Computer Science at Kent

Model Transformations in YATL. Studies and Experiments

Octavian Patrascoiu

Technical Report No. 3-04
March 2004

This report describes three examples of model transformations, which have been implemented using YATL and the support provided by Kent Modelling Framework [KMF]. Model transformations are supported in KMF by a set of tools such as YATL-Studio, KMF-Studio, OCLCommon, and OCL4KMF. The core of the model transformations in KMF is YATL-Studio, a software environment used to create YATL projects and perform model transformations on them. The implementations of the source and target model are generated by KMF-Studio. The OCL 2.0 support is provided by OCLCommon and OCL4KMF, described in more details in [AP03][ALP03], which implement the OCL 2.0 standard.

Chapter 1. TRANSFORMATION ENVIRONMENT

The OMG's MDA is a new approach to develop large software systems. The core technologies of MDA are the Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Meta-Data Interchange (XMI) and Common Warehouse Metamodel (CWM). These standards are used to facilitate the design, description, exchange, and storage of models. MDA also introduces other important conceps: Platform-Independent Model (PIM), Platform-Specific Model (PSM), transformation language, and transformation engine. The relations and interactions between these concepts in KMF is depicted in Figure 1.1.

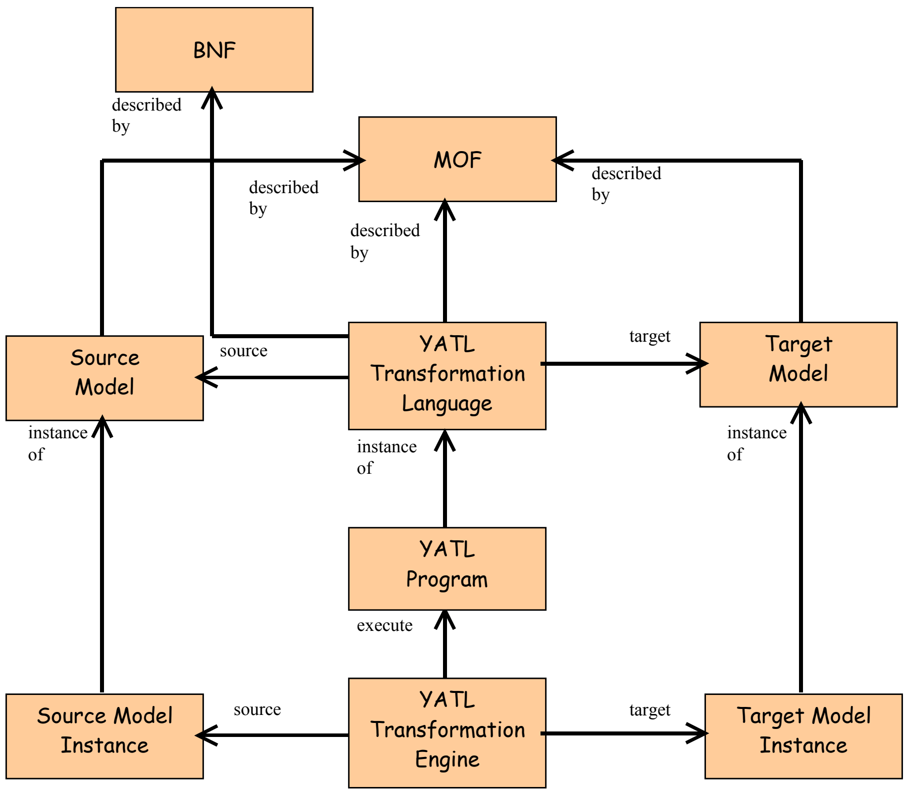


Figure 1.1 Transformation Environment

In our approach, the source and target models are described using the MOF language, which in this case acts like a metalanguage. The transformation language, in our case

YATL, is described using two metalanguages: BNF and MOF. BNF is used to describe the concrete syntax, while MOF is used to describe the abstract syntax. The transformation engine performs the mapping from a source model instance to a target model instance, executing a YATL program, which is an instance of the YATL transformation language.

The entire transformation process is performed in KMF following the steps:

- The source and target models are defined using a MOF editor (e.g. Rational Rose or Poseidon)
- KMF-Studio is used to generate Java implementations of the source and target models.
- The source model repository is populated used either Java hand-written code or GUI provided by the modelling tool generated by KMF-Studio.
- YATL-Studio is used to create a YATL project and perform the requested transformation.

Chapter 2. TRANSFORMATION FROM THE UML MODEL TO THE JAVA MODEL

Figure 2.1 contains a possible model of the Java programming language. This model is derived from the Java standard [Java] and covers only a subset of the language. The main elements of the Java model are:

- *JavaElement* denotes a generic element in the Java language and represents a generalization of all the elements from Java.
- *JavaPackageElement* denotes a *JavaElement* that can be included in a package.
- *JavaClassifier* denotes a generalization of the types used in Java
- *JavaPackage*, *JavaClass*, and *JavaInterface* denote Java packages, classes, and interfaces.
- Members contained within a class are represented by *JavaField* and *JavaMethod*.
- Parameters of Java operations are described using *JavaParameter*.
- Basic types are described using *DataType*.

The transformation that maps from UML model to Java model is performed in two phases. In the first phase 1-1 mappings are established between equivalent concepts:

- For every UML *Package* rule *umlPkg2JavaPkg* creates an instance of *JavaPackage.*
- For every UML *Class* rule *umlClass2JavaClass* creates an instance of *JavaClass.*
- For every UML *Attribute* rule *umlAttribute2JavaField* creates an instance of *JavaField*.
- For every instance of UML *AssociationEnd* rule *umlAssociationEnd2JavaField* creates an instance of *JavaField*.
- For every UML *Operation* rule *umlOperation2JavaMethod* creates an instanmce of *JavaMethod*.

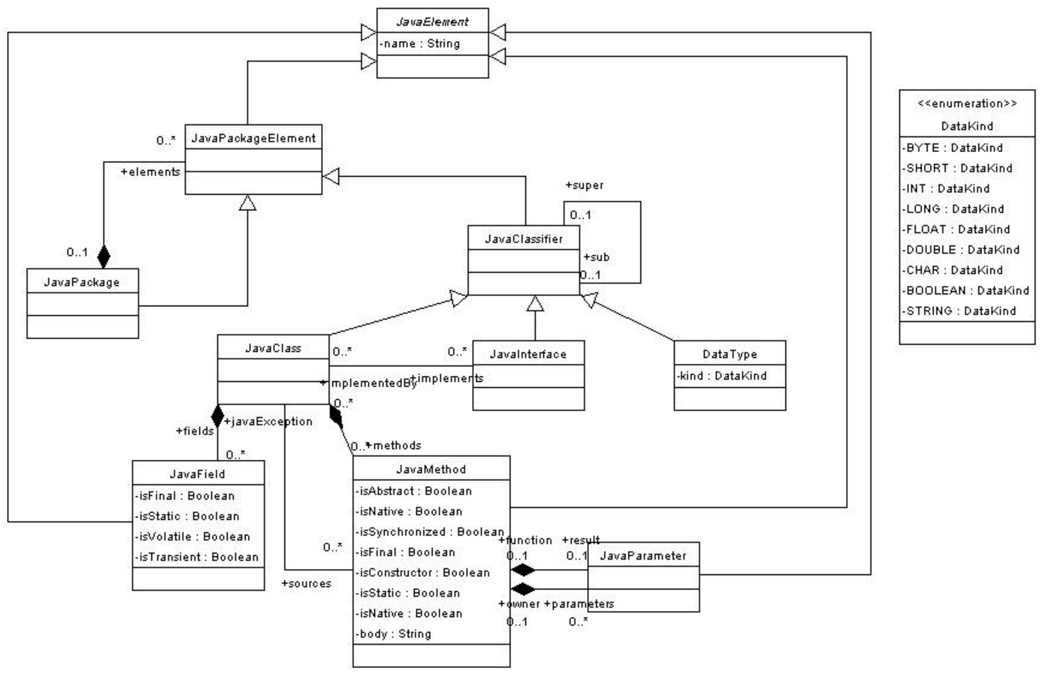


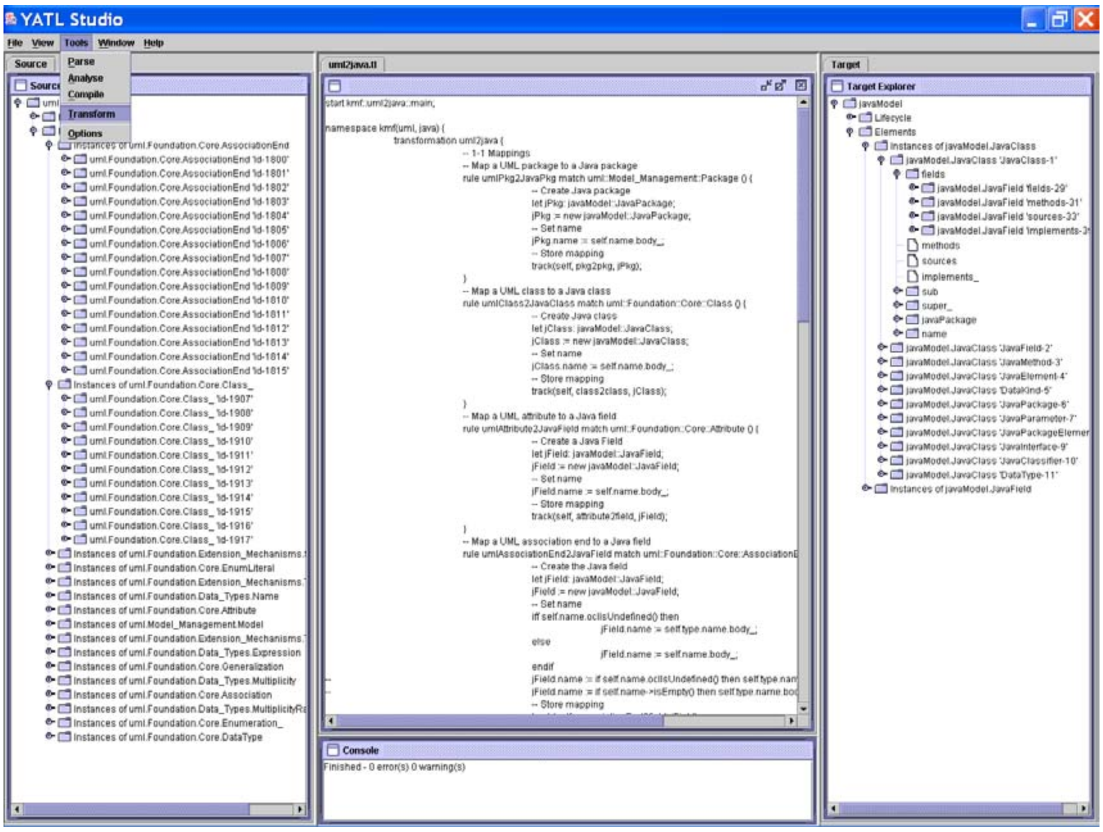
Figure 2.1 A possible Java model

The above rules create new instances of the required types and store the mappings using *track* constructions. This information is required in the second phase, which is responsible for filling the containment fields of Java model elements:

- Rule *linkElement2Package* scans all the *ownedElement*s of all the UML *Packages*, retrieves the corresponding *JavaPackageElements* and includes them it into the *elements* collection.
- Rules *linkAttribute2Class* and *linkAssociationEnd2Class* set the correct content of *fields* property.
- Rule *linkOperation2Class* sets the value of *methods* property.

The YATL program that performs this transformation is described in more details in Appendix 1.

The above transformation rules were tested on a source model instance that was populated using the XMI file that describes the Java model. The result of the mapping of the UML model instance described in Figure 2.1 to a Java model instance, using YATL-Studio and the YATL program from Appendix 1, is described in Figure 2.2.



```
uml
    Instances of uml.Foundation.Core.AssociationEnd
        uml.Foundation.Core.AssociationEnd 'id-1800'
        uml.Foundation.Core.AssociationEnd 'id-1801'
        uml.Foundation.Core.AssociationEnd 'id-1802'
        uml.Foundation.Core.AssociationEnd 'id-1803'
        uml.Foundation.Core.AssociationEnd 'id-1804'
        uml.Foundation.Core.AssociationEnd 'id-1805'
        uml.Foundation.Core.AssociationEnd 'id-1806'
        uml.Foundation.Core.AssociationEnd 'id-1807'
        uml.Foundation.Core.AssociationEnd 'id-1808'
        uml.Foundation.Core.AssociationEnd 'id-1809'
        uml.Foundation.Core.AssociationEnd 'id-1810'
        uml.Foundation.Core.AssociationEnd 'id-1811'
        uml.Foundation.Core.AssociationEnd 'id-1812'
        uml.Foundation.Core.AssociationEnd 'id-1813'
        uml.Foundation.Core.AssociationEnd 'id-1814'
        uml.Foundation.Core.AssociationEnd 'id-1815'
    Instances of uml.Foundation.Core.Class_
        uml.Foundation.Core.Class_ 'id-1907'
        uml.Foundation.Core.Class_ 'id-1908'
        uml.Foundation.Core.Class_ 'id-1909'
        uml.Foundation.Core.Class_ 'id-1910'
        uml.Foundation.Core.Class_ 'id-1911'
        uml.Foundation.Core.Class_ 'id-1912'
        uml.Foundation.Core.Class_ 'id-1913'
        uml.Foundation.Core.Class_ 'id-1914'
        uml.Foundation.Core.Class_ 'id-1915'
        uml.Foundation.Core.Class_ 'id-1916'
        uml.Foundation.Core.Class_ 'id-1917'
    Instances of uml.Foundation.Extension_Mechanisms.
    Instances of uml.Foundation.Core.EnumLiteral
    Instances of uml.Foundation.Extension_Mechanisms.
    Instances of uml.Foundation.Data_Types.Name
    Instances of uml.Foundation.Core.Attribute
    Instances of uml.Model_Management.Model
    Instances of uml.Foundation.Extension_Mechanisms.
    Instances of uml.Foundation.Data_Types.Expression
    Instances of uml.Foundation.Core.Generalization
    Instances of uml.Foundation.Data_Types.Multiplicity
    Instances of uml.Foundation.Core.Association
    Instances of uml.Foundation.Data_Types.MultiplicityRa
    Instances of uml.Foundation.Core.Enumeration_
    Instances of uml.Foundation.Core.DataType
```

```
start kmf::uml2java::main;

namespace kmf(uml, java) {
          transformation uml2java {
                    -- 1-1 Mappings
                    -- Map a UML package to a Java package
                    rule umlPkg2JavaPkg match uml::Model_Management::Package () {
                              -- Create Java package
                              let jPkg: javaModel::JavaPackage;
                              jPkg := new javaModel::JavaPackage;
                              -- Set name
                              jPkg.name := self.name.body_;
                              -- Store mapping
                              track(self, pkg2pkg, jPkg);
                    }
                    -- Map a UML class to a Java class
                    rule umlClass2JavaClass match uml::Foundation::Core::Class () {
                              -- Create Java class
                              let jClass: javaModel::JavaClass;
                              jClass := new javaModel::JavaClass;
                              -- Set name
                              jClass.name := self.name.body_;
                              -- Store mapping
                              track(self, class2class, jClass);
                    }
                    -- Map a UML attribute to a Java field
                    rule umlAttribute2JavaField match uml::Foundation::Core::Attribute () {
                              -- Create a Java Field
                              let jField: javaModel::JavaField;
                              jField := new javaModel::JavaField;
                              -- Set name
                              jField.name := self.name.body_;
                              -- Store mapping
                              track(self, attribute2field, jField);
                    }
                    -- Map a UML association end to a Java field
                    rule umlAssociationEnd2JavaField match uml::Foundation::Core::AssociationE
                              -- Create the Java field
                              let jField: javaModel::JavaField;
                              jField := new javaModel::JavaField;
                              -- Set name
                              iff self.name.oclIsUndefined() then
                                        jField.name := self.type.name.body_;
                              else
                                        jField.name := self.name.body_;
                              endif
--                            jField.name := if self.name.oclIsUndefined() then self.type.nam
--                            jField.name := if self.name->isEmpty() then self.type.name.bo
                              -- Store mapping
```

```
Finished - 0 error(s) 0 warning(s)
```

```
javaModel
    Lifecycle
    Elements
        Instances of javaModel.JavaClass
            javaModel.JavaClass 'JavaClass-1'
                fields
                    javaModel.JavaField 'fields-29'
                    javaModel.JavaField 'methods-31'
                    javaModel.JavaField 'sources-33'
                    javaModel.JavaField 'implements-3
                methods
                sources
                implements_
                sub
                super_
                javaPackage
                name
            javaModel.JavaClass 'JavaField-2'
            javaModel.JavaClass 'JavaMethod-3'
            javaModel.JavaClass 'JavaElement-4'
            javaModel.JavaClass 'DataKind-5'
            javaModel.JavaClass 'JavaPackage-6'
            javaModel.JavaClass 'JavaParameter-7'
            javaModel.JavaClass 'JavaPackageElemen
            javaModel.JavaClass 'JavaInterface-9'
            javaModel.JavaClass 'JavaClassifier-10'
            javaModel.JavaClass 'DataType-11'
        Instances of javaModel.JavaField
```

Figure 2.2 Example of mapping from UML model to Java model

Chapter 3. TRANSFORMATION FROM SPIDER DIAGRAMS MODEL TO OCL MODEL

This section contains the description of the transformation from the spider diagrams model to the OCL model. The first subsection contains a brief description of the concepts related to spider diagrams. The subsequent subsections describe briefly the mapping process.

3.1.1. Spider diagrams

This section introduces the main syntax and semantics of spider diagrams. Spider diagrams, introduced in [GHK99] are based on Euler diagrams rather than Venn diagrams. Spider diagrams considered here are adapted so that we can infer lower bounds for the cardinalities of the sets represented by the non-empty regions.

A *contour* is a simple closed plane curve. A *boundary rectangle* properly contains all other contours. A *basic region* is the bounded subset of the plane enclosed by a contour. A *region* is defined, recursively, as follows: any district is a region; if r_1 and r_2 are regions, then the union, intersection, or difference, of r_1 and r_2 are regions provided these are non-empty. A *zone* or *minimal region* is a region having no other region contained within it. Contours and regions denote sets. Every region is a union of zones. A region is *shaded* if each of its component zones is shaded. A shaded region denotes the empty set.

A *spider* is a tree with nodes, called *feet*, placed in different zones. The connecting edges, called *legs*, are straight lines. A spider *touches* a zone if one of its feet appears in that region. A spider may touch a zone at most once. A spider is said to *inhabit* the region that is the union of the zones it touches. For any spider *s*, the *habitat* of *s* is the region inhabited by *s*. A spider denotes the existence of an element in the set denoted by the habitat of the spider. Two distinct spiders denote distinct elements.

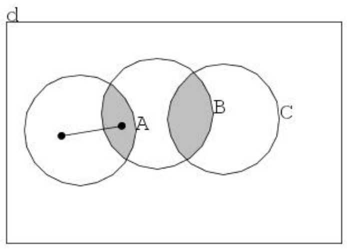


Figure 3.1 A spider diagram

```
context OclVoid inv:
let
      setA: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::A) and
            not x.oclIsKindOf(RwD::B) and  not x.oclIsKindOf(RwD::C)),
      setB: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::B) and
            not x.oclIsKindOf(RwD::A) and  not x.oclIsKindOf(RwD::C)),
      setA_B: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::A) and x.oclIsKindOf(RwD::B) and
            not x.oclIsKindOf(RwD::C)),
      setC: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::C) and
            not x.oclIsKindOf(RwD::A) and  not x.oclIsKindOf(RwD::B)),
      setA_C: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::A) and x.oclIsKindOf(RwD::C) and
            not x.oclIsKindOf(RwD::B)),
      setB_C: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::B) and x.oclIsKindOf(RwD::C) and
            not x.oclIsKindOf(RwD::A)),
      setA_B_C: Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            x.oclIsKindOf(RwD::A) and x.oclIsKindOf(RwD::B) and
            x.oclIsKindOf(RwD::C)),
      out :Set(OclAny) = OclAny.allInstances()->select(x : OclAny |
            not x.oclIsKindOf(RwD::A) and  not x.oclIsKindOf(RwD::B) and
            not x.oclIsKindOf(RwD::C))
in
      (setA_B->size() = 1) and (setB_C->size() = 0) or
      (setA->size() >= 1) and (setA_B->size() = 0) and (setB_C->size() = 0)
```

Figure 3.2 OCL equivalent expression

Figure 3.1 contains a spider diagram with contours A, B, and C, six zones, two shaded zones, and a spider with o leg and two feet. The construction of the equivalent OCL expression, presented in Figure 3.2, is based on the following basic ideas:

- Every spider diagram maps to an OCL let expression.
- Every zone maps to a variable declaration of Set type.
- Every boundary condition regarding a zone maps to an OCL expression that checks the size of the corresponding variable.

The transformation rules and their meaning are described briefly in Table 3.1.

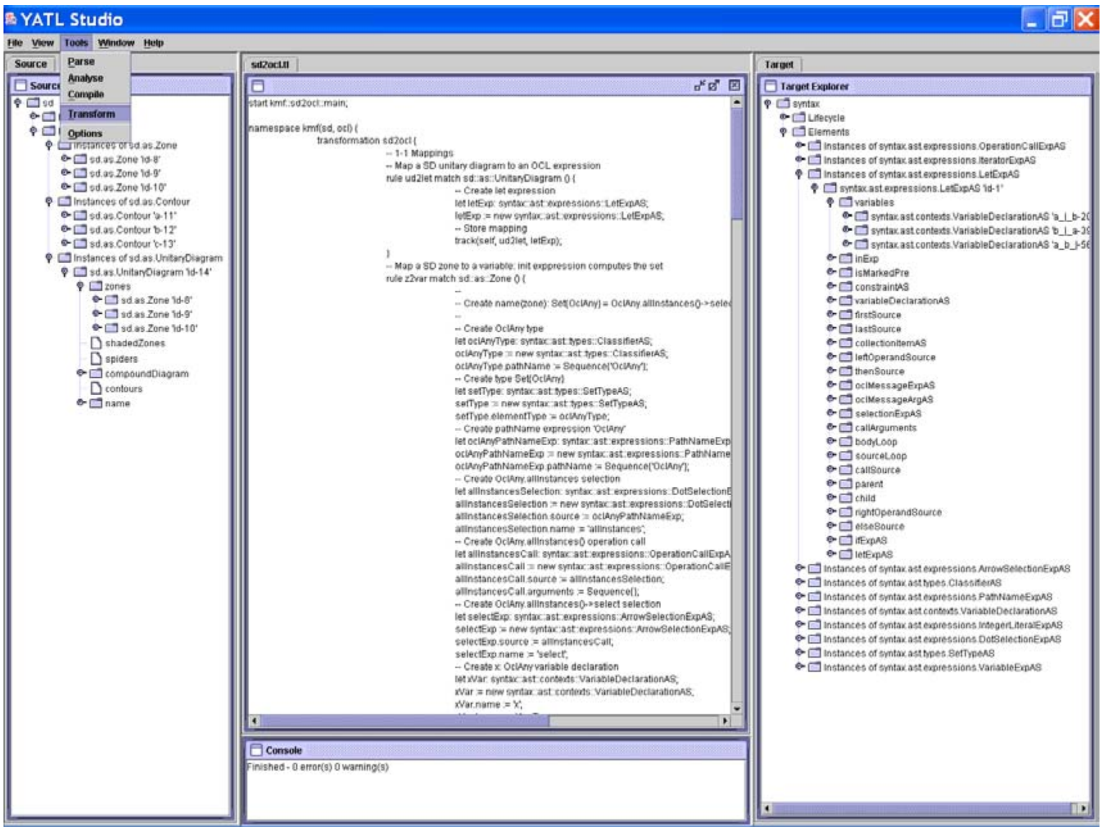
Rule name	**Rule description**
ud2let	Creates an OCL *LetExpression* for each spider diagram *Diagram* and stores the mapping using the *track* mechanism.
z2var	Creates an OCL *VariableDeclaration* for each spider diagram *Zone* and stores the mapping using the *track* mechanism.
ud2in	Creates an OCL *Expression,* representing the body of the *LetExpression*, for each spider diagram *Diagram* and stores the mapping using the *track* mechanism.
linkLet2Variables	Sets the correct value for *variables* property for each OCL *LetExpression*.
linkLet2In	Sets the correct value for *body* property for each OCL *LetExpression*

main	Invokes the above rules in the following order:
	`apply ud2var();`
	`apply z2var();`
	`apply ud2in();`
	`apply linkLet2Variables();`

	`apply linkLet2In();`

Table 3.1 *Transformation rules from spider diagrams to OCL*

The entire YATL program that performs this transformation is described in more details Appendix 2. Appendix 2 contains also the Java code that has been used to populate a source model instance. The result of the mapping of this spider diagram model instance to an OCL model instance, using YATL-Studio and the YATL program from Appendix 2, is described in Figure 3.3.



```
sd
  Instances of sd.as.Zone
    sd.as.Zone 'id-8'
    sd.as.Zone 'id-9'
    sd.as.Zone 'id-10'
  Instances of sd.as.Contour
    sd.as.Contour 'a-11'
    sd.as.Contour 'b-12'
    sd.as.Contour 'c-13'
  Instances of sd.as.UnitaryDiagram
    sd.as.UnitaryDiagram 'id-14'
      zones
        sd.as.Zone 'id-8'
        sd.as.Zone 'id-9'
        sd.as.Zone 'id-10'
      shadedZones
      spiders
      compoundDiagram
      contours
      name
```

```
start kmf::sd2ocl::main;

namespace kmf(sd, ocl) {
            transformation sd2ocl {
                        -- 1-1 Mappings
                        -- Map a SD unitary diagram to an OCL expression
                        rule ud2let match sd::as::UnitaryDiagram () {
                                    -- Create let expression
                                    let letExp: syntax::ast::expressions::LetExpAS;
                                    letExp := new syntax::ast::expressions::LetExpAS;
                                    -- Store mapping
                                    track(self, ud2let, letExp);
                        }
                        -- Map a SD zone to a variable: init exppression computes the set
                        rule z2var match sd::as::Zone () {
                                    --
                                    -- Create name(zone): Set(OclAny) = OclAny.allInstances()->selec
                                    --
                                    -- Create OclAny type
                                    let oclAnyType: syntax::ast::types::ClassifierAS;
                                    oclAnyType := new syntax::ast::types::ClassifierAS;
                                    oclAnyType.pathName := Sequence{'OclAny'};
                                    -- Create type Set(OclAny)
                                    let setType: syntax::ast::types::SetTypeAS;
                                    setType := new syntax::ast::types::SetTypeAS;
                                    setType.elementType := oclAnyType;
                                    -- Create pathName expression 'OclAny'
                                    let oclAnyPathNameExp: syntax::ast::expressions::PathNameExp
                                    oclAnyPathNameExp := new syntax::ast::expressions::PathName
                                    oclAnyPathNameExp.pathName := Sequence{'OclAny'};
                                    -- Create OclAny.allInstances selection
                                    let allInstancesSelection: syntax::ast::expressions::DotSelectionE
                                    allInstancesSelection := new syntax::ast::expressions::DotSelecti
                                    allInstancesSelection.source := oclAnyPathNameExp;
                                    allInstancesSelection.name := 'allInstances';
                                    -- Create OclAny.allInstances() operation call
                                    let allInstancesCall: syntax::ast::expressions::OperationCallExpA
                                    allInstancesCall := new syntax::ast::expressions::OperationCallE
                                    allInstancesCall.source := allInstancesSelection;
                                    allInstancesCall.arguments := Sequence{};
                                    -- Create OclAny.allInstances()->select selection
                                    let selectExp: syntax::ast::expressions::ArrowSelectionExpAS;
                                    selectExp := new syntax::ast::expressions::ArrowSelectionExpAS;
                                    selectExp.source := allInstancesCall;
                                    selectExp.name := 'select';
                                    -- Create x: OclAny variable declaration
                                    let xVar: syntax::ast::contexts::VariableDeclarationAS;
                                    xVar := new syntax::ast::contexts::VariableDeclarationAS;
                                    xVar.name := 'x';
```

```
Finished - 0 error(s) 0 warning(s)
```

```
syntax
  Lifecycle
  Elements
    Instances of syntax.ast.expressions.OperationCallExpAS
    Instances of syntax.ast.expressions.IteratorExpAS
    Instances of syntax.ast.expressions.LetExpAS
      syntax.ast.expressions.LetExpAS 'id-1'
        variables
          syntax.ast.contexts.VariableDeclarationAS 'a_i_b-20
          syntax.ast.contexts.VariableDeclarationAS 'b_i_a-39
          syntax.ast.contexts.VariableDeclarationAS 'a_b_i-56
        inExp
        isMarkedPre
        constraintAS
        variableDeclarationAS
        firstSource
        lastSource
        collectionItemAS
        leftOperandSource
        thenSource
        oclMessageExpAS
        oclMessageArgAS
        selectionExpAS
        callArguments
        bodyLoop
        sourceLoop
        callSource
        parent
        child
        rightOperandSource
        elseSource
        ifExpAS
        letExpAS
    Instances of syntax.ast.expressions.ArrowSelectionExpAS
    Instances of syntax.ast.types.ClassifierAS
    Instances of syntax.ast.expressions.PathNameExpAS
    Instances of syntax.ast.contexts.VariableDeclarationAS
    Instances of syntax.ast.expressions.IntegerLiteralExpAS
    Instances of syntax.ast.expressions.DotSelectionExpAS
    Instances of syntax.ast.types.SetTypeAS
    Instances of syntax.ast.expressions.VariableExpAS
```

Figure 3.3 *Mapping spider diagrams to OCL*

Chapter 4. TRANSFORMATION FROM A SUBSET OF EDOC TO WEB SERVICES

This section provides a mapping of a distributed system described using a subset of EDOC into an equivalent system described using Web Services. The subset contains only distributed systems described by EDOC's Model Document and Component Collaboration Architecture profiles. The equivalence between source and target system is established using the behaviour of the system from the users point of view. The first two subsections contain a brief description of EDOC and Web Services. The subsequent sections describe the system and the transformation that performs the mapping. The entire transformation from Model Document to XML Schema is described in Appendix 3.

4.1.1. EDOC: the UML profile for Enterprise Distributed Object Computing Specification

The EDOC profile of UML was adopted by the OMG in November of 2001 as the standard for modeling enterprise systems. It is the modeling standard for Internet computing - providing for model driven development of enterprise systems based on the OMG's MDA.

EDOC is proposed as the *modeling framework for Internet computing*, integrating web services, messaging, ebXML, .NET and other technologies under a common technology-independent model. It comprises a set of profiles, which define the Enterprise Collaboration Architecture (ECA), the Patterns, and the Technology Specific Models and Technology Mappings.

The ECA allows the definition of PIMs and provides five UML profiles:

- The *Component Collaboration Architecture* (CCA) uses UML classes, collaborations, and activity graphs to model the structure and behaviour of components that are part of a system.

- The *Entity profile* describes a set of UML extensions that may be used to model entity objects.
- The *Events profile* describes a set of UML extensions that may be used to model event driven systems.
- The *Business Process* profile specializes the CCA and comprises a set of UML extensions that can be used to model business processes.
- The *Relationship* profile contains extensions of the UML core to rigorously specify relationships.

The *Patterns* profile defines a standard means, Business Function Object Patterns that can be used to describe object models using the UML package notation.

The *Technology Specific Models* and the *Technology Specific Mappings* take into account the mapping from ECA specification to technology specific models. It defines and EDOC profile for Enterprise Java Beans (EJB) and another for Flow Composition Model (FCM).

4.1.2. Web Service

The purpose of web services is to enable a distributed environment in which any number of applications, or application components, can communicate in a platform-independent, language-independent fashion. A web service is a piece of software application, located on the Internet, that is accessible through standard-based Internet protocols such as HTTP or SMTP.

Given this definition, several technologies used in recent years could have been classified as web service technologies, but were not. These technologies include win32 technologies, J2EE, CORBA, and CGI scripting. These technologies are not web services technologies mainly because are based on a proprietary binary standard, which is not supported globally by most major technologies firms. The core of the web services technologies is made of eXtensible Markup Language [XML], Simple Object Access Protocol [SOAP], Web Service Description Language [WSDL], and Universal Description, Discovery and Integration [UDDI].

XML is a widely used standard from the World Wide Web Consortium (W3C) that facilitates the interchange of data between computer applications. XML uses use markup codes (tags) is, just like the language used for Web pages, the HyperText

Markup Language (HTML), in that both. Computer programs can automatically extract data from an XML document, using its associated DTD as a guide.

SOAP provides a standard packaging structure for exchanging XML documents over a variety of Internet protocols, including HTTP, SMTP, and FTP. The existence of a standard transport mechanism allows heterogeneous clients and servers to communicate. For example, .NET clients can invoke EJBs and Java clients can invoke .NET Components through SOAP.

WSDL is an XML technology that provides a standard description of web services. WSDL can be used to describe the representation of input and output parameters of an invocation, the function's structure, the nature of the invocation, and the protocol used for transport.

UDDI provides a worldwide registry of web services for description, discovery, and integration purposes. Analysts and technologist use UDDI to discover available web services by searching for categories, names or identifiers.

4.1.3. Mapping from Document Model to XML Schema

Both EDOC and WS models describe business processes. A business process manipulates and exchange information with other business processes. To describe the information that is manipulated or exchanged during a business process, both EDOC and WS have dedicated components: *Model Document* and *XML Schema* respectively.

The first step in the mapping from EDOC to WS is to map the models that are used to describe the information that is manipulated. This section contains the description of the mapping process from Model Document to XML Schema.

The Document Model package from the EDOC profile defines the information that can be manipulated by EDOC *ProcessComponent*s. The document model is based in *data elements* that can be either primitive *data types* or *composite data*. A *CompositeData* contains several attributes. An *attribute* has a specific type, an initial value and can be marked as *required* or as *many* to indicate the cardinality. An *enumeration* defines a type with a fixed set of values. The document model is described in Figure 4.1.

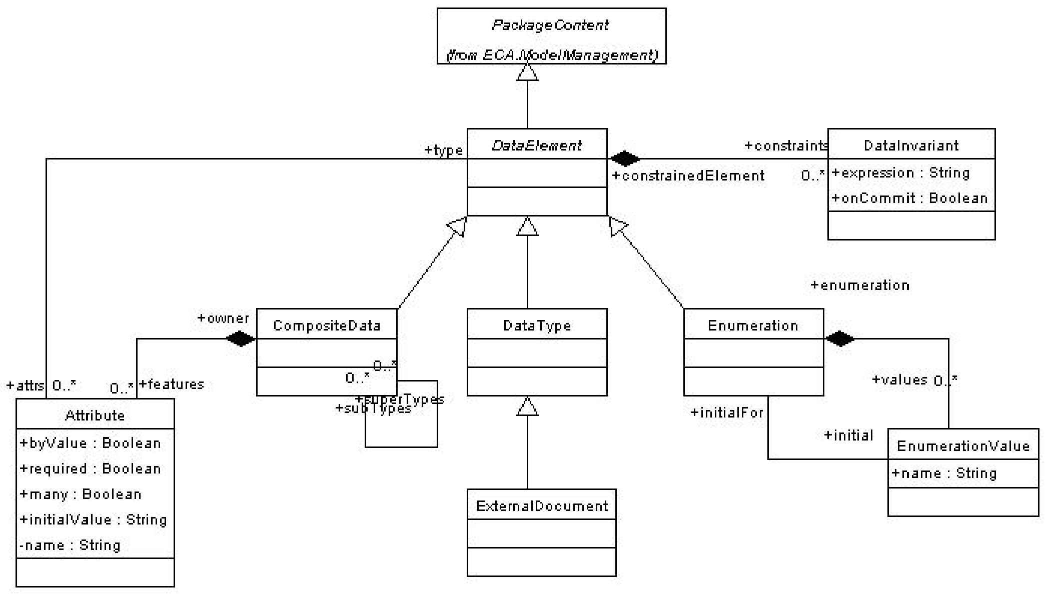


Figure 4.1 Document Model profile

The XML Schema [XMLS] describes the information that can be manipulated by web services. It contains types that can be *simple*, such as string or decimal, or *complex*. A *ComplexType* contains a sequence of *attributes*. An *Attribute* has a name and a given type. A partial model of XML Schema is given in Figure 4.2.

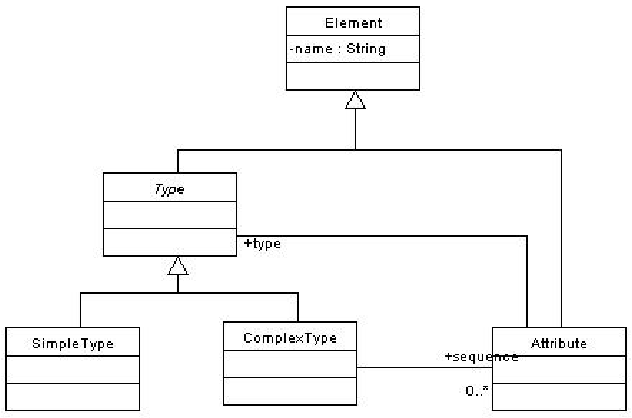


Figure 4.2 XML Schema

It is obvious that mapping from Model Document to XML Schema means mapping from DataElement, DataType and CompositeData to Type, SimpleType and ComplexType respectively. The transformation process and the rules that perform the mapping are described briefly in Table 4.1.

Rule name	Rule description
dt2st	Creates a XML Schema *SimpleType* for each Document Model *DataType* and stores the mapping using the *track* mechanism.
cd2ct	Creates a XML Schema *ComplexType* for each Document Model *CompositeData* and stores the mapping using the *track* mechanism.
at2at	Creates a XML Schema *Attribute* for each Document Model *Attribute* and stores the mapping using the *track* mechanism.
linkAttribute2Type	Sets the correct value for *type* property for each XML Schema *Attribute*.
linkComplexType2Attribute	Sets the correct value for *sequence* property for each XML Schema *CompositeType*

documentModel2xsd	Invokes the above rules in the following order:
	`apply dt2st();`
	`apply cd2ct();`
	`apply at2at();`
	`apply linkAttribute2Type();`
	`apply linkComplexType2Attribute();`

Table 4.1 Transformation rules for Document Model to XML Schema mapping

4.1.4. Mapping from CCA to WSDL

The CCA profile details how the UML concepts of classes and collaboration graphs can be used to model the structure and the behaviour of the components that comprise a system. In CCA *process components* interact with other process components using a set of *ports*. A *ProcessComponent* describes the contract for a component that performs actions. A *Port* defines a point of interaction between process components. Ports can be classified according to the complexity of the interaction in *FlowPorts*, *ProtocolPorts, OperationPorts, and MultiPorts.* A *FlowPort* is a port capable to produce and consume a single data type. *ProtocolPorts* describe more complex interactions based on *Protocols*. A *Protocol* is a method by which two components can communicate. An *OperationPort* is a port that realizes a typical request/response operation. A *MultiPort* is a group of ports whose actions are tied together. The specification of a *ProcessComponent* may include a *Choreography* to specify the sequence of interactions performed through ports. Figure 4.3 describes the CCA profile.

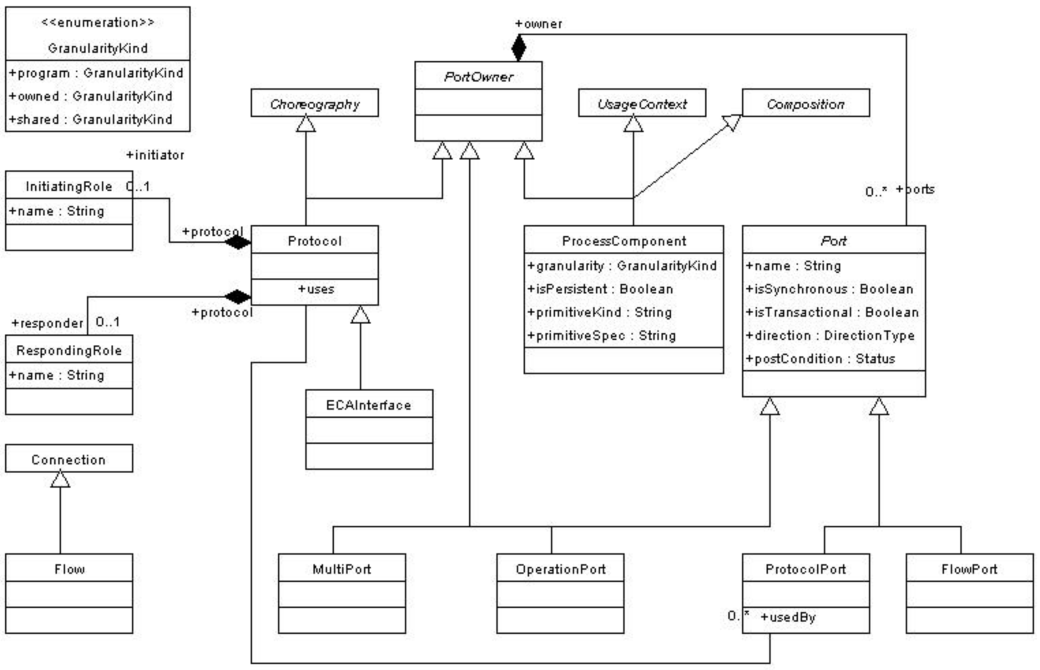


Figure 4.3 CCA profile

In WSDL the *Definition* element acts as a container for the service description. The *Import* element serves a purpose similar to the #include directive in the C/C++ programming language. It lets the modeller separate the elements of a service definition into separate documents and include them in the main document. The *Type* element acts as a container for the definition of datatypes that are used in the *Message* elements. The *Message* element is used to model the data exchanged in a web service. A message is made of several *parts*, each part having a name and a type. The *PortType* element specifies a subset of operations supported for an endpoint of a web service. The *Operation* element models an operation. A WSDL operation can have input, output, and fault messages as part of its action. The *Binding* element specifies the protocol and data format of a *PortType* element. The bindings can be standard - HTTP, SOAP, or MIME – or can be created by the user. The *Service* element typically appears at the end of a WSDL document and identifies a web service. The primary purpose of a WSDL document is to describe the abstract interface. A *Service* element is used only to describe the actual endpoint of a service. Figure 4.4 contains the WSDL model.

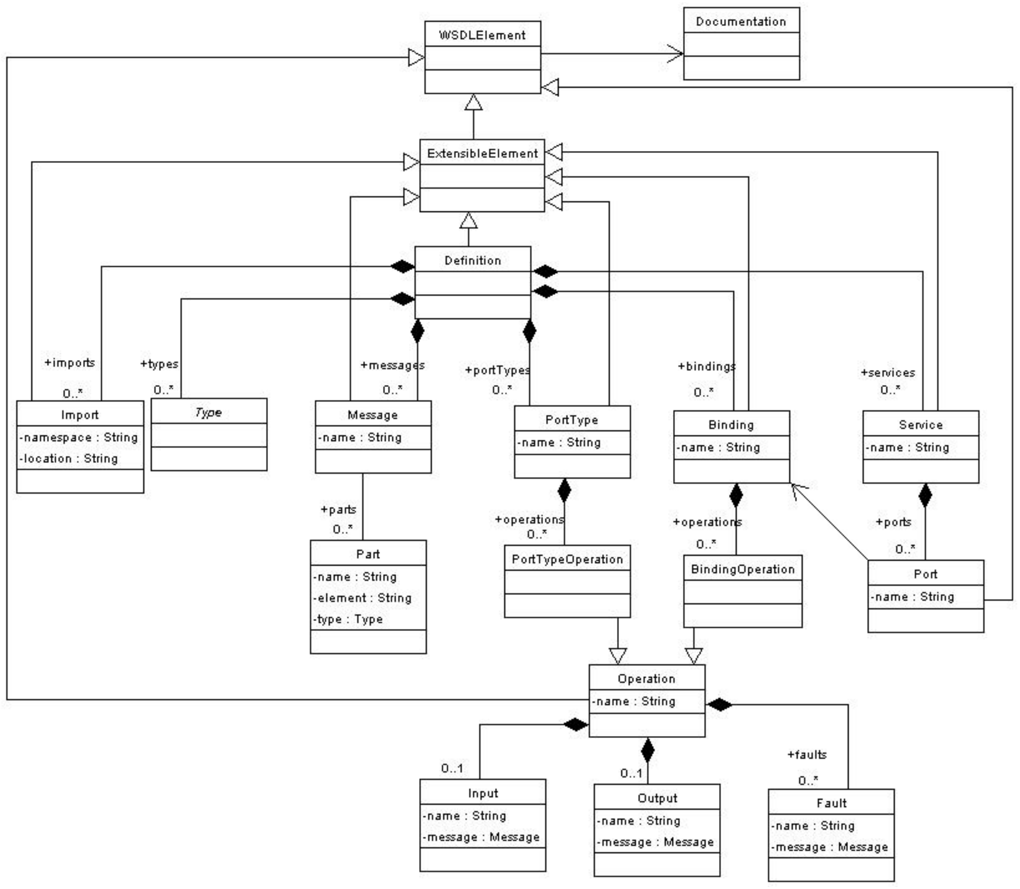


Figure 4.4 WSDL model

The transformation from CCA to WSDL obeys the well-known compositional principal of Frege [JB81], which states that "the meaning of a syntactic construct is a function of the meanings of its constituents". The transformation process and transformation rules are described in Table 4.2.

Rule name	Rule description
flowPort2message	Creates a WSDL Message for each CCA FlowPort and stores the mapping using the track mechanism.

operationPort2operation	Creates a WSDL Operation for each CCA OperationPort and stores the mapping using the track mechanism. The input and output properties of the WSDL Operation are computed using the initiator and the responder port from the OperationPort.
protocolPort2portType	Creates a WSDL PortType for each CCA ProtocolPort and stores the mapping using the track mechanism.
processComponent2service	Creates a WSDL Service for each CCA ProcessComponent and stores the mapping using the track mechanism. The definition of the dervice is instantied by this rule. The values of the properties are assigned by the other brules.
LinkDefinition2X	Computes the types, messages, and portTypes properties for every WSDL Definition. Uses the track mechanism to retrieve the mapping information stored by previous rules.
cca2wsdl	Invokes the above rules in the following order: `apply flowPort2message();` `apply operationPort2operation();` `apply protocolPort2portType();` `apply processComponent2service();` `apply linkDefinition2X();`

Table 4.2 Transformation from CCA to WDSL

4.1.5. An example

To study the mapping from EDOC to WS using YATL and YATL-Studio we consider a simplified model of a travel agency. In general a travel agency provides services such as: reserves and purchases flights and charters tickets, reserves hotel rooms, rents cars, books holidays and cruises, and sells travel insurance. To provide such services a travel agency needs to establish business links with companies such as airlines, hotels, and banks.

Figure 4.6 contains the description of a travel agency community process. The activities in the TravelAgency Community Process start by the Client initiating the interactions on its Buy ProtocolPort, according to the BuySell protocol. The TravelAgency is connected through the Sell ProtocolPort with the Client and responds to the BuySell protocol initiated by the Client. The TravelAgency uses the dedicated ports BuyFlight, ReserveRoom, RentCar, and Payment to communicate with the other processes: Airline, Hotel, CarCompany, and Bank. The TravelAgency initiates the communication through these ports, according to Client's requests. Figure 4.6 contains the description of choreographies for BuySell and BuyFlight protocols. Similar choreographies can be derived for ReserveRoom

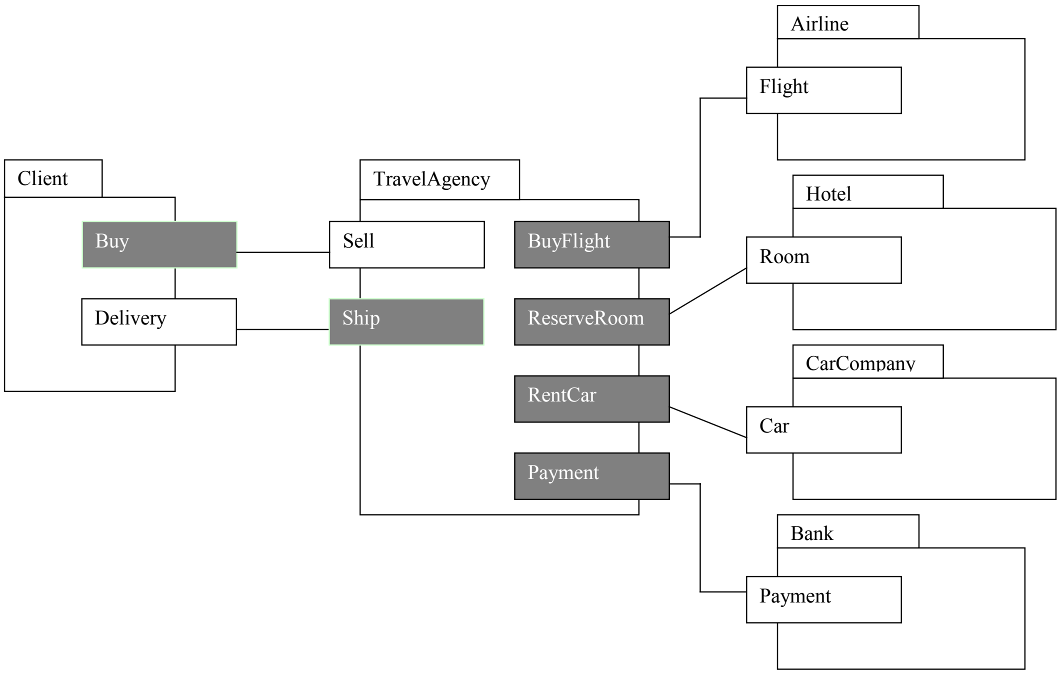
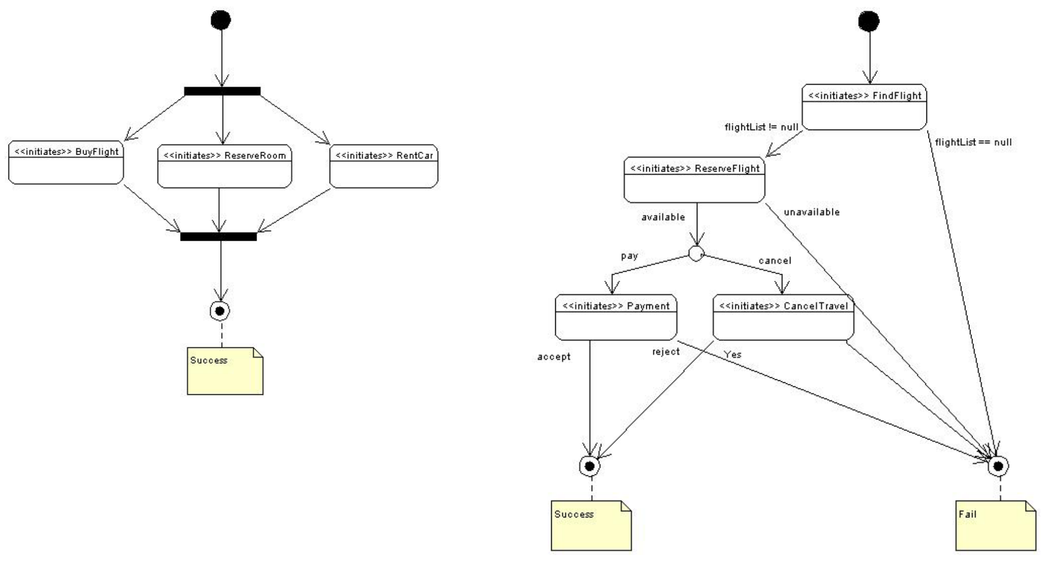


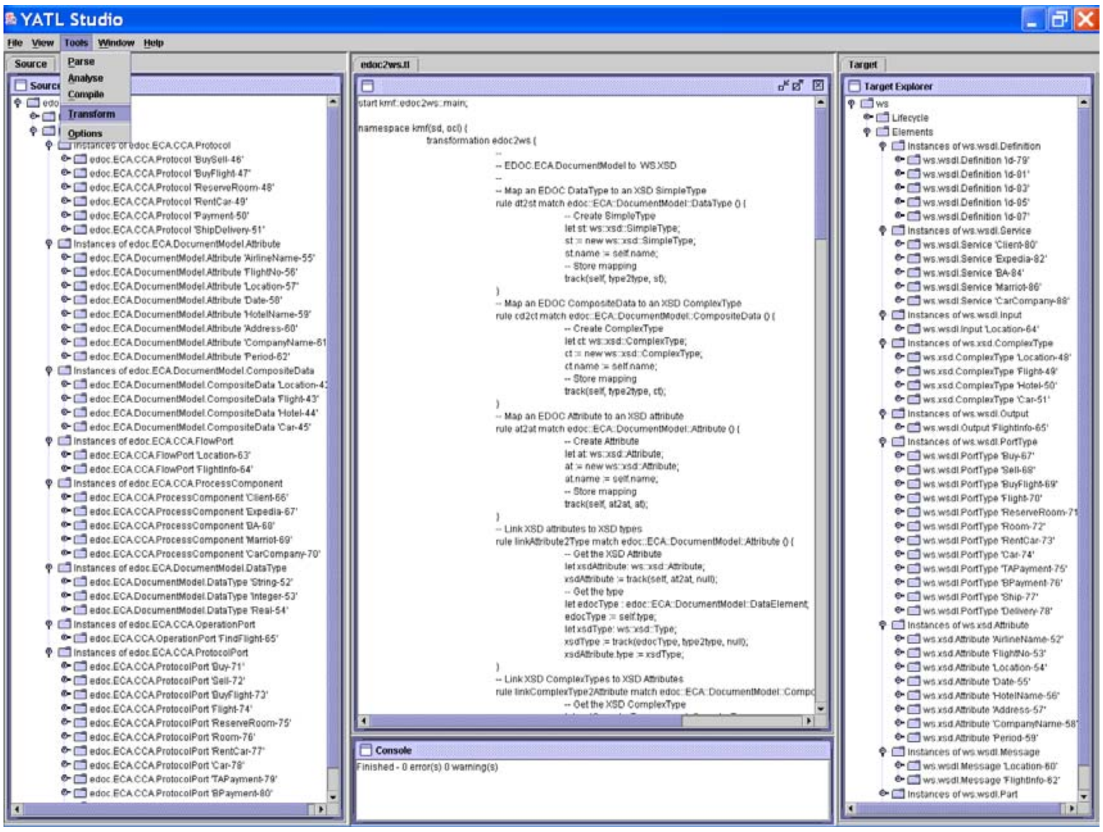
Figure 4.5 Travel agency community process



a) BuySell choreography b) BuyFlight choreography

Figure 4.6 BuySell and BuyFlight coreography

Appendix 3 contains the Java code that has been used to populate a source model instance. It also contains the entire description of transformation rules. The result of the mapping performed by the YATL program from Appendix 3 over this source model instance is described in Figure 4.7.



```
Source
edo
    Instances of edoc.ECA.CCA.Protocol
        edoc.ECA.CCA.Protocol 'BuySell-46'
        edoc.ECA.CCA.Protocol 'BuyFlight-47'
        edoc.ECA.CCA.Protocol 'ReserveRoom-48'
        edoc.ECA.CCA.Protocol 'RentCar-49'
        edoc.ECA.CCA.Protocol 'Payment-50'
        edoc.ECA.CCA.Protocol 'ShipDelivery-51'
    Instances of edoc.ECA.DocumentModel.Attribute
        edoc.ECA.DocumentModel.Attribute 'AirlineName-55'
        edoc.ECA.DocumentModel.Attribute 'FlightNo-56'
        edoc.ECA.DocumentModel.Attribute 'Location-57'
        edoc.ECA.DocumentModel.Attribute 'Date-58'
        edoc.ECA.DocumentModel.Attribute 'HotelName-59'
        edoc.ECA.DocumentModel.Attribute 'Address-60'
        edoc.ECA.DocumentModel.Attribute 'CompanyName-61
        edoc.ECA.DocumentModel.Attribute 'Period-62'
    Instances of edoc.ECA.DocumentModel.CompositeData
        edoc.ECA.DocumentModel.CompositeData 'Location-4
        edoc.ECA.DocumentModel.CompositeData 'Flight-43'
        edoc.ECA.DocumentModel.CompositeData 'Hotel-44'
        edoc.ECA.DocumentModel.CompositeData 'Car-45'
    Instances of edoc.ECA.CCA.FlowPort
        edoc.ECA.CCA.FlowPort 'Location-63'
        edoc.ECA.CCA.FlowPort 'FlightInfo-64'
    Instances of edoc.ECA.CCA.ProcessComponent
        edoc.ECA.CCA.ProcessComponent 'Client-66'
        edoc.ECA.CCA.ProcessComponent 'Expedia-67'
        edoc.ECA.CCA.ProcessComponent 'BA-68'
        edoc.ECA.CCA.ProcessComponent 'Marriot-69'
        edoc.ECA.CCA.ProcessComponent 'CarCompany-70'
    Instances of edoc.ECA.DocumentModel.DataType
        edoc.ECA.DocumentModel.DataType 'String-52'
        edoc.ECA.DocumentModel.DataType 'Integer-53'
        edoc.ECA.DocumentModel.DataType 'Real-54'
    Instances of edoc.ECA.CCA.OperationPort
        edoc.ECA.CCA.OperationPort 'FindFlight-65'
    Instances of edoc.ECA.CCA.ProtocolPort
        edoc.ECA.CCA.ProtocolPort 'Buy-71'
        edoc.ECA.CCA.ProtocolPort 'Sell-72'
        edoc.ECA.CCA.ProtocolPort 'BuyFlight-73'
        edoc.ECA.CCA.ProtocolPort 'Flight-74'
        edoc.ECA.CCA.ProtocolPort 'ReserveRoom-75'
        edoc.ECA.CCA.ProtocolPort 'Room-76'
        edoc.ECA.CCA.ProtocolPort 'RentCar-77'
        edoc.ECA.CCA.ProtocolPort 'Car-78'
        edoc.ECA.CCA.ProtocolPort 'TAPayment-79'
        edoc.ECA.CCA.ProtocolPort 'BPayment-80'
```

```
start kmf::edoc2ws::main;

namespace kmf(sd, ocl) {
            transformation edoc2ws {
                        --
                        -- EDOC.ECA.DocumentModel to WS.XSD
                        --
                        -- Map an EDOC DataType to an XSD SimpleType
                        rule dt2st match edoc::ECA::DocumentModel::DataType () {
                                    -- Create SimpleType
                                    let st: ws::xsd::SimpleType;
                                    st := new ws::xsd::SimpleType;
                                    st.name := self.name;
                                    -- Store mapping
                                    track(self, type2type, st);
                        }
                        -- Map an EDOC CompositeData to an XSD ComplexType
                        rule cd2ct match edoc::ECA::DocumentModel::CompositeData () {
                                    -- Create ComplexType
                                    let ct: ws::xsd::ComplexType;
                                    ct := new ws::xsd::ComplexType;
                                    ct.name := self.name;
                                    -- Store mapping
                                    track(self, type2type, ct);
                        }
                        -- Map an EDOC Attribute to an XSD attribute
                        rule at2at match edoc::ECA::DocumentModel::Attribute () {
                                    -- Create Attribute
                                    let at: ws::xsd::Attribute;
                                    at := new ws::xsd::Attribute;
                                    at.name := self.name;
                                    -- Store mapping
                                    track(self, at2at, at);
                        }
                        -- Link XSD attributes to XSD types
                        rule linkAttribute2Type match edoc::ECA::DocumentModel::Attribute () {
                                    -- Get the XSD Attribute
                                    let xsdAttribute: ws::xsd::Attribute;
                                    xsdAttribute := track(self, at2at, null);
                                    -- Get the type
                                    let edocType : edoc::ECA::DocumentModel::DataElement;
                                    edocType := self.type;
                                    let xsdType: ws::xsd::Type;
                                    xsdType := track(edocType, type2type, null);
                                    xsdAttribute.type := xsdType;
                        }
                        -- Link XSD ComplexTypes to XSD Attributes
                        rule linkComplexType2Attribute match edoc::ECA::DocumentModel::Compo
                                    -- Get the XSD ComplexType
```

```
Finished - 0 error(s) 0 warning(s)
```

```
ws
    Lifecycle
    Elements
        Instances of ws.wsdl.Definition
            ws.wsdl.Definition 'Id-79'
            ws.wsdl.Definition 'Id-81'
            ws.wsdl.Definition 'Id-83'
            ws.wsdl.Definition 'Id-85'
            ws.wsdl.Definition 'Id-87'
        Instances of ws.wsdl.Service
            ws.wsdl.Service 'Client-80'
            ws.wsdl.Service 'Expedia-82'
            ws.wsdl.Service 'BA-84'
            ws.wsdl.Service 'Marriot-86'
            ws.wsdl.Service 'CarCompany-88'
        Instances of ws.wsdl.Input
            ws.wsdl.Input 'Location-64'
        Instances of ws.xsd.ComplexType
            ws.xsd.ComplexType 'Location-48'
            ws.xsd.ComplexType 'Flight-49'
            ws.xsd.ComplexType 'Hotel-50'
            ws.xsd.ComplexType 'Car-51'
        Instances of ws.wsdl.Output
            ws.wsdl.Output 'FlightInfo-65'
        Instances of ws.wsdl.PortType
            ws.wsdl.PortType 'Buy-67'
            ws.wsdl.PortType 'Sell-68'
            ws.wsdl.PortType 'BuyFlight-69'
            ws.wsdl.PortType 'Flight-70'
            ws.wsdl.PortType 'ReserveRoom-71
            ws.wsdl.PortType 'Room-72'
            ws.wsdl.PortType 'RentCar-73'
            ws.wsdl.PortType 'Car-74'
            ws.wsdl.PortType 'TAPayment-75'
            ws.wsdl.PortType 'BPayment-76'
            ws.wsdl.PortType 'Ship-77'
            ws.wsdl.PortType 'Delivery-78'
        Instances of ws.xsd.Attribute
            ws.xsd.Attribute 'AirlineName-52'
            ws.xsd.Attribute 'FlightNo-53'
            ws.xsd.Attribute 'Location-54'
            ws.xsd.Attribute 'Date-55'
            ws.xsd.Attribute 'HotelName-56'
            ws.xsd.Attribute 'Address-57'
            ws.xsd.Attribute 'CompanyName-58'
            ws.xsd.Attribute 'Period-59'
        Instances of ws.wsdl.Message
            ws.wsdl.Message 'Location-60'
            ws.wsdl.Message 'FlightInfo-62'
        Instances of ws.wsdl.Part
```

Figure 4.7 Mapping the travel agency model to a WS model

4.2. Conclusions

We have learned a lot during this work. The experiments forced us to add new features to YATL and improve the implementation, especially the mapping from spider diagrams to OCL because is not a conventional mapping from a visual language to a textual language. YATL is still evolving because one of our main goals is to make it complaint to the QVT standard. But we also hope to add many original features to the YATL development environment and to integrate it with KMF and EMF

Appendix 1. MAPPING FROM UML MODEL TO JAVA MODEL

```
start kmf::uml2java::main;

namespace kmf(uml, java) {
  transformation uml2java {
    -- 1-1 Mappings
    -- Map a UML package to a Java package
    rule umlPkg2JavaPkg
         match uml::Model_Management::Package () {
     -- Create Java package
     let jPkg: javaModel::JavaPackage;
     jPkg := new javaModel::JavaPackage;
     -- Set name
     jPkg.name := self.name.body_;
     -- Store mapping
     track(self, pkg2pkg, jPkg);
    }

    -- Map a UML class to a Java class
    rule umlClass2JavaClass
         match uml::Foundation::Core::Class () {
     -- Create Java class
     let jClass: javaModel::JavaClass;
     jClass := new javaModel::JavaClass;
     -- Set name
     jClass.name := self.name.body_;
     -- Store mapping
     track(self, class2class, jClass);
    }

    -- Map a UML attribute to a Java field
    rule umlAttribute2JavaField
         match uml::Foundation::Core::Attribute () {
     -- Create a Java Field
     let jField: javaModel::JavaField;
     jField := new javaModel::JavaField;
     -- Set name
     jField.name := self.name.body_;
     -- Store mapping
     track(self, attribute2field, jField);
    }

     -- Map a UML association end to a Java field
    rule umlAssociationEnd2JavaField
         match uml::Foundation::Core::AssociationEnd (){
     -- Create the Java field
     let jField: javaModel::JavaField;
     jField := new javaModel::JavaField;
     -- Set name
     iff self.name.oclIsUndefined() then
      jField.name := self.type.name.body_;
     else
      jField.name := self.name.body_;
     endif
     -- Store mapping
     track(self, associationEnd2field, jField);
    }
```

```
-- Map a UML method to a Java operation
rule umlOperation2JavaMethod
     match uml::Foundation::Core::Operation () {
 -- Create a Java Method
 let jMethod: javaModel::JavaMethod;
 jMethod := new javaModel::JavaMethod;
 -- Set name
 jMethod.name := self.name.body_;
 -- Store mapping
 track(self, operation2method, jMethod);
}

-- Link all the elements to the corresponding package
rule linkElements2Pkg
     match uml::Model_Management::Package () {
 -- Get the corresponding JavaPackage
 let jPkg: javaModel::JavaPackage;
 jPkg = track(self, pkg2pkg, null);
 -- For each owned element
 foreach e:uml::Foundation::Core::Classifier
       in self.ownedElement do {
  -- Get the Java classifier
  let jCls: javaModel::JavaClassifier;
  jCls := track(e, class2class, null);
  jPkg.elements := jPkg.elements->including(jCls);
 }
}

-- Link all the fields to the corresponding class
rule linkAttribute2Class
     match uml::Foundation::Core::Attribute () {
 -- Get the Java Class that owns the corresponding field
 let umlOwner: uml::Foundation::Core::Classifier,
      jClass : javaModel::JavaClass;
 umlOwner := self.owner;
 jClass := track(umlOwner, class2class, null);
 -- Get the Java Field
 let jField: javaModel::JavaField;
 jField := track(self, attribute2field, null);
 -- Link field and class
 jClass.fields := jClass.fields->including(jField);
 jField.javaClass := jClass;
}
rule linkAssociationEnd2Class
     match uml::Foundation::Core::AssociationEnd () {
 -- Get the AssociationEnds
 let ends: Set(uml::Foundation::Core::AssociationEnd) =
             self.association.connection->asSet();
 let otherEnd: uml::Foundation::Core::AssociationEnd =
             (ends->asSet()-Set{self})->asSequence()->at(1);
 -- Get the Java Class that owns the corresponding field
 let umlOwner: uml::Foundation::Core::Classifier,
      jClass: javaModel::JavaClass;
 umlOwner := otherEnd.type;
 jClass := track(umlOwner, class2class, null);
 -- Get the Java Field
 let jField: javaModel::JavaField;
 jField := track(self, associationEnd2field, null);
 -- Link field and class
 jClass.fields := jClass.fields->including(jField);
 jField.javaClass := jClass;
}

-- Link all the operations to the corresponding class
rule linkOperation2Class
     match uml::Foundation::Core::Operation () {
 -- Get the UML Class that owns the attribute
 let umlOwner: uml::Foundation::Core::Classifier,
      jClass: javaModel::JavaClass;
      umlOwner := self.owner;
      jClass := track(umlOwner, class2class, null);
      -- Get the Java Method
      let jMethod: javaModel::JavaMethod;
      jMethod := track(self, operation2field, null);
      -- Link method and class
      jClass.methods := jClass.methods->including(jMethod);
      jMethod.javaClasses := jMethod.javaClasses->including(jClass);
     }

     -- main rule
     rule main () {
      -- Map individual elements
      apply umlPkg2JavaPkg();
      apply umlClass2JavaClass();
      apply umlAttribute2JavaField();
      apply umlAssociationEnd2JavaField();
      apply umlOperation2JavaMethod();
      -- Add element to Java packages
      apply linkElements2Pkg();
      -- Add fields to Java classes
      apply linkAttribute2Class();
      apply linkAssociationEnd2Class();
      -- Add operations to Java classes
      apply linkOperation2Class();
     }
    }
  }
```

Appendix 2. MAPPING FROM SPIDER DIAGRAMS MODEL TO OCL MODEL

Java program that populates the spider diagram model instance

```
SdRepository rep = new SdRepository$Class();
// Create contours
Contour a = (Contour)rep.buildElement("sd.as.Contour");
a.setName("a");
Contour b = (Contour)rep.buildElement("sd.as.Contour");
b.setName("b");
Contour c = (Contour)rep.buildElement("sd.as.Contour");
c.setName("c");
// Create zone (a | b)
Zone z1 = (Zone)rep.buildElement("sd.as.Zone");
z1.getContainingContours().add(a);
z1.getExcludingContours().add(b);
// Create zone (b | a)
Zone z2 = (Zone)rep.buildElement("sd.as.Zone");
z2.getContainingContours().add(b);
z2.getExcludingContours().add(a);
// Create zone (a, b |)
Zone z3 = (Zone)rep.buildElement("sd.as.Zone");
z3.getContainingContours().add(a);
z3.getContainingContours().add(b);

// Create diagram containing all the zones
UnitaryDiagram ud1 =
    (UnitaryDiagram)rep.buildElement("sd.as.UnitaryDiagram");
ud1.getZones().add(z1);
ud1.getZones().add(z2);
ud1.getZones().add(z3);
// Save repository
rep.saveXMI("src/test/scripts/sdRep.xml");
```

YATL program

```
start kmf::sd2ocl::main;

namespace kmf(sd, ocl) {
 transformation sd2ocl {
  -- 1-1 Mappings
  -- Map a SD unitary diagram to an OCL expression
  rule ud2let match sd::as::UnitaryDiagram () {
    -- Create let expression
    let letExp: syntax::ast::expressions::LetExpAS;
    letExp := new syntax::ast::expressions::LetExpAS;
    -- Store mapping
  track(self, ud2let, letExp);
}

-- Map a SD zone to a variable: init exppression computes the set
rule z2var match sd::as::Zone () {
 --
 -- Create name(zone): Set{OclAny} = OclAny.allInstances()
 -- ->select(x:OclAny | x.isKindOf() and ... and not x.isKindOf()
 -- and ... and not )
 --
 -- Create OclAny type
 let oclAnyType: syntax::ast::types::ClassifierAS;
 oclAnyType := new syntax::ast::types::ClassifierAS;
 oclAnyType.pathName := Sequence{'OclAny'};
 -- Create type Set{OclAny}
 let setType: syntax::ast::types::SetTypeAS;
 setType := new syntax::ast::types::SetTypeAS;
 setType.elementType := oclAnyType;
 -- Create pathName expression 'OclAny'
 let oclAnyPathNameExp: syntax::ast::expressions::PathNameExpAS;
 oclAnyPathNameExp := new syntax::ast::expressions::PathNameExpAS;
 oclAnyPathNameExp.pathName := Sequence{'OclAny'};
 -- Create OclAny.allInstances selection
 let allInstancesSelection:
     syntax::ast::expressions::DotSelectionExpAS;
 allInstancesSelection :=
     new syntax::ast::expressions::DotSelectionExpAS;
 allInstancesSelection.source := oclAnyPathNameExp;
 allInstancesSelection.name := 'allInstances';
 -- Create OclAny.allInstances() operation call
 let allInstancesCall:
     syntax::ast::expressions::OperationCallExpAS;
 allInstancesCall :=
     new syntax::ast::expressions::OperationCallExpAS;
 allInstancesCall.source := allInstancesSelection;
 allInstancesCall.arguments := Sequence{};
 -- Create OclAny.allInstances()->select selection
 let selectExp: syntax::ast::expressions::ArrowSelectionExpAS;
 selectExp := new syntax::ast::expressions::ArrowSelectionExpAS;
 selectExp.source := allInstancesCall;
 selectExp.name := 'select';
 -- Create x: OclAny variable declaration
 let xVar: syntax::ast::contexts::VariableDeclarationAS;
 xVar := new syntax::ast::contexts::VariableDeclarationAS;
 xVar.name := 'x';
 xVar.type := oclAnyType;
```

```
-- Create filters: isKindOf and notIsKindOf
let filters: Sequence(syntax::ast::expressions::OclExpressionAS);
filters := Sequence{};
let isKindOfSelection:
    syntax::ast::expressions::DotSelectionExpAS;
let isKindOfCall: syntax::ast::expressions::OperationCallExpAS;
let contourPathNameExp: syntax::ast::expressions::PathNameExpAS;
foreach c: sd::as::Contour in self.containingContours do {
 -- Create name(c) path name
 contourPathNameExp :=
    new syntax::ast::expressions::PathNameExpAS;
 contourPathNameExp.pathName := Sequence{c.name};
 -- Create x.isKindOf
 isKindOfSelection :=
    new syntax::ast::expressions::DotSelectionExpAS;
 isKindOfSelection.source := xVar;
 isKindOfSelection.name := 'isKindOf';
 -- Create x.isKindOf(c.name)
 isKindOfCall :=
    new syntax::ast::expressions::OperationCallExpAS;
 isKindOfCall.source := isKindOfSelection;
 isKindOfCall.arguments := Sequence{contourPathNameExp};
 -- Add it to filters
 filters := filters->including(isKindOfCall);
}
foreach c: sd::as::Contour in self.excludingContours do {
 -- Create name(c) path name
 contourPathNameExp :=
    new syntax::ast::expressions::PathNameExpAS;
 contourPathNameExp.pathName := Sequence{c.name};
 -- Create x.isKindOf
 isKindOfSelection :=
    new syntax::ast::expressions::DotSelectionExpAS;
 isKindOfSelection.source := xVar;
 isKindOfSelection.name := 'isKindOf';
 -- Create x.isKindOf(c.name)
 isKindOfCall :=
    new syntax::ast::expressions::OperationCallExpAS;
 isKindOfCall.source := isKindOfSelection;
 isKindOfCall.arguments := Sequence{contourPathNameExp};
 -- Create not x.isKindOf(c.name)
 let notSelection: syntax::ast::expressions::DotSelectionExpAS;
 notSelection := new syntax::ast::expressions::DotSelectionExpAS;
 notSelection.source := isKindOfCall;
 notSelection.name := 'not';
 let notCall: syntax::ast::expressions::OperationCallExpAS;
 notCall := new syntax::ast::expressions::OperationCallExpAS;
 notCall.source := notSelection;
 notCall.arguments := Sequence{};
 -- Add it to filters
 filters := filters->including(notCall);
}
-- Compute iterator's body
let itBody: syntax::ast::expressions::OclExpressionAS;
itBody := filters->at(1);
let i:Integer = 2;
while i <= filters->size() do {
 -- Create itBody.and
 let andSelection: syntax::ast::expressions::DotSelectionExpAS;
 andSelection := new syntax::ast::expressions::DotSelectionExpAS;
 andSelection.name := 'and';
 andSelection.source := itBody;
 -- Create itBody.and(args)
 let andCall: syntax::ast::expressions::OperationCallExpAS;
 andCall := new syntax::ast::expressions::OperationCallExpAS;
 andCall.source := andSelection;
 andCall.arguments := Sequence{filters->at(i)};
 -- Set new value for itBody
 itBody := andCall;
 -- Next filter
 i := i + 1;
}
-- Create iterator expression OclAny.allInstances()->select(...)
let iteratorExp: syntax::ast::expressions::IteratorExpAS;
iteratorExp := new syntax::ast::expressions::IteratorExpAS;
iteratorExp.source := selectExp;
iteratorExp.iterator := xVar;
iteratorExp.loopBody := itBody;
-- Compute zone's name
let zName: String = '';
foreach c: sd::as::Contour in self.containingContours do {
 zName := zName.concat(c.name);
 zName := zName.concat('_');
}
zName := zName.concat('|');
foreach c: sd::as::Contour in self.excludingContours do {
 zName := zName.concat('_');
 zName := zName.concat(c.name);
}
-- Create name(zone):Set{OclAny} :=
--    OclAny.allInstances()->select( ... )
let var: syntax::ast::contexts::VariableDeclarationAS;
```

```
 var := new syntax::ast::contexts::VariableDeclarationAS;
 var.name := zName;
 var.type := setType;
 var.initExp := iteratorExp;
 -- Store mapping
 track(self, z2var, var);
}

-- Map a SD to let's body (in expression)
rule ud2in match sd::as::UnitaryDiagram () {
 -- Make a list of conditions for each zone
 let ands: Sequence(syntax::ast::expressions::OclExpressionAS) =
     Sequence{};
 -- For each zone
 foreach z: sd::as::Zone in self.zones do {
  -- Compute the number of spiders touching the zone
  -- All spiders are single footed
  let feetNo: Integer = 0;
  foreach s: sd::as::Spider in self.spiders do {
   iff s.habitat->includes(z) then
    feetNo := feetNo + 1;
   endif
  }
  -- Compute is shaded flag
  let isShaded: Boolean = self.shadedZones->includes(z);
  -- Make the expression that checks the size
  -- name(z)->size() operator feetNo
  -- Make name(z) expression
  let varExp: syntax::ast::expressions::VariableExpAS;
  varExp := new syntax::ast::expressions::VariableExpAS;
  varExp.variableDeclarationAS := track(z, z2var, null);
  -- Make name(z)->size
  let selectExp: syntax::ast::expressions::ArrowSelectionExpAS;
  selectExp := new syntax::ast::expressions::ArrowSelectionExpAS;
  selectExp.source = varExp;
  selectExp.name := 'size';
  -- Make name(z)->size()
  let callExp: syntax::ast::expressions::OperationCallExpAS;
  callExp := new syntax::ast::expressions::OperationCallExpAS;
  callExp.source := selectExp;
  -- Make operator
  let opName: String = '>=';
  iff isShaded then
   opName := '=';
  endif
  -- Make name(z)->size() <=
  let selExp: syntax::ast::expressions::DotSelectionExpAS;
  selExp := new syntax::ast::expressions::DotSelectionExpAS;
  selExp.source := callExp;
  selExp.name := opName;
  -- Make feetName exp
  let argExp: syntax::ast::expressions::IntegerLiteralExpAS;
  argExp := new syntax::ast::expressions::IntegerLiteralExpAS;
  argExp.value := feetNo;
  -- Make name(z)->size() <= feetNo
  let relCall: syntax::ast::expressions::OperationCallExpAS;
  relCall :=  new syntax::ast::expressions::OperationCallExpAS;
  relCall.source := selExp;
  relCall.arguments := relCall.arguments->including(argExp);
  --
  -- Add exp to ands
  --
  ands := ands->including(relCall);
 }
 -- Make a logical expression from ands
 iff ands->size() >= 1 then {
  let inExp: syntax::ast::expressions::OclExpressionAS;
  inExp := ands->at(1);
  let i:Integer = 2;
  while i<=ands->size() do {
   -- Make an and
   let andSel: syntax::ast::expressions::DotSelectionExpAS;
   andSel := new syntax::ast::expressions::DotSelectionExpAS;
   andSel.source := inExp;
   andSel.name := 'and';
   let andCall: syntax::ast::expressions::OperationCallExpAS;
   andCall := new syntax::ast::expressions::OperationCallExpAS;
   andCall.source := andSel;
   andCall.arguments := andCall.arguments->including(ands->at(i));
   -- Update inExp for next iteration
   inExp := andCall;
   -- Next
   i := i+1;
  }
  -- Store mapping
  track(self, ud2in, inExp);
 }
 endif
}

-- Link let expressions to variables
rule linkLet2Variables match sd::as::UnitaryDiagram () {
     -- Get let expression
     let letExp: syntax::ast::expressions::LetExpAS;
     letExp := track(self, ud2let, null);
     -- For each zone
     foreach z: sd::as::Zone in self.zones do {
      let var:syntax::ast::contexts::VariableDeclarationAS;
      var := track(z, z2var, null);
      letExp.variables := letExp.variables->including(var);
     }
    }

    -- Link let expressions to variables
    rule linkLet2In match sd::as::UnitaryDiagram () {
     -- Get let expression
     let letExp: syntax::ast::expressions::LetExpAS;
     letExp := track(self, ud2let, null);
     -- Get in expression
     let inExp: syntax::ast::expressions::OclExpressionAS;
     inExp := track(self, ud2in, null);
     -- Link them
     letExp.inExp := inExp;
    }

    -- main rule
    rule main () {
     -- Create a let expression for each unitary diagram
     apply ud2let();
     -- Create a variable declaration for each zone
     apply z2var();
     -- Create the in expression
     apply ud2in();
     -- Link diagrams to variables
     apply linkLet2Variables();
     -- Link diagrams to in
     apply linkLet2In();
    }
   }
  }
```

Appendix 3. MAPPING FROM EDOC TO WS

Java code to populate the source model instance

```java
//
// Create EDOC population
//
protected static DataType makeDataType(Repository rep, String type) {
      DataType dt = (DataType)rep.buildElement("edoc.ECA.DocumentModel.DataType");
      dt.setName(type);
      return dt;
}
protected static Attribute makeAttribute(Repository rep, String name,
            DataElement type) {
      Attribute at = (Attribute)rep.buildElement("edoc.ECA.DocumentModel.Attribute");
      at.setName(name);
      at.setType(type);
      return at;
}
protected static CompositeData makeCompositeType(Repository rep, String name,
            List dataElements) {
      CompositeData dt =
            (CompositeData)rep.buildElement("edoc.ECA.DocumentModel.CompositeData");
      dt.setName(name);
      dt.setFeatures(dataElements);
      return dt;
}
protected static Protocol makeProtocol(Repository rep, String name) {
      Protocol p = (Protocol)rep.buildElement("edoc.ECA.CCA.Protocol");
      p.setName(name);
      return p;
}
protected static FlowPort makeFlowPort(Repository rep,String name,DataElement type) {
      FlowPort fp = (FlowPort)rep.buildElement("edoc.ECA.CCA.FlowPort");
      fp.setName(name);
      fp.setType(type);
      return fp;
}
protected static ProtocolPort makeProtocolPort(Repository rep, String name) {
      ProtocolPort pp = (ProtocolPort)rep.buildElement("edoc.ECA.CCA.ProtocolPort");
      pp.setName(name);
      return pp;
}
protected static OperationPort makeOperationPort(Repository rep, String name,
            FlowPort call, FlowPort ret) {
      OperationPort op =
            (OperationPort)rep.buildElement("edoc.ECA.CCA.OperationPort");
      op.setName(name);
      op.getPorts().add(call);
      op.getPorts().add(ret);
      return op;
}
protected static Repository initEDOCPopulation() {
      EdocRepository rep = new EdocRepository$Class();
      // Create simple types
      DataType stringType = makeDataType(rep, "String");
      DataType integerType = makeDataType(rep, "Integer");
      DataType realType = makeDataType(rep, "Real");
      // Create attributes
      Attribute airlineName = makeAttribute(rep, "AirlineName", stringType);
      Attribute flightNo = makeAttribute(rep, "FlightNo", integerType);
      Attribute location = makeAttribute(rep, "Location", stringType);
      Attribute date = makeAttribute(rep, "Date", stringType);
      Attribute hotelName = makeAttribute(rep, "HotelName", stringType);
      Attribute address = makeAttribute(rep, "Address", stringType);
      Attribute companyName = makeAttribute(rep, "CompanyName", stringType);
      Attribute period = makeAttribute(rep, "Period", integerType);
      // Create composite types
      List locationInfo = new Vector();
      locationInfo.add(location);
      locationInfo.add(date);
```

```
CompositeData locationType = makeCompositeType(rep, "Location", locationInfo);
List flightInfo = new Vector();
flightInfo.add(airlineName);
flightInfo.add(flightNo);
flightInfo.add(date);
CompositeData flightType = makeCompositeType(rep, "Flight", flightInfo);
List hotelInfo = new Vector();
hotelInfo.add(hotelName);
hotelInfo.add(address);
hotelInfo.add(date);
hotelInfo.add(period);
CompositeData hotelType = makeCompositeType(rep, "Hotel", hotelInfo);
List carInfo = new Vector();
carInfo.add(companyName);
carInfo.add(address);
carInfo.add(date);
carInfo.add(period);
CompositeData carType = makeCompositeType(rep, "Car", carInfo);
// Create BuySell protocol
Protocol buySellProt = makeProtocol(rep, "BuySell");
ProtocolPort buyPort = makeProtocolPort(rep, "Buy");
buyPort.setDirection(DirectionType$Class.Initiates);
buyPort.setOwner(buySellProt);
buyPort.setUses(buySellProt);
ProtocolPort sellPort = makeProtocolPort(rep, "Sell");
sellPort.setDirection(DirectionType$Class.Responds);
sellPort.setOwner(buySellProt);
sellPort.setUses(buySellProt);
buySellProt.getPorts().add(buyPort);
buySellProt.getPorts().add(sellPort);
// Create BuyFlight protocol
Protocol buyFlightProt = makeProtocol(rep, "BuyFlight");
ProtocolPort buyFlightPort = makeProtocolPort(rep, "BuyFlight");
buyFlightPort.setDirection(DirectionType$Class.Initiates);
buyFlightPort.setOwner(buyFlightProt);
buyFlightPort.setUses(buyFlightProt);
ProtocolPort flightPort = makeProtocolPort(rep, "Flight");
flightPort.setDirection(DirectionType$Class.Responds);
flightPort.setOwner(buyFlightProt);
flightPort.setUses(buyFlightProt);
buyFlightProt.getPorts().add(buyFlightPort);
buyFlightProt.getPorts().add(flightPort);
// Add operation protocols
FlowPort locationPort = makeFlowPort(rep, "Location", locationType);
locationPort.setDirection(DirectionType$Class.Initiates);
FlowPort flightFlowPort = makeFlowPort(rep, "FlightInfo", flightType);
flightFlowPort.setDirection(DirectionType$Class.Responds);
OperationPort    findFlightPort    =    makeOperationPort(rep,    "FindFlight",
locationPort, flightFlowPort);
buyFlightProt.getPorts().add(findFlightPort);
// Create reserveRoom protocol
Protocol reserveRoomProt = makeProtocol(rep, "ReserveRoom");
ProtocolPort reserveRoomPort = makeProtocolPort(rep, "ReserveRoom");
reserveRoomPort.setDirection(DirectionType$Class.Initiates);
reserveRoomPort.setOwner(reserveRoomProt);
reserveRoomPort.setUses(reserveRoomProt);
ProtocolPort roomPort = makeProtocolPort(rep, "Room");
roomPort.setDirection(DirectionType$Class.Responds);
roomPort.setOwner(reserveRoomProt);
roomPort.setUses(reserveRoomProt);
reserveRoomProt.getPorts().add(reserveRoomPort);
reserveRoomProt.getPorts().add(roomPort);
// Create rentCar protocol
Protocol rentCarProt = makeProtocol(rep, "RentCar");
ProtocolPort rentCarPort = makeProtocolPort(rep, "RentCar");
rentCarPort.setDirection(DirectionType$Class.Initiates);
rentCarPort.setOwner(rentCarProt);
rentCarPort.setUses(rentCarProt);
ProtocolPort carPort = makeProtocolPort(rep, "Car");
carPort.setDirection(DirectionType$Class.Responds);
carPort.setOwner(rentCarProt);
carPort.setUses(rentCarProt);
rentCarProt.getPorts().add(rentCarPort);
rentCarProt.getPorts().add(carPort);
// Create payment protocol
Protocol paymentProt = makeProtocol(rep, "Payment");
ProtocolPort taPaymentPort = makeProtocolPort(rep, "TAPayment");
taPaymentPort.setDirection(DirectionType$Class.Initiates);
taPaymentPort.setOwner(paymentProt);
taPaymentPort.setUses(paymentProt);
ProtocolPort bPaymentPort = makeProtocolPort(rep, "BPayment");
bPaymentPort.setDirection(DirectionType$Class.Responds);
bPaymentPort.setOwner(paymentProt);
bPaymentPort.setUses(paymentProt);
paymentProt.getPorts().add(taPaymentPort);
paymentProt.getPorts().add(bPaymentPort);
// Create ShipDelivery protocol
```

```
        Protocol shipDeliveryProt = makeProtocol(rep, "ShipDelivery");
        ProtocolPort shipPort = makeProtocolPort(rep, "Ship");
        shipPort.setDirection(DirectionType$Class.Initiates);
        shipPort.setOwner(shipDeliveryProt);
        shipPort.setUses(shipDeliveryProt);
        ProtocolPort deliveryPort = makeProtocolPort(rep, "Delivery");
        deliveryPort.setDirection(DirectionType$Class.Responds);
        deliveryPort.setOwner(shipDeliveryProt);
        deliveryPort.setUses(shipDeliveryProt);
        shipDeliveryProt.getPorts().add(shipPort);
        shipDeliveryProt.getPorts().add(deliveryPort);
        // Create Client
        ProcessComponent client =
              (ProcessComponent)rep.buildElement("edoc.ECA.CCA.ProcessComponent");
        client.setName("Client");
        client.getPorts().add(buyPort);
        client.getPorts().add(deliveryPort);
        buyPort.setOwner(client);
        deliveryPort.setOwner(client);
        // Create Travel Agency
        ProcessComponent travelAgency =
              (ProcessComponent)rep.buildElement("edoc.ECA.CCA.ProcessComponent");
        travelAgency.setName("Expedia");
        travelAgency.getPorts().add(sellPort);
        travelAgency.getPorts().add(buyFlightPort);
        travelAgency.getPorts().add(findFlightPort);
        travelAgency.getPorts().add(reserveRoomPort);
        travelAgency.getPorts().add(rentCarPort);
        travelAgency.getPorts().add(taPaymentPort);
        sellPort.setOwner(travelAgency);
        buyFlightPort.setOwner(travelAgency);
        reserveRoomPort.setOwner(travelAgency);
        rentCarPort.setOwner(travelAgency);
        taPaymentPort.setOwner(travelAgency);
        // Create Airline
        ProcessComponent airline =
          (ProcessComponent)rep.buildElement("edoc.ECA.CCA.ProcessComponent");
        airline.setName("BA");
        airline.getPorts().add(flightPort);
        // Create Hotel
        ProcessComponent hotel =
              (ProcessComponent)rep.buildElement("edoc.ECA.CCA.ProcessComponent");
        hotel.setName("Marriot");
        hotel.getPorts().add(roomPort);
        // Create CarCompany
        ProcessComponent carCompany =
              (ProcessComponent)rep.buildElement("edoc.ECA.CCA.ProcessComponent");
        carCompany.setName("CarCompany");
        carCompany.getPorts().add(carPort);
        // Save repository into an xml
        rep.saveXMI("src/test/scripts/edocRep.xml");
        return rep;
}
```

The YATL program

```
start kmf::edoc2ws::main;

namespace kmf(sd, ocl) {
  transformation edoc2ws {
    --
    -- EDOC.ECA.DocumentModel to  WS.XSD
    --
    -- Map an EDOC DataType to an XSD SimpleType
    rule dt2st match edoc::ECA::DocumentModel::DataType () {
      -- Create SimpleType
      let st: ws::xsd::SimpleType;
      st := new ws::xsd::SimpleType;
      st.name := self.name;
  -- Store mapping
  track(self, type2type, st);
}
-- Map an EDOC CompositeData to an XSD ComplexType
rule cd2ct match edoc::ECA::DocumentModel::CompositeData () {
  -- Create ComplexType
  let ct: ws::xsd::ComplexType;
  ct := new ws::xsd::ComplexType;
  ct.name := self.name;
  -- Store mapping
  track(self, type2type, ct);
}
-- Map an EDOC Attribute to an XSD attribute
rule at2at match edoc::ECA::DocumentModel::Attribute () {
  -- Create Attribute
  let at: ws::xsd::Attribute;
  at := new ws::xsd::Attribute;
  at.name := self.name;
  -- Store mapping
  track(self, at2at, at);
}
-- Link XSD attributes to XSD types
rule linkAttribute2Type
     match edoc::ECA::DocumentModel::Attribute () {
  -- Get the XSD Attribute
  let xsdAttribute: ws::xsd::Attribute;
  xsdAttribute := track(self, at2at, null);
  -- Get the type
  let edocType : edoc::ECA::DocumentModel::DataElement;
  edocType := self.type;
  let xsdType: ws::xsd::Type;
  xsdType := track(edocType, type2type, null);
  xsdAttribute.type := xsdType;
}
-- Link XSD ComplexTypes to XSD Attributes
rule linkComplexType2Attribute
     match edoc::ECA::DocumentModel::CompositeData () {
  -- Get the XSD ComplexType
  let xsdComplexType: ws::xsd::ComplexType;
  xsdComplexType := track(self, type2type, null);
  -- Add every attribute
  foreach edocAttribute : edoc::ECA::DocumentModel::Attribute
          in self.features do {
    let xsdAttribute : ws::xsd::Attribute;
    xsdAttribute := track(edocAttribute, at2at, null);
    xsdComplexType.sequence :=
      xsdComplexType.sequence->including(xsdAttribute);
    }
  }
  -- Map concepts from EDOC.ECA.DocumentModel to WS.XSD concepts
  rule documentModel2xsd() {
    -- Create a SimpleType for each DataType
    apply dt2st();
    -- Create a ComplexType for each CompositeData
    apply cd2ct();
    -- Create an XSD Attribute for each EDOC Attribute
    apply at2at();
    -- Link XSD Attributes to XSD Types
    apply linkAttribute2Type();
    -- Link XSD ComplexTypes to XSD Attributes
    apply linkComplexType2Attribute();
  }

  --
  -- Map concepts from EDOC.ECA.CCA to WS:WSDL
  --
  -- Create a WSDL Message for each EDOC FlowPort
  rule flowPort2message match edoc::ECA::CCA::FlowPort () {
    -- Create Message
    let m: ws::wsdl::Message;
    m := new ws::wsdl::Message;
    m.name := self.name;
    -- Create part and add it
    let part: ws::wsdl::Part;
    part := new ws::wsdl::Part;
    part.name := self.name;
    part.type := track(self.type, type2type, null);
    m.parts := m.parts->including(part);
    -- Store mapping
    track(self, fp2m, m);
  }
  -- Create a WSDL Operation for each EDOC OperationPort
  rule operationPort2operation
      match edoc::ECA::CCA::OperationPort () {
    -- Get input and output port
    let iPort : edoc::ECA::CCA::OperationPort;
    iPort := self.ports->asSequence()->at(1);
    let oPort : edoc::ECA::CCA::OperationPort;
    oPort := self.ports->asSequence()->at(2);
    -- Create input
    let input: ws::wsdl::Input;
    input := new ws::wsdl::Input;
    input.name := iPort.name;
    input.message := track(iPort, fp2m, null);
    -- Create outpout
    let output: ws::wsdl::Output;
    output := new ws::wsdl::Output;
    output.name := oPort.name;
    output.message := track(oPort, fp2m, null);
    -- Create Operation
    let o: ws::wsdl::Operation;
    o := new ws::wsdl::Operation;
    o.name := self.name;
    o.input := input;
    o.output := output;
    input.operation := o;
    output.operation := o;
    -- Store mapping
    track(self, op2o, o);
  }
  -- Create a WSDL PortType for each EDOC ProtocolPort
  rule protocolPort2portType
       match edoc::ECA::CCA::ProtocolPort () {
    -- Create a portType
    let pt: ws::wsdl::PortType;
    pt := new ws::wsdl::PortType;
    pt.name := self.name;
    -- Add operations
    let ps: Set(edoc::ECA::CCA::Port) = self.owner.ports->asSet();
    let fps: Set(edoc::ECA::CCA::Port) =
       ps->select(e | e.oclIsKindOf(edoc::ECA::CCA::FlowPort));
    let ops: Set(edoc::ECA::CCA::Port) =
       ps->select(e|.oclIsKindOf(edoc::ECA::CCA::OperationPort));
    foreach op: edoc::ECA::CCA::OperationPort in ops do {
      -- Find operation
      let o: ws::wsdl::Operation;
      o := track(op, op2o, null);
      pt.operations := pt.operations->including(o);
    }
    -- Store mapping
    track(self, pp2pt, pt);
  }
  -- Create a WSDL Definition for each EDOC ProcessComponent
  rule processComponent2service
       match edoc::ECA::CCA::ProcessComponent () {
    -- Create Definition
    let d: ws::wsdl::Definition;
    d := new ws::wsdl::Definition;
    -- Create service
    let s: ws::wsdl::Service;
    s := new ws::wsdl::Service;
    s.definition := d;
    s.name := self.name;
    -- Store mapping
    track(self, pc2s, s);
  }
  -- Link Definition to Types
  rule linkDefinition2X
      match edoc::ECA::CCA::ProcessComponent () {
    -- Get the WSDL Service
    let s: ws::wsdl::Service;
    s := track(self, pc2s, null);
    let d : ws::wsdl::Definition;
    d := s.definition;
    -- Add every portType
    let ps : Set(edoc::ECA::CCA::Port) = self.ports->asSet();
    let fps: Set(edoc::ECA::CCA::Port) =
       ps->select(e | e.oclIsKindOf(edoc::ECA::CCA::FlowPort));
    let ops: Set(edoc::ECA::CCA::Port) =
       ps->select(e|e.oclIsKindOf(edoc::ECA::CCA::OperationPort));
    let pps: Set(edoc::ECA::CCA::Port) =
       ps->select(e|e.oclIsKindOf(edoc::ECA::CCA::ProtocolPort));
    let m: ws::wsdl::Message;
    let ms: Set(ws::wsdl::Message);
    let ts: Set(ws::xsd::Type);
    foreach fp : edoc::ECA::CCA::FlowPort in fps do {
      m := track(fp, fp2m, null);
      ms := ms->including(m);
      foreach p:ws::wsdl::Part in m.parts do {
        ts := ts->including(p.type);
      }
    }
    foreach op : edoc::ECA::CCA::OperationPort in ops do {
      -- Get input and output port
      let iPort : edoc::ECA::CCA::OperationPort;
      iPort := op.ports->asSequence()->at(1);
      let oPort : edoc::ECA::CCA::OperationPort;
      oPort := op.ports->asSequence()->at(2);
      m := track(iPort, fp2m, null);
      ms := ms->including(m);
      foreach p:ws::wsdl::Part in m.parts do {
        ts := ts->including(p.type);
      }
      m := track(oPort, fp2m, null);
```

```
        ms := ms->including(m);
        foreach p:ws::wsdl::Part in m.parts do {
          ts := ts->including(p.type);
        }
      }
      let pts : Set(ws::wsdl::PortType);
      foreach pp : edoc::ECA::CCA::ProtocolPort in pps do {
        let pt : ws::wsdl::PortType;
        pt := track(pp, pp2pt, null);
        pts := pts->including(pt);
      }
      d.messages := ms->asSequence();
      d.types := ts->asSequence();
      d.portTypes := pts->asSequence();
    }
    --- Map CCA to WSDL
    rule cca2wsdl() {
      -- Create a WSDL Message for each EDOC FlowPort
      apply flowPort2message();
      -- Map Operation Ports
      apply operationPort2operation();
      -- Map Protocol Ports
      apply protocolPort2portType();
      -- Map ProcessComponent
      apply processComponent2service();
      -- Link Definition to types, messages, and portTypes
      apply linkDefinition2X();
    }

    -- main rule
    rule main () {
      -- Map DocumentModel to XSD
      apply documentModel2xsd();
      -- ECA to WSLD
      apply cca2wsdl();
    }
  }
}
```

BIBLIOGRAPHY

[AP03] Akehurst D. and Patrascoiu O. (2003). OCL 2.0 – Implementing the Standard for Multiple Metamodels. In *OCL2.0-"Industry standard or scientific playground?" - Proceedings of the UML'03 workshop*, page 19. Electronic Notes in Theoretical Computer Science.

[ALP03] Akehurst D., Linington P., and Patrascoiu O. (2003). Technical Report No. 12-03, Computer Laboratory, University of Kent, UK.

[AKP03] Akehurst D., Kent S., and Patrascoiu O. (2003). A relational approach to defining and implementing transformations between metamodels. In *Journal of Software and Systems Modeling* (SoSym), 2(4), 215-239.

[CH03] Czarnecki K., and Helsen S. (2003). Classification of Model Transformation Approaches. In *Generative techniques in the context of MDA – Proceedings of OOPSLA 2003 workshop*.

[CWM] OMG, Common Warehouse Metamodel Specification. OMG Document formal/2003-03-02, available at http://www.omg.org/cwm.

[EMF] IBM, Eclipse Modeling Framework. http://www.eclipse.org.

[Fra03] Frankel D. S. (2003) *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley & Sons, 2003.

[GLRSW02] Gerber A., Lawley M., Raymond K., Steel J., and Wood A. (2002). Transformation: The Missing Link of MDA, in A. Corradini, H. Ehring, H. J. Kreowsky, G. Rozenberg (Eds): In *Proc. of Graph Transformation: First International Conference (ICGT 2002)*

[GHK99] Gil J., Howse J, and Kent S. (1999) Formalising Spider Diagrams, In *Proc.of IEEE Symposium on Visual Languages (VL99)*, IEEE Press, 130-137.

[Java] Java standard http://www.sun.com

[JB81] Janssen T. M. V. and van Emde Boas. (1981) *Some observations on compositional semantics*. Report 81-11. University of Amsterdam.

[KMF] Kent Modeling Framework. http://www.cs.kent.ac.uk/projects/kmf.

[MDA] MDA. Model Driven Architecture Specification. OMG document omg/03-06-01, available at. http://www.omg.org/mda.

[MOF] OMG, MOF Meta Object Facility Specification, OMG Document formal/2002-04-03, available at http://www.omg.org/mof

[OCL] OMG, OCL Object Constraint Language Specification Revised Submission, Version 1.6, January 6, 2003, OMG document ad/2003-01-07.

[OCL2P] OCL Open source project: Object Constraint Language for Kent Modeling Framework and Eclipse Framework. http://www.cs.kent.ac.uk/projects/kmf.

[OMG] OMG Object Management Group. http://www.omg.org.

[UML] OMG, Unified Modeling Language Specification, Version 1.5, 2003, OMG Document formal/2003-03-01, available at. http://www.omg.org/uml.

[QVT02] OMG, QVT Query/Views/Transformations RFP, OMG Document ad/02-04-10, revised on April 24, 202. http://www.omg.org/cgi-bin/doc?ad/2002-4-10

[QVTD] OMG, MOF Query/Views/Transformation, Initial submission, DSTC and IBM.

[QVTP] OG, MOF Query/Views/Transformation, Initial submission, QVT Partners.

[QVTF] OMG, MOF Query/Views/Transformation, Initial submission, Alcatel, SoftTeam, Thales, TNI-Valiosys.

[Pat04a] Patrascoiu O. (2004) YATL:Yet Another Transformation Language. In Proc. of First European Workshop MDA-IA, University of Twente, the Nederlands.

[Pat04b] Patrascoiu O. (2004) YATL:Yet Another Transformation Language. Reference Manual. Version 1.0. Technical Report 2-04, University of Kent, UK.

[RJB99] Rumbaugh, J., Jacobson I., and Booch G.. (1999). The Unified Modeling Language – Reference Manual. Addison-Wesley.

[SOAP] W3C, Simple Object Access Protocol http://www.w3.org/TR/soap

[UDDI] Universal Description, Discovery, and Integration http://uddi.org/specification.html

[UNI] Unicode standard. http://www.unicode.org

[XMI] OMG, MOF Meta Object Facility Specification OMG Document 2003-05-02, available at http://www.omg.org/uml

[XML] W3C, Extensible Markup Language http://www.w3.org/TR/2004/REC-xml-20040204/

[XMLS] XML Schema http://www.w3.org/XML/Schema

[WK99] Warmer, J. and Kleppe A. (1999). The Object Constraint Language: Precise Modeling with UML. Addison-Wesley.

[WIK] Wikipedia The Free Encyclopedia http://www.wikipedia.org

[WSDL] W3C, Web Service Description Language http://www.w3.org/TR/wsdl