

# Creating Multi-Tier Web Applications with PHP



Harald Radi <harald.radi@nme.at>

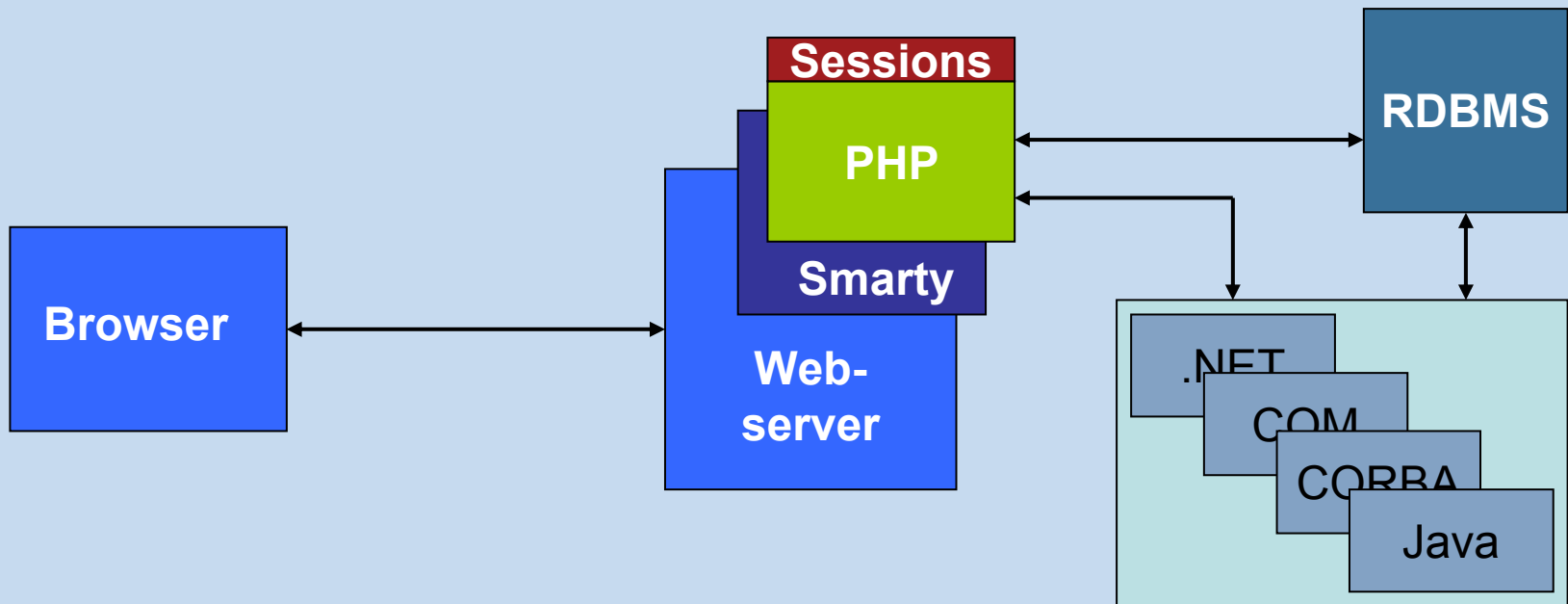
PHP – Conference 2002

# Overview



- Multi-Tier Architecture
- Component Models
- Java
- (D)COM
- .NET
- CORBA
- Outlook (PHP5)

# Multi-Tier Architecture (1)



# Multi-Tier Architecture (2)



## Advantages - RDBMS

- Data is well defined by a Database Schema
- Adding scripts can't corrupt data accidentally
- RDBMS takes care of data integrity
- Improved security and availability
- User specific permissions and data views
- Data is available for other applications (e.g. monthly turnover reports)

# Multi-Tier Architecture (3)



## Advantages - Template System

- Separate layout
- Designer and developer can work independently
- Different templates for each browser version

# Multi-Tier Architecture (4)



## Advantages – Using Components

- Operations are atomic from the scripts point of view
- Scripts only serve as a glue between the different components
- Combines the robustness of strong typed languages with the ease of scripting

# Component Models



- Using components instead of custom PHP extensions your reduces maintenance effort
- PHP ships with Java, COM, CORBA (satellite) and .NET extensions
- PHP also supports SOAP and CORBA (universe) through third party extensions not bundled in the PHP distribution

# Java (1)



- Java's component model JavaBeans is not a real component model
- PHP's Java extension enables you to instantiate arbitrary Java classes
- Java objects can be used like ordinary PHP objects
- All of Java's extensive libraries and third party products can be used

# Java (2)

```
<?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");
?>
```

# Java (3)



## Limitations

- Java is case sensitive, PHP is not
- Parameter are not passed by reference
- Native type arrays are not supported
- A class is always instantiated whether you only want to access static members or not
  - Except if the class doesn't have a public constructor, the Class object is returned in this case

# Java (4)

java.util.Calendar does not have a public constructor

```
<?php
    $class = new Java("java.util.Calendar");
    $calendar = $class->getInstance();

    $millis = $calendar->getTimeInMillis();
    echo date("m/d/y h:m:s", $millis / 1000);
?>
```

# Java (5)

## Exception Handling

```
java_last_exception_get()
```

```
java_last_exception_clear()
```

## Exceptions can be caught by

- Using '@' to disable warnings and check for an exception after each function call
- Overloading the error handler

# Java (6)

```
<?php
    $php = new Java("PHPJavaTest");
    @$php->throwException();

    $exception = java_last_exception_get();
    echo $exception->getLocalizedMessage();

    $callstack = $exception->getStackTrace();
    foreach ($callstack as $element) {
        echo $element->toString();
    }

    java_last_exception_clear();

?>
```

# Java (7)

```
<?php
function exception_handler($errno, $errstr,
                           $errfile, $errline)
{
    $exception = java_last_exception_get();
    echo $exception->getLocalizedMessage();
    java_last_exception_clear();
}

set_error_handler("exception_handler");

$php = new Java("PHPJavaTest");
$php->throwException();

?>
```

# Java - RMI (1)



## Remote Method Invocation (RMI)

- RMI is a mechanism to access Java objects living in a different Virtual Machine (VM)
- The calling VM loads a stub and dynamically creates a proxy object
- Such proxies can be used like local objects
- Objects passed as parameters have to be serialisable

# Java - RMI (2)

```
<?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();
?>
```

# Java - JMS (1)



## Java Messaging Service (JMS)

- Lets applications communicate asynchronously
- Provides anonymity between producer and consumer
- Producer and consumer are totally decoupled, they even don't have to run at the same time

# Java - JMS (2)



- Point-to-Point model
  - Only one consumer
  - Messages stay in a queue till they are fetched
  - Messages can be fetched anytime
- Publish/Subscribe model
  - Multiple consumers
  - Consumers announce their interest in a topic
  - Consumers have to be running to receive a message

# Java – JMS (3)

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

    $queue_factory = $jndi->lookup("QueueConnectionFactory");
    $queue = $jndi->lookup("myQueue");

    $connection = $queue_factory->createQueueConnection();
    $session = $connection->createQueueSession(FALSE, 1);
```

# Java – JMS (4)

```
$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);  
}
```

# Java - JMS (5)

```
$receiver = $session->createReceiver($queue);

for ($i=0; $i < 2; $i++) {
    $message = $receiver->receive(1);
    echo $message->getText();
}

for ($i=0; $i < 2; $i++) {
    $message = $receiver->receive(1);
    echo $message->getObject();
}

$connection->close();

?>
```

# (D)COM (1)



- (D)COM is a Windows proprietary component model
- PHP provides a functional and an object-orientated API
- Using COM components basically works like using Java classes

# (D)COM (2)

- InProc server
  - Functions are loaded into the executing processes memory
  - Shipped as .dll
  - Not easily usable with DCOM
- OutProc server
  - Components run as a separate server
  - Shipped as .exe
  - Can be used with DCOM

# (D)COM (3)

```
<?php
    $obj = new COM("ProgId");
    $obj->method();
?>
```

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

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

# (D)COM (4)

`com_load()`

creates a new instance

`com_get()`

gets the value of a property

`com_set()`

sets the value of a property

`com_invoke()`

invokes a method

`com_addref()`

call the components `AddRef()`

`com_release()`

call the components `Release()`

`com_load_typelib()`

loads constants from a typelib

`com_isenum()`

checks for `IEnumVARIANT`

# (D)COM (5)

```
<?php
    $fs = new COM("Scripting.FileSystemObject");
    $drives = $fs->Drives;

    // possibility one
    $drivearray = $drives->Next($drives->Count);
    foreach ($drivearray as $drive) {
        echo $drive->DriveLetter;
    }

    $drives->Reset();

    // possibility two
    while ($drive = $drives->Next()) {
        echo $drive->DriveLetter;
    }
?>
```

# (D)COM (6)



## Type libraries

- Are interface definitions for the component
- Give information about return value and parameter types
- Can declare enumerations and parameter default values
- Might contain metadata like additional descriptions and help IDs

# (D)COM (7)

## php.ini COM section

```
[com]
com.autoregister_casesensitive=false
com.autoregister_typelib=true
com.typelib_file=c:\typelib
com.allow_dcom=true
```

## Example typelib file

```
MSMQ.MSMQQueueInfo
Microsoft WMI Scripting V1.1 Library #cis
{0BAC5750-44C9-11D1-ABEC-00A0C9274B91}
```

# (D)COM – VARIANT (1)

The `VARIANT` class allows you to bypass PHP's automatic variable conversion and pass a variable of a specific type.

```
<?php
    $var = new VARIANT(123.45, VT_DATE);
?>
```

You can even specify the type of array values which by default would be `VT_ARRAY|VT_VARIANT`

# (D)COM – VARIANT (2)

Predefined members are

- >value            containing the actual value
- >type            containing the variant type  
                    as a VT\_\* constant

You can also set a value using the c/c++ member names (e.g. bVal, iVal, lVal, fltVal, dblVal, boolVal, cyVal, date, ...)

# (D)COM – VARIANT (3)

Unlike the Java extension, the COM extension supports passing variables by reference

```
<?php
    $var = new VARIANT();

    $obj->call($var);

    echo $var->value;
?>
```

# (D)COM - Monikers

Monikers are used to access a specific instance of a component. PHP is directly able create instances from Moniker strings.


```
<?php
    $doc = new COM("C:\example.doc");
?>
```

```
<?php
    com_load_typelib("Microsoft WMI Scripting V1.1 Library");
    $wmi = new COM("WinMgmts:{impersonationLevel=impersonate}");
?>
```

# .NET

- Is basically a wrapper for the COM extension
- Implements the infrastructure to load .NET assemblies and wraps them as COM components
- Delegates everything else to the COM extension and therefore providing the same features

# CORBA (Universe) (1)



- Universe is not bundled with PHP, Satellite does not run on every platform
- Available at <http://universe.2good.nu>
- A case insensitive version of MICO is needed

# CORBA (Universe) (2)

```
<?php
// Interoperable Object Reference
$ior = "IOR:00000000000000000f49444c3a52616".
    "e646f6d3a312e300000000000001000000".
    "000000005000010000000000016706c616".
    "e7874792e6473672e63732e7463642e69".
    "650006220000002c3a5c706c616e78747".
    "92e6473672e63732e7463642e69653a52".
    "616e646f6d3a303a3a49523a52616e646".
    "f6d00";

$obj = new UniverseObject($ior);

$value = $obj->lrand48();
echo $value;

?>
```

# CORBA (Universe) (3)

## PHP as CORBA server

- It is possible to write a CORBA server in PHP
- IDL file has to be added to the local repository

```
interface PHPTestServer
{
    string version();
    string foobar(in string value);
};
```

- PHP class has to implement that interface

# CORBA (Universe) (4)

```
<?php
class PHPTestServer {
    function version() {
        return phpversion();
    }
    function foobar($value) {
        return "foo{$value}bar";
    }
}

// create server object
$object = new UniverseObject(new PHPTestServer(),
                              "IDL:PHPTestServer:1.0");
$iior = universe_object_to_string($object);
echo $iior;

universe_run();

?>
```

# CORBA (Universe) (5)

```
<?php
    $server = new UniverseObject("....."); // previous IOR

    // call our methods
    echo $server->version();
    echo $server->foobar("blah");
?>
```

# PHP 5

- Unified RPC abstraction
- Unified naming scheme
- Unified error propagation using exceptions
- Caching of method lookups
- Object pools
- Real singletons
- Serialisable instances
- Case sensitive properties (?)

# End



<http://www.nme.at>