Variations in Model-Based Composition of Domains

Anca Daniela Ionita

University "Politehnica" of Bucharest, CIID-UPB, Romania

Jacky Estublier, German Vega

Institut d'Informatique et Mathématiques Appliquées de Grenoble, LIG-IMAG, France

Summary

- Variation and composition with model driven approaches
- Variation points in domain architecture
  - Application model definition
  - Feature selection
  - Component selection
- Variations inside the domain composition mechanism
  - inter-domain relationship definition
  - inter-model relationship definition
  - inter-domain relationship properties
- Conclusions

Variation and composition with model driven approaches

captured in the model

- Predictable variation points
- Unpredictable variation points

model composition

  - Composition as a mechanism to obtain variation

composition model

  - Variation points in the composition mechanism

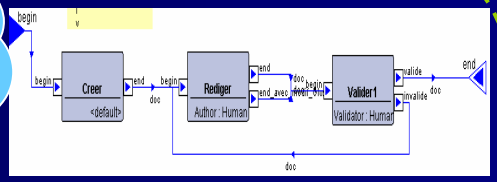
The Domain Approach

Composition of domains

Variability inside domains

Variation points in the composition mechanism

Activity



realization

agenda

process engine

Resource

realization

BD

server

Compose heterogenous components

Document

realization

CVS

Doc editor

Product

realization

SQL Storage

Viewer

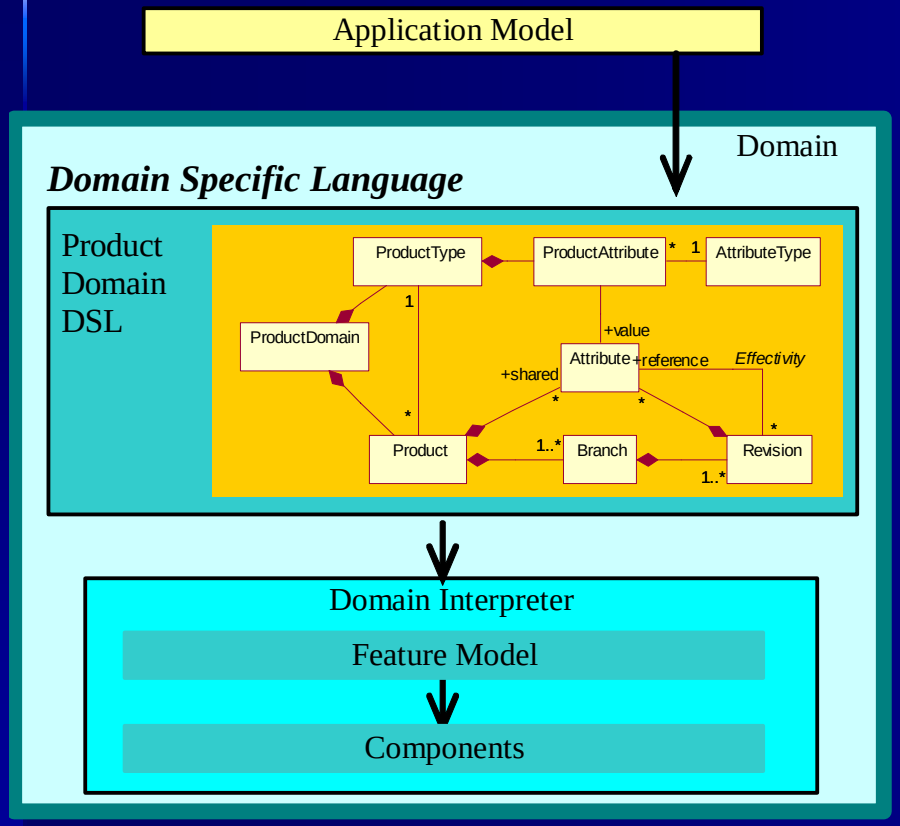
Summary

- Variation and composition with Model Driven Approaches
- Variation points in domain architecture
  - Application model definition
  - Feature selection
  - Component selection
- Variations inside the domain composition mechanism
  - inter-domain relationship definition
  - inter-model relationship definition
  - inter-domain relationship properties
- Conclusions

Domain architecture

E.g. Product domain

Domain Architecture | **Domain Variation Points**



(V1) Application Model definition

e.g. domain architecture model

e.g. persistency, visibility

(V2) Feature selection

(V3) Component selection

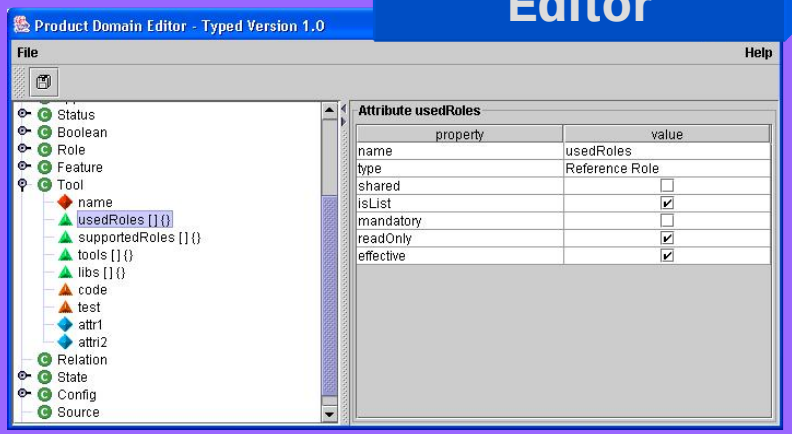
e.g. SQL or CVS storage

Application Model Definition

E.g. Domain Architecture Model

- Often structural only ⟶ Application development does not need any programming
- E.g. Domain Architecture Model

Product Model Editor



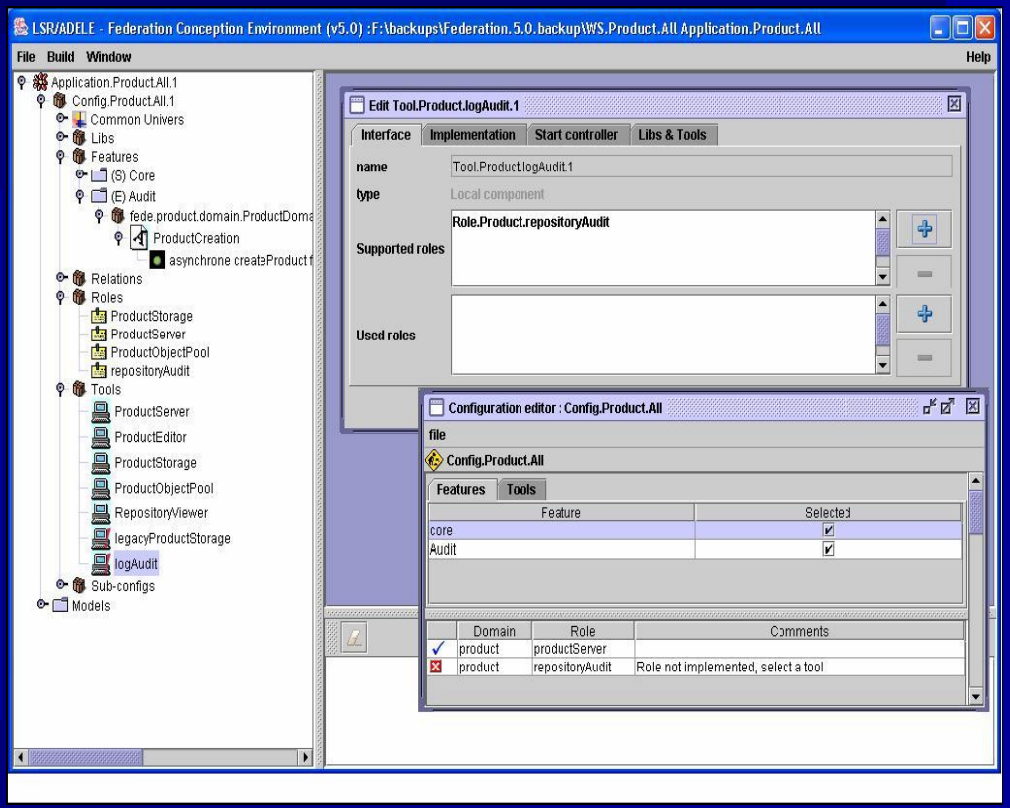
XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<TypeDefinition name="Domain"
        xmlns="http://fr.imag.adele/fede/product/model">
 <TypeRoot xsi:type="StructDefinition"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<attribute name="name" version="false" mandatory="true" readonly="true"
        key="true">
 <TypeRoot xsi:type="StringDefinition" />
 </attribute>
<attribute name="features" maxOccurs="-1" version="true" key="false">
<TypeRoot xsi:type="TypeReference">
 <TypeRoot type="Feature" xsi:type="NameTypeDefinition" />
 </TypeRoot>
 </attribute>
 <attribute name="roles" maxOccurs="-1" version="true" key="false">
 <TypeRoot xsi:type="TypeReference">
 <TypeRoot type="Role" xsi:type="NameTypeDefinition" />
 </TypeRoot>
 </attribute>
<attribute name="tools" maxOccurs="-1" version="true" key="false">
<TypeRoot xsi:type="TypeReference">
 <TypeRoot type="Tool" xsi:type="NameTypeDefinition" />
 </TypeRoot>
 </attribute>
<attribute name="models" maxOccurs="-1" version="true" key="false">
<TypeRoot xsi:type="TypeReference">
 <TypeRoot type="Model" xsi:type="NameTypeDefinition" />
 </TypeRoot>
 </attribute>
 <attribute name="comment" version="true">
 <TypeRoot xsi:type="StringDefinition" />
 </attribute>
 </TypeRoot>
 </TypeDefinition>
```

Feature Selection

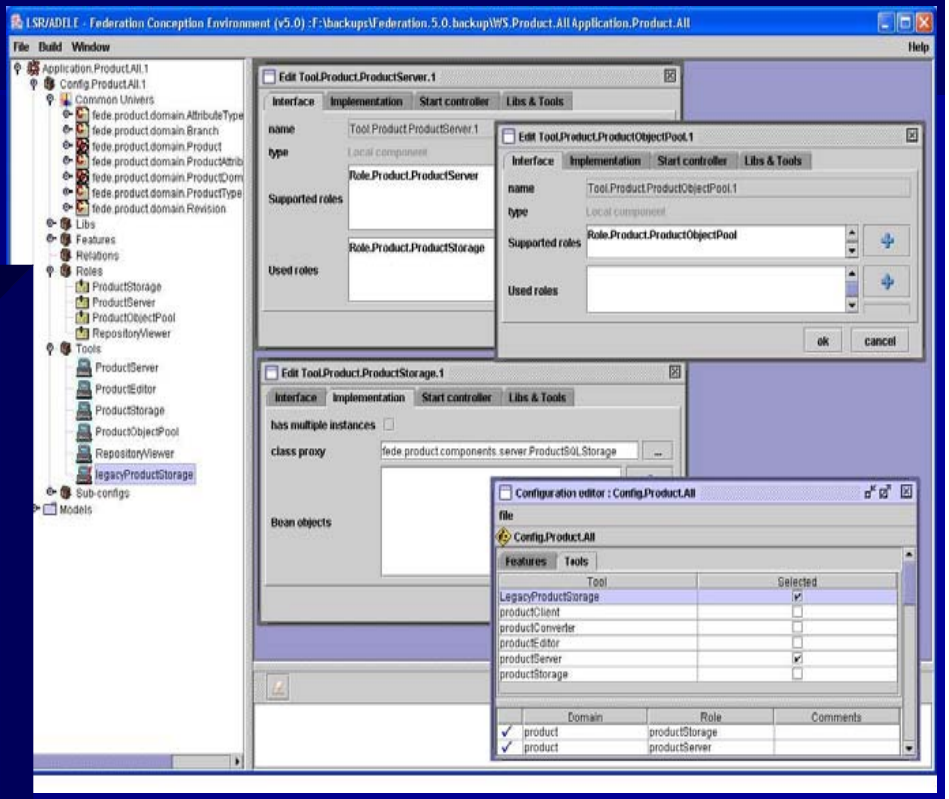
- The feature model does not try to grasp all the domain concepts

- variability at the *implementation* level
- variations regarding *non-functional properties*, which are not related to a single concept



Component (Tool) Selection

• mapping the abstract features to concrete components tools



Tools (components)

• encapsulate most of the implementation code

• reuse highly non-homogenous tools

- ✓ COTS
- ✓ legacy software
- ✓ programs available on the market

Summary

- Variation and composition with Model Driven Approaches
- Variation points in domain architecture
  - Application model definition
  - Feature selection
  - Component selection
- Variations inside the domain composition mechanism
  - inter-domain relationship definition
  - inter-model relationship definition
  - inter-domain relationship properties
- Conclusions

Domain composition

Given:

authonomous domains driven by application models

Purpose:

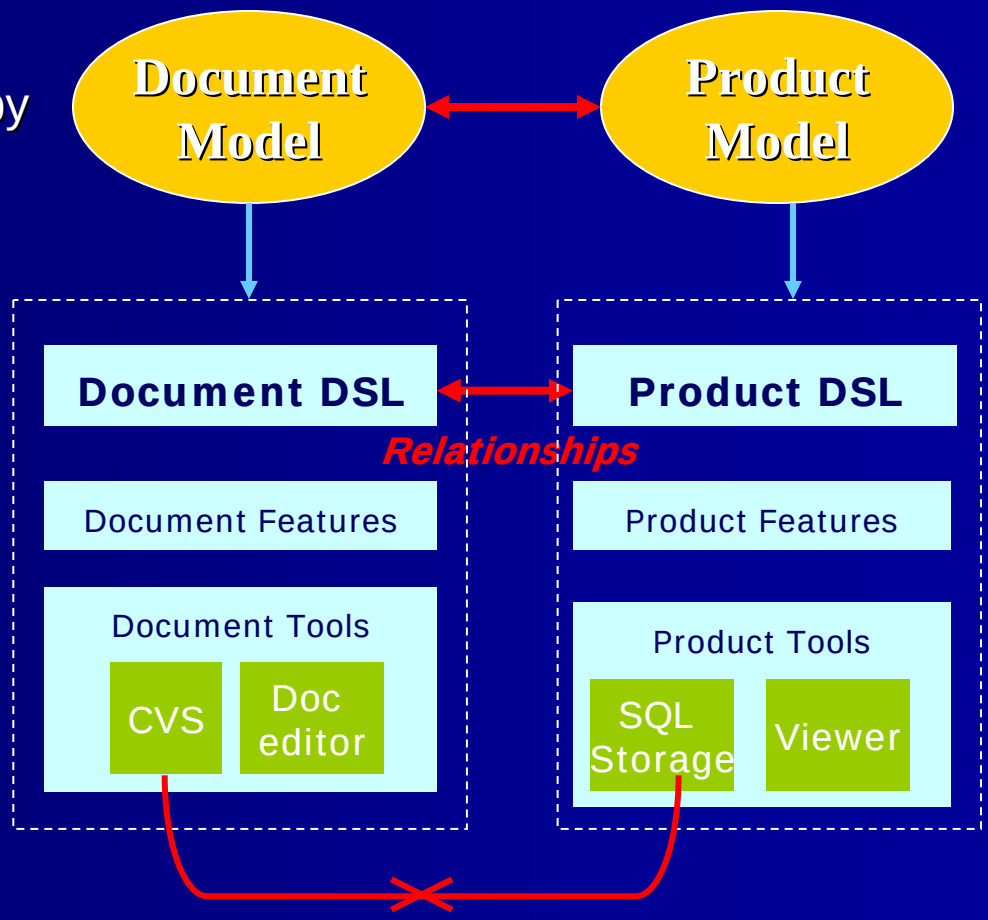
develop wide area applications, that crosscut several domains

Problem:

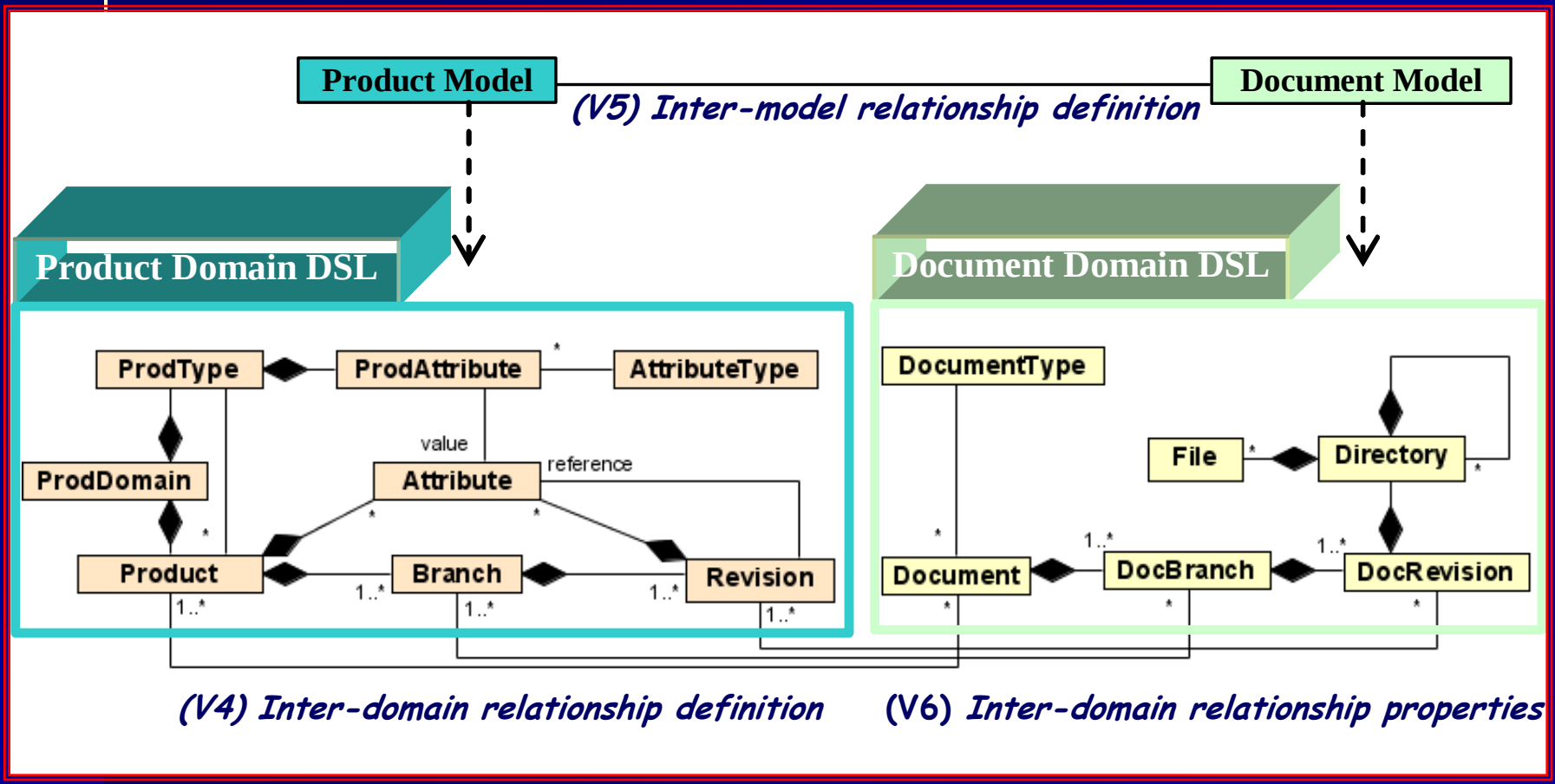
domains are narrow

Solution:

compose existing domains by establishing relationships

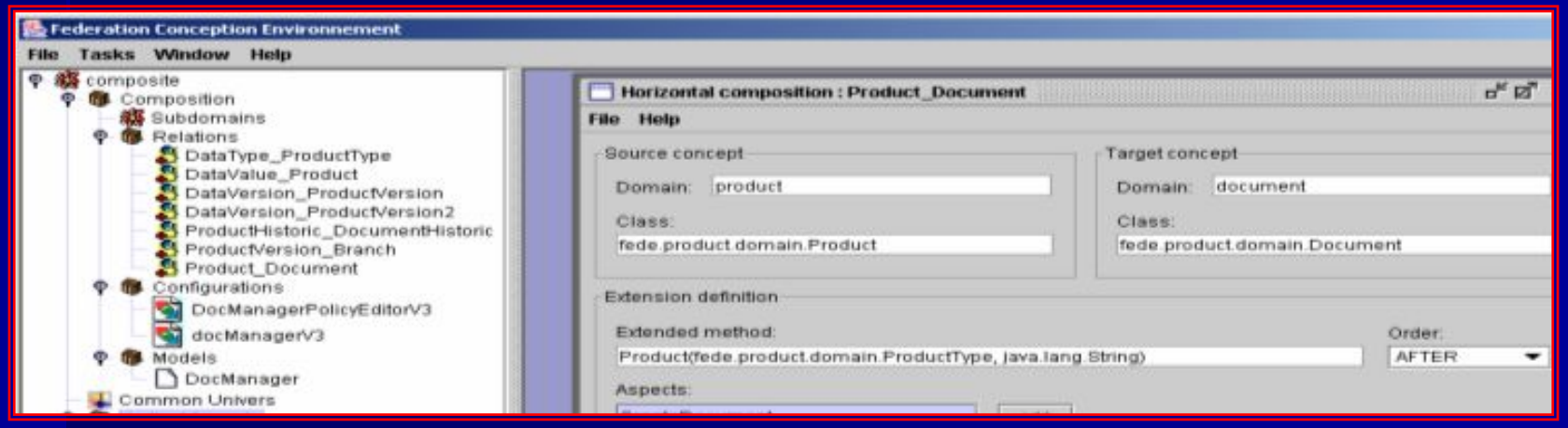


Variations inside the domain composition mechanism

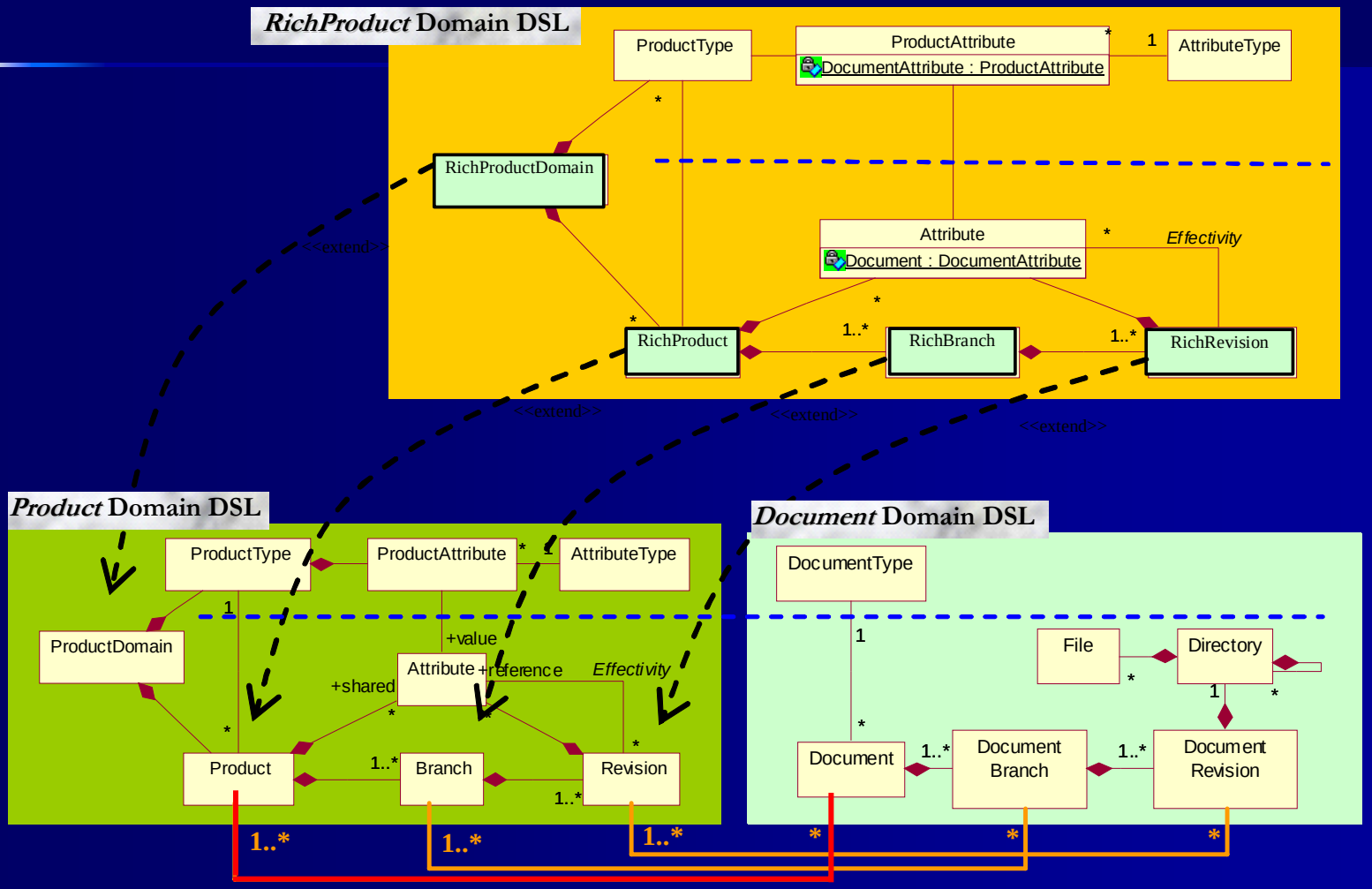


Inter-domain Relationship Definition

- establish *interactions* characteristic to the new composite domain
- establish *correspondences* - link “the same” concept found in both domains



Inter-domain Relationship Definition E.g. RichProduct

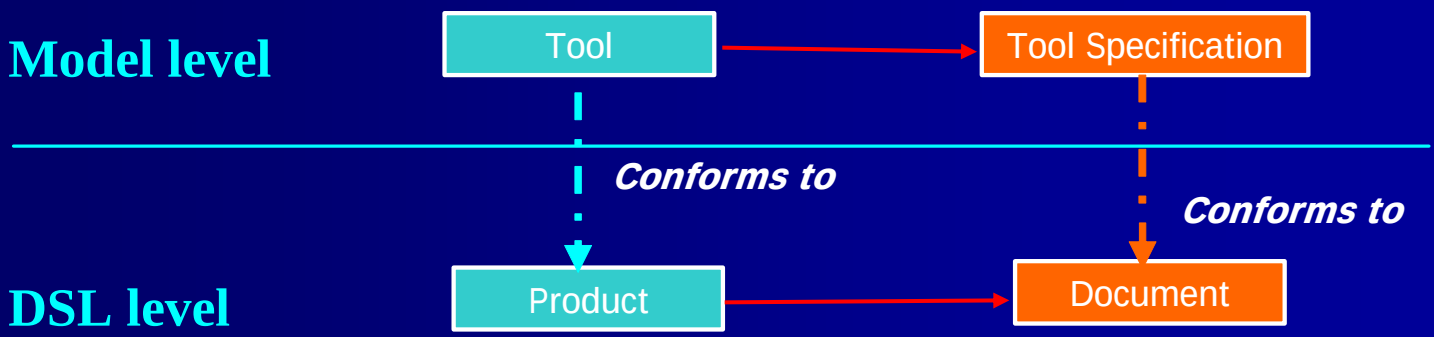


Implementation with AOP

```
public aspect RelH_Product_Document extends FedeAbstractAspect {

public RichProduct ProductDomain.createProductWithContent
(ProductType productType, Map attributesMap , String host, URI directory, Pattern exclusion)

{
   Product product = createProduct( productType,  attributesMap );
   DocumentDomain documentDomain = ( DocumentDomain ) MelusineCore
        .getDomainRoot ( DocumentDomain.DOMAIN_NAME );
   DocumentType documentType =
        getDocumentType(product.getProductType(), documentDomain);
   Document document = documentDomain.createDocument(documentType,
        product.getIdentificator(), host, directory, exclusion);
   product.link(document);
   return (RichProduct) product;
 }...

}
```

Inter-model Relationship Definition

- *automatically* - if there are enough criteria for matching the model elements
- *manually*



Relationships properties

- ***Destination Life Cycle Management***
  - *Source Independent*
  - *Source Dependent*
- ***Multiplicity***
- ***Link Creation Moment***
  - *Early, at instantiation*
  - *Late, at navigation*
- ***Persistence***
- ***Captures***

Destination Life Cycle Management

2 *types of domains*:

- *active* – a domain whose components are interactive and are influenced by non deterministic actors, like humans;
- *passive* – a domain that only performs actions

E.g.

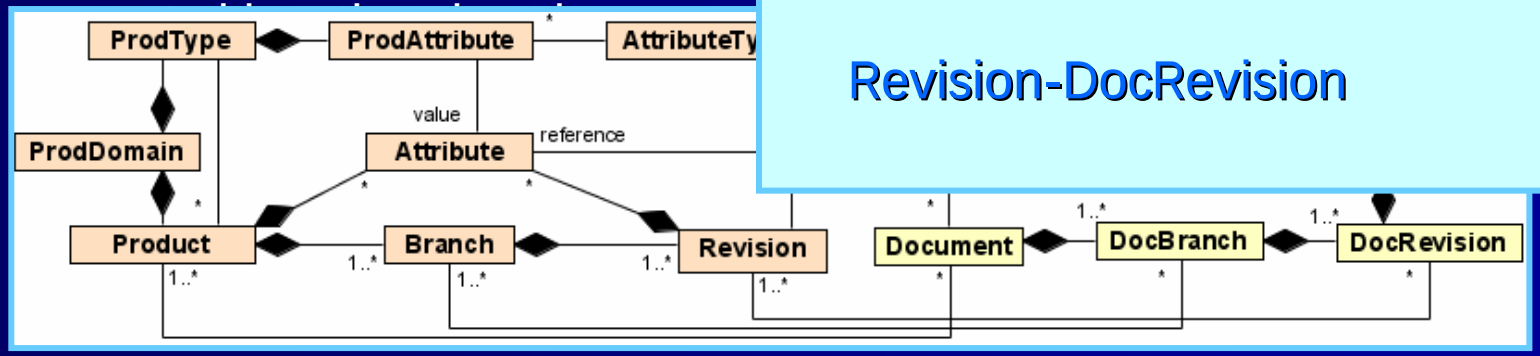
- *Product* domain active
- *Document* domain passive

Source dependent relationships:

Product-Document

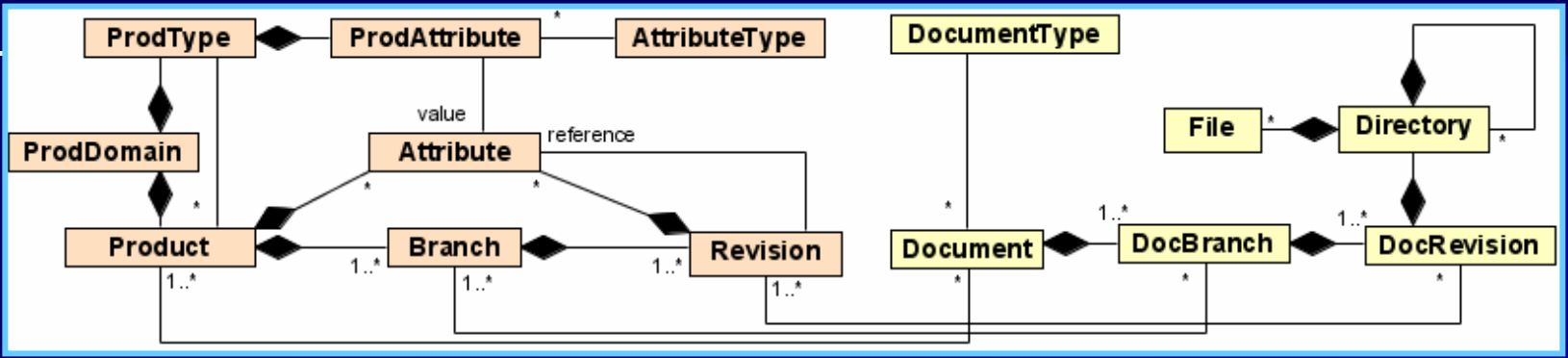
Branch-DocBranch

Revision-DocRevision



Multiplicity

E.g. Revision - DocRevision



1..* - a document revision may be linked to more product revisions

productRevision.setDocumentRevision (originalDocumentRevision);

1 - create a new document revision for each product revision, as a clone of the previous one

DocRevision clonedDocumentRevision = docBranch.createRevision(
originalDocumentRevision);
productRevision.setDocumentRevision (clonedDocumentRevision);

Summary

- Variation and composition with Model Driven Approaches
- Variation points in domain architecture
  - Application model definition
  - Feature selection
  - Component selection
- Variations inside the domain composition mechanism
  - inter-domain relationship definition
  - inter-model relationship definition
  - inter-domain relationship properties
- Conclusions

Conclusions

- The increase of size, complexity and evolution requires *more flexible composition mechanisms*.
- *Variation points*
  - inside the units of reuse
  - *in the composition mechanisms*

model-based composition

- *Domains* - high granularity units of reuse
  - any pair of domains may be composed
  - establishing inter-domain relationships
  - *relationship properties - variation points of the composition*