Prof. Dr. Claudia Müller-Birn
Institute for Computer Science, Networked Information Systems

Freie Universität Berlin

Web application development (part 1)

01-09-2012

Netzprogrammierung

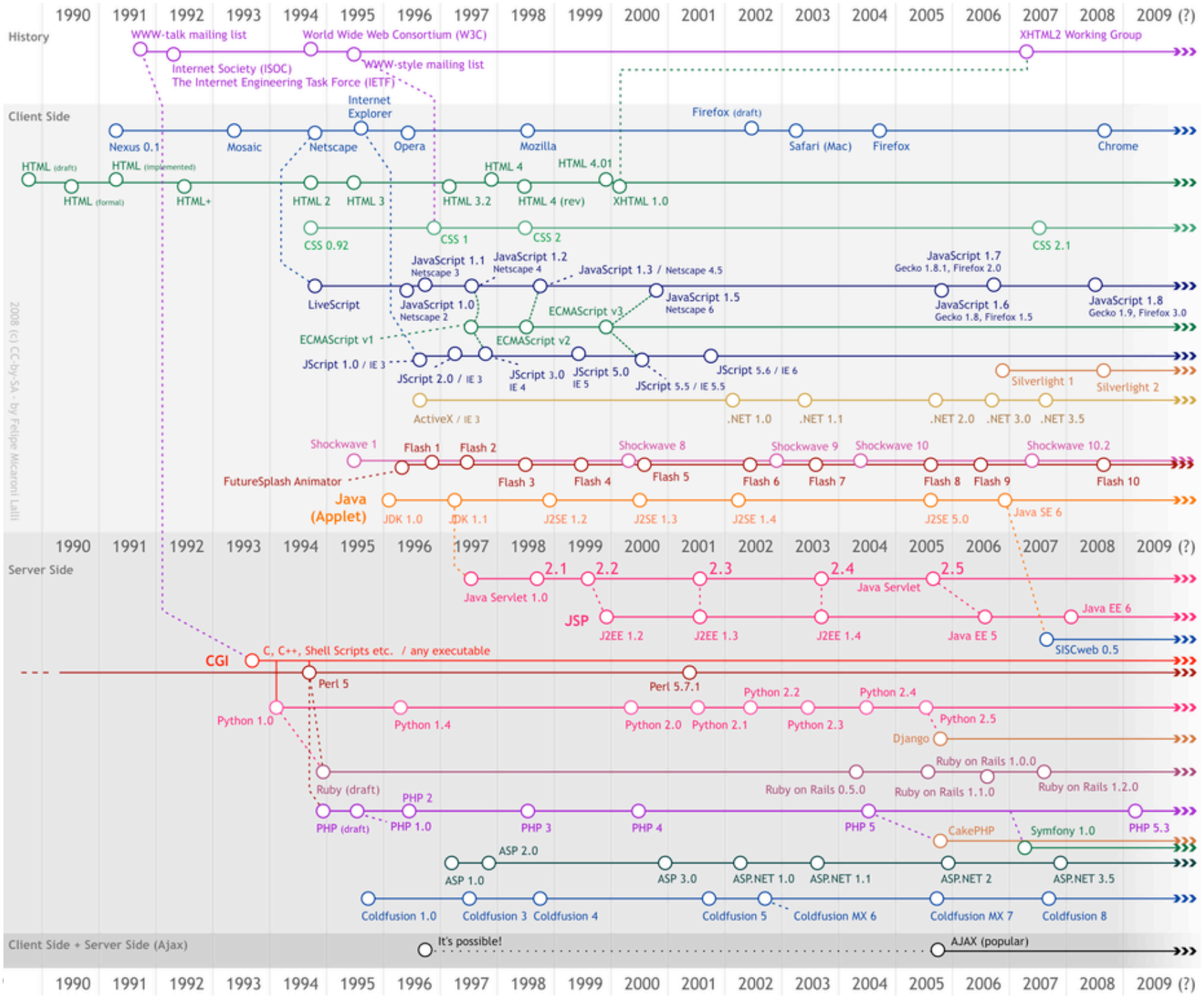
(Algorithmen und Programmierung V)

Our topics today

Finishing material from last lecture, in other words, talking about (the) REST

Client-side programming of static webpages using Hyper Text Markup Language (HTML) and Cascading Stylesheets (CSS)

Server-side programming of dynamic webpages using Common Gateway Interface (CGI) and Hypertext Preprocessor (PHP)



http://en.wikipedia.org/wiki/File:Web_development_timeline.png

Representational State Transfer (REST)

Representational State Transfer (REST)

What is REST?

REST stands for **Re**presentational **S**tate **T**ransfer and it was invented by Roy Fielding in 2000.

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

REST is *an architecture style* for designing networked applications. The idea is that, rather than using complex mechanisms such as CORBA, RPC or SOAP to connect between machines, simple HTTP is used to make calls between machines.

The World Wide Web itself, based on HTTP, can be viewed as a REST-based architecture.

(Elkstein, 2008)

REST design principles

Stateless Client/Server Protocol: Each message contains all the information needed by a receiver to understand and/or process it. This constraint attempts to “keep things simple” and avoid needless complexity.

A set of uniquely addressable resources enabled by a universal syntax for resource identification; “Everything is a Resource” in a RESTful system.

A set of well-defined operations that can be applied to all resources.

Resources are typically stored in a structured data format that supports hypermedia links, such as HTML or XML.

Resources

Resources are defined by URIs

- Resources can never be accessed or manipulated directly
- REST works with resource representations

Resources are all the things we want to work with

- If you cannot name something, you cannot do anything with it
- A popular resource type on the Web are documents
- Documents usually are a structured collection of information

Documents are abstract concepts of descriptive resources

- They may be used in different contexts (e.g., formats)
- Different applications may be interested in different representations
- The underlying resource is always the same

State

State is represented as part of the content being transferred

- Server interruptions do not create problems for the client
- It is possible to switch between servers for different interactions
- Clients can simply store the representation to save the state

State transfer makes the system scalable

- Data transfer is not state-specific (no stateful connection handling)
- State is transferred between client and server

Establishing a common model

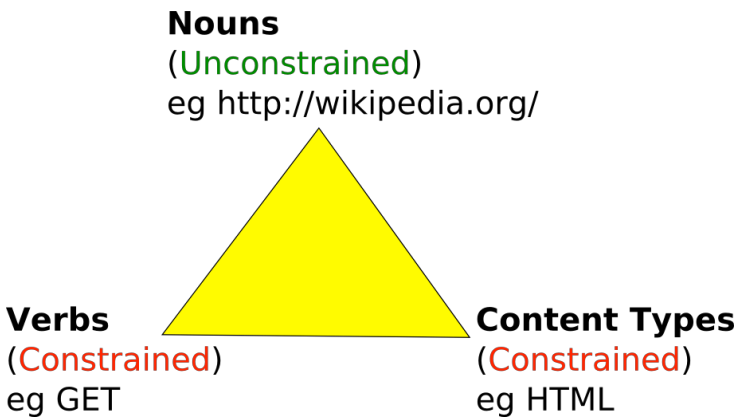
Distributed systems must be based on a shared model

- Traditional systems must agree on a common API
- REST systems structure agreement into three areas

REST is built around the idea of simplifying agreement

- *nouns* are required to name the resources that can be talked about
- *verbs* are the operations that can be applied to named resources
- *content types* define which information representations are available

REST triangle



Nouns

Nouns are the names of resources

- In most designs, these names will be URIs
- URI design is a very important part of a REST-based system design

Everything of interest should be named

- By supporting well-designed names, applications can talk about named things
- New operations and representations can be introduced

Separating nouns from verbs and representations improves extensibility

- Applications might still work with resources without being able to process them
- Introducing new operations on the Web does not break the Web
- Introducing new content types on the Web does not break the Web

Verbs

Operations which can be applied to resources.

The core idea of REST is to use universal verbs only

- Universal verbs can be applied to all nouns

For most applications, HTTP's basic methods are sufficient

GET: Fetching a resource (there must be no side-effects)

PUT: Transfers a resource to a server (overwriting if there already is one)

POST: Adds to an existing resource on the server

DELETE: Discards a resource (its name cannot be used anymore)

Corresponding to the most popular basic database operations - CRUD: Create, Read, Update, Delete

POSTing

POST adds instead of an overwriting update.

POST can have different effects

- By POSTing, state is changed and a new resource is created
- By POSTing, only the existing resource is changed
- The server signals the difference using HTTP responses (200 OK or 201 Created)

This is a design choice

- If the added information needs to be accessible individually, create a new resource
- For changes of an existing resource, no new resource has to be created

Make sure that resources are navigable using URIs

- If appropriate, a relationship can be represented in the resource format

Content types

Representations should be machine-processable

- They don't have to, they may be opaque to applications
- In many cases, machine-processable representations are advantageous

Resources are abstractions, REST passes representations around

- Resources can have various representations (i.e., content types)
- Clients can request content types they are interested in

Adding or changing content types does not change the system architecture

- Different clients and servers support different content types
- Content Negotiation allows content types to be negotiated dynamically

Stateless interactions

For many RESTful applications, state is an essential part but the idea of REST is to avoid long-lasting transactions *in applications.*

Statelessness in this context means to move state to clients or resources and the most important consequence is to avoid state in server-side applications.

Resource state is managed on the server. It is the same for every client working with the service and when a client changes resource state other clients see this change as well.

Client state is managed on the client and it is specific for a client and thus has to be maintained by each client. It may affect *access* to server resources, but not the resources themselves.

State management

Essential for supporting stateless interactions

Cookies are a frequently used mechanism for managing state

- In many cases used for maintaining session state (login/logout)
- More convenient than having to embed the state in every representation

Cookies have two interesting client-side side-effects

- They are stored persistently independent from any representation
- They are “shared state” within the context of one browser

Cookie

A cookie is like a special bookmark shared between client and server in a "session." (RFC2109)

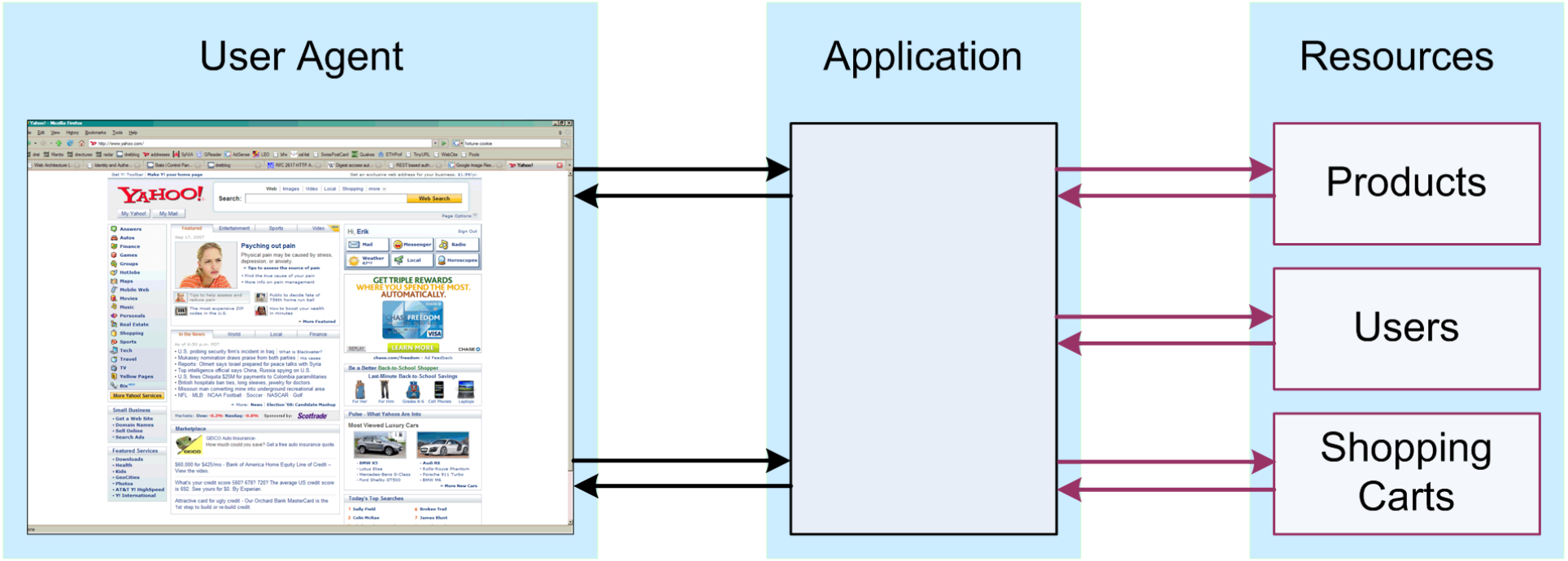
- Server sets a cookie in HTTP response headers
- Client will always include this cookie with requests
- Cookie can be any key-value pairs as text.
- Server typically uses the cookie to identify a client's data in a database.

Example in PHP

```
<?php
setcookie("user", "Alex Porter", time()+3600);
?>

<html>
.....
```

State in HTML or HTTP



Application State (User Session)

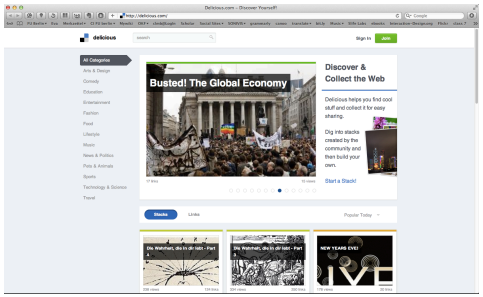
State as a Resource

Representational State Transfer (REST)

A RESTful Web service, an example

Please read the complete example on
http://www.peej.co.uk/articles/restfully-delicious.html

delicious



Delicious is a social bookmarking web service for storing, sharing, and discovering web bookmarks.

Delicious has “a simple REST API”, that means a simple POX over HTTP API or REST-RPC hybrid service

Delicious’s API isn't very RESTful. Why not?

- First class objects aren't exposed as resources, so bookmarks or tags can not be accessed directly.
- HTTP methods not used correctly, everything is done via GET even operations that change things.
- Resource representations not interconnected, you can't traverse from a list of bookmarks to a single bookmark.

What do we want to do with such a web service?

- Get a list of all our bookmarks and to filter that list by tag or date or limit by number
- Get the number of bookmarks created on different dates
- Get the last time we updated our bookmarks
- Get a list of all our tags
- Add a bookmark
- Edit a bookmark
- Delete a bookmark
- Rename a tag

Our two resources are bookmarks and tags:

- http://del.icio.us/api/[username]/bookmarks
- http://del.icio.us/api/[username]/tags

Getting Bookmarks

In the POX Delicious API bookmarks are accessed by a RPC style request to

https://api.del.icio.us/v1/posts/get

with a number of optional query string parameters that influence the returned results.

To do a similar thing RESTfully, we'll define a similar resource at

http://del.icio.us/api/[username]/bookmarks/

that returns a list of bookmarks.

We'll also define an infinite number of resources at

http://del.icio.us/api/[username]/bookmarks/[hash]

that represent our individual bookmarks where [hash] is the Delicious hash used to identify a bookmark.

Get all bookmarks

URL	http://del.icio.us/api/[username]/bookmarks/	
Method	GET	
Querystring	tag=	Filter by tag
	dt=	Filter by date
	start=	The number of the first bookmark to return
	end=	The number of the last bookmark to return
Returns	200 OK & XML (delicious/bookmarks+xml)	
	401 Unauthorized	
	404 Not Found	

An example delicious/bookmarks+xml document

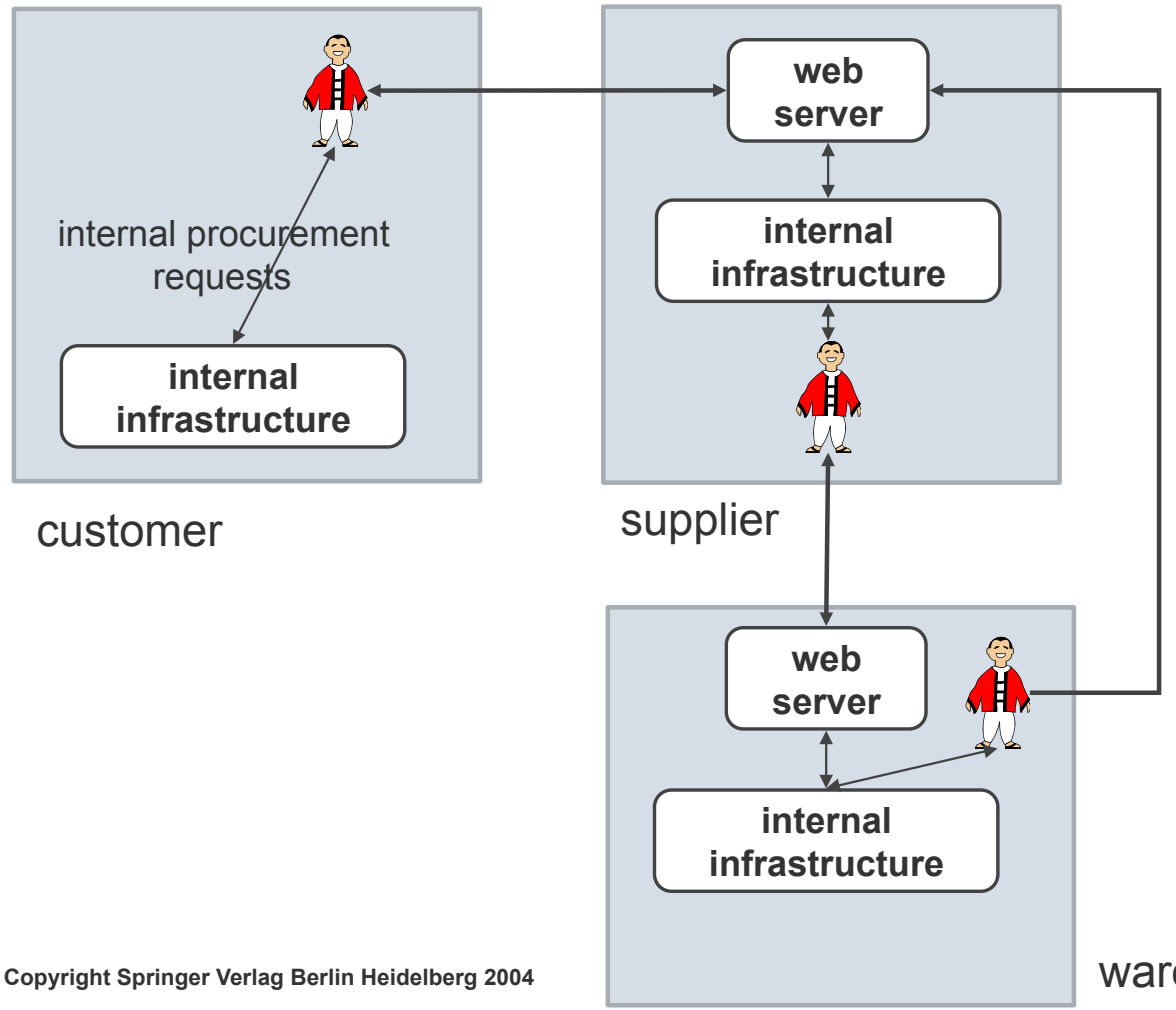
GET http://del.icio.us/api/peej/bookmarks/?start=1&end=2

```
<?xml version="1.0"?>
<bookmarks start="1" end="2"
   next="http://del.icio.us/api/peej/bookmarks?start=3&amp;end=4">
   <bookmark url="http://www.example.org/one" tags="example,test"
      href="http://del.icio.us/api/peej/bookmarks/a211528fb5108cddaa4b0d3aeccdbdcf"
      time="2005-10-21T19:07:30Z">
      Example of a Delicious bookmark
   </bookmark>
   <bookmark url="http://www.example.org/two" tags="example,test"
      href="http://del.icio.us/api/peej/bookmarks/e47d06a59309774edab56813438bd3ce"
      time="2005-10-21T19:34:16Z">
      Another example of a Delicious bookmark
   </bookmark>
</bookmarks>
```

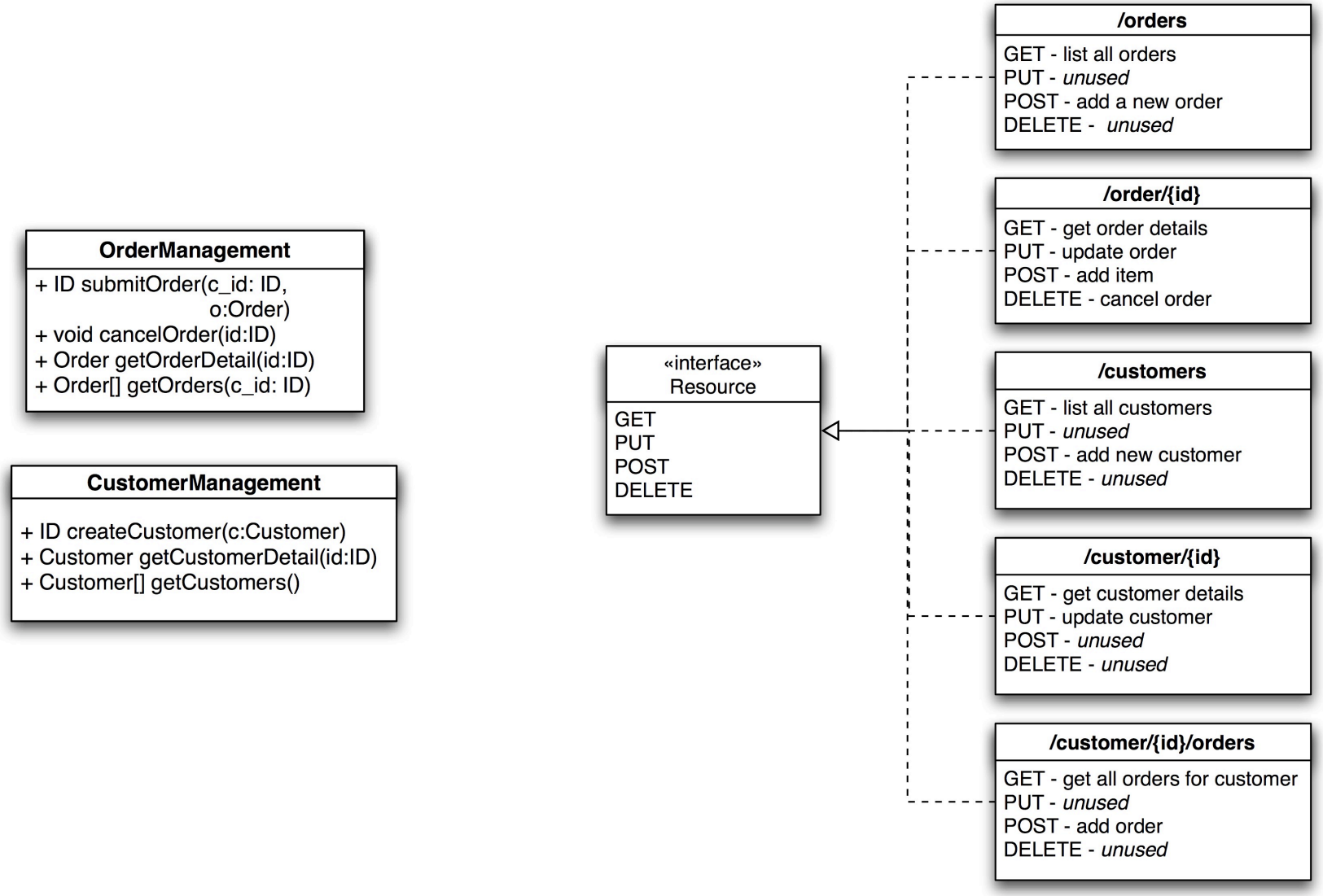
Representational State Transfer (REST)

SOAP vs. REST

Do you remember our example?



SOAP vs. REST



http://www.innoq.com/blog/st/2006/06/30/rest_vs_soap_oh_no_not_again.html

REST vs. SOAP-based web services

REST is a description of the Web's design principles

- It is not something new, it is simply a systematic view of the Web
- REST's claim is to be able to learn from the Web's success

Web Services (in their narrow sense) do not build on REST

- They use HTTP as a transport protocol
- They re-create Web functionality through additional specifications (WS-*)
- They have been built by programmers using a top-down approach

REST and Web Services have different design approaches

- REST starts at the resources and takes everything from there
- Web Services focus on messages, which in most cases are operations

Freie Universität Berlin

Web application development (part 1)

Hyper Text Markup Language (HTML) and Cascading Stylesheets (CSS)

Basics of HTML

HTML is not a programming language, it is a markup language which means it consists of a set of markup tags. These markup tags are used to describe web pages.

HyperText Markup Language (HTML) is the language for specifying the static content of Web pages (based on SGML, the Standard Generalized Markup Language)

- Hypertext refers to the fact that Web pages are more than just text; it can contain multimedia, provide links for jumping within the same document & to other documents

- Markup refers to the fact that it works by augmenting text with special symbols (tags) that identify the document structure and content type

If you don't know how to use it, check out: W3C School HTML Tutorial

Tags and Elements

HTML specifies a set of tags that are enclosed in < > which identify structure of the document and the content type

 specifies an image

Most tags come in pairs, marking a beginning and ending and an HTML element is an object enclosed by a pair of tags

<title>My Home Page</title> is a TITLE element

This text appears bold. is a BOLD element

Part of this text is bold.

is a PARAGRAPH element that contains a BOLD element

An HTML document is a collection of elements (text/media with context).

<head> and <body> elements

The <head> element is where you include a <title> element (that appears in the title bar of the browser).

- You can also include lots of other type of information in the <head> element.
- Cascading Style sheet information, or a link to an external style sheet (or several)
- “Meta” data, such as who authored the page, the type of content, and clues that search engines may (or may not) use to help categorize your page
- JavaScript code

The <body> element contains the main bulk of the material to be displayed on the webpage.

- Paragraphs, tables and lists as well as images
- JavaScript code
- PHP code can be included here too (if passed through a PHP parser)
- Other embedded objects

Example of a basic web page

```
<html>
<head>
    <title> Bill Smiggins Inc. </title>
  </head>
  <body>
    <h1>Bill Smiggins Inc.</h1>
    <h2>About our Company...</h2>
    <p>This Web site provides clients, customers,
      interested parties and our staff with all of
      the information that they could want on
      our products, services, success and failures.
    </p>
    <hr/>
    <h3> Products </h3>
    <p> We are probably the largest
    supplier of custom widgets, thingummybobs, and bits
    and pieces in North America. </p>
    <hr/>
  </body>
</html>
```

Text Appearance

You can specify styles for fonts

…	specify bold
…	specify italics
<tt>… </tt>	specify typewriter-like (fixed-width) font
<big>… </big>	increase the size of the font
…	decrease the size of the font
…	put emphasis
…	put even more emphasis
_…	specify a subscript
[…]	a superscript

….

Content vs. presentation

Most HTML tags define content type, independent of presentation.

Exceptions?

Style sheets associate presentation formats with HTML elements.

- CSS1: developed in 1996 by W3C
- CSS2: released in 1998, but still not fully supported by all browsers
- CSS3: specification still under development by the W3C, “completely backwards compatible with CSS2” (according to the W3C)

The trend has been towards an increasing separation of the content of webpages from the presentation of them.

Style sheets allow us to maintain this separation, which allows for easier maintenance of webpages, and for a consistent look across a collection of webpages.

Content vs. presentation (*cont.*)

Style sheets can be used to specify how tables should be rendered, how lists should be presented, what colors should be used on the webpage, what fonts should be used and how big/small they are, etc.

HTML style sheets are known as *Cascading Style Sheets*, since can be defined at three different levels

1. Inline style sheets apply to the content of a single HTML element
2. Document style sheets apply to the whole BODY of a document
3. External style sheets can be linked and applied to numerous documents, might also specify how things should be presented on screen or in print

(*Lower-level style sheets can override higher-level style sheets*)

Use-defined style sheets can also be used to override the specifications of the webpage designer. These might be used, say, to make text larger (e.g. for visually-impaired users).

HTML5

In HTML5 elements some elements of HTML 4.01 are obsolete, never used, or not used the way they were intended to and they are deleted or re-written.

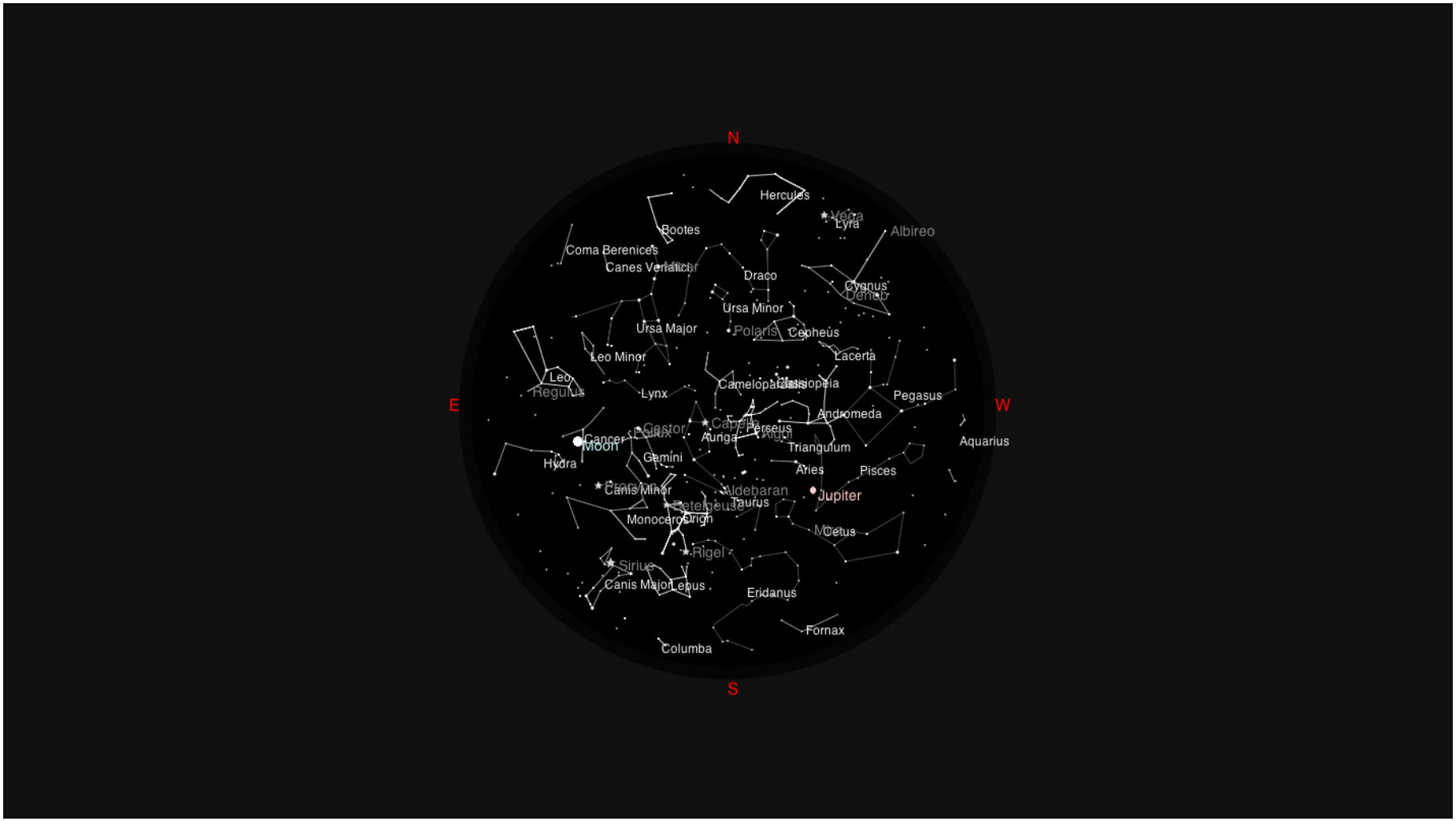
Some of the most interesting new features in HTML5:

- The canvas element for drawing
- The video and audio elements for media playback
- Better support for local offline storage
- New content specific elements, like article, footer, header, nav, section
- New form controls, like calendar, date, time, email, url, search

Note: HTML5 is not yet an official standard, and no browsers have full HTML5 support. Therefore, test always!

If you want to use it HTML5, check out: W3C School HTML5 Tutorial

One nice example: Art of Stars



Freie Universität Berlin

Web application development (part 1)

Common Gateway Interface (CGI)

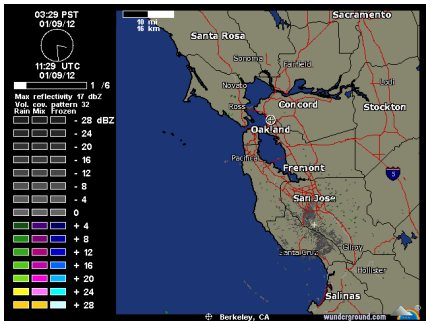
CGI basics

The Common Gateway Interface (CGI) is a standard (protocol) for interfacing external applications with an HTTP server. A CGI program is executed in real-time, so that it can output dynamic information. (RFC 3875)

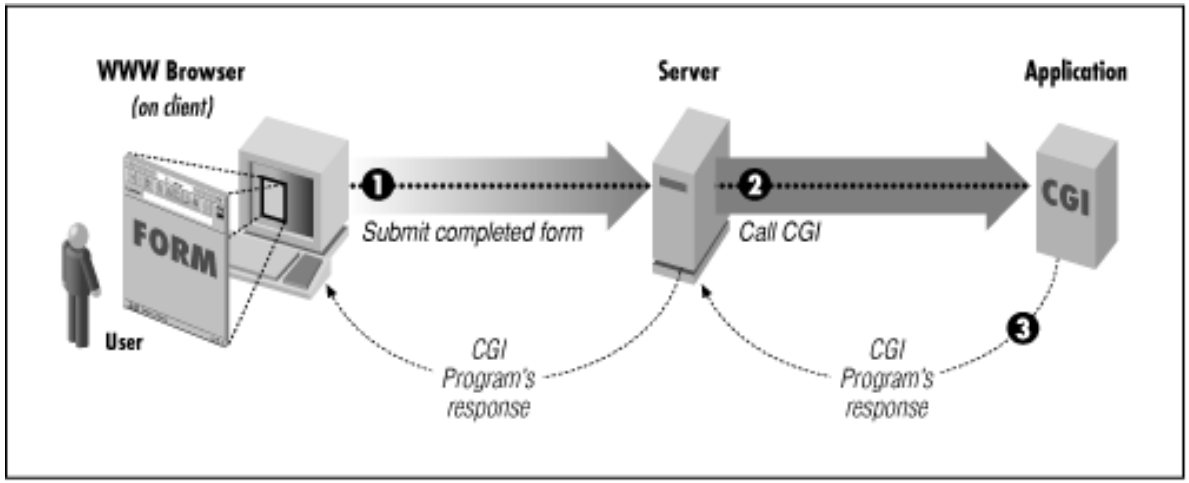
CGI defines the way by which web server software communicates with "backend programs" running on the server host. CGI is completely independent of programming language, operating system and web server.

CGI can be embedded into pages or produce complete pages

- embedded CGI results are expected to produce HTML fragments
- complete CGI results are expected to produce complete HTML content
- Check out example

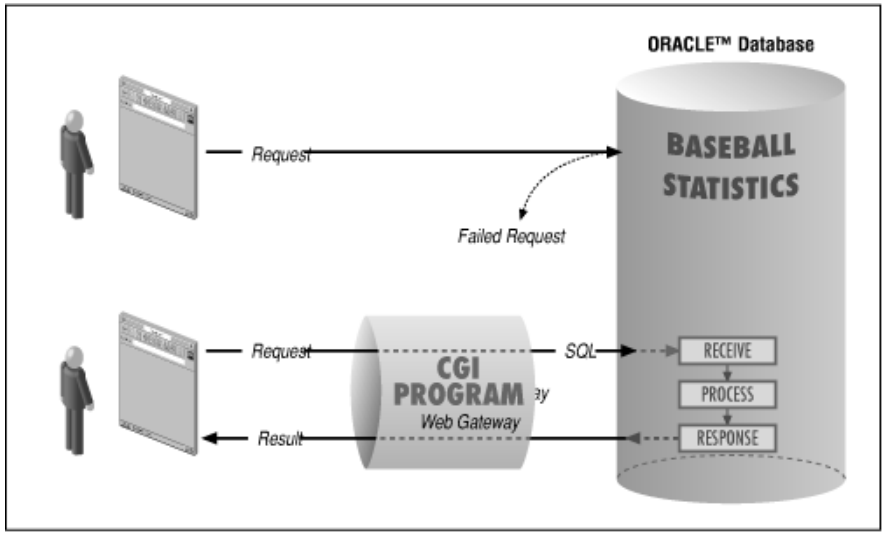


Application areas



The Web server can call up a program, while passing user-specific data to the program.

The program then processes that data and the server passes the program's response back to the Web browser.



Forms, e.g. shopping, booking.

Gateways, e.g. search engine, database

Virtual documents, e.g. guestbook, chat, bulletin board, dictionary.

Internal workings of CGI

Idea 1. CGI programs:

- a special directory (e.g. http://www.whatever.de/cgi-bin/cgiwrap/{user}/{script}),
- a certain file extension (e.g pl, cgi)

```
GET /cgi-bin/cgiwrap/claudia/welcome.cgi HTTP/1.1
Host: www.whatever.de
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Linux 2.4.25 i686) Opera 7.11  [en]
Connection: close
```

Idea 2. HTTP requests (GET, POST)

- specify URL (e.g. /cgi-bin/welcome.cgi)
- specify protocol (e.g. HTTP/1.1)
- specify accept formats
- specify user-agent (e.g. Mozilla/4.0)
- specify user's data

Idea 3. Web servers:

- recognize it is a CGI program
- execute the program (welcome.cgi)
- makes user's info available to the CGI program along with some server info

Internal workings of CGI (*cont.*)

Idea 4. Input to CGI program (UNIX)

- either from (STDIN), or
- from UNIX environment variables (%ENV hash in Perl)

Idea 5. Output from CGI program (UNIX)

- either to the client directly, or
- to STDOUT as a data stream

```
HTTP/1.1 200 OK
Date: Tue, 22 Nov 2005 18:25:25 GMT
Server: Apache/2.0.50 (Fedora)
Content-Length:91
Connection: close
Content-Type: text/html; charset=ISO-8859-1
<html>
<head><title>Welcome!</title></head>
<body><h1>Hello World!</h1>
</body></html>
```

```
Content-type: text/html

<html>
<head><title>Welcome!</title></head>
<body><h1>Hello World!</h1>
</body></html>
```

Idea 6. Output as data stream (UNIX)

HTTP header + a blank line + body

- if a complete HTTP header, to the client
- if not, the server must do it.

Programming in CGI

Now that I know how CGI works, what programming language can I use?

You can use whatever language you want, but it should ideally include

- Ease of text manipulation
- Ability to interface with other software libraries and utilities
- Ability to access environment variables (in UNIX)

Perl (UNIX, Windows, Macintosh) is the most widely used for CGI programming!

- Highly portable and readily available
- Powerful string manipulation operators
- Very simple and concise constructs
- Calling shell commands, and useful equivalents of certain UNIX system functions
- Extensions on top of Perl for specialized functions

Drawbacks

Calling a command generally means the invocation of a newly created process on the server.

Starting the process can consume much more time and memory than the actual work of generating the output, especially when the program still needs to be interpreted or compiled. If the command is called often, the resulting workload can quickly overwhelm the web server.

Possible solutions

- Server side *inside out programming* methods (ASP, *PHP, etc.*)
- Reducing the overhead involved in process creation such as FastCGI or run the application code entirely within the web server using extension modules such as mod_php

Freie Universität Berlin

Web application development (part 1)

Hypertext Preprocessor (PHP)

Example: Printing a dynamic date

Using CGI

- HTML is embedded within the script-code
- Problem is that it becomes unreadable if there is more HTML than code

```
print("<HTML>");
print("<H1>Sample</H1>");
var now = new Date();
print("It is now <strong>"+ now + "</strong>"); print("</HTML>");
```

Using PHP

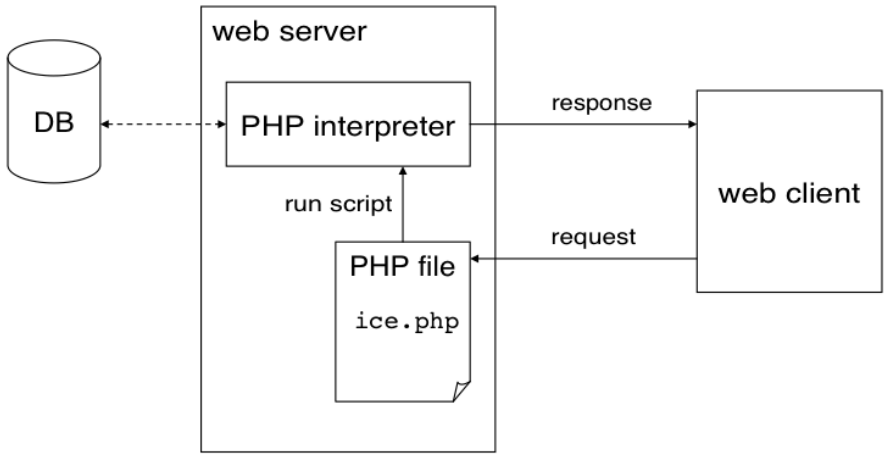
- Script-code is embedded in HTML instead of other way around!
- *Inside-out programming*

```
<HTML>
<H1>Sample</H1>
It is now <strong> <?php echo date ?> </strong>
</HTML>
```

PHP basics

In 1995 developed by Rasmus Lerdorf (member of the Apache Group)

- Originally designed as a tool for tracking visitors at Lerdorf's Web site
- Within 2 years, widely used in conjunction with the Apache server
- Developed into full-featured, scripting language for server-side programming
- Free, open-source, recent version PHP 5.3.8



New to PHP? Then go to this webpage.

Basic application areas

Server-side scripting

- Most traditional and main target field
- Ingredients: PHP parser (e.g., CGI), a web server and a web browser

Command line scripting

- Use a PHP script to run it without any server or browser
- Ingredients: PHP parser

Writing desktop applications

- Well, is not the very best language to create a desktop application with a graphical user interface
- Ingredients: you know PHP very well, and if you need some advanced PHP features in your client-side applications use PHP-GTK

Basic PHP syntax

A PHP scripting block always starts with <?php and ends with ?>. A PHP scripting block can be placed (almost) anywhere in an HTML document.

```
<html>
<!-- hello.php COMP519 -->
<head><title>Hello World</title></head>
<body>
 <p>This is going to be ignored by the PHP interpreter.</p>
   <?php echo ‘<p>While this is going to be parsed.</p> ‘; ?>
 <p>This will also be ignored by the PHP preprocessor.</p>
   <?php print(‘<p>Hello and welcome to <i>my</i> page!</p>'); ?>
</body>
</html>
```

print and echo
for output

a semicolon (;)
at the end of each statement

Variables and arrays

All variables in PHP start with a $ sign symbol. A variable's type is determined by the context in which that variable is used (i.e. there is no strong-typing in PHP).

```
$dutch_dessert = Vanillevla ;        // string
$x = 28;                             // integer
$pi = 3.1415;                        // double
$whatever = TRUE;                    // boolean
```

Variables are case-sensitive, beware! $alp5 != $ALP5

An array in PHP is actually an ordered map. A map is a type that maps values to keys.

```
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

array() = creates arrays

key = either an integer or a string.
value = any PHP type.

Functions in PHP

The PHP function reference is available on its web site. It shows the impressive list of functions within PHP.

For example:

date() is a built-in PHP function that can be called with many different parameters to return the date (and/or local time) in various formats

But you also can create your own functions. If you recreate one of the built-in functions, your own function will have no effect.

File Open

The fopen("file_name","mode") function is used to open files in PHP.

r	Read only.	r+	Read/Write.
w	Write only.	w+	Read/Write.
a	Append.	a+	Read/Append.
x	Create and open for write only.	x+	Create and open for read/write.

```
<?php
$fh=fopen("welcome.txt","r");
?>
```

For w, and a, if no file exists, it tries to create it (use with caution, i.e. check that this is the case, otherwise you will overwrite an existing file).

```
<?php
if
( !($fh=fopen("welcome.txt","r")) )
exit("Unable to open file!");
?>
```

For x if a file exists, this function fails (and returns 0).

If the fopen() function is unable to open the specified file, it returns 0 (false).

Form handling

Any form element is automatically available via one of the built-in PHP variables (provided that element has a “name” defined with it).

```
<html>
<-- form.html -->
<body>
<form action="welcome.php" method="POST">
Enter your name: <input type="text" name="name" /> <br/>
Enter your age: <input type="text" name="age" /> <br/>
<input type="submit" /> <input type="reset" />
</form>
</body>
</html>
```

```
<html>
<!–- welcome.php -->
<body>

Welcome <?php echo $_POST["name"].”.”; ?><br />
You are <?php echo $_POST["age"]; ?> years old!

</body>
</html>
```

$_POST
contains all POST data.

$_GET
contains all GET data.

Freie Universität Berlin

Web application development (part 1)

Summary

What have we discussed today?

- We discussed the basic features of a ROA – resource-oriented architecture
- We repeated the properties of POST.
- We talked about stateless interactions and cookies.
- We had a look at an example of an almost RESTful Web service.
- We discussed the differences of SOAP vs. REST
- We had a quick look of how to create static web pages with Hyper Text Markup Language (HTML) and Cascading Stylesheets (CSS).
- We extended our perspective by dynamic pages using Common Gateway Interface (CGI) and Hypertext Preprocessor (PHP).
- That was all.

References

Wilde, Erik: Web Architecture. Lecture INFO 290-03 (CCN 42584). UC Berkeley. 2008. URL: http://dret.net/lectures/web-fall08/

Roy Thomas Fielding: Architectural Styles and the Design of Network-based Software Architectures. Dissertation. University of California Irvine. 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm