

Model-Driven Engineering: Realizing the vision



Robert B. France

Dept. of Computer Science

Colorado State University

Fort Collins, Colorado, USA

france@cs.colostate.edu

About the author

- Organizer and steering committee member of the MODELS conference series
- Founding editor-in-chief, Software and System Modeling journal (SoSyM)
- Member of UML 1.X revision task forces
- Current research focus on aspect-oriented modeling (AOM), model composition

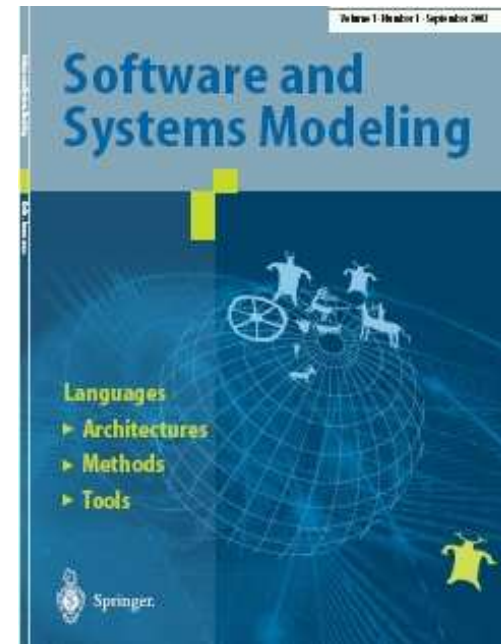
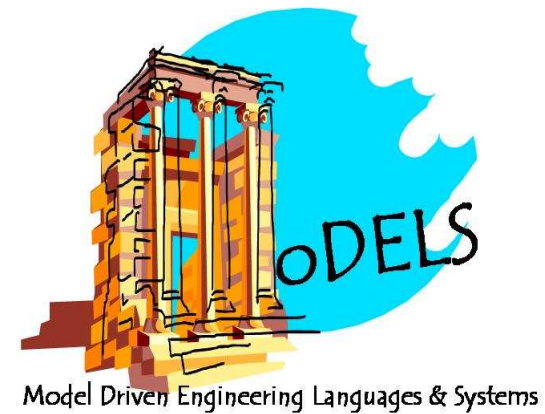




Diagram provided by Arnor Solberg, SINTEF, Norway

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

MDE is concerned with ...

- reducing accidental complexities associated with bridging wide problem-implementation gaps
- through use of technologies that support systematic transformation of abstractions to software implementations

MDE is concerned with developing software to support the work of software developers

Why modeling techniques?

Software development is a modeling activity

Programmers build and evolve (mentally-held) 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?

What is a model?

A description of an aspect of a software-based system that may or may not exist. A model is created to serve particular purposes.

MDE \neq use of UML

2 broad classes of models

- **Development models**
 - Models produced and used during the development of software systems
 - Focus on models that describe systems above the code level of abstraction
 - Current focus of MDE researchers
- **Runtime models**
 - Models produced or used during system runtime
 - Models describe some aspect of an executing system
 - Small, but growing community of MDE researchers

Where are we now?

- **Generation 1 (Informal Models):** Models as documentation/communication artifacts (informal camp)
 - a.k.a the Computer Aided Software Engineering (CASE) generation
- **Generation 1 (Formal Models):** Models as analysis mechanisms (formal camp)
 - Use of mathematically-base specification languages: VDM, Z, algebraic specs
- **Generation 2:** Models as artifacts for generating implementation and deployment artifacts
 - Exemplified by OMG's work on MDA/MDD
 - Emphasis on separation of platform independent concerns from platform specific concerns

Is it enough to just create languages that raise the level of abstraction?

- Each successful attempt at raising the level of abstraction triggers concerted effort to develop even more complex software
 - New technologies also give rise 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 gap widens

The abstraction-raising dilemma

- The growing complexity of the problem and solution spaces overwhelms the abstractions provided by current implementation languages
 - Leads to ad-hoc development of technology layers that extend current languages in an attempt to provide needed abstractions (**some gap contraction**)
- Complexity of technology-based abstraction layers leads to reliance on expert software developers (**gap widening**)
- Eventually, complexity overwhelms even the experts (**accelerated gap widening**)
- Need for raising the “level of abstraction” is painfully apparent – a “software crisis” is declared ...

The nature of the “software crisis” evolves

MDE research questions

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

How can MDE be used to manage the rate at which the gap is widening?

Where are we going?

- Providing support for engineering domain-specific languages and associated tool sets
- A MDE framework provides concepts and tools that developers can use to build domain-specific development environments
 - Can be based on a family of languages (e.g., the UML)
 - or on meta-metamodel facilities
 - Jack Greenfield, Keith Short, Steve Cook, Stuart Kent, “*Software Factories, Assembling Applications with Patterns, Models, Frameworks and Tools*. Wiley, 2004
 - Jean Bézivin, Reiko Heckel (Eds.) “*Language Engineering for Model-Driven Software Development*”, [Dagstuhl Seminar Proceedings](#), 2005

Getting there

- Requires deep understanding of gap-bridging activity
 - Understanding can only be gained through development of solutions, costly experimentation, and systematic accumulation and examination of modeling and software development experience
 - ... development frameworks that realize the MDE vision are not likely to appear in the near future
- Development has to be incremental and iterative
 - Each new generation of MDE tools should address the accidental complexities of past generations and thus move us closer to better approximations of the MDE vision

Next steps

- **Generation 3:**
 - MDE frameworks that support creation of DSML editors and some code generation facilities
- **Generation 4:**
 - MDE frameworks with support for mega-modeling
 - Rudimentary support for models@run.time

MDE challenges

- Modeling languages
 - Providing support for creating and using **appropriate abstractions**
- Separation of concerns
 - Providing support for modeling and analyzing **views** possibly expressed in different languages
- Model Manipulation and Management
 - Providing support for model transformation, composition, evolution, analysis, roundtrip engineering, ...

Modeling Languages

Tackling the abstraction challenge



Finding the “right”
abstractions

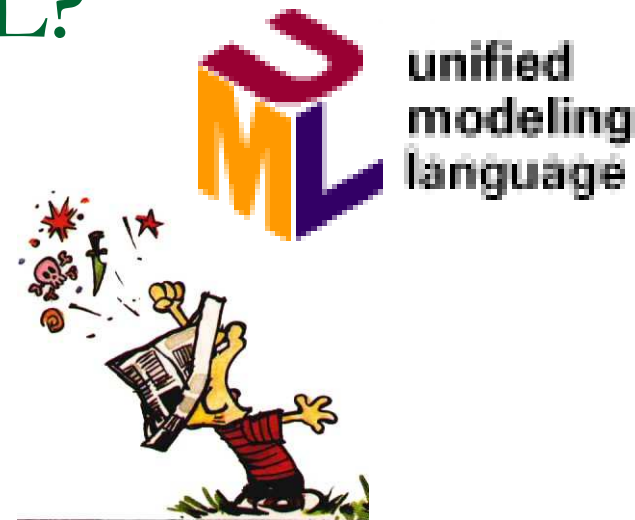
Two schools of thought

- The extensible general-purpose modeling language (GPML) school
 - UML with semantic variation points and profiles
- The domain-specific language (DSML) school
 - Software factories, GME

What about the UML?

B. Henderson-Sellers, S. Cook, S. Mellor, J. Miller, B. Selic,
UML – the Good, the Bad or the Ugly? Perspectives from a panel of experts, Software and Systems Modeling 4(1), 2005.

R. France, S. Ghosh, T. Dinh-Trong, A. Solberg,
Model-Driven Development Using UML 2.0: Promises and Pitfalls,
IEEE Computer, Vol. 39, No. 2, February 2006, 59-66



“It is easier to perceive error than to find truth, for the former lies on the surface and is easily seen, while the latter lies in the depth, where few are willing to search for it.”

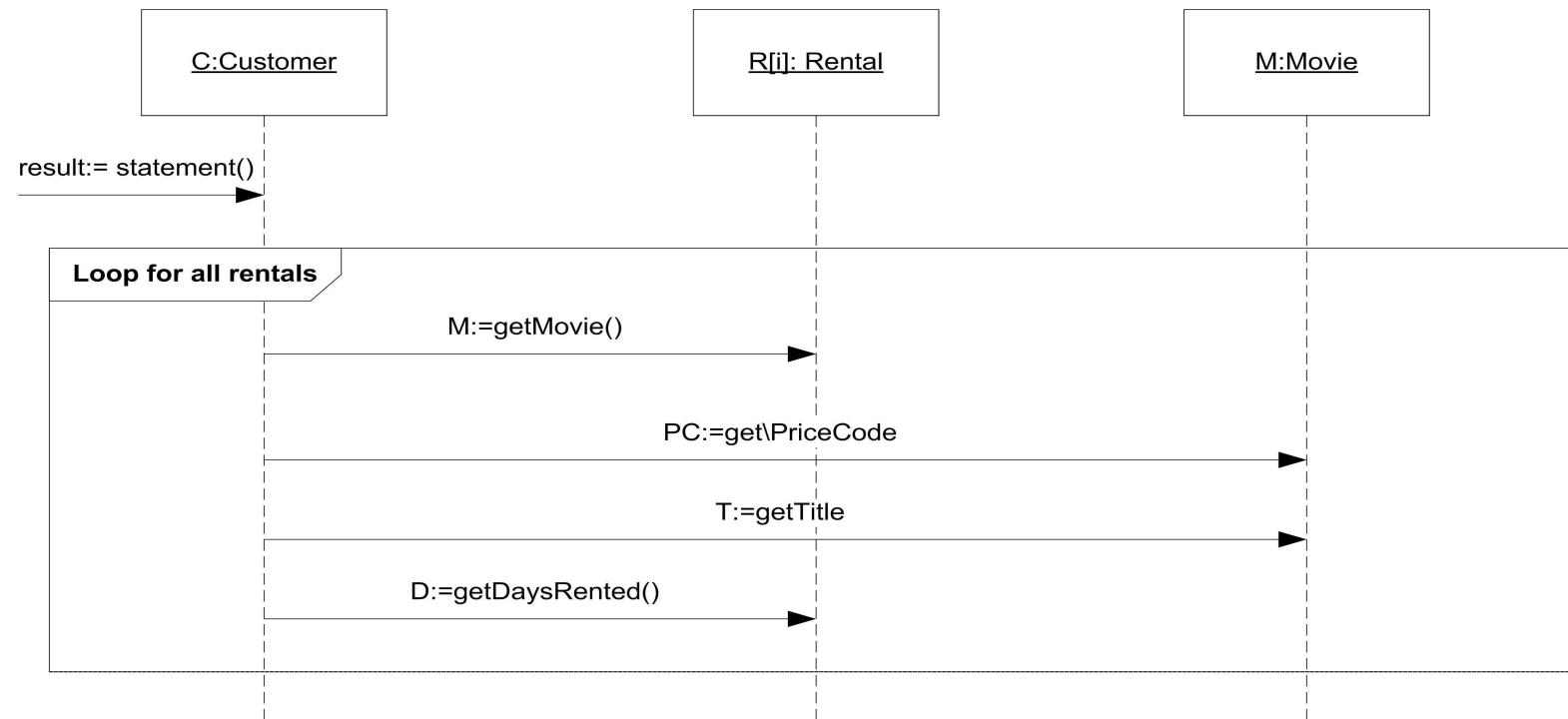
Johann Wolfgang von Goethe

Learning from the UML experience: Avoiding bloat

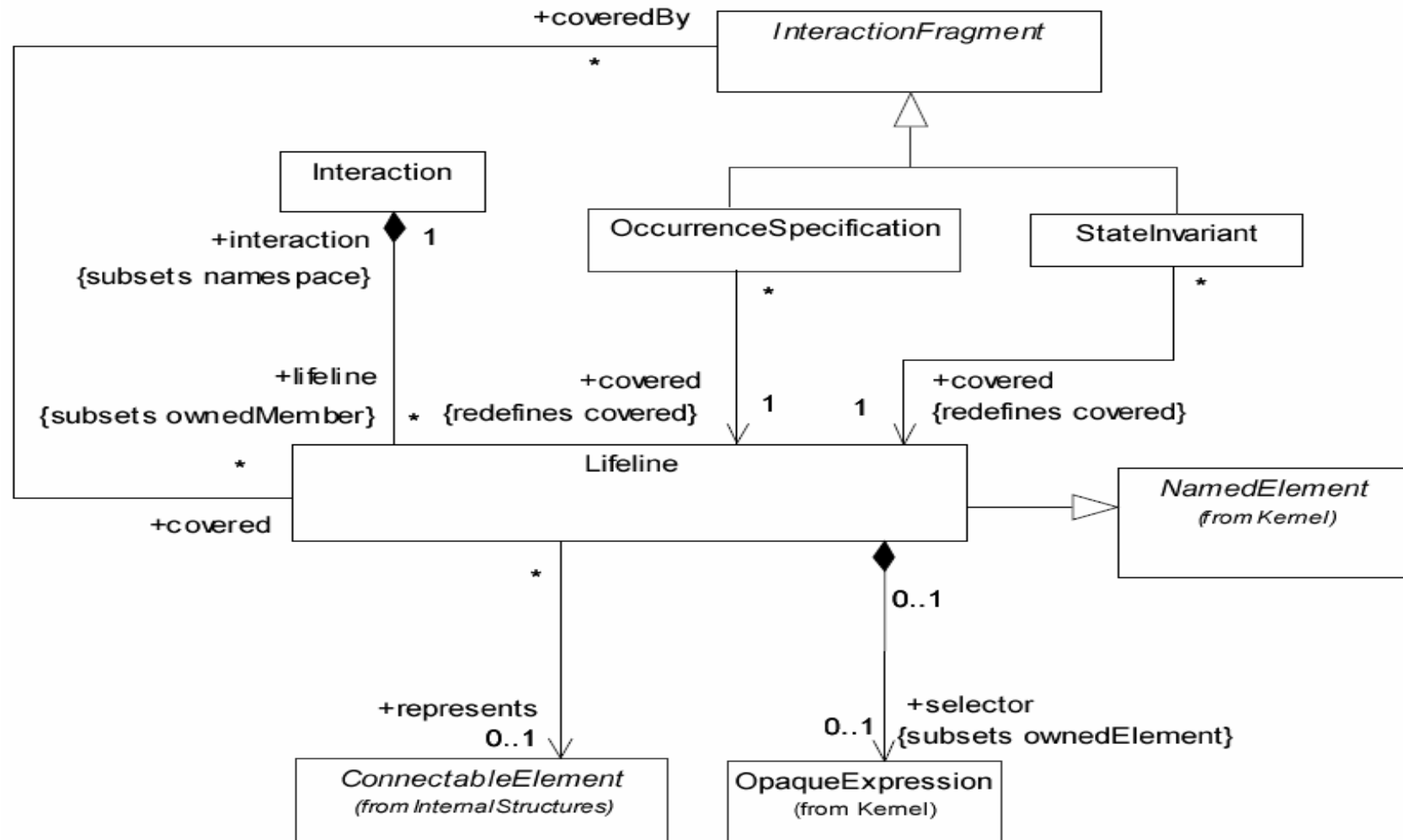
- Difficulty of identifying a small base set of modeling concepts that can be used to express a broad range of abstractions
- Result: UML 2.0 provides support for many types of abstractions but the language definition is not well-structured
 - Gives rise to “meta-muddles”

Navigating the “meta-muddle”

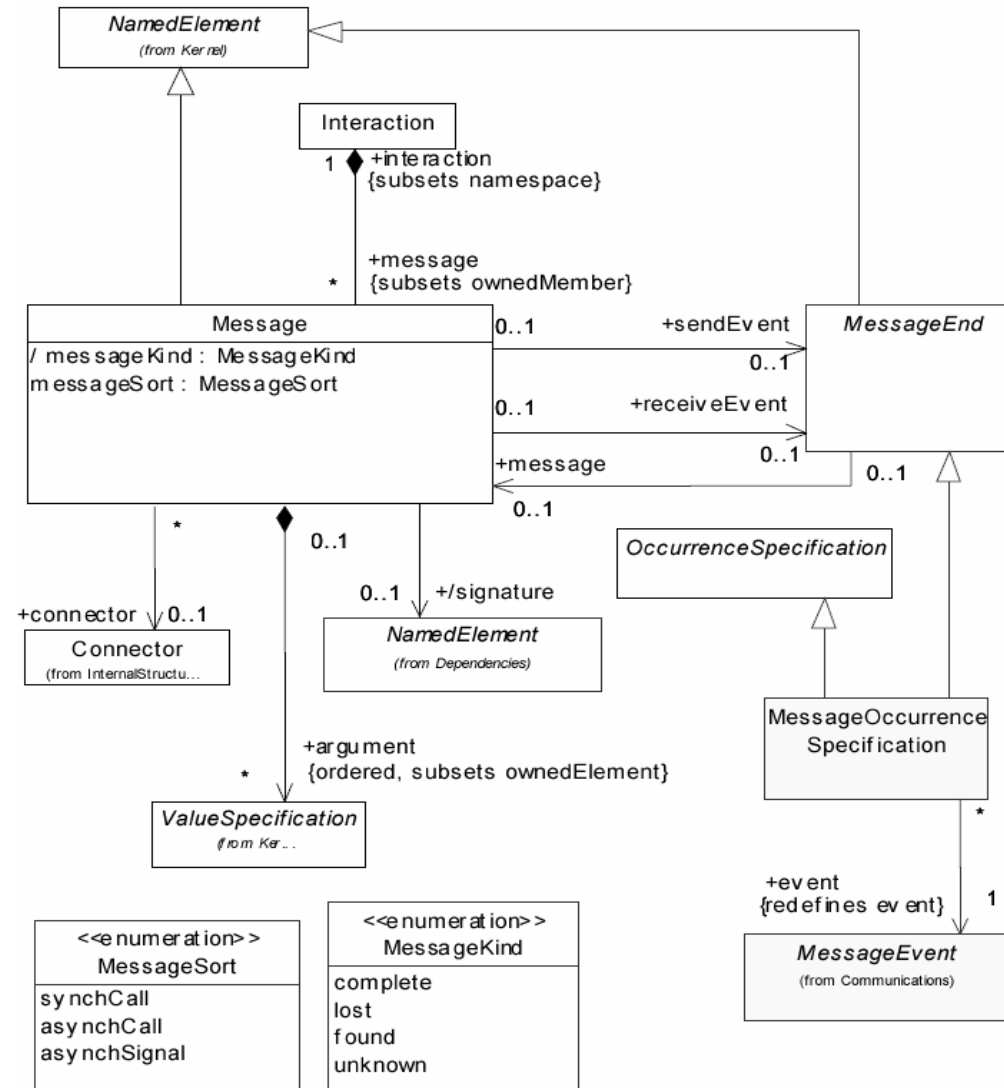
What relationship does a message end have with a lifeline?

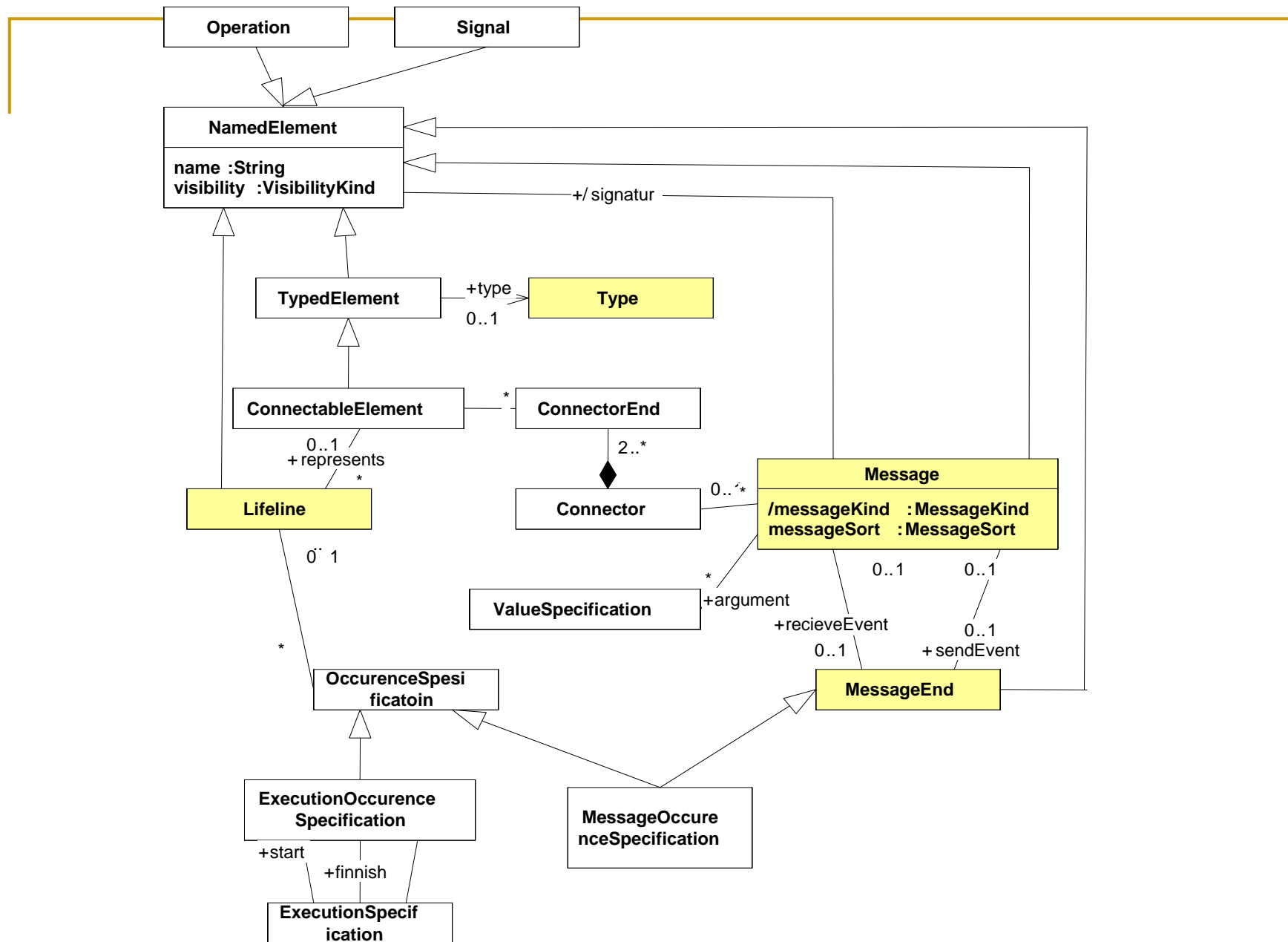


Lifeline (from UML 2.0 metamodel)



Messages (from the UML 2.0 specification)





Learning from the UML experience: Avoiding “meta-muddles”

- UML abstract syntax described by a complex meta-model
 - Complexity problematic for application developers, modeling tool developers, and for language standards committees
- Need tools for navigating and for extracting views from meta-models
- Reflects need for research on engineering modeling languages

DSML challenges - 1

- Creating and evolving languages and their toolsets
 - Need facilities for engineering languages quickly
 - Composing language units
 - Domains evolve and thus languages and supporting tools must evolve
 - Tool challenge: Need to provide a foundation for building meta-toolsets
 - Language challenge: Need to provide support for language versioning and migration

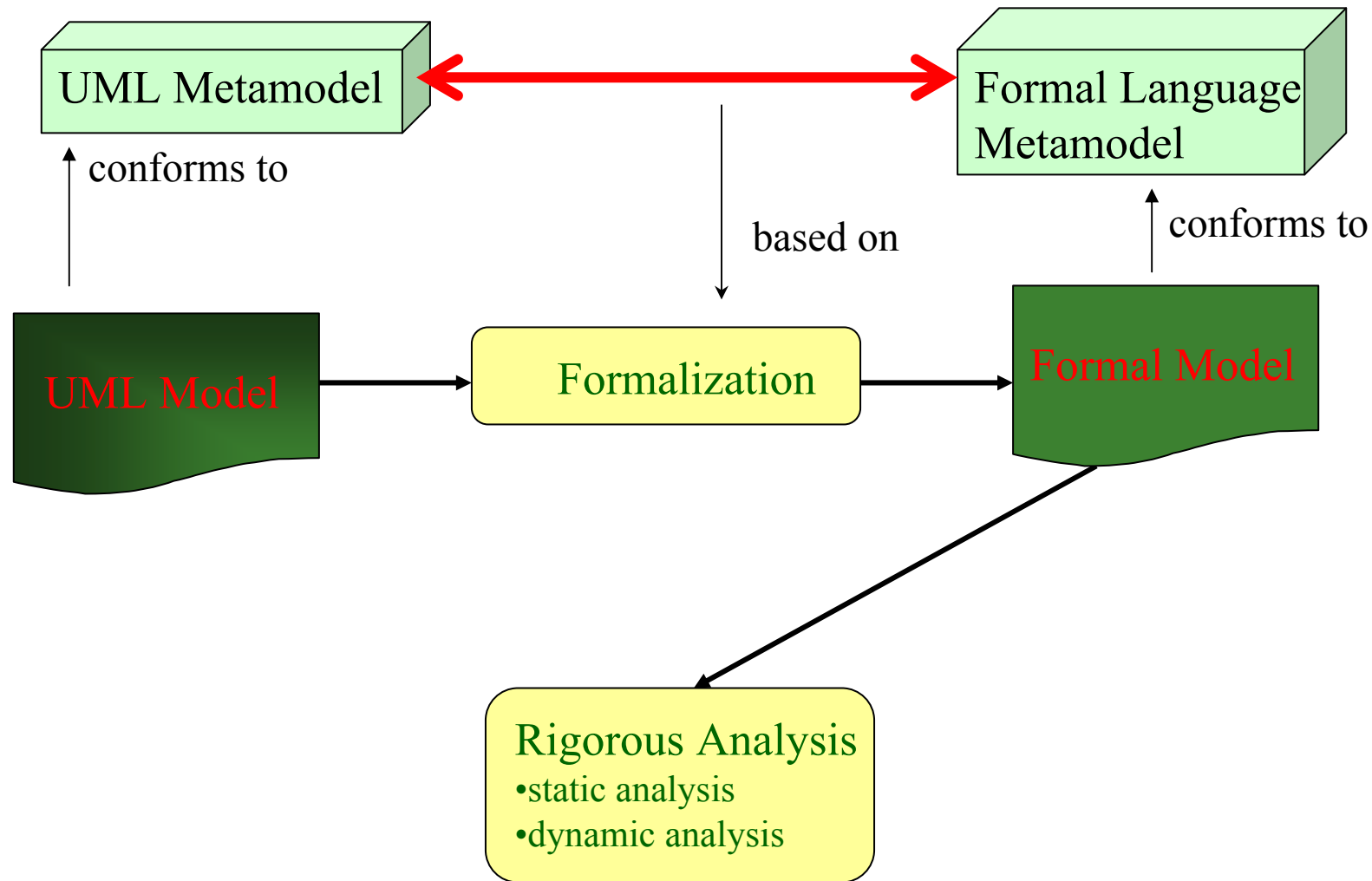
DSML challenges - 2

- Avoiding the “DSL-Babel”
 - A project may use many DSMLs and thus language interoperability is a concern
 - Need to relate concepts across different DSLs and provide support for maintaining consistency of concept representations across the languages

Tackling the formality challenge: Why not just use formal methods?

- Does formal methods research subsume MDE research?
- Not likely
 - MDE research provides a context for FST research
 - Current formal techniques are applicable in specific views of a system
 - MDE concerns go beyond describing and analyzing systems using a limited set of viewpoints

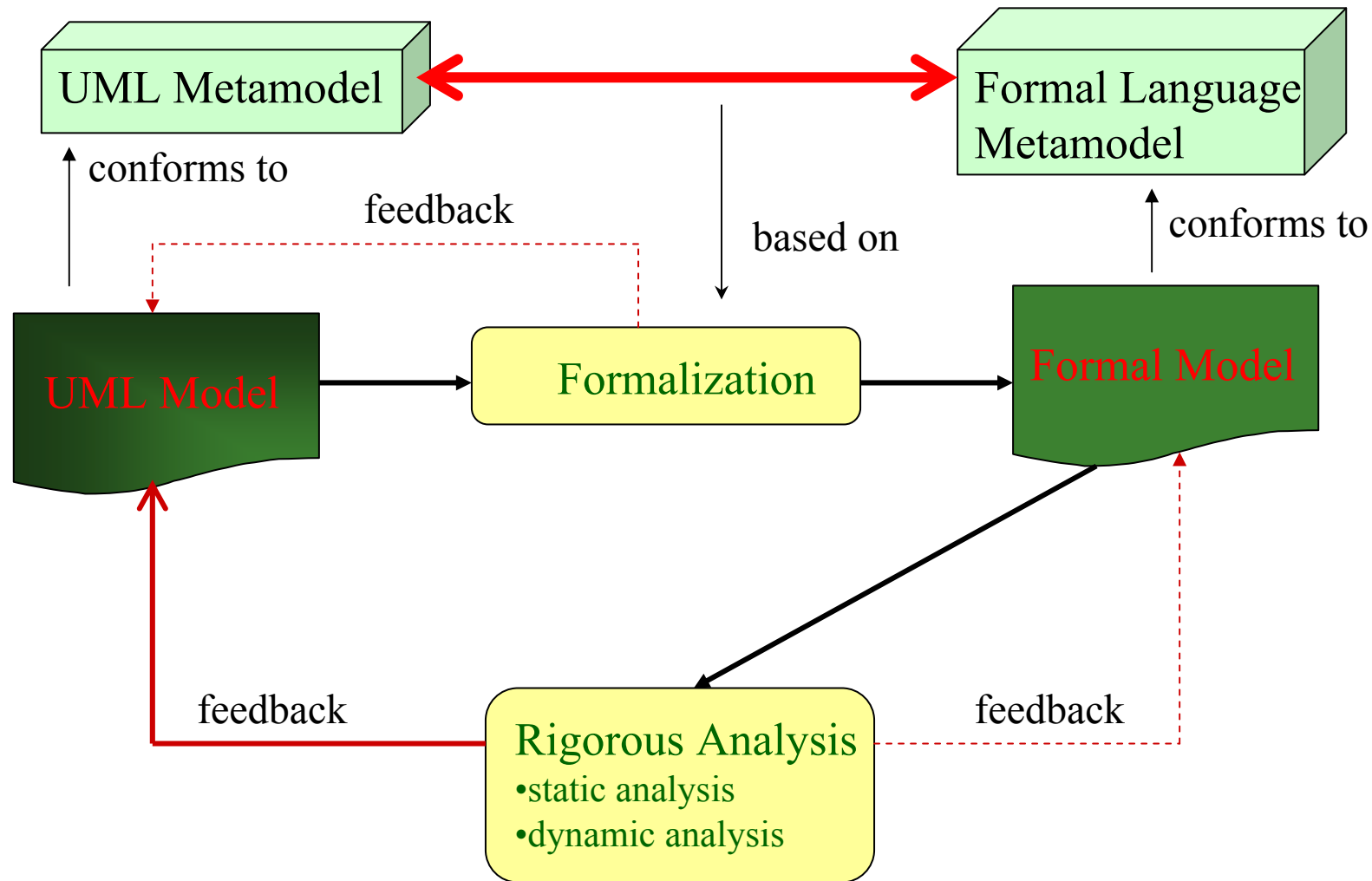
Integrated Methods



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, ...)

Integrated Methods



Separating Concerns

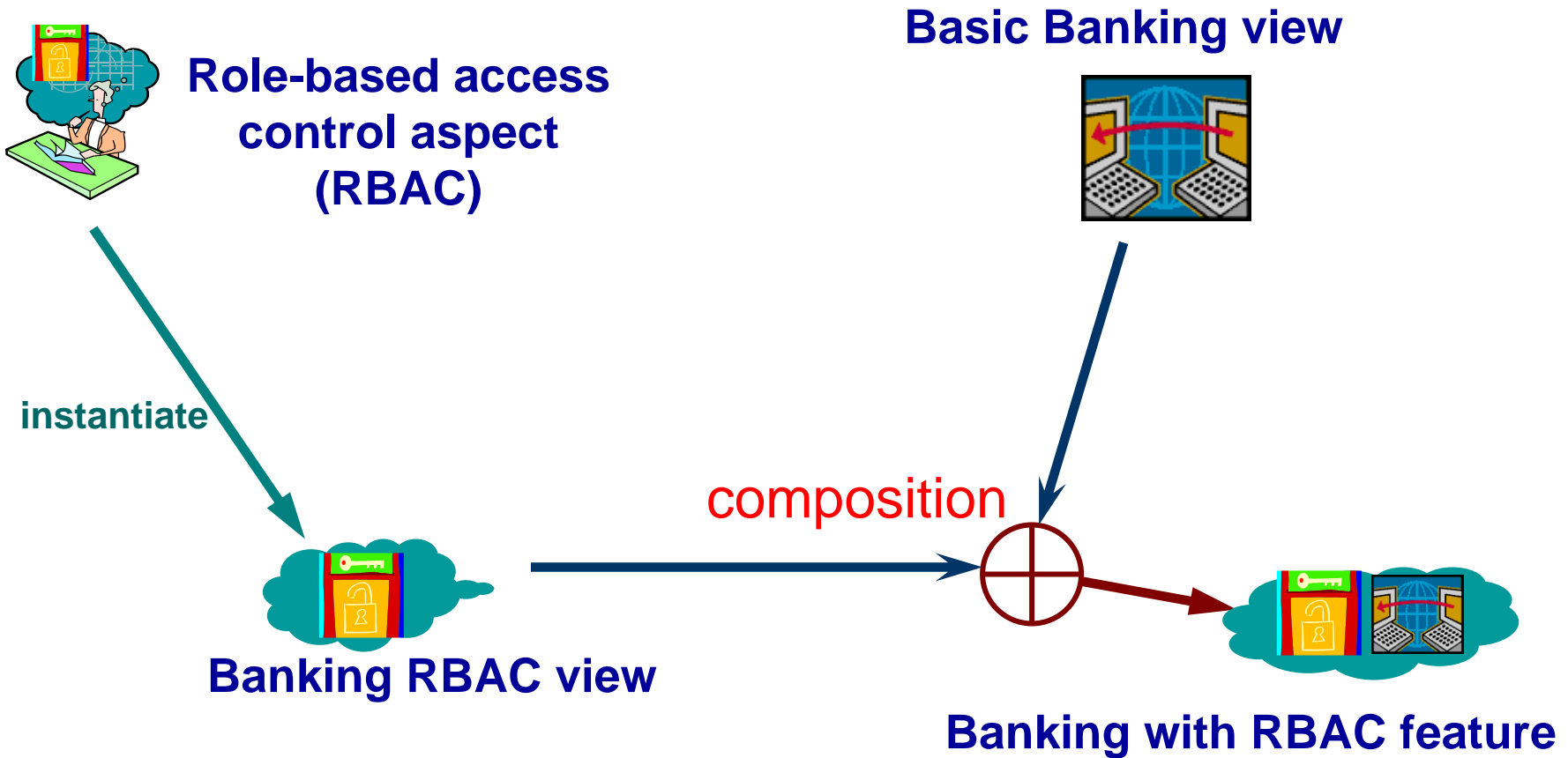


Developers of mission-critical open distributed software systems need to balance multiple, interdependent design concerns such as **availability**, **performance**, **survivability**, **fault tolerance**, and **security**.

Supporting separation of concerns

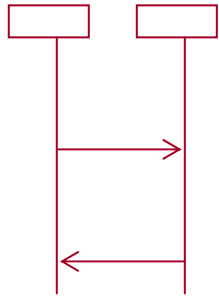
- UML 2.0: Supports modeling of artifacts using 13 diagram types
- OMG's MDA: advocates modeling systems using a fixed set of viewpoints (CIM, PIM, PSM)
- Challenge: Providing support for more flexible separation of concerns
 - Aspect Oriented Modeling

An AOM Example

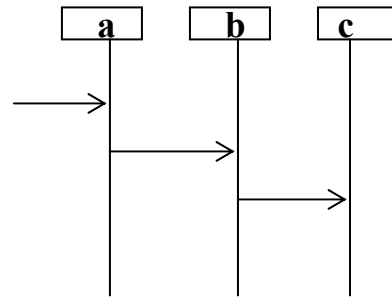


Aspect-Oriented Modeling

access control aspect

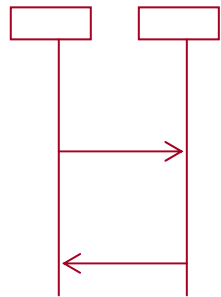


primary model



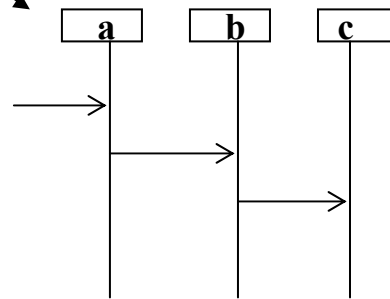
Aspect-Oriented Modeling

access control aspect



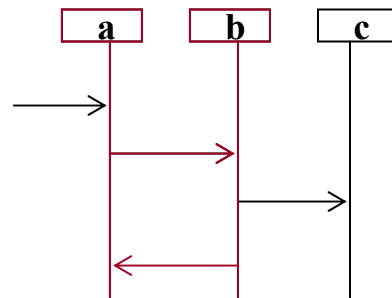
compose with

primary model



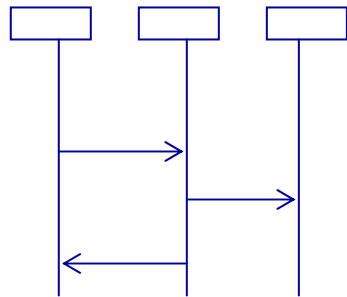
to produce

composed model



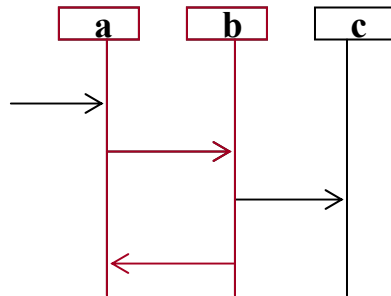
Aspect-Oriented Modeling

error recovery aspect

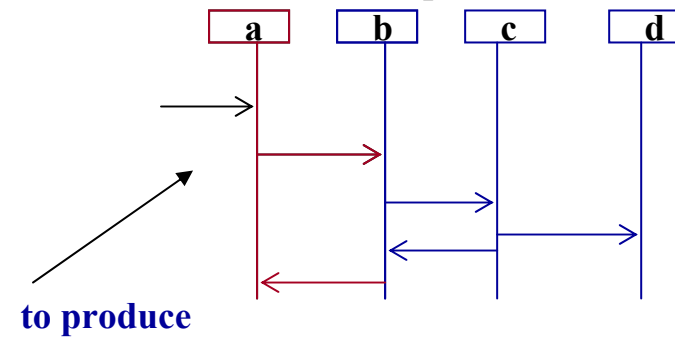


compose with

composed model

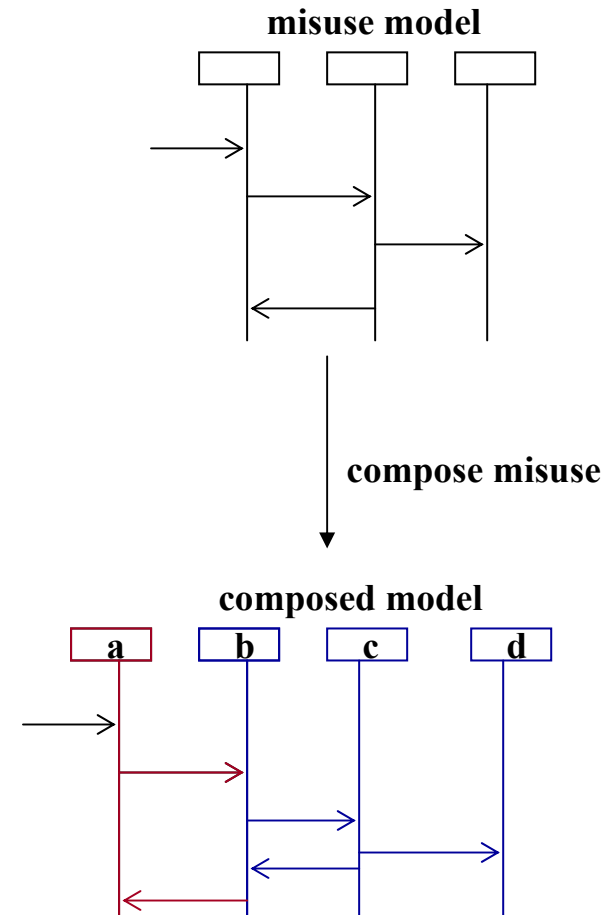


composed model



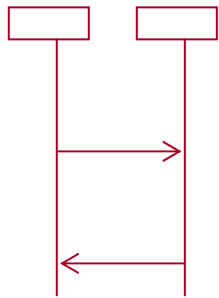
to produce

Aspect-Oriented Modeling



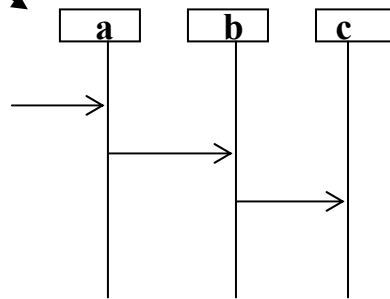
Aspect-Oriented Modeling

access control aspect

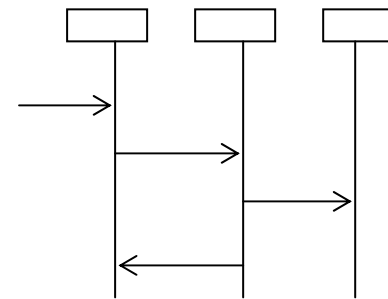


compose with

primary model

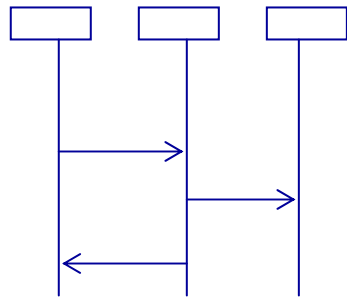


misuse model



compose misuse

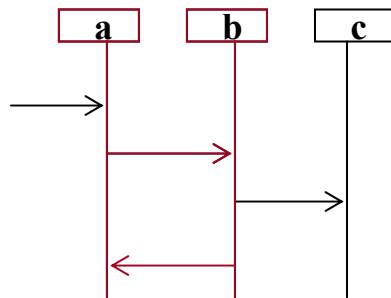
Error recovery aspect



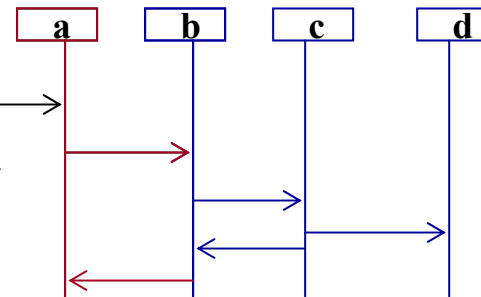
compose with

to produce

composed model



composed model



to produce

Separation of concerns challenges

- Supporting verifiable integration of views
 - Property preservation
 - Establishing presence or absence of emergent properties
- Use of aspect contracts
- Supporting evolution of models consisting of multiple overlapping views

Research in the viewpoint analysis and feature interactions domains should be leveraged

Model Manipulation and Management

“A model is something you can play with. You can change it, see how it responds, and in many other ways experiment with it.”

(from a slide show produced by Pille Bunnell, Douglas Tait, Inst. of Animal Resource Ecology, University of British Columbia)

Challenges

- Supporting model analysis
 - What is a “good” model?
 - Analyzing functional properties and system attributes using models
- Supporting model transformations, evolution, roundtrip engineering
 - Traceability
 - Model versioning
 - Integrating generated code and legacy systems
 - Analyzing model transformations
- Supporting distributed multi-developer modeling environments
 - ModelBus (ModelWare/ModelPlex EU project)
- Supporting mega-modeling
 - Models as manipulable entities
 - Model type systems

Beyond development models ...

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

Summary

- Software engineering (technical aspects) is essentially a modeling activity
 - MDE highlights the importance of models as explicit representations of abstractions
- Focus should be on developing a foundation for building MDE frameworks (meta-tool environments)
- Technologies for effectively managing the web of models produced in an MDE project are critical to adoption of MDE in industry
- The MDE vision may not be realizable in its entirety
 - But close approximations can reap benefits
 - Building close approximations will require developing successive generations of technologies;
 - Each new generation should address the accidental complexities of the previous generation

Accelerating MDE research

- Need facilities for collecting, analyzing and sharing modeling experience
 - A number of initiatives are taking form: PlanetMDE, ZoooM, Open Models Initiative, REMODD

Learn more about MDE by



Model Driven Engineering Languages & Systems

attending IEEE/ACM MODELS 2007

Nashville, Tennessee, USA

<http://www.modelsconference.org>

**reading Software and System
Modeling Journal (SoSyM)**

Springer

<http://www.sosym.org>

