Component and Event-Driven Architectures

Building Flexible Applications in PHP5

Stephan Schmidt, 1&1 Internet AG

international PHP2005 conference - spring edition -

Agenda

- Desktop vs. Web Applications
- Components in Web Development
- ASP.NET
- Java Server Faces
- PHP Frameworks
- Events in Web Development
- PEAR::Event_Dispatcher
- patPortal

The speaker

- Working for 1&1 Internet AG
- Founding member of PHP Application Tools (pat)
- Active developer of the PEAR community
- Writing for several PHP magazines

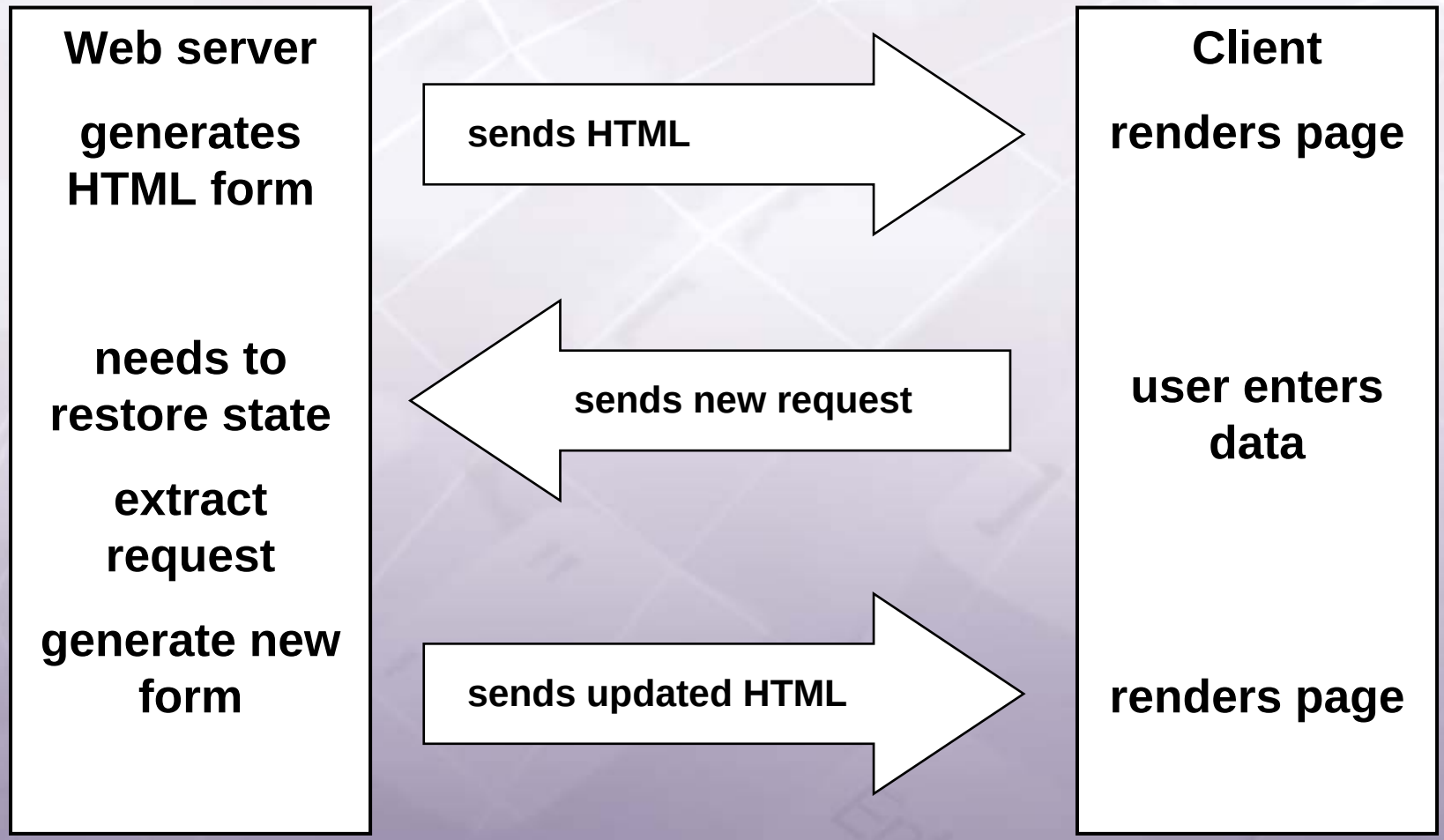
Desktop Applications

- Interactive User Interfaces
- Consist of components
  - Menus
  - Toolbars
  - Input fields
- Event-based
  - Events are triggered by components
  - Handled by listeners

Web Applications

- static HTML front end
- Interactivity using JavaScript
- No persistent connection between front end and application
- Applications are stateless

Web Applications



Web Applications

- 50-75% of the work done in web applications is related to generating the frontend or parsing/validating request data.
- a lot of repetitive work
  - Fetch user submitted variables from $_REQUEST
  - Validate the data
  - Generate `<input/>` tags with submitted values

Web Components

A component is an object written to a specification .

- atomized part of a website that fulfills exactly one task
- Multiple-use
- Non-context-specific
- Implements an interface
- Composable with other components

Web Components

Examples for Components:

- User Interface Components
  - Button, Input field, Checkbox
- Data Storage Components
  - Beans, ORM
- Components that contain logic
  - Validators, Converters

Web Components

User Interface Components

- Represent one element in the GUI
- Are able to render themselves
- Extract their state from the HTTP-Request or session
- Able to validate their state (only in some frameworks)

The ASP.NET Way

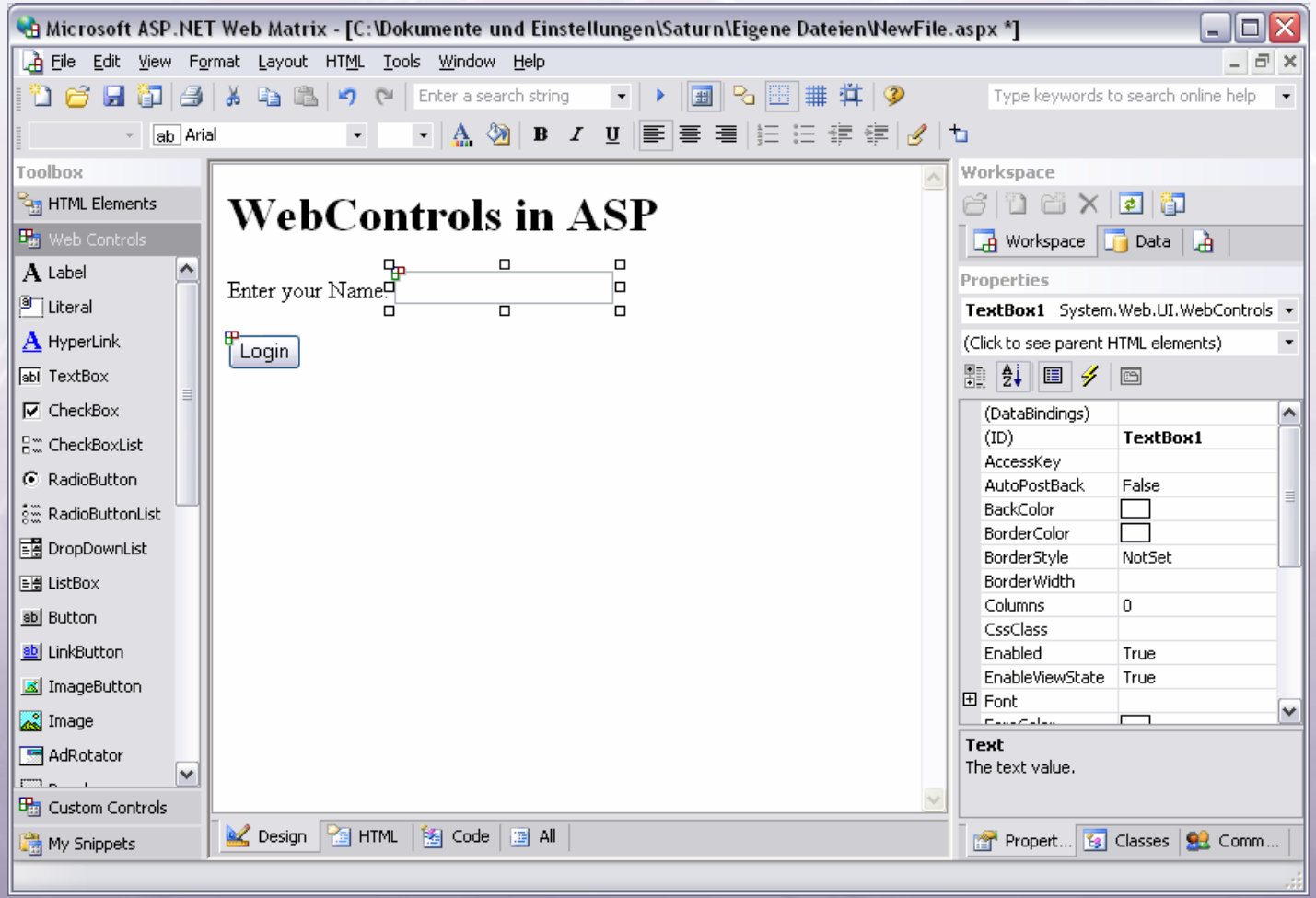
- Components (called web-controls) are embedded into HTML
- Uses standard HTML-Tags plus some ASP attributes as well as namespaced XML tags

```
<form runat="server">
  <asp:Button id="Button1" runat="server"
              Text="Click Me">
  </asp:Button>
</form>
```

The ASP.NET Way

- Designers do not need to touch the actual code
- Designers have full control over the page layout
- Microsoft even provides the designers a graphical editor to compose pages

ASP.Net Web Matrix



User Interaction

- Each Web Control is represented by an object on the server
- You can easily bind methods of these classes to events that are triggered on the client
- Enabling the ViewState of a control will automatically repopulate all elements with the submitted values

ASP.NET: Event Handling

```
public class WebForm1 : System.Web.UI.Page{
  protected System.Web.UI.WebControls.Button Button1;

  // Handler for onClick
  private void Button1_ClickHandler (object sender,
                                     System.EventArgs e) {
    Response.Write( "You clicked Button1" );
  }

  // register the
  override protected void OnInit(EventArgs e) {

    this.Button1.Click += new
       System.EventHandler(this.Button1_ClickHandler);
    base.OnInit(e);
  }
}
```

Java Server Faces

- Specification by Java Community Process Group (Apache, Oracle, Sun...)
- Several implementations available
- Defines a user interface component model
- Can be embedded in HTML
  - Theoretically not limited to HTML

Java Server Faces

- Splits the development process into three main tasks
  - Develop the backend (models)
  - Develop components and integration code (like event handlers)
  - Develop the user interface pages using components
- Resembles ASP.NET

JSF Example

```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsf/html"
           prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core"
           prefix="f" %>
<f:view>
  <h:form>
    Name: <h:inputText id="name" size="8"/>
    Pass: <h:inputSecret id="pass" size="8"/>
    <h:commandButton value="Login"
                     action="#{authHandler.login}"/>
  </h:form>
</f:view>
```

JSF Components

- Always consist of a "component type" and a "renderer type"
  - inputText, inputSecret
  - commandButton
- Not restricted to UI components
  - Validators, converters
- Can be bound to model properties
- support events in a similar way to ASP

Components in PHP

- Not yet a standard interface for components
- No specifications like JSF available
- Packages slowly evolve from form handling packages
- Since the release of PHP5 several frameworks have been published that provide more components

patForms

- PHP4 Open Source Project
- Abstraction for (HTML-)forms
- Each element is represented by one object (i.e. Component)
- All elements provide the same API
- All elements are able to render themselves

patForms Components

Each component is an object.

Only provides form elements, could be extended.

- String, Number, Text, File
- Checkbox, Radio, Enum, Set
- Date, Pool
- Group (acts as a container)

patForms Page Structure

patForms Example

```
$form = patForms::createForm();
$form->setAttribute('name', 'myForm');
$username = array(
    'required'    => 'yes',
    'display'     => 'yes',
    'edit'        => 'yes',
    'label'       => 'Username',
    'description' => 'Enter your username here.',
    'default'     => '',
    'maxlength'   => '15',
    'minlength'   => '4',
    'title'       => 'Username',
);
$user = $form->createElement('user', 'String',
                                    $username);
$form->addElement($user);
```

patForms Example

```
// more elements are created and added
……

$renderer = patForms::createRenderer('Array');
$renderer->
$form->setRenderer($renderer);
$form->setAutoValidate('mySubmitVar');

if ($form->isSubmitted()) {
   // check for errors
}
$elements = $form->renderForm();

// insert the elements in the HTML template
```

patForms Example

```
<form name="myForm" method="post"
      action="example.php">
  Username<br />
  <input id="pfo1" name="user" type="text"
         title="Username" value="" maxlength="15" />
  <br />
  Password<br />
  <input id="pfo2" name="pass" type="password"
         title="Password" value="" maxlength="15" />
  <br />
  <input id="pfo3" type="submit" name="save"
         value="Save form"/>
</form>
```

patForms Parser

Disadvantages of this method:

- Creating forms requires a lot of code
- Designers cannot set attributes of the components

There should be a way to embed components in HTML like ASP.NET.

patForms Example 2

```
<div>
  Username:<br />
  <patForms:String name="username" required="yes"
            description="Please enter your name here"
            size="8" maxlength="8" accesskey="U" />
  <br />
  Password:<br />
  <patForms:String name="password" type="password"
            required="yes" size="6" maxlength="6"
            accesskey="P" />
</div>
```

patForms Example 2

```
patForms_Parser::setNamespace('patForms');
patForms_Parser::setCacheDir('cache');
$form =& patForms_Parser::createFormFromTemplate(
                'SimpleRenderer',
                'templates/example_parser_simple.fhtml',
                'templates/example_parser_simple.html'
        );

$form->setAutoValidate('save');
$form->registerEventHandler('onSuccess',
                                'handleUserInput');
$form->registerEventHandler('onError',
                                'displayErrors');
print $form->renderForm();
```

Interaction with PHP

- Provides basic event management for the form
  - onSubmit, onSuccess, onError
  - any valid PHP callback
- Provides observers for all components
- Events cannot (yet) be registered in the HTML code
- DataSources can be bound to <select/> or <radio/>

patForms Example 3

```
<patForms:Enum name="heroDC" required="yes"
               label="DC-Heroes">
  <patForms:Datasource type="custom">
    <hero:getDatasource id="DC"/>
  </patForms:Datasource>
</patForms:Enum>
```

```
class Datasource_DC {
  public function getValues() {
    // open DB connection and fetch the
    // possible values from the database
  }
}
```

patForms Example 3

```
class heroHandler {
  public function getDatasource($attributes,$content){
    $dsName =  'Datasource_'.$attributes['id'];
    return new $dsName;
  }
}
```

```
$parser = patForms_Parser::createParser(…);
$heroHandler = new heroHandler;
$parser->addNamespace('hero', $heroHandler);
$parser->parseFile( 'form.fhtml');

$form = $parser->getForm();
$form->setRenderer($parser);
$content = $form->renderForm();
```

PRADO

- PHP5 Open Source Project
- Focus on components
- Very similar to ASP.NET and JSF
- Components embedded in HTML
- Page represented by an object
- Glued together using an XML configuration

PRADO Example

```
<html>
<head>
  <title>Hello, world!</title>
</head>
<body>
  <com:TForm>
    <com:TButton Text="Click me" OnClick="clickMe" />
  </com:TForm>
</body>
</html>
```

```
class HomePage extends TPage {
  public function clickMe($sender,$param) {
    $sender->Text="Hello, world!;
  }
}
```

PRADO Example

```
<?xml version="1.0" encoding="UTF-8"?>
<application state="debug">
  <request default="HomePage" />
  <session enabled="true" />
  <vsmanager enabled="true" />
  <alias name="Pages" path="." />
  <using namespace="System.Web.UI.WebControls" />
  <using namespace="Pages" />
</application>
```

```
require_once '/path/to/prado.php';

$app = pradoGetApplication('helloworld.spec');
$app->run();
```

PRADO Component

Represented by a class that extends TComponent.

Properties and events are defined in an XML specification:

```
<component>
  <property name="Text" get="getText" set="setText"
            type="string" />
  …more properties…
  <event name="OnClick" />
  <event name="OnCommand" />
</component>
```

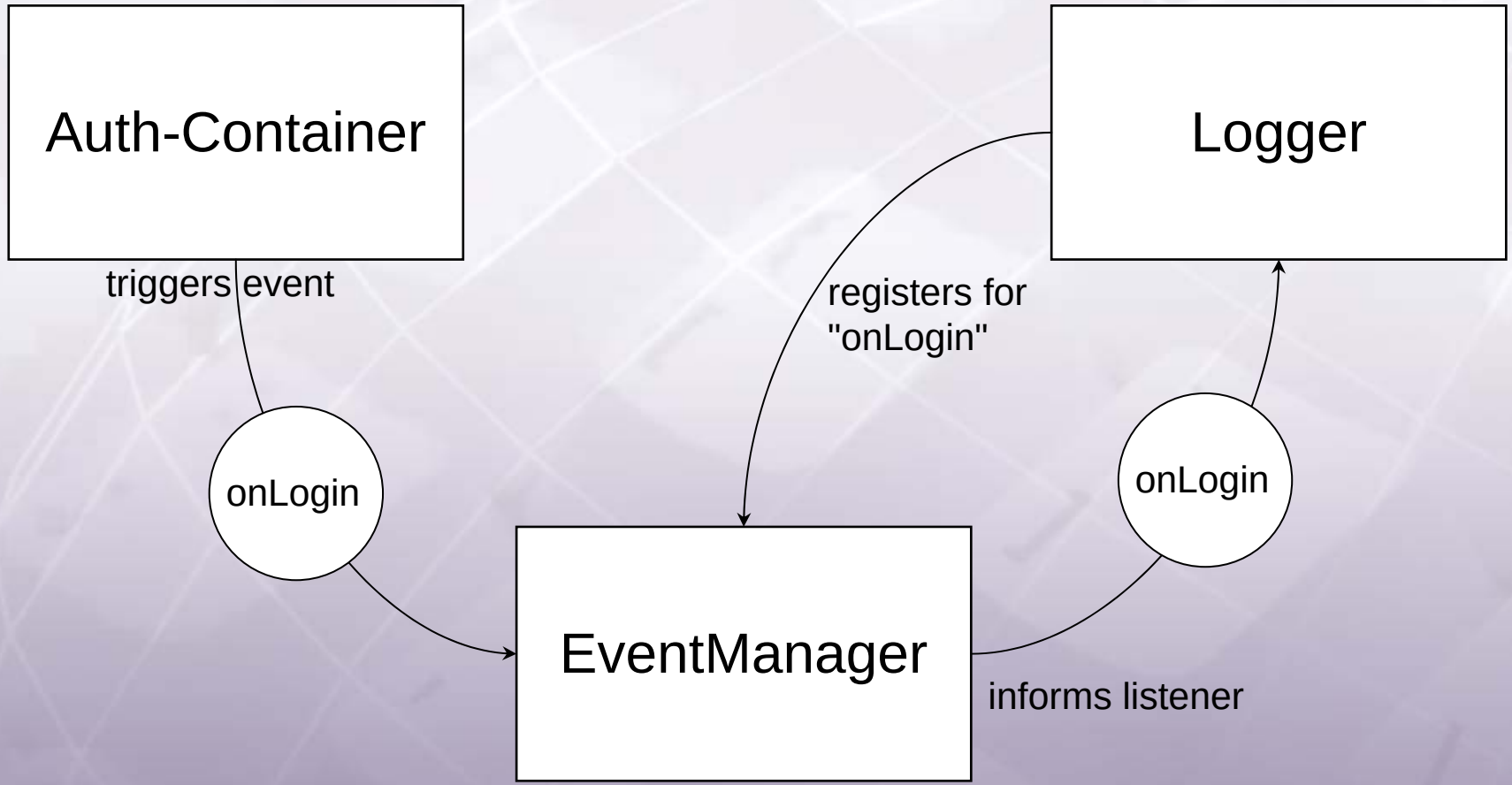
Problems

- These frameworks often support only one-to-one binding of an event to a method.
- Designer has to know about the event handlers on the server.
- Often lacks flexibility that is needed to plug in new functionality.

Event-based development

- Events enable the developer to loosely glue different components together to one application
- Derived from desktop applications and JavaScript
- Often also referred to as "Notifications" or the "Subject/Observer"-Pattern

Example


Auth-Container
Logger
triggers event
registers for
"onLogin"
onLogin
onLogin
EventManager
informs listener

Advantages of events

- Build more flexible applications
  - Event listeners instead of hardcoded relationships
  - Easily plug in new functionality (like additional checks and logging mechanisms)
  - more than one listener per event
- Easier to maintain
  - Classes are smaller
  - One class per functionality

Cocoa's Notification Center

- Used for sending messages from one object to another
- Similar to delegation, but better:
  - Any number of objects may receive the notification, not just the delegate object.
  - An object may receive any message, not just the predefined delegate methods.
  - The object posting the notification does not even have to know the observer exists.

Notification

- Encapsulates information about the event:
  - name
  - object
  - optional dictionary (user info)
- Can be posted by any object
- Posted to the NotificationCenter

Notification Center

- Manages sending and receiving of notifications
- Notifies all observers/listeners if a new notification is posted
- Sending object can also be the observing object

Notification Queue

- Acts as a buffer for notification centers
- Stores notifications so they can be sent to observers that are added at later time
  - Object A registers as observer for "onLogin"
  - Object B posts "onLogin" notification
  - Object A receives "onLogin" notification
  - Object C registers as observer for "onLogin"
  - Object A receives "onLogin" notification

PEAR::Event_Dispatcher

- PHP implementation of the NotificationCenter
- Class names differ from Cocoa
  - NotificationCenter = Event_Dispatcher
  - NotificationQueue = implemented in Event_Dispatcher as an array
  - Notification = Event_Dispatcher_

The Auth class

```
class Auth {
  private $dispatcher = null;
  private $username = null;
  public function __construct() {
  }
  public function login($user, $pass) {
    // your auth code goes here
    $this->username = $user;
    $this->dispatcher->post($this, 'onLogin');
  }
  public function getUsername() {
    return $this->username;
  }
}
```

Adding the dispatcher

```
class Auth {
  private $disp = null;
  private $username = null;
  public function __construct() {
    $this->disp = Event_Dispatcher::getInstance();
  }
  public function login($user, $pass) {
    // your auth code goes here
    $this->username = $user;
    $this->disp->post($this, 'onLogin');
  }
  public function getUsername() {
    return $this->username;
  }
}
```

The Logger

```
class UserLogger {
  var $fp;
  public function __construct($fname) {
    $this->fp = fopen($fname, 'a');
  }
  public function __destruct() {
    fclose($this->fp);
  }
  public function append($event) {
    $data = array(
       date('Y-m-d H:s:s', time()),
       $event->getNotificationName(),
       $event->getNotificationObject()->getUsername()
    );
    fputs($this->fp, implode('|', $data) . "\n");
  }
}
```

The Glue

```
$auth   = new Auth();
$logger = new UserLogger('./user.log');

$dispatcher = Event_Dispatcher::getInstance();
$dispatcher->addObserver(array($logger, 'append'),
                         'onLogin');

$auth->login('schst', 'secret');
```

```
2005-04-30 21:34:34|onLogin|schst
```

Adding a second event

```
class Auth {
  …
  public function logout() {
    // your logout code goes here
    $this->disp->post($this, 'onLogout');
    $this->username = null;
  }
}
```

Register the observer

```
$dispatcher->addObserver(array($logger, 'append'),
                         'onLogout');
```

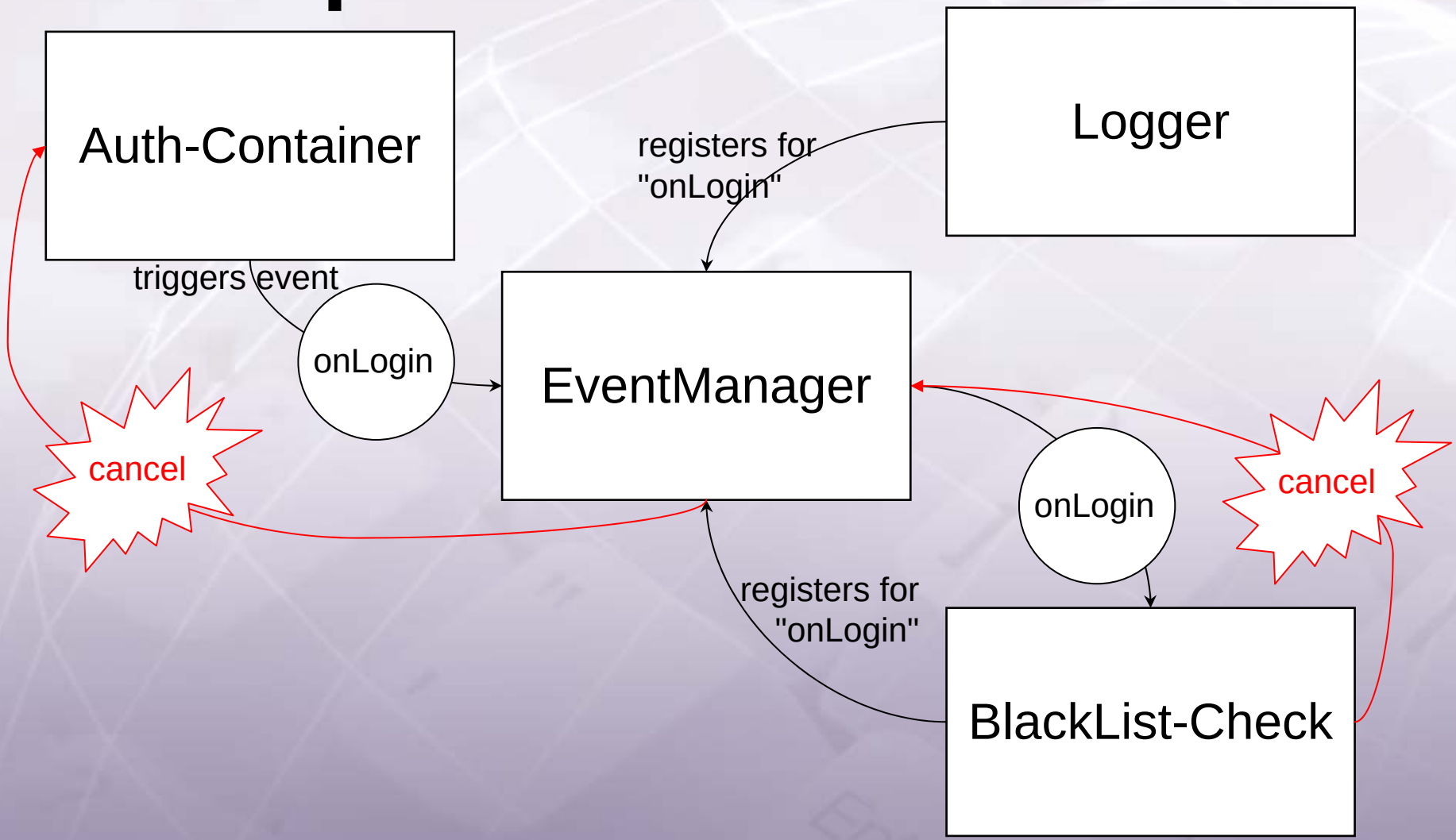
Catch'em all

```
$dispatcher = Event_Dispatcher::getInstance();
$dispatcher->addObserver(array($logger, 'append'));
```

Logfile

```
2005-04-30 22:43:43|onLogin|schst
2005-04-30 22:43:43|onLogout|schst
```

Example 2


Auth-Container
Logger
registers for
"onLogin"
triggers event
onLogin
EventManager
cancel
onLogin
cancel
registers for
"onLogin"
BlackList-Check

Canceling events

Event_Dispatcher provides more features than NotificationCenter:

- cancelNotification() aborts the event
- the following observers are not notified
- sender may check, whether notification has been cancelled

Canceling events

```
function blacklist($event) {
  $uname =
       $event->getNotificationObject()->getUsername();
  if ($uname == 'schst') {
    $event->cancelNotification();
  }
}
```

```
$dispatcher = Event_Dispatcher::getInstance();
$dispatcher->addObserver('blacklist', 'onLogin');
$dispatcher->addObserver(array($logger, 'append'));

$auth->login('schst', 'secret');
$auth->login('luckec', 'pass');
```

```
2005-04-30 22:49:49|onLogin|luckec
```

Canceling events

The sender should check, whether the event has been cancelled:

```
public function login($user, $pass) {
   // your auth code goes here
   $this->username = $user;
   $event = $this->disp->post($this, 'onLogin');
   if ($event->isNotificationCancelled()) {
     $this->username = null;
   }
}
```

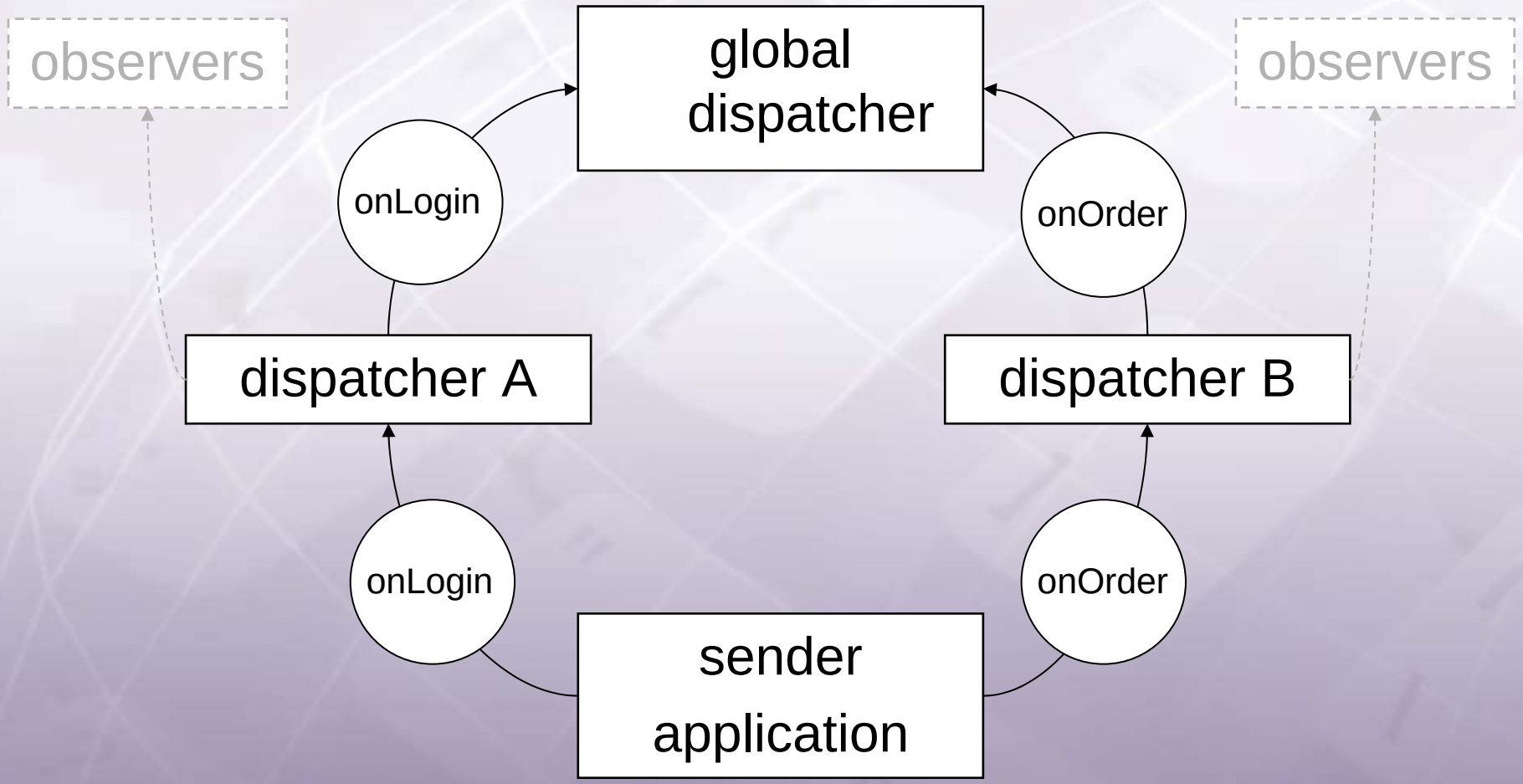
Logging blocked users

Observers may also post notifications:

```
function blacklist($event) {
  $uname =
       $event->getNotificationObject()->getUsername();
  if ($uname == 'schst') {
    $event->cancelNotification();
    $disp = Event_Dispatcher::getInstance();
    $disp->post($event->getNotificationObject(),
                'onBlocked');
  }
}
```

```
2005-04-30 22:28:28|onBlocked|schst
2005-04-30 22:28:28|onLogin|luckec
```

Event bubbling



Event bubbling

Event_Dispatcher allows more than one dispatcher:

```
$disp1 = Event_Dispatcher::getInstance('foo');
$disp2 = Event_Dispatcher::getInstance('bar');
```

Dispatchers can be nested:

```
$disp2->addNestedDispatcher($disp1);
```

Notifications will be broadcasted:

```
$disp2->post($this, 'onBar');
```

Event bubbling Example

Each Auth container needs its own dispatcher and nests the global dispatcher:

```
class Auth {
  private $disp = null;
  private $username = null;
  public function __construct($type) {
    $this->disp =
                Event_Dispatcher::getInstance($type);
    $global = Event_Dispatcher::getInstance();
    $this->disp->addNestedDispatcher($global);
  }
  …
}
```

Event bubbling Example

```
$authDB   = new Auth('DB');
$authLDAP = new Auth('LDAP');
$logger     = new UserLogger('./user.log');
$dbLogger   = new UserLogger('./user-db.log');
$ldapLogger = new UserLogger('./user-ldap.log');

$global = Event_Dispatcher::getInstance();
$global->addObserver(array($logger, 'append'));

$db = Event_Dispatcher::getInstance('DB');
$db->addObserver(array($dbLogger, 'append'));

$ldap = Event_Dispatcher::getInstance('LDAP');
$ldap->addObserver(array($ldapLogger, 'append'));

$authDB->login('schst', 'secret');
$authLDAP->login('luckec', 'pass');
```

patPortal

- PHP5 framework
- far from finished :(
- Uses concepts from NotificationCenter to glue components together
  - Event-Listeners are added using XML files
  - Each component has its own EventManager
  - Events bubble to the top if not handled

patPortal Example

- Page is requested from the client
- Controller checks whether the requested page exists
  - If yes => load and render
  - If no => display a default page and log an error
- This is not flexible, it would be better to allow the developer to handle this problem

patPortal Example

```
private function callPage($path) {
  if (!$this->sitemap->pageExists($path)) {
    $event = $this->eventManager->raiseEvent
                                  ('PageNotFound', $path);
    $path  = $event->getContext();

    if (!$event->isCancelled()) {
      …handle the error internally…
    }
  }
}
```

```
<event:event name="onPageNotFound">
  <event:listener name="Redirect" cancelEvent="true">
  <event:params>
    <event:param name="path" value="HomePage"/>
  </event:params>
</event:listener>
```

patPortal built-in events

- Request/Response
  - RequestStarted, RequestCancelled
  - ResponseSent
- Session
  - SessionStarted, SessionExpired
- Auth
  - AuthSuccess, AuthFailure, AuthCleared
- Global
  - PageNotFound

patPortal custom events

Easy to trigger your own events in the components:

```
$this->eventManager->raiseEvent(
        'EventName', $context);
```

Events triggered by the request:

```
<request:request type="HTTP">
  <request:properties>
    <request:property name="argh"
                      trigger="ArghReceived"/>
    <request:property name="__action"
                  match="/^Logout$/i" trigger="Logout"/>
  </request:properties>
</request:request>
```

Further Reading

- patForms
  http://www.php-tools.net/patForms
- PRADO
  http://prado.sourceforge.net/
- Event_Dispatcher
  http://pear.php.net/package/Event_Dispatcher
- Cocoa's Notification Center
  http://developer.apple.com/documentation/Cocoa/Conceptual/Notifications
- patPortal
  http://www.php-tools.net/patPortal

The end

Thank you for your attention!

ANY QUESTIONS ?

schst@php-tools.net
http://www.php-tools.net