Realizing the Model-Driven Engineering (MDE) Vision

Robert B. France
Dept. of Computer Science
Colorado State University
france@cs.colostate.edu

Colorado State University Computer Science Department

Outline of talk

- On the difficulty of developing complex software
- Why model software?
- Meeting the MDE Challenges
- Beyond MDE

Colorado State University Computer Science Department

"We know why (software) projects fail, we know how to prevent their failure … so why do they still fail?"

Martin Cobb
Treasury Board of Canada
Secretariat

The Problem-Implementation Gap

- A problem-implementation gap exists when software is implemented using abstractions that are different than those used to understand and describe the problem
  - when the gap is wide significant effort is required to implement solutions
  - bridging the gap using manual techniques introduces significant accidental complexities

Factors behind the gap widening

- Technology advances open the door to new software opportunities that are acted upon
  - Result is a new generation of software that gives rise to a new breed of software development problems
- The growing complexity of the problem spaces tackled by software overwhelms the available implementation abstractions
  - Leads to a dependence on "expert" developers who have developed mentally-held "patterns" to cope with growing complexity
  - Eventually, growing problem complexity can overwhelm the patterns held by the experts

The abstraction-raising dilemma

- Each successful attempt at raising the level of abstraction triggers concerted effort to develop even more complex software
- Existing forms of software development problems and concerns change and new problems/concerns emerge

The nature of the "software crisis" evolves

Evolving development concerns

- 60’,70’s: Standalone applications
  - Functional correctness
- 70’s,80’s: Closed network systems
  - Network reliability, availability
  - data integrity
  - Functional correctness of concurrent distributed systems
- 90’s: Open network systems
  - Transparent access to distributed resources
  - Usability
  - Service availability, reliability, security
- 2000’s: Open, mobile, pervasive, embedded systems of systems
  - Reasoning about correctness in the presence of mobility
  - Balancing multiple dependability concerns

Colorado State University
Computer Science Department

Developers of mission-critical open distributed software systems need to balance multiple, interdependent design concerns such as availability, performance, survivability, fault tolerance, and security.

Colorado State University
Computer Science Department

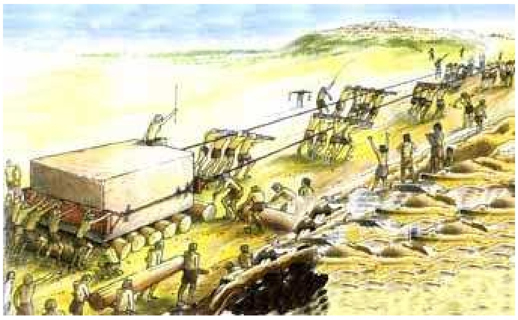
Balancing dependability concerns

- Balancing requires making trade-offs and assessing risks associated with design features that address the concerns
  - Organizations seldom have all the resources needed to build software systems that have the desired levels of dependability
  - Need to consider and evaluate alternative features to determine the extent they
    - address concerns
    - cost-effectively mitigate product-related risks.
  - Pervasiveness of dependability features complicates their evaluation and evolution

Colorado State University
Computer Science Department

Building software pyramids

Building dependable software with current tools is akin to building pyramids in ancient Egypt



Colorado State University
Computer Science Department

Colorado State University
Computer Science Department

Why consider modeling techniques?

Software development is a modeling activity!

Programmers build and evolve mental models of problems and solutions as they develop code

Programmers express solutions in terms of abstractions provided by a programming language

How can we better leverage modeling techniques?

Colorado State University
Computer Science Department

Model Driven Engineering Research Question

How can modeling techniques be used to tame the complexity of bridging the gap between the problem domain and the software implementation domain?

A peek ahead …

What MDE researchers should target:

- Development of technologies supporting the generation of domain-specific software development environments

The reality …

- Requires codifying knowledge that reflects a deep understanding of the gap bridging process
- Understanding can only be gained through costly experimentation and systematic accumulation and examination of experience

Accomplishing the MDE vision of development is a wicked problem!

Realizing the vision

The MDE vision may not be realized in its entirety

but close approximations can have a significant impact on quality and productivity

- … MDE/software engineering research will be needed well into the future
  - Building close approximations will require developing successive generations of technologies;
  - Each new generation should address the accidental complexities of the previous generation

MDE Challenges

- Abstraction
- Separation of Concerns
- Model Analysis
- Model Manipulation and Management

The Abstraction Challenge

The Abstraction Challenge

- MDD claims to raise the level of abstraction at which software is conceived, designed, analyzed and implemented
- Abstraction: Can we build an analyzable general-purpose, language for describing software systems at a level of abstraction above the current high-level programming languages?

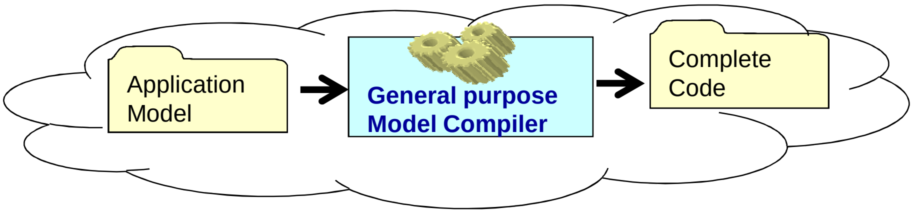
Colorado State University
Computer Science Department

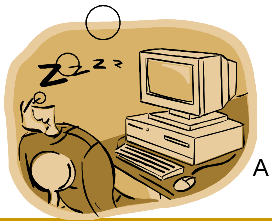
Looking for the "right" abstractions

Colorado State University
Computer Science Department

The Automation Challenge

- Automation: Can we build it such that it is "compilable"?





A modeler's daydream

Colorado State University
Computer Science Department

Bridging the Abstraction Gap

The gap between models that we typically build using languages such as the UML and code seems wider than the gap between models we build in a programming language like Java and machine code.

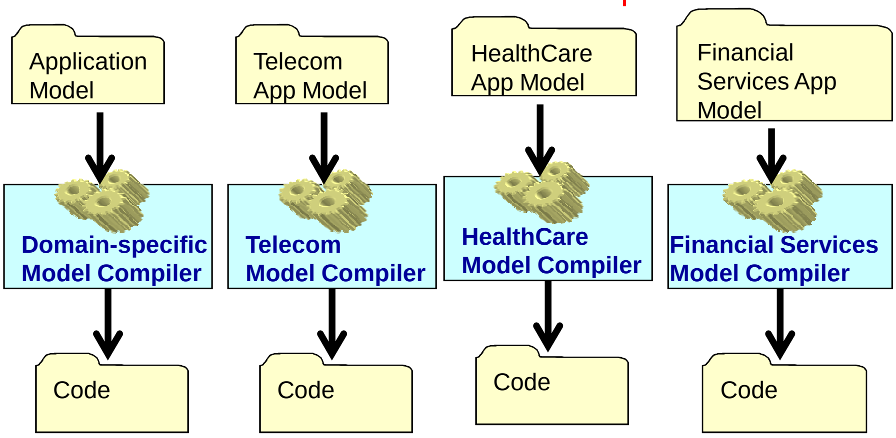
The needed abstractions tend to be domain-specific and thus general-purpose "compilers" that attempt to bridge the gap have to incorporate domain-knowledge in some form

Leads to consideration of domain-specific languages and frameworks for building domain-specific development environments

Colorado State University
Computer Science Department

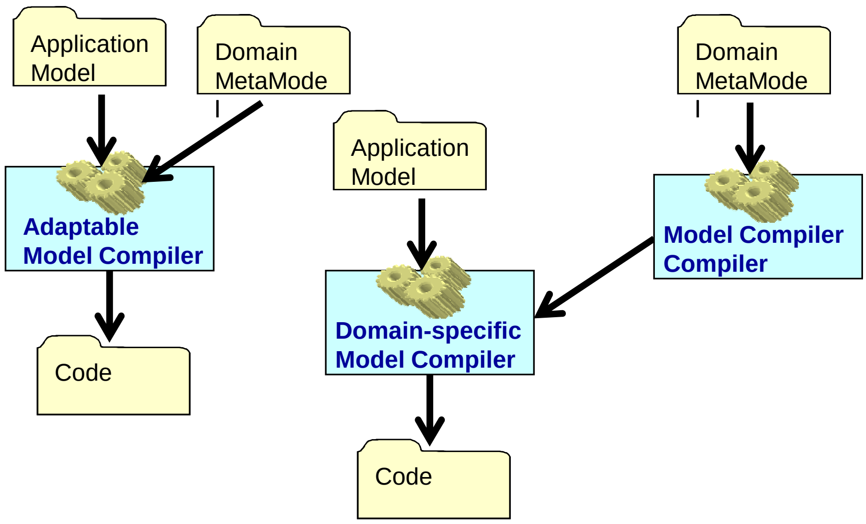
The Domain Specific Modeling Language Challenge

How can we avoid the "Tower of Babel" problem?



Colorado State University Computer Science Department

Towards Model Compiler Compilers



Colorado State University Computer Science Department

MDD Frameworks

- A MDD framework provides concepts and tools that domain architects can use to build domain-specific MDD development environments
  - Can be based on a family of languages (e.g., the UML) … a formal notion of profiles is needed to support MDD frameworks
  - Focus of Software Factory research at Microsoft [Jack Greenfield, Keith Short, Steve Cool, Stuart Kent, "*Software Factories, Assembling Applications with Patterns, Models, Frameworks and Tools. Wiley, 2004*]
  - Requires deep understanding of modeling phenomena
  - … and thus full frameworks are not likely to appear in the near future

Colorado State University
Computer Science Department

Separating Concerns Challenges

Colorado State University
Computer Science Department

Going beyond traditional support for separation of concerns

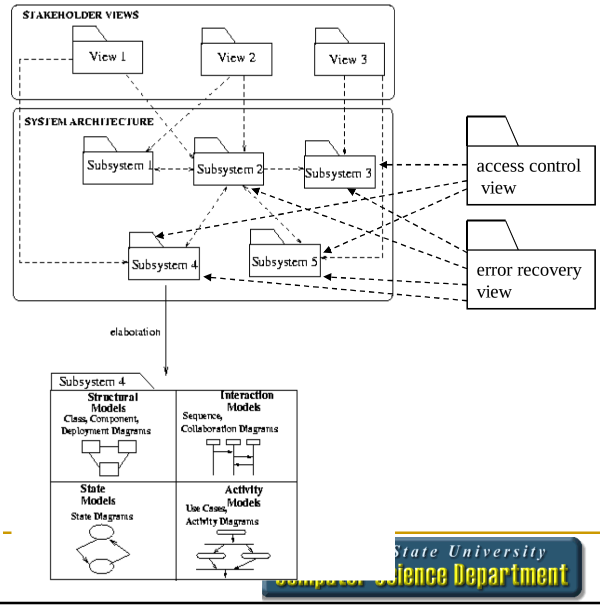
- The Separation of Concerns principle is considered fundamental in software engineering
- Much attention paid to providing support for modularizing descriptions of problems and solutions (separation of parts)
- Less attention has been paid to providing support for understanding interactions across separated parts

On the importance of understanding interactions across features: An example

The first launch of the space shuttle Columbia was delayed because "(b)ackup flight software failed to synchronize with primary avionics software system"

(http://science.ksc.nasa.gov/shuttle/missions/sts-1/mission-sts-1.html)

Crosscutting Design Views



- Trade-offs necessary when crosscutting solutions interact to produce undesirable behavior
- Current modeling techniques do not support separation of crosscutting solutions.

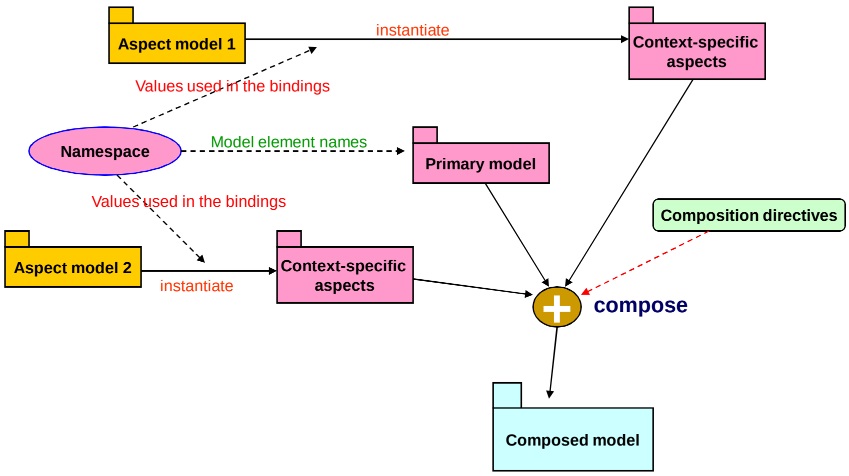
Colorado State University
Computer Science Department

Aspect-Oriented Design

- Separation of Concerns
  - *Primary Model* : A model of core functionality; determines dominant structure
  - *Aspect Model* : Describes a feature that cross-cuts modules in the dominant design structure
- Design Patterns
  - Aspects are patterns: Promotes reuse

Colorado State University
Computer Science Department

Aspect Oriented Modeling



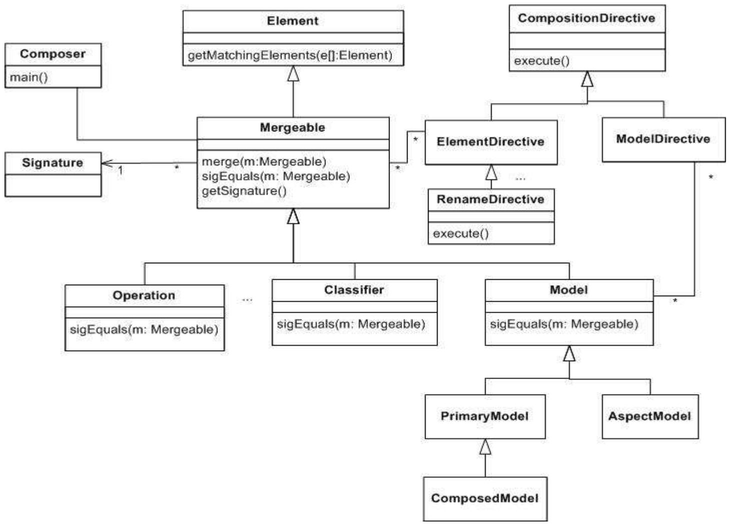
Colorado State University
Computer Science Department

Composition Directives

- Used to ensure that basic composition procedure produces desired result
- Two types
  - **Model directives**: determine the order in which aspect models are merged
  - **Element directives**: affect how model elements are composed
- Types of element directives
  - **Matching directives**: used to force or prevent model element matching
  - **Merge directives**: used to override merging rules used in basic procedure
  - **Build directives**: used to introduce or remove elements in models

Colorado State University
Computer Science Department

Composition Metamodel

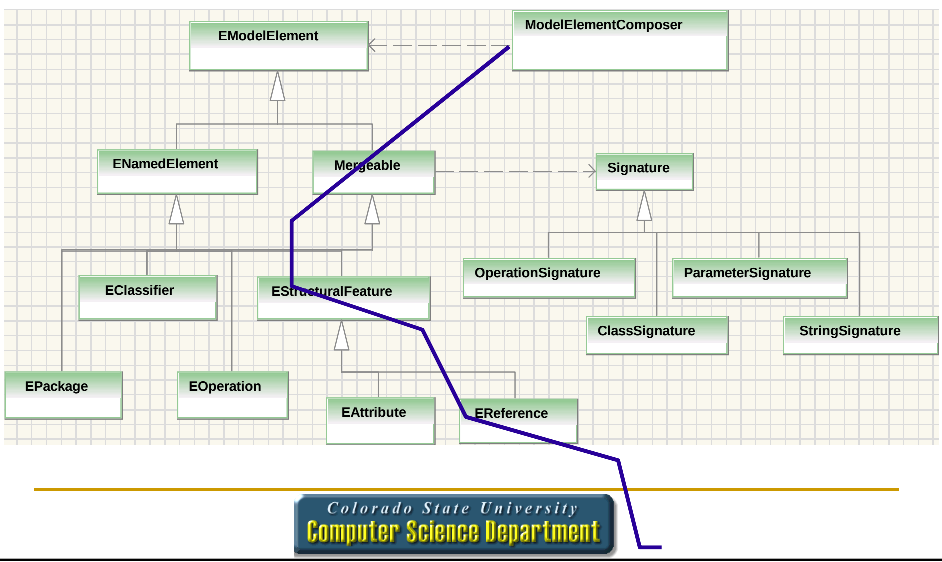


Colorado State University
Computer Science Department

Composition Meta-model (merge)

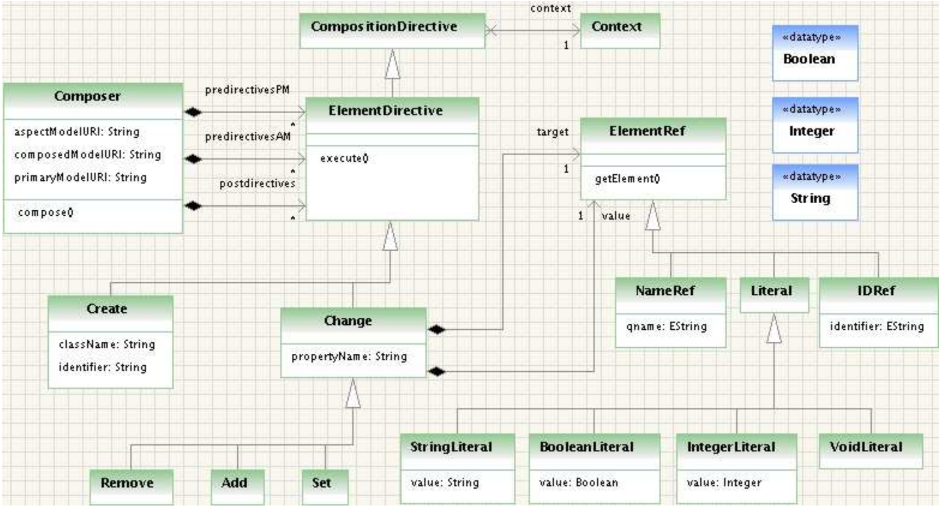
Class diagram

Composition



Colorado State University
Computer Science Department

Composition Meta-model (directives)



Colorado State University
Computer Science Department

AORDD Framework

- AORDD: **Aspect-Oriented Risk-Driven Development**
- The framework
  - Models security solutions as security aspects
  - Includes security analysis and verification of security aspects
  - Includes verification of composed models using properties
  - Support cost-effective development through the AORDD security solution design trade-off analysis

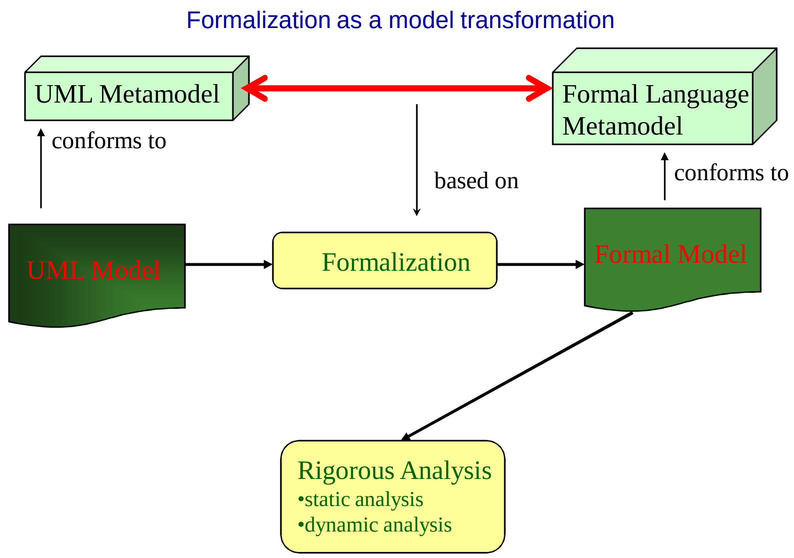
Colorado State University
Computer Science Department

Analyzing Models

Role of Formal Methods

- Some formal method researchers have commented that FST research subsumes MDE research
- My opinion: MDE research provides a context for FST research
  - Current formal techniques are applicable to only specific views of a system
  - MDE concerns go beyond describing and analyzing systems from a limited set of viewpoints

Integrated Methods



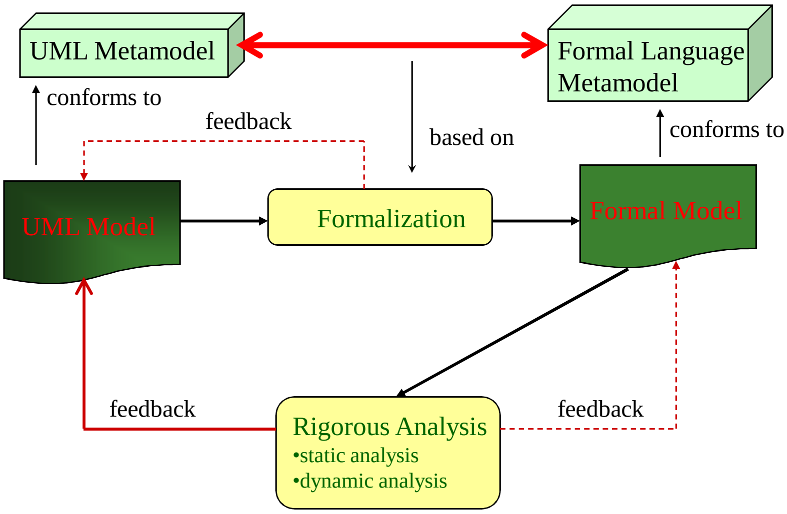
Colorado State University
Computer Science Department

Pitfalls

- Many UML to formal X notation approaches
  - Most force users to be familiar with both UML and formal notation
  - Not aware of any that express formal analysis results in terms of UML
- Assumption that a single semantics for the UML will suffice
  - UML is a family of languages
  - BUT a semantic core for some models may be useful
    - Formalizing class model concepts (Evans, France, Bruel, …)
    - Formalizing interaction diagrams (Engels, Knapp, …)

Colorado State University
Computer Science Department

Integrated Methods

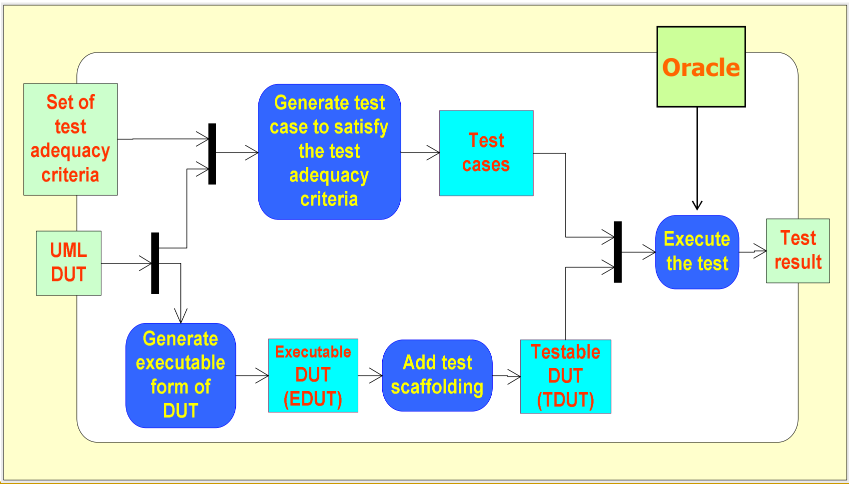


A Systematic Approach to Testing UML Designs

Robert France, Sudipto Ghosh, Trung T. Dinh-Trong
Department of Computer Science, Colorado State University

Sponsors: NSF, IBM Eclipse Innovation Grants

Test approach

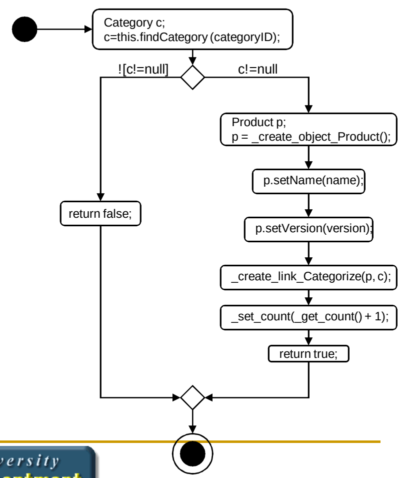


Execution semantics

- UML action semantics
  - E.g., Create/destroy instances, links; read/set attributes
- Java-like Action Language (JAL)
  - Java syntax
  - Surface syntax for UML action semantics
  - Used to describe sequence of actions performed by a class instance during execution of operation call
  - Currently, no support for parallel structures or asynchronous actions

A JAL Example

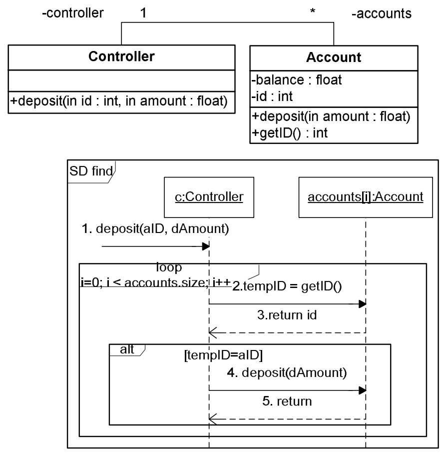
```
boolean addProduct(int categoryID, int version,
    String name)
{
    Category c;
    c = this.findCategory (categoryID);
    if(c != null) {
        Product p;
        p = _create_object_Product();
        p.setName(name);
        p.setVersion(version);
        _create_link_Categorize(p, c);
        return true;
    }
    else
        return false;
}
```



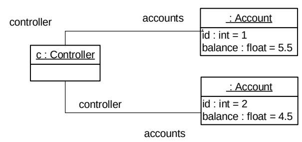
Colorado State University
Computer Science Department

Test input

- Each test input tests one scenario described in a sequence diagram
- Test input (S, Pr)
  - S: Start configuration
  - Pr: System operation with parameter values



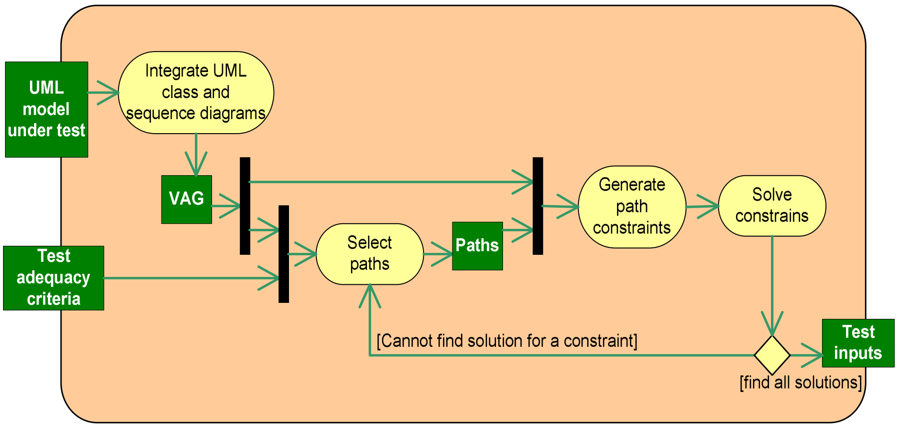
S:



Pr: c.deposit(1, 2)

Colorado State University
Computer Science Department

Test Input generation process



Colorado State University
Computer Science Department

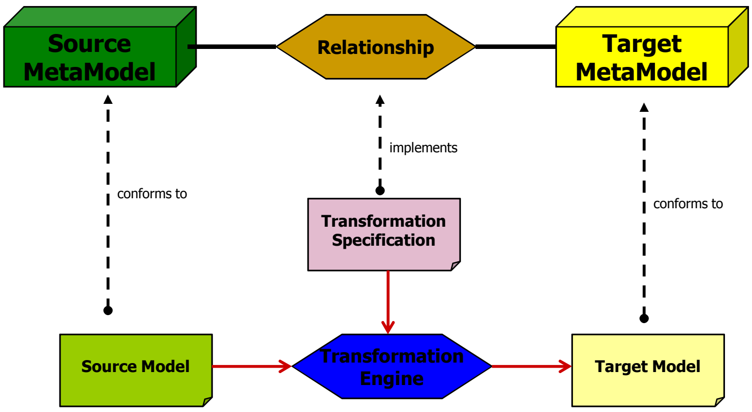
Model Manipulation and Management

Colorado State University
Computer Science Department

Model Management Initiatives

- Megamodeling (Jean Bezivin, Univ. of Nantes)
- Model Repository (Dan Matheson, Colorado State University)
- Community-based model repository (PlanetMDE, REMODD, models.org, …)

Model Transformations



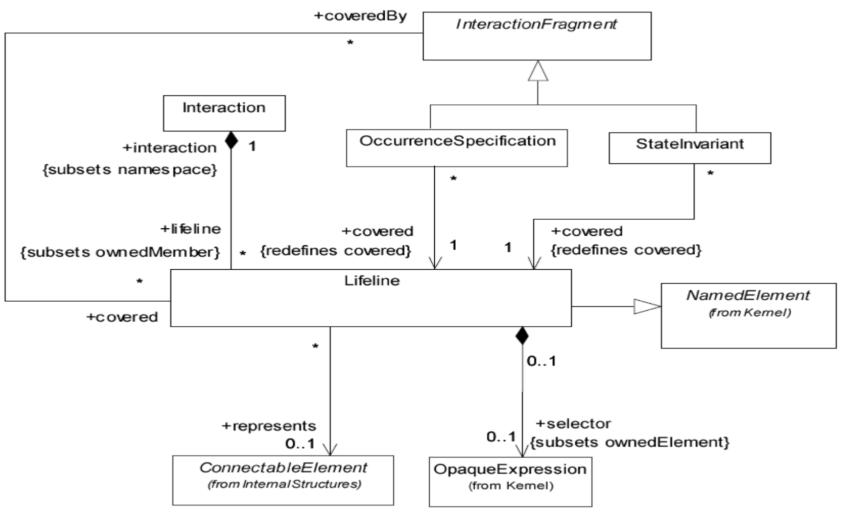
Work on model transformations

- QVT
  - Creating a standard when there is very little practical experience is challenging!
- Transformations that preserve QoS properties
- Middleware transparent software development

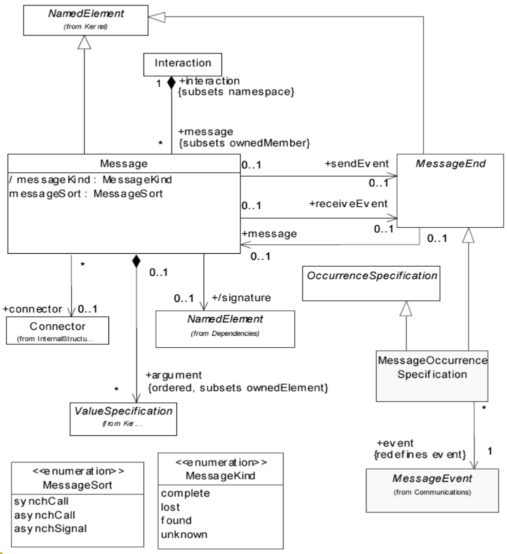
The UML metamodel challenge: Navigating the metamuddle

Using the UML 2.0 metamodel find the relationships between message ends and lifelines

Lifelines (from the UML 2.0 specification)

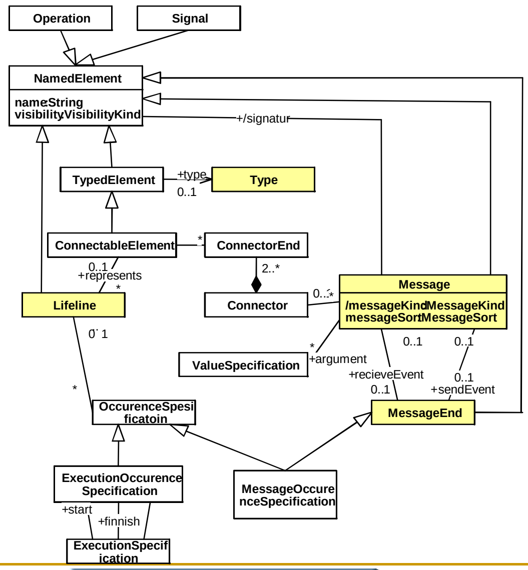


Messages (from the UML 2.0 specification)



Colorado State University
Computer Science Department

A *simple* UML interactions metamodel



Colorado State University
Computer Science Department

Beyond MDE

Models@run.time

Models can be used at runtime to:

- Present aspects of runtime phenomenon
- Support software adaptation
  - Adaptation agents can use runtime models to determine the need for adaptation and to determine the adaptation needed
- Support controlled evolution of software
  - Change agents can use runtime models to correct design errors and to introduce new features during runtime

Colorado State University
Computer Science Department

Summary

- Realizing the MDE vision is a wicked problem
- Software engineering (technical aspects) is essentially a modeling activity
  - MDE highlights the importance of models as explicit representations of intent.
  - Reduces the cognitive burden and accidental complexities associated with maintaining mental models.
- It may seem that MDE contributes to development complexity but the web of models produced in a MDE project reflect inherent complexity (when done well!)
- There will always be accidental complexities associated with using modeling languages and MDE technologies

Colorado State University
Computer Science Department

Conclusion

Colorado State University
Computer Science Department