

ADWYS CON '11

# Arquitectura RIA Empresarial Restbox y beDesk

  
at sistemas

ADWYS  
CON 11



at sistemas

ADWYS  
CON 11



**Antonio David Fernández.**

Responsable Centro de Desarrollo de  
atSistemas del Puerto de Santa María.

[adfernandez@atsistemas.com](mailto:adfernandez@atsistemas.com)

@antoniodfr

# Agenda

## 1. Conceptos

RIA / JSON / REST

Demo REST

## 2. Arquitectura RIA de alto nivel

## 3. ¿Qué es un WebTop?

Demo beDesk

## 4. Implementación de Arquitectura RIA

## 5. Restbox JEE / beDesk / xQuiD / jQuery

Demo Restbox

# Conceptos





- RIA: Rich Internet Application.
- Nuestra visión de RIA está basada en los estándares abiertos de la Web: HTML, CSS y Javascript.
- Sin plugins adicionales en el navegador web.
- Implica principalmente que **no existe recarga de página**.
- Permite lógica de presentación en el cliente, como uso del patrón MVC.
- Se convierte en un modelo Cliente - Servidor a través de la Web.

El salto al desarrollo RIA, requiere de una buena estructuración y gestión de la parte menos controlable: ***Código en la parte cliente.***



- Escala mejor, al trasladar lógica del servidor a cliente.
- Mejores tiempos de respuesta, ya que muchas operaciones se hacen en el cliente.
- Mejor experiencia de usuario: Una llamada a servidor puede ser no bloqueante, permitiendo al usuario hacer otras operaciones.
- No existe sesión en servidor: Esto nos permite escalar más fácilmente.

**Necesidades:** Necesitamos frameworks que gestionen y simplifiquen:

- Crossbrowsing.
- Gestión visual.
- Reutilización de código.

## 1.2

# ¿Qué es JSON?

- JSON: **J**avascript **O**bject **N**otation.
- Notación empleada para representar textualmente objetos Javascript .
- Javascript permite instanciar directamente objetos a partir de una cadena JSON.

## Ejemplos

```
var coche = {
  "nombre" : "Renault Megane",
  "matricula" : "8494-BGJ",
  "color": "gris"
};
```

```
var coches = [{ "nombre": "Fiat Punto", "matricula": "2345-GHV", "color": "rojo" },
  { "nombre": "Seat León", "matricula": "8474-FGH", "color": "azul" } ];
```



- REST: **RE**presentational **St**ate **T**ransfer.
- Es un estándar de publicación de servicios en la web a nivel mundial.
- Aprovecha la infraestructura de la web.
- Basado en el protocolo HTTP.
- Es una alternativa a SOAP.
- Fácil de usar.
- Formalizado por Roy Fielding.

**/personas/1**

```
{ id: 1,  
  nombre: "Fernando",  
  provincia: /provincias/1 }
```

**/provincias/1**

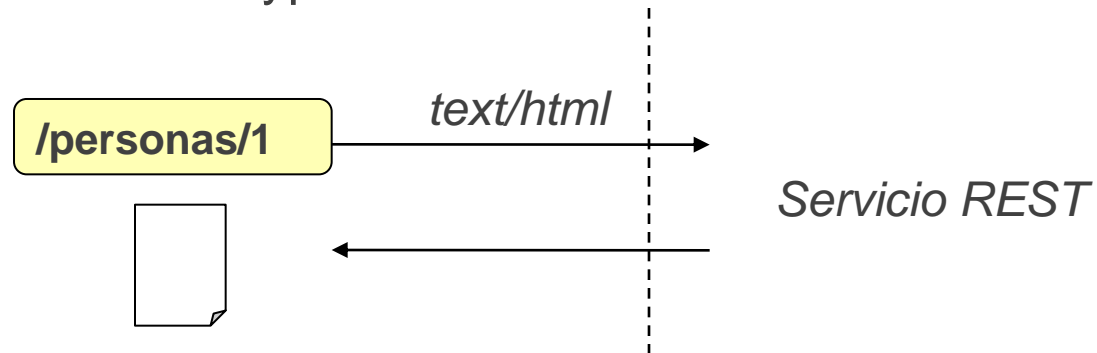
```
{ id: 5,  
  nombre: "Cádiz" }
```



- REST publica **RECURSOS**, no métodos. Pensamos en nombres.
- REST no es un mecanismo de RPC como SOAP.
- Un recurso representa un concepto de negocio sobre el que queremos trabajar. Ej. persona, pedido, coche, etc.
- La definición de un recurso consta de:
  - Una URI que la identifica de forma global y sin ambigüedades.
    - Es el “nombre” del recurso.
  - Un modelo de información o de datos.
  - El conjunto de “representaciones” que soporta, en forma de mime types.
- Sobre un recurso sólo podemos hacer operaciones CRUD.



- Cada recurso puede soportar una o más “representaciones” de su modelo de información.
- Cada representación es un formato de datos concreto estandarizado en un mime type. Ej. application/json, text/html, image/jpeg, etc..
- Podemos crear nuestros mime types sin estandarizarlos si no nos interesa interoperabilidad a nivel mundial: ej: application/x-persona+json.
- El protocolo HTTP se encarga de negociar el mime type a usar con las cabeceras Accept y Content-Type.





- A diferencia de un mecanismo RPC las operaciones son limitadas a CRUD, no podemos definir nuevos métodos.
- Cada operación se mapea de forma estándar a un verbo HTTP:
  - **READ** usamos el método GET sobre la URI.
  - **UPDATE** usamos el método PUT sobre la URI y una representación del nuevo estado del recurso en el body.
  - **DELETE** usamos el método DELETE sobre la URI.
  - **CREATE** usamos el método POST sobre la URI.  
Opcionalmente pasamos el estado inicial del recurso en el body.
- La única operación obligatoria es READ, las demás son opcionales. Se debe documentar qué operaciones se soportan.

1.7

## DEMO REST

 at sistemas

 ADWYS  
CON

*DEMO con Curl como cliente HTTP*  
(Curl: <http://curl.haxx.se/> )



- REST es sencillo ya que trabaja sólo sobre el protocolo HTTP o HTTPS:
  - No es necesario definir “bindings” con otros protocolos.
  - No es necesario definir métodos arbitrarios.
- Aprovecha toda la infraestructura de la web usando HTTP:
  - Cache.
  - Concurrencia optimista y versionado.
  - Seguridad.
  - Compresión.
  - Multilenguaje.
- Sencillo de entender, mayor productividad y mantenibilidad.
- Sencillo de procesar por las máquinas, ideal para clientes AJAX.
- No es necesario mantener una sesión en servidor.



- El mejor ejemplo REST sencillo que podemos encontrar y que funciona a nivel Mundial, es la propia **World Wide Web**.
- Una página web no es más que un recurso REST que:
  - Soporta sólo tipos mime text/html y similar.
  - Soporta sólo la operación READ.
  - Permite vínculos a otras páginas web.

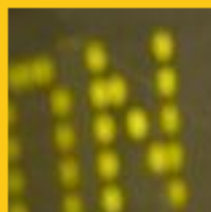
## 1.10

## REST vs SOAP



	SOAP	REST
<b>Protocolo</b>	<b>Cualquiera, pero necesitas especificar el mapeo en WSDL</b>	<b>HTTP/S</b>
<b><i>Definition Language</i></b>	<b>WSDL</b>	<b>No es necesario, se basa en tipos mime estándar</b>
<b>Publica</b>	<b>Métodos y tipos de datos para argumentos</b>	<b>Recursos/entidades</b>
<b>Aprovecha HTTP</b>	<b>NO</b>	<b>SI</b>
<b>Complejidad</b>	<b>ALTA. NECESITA IDE.</b>	<b>BAJA</b>
<b>Multiformato</b>	<b>Sólo XML o XML with attachments</b>	<b>Cualquier tipo mime</b>
<b>Ámbito</b>	<b>Local a la empresa o entre dos empresas partners</b>	<b>Global</b>

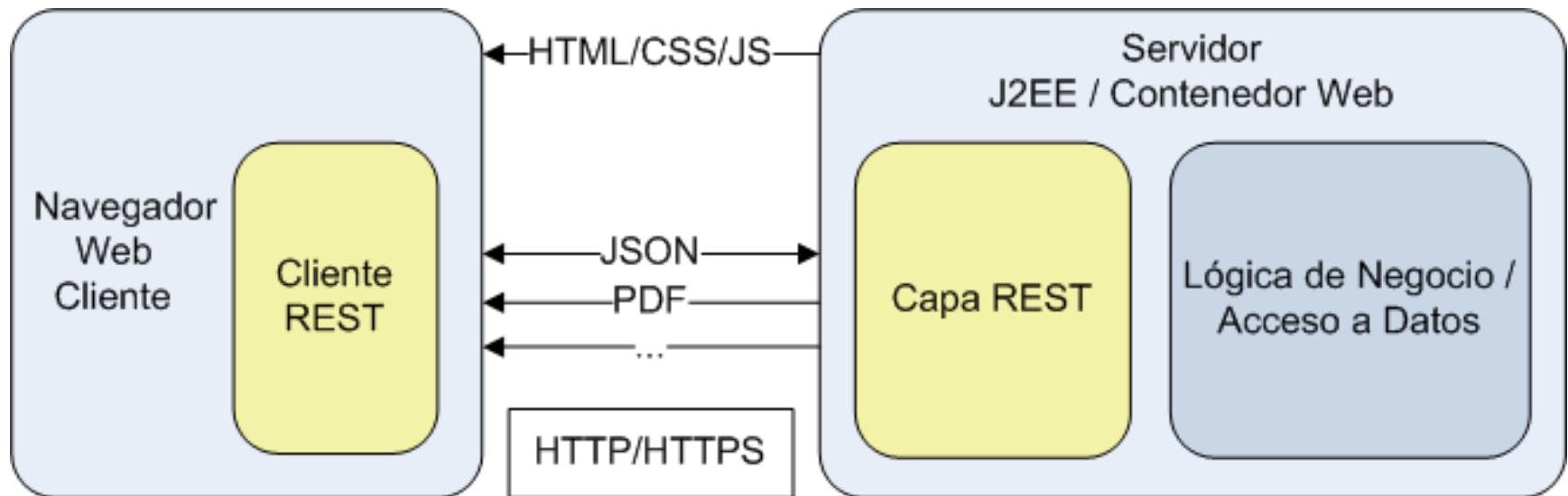
# Arquitectura RIA de alto Nivel



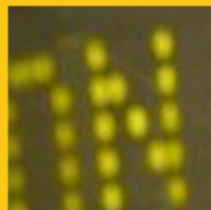


## Parte Cliente: Browser

## Parte Servidora: JEE



# Introduciendo un WebTop





- Un WebTop permite, **sin recarga de página**:
  - Presentar al usuario el **conjunto de aplicaciones** a las que tiene acceso (Mediante Menús, iconos, etc.)
  - Por cada aplicación, pueden lanzarse **varias instancias simultáneas**, pudiendo seleccionar cuál está activa entre ellas.
- **Ofrece a las aplicaciones una infraestructura común**:
  - Servicio de log.
  - Cliente REST para acceso a la parte servidora.
  - Notificaciones homogéneas para el usuario (Aplicación Cargando datos, diálogos, etc.).
  - Comunicación entre aplicaciones.

## 3.1

# Ventajas de un WebTop

- Con este concepto, pueden agruparse todas las aplicaciones empresariales en una misma página, siendo una alternativa a los portales.
- Refleja un comportamiento cercano al usuario.
- Capacidad de descargar el código de cada aplicación de forma dinámica y perezosa (bajo demanda).
- Todos los servicios comunes los proporciona la plataforma WebTop y no necesitan ser reimplementados en cada aplicación.

## 3.2

# DEMO WebTop: beDesk

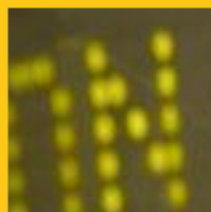
  
at sistemas

ADWYS  
CON 11

*DEMO del WebTop de atSistemas:*

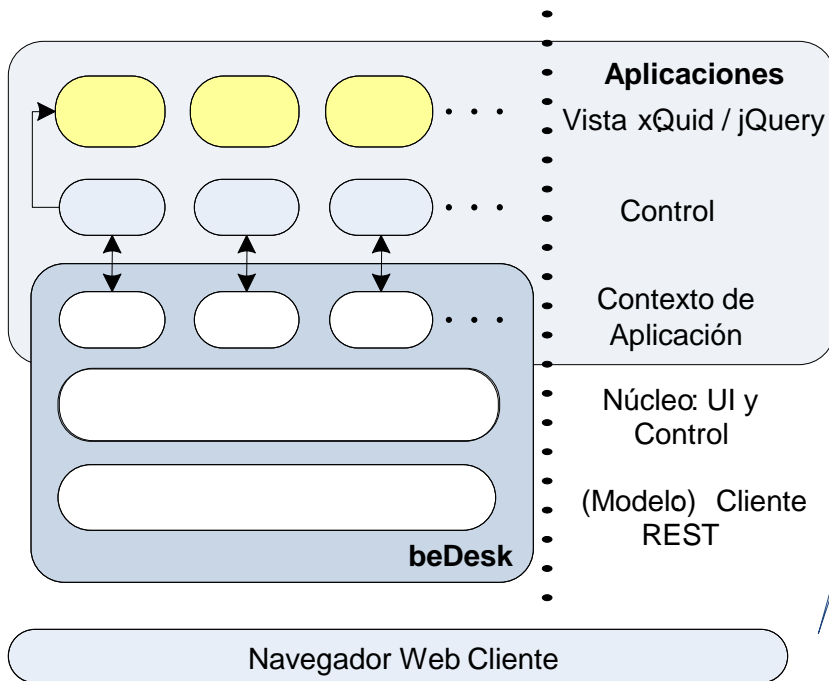


# Arquitectura RIA: Propuesta de Implementación

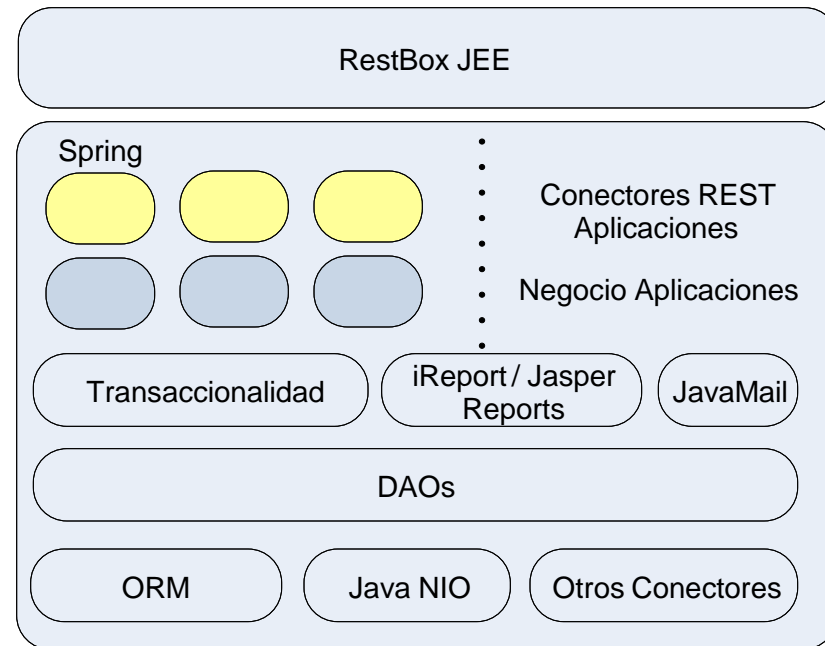




## Parte Cliente: Browser

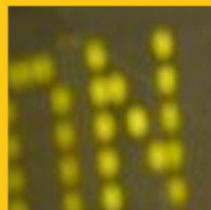


## Parte Servidora: J2EE



HTTPS / REST

# Detalles de la Arquitectura Propuesta



## 5.0

## Restbox JEE (1)

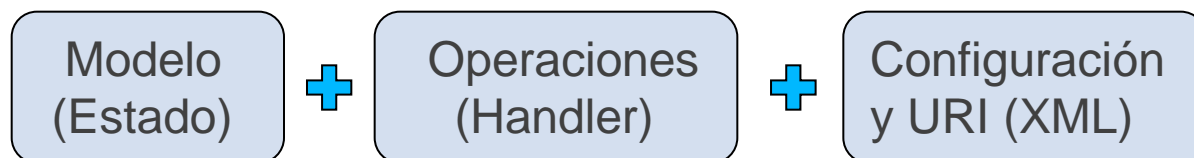
- Framework ligero para publicar servicios REST usando J2EE y JDK5.
- Fue desarrollado antes de que se cerrara la especificación (JSR-311).
- Aprovecha las anotaciones de JDK5 y JAVA NIO.
- Se integra con Spring Framework.
- Nos permite trabajar directamente con POJOs abstrayendo completamente el mapeo de las operaciones CRUD a HTTP.
- Conversión automática de beans a tipos mime y viceversa.
- Proporciona un sistema extensible para soportar múltiple tipos mime.
- Configurable mediante Spring y un sencillo XML.



## 5.1

# Restbox JEE (2)

- Su objetivo es aislar los detalles de REST.
- Para publicar un servicio primero debemos definir el modelo de información del recurso REST → Un simple JavaBean.
- Podemos anotar este bean mediante anotaciones JDK5 para customizar la forma en que el framework hará la conversión al tipo mime pedido por el cliente.
- Las operaciones CRUD se implementan como métodos en un POJO al que llamaremos “handler”.
- En el XML definimos la URI del recurso y que “handler” y bean usar para cada recurso.



## 5.2

# Restbox JEE (3)

- Nos permite implementar 6 operaciones en los handler:
  - Operaciones para entidades: Ej URI: /clientes/1287
    - **retrieve** (GET) – **update** (POST) – **delete** (DELETE)
  - Operaciones para colecciones: Ej: URI: /clientes
    - **listAll** (GET sin parámetros) – **find** (GET con parámetros) – **Insert** (PUT).
- En la actualidad se han afianzado otros frameworks:
  - Especificación JEE: JAX-RS (Apache CXF, Jersey, RESTEasy)
  - Alternativa de Spring: Spring RESTful.

5.3

## DEMO RestBox



  
at sistemas

  
ADWYS  
CON

*DEMO de RestBox JEE*



- Es el WebTop de atSistemas.
- Permite desarrollar las aplicaciones con dos frameworks cliente:
  - jQuery (<http://jquery.com/>)
    - Comunidad muy activa.
    - Gran número de plugins y componentes disponibles.
  - xQuid (<http://www.xquid.org/>)
    - Orientado a aplicaciones empresariales, gracias a su editor visual WYSIWYG para eclipse.
    - Donado por atSistemas a la comunidad Open Source.

- Cada usuario de la aplicación tiene un menú de aplicaciones, configurado en el propio sistema en base a autorizaciones:
  - a nivel de usuario o a nivel de rol.
- Además de ese nivel de seguridad, se securiza también a nivel de URL, ya que por cada aplicación, se define que URIs REST se van a consumir. Se implementa con Spring Security.



atdesk Web Enterprise Desktop Consult

Consulta y gestión de las incidencias enviadas al departame

Búsqueda de Incidencias

Palabras Clave:

Estado:

Buscar

Resultados

Asunto	Fecha	Est.
IE8 Mis clientes y servicios- No ref...	10/11/2009 a...	Cerrada

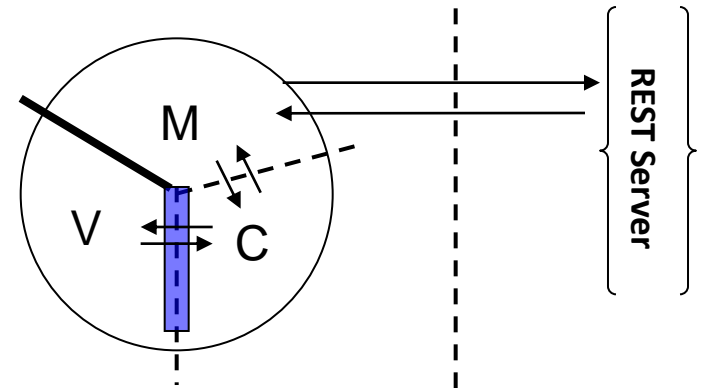
GET /estadosIncidencia

GET /incidencias?pClave=IE8&  
estadold=2



Establece un modelo de desarrollo basado en **MVC en cliente**:

- Cada aplicación recibe un contexto para interactuar con el WebTop y requiere que se defina:
  - **La vista:** Formada por los recursos visuales (paneles xQuid / HTML+CSS+jQuery)
  - **El controlador:** fichero javascript con el control de la aplicación, encargado de instanciar la vista, escuchar a sus eventos, y solicitar datos al modelo (cliente REST).
  - **El modelo:** cliente REST para consumo de recursos REST. El cliente REST es ofrecido en el contexto de aplicación.



## 5.7

# beDesk (4): Ciclo de Vida

- Todas las aplicaciones siguen el mismo ciclo de vida, que se refleja en los métodos que debe implementar su controlador:
  - **Al abrir la aplicación:** `init()` para renderizar la vista, `reload()` para cargar los datos iniciales.
  - **Al cerrar la aplicación:** `destroy()`.
  - **Al cambiar de pestaña**, y dejar la aplicación en background: `pause()`.
  - **Al seleccionar una pestaña** y reactivar una aplicación: `resume()`.



atsistemas



**Gracias por vuestra atención**

**Antonio David Fernández.  
adfernandez@atsistemas.com  
@antoniodfr**