

Automatic generation of behavioral code - too ambitious or even unwanted?



Gregor Engels

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft



University of Twente, The Netherlands
23 June 2009



Professional Activities

Gregor Engels

- University of Paderborn
 - Head of Board s-lab (Software Quality Lab)
 - Member Board International Graduate School (IGS)
- Scientific Director Capgemini sd&m Research, Munich
- International
 - Member Founding Board Informatics Europe (<http://www.informatics-europe.org/>)
 - Steering Committee MODELS Conference, Visual Languages and Human-Centric Computing (VLHCC), International Conference on Graph Transformations (ICGT)



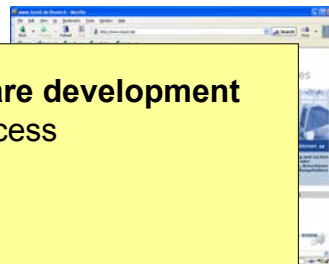
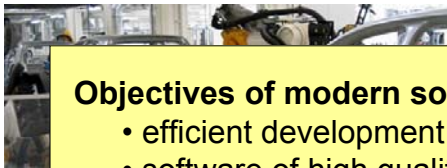
Software is everywhere!



Gregor Engels, BM-MDA 2009

3

Software is everywhere!



Objectives of modern software development

- efficient development process
- software of high quality
 - **correct**
 - adaptable
 - reusable
 - ...

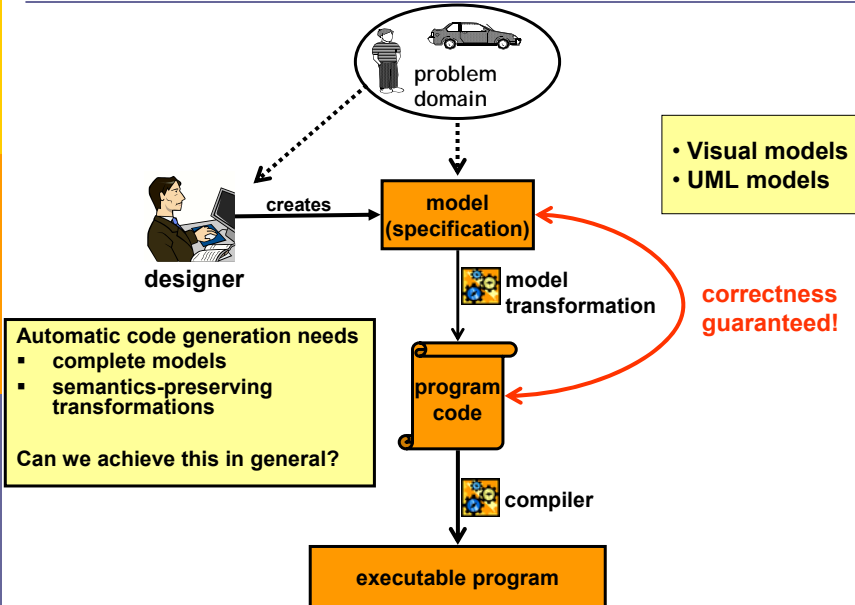
Solution / Dream (?)

- Model-driven Development (MDD)
 - Model-driven Architecture (MDA)
- Service-oriented Architecture (SOA)

Gregor Engels, BM-MDA 2009

4

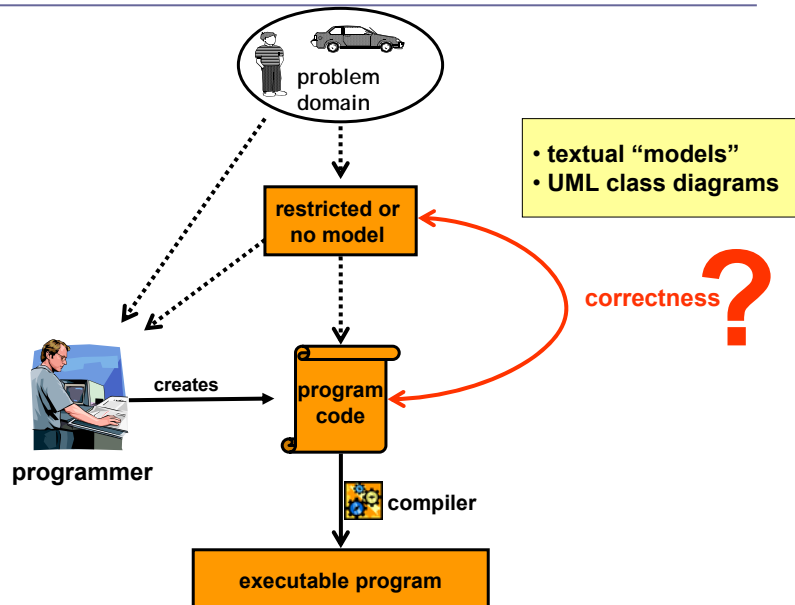
Model-driven Software Development (MDD)



Gregor Engels, BM-MDA 2009

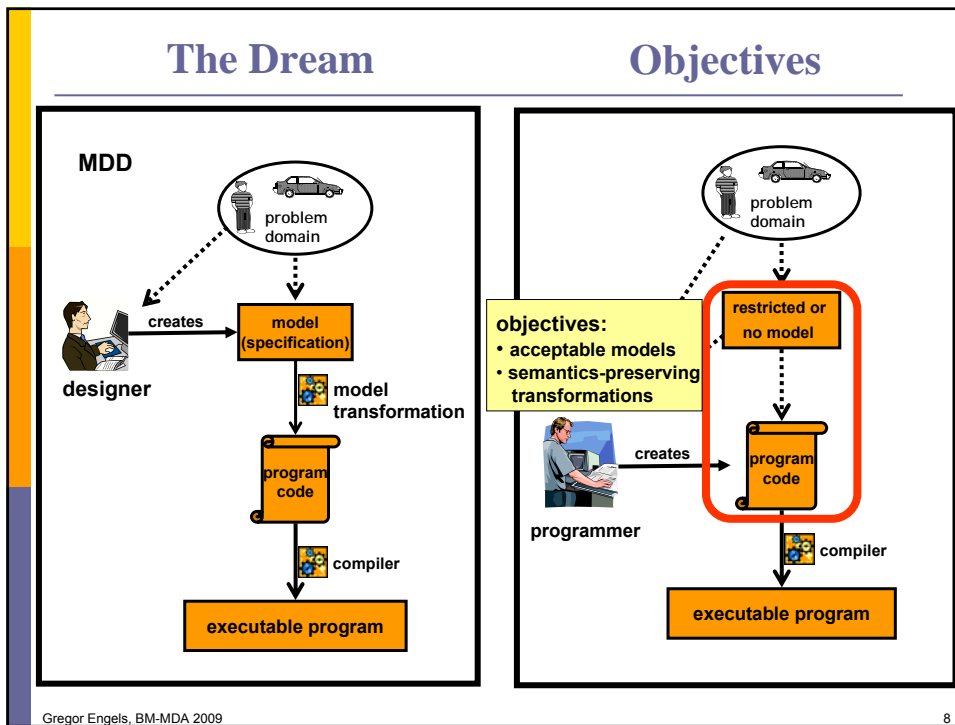
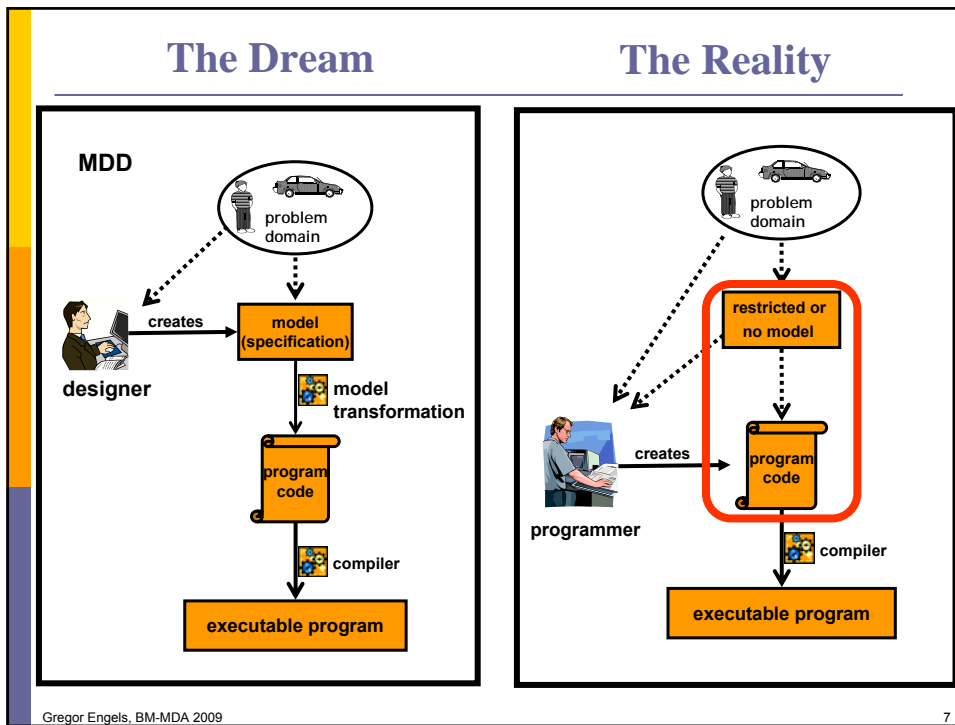
5

Traditional Manual Software Development

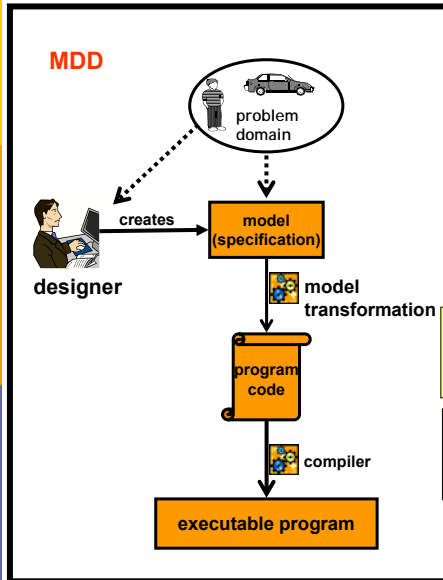


Gregor Engels, BM-MDA 2009

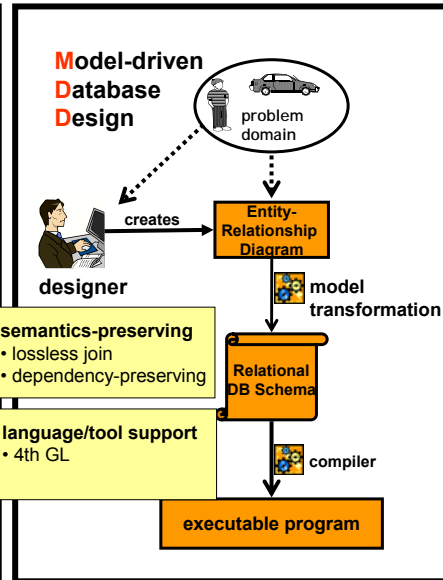
6



Only a Dream?



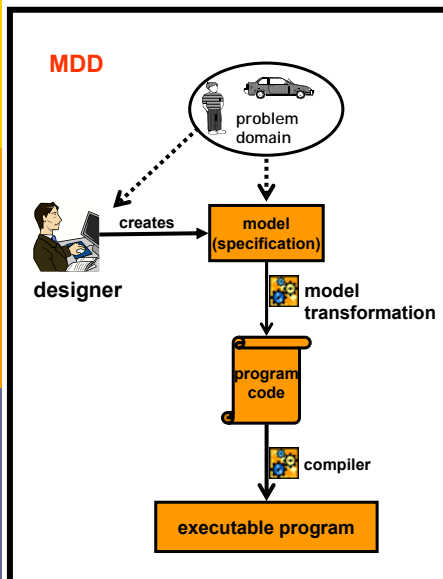
No! Database Design!



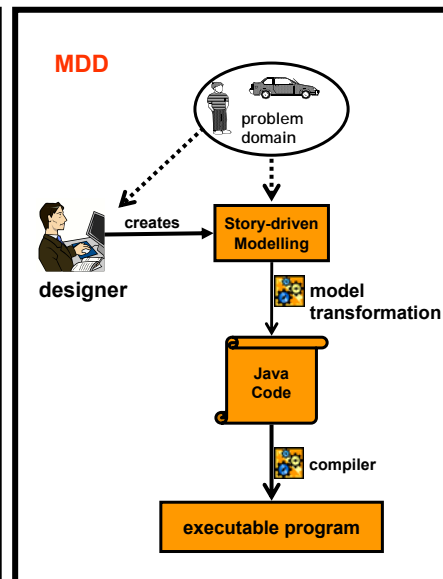
Gregor Engels, BM-MDA 2009

9

Only a Dream?



No!?! Fujaba



Gregor Engels, BM-MDA 2009

10

The Fujaba Approach

Fujaba - From UML to Java and Back Again

- Open Source UML CASE Tool Project
- started in 1997 at the University of Paderborn
- Wilhelm Schäfer (University of Paderborn)
 - <http://wwwcs.uni-paderborn.de/cs/fujaba/>
- Albert Zündorf (University of Kassel)
 - <http://www.se.eecs.uni-kassel.de/se/index.php?fujabaproject>
- Holger Giese (HPI Potsdam)
 - <http://www.hpi.uni-potsdam.de/giese/projekte/fujaba.html>

Fujaba – main features

- based on UML
- extended by Story Driven Modeling (SDM)
- deploys graph transformation platform

- combines UML class diagrams and UML behavior diagrams (Story Diagrams) to a powerful, easy to use, yet formal system design and specification language

- generation of Java source code out of the whole design which results in an executable prototype
- re-engineering so that Java source code can be parsed and represented within UML

Behavior modeling with Fujaba

- **object diagrams**
 - are used as typed graphs
 - are the base to define graph transformation rules
- **control flow** described by UML activity diagrams
- **object behavior** described by graph transformation rules

- Running example:
 - autonomous shuttles



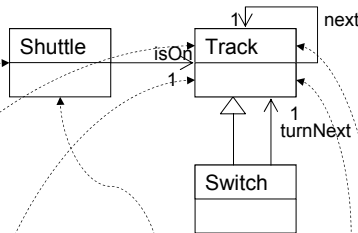
Gregor Engels, BM-MDA 2009

Provided by: Software Engineering Group, Prof. Dr. Wilhelm Schäfer

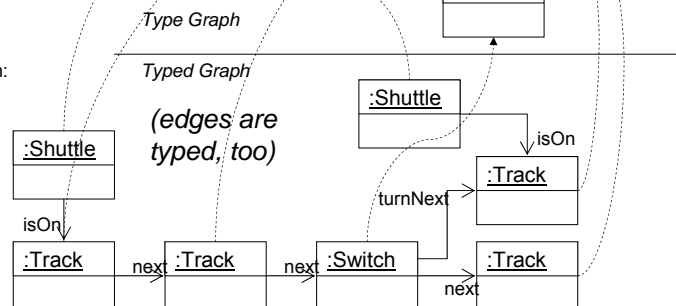
13

Object diagrams as typed graphs

Class Diagram:



Object graph:



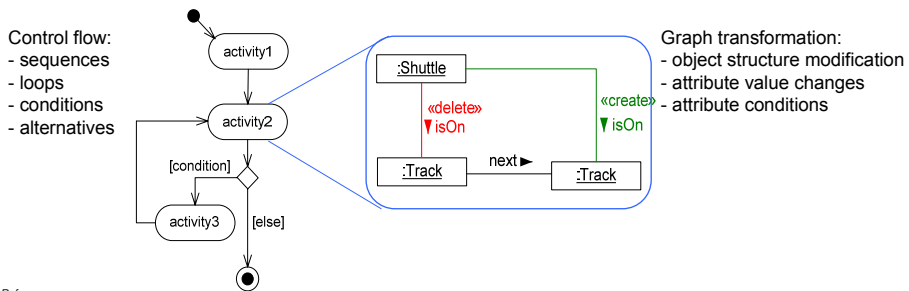
Gregor Engels, BM-MDA 2009

Provided by: Software Engineering Group, Prof. Dr. Wilhelm Schäfer

14

Story Diagrams

- combine:
 - UML Activity diagrams specify **control flow**
 - Story patterns specify **graph transformation rules**
- specify (more complex) operations on object structures
- have formally defined semantics

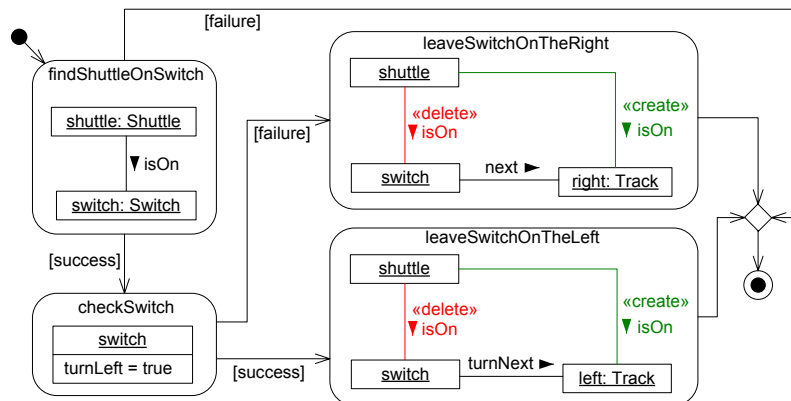


References:
 Albert Zündorf: "Rigorous Object Oriented Software Development", Habilitation draft, 2001
 T. Fischer et al.: "Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language", in Proc. of the 6th International TAGT Workshop, LNCS 1764, pp. 296-309, Springer Verlag, 1998
 Gregor Engels, BM-MDA 2009 Provided by: Software Engineering Group, Prof. Dr. Wilhelm Schäfer

15

Story Diagrams - Control Flow Example

- Modeling alternatives with Story Diagrams



Guards *success* and *failure*:

Success: Transition fires iff previous activity has been executed successfully, i.e. pattern completely matched and all conditions satisfied.

Gregor Engels, BM-MDA 2009

Provided by: Software Engineering Group, Prof. Dr. Wilhelm Schäfer

16

Motivation for code generation

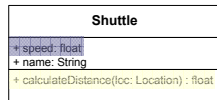
- model-based software engineering
 - (mainly) working on design level (with models)
 - reduces complexity
 - automatically generate code
 - reduces implementation effort
 - reduces implementation errors
- maintainability and readability of generated code
 - reason:* sometimes code has to be reviewed or adapted manually
 - integration with frameworks, platforms, etc.
 - manual code optimization (e.g. for efficiency)

Code Generation with Fujaba

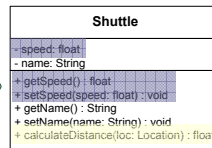
- Structural diagrams (Class diagrams)
 - Inheritance
 - Associations
- Behavioral diagrams which refine class diagrams
 - Activity diagrams (usually for control flow of methods)
 - Story diagrams (refine activity diagrams)

Code generation of structural information

Design Level
Class Diagram



Code Level
Class Diagram



Generated
Source Code

```

class Shuttle {
    public Shuttle() {}

    private float speed;
    public float getSpeed() {
        return this.speed;
    }
    public void setSpeed(float speed) {
        this.speed = speed;
    }

    private String name;
    public String getName() {
        return this.name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public float calculateDistance(Location loc) {
        ...
    }
}
    
```

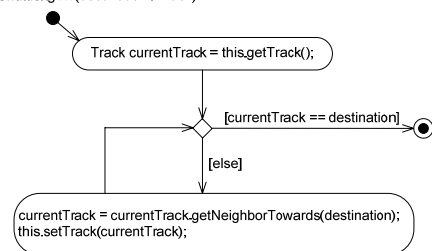
Code generation defined for:

- inheritance
- bidirectional associations
- composition and aggregation
- ...

Code generation of activity diagrams

- Activity Diagram specifies single method
- (well-formed) control flow maps to (Java) control structures
- actions / guards translate directly to code (makes diagrams language dependent)

Shuttle: goto(destination : Track)

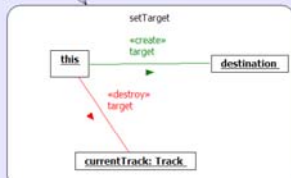


```

class Shuttle {
    public void goto(Track destination) {
        Track currentTrack = this.getTrack();
        while(!currentTrack == destination){
            currentTrack = currentTrack.getNeighborTowards(destination);
            this.setTrack(currentTrack);
        }
    }
}
    
```

Generated Java code for story diagrams

Shuttle.goto(destination: Track): Void



JavaSDM.ensure(...) throws a JavaSDM-Exception if the given expression is not true

```
public class Shuttle {
    ...

    public void gotoTrack(Track destination)
    {
        boolean fujaba_Success = false;
        Track currentTrack = null;
        try {
            fujaba_Success = false;
            // check object destination is really bound
            JavaSDM.ensure(destination != null);
            // bind currentTrack: Track
            currentTrack = this.getTarget();
            JavaSDM.ensure(currentTrack != null);
            // check isomorphic binding
            JavaSDM.ensure(!(destination.equals(currentTrack)));
            // delete link
            this.setTarget(null);
            // create link
            this.setTarget(destination);
            fujaba_Success = true ;
        }
        catch (JavaSDMException fujaba_InternalException) {
            fujaba_Success = false ;
        }
    }
    ...
}
```

Summary of Fujaba Approach

- complete structure needs to be modelled to provide object diagrams
- control flow definition is programming language dependent
- code generation must embed methods and its behavior into structural skeleton
- syntax and semantics definition of story diagrams is mapped to programming language instructions

Evaluation of Fujaba

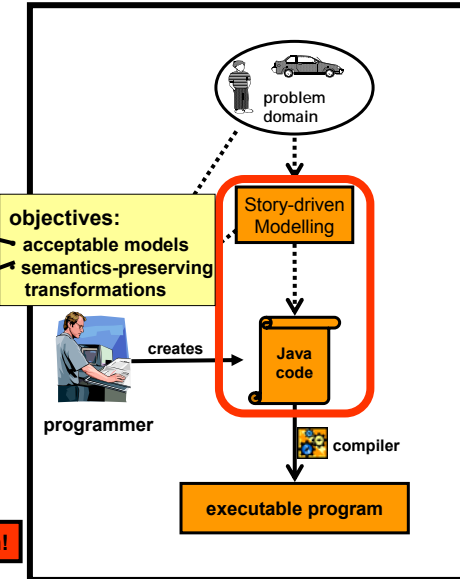
acceptable model?

- detailed, fine-grained specification
- programming-language dependent

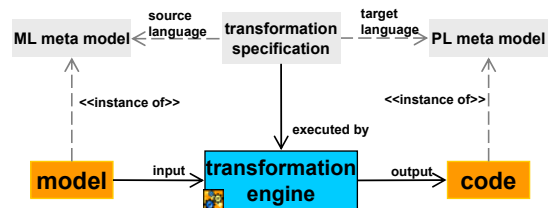
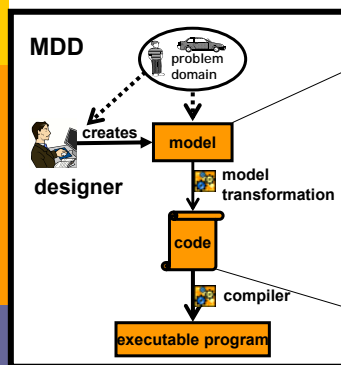
semantics-preserving transformation?

- compiler-based semantics of Fujaba
- transformation as semantics definition

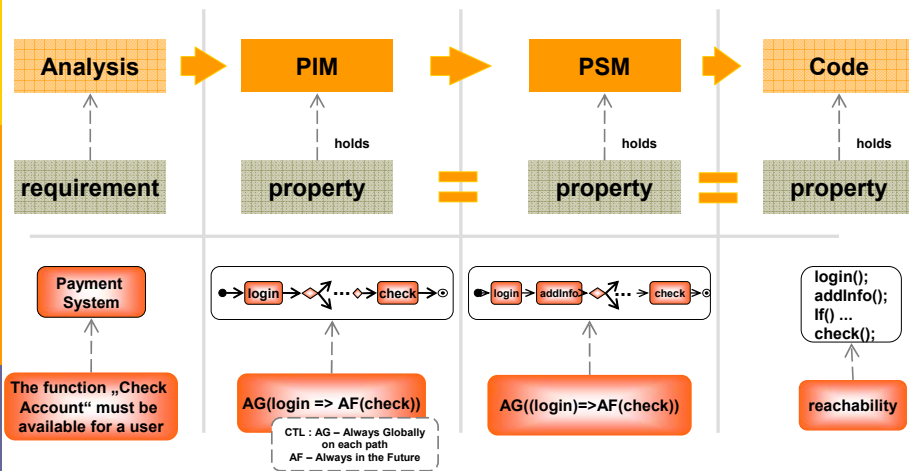
Not a semantics-preserving transformation!



Semantics-preserving model transformation



What does semantics-preserving transformation mean?



Semantics preserving means that the resulting model/code still conforms to the initial requirements

Correctness of transformation in the sense of semantics preserving

Gregor Engels, BM-MDA 2009

25

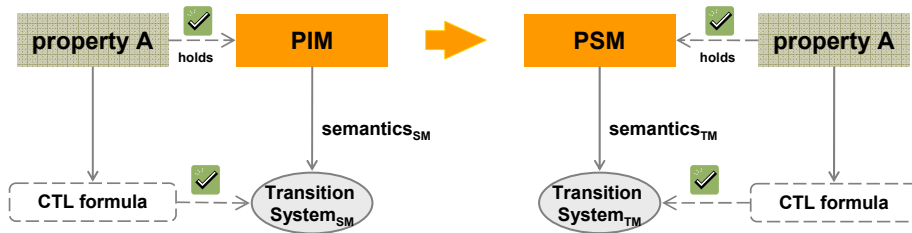
What is needed for a proof?

- formal semantics of behavioral models
- formalized properties
- formal transformation

Gregor Engels, BM-MDA 2009

26

Semantics-preserving model transformation

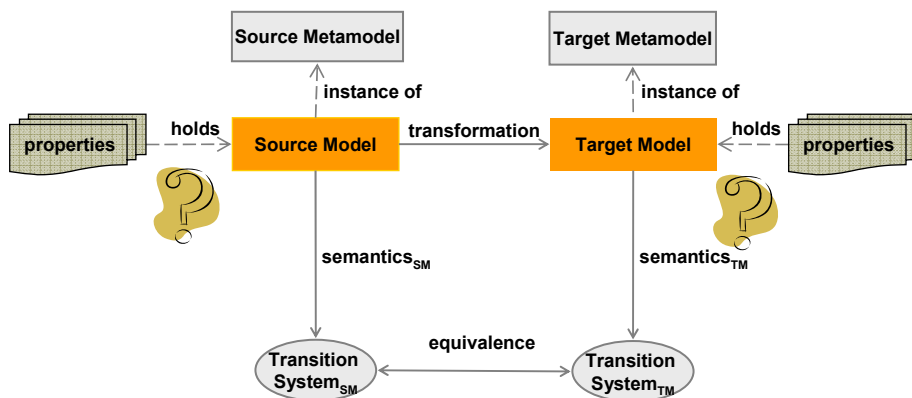


- semantics-preserving means that **all** properties of the source model hold for the target model, too

Gregor Engels, BM-MDA 2009

27

The way to show „semantics-preservation“



Transition systems are equivalent \Leftrightarrow All properties that hold on a source model also hold on a target model

Gregor Engels, BM-MDA 2009

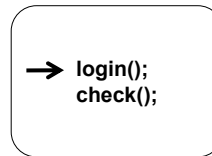
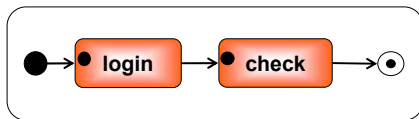
28

Why is it difficult to prove this?

- languages are different – not every element has a counterpart in the other language.

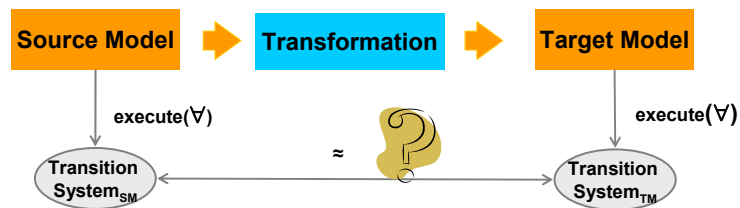
Example:

- UML Activity Diagram
 - several tokens may traverse through a diagram
- Textual Language (e.g., Java)
 - only one program counter



How to map the semantics in this case?

Proof of bisimulation

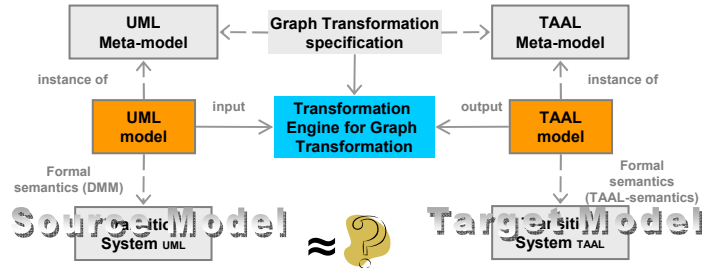


- Goal: to show that transition system (TS) of each source model is bisimilar to the TS of target model received as result of transformation

Double Check Approach

Joint Project:

- University of Paderborn: G. Engels, M. Semenyak, Chr. Soltenborn, H. Wehrheim
- TU Twente: A. Rensink (Groove approach)



- Bisimilar relation between Transition Systems
 - Systems behave in the same way in the sense that one system simulates the other and vice-versa
- G. Engels, A. Kleppe, A. Rensink, M. Semenyak, Chr. Soltenborn, H. Wehrheim: From UML Activities to TAAL - Towards Behaviour-Preserving Model Transformations. Proceedings of ECMDA 2008, LNCS 5095, pp. 94-109, Springer-Verlag 2008

Gregor Engels, BM-MDA 2009

31

Evaluation

acceptable model?

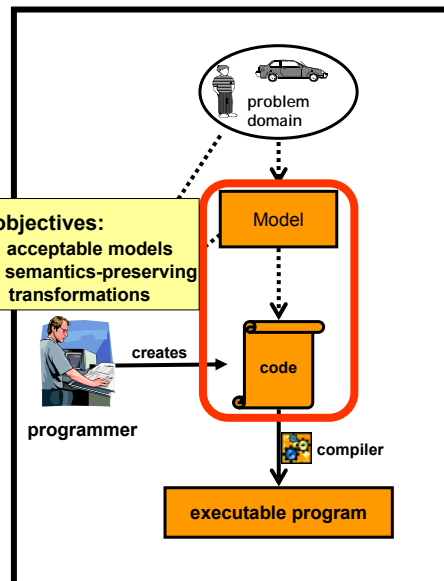
- detailed, fine-grained specification
- programming-language dependent

semantics-preserving transformation?

- hard to prove
- needs semantics definition of languages

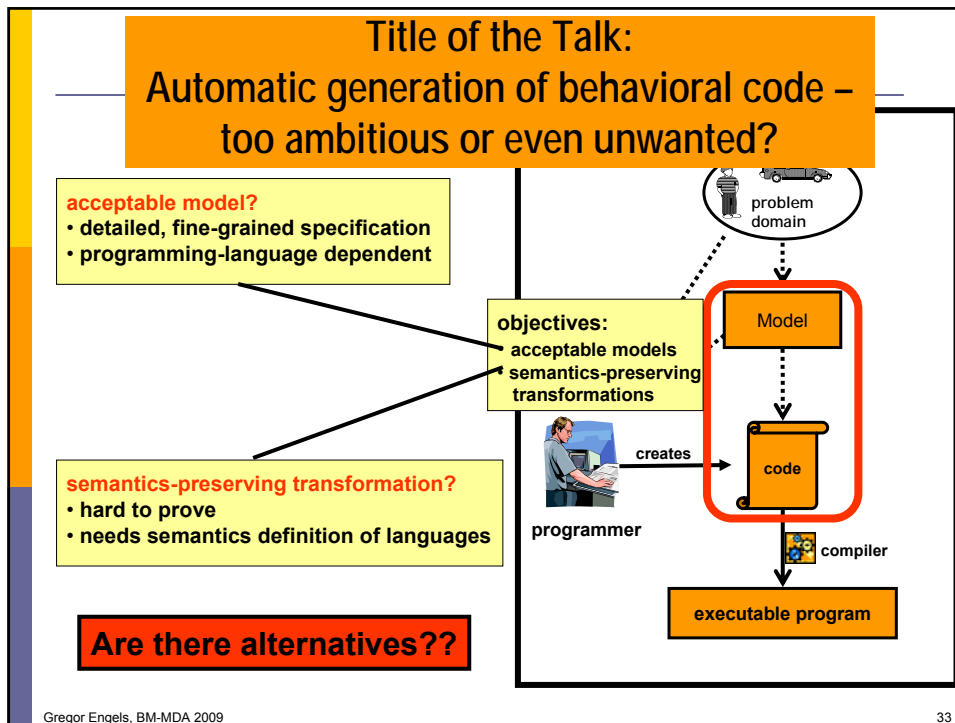
objectives:

- acceptable models
- semantics-preserving transformations



Gregor Engels, BM-MDA 2009

32



- ## Alternative Approaches
- Use of partial models
 - visual contracts
 - Model-driven Monitoring (MdM)
 - Model-based Testing (MbT)
- Gregor Engels, BM-MDA 2009 34

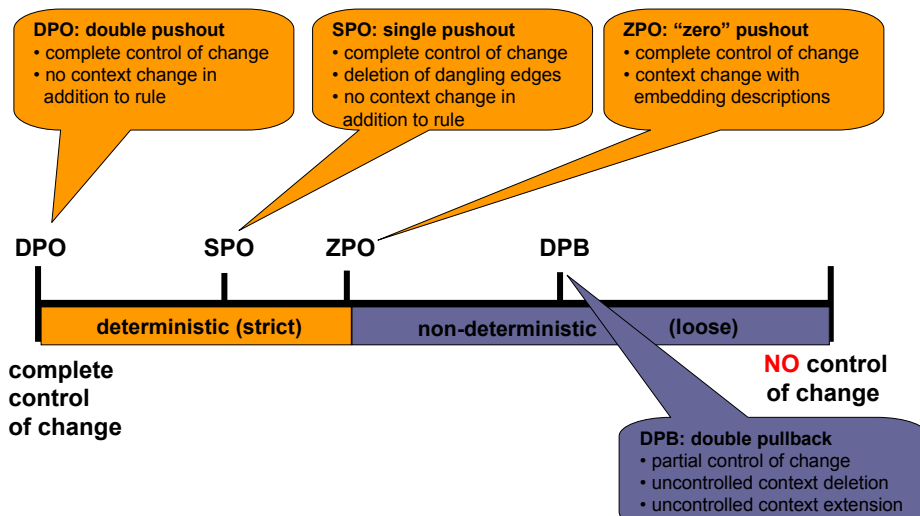
Alternative Approaches

- **Use of partial models**
 - **visual contracts**
- Model-driven Monitoring (Mdm)
- Model-based Testing (MbT)

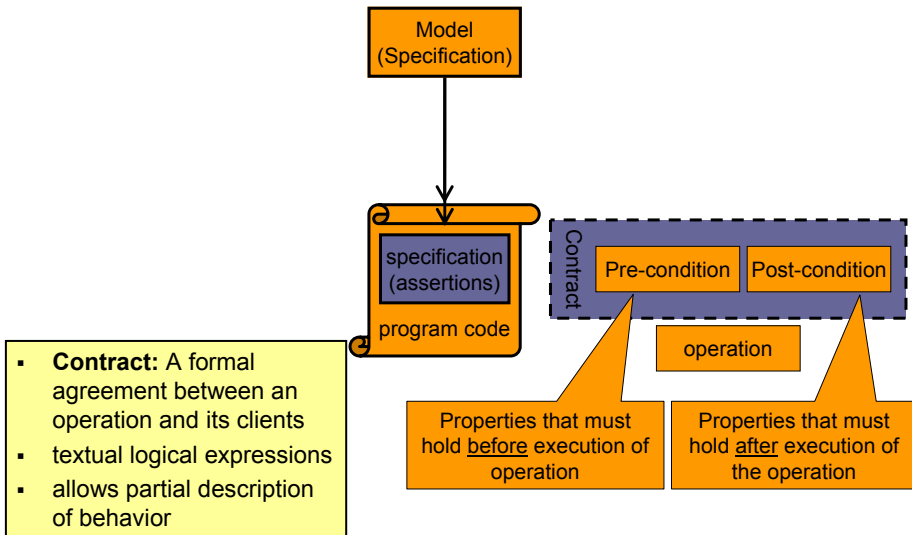
Graph-Transformation-Based Modeling



Modelling Alternatives



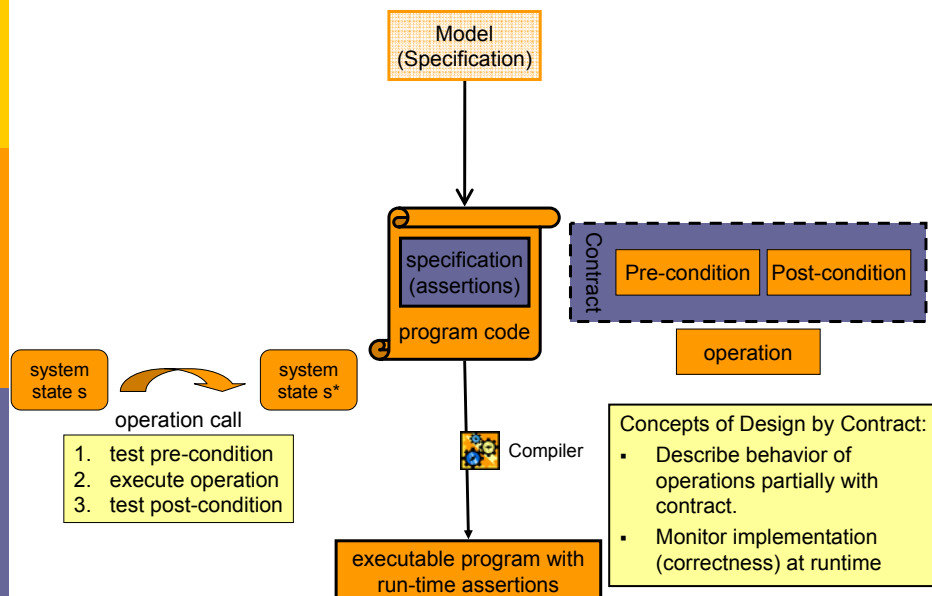
Reuse of an old idea: Design by Contract (Eiffel)



Gregor Engels, BM-MDA 2009

37

Reuse of an old idea: Design by Contract (Eiffel)



Gregor Engels, BM-MDA 2009

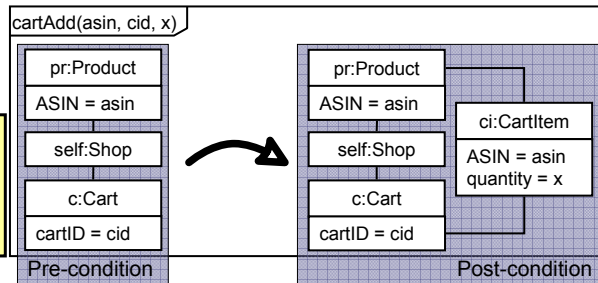
38

Visual Contract Example

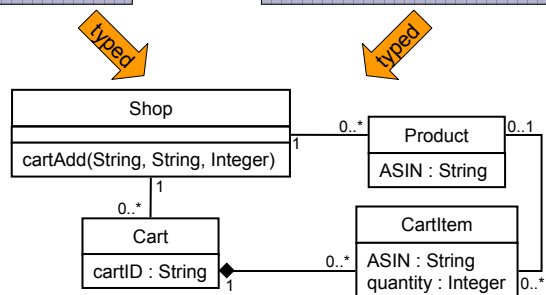
Behavioral Aspects:
Visual Contract

Visual Contracts:

- define "minimal" requirements
- semantic concept of loose graph transitions
- formalized by double-pullback



Static Aspects:
Class Diagram



Gregor Engels, BM-MDA 2009

39

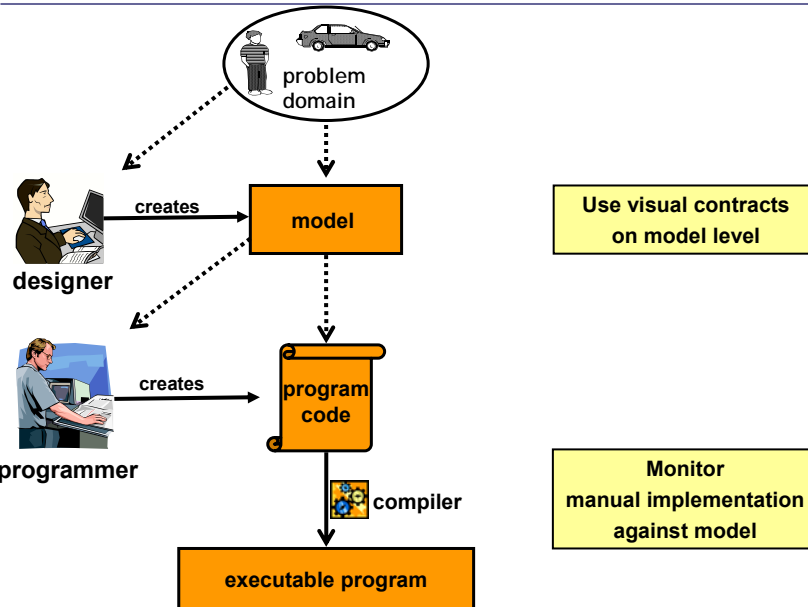
Alternative Approaches

- Use of partial models
 - visual contracts
- **Model-driven Monitoring (MdM)**
- Model-based Testing (MbT)

Gregor Engels, BM-MDA 2009

40

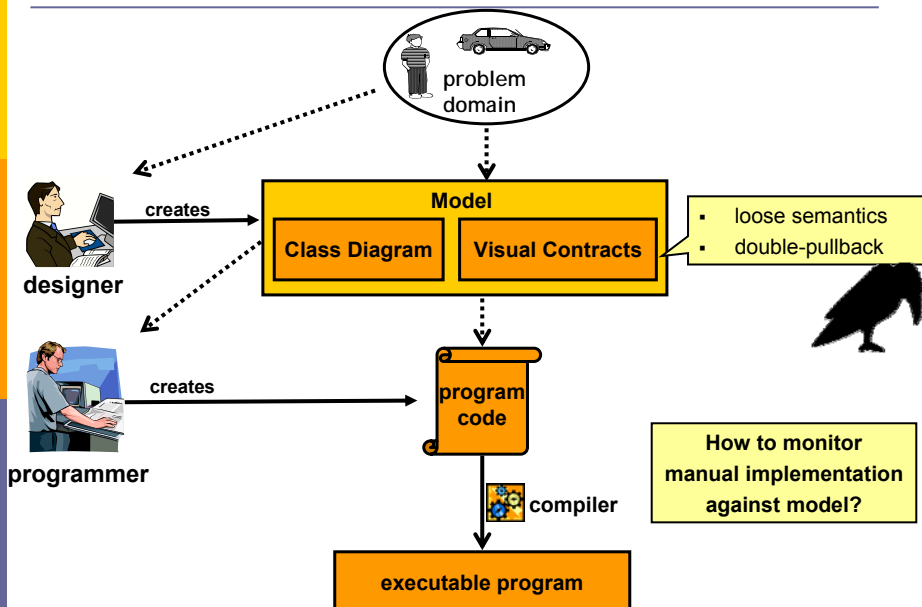
Model-Driven Monitoring



Gregor Engels, BM-MDA 2009

41

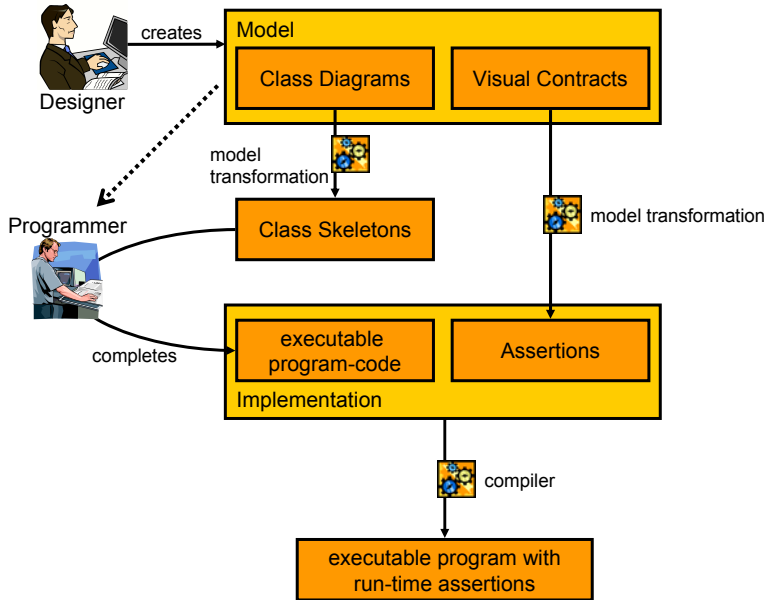
Visual Contracts



Gregor Engels, BM-MDA 2009

42

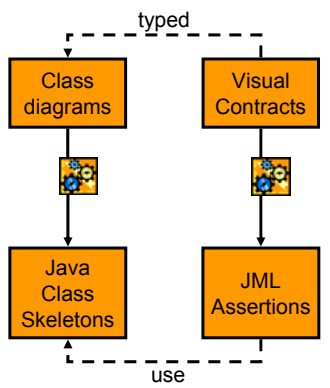
Model-Driven Monitoring



Gregor Engels, BM-MDA 2009

43

Transformation into Java and JML



JML (Java Modeling Language)

- Design by Contract Extension for Java
- JML Assertions are based on Java expressions

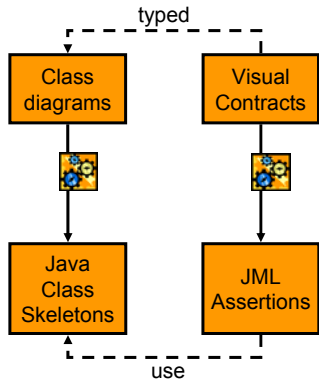
```

/*@ public normal_behavior
  @ requires (\exists Product pr;
  @   getProducts().values().contains(pr);
  @   pr.getASIN().equals(asin)
  @   && (\exists Cart c;
  @     getCarts().values().contains(c);
  @     c.getCartID().equals(cid)));
  @
  @ ensures (\exists Product pr;
  @   getProducts().values().contains(pr);
  @   pr.getASIN().equals(asin)
  @   && (\exists Cart c;
  @     getCarts().values().contains(c);
  @     c.getCartID().equals(cid)
  @   && (\exists CartItem citem;
  @     c.getItems().values().contains(citem);
  @     item.getASIN().equals(asin)
  @     && citem.getQuantity() == x));
  @*/
public String cartAdd (String asin, String cid, int x);
  
```

Gregor Engels, BM-MDA 2009

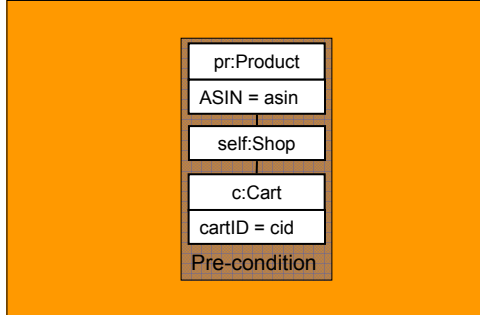
44

Transformation into Java and JML



```

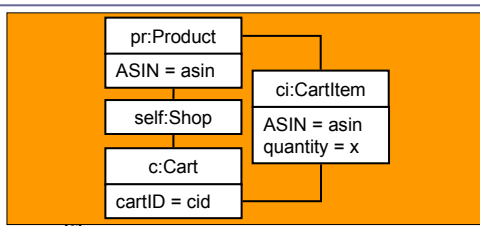
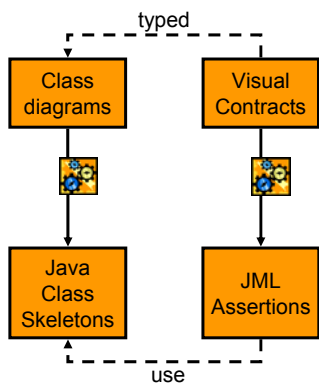
    /*@ public normal_behavior
    @ requires (\exists Product pr;
    @   getProducts().values().contains(pr);
    @   pr.getASIN().equals(asin)
    @   && (\exists Cart c;
    @     getCarts().values().contains(c);
    @     c.getCartID().equals(cid)));
    @
  
```



JML (Java Modeling Language)

- Design by Contract Extension for Java
- JML Assertions are based on Java expressions

Transformation into Java and JML



```

    @
    @ ensures (\exists Product pr;
    @   getProducts().values().contains(pr);
    @   pr.getASIN().equals(asin)
    @   && (\exists Cart c;
    @     getCarts().values().contains(c);
    @     c.getCartID().equals(cid)
    @     && (\exists CartItem citem;
    @       c.getCartItems().values().contains(citem);
    @       item.getASIN().equals(asin)
    @       && citem.getQuantity() == x));
    @
  
```

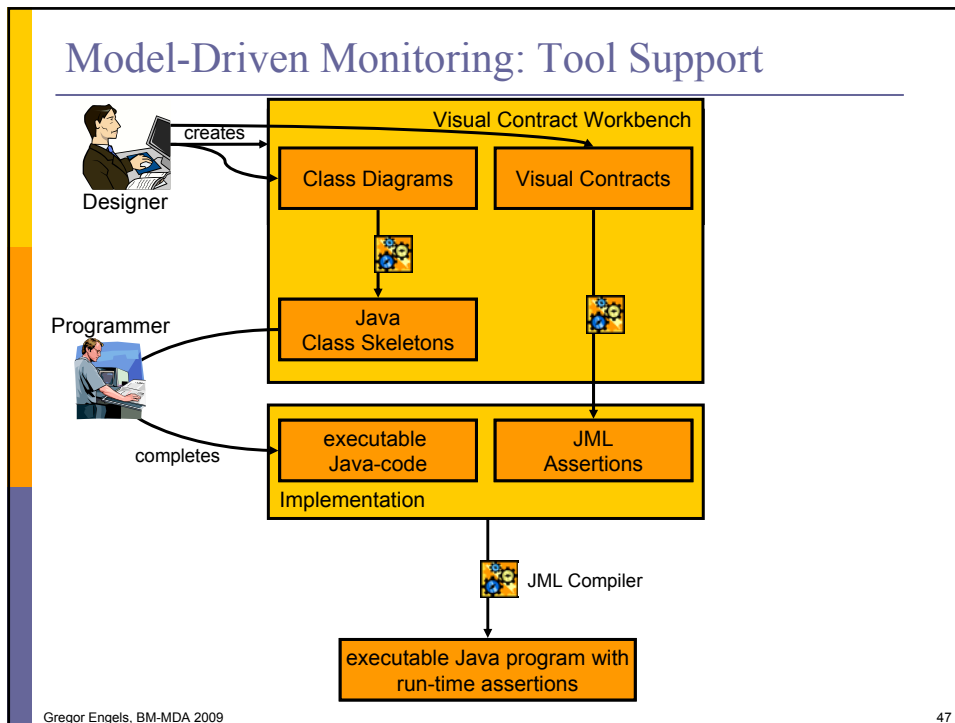
```

    /*@
    public String cartAdd (String asin, String cid, int x);
  
```

JML (Java Modeling Language)

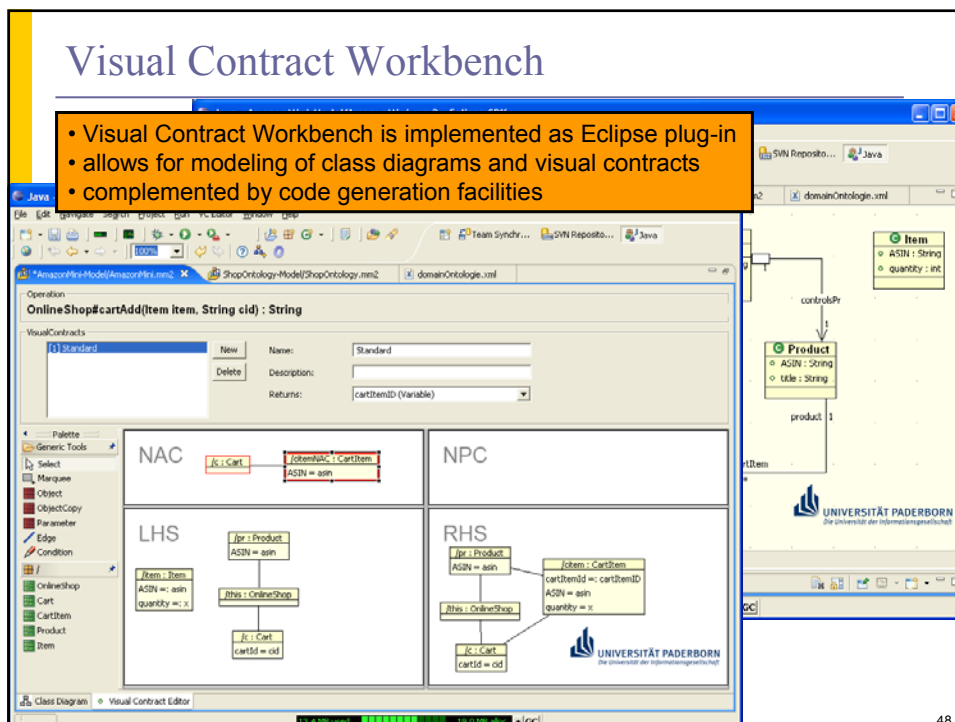
- Design by Contract Extension for Java
- JML Assertions are based on Java expressions

Model-Driven Monitoring: Tool Support



Visual Contract Workbench

- Visual Contract Workbench is implemented as Eclipse plug-in
- allows for modeling of class diagrams and visual contracts
- complemented by code generation facilities



Alternative Approaches

- Use of partial models
 - visual contracts
- **Model-driven Monitoring (MdM)**
- Model-based Testing (MbT)

Quality check
by accident!!

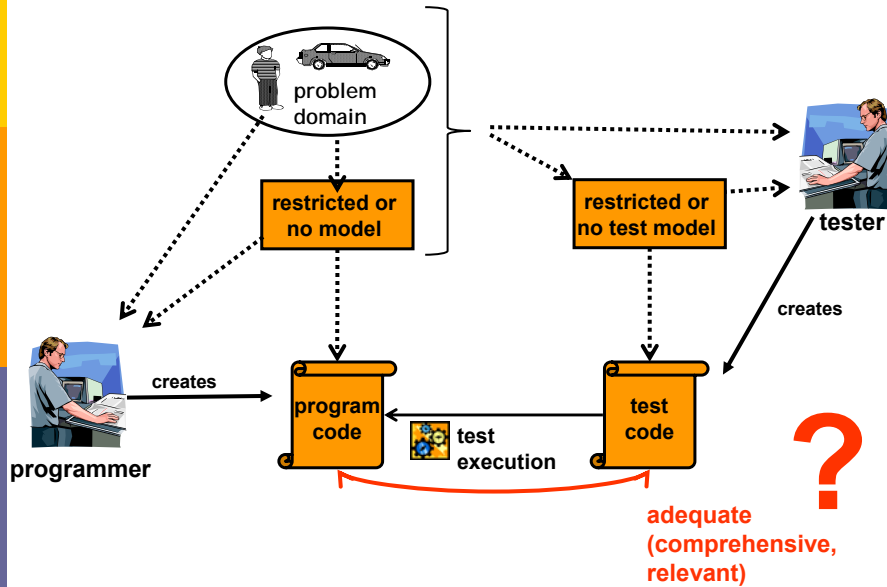
Alternative Approaches

- Use of partial models
 - visual contracts
- **Model-driven Monitoring (MdM)**
- **Model-based Testing (MbT)**

Quality check
by accident!!

Planned
quality check!!

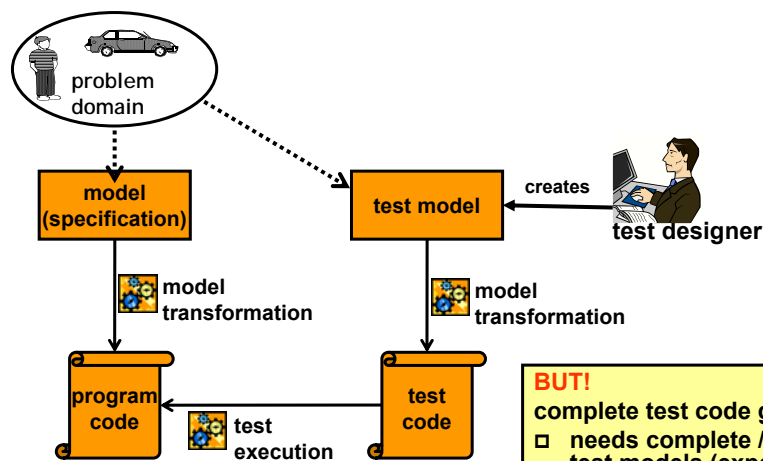
Traditional Manual Software Testing



Gregor Engels, BM-MDA 2009

51

Model-based Testing (MbT): Scenario 1

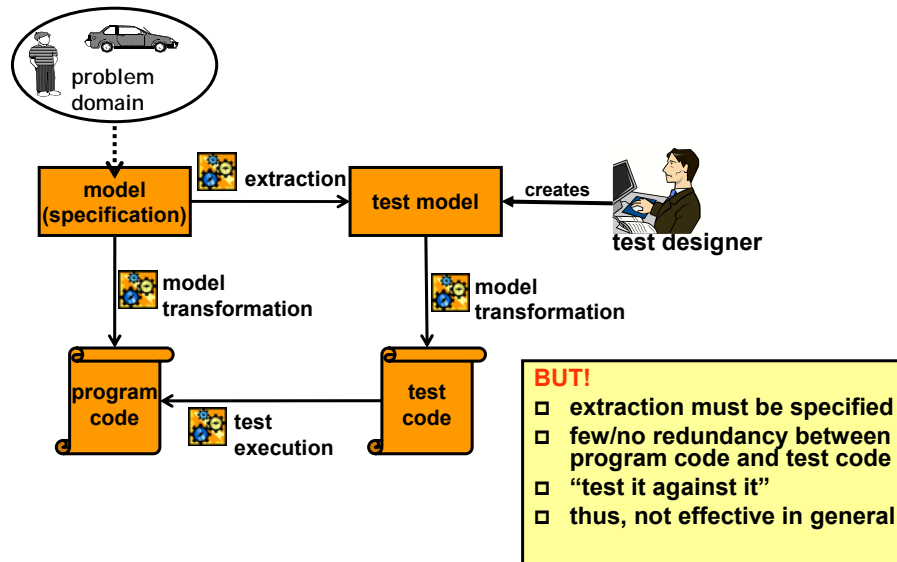


- BUT!**
- complete test code generation
 - needs complete / low level test models (expensive)
 - needs flexible transformation (difficult)
 - thus, not realistic in general

Gregor Engels, BM-MDA 2009

52

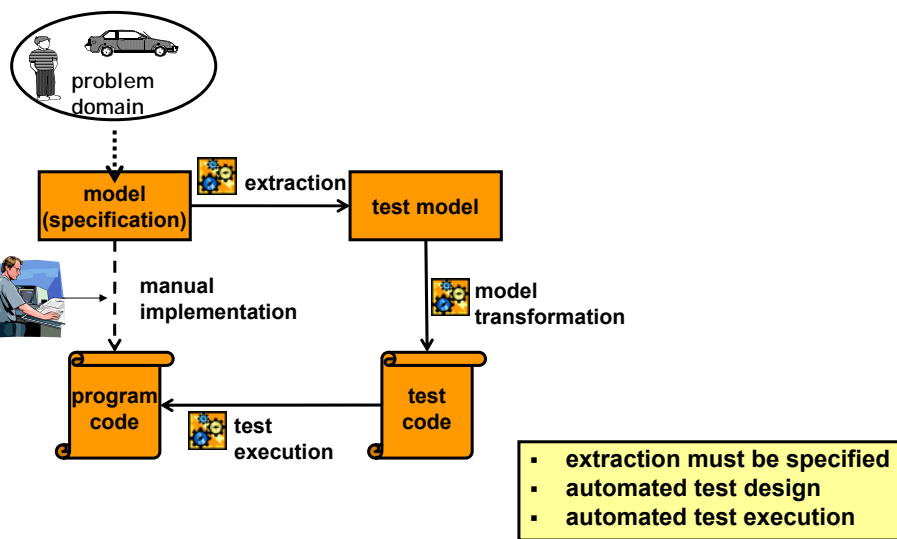
Model-based Testing (MbT): Scenario 2



Gregor Engels, BM-MDA 2009

53

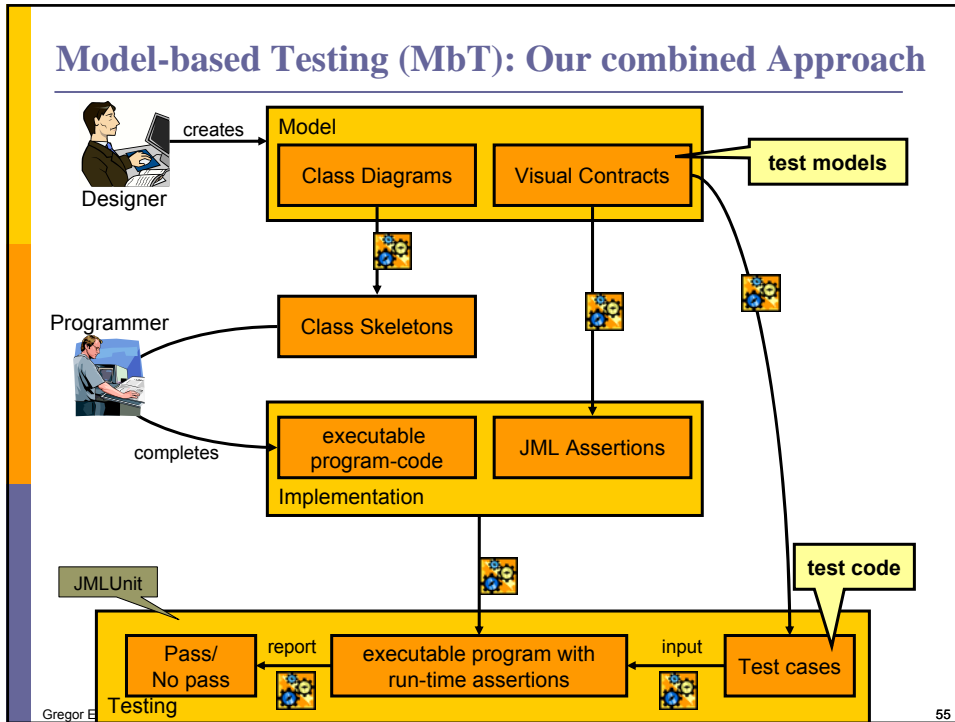
Model-based Testing (MbT): Scenario 3



Gregor Engels, BM-MDA 2009

54

Model-based Testing (MbT): Our combined Approach



Unit Testing: Test Case Generation

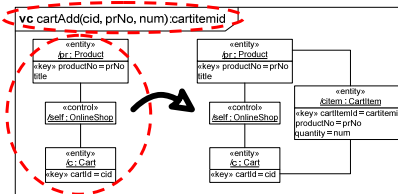
Test case inputs: call parameters + system state

1. Generation of call parameters

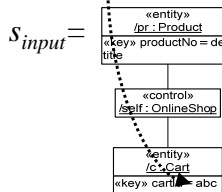
$P = \{cid=„abc“, prNo=„def“, num=1\}$

Well-known techniques:

- Boundary analysis,
- Equivalence classes,
- Random



2. Generation of system state



3. Setting system state

- Simulate
- Naturally generate

Unit Testing: Test case generation

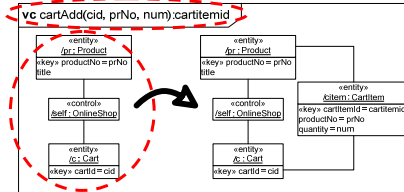
Test case inputs: call parameters + system state

1. Generation of call parameters

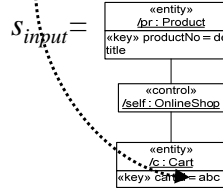
$P = \{cid=„abc“, prNo=„def“, num=1\}$

Well-known techniques:

- Boundary analysis,
- Equivalence classes,
- Random



2. Generation of system state



e.g. using mock objects or stubs

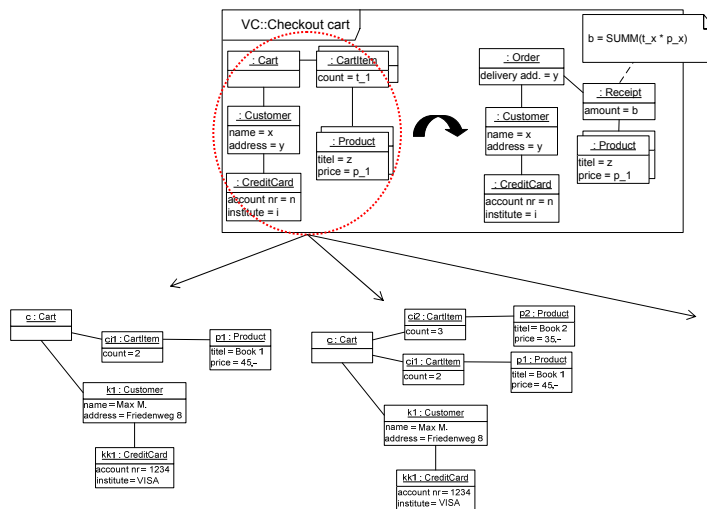
e.g. using model checking

3. Setting system state

$s \supseteq s_{input}$

- simulate system state
- call other class operations until desired system state is reached naturally

Unit Testing: Simulating Test Cases

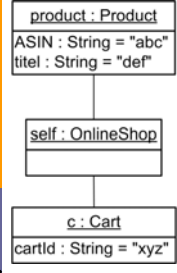


Unit Testing: JUnit Test Script

cartAdd(Product product, int quantity, String cid)

Test input

quantity = 1
cid = "xyz"



generate

```

public void testCartAdd_0()
{
    Product product = new Product();
    product.setTitle("abc");
    product.setASIN("def");
    int quantity = 1;
    String cid = "xyz";

    OnlineShop self = new OnlineShop();
    Cart c = new Cart();

    c.setCartId("xyz");

    self.addCart(c);
    self.addProduct(product);

    self.cartAdd(product, quantity, cid);
}
  
```

Call parameters

further Objects

Object variables

Object relations

Invoke method

Model-driven Monitoring used to check correctness!

Unit Testing: Test case generation

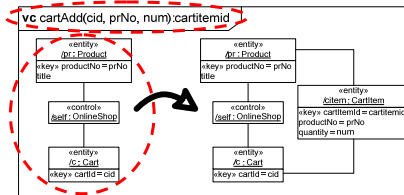
Test case inputs: call parameters + system state

1. Generation of call parameters

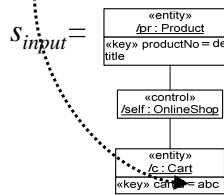
$P = \{cid=,abc, prNo=,def, num=1\}$

Well-known techniques:

- Boundary analysis,
- Equivalence classes,
- Random



2. Generation of system state



e.g. using mock objects or stubs

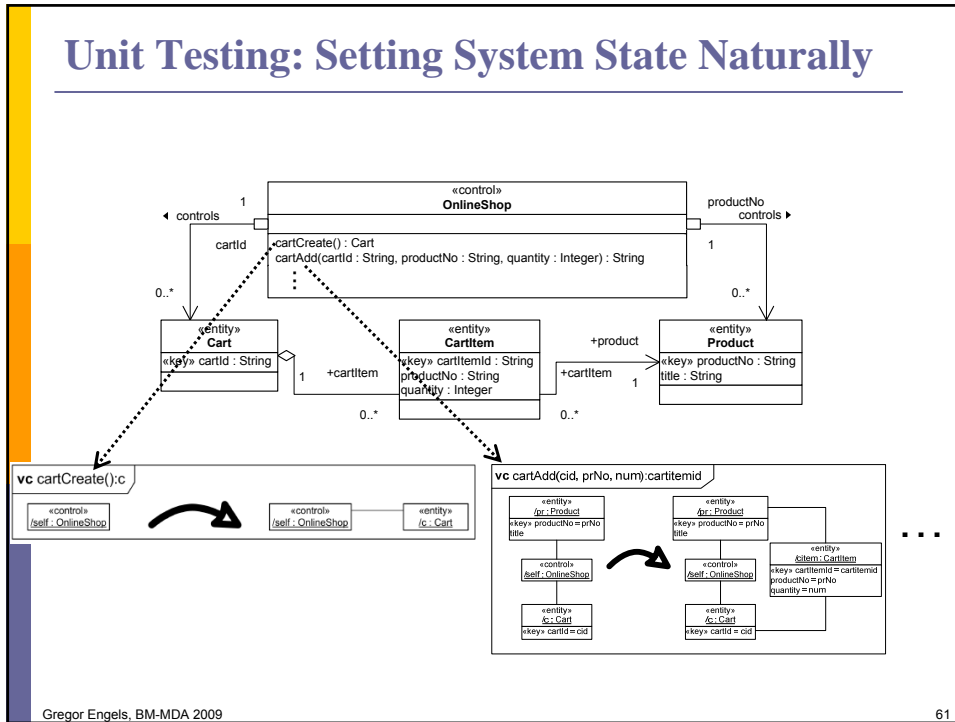
e.g. using model checking

3. Setting system state

$s \supseteq s_{input}$

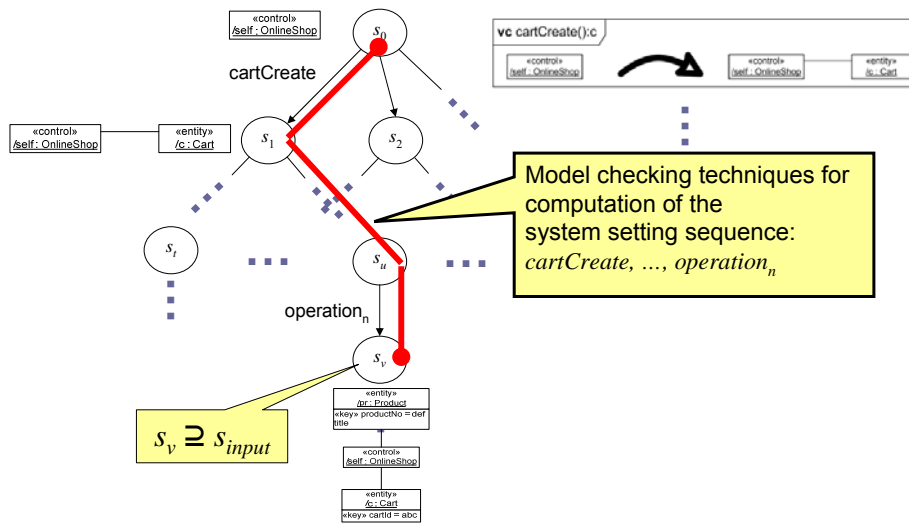
- simulate system state
- call other class operations until desired system state is reached naturally

Unit Testing: Setting System State Naturally



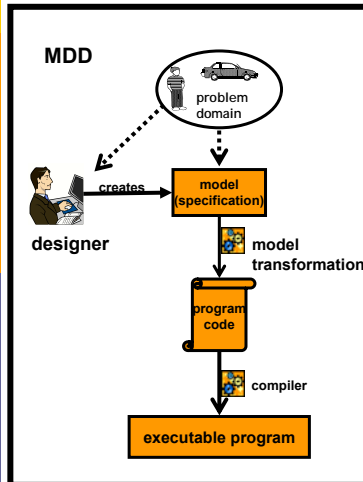
Unit Testing: Setting System State

3. Generation of system setting sequence

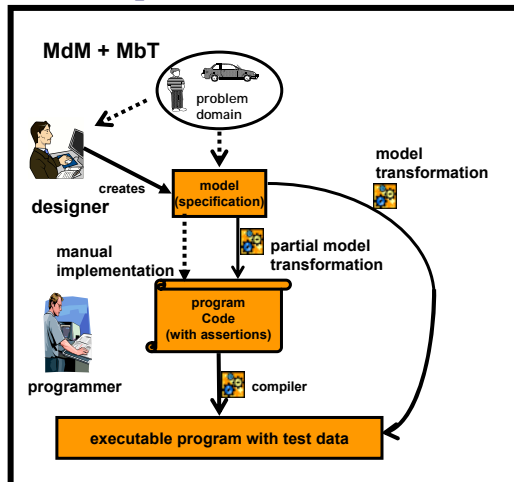


Title of the Talk: Automatic generation of behavioral code – too ambitious or even unwanted?

The Dream



Our Proposal: MdM + MbT



Gregor Engels, BM-MDA 2009

63

Acknowledgments

Thanks to the PhD students who helped me in preparing this talk:

Baris Gldali



Maria Semenyak



Michael Spijkermann



Gregor Engels, BM-MDA 2009

64

References Model-based Testing

- G. Engels, B. Güldali, C. Soltenborn, H. Wehrheim:
Assuring Consistency of Business Process Models and Web Services using Visual Contracts.
In A. Schürr/ M. Nagl/ A. Zündorf (Eds.): Proc. 3rd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (ACTIVE'07), LNCS 5088, pp. 17-31, Springer 2008.
- G. Engels, B. Güldali, M. Lohmann
Towards Model-Driven Unit Testing
In T. Kühne (ed.), Satellite Events at the MoDELS 2006 Conference, Genua, Italy, October 1-6, 2006, Revised Selected Papers, LNCS 4364, pp. 182 - 192, Springer 2007
- B. Güldali, M. Mlynarski, A. Wübbeke, G. Engels:
Model-Based System Testing Using Visual Contracts.
In Proceedings of Euromicro SEAA Conference 2009, Special Session on "Model Driven Engineering". IEEE, to be published.

References Model-driven Monitoring

- G. Engels, M. Lohmann, S. Sauer, R. Heckel:
Model-Driven Monitoring: An Application of Graph Transformation for Design by Contract
In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, G. Rozenberg (Eds.): Proc. Third International Conference on Graph Transformations (ICGT 2006), LNCS 4178, pp. 336 – 350, Springer 2006.
- M. Lohmann, S. Sauer, G. Engels:
Executable Visual Contracts
2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), pp. 63-70, 2005.

References Semantics

- G. Engels, A. Kleppe, A. Rensink, M. Semenyak, Chr. Soltenborn, H. Wehrheim
From UML Activities to TAAL - Towards Behaviour-Preserving Model Transformations
Proc. of ECMDA 2008, LNCS 5095, pp. 94-109, Springer 2008.
- G. Engels, C. Soltenborn, H. Wehrheim:
Analysis of UML Activities Using Dynamic Meta Modeling
In M. M. Bosangue, E. Broch Johnsen (Eds.): FMOODS 2007, LNCS 4468, pp. 76-90, Springer 2007.
- G. Engels, J. H. Hausmann, R. Heckel, S. Sauer
Dynamic Meta-Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML
In A. Evans, S. Kent, B. Selic (eds.): UML 2000 - The Unified Modeling Language, Third International Conference, October 2000, York, UK, LNCS 1939, pp. 323-337. Springer 2000.

The End

engels@upb.de