafa server communicates directly with Windows clients using a variant of MAPI, it provides a high level of Outlook compatibility with minimal client configuration. See the box "Buying Zarafa" for a price summary as given on the Zarafa website. Although these prices are certainly higher than free alternatives such as Kolab, the cost of Zarafa compares favorably with the cost of Microsoft Exchange, and because the server runs on Linux, you can avoid many of the issues associated with running Windows server systems. Contact the company for information on local business partners and support options.

MAPI4Linux

MAPI4Linux supports a compatible Exchange counterpart on Linux. Zarafa comprises the MAPI4Linux library, which controls access to the MAPI store, at is core, and a collection of peripheral tools. A MySQL database is used for storage, and this makes it easier to back up or replicate the data.

MAPI4Linux controls the read and write operations. Direct access to the database is not recommended because it would break the caching and affect the response time.

Open Interfaces

Zarafa relies on open interfaces and tried-and-trusted server components; it

uses Postfix, for example, to send email and Apache as its web server, making it easy to integrate into environments that already implement these services.

Figure 1 shows the Zarafa server's major components. The MAPI kernel is surrounded by various connectors. The figure shows the path an email takes from physical reception by the Mail Transfer Agent (MTA) through to the Zarafa store. The MTA can be supplemented with various tools, such as spam filters or virus scanners. When an incoming mail message is ready for delivery to the receiver, the MTA passes it on to the Zarafa D(elivery) Agent. This process is controlled by the *mailbox_command* variable in */etc/postfix/main.cf*:

```
mailbox_command = ➥
 /usr/bin/zarafa-dagent "$USER"
```

For QMail, the *~/.qmail* file needs:

```
| /usr/bin/zarafa-dagent ➥
 -q user_name
```

The *-q* option tells the D Agent to use Qmail error codes in its reply. The mail is then passed by the agent to MAPI-4Linux, which converts it into a MAPI store object for storage in the database.

Outlook Connection

The Zarafa MAPI provider gives Windows clients that only speak the native language of Microsoft's Exchange server access to the MAPI store. Figure 2 shows the configuration menu.

SOAP messages handle communications between the MAPI provider and the Outlook share on the server. A proxy or Apache web server transparently advertises this service on Intranets or the Internet. To allow this to happen, you just need the following entry in your Apache configuration:

```
<IfModule mod_proxy.c>
 ProxyPass /zarafa ➥
http://127.0.0.1:236/
 ProxyPassReverse ➥
 /zarafa ➥
http://127.0.0.1:236/
 <Location /zarafa>
  Order Allow,Deny
  Allow from all
 </Location>
</IfModule>
```

This supports Outlook access without VPN access or port forwarding on the firewall. Connections between the server and the client are simple web connections. Using Apache's own tools, you can enable standard and SSL connections. Additionally, administrators can restrict access to specific subnets. Figure 3 shows how Outlook is mapped to MAPI4Linux.

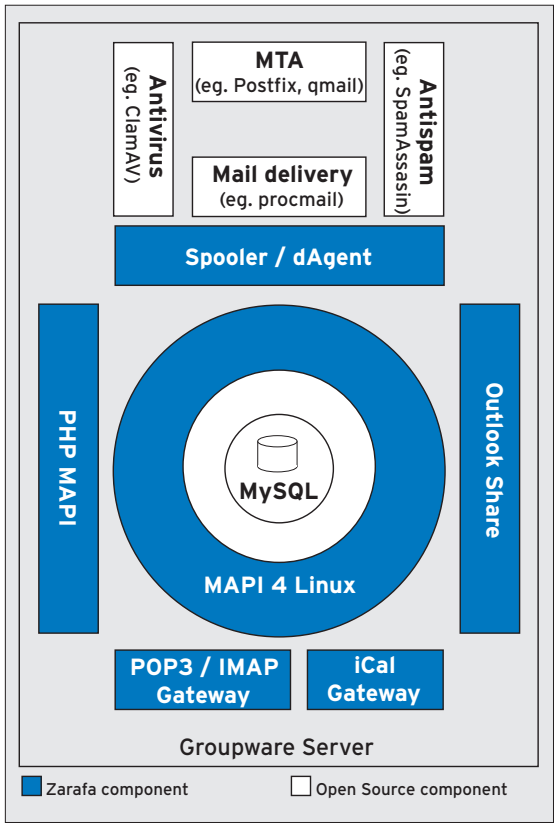

Figure 1: Overview of the Zarafa server's major components.

Alternative Applications Still Supported

You can continue to use POP or IMAP clients like Mozilla Thunderbird. A POP/IMAP gateway gives you access to the email folders, which is easy to configure as all you need to do is specify the services and ports to enable in */etc/zarafa/gateway.cfg*. The gateway converts email from the MAPI format to regular plain text mail before it reaches the client. Double conversion of mail – into MAPI format for incoming mail, and back again before being dispatched via the Gateway – would appear to be a waste of resources at first glance; however, the benefits in terms of compatibility with any component outweigh the overhead.

Alternative calendaring applications such as Mozilla Sunbird are supported thanks to the Zarafa iCalendar interface. The iCal interface emulates server profiles to allow Sunbird to work with live data from the Zarafa store.

Changes or new appointments are immediately stored in the MAPI store, where they are available in real time to all users via all supported interfaces.

Large Systems

A multiple-server setup is useful for large installations with thousands of users. Although you can't install the Zarafa core itself on multiple systems, the service-based architecture does

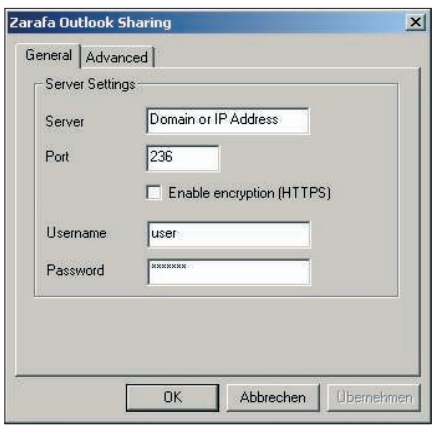

Figure 2: MAPI provider configuration menu.

Buying Zarafa

The Zarafa website lists the following prices:

- Base price for up to 5 users: EUR 300 (US$ 439)
- Every additional 5 users: EUR 150 (US$ 219)
- More than 100 users: 5 percent reduction
- More than 250 users: 10 percent reduction
- More than 1,000 users: 15 percent reduction

Municipalities receive a 25 percent reduction. An education version for schools has a 25 percent baseline reduction, with greater discounts available for higher volumes.

For updates and upgrades after the first year, you'll need to pay a yearly fee of 20 percent of the list price.

support the following configuration:

- Server 1: MySQL database
- Server 2: Zarafa core
- Server 3: MTA + antispam/virus
- Server 4: web server

The Zarafa core server configuration defines the connection between the database and the Zarafa core, which is the only entity to talk to it. All the other components can use TCP port 236 to access the core server.

The MTA server hands incoming email to the delivery agent, which runs on the MTA server and uses an SSL certificate to authenticate with the Zarafa core server. The web server follows a similar approach to communicate with the Zarafa core and thus bind the web components to the MAPI store.

If your spam volumes are particularly high, it might make sense to distribute the MTA and antispam or antivirus software to separate servers.

This scenario also shows how to run the web server in the DMZ, while the database and the Zarafa core reside on the secure internal network.

Data Import

Zarafa offers various approaches to importing data sets into a store. Using the open source tool *imapsync* [2], you can migrate data from other IMAP servers. To do so, the admin users would create a user in Zarafa and then use the Zarafa IMAP gateway to handle the synchronization process.

If you need to migrate multiple users, scripts could be the answer. For existing Outlook systems, whether standalone or with a Microsoft Exchange server, Zarafa has its own migration tool (see Figure 4) for importing the *.pst* files. This tool must be run on a Windows system; it has an unattended mode and can handle about 7GB of data per hour.

Notifications

One of MAPI's biggest strengths is its notification mechanism, which servers can use to push messages to clients. Changes thus become visible on the system after a short interval of less than a second and do not require user interaction.

Network topologies often do not allow servers to reach clients directly when sending messages of this kind. To work around this problem, the client opens a long open http request (maximum 60 seconds), which waits for a result that is relevant to the client. After this interval has expired, the connection terminates; the request then recommences immediately after this. Each Outlook client establishes four or five connections of this kind to avoid interruptions.

If you have many clients, it makes sense to increase the maximum number of parallel connections for the web server – Apache restricts this to 100 by default. Because the lightweight requests generate a couple of bytes of network transfer traffic, this isn't a problem.

User Management

Internal user management is fine for smaller environments. Currently, this involves using a command-line tool that also supports OpenLDAP and Active Directory. The admin simply adds required attributes to a configuration file. Because changes to the directory service do not trigger events to update the data, Zarafa authenticates each user at login.

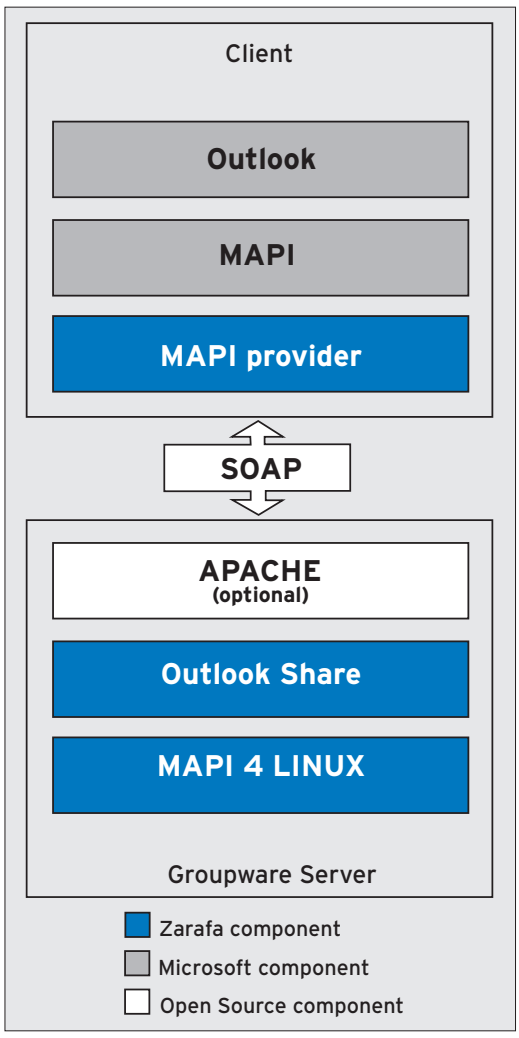

Figure 3: How Outlook maps to MAPI4Linux.

Changes immediately take effect in the store. For example, Zarafa will immediately create mail boxes for new users. Admins can use policies and scripts to tell the system how to react to other kinds of change (such as when groups are modified at directory service level).

Backup

The Zarafa Backup Utility does what its name implies. The utility creates two files: One contains the data, the second contains an index. Creating a consistent snapshot of the complete store without blocking the database is not possible, which means that elements that change or are created during the backup process are not included in the backup.

The current 5.20 version of Zarafa introduces advanced backup options. The new features include support for brick-level backup. Individual stores, including the public store, can now be backed up fully or incrementally. Brick-level backup now allows administrators to restore the whole store, individual messages, or complete directories.

Because it does not store the meta data, this method is not useful for disaster recovery. If you do a full restore from the brick-level backup, you use all view settings, rules, and unique user and store IDs. Also, administrators must create the user MAPI profiles again from scratch under Windows.

Information on other features, such as synchronization tools for PDAs (via SyncML), privilege management via ACLs, or the mobile web interface for HTML-capable mobiles, is available online [3].

Fortunately, Zarafa supports not only commercial Linux distributions, such as Red Hat and Novell SLES, but also Debian and Ubuntu. Also, Zarafa 5 is now available for 64-bit systems.

MAPI via PHP

Many online applications are programmed in PHP, a programming language that is an obvious choice due to its rapid development potential and widespread use.

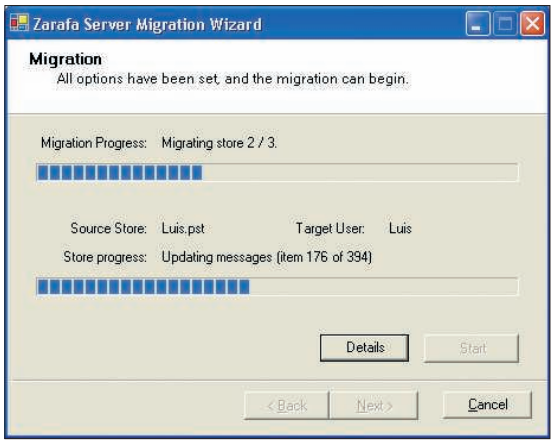

Figure 4: The Zarafa migration tool can handle 2GB of data per hour.

Zarafa features programmed in PHP include the web front end, Webaccess. Webaccess resembles the Outlook client and communicates directly with the MAPI4Linux layer via the PHP-MAPI module. PHP-MAPI has useful options for adding groupware functionality to open source solutions in areas such as Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), document management systems, or wikis.

The module is a prebuilt *.so* file. The PHP configuration file, *php.ini*, loads the module like this: *extension = mapi.so*. Developers can use this module to implement MAPI functionality in PHP. Zarafa also offers a detailed interface description online [4], and general information on MAPI is available [5].

Although Zarafa stores data in a MySQL database, the use of PHP-MAPI is the recommended approach for access by web applications to the MAPI store. Figure 5 shows PHP-MAPI's web server integration.

Programmers can use special PHP functions to connect to the store and, for example, read its properties.

Z-Push

Previously, the only way to synchronize data with a PDA was to use the cradle and a desktop tool with SyncML. In April 2007, Zarafa introduced Z-Push, an open source implementation of Microsoft's ActiveSync protocol.

PDAs with Windows Mobile 2003, 2005, and 6.0 can synchronize their local contacts, email, appointments, and various tasks with the server online via GPRS or UMTS.

Zarafa Z-Push is based on WAP Binary XML (WBXML), like Microsoft AirSync, which is used by the original; this is a kind of low-overhead XML for narrow bandwidths. Users do not need to install any additional software on the PDA since Z-Push handles synchronization natively. Previously, back ends integrated Zarafa and MailDirs. A comprehensive interface description supports collaboration between Z-Push and any groupware system. Additionally, Z-Push now supports IMAP as a back end, bringing push services with IMAP to cell phones for free [6].

PHP-MAPI Technology

Enterprises use a variety of Intranet applications that benefit from the ability to display and modify appointments or contacts from groupware. PHP-MAPI offers the ability of integrating MAPI functionality with existing web solutions.

Basically, you have two options for accessing the Zarafa server from a PHP application: via a UNIX socket:

```
$zarafaserver = ➥
 "file:///var/run/zarafa"
```

or via the SOAP interface:

```
$zarafaserver = ➥
"http://url_zum_zarafaserver➥
:236/zarafa".
```

To make it easier to log onto a server via a socket, you can make the user who runs the application a Zarafa admin, thus enabling access to the store without entering a password. This gives any PHP script administrative access to the server, but this method is not recommended for security reasons.

Example of a PHP-MAPI Application

Listing 1 is a sample calendar function that shows how MAPI integration works [8]. The first step is to set up a connection to the MAPI store:

```
mapi_openmsgstore_zarafa➥
(string $user , ➥
string $password, ➥
string $server)
```

This example uses a socket connection. Note that the application logs on to the MAPI store as an administrative user and that a password is not required to authenticate.

Successful Login

A successful login returns an array with two stores: the user's private store with data from the user's own PIM and the public store with data for shared use.

MAPI stores have a tree structure. To access a branch or leaf, you need to know its address. Properties are used for addressing purposes. The *mapi_prop_tags()* function creates addresses from a type and an ID.

The *mapitags.php* file in the {*webaccess*}*/mapi* folder (for Zarafa 5) or in

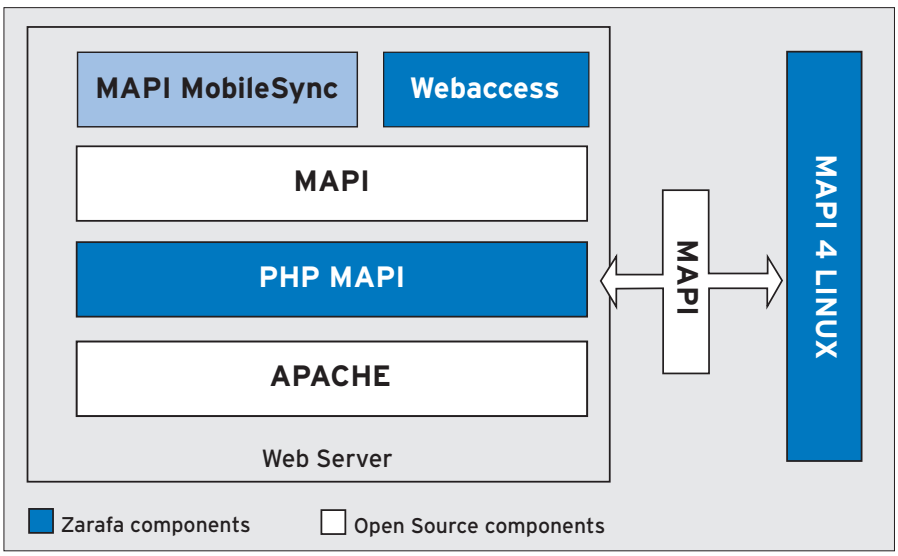

Figure 5: PHP-MAPI web server integration.

Listing 1: PHP-MAPI Sample Application

```
<?
//own definition file
require "mapidefs.php";

//connect to store and get properties of calendar
$store = mapi_openmsgstore_zarafa("john", "", "file:///var/run/zarafa");

$receivefolder = mapi_msgstore_getreceivefolder($store[0]);
$rcvFProps = mapi_getprops($receivefolder, Array(PR_IPM_APPOINTMENT_ENTRYID));

//get entryid of calender in binary form
$entryID = $rcvFProps[PR_IPM_APPOINTMENT_ENTRYID];

//get calendar folder
$folder = mapi_msgstore_openentry($store[0], $entryID);

//get contents of calender-folder
$contents = mapi_folder_getcontentstable($folder);

//guids for calender
$guid = makeguid("{00062002-0000-0000-C000-000000000046}");
$guid2 = makeguid("{00062008-0000-0000-C000-000000000046}");

//get ids of properties
$namedproperties = mapi_getIdsFromNames($store[0], array(0x820D, 0x820E, 0x8223,
0x8216, 0x8205, 0x8214, 0x8215, 0x8506, 0x8217, 0x8235, 0x8236, 0x8208, 0x8233),
array($guid, $guid, $guid, $guid, $guid, $guid, $guid, $guid2, $guid, $guid,
$guid, $guid, $guid));

// generate mapi like ids and put them in an array
$props[0] = mapi_prop_tag(PT_SYSTIME, mapi_prop_id($namedproperties[0])); //
Start
$props[1] = mapi_prop_tag(PT_SYSTIME, mapi_prop_id($namedproperties[1])); // End
$props[2] = mapi_prop_tag(PT_BOOLEAN, mapi_prop_id($namedproperties[2])); //
Recurring
$props[3] = mapi_prop_tag(PT_BINARY, mapi_prop_id($namedproperties[3])); //
Recurring data
$props[4] = mapi_prop_tag(PT_LONG, mapi_prop_id($namedproperties[4])); //
BusyStatus - [Free, Temporarily Occupied, Occupied, Not Available]
$props[5] = PR_ENTRYID; // unique meeting ID
$props[6] = PR_DISPLAY_TO; // Recipients
$props[7] = PR_SUBJECT; // Subject
$props[8] = mapi_prop_tag(PT_LONG, mapi_prop_id($namedproperties[5])); // label
$props[9] = mapi_prop_tag(PT_BOOLEAN, mapi_prop_id($namedproperties[6])); //
alldayevent
$props[10] = mapi_prop_tag(PT_BOOLEAN, mapi_prop_id($namedproperties[7])); //
private
$props[11] = mapi_prop_tag(PT_LONG, mapi_prop_id($namedproperties[8])); //
meeting
$props[12] = mapi_prop_tag(PT_SYSTIME, mapi_prop_id($namedproperties[9])); //
Start Recurring
$props[13] = mapi_prop_tag(PT_SYSTIME, mapi_prop_id($namedproperties[10])); //
End Recurring
$props[14] = mapi_prop_tag(PT_STRING8, mapi_prop_id($namedproperties[11])); //
location
$props[15] = mapi_prop_tag(PT_BINARY, mapi_prop_id($namedproperties[12])); //
```

Listing 1: continued

```
057 timezone data
058 $props[16] = PR_BODY;
059
060 //restrictions' array
061 $restriction = Array(RES_OR,
062    Array(
063       // OR
064       // (item[start] >= start && item[start] <= end)
065          Array(RES_AND,
066             Array(
067                Array (RES_PROPERTY, Array(RELOP =>
068                RELOP_GE, ULPROPTAG => $props[0], VALUE
069                => $start)),
070                Array (RES_PROPERTY, Array(RELOP =>
071                RELOP_LE, ULPROPTAG => $props[0], VALUE
072                => $end))
073             )
074          ),
075        // OR
076        // (item[end]   >= start && item[end]   <= end)
077           Array(RES_AND,
078              Array(
079                Array(RES_PROPERTY, Array(RELOP =>
080                RELOP_GE, ULPROPTAG => $props[1], VALUE
081                => $start)),
082                Array(RES_PROPERTY, Array(RELOP =>
083                RELOP_LE, ULPROPTAG => $props[1], VALUE
084                => $end))
085              )
086           ),
087        // OR
088        // (item[start] <  start && item[end]   >  end)
089           Array(RES_AND,
090              Array(
091                Array(RES_PROPERTY, Array(RELOP =>
092                RELOP_LT, ULPROPTAG => $props[0], VALUE
093                => $start)),
094                Array(RES_PROPERTY, Array(RELOP =>
095                RELOP_GT, ULPROPTAG => $props[1], VALUE
096                => $end))
097              )
098           ),
099        )
100 );             // global OR
101
102 $start = mktime(0, 0, 0, 9, 1, 2007);
103 $end = mktime(23, 59, 59, 9, 30, 2007);
104
105 //get the required calender items
106 $rows = mapi_table_queryallrows($contents, $props, $restriction);
107 foreach ($rows as $appointment) {
108 /* do something */
109 }
110 ?>
```

{*webaccess*}*/include/mapi* (for Zarafa 4) contains a list of constants.

Alternatively, you can use tools like OutlookSpy [7] to search for the required properties. If the programmer uses MAPI *include* files from the directories we just mentioned, there is no need to create addresses.

Once the connection to the store has been set up, you can open the user's inbox to access all the objects in the store:

```
mapi_msgstore_getreceivefolder➥
(mapimsgstore $store);
```

The *mapi_getprops()* function reads the properties of the required object. In this case, it is an entry ID for the calendar. *mapi_msgstore_openentry()* lets us create a pointer to the calendar and access further calendar object properties. *mapi_folder_getcontentstable()* then opens the messages in the folder.

Before you can start reading appointments, you need to generate the IDs for the required properties (start, end, location, and so on) using *mapi_prop_tag()* and *mapi_getIdsFromNames()*.

Restrictions

The example only shows appointments for the month of September; it uses PHP-MAPI restrictions to do so.

The *restriction* array contains a time stamp for the start and end of the required period. Then, the array is passed in to the MAPI request.

Now you know where the folder with the calendar entries is, which appointment properties you want to query, and what restrictions they are subject to. The *mapi_table_queryallrows()* function stores the relevant entries in a result array, giving you the ability to display it.

Conclusion

Zarafa is a robust groupware server that integrates seamlessly with existing Linux environments. The core is supplemented with open source components such as Apache, Postfix, or MySQL.

Besides native Outlook access, clients benefit from POP3, IMAP, or iCalendar interfaces. PHP-MAPI gives developers rapid access to data and the ability to manipulate the data in the store.

Webaccess comes with an AJAX interface and new functions. Because the fea-

ture scope is similar to that of Outlook, many users have started to use Webaccess exclusively.

Z-Push gives PDAs continuous access to the latest data.

The well-documented and completely open interfaces facilitate the integration of Zarafa with existing systems. Right now, Zarafa is working on server-to-server integration with CRM systems such as SugarCRM and vTiger, thus providing the same data to all of these systems in real time.

Zarafa is a commercial product, but it is a complete groupware solution for a fair price and development work is in full swing. ■

INFO

[1] Zarafa: *http://www.zarafa.com/*

[2] ImapSync: *http://freshmeat.net/projects/imapsync*

[3] Zarafa features: *http://zarafa.com/features.html*

[4] PHP-MAPI: *http://download.zarafa.com/zarafa/en/zarafa_php_ext_5.00.pdf*

[5] MSDN-MAPI: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/mapi/html/ef00004c-893c-4136-8cd4-89b729b7401e.asp*

[6] Z-Push: *http://z-push.sourceforge.net*

[7] Code listings: *http://www.linuxpromagazine.com/resources/article_code*

[8] OutlookSpy: *http://www.dimastr.com/outspy*

THE AUTHORS

Sebastian Kummer studied Computer Science at HAW, the University of Applied Science in Hamburg Germany. He has worked as a free software developer for various enterprises since 2000.

Sebastian has also worked on Zarafa migration and integration for inmedias.it GmbH in Hamburg. Since September 2006, Sebastian has been working on developing the mobile security design for the *colamo.org* project.

Manfred Kutas studied Computer Science at HAW, the University of Applied Science in Hamburg Germany. He works as a freelance developer, focusing on PHP and Java. In the scope of the open source project, *colamo.org*, Manfred has worked exhaustively with PHP-MAPI and implemented read/write access to the Zarafa server for *inmedias.it*.

Listing 2: Definition File

```
01 <?
02 /* Objects' definitions */
03 define ('PT_LONG', 3);    /* Signed 32-bit value */
04 define ('PT_BOOLEAN', 11); /* 16-bit boolean (non-zero true) */
05 define ('PT_SYSTIME', 64); /* FILETIME 64-bit int w/ number of 100ns
   periods
06 since Jan 1,1601 */
07 define ('PT_STRING8', 30); /* Null terminated 8-bit character string
   */
08 define ('PT_BINARY', 258); /* Uninterpreted (counted byte array) */
09 define ('PT_TSTRING', PT_STRING8); /* Alternate property type names
   for ease of
10 use */
11
12 /* Property tags' definitions for standard properties of MAPI
   objects.*/
13 define ('PR_SUBJECT', mapi_prop_tag( PT_TSTRING, 0x0037));
14 define ('PR_BODY', mapi_prop_tag( PT_TSTRING, 0x1000));
15
16 /* Message non-transmittable properties */
17 define ('PR_DISPLAY_TO', mapi_prop_tag( PT_TSTRING, 0x0E04));
18 define ('PR_NORMALIZED_SUBJECT', mapi_prop_tag( PT_TSTRING,
   0x0E1D));
19
20 /* properties that are common to multiple objects (including message
   objects));
21  * -- these ids are in the non-transmittable range */
22 define ('PR_ENTRYID', mapi_prop_tag( PT_BINARY, 0x0FFF));
23
24 /* Extra properties */
25 define ('PR_IPM_APPOINTMENT_ENTRYID', mapi_prop_tag( PT_BINARY,
   0x36D0));
26
27 /* restrictions */
28 define('RES_AND',                    0);
29 define('RES_OR',                     1);
30 define('RES_PROPERTY',          4);
31
32 /* restriction compares */
33 define('RELOP_LT', 0);
34 define('RELOP_LE', 1);
35 define('RELOP_GT', 2);
36 define('RELOP_GE', 3);
37
38 /* array index values of restrictions */
39 define('VALUE',              0);          // propval
40 define('RELOP',              1);          // compare method
41 define('ULPROPTAG',          6);          // property
42 ?>
```