REST Web Services

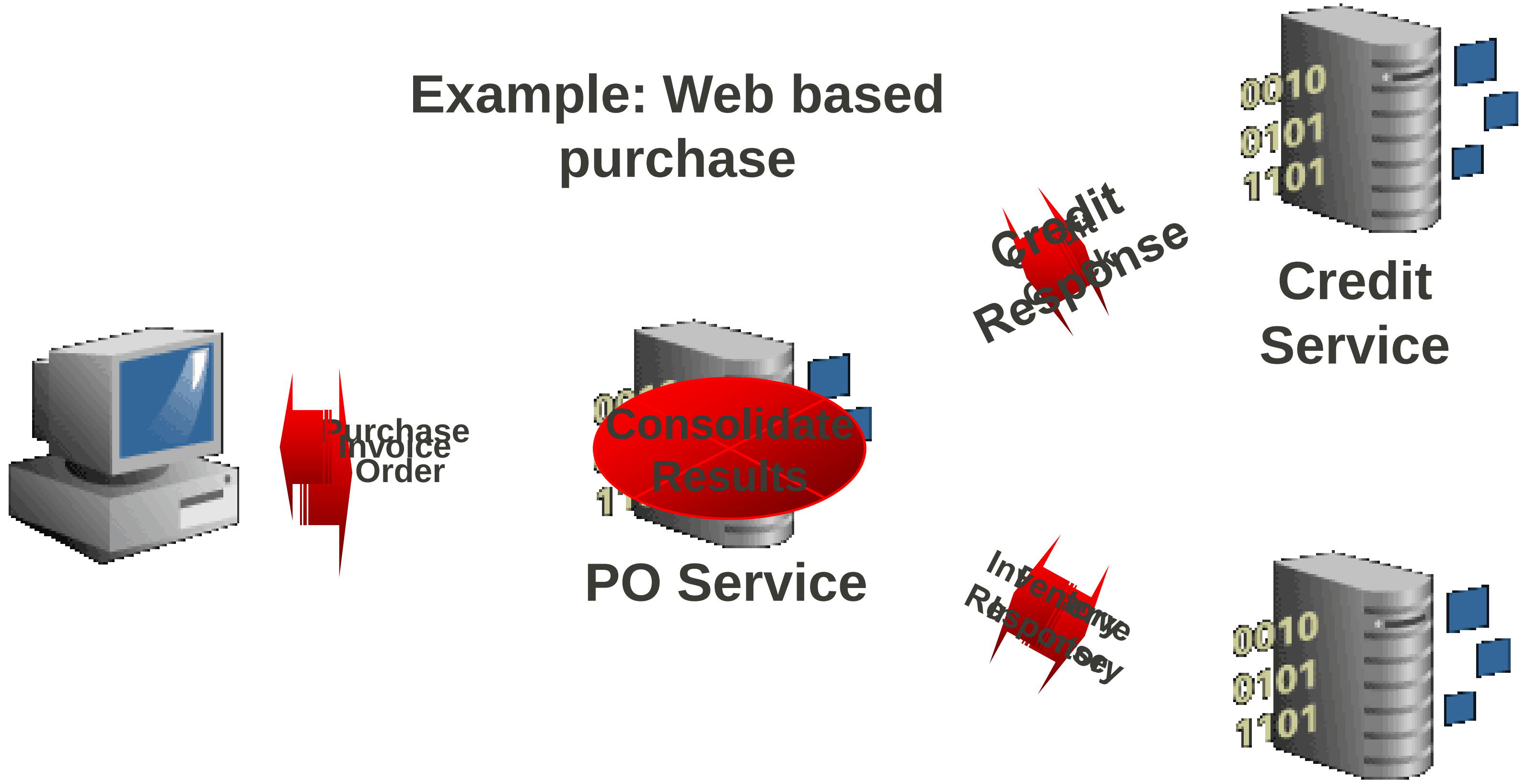
What are Web Services?

- Microsoft: XML Web Services
  - “.. **provide a way to describe their interfaces in enough detail to allow a user to build a client application to talk to them**”
- IBM

“**Web services are self contained, self describing, modular applications that can be published, located, and invoked across the web**. Web services **perform functions, which can be anything from simple requests to complicated business processes**".

What is Web Services

Example: Web based purchase

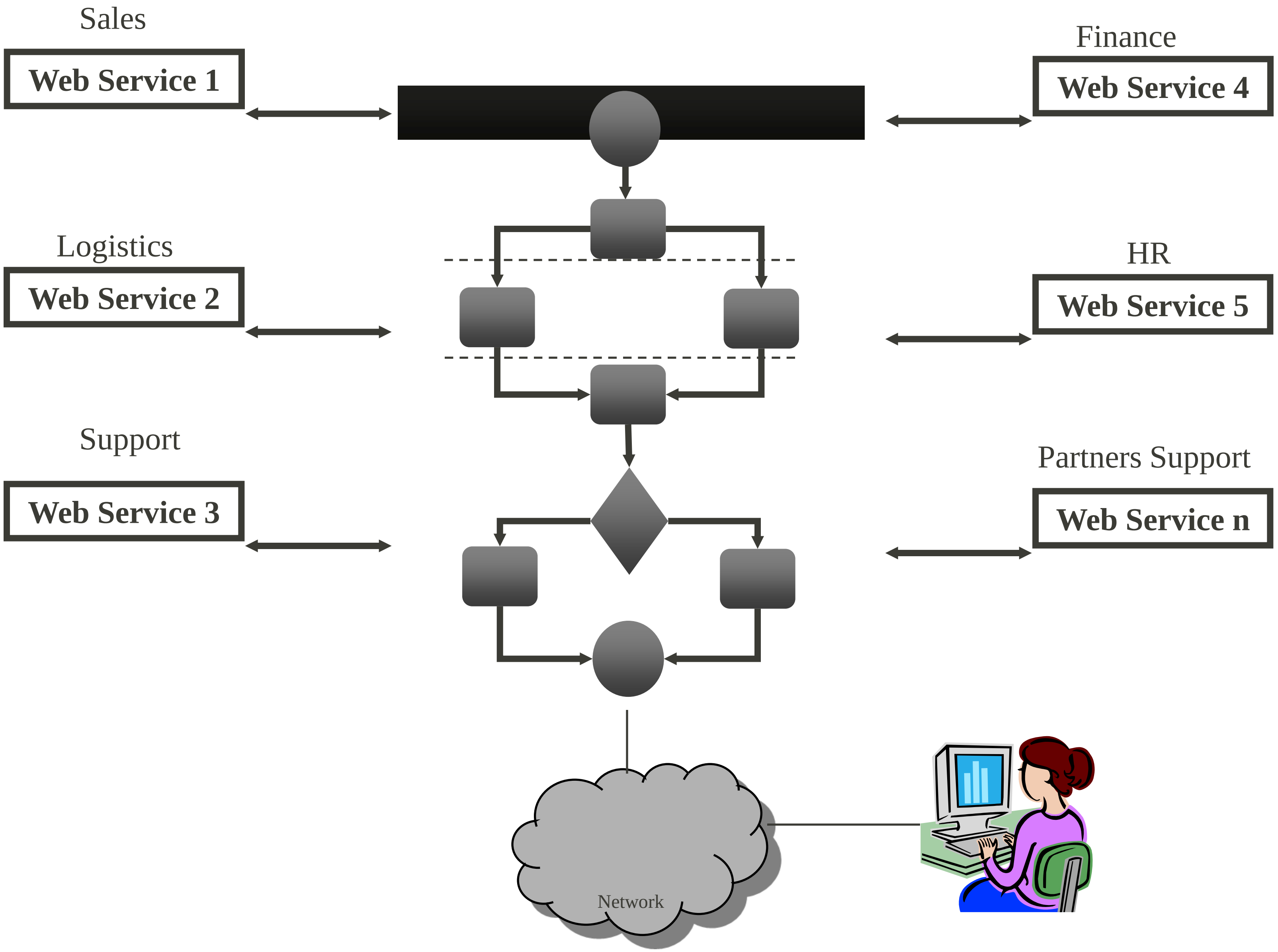


Why Web Services?

From Business:

- **Integration**
  - Within an organization
  - Between companies
  - **Allows time/cost efficiencies**
    - Purchase orders
    - Answering inquiries
    - Processing shipment requests
  - Do this without locking in to a single partner

Web Services Meet Business Processes



REST

REST stands for **Re**presentational **S**tate **T**ransfer

- REST is a term coined by Roy Fielding to describe an **architecture style** of networked systems.
- It is an **architectural pattern** for developing web services as **opposed to a specification**.
- REST web services communicate over the **HTTP** specification:
  - **Methods** (GET, POST, etc.)
  - **HTTP URI** syntax (paths, parameters, etc.)
  - Media types (**xml, json, html, plain text, etc**)
  - **HTTP Response codes.**

"**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."

Roy Fielding.

REST + Web Services = **RESTful Web services**

REST

- **Representational**
  - Clients possess the information necessary to identify, modify, delete a web resource.
- **State**
  - All resource state information is stored on the client.
- **Transfer**
  - Client state is passed from the **client** to the service through **HTTP.**

Rest – An architectural Style

Elements

- **Components**: Proxy, gateway, etc
- **Connectors**: client, server, etc
- **Data**: resource, representation, etc

REST

- **Ignores component implementation details**.
- **Focus** on **roles of components, their interactions and their interpretation of data elements**.

REST - An Architectural Style of Networked System

- Underlying Architectural model of the world wide web.
- Guiding framework for Web protocol standards.

REST based web services

- Online shopping
- Search services
- Dictionary services

REST

The six characteristics of REST:

1. **Uniform interface**
2. **Decoupled client-server interaction**
3. **Stateless**
4. **Cacheable**
5. **Layered**
6. **Extensible through code on demand (optional)**

Services that do not conform to the above required constraints **are not strictly RESTful web services.**

1. **Uniform interface**
    - This is the **API** of the web service, **describing operations and data structures.**
    - It **simplifies and decouples** the **architecture of both client and server**, enabling each to evolve independently.
2. **Client–server decoupling**

    Clients are separated from servers by a **uniform interface.**

    - For portability, clients **must not concern** themselves with data storage.
    - For simplicity and scalability, servers **must not concern** themselves with the UI or user state.
    - **Servers and clients may be replaced and developed independently, as long as the interface is not altered.**

3. Stateless

- No client context should be stored on the server between requests.
- **Each request contains all of the information necessary to service the request and session state is held in the client.**
- It makes servers:
  - **More scalable.**
  - **More visible for monitoring.**
  - **More reliable in the event of partial network failures**

4. **Cacheable**
   - **Clients may cache responses**, so responses must define whether they are cachable to prevent clients reusing stale or inappropriate data in response to further requests.
   - Well-managed caching **can eliminate repetitive client–server interactions**, further **improving scalability and performance**.

5. Layered system

A client's connection to a server may pass **directly to the service or through several intermediaries**, allowing:

- Scalability by **enabling load balancing** and by providing shared caches
- The enforcement of security policies.

6. Code on demand (optional)

Servers may temporarily extend or customize the functionality of a client by transferring logic to be executed. (e.g. client-side JavaScript)

REST web service

If a service violates any constraint other than "**code on demand**", it cannot strictly be referred to as REST web service.

Complying with these constraints, and thus conforming to the REST architectural style, will **improve** a service's

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability

HTTP-REST Request Basics

- The **HTTP request** is sent from the client.
  - Identifies the location of a **resource**.
  - Specifies the **verb**, or HTTP **method** **(**GET, POST, PUT, DELETE, etc.) to use when accessing the resource.
  - Supplies optional **request headers** (name-value pairs) that provide additional information the server may need when processing the request.
  - Supplies an optional **request body** that identifies additional data to be uploaded to the server (e.g. form parameters, attachments, etc.)

HTTP-REST Request Basics

Sample Client Requests:

- A typical client **GET** request:

```
GET /view?id=1 HTTP/1.1              } Requested Resource (path and query string)
User-Agent: Chrome                   }
Accept: application/json             } Request Headers
[CRLF]                                (no request body)
```

- A typical client **POST** request:

```
POST /save HTTP/1.1                  } Requested Resource (typically no query string)
User-Agent: IE                                               }
Content-Type: application/x-www-form-urlencoded              } Request Headers
[CRLF]
name=x&id=2              } Request Body (e.g. form parameters)
```

CRLF: Carriage Return and Line Feed

HTTP-REST Response Basics

- The **HTTP response** is **sent *from the server***.
  - Gives the **status** of the processed request. (e.g. 404 Not Found, 200 OK)
  - Supplies **response headers** (name-value pairs) that provide additional information about the response.
  - Supplies an optional **response body** that identifies additional data to be downloaded to the client (html, xml, binary data, etc.)

HTTP-REST Response Basics

Sample Server Responses:

```
HTTP/1.1 200 OK                 } Response Status
Content-Type: text/html         } Response Headers
Content-Length: 1337            }
[CRLF]
<html>                                        }
  <!-- Some HTML Content. -->                 } Response Body (content)
</html>                                       }
```

```
HTTP/1.1 500 Internal Server Error } Response Status
```

```
HTTP/1.1 201 Created       } Response Status
Location: /view/7          } Response Header
[CRLF]
Some message goes here.    } Response Body
```

HTTP-REST Vocabulary

HTTP Methods supported by REST:

- GET – Requests a resource at the request URL
- POST – Submits information to the service for processing
- PUT – Add a new resource at the request URL
- DELETE – Removes the resource at the request URL
- OPTIONS – Indicates which methods are supported
- HEAD – Returns meta information about the request URL

HTTP-REST Vocabulary

A typical HTTP REST URL:

$$\underbrace{\texttt{http}}_{\text{protocol}}\texttt{://}\underbrace{\texttt{my.store.com}}_{\text{host name}}\underbrace{\texttt{/fruits/list}}_{\text{path to a resource}}\texttt{?}\underbrace{\texttt{category=fruit\&limit=20}}_{\text{query string}}$$

- The **protocol** identifies the transport scheme that will be used to process and respond to the request.
- The **host name** identifies the server address of the resource.
- The **path** and **query string** can be used to identify and customize the accessed resource.

HTTP and REST

A REST service framework provides a **controller** for routing HTTP requests to a request handler according to:

- The HTTP method used (e.g. GET, POST)
- Supplied path information (e.g /service/listItems)
- Query, form, and path parameters
- Headers, cookies, etc.

Producing REST Services

REST services in Java web applications can be implemented in several ways:

- As a plain Java Servlet
  - Adequate for very simple REST services.
- Using a REST service framework.
  - Typically integrates with other technologies, such as Spring.

Java provides the JAX-RS specification for use by providers of REST service frameworks.