

# Mapping EDOC to Web Services using YATL

Octavian Patrascoiu  
University of Kent

## YATL - Overview

- YATL: Yet Another Transformation Language
- Design goals
  - Easy to learn and efficient implementation
  - Deterministic behaviour
  - Transformations
    - Made of a LHS and RHS
    - LHS matches patterns
    - RHS builds the output
  - Solution
    - LHS implemented using OCL – declarative
    - RHS implemented using statements - procedural

## YATL- Features

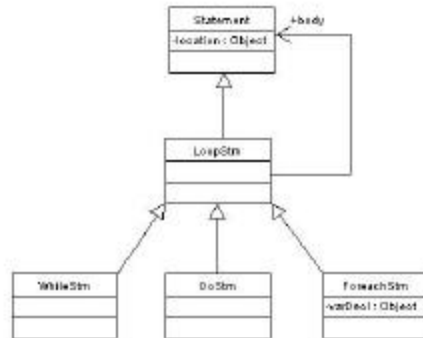
- Specialized language: model driven engineering
- Platform independent language
- Operates at abstract level
- Input
  - Source model
  - Target model
  - Source model instance (source population)
- Output
  - Target model instance (target population)
- Simple and with an efficient implementation

## YATL - Benefits

- **Improved portability** due to separating the application knowledge from the mapping to a specific implementation technology.
- **Increased productivity** due to automating the mapping.
- **Improved quality** due to reuse of well-proven patterns and best practices in the mapping.
- **Improved maintainability** due to better separation of concerns
- Enables different applications to be integrated by explicitly relating their models: this facilitates **integration** and **interoperability** and supports system evolution as platform technologies change.



## YATL - Abstract Syntax



## YATL - Concrete Syntax

Textual Syntax – Graphic Syntax in the future

Examples

[UML -> Java](#)

## YATL - Semantics

- Uses OCL semantics for queries
- Operational semantics for transformations, rules, and statements

## Transformation steps

- Edit UML models – Rational Rose, Gentleware
- Create a ksp project and generate code to implement UML concepts KMStudio
- Instantiate the source model instance – either programmatically or using the GUI generated by KMStudio
- Write the transformation written in YATL
- Create a ysp project and use YATLStudio to execute the transformation – YATL interpreter

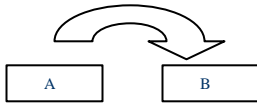
## KMFStudio projects

```
<?xml version="1.0"?>
<project type="">
  <path>C:\D\Work\Java Projects\YATL4KMF</path><name>edocksp</name>
  <xmi>C:\D\Work\Java Projects\YATL4KMF\src\test\scripts\EDOC.xmi</xmi>
  <license></license><offset></offset>
  <modelName></modelName>
  <visitor>true</visitor><HUTNvisitor>true</HUTNvisitor>
  <JTreevisitor>true</JTreevisitor><ViewEditvisitor>true</ViewEditvisitor>
  <XMLvisitor>true</XMLvisitor><observer>false</observer>
  <invariant>false</invariant><id>true</id>
  <factory>true</factory><repository>true</repository><browser>true</browser>
  <startup>true</startup>
  <bidirectional>true</bidirectional>
  <interfacePrefix></interfacePrefix><interfaceSuffix></interfaceSuffix>
  <classPrefix></classPrefix><classSuffix>$Class</classSuffix>
  <collectionInterface>java.util.List</collectionInterface>
  <collectionClass>java.util.Vector</collectionClass>
  <listInterface>java.util.List</listInterface>
  <listClass>java.util.Vector</listClass>
  <setInterface>java.util.Set</setInterface>
  <setClass>java.util.HashSet</setClass>
</project>
```

## YATLStudio projects

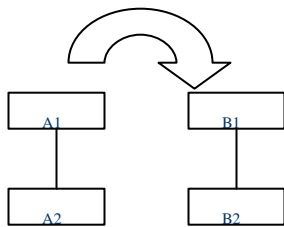
```
<?xml version="1.0"?>
<project>
  <pathname>C:\D\Work\Java Projects\YATL4KMF\edoc2ws. ysp</pathname>
  <transformation>C:\D\Work\Java Projects\YATL4KMF\src\test\scripts\edoc2ws. tl</transformation>
  <source>
    <model>C:\D\Work\Java Projects\YATL4KMF\src\test\scripts\EDOC.xmi</model>
    <repositoryClass>edoc.repository.EdocRepository$Class</repositoryClass>
    <repositoryContext>
      C:\D\Work\Java Projects\YATL4KMF\src\test\scripts\edocRep.xml
    </repositoryContext>
  </source>
  <target>
    <model>C:\D\Work\Java Projects\YATL4KMF\src\test\scripts\WS.xmi</model>
    <repositoryClass>ws.repository.WsRepository$Class</repositoryClass>
    <repositoryContext></repositoryContext>
  </target>
</project>
```

## Transformation patterns



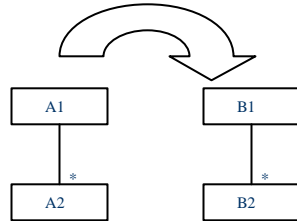
```
rule A2B match A () {  
  -- Create B  
  let b: B;  
  b := new B;  
  -- Set attributes  
  ...  
  -- Store mapping  
  track(self, a2b, b);  
}
```

## Transformation patterns



```
-- Link all the fields to the corresponding class  
rule linkB2andB1 match A1 () {  
  -- get B1  
  let b1: B1;  
  b1 := track(self, a1toB1, null);  
  -- get A2  
  let a2: A2;  
  a2 := self.a2;  
  -- get B2  
  let b2: B2;  
  b2 := track(a2, a2toB2, null);  
  -- Link B1 and B2  
  b1.b2 := b2;  
  b2.b1 := b1;  
}
```

## Transformation patterns



```
-- Link all the elements to the corresponding package
rule linkB2andB1 match A1 () {
  -- get b1
  let b1: B1;
  b1 := track(self, a12b1, null);
  -- For each a2
  foreach a2:A2 in self.a2 do {
    -- get b2
    let b2: B2;
    b2 := track(a2, a2toB2, null);
    -- link b1 and b2
    b1.b2 := b1.b2->including(b2);
    b2.b1 := b1;
  }
}
```

## EDOC to WS

EDOC -> WS

## Future Work

- YATL
  - Graphic syntax
  - Bridges to other modelling frameworks/tools
    - IBM's EMF
    - .NET
  - More examples
  - Add new features to YATLStudio
  - Model editing
  - Population editing
  - Debugging/tracing
  - Persistence
- EDOC 2 WS
  - map the behaviour - BPEL

## Questions

KMF-Studio

YATL-Studio

[www.cs.kent.ac.uk/projects/kmf](http://www.cs.kent.ac.uk/projects/kmf)