Dynamic websites with Midgard

PHP INSIDE

JOCHEN LILLICH

Modern websites are dynamic – their contents come from databases, not just static HTML pages. Midgard Web Application Server is designed to help you create just such a site and will save you a lot of boring programming

Modern websites need to have comprehensive and, most importantly, up-to-date information – not just a pretty lot pictures and some boring text. PHP has established itself as a powerful language in the development of database-enabled websites, but you'll soon find that performing even routine functions can be a real pain. Indeed, editing text and other database contents, creating templates for separating content and layout, and creating interfaces for updating the database and so on require you to do a great deal of work. These basic functions are therefore often achieved via some form of middleware that sits between the database and the web application. One fairly new representative of such an application server, as it's often known is Midgard. This was written just over a year ago as the basis of an editorial system for a Finnish online magazine, hence its Norse mythological name. Since then, Midgard has become an international Open Source Software Project and is developing rapidly. Have a look at *http://www.midgard-project.org for more details*.

Teamwork

Midgard is an expansion of the widely-used LAMP combination: Linux (L) as operating system, Apache (A) as webserver, mySQL (M) as database and PHP (P) as programming language. Midgard adds important building blocks to this combination by providing PHP with a pre-defined, yet expandable, data model. This data model splits the individual applications of a Midgard server into "Hosts", in "Styles" and "Groups" data. "Content" refers to he actual information required from a website. For this Midgard supports text (raw text or HTML), calendar entries and binary data. "Styles" are freely definable style and page elements. This means there is complete separation in Midgard between application logic, contents and layout. Which users can process which parts of these is defined in "Groups", the management of user accounts and user groups.

Installation steps

Before installing Midgard you should already have an Apache webserver and a my SQL database running on your system. As Midgard brings with it an expanded PHP3, any PHP module already installed for Apache must first be removed.

From *http://www.midgard-project.org/download/* you'll need to download the midgard-data, midgard-lib, midgard-PHP and mod_midgard packages. These should each be unpacked, compiled and installed in a working directory.

The Library Midgard-lib contains the basic Midgard functions and is installed as follows: The parameter "--with-mysql" should now refer to the directory in which the Include files and Libraries of the database can be found.

```
tar xvzf midgard-lib-1.2.5.tar gz
cd midgard-lib-1.2.5
./configure --with-mysql=/usr
make
make install
```

mod_midgard adds a Midgard interface to Apache. The configuration parameter "--with-apache" refers to the configuration directory of the webserver, and "--with-midgard" indicates where the Midgard libraries have been filed.

```
tar xvzf mod_midgard-1.2.5.tar.gz
cd mod_midgard-1.2.5
./configure--with-apache=/etc/httpd \
--with-midgard=/usr/local
make
make install
```

midgard-php now contains the new PHP functions.

```
tar xvzf midgard-php-1.2.5.tar.gz
cd midgard-php-1.2.5
./configure --with-apxs=/usr/sbin/apxs \
with-mysql=/usr \
  --with-idgard=/usr/local --enable-
tracks-vars \
 --with-system-regex
make
make install
```

To enable the connection between Midgard and the Apache webserver, the configuration file normally found in /etc/httpd/httpd.conf has to be expanded by a few lines (see box). Some of these new lines should already have been added by the procedure we've already outlined, but do make sure that these have been entered at the right place and not, for example, in *an/ifdef/block*.

Finally we need to deal with the Midgard database and the websites (see box).

After a new log-in to the database, using the command *mysql -u midgard -pmidgard midgard*, this time as user "midgard", we enter the two webhosts:

```
mysql> update host set name='localhost',
online=1, port=8101 \
 where id=1;
mysql> update host set name='localhost',
onlne=1, port=8099 \
 where id=2;
```

All that is left now is to copy the webserver data. The file midgard-root.php3 specifies the basic structure of a Midgard page and should be moved into its own directory:

```
mkdir -p /usr/local/midgard/root
mv /usrlib/apache/midgard-root.php3 \
/usr/local/midgard/root/
```

Using *cp -r htdocs /usr/local/midgard/* we can then copy the graphics for the pre-installed Midgard sites and start the Apache server again, but that's it. All being well, you should find sample website *http://localhost:8099* and an administration website at *http://localhost:8101*.

Midgard expansions in the httpd.conf

```
LoadModule midgard_module libexec/mod_midgard.so
LoadModule php3_module libexec/libphp3.so
LoadModule midgard_module libexec/mod_midgard.so

AddModule mod_php3.c
AddModule mod_midgard.c

AddType application/x-httpd-php3 .phtml .php3 .php

DirectoryIndex ndex.html index.shtml index.php3 \
  index.phtml index.php

MidgardEngine on
MidgardRootfile /usr/local/midgard/root/midgard-root.php3

<Directory /usr/local/midgard>
Order allow,deny
Allow from all
</Directory>

<Directory /usr/local/midgard/oot>
require valid-user
AuthName Midgard
AuthType Basic
</Directory>

NameVirtualHost localhost:8101
Listen localhost:8101
<VirtualHost localhost:8101>
ServerName localhost
Port 8101
DocumentRoot /usr/local/midgard/htdocs
</VirtualHost>

NameVirtalHost localhost:8099
Listen localhost:8099
<VirtualHost localhost:8099>
ServerName localhost
Port 8099
DocumentRoot /usr/local/midgard/htdocs
</VirtualHost>
```

Instllation of the Midgard database

```
$ tar xvzf midgard-data-1.2.5.tar.gz
$ cd midgard-data-1.2.5.tar.gz
$ mysqladmin create midgard
$ mysql midgard < midgard.sql
$ mysql mysql
mysql> insert into user (host, user, password) \
 values ('localhost', 'idgard', password('midgard'));
mysql> insert into db (host, user, db, select_priv, \
  insert_priv, update_priv, delete_priv) \
  values ('localhost', 'midgard', 'midgard', 'y', 'y', 'y', 'y');
mysql> flush privileges;
```

For access to the latter you need to enter a user name of "admin" and a password of "password".

Now the fun starts

To show you how the whole thing works, let's create a simple dynamic site – a Midgard Club homepage. Our "Club for Midgard Users", CMU for short (you can guess why we didn't opt for Midgard User's Club as a name), wants to provide general information about itself, news about the club and the dates of special events.

The club's committee are an impatient lot, so let's get straight down to business and use our browser to call up the Midgard admin-site, *http:///localhost:8101*. As you'll see, navigation is easy and detailed information just about everything is available (see Figure 1).

We'll start by setting up some users – this is easily done using the "New Person" option. Next we'll set up a new group for the administrators of the site using the "Group Administration" option. Having selected this option, clicking on "New group" displays an input form in which we can enter information about the administrator group. As a name, we'll select "CMU-Admis", and at the bottom, in the member list, we can allow any or all the people we set up previously who we want to act as administrators.

Since the layout defines important elements of our new club site, the next thing to do is establish a main style that will apply for the entire website. This is made easy for us thanks to the "New top level style" option in "Layout Administration". We only have to give it a name, so it seems sensible to select "CMU-Layout".

Now we need to define a new host for the application logic. To do this, we go to "Host Administration" and click on "New host". Let's name our host "CMU-Site" and specify that it should be accessible on local host, Port 8110. We set the status to "online" and the authentication to "not necessary". As owners of the site we also select the "CMU-Admins" and the style "CMU-Layout". The "Root Page", in other words the start page for our application, is created from new with "new root page". This then also gets named "CMU-Site".

We also, of course, have to attach this new host to the Apache server. To do this, before restarting the server, edit /etc/httpd/httpd.conf as follows:

```
NameVirtualHost localhost:8110
Listen localhost:8110
<VirtualHost localhost:8110>
ServerName localhost
Port 8110
DocumentRot /usr/local/midgard/vnm
</VirtualHost>
```

We'll also want to change the sample information pre-configured in the Root Page, and to do this we just click on the "Modify" link. Here we'll change the title to "Welcome to the CMU!" and enter the content from Listing 1.

Now we come to the content; on the starting page, represented by the Midgard Root Page, we offer links to club info, news and events. All these contents can be filed, thanks to the comprehensive data model, as normal Midgard articles. Midgard organises articles hierarchically into "Topics" and in this way builds a contents tree. Let's create such a tree called "CMU-Content" owned by 'CMU-Admins" in "Content Administration" using the "New top level topic" option. We can then set up the three main topics – "Info", "News" and "Events", as a New subtopic" under "CMU Content". Other settings can be kept the same as the parent topic using the "same as parent" option. At this point we can start filling the individual topics using the "New article" option. In the article fields, bear in mind that the title entered will later appear as the title of the HTML page, but the name has merely internal significance. For the articles on events, the fields "Start date" and "End date" can be used in order to show the start and end of the event. By entering to or three articles per topic, you can get yourself a bit of practice in handling the administration interface.

All we need do now is program the application logic, which ensures that our carefully updated contents also appear in the appropriate places of the website.

To this end we'll set up the sub-pages to which the links from the root page refer to, in each case as a "New subpage". As a name, the respective link name should be entered "info", "news" or "events". For the title we can enter meaningful phrases, and again we can inherit the style from the

Listing 1: Starting page

```
<h1>Welcome to the CMU!</h1>

<p><a href="/info">info</p>
<p><a href="/news">news</p>
<p><a href="/events">events</p>
```

Listing 2: Display of current events

```
<?
if ($article) {
?>
  <h1>&(article.title);</h1>
  <p>&(article.alcalendar);<p>
  <p>&(article.abstract);</p>
  &(article.content:h);

<?
} else {
?>

  <h1><(title)></h1>

  <!– Sort events by calendar date –>
<?
  $calendar = mgd_list_topic_calendar_all(17);
  if ($calendar) {
    while ($calendar->fetc()) {
?>
      <p>
      &(calendar.acalendar);<br>
      <strong><a
href="&(calendar.id);.html">&(calendar.title)
;</a></strong><br>
      &(calendar.abstract);
      </p>

<?
    }
  }
}
?>
```

parent site. As we want to use some PHP for these pages, in order to read the events from the database, for example, we must set the type to "active".

With us so far? Good. Now, using the events page, we can learn about programming with Midgard: The content of the events page "events" is to be taken from Listing 2.

In this code, a check is first made as to whether the object is defined as an $article or not. If it is, some HTML is created from its fields "title", "alcalendar" and "abstract", in order to display the appropriate article. "&(object.field);" is the Midgard syntax for quickly calling up information on a Midgard object. In article.content ":h" has also been appended in order to show that this field contains HTML, which must be interpreted by the browser. Without an ":h" the field content would be displayed word for word, including the HTMLtags.

On opening the "Events" page the user should be given a list of events to choose from, so he has yet to select any specific articles. In this situation the else-branch is executed. Next, the page title, as entered in the host administration, is displayed. After that all calendar contents are queried under the topic "Events":

```
$calendar = mgd_list_topic_calendar_all(17);
```

Listing 3: code-init

```
<?
if ($argc == 1) $id = $agv[0];
if ($id && !mgd_is_article_in_topic_tree(17,?
$id)) $id = 0;
if ($id) $article = mgd_get_article($id);
if ($article) $author  =
mgd_get_person($article->author);
?>
```

The parameter "17" is the numerical identifier of the topic "Events". Such identifiers are created when topics and articles are created and therefore differ depending on the user. The correct identifier can be found by displaying the corresponding topic in the content administration.

With the while loop, we now get $calendar to show us articles. Using $calendar->fetch we then call the next article and output its "acalendar" (date) and "title" fields, using a link to an URL consisting of its ID with the suffix ".html".

So, if the user clicks on "/events", on the link to the event with the ID "22", the page "/evnts/22.html" will be queried. Midgard reacts to this by again calling the active content of the page "events", but this time with the additional parameter "22". The next step is therefore to ensure that article No. 22 is retrieved and its data is filed in the object $article mentioned above. We file the logic necessary for this in a page element of the

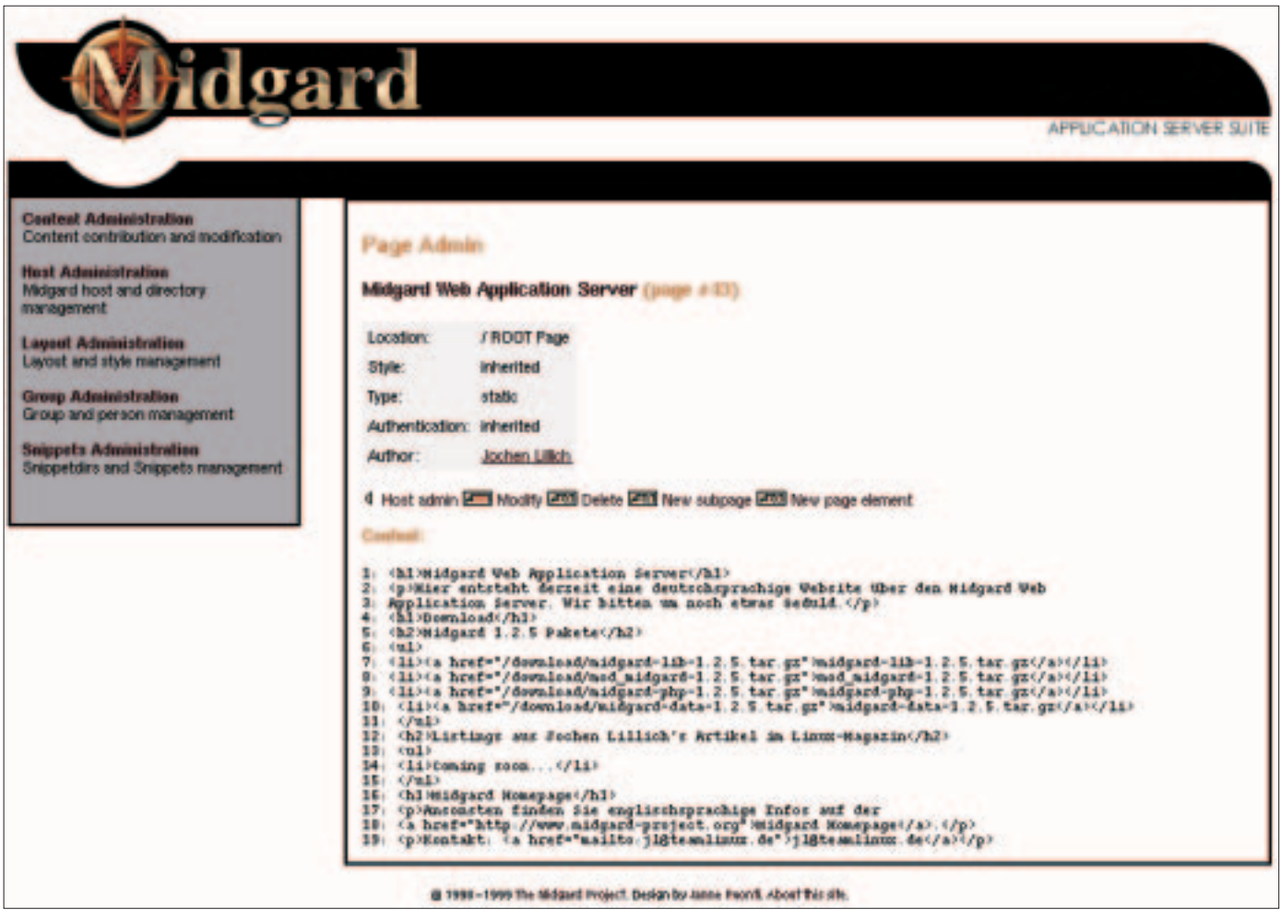

Figure 1: Simple administration by web browser

site "Events" named "code-init". This page element is always executed before the actual page content is dealt with (see midgard-root.php3) and can therefore pre-process $article. To store the page element we click on the Host Administration of the page /events on "New page element" and enter the title "code-init" and the content from Listing 3.

First we check whether a parameter is available and store this in $id. For security reasons a check is then made as to whether this ID does in fact belong to an article from the part-tree "Events". If everything is correct, the function *mgd_get_article* ensure that its data is retrieved and stored in the object $article. The data on the article's author is filed in $author.

And that's it! A visitor to our site will find a list of our events and at a click of a button can get further information. The calendar logic of Midgard, by the way, automatically drops events on dates prior to the current one. So now all we'll ever have to do is regularly update the events using content administration, so that the homepage of our club is always able to provide the latest information.

The pages for information and news of our club can be created in a similar manner to the events page but will use a different topic ID.

As you can plainly see, developing dynamic websites with Midgard is really easy: Once you get used to web administration, creating a website from database contents and a separate layout will take you very little time. And if topic or article structure provided suffices for your own purposes, you won't even have to worry about even one single SQL instruction, sidestepping a potential trouble-spot.

Having said we've finished, though, we have to admit that our site is a bit lacking in style. This is what Layout Administration is for (Figure 2), though, and we'll leave it to you to create something individual for your CMU test site. In doing so, note that layout can, starting from the ROOT" element, be broken down into building blocks known as „Style elements". Subordinate elements are then simply integrated in the form *<[element]>*.

Midgard evolution

In order to prevent Midgard's data model from acting as a restriction, the Midgard development team is currently working on a data model that can be expanded as required. The current developer versions already contain support for inary data (BLOBs) and sophisticated calendar management (for example for assigning people to events).

If several Midgard hosts are running on one server they must each be administered separately. After all, a host administrator or even a customer should only be able to access his own data. With the site group concept, which made its debut in Version 1.4 beta, this is now possible without any problems.

Another task that the Midgard developers are working on is a universal database interface. While Midgard at present only supports mySQL, for Version 2.0 of the application the choice is expected to be extended to cover a wide variety of databases and even other backends, such as LDAP.

This is all very exciting, and also very useful, so this Web Application Server undoubtedly has a very interesting future ahead of it. ■

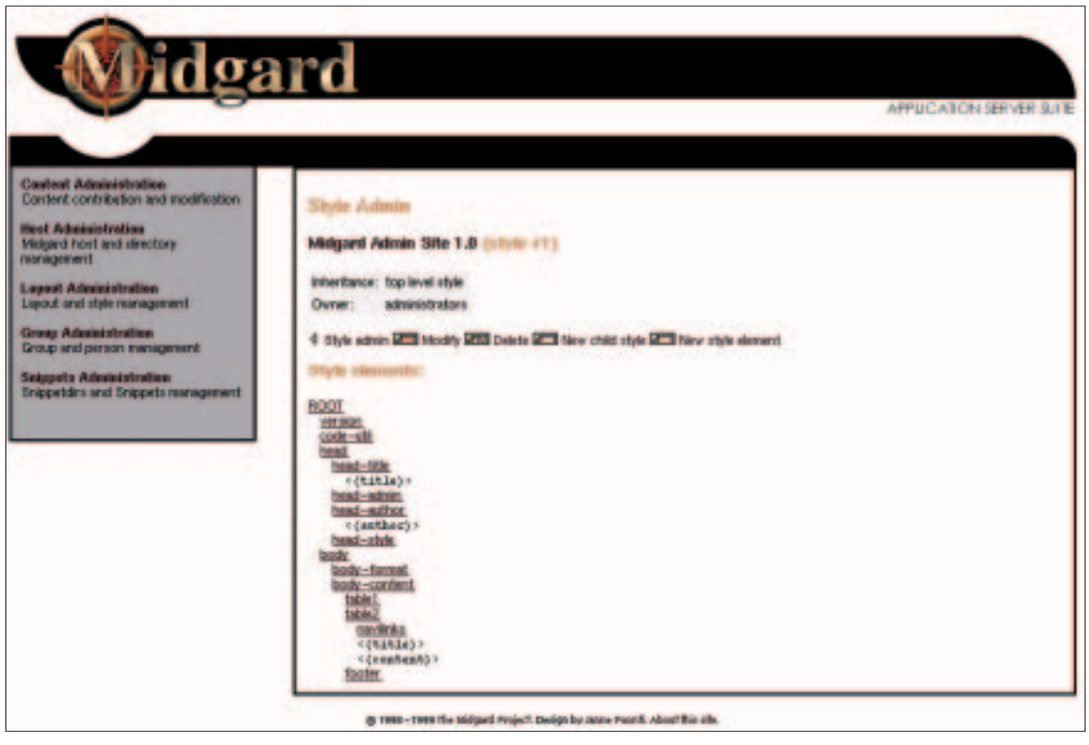

Figure 2: A wide range of layout styles can be used with Midgard