The Architecture of the World Wide Web

Distributed Systems L-A
Sistemi Distribuiti L-A

Andrea Omicini
after Giulio Piancastelli
`andrea.omicini@unibo.it`

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2008/2009

Disclaimer

These slides contain material from [Fielding, 2000]

- A lot of material from that PhD Thesis has been re-used in the following, and integrated with new material according to the personal view of the teacher of this course
- A well-founded synthesis can be found in [Fielding and Taylor, 2002]
- Also, the following slides were first developed by Giulio Piancastelli
- Every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of the teacher of this course

Outline

Part I

Software Architecture

What is a Software Architecture?

Software architecture

A **software architecture** is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture

Architectural elements

A software architecture is defined by a configuration of **architectural elements**—components, connectors, and data—constrained in their relationships in order to achieve a desired set of architectural properties

Architectural Elements

Components

A **component** is an abstract unit of software instructions and internal state that provides a transformation of data via its interface

Connectors

A **connector** is an abstract mechanism that mediates communication, coordination, or cooperation among components

Data

A **datum** is an element of information that is transferred from a component, or received by a component, via a connector

Architectural Properties & Constraints

The set of **architectural properties** of a software architecture is derived from the selection and arrangement of components, connectors, and data within a system

- *functional properties*
- *quality attributes* such as ease of evolution, reusability of components, efficiency, and dynamic extensibility

Properties are induced by the *set of constraints* within an architecture

Architectural constraints are often motivated by the application of a software engineering principle to an aspect of the architectural elements

Architectural Styles

Architectural Style

An **architectural style** is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style

Architectural styles

- are a mechanism for categorizing architectures and defining their common characteristics
- provide an abstraction for the interactions of components, capturing the essence of a pattern of interaction by ignoring the accidental details of the rest of the architecture

Part II

Network-based Application Architectures

WWW as a Network-based Application

The World Wide Web is a **network-based** application because

- communication between components is restricted to *message passing*, unlike more general applications
- operations across the network are performed in a fashion that is *not necessarily transparent* to the user, unlike classic distributed systems that look to their users like ordinary centralized systems
- applications represent “business aware” functionalities, unlike operating systems, networking software, and support systems
- in application architectures the goals of a user action are representable as functional architectural properties, such as the location of information, performing requests, and rendering data streams
  - this is in contrast with e.g. a networking abstraction, where the goal is to move bits from one location to the other without regard to why those bits are being moved

Architectural Properties of Interest for Network-based Applications

- Performance
  - Network performance
  - User-perceived performance
- Scalability
- Simplicity
- Modifiability
  - Evolvability
  - Extensibility
  - Customizability
  - Configurability
  - Reusability
- Visibility
- Portability
- Reliability

Part III

Application Domain Requirements for the World Wide Web

The Application Domain of the World Wide Web

The major goal of the World Wide Web was to be a “shared information space through which people and machines could communicate.” That goal arose two basic needs

- a way for people to store and structure their own information
- a way to be able to reference and structure the information stored by others so that it would not be necessary for everyone to keep and maintain local copies

More requirements came from

- *distribution* of intended end-users located around the world
- *heterogeneity* of machines, operating systems, and file formats in use

The Web as a Distributed Hypermedia System

The World Wide Web was intended as a *distributed hypermedia system*

Hypermedia is defined by the presence of application control information embedded within, or as a layer above, the presentation of information

Distributed hypermedia allows the presentation and control information to be stored at remote locations

Simplicity

A low entry-barrier was necessary to enable sufficient adoption by *readers*, *authors*, and *application developers*

Readers

Hypermedia was chosen as the user interface because of

- simplicity and generality
- flexibility of relationships (links) allowing for unlimited structuring

Authors

Hypertext allowed partial availability of content and references without preventing their creation

Developers

Text-based protocols were the basis for simplifying application development

Extensibility

While simplicity makes it possible to deploy an initial implementation of a system, extensibility allows the system to evolve beyond the the limitations of what was initially deployed

Latency

User actions within a distributed hypermedia system require the transfer of large amounts of data from where the data is stored to where it is used

- the World Wide Web architecture must be designed for large-grain data transfer

The usability of hypermedia interaction is highly sensitive to user-perceived latency: the time between selecting a link and the rendering of a usable result

- the World Wide Web architecture needs to minimize network interactions

Scalability

The Web is intended to be an Internet-scale distributed hypermedia system

- the entire system is not under the control of a single entity
- the system is about interconnecting information networks across multiple organizational boundaries
- all entities participating in the system may be acting towards different or crossing purposes

Scalability

Architectural elements need to continue operating when they are subjected to unanticipated load, or when given malformed or maliciously constructed data, since they may be communicating with elements outside their organizational control

- the architecture must feature mechanisms enhancing visibility and scalability

Security

Multiple organizational boundaries implies that multiple trust boundaries could be present in any communication

Security

This requires that the architecture be capable of communicating authentication data and authorization controls. However

- authentication degrades scalability
- the architecture's default operation should be limited to actions that do not need trusted data

Independent Deployment

Multiple organizational boundaries also means that the system must be prepared for gradual and fragmented change

- old and new implementations co-exist
- old implementations must not prevent the new implementations from making use of their extended capabilities

Deployment

The architecture as a whole must be designed to ease the deployment of architectural elements in a partial, iterative fashion

- Existing architectural elements need to be designed with the expectation that architectural features will be added later
- Older implementations need to be easily identified so that legacy behavior can be encapsulated without adversely impacting newer architectural elements

Deriving the Web Architectural Style

From those requirements, an architectural style can be derived and used to define the principles behind the World Wide Web architecture. The formalization process for the Web architectural style works under two hypothesis:

Hypothesis I

The design rationale behind the WWW architecture can be described by an architectural style consisting of the set of constraints applied to the elements within the Web architecture

Hypothesis II

Constraints can be added to the WWW architectural style to derive a new hybrid style that better reflects the desired properties of a modern Web architecture

Part IV

The Representational State Transfer (ReST) Architectural Style

Deriving ReST as the Web Architectural Style

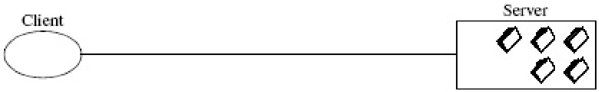
The design rationale behind the Web architecture can be described by an architectural style consisting of the set of constraints applied to elements within the architecture

By examining the impact of each constraint as it is added to the evolving style, we can identify the properties induced by the Web constraints

Starting From the Null Style

The Null style starts with the system needs as a whole, without constraints, and then constraints are incrementally identified and applied to elements of the system in order to differentiate the design space and allow the forces that influence system behavior to flow naturally, in harmony with the system

Client-Server (I)



Principle

Separation of concerns (between user interface and data storage)

Constraints

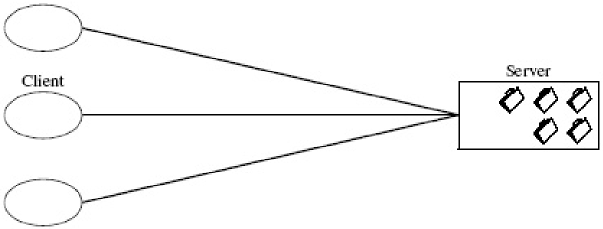
A server component, offering a set of services, listens for requests upon those services. A client component, desiring that a service be performed, sends a request to the server via a connector. The server either rejects or performs the request and sends a response back to the client.

Client-Server (II)

Properties

- Improve *portability* of the user interface across multiple platforms
- Improve *scalability* by simplifying the server components
- Allow components to *evolve* independently

Stateless (I)



Constraint

Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server

Stateless (II)

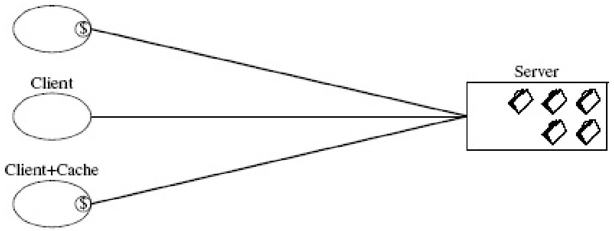
Properties

- Improve *visibility* by allowing monitoring systems to determine the full nature of a request without looking beyond it
- Improve *reliability* by easing the recovering from partial failures
- Improve *scalability* by allowing server components to quickly free resources
- Improve *simplicity* by simplifying implementation because the server does not have to manage resource usage across requests

Disadvantages

- Decrease network performance by increasing the per-interaction overhead repetitive data sent in a series of requests
- Reduce the server control over consistent application behavior

Cache (I)



Constraint

Data within a response to a request be implicitly or explicitly labeled as cacheable or not. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

Cache (II)

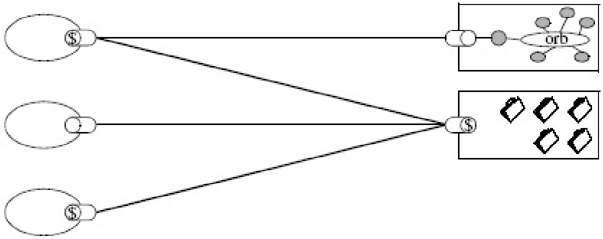
Properties

- Improve *efficiency*, *scalability*, and *user-perceived performance* by reducing the average latency of a series of interactions

Disadvantages

- Decrease *reliability* if stale data within the cache differs significantly from the data that would have been obtained had the request been sent directly to the server

Uniform Interface (I)



The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components

Principle

Generality (applied to the component interface)

Uniform Interface (II)

Constraint

Multiple architectural constraints are needed to guide the behavior of components:

- identification of resources
- manipulation of resources through representations
- self-descriptive messages
- hypermedia as the engine of application state

Properties

- *Simplify* the overall system architecture
- Improve the *visibility* of interactions
- Encourage independent *evolvability* by decoupling implementations from the services they provide

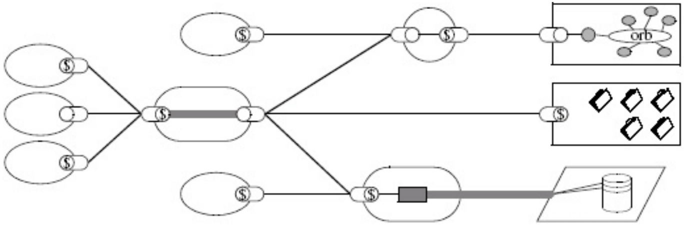
Uniform Interface (III)

The ReST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction

Disadvantages

- Degrades *efficiency* by transferring information in a standardized form rather than one which is specific to an application's needs

Layered System (I)



Constraint

Compose an architecture of hierarchical layers by constraining component behavior such that each component cannot see beyond the immediate layer with which they are interacting

Layered System (II)

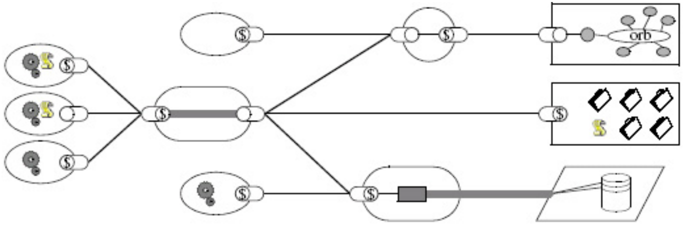
Properties

- Improve the overall system *simplicity* and promote *independence* by restricting knowledge of the system to a single layer
- Improve system *scalability* by enabling load balancing of services at intermediaries
- Improve *security* by allowing policies to be enforced on data crossing organizational boundaries

Disadvantages

- Reduce *user-perceived performance* by adding overhead and latency to data processing
  - the effect is countered by shared caching at intermediaries

Code-On-Demand (I)



It is an *optional* constraint, so that the architecture only gains the benefit and suffer the disadvantage of it when they are known to be in effect for some realm of the overall system

Constraint

Client functionalities can be extended by downloading and executing code (typically in the form of applets and scripts)

Code-On-Demand (II)

Properties

- *Simplify* clients by reducing the number of features required to be pre-implemented
- Improve system *extensibility* by allowing features to be downloaded after deployment

Disadvantages

- Reduce *visibility* and thus is only an optional constraint

Part V

ReST Architectural Elements

ReST General Overview

The Representational State Transfer (ReST) is an abstraction of the architectural elements within a distributed hypermedia system. ReST ignores the details of component implementation and protocol syntax in order to focus on

- the roles of components
- the constraints upon interaction between components
- the component interpretation of significant data elements

ReST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource. Whether the representation is in the same format as the raw source, or is derived from the source, remains hidden behind the component uniform interface.

ReST Data Elements

Data Element	**Modern Web Examples**
resource	the conceptual target of a hypertext link
resource identifier	URI (URL, URN)
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Resources

The key abstraction of information in ReST is a **resource**. Any information that can be named and is important enough to be referenced as a thing in itself can be a resource:

- a document
- an image
- a temporal service
  - e.g. today's weather in any Italian city
- a collection of other resources
  - e.g. a list of open bugs in a bug database
- a non-virtual object
  - e.g. a physical object like a lamp, an abstract concept like fear

Resources as Conceptual Mappings

A resource is a **conceptual mapping** to a set of entities, not the entity that corresponds to the mapping at any particular point in time. The entities in the set can be:

- resource *representations*
- resource *identifiers*

A resource can map to the empty set

- References can be made to a concept before any realization of that concept exists

Resources can be

- static, in the sense that, when examined at any time after their creation, they always correspond to the same entity set
- dynamic, otherwise

The only thing that is required to be static for a resource is the semantics of the mapping, since the semantics is what distinguishes one resource from another

Resources and the Web Architecture

The abstract definition of resources enables key features of the Web architecture

- provides *generality* by encompassing many sources of information
  - avoids artificially distinguishing information sources by type or implementation
- allows *late binding* of the reference to a representation
  - enables content negotiation to take place based on characteristics of the request
- allows an author to reference the concept rather than some singular representation of that concept
  - thus removing the need to change all existing links whenever the representation changes

Resource Identifiers

Each resource has to have at least one identifier in the form of a **URI** (RFC 2396, [Berners-Lee et al., 1998]). The URI is the name and address of a resource.

> *The URI is the fundamental technology of the Web. There were hypertext systems before HTML, and Internet protocols before HTTP, but they didn't talk to each other. The URI interconnected all these Internet protocols into a Web.*
>
> *The web kills off other protocols because it has something most protocols lack: a simple way of labeling every available item. Every resource on the Web has at least one URI.*

Design Guidelines

URIs should have a structure. Their structure should vary in predictable ways.

The Relationship Between URIs and Resources

No two resources can be the same, since each resource maps a different concept

However, at some moment in time, two different resources may point to the same data, e.g.

- `http://example.com/software/release/1.0.3/`
- `http://example.com/software/release/latest/`

A resource may have one URI or many. Every URI designates exactly one resource.

Design Guidelines

A resource and its URI ought to have an intuitive correspondence

Representations

A **representation** is a sequence of bytes, plus *representation metadata* to describe those bytes

- Other commonly used but less precise names for a representation include: document, file, HTTP message

A representation contains *any useful information* about the current state of a resource

Representations Metadata

Representation **metadata** is in the form of name-value pairs, where the name corresponds to a standard that defines the structure and semantics of the value. Response messages may include both

- representation metadata, and
- *resource metadata*, i.e. information about the resource that is not specific to the supplied representation

Control Data

Control data defines the purpose of a message between components

- e.g. the action being requested or the meaning of a response

Control data is also used to parameterize requests and override the default behavior of some connecting elements

- e.g. cache behavior

Media Types

The data format of a representation is known as a **media type**

A representation can be included in a message and processed by the recipient according to the control data of the message and the nature of the media type

The design of a media type can directly impact the *user-perceived performance* of a distributed hypermedia system.

- Any data that must be received before the recipient can begin rendering the representation adds to the latency of an interaction

Connectors

Connector	**Modern Web Examples**
client	libwww, libwww-perl
server	libwww, Apache API, NSAPI
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT
cache	browser cache

- provide a generic interface for accessing and manipulating the value set of a resource
  - enhancing *simplicity* by providing a clean separation of concerns
- encapsulate the activities of accessing resources and transferring resource representations
  - enabling *substitutability* by hiding the underlying implementation of resources and communication mechanisms

Connector Types

Client and Server

The primary connector types are client and server

- a client initiates communication by making a request
- a server listens for connections and responds to requests in order to supply access to its services

Resolver

A resolver translates partial or complete resource identifiers into the network address information needed to establish an inter-component connection

Tunnel

A tunnel simply relays communication across a connection boundary, such as a firewall or lower-level network gateway. Some ReST components (e.g. proxy) may dynamically switch from active component behavior to that of a tunnel.

Cache (I)

The cache connector is located on the interface to a client or server connector in order to save cacheable responses to current interactions so that they can be reused for later requested interactions

Some cache connectors are shared, meaning that cached responses may be used in answer to a client other than the one for which the response was originally obtained

- can be an effective way to reduce the impact of “flash crowds” on the load of a popular server
- can also lead to errors if the cached response does not match what would have been obtained by a new request

Cache (II)

A cache is able to determine the cacheability of a response because the interface is generic rather than specific to each resource.

Default cache behavior can be overridden by including proper control data in the interaction.

Components (I)

Component	**Modern Web Examples**
origin server	Apache httpd, Microsoft IIS
gateway	Squid
proxy	CERN Proxy, Netscape Proxy
user agent	Mozilla Firefox, Safari

Origin Server

Uses a server connector to govern the namespace for a requested resource

- the server is the definitive source for representations of its resources and must be the ultimate recipient of any request that intends to modify the value of its resources
- provides a generic interface to its services as a resource hierarchy

Components (II)

User Agent

Uses a client connector to initiate a request and becomes the ultimate recipient of the response

Proxy and Gateway

Intermediary components act as both a client and a server in order to forward, with possible translation, requests and responses

- a client determines when it will use a proxy
- a gateway is imposed by the network or origin server

Part VI

The HyperText Transfer Protocol

HyperText Transfer Protocol (HTTP)

ReST components on the World Wide Web communicate through the **HTTP protocol** (RFC 2616, [Fielding et al., 1999]). HTTP is a synchronous document-based protocol where transactions are carried out in two steps

1. the client creates a HTTP **request** by putting a document in an envelope, and sends it to the server
2. the server creates a HTTP **response** by putting a response document in an envelope, and sends it to the client

HTTP Request Example

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15;utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

HTTP Request Example: the HTTP Method

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15;utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

The name of the HTTP method indicates how the client expects the server to process this request

HTTP Methods

Unlike RPC, requests in HTTP are directed to resources using a generic interface with standard semantics that can be interpreted by intermediaries almost as well as by the machines that originated services

The most important methods in the generic HTTP interface and their semantics are

GET	to retrieve a representation of the resource in an idempotent way
POST	to create a new subordinate entity of the specified resource using the content sent in the request
PUT	to create or modify the specified resource using the content sent in the request
DELETE	specifying that the resource must be deleted

HTTP Request Example: The Path

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15;utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

This is the portion of the URI to the right of the host name and indicates which part of the server data the client expects to be processed by this request

HTTP Request Example: The Headers

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15;utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Request headers are key-value pairs of metadata. There is a standard list of HTTP headers, and applications can define their own.

HTTP Request Example: The Representation

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15;utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

The representation (also called “entity body”) is the document inside the HTTP envelope. HTTP GET requests have no entity body.

HTTP Response Example

```
HTTP/1.1 200 OK
Date: Fri, 17 Nov 2006 15:36:32 GMT
Server: Apache
Last-Modified: Fri, 17 Nov 2006 09:05:32 GMT
Etag: "7359b7-a7fa-455d8264
Accept-Ranges: bytes
Content-Length: 43302
Content-Type: text/html
X-Cache: MISS from www.oreilly.com
Keep-Alice: timeout=15, max=1000
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
</html>
```

HTTP Response Example: the Response Code

```
HTTP/1.1 200 OK
Date: Fri, 17 Nov 2006 15:36:32 GMT
Server: Apache
Last-Modified: Fri, 17 Nov 2006 09:05:32 GMT
Etag: "7359b7-a7fa-455d8264
Accept-Ranges: bytes
Content-Length: 43302
Content-Type: text/html
X-Cache: MISS from www.oreilly.com
Keep-Alice: timeout=15, max=1000
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
</html>
```

The response code is a numeric code that tells the client whether its request has been successfully processed or not, and how the client should therefore regard the response

HTTP Response Example: the Headers

```
HTTP/1.1 200 OK
Date: Fri, 17 Nov 2006 15:36:32 GMT
Server: Apache
Last-Modified: Fri, 17 Nov 2006 09:05:32 GMT
Etag: "7359b7-a7fa-455d8264
Accept-Ranges: bytes
Content-Length: 43302
Content-Type: text/html
X-Cache: MISS from www.oreilly.com
Keep-Alice: timeout=15, max=1000
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
</html>
```

Response headers are key-value pairs of metadata. There is a standard list of HTTP headers, and applications can define their own headers.

HTTP Response Example: the Representation

```
HTTP/1.1 200 OK
Date: Fri, 17 Nov 2006 15:36:32 GMT
...
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
...
</html>
```

The representation contained in the response is the fulfillment of the request.

The *media type* of the representation is given in the `Content-Type` header. The indication of a media type lets clients correctly display the representation of the request target resource.

Method and Scoping Information

The **method information** in a HTTP request indicates how the client expects the server to process the request

The **scope information** indicates on which part of the data set the server should operate the method requested by the client

On systems respectful of ReST constraints, such as the World Wide Web, the method information is contained in the HTTP request method and the scope information is the URI (host + path) of the resource to which the request is directed.

Bibliography

Berners-Lee, T., Fielding, R., and Masinter, L. (1998).
Uniform Resource Identifiers (URI): Generic syntax.
http://www.ietf.org/rfc/rfc2396.txt.
Internet RFC 2396.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999).
Hypertext Transfer Protocol – HTTP/1.1.
http://www.w3.org/Protocols/rfc2616/rfc2616.html.
Internet RFC 2616.

Fielding, R. T. (2000).
Architectural Styles and the Design of Network-based Software Architectures.
PhD thesis, University of California, Irvine, CA, USA.

Fielding, R. T. and Taylor, R. N. (2002).
Principled design of the modern Web architecture.
ACM Transactions on Internet Technology, 2(2):115–150.

Richardson, L. and Ruby, S. (2007).
RESTful Web Services.
O'Reilly.

The Architecture of the World Wide Web

Distributed Systems L-A
Sistemi Distribuiti L-A

Andrea Omicini
after Giulio Piancastelli
`andrea.omicini@unibo.it`

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2008/2009