

Harald Radi ([harald.radi@nme.at](mailto:harald.radi@nme.at)) - <http://www.nme.at>

# 1 Creating Multi-Tier Web Applications with PHP

*PHP is attractive as a language because of its simplicity and ease of learning, in return PHP lacks important features needed for building complex web applications. These are mostly required when following the multi-tier pattern to achieve a separation of presentation, logic and data storage and to reuse already existing interfaces of your information system. Because of its straightforwardness PHP is best suited for building the presentation layer of a web application. The extensions bundled with PHP enable you to integrate existing components that are available as Java classes, CORBA- and (D)COM components or .NET assemblies. Another advantage is that most of these component models already provide various frameworks for transactions and messaging between processes and furthermore message queuing if these processes are not running at the same time. Efficient process communication can therefore be realised with very little effort.*

## 1.1 Web Applications and Multi-Tier Architecture

Due to the fact that each software manufacturer will interpret these terms to its own benefit to try and make its product as buzzword compliant as possible, you will have difficulty in finding a widely accepted definition for these terms. Therefore we should first define what we consider to be a web application.

A web application consists of pages not only linked with each other but furthermore interacting with each other by keeping an inner state. In the simplest form this state can be kept as a session using PHP's built in session management. As a result you will get an application like set of web pages which will behave differently depending on your actions (e.g. an enrolment process, a shopping cart, etc.).

You will soon however come across the limitations of using only sessions:

- The lifetime of session data is limited to the duration of the user session plus some threshold, thus you cannot persist data between different sessions of the same user.
- Moreover you cannot store application wide data which is accessible from every session and without additional measures session data cannot be shared among servers of a web farm hence those web applications don't scale very well.
- Even the structure of such applications is not very transparent; a newly added script might store data in the session memory that overwrites existing data because of the lack of knowledge of data structures in other scripts.

One way to circumvent these limitations is the use of a relational database management system (RDBMS). Application data can be stored persistently in the database and is available across all user sessions. All data structures are described by the database schema and data integrity is checked by the RDBMS. Changing or adding scripts cannot corrupt data accidentally as all the data structures are well defined.

The application can now be deployed on a web farm because the data is kept separate. Separating the data out also greatly improves security because the RDBMS can be secured more restrictively by the firewall, preventing direct access from outside the server network. Keeping the data centralised in an RDBMS also opens further possibilities such as running monthly turnover reports or tracking user habits. As you surely would have guessed this is a two-tier web application, we have an application layer and a data layer.

As soon as more people get involved in software development things start getting more complicated. You will soon switch to Smarty or other templating systems to keep code and layout separated as much as possible. Developers can work on their code without influencing the layout and designers can change the layout without having to worry about breaking the scripts. The former application layer is split into a presentation layer and a business logic layer respectively representing templates and scripts.

With increasing complexity of your web application you will encounter several other limitations. Functionality starts influencing the layout too much and critical operations cannot be efficiently implemented in a loosely typed scripting language. If you deal with financial transactions or other critical scenarios, an aborted script will most likely leave an inconsistent state or some uncommitted transactions. If you have to deal with third-party systems and other servers on the Internet, you cannot always catch those situations with standard database integrity checks but have to check transaction results. If one of your scripts fails for some reason a result could be misinterpreted and a bogus action will be initiated. As a consequence you should keep your scripts as short and simple as possible to reduce the risk that it will quit or break in the middle of a transaction. Long and complex scripts are much more vulnerable to bogus user input and other unpredictable side effects and should therefore be avoided.

The next logical step after separating the data out of the web application is to remove all the logic and functionality. This might sound odd at first glance but you will soon see the advantages.

Your scripts will drastically lose complexity and the risk of breaking something will be dramatically reduced. All logical operations are atomic from the scripts point of view as they are API calls now. The script can terminate either before or after a transaction is done, but not in between. The potential points of vulnerability are reduced to a minimum. You can use a strongly typed language to implement the functionality gaining even more robustness.

## 1.2 Component Models

The first approach would be writing a PHP extension in C or C++ that exports a few functions to the PHP script scope and handles all the critical logic. This isn't as ideal as it sounds because you will have to maintain and recompile your extension with every new version of PHP. If you depend on other third-party libraries you will also have to recompile your module if they change. You will possibly spend more time maintaining the extension than benefiting from it.

Given that most third-party libraries are shipped with an interface accessible by one of the various component models it would be preferable that you can use it directly from PHP like from any other language that supports the specific model. You could even write your own components and use them from your PHP scripts and any other application you need them for.

PHP supports various modern component models like (D)COM, .NET, JavaBeans, CORBA, SOAP and Bananas. The task of a script is no longer implementing functionality but using existing components and plugging them together in the right order.

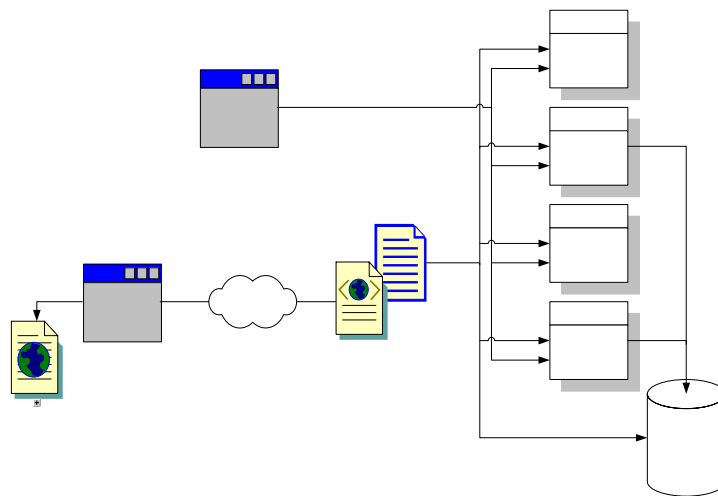


Fig. 1: (Re)Using components to build a web application

Using components in software development gives you much more flexibility than developing ordinary software products. As shown in Fig. 1 you can build thin as well as thick clients on top of your components without any additional effort. You can provide a web interface, an application interface and an interface for handhelds or mobile phones as front ends for your components.

The number of components and different kinds of clients is not limited in any aspect; the number of layers is unlimited. In this case we are talking of multi-tier web applications.

Components are usually instantiated once and keep running as a service which means that you don't have the overhead of loading everything for each request if you use PHP via CGI. Beyond that most component model systems provide facilities to distribute the load among different servers which means that components automatically scale very well.

Components are only accessible through an interface that provides a "black box" view to the programmer. Each component can be tested and deployed on its own which eases tracking down bugs greatly.

### 1.2.1 Java

#### Concept

The Java Standard Edition only contains a component model called JavaBeans. JavaBeans is not really a component model like others we will see later, but because of Java's ability to interface with almost every component model due to its extensive libraries, it is still worth discussing here.

The PHP Java extension enables you to instantiate arbitrary Java classes and to invoke methods on their objects. PHP has to start a Java Virtual Machine (JVM) in order to communicate with the Java object. The JVM is invoked through the Java Native Interface (JNI) and runs inside the PHP process or the web server's process if PHP is loaded as a server module.

```
<?php
    // instantiate a java class
    $system = new Java("java.lang.System");

    // call methods
    $java_version = $system->getProperty("java.version");
    $java_vendor   = $system->getProperty("java.vendor");
    $os_name       = $system->getProperty("os.name");
    $os_version    = $system->getProperty("os.version");

    echo "Java $java_version, $java_vendor \n";
    echo "running on $os_name $os_version \n";
?>
```

Lst. 1: Instantiating a Java class

The above code listing illustrates how the Java extension maps Java classes into the PHP namespace. Though they seem to be well integrated like ordinary PHP classes there are still a few limitations:

- Java is case sensitive, PHP is not; the Java extension tries to find the correct method by comparing the whole signature but if two functions only differ by case PHP cannot distinguish them.

- Parameters are not passed by reference, thus if the Java method changes one of the parameters PHP won't reflect that change.
- The Java extension always creates an instance of a specified class; you cannot just access a static member without instantiating the class. There is one exceptional case where this rule doesn't apply: If the specified class doesn't have a public constructor then the Java extension will return the objects Class object and you can access all the static members. This way you can call factory methods that return the actual instance of a class.
- You cannot call functions that expect an array of a native data type such as int[], char[], etc.

If you know these limitations you will see that avoiding them and working around them is not difficult. The more interesting things are how to use enterprise level services such as accessing remote objects and using the messaging service.

### Remote Method Invocation (RMI)

RMI is a mechanism to access Java objects that reside in a separate virtual machine and thus probably in another physical machine. If you request an instance of a remote object your JVM transparently loads a stub from the remote object and creates a proxy object. This proxy object can be used the same way as a local object save for the fact that if you pass objects as parameters to methods of the remote object these objects have to be serialisable.

```
<?php
    // instantiate the necessary classes
    $system = new Java("java.lang.System");
    $naming = new Java("java.rmi.Naming");
    $security = new Java("java.rmi.RMISecurityManager");

    // init the security manager
    $system->setSecurityManager($security);

    // request the remote interface
    $remote = $naming->lookup("//host/remote/interface");
    // invoke a remote method
    $result = $remote->method();
?>
```

Lst. 2: Accessing a remote object

You do of course have to set up your Java infrastructure accordingly so that the above sample will work.

### Java Messaging Service (JMS)

The Java Messaging Service provides a way to let different applications communicate asynchronously. Using a messaging service differs from conventional messaging by the anonymity between producer and consumer. Messages are kept in a queue managed by the messaging service and the consumer can fetch them whenever it is ready for processing. The consumer is totally decoupled from the producer; it doesn't even know how many producers there are.

From a web applications point of view this is a very interesting mechanism because you can put all transactions into a queue and batch process them later on. If you perform financial transactions or orders it might be cheaper to commit them as a batch because you usually pay per transaction regardless how many records the transaction contains.

If this situation changes you can simply run the batch processing task more often (even per message), without affecting the web application.

There are two different messaging models, the point-to-point model and the publish/subscribe model:

- The point-to-point model is used if there is only one consumer. The messages stay in the queue until the consumer fetches them; there is no need for the consumer to be running when the message arrives. Fetching can be done either synchronously or asynchronously.
- The publish/subscribe model is used when there are multiple consumers. The consumers announce their interest in a particular kind of message and each time such a message is published by a producer all the consumers will be notified. The consumers have to be running in order to be notified, messages will not be kept in a queue.

```
<?php
$system = new Java("java.lang.System");
$system->setProperty("jms.properties",
                    "/path/to/jms_client.properties");
$jndi = new Java("javax.naming.InitialContext");

/* QueueConnectionFactory ... point-to-point model
 * TopicConnectionFactory ... publish/subscribe model
 */
$queue_factory = $jndi->lookup("QueueConnectionFactory");
$queue = $jndi->lookup("myQueue");

/* replace 'Queue' with 'Topic' in the method calls
 * if you are using the TopicConnectionFactory
 */
$connection = $queue_factory->createQueueConnection();
$session = $connection->createQueueSession(FALSE, 1);
```

```
$sender = $session->createSender($queue);

$message = $session->createTextMessage();
for ($i=0; $i < 2; $i++) {
    $message->setText("message $i");
    $sender->send($message);
}

$message = $session->createObjectMessage();
for ($i=0; $i < 2; $i++) {
    $message->setObject($i);
    $sender->send($message);
}

$connection->close();
?>
```

Lst. 3: Using the Java Messaging Service

### 1.2.2 (D)COM

(D)COM is a Windows proprietary component model not available on other platforms. PHP provides a functional as well as an object-orientated API to access COM components. The object-orientated API is more consistent so I therefore recommend using it instead of the functional API.

Loading COM components basically works like instantiating Java classes, if you want to instantiate DCOM components you have to pass a few more parameters to the constructor.

```
<?php
    $obj = new COM("ProgId", "server.test.org");

    $obj->method();
?>
```

Lst. 4: Instantiating a DCOM component

Like in Lst. 4 PHP would authenticate to the DCOM server using its current security token (most likely the web servers account), but there might be situations where you want to authenticate using a different account like in Lst. 5.

```
<?php
    $obj = new COM("ProgId", array(
        "Server" => "server.test.org",
        "Username" => "user",
        "Password" => "xxx",
        "Flags" => CTX_REMOTE_SERVER));

    $obj->method();
?>
```

### Lst. 5: Using a different security token

The COM extension also contains a VARIANT class to pass parameters by reference or to set special parameter types like date, money, etc.

```
<?php
    $var = new VARIANT(); // create an empty variant container
                        // which will be able to hold the
                        // result of a by-reference call

    $obj->call($var); // of course $obj must be a valid
                    // com instance

    echo $var->value; // in case the previous call passed
                    // back a value, it will be stored
                    // in the value member, otherwise it
                    // will be NULL
?>
```

### Lst. 6: Using the VARIANT class

## 1.2.3 .NET

PHP's .NET extension only servers as a wrapper to the COM extension in order to allow loading .NET assemblies. Everything that applies to the COM extension also applies to the .NET extension.

## 1.2.4 CORBA

The CORBA PHP extension is not included in the standard PHP distribution, but is available for download at <http://universe.2good.nu/>. The extension is called Universe and is available for Windows as well as Unix. The API doesn't differ very much from the other extensions described so far.

```
<?php
// available untangled at http://www.random.org/Random.ior
$ior = "IOR:000000000000000f49444c3a52616e646f6d3a312e30".
      "0000000000010000000000000050000100000000016706c".
      "616e7874792e6473672e63732e7463642e69650006220000".
      "002c3a5c706c616e7874792e6473672e63732e7463642e69".
      "653a52616e64646f6d3a303a3a49523a52616e646f6d00";

$obj = new UniverseObject($ior);
$value = $obj->lrand48();

echo "random value: $value\n";
?>
```

Lst. 7: Using CORBA objects

### 1.3 Outlook

With the upcoming version of PHP including the new Zend Engine 2 and its much improved object model there will also be improvements in the extensions described here. Besides making use of the new features like support for exceptions and faster resource management the extensions will be based on a unified abstraction layer providing a consistent API (functional as well as OO) and faster execution due to caching of method signature lookups.

The aim is to combine the critical parts of all those extensions into one code base and reuse the code wherever possible. This step will greatly improve stability and thread safety on one hand and will reduce possible memory leaks on the other hand.