

UML Modellierung und Model Driven Architecture (MDA) für Java mittels Rational Software Architect (RSA)



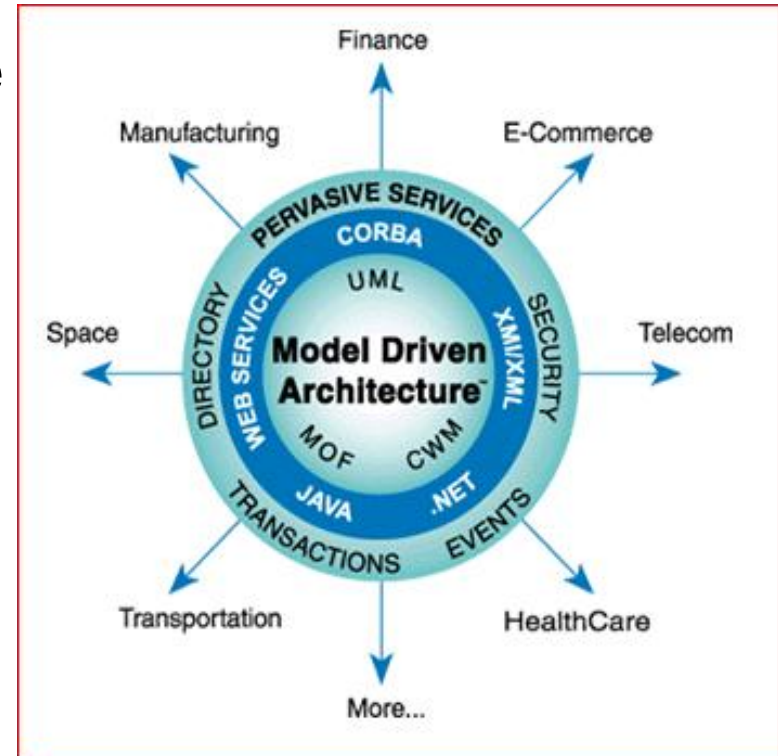
Michail Matjuchin
IBM Software Group, Rational
Austria

Rational. software

Agenda

- Was ist MDA und welche Probleme adressiert werden.
- MDD Umsetzung in RSA
 - Modellierungstechniken für MDA
 - Out-of-the-Box Transformationen
 - UML Patterns in RSA
 - Eigene Transformationen
 - Out-of-the-Box Transformationen anpassen
- Fragen

OMG Model Driven Architecture

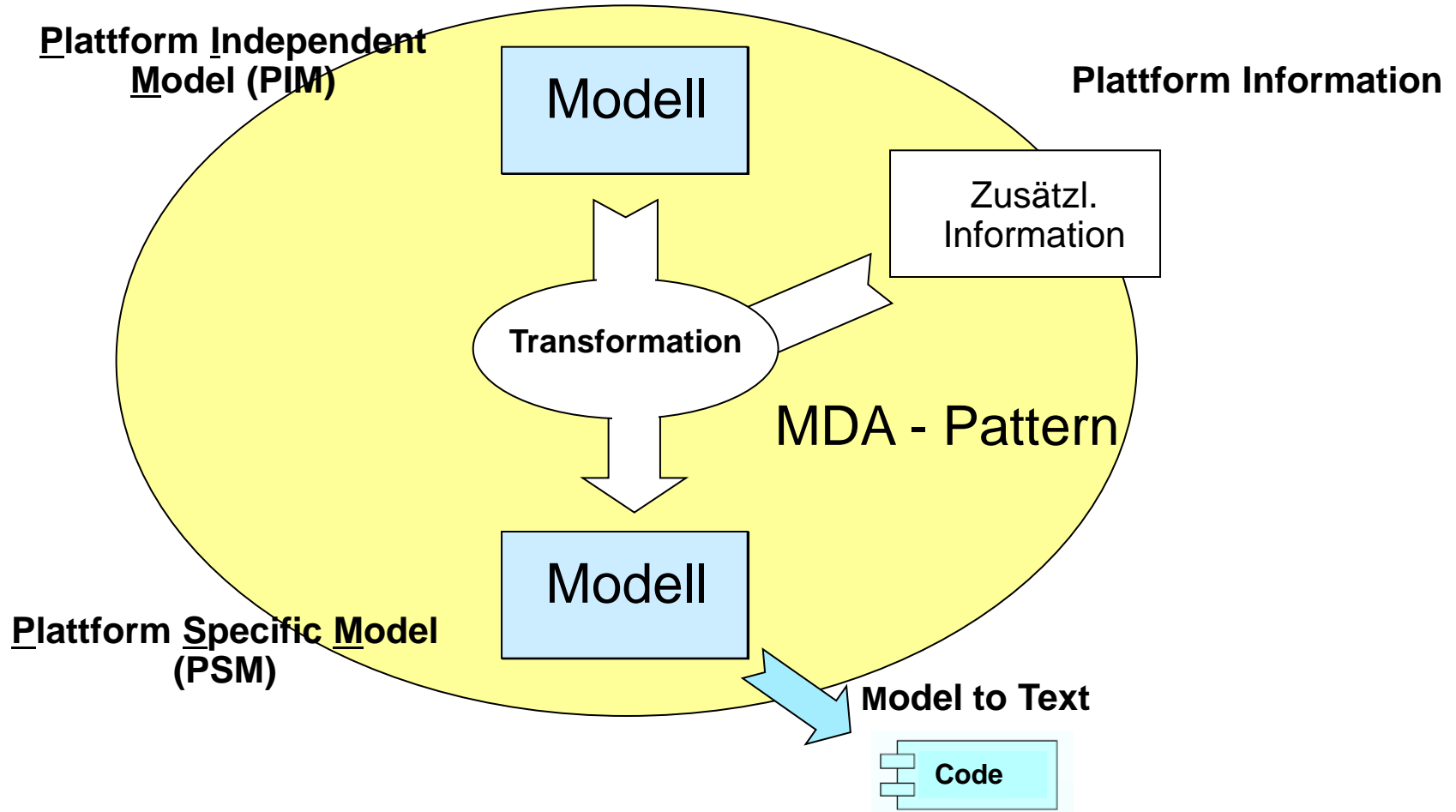


MDA ist eine Initiative der Object Management Group, 2003
(<http://www.omg.org/mda/>)

Folgende Ziele wurden dabei verfolgt:

- Verschiedene Middleware- und Implementierungstechniken, denen sich der Softwareentwickler heute stellen muss, zu integrieren und den Übergang auf neu entstehende Standards zu erleichtern.
- Der Fokus des Entwicklers sollte sich weg von der Implementierung hin zu Modellen verlagern.
- Technikunabhängige Modellierung sollte die Aufwände wesentlich verringern.

MDA-Vorgehensweise

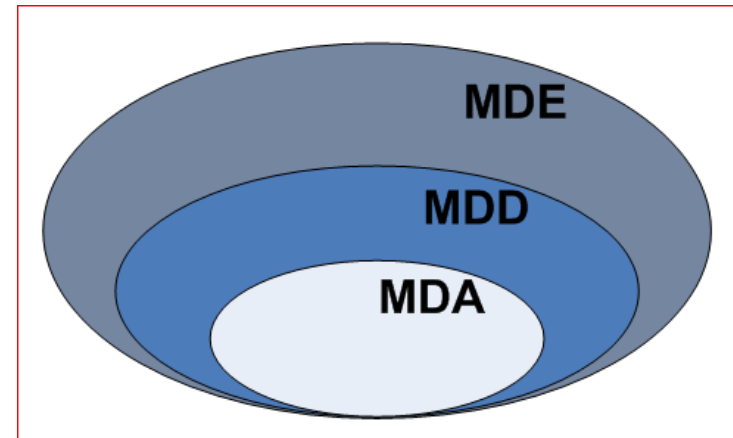


MDA Vorgehensweise verspricht folgende Vorteile:

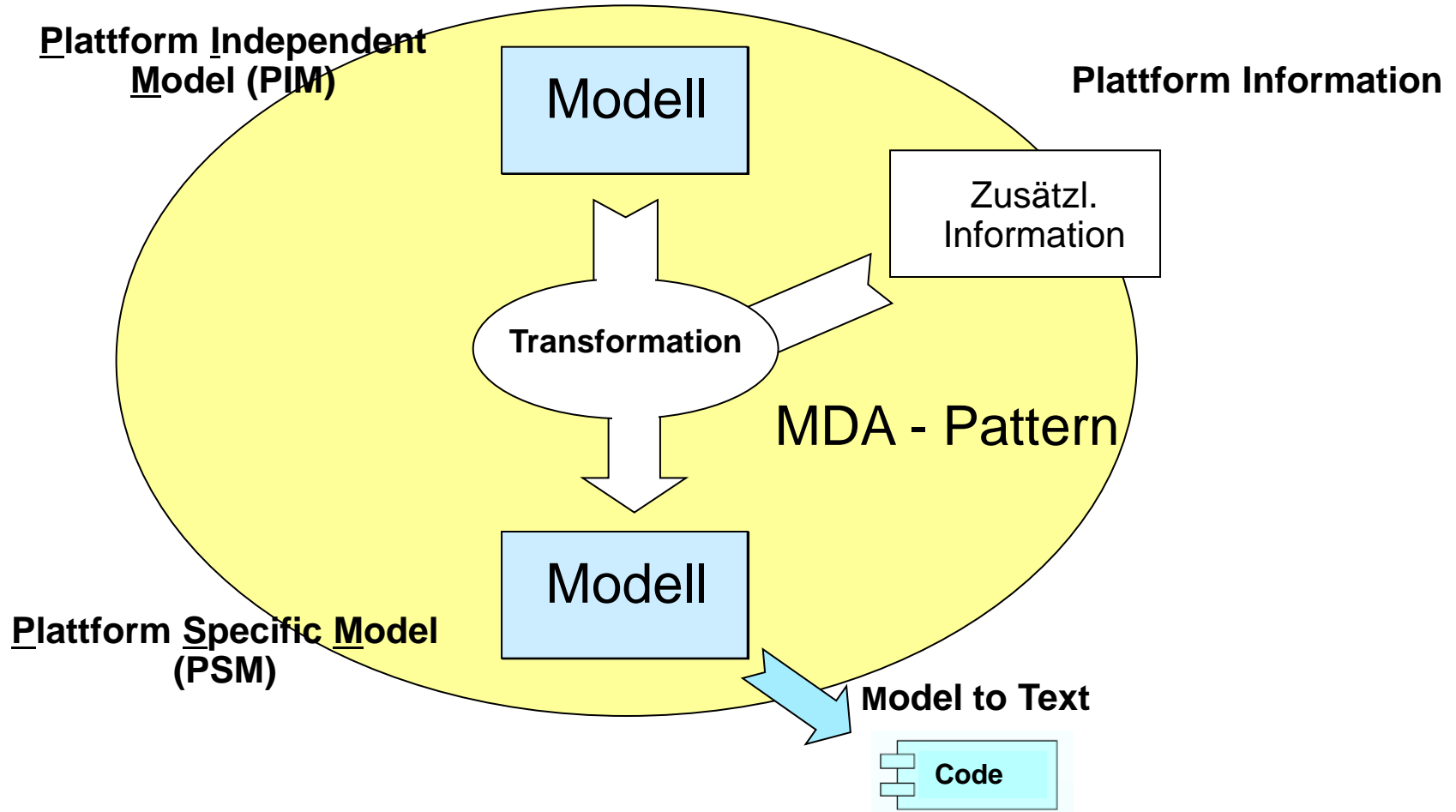
- Investitionsschutz (Fachmodelle leben länger als Technologie)
- Flexibilität
 - Trennung der Technik und Fachlichkeit
 - Technologische Änderungen einfach möglich / Plattformunabhängigkeit
- Steigerung der Produktivität
- Wiederverwendung
 - Automatisiertes / Kontrollierbares "Copy&Paste" mittels "Transformationen,,
 - Modelle sind Basis für Wiederverwendung
- Steigerung der Qualität / Reduzierung der Restfehlerrate
- Know How wird vervielfältigt (in Transformationen)
- Schnellere Einsatz neuer Technologien
- Reduzierung von Zeit, Kosten und Risiken
- Wartbarkeit

MDA, MDD und MDE

- Model Driven Architecture (MDA) ist OMG's Vision des Entwicklungsprozesses, die in mehreren Standards festgelegt ist.
- Model Driven Development (MDD) ist ein Softwareentwicklung - Paradigma, das die Modelle als primäre Artefakte bei der Softwareentwicklung sieht.
 - Üblicherweise wird die Implementierung aus den Modellen quasi-automatisch generiert.
- Model Driven Engineering (MDE): „E“ geht drüber Softwareentwicklung hinaus und umfasst andere model-basierte Tasks in dem komplettem Softwareentwicklung Prozess.

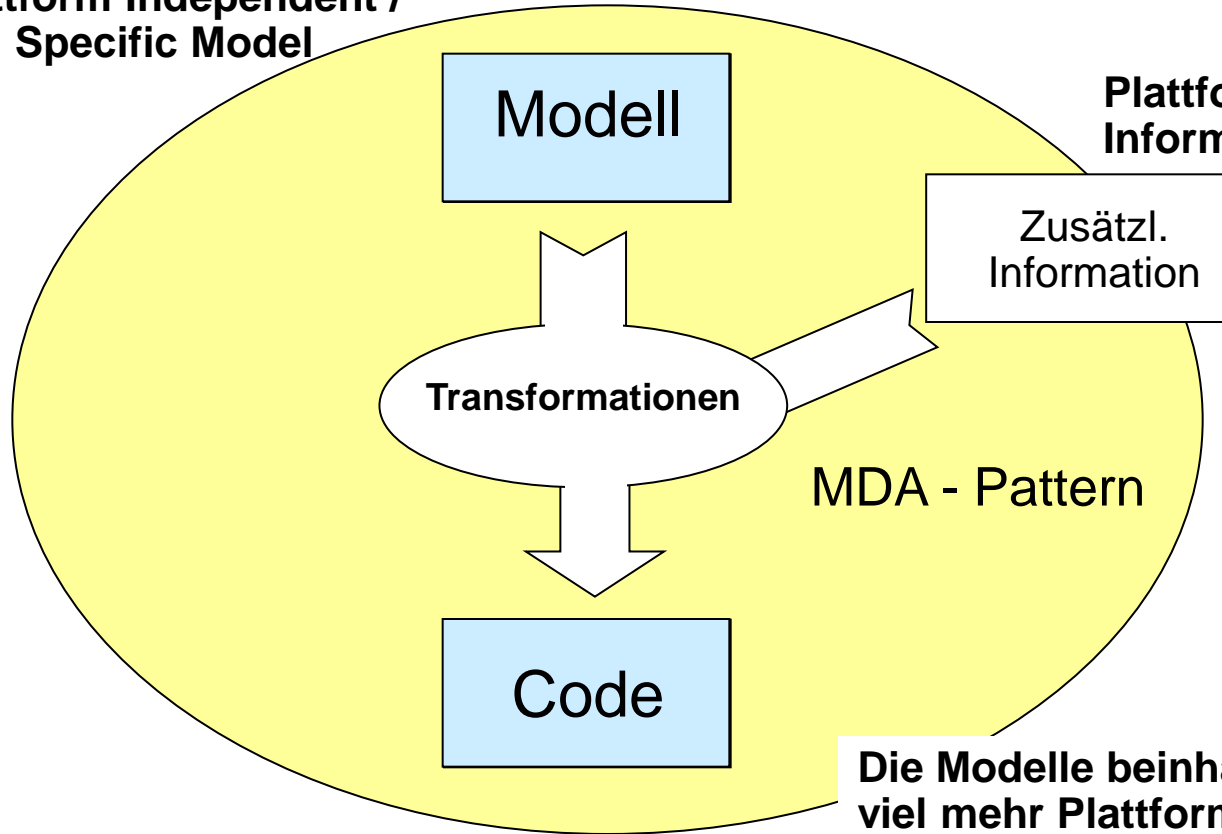


Mehrstufige Transformationen sind in MDA-Vorgehensweise vorgesehen



Im MDD wird üblicherweise nur eine Transformationsebene realisiert

Plattform Independent /
Specific Model

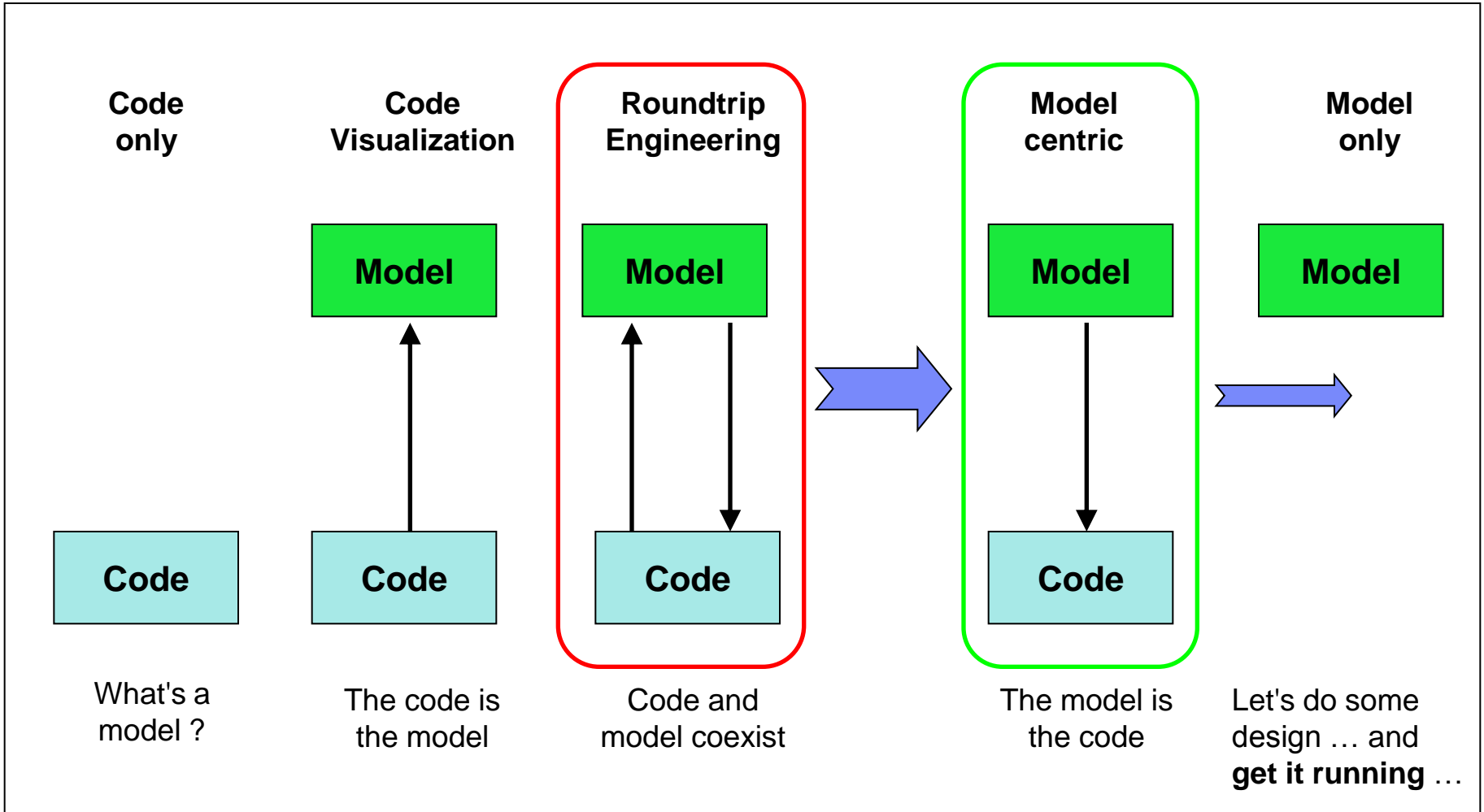


Plattform und Technologie
Informationen

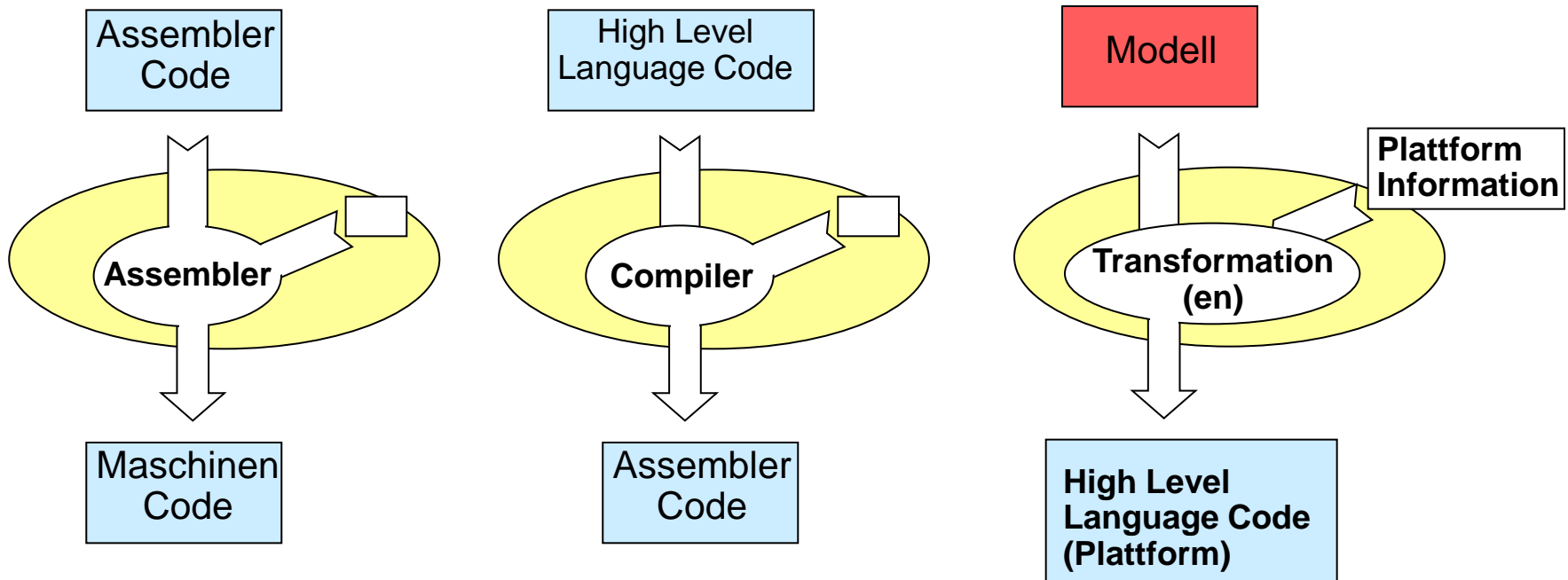
MDA - Pattern

Die Modelle beinhalten üblicherweise
viel mehr Plattform- und Technologie
Spezifikum

MDA Technologie ist evolutionär entstanden / Entwicklung des Modellierungsspektrums



Modell-Transformationen waren schon immer in der Softwareentwicklung vorhanden



MDA hat auch Mängel

- Statische und dynamische Aspekte sind für die Entwicklung sind gleich wichtig.
 - In der UML liegt der Fokus aber auf der Statik.
- Dynamisches Verhalten kann im UML zwar modelliert werden, wird aber üblicherweise sehr kompliziert.
- Entstehender Code hat eventuell mehr oder wenige große Lücken, die manuell ausprogrammiert werden müssen
- MDA Patterns, die für Code-Generierung verwendet werden, müssen erstellt und gepflegt werden

MDD Vorgehensweise ist in mehreren IBM Tools implementiert

- Allgemeiner Gebrauch
 - Rational Software Architect
 - Allgemeine MDD Unterstützung
 - Kann angepasst werden
- Domain – spezifisch
 - IBM Rational Rhapsody
 - Wird am meistens in Systems Development verwendet
 - Kann angepasst werden
- Supporting
 - WebSphere Business Modeler
 - IBM Integration Designer
 - Modellieren von Business Prozessen
 - Transformationen können nicht angepasst werden

RSA ist primäres Tool für JEE-Entwicklung

Rational Software Architect

UML 2

IBM / RSA Transformation Framework

OCL

UML2 Utilities for Model Driven Development

Patterns

Off-the-Shelf Transformations

IBM Reusable MDA Assets

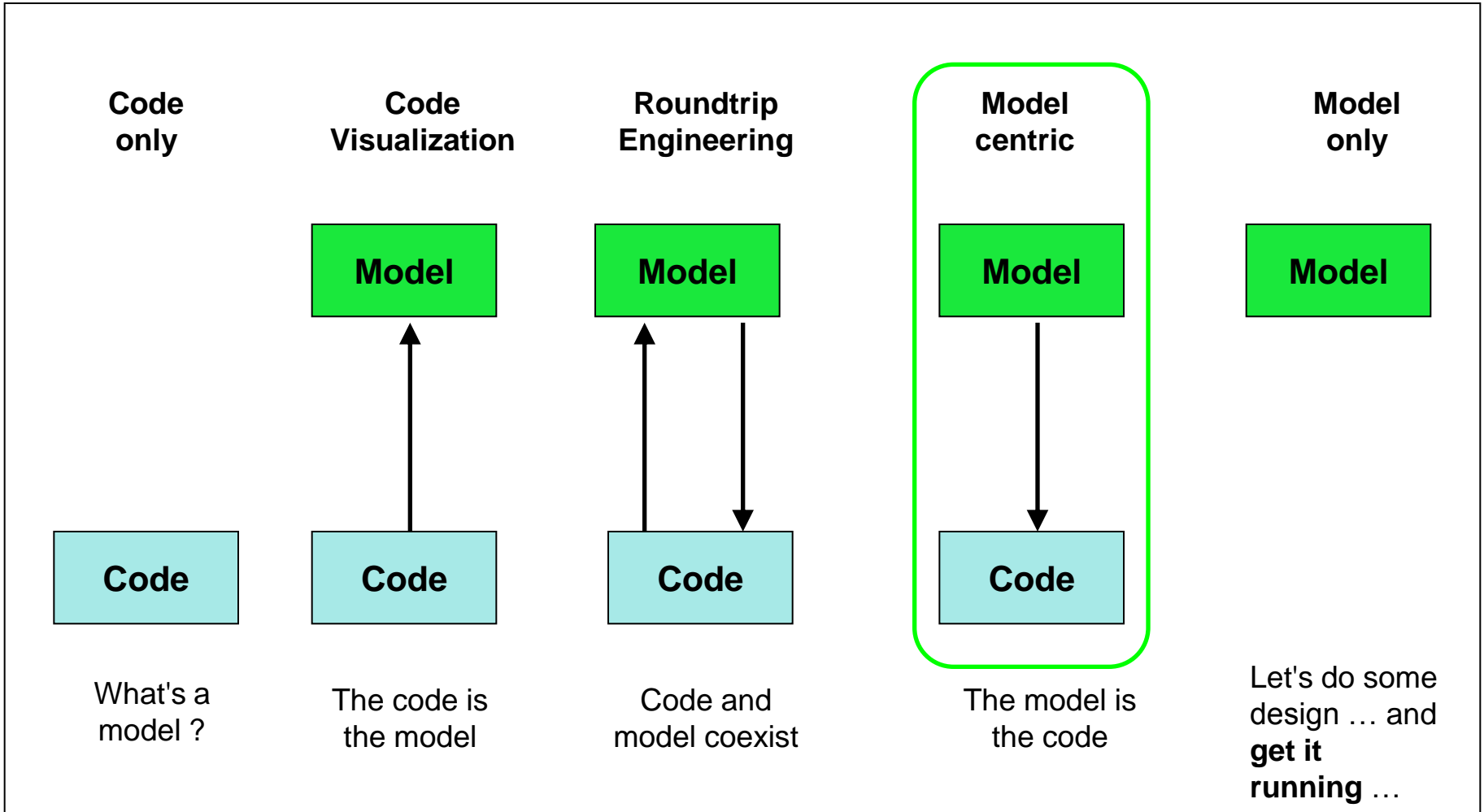
Business Prozesse

SOA

J2EE

Deployment und Security

MDD Vorgehensweise geht davon aus, dass die Modelle in Zentrum der Softwareentwicklung stehen

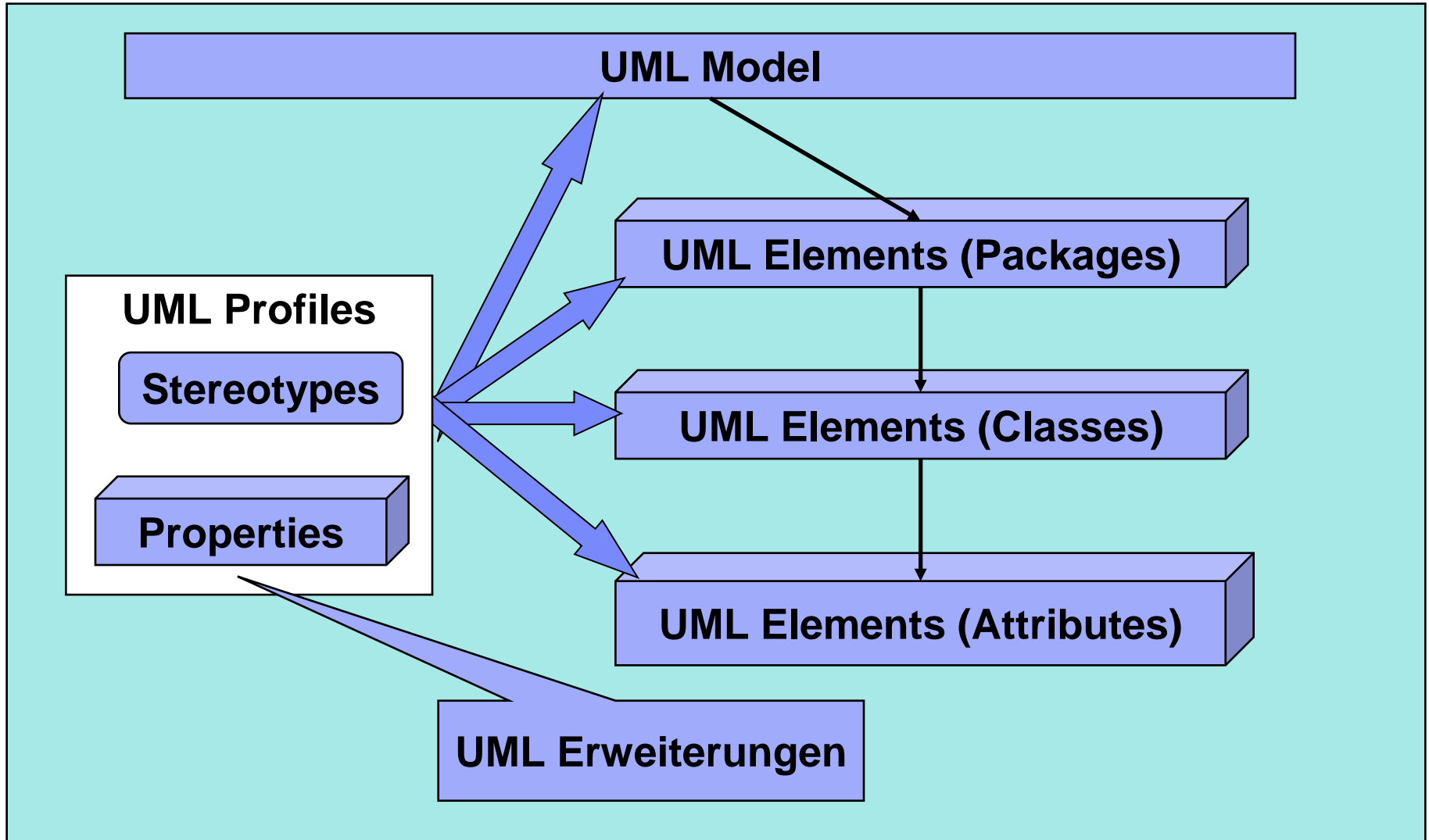


MDD stellt bestimmte Anforderungen an die Modelle

Modelle müssen detailliert genug sein

- "Full System Implementation" kann aus Modell generiert werden
- Zusätzliche Informationen sind nicht notwendig, alles ist im Modell vorhanden
- Kein Round-trip ist notwendig
- "Single Source" approach
- Modelle sind nicht "Sketches" oder "Blueprints" - sie sind "Primary Artifacts",
- The Model is the code

UML 2 mit UML-Profiles wird für die Modellierung verwendet

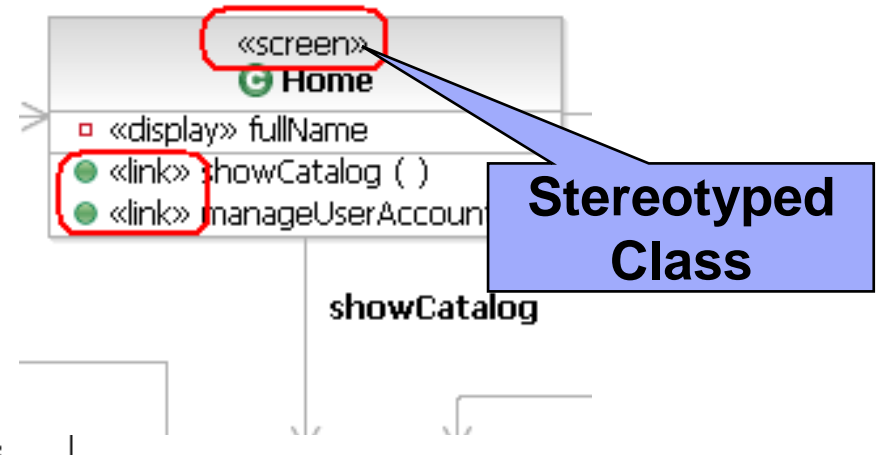
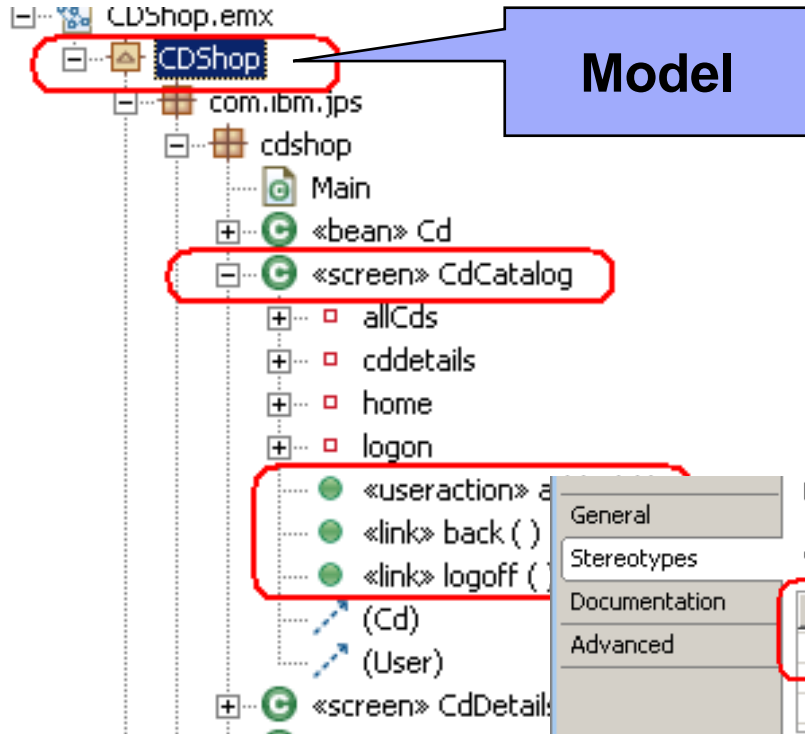


MDA Beispiel

The screenshot displays the Rational software interface with several key components:

- Model:** A tree view on the left shows a project structure. A red box highlights the package `«DalModel, EquirionBaseModel» mmdal`, which contains elements like `MoneyMarket`, `(EquirionTypes)`, `(UML2)`, `(Config)`, and `(mmdos)`.
- Stereotyped Element:** A callout box points to the `«DO» MmHyperReference` class in the central editor. Its attributes include `hyperReference : String`, `beginValueDate : Date`, `endValueDate : Date`, `maturityDate : Date`, `callFlag : boolean`, `callCode : MMCallCodeENUM`, `trading : BigDecimal`, `interestRate : BigDecimal` (highlighted), `interestMethod : BigDecimal`, `state : int`, `optLocking : boolean`, `currencyKey : String`, and `currencyBk : int`.
- Available Transformations:** A callout box points to a list of transformation options in the bottom right. The `DosModelToCodeTransformation` option is highlighted. Other options include `BLModelToCodeTransformation`, `BLModelToCodeTransformationExt`, `CalModelToCodeTransformation`, `DalModelToCodeTransformation`, `UML to C++`, `UML to CORBA`, `UML to EJB`, `UML to Java`, `UML to XSD`, `UX to JSF`, and `UX to Struts`.

Stereotypes werden verwendet, um „das Verhalten“ der Elemente zu definieren



Keywords:

Applied Stereotypes:

Stereotype	Profile	Required
inputtext	UXModeling2	False

Add Stereotypes... Remove Stereotypes...

Stereotype Properties:

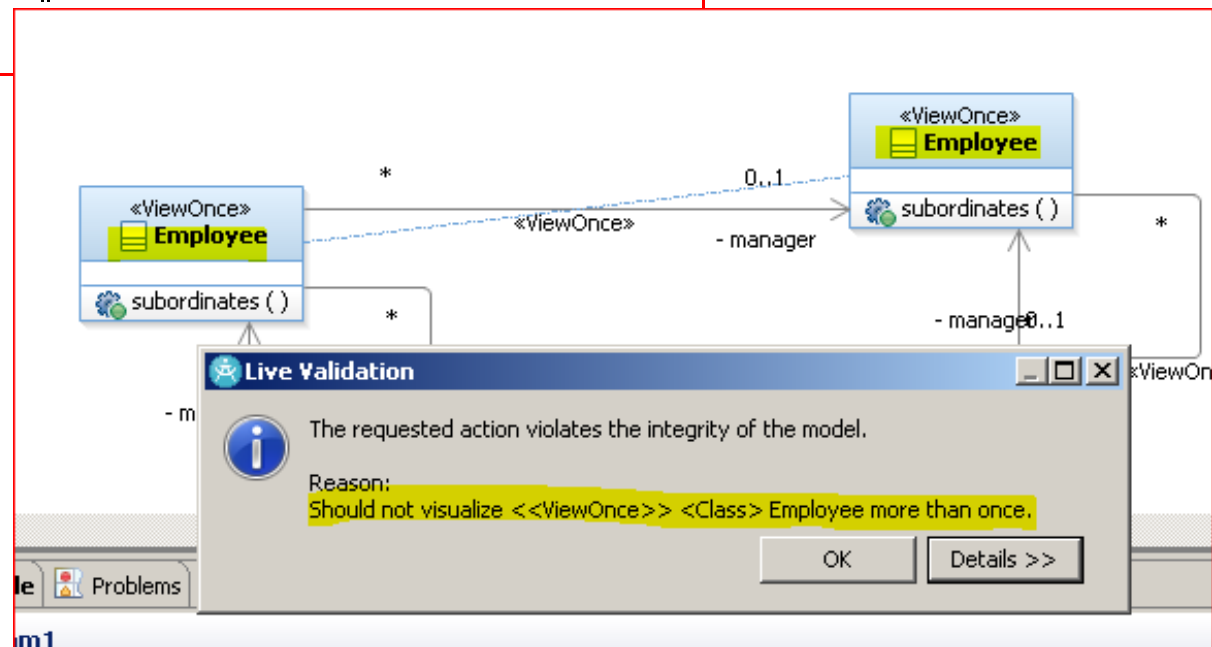
Property	Value
inputtext	
label	Please enter your u
max	
min	
password	False
required	True

Object Constraint Language (OCL) wird verwendet, um sicherzustellen, dass Modelle korrekt sind

```

-- this constraint accesses the metaclasses from the GMF Notation metamodel and uses
-- the allInstances() operation to access all node/edge views in the model
let elem : NamedElement = self.base_NamedElement, views : Set(notation::View) =
if elem.oclIsKindOf(Relationship) then
    notation::Edge.allInstances()
else
    -- consider top-level nodes only (not compartments)
    notation::Node.allInstances()->select(n | n.diagram.children->includes(n))
endif in
views->select(v | v.element = elem)->size() <= 1

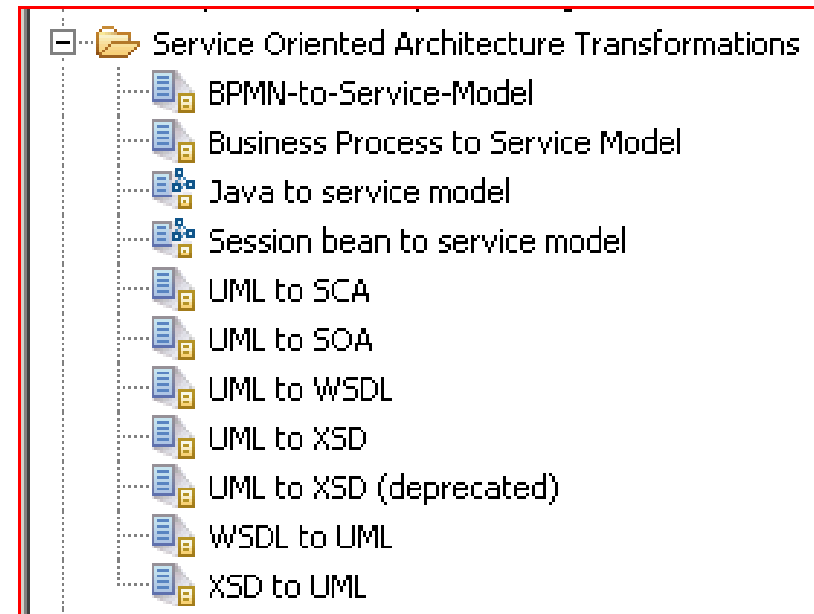
```



Mehrere Transformationen sind Out-of-the-Box vorhanden

- UML-to-EJB 3
- UML-to-JPA (Java Persistence API)
- UML-to-WSDL (Web Services)
- UML-to-SCA (Service Component Arcitecture)
- UML-to-SOA (WSDL, XSD, BPEL)
- UML-to-XSD (XML Schema Definition)

- C#, C++ und VB



Entsprechende UML-Profile und Integration mit User-Interface erleichtern die Modellierung

The image shows two screenshots from the Rational UML tool. The left screenshot, titled "Applied Profiles", lists various UML profiles applied to a model. The right screenshot shows the UML class diagram palette with two profiles expanded: "Java Persistence API Transformation" and "EJB 3.0 Transformation". Red arrows indicate the mapping from the profiles in the left screenshot to the corresponding UML elements in the right screenshot.

Applied Profiles
This section lists UML profiles applied in this model.

- Business Modeling
- Java Persistence API Transformation
- Deployment
- XSD Transformation
- JavaTransformation
- REST
- Default
- SCA Transformation Profile
- Standard
- WSDL Transformation
- EJB 3.0 Transformation
- JAX-RS
- SoaML

Java Persistence API Transformation

- <Entity> Class
- Entity From Table
- Bidirectional Association
- Generalization
- <Datasource> Actor
- <Embeddable> Class

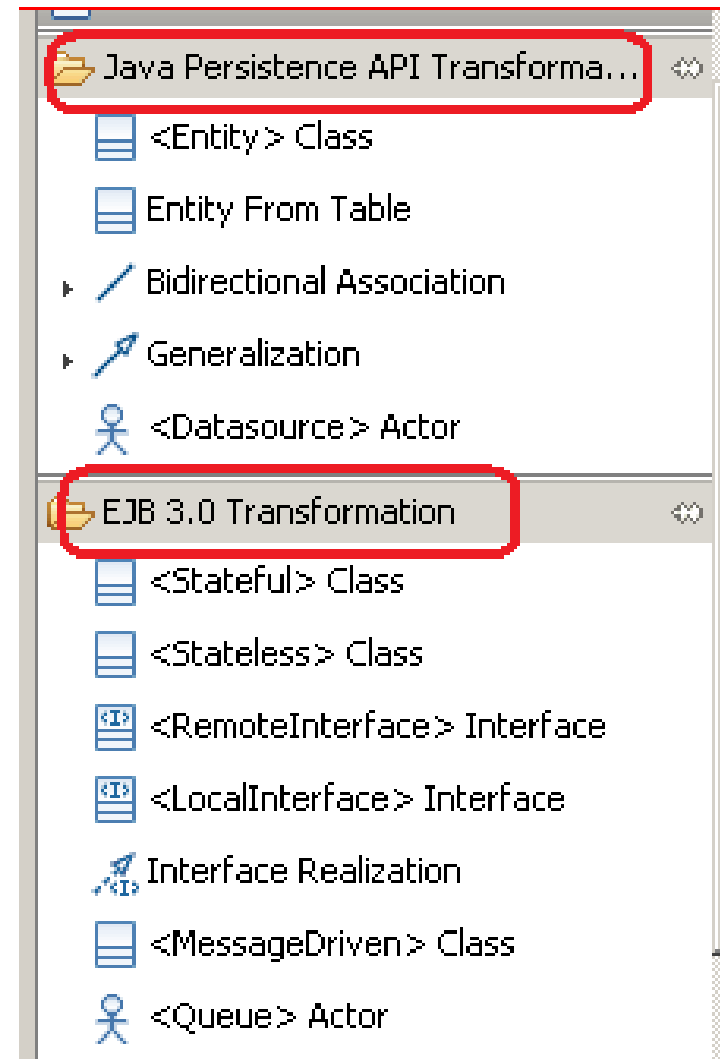
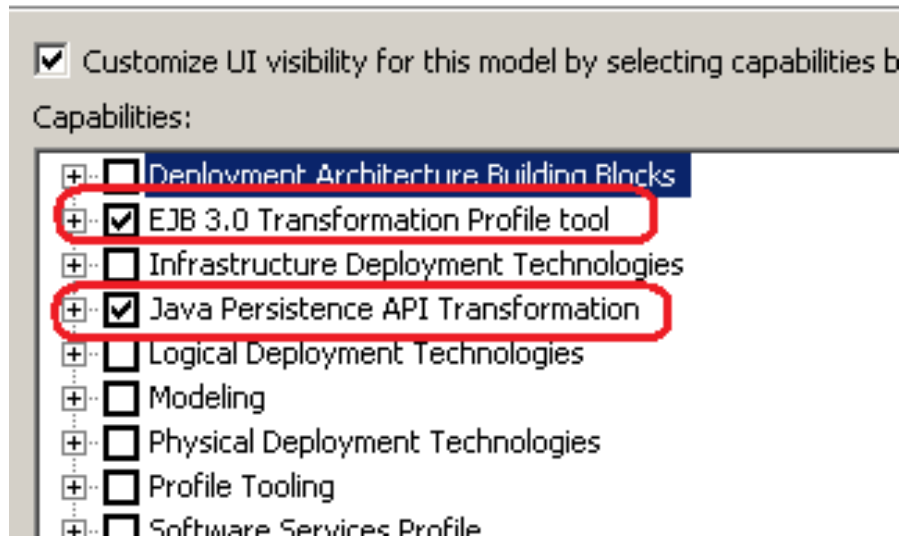
EJB 3.0 Transformation

- <Singleton> Class
- <Stateful> Class
- <Stateless> Class
- <RemoteInterface> Interface
- <LocalInterface> Interface
- Interface Realization
- <MessageDriven> Class
- <Queue> Actor

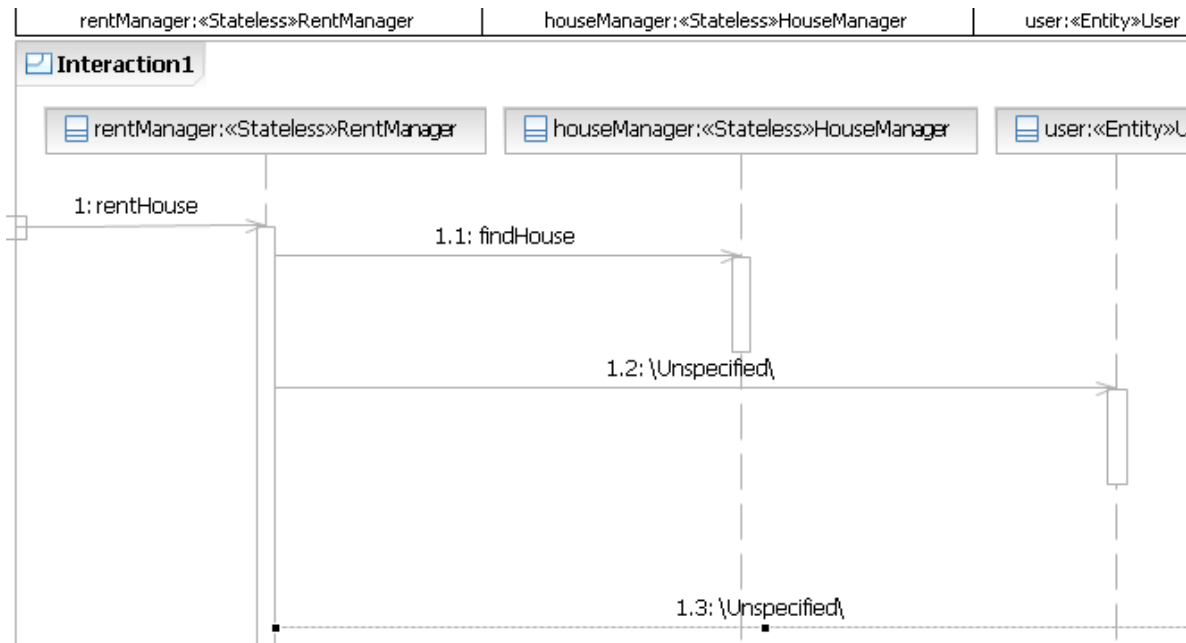
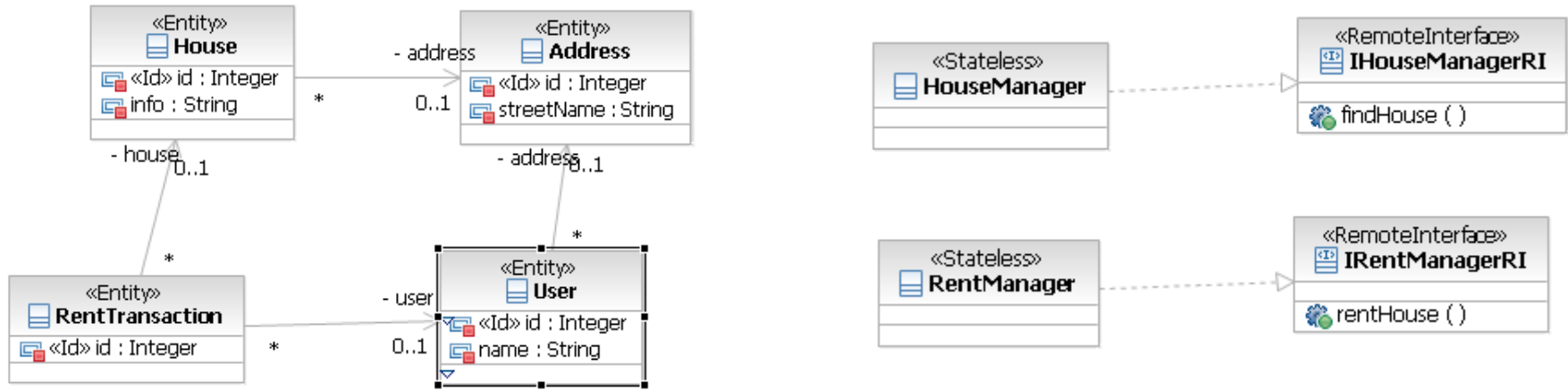
EJB 3 Transformation

Model Capabilities

Select the capabilities to associate with the new model.



EJB 3.0 Beispiel

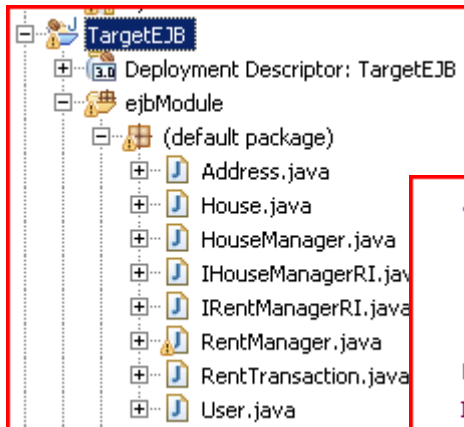


EJB 3.0 Beispiel

```

*/
@Entity
@NamedQueries( {
    @NamedQuery(name = "House.findById", query = "select obj from House where obj.id = :id"),
    @NamedQuery(name = "House.findByinfo", query = "select obj from House where obj.info = :info")
    @NamedQuery(name = "House.findByaddress", query = "select obj from House where obj.address = :")
}
)
public class House implements Serializable {
    /**

```



```

/**
 * <!-- begin-UML-doc -->
 * <!-- end-UML-doc -->
 * @generated "UML-to-EJB 3.0"
 */
@ManyToOne
private Address address;

/**
 * @return the address
 * @generated "UML-to-EJB 3.0"
 */
public Address getAddress() {
    // begin-user-code
    return address;
    // end-user-code
}

```

```

public void rentHouse() {
    //IHouseManagerRI.findHouse();
    //TODO Add method parameters

    Integer id = null; //TODO initialize and set the v
    User entity = entityManager.find(User.class, id);

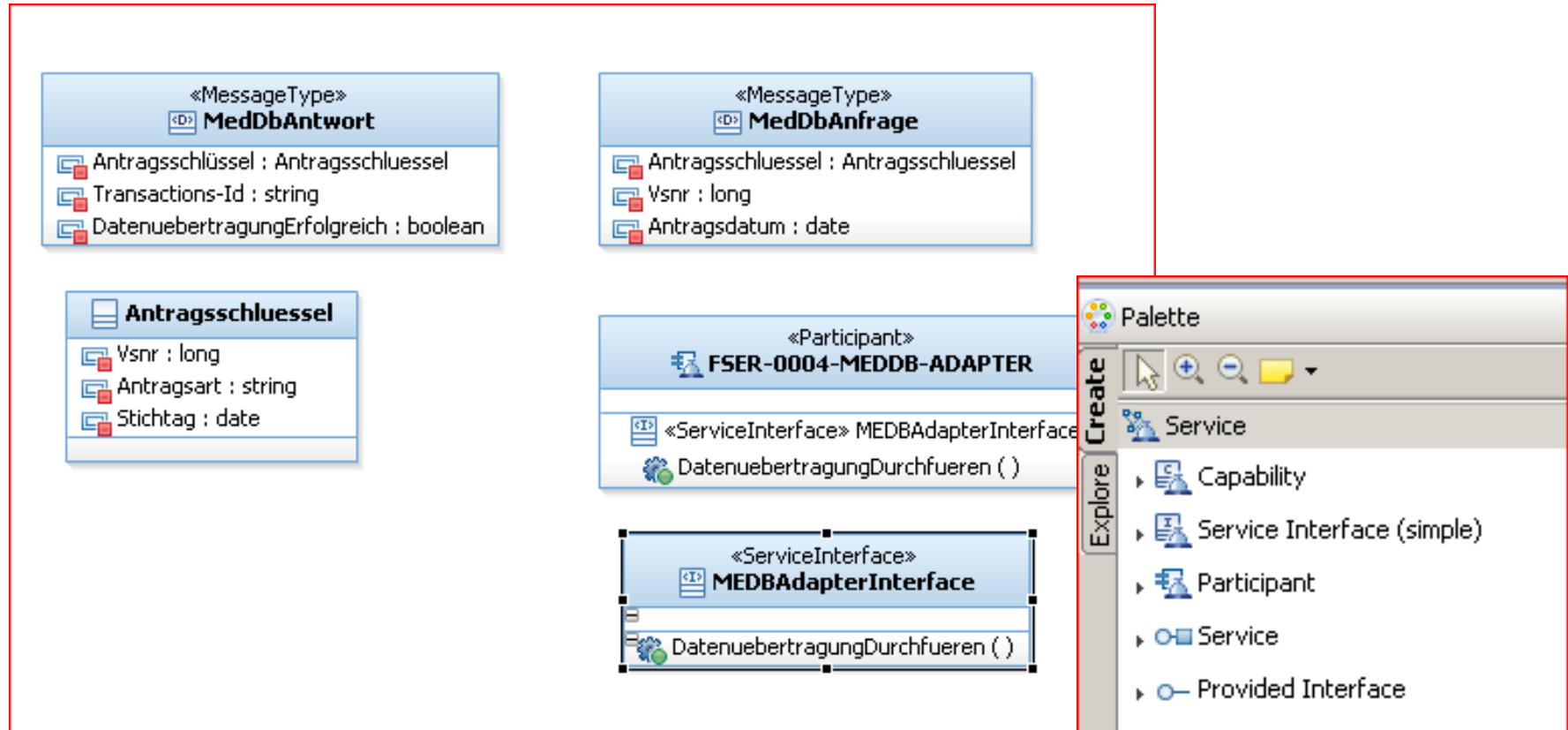
    RentTransaction entity1 = new RentTransaction();
    //TODO set the entity properties
    entityManager.persist(entity1);

    // begin-user-code
    // TODO Auto-generated method stub

    // end-user-code
}

```

Web Services Transformation



Palette

Tools: Mouse, Zoom In, Zoom Out, Refresh

Create

- Service

Explore

- Capability
- Service Interface (simple)
- Participant
- Service
- Provided Interface
- Part
- Service Channel
- Message Type (Data Type)

- Services Architecture
- Service Contract

Web Services Transformation

UML to WSDL Transformation: sample_conf.tc ▾

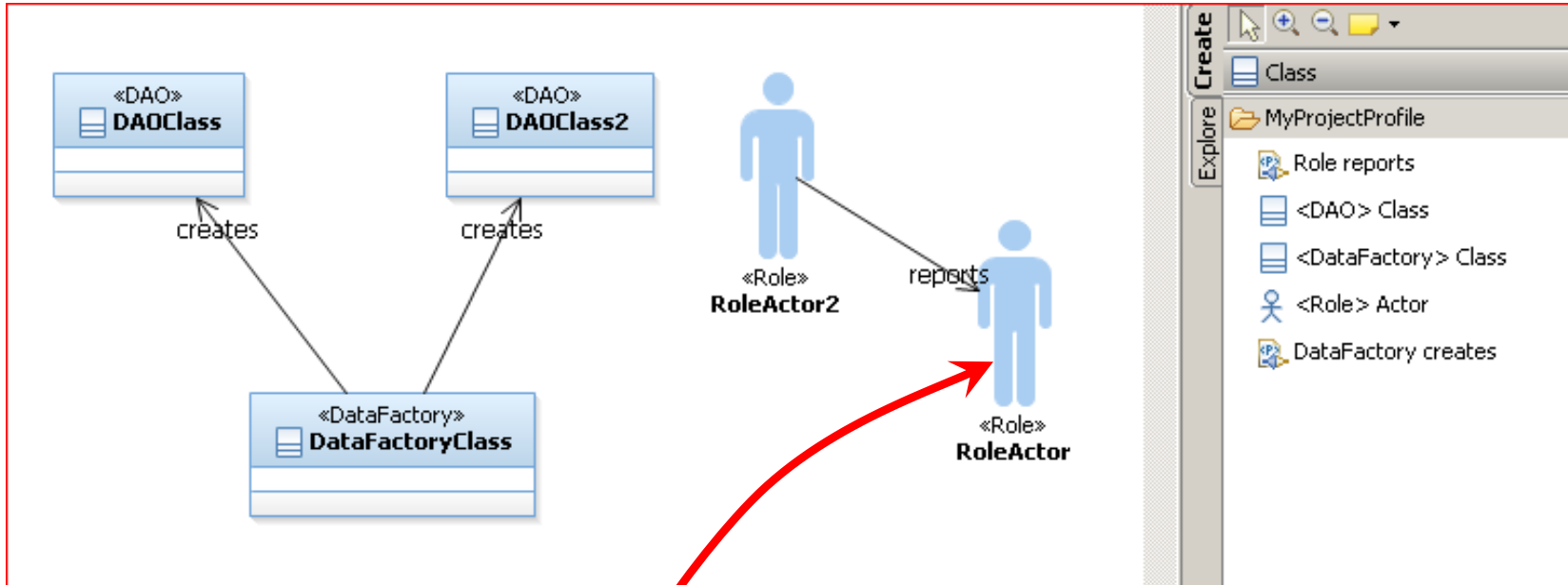
Binding options

Interface	Binding ▾	SOAP Vers
MEDDB-ADAPTER::demo.pva::service::MEDBAdapterInterface	SOAP-DOCUMENT-LITERAL ▾	1.1
	HTTP-POST	
	SOAP-DOCUMENT-LITERAL	
	SOAP-RPC-ENCODED	
	SOAP-RPC-LITERAL	
	WRAPPED-DOCUMENT-LITERAL	

FSER0004MEDDBADAPTER.wsdl

The diagram illustrates the transformation of a service port into a WSDL interface. On the left, a service box contains a port named 'MEDBAdapterInterfacePort' with the URL 'http://demo.pva/service/...'. An arrow points from this port to a central icon representing the transformation. On the right, the resulting WSDL interface 'MEDBAdapterInterface' is shown with the operation 'DateneubertragungDurchfueren'. This operation has two messages: an input message 'MEDBAdapterInterfaceDateneubertragungDurchfuerenRequest1' with the type 'MedDbAnfrage', and an output message 'MEDBAdapterInterfaceDateneubertragungDurchfuerenResponse1' with the type 'MedDbAntwort'.

Domain Specific Languages (DSL) vereinfachen die Modellierung



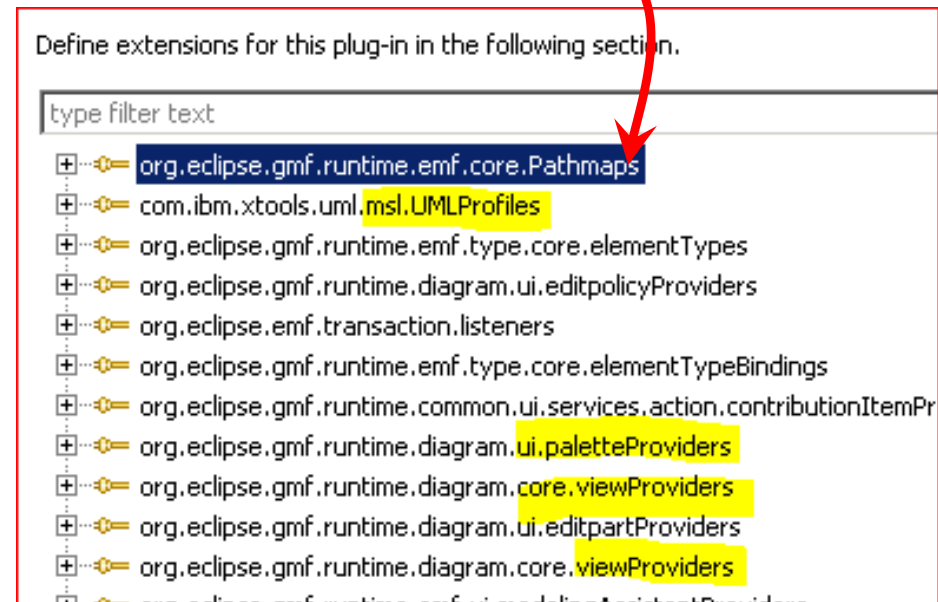
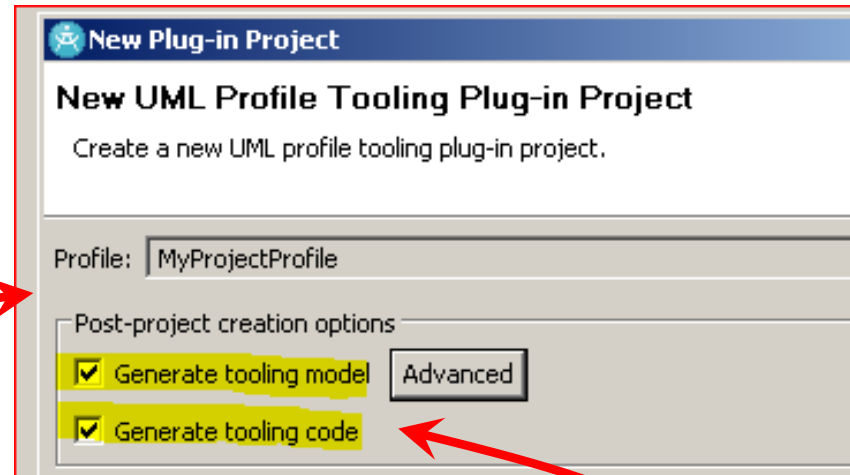
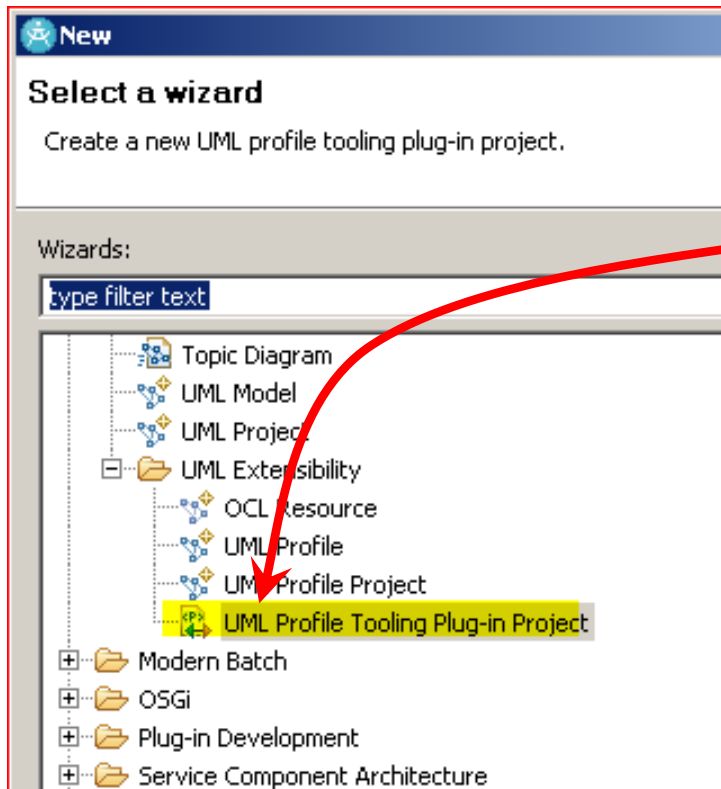
Properties view for «Role» Actor:

Apply Stereotypes... Unapply Stereotypes

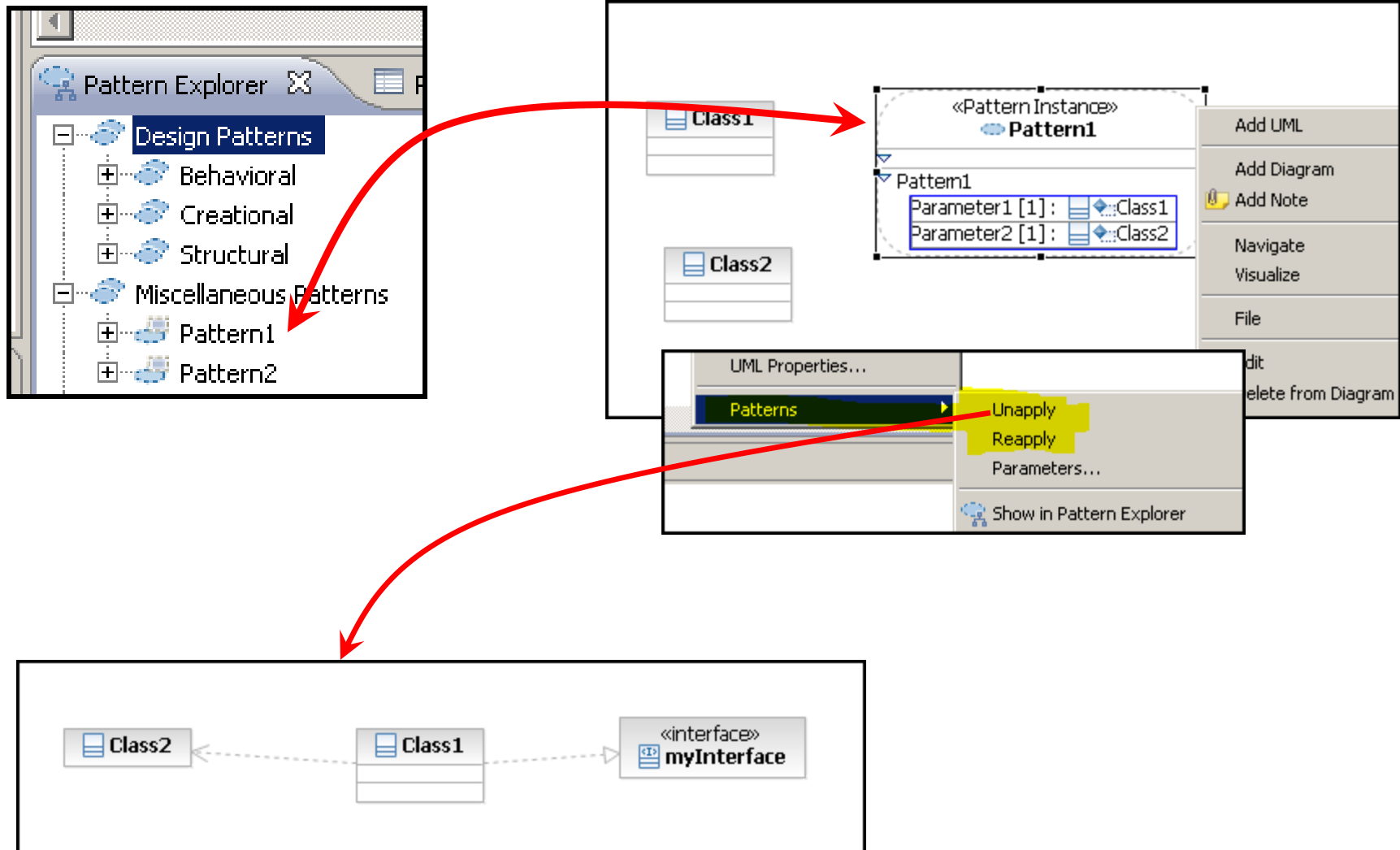
Stereotype Properties:

Property	Value
Role	
department	1 - Finance
designation	2 - CFO
reports	0 - CEO 1 - HR manager 2 - CFO

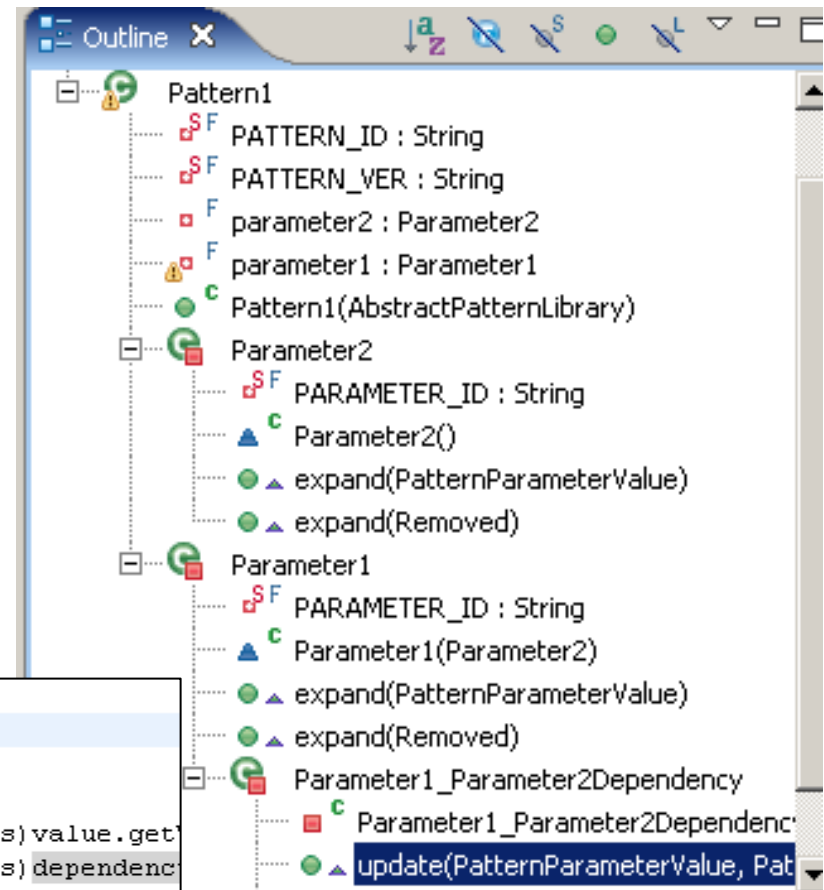
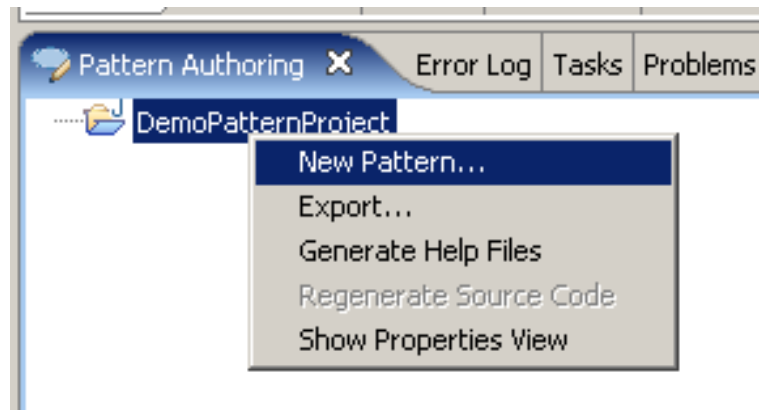
UML Profile Tooling ist vorhanden, um DSL –Funktionalitäten „zu aktivieren“



UML Patterns erhöhen die Produktivität bei der Modellierung



Eigene Patterns können entwickelt werden



```

public boolean update(PatternParameterValue value,
    PatternParameterValue dependencyValue) {
    //TODO: implement the dependency's update method

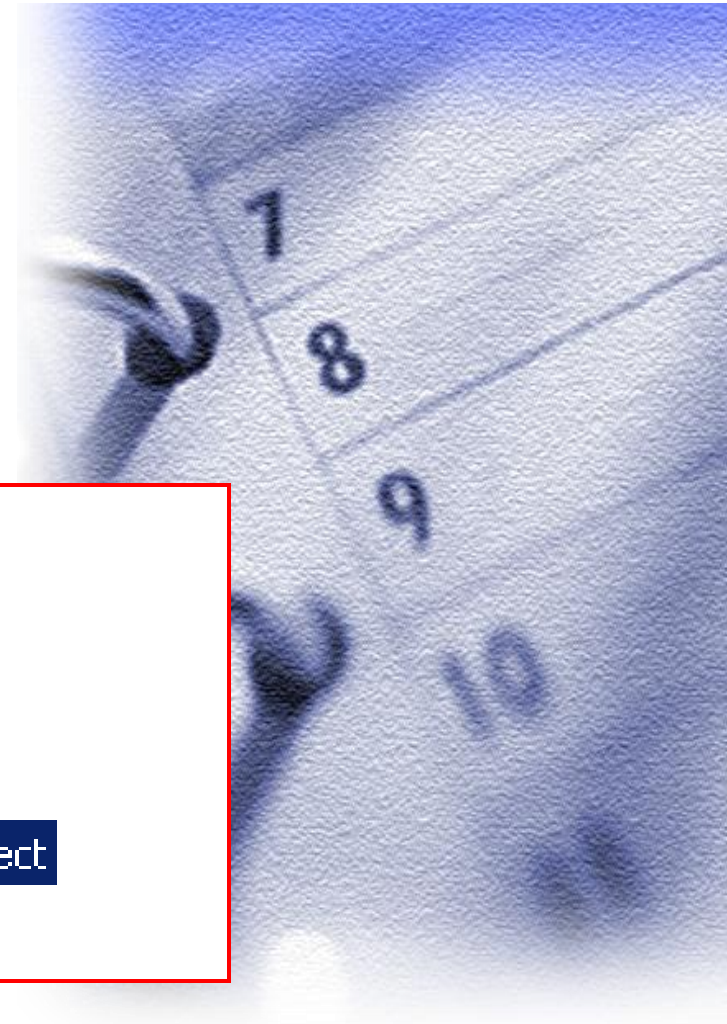
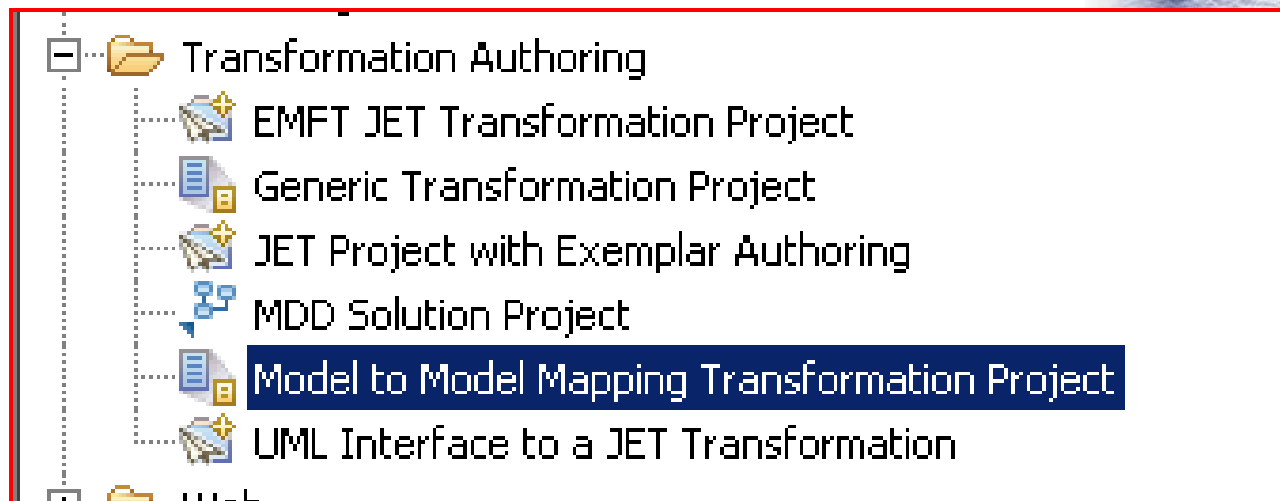
    org.eclipse.uml2.uml.Class a=(org.eclipse.uml2.uml.Class) value.get
    org.eclipse.uml2.uml.Class b=(org.eclipse.uml2.uml.Class) dependenc
    a.createDependency(b) ;

    return true;
}

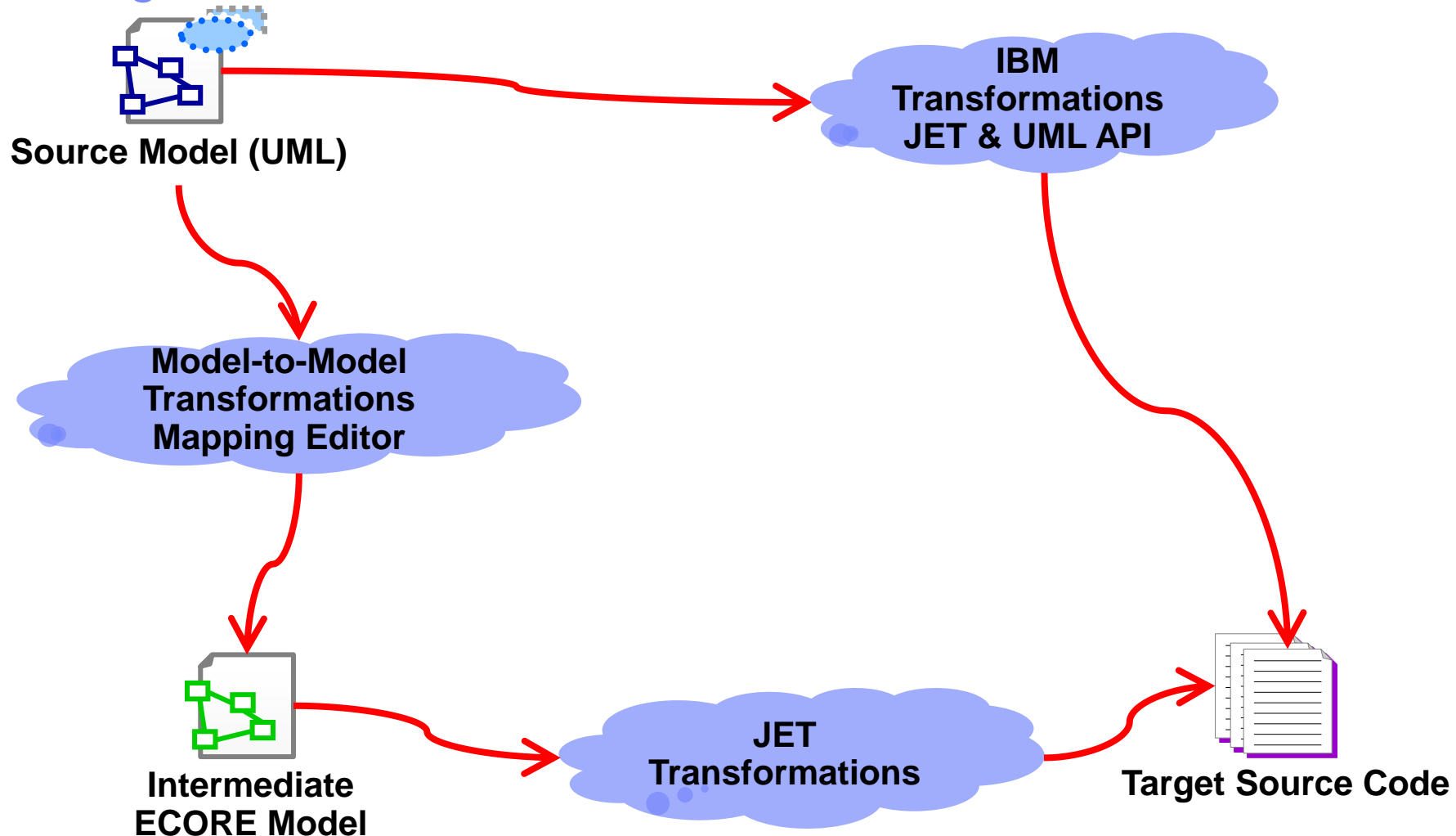
```

RSA stellt „Transformation Authoring Framework“ zur Verfügung, um eigene Transformationen zu entwickeln

- Model-to-Text Transformations
- Model-to-Model Transformations
- Transformations – Extensions



Folgende Möglichkeiten bei der Transformationsentwicklung sind vorgesehen



Java Emitter Templates (JET) werden als Basis für die Transformationen verwendet

JETs vs. JSPs

- Templates lesen die Informationen aus dem „Input Model“ und erzeugen Text, der in Workspace-Files gespeichert wird.
- Mehrere Templates werden in die Transformation zusammengefasst.
- Transformationen werden aus dem User-Interface oder mittels API gestartet.
- Beliebige Artefakte können im Workspace erzeugt werden
- Mehrere Tag-Libraries sind vorhanden, um an „Input Model“ zuzugreifen und den Ablauf zu steuern
- XPath wird verwendet, um über das Modell zu „navigieren“

Java Emitter Templates (JET) - Beispiel

```

private void init() {
    Vector<ICommandHandler> v = new Vector<ICommandHandler> ();
    <c:iterate select="$console/command" var="command">
        v.addElement(new <c:get select="$command/@handlerName" />());
    </c:iterate>

```

```

<c:iterate select="$root/console" var="console">
    <%-- actions for $console --%>

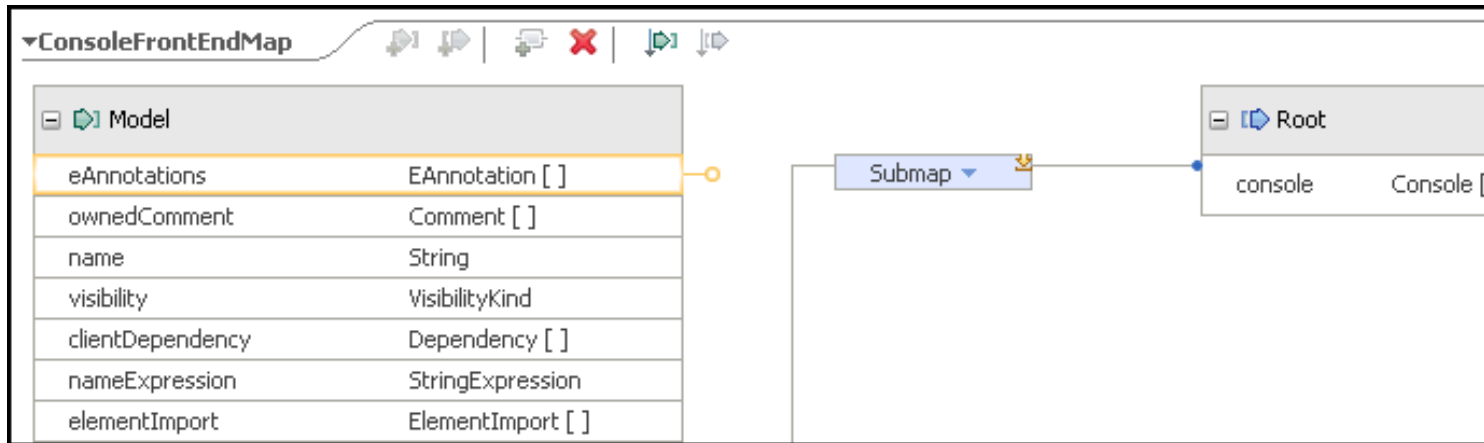
    <ws:project name="{ $console/@projectName}"/>
    <ws:file path="{ $console/@projectName}/.classpath" replace="true" template=
    <ws:file path="{ $console/@projectName}/.project" replace="true" template=
    <ws:file path="{ $console/@projectName}/src/{ $console/@dir}/Console.java"
    <ws:file path="{ $console/@projectName}/src/{ $console/@dir}/ICommandHandle

    <c:iterate select="$console/command" var="command">
        <%-- actions for $command --%>

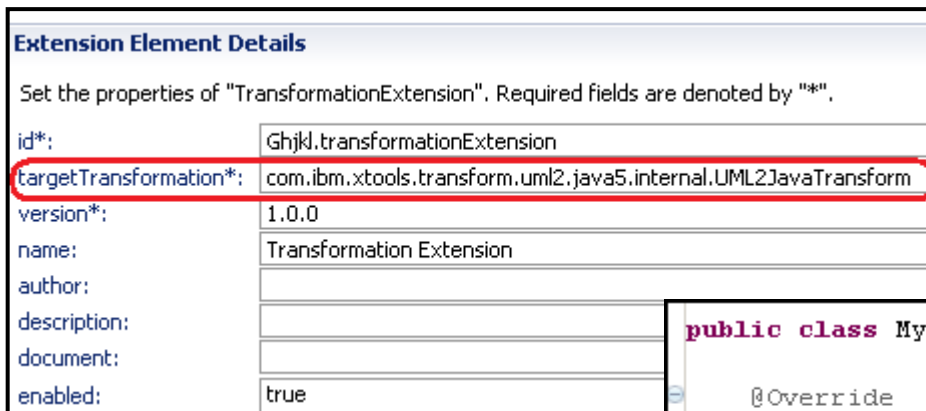
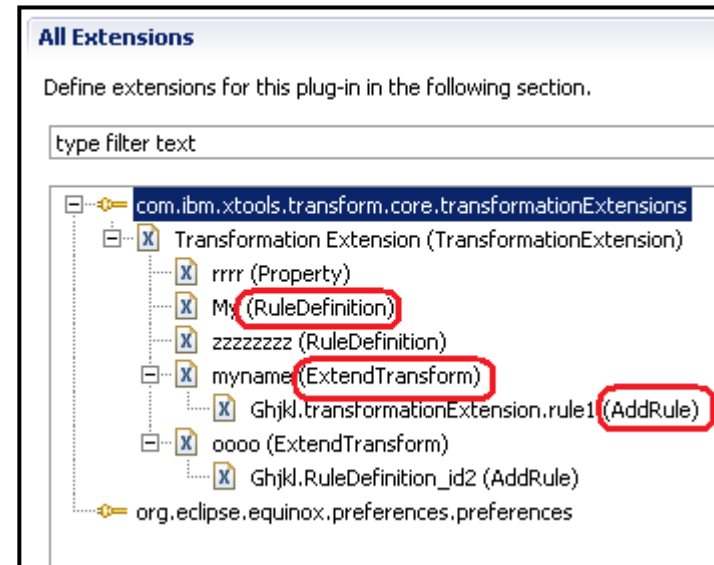
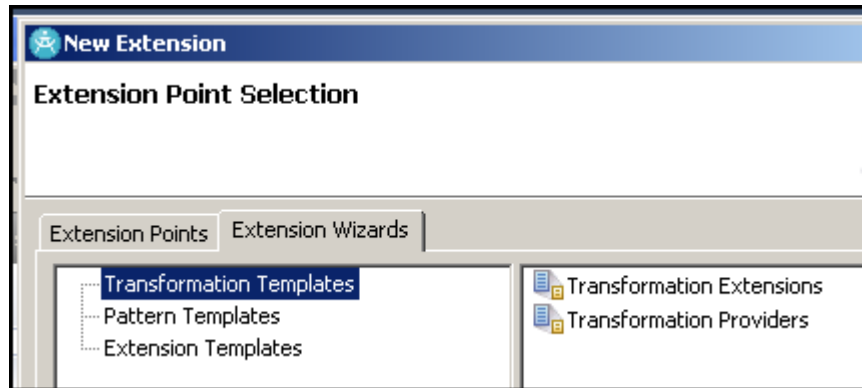
        <ws:file path="{ $console/@projectName}/src/{ $console/@commandDir}/{ $com
        <ws:file path="{ $console/@projectName}/src/{ $console/@commandDir}/{ $com

```

Graphischer Editor steht zur Verfügung, um die Mapping zwischen UML-Modellen und JET-Input zu erzeugen



Transformation - Extensions



```
public class MyRule1 extends ClassRule {

    @Override
    protected Object createTarget(ITransformContext context) {
        // TODO Auto-generated method stub
        TypeDeclaration target = (TypeDeclaration) context.getTarget();
        Javadoc javadoc = target.getJavadoc();
        List tags = javadoc.tags();
    }
}
```

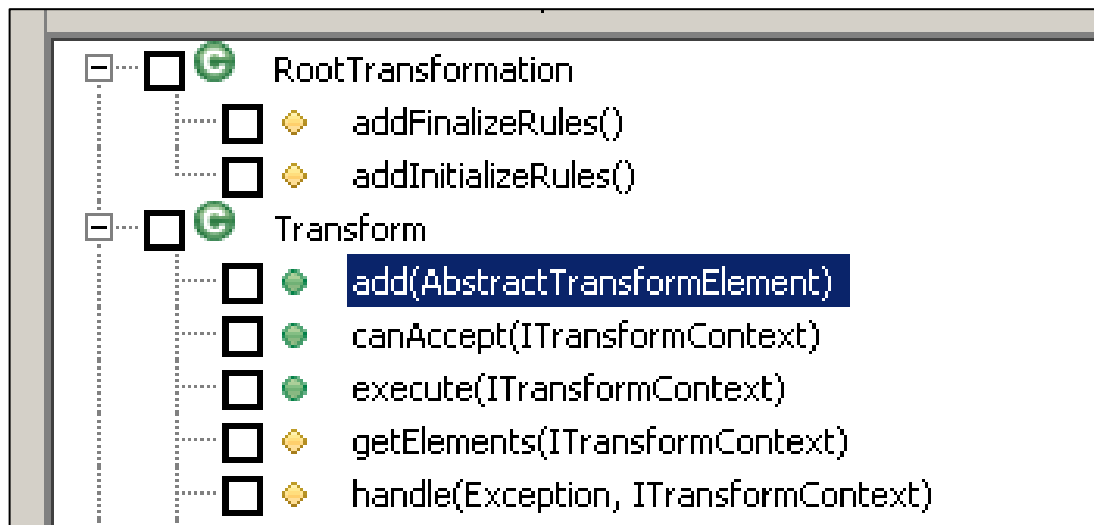
Transformation - Cloning

```
public Transformation(ITransformationDescriptor descriptor) {  
    super(descriptor);  
    RootTransform umlkindTransform = getClone();  
    if(umlkindTransform!=null) {  
        umlkindTransform.addToInit(new InitializeRule());  
        add(umlkindTransform);  
    }  
}
```

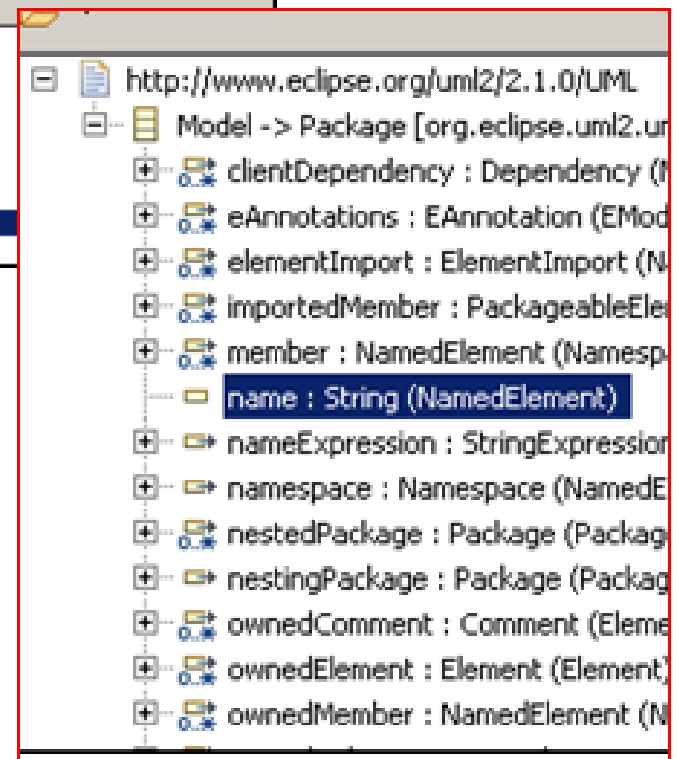
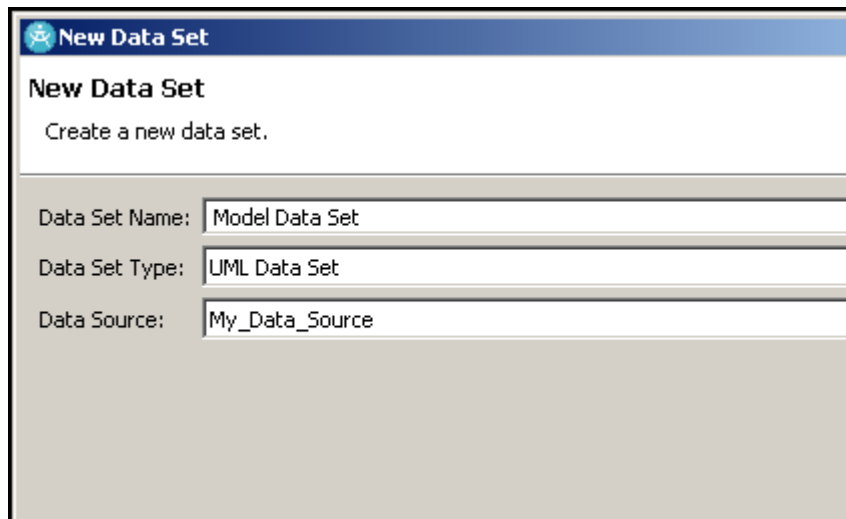
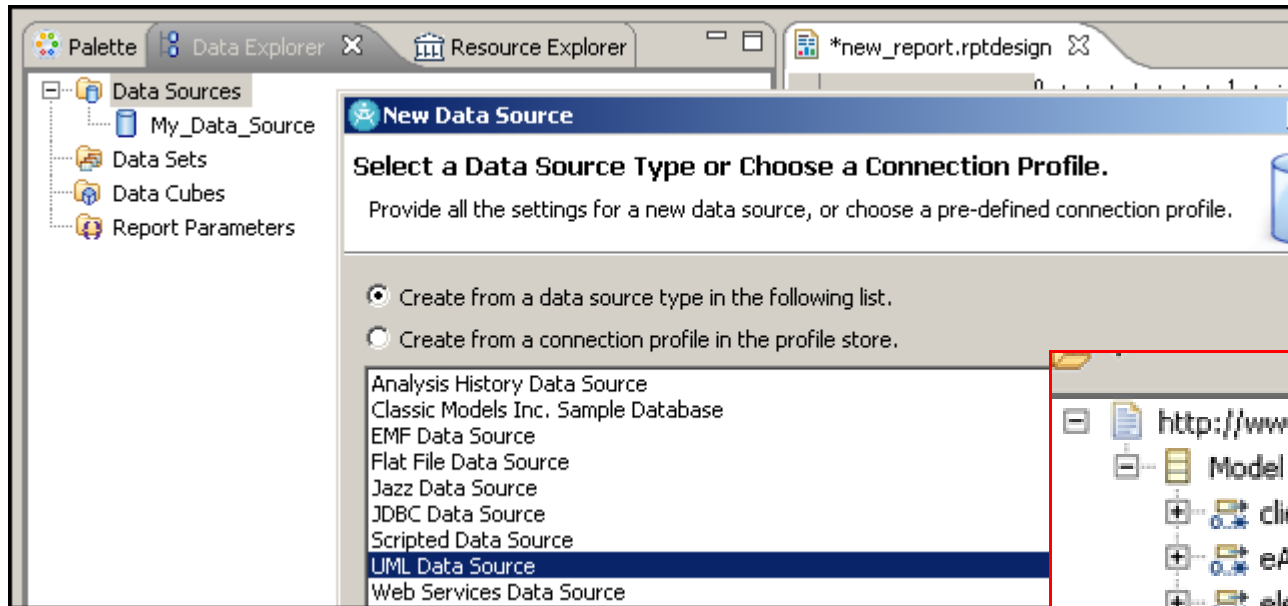
```
private RootTransform getClone() {  
    AbstractTransform tmp_result=null;  
    ITransformationDescriptor descriptor=TransformationServiceUtil.  
        getTransformationDescriptor("com.ibm.at.basisTransformation");  
    tmp_result=TransformationServiceUtil.createTransformation(descriptor);  
    if(tmp_result instanceof RootTransform) {  
        return (RootTransform)tmp_result;  
    }  
    return null;  
}
```

Transformation - Chaining

```
protected RootTransformation createRootTransformation(ITransformContext context) {  
    return new RootTransformation(descriptor, new MainTransformation(descriptor, context));  
    protected void addPostProcessingRules() {  
        add(new JETRule("lab.console.transform"));  
    }  
};
```



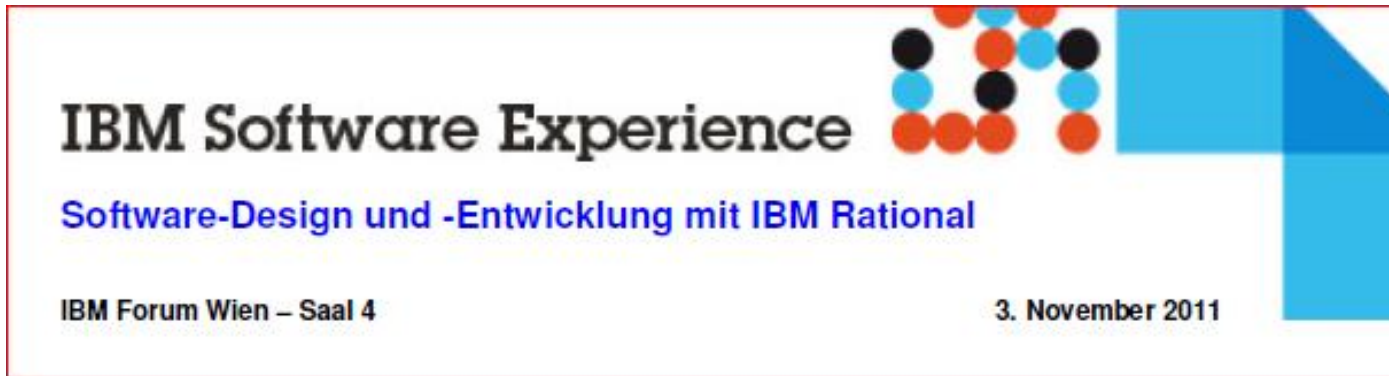
UML Reporting mit BIRT



Zusammenfassung

- MDA / MDD funktioniert, hat sich in der Praxis bewährt
- Mehrere OMG's Versprechungen sind erfüllt
- Voraussetzungen und Tools sind vorhanden
- MDA/MDD Vorteile:
 - Steigerung der Qualität / Reduzierung der Restfehlerrate
 - Schnellere Einsatz neuer Technologien
 - Reduzierung von Zeit, Kosten und Risiken
 - Wiederverwendung
 - Automatisiertes / Kontrollierbares "Copy&Paste" mittels "Transformationen,,
- Die Vorteile entstehen durch:
 - Automatisierung
 - Wiederholbarkeit und Nachvollziehbarkeit
 - Know How Integration
- MDA / MDD kann und wird Software Development verändern

Zusammenfassung



http://www-05.ibm.com/at/events/software_experience/



Questions

Thank You