

# An Integration Concept for Complex Modelling Techniques

Benjamin Braatz  
Technische Universität Berlin

Workshop on Multi-Paradigm Modeling: Concepts and Tools  
Genova, Italy

3 October 2006

# Motivation

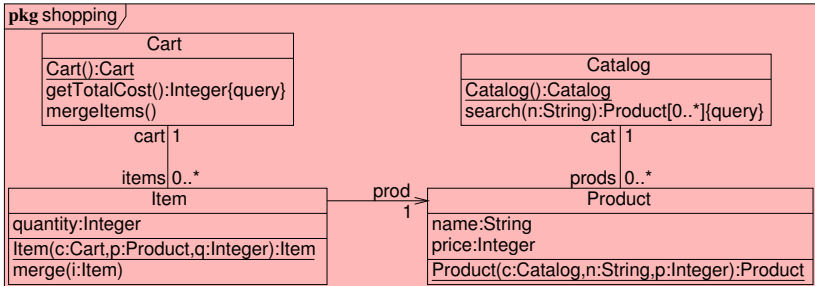
- Contemporary Modelling Techniques: Complex Integrated Techniques
- Need for Integrated Semantics to describe and analyse Interdependencies
- Rich Variety of Multi-Paradigm Behaviour Techniques mapped to Single Semantic Domain
- Semantic Domain should facilitate Verification and Code Generation

# CUML – The Complex Modelling Technique

- Compact, Comprehensive and Constructive UML Profile
- Class Diagrams for Structural Constraints
- OCL Constraints for Method Effects and Query Behaviour
- Transformation Rules for Local Changes
- Flowcharts for Algorithms
- Activity Diagrams for Workflows

# Example – Class Diagram

## Class Diagram: Specification of Structural Constraints



# Example – OCL Constraints

## Pre-Post Constraint: Descriptive Specification of Method Effect

```

context shopping::Cart::mergeItems()
post: self.items->forAll(i1,i2:Item|
    i1<>i2 implies i1.prod<>i2.prod)
    self.items@pre.prod->asSet()=
    self.items.prod->asSet()
  
```

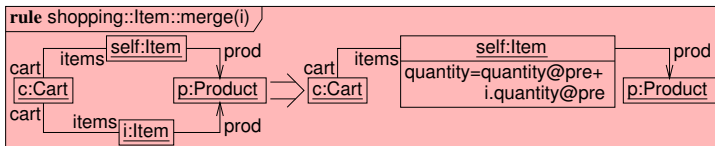
## Body Constraint: Complete Specification of Query Behaviour

```

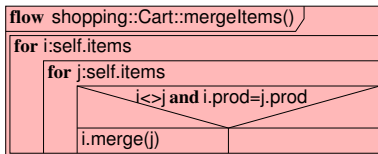
context shopping::Cart::getTotalCost()
body: self.items->iterate(i:Item,sum:Integer=0|
    sum+i.quantity*i.prod.price)
  
```

# Example – Transformation Rule and Flowchart

## Transformation Rule: Specification of Local Changes



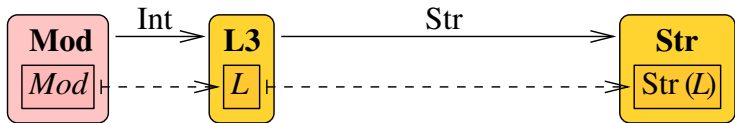
## Structured Flowchart: Specification of Algorithms



# L3 – The Low-Level Language

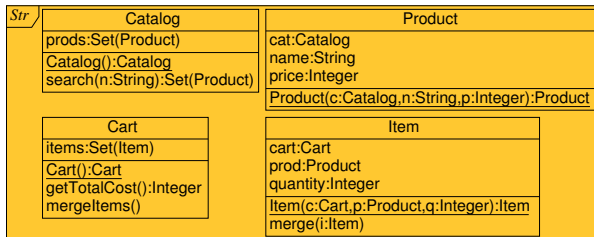
- Common Semantic Domain for CUML (and possible Extensions)
- Basis for Verification and Code Generation
- Strict Separation of Structure, Descriptive and Constructive Parts

# L3 – Structure Modelling

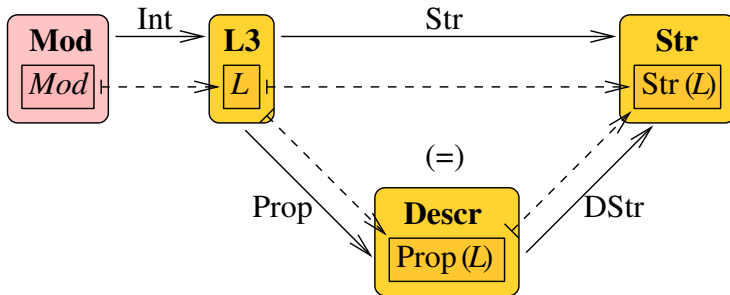


# Example – Structure

Structure Model: Pure Structure without Associations, Cardinalities, etc.



# L3 – Property Modelling

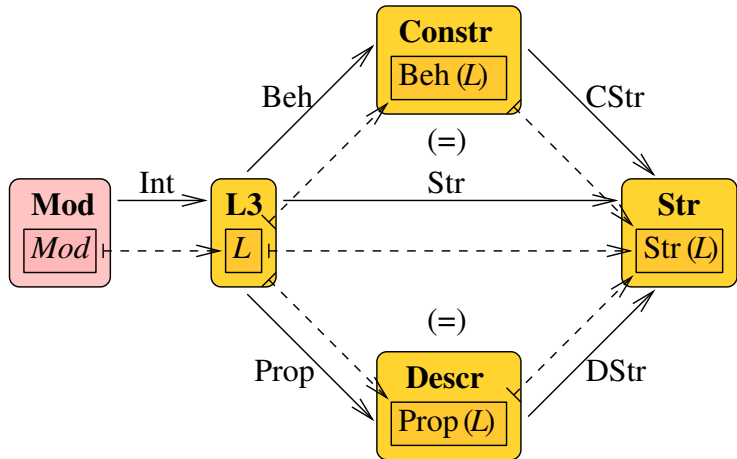


# Example – Properties

## Property Model: Associations, Cardinalities, etc. from Class Diagram, OCL Pre-Post Constraints

<i>Prop</i>	
<i>assocCartItems</i> : <b>for all</b> i:Item,c:Cart: i.cart=c <=> i <b>in</b> c.items	
<i>assocCatProds</i> : <b>for all</b> p:Product,c:Catalog: p.cat=c <=> p <b>in</b> c.prods	
<i>multCart</i> : <b>for all</b> i:Items: i.cart <b>defined</b>	
<i>multProd</i> : <b>for all</b> i:Items: i.prod <b>defined</b>	
<i>multCat</i> : <b>for all</b> p:Product: p.cat <b>defined</b>	
Cart::getTotalCost()	Catalog::search(n:String):Set(Product)
<b>query</b>	<b>query</b>
Item::Item(c:Cart,p:Product,q:Integer):Item	Product::Product(c:Catalog,n:String,p:Integer):Product
<b>pre</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat</i>	<b>pre</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat</i>
<b>post</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat</i>	<b>post</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat</i>
Item::merge(i:Item)	Cart::mergeItems()
<b>pre</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat, self.cart=i,self.prod=i, q1:=self.quantity,q2:=i.quantity</i>	<b>pre</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat, oldits:=self.items.prod</i>
<b>post</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat, self.quantity=q1+q2 not i in self.cart.items</i>	<b>post</b> : <i>assocCartItems,assocCatProds, multCart,multProd,multCat, for all i1,i2:Item: i1,i2 in self.items =&gt; (i1&lt;&gt;i2 =&gt; i1.prod&lt;&gt;i2.prod) for all p:Product: p in oldits &lt;=&gt; p in self.items.prod</i>

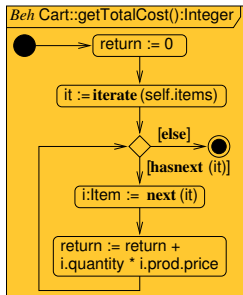
# L3 – Behaviour Modelling



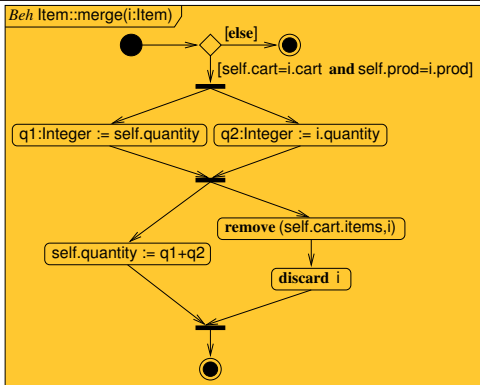
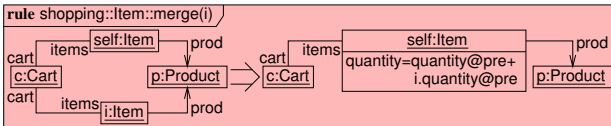
# Example – Behaviour for OCL Body Constraint

**Behaviour Model:** Executable, Constructive Behaviour for Different Modelling Techniques

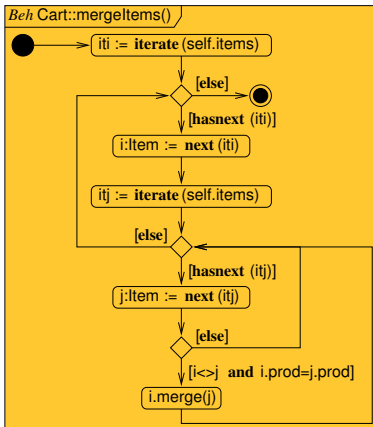
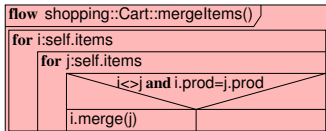
**context** shopping::Cart::getTotalCost()  
**body:** self.items->iterate(i:Item,sum:Integer=0|  
 sum+i.quantity\*i.prod.price)



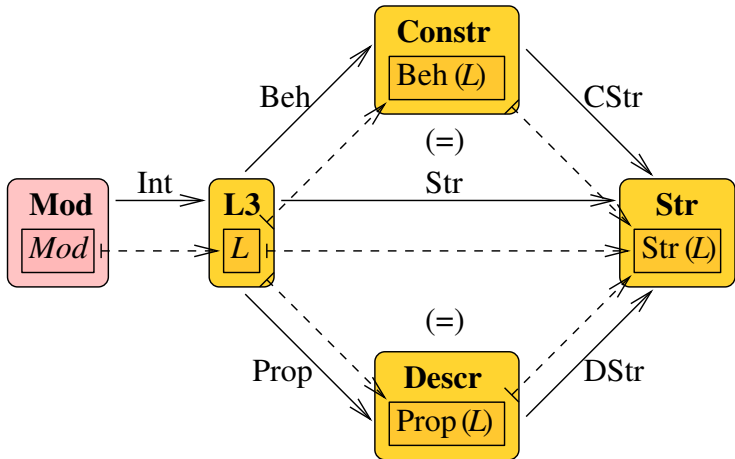
# Example – Behaviour for Transformation Rule



# Example – Behaviour for Structured Flowchart



# Summary – Abstract Integration Concept



# Comparison to Other Approaches

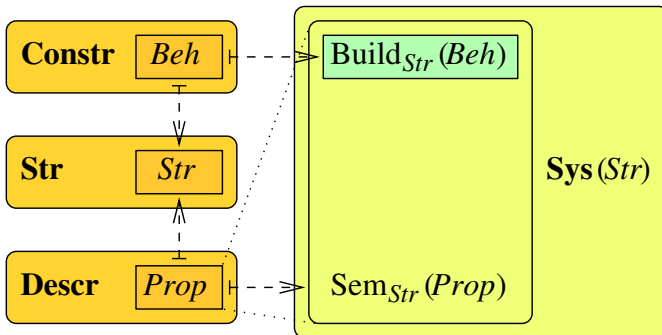
## Abstract State Machines and Process Algebras:

- Unintuitive Encoding of OO Concepts necessary
- Detour on the Way to Code Generation
- Advantage: Tools and Theory readily available

## Petri Nets:

- No existing Variant supports all OO Features
- New Formalism necessary, as well

# Future Work – Semantics



# Future Work

## Formal Syntax:

- UML Profile for CUML
- Meta Model for L3
- Model Transformation from CUML to L3

## Formal Semantics:

- Behaviour Semantics by Simulating the Execution
- Property Semantics by Satisfaction Relation
- Verification Techniques?

## Code Generation:

- Generation of Java Byte Code Classes from Behaviour
- Generation of Test Cases from Properties
- Other Target Platforms: .NET-CIL, ...