Best Practices for Building Optimized Wireless Solutions for Web Services

Michael Shenfield

Director, Standards Architecture
RIM

Bryan Goring

Architect, Advanced Technologies
RIM

TS-1293

Goals

Learn principles for building optimized Java™ ME frameworks to host Web Service client applications

Learn patterns for designing effective device applications and wireless friendly Web Services

Agenda

Challenges of Wireless Access to WS

Java ME Container + Services + Components

WS Client Applications: Selected Patterns

Designing Wireless Friendly WS

Demo

Agenda

Challenges of Wireless Access to WS

Java ME Container + Services + Components

WS Client Applications: Selected Patterns

Designing Wireless Friendly WS

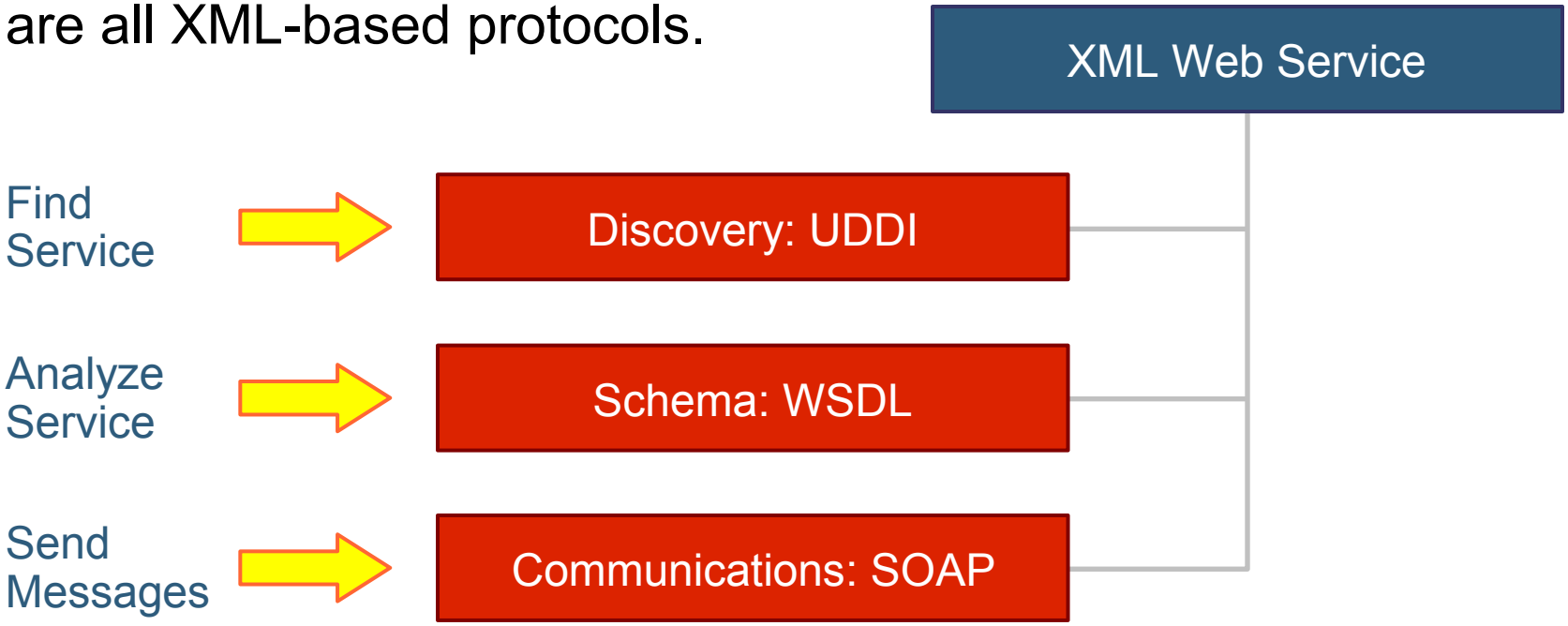
Demo

What Is a Web Service?

- An application component that can be called remotely using standard Internet Protocols such as HTTP and XML
- A unit of code that can be activated using HTTP requests
- Web Service is a programmable URL
- The purpose of Web Services is to deliver distributed computing over the Internet
- Web Services architecture allows programs written in different languages on different platforms to communicate with each other in a standards-based way

“Classic” Web Services

XML Web Service is a software service exposed on the Web through SOAP protocol, described with a WSDL file and registered in UDDI registry. UDDI, WSDL, and SOAP are all XML-based protocols.



Challenges for Wireless

Typical WS is not designed/optimized for a wireless client

- Use of multiple levels of nesting for complex data types
- Auto-generation of WS from J2EE, DB, or .NET applications results in late binding and loosely defined types:
  - Endpoint redirect at execution time
  - Complex type polymorphism
  - Runtime type resolution (e.g., “xsd:any”, “xsd:union”, etc.)
- In wireless space, neither storage nor traffic are free; Many WSs are designed to return large datasets (e.g., “xsd:maxOccurs=unbounded”):
  - Weaker processors implies slow message processing
  - Limited storage space for large datasets
  - Excessive garbage collection
  - Waste of battery life

Challenges for Wireless (Cont.)

- Latency is the #1 killer for user experience. The best wireless applications use wireless the least (i.e., the data is present on a device when the user needs it) and are based on a push paradigm. Very few asynchronous Web Services are available due to lack of common standard and tools.
- Coverage loss or intermittent coverage. Loss of coverage could result in an inconsistent state of the device application, Web Service, or both.
- The high complexity of implementing Web Service security model in wireless (enterprise, banking, insurance, payments, etc.)

Agenda

Challenges of Wireless Access to WS

Java ME Container + Services + Components

WS Client Applications: Selected Patterns

Designing Wireless Friendly WS

Demo

Path to Solution: Container Framework

Critical Requirements for Wireless Applications:

- Small size and execution efficiency
- Security of data access on and off the device
- OTA manageability: Installation, upgrade, removal, data preservation on upgrade, version management, lifecycle management

Solution:

- Container framework—Controlled environment that encapsulates the execution of wireless applications

50-50 Rule

When building Java ME applications:

~ 50% of the code is application-specific workflow

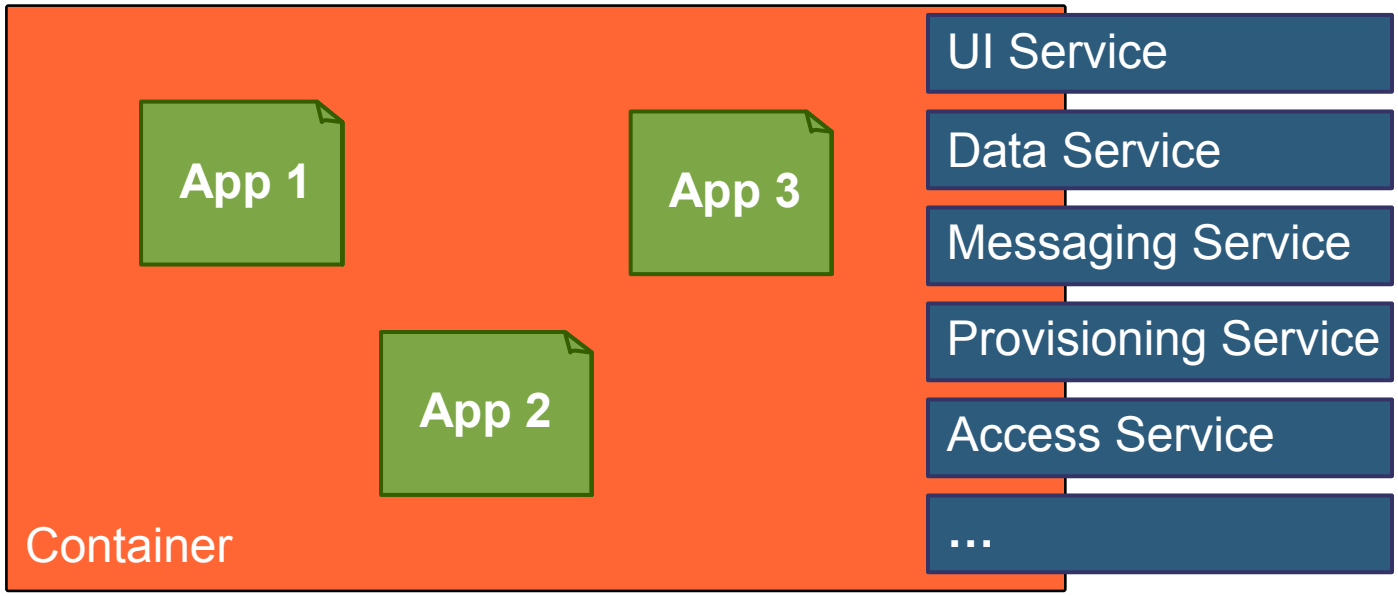
- Code complexity is not very high
- Code is unique for each application

~ 50% is the code for “common tasks” such as data management, building screens, and message processing

- More complex Java ME code
- Same patterns for most applications
- Could be reduced if high-level APIs were available

Path to Solution: Common Services

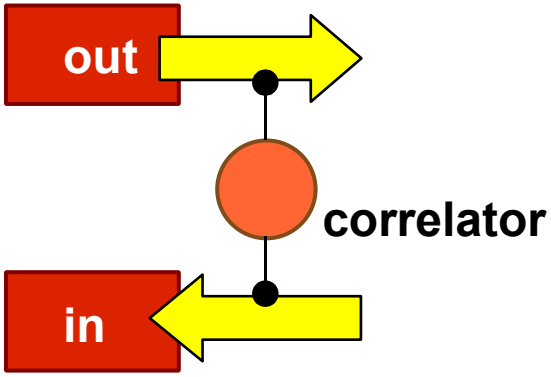
- Common services are shared between applications and represent high-level, template-based APIs
- To display a screen, an application passes a screen template to the UI service; to send a message, it passes a data template to the messaging service, etc.
- Access to services is mediated by an application container



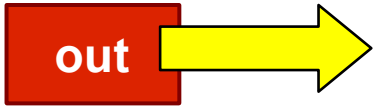
Path to Solution: Forget Synchronous

Focus on WS messages instead of operations

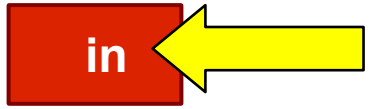
Correlated In-out Messages
(Sync WS Operation)



Oneway Out Message
(Async Request)



Uncorrelated Notification
(Async Response)

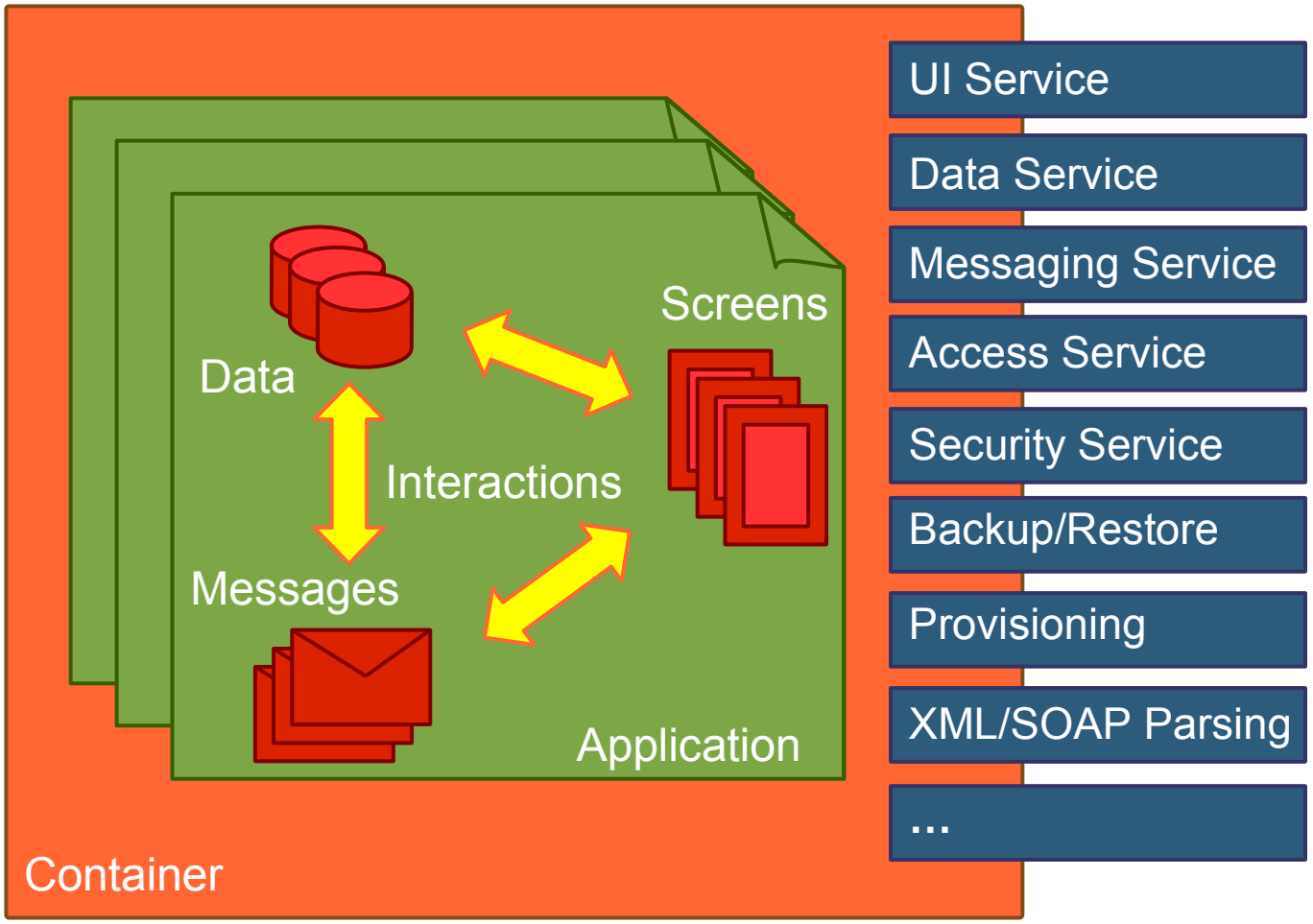


Path to Solution: Application as a Set of Component

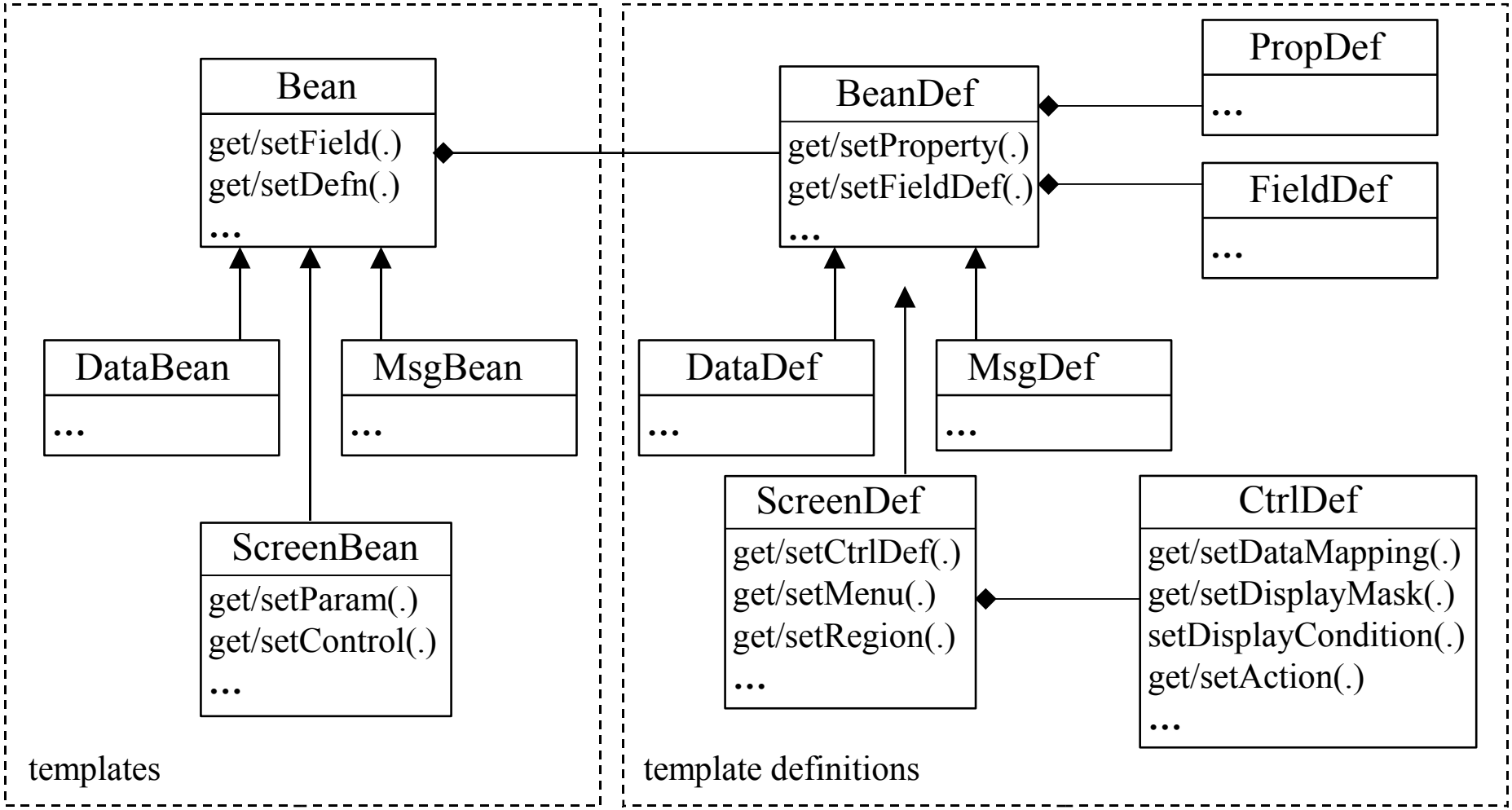
Why components?

- Messages, data, and screens are modular entities with predefined sets of interactions such as events, actions, data bindings, etc.
- Message and data components can be derived from WS messages
- Component structure can be expressed using meta-data
- Components and interactions can be expressed in the Java programming language, XML, etc.
- Component application can be provisioned as data and executed using templates within a container (“executable metadata” model)
- Component templates can be exchanged between the application container and common services to execute an application’s workflow

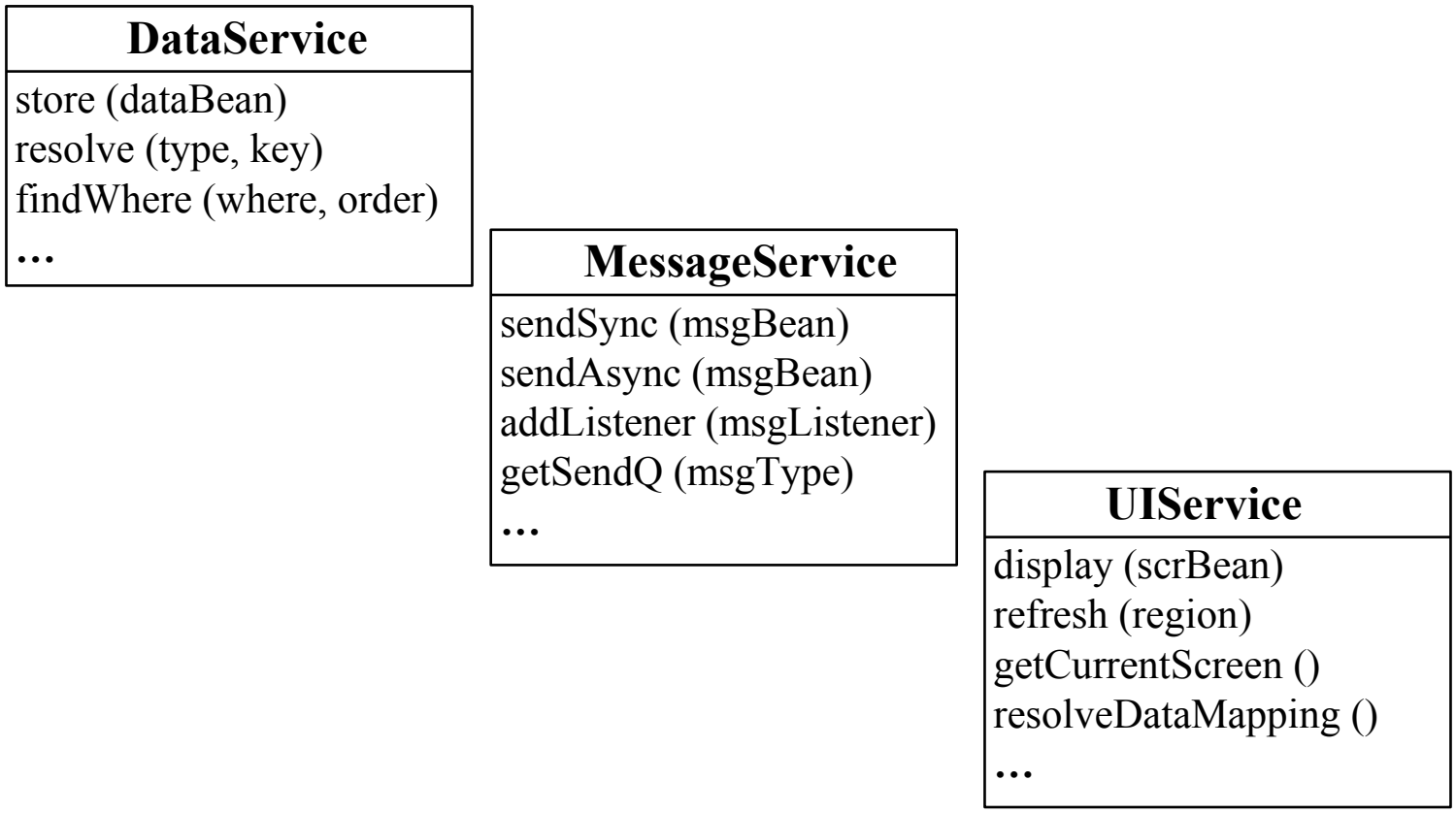
Container + Services + Components



Component Applications (Java Based)



Common Services (Java Based)



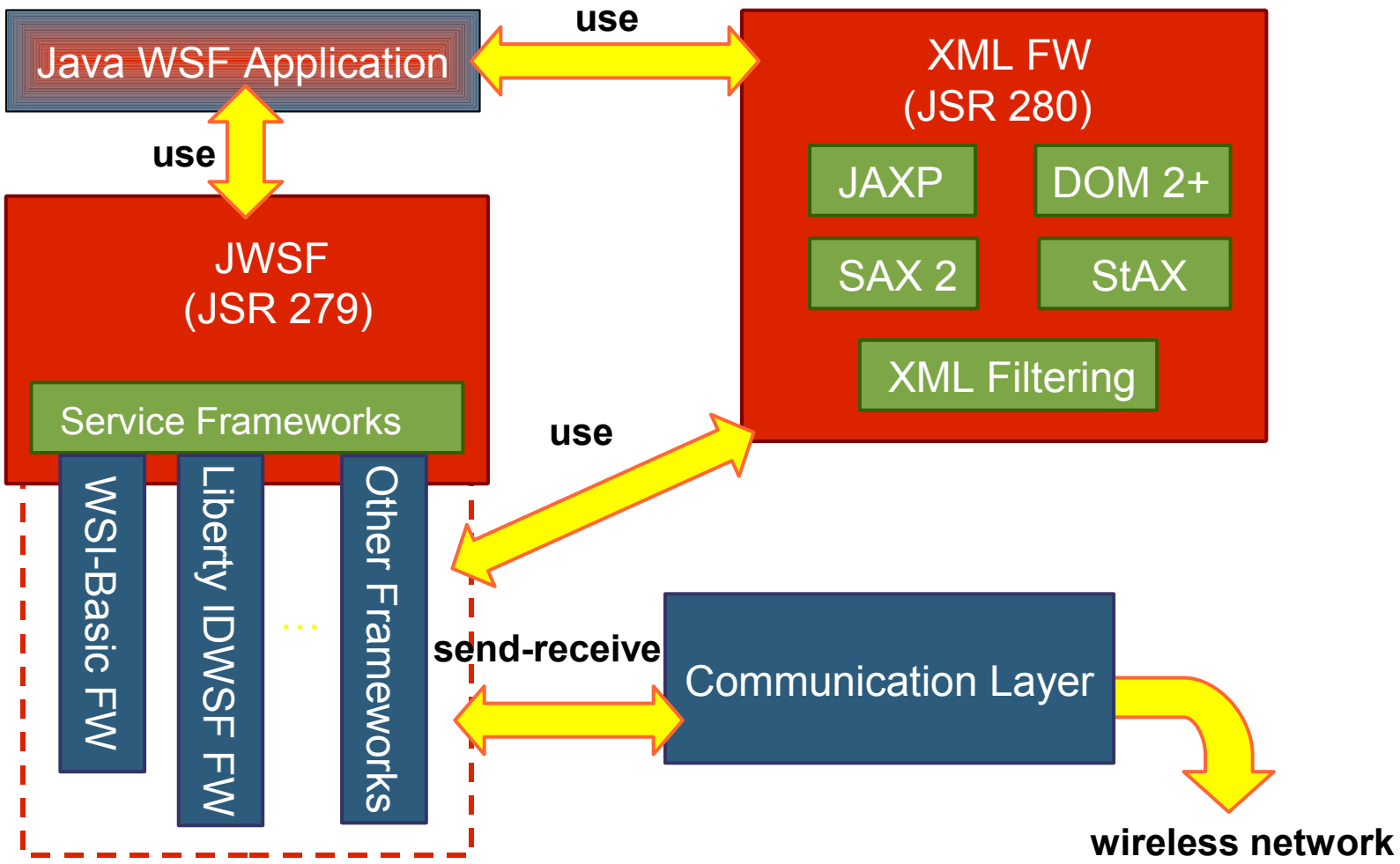
templates

template definitions

Component Applications (XML)

```
<!ENTITY % commonFieldAttribs 'name CDATA #REQUIRED
    type (string | integer | long | decimal | boolean | date | data | enumeration) "string"
    component IDREF #IMPLIED
    array (true | false) "false"'>
...
<!ELEMENT app (desc?, resource*, global*, enum*, data*, msg*, screen*, script*)>
…
<!ELEMENT data (field*)>
<!ATTLIST data
    name ID #REQUIRED
    prototype CDATA #IMPLIED
    persist (true | false) #IMPLIED
    key CDATA #IMPLIED
>
<!ELEMENT field EMPTY>
<!ATTLIST field
    %commonFieldAttribs;
    default CDATA #IMPLIED
>
…
```

JSRs 279, 280


Java WSF Application
use
XML FW
(JSR 280)
JAXP
DOM 2+
SAX 2
StAX
XML Filtering
use
JWSF
(JSR 279)
Service Frameworks
use
WSI-Basic FW
Liberty IDWSF FW
...
Other Frameworks
send-receive
Communication Layer
wireless network

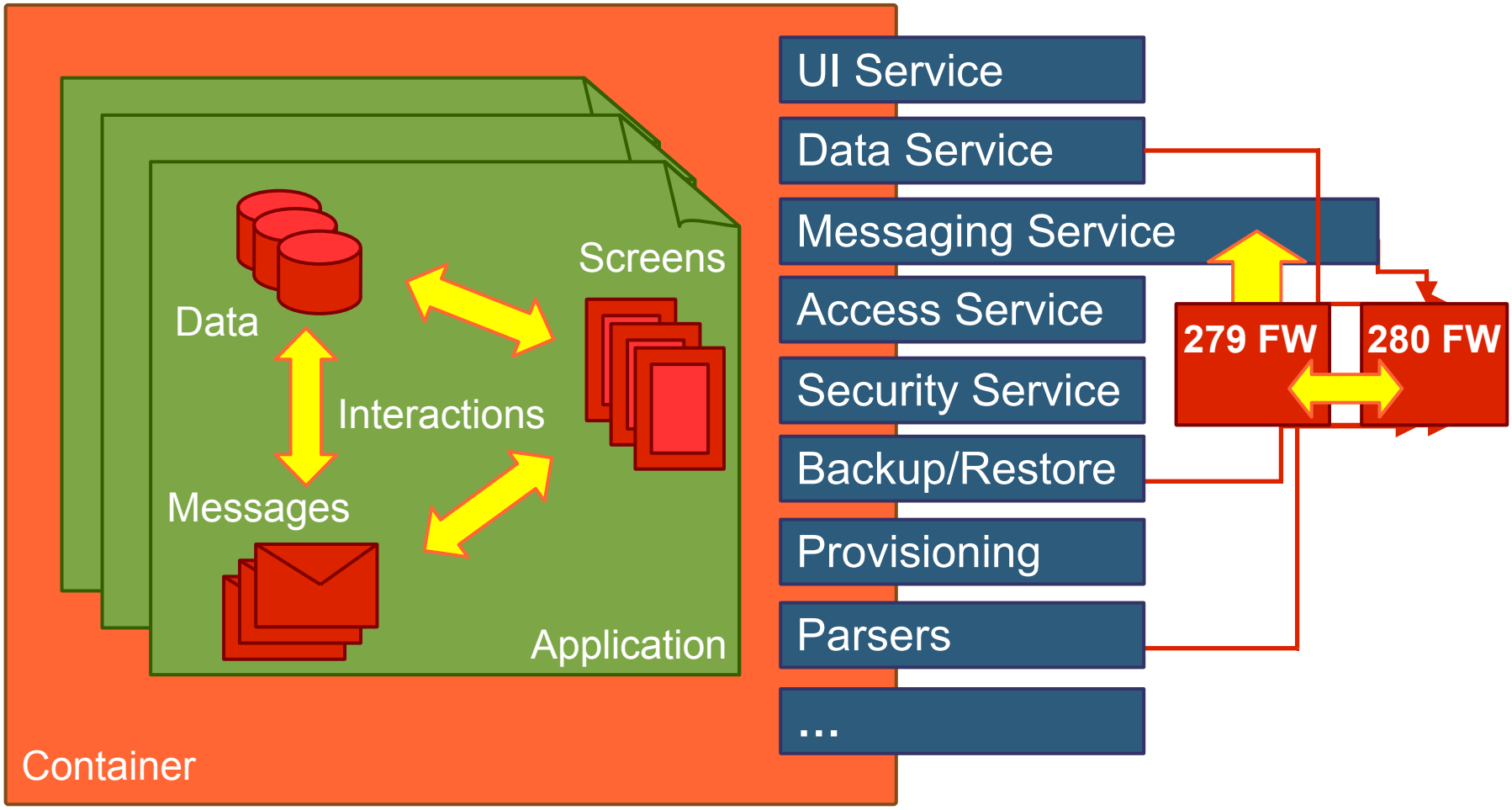
JSR 279, 280— Sample Flow for a Sync Call

```
…
// use XML API for Java ME (JSR 280) API to prepare WS request (StAX
    approach shown)
XMLOutputFactory outFactory = XMLOutputFactory.newInstance();
XMLStreamWriter sw = outFactory.createXMLStreamWriter (osRequest);
sw.writeStartDocument();
…
// use Service Connection API for Java ME (JSR 279) to obtain a connection and
    call the web service
ServiceDescriptor sd = new ServiceDescriptor (fwID, contract, endpoint);
ServiceConnection con = ServiceManager.getServiceConnection (sd);
InputStream isResponse = con.sendRcv (osRequest);
// use XML API for Java ME to parse WS response (DOM approach shown)
DocumentBuilder db = dbFactory.newDocumentBuilder();
Document docResponse = db.parse (isResponse);
…
```

JSR 279 API Used by Messaging Service

```
public MsgBean sendSync (MsgBean msg) {
    String contract = msg.getBeanDef().getContract();
    String endpoint = msg.getBeanDef().getEndpoint();
    OutputStream os = toSOAPStream(msg);
    // use Service Connection API for Java ME to obtain a connection and call the web service
    ServiceDescriptor sd = new ServiceDescriptor (WSI_B, contract, endpoint);
    ServiceConnection con = ServiceManager.getServiceConnection (sd);
    InputStream isResponse = con.sendRcv (osRequest);
    return fromSOAPStream(isResponse);
}
```

JSR 279, 280 FWs as Common Services



Agenda

Challenges of Wireless Access to WS

Java ME Container + Services + Components

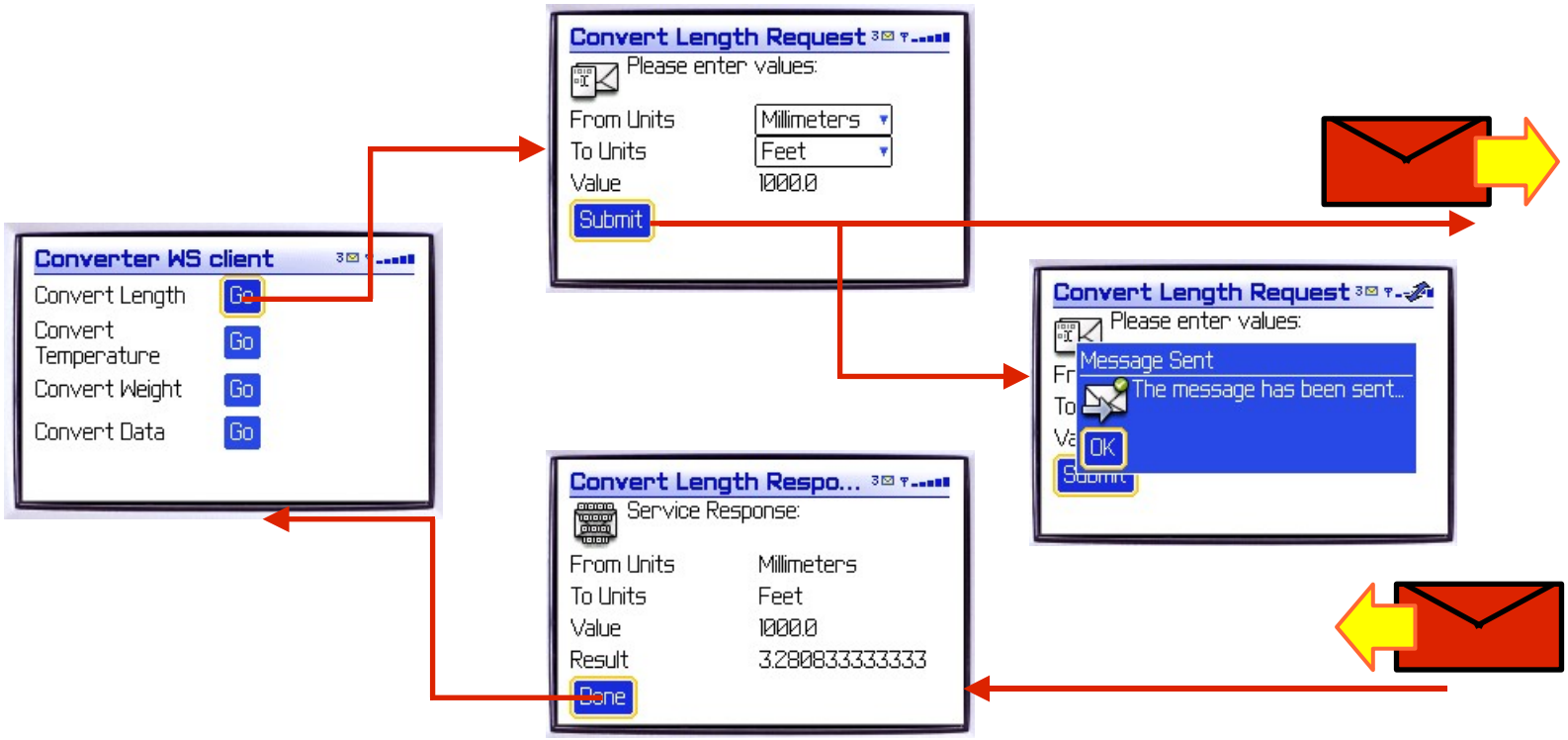
WS Client Applications: Selected Patterns

Designing Wireless Friendly WS

Demo

Form Set Pattern

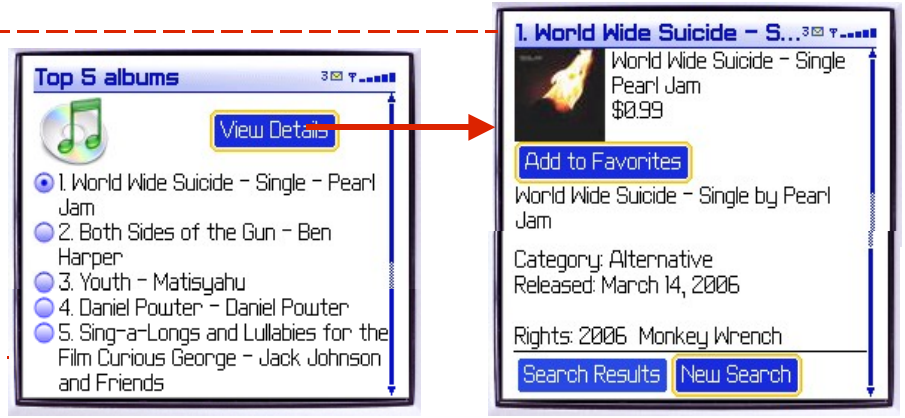
- Used for synchronous Web Services
- The main screen lists supported operations
- One screen/form per request, one per response



Drill Down Screens

- Could be used for nested complex types in requests and responses
- Could be used for arrays of complex types or arrays of elements of “any” type
- The level of nesting could be reduced by “flattening” the complex type

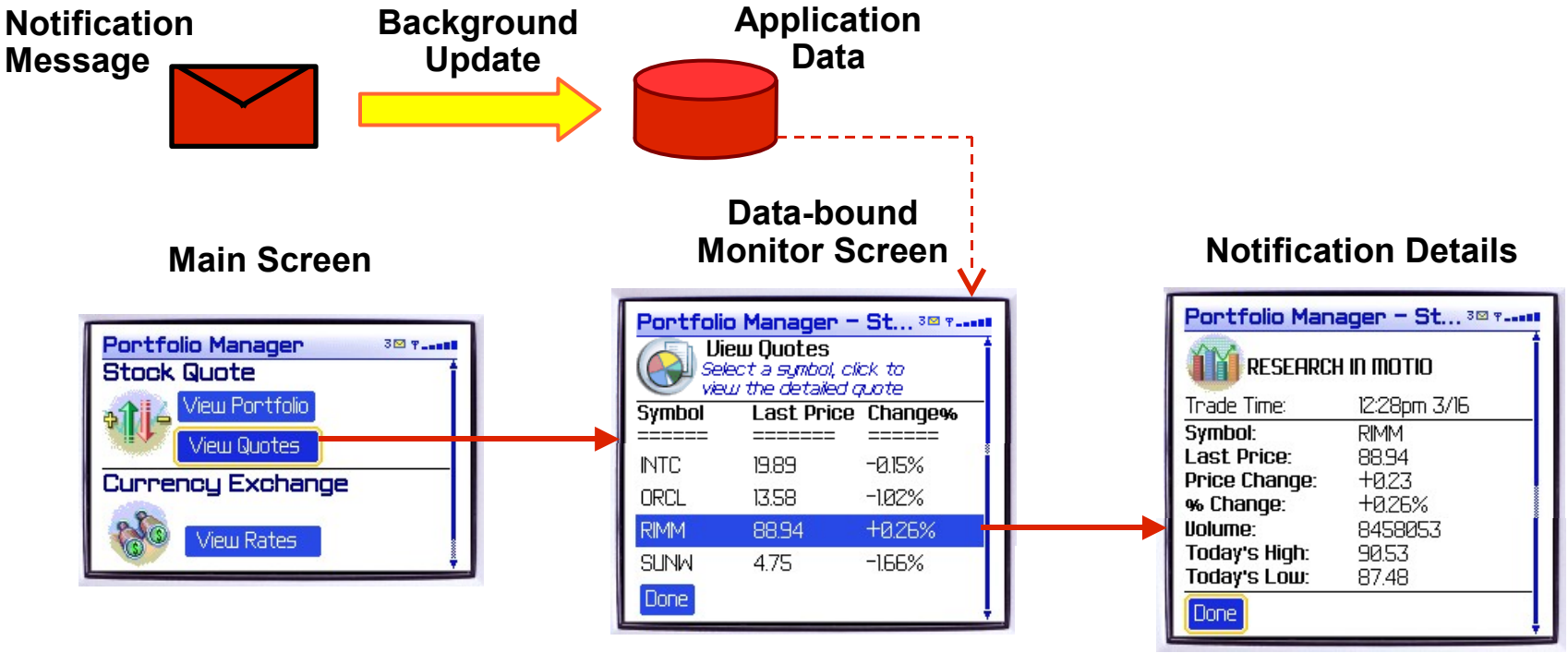
```
<complexType name="Music">
    <sequence>
        <element name="title" type="xsd:string"/>
        <element name="link" type="xsd:string"/>
         …
    </sequence>
</complexType>
<complexType name="ArrayOfMusic">
    <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="item" type="impl:Music"/>
    </sequence>
</complexType>
<complexType name="MusicSearchResponse">
    <sequence>
        <element name="music" type="impl:ArrayOfMusic"/>
       …
    </sequence>
</complexType>
```



This complex type could be flattened

Notification Monitor Pattern

- Used for asynchronous Web Services
- Notification messages update application data on the background
- Monitor screen displays selected data for notification channels
- The latest data is available when the user opens a monitor screen



Agenda

Challenges of Wireless Access to WS

Java ME Container + Services + Components

WS Client Applications: Selected Patterns

Designing Wireless Friendly WS

Demo

Recommended WSDL Types

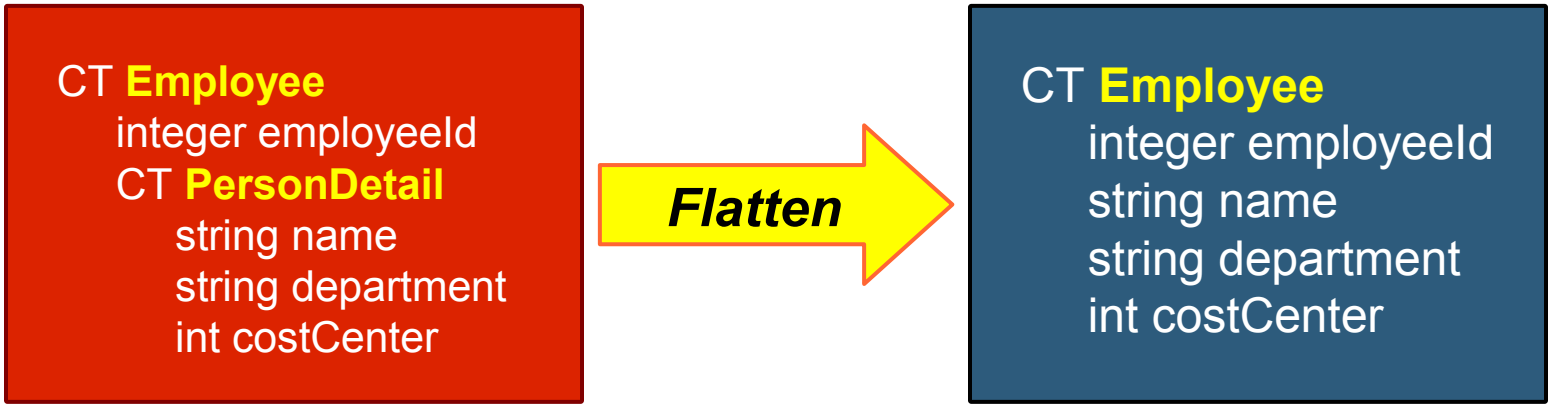
- Use XMLSchema anySimpleType and common derivations
  - Avoid use of soap encoded types
- Date types are commonly supported
  - xs:Date, xs:time, xs:dateTime
- Integer-based types
  - xs:boolean, xs:float, xs:double, xs:integer, xs:long and derivatives
  - Beware large xs:decimal results
  - Handhelds may have restrictions on storing large integers
- xs:string

Derived Types Can Be Easily Mapped to These Base Types

Efficient Complex Types: Structure

Flatten unnecessary containments

- Deep nesting or unnecessary nesting is discouraged
- Weigh tradeoffs of reusability against processing efficiencies

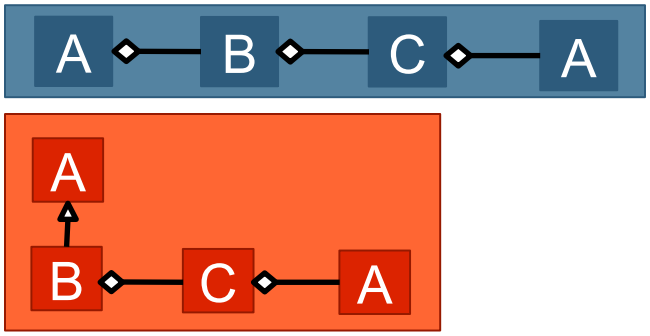


Name complex type fields judiciously

- Element names are carried in the SOAP
- Binary encoding mitigates this effect

Efficient Complex Types: Recursive Declarations

- Resource intensive processing
  - Recursive algorithms incur memory and execution time overhead
- Neither template nor XML parsing approach make treatment of recursion straight-forward
  - Try to avoid using recursive types where possible
- Recognize forms of recursion
  - Containment reference
  - Indirect reference by extension

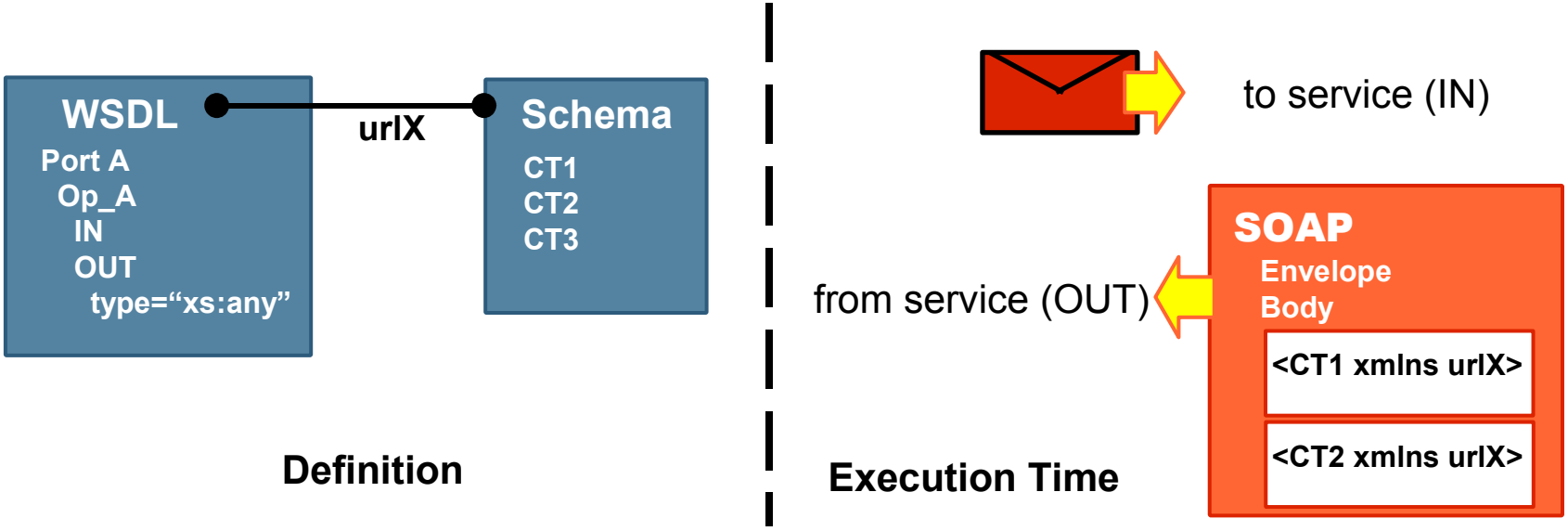


Loosely Defined Types

- Types are not known at design time!
  - Has an impact on how the mobile client is designed
- xs:any can be used to pass arbitrary XML
  - Useful for making an interface very extensible
  - Extensibility comes at the cost of logical complexity on the client end
- xs:anyType also falls into this category
- xs:extension to deliver super type is preferable
  - Sub types are at least declared in the service
- xs:choice is another alternative

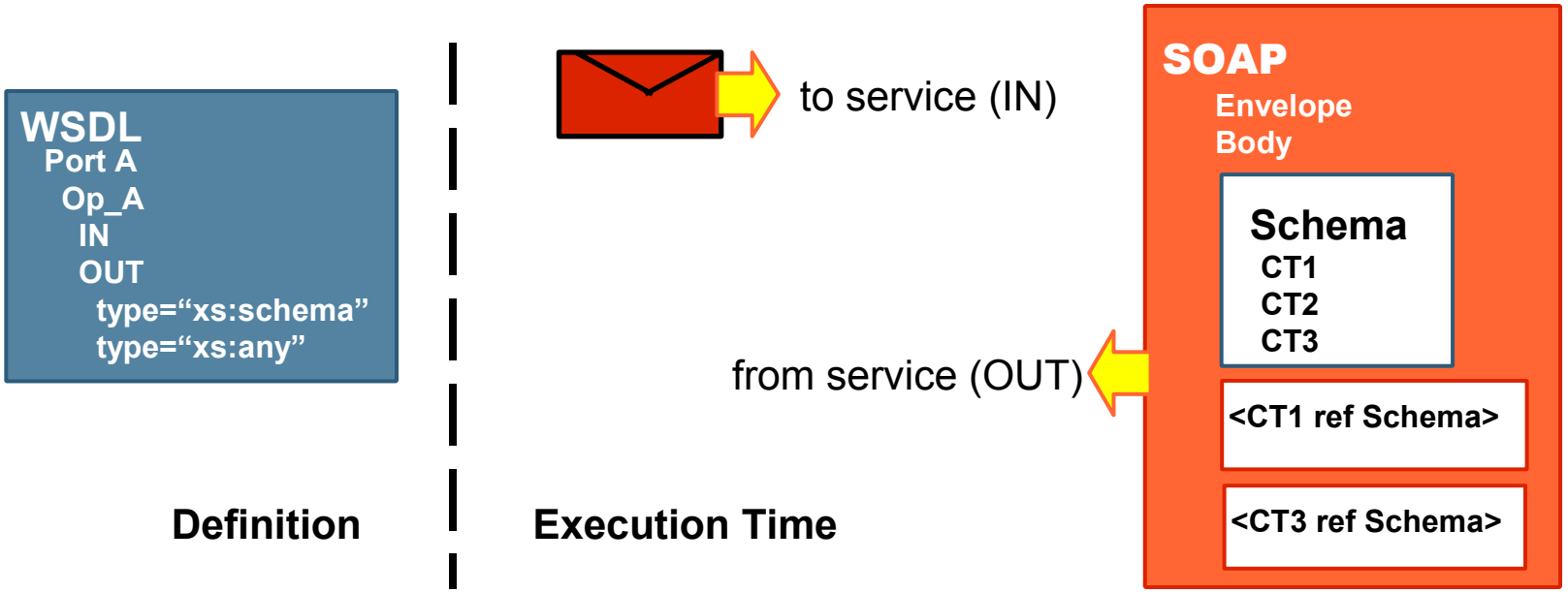
Loosely Defined Types: Raw XML and Declared Types

- Types transmitted are defined elsewhere in the schema or WSDL
- Easier to handle since the client application has the type definitions available



Loosely Defined Types: Raw XML and Schema

- Requires custom processing and parsing
- Types and content/structure is not known until message delivery



Choice of Binding

- When using SOAP bindings there are several choices of style/use
  - rpc/literal (WSI-compliant)
  - document/literal (WSI-compliant with wrapping)
  - rpc/encoded (not WSI-compliant)
  - document/encoded (not WSI-compliant)
- doc/literal and rpc/literal are encouraged
  - rpc/encoded is, however, commonly encountered

Other Advanced Features to Avoid

- Operation overloading
  - Depending on choice of binding style and use operations may be indistinguishable
- Multi-dimensional arrays
- Soap encoded types
  - Stick to schema xsd types

Service Aggregation: Handling Large Datasets

- Some datasets are easily separated: Arrays
- Others are less intuitive to separate: Binary data
- Large datasets are difficult to transmit
  - Consider using aggregator to partition large datasets into chunks and maintain a dataset cursor
- Large datasets may pose storage and processing issues when received by device
  - Consider describing the data and allowing the client to selectively obtain records, e.g., key information

Service Aggregation: Endpoint Redirection

- Often used by WS for registration handoff
  - Initial request comes to endpoint predefined by WSDL
  - The response supplies redirected service endpoint
  - Subsequent requests are redirected by the client
- Aggregator service can negotiate handoff on behalf of the device client
- Session must be maintained by aggregator to associate client instance to dynamic endpoint

Aggregator Maintains Single Endpoint, Redirects Web Service Interactions as Required

Service Aggregation: Dynamic Binding

xs:any type handling

- Aggregator Web Service processes XML data and constructs appropriate complex type
- Aggregator exposes structural information of the XML data for early binding

Aggregator Provides Essential Design Time Information and Performs Runtime Transformation

Mobile Proxy: Transforming SOAP

- Message definitions are well known at the client and mobile proxy: WSDL
- Mobile proxy can extend optimized protocols down to the device
  - The proxy executes protocol transformation between optimized mobile format and SOAP
  - The proxy constructs SOAP requests and parses SOAP responses
  - While data payload of the messages is transmitted, structural/tag information is withheld

Mobile Proxy: Shift to Asynchronous Messaging

- Requests from device are queued at proxy and dispatched separately
- Mobile client receives immediate acknowledgment; fire-and-forget model
- Mobile proxy unites asynchronous device messaging with synchronous WS interaction
- Handles service unavailability through resubmission
- Mobile proxy issues a push to deliver the response data
- Out-of-coverage scenarios are covered inherently

Mobile Proxy: Challenges

- Correlation of request/response data
- Cracking the synchronous programming mindset (browser form-based approach)
- Handling advanced WSDL features during transformations
- Extending security protocols to the device
  - Avoiding the security gap at proxy
  - NTLM, Basic authentication over https are popular solutions
  - Mobile proxy must hold connections, mediate secure connection

Notifications: Where Are Async Web Services?

- Standardized async web services are still very much in their infancy
- Widespread adoption of standards by toolkits is yet to materialize
  - It's difficult to create these types of web services as a result
- Most existing public web services define their interface as request/response
- On the other hand some push-based wireless solutions are extremely mature (e.g., email)

Strong Motivation to Combine Advantages of Web Services, Push Capabilities of Wireless Networks, to Produce Compelling Mobile Apps

Notifications: Benefits

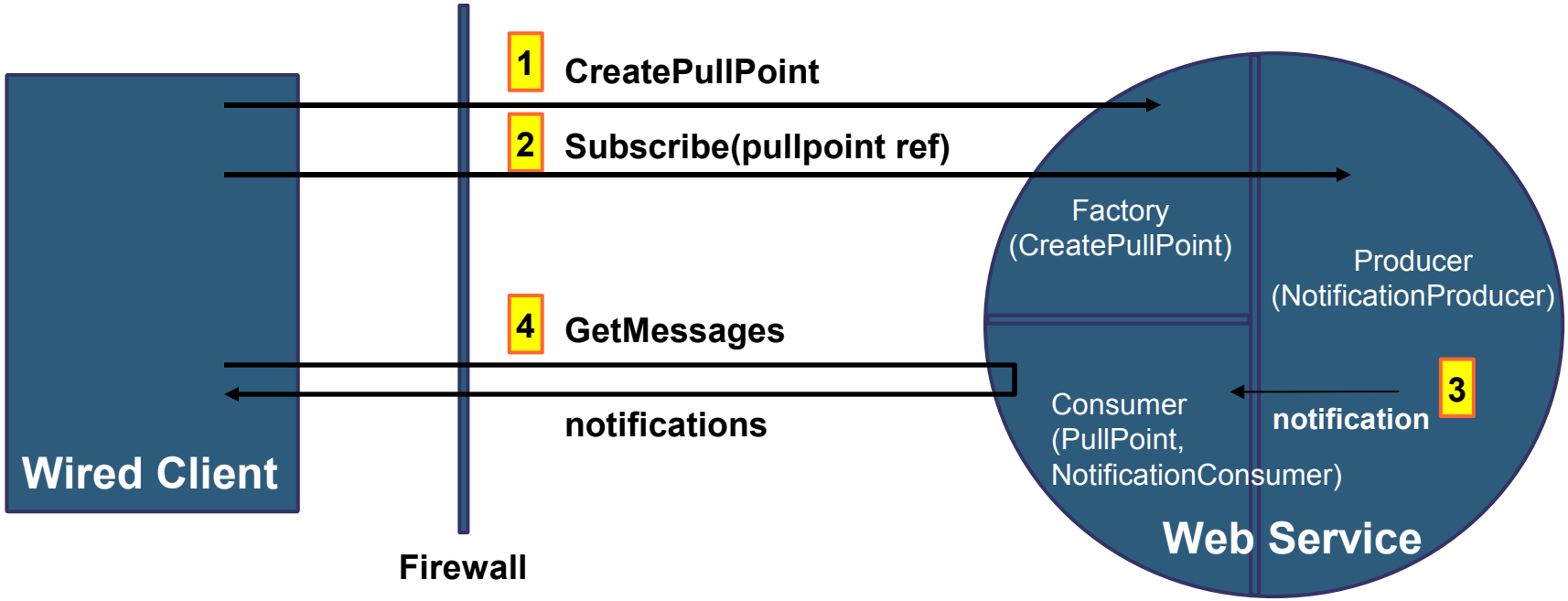
- Better use of wireless network/device resources
  - Notification pattern (single subscribe and multiple push) results in less network traffic
  - Less message processing results in less demand on battery
- A better model where unpredictable connectivity is concerned
  - New data is queued and sent when possible
- Better usability model
  - User doesn't wait for responses (zero latency)
  - Device view of data is always up to date

Notifications: Common Models

- Prevalent standards
  - WS-EVENTING (WSE)
  - WS-NOTIFICATION (WSN)
- WSE:EventSource or WSN:NotificationProducer expose the notification object
  - WS-NOTIFICATION has far more rigor around this matter through Topics
- Application passes a Filter on the subscribe action
  - Indicates the parameters to decide a notification
  - Must evaluate to a true/false condition to trigger a notification
- Web service monitors changing service state
  - Calculates changing criteria across known filters or “Situations”
  - Delivers a notification when criteria matches

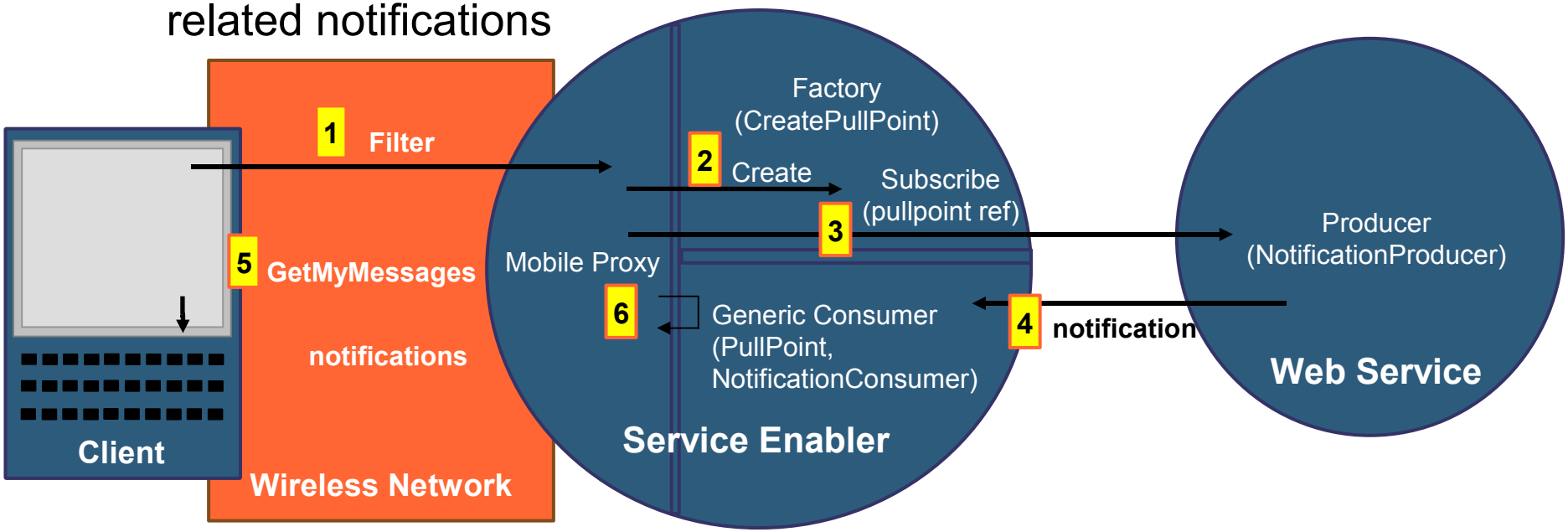
Notifications: Firewall Impedes Push

- WS-NOTIFICATION defines a pullpoint
  - Can be useful when there are firewalls to overcome
  - Pullpoint is created by the client through a factory, becomes consumer
  - Asynchronous notification to the pullpoint by producer
  - Synchronous Subscribe and GetMessages pass through firewall



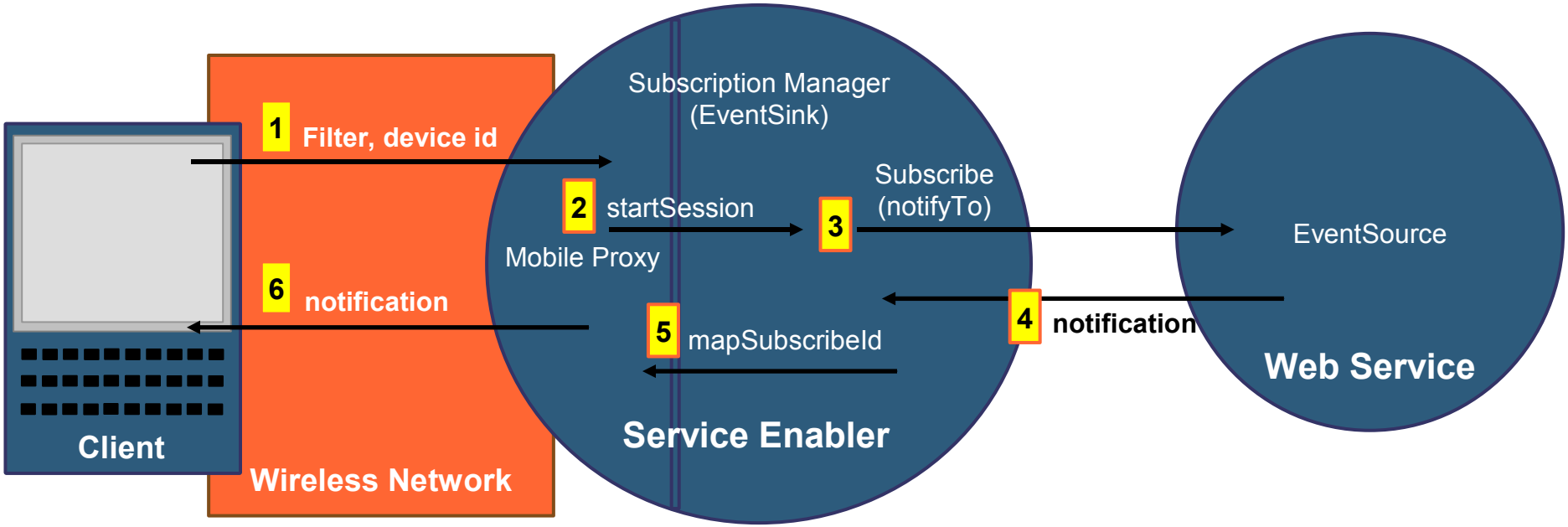
Notifications: Wireless Solution Impedes Push

- Pullpoint concept could be extended to wireless solutions without push capability
- Service Enabler can manage subscription, aggregate consumer and pullpoint
- Mediator tracks subscription and periodically prods pullpoint for related notifications



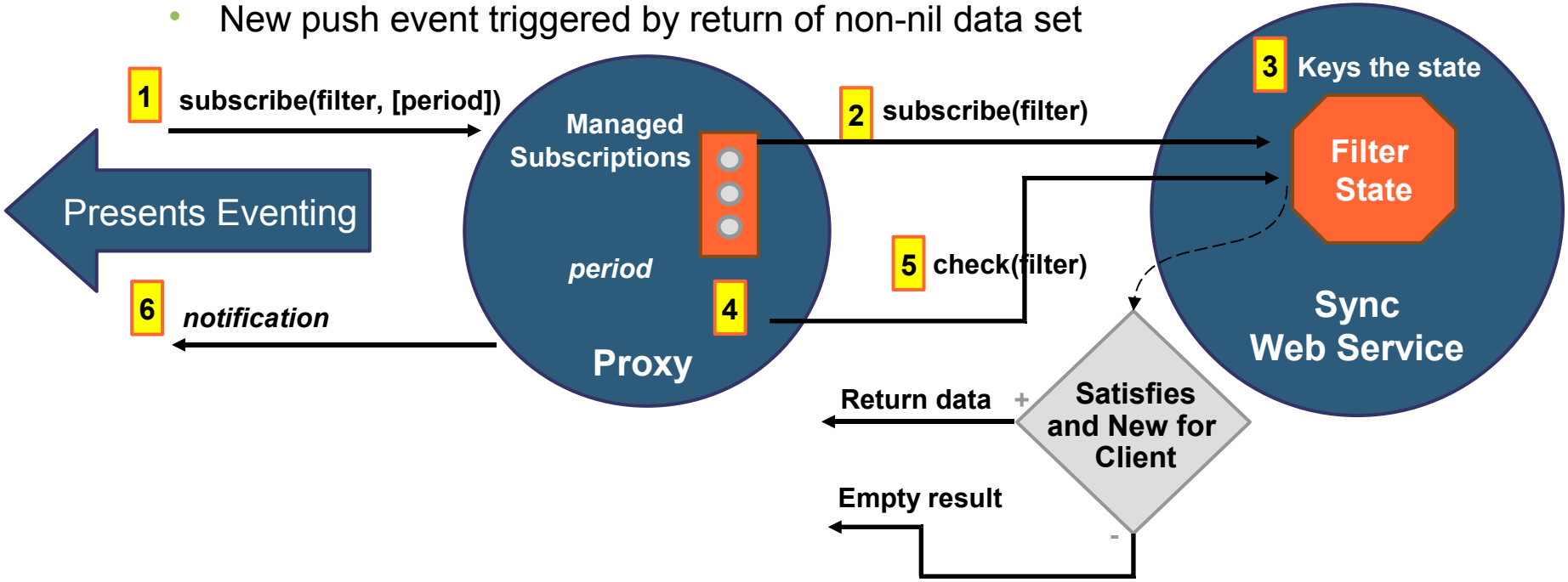
Notifications: Push-Based Wireless Solutions

- Promotes push data directly to the device
- Mobile proxy is useful to simplify wireless messaging
  - Proxy assumes the knowledge of the WS notification approach
  - Device view is just a conventional request (subscribe) and push (notifications) responses



Notifications: Proxy for Synchronous WS

- Simplicity of synchronous web services
  - Can be created with standard approach, existing toolkits
- Proxy service nearly stateless, acts like a simple “pass-through” for eventing
  - Implies better scalability
  - New push event triggered by return of non-nil data set



DEMO

Tying It All Together…

More Information:

- BlackBerry solutions for Web Services:
  http://www.blackberry.com/developers/downloads/studio/index.shtml
- Wireless Component Architecture (WCA):
  http://developers.sun.com/learning/javaoneonline/2005/mobility/TS-7888.pdf
- Service Connection API for Java ME (JSR 279)/XML API for Java ME (JSR 280):
  http://www.jcp.org/en/jsr/detail?id=279 (…?id=280)
- WS-Eventing:
  http://schemas.xmlsoap.org/ws/2004/08/eventing/
- WS-Notification:
  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

Q&A

Best Practices for Building Optimized Wireless Solutions for Web Services

Michael Shenfield

Director, Standards Architecture
RIM

Bryan Goring

Architect, Advanced Technologies
RIM

TS-1293