WebSphere software

IBM

e business software

# Web Services for Remote Portlets (WSRP)

WSRP Kickoff Meeting
March 18-20 2002

Dr. Carsten Leue
Thomas Schäck
Peter Fischer

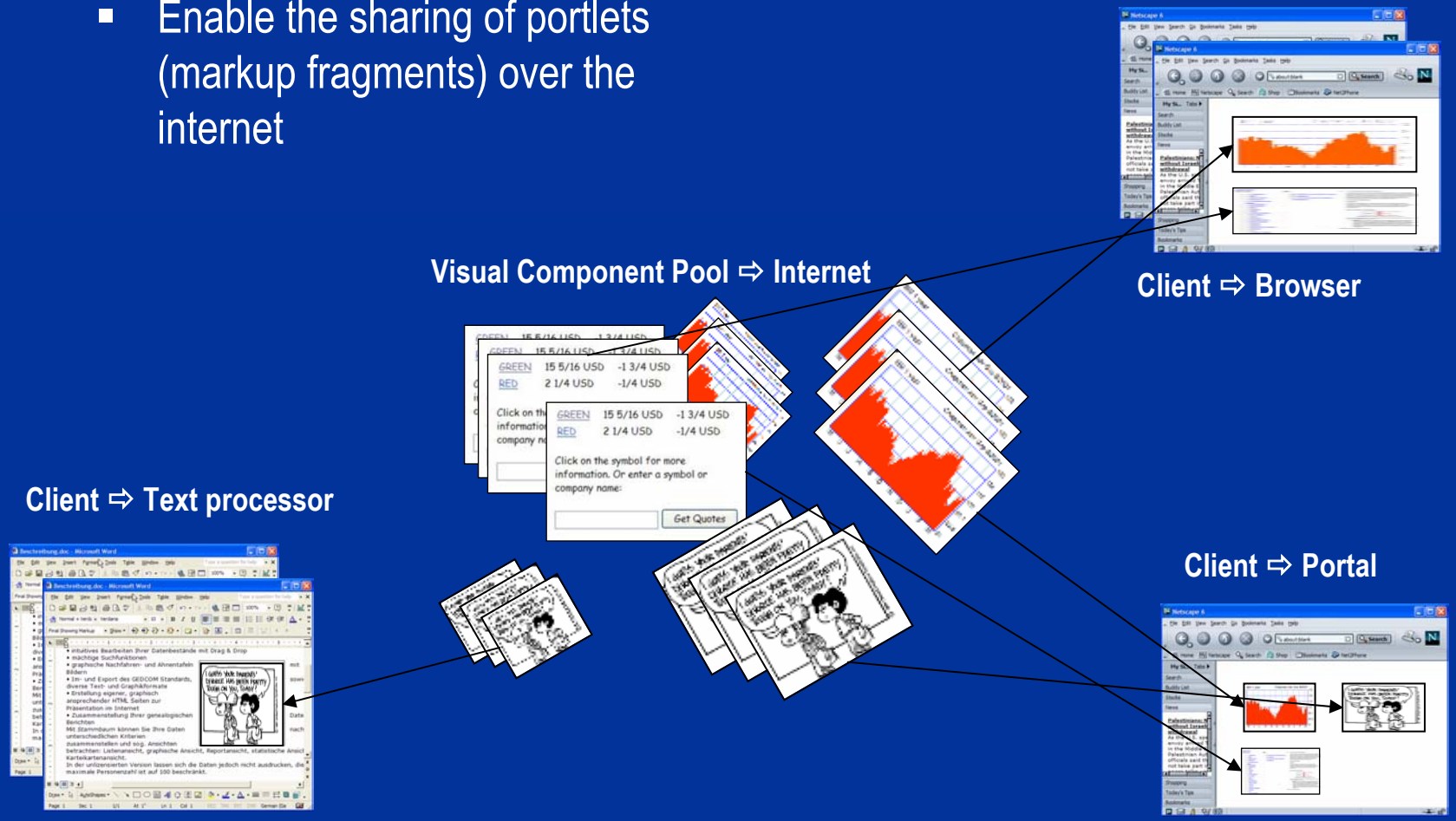
IBM Software Group

# Summary

- Web Services for Remote Portlets
  - Idea and Goals
  - Architecture and Design
  - Markup, Actions and Persistence
- Implementation
  - J2EE standalone version
  - Integration into portal servers
- Standards
  - Relationship to WSIA
  - Interoperability

# WSRP Motivation

- Enable the sharing of portlets (markup fragments) over the internet

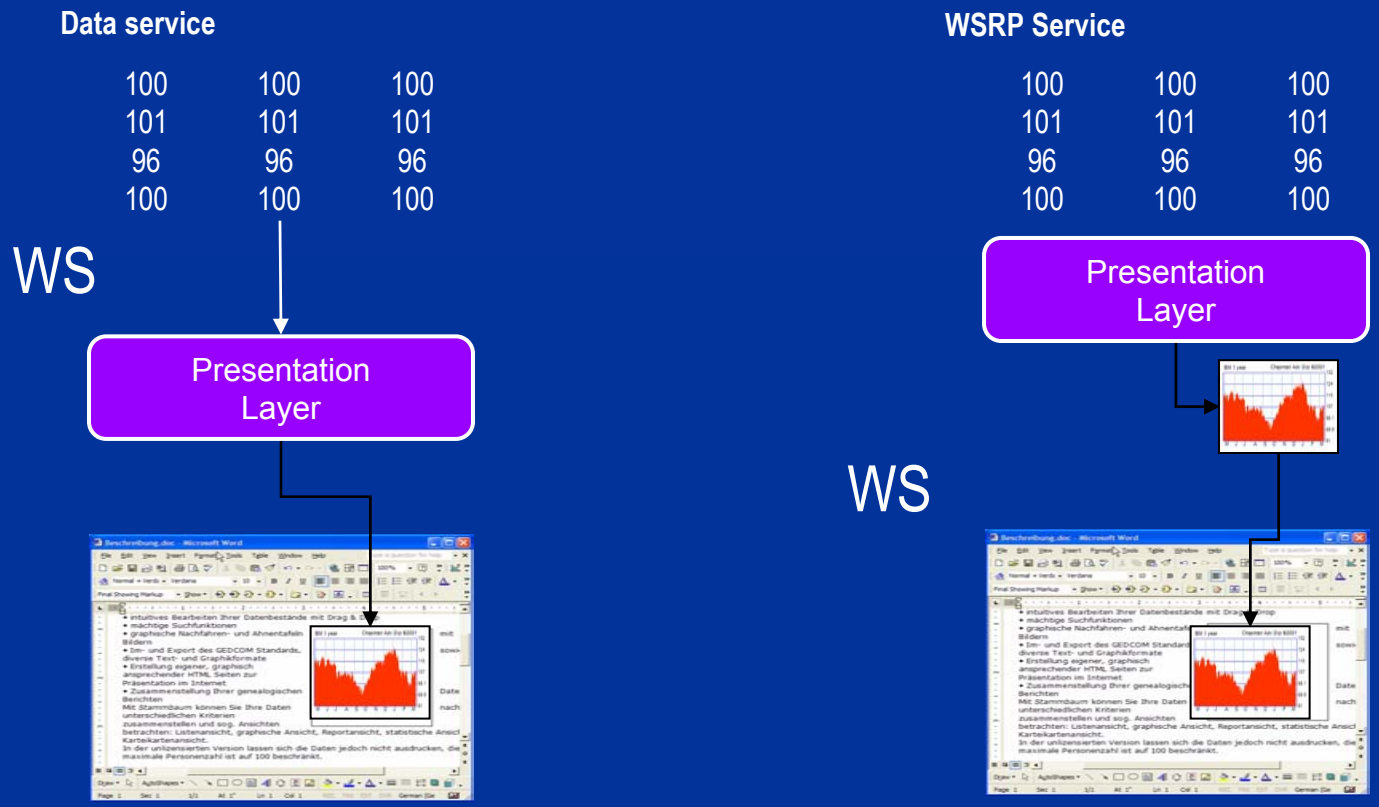


# Goals of Web Services for Remote Portlets (WSRP)

- Allow **visual**, **interactive**, user-facing web services to be **easily plugged into** all standards-compliant portals
- Let anybody **create and publish** their content and applications as user-facing web services
- Portal administrators can browse public or private UDDI directories for user-facing web services to plug into their portals as new portlets, **without any programming effort**

- Let portals **interact** and publish portlets so that they can be consumed by other portals
- Make the internet a **pool** of visual web services, waiting to be integrated

# Remote Portlets vs. data oriented WS

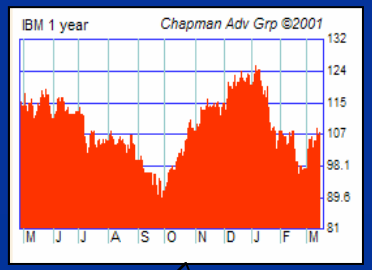



- WSRP ⇔ visual & user facing & interactive

# WSRP Sample Usage

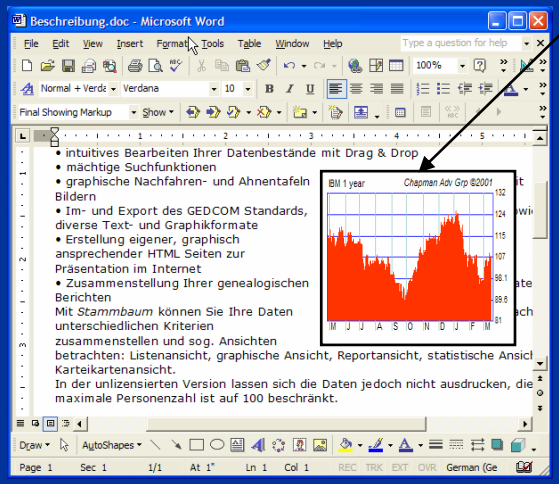
Document service

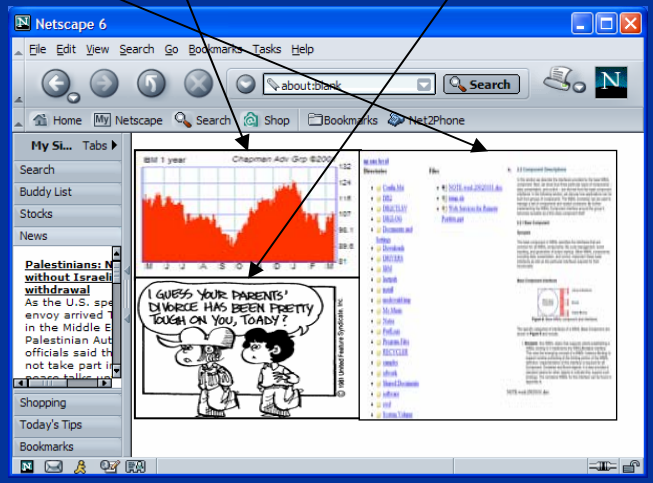
Stocks service




Cartoon service


Internet or Intranet via SOAP

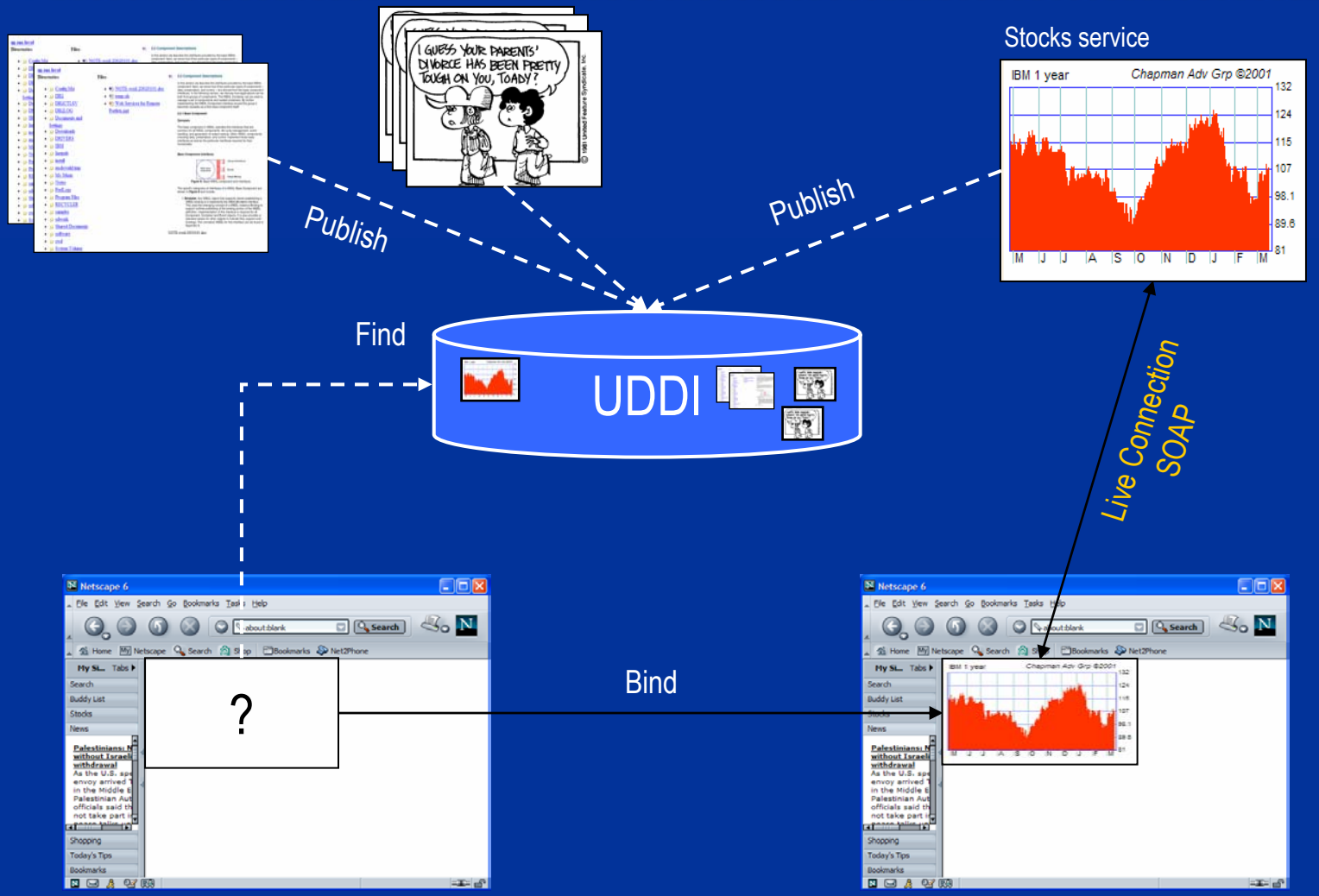



WebSphere software

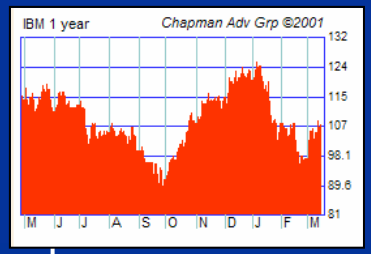
IBM

# WSRP Advertising



# WSRP Entities

- Service
  - exposes the WSRP interface
- Aggregation (client)
  - consumes multiple WSRP services
  - aggregates the services onto pages
- Device
  - displays the aggregated markup to the end user
  - handles user input

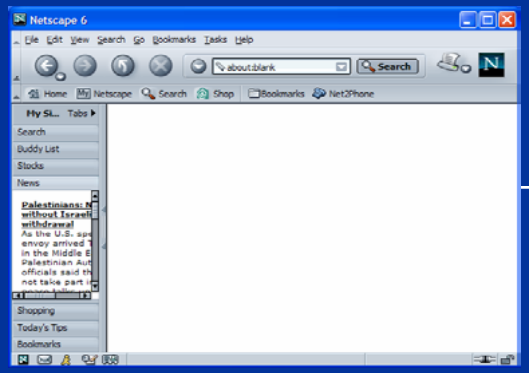



**Service Component**

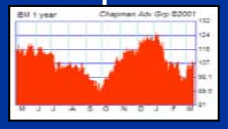
WSRP

**Aggregation Component (client)**

**Device Component**




e.g. HTTP



portlet


portlet

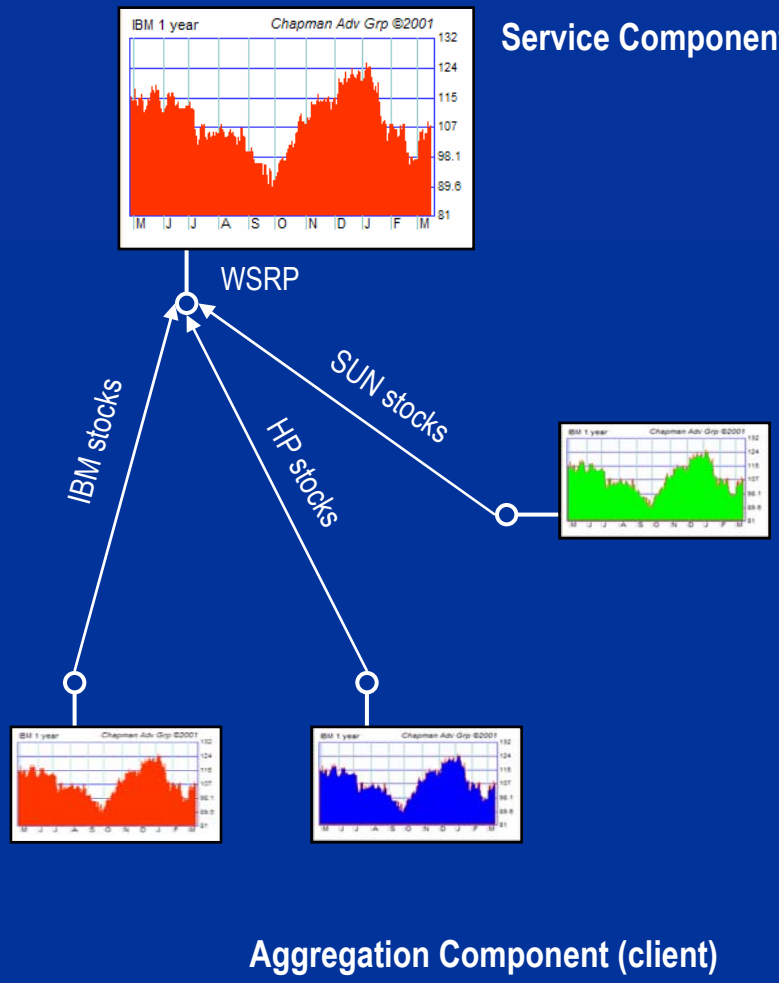
portlet



IBM

# WSRP Instances

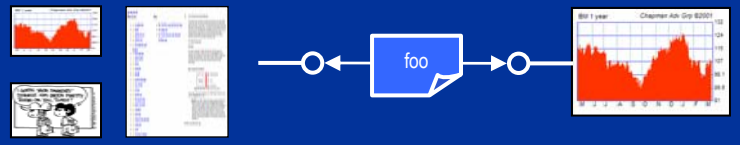
- Scenario
  - The same service may be accessed multiple times with different settings
  - The server must manage and identify each of these settings
- Solution
  - Service + settings form a „**remote instance**“
  - Clients always integrate instances of WSRP services
  - The management of the instance‘s settings can be negotiated between client and server



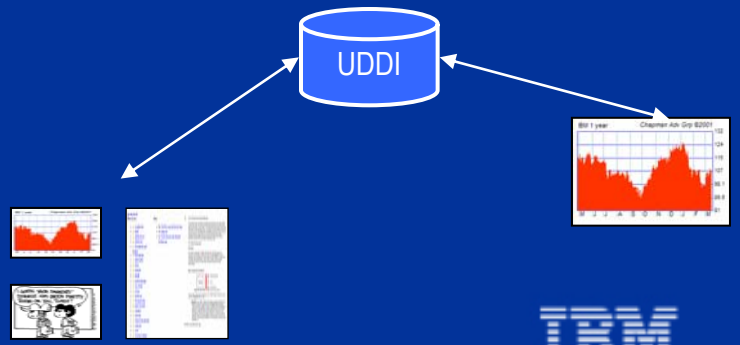
# What needs to be defined?

- Interfaces
  - Management of remote instances
  - Markup retrieval and action processing

- Protocol
  - Sequence of calls
  - Markup rules
  - Action and namespace encoding

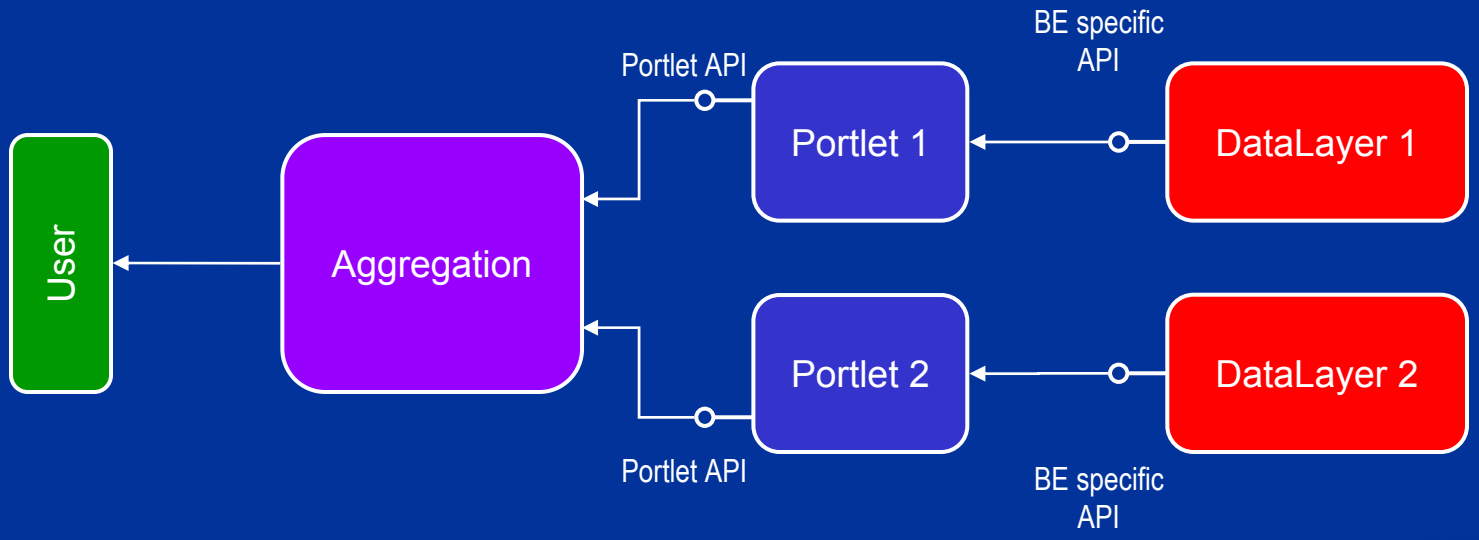



- UDDI configuration
  - How to publish?
  - What to publish?



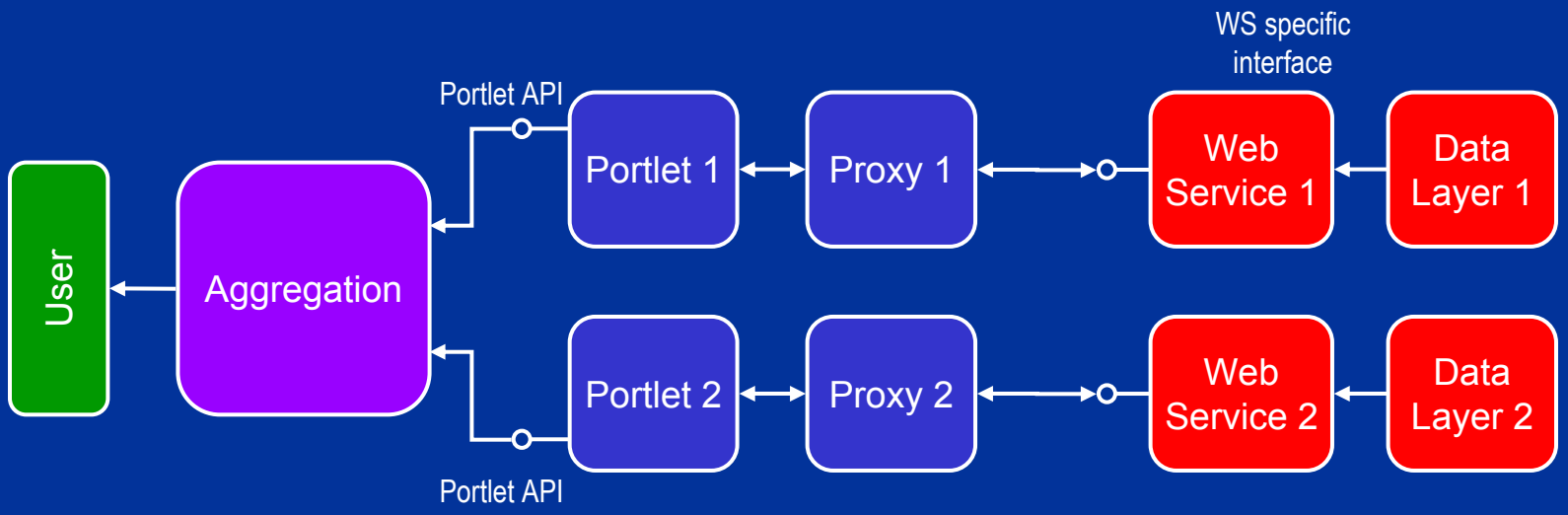
# Summary: Traditional Back-End Usage Scenario

- Local Portlets
  - Efficient ☺
  - Local deployment of code ☹
  - Specific UI for each deployed portlet ☹
  - Business layer and presentation layer both located on the portal server ☹
  - Portlets cannot be shared between portals!! ☹



# „Traditional“ Web Service Usage Scenario

- Portlets using Web Services
  - Different Web Services expose different interfaces ☹
  - Specialized UI and proxy code required for each WS ☹
  - Local deployment of code is still necessary ☹
  - Data layer separated from presentation layer ☺

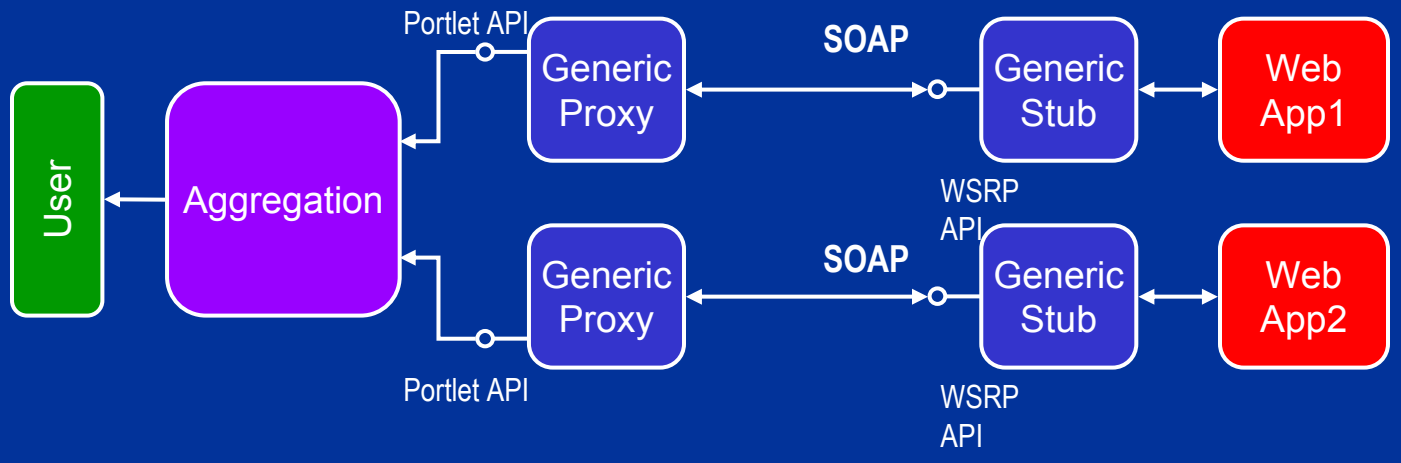


# Wish list

- Client‘s view
  - Plug-and-play
  - Configurable
  - Interactive
  - Markup and user aware
- Server‘s view
  - Modest implementation overhead
  - Scalable
  - Client aware
- Users‘ view
  - Does not want to bother

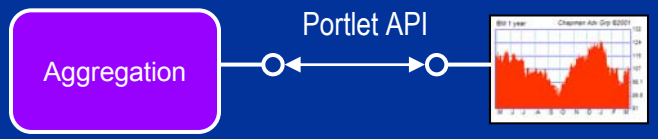
# WebServices for Remote Portlets (WSRP)

- All remote connections share a unified API ☺
- No coding required, proxy and stub are coded **once** or generated automatically ☺
- Stable and standardized transport mechanism (e.g. SOAP) ☺
- Visual and user-facing ☺



# Requirements for a remote API

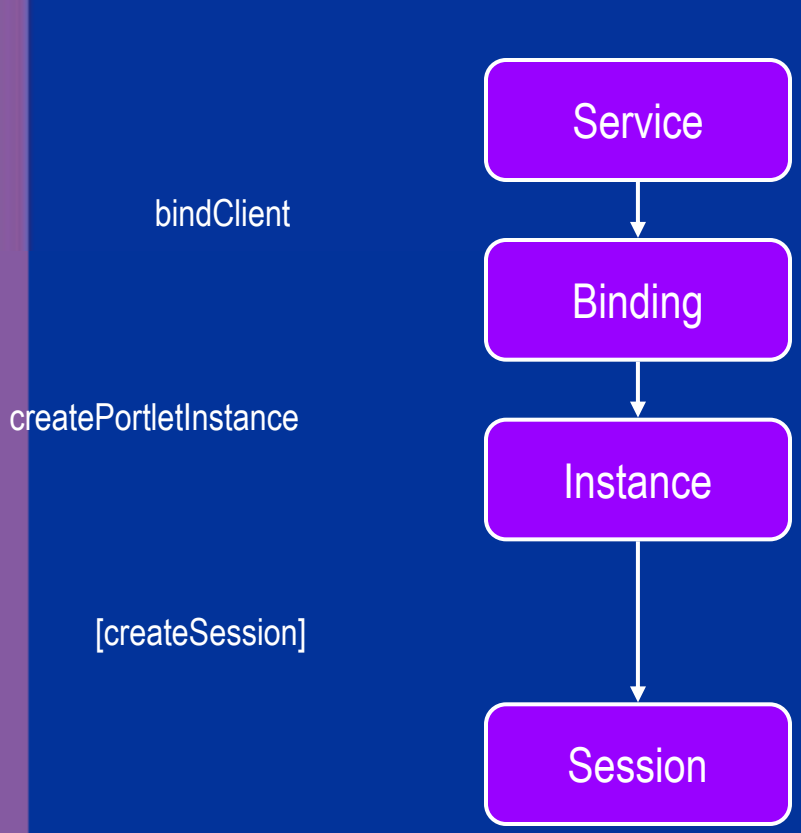
- Local case
  - Each Portlet forms a logical instance
  - Portlets generate markup based on user and device profile
  - Portlets can store state data in a database
  - Portlets can encode actions as URLs
- Remote case
  - The service must be instance aware
  - User and device data must be transmitted to the service
  - The service must either be persistent or it must delegate the persistence to the caller
  - Actions encoded by the service must be recognized and remoted by the caller
  - To authorize calls, remote instances are embedded in a binding context

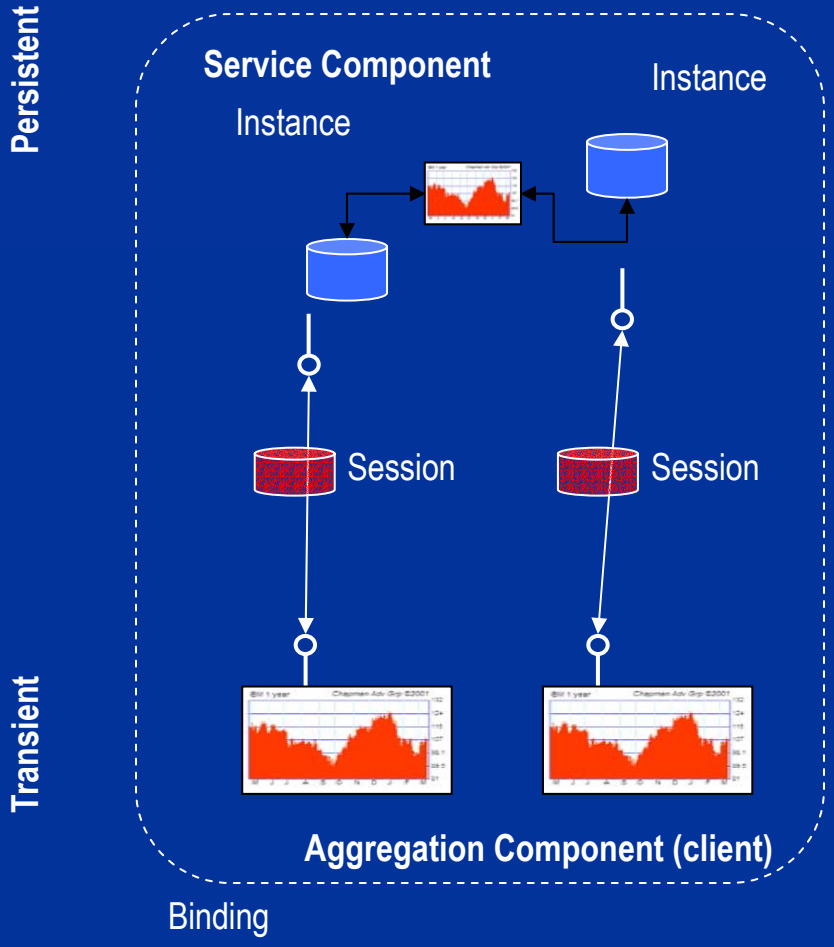



# WSRP Contract

- WSRP technical contracts define
  - Action handling and embedding in URLs
  - Namespacing of named entities
  - Restrictions on markup produced
  - Allowed order of method invocation
- WSRP interfaces define
  - Lifecycle handling
  - Markup Retrieval
  - Action handling

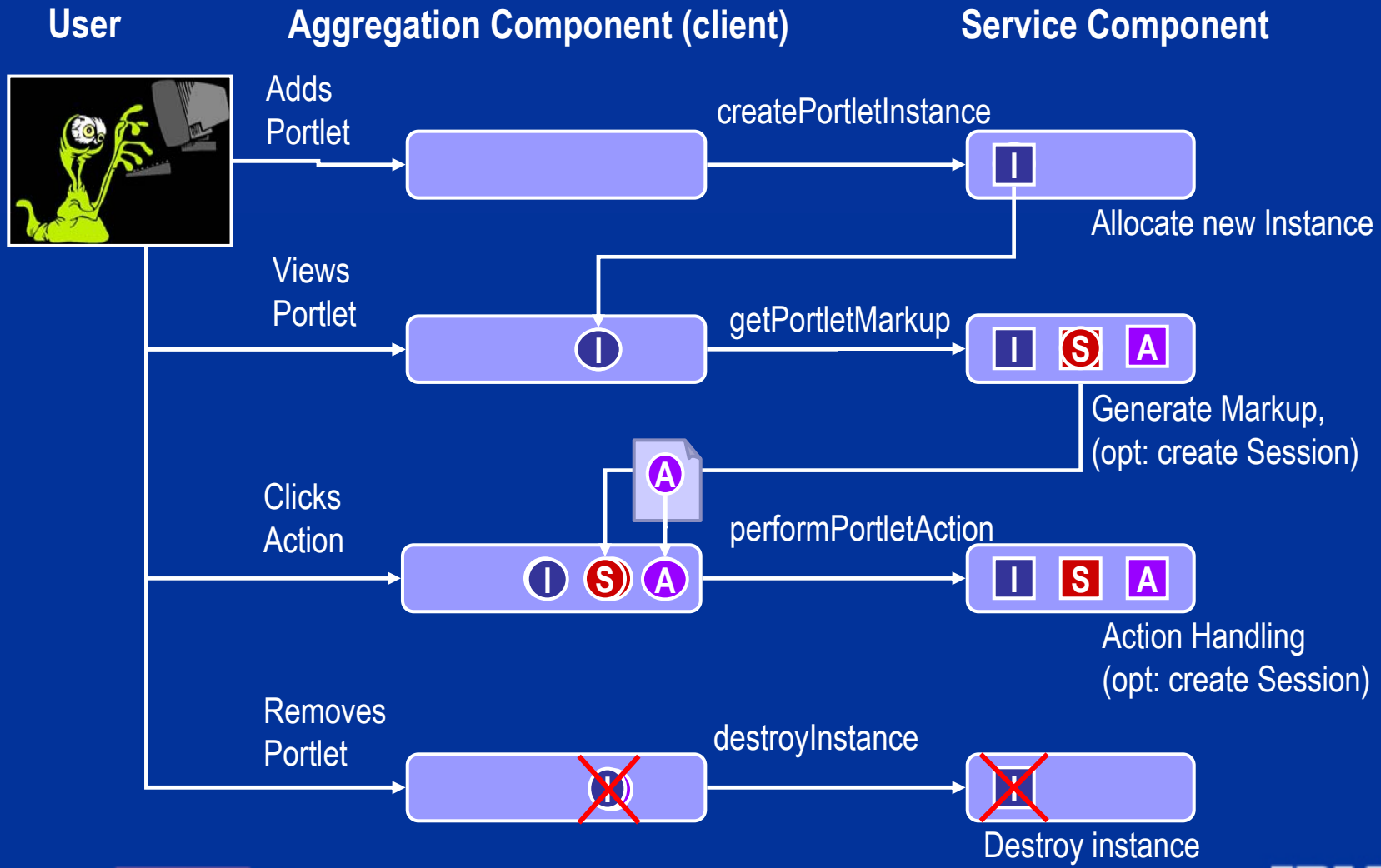
# Life Cycle Management




Instances are identified by **handles**

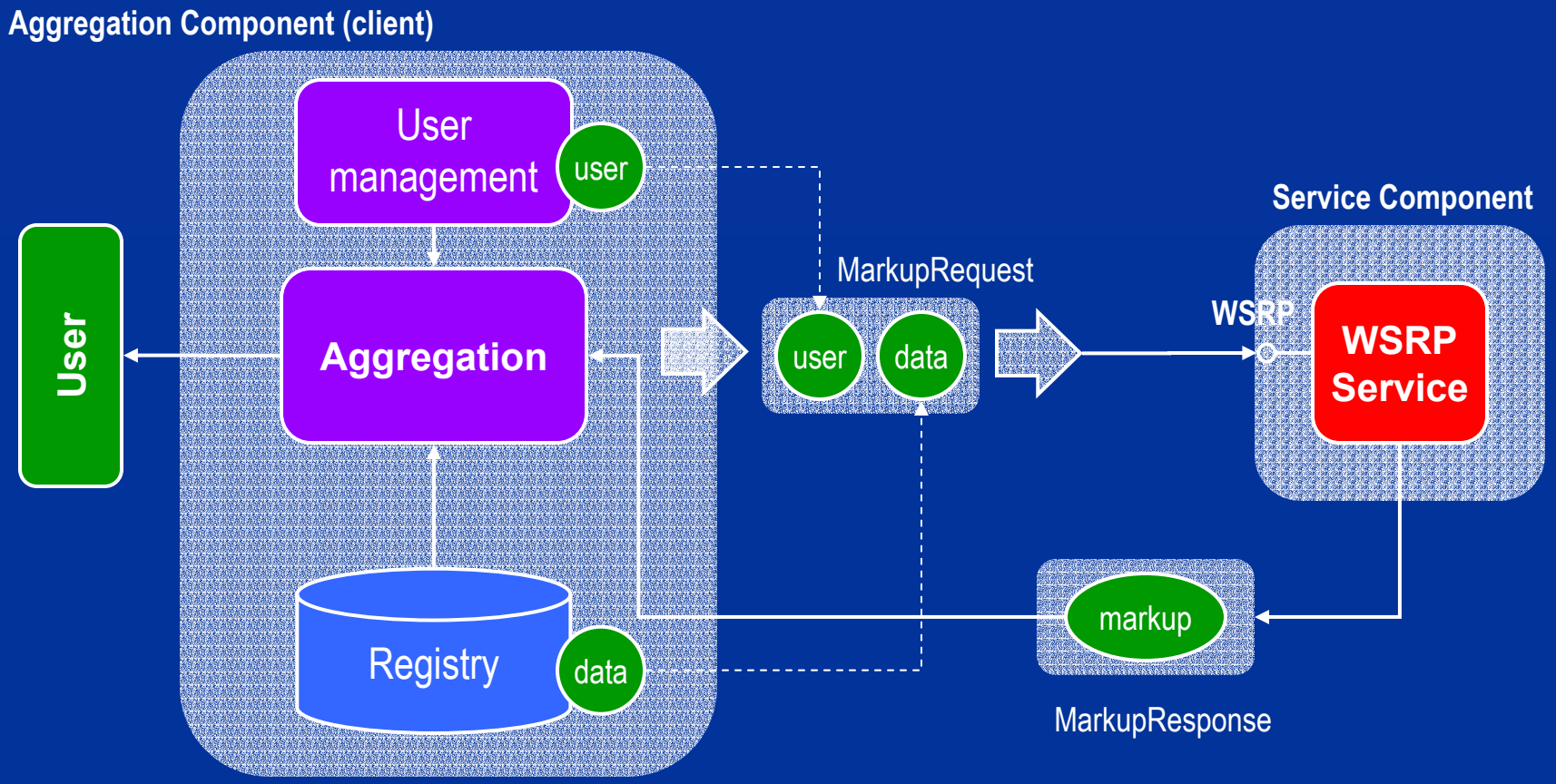
# Example of Portal ⇔ WSRP Service Interaction



# Markup Retrieval

- Client
  - User information
  - Client state
  - Locale
  - Instance/session handle
  - Markup type
  - Request parameters
- Server
  - Generates markup based on the client‘s request data
  - May have internal state
  - May embed encoded action URLs in the markup
  - Use namespace to encode named entities

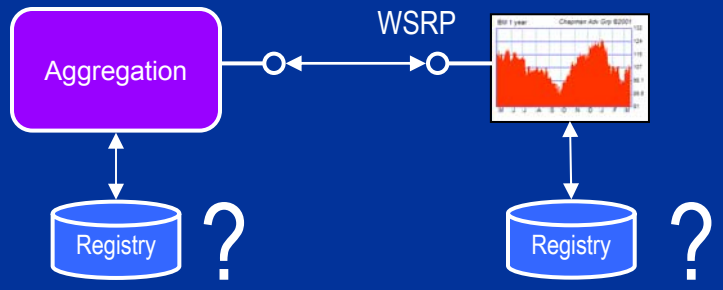
# Markup Retrieval (cont'd)



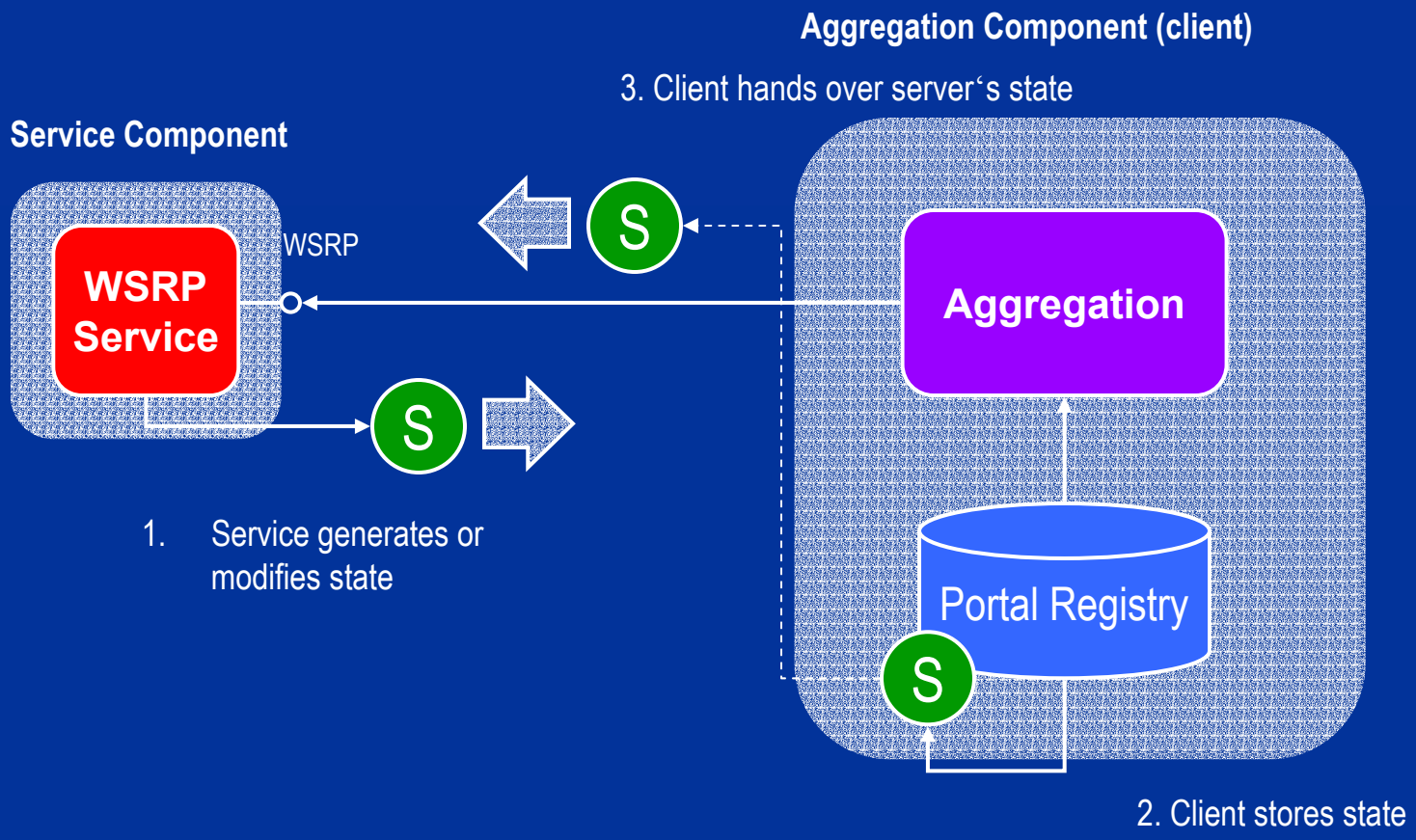

2. Client stores state

# State Handling

- Idea
  - Let the server decide whether to store persistent data on the server or client
- Concept
  - Allow the server to return its (modified) state in a serialized form to the client
  - The client persists the server‘s state and passes it to the server in each request
  - Servers may choose to persist only parts of its state (security)



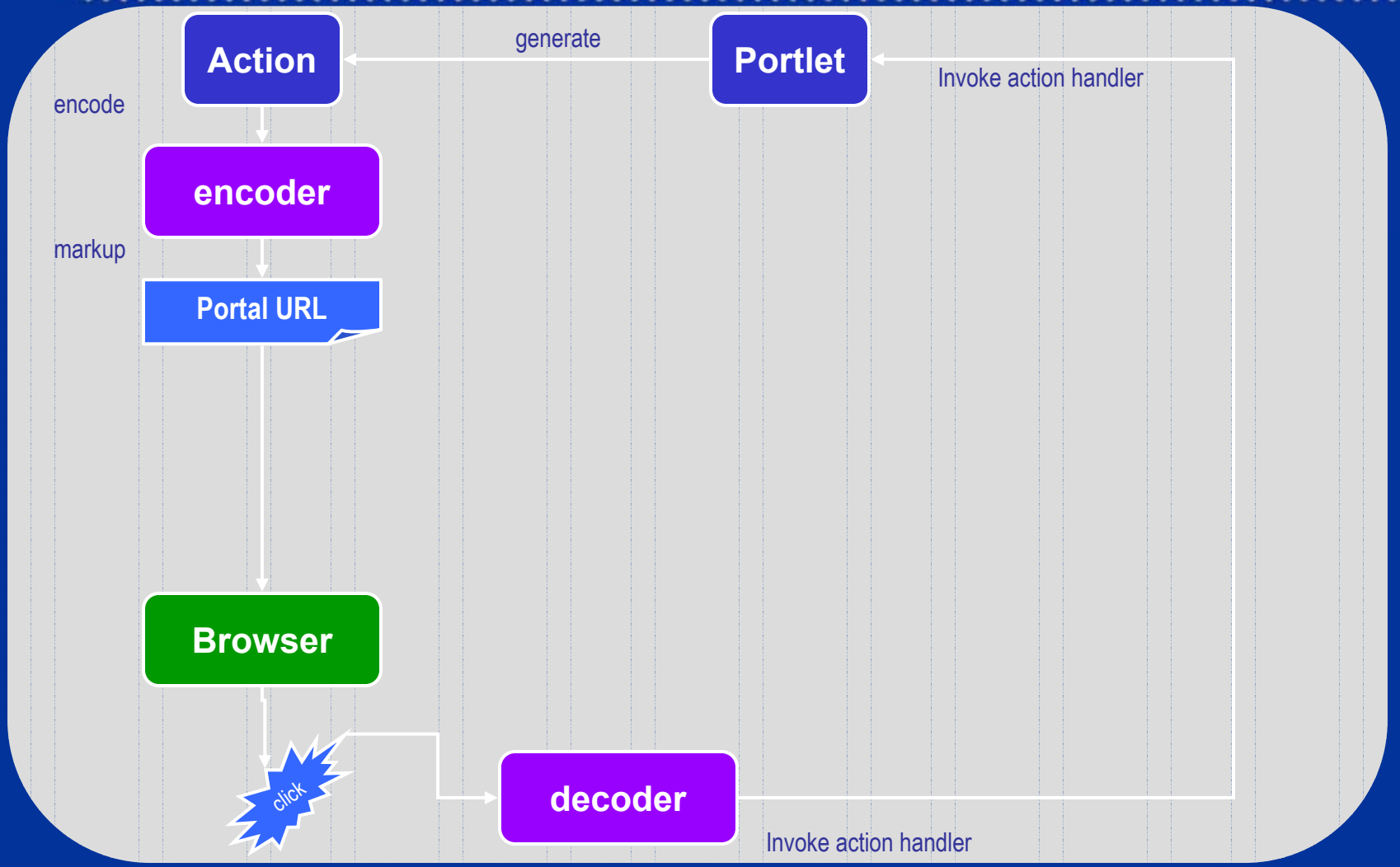
# State Handling (cont'd)



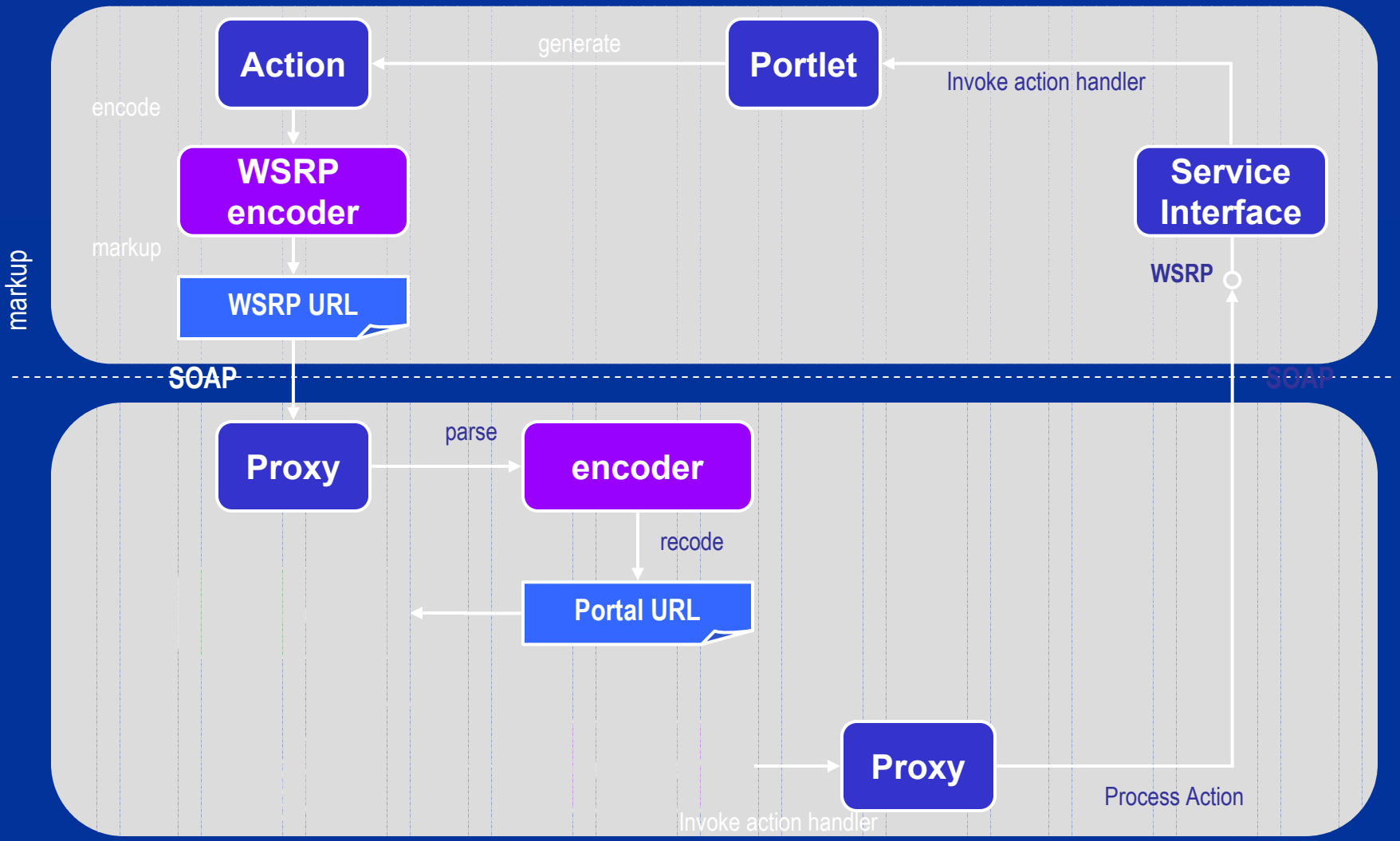
# Action Handling

- Server
  - Encodes actions in a special WSRP URL syntax
- Client
  - Recodes WSRP action URLs to match the portal‘s URL syntax
  - Intercepts WSRP URL clicks
  - Invokes action processing via WSRP
- Server
  - Processes an action and optionally invalidates its markup
- Client
  - Requests new markup if necessary

# Action Handling (local case)



# Action Handling (remote case)

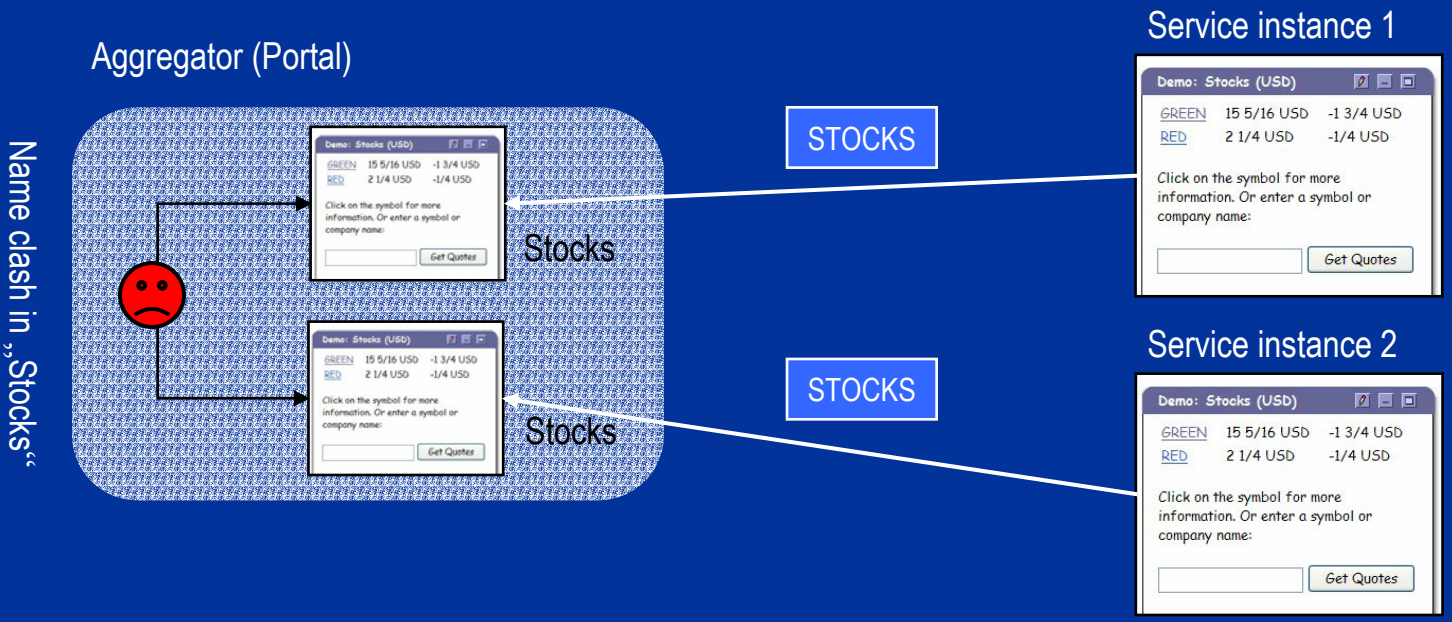


# Action Handing (summary)

- Transparency
  - Actions are represented by handles
  - No changes in portlet programming required
  - Client can handle WSRP URLs without knowledge of the server‘s details
  - Neither client nor server needs to be a portal
- Uniqueness
  - URLs are automatically unique by using a GUID
- Efficiency
  - Simple string replacement required on client side (eg. BM algorithm)

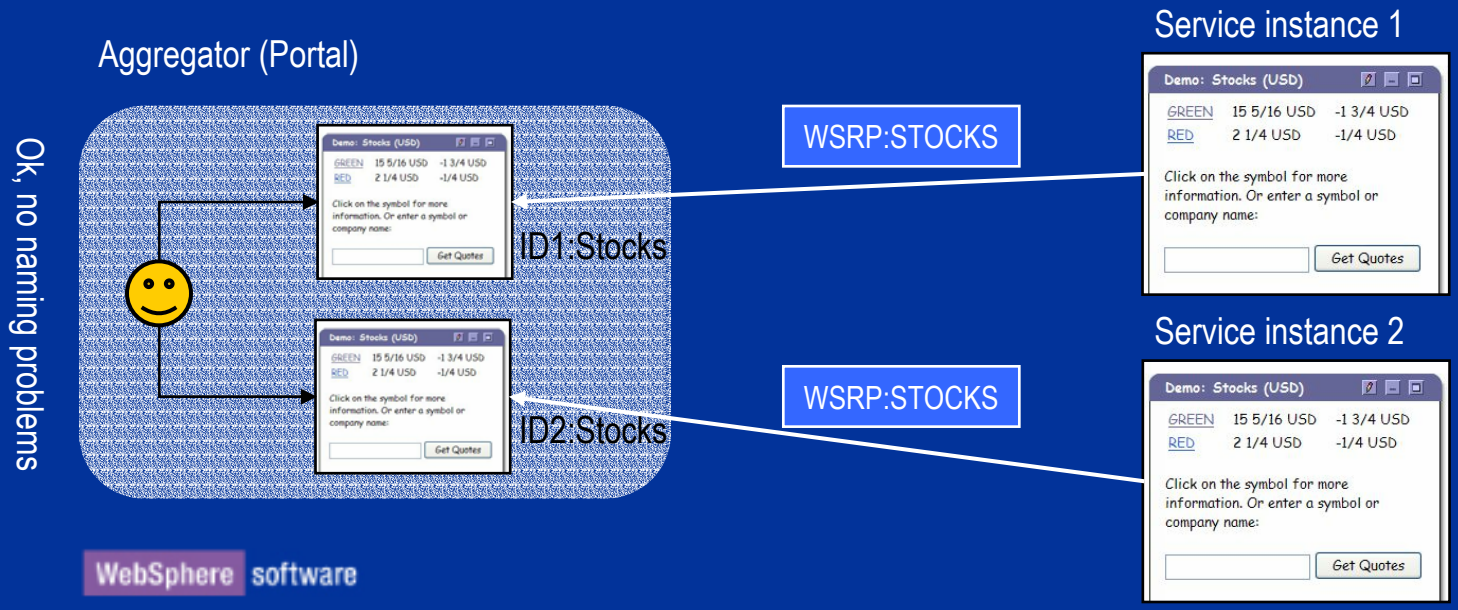
# Namespace Encoding

- Problem
  - The portlet's markup may contain named entities (e.g. form names)
  - Names from different portlets that are aggregated onto a single page may conflict
  - The same portlet may be aggregated multiple times which leads to conflicting names

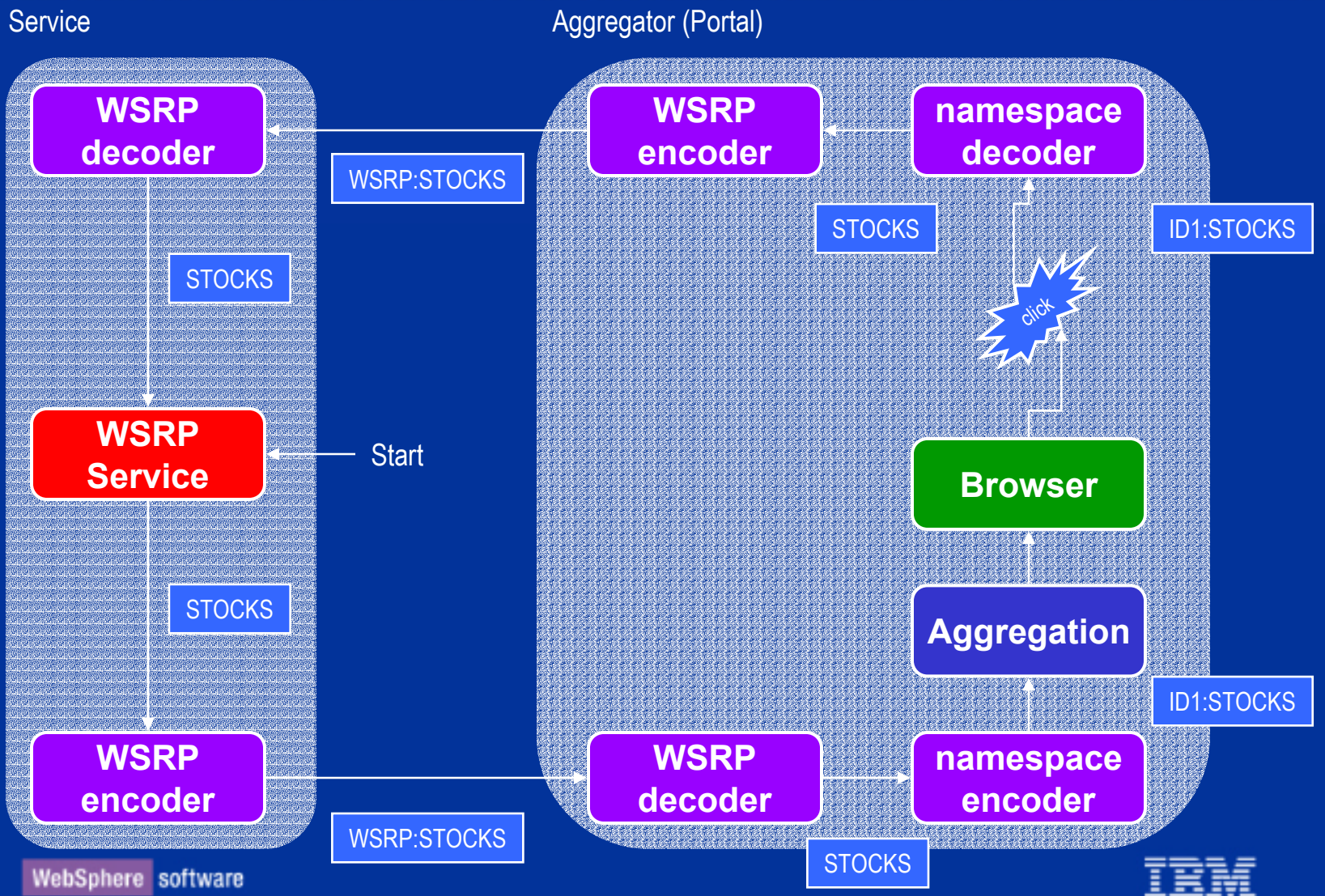


# Namespace Encoding (cont'd)

- Solution
  - The aggregator must lift every named entity into a unique namespace
  - When passing the names to the portlets the namespace must be resolved for each destination portlet
  - **How to locate the named entities in the markup?**
    - Write a parser for each markup type ☹
    - Let the portlet indicate the names by tagging them ☺



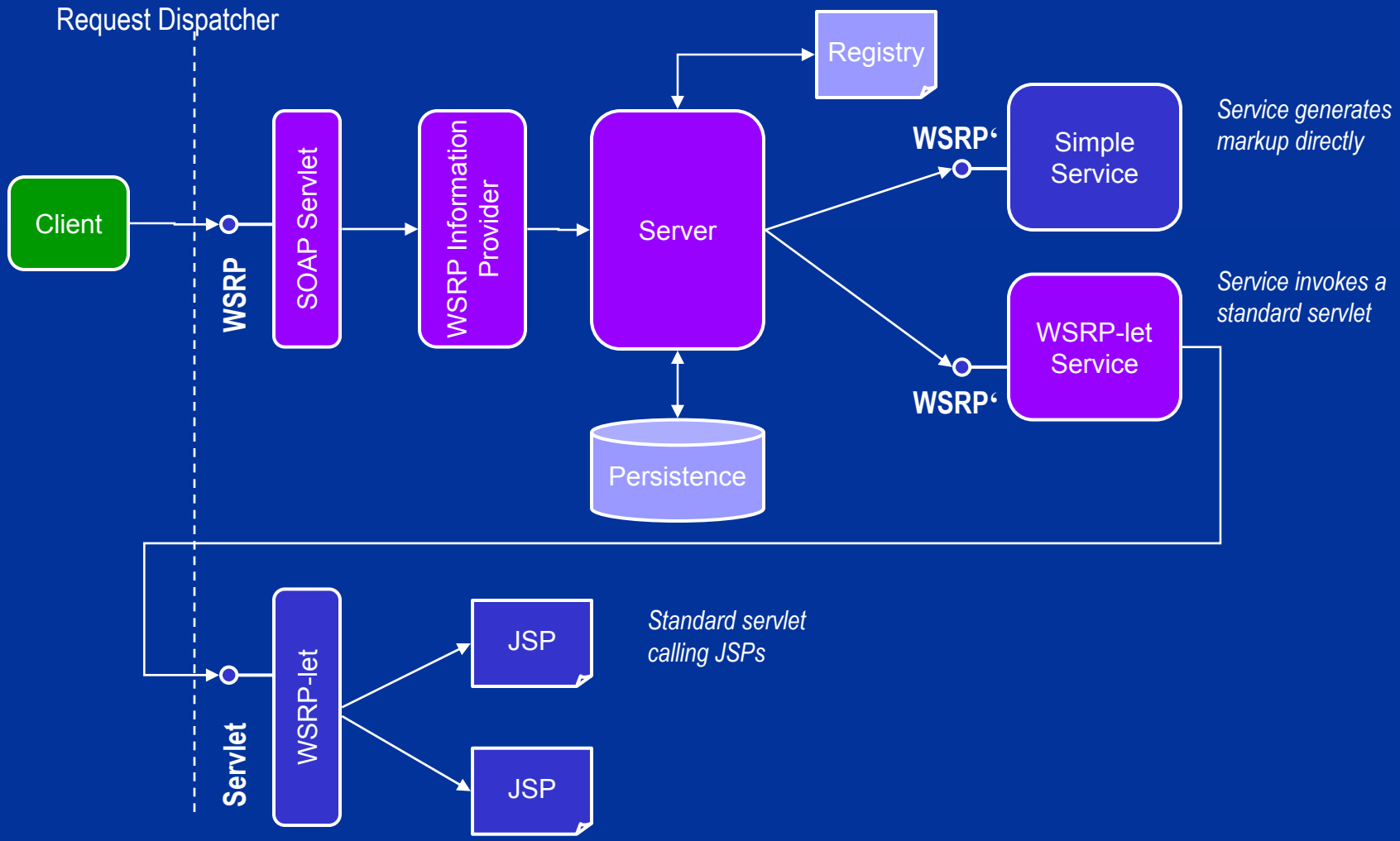
# Namespace Encoding (cont'd)



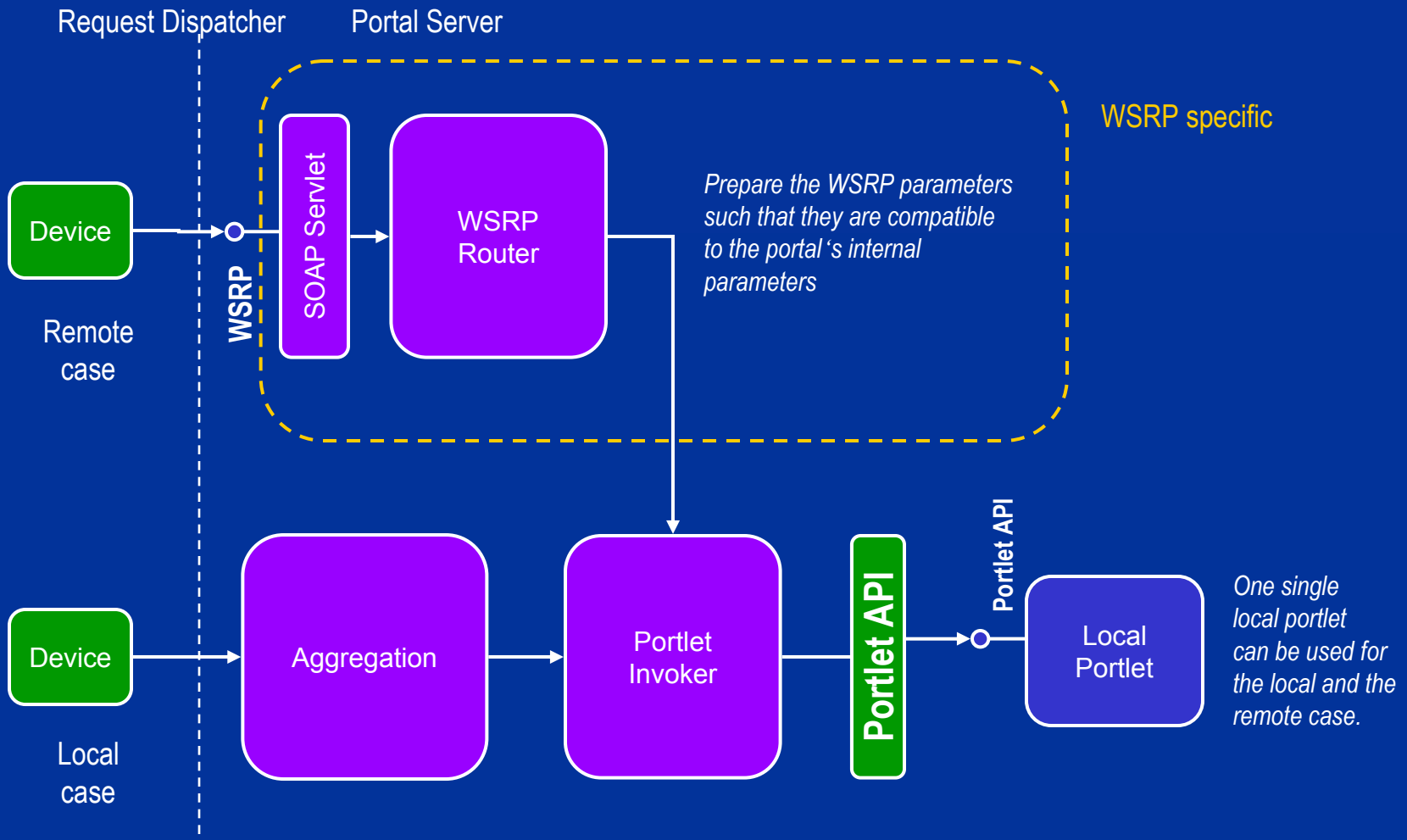
# Possible Implementations

- Baseline
  - Implementing both a WSRP server and a WSRP client is **very simple**!
- Sample Implementations
  - Java based WSRP Server (on Tomcat)
  - Java (Swing) based WSRP Client
  - .NET service as WSRP Server
  - ActiveX Control as WSRP Client

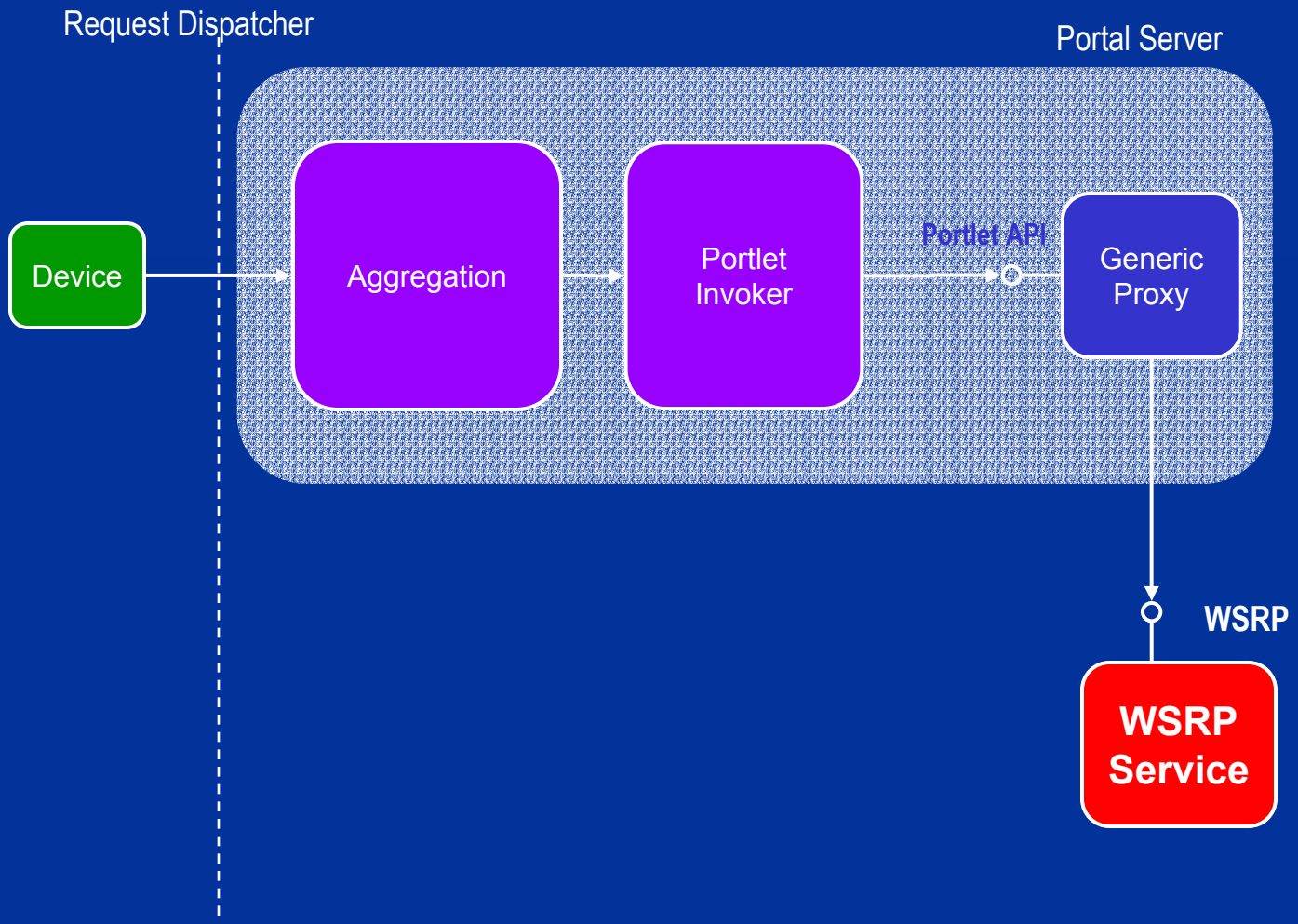
# Implementing a WSRP Server on Tomcat



# Making a portal server a WSRP server



# Making a portal server a WSRP client



# Open Source Projects used for Reference Impl.

- Projects for server impl.
  - Jakarta Tomcat
    - Application server
  - Axis
    - SOAP implementation
  - UDDI4J
    - UDDI access
  - Xerces
    - XML parsing
- Standards
  - SOAP
    - Basis for the communication layer
    - Guarantees interoparability
  - WSDL
    - Used as the interface definition
  - UDDI
    - Serving portals publish the UDDI to make their content available
    - Client portals can query published portals from UDDI
    - UDDI registry strores binding and configuration information
  - WSIA
    - WSRP is a special case of a WSIA service
    - Remote portlets will be accessible from WSIA clients
  - Portlet API

# WSIA ⇔ WSRP Relationship

- Basic Issues
  - WSIA provides a **generic** set of basic **interfaces** for life cycle, presentation, persistence and event handling
  - WSRP **extends** selected WSIA interfaces and provides an own specialized interface to its services

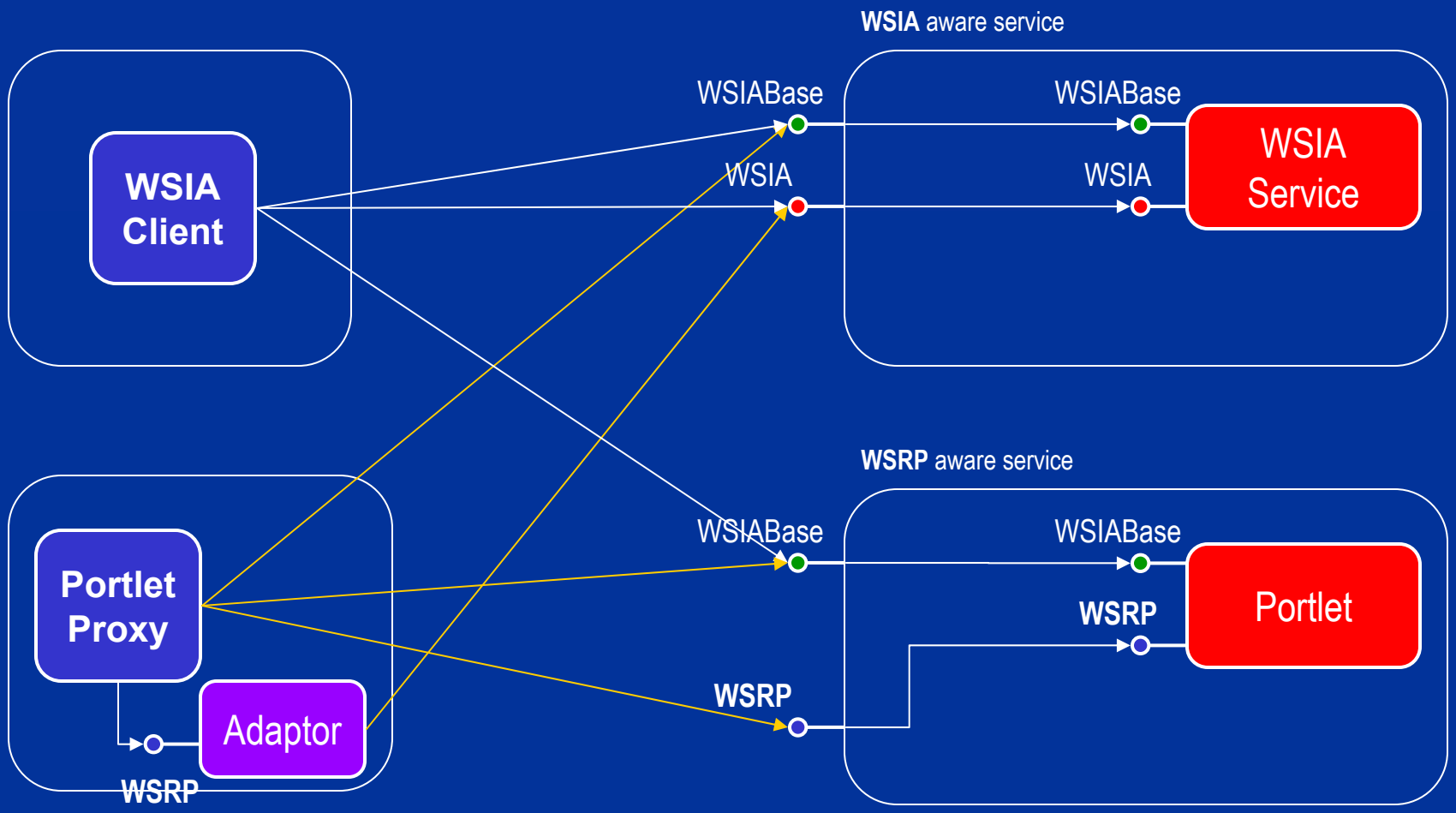
- Design Goals
  - Implementing a compliant service should be as **easy as possible**
  - Services should be extensible
  - Accessing interfaces should be **efficient**

# WSIA ⇔ WSRP Interoperability

- Interoperability goals
  - Specialized clients can access WSRP services by only using the WSRP interface
  - Generic clients can make use of WSRP services by only using the WSIA interfaces

- Implications
  - WSRP services must implement the WSIA interfaces for special cases only (the WSRP relevant ones)

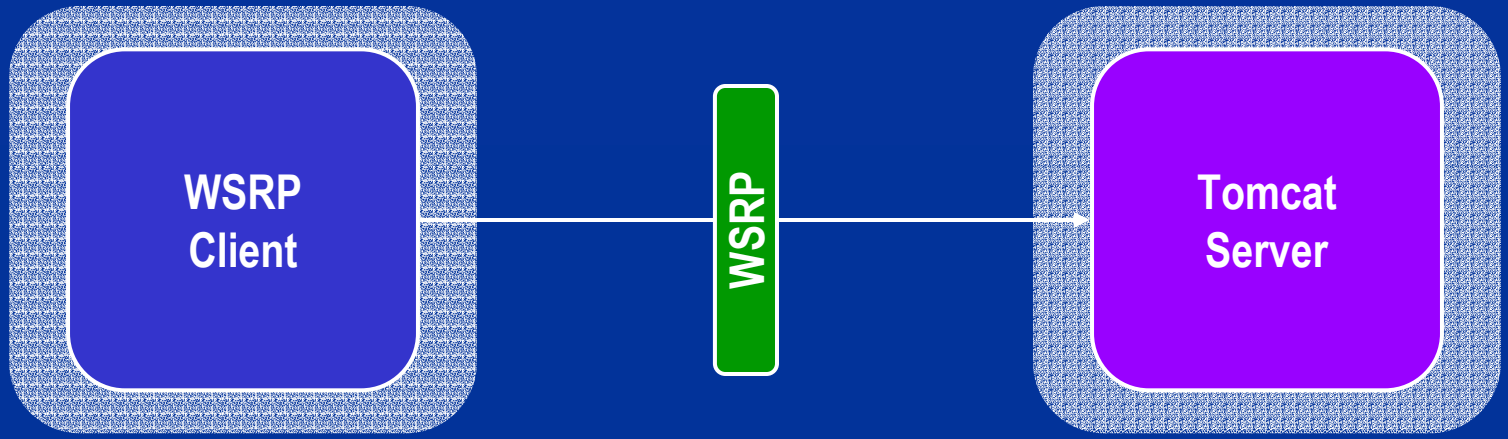
# WSIA ⇔ WSRP Interoperability (cont'd)



# Summary

- A standardized portlet model is key to interoperability between portal servers and portlets
- Portlets can be published as WSRP services
- WSRP services can be wrapped in Portlets
- Interoparability with different platforms
  - J2EE client and server
  - .NET client and server
- In sync with WSIA

# Client Server Communication (Java)




- To be demonstrated
  - Find a portlet via UDDI
  - Bind to the portlet
  - React on an action

# Java client consuming at .NET service

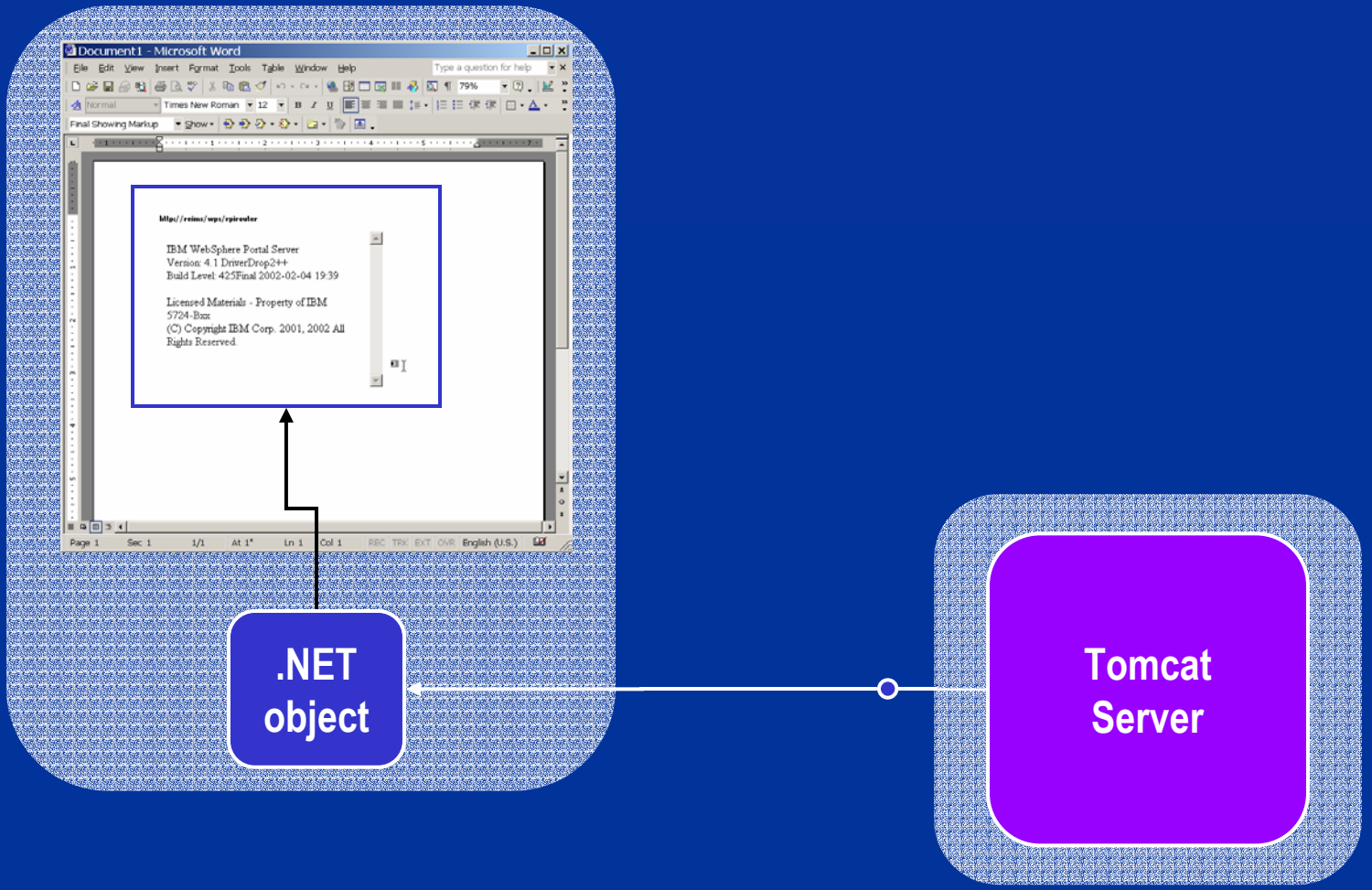

WSRP

MS Office as OLE server

IIS

.NET object

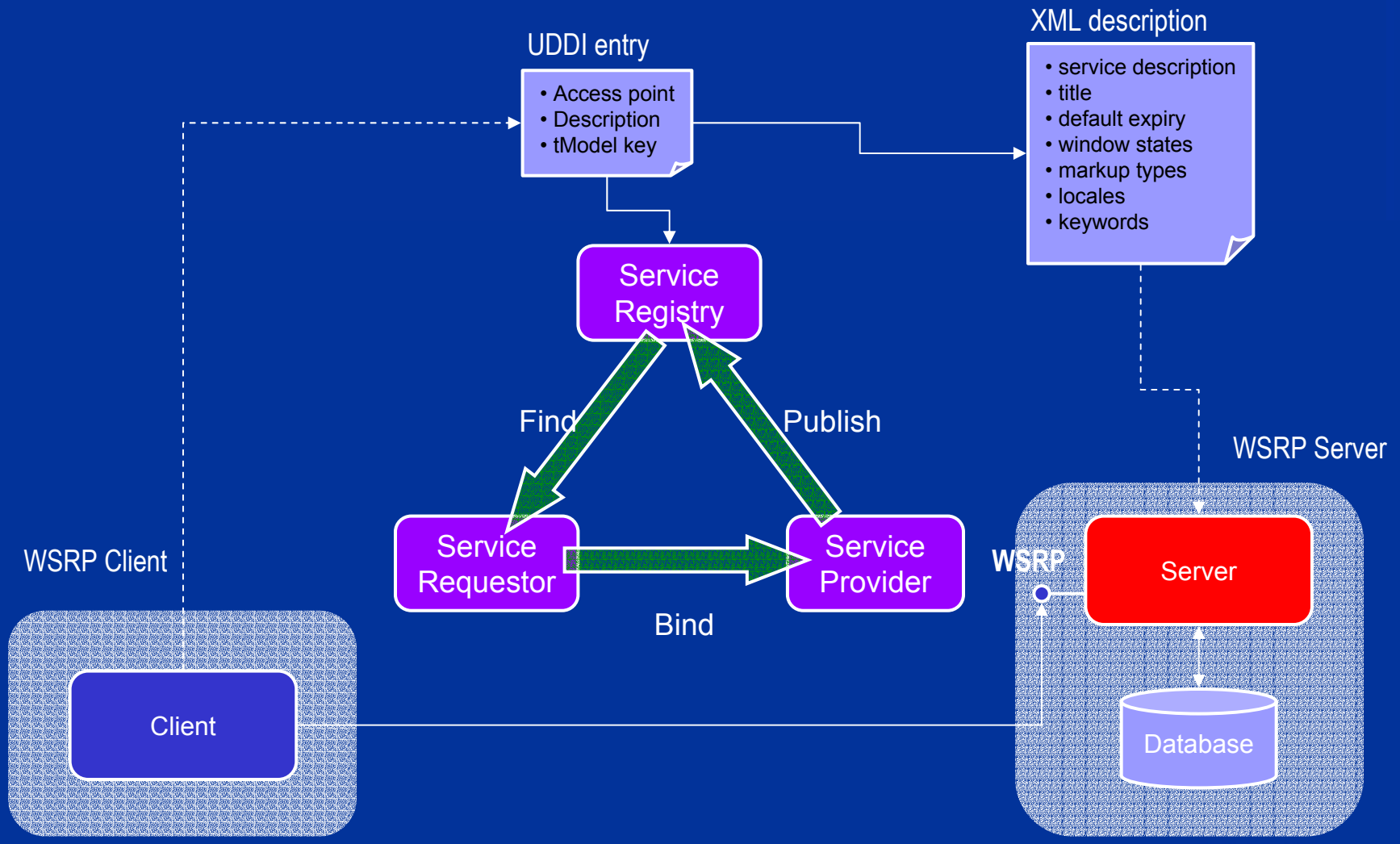
# WSRP service inside a Word Document



# Questions!?

# Thank you for your attention!

# Publish/Find/Bind



# Action Handling (URL recoding)

- Portlets
  - Use the same URL encoder in local and remote case
- Client
  - Provides different encoders for the local and remote (same interface) case
  - Each action is assigned a **locally unique** action handle
  - Recognizes WSRP URLs in the markup
  - Decodes the URL to distinguish between window state changes and remote action handles
- WSRP
  - Defines globally unique action identifiers which prefix action handles

# Information Flow

- Persistent Attributes
  - Portal registration ⇨ initialization per portal
  - Portlet ID to bind to ⇨ initialization per portlet

- Transient Attributes
  - Markup type (HTML, VoiceXML, etc.)
  - Locale
  - User information
  - Window state
  - Session
  - Actions

  ⇨ transfer per request

WebSphere software

IBM

e business software

# Web Services for Remote Portlets (WSRP) – some Details

March 18-20 2002

Dr. Carsten Leue

Thomas Schäck

Peter Fischer

IBM Software Group

# Summary

- WebServices for Remote Portlets
  - Recapitulation: Architecture and Design
  - Interfaces
  - Protocoll
- Implementation
  - Client Sample
  - Server Sample
- Misc
  - Performance, Caching
  - Discussion

# Portlet / User Interaction

- User actions
  - Read the markup
  - Trigger actions by clicking on links
  - Enter data into forms and send it to the server
  - Edit the portlet

- Portlet
  - Generate markup based on user settings (identitiy, markup type, locale)
  - Encode actions as links in the markup
  - Encode namespaces
  - React on actions triggered by the user by modifying the portlet‘s state
  - Receive form data entered by the user and process it

# Performance

- Goal
  - Use as little rountrips as possible
    - two roundtrips to setup a portlet (*bind, createInstance*)
    - one single roundtrip per markup request (*getPortletMarkup*)
    - one or two roundtrips per action (*invokePortletAction*)
  - Enable caching of remote services
  - Efficient markup interpretation
    - Fast string search algorithms
    - Use of escape tokens for efficient parsing

# Caching

- Requirements
  - WSRP clients may want to provide caching to relieve the WSRP service
  - The WSRP service must provide information on the expiry of its content
- Solution
  - The WSRP service returns an expiry flag to the client on each markup call
  - Each action event may explicitely expire the markup
    - either a  subsequent getPortletMarkup call follows
    - or the service returns its markup in a single roundtrip

# Boyer – Moore String Search

- Requirements
  - String search is needed for URL rewriting and namespace encoding
- BM offers a verify efficient string search algorithm
  - Run time typically of order O(N/M)
    N = len(text), M = len(token)
  - Small initialization costs
- Implications for WSRP
  - Select the same escape token for URLs and namespaces
  - Choose the token as long as possible consisting of unlikely characters

(Robert S. Boyer, J Strother Moore: A Fast String Searching Algorithm, Communications of the ACM, October 1977, Volume 20, Number 10)

# WSRP Interface

```
public interface WSRPComponent extends WSXLBindable
{
    // markup and actions
    public WSRPMarkupResponse getPortletMarkup(
          String hPortlet,
          WSRPMarkupRequest request) throws WSXLException;

    public WSRPActionResponse invokePortletAction(
          String hPortlet,
          String hAction,
          WSRPActionRequest request) throws WSXLException;

    // instance management
    public String createPortletInstance(
          String hBinding,
          String classID) throws WSXLException;

    public String createSession (String hPortlet) throws WSXLException;

    public void destroyInstance (String hGeneric) throws WSXLException;

    public String bindClient () throws WSXLException;

};
```

# WSRP Interface – bindClient

```
public String bindClient () throws WSXLException;
```

- Input:
  - None
- Output:
  - Handle identiying the binding

- Remarks:
  - Can be omitted if the binding information has been transfered otherwise

# WSRP Interface – createPortletInstance

```
public String createPortletInstance(
        String hBinding, String classID) throws WSXLException;
```

- Input:
  - Binding handle
  - Identifier of the service within the server (LUID)
- Output:
  - Handle identiying the remote instance

```
String hBinding = bindClient();
String hInstance = createPortletInstance(hBinding,"107");
...
destroyInstance(hInstance);
destroyInstance(hBinding);
```

# WSRP Interface – destroyInstance

```
public void destroyInstance (String hGeneric) throws WSXLException;
```

- Input:
  - Arbitrary handle
- Output:
  - none

```
String hBinding = bindClient();
String hInstance = createPortletInstance(hBinding,"107");
...
destroyInstance(hInstance);
destroyInstance(hBinding);
```

# WSRP Interface – getPortletMarkup

```
public WSRPMarkupResponse getPortletMarkup(
        String hPortlet,
        WSRPMarkupRequest request) throws WSXLException;
```

- Input:
  - Instance or session handle
  - Markup request containing client data, user data, portlet state etc.
- Output:
  - Markup response

```
String hBinding = bindClient();
String hInstance = createPortletInstance(hBinding,"107");

WSRPMarkupResponse res = getPortletMarkup(hInstance,request);
System.out.println(res.getResultString());

destroyInstance(hInstance);
destroyInstance(hBinding);
```

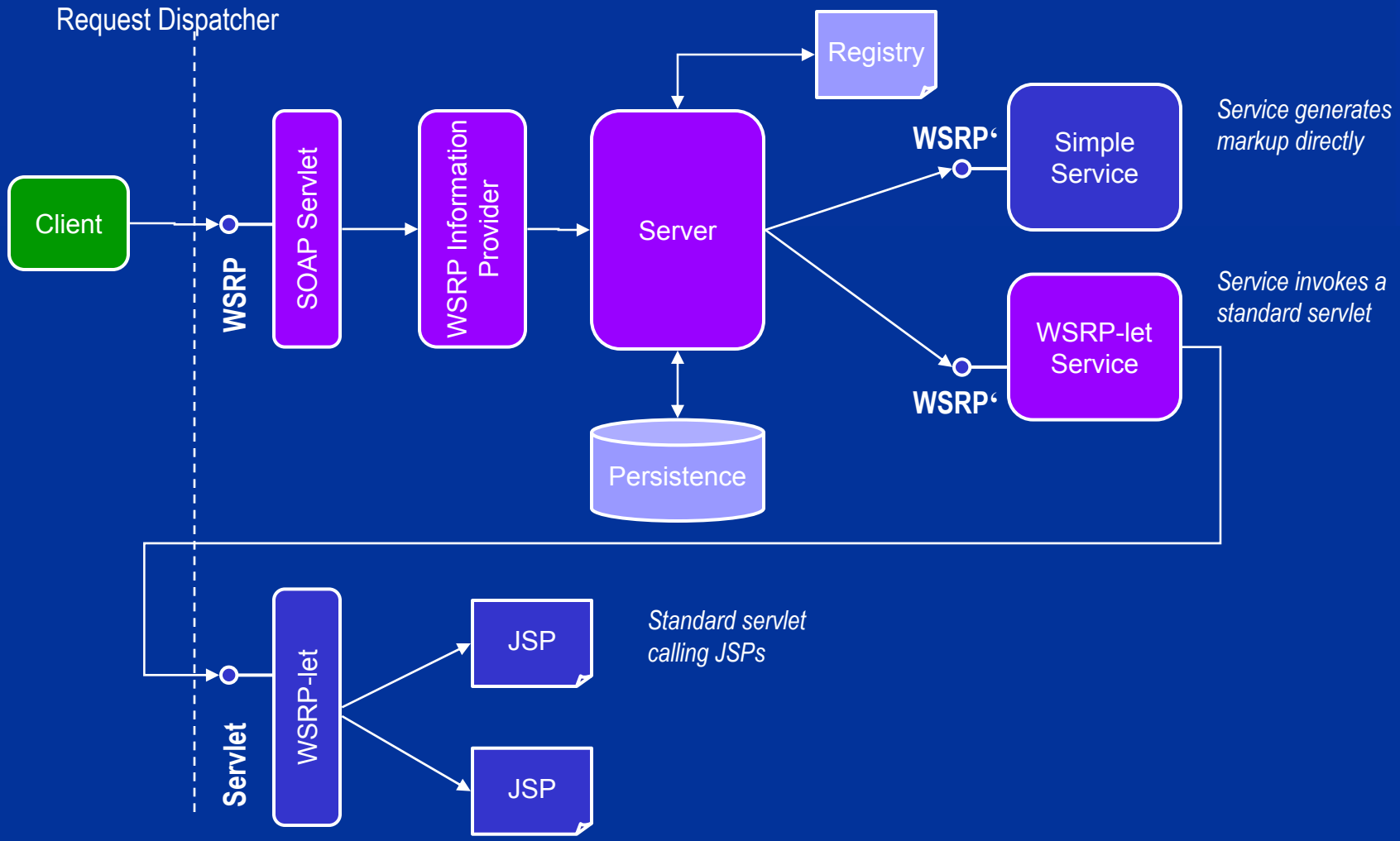
# WSRP Interface – invokePortletAction

```
public WSRPActionResponse invokePortletAction(
    String hPortlet, String hAction, WSRPActionRequest request)
     throws WSXLException;
```

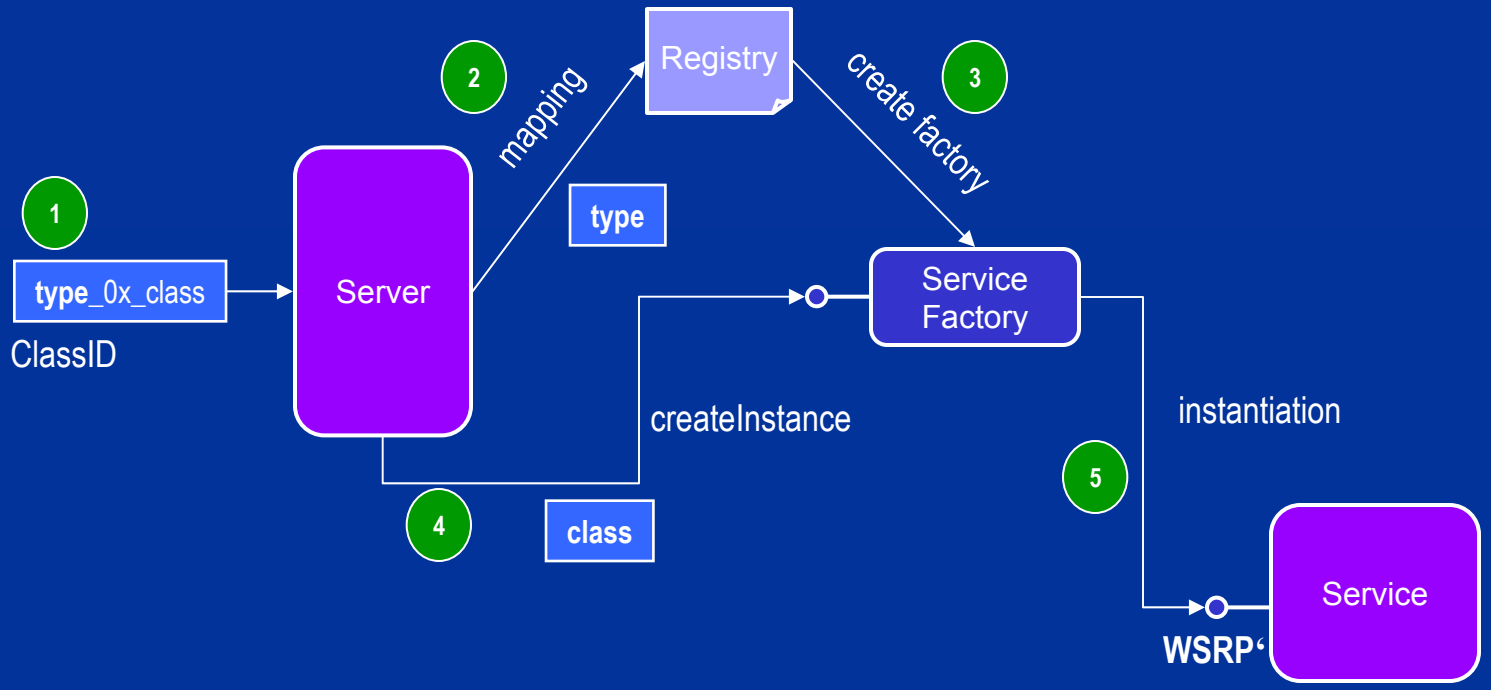
- Input:
  - Instance or session handle
  - Action handle
  - Action request containing client data, user data, portlet state etc.
- Output:
  - Action response

```
...
WSRPActionResponse res = invokePortletAction(hInstance,hAction,request);
String out = res.getResultString();
if (out==null)
    out = getPortletMarkup(hInstance,request).getResultString();
System.out.println(res.getResultString());
```

# Implementing a WSRP Server on Tomcat



# Instantiating a WSRP service




- the server defers the service type from the incoming class handle
- the registry maps the service type against a service factory
- the factory instantiates the service

# Summary

# Questions!?

# Thank you for your attention!