

Web

Master 1 IFI



Andrea G. B. Tettamanzi

Université de Nice Sophia Antipolis

Département Informatique

andrea.tettamanzi@unice.fr

Unit 5

Persistence, AJAX, and REST

Agenda

- Persistence (and serialization)
- Event-Driven Programming in JavaScript
- AJAX
- Logical Organization of a Web Application
- REST

Persistence

- General mechanism to save and restore
 - Data
 - State of computation
- Allow a program to terminate without its data and execution state being lost
- A central subject in Web programming
 - Server-side and client-side
 - Web protocols are stateless!
- Saving
 - Locally on disk
 - Remotely on a server
 - E.g., a relational DB server

Client-Side Persistence (JavaScript)

- Client-side persistence requires serialization
- Serialization = translating data structures or object state into a format that can be stored or transmitted and reconstructed later
- Possible choices for serialization
 - Binary
 - XML
 - JSON
- Two persistence mechanisms for client-side persistence
 - HTTP Cookies (traditional solution, predates JavaScript)
 - Web Storage API (convenient for small amounts of data)
 - IndexedDB API (suitable for large, structured data)

XML

- XML : e**X**tensible **M**arkup **L**angage
- W3C standard since 1998
- Version 1.0 (February 1998); Version 1.1 (February 2004)
- A description language for a class of data objects called “XML documents”
- The standard partially describes the behavior of programs that process them
- XML is a restricted form of SGML (1986)

XML Document Example

```
<!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<glossary><title>example glossary</title>
  <GlossDiv><title>S</title>
    <GlossList><GlossEntry ID="SGML" SortAs="SGML">
      <GlossTerm>Standard Generalized Markup Language</GlossTerm>
      <Acronym>SGML</Acronym>
      <Abbrev>ISO 8879:1986</Abbrev>
      <GlossDef>
        <para>A meta-markup language, used to create markup
          languages such as DocBook.</para>
        <GlossSeeAlso OtherTerm="GML">
        <GlossSeeAlso OtherTerm="XML">
      </GlossDef>
      <GlossSee OtherTerm="markup">
    </GlossEntry></GlossList>
  </GlossDiv>
</glossary>
```

XML Documents

- Consist of storage units called entities, containing data, which may be parsed or not
- Data are characters representing either simple values or tagged content
- The markup (tags) describes the logical/storage structure of the document
- An XML document is **well-formed** if it respects XML syntax
- XML provides a mechanism to constrain this syntax, namely the DTD (Document Type Definition)
- An XML document may be **valid** w/ respect to one or more DTDs

DTD

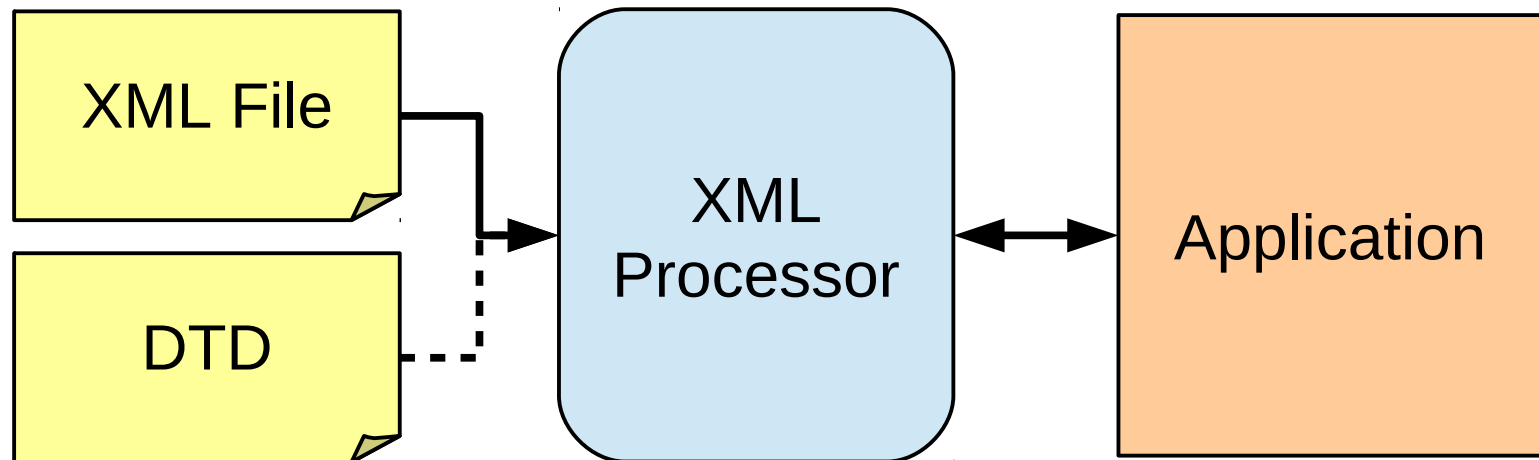
- Element Definition: `<! ELEMENT Name Content >`
- Content:
 - ANY: The element may contain data of any type
 - EMPTY: The element contains no data
 - #PCDATA: The element must contains a character string (parsed character data)
 - Operators:
 - + at least one
 - * zero or more
 - ? optional
 - | alternative
 - , concatenation
 - () grouping

DTD Example

```
<!ELEMENT contact (address, url?, tel*)>  
<!ELEMENT address (addrline, zip, city)>  
<!ELEMENT addrline (line+)>  
<!ELEMENT line (#PCDATA)>  
<!ELEMENT zip (#PCDATA)>  
<!ELEMENT city (#PCDATA)>  
<!ELEMENT url (#PCDATA)>  
<!ELEMENT tel (#PCDATA)>  
<!ATTLIST tel t (fixed|fax|mobile) "fixed">
```

XML Processor

- A software module used to read XML documents and access their content (with or without validation)
- An XML processor does the work on behalf of another module, an application



XML in JavaScript

- The XML processor in JavaScript is the DOM API
- The XML DOM defines the properties and methods for accessing and editing XML
- Before an XML document can be accessed, it must be loaded into an XML DOM object
- DOMParser

```
parser = new DOMParser();  
xmlDoc = parser.parseFromString(xml, "text/xml");  
title = xmlDoc.getElementsByTagName("title")[0];  
...
```

JSON

- JSON = JavaScript Object Notation
 - Lighter than XML
 - Native to JavaScript
 - Platform- and Programming-language-independent

object ::= '{' | '{' members '}'

members ::= pair | pair ',' members

pair ::= string ':' value

array ::= '[' | '[' elements ']'

elements ::= value | value ',' elements

value ::= string | number | object | array | 'true' | 'false' | 'null'

JSON Example

```
{ "glossary":  
  { "title": "example glossary",  
    "GlossDiv": {  
      "title": "S",  
      "GlossList": {  
        "GlossEntry": {  
          "ID": "SGML",  
          "SortAs": "SGML",  
          "GlossTerm": "Standard Generalized Markup Language",  
          "Acronym": "SGML",  
          "Abbrev": "ISO 8879:1986",  
          "GlossDef": {  
            "para": "A meta-markup language, used to create  
                    markup languages such as DocBook.",  
            "GlossSeeAlso": ["GML", "XML"] },  
            "GlossSee": "markup" } } }  
      }  
    }  
  }  
}
```

The JSON Object

- **JSON.stringify**(*value*[, *replacer*[, *space*]])
 - Returns a JSON representation of *value* as a string
 - Throws a `TypeError` exception if *value* is cyclic
 - *space* is a `Number` (0, 1, ...) or `String` (like `""`, `" "`, `"\t"`, `"\n"`)
 - *replacer* is a function(*key*, *value*) → JSON string
- **JSON.parse**(*json_string*[, *reviver*])
 - Throws a `SyntaxError` exception if *json_string* is not valid
 - *reviver* is a function(*key*, *json_string*) → value

Cookies

- An HTTP cookie is a small piece of data that a server sends to the user's browser
- The browser may store it and send it back with the next request to the same server
- Cookies are used for
 - Session management
 - Personalization (preferences, themes, settings)
 - Tracking (bad, bad, ...)
 - General client-side persistence (but that's old-fashioned)
- In JavaScript:
 - `document.cookie` is a pseudo-variable to access cookies

Web Storage API

- Two storage object are available in an HTML5 client:
 - **sessionStorage** maintains a separate storage area for each given origin that's available for the duration of the page session (as long as the browser is open, including page reloads and restores)
 - **localStorage** does the same thing, but persists even when the browser is closed and reopened
- They expose the same interface:
 - length: the number of data items stored in the area
 - setItem(key, serializedValue), getItem(key), removeItem(key)
 - Clear() – clear the whole area, key(n) – returns the *n*th key
 - The “storage” event is fired when the a storage area changes

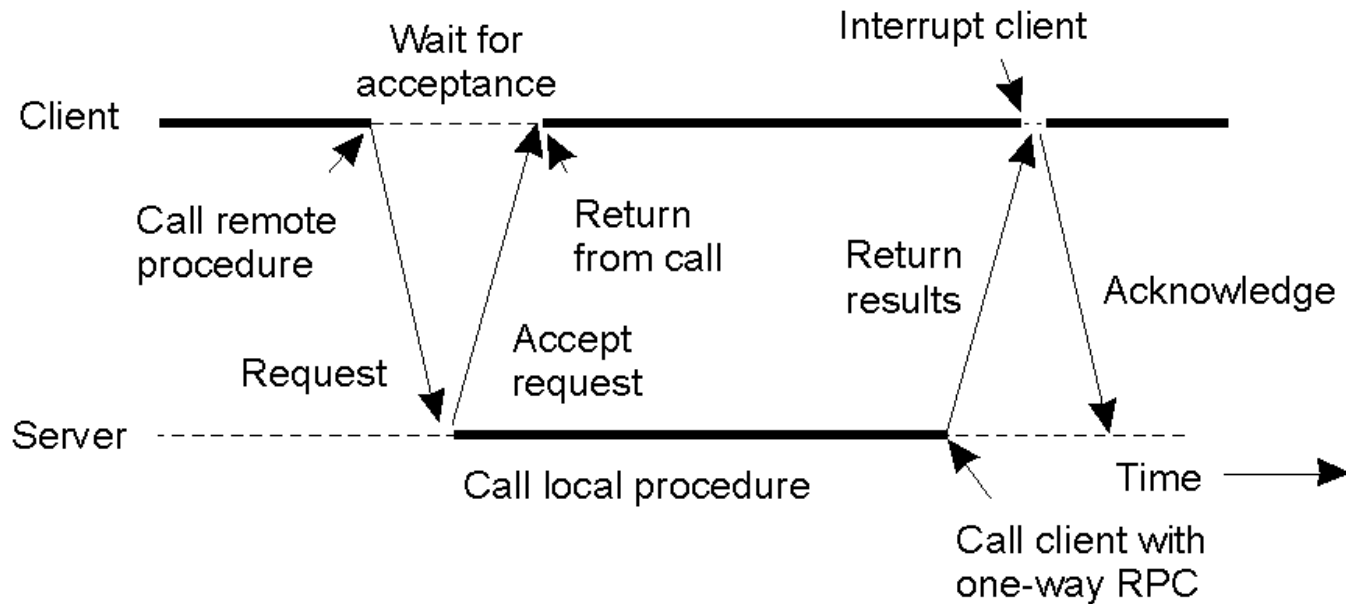
Event-Driven Programming

- A programming style based on events
- Contrasts with traditional sequential programming
- The program is mainly defined by its reactions to the various events that can occur
 - Variable state changes
 - User actions
 - Message reception
 - End of a long task delegated to another process/thread
- Much used in graphical user interfaces (GUIs)
- Typical of distributed systems
 - Parallelism and asynchronous execution
- Node.js is a JavaScript-based event-driven software platform

Callback Functions

- Web programming promotes an event-driven style for two main reasons
 - Web clients often implement reactive GUIs
 - Web applications are distributed (client-server)
- In this style, so-called *callback* functions are first-class citizens
 - HTML event handlers are one remarkable example
 - Interrupt handlers in system programming are another
- A callback function is a function
 - Passed as an argument to another function/method
 - This latter calls it (asynchronously) at the end of its execution, to pass back some results, for instance
 - Other uses/schemes are possible and popular

Asynchronous Remote Procedure Call



AJAX

- Acronym of “Asynchronous JavaScript and XML”
- Principle: to make an asynchronous request to a server/service
 - To verify/validate
 - Po get/retrieve information
- Asynchronous HTTP request (to a CGI script or similar)
- The HTTP response will consist of XML (or JSON, or other...)
 - Such content will then be exploited to change the Web page (e.g., getElementById + innerHTML)
- The whole process does not require to reload the current page or to download a new page!

AJAX: XMLHttpRequest

- A JavaScript constructor providing an API to exchange data between an HTTP client and an HTTP server:
 - <http://www.w3.org/TR/XMLHttpRequest/>
- The name has historical reasons, however
 - Any textual format can be used, not just XML
 - It allows to use HTTP as well as SHTTP (or other protocols)
 - Request, here, is in a very broad sense (any HTTP method)
- Working principle:
 - Create an object with this constructor
 - Assign a reference to a handler (i.e., a callback function) to the “onreadystatechange” property/event
 - Call the open() et send() methods to submit the request

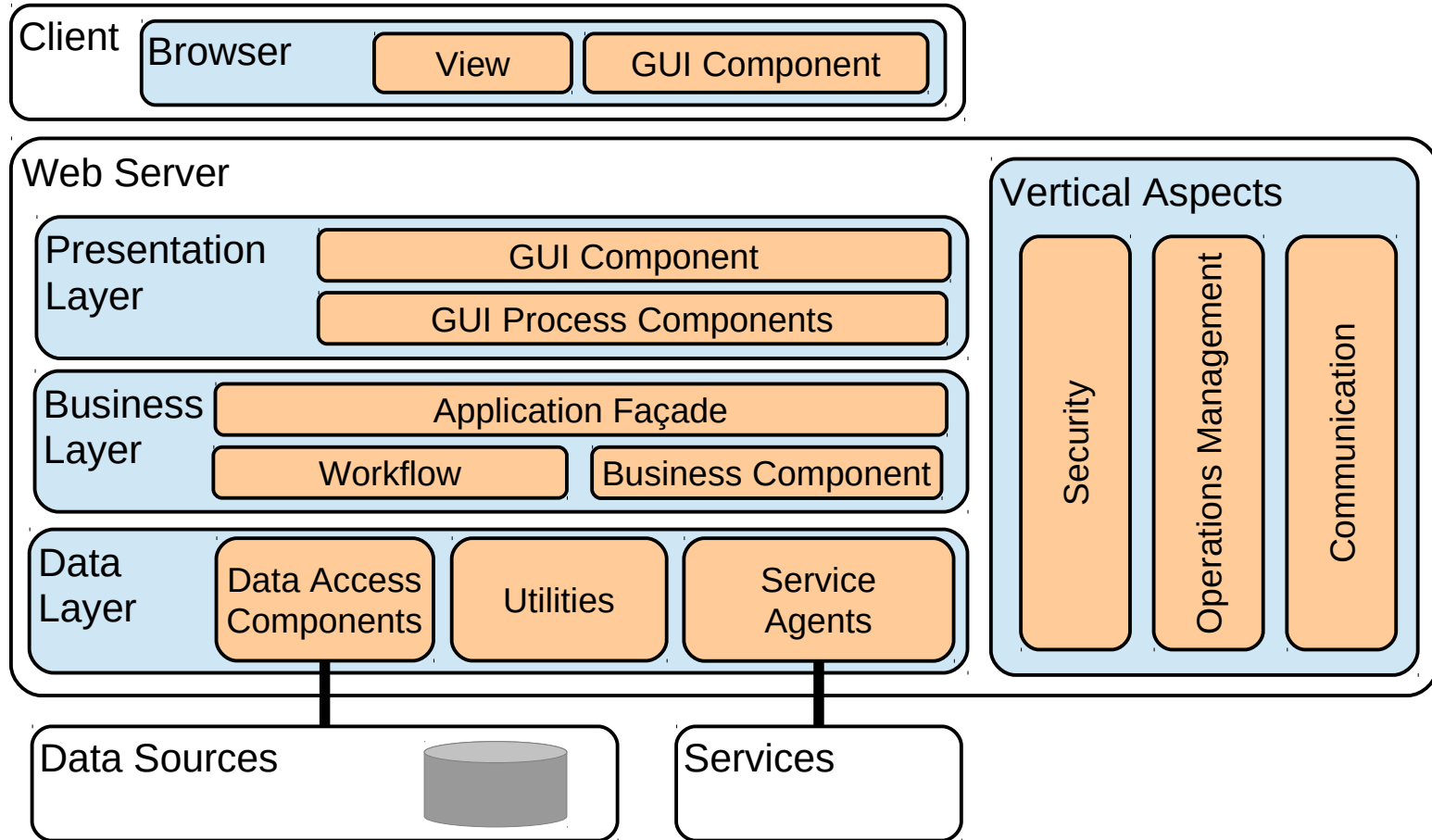
Handler Example

```
function my_handler( ) {  
    // test the processing state of the request  
    if ((this.readyState==4) && (this.status==200))  
    {  
        // retrieve the response in XML or text format  
        var myXML = this.responseXML; // a DOM object  
        var myText = this.responseText; // a string  
        // ... do something with the response  
    }  
}
```

Using XMLHttpRequest

```
var client = new XMLHttpRequest();  
client.onreadystatechange = my_handler;  
  
client.open("GET", url);  
client.send();  
  
// alternatively:  
client.open("POST", url);  
client.setRequestHeader("Content-Type",  
    "text/plain;charset=UTF-8");  
client.send("var1=val1&var2=val2&...");
```


Architecture of a Web Application



Request Handling

- Two approaches to request handling:
 - “Post-Back” Approach (we know that already, cf. CGIs)
 - Form-based development
 - Server-centric
 - Rapid prototyping
 - “RESTful” Approach, based on the notion of Web service
 - More client-centric
 - Finer control of the GUI and more flexibility
- The choice between these two approaches must take into account the desired degree of control on GUI, the development process and scalability

Representational State Transfer (REST)



- Architectural style for distributed hypermedia systems
- Proposed in 2000 by Roy Fielding
- Defined by six architectural constraints:
 - 1) Client-server architecture: clear separation of concerns
 - 2) Statelessness: each requests must also contain its context
 - 3) Cacheability: responses must define if they are cacheable
 - 4) Uniform Interface: (a) resource identification in requests;
(b) resource manipulation through representations;
(c) self-descriptive messages;
(d) Hypermedia as the engine of application state (HATEOAS)
 - 5) Layered System
 - 6) Code on Demand (→ client-side scripting)

