



The Java Architecture For XML Binding (JAXB)

By:

Yoav Zibin

Sharon Krisher

Motivation for JAXB

- Problem: How to manipulate this **data model**?
- **DOM** (data object model) solution:
 - **Pros**: simple, general (a schema is not even required)
 - **Cons**: no types, no compile-time checking

DOM pseudo-code example

```
root.getChild("Address").getChild("Number").getText()
```

returns a string

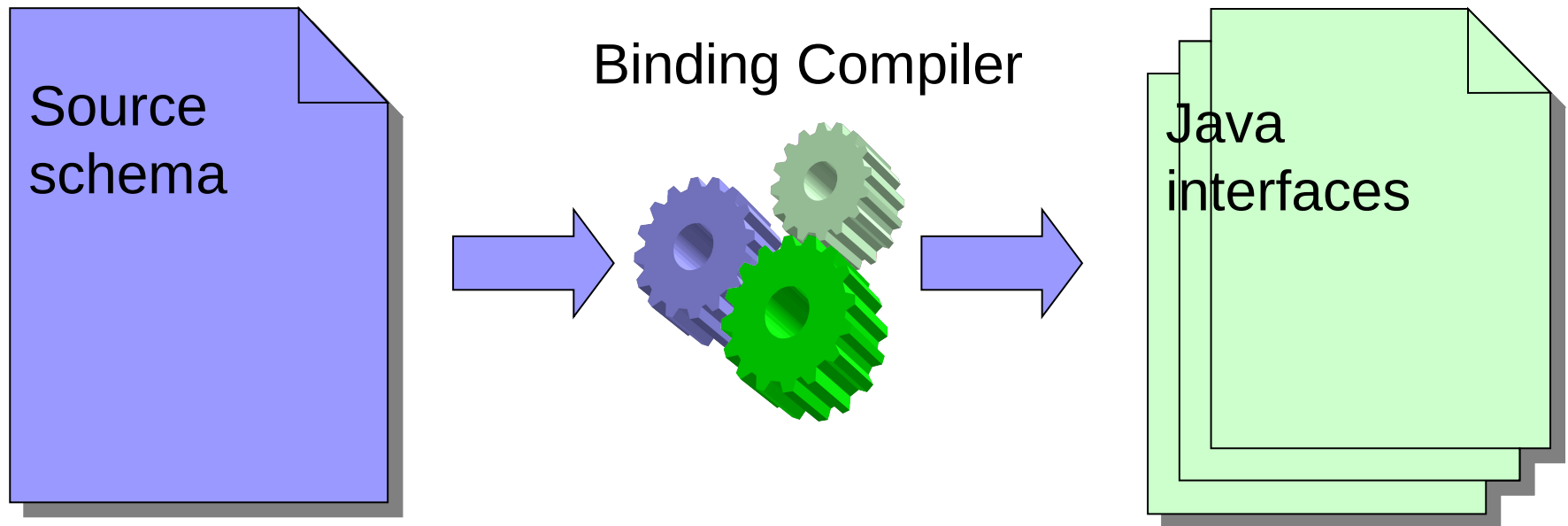
- I wish to write ...

```
root.getAddress() .getNumber()
```

returns a number

JAXB solution:

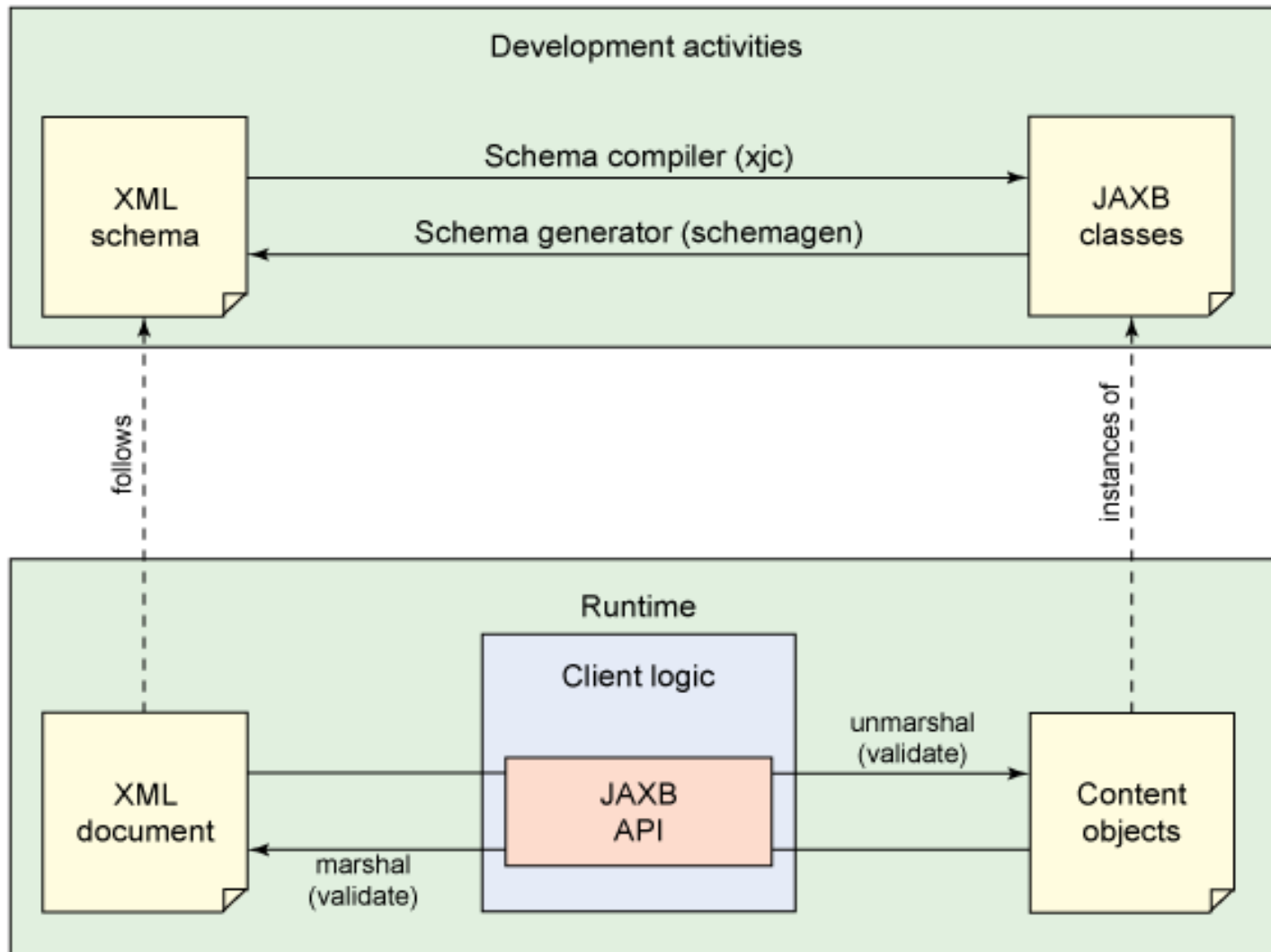
Mapping XML Schema to Java interfaces



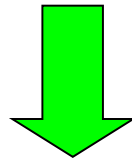
Pros: preserve types, compile-time checking

Cons: complex, specific to a certain schema

Mapping XML Schema to Java interfaces



```
<xs:complexType name="AddressType">
  <xs:sequence>
    <xs:element name="Number" type="xs:unsignedInt"/>
    <xs:element name="Street" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```



Binding Compiler

```
public interface AddressType {
  long getNumber();
  void setNumber(long value);

  String getStreet();
  void setStreet(String value);
}
```

Must be non-negative

Must be non-null

Main Features

- **Unmarshal**: xml \rightarrow objects
- Create / Read / Update / Delete objects
- **Validate** objects
- **Marshal**: objects \rightarrow xml
- No **roundtrip** guarantees
 - $\text{Marshal}(\text{Unmarshal}(\text{xml})) \neq \text{xml}$
 - We found that order is not always preserved
 - But usually roundtrip holds

Step 1: Create XML Schema

Demo.xsd

```
<xs:element name="Person" type="PersonType"/>
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Address" type="AddressType"
      minOccurs="1" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AddressType">
  <xs:sequence>
    <xs:element name="Number" type="xs:unsignedInt"/>
    <xs:element name="Street" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Step 2: Create XML Document

Demo.xml

```
<Person
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="C:\JAXB Demo\demo.xsd">
  <Name>Sharon Krisher</Name>
  <Address>
    <Street>Iben Gevirol</Street>
    <Number>57</Number>
  </Address>
  <Address>
    <Street>Moshe Sharet</Street>
    <Number>89</Number>
  </Address>
</Person>
```


Step 3: Run the binding compiler

- `xjc -p demo demo.xsd`
 - A package named **demo** is created (in the directory `demo`)
 - The package contains (among other things):
 - interface **AddressType**
 - interface **PersonType**

AddressType and PersonType

```
public interface AddressType {  
    long getNumber();  
    void setNumber(long value);  
  
    String getStreet();  
    void setStreet(String value);  
}
```

Must be non-negative

Must be non-null

```
public interface PersonType {  
    String getName();  
    void setName(String value);  
  
    /* List of AddressType */  
    java.util.List getAddress();  
}
```

Must be non-null

Must contain at least one item

Step 4: Create Context

- The context is the entry point to the API
- Contains methods to create Marshaller, Unmarshaller and Validator instances

```
JAXBContext context = JAXBContext.newInstance("demo");
```

The package name is **demo**
(Recall: `xjc -p demo demo.xsd`)

Step 5: Unmarshal: xml -> objects

```
Unmarshaller unmarshaller =  
    context.createUnmarshaller();  
unmarshaller.setValidating(true);
```

Enable validation of
xml according to the
schema while
unmarshalling

```
PersonType person =  
    (PersonType) unmarshaller.unmarshal(  
        new FileInputStream("demo.xml") );
```

Step 6: Read

```
System.out.println("Person name=" +  
    person.getName() );
```

```
AddressType address = (AddressType)  
    person.getAddress().get(0);
```

```
System.out.println("First Address: " +  
    " Street=" + address.getStreet() +  
    " Number=" + address.getNumber() );
```

Step 7: Manipulate objects

```
// Update
person.setName("Yoav Zibin");

// Delete
List addressList = person.getAddress();
addressList.clear();

// Create
ObjectFactory objectFactory = new ObjectFactory();
AddressType newAddr = objectFactory.createAddressType();
newAddr.setStreet("Hanoter");
newAddr.setNumber(5);
addressList.add( newAddr );
```

part of the demo package
uses the factory pattern

Step 8: Validate on-demand

```
Validator validator = context.createValidator();
```

```
validator.validate(newAddr);
```

Check that we have set **Street** and **Number**,
and that **Number** is non-negative

```
validator.validate(person);
```

Check that we have set **Name**, and that
Address contains at least one item

Step 9: Marshal: objects -> xml

```
Marshaller marshaller = context.createMarshaller();  
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,  
    Boolean.TRUE);  
marshaller.marshal(person,  
    new FileOutputStream("output.xml"));
```

output.xml

```
<Person>  
  <Name>Yoav Zibin</Name>  
  <Address>  
    <Street>Hanoter</Street>  
    <Number>5</Number>  
  </Address>  
</Person>
```