

Modular Application Architecture

PHPDay Italy 2011

Kore Nordmann & Tobias Schlitt

May 12, 2011



About us

- ▶ Degree in computer science in 2010
- ▶ Each more than 10 years of professional PHP
- ▶ Open source enthusiasts
- ▶ Contributing to various FLOSS projects
- ▶ 2 of 3 founders of **Qafoo GmbH**, which provides **Services all around high quality PHP**

Outline

Application modules

Extending code

Summary



Application modules

- ▶ Develop modules separately from main application
 - ▶ External developers
 - ▶ Seperate release cycles

Application modules

Application modules

Module search paths

Configuration with module paths

Module-core interaction



Application modules

- ▶ Develop modules separately from main application
 - ▶ External developers
 - ▶ Separate release cycles
- ▶ Integrate modules
 - ▶ Locate modules
 - ▶ Module search paths
 - ▶ Configuration with module paths
 - ▶ Module-core interaction



Application modules

- ▶ Develop modules separately from main application
 - ▶ External developers
 - ▶ Separate release cycles
- ▶ Integrate modules
 - ▶ Locate modules
 - ▶ Module search paths
 - ▶ Configuration with module paths
 - ▶ **Module-core interaction**



Static files

- ▶ How to handle modules with data?
 - ▶ Templates
 - ▶ Translations
 - ▶ Images
 - ▶ CSS

Static files processed by code is "easy":

Static files not processed by code is possible to provide "overrides"

Static files are not

Static files are in a web accessible path?

Static files / link static data to htdocs/ ?

Static files through PHP?

Static files through server configuration?

Static files

- ▶ How to handle modules with data?
 - ▶ Templates
 - ▶ Translations
 - ▶ Images
 - ▶ CSS
- ▶ Everything processed by code is “easy”:
 - ▶ Make it possible to provide “overrides”

How do we not?

• How are static files in a web accessible path?

• How do we link static data to httdocs/ ?

• How do we serve static files through PHP?

• How do we do server configuration?

Static files

- ▶ How to handle modules with data?
 - ▶ Templates
 - ▶ Translations
 - ▶ Images
 - ▶ CSS
- ▶ Everything processed by code is “easy”:
 - ▶ Make it possible to provide “overrides”
- ▶ Static files are not
 - ▶ Modules are in a web accessible path?
 - ▶ Copy / link static data to htdocs/ ?
 - ▶ Pipe static files through PHP?
 - ▶ Webserver configuration?

Outline

Application modules

Extending code

Summary



Outline

Extending code

Patching

Hooks

Pipes & Filters

Inheritance

Subject-Observer

Signal-Slot



Patching the source

- ▶ The naive approach

works surprisingly well for some of the largest modules
in the world (e.g. phpBB)

Patching the source

- ▶ The naive approach
 - ▶ Works suprisingly well for some of the largest module ecosystems: phpBB



phpBB MODx format

```
1 <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
2 <?xml-stylesheet type="text/xsl" href="1.2.0/modx.prosilver.en.xsl"?>
3 <mod xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.
  phpbb.com/mods/xml/modx-1.2.0.xsd">
4   <header>
42  </header>
43   <open src="index.php">
44     <edit>
45       <comment lang="en">Here is a comment</comment>
46       <comment lang="nl">Hier is een stukje commentaar</comment>
47       <find>text to find</find>
48       <action type="replace-with">text to be replaced with</action>
49     </edit>
50     <edit>
51       <find>text to find</find>
52       <action type="after-add">text to be added on the line after</
        action>
53     </edit>
54     <edit>
55       <find>text to find</find>
56       <action type="before-add">text to be added on the line before</
        action>
57     </edit>
```

Pro & Contra

► Benefits:

- Trivial to get started with (high "hackability")
- You can change anything

► Drawbacks:

- Easy to break
- Easy to get into unparseable code
- Complex modules require deep knowledge

Pro & Contra

- ▶ Benefits:

- ▶ Trivial to get started with (high “hackability”)

- ▶ Drawbacks:

- ▶ Easy to break
 - ▶ Easy to get stuck in circular dependencies or to unparsable code
 - ▶ Complex modules require deep knowledge

Pro & Contra

► Benefits:

- Trivial to get started with (high “hackability”)
- You can change anything

► Drawbacks:

- Easy to break
- Easy to get into unparseable code
- Complex modules require deep knowledge

Pro & Contra

▶ Benefits:

- ▶ Trivial to get started with (high “hackability”)
- ▶ You can change anything

▶ Drawbacks:

- ▶ Will definitely break

▶ Will produce unparseable code
▶ Complex modules require deep knowledge

Pro & Contra

▶ Benefits:

- ▶ Trivial to get started with (high “hackability”)
- ▶ You can change anything

▶ Drawbacks:

- ▶ Will definitely break
- ▶ Can lead to unparsable code

Complex modules require deep knowledge

Pro & Contra

- ▶ **Benefits:**
 - ▶ Trivial to get started with (high “hackability”)
 - ▶ You can change anything
- ▶ **Drawbacks:**
 - ▶ Will definitely break
 - ▶ Can lead to unparsable code
 - ▶ Complex modules require deep knowledge

Outline

Extending code

Patching

Hooks

Pipes & Filters

Inheritance

Subject-Observer

Signal-Slot



Registering to hooks

- ▶ Hook indicates occurrence of event
 - ▶ Can carry arbitrary data
- ▶ Plugin can register to react on hooks
 - ▶ Manipulate data
 - ▶ Trigger action

Serendipity hook announcement

```
1 <?php
2
3 // ... in CSS code ...
4
5 // $out is CSS string
6 serendipity_plugin_api::hook_event($css_hook , $out);
7
8 echo $out;
9
10 // ... in entry display code ...
11
12 // $entry is blog entry
13 // $addData is meta data
14 serendipity_plugin_api::hook_event('frontend_display' , $entry , $addData);
```

Serendipity hook reaction

```
18 function event_hook($event, &$bag, &$eventData) {
19     global $serendipity;
20
21     $hooks = &$bag->get('event_hooks');
22
23     if (isset($hooks[$event])) {
24         switch($event) {
25             case 'frontend_display':
26                 if ( $condition /* ... */ ) {
27                     $element = $temp['element'];
28                     $eventData[$element] = $this->bbcode(
29                         $eventData[$element]
30                     );
31                 }
32                 return true;
33                 break;
```


Serendipity hook reaction

```
18 function event_hook($event, &$bag, &$eventData) {
19     global $serendipity;
20
21     $hooks = &$bag->get('event_hooks');
22
23     if (isset($hooks[$event])) {
24         switch($event) {
25             case 'css':
26                 if (strpos($eventData, '.bb-code') !== false) {
27                     // class exists in CSS, so a user has customized it and
28                     // we don't need default
29                     return true;
30                 }
31             ?>
32             .bb-quote, .bb-code, .bb-php, .bb-code-title, .bb-php-title {
33                 margin-left: 20px;
34                 margin-right: 20px;
35                 /* ... */
36             }
37             /* ... */
38             <?php
39                 return true;
40                 break;
41         }
42     }
43 }
```

Pro & Contra

► Benefits:

- High flexibility
- Low coding efforts

► Drawbacks:

- Can easily break hook data
- Fixed data formats
- "hook substitution principle" limits what you are allowed to

Pro & Contra

- ▶ Benefits:

- ▶ High flexibility

▶ Reduced coding efforts

- ▶ Drawbacks:

▶ Can't easily break hook data

▶ Limited data formats

▶ "hook substitution principle" limits what you are allowed to

Pro & Contra

- ▶ **Benefits:**
 - ▶ High flexibility
 - ▶ Low coding efforts
- ▶ **Drawbacks:**

• You can't easily break hook data
• You can't use all data formats
• "hook substitution principle" limits what you are allowed to

Pro & Contra

- ▶ **Benefits:**
 - ▶ High flexibility
 - ▶ Low coding efforts
- ▶ **Drawbacks:**
 - ▶ Plugin can easily break hook data

different data formats
"substitution principle" limits what you are allowed to

Pro & Contra

- ▶ **Benefits:**
 - ▶ High flexibility
 - ▶ Low coding efforts
- ▶ **Drawbacks:**
 - ▶ Plugin can easily break hook data
 - ▶ No defined data formats

"Liskov substitution principle" limits what you are allowed to

Pro & Contra

- ▶ **Benefits:**
 - ▶ High flexibility
 - ▶ Low coding efforts
- ▶ **Drawbacks:**
 - ▶ Plugin can easily break hook data
 - ▶ No defined data formats
 - ▶ “Liskov substitution principle” limits what you are allowed to do

Outline

Extending code

Patching

Hooks

Pipes & Filters

Inheritance

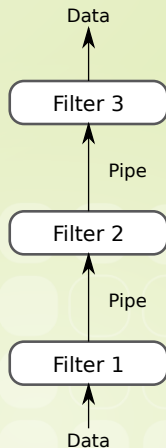
Subject-Observer

Signal-Slot



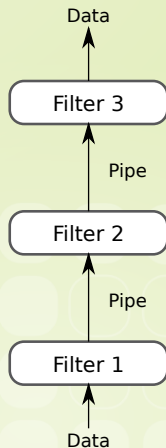
Pipes & Filters architecture

- ▶ Pipes
 - ▶ Transport data
- ▶ Filters
 - ▶ Manipulate data



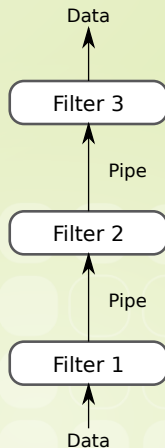
Pipes & Filters architecture

- ▶ Pipes
 - ▶ Transport data
- ▶ Filters
 - ▶ Manipulate data



Pipes & Filters architecture

- ▶ Pipes
 - ▶ Transport data
- ▶ Filters
 - ▶ Manipulate data



Popoon

- ▶ <http://bit.ly/popoon>
- ▶ Web framework
- ▶ Pipes and filters
- ▶ Inspired by Cocoon
<http://bit.ly/cocoon23>
- ▶ Defines pipelines in XML



Popoon example

```
1 <?xml version="1.0" ?>
2
3 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4 <map:pipelines>
5   <map:pipeline>
6     <map:match type="uri" pattern="examples.tgz">
7       <map:read type="tgz" src="." name="examples.tgz" />
8     </map:match>
9   </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon example

```
1 <?xml version="1.0" ?>
2
3 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4 <map:pipelines>
5   <map:pipeline>
6     <map:match type="uri" pattern="examples.tgz">
7       <map:read type="tgz" src="." name="examples.tgz" />
8     </map:match>
9   </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon example

```
1 <?xml version="1.0" ?>
2
3 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4 <map:pipelines>
5   <map:pipeline>
6     <map:match type="uri" pattern="examples.tgz">
7       <map:read type="tgz" src="." name="examples.tgz" />
8     </map:match>
9   </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Popoon example

```
1 <?xml version="1.0" ?>
2
3 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4 <map:pipelines>
5   <map:pipeline>
6     <map:match type="uri" pattern="examples.tgz">
7       <map:read type="tgz" src="." name="examples.tgz" />
8     </map:match>
9   </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```


Popoon example

```
1 <?xml version="1.0" ?>
2
3 <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
4 <map:pipelines>
5   <map:pipeline>
6     <map:match type="uri" pattern="examples.tgz">
7       <map:read type="tgz" src="." name="examples.tgz" />
8     </map:match>
9   </map:pipeline>
10
11   <map:pipeline >
12     <map:generate type="xmlfile" src="examples.xml" />
13     <map:transform type="libxslt" src="examples.xsl" />
14     <map:serialize type="html" />
15   </map:pipeline>
16 </map:pipelines>
17 </map:sitemap>
```

Pro & Contra

► Benefits:

- Clean architectural approach
- Might gain high re-usability

► Drawbacks:

- Substitution principle limits what you are allowed to do
- Might break data easily
- What forces linear code flow

Pro & Contra

- ▶ **Benefits:**

- ▶ Clean architectural approach

- ▶ Gain high re-usability

- ▶ **Drawbacks:**

- ▶ Abstraction principle limits what you are allowed to do

- ▶ Might break data easily

- ▶ What forces linear code flow

Pro & Contra

- ▶ **Benefits:**
 - ▶ Clean architectural approach
 - ▶ Might gain high re-usability
- ▶ **Drawbacks:**

Abstraction principle limits what you are allowed to do
Might break data easily
What forces linear code flow

Pro & Contra

► Benefits:

- Clean architectural approach
- Might gain high re-usability

► Drawbacks:

- Liskov substitution principle limits what you are allowed to do

might break data easily
what forces linear code flow

Pro & Contra

▶ Benefits:

- ▶ Clean architectural approach
- ▶ Might gain high re-usability

▶ Drawbacks:

- ▶ Liskov substitution principle limits what you are allowed to do
- ▶ Filters might break data easily

→ what forces linear code flow

Pro & Contra

► Benefits:

- Clean architectural approach
- Might gain high re-usability

► Drawbacks:

- Liskov substitution principle limits what you are allowed to do
- Filters might break data easily
- Somewhat forces linear code flow

Outline

Extending code

Patching

Hooks

Pipes & Filters

Inheritance

Subject-Observer

Signal-Slot



Inheritance

- Generally: Use Aggregation instead of inheritance for code re-use.

Example: Sales (OS shop software) has an interesting extension
that could entirely on inheritance

... but a number of modules can inherit from "any" class
... and each inheriting class will be used anywhere the original
class would be used.

How can that be possible?

Inheritance

- ▶ Generally: Use Aggregation instead of inheritance for code re-use.
- ▶ Oxid eSales (OS shop software) has an interesting extension model build entirely on inheritance
 - ▶ Any number of modules can inherit from “any” class ...

... which inheriting class will be used anywhere the original class could be used.

... can that be possible?

Inheritance

- ▶ Generally: Use Aggregation instead of inheritance for code re-use.
- ▶ Oxid eSales (OS shop software) has an interesting extension model build entirely on inheritance
 - ▶ Any number of modules can inherit from “any” class ...
 - ▶ ... and each inheriting class will be used anywhere the original object would be used.

... can that be possible?

Inheritance

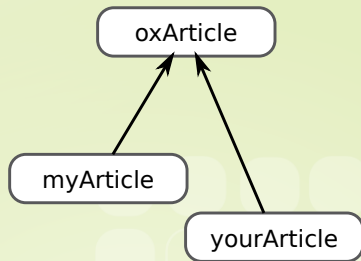
- ▶ Generally: Use Aggregation instead of inheritance for code re-use.
- ▶ Oxid eSales (OS shop software) has an interesting extension model build entirely on inheritance
 - ▶ Any number of modules can inherit from “any” class ...
 - ▶ ... and each inheriting class will be used anywhere the original object would be used.
 - ▶ *How can that be possible?*

Modular inheritance

- ▶ Objects are instantiated with a special function instead of the `new` operator.
- ▶ Inheritance graph is created on-the-fly by generating intermediate classes

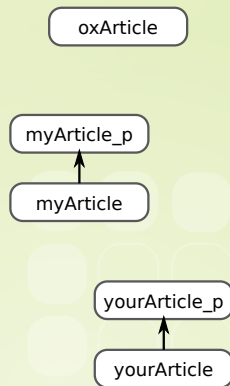
Example

```
1 <?php
2
3 class oxArticle
4 {
5     public function calculatePrice()
6     {
7         // ...
8     }
9 }
10
11 class myArticle
12     extends oxArticle
13 {
14     // ...
15 }
16
17 class yourArticle
18     extends oxArticle
19 {
20     // ...
21 }
```



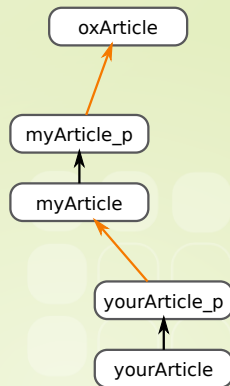
Example

```
1 <?php
2
3 class oxArticle
4 {
5     public function calculatePrice()
6     {
7         // ...
8     }
9 }
10
11 class myArticle
12     extends myArticle_parent
13 {
14     // ...
15 }
16
17 class yourArticle
18     extends yourArticle_parent
19 {
20     // ...
21 }
```



Example

```
1 <?php
2
3 class oxArticle
4 {
5     public function calculatePrice()
6     {
7         // ...
8     }
9 }
10
11 class myArticle
12     extends myArticle_parent
13 {
14     // ...
15 }
16
17 class yourArticle
18     extends yourArticle_parent
19 {
20     // ...
21 }
```



Pro & Contra

- ▶ **Benefits:**

- ▶ You can extend about everything...

- ▶ **Drawbacks:**

- ▶ Everything will be extended...

- ▶ You can't use the new operator – but use something like:

- ▶ `new Article("article")`

- ▶ You can't use object-oriented design principles

- ▶ You can't use enforcable constraints (`parent::method()`)

- ▶ The substitution principle limits what you are allowed to do

Pro & Contra

- ▶ Benefits:
 - ▶ You can extend about everything. . .
- ▶ Drawbacks:

▶ Everything will be extended. . .

▶ You can't use the new operator – but use something like:

```
class Article {  
    ...  
    "article"  
}
```

▶ Violates object-oriented design principles

▶ Enforceable constraints (parent::method())

▶ Substitution principle limits what you are allowed to do

Pro & Contra

- ▶ **Benefits:**

- ▶ You can extend about everything. . .

- ▶ **Drawbacks:**

- ▶ About everything will be extended. . .

→ you can use the new operator – but use something like:

```
class Article {  
    "article"  
}
```

→ object-oriented design principles

→ enforceable constraints (parent::method())

→ substitution principle limits what you are allowed to do

Pro & Contra

- ▶ Benefits:

- ▶ You can extend about everything...

- ▶ Drawbacks:

- ▶ About everything will be extended...
- ▶ You may not use the `new` operator – but use something like:
`oxNew("oxArticle")`

object-oriented design principles

enforceable constraints (`parent::method()`)

substitution principle limits what you are allowed to do

Pro & Contra

- ▶ **Benefits:**

- ▶ You can extend about everything...

- ▶ **Drawbacks:**

- ▶ About everything will be extended...
- ▶ You may not use the `new` operator – but use something like:
`oxNew("oxArticle")`
- ▶ **Violates object-oriented design principles**

flexible constraints (parent::method())

substitution principle limits what you are allowed to do

Pro & Contra

- ▶ **Benefits:**

- ▶ You can extend about everything...

- ▶ **Drawbacks:**

- ▶ About everything will be extended...
- ▶ You may not use the `new` operator – but use something like:
`oxNew("oxArticle")`
- ▶ Violates object-oriented design principles
- ▶ Non-enforcable constraints (`parent::method()`)

the substitution principle limits what you are allowed to do

Pro & Contra

- ▶ **Benefits:**

- ▶ You can extend about everything...

- ▶ **Drawbacks:**

- ▶ About everything will be extended...
- ▶ You may not use the `new` operator – but use something like:
`oxNew("oxArticle")`
- ▶ Violates object-oriented design principles
- ▶ Non-enforcable constraints (`parent::method()`)
- ▶ Liskov substitution principle limits what you are allowed to do

Outline

Extending code

Patching

Hooks

Pipes & Filters

Inheritance

Subject-Observer

Signal-Slot



Subject-Observer

- ▶ Loose coupling approach

“observers” on a subject to get informed about events
register for certain signal types

Subject-Observer

- ▶ Loose coupling approach
- ▶ Register “observers” on a subject to get informed about events

register for certain signal types

Subject-Observer

- ▶ Loose coupling approach
- ▶ Register “observers” on a subject to get informed about events
- ▶ Others may register for certain signal types

Subject-Observer

```
1 <?php
2
3 class Subject
4 {
5     public function doSomething()
6     {
7         $this->notify( 'doSomethingStart' );
8         // ...
9         $this->notify( 'doSomethingEnd' );
10    }
11 }
```

Subject-Observer

```
1 <?php
2
3 class Subject
4 {
5     protected $observers = array();
6
7     public function addObserver( Observer $observer )
8     {
9         $this->observers [] = $observer;
10    }
11
12    public function notify( $event, $data = null )
13    {
14        foreach ( $this->observers as $observer )
15        {
16            $observer->$event( $data );
17        }
18    }
19
20    public function doSomething()
21    {
22        $this->notify( 'doSomethingStart' );
23        // ...
24        $this->notify( 'doSomethingEnd' );
25    }
26 }
```

Subject-Observer

```
1 <?php
2
3 class Observer
4 {
5     public function doSomethingStart ()
6     {
7         // ...
8     }
9
10    public function doSomethingEnd ()
11    {
12        // ...
13    }
14 }
```

Pro & Contra

► Benefits:

- Transparent – any number of observers can register
- (Documented) clearly defined extension API
- (Optional) optionally with clearly defined transmitted data structs

► Drawbacks:

- Not transparent – you have no idea how long a signal will take to process
- Limited to defined extension points
- Requires implementation in each subject

Pro & Contra

► Benefits:

- Transparent – any number of observers can register

- (Documented) clearly defined extension API
 - optionally with clearly defined transmitted data structs

► Drawbacks:

- Not transparent – you have no idea how long a signal will process

- Limited to defined extension points
 - requires implementation in each subject

Pro & Contra

► Benefits:

- Transparent – any number of observers can register
- (Documented) clearly defined extension API

optionally with clearly defined transmitted data structs

► Drawbacks:

not transparent – you have no idea how long a signal will take to process

not limited to defined extension points

requires implementation in each subject

Pro & Contra

► Benefits:

- Transparent – any number of observers can register
- (Documented) clearly defined extension API
- ... optionally with clearly defined transmitted data structs

► Drawbacks:

- Not transparent – you have no idea how long a signal will take to process
- Not limited to defined extension points
- Requires implementation in each subject

Pro & Contra

- ▶ **Benefits:**
 - ▶ Transparent – any number of observers can register
 - ▶ (Documented) clearly defined extension API
 - ▶ ... optionally with clearly defined transmitted data structs
- ▶ **Drawbacks:**
 - ▶ Fully transparent – you have no idea how long a signal will take to process

limited to defined extension points
requires implementation in each subject

Pro & Contra

► Benefits:

- Transparent – any number of observers can register
- (Documented) clearly defined extension API
- ... optionally with clearly defined transmitted data structs

► Drawbacks:

- Fully transparent – you have no idea how long a signal will take to process
- Limited to defined extension points

requires implementation in each subject

Pro & Contra

- ▶ **Benefits:**
 - ▶ Transparent – any number of observers can register
 - ▶ (Documented) clearly defined extension API
 - ▶ ... optionally with clearly defined transmitted data structs
- ▶ **Drawbacks:**
 - ▶ Fully transparent – you have no idea how long a signal will take to process
 - ▶ Limited to defined extension points
 - ▶ **Requires implementation in each subject**

Outline

Extending code

Patching

Hooks

Pipes & Filters

Inheritance

Subject-Observer

Signal-Slot



Signal-Slot

- ▶ Very loose coupling approach

emit signal for "interesting" events

connect to slot for certain signal types



Signal-Slot

- ▶ Very loose coupling approach
- ▶ Send signals for “interesting” events

Register for certain signal types



Signal-Slot

- ▶ Very loose coupling approach
- ▶ Send signals for “interesting” events
- ▶ Others may register for certain signal types

Signal slot

```
1 <?php
2
3 $handler = new arbitSignalSlot();
4
5 $handler->register( 'signalA', array( new myModule(), 'handleSignalA' ) );
6 $handler->register( 'signalA', array( new yourModule(), 'handleSignalA' ) );
7
8 // In module c
9 $handler->emit( 'signalA', new signalADataStruct( /* ... */ ) );
10
11 // Now all modules registered for this signal are called with the provided data
12 class myModule
13 {
14     public function handleSignalA( $name, signalADataStruct $data )
15     {
16         // ...
17     }
18 }
```


Pro & Contra

► Benefits:

- Fully transparent – nobody needs to know who is doing what
- (Documented) clearly defined extension API
 - can be optionally with clearly defined transmitted data structs
 - can easily be made asynchronous

► Drawbacks:

- Not transparent – you have no idea how long a signal will take to process
- Limited to defined extension points

Pro & Contra

► Benefits:

- Fully transparent – nobody needs to know who is called
 - (Documented) clearly defined extension API
 - optionally with clearly defined transmitted data structs
 - can be made asynchronous

► Drawbacks:

- Fully transparent – you have no idea how long a signal will take to process
- limited to defined extension points

Pro & Contra

► Benefits:

- Fully transparent – nobody needs to know who is called
- (Documented) clearly defined extension API

can be optionally used in combination with clearly defined transmitted data streams
can be made asynchronous

► Drawbacks:

not fully transparent – you have no idea how long a signal will take to reach the process
not fully documented – limited to defined extension points

Pro & Contra

► Benefits:

- Fully transparent – nobody needs to know who is called
- (Documented) clearly defined extension API
- ... optionally with clearly defined transmitted data structs

► Drawbacks:

- Not fully transparent – you have no idea how long a signal will take to reach the process
- Limited to defined extension points

Pro & Contra

► Benefits:

- Fully transparent – nobody needs to know who is called
- (Documented) clearly defined extension API
- ... optionally with clearly defined transmitted data structs
- Can easily be made asynchronous

► Drawbacks:

- Not transparent – you have no idea how long a signal will take to reach the process
- Limited to defined extension points

Pro & Contra

► Benefits:

- Fully transparent – nobody needs to know who is called
- (Documented) clearly defined extension API
- ... optionally with clearly defined transmitted data structs
- Can easily be made asynchronous

► Drawbacks:

- Fully transparent – you have no idea how long a signal will take to process

► Not possible to define extension points

Pro & Contra

- ▶ **Benefits:**
 - ▶ Fully transparent – nobody needs to know who is called
 - ▶ (Documented) clearly defined extension API
 - ▶ ... optionally with clearly defined transmitted data structs
 - ▶ Can easily be made asynchronous
- ▶ **Drawbacks:**
 - ▶ Fully transparent – you have no idea how long a signal will take to process
 - ▶ Limited to defined extension points

Outline

Application modules

Extending code

Summary



Summary

- ▶ Patching
- ▶ Hooks
- ▶ Pipes & Filters
- ▶ Inheritance
- ▶ Subject-Observer
- ▶ Signal-Slot



Thanks for listening

Please rate this talk at
<http://joind.in/3018>
and / or give me some feedback right now!

Thanks for listening

Please rate this talk at
<http://joind.in/3018>
and / or give me some feedback right now!

Stay in touch

- ▶ Kore Nordmann / <kore@qafoo.com> / @koredn
- ▶ Tobias Schlitt / <toby@qafoo.com> / @tobySen
- ▶ **Rent a PHP quality expert:**
<http://qafoo.com> / @qafoo