

**Development of UNICORE 6
Web-Services
UNICORE Tutorial
25.-26.07.2007**

Bastian Demuth



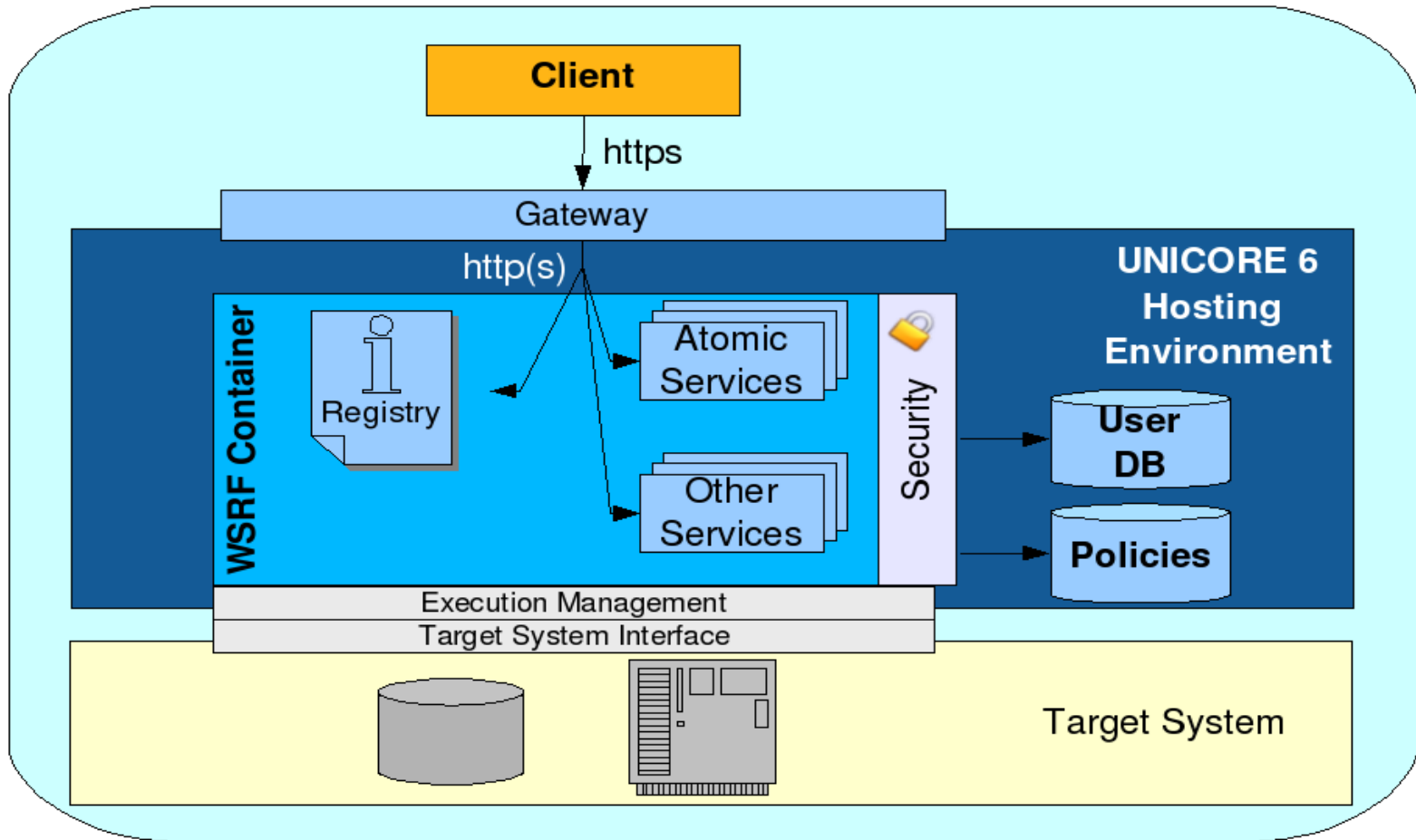
Forschungszentrum Jülich
in der Helmholtz-Gesellschaft

UNICORE Example Projects

- ▶ Java projects used in this presentation are available at
<http://zam461.zam.kfa-juelich.de:9129/dgrid/>
- ▶ Java 5 and ant are needed for compilation
- ▶ Check out the README files



UNICORE Hosting Environment



UNICORE Software Layers

UNICORE Atomic Services (UAS)

WSRFlite

Xfire

(SOAP Framework)

XMLBeans

(XMLSchema to Java)

Jetty

(Web Server)



UNICORE Resources vs. Plain Web Services

WSRFlite Resources

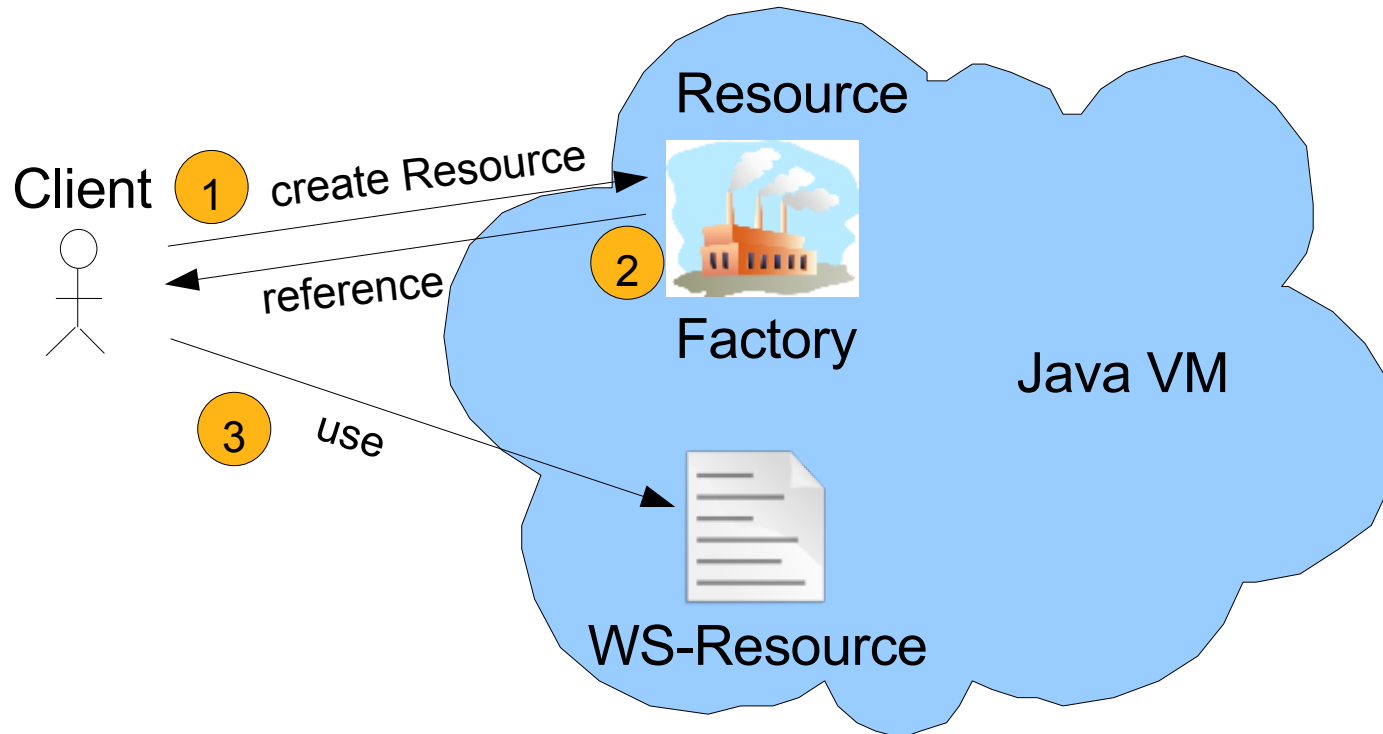
- ▶ can have states
- ▶ often one per client/action
- ▶ WS-ResourceLifetime
- ▶ standardised operations
- ▶ easy-to-use persistence mechanisms

Plain Web Services

- ▶ same behaviour for each call
- ▶ shared by all clients
- ▶ no defined lifetime
- ▶ arbitrary operations
- ▶ no ready-made persistence

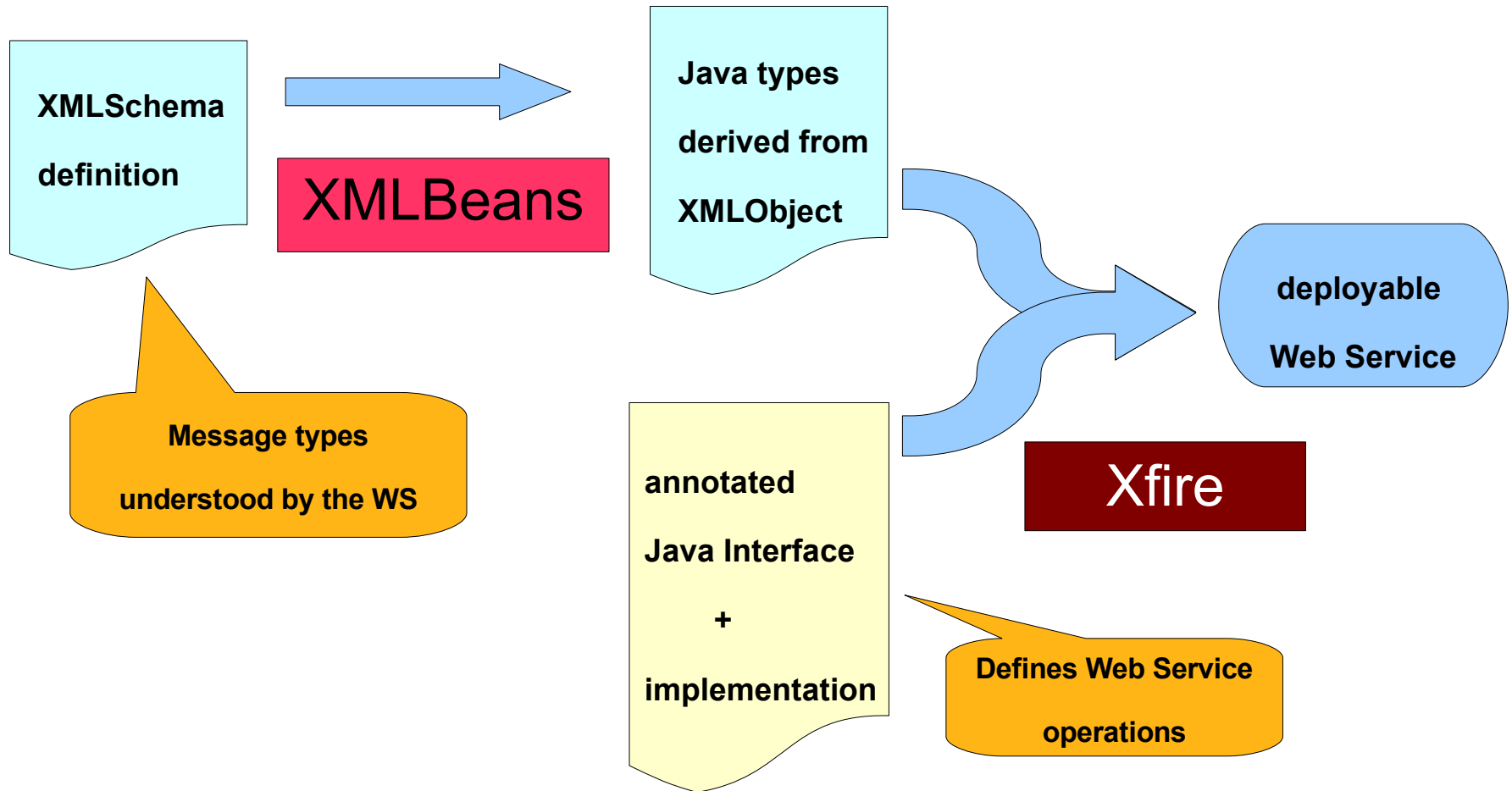


UNICORE WS-Resource creation



- ▶ Clients get their own WS-Resource instances which they can work with
- ▶ Resource Factory often a plain Web Service

UNICORE Overview: Writing a WS



UNICORE Defining a plain WS

XMLSchema definition

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.unicore.eu/2007/01/WSExample" >

  <xsd:element name="LookupRequest">
    <xsd:complexType>
      <xsd:sequence> <xsd:element name="ISBN" type="xsd:string" /> </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Document name

single String argument

annotated Java Interface

```
@WebService(targetNamespace="http://www.unicore.eu/WSExample", portName="LookupService")
public interface ILookupWS {

  @WebMethod(action="http://www..unicore.eu/WSExample/WSExample/LookupService/lookupRequest")
  public LookupResponseDocument lookup(LookupRequestDocument req);

}
```



UNICORE Implementing a plain WS

Implementation class

```
public class LookupWSImpl implements ILookupWS {  
  
    public LookupResponseDocument lookup(LookupRequestDocument req) {  
        // retrieve the book's ISBN from the input document  
        String isbn = req.getLookupRequest().getISBN();  
  
        // create response Document, enter the correct data and return  
  
        LookupResponseDocument resp = LookupResponseDocument.Factory.newInstance();  
  
        String title = catalogue.getBookTitle(isbn);  
  
        resp.addNewLookupResponse().setTitle(title);  
  
        return resp;  
    }  
}
```

Types generated by
XMLBeans



UNICORE Setup and deployment

- ▶ Add an entry to the wsrf-lite.xml file that configures the service container:

```
<!-- SERVICES -->  
<service name="LookupService" wsrf="false" persistent="false">  
  <interface class="de.fzj.unicore.wsexample.ILookupWS" />  
  <implementation class="de.fzj.unicore.wsexample.LookupWSImpl"/>  
</service>
```

- ▶ Copy the .jar file containing your web service to the /lib folder of the UNICORE 6 installation
- ▶ Restart Unicore 6
- ▶ The new web service is available at [http\(s\)://host:port/services/LookupService](http(s)://host:port/services/LookupService)



UNICORE Creating a WSRF service: Scenario

- ▶ Modelling a shopping cart for an online shop
- ▶ Each customer has his/her own WSResource representing his/her shopping cart
- ▶ Shopping carts contain items
- ▶ An item can be put in the cart more than once

```
<xsd:element name="Item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="ID" type="xsd:string"/>
      <xsd:element name="Price" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```



UNICORE Creating a WSRF service: ResourcePropertyDocument

- ▶ The schema defines the Resource Property Document representing the state of the Resource:

```
<!-- a shopping cart -->
```

```
<xsd:element name="ShoppingCartProperties">
```

```
<xsd:complexType>
```

```
<xsd:sequence>
```

```
<xsd:element ref="test:Entry" maxOccurs="unbounded"/>
```

```
<xsd:element ref="test:TotalPrice"/>
```

```
</xsd:sequence>
```

```
</xsd:complexType>
```

```
</xsd:element>
```

**Content of the cart
(list of entries)**

Total price of all items in the cart



UNICORE Creating a WSRF service: Resource Properties

- ▶ Each Resource Property is also defined in the schema:

```
<!-- a shopping cart entry -->
<xsd:element name="Entry">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="test:Item"/>
      <xsd:element name="Number" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- the current total price of items in the shopping cart -->
<xsd:element name="TotalPrice" type="xsd:float"/>
```

Each entry states how often a certain item is in the cart

The total price is a float



UNICORE Creating a WSRF service: Java Interface

- ▶ Make RP-related QNames available to other classes
- ▶ Define operations

Extend the WSRResource Interface

```
public interface ShoppingCart extends WSRResource {  
  
    public static final QName RPShoppingCartQName=new QName("http://example.org","ShoppingCartProperties");  
  
    public static final QName RPtotalPriceQName=new QName("http://example.org","TotalPrice");  
  
    public static final QName RPEntryQName=new QName("http://example.org","Entry");  
  
    /**  
     * method to add an item to the registry  
     */  
  
    @WebMethod(action = "http://example.org/AddItemRequest")  
    public AddItemResponseDocument add(AddItemRequestDocument request);  
}
```

QNames of Resource Property
Document and Resource Properties

These messages are defined
in the schema, too



UNICORE Creating a WSRF service: Java Implementation

- ▶ Populate the ResourcePropertyDocument
- ▶ Implement operations

```
public class ShoppingCartImpl extends WSResourceImpl implements ShoppingCart {  
    public QName getResourcePropertyDocumentQName() {  
        return RPShoppingCartQName;  
    }  
    public void initialise(String serviceName, Map<String, Object> initobjs) throws Exception {  
        super.initialise(serviceName, initobjs);  
        properties.put(RPTotalPriceQName, new TotalPriceProperty());  
        properties.put(RPEntryQName, new EntriesProperty());  
    }  
    public AddItemResponseDocument add(AddItemRequestDocument request) { ... }  
}
```

Hint to the Resource Property
Document from the schema

Provide Java implementations of
the Resource Properties



UNICORE WSRF: Setup and deployment

- ▶ Add an entry to the wsrflite.xml file that configures the service container:

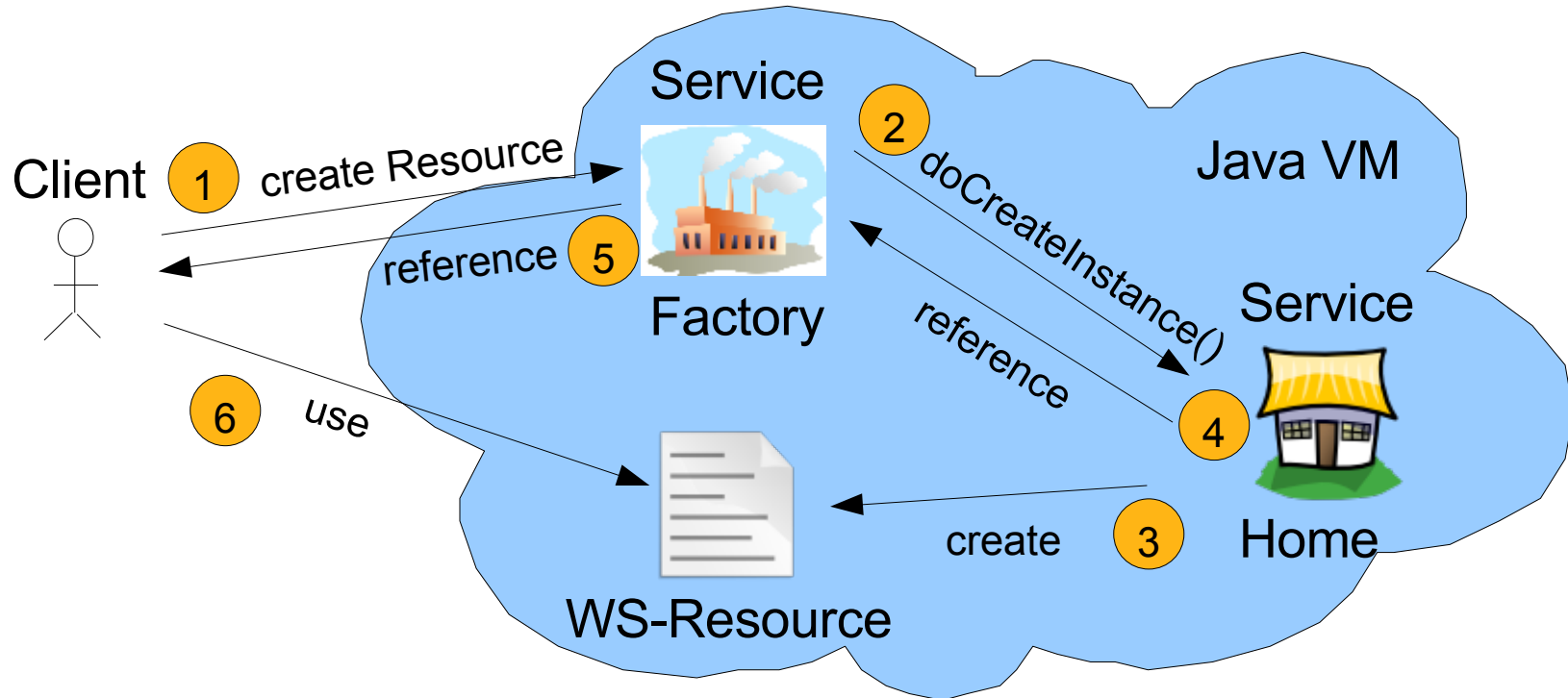
```
<!-- SERVICES -->  
<service name="Cart" wsrf="true" persistent="true">  
  <interface class="example.ShoppingCart" />  
  <implementation class="example.ShoppingCartHomeImpl"/>  
</service>
```

provide the Service Home
here (next slide)

- ▶ update /lib folder of the UNICORE 6 installation
- ▶ Restart Unicore 6
- ▶ Each Cart WS-Resource will have its own URL:
http(s)://host:port/services/Cart?res=some_unique_id



UNICORE The Service Home

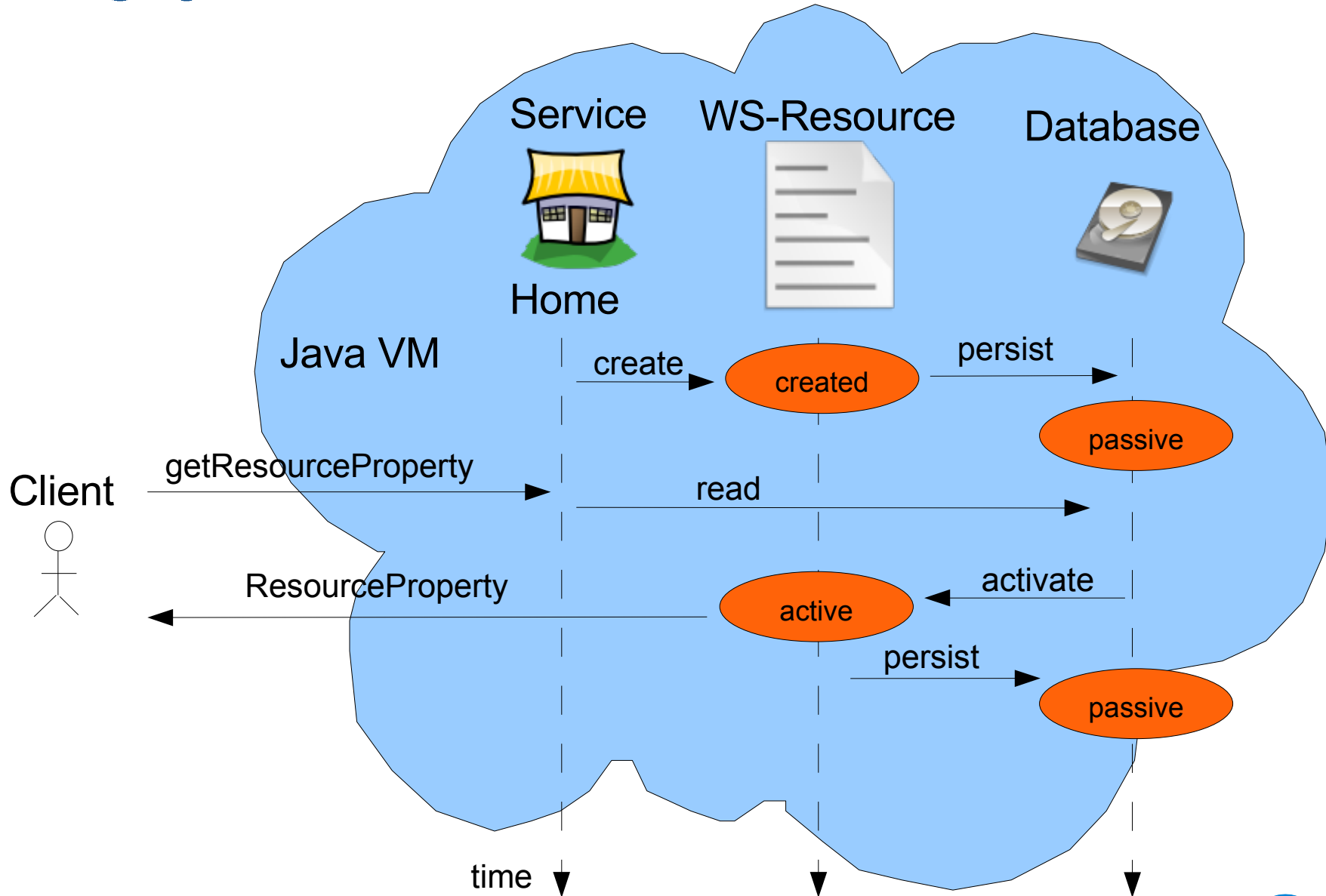


- ▶ Used to create and manage WS-Resources
- ▶ Usually trivial implementation:

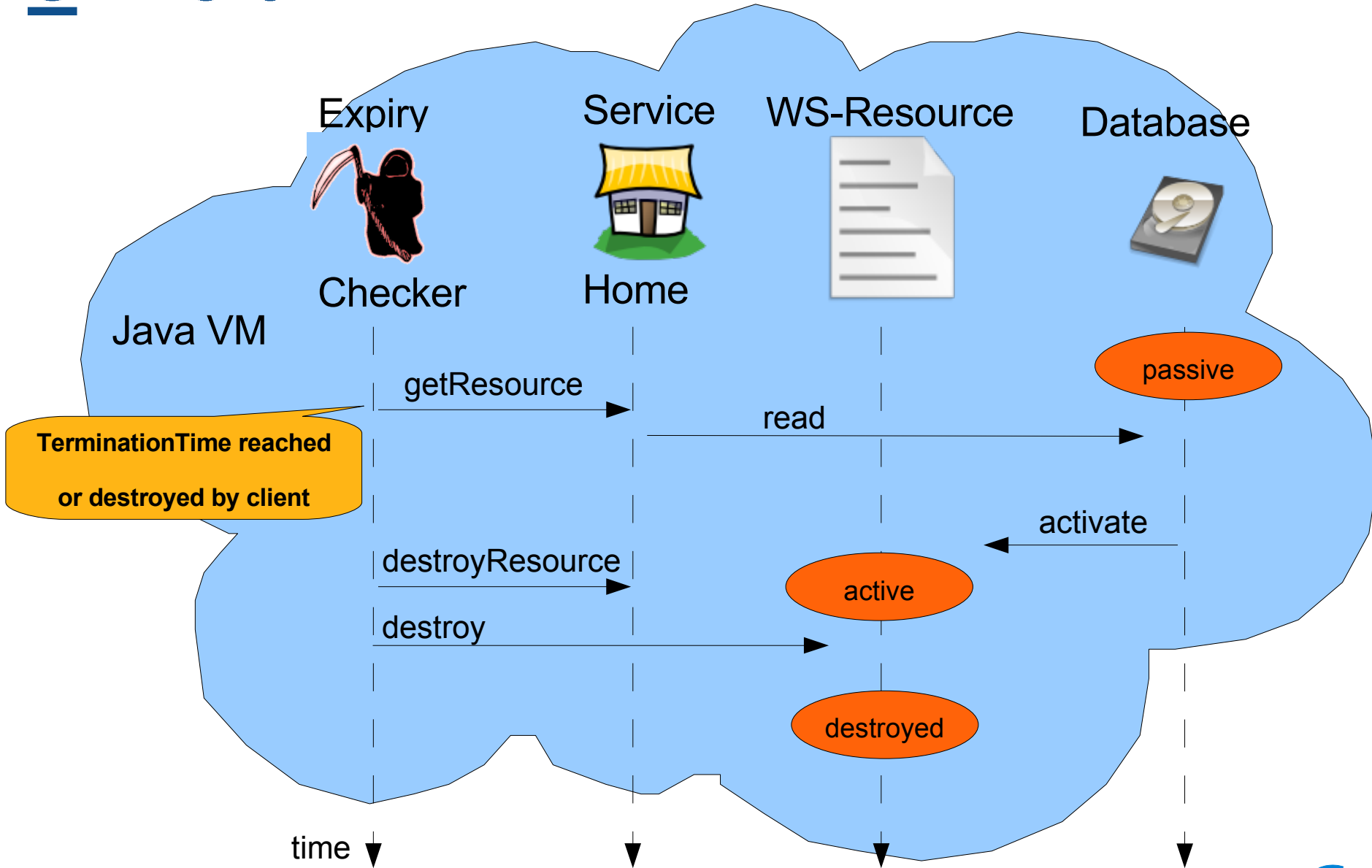
```
public class ShoppingCartHomeImpl extends WSResourceHomeImpl{  
    protected WSResourceImpl doCreateInstance() { return new ShoppingCartImpl(); }  
}
```



UNICORE Resource Persistence



UNICORE Resource Termination



UNICORE Client Proxies

- ▶ Proxy: Java object that can be treated like any other object implementing the Java WS interface
- ▶ Web service calls look like method invocations
- ▶ Proxies are created using a client factory
- ▶ Arguments for Factory:
 - ▶ Java interface of the web service
 - ▶ URL of the web service
 - ▶ UNICORE 6 security properties

```
IlookupWS proxy = (IlookupWS) new UASClientFactory.createPlainWSProxy(ILookupWS.class,url,securityProperties);
```

```
LookupRequestDocument req = LookupRequestDocument.Factory.newInstance();
```

```
req.addNewLookupRequest();
```

```
req.getLookupRequest().setISBN(isbn);
```

```
LookupResponseDocument resp = proxy.lookup(req);
```

```
String title = resp.getLookupResponse().getTitle();
```



UNICORE Tips and Hints

- ▶ Each client call to a WS is processed within its own thread => beware of concurrency problems
- ▶ For plain web services: each client call results in a new instance of your implementation type
- ▶ Implementation type should contain very little logic => delegate to the actual implementation
- ▶ Check out uas-core and have a look at the UAS!

