Services made simple with PHP

Caroline Maynard

IBM
caroline.maynard@uk.ibm.com
cem@php.net

15 May 2007

Agenda

- Introduction and rationale
- A simple service using SOAP
- Compound data structures
- Other RPC-style protocols
- Resource-oriented protocols
- A custom protocol
- Wrap-up

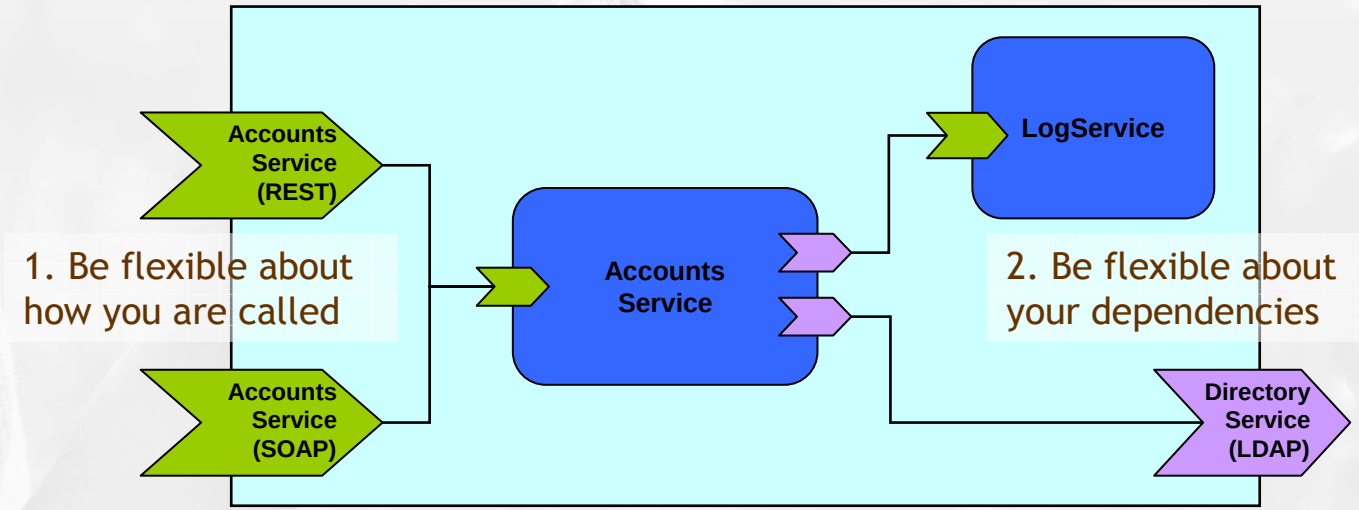
What is SCA for ?

- Service Component Architecture (SCA) is for PHP developers working with a mixture of technologies in a changing environment
- SCA is intended to help you to:
  - write reusable code by keeping separate the business logic and the communications code
  - support several protocols at once without duplicating code
  - provide local / remote transparency
  - easily consume external Web services
  - easily expose Web services to others
- without having to:
  - hand-edit service descriptions (like WSDL)
  - create external configuration files
  - introduce new deployment steps

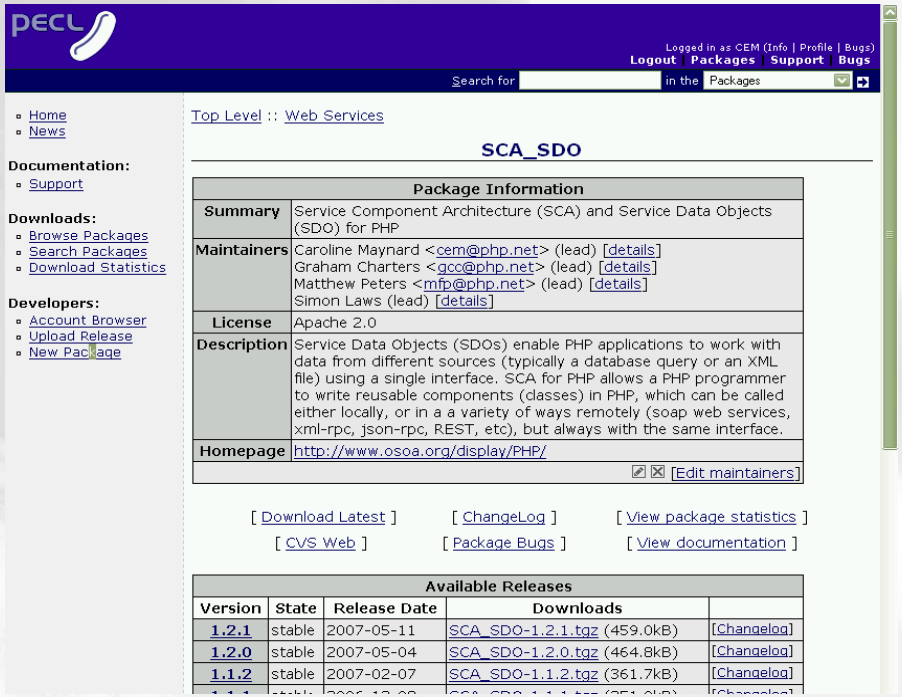
Making your component reusable

- Do not entangle the business logic with the “wiring”

1. Be flexible about how you are called
   - Expose as many ‘bindings’ as needed – make sure your business logic does not need to know how it was called
2. Be flexible about your dependencies
   - Declare the dependencies – but make sure your business logic does not need to know how to resolve these
   - Ideally get something else to “wire up” the components (Inversion of Control; Dependency Injection patterns)



The SCA_SDO PECL package



pecl

Logged in as CEM (Info | Profile | Bugs)
Logout | Packages | Support | Bugs

Search for in the Packages

- Home
- News

Documentation:
- Support

Downloads:
- Browse Packages
- Search Packages
- Download Statistics

Developers:
- Account Browser
- Upload Release
- New Package

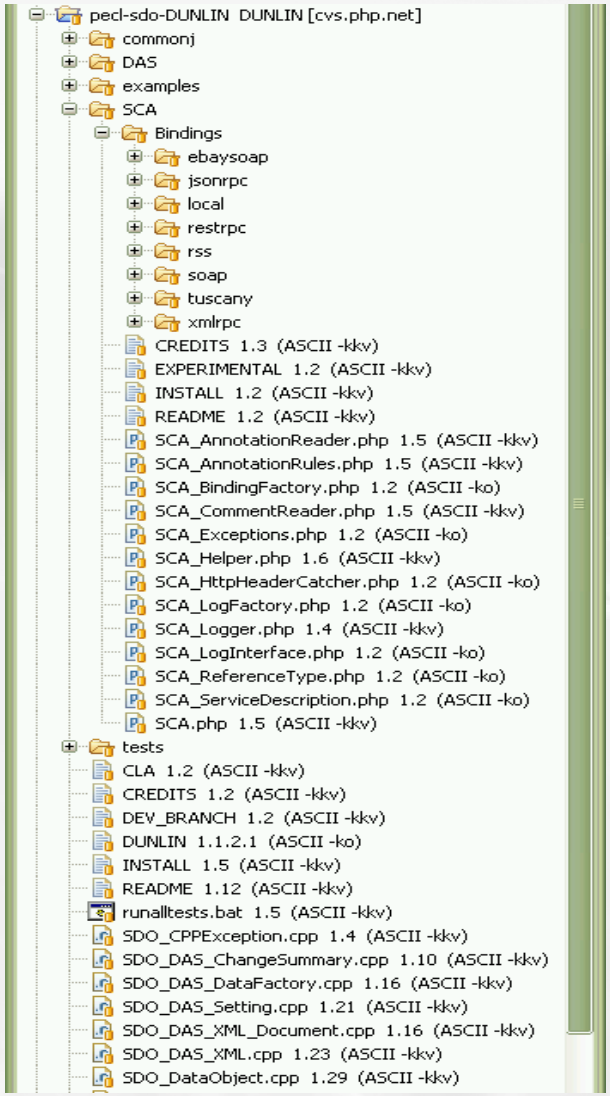
Top Level :: Web Services

SCA_SDO

Package Information	
Summary	Service Component Architecture (SCA) and Service Data Objects (SDO) for PHP
Maintainers	Caroline Maynard <cem@php.net> (lead) [details] Graham Charters <gcc@php.net> (lead) [details] Matthew Peters <mfp@php.net> (lead) [details] Simon Laws (lead) [details]
License	Apache 2.0
Description	Service Data Objects (SDOs) enable PHP applications to work with data from different sources (typically a database query or an XML file) using a single interface. SCA for PHP allows a PHP programmer to write reusable components (classes) in PHP, which can be called either locally, or in a a variety of ways remotely (soap web services, xml-rpc, json-rpc, REST, etc), but always with the same interface.
Homepage	http://www.osoa.org/display/PHP/
	[Edit maintainers]

[Download Latest] [ChangeLog] [View package statistics]
[CVS Web] [Package Bugs] [View documentation]

Available Releases				
Version	State	Release Date	Downloads	
1.2.1	stable	2007-05-11	SCA_SDO-1.2.1.tgz (459.0kB)	[Changelog]
1.2.0	stable	2007-05-04	SCA_SDO-1.2.0.tgz (464.8kB)	[Changelog]
1.1.2	stable	2007-02-07	SCA_SDO-1.1.2.tgz (361.7kB)	[Changelog]



- pecl-sdo-DUNLIN DUNLIN [cvs.php.net]
  - commonj
  - DAS
  - examples
  - SCA
    - Bindings
      - ebaysoap
      - jsonrpc
      - local
      - restrpc
      - rss
      - soap
      - tuscany
      - xmlrpc
    - CREDITS 1.3 (ASCII -kkv)
    - EXPERIMENTAL 1.2 (ASCII -kkv)
    - INSTALL 1.2 (ASCII -kkv)
    - README 1.2 (ASCII -kkv)
    - SCA_AnnotationReader.php 1.5 (ASCII -kkv)
    - SCA_AnnotationRules.php 1.5 (ASCII -kkv)
    - SCA_BindingFactory.php 1.2 (ASCII -ko)
    - SCA_CommentReader.php 1.5 (ASCII -kkv)
    - SCA_Exceptions.php 1.2 (ASCII -ko)
    - SCA_Helper.php 1.6 (ASCII -kkv)
    - SCA_HttpHeaderCatcher.php 1.2 (ASCII -ko)
    - SCA_LogFactory.php 1.2 (ASCII -ko)
    - SCA_Logger.php 1.4 (ASCII -kkv)
    - SCA_LogInterface.php 1.2 (ASCII -ko)
    - SCA_ReferenceType.php 1.2 (ASCII -ko)
    - SCA_ServiceDescription.php 1.2 (ASCII -ko)
    - SCA.php 1.5 (ASCII -kkv)
  - tests
    - CLA 1.2 (ASCII -kkv)
    - CREDITS 1.2 (ASCII -kkv)
    - DEV_BRANCH 1.2 (ASCII -kkv)
    - DUNLIN 1.1.2.1 (ASCII -ko)
    - INSTALL 1.5 (ASCII -kkv)
    - README 1.12 (ASCII -kkv)
    - runalltests.bat 1.5 (ASCII -kkv)
    - SDO_CPPException.cpp 1.4 (ASCII -kkv)
    - SDO_DAS_ChangeSummary.cpp 1.10 (ASCII -kkv)
    - SDO_DAS_DataFactory.cpp 1.16 (ASCII -kkv)
    - SDO_DAS_Setting.cpp 1.21 (ASCII -kkv)
    - SDO_DAS_XML_Document.cpp 1.16 (ASCII -kkv)
    - SDO_DAS_XML.cpp 1.23 (ASCII -kkv)
    - SDO_DataObject.cpp 1.29 (ASCII -kkv)

PECL : http://pecl.php.net/sca_sdo
Mail : http://www.google.com/group/phpsoa
Web : http://www.osoa.org/display/PHP
Specs: Google for “osoa”

Using SCA to expose and consume services

php|tek PHP::innovation() Chicago, May 15-18, 2007

An SCA service

- It's just a PHP class
  - includes `SCA.php` (last)
  - uses phpDocumentor-style annotations to *declare* capabilities
  - methods must assume pass-by-value
- But other than this, job done!
  - make sure SDO extension is loaded (`sdo.so` or `php_sdo.dll`)
  - drop the class file into Apache
  - the EmailService is now exposed as a Web service

```php
<?php
include 'SCA/SCA.php';

/**
 * Service for sending emails
 *
 * @service
 * @binding.soap
 */
class EmailService {
    ...
    /**
     * Send a simple text email
     *
     * @param string $to The "to" email address
     * @param string $from The "from" email address
     * @param string $subject The subject of the email
     * @param string $message The email message
     * @return boolean
     */
    public function send ($to, $from, $subject, $message) {
        ...
    }
}

?>
```

Consuming a service from a PHP script

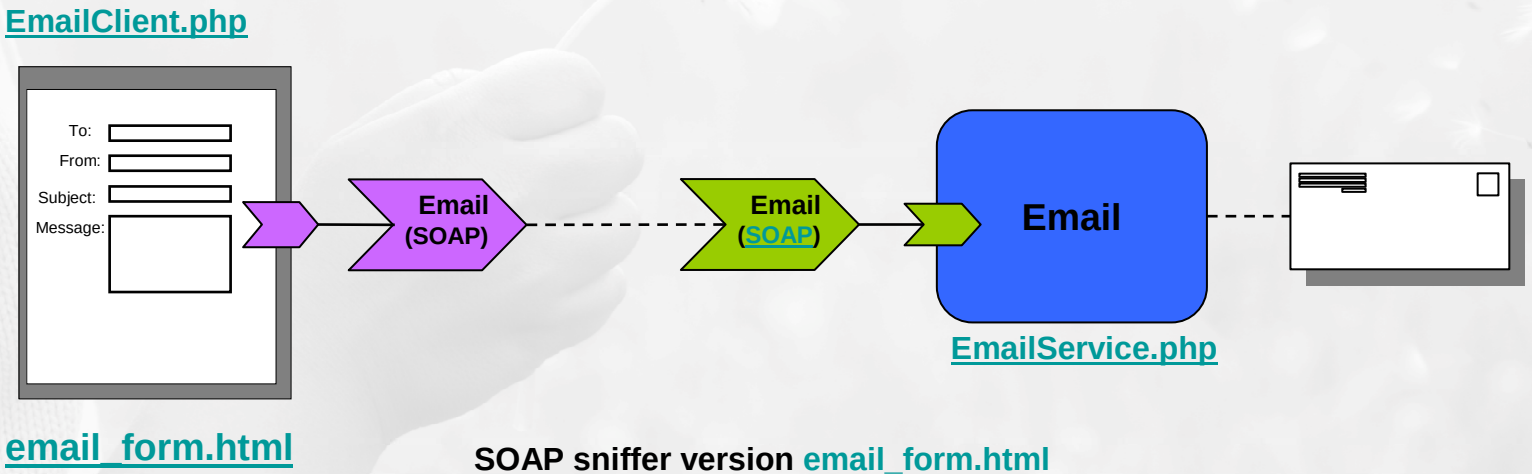

```
$to      = $_POST['to'];
$from    = $_POST['from'];
$subject = $_POST['subject'];
$message = $_POST['message'];

include 'SCA/SCA.php';

$email_service = SCA::getService(
                ‘http://www.example.com/EmailService.php?wsdl');
$success = $email_service->send($to, $from, $subject, $message);
```

A simple email form

- Write an SCA Component
- Expose it as a Web service
- Generate the WSDL
- Consume it in a client script



SOAP sniffer version email_form.html

Aside: what would that look like with ext/soap?

- first generate the WSDL (somehow) and copy it to the client
- service must create a SoapServer and add the Email service to it
- client must
  - create a SoapClient
  - wrap and unwrap parameters

```
$server = new
            SoapServer('./EmailService.wsdl');
$server->setClass('EmailService');
$server->handle();

class EmailService {
   ...
}
```

```
$soap_client = new
            SoapClient('./EmailService.wsdl');
$send_params = array(
            'to'=>$to,
            'from'=>$from,
            'subject'=>$subject,
            'message'=>$message);
$send_response =
            $soap_client->send($send_params);
$success = $send_response->sendReturn;
```

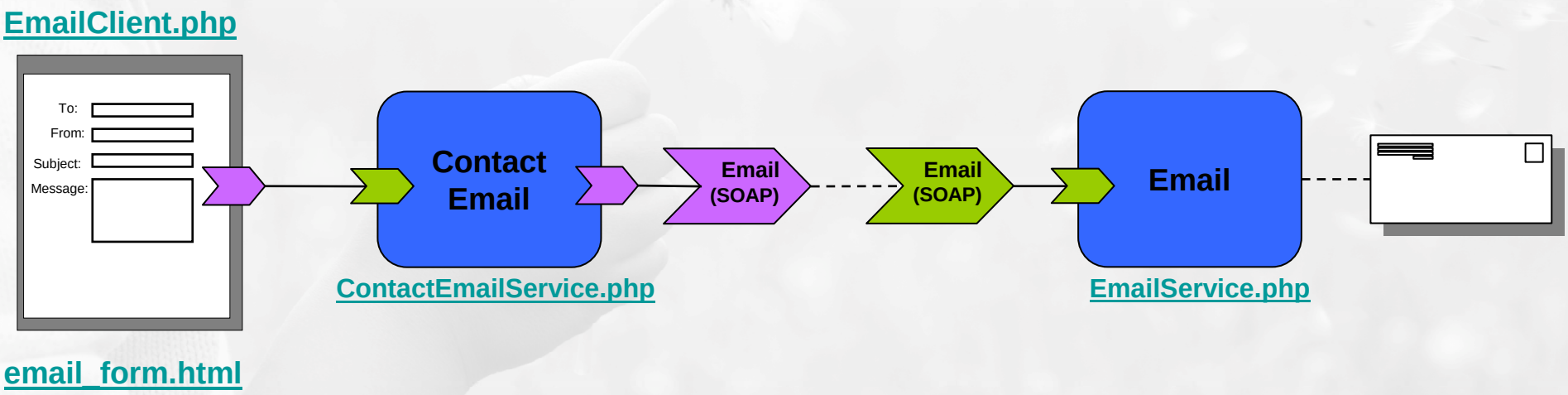
Consuming a service from a component


```
/**
 * Service for sending emails (supports shortnames)
* @service
 */
class ContactEmailService {

  /**
   * @reference
   * @binding.soap ./EmailService.wsdl
   */
  public $email_service;

  /** ... */
  public function send($to, $from, $subject, $message) {
    ...
    // a proxy to the service is 'injected' so we
    // can just use it...
    $this->email_service->
          send($to, $from, $subject, $message);
    ...
  }
}
```

- Write a new SCA component
- Expose it as a Local service
- Have it reference the Email service
- Consume it in the client script



Using SDO to work with compound data

Handling compound data

- Not all services exchange scalars!
- SCA uses Service Data Objects to handle compound data
- SDO requires a description of the data structures
  - currently XML schema
  - future: annotated PHP classes

Annotations for compound data

- Three steps to providing a service with complex types
  1. create a schema for the data structure
  2. annotate class to map namespaces to schema files
  3. document the class methods to show where the types are used

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://example.org/contacts">

 <element name="contact">
  <complexType>
   <sequence>
    <element name="shortname" type="string" />
    <element name="fullname" type="string" />
    <element name="email" type="string" />
   </sequence>
  </complexType>
 </element>

</schema>
```

1

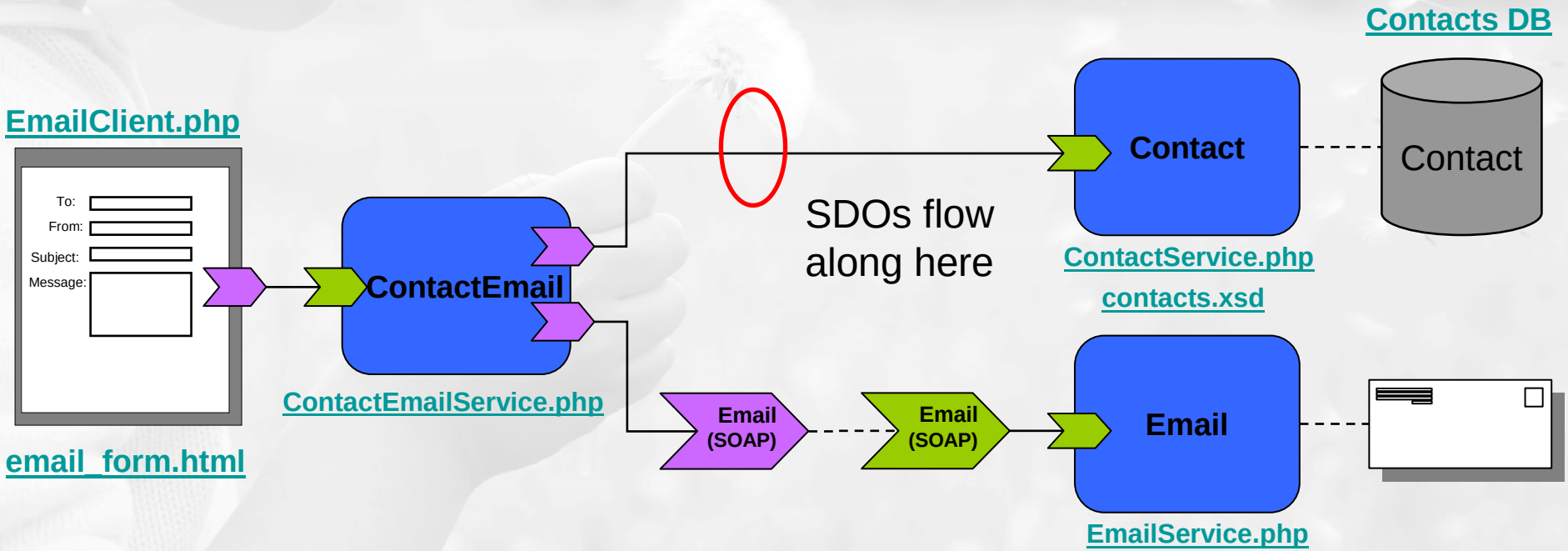
```
/**
 * Service for managing email contacts
 * @service
 * @types http://example.org/contacts contacts.xsd
 */
class ContactService {

   /**
    * Retrieve contact details
    *
    * @param string $shortname Short name of the contact
    * @return contact http://example.org/contacts The contact
    */
   public function retrieve($shortname) {
      $contact = SCA::createDataObject(
            'http://example.org/contacts', 'contact');
      ...
      return $contact;
   }
}
```

2

3

- Create a new Contact service
- Design the data structure to represent a contact
- Adding data structures to the contact service
- Reference the Contact service from the ContactEmail service
- Use data structures in the ContactEmail service

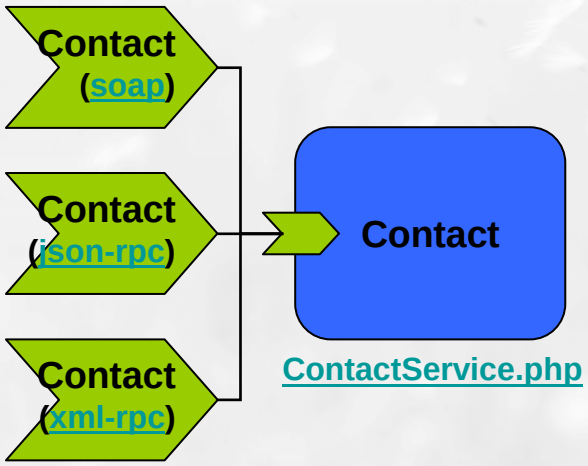


Other RPC-style bindings

Other RPC-style bindings

Supporting several bindings

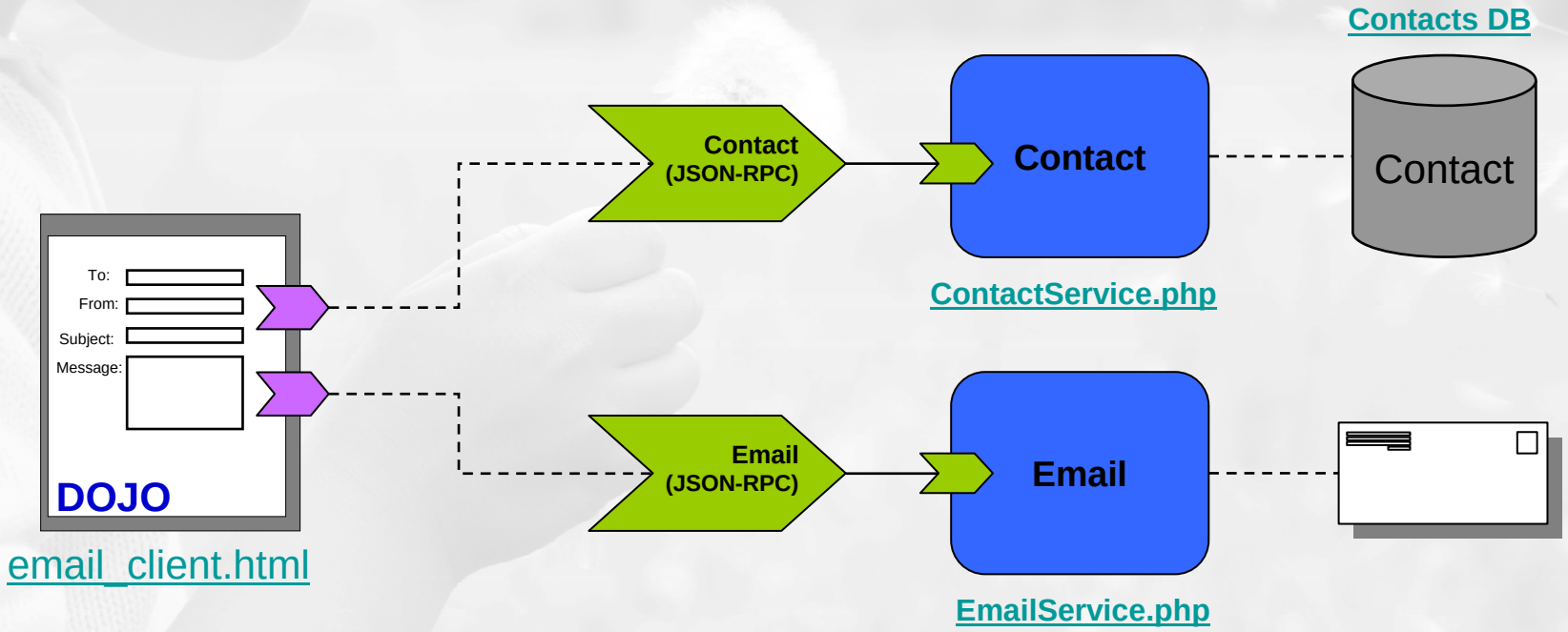
- Need to be able to choose protocols
  - As a provider: different clients (customers) prefer or require different protocols
    - Java client (soap/http), JavaScript client (json-rpc), …
  - As a consumer: no one protocol is supported by all service providers
- Various bindings available
  - Local
  - SOAP
  - JSON-RPC
  - XML-RPC
  - REST-RPC
- Intend to provide others



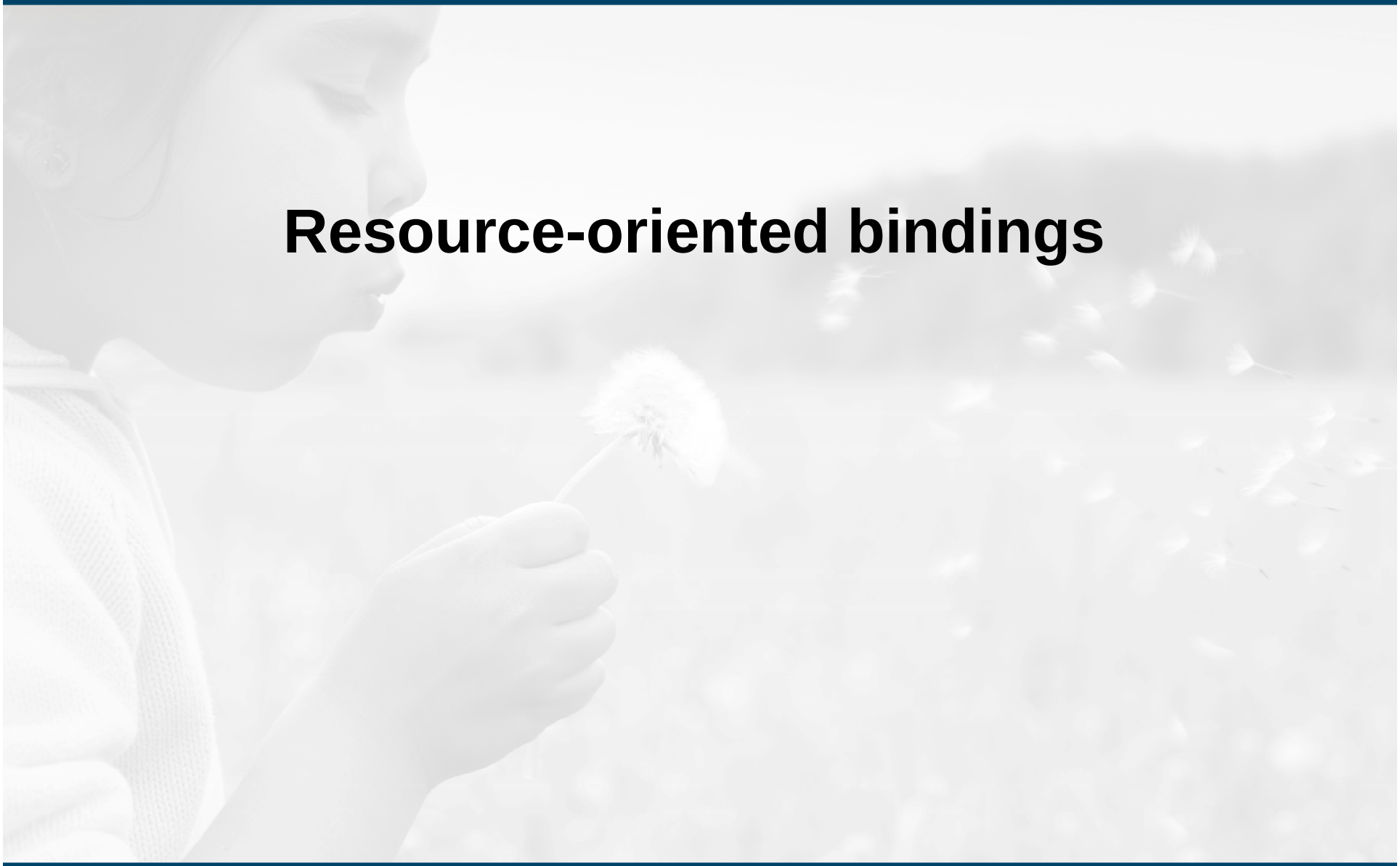
ContactService.smd **(formatted)**

AJAX application calling SCA service

- Add a json-rpc binding to the Contact and Email services
- Call the services directly from a DOJO-based AJAX application via json-rpc



Resource-oriented bindings



Other styles of services

- What we've seen up to now is a number of RPC-style services
- Other styles exist that are equally valid
  - Resource-Oriented REST (**RE**presentational **S**tate **T**ransfer)
  - Plain Old XML (POX)
  - Syndication (Atompub, RSS)
  - …and many more…
- No clean taxonomy/terminology exists
  - http://www.intertwingly.net/blog/2006/11/03/REST-Web-Services
  - http://www.trachtenberg.com/blog/2006/11/06/rest-vs-httppox-vs-soap/
  - http://www.ibm.com/developerworks/xml/library/ws-restvsoap/

Resource-Oriented REST background

- An architectural style for well-designed Web applications, not a standard
- Considers the Web to be a state machine
  - A network of Resources (e.g. Web pages) – a virtual state machine
  - Navigating resources via links results in **representations** of **states** being **transferred** to the user agent (e.g. browser)

- This concept is used to describe a class of Web services
  - URIs identify Resources on the Web
  - HTTP used to access and modify these Resources

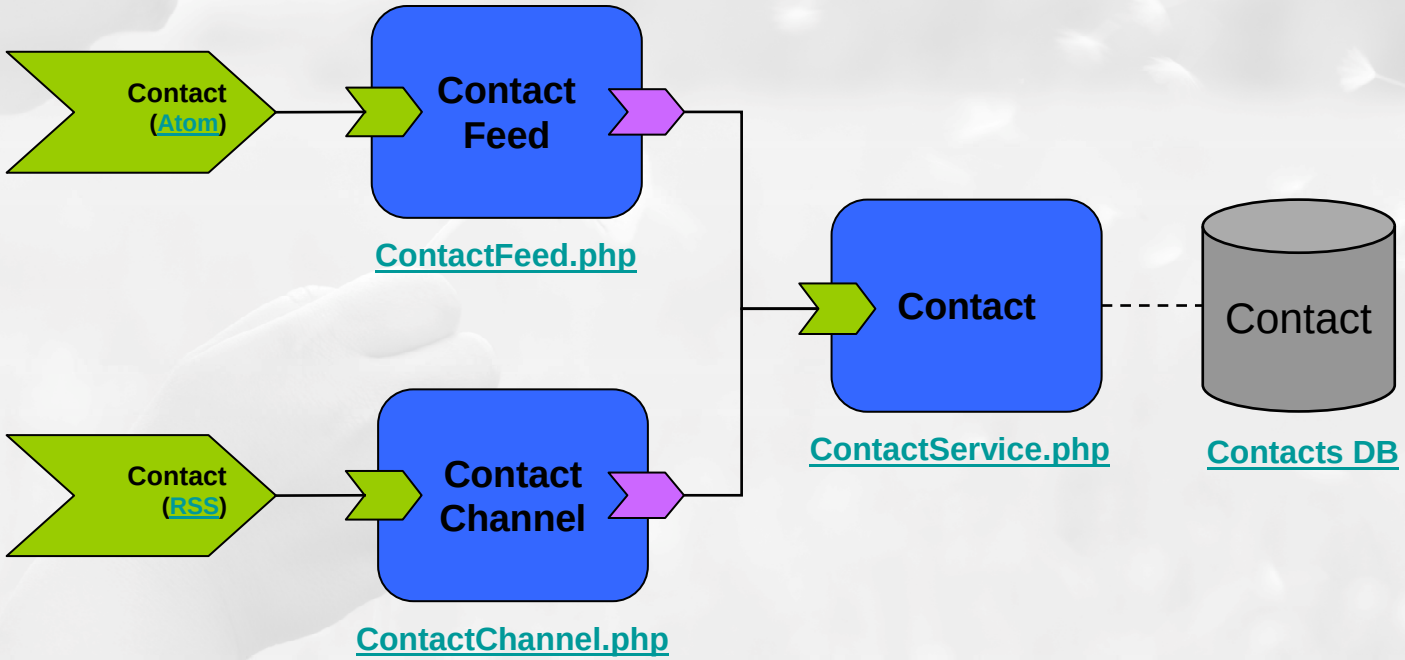
HTTP verb	Operation
Post	Create
Get	Retrieve
Put	Update
Delete	Delete

- REST says nothing about the representations (formats) – might be HTML, XML, JSON, serialized PHP, …

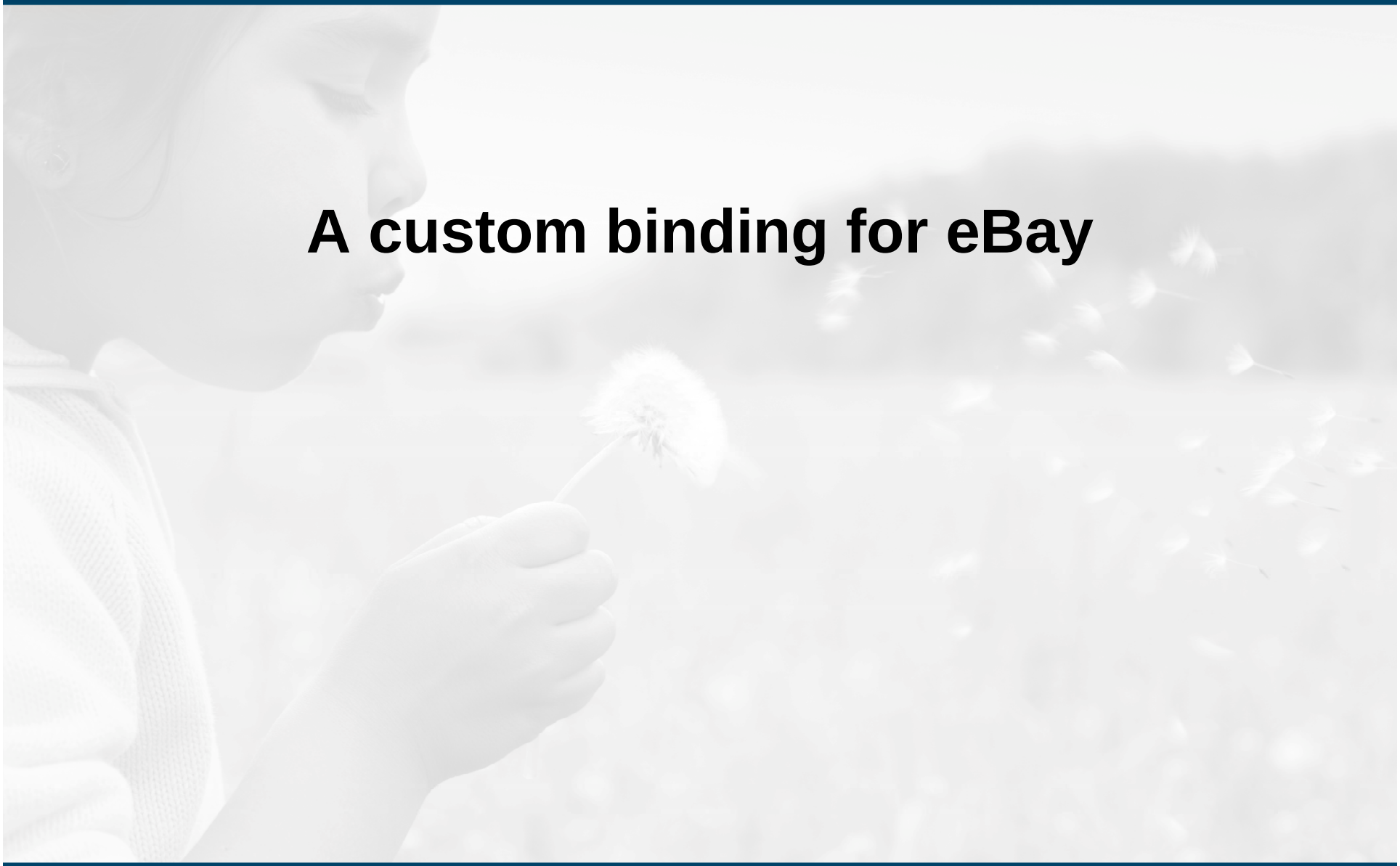
Syndication Services

- RSS and Atom are service types used to publish information
- Give the appearance of publish-subscribe but actually still request-response under the covers
- Not just about syndicating news feeds
- Can be thought of as standardized Resource-oriented REST services

- Syndication services through SCA



A custom binding for eBay



Custom Bindings

- Many real-world services are complex and difficult to call through a generic binding (eBay, Google GData, and so on)
- SCA allows people to write and contribute custom bindings

- eBay Soap requires a client to provide:
  - Soap Body (the main request)

```
<SOAP-ENV:Body>
  <GetSearchResultsRequest...>
    <Version>495</Version>
    <Query>ipod</Query>
    <Pagination>
      <EntriesPerPage>10</EntriesPerPage>
    </Pagination>
  </GetSearchResultsRequest>
</SOAP-ENV:Body>
```

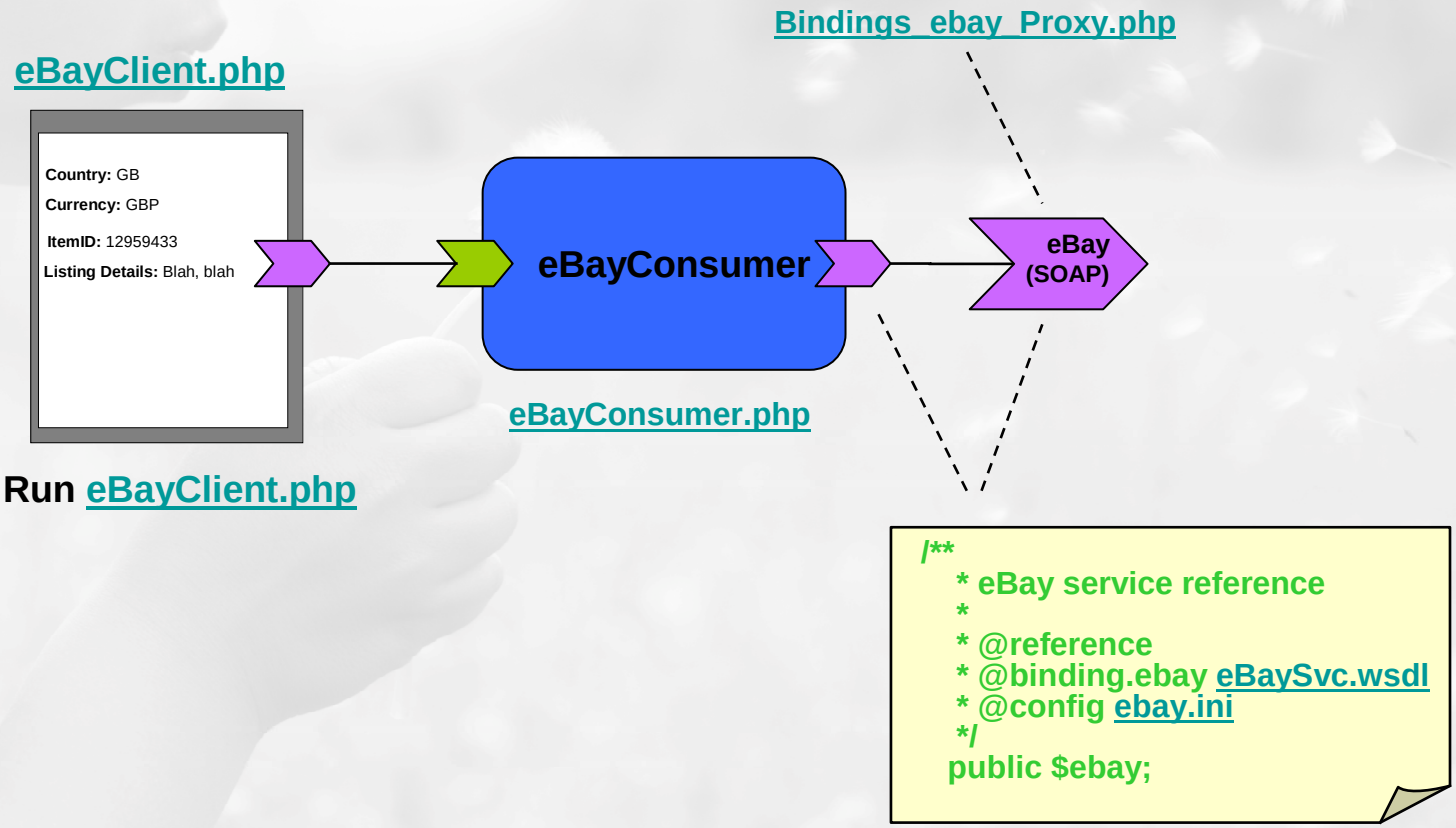
  - Soap Header (the security information)

```
<SOAP-ENV:Header>
  <RequesterCredentials ...>
    <eBayAuthToken>AgAAAA**AQAAA...ST+aWf1</eBayAuthToken>
    <Credentials>
      <AppId>IBMUN...</AppId>
      <DevId>...</DevId>
      <AuthCert>...</AuthCert>
    </Credentials>
  </RequesterCredentials>
</SOAP-ENV:Header>
```

  - Url Query String Parameters (for eBay to route requests)

```
POST /?callname=GetSearchResults&siteid=1&version=495&appid=…&Routing=default HTTP/1.1
Host: api.sandbox.ebay.com
```

- Solution: create “ebaysoap” binding extending the “soap” binding with eBay-specific configuration



Almost the end

- PHP classes for data structures
  - Simpler but less capable than xsd
- Simple database services
  - A CRUD service for a table
- Other bindings
  - Improve: Atom, RSS
  - New: Resource-oriented REST, Google GData, Yahoo!
- Annotation overriding
  - Externally changing service targets, bindings, properties

Summary

- SCA for PHP enables a PHP programmer to write components in PHP which are unaware of local/remote and protocol differences and can focus on providing reusable business logic.
- Components use PHP annotations both to declare their dependencies on other components, and to define the interface which they expose as a service. The SCA for PHP runtime resolves all of these.
- Deploying a PHP component as a 'Web service' can be as simple as copying it into a web server's document root. The SCA for PHP runtime automatically generates service descriptions (WSDL, SMD) for these when requested.

How To Find Out More…

- The PECL package
  - Go to PECL and search for SCA, SDO or SCA_SDO
  - http://pecl.php.net/sca_sdo
- Web Site
  - As well as the information in the PHP Manual there is a web site.
  - http://www.osoa.org/display/PHP/
- Mail List
  - For rants, questions, feedback etc. there is a Google Groups mail list called PHPSOA
  - http://groups.google.com/group/phpsoa
- Documents Describing SCA and SDO in more detail
  - Google for OSOA
  - http://www.osoa.org/display/Main/Home

The end