An S-Module Generator for Web Services

Bachelor's Thesis of Paulus Santoso
2004 Hans-Werner Sehring

Requirement

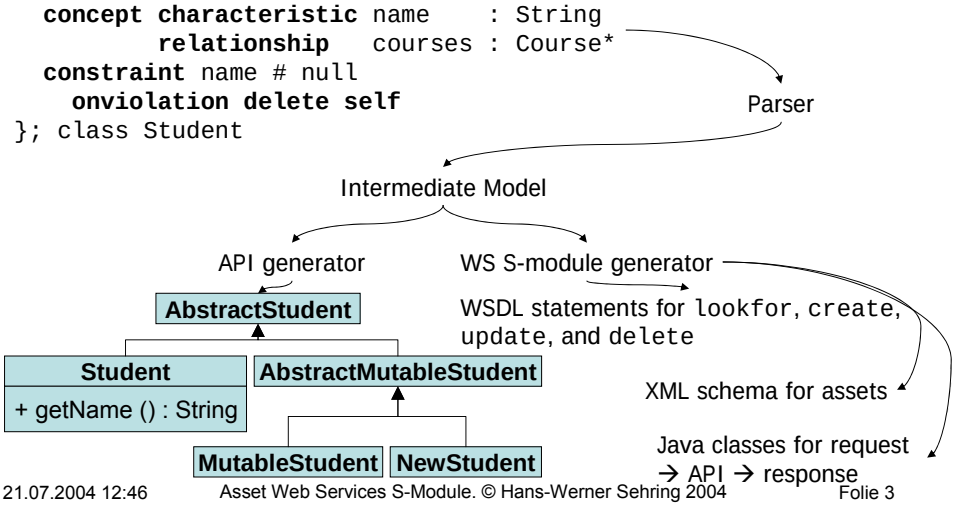
Task:
Access to concept-oriented content management systems (COCoMaSs) using Web Services (WS).

Solution:
A generator for the model compiler which generates an S-module.

- Creation of WSDL definitions of asset query and manipulation operations.
- Creation of an XML schema definition (?) describing assets and asset sets(?).
- Java code to map WS requests to the module API (see module API documentation) and from the module API to WS responses.

Basic Translation Scenario

```
module students
class Student refines Person {
  content photo : java.awt.Image
  concept characteristic name    : String
          relationship   courses : Course*
  constraint name # null
    onviolation delete self
}; class Student
```



Open Questions

Transactions

- If using API directly transactions are needed:
  `lock () ; set…() ; commit ()` / `abort () ;`
- No transactions in WS, only in extensions.
- Unknown implementation state of extensions.

Events

- To notify clients about changes use event extensions to WS.

More?

Transactions

Answer: not methods, but operations.

Instead: use AQL/AML operation statements as service methods.

E.g.: `modify a b` translated to

```
ma = a.lock () ;
ma.setName (b.getName ()) ;
…
a = ma.commit () ;
```

Make each service method have transactional behavior.

Events

???

Generating WSDL

Implementation idea:

- Generate Java classes implementing Web Service Interface. E.g.,

```
class StudentComponentInterface
  implements ComponentInterface
{
  public AssetIterator lookfor (AssetClass c,
                                QueryConstraint [] qc) {
    … access a module
  }// lookfor
  public AssetIterator lookfor (AssetClass c,
                                Asset p) {
    … access a module
  }// lookfor
  public AssetIterator lookfor (AssetClass c,
                                Asset [] ps) {
    … access a module
  }// lookfor
}// class StudentComponentInterface
```

  This means, the interface is uniform for all COCoMaSs. Implementations are created specifically for the classes at hand.
- Generate WSDL files using Systinet WASP (how is it called now?).

Generic Component Interface

Interfaces:

```
public interface ComponentInterface {
  AssetIterator lookfor (AssetClass c, QueryConstraint [] qc) ;
  AssetIterator lookfor (AssetClass c, Asset p) ;
  AssetIterator lookfor (AssetClass c, AssetIterator ps) ;
  NewAsset create (AssetClass c, AttributeInitialization [] ai) ;
  NewAsset create (AssetClass c, AbstractAsset prototype) ;
  AssetIterator create (AssetClass c , AssetIterator ps) ;
  MutableAsset modify (AbstractMutableAsset a, AssetInitialization [] ai) ;
  MutableAsset modify (AbstractMutableAsset a, AbstractAsset p) ;
  AssetIterator modify (AssetIterator as, AssetInitialization [] ai) ;
  AssetIterator modify (AssetIterator as, AssetIterator ps) ;
  NewAsset delete (MutableAsset a) ;
  AssetIterator delete (AssetIterator as) ;
}// class ComponentInterface
public interface QueryConstraint {
  String getAttributeName () ;
  Constraint.Comparator getComparator () ;
  Object getValue () ;
}// interface QueryConstraint
public interface AssetInitialization {
  String getAttributeName () ;
  Object getValue () ;
}// interface AssetInitialization
```

Generated Component Implementation

Implementation (what do we get from WASP???):

```
public class StudentsComponent {
  public AssetIterator lookfor (AssetClass c,QueryConstraint [] qc) {
    if ("Student".equals (c.getName ()) {
      StudentQuery q = (StudentQuery)c.startQuery () ;
      for each qc_i:
        if ("Name".equals (qc [i].getName ())
          if (qc [i].getComparator () == Constraint.Comparator.EQUAL)
            q.constrainNameEqual (qc [i].getValue ()) ;
          else if (qc [i].getComparator()==Constraint.Comparator.LESS)
            q.constrainNameLess (qc [i].getValue ()) ;
          …
    }// if Student
    …
  }// lookfor
  …
}// class StudentsComponent
```