



TECHNISCHE
UNIVERSITÄT
DRESDEN



Fakultät Informatik, Institut für Software- und Multimediatechnologie, Lehrstuhl Softwaretechnologie

Controlling Model-Driven Software Development through Composition Systems

Jendrik Johannes, NW-MODE'09, Tampere, 29 August 2009

- Motivation
- Invasive Software Composition (ISC) / Reuseware
 - History
 - Concepts of Invasive Software Composition Systems
- Applying ISC in MDSD (shown on an example process)
- Conclusion & Outlook

- **Problem**

- Separation (and composition) of Concerns in MDSD
- Different languages require transformations
 - Risk of mixing generated and manually modelled parts high
 - Application model becomes inconsistent
- Required *composition systems* are missing in languages
 - SoC can not be performed
 - Generic solution to define new *composition systems* may help

- **Problem**

- Separation (and composition) of Concerns in MDSD
- Different languages require transformations
 - Risk of mixing generated and manually modelled parts high
 - Application model becomes inconsistent
- Required *composition systems* are missing in languages
 - SoC can not be performed
 - Generic solution to define new *composition systems* may help

- (One) **Solution:**

Metamodel-based Universal
Invasive Software Composition (ISC)

- History

- BETA's fragment system

- Abstraction mechanisms in the BETA programming language (Madsen et al, 83)

- Invasive Software Composition

- Book: Invasive Software Composition (Aßmann, 2001)

- Extends idea, shows applicability for Java

- Universal Invasive Software Composition

- PhD: A Lightweight Framework for Universal Fragment Composition (Henriksson, 08)

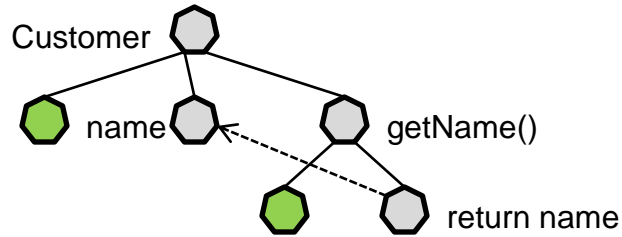
- Generalizes approach based on Context-Free Grammars
 - Introduces generative framework as tool support

- Metamodel-based Universal Invasive Software Composition

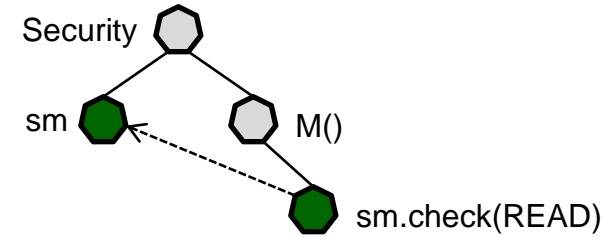
- On language-independent model modularisation (Johannes et al, 09) + ongoing work

- Extends approach for Metamodels
 - Interpretative framework (**Reuseware**) as extension to Eclipse (meta)modelling environments

- Invasive Composition Systems
 - **Composition Technique** (composition engine)
 - Generic, based on graph rewriting
 - Merging fragments
 - **Component Model** (what are the interfaces of components?)
 - Generic core
 - Specification of complex component types (e.g., aspects)
 - For arbitrary languages (based on metamodel)
 - **Composition Language** (def. composition of component instances)
 - Generic language
 - Arbitrary languages can be interpreted as composition language

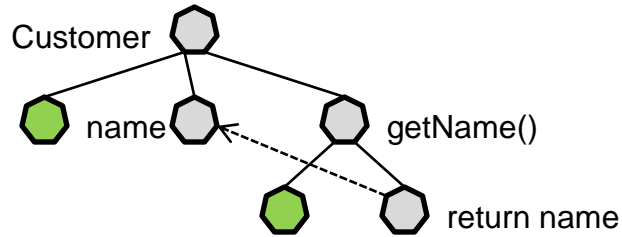


```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```



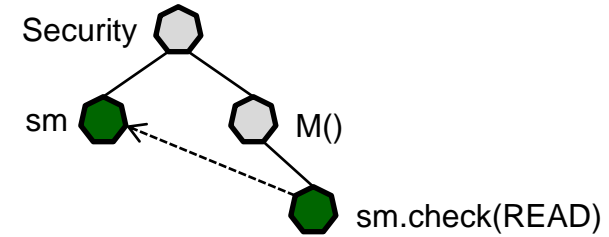
```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

ISC – an Example



```

public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
    
```



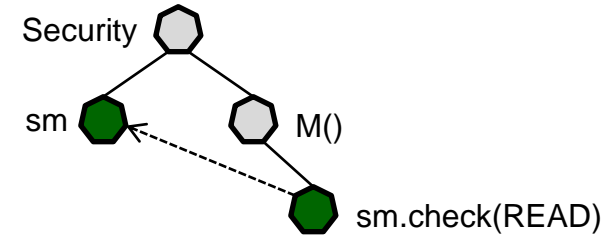
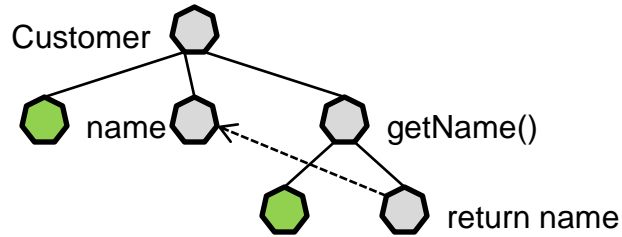
```

public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
    
```

```

CustomerFI1 = instantiate(Customer.java);
SecurityFI1 = instantiate(Security.java);
link(CustomerFI1, SecurityFI1);
    
```

ISC – an Example



```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```

```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

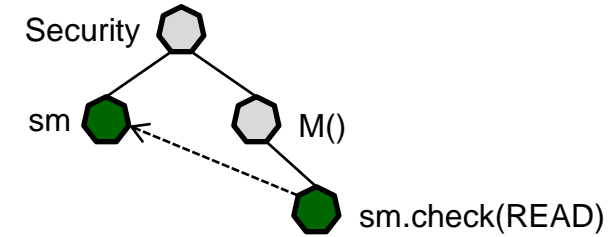
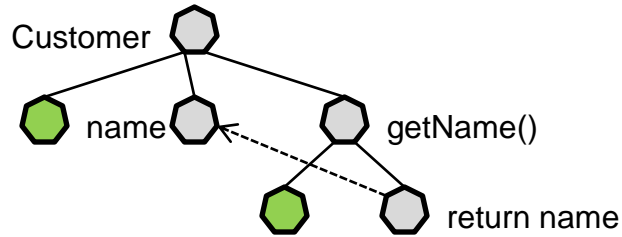
INSTANCE-OF

CustomerFI1

Instance-of myCL::Instantiate

```
CustomerFI1 = instantiate(Customer.java);
SecurityFI1 = instantiate(Security.java);
link(CustomerFI1, SecurityFI1);
```

ISC – an Example



```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```

```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

INSTANCE-OF

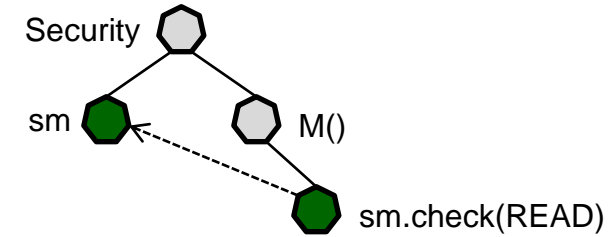
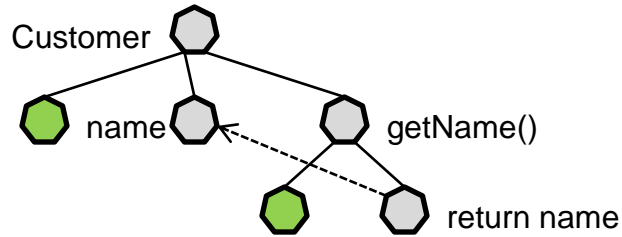
CustomerFI1

SecurityFI1

Instance-of myCL::Instantiate

```
CustomerFI1 = instatiate(Customer.java);
SecurityFI1 = instantiate(Security.java);
link(CustomerFI1, SecurityFI1);
```

ISC – an Example



```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```

```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

INSTANCE-OF

CustomerFI1

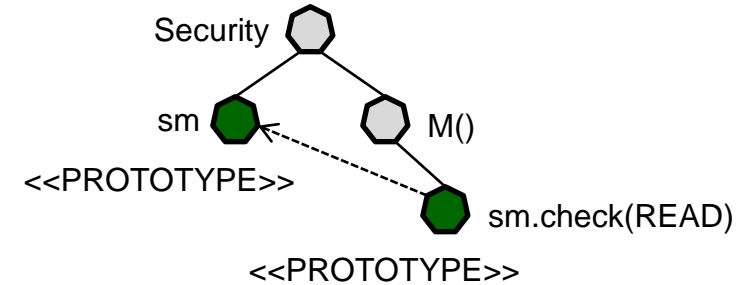
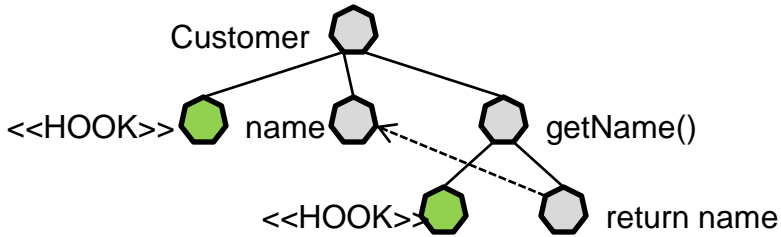
SecurityFI1

COMPOSE

Instance-of myCL::Instantiate
compose myCL::Link

```
CustomerFI1 = instantiate(Customer.java);
SecurityFI1 = instantiate(Security.java);
link(CustomerFI1, SecurityFI1);
```

ISC – an Example



```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```

```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

INSTANCE-OF

CustomerFI1

hook java::JumpLabel
protoype java::Field

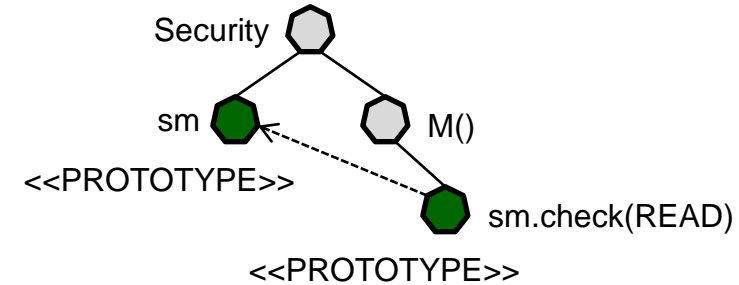
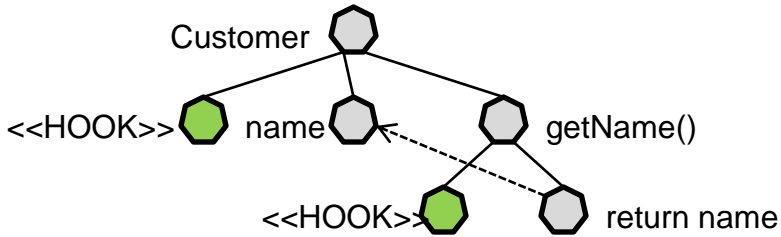
SecurityFI1

COMPOSE

Instance-of myCL::Instantiate
compose myCL::Link

```
CustomerFI1 = instatiate(Customer.java);
SecurityFI1 = instantiate(Security.java);
link(CustomerFI1, SecurityFI1);
```

ISC – an Example



```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```

```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

INSTANCE-OF

CustomerFI1

hook java::JumpLabel
protoype java::Field

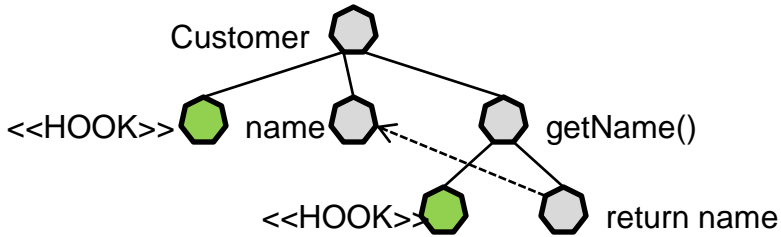
SecurityFI1

COMPOSE

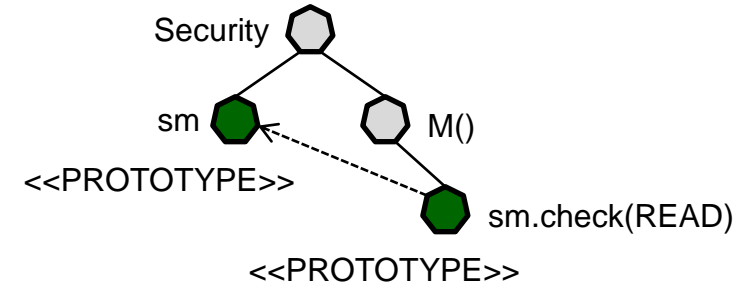
Instance-of myCL::Instantiate
compose myCL::Link

```
CustomerFI1 = instatiate(Customer.java);
SecurityFI1 = instantiate(Security.java);
link(CustomerFI1, SecurityFI1);
```

ISC – an Example



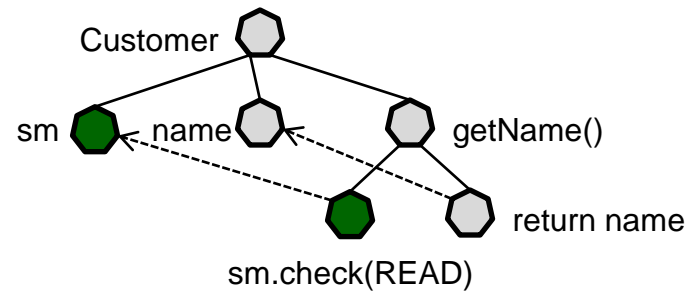
```
public class Customer {
    HOOK : ;
    String name;
    public String getName(){
        HOOK : ;
        return name; } }
```



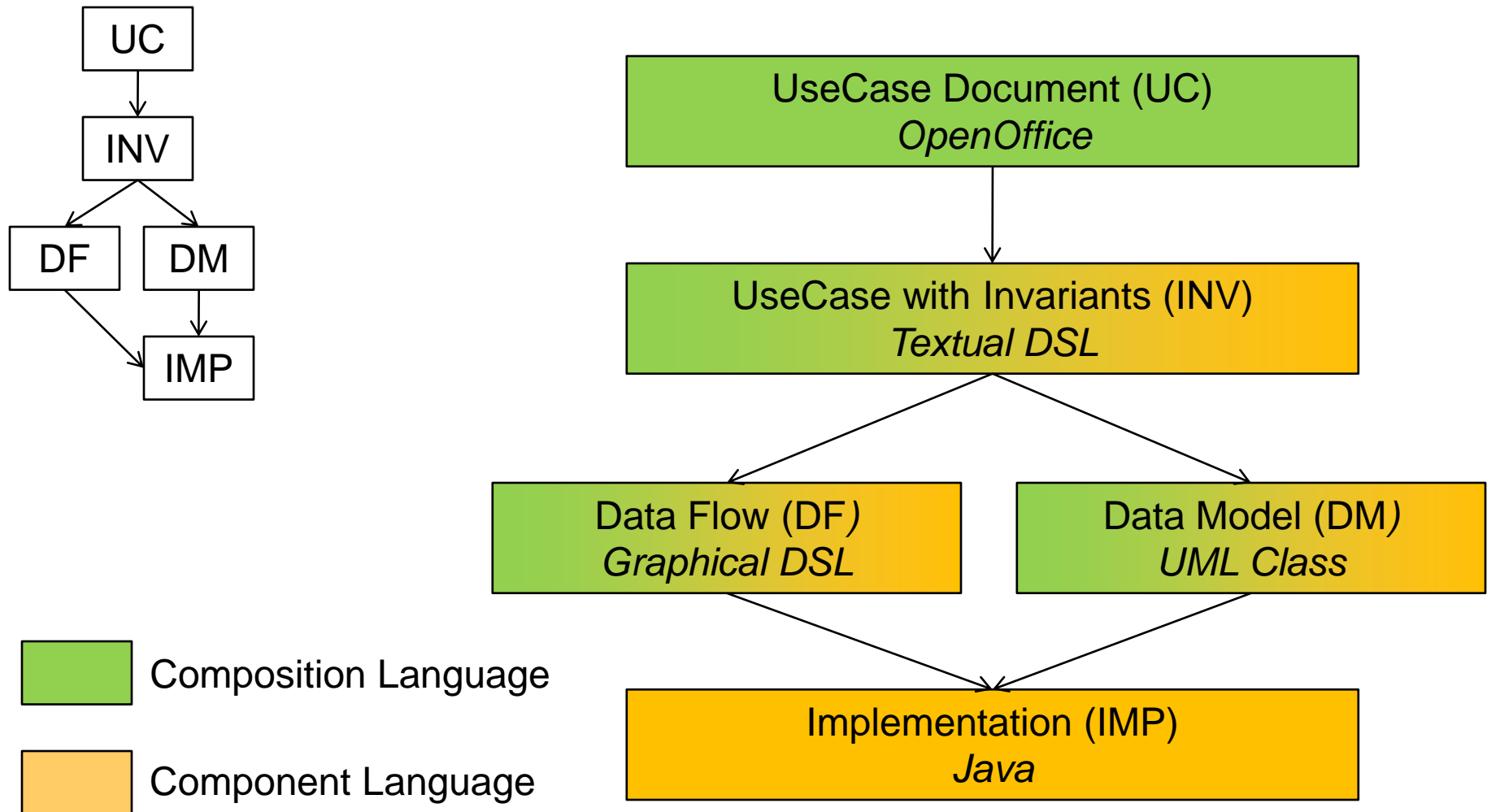
```
public class Security {
    SecurityManager sm =
        new SecurityManager();
    public void M(){
        sm.check(READ); } }
```

INVASIVE COMPOSITION

```
public class Customer {
    SecurityManager sm =
        new SecurityManager();
    String name;
    public String getName(){
        sm.check(READ);
        return name;
    }
}
```

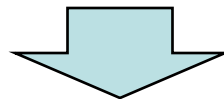
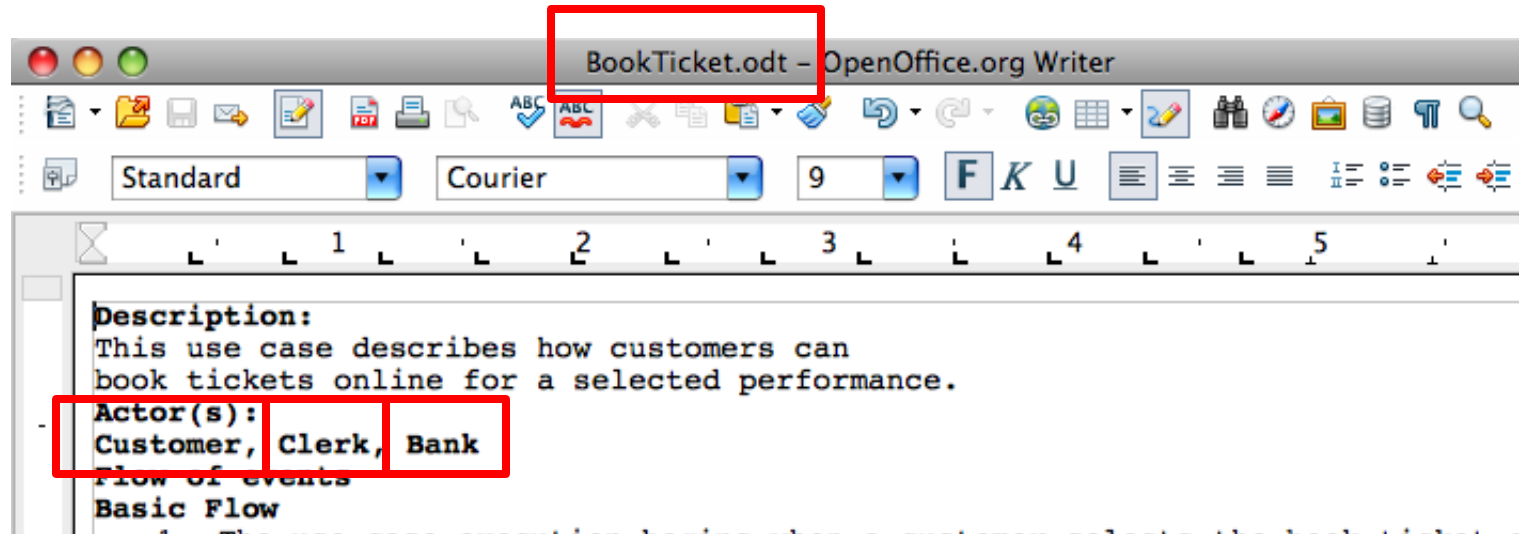
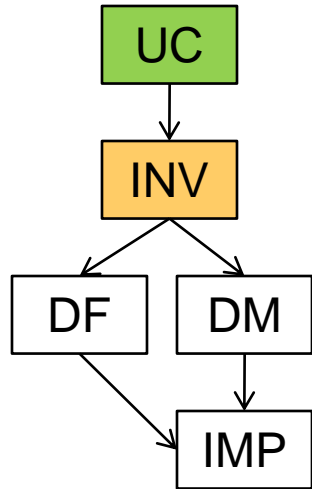


Applying ISC in an MDSD process

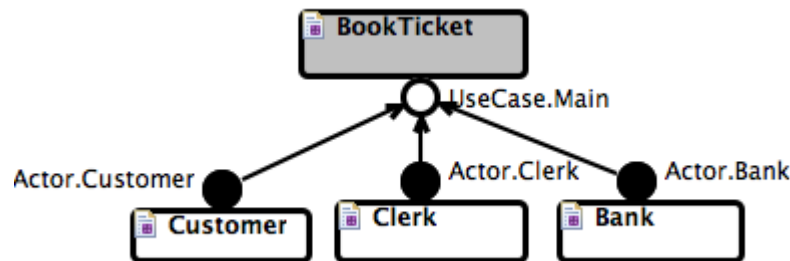


Based on: Roussev, B., Wu, J.: Transforming use case models to class models and ocl-specifications. (2007)
Full example available at: www.reuseware.org/index.php/MDSD

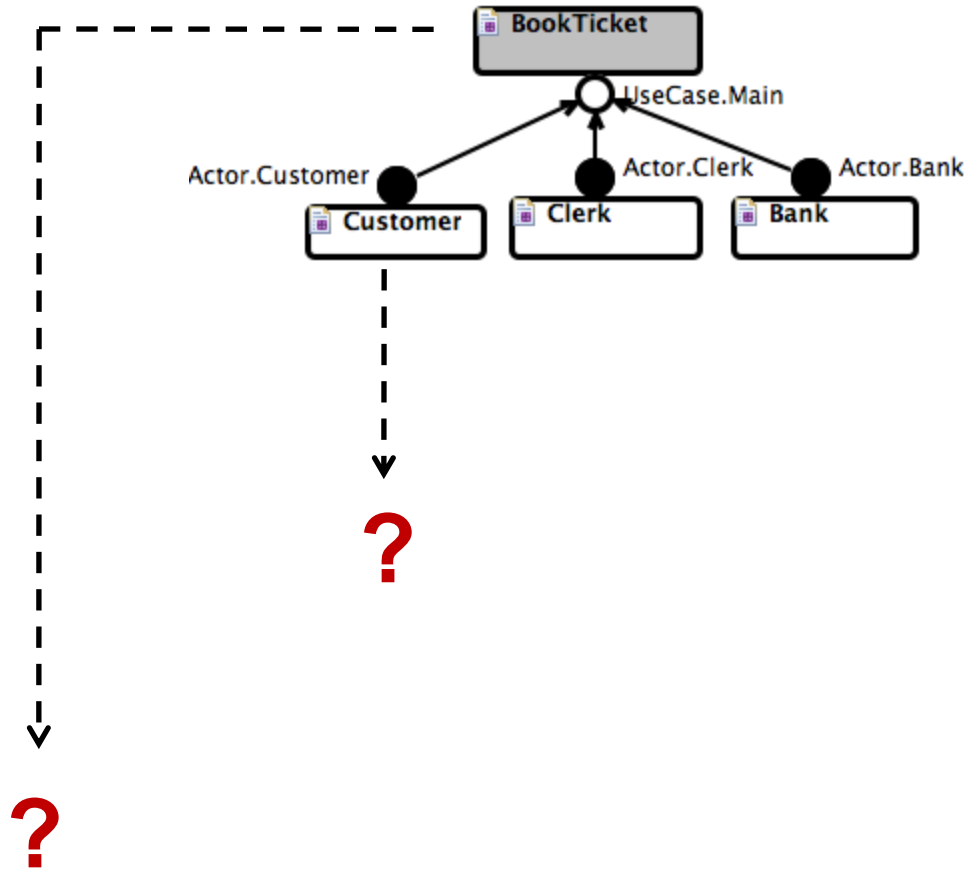
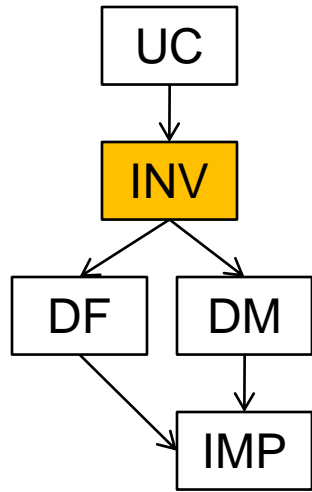
UseCase Document (Level 1)



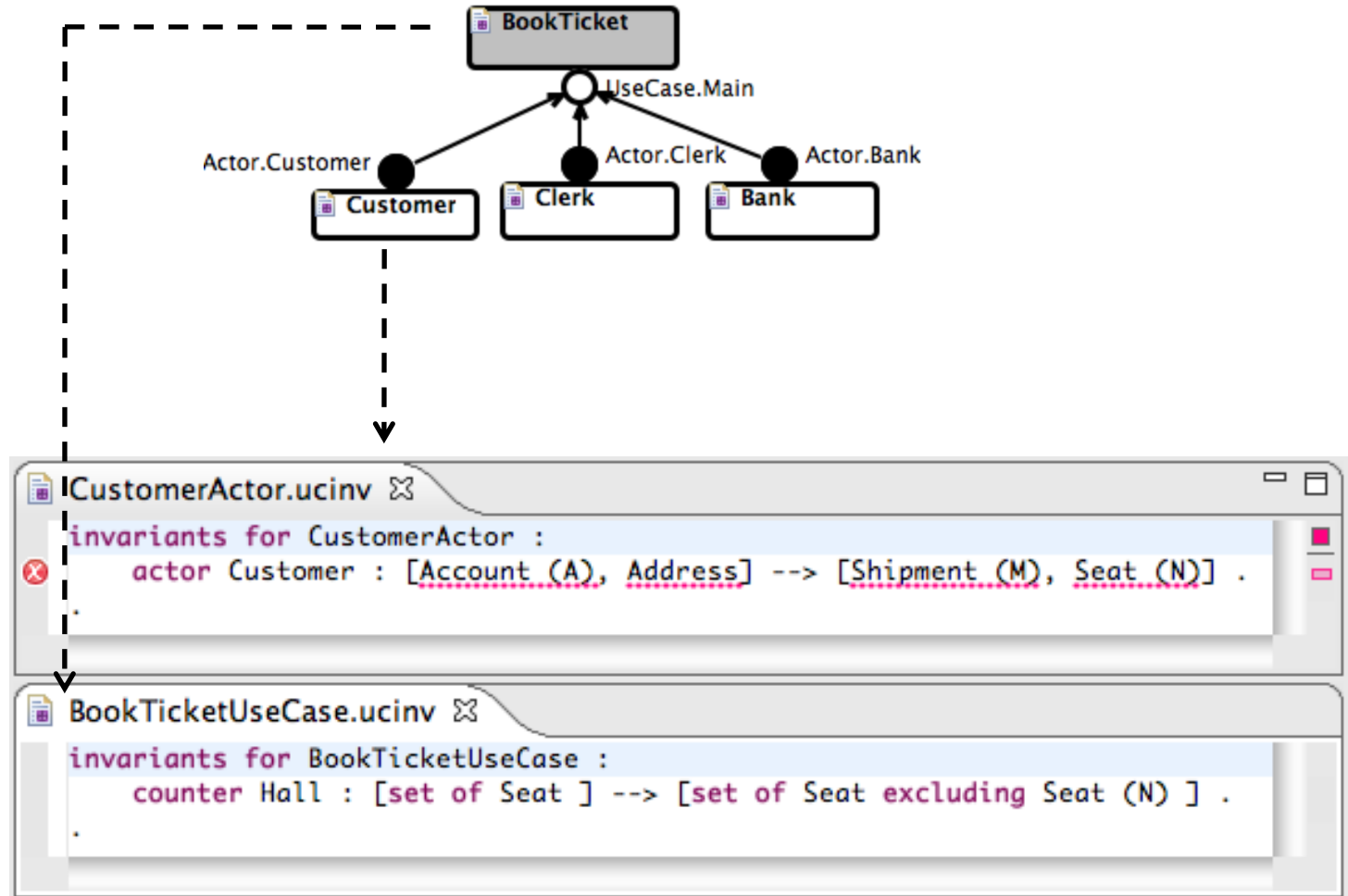
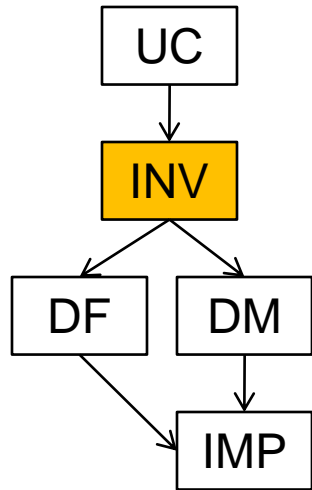
Instance-of odf:text::SpanType if \$styleName = 'Actor'\$



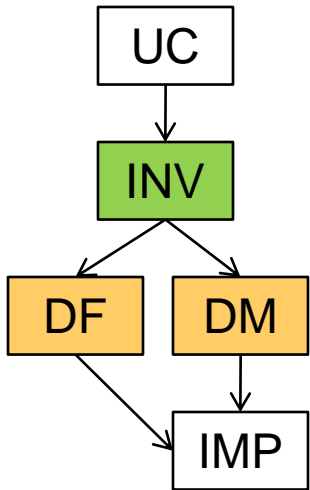
UseCase w/ Invariants (Level 2)



UseCase w/ Invariants (Level 2)

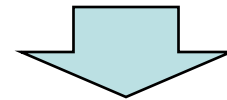
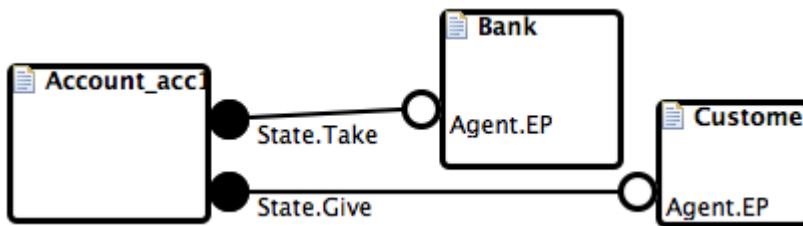
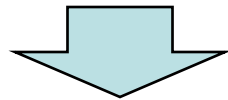


UseCase w/ Invariants (Level 2)

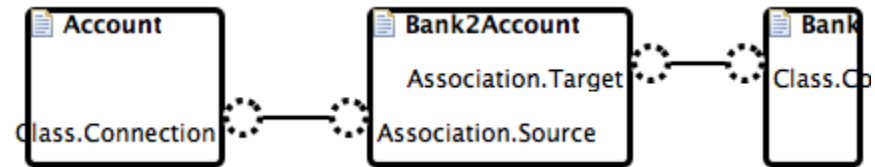


```

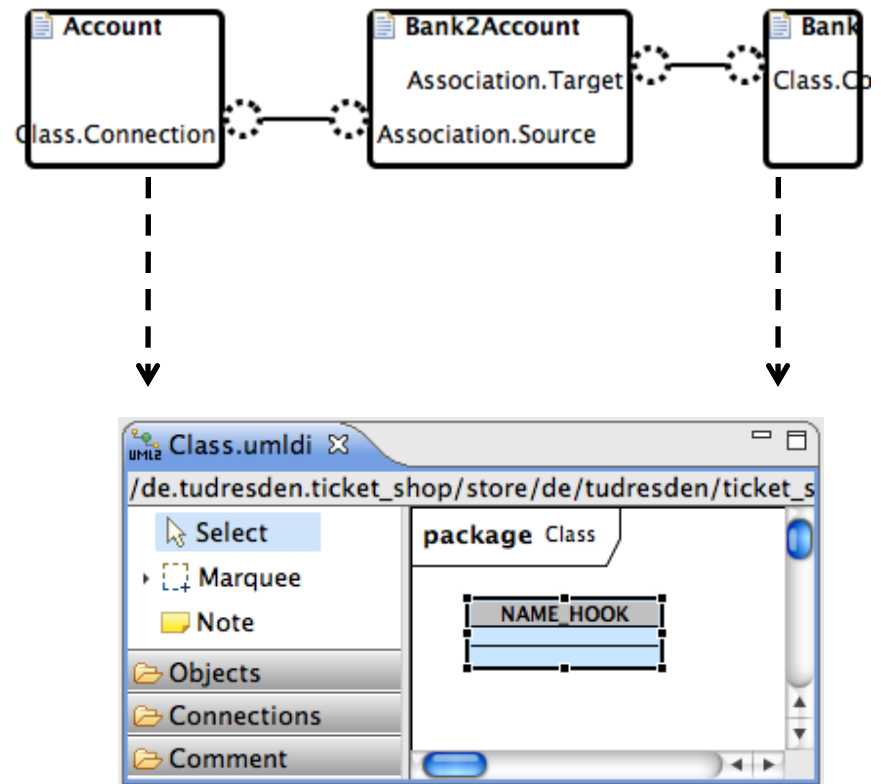
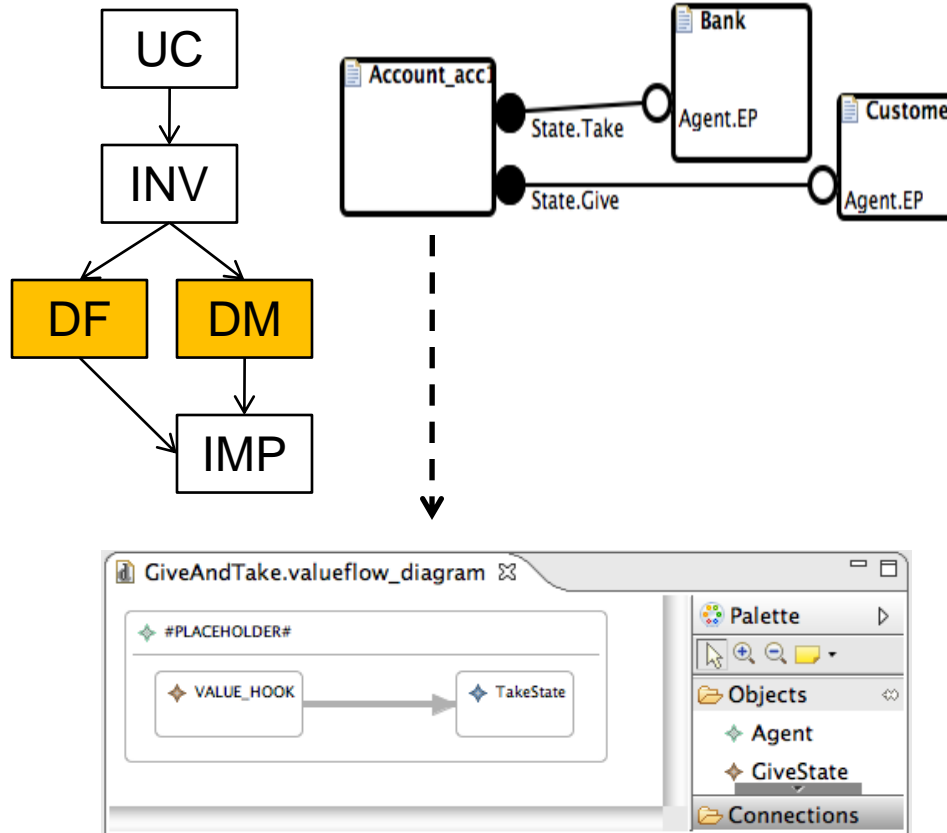
BookTicket.ucinv
Invariants for BookTicket :
  counter Hall : [ set of Seat ] --> [ set of Seat excluding Seat (N) ] .
  actor Customer : [ Account (acc1) , Address (add1) ] --> [ Shipment (M) , Seat (N) ] .
  actor Clerk : [ set of Shipment ] --> [ set of Shipment excluding Shipment (M) , Address (add1) ] .
  actor Bank : [ set of Account ] --> [ set of Account including Account (acc1) ] .
  
```



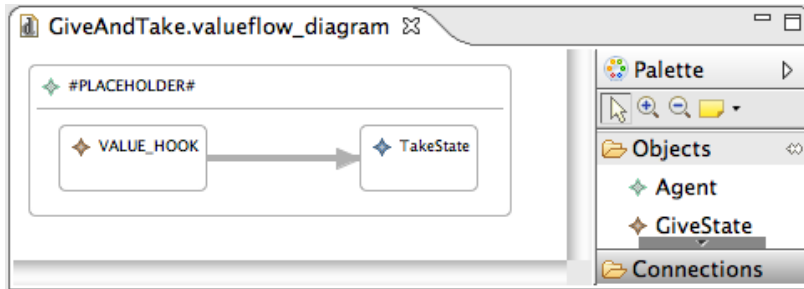
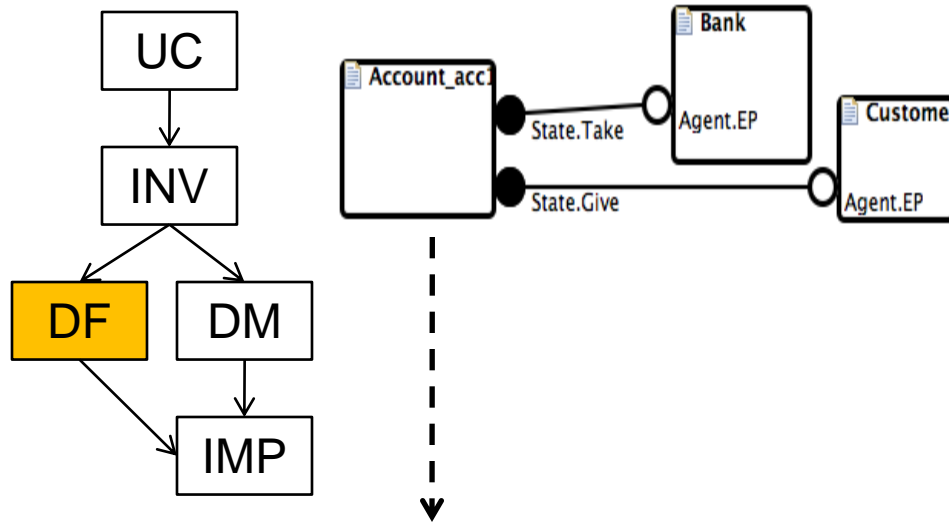
Instance-of ucinv::Value {
 name = '\$self.name'\$
}



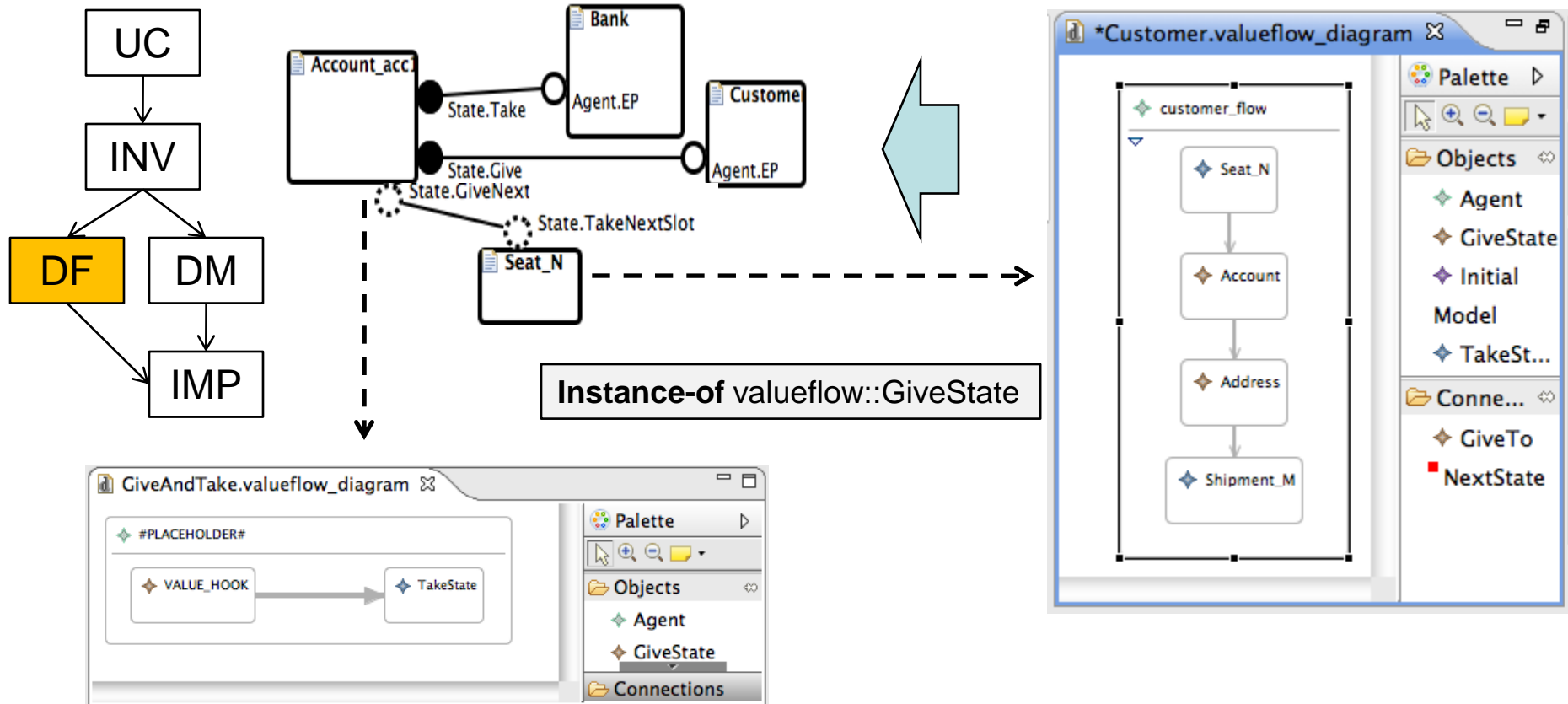
DataFlow & DataModel (Level 3)

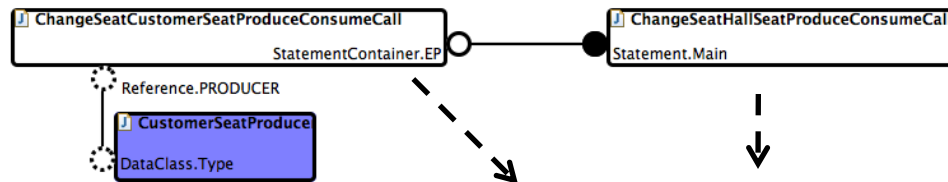
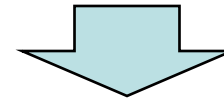
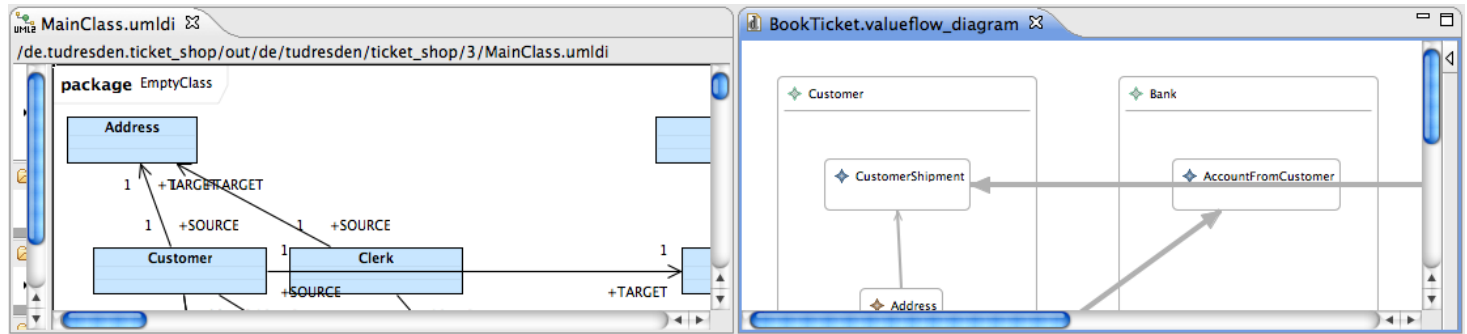
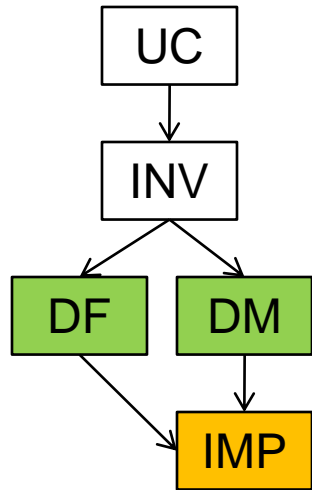


Refinement of DF (Level 3)



Refinement of DF (Level 3)

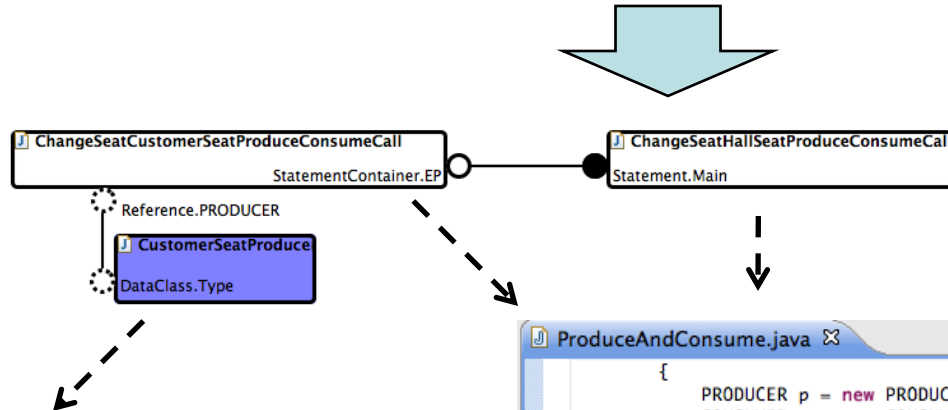
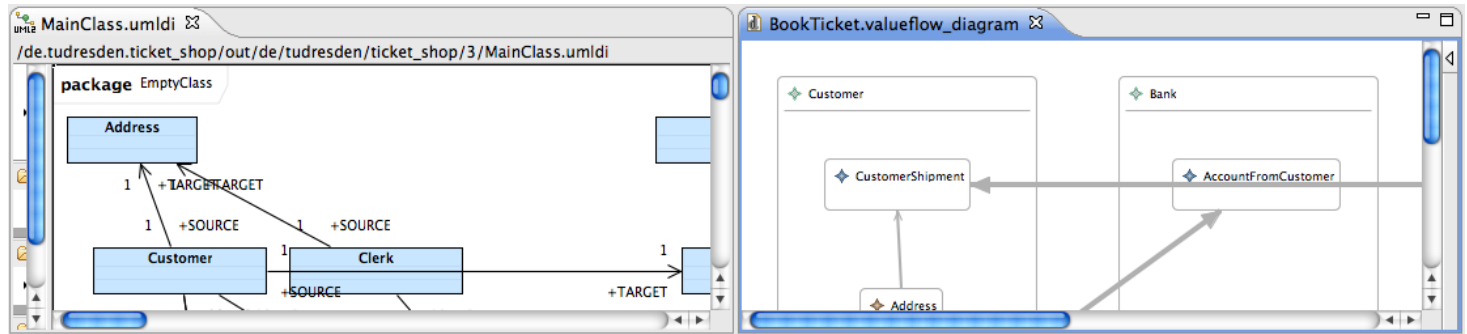
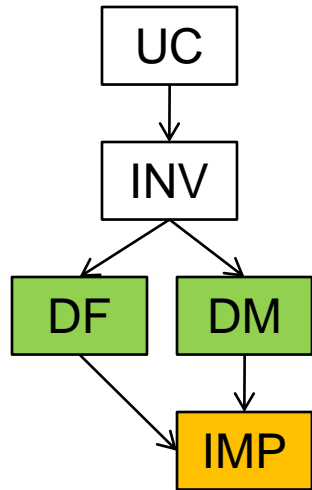




```

    ProduceAndConsume.java
    {
      PRODUCER p = new PRODUCER();
      CONSUMER c = new CONSUMER();

      VALUE v = p.produce(NAME_HOOK);
      if (v == null) {
        return;
      }
      else {
        c.consume(NAME2_HOOK, v);
      }
    }
    EP ;;
  }
  
```



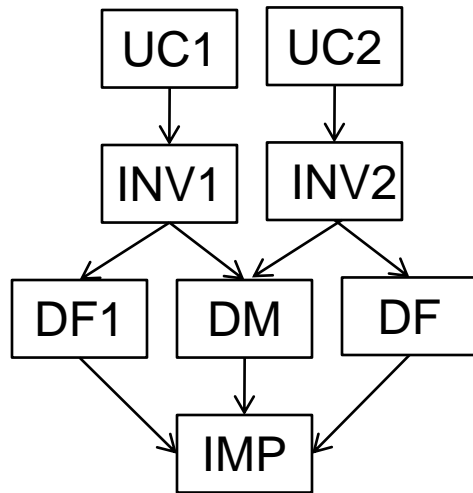
```

    ProduceSeat.java
    public class ProduceSeat implements IProducer<Seat,
    {
        public Seat produce(Customer agent) {
            Seat seat = agent.getSeat();
            String no = "";
            if (seat != null) {
                no = seat.getID();
            }
            else {
                seat = new Seat();
                agent.setSeat(seat);
            }
            InputDialog dialog = new InputDialog(null,
                "Customer Question", "Please enter
  
```

```

    ProduceAndConsume.java
    {
        PRODUCER p = new PRODUCER();
        CONSUMER c = new CONSUMER();

        VALUE v = p.produce(NAME_HOOK);
        if (v == null) {
            return;
        }
        else {
            c.consume(NAME2_HOOK, v);
        }
    }
    EP ;;
  
```



- One composition can be influenced by several models
 - E.g., Data Model composition
- Repeated information is merged/reused
 - E.g, Actor participated in other Use Case – components of Data Model and Implementation are reused
- Missing components are immediately detected
 - E.g., invariants must be defined
 - E.g., new actors appear; influences Impl.

- General Observations
 - Composition Systems can be customized for an MDSD process individually
 - Trade-of power of composition abstraction vs. simplicity
 - Information about system architecture is made explicit
 - SoC increased in several dimensions
 - Reduces duplication of information
- Evaluation
 - Demonstrator system shows that it is possible
 - Different ideas applied to industrial case studies
- Outlook
 - How to combine with other transformation approaches (when to use what?)
 - How can the specification of composition systems be simplified?
 - In how far can those specifications be reused?
 - Editing of composed models via round-trip mechanism



Thank You!



Questions?



reuseware.org



emftext.org

JaMoPP

jamopp.inf.tu-dresden.de

reuseware.org/update