# Web Services to Go: Mobile Access to Web Services With JSR-279 and JSR-280


Pia Niemelä

Nokia Corporation

http://jcp.org/en/jsr/detail?id=279

Jean-Yves Bitterlich

Sun Microsystems

http://jcp.org/en/jsr/detail?id=280


Session TS-5188

# Goal

New tools for creating a rich mobile Web Services client-side ecosystem

Overview of XML, Web services and SOA and introduction to Java™ Specification Request (JSR) 279 and 280

# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plugins?
- Interaction patterns and basic building blocks of JSR 279 web services
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plugins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

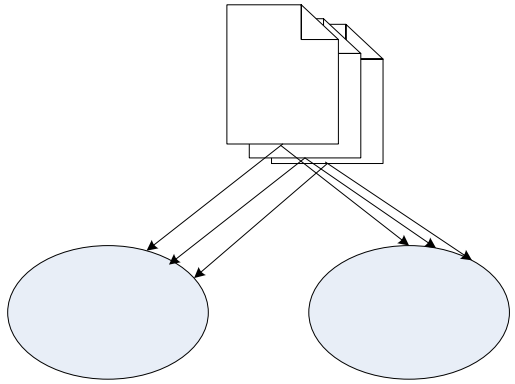
# What Is Service-Oriented Architecture (SOA)?

- SOA is an architectural style whose goal is to achieve loose coupling among interacting entities; two entities involved in service transactions are:
  - A service consumer and a service provider
    - Consumers and providers may build larger networks
  - The third entity usually mentioned is a service directory/repository which helps in publishing and finding the desired services
- A service is a unit of work done by a service provider to achieve the desired end results for a service consumer
  - Services offer methods for extracting or modifying data

# What Is Service-Oriented Architecture (SOA)?

## Conventions and rules to improve application interoperability

| Resource centric service abstraction | Universal interfaces | Robustness |
|---|---|---|
| Service contract (WSDL) abstracts the service, other service logic is hidden; each resource has a URI | **All** services provide same simple interface<br>**No** application specific interfaces | **Loosely coupled** services<br>• The same message works with several services<br>• The same service works with several messages |



# What Is a Web Service?

- An application component that can be called remotely using standard Internet Protocols such as HTTP and XML
- RPC uses mostly HTTP, whereas messaging may also use other mechanisms; for example:
  - FTP, SMTP, Java Message Service (JMS) API, IIOP, even SMS
- A unit of code that can be activated using service requests
- 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

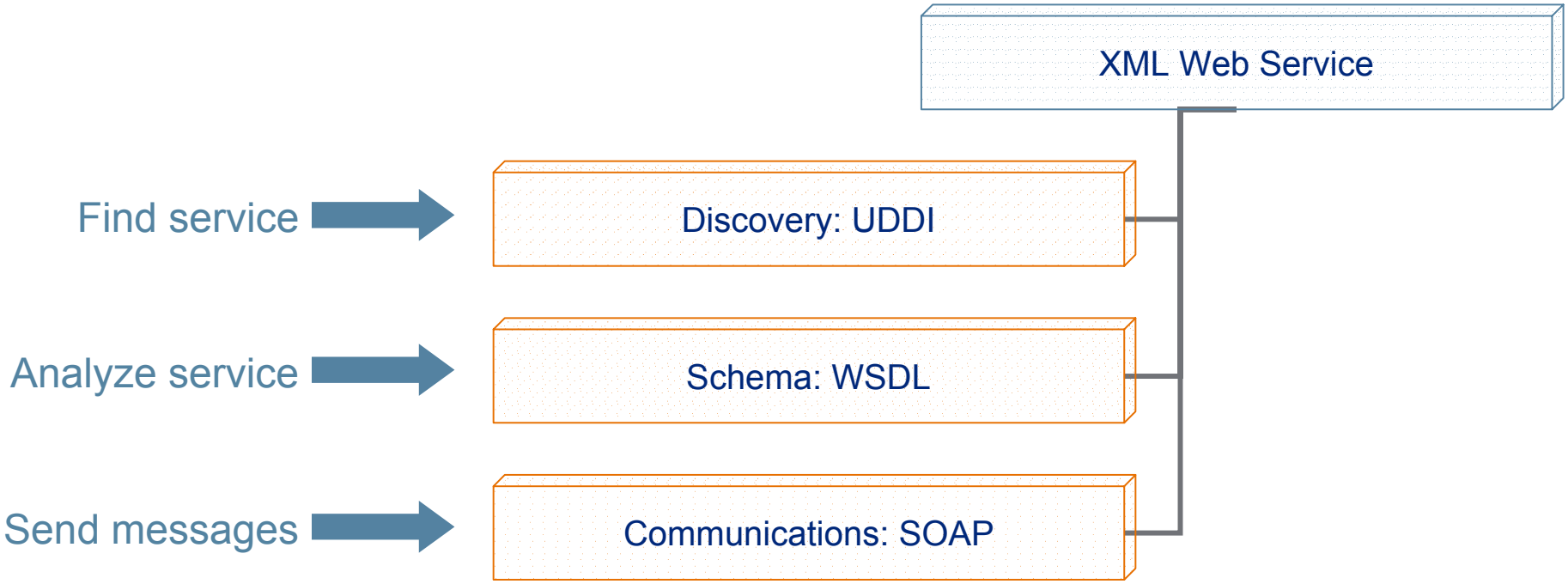
# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plugins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

# “Classic” Web Services

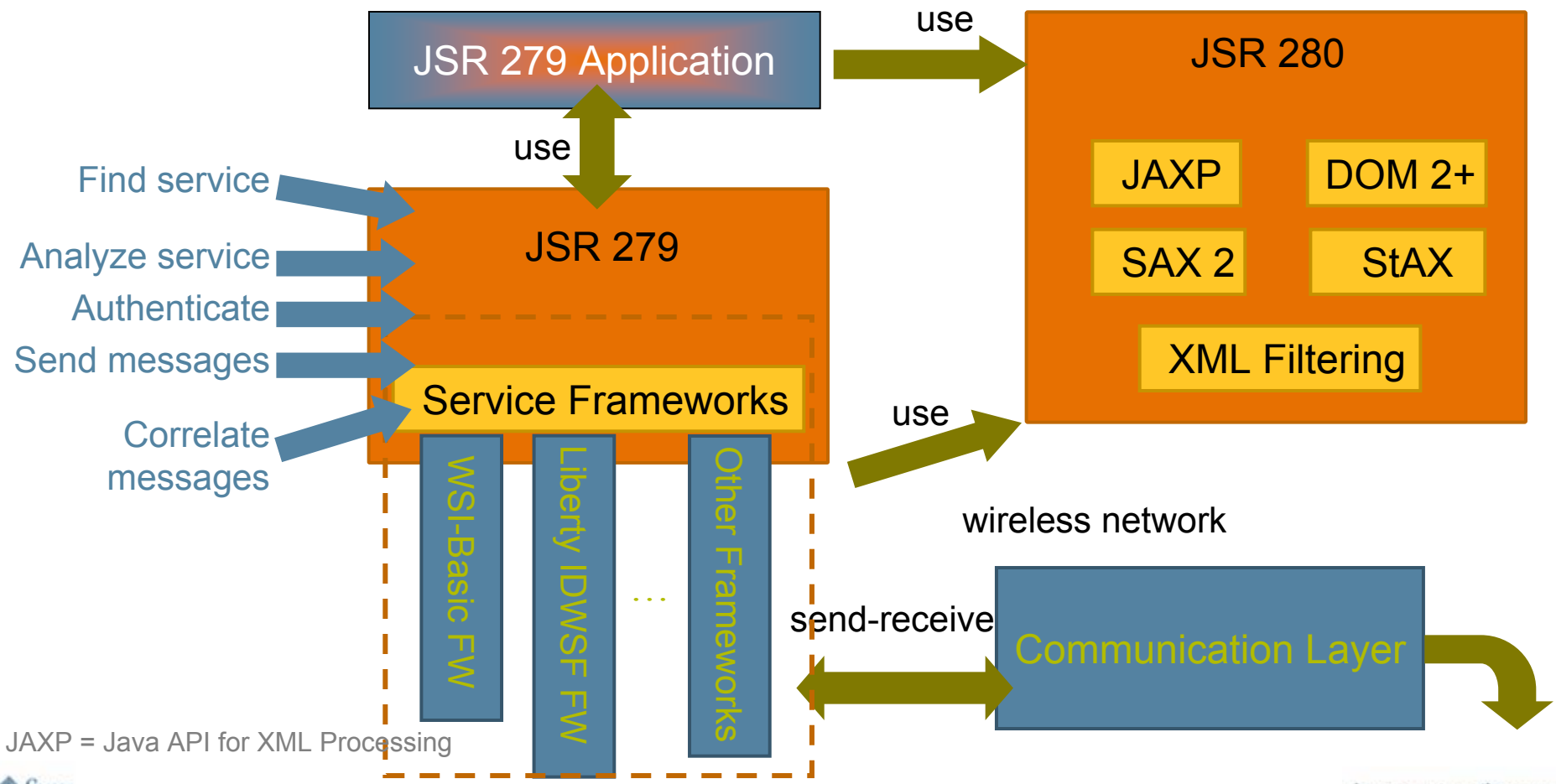
XML Web Service is a software service exposed on the Web through SOAP protocol, described with a WSDL file and registered in UDDI registry. UDDI, WSDL, and SOAP are all XML based protocols.

JSR 172 provides API for SOAP web services



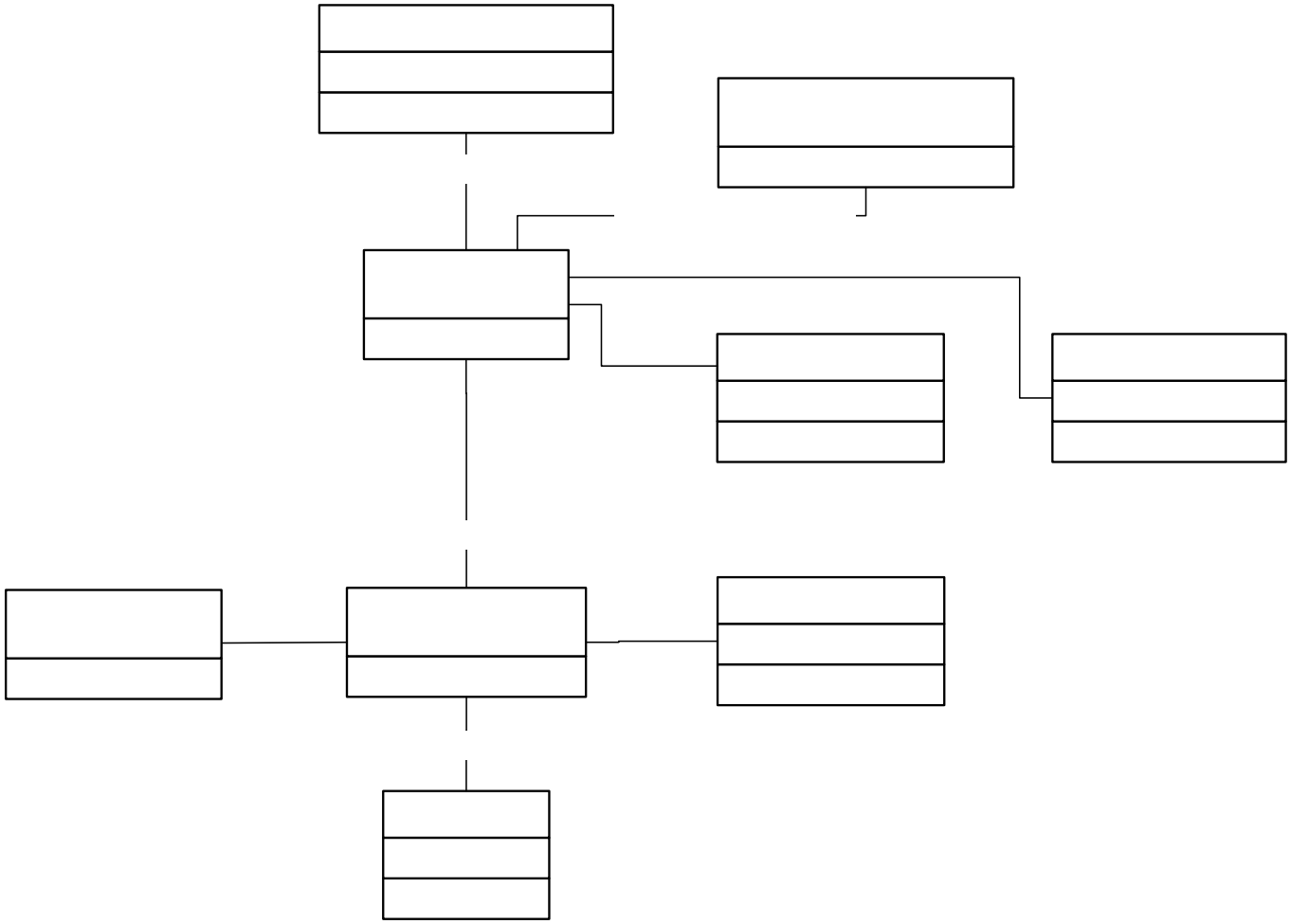
# “Modern” JSR 279 Web Services

- “Seamless SSO" decreasing the app developer’s work
- Ability to use the same API to access services that use different frameworks for auth/SSO and service discovery (e.g., Liberty ID-WSF and UPnP)




JAXP = Java API for XML Processing

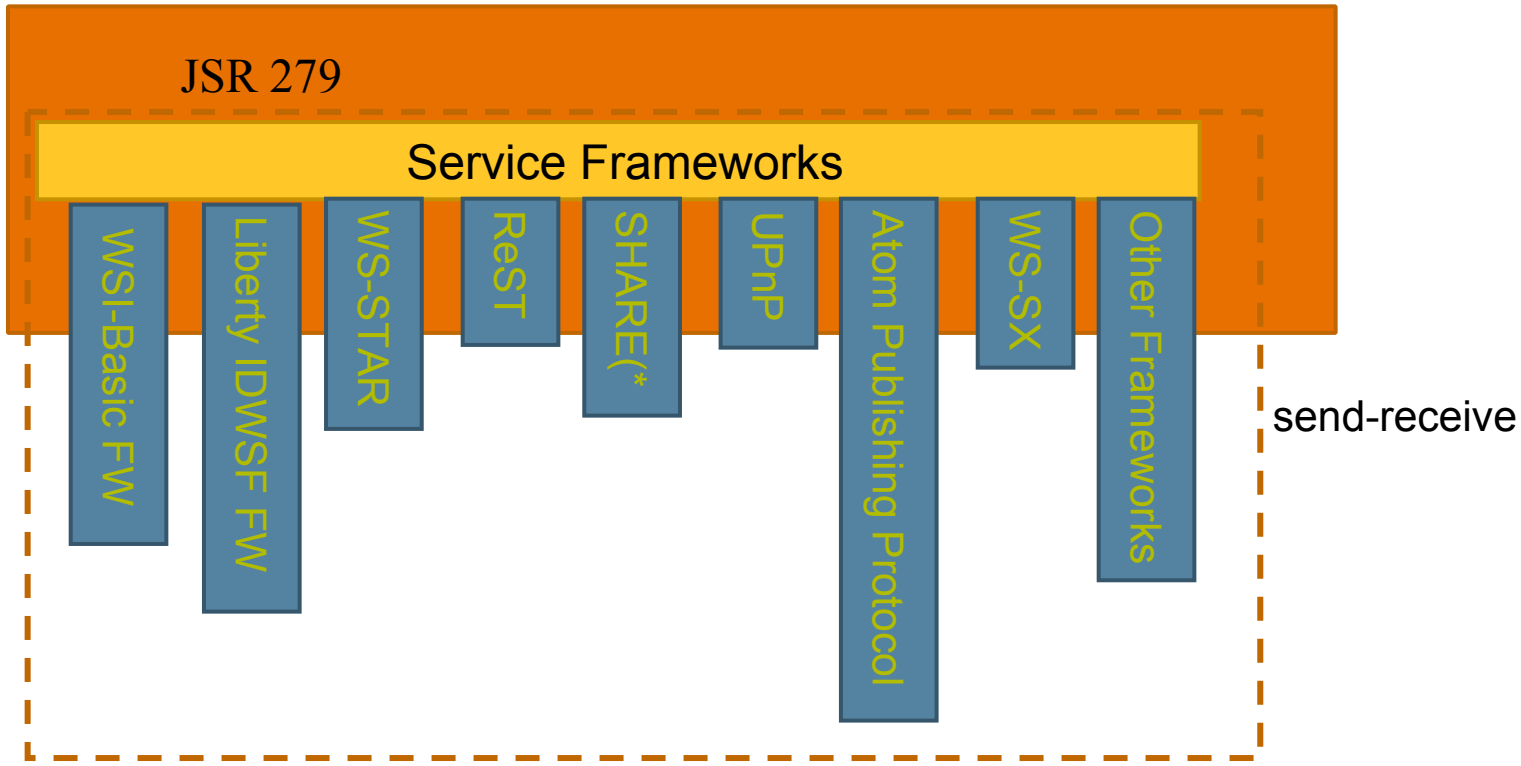
# JSR 279 Class Diagram



# Agenda

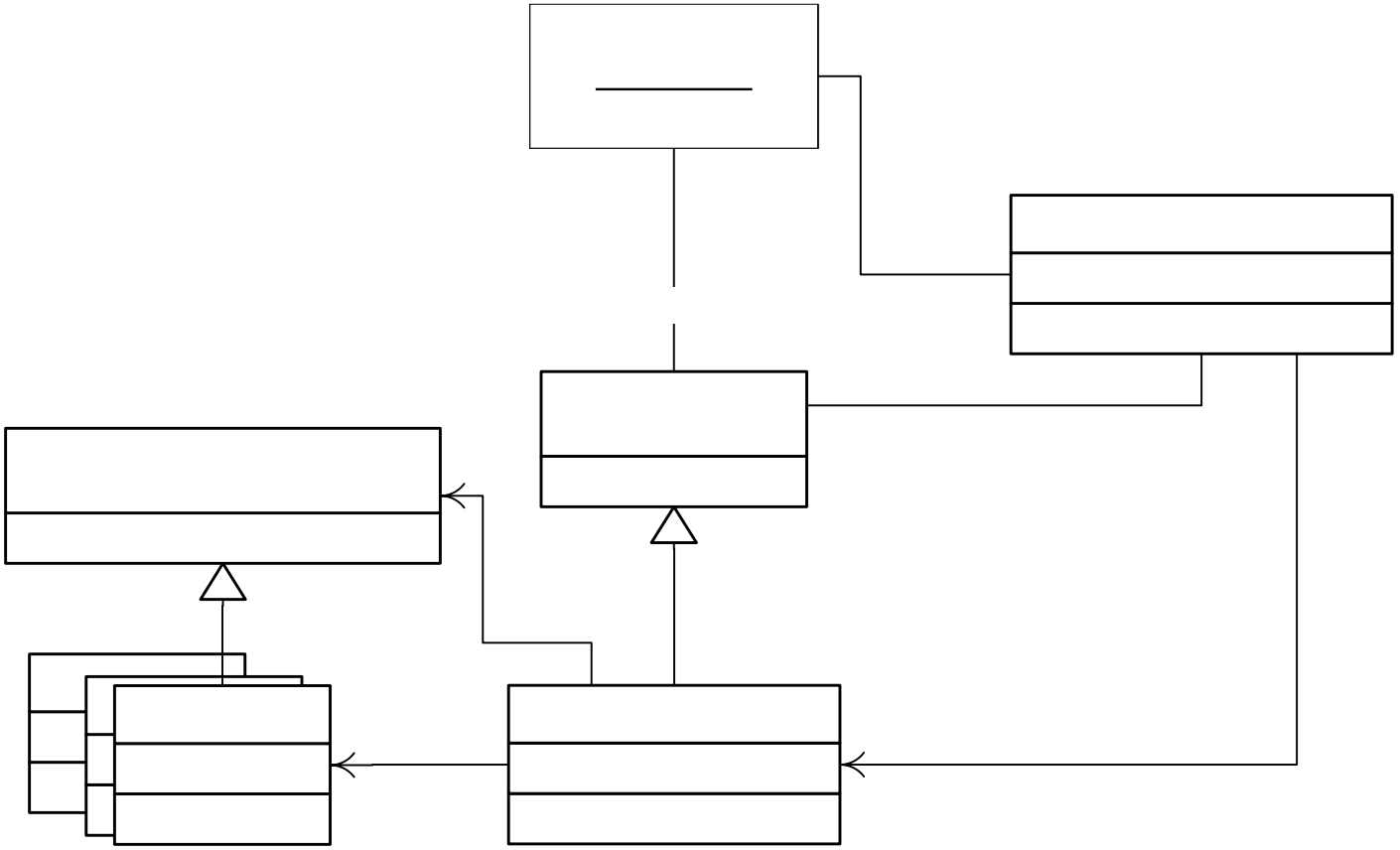
- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plugins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

# Framework Profile Plug-In Examples




New framework profile plugins can be written and installed. The framework ID must be unique.
*)SHARE" or "Simple HTTP API with RPC and encoded data". (http://asynchronous.org/blog/archives/2005/03/index.html)

# Plugging in a New Framework

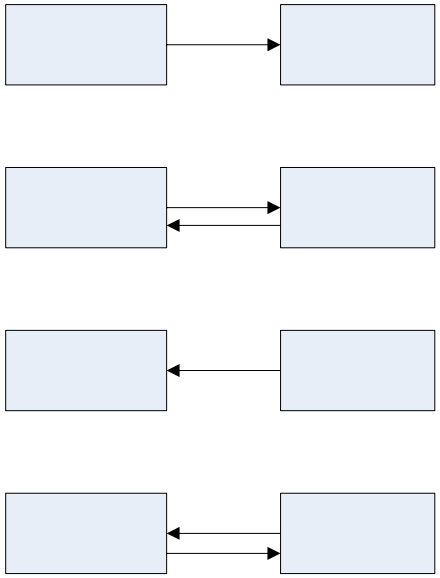


The ServiceManager uses an implementation-specific mechanism to locate framework implementations i.e., providers.

# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plugins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

# Interaction Patterns

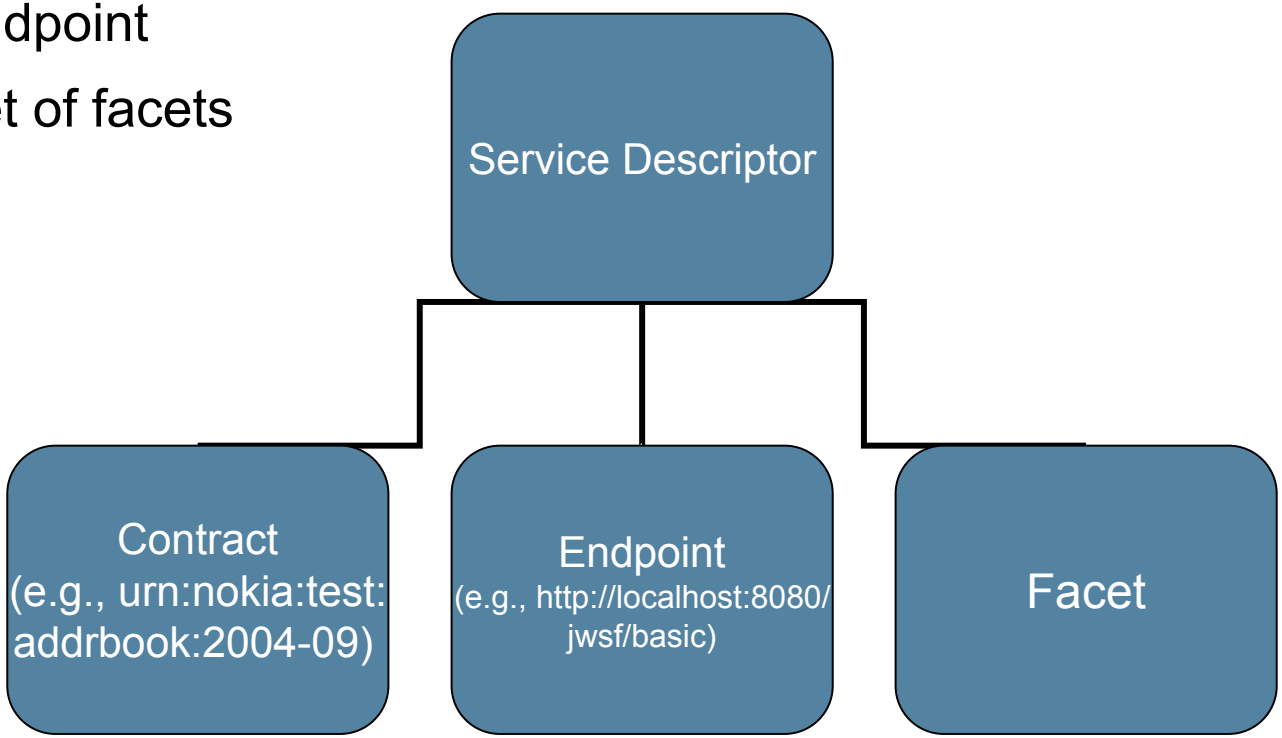


| Pattern | JSR 279 | Mode |
|---|---|---|
| One-way | send() | async |
| Request-response | sendReceive() | sync |
| Notification | receive(),<br>setMessageListener() – handle() notification | receive() is synchronous,<br>MessageListener is asynchronous |
| Notification-response | No direct support | async |

# Service Descriptor

Service descriptor specifies the desired service; it has three parts:

1. Contract
2. Endpoint
3. Set of facets

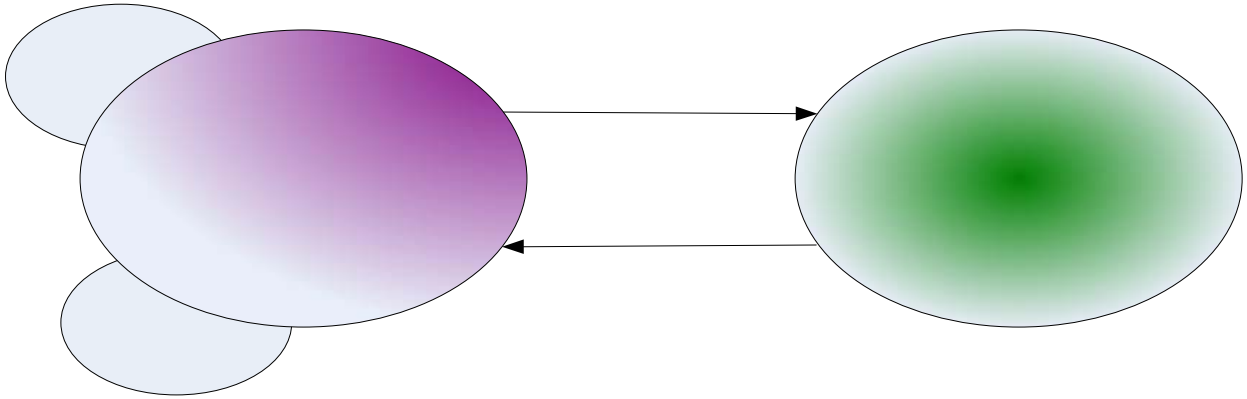


# Service Descriptor

- Contract
  - An abstract way to refer to a service, “name” of the service
    - Contract is defined by specifications implemented into a JSR 279 framework plug-in
  - In WSDL terms there is no interoperable equivalent in standard WSDL
  - For Liberty services, the contract is likely to be the namespace within which all of the XML elements of the messages are placed
- Endpoint
  - A concrete way to define a service, a network endpoint
  - WSDL soap:address location equals the endpoint
- Set of facets
  - WSDL content mapped as facets
    - Facet names are presented in XPath notations
  - Metadata, policy

# Service Descriptor vs. WSDL

- The **service should be describable** in WSDL 1.0
  - ReST support is defined in WSDL 2.0
    - http://www.w3.org/TR/2007/WD-wsdl20-primer-20070326/#reservationDetails_HTTP



A ServiceDescriptor can be generated from WSDL with the constructor:

```
ServiceDescriptor desc = new ServiceDescriptor(WSDL);
```

WSDL can be generated from a ServiceDescriptor:

```
String wsdl = serviceDescriptor.getWSDL();
```

# ServiceDescriptor Generation: Facets

| Facet Name (XPath/URN) | Facet value |
|---|---|
| | Specifies an address for the port |
| | Specifies the sequence of operations i.e. interaction pattern |
| price:0 | QoS definition, free of charge service desired |
| accuracy:high | QoS definition, high accurate service desired |

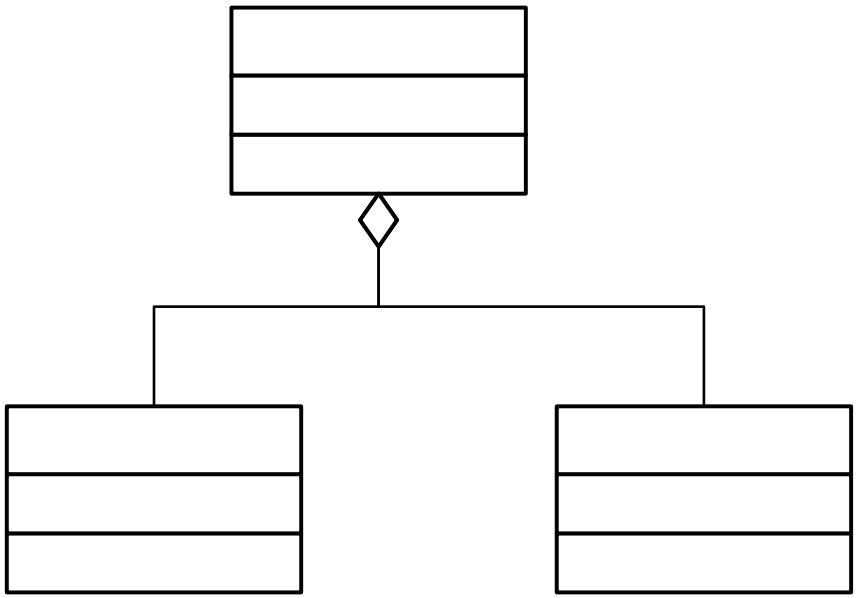
To help the app developer in constructing the ServiceDescriptor a following method can be used:

```
String[] ServiceDescriptor.getFacetValueOptions(frameworkId, facetName);
```

If there is a closed set of options, the method returns them; otherwise null is returned

# Message Object Is the Actual Content

- The Element[] array is provided because the soap body of the message may have multiple elements without necessarily having a single top-level element
- Properties contain, for example, correlation and authentication headers
- Properties can also override facets of ServiceDescriptor



# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plugins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- **Adding identity to Web services (e.g., Liberty)**
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

# Authentication

- The most basic operation in a high-value relationship with customers, employees, citizens or business partners
- Has to be done with great care to proactively fight fraud and identity theft
  - Secure solutions are essential
  - User consent must be supported
- Common mechanisms to handle authentication information is required
  - Technically, to enable interoperability and seamless user experiences
  - Legally, to enable a business relationship between different entities in a distributed environment

# Authentication

- Steps to use authentication with JSR 279
  1. Populate a AuthenticationInfo object
  2. Associate the AuthenticationInfo and the identity provider
  3. Associate the service and the identity provider
  - The framework takes care of the rest
- The authenticated service connection can be reused while it is not expired

# AuthenticationInfo

Current authentication facets:

public static final String FCT_USERNAME=“ USERNAME“;

public static final String FCT_PASSWORD=“ PASSWORD“;

public static final String FCT_X509= "X509_CERTIFICATE“;

public static final String FCT_JAAS="JAAS_SUBJECT“;

public static final String FCT_IMEI=“IMEI”;

# Authentication: Liberty Example

```
// create an AuthenticationInfo object
AuthenticationInfo authInfo = new AuthenticationInfo();
authInfo.setFacet(AuthenticationInfo.FCT_USERNAME, "uname");
authInfo.setFacet(AuthenticationInfo.FCT_PASSWORD, "pwd");
// set up the Identity Provider
ServiceDescriptor idp = new ServiceDescriptor(null, null,
    "http://localhost:8080/jwsf/as");
// set up associations between IdP and authinfo, and service and idp
serviceManager.associateAuthenticationInfo(idp, authInfo);
// create the descriptor giving the contract
ServiceDescriptor pattern = new ServiceDescriptor(null,
    "urn:nokia:test:addrbook:2004-09", null);
serviceManager.associateIdentityProvider(pattern, idp);
```

# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plug-ins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- **XML tooling**
  - **JSR 280 introduced**
  - **Three programming models StAX, SAX, and DOM compared**
- Current XML standardization landscape
- DEMO 279 and 280
- Q&A

# JSR 280

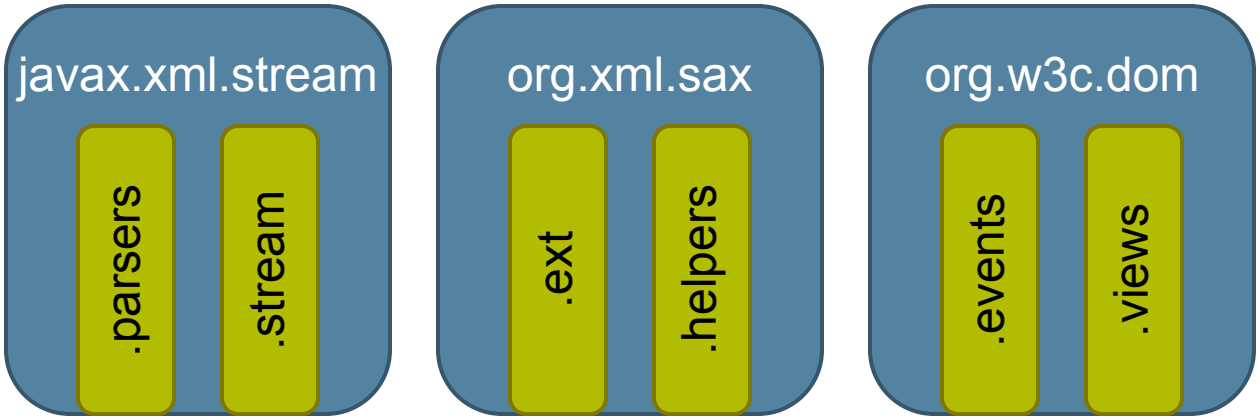
- The starting point for JSR 280 was about:
  - Revising JSR 172, concentrating on XML and leaving Web services to JSR 279
  - Defragmenting XML APIs from the different JSRs
  - Be synced and backward compatible with JSR 172, 173, 226, 279
  - Offering a state of the art and rich XML parsing ecosystem

# XML Use Cases

- Web services
- MMS messages
- Multimedia presentations (SMIL)
- Browsing (WML, xHTML)
- Scalable Vector Graphics (SVG)
- Office applications (many document mark-ups, some proprietary)
- P2P and proximity protocols like UPnP and JXTA™ technology use SOAP and XML mark-ups
- Small, simple, fit-for-purpose protocols/formats, e.g., RSS and ATOM
- Synchronizing (SyncML)

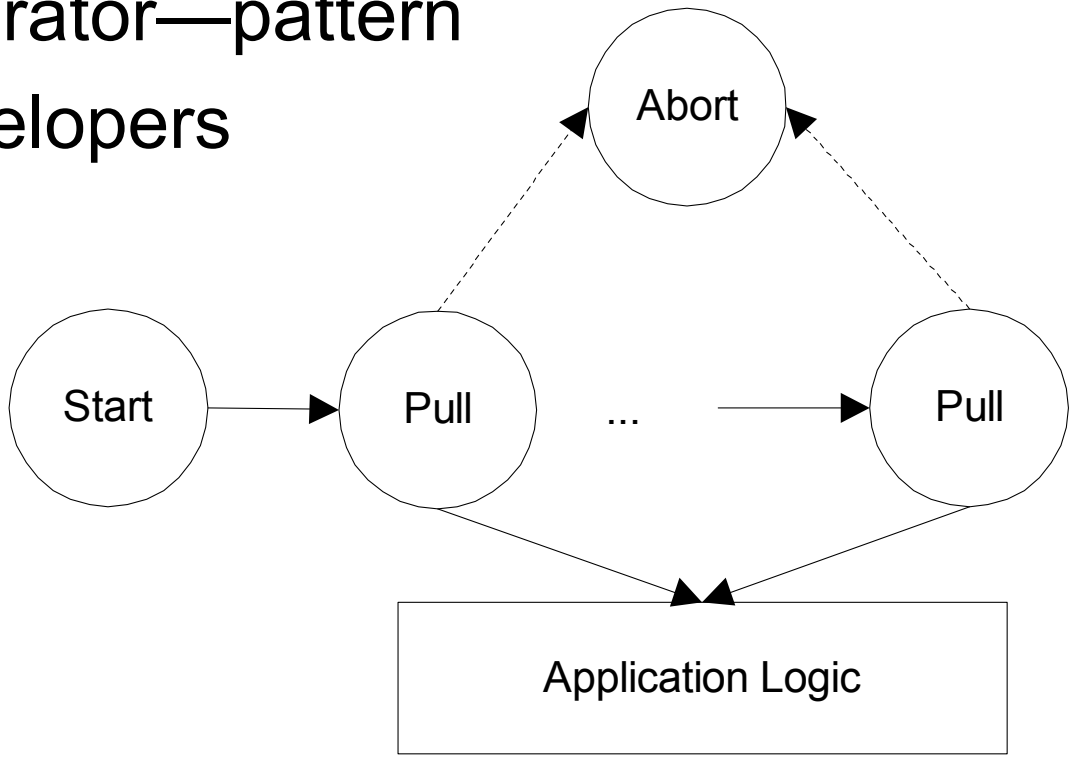
# XML Parser Programming Model

- The following technologies for parsing XML exists
  - Pull parsers (e.g., StAX)
  - Push parsers (e.g., SAX)
  - DOM (document object model) parser
- JSR 280 covers them all:

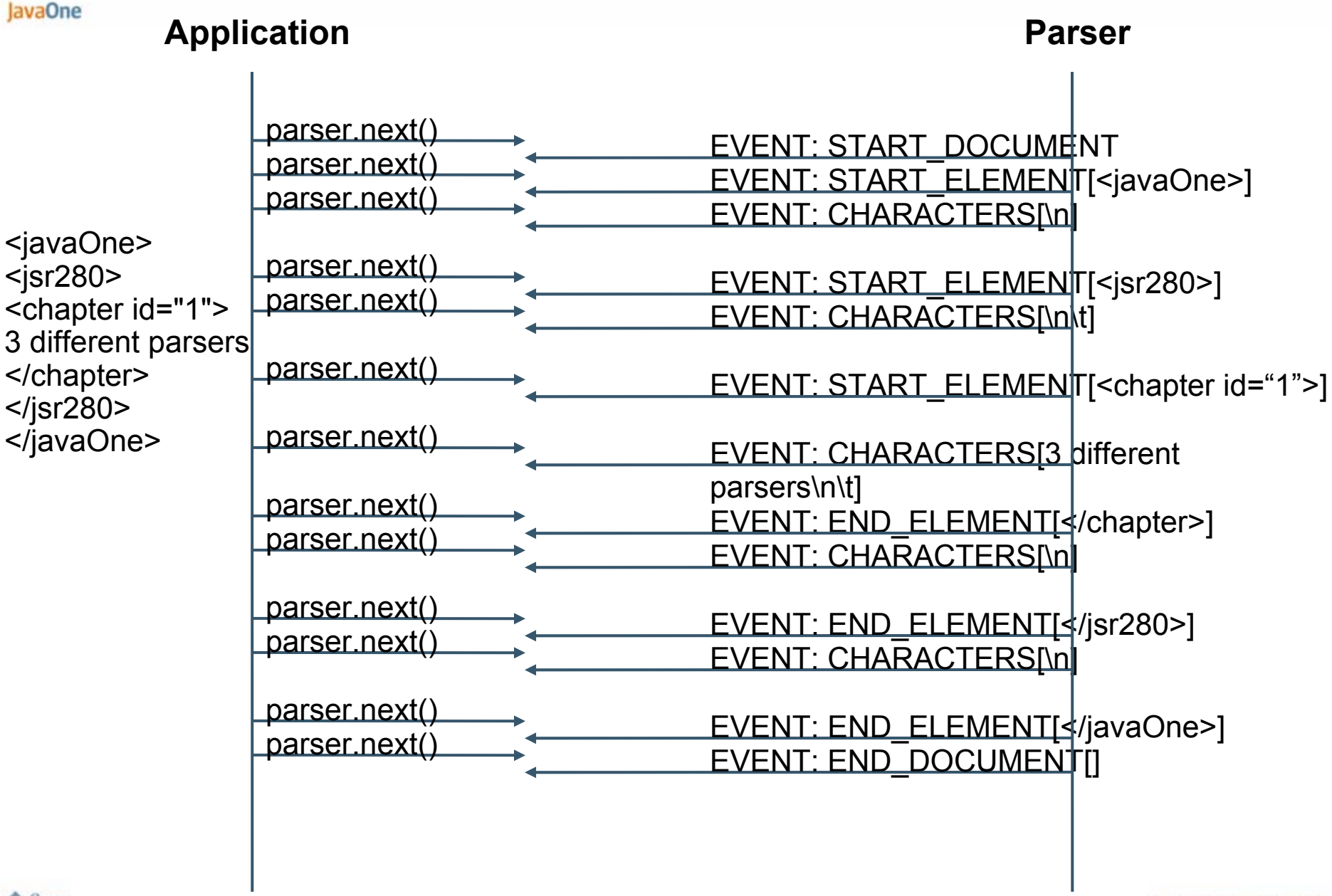


# Pull

- Based on an Iterator—pattern
- Intuitive for developers

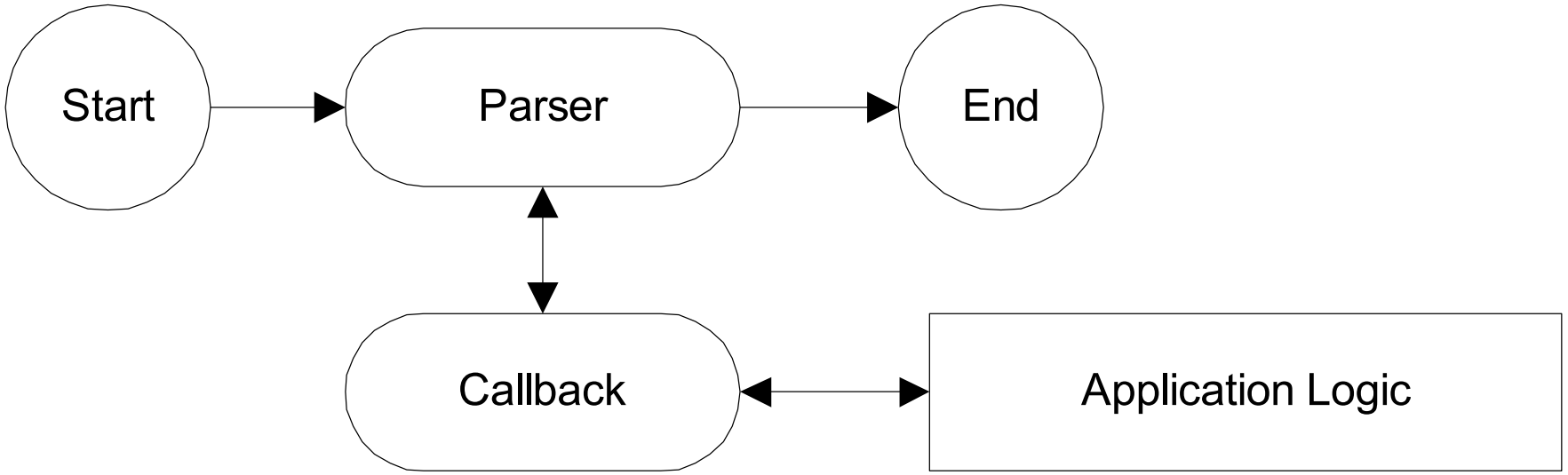



XMLPull parsing process

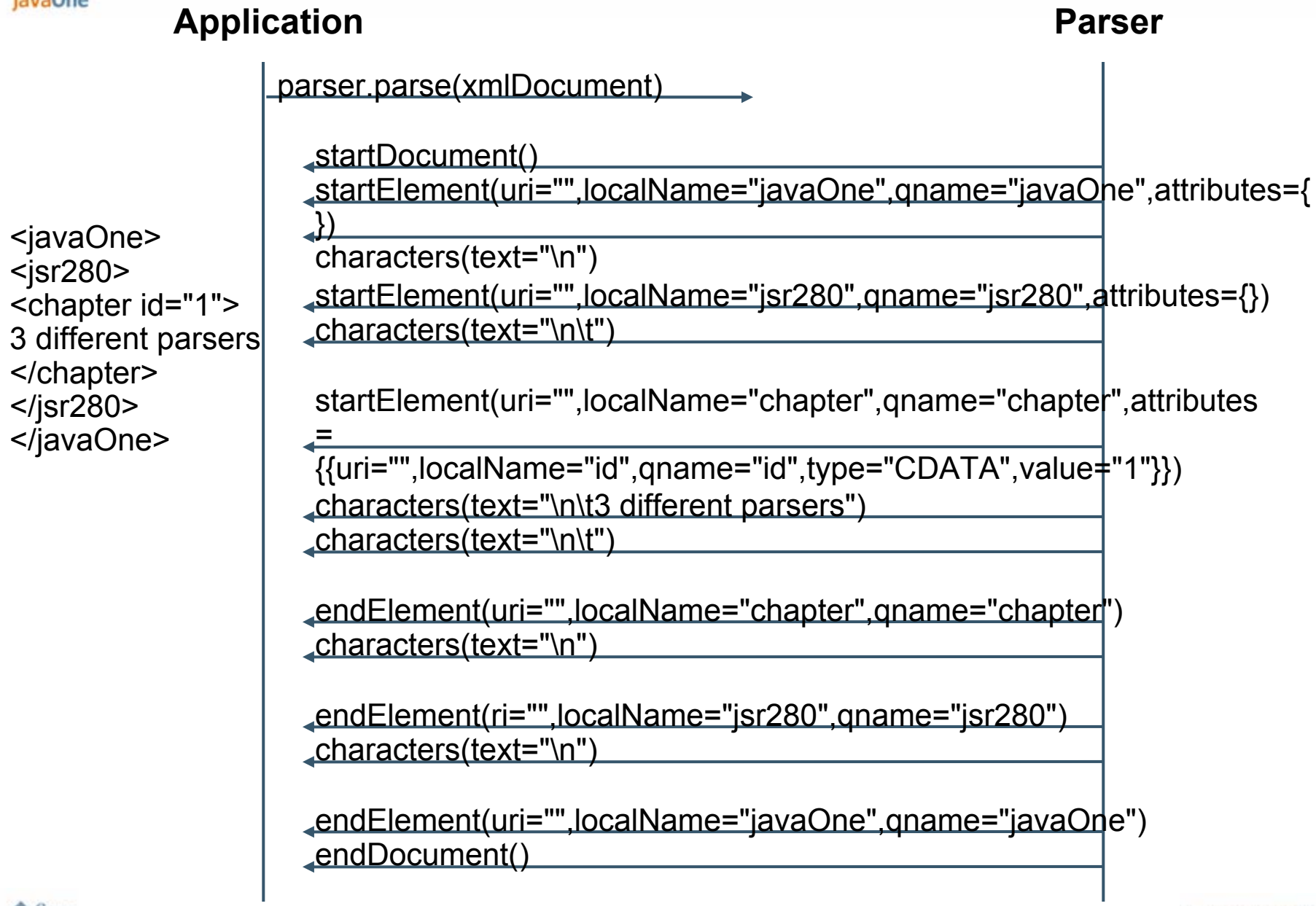

Application
Parser
<javaOne>
<jsr280>
<chapter id="1">
3 different parsers
</chapter>
</jsr280>
</javaOne>
parser.next()
EVENT: START_DOCUMENT
parser.next()
EVENT: START_ELEMENT[<javaOne>]
parser.next()
EVENT: CHARACTERS[\n]
parser.next()
EVENT: START_ELEMENT[<jsr280>]
parser.next()
EVENT: CHARACTERS[\n\t]
parser.next()
EVENT: START_ELEMENT[<chapter id="1">]
parser.next()
EVENT: CHARACTERS[3 different parsers\n\t]
parser.next()
EVENT: END_ELEMENT[</chapter>]
parser.next()
EVENT: CHARACTERS[\n]
parser.next()
EVENT: END_ELEMENT[</jsr280>]
parser.next()
EVENT: CHARACTERS[\n]
parser.next()
EVENT: END_ELEMENT[</javaOne>]
parser.next()
EVENT: END_DOCUMENT[]

# Push

- Based on events
- Parser emits events, which are captured by event handlers

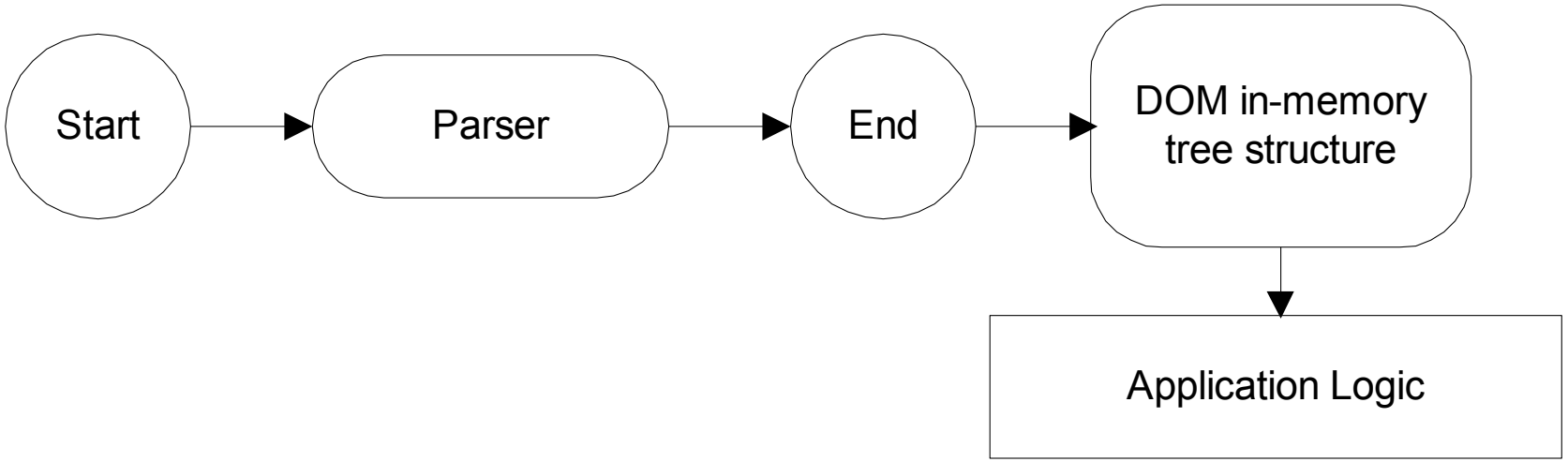



SAX parsing process


Application
Parser
parser.parse(xmlDocument)
startDocument()
startElement(uri="",localName="javaOne",qname="javaOne",attributes={
})
characters(text="\n")
startElement(uri="",localName="jsr280",qname="jsr280",attributes={})
characters(text="\n\t")
startElement(uri="",localName="chapter",qname="chapter",attributes
=
{{uri="",localName="id",qname="id",type="CDATA",value="1"}})
characters(text="\n\t3 different parsers")
characters(text="\n\t")
endElement(uri="",localName="chapter",qname="chapter")
characters(text="\n")
endElement(ri="",localName="jsr280",qname="jsr280")
characters(text="\n")
endElement(uri="",localName="javaOne",qname="javaOne")
endDocument()
<javaOne>
<jsr280>
<chapter id="1">
3 different parsers
</chapter>
</jsr280>
</javaOne>

# DOM

- Models the whole document in memory
- Easy to access and manipulate data in the DOM tree




DOM parsing

# Pull and Push Compared

| Push (e.g. SAX 2.0) | Pull (e.g. StAX) |
|---|---|
| - act like I/O stream: cannot seek back or leap ahead. Perfect for having application reacting on incoming data | |
| - public domain | - Intuitive usage for the developer (iteration) |
| - widely used in Java SE | - can be easily stopped whenver the developer wishes to |
| - can be stopped by throwing an application specific exception (not intuitive) | - widely used and accepted tools for Java ME (kxml, XMLPull) - “de facto” standards, which do no cessarily conform to XMI 1.0 |
| - quite usefull subset for Java ME already in JSR 172 | - StAX in JSR 280 is compatible with JSR 173 defined StAX Java ME subset |

Java ME = Java Platform, Micro Edition (Java ME platform)
Java SE = Java Platform, Standard Edition (Java SE platform)

# Pull/Push and DOM Compared

| Push/Pull | DOM |
|---|---|
| NOT Competitors: DOM is a complementing tool for extensive document handling | |
| - performance/memory sensitive<br>- structure of the document does not need to be recognized | - modifying XML doc structurally<br>- sharing the XML doc in memory with other apps<br>- handling small/middle sized docs<br>- sorting<br>- when processing is started after validation |

- DOM parsing is the right tool when you need to process both the document and data
- DOM can be understood as SAX/XMLPull parser with event handler that stores XML document into an in-memory tree

# Agenda

- What are SOA, Web services, why to use them?
- Classic versus modern Web services
  - JSR 279 introduced
- How to extend Service Framework with new framework plug-ins?
- Interaction patterns and basic building blocks of JSR 279 web services:
  - ServiceDescriptor
  - Message
- Adding identity to Web services (e.g., Liberty)
- XML tooling
  - JSR 280 introduced
  - Three programming models StAX, SAX, and DOM compared
- **Current XML standardization landscape**
- DEMO 279 and 280
- Q&A

# XML Parser Landscape

| JSR | Title | XML parser Status |
|---|---|---|
| 172 | J2ME Web Services | SAX subset, lacking some substantial classes (e.g. XMLReader) |
| 173 | Streaming API for XML | Targets JavaSE. The “only standard pull parser”. Maintenance release includes 280 change proposals |
| 226 | Scalable 2D Vector Graphics API for J2ME | DOM subset restricted to SVG docs only |
| 279 | ServiceConnection API | Web services part of JSR 172 DOM Element is used |
| 280 | XML API for JavaME | State of the art XML tooling for Java ME. To be released in June |
| 287 | Scalable 2D Verctor Grphics API 2.0 for J2ME | Successor of JSR 226. Proper subset of JSR280 as needed for SVG purposes |
| 290 | Java Language XML UI Markup Integration | Refers to JSR280 DOM Core and Events |
| 102 | JDOM 1.0 | DOM API for Java. No activities since 2001 |

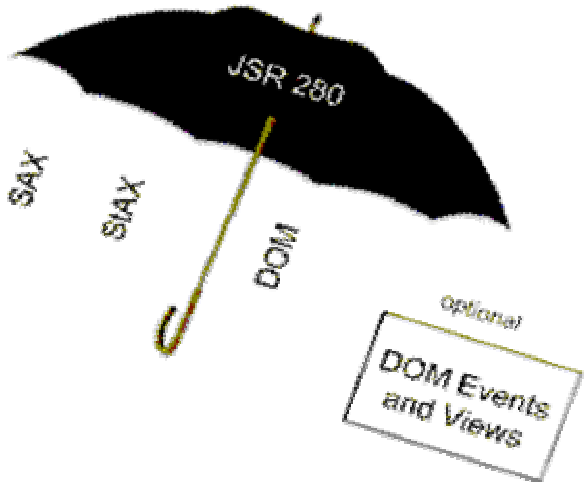
J2ME = Java 2 Platform, Micro Edition (J2ME™ platform)

# Target: Defragmenting XML API

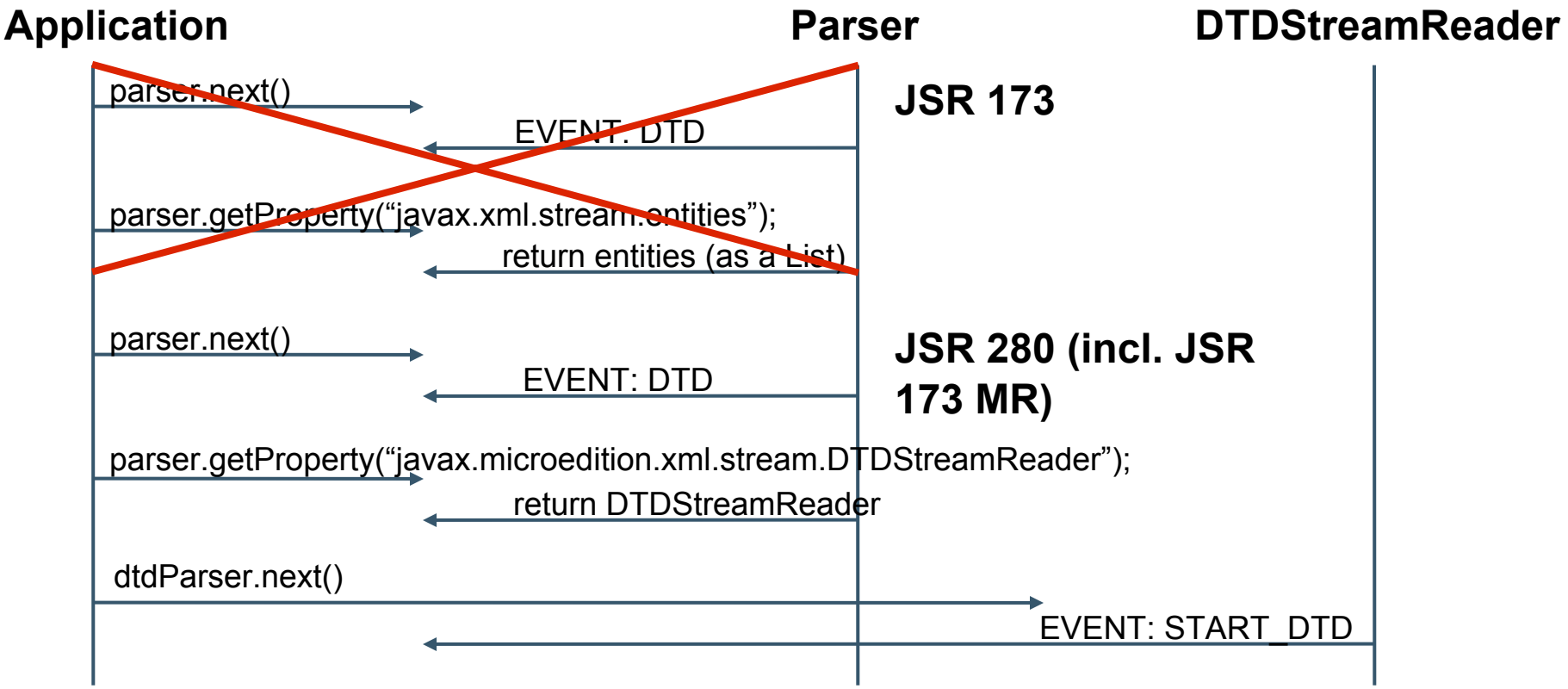
- New JSRs should either refer to the whole package or define a strict subset of any XML parsers of JSR 280
  - Otherwise the co-existence of JSRs in a same device is at risk
  - Referring directly to JSR 280 is the best way to avoid fragmentation
  - TCK signature tests have to be modified every time a new subset is defined
- For example: In JSR 287 the current proposal is:
  - If JSR 280 and 287 are implemented in the same device, then signatures of the DOM interfaces need to be according to 280; however, the classes implementing them for 287 use only need to have proper implementation of the methods in the 287 micro-DOM
  - If JSR 287 is alone then micro-DOM is used

# Target: Offer a Rich XML Ecosystem

- Push, pull, and DOM programming models, are provided
  - The ultimate goal has been to take whole packages, no subsets
    - Having added plenty of DOM Core Level 3 methods to DOM Core Level 2 package, breaks against the rule the most
    - The additions are due to alignment of JSR 280 with JSR 287 and 290
    - In StAX streamlining was done by removing duplicate methods
  - In StAX, a new tool for DTD parsing was introduced: the DTDStreamReader



# DTDStreamReader




```
<!DOCTYPE life[
<!ENTITY hum SYSTEM "humility.xml">]>
<life>Life is pleasant. Death is peaceful. But the transition is troublesome&hum;</life>
```

# JSR 279 and 280: Sample Flow

```
…
// use JSR 280 API to prepare WS request (StAX approach shown)
XMLOutputFactory outFactory = XMLOutputFactory.newInstance();
XMLStreamWriter sw = outFactory.createXMLStreamWriter (osRequest);
sw.writeStartDocument();
…
// use JSR 279 API to obtain a connection and call the web service
ServiceDescriptor sd = new ServiceDescriptor (fwID, contract, endpoint);
ServiceConnection con = ServiceManager.getServiceConnection (sd);
Message response = con.sendReceive (message);
// use JSR 280 API to parse WS response (DOM)
```

# DEMO

279: Authentication using Liberty (local and remote) and Basic Service Profile

280: Create sample XML using DOM and StAX and parse




# Q&A

# Web Services to Go: Mobile Access to Web Services With JSR-279 and JSR-280


Pia Niemelä

Nokia Corporation

http://jcp.org/en/jsr/detail?id=279

Jean-Yves Bitterlich

Sun Microsystems

http://jcp.org/en/jsr/detail?id=280


Session TS-5188