CSC2125: Modeling Methods, Tools and Techniques
Winter 2018

Marsha Chechik

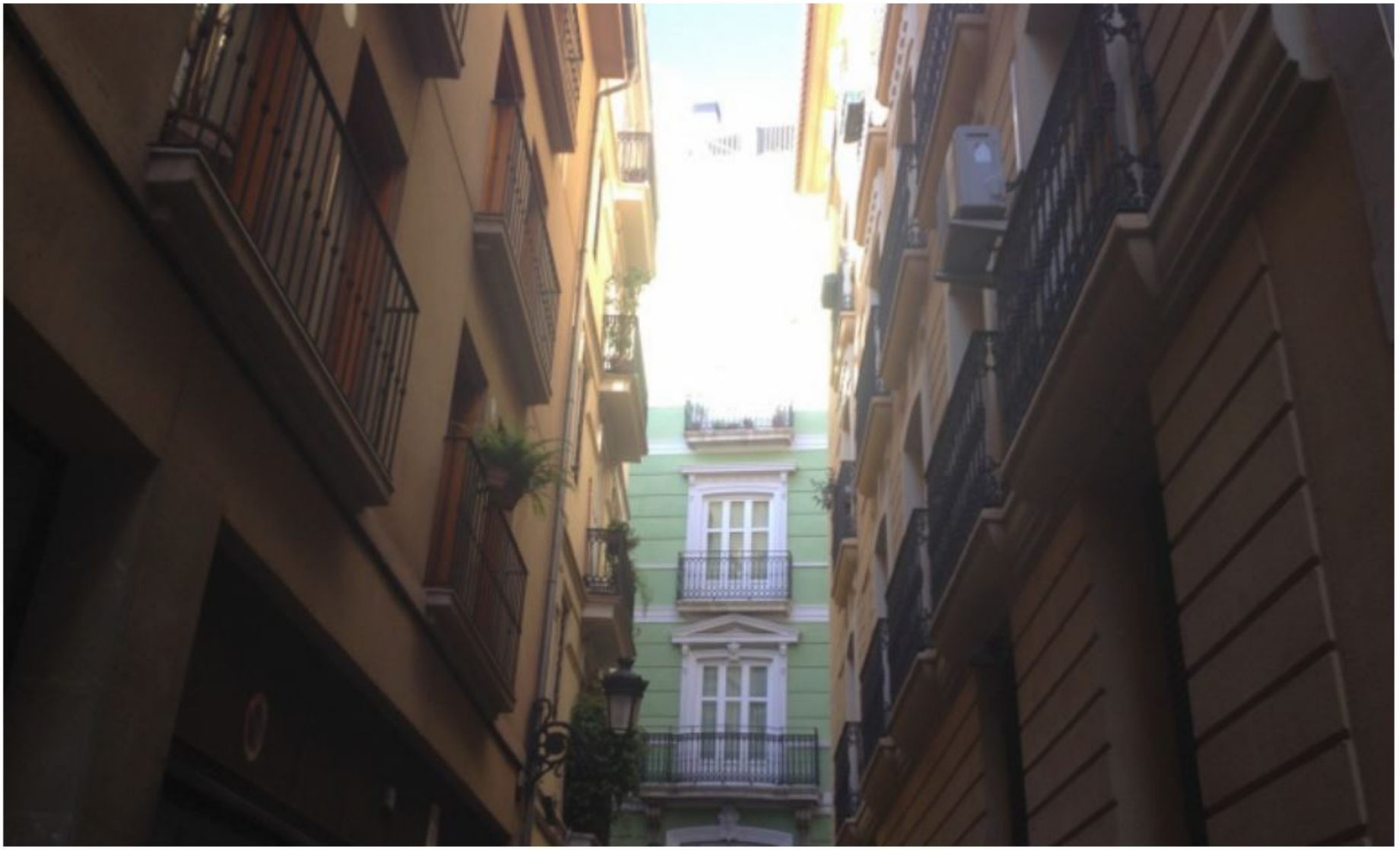
Department of Computer Science
University of Toronto

Software Models

http://www.cs.toronto.edu/~chechik/courses18/csc2125

Plan for the rest of the lecture

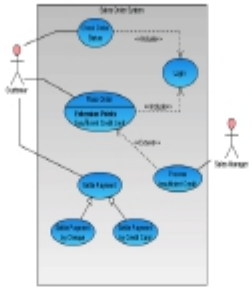
- Part 0. Modeling
- Part 1. Software Models, UML, OCL
- Part 2. Meta-modeling, model mappings, DSLs / generic languages, model transformations
- Part 3 (if we have time). How usable are models?

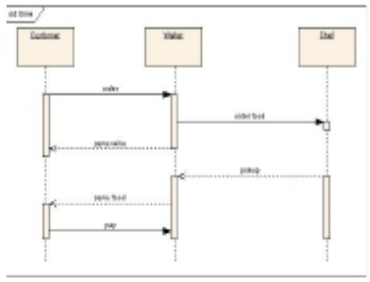


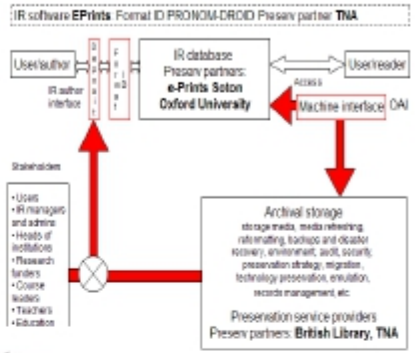
Model Driven Engineering

First: Models

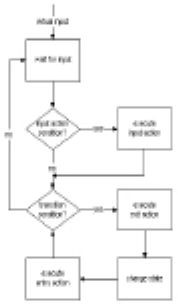
What is Being Modelled?

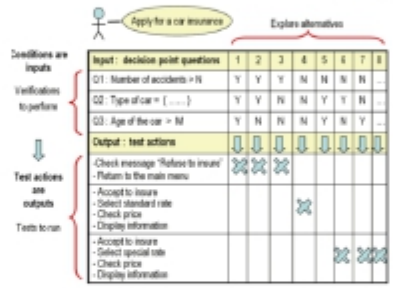


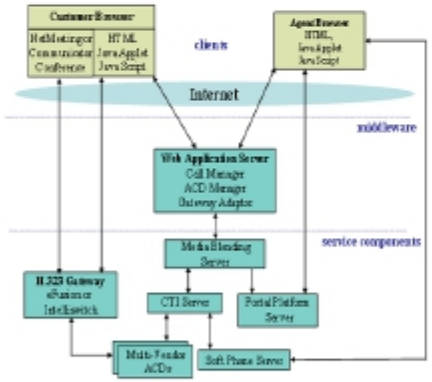


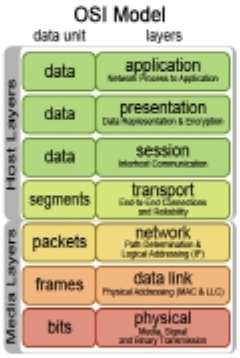


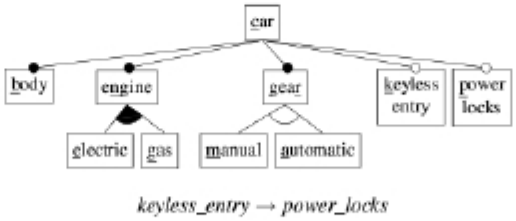
<postal-address> ::= <name-part> <street-address> <zip-part>
<name-part> ::= <personal-part> <last-name> <opt-jr-part> <EOL>
| <personal-part> <name-part>

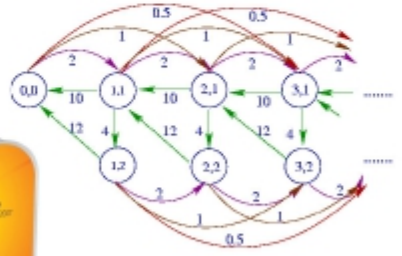






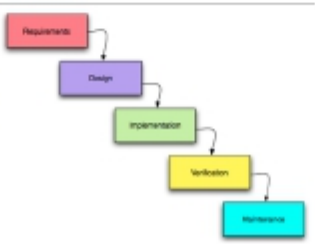


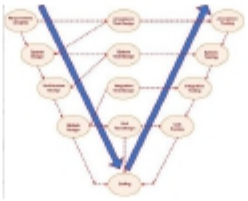




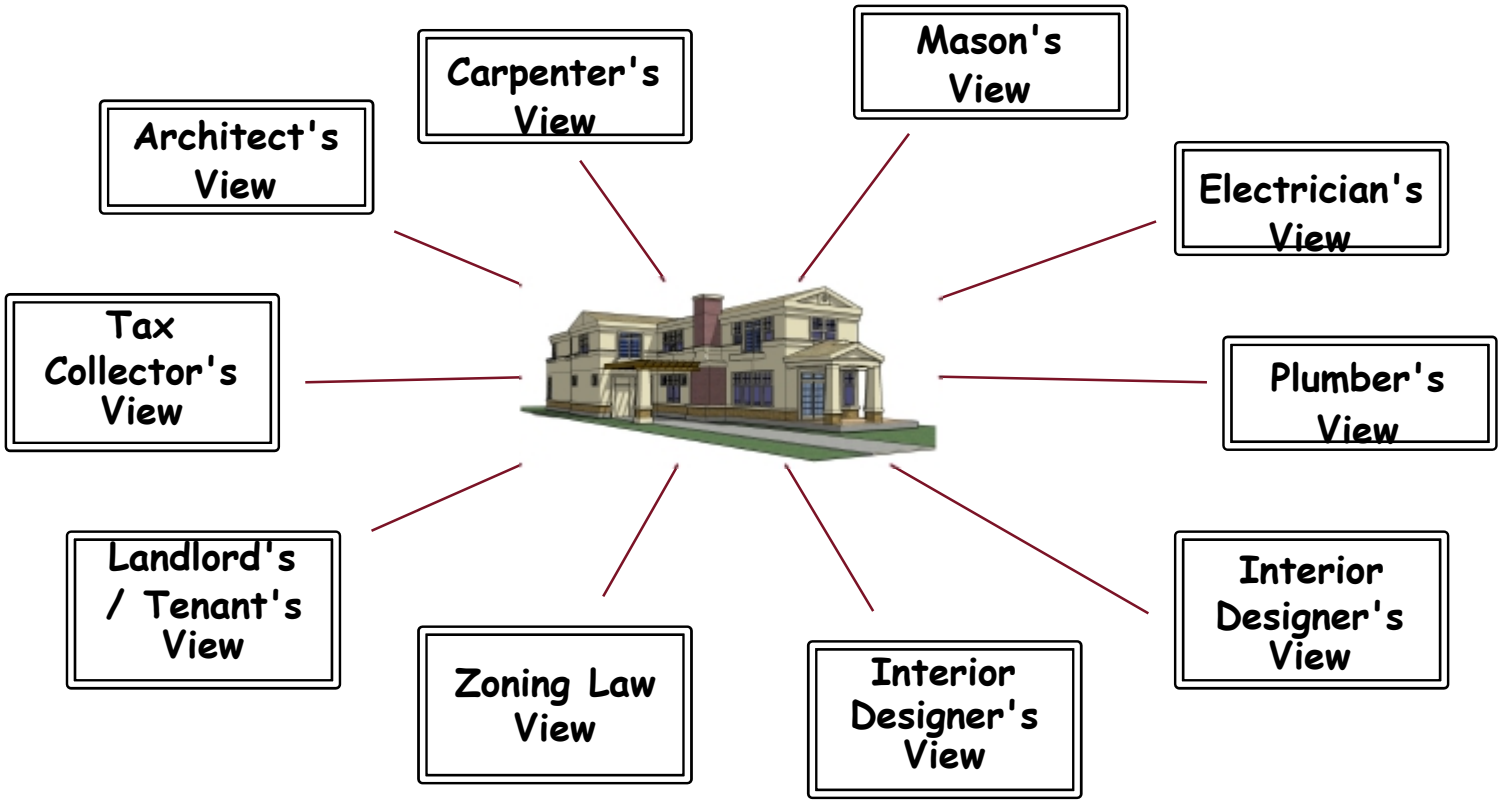
E = a KLOC**b**

	Risks (Failure Modes)					
Countermeasures	Participant does not read emails	Participant does not reply to requests	Room with required equipment is not available	System response is too close to meeting	Important participant has last minute change	**Overall single effect of countermeasure**
Criticality (risk)	**0.264**	**0.468**	**0.01**	**0.297**	**0.375**	
Email reminder sent	0.7	0.7	0	0.1	0	0.542
Change the meeting, increase time range	0.2	0.2	0	0.1	0	0.176
System has access to personal e-agendas	0.3	0.2	0.1	0.2	0.3	0.346
Change the meeting, fewer constraints (equipment)	0	0	0.9	0	0	.009
Cancel a meeting and send email confirmation	0.8	0.8	1	0.7	0.9	1.141
Combined risk reduction	**0.966**	**0.962**	**1**	**0.806**	**0.93**	





Models as Views

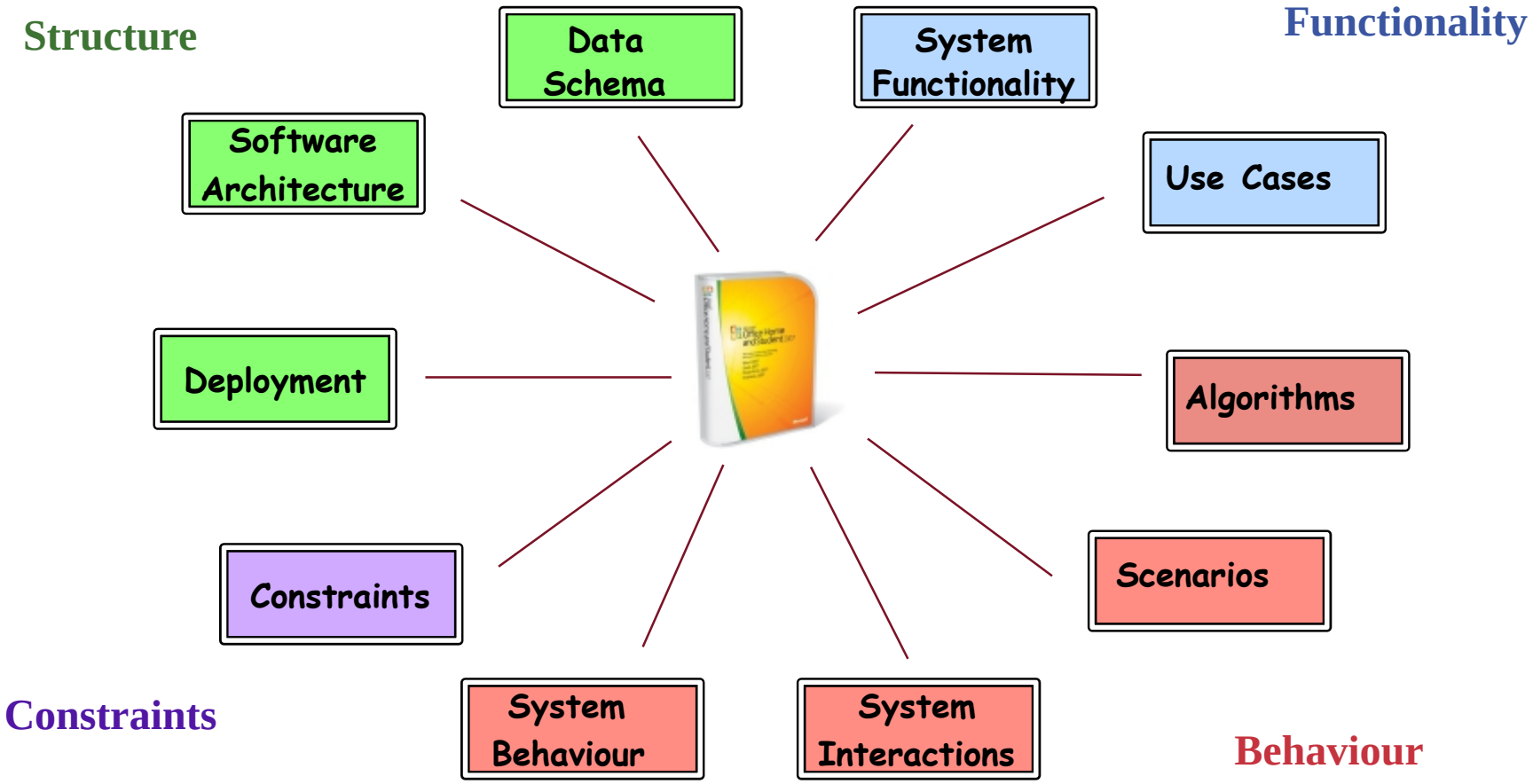


Every view

- obtained by a different projection, abstraction, translation
- may be expressed in a different notation (modelling language)
- reflects a different intent

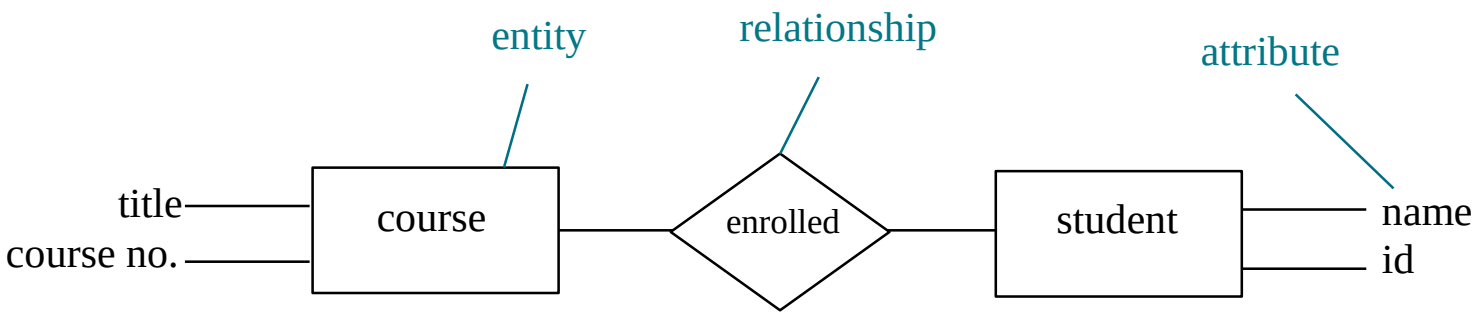
Modelling Paradigms

Fundamental modelling paradigms, each emphasizing some basic view of the software to be developed.



Entity-Relationship Diagrams

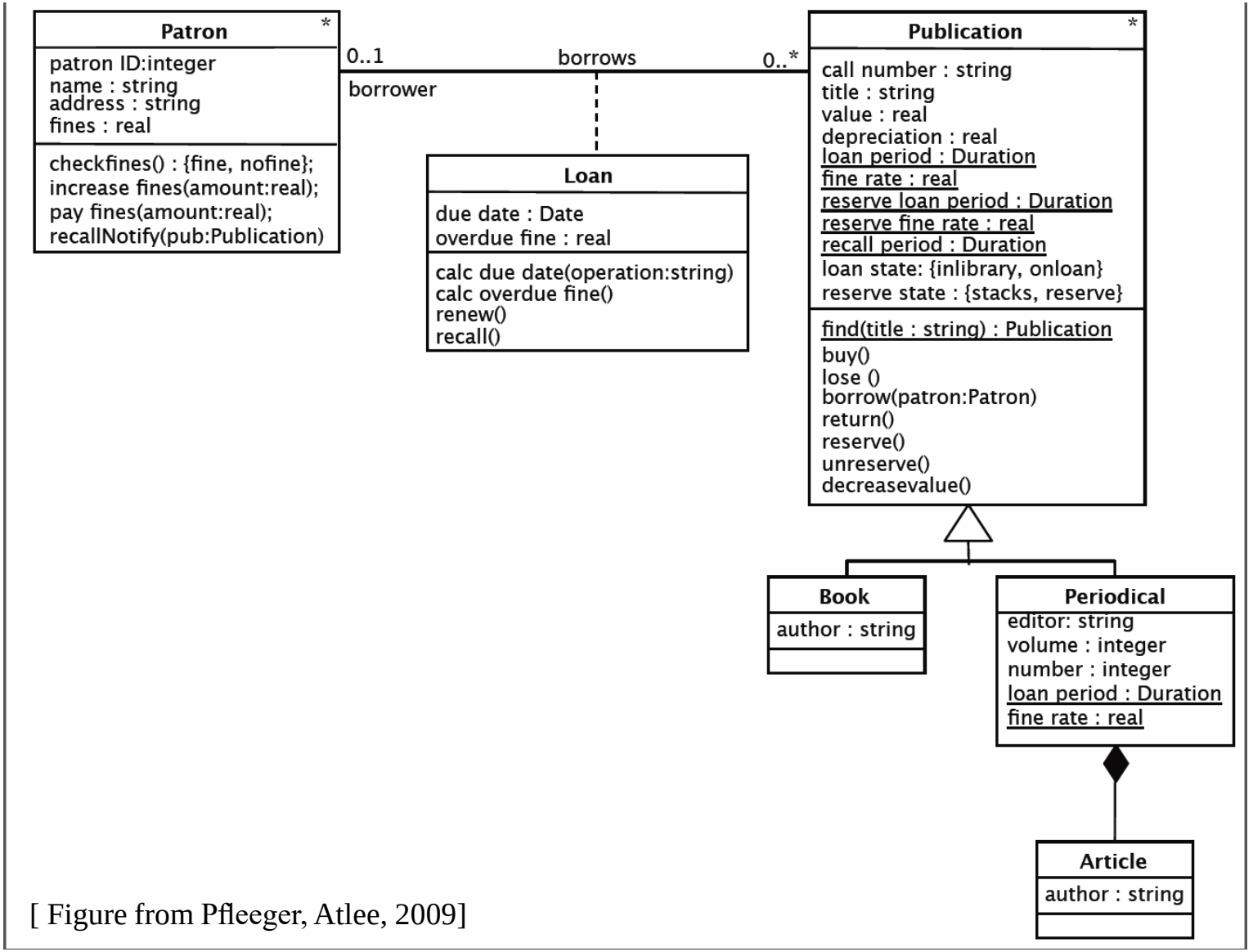
An ER diagram is a *structural model* representing a software system's data elements and relationships among them.



- originally invented for model database design (Chen, 1976)
- emphasizes concepts/data
- relationships can represent associations, navigability, containment, dependencies, etc.

UML Class Diagrams

UML Class Diagrams are an elaborate form of ER diagram.

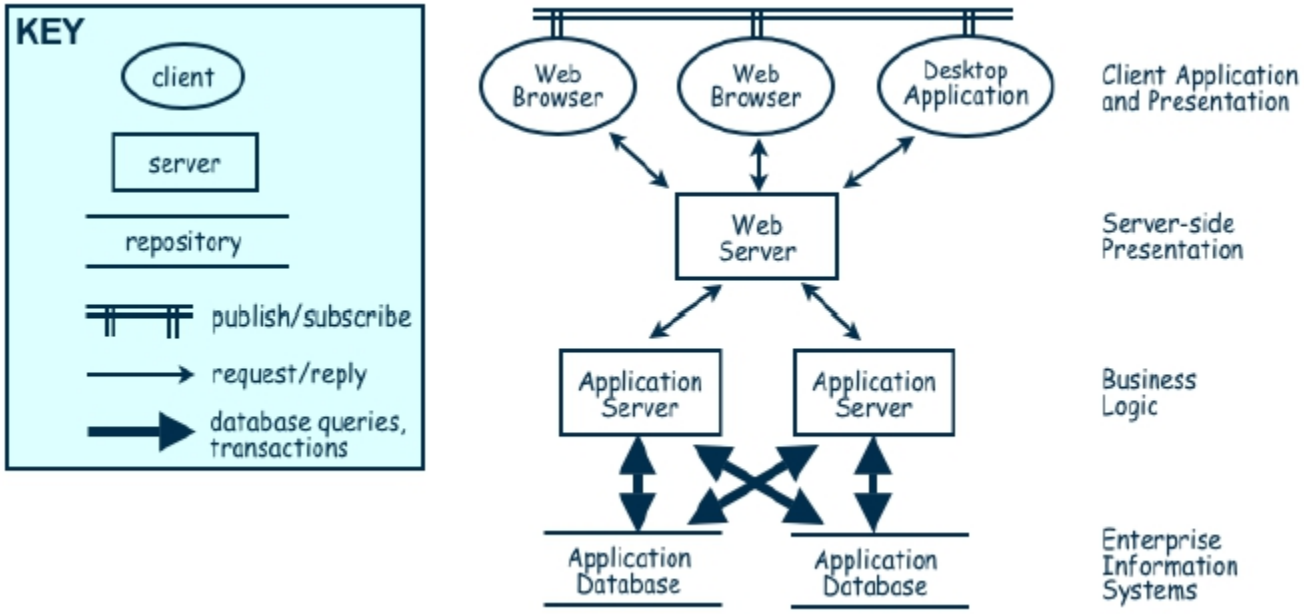


[Figure from Pfleeger, Atlee, 2009]

Software Architecture

A software architecture is high-level model of code *structure*.

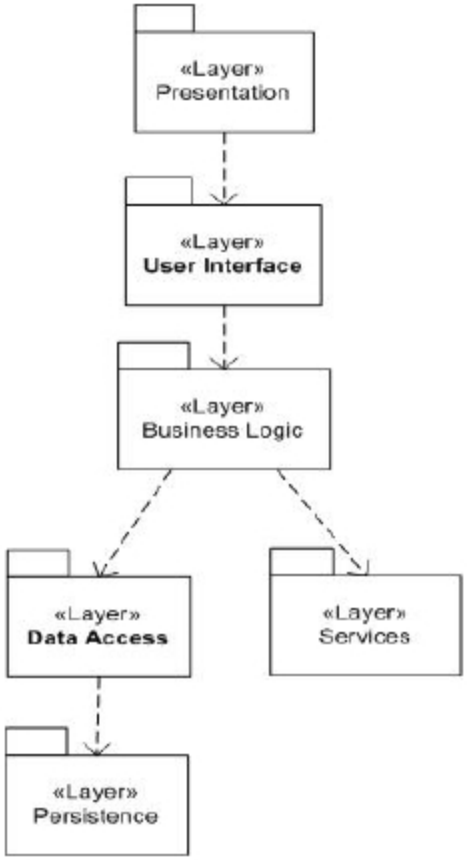
Typically modelled as a "box and arrow" diagram, with a key explaining the types of components (boxes) and connectors (arrows).



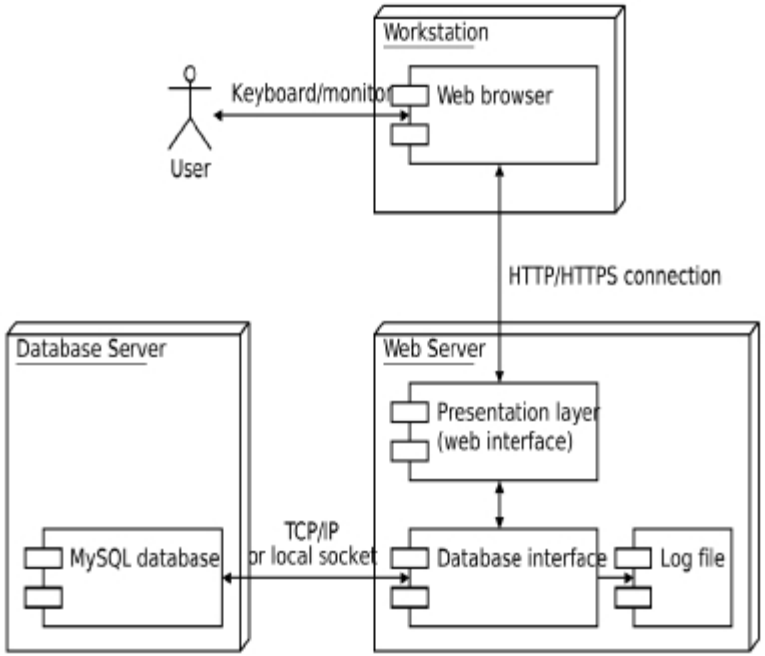
[Figure from Pfleeger, Atlee, 2009]

UML "Software Architecture" Models

The closest that UML comes to a software architecture model are UML Package Diagrams and UML Deployment Diagrams.



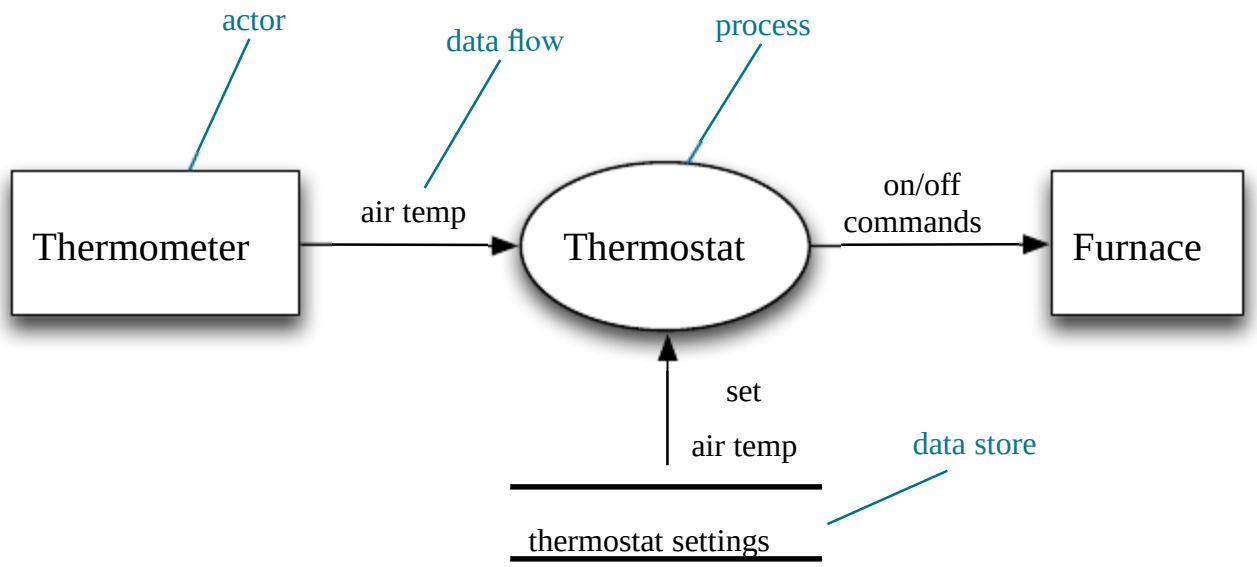
UML Package Diagram



UML Deployment Diagram

Data Flow Diagrams (DFDs)

Descriptive model of functional decomposition of the system, and the data dependencies between functions.

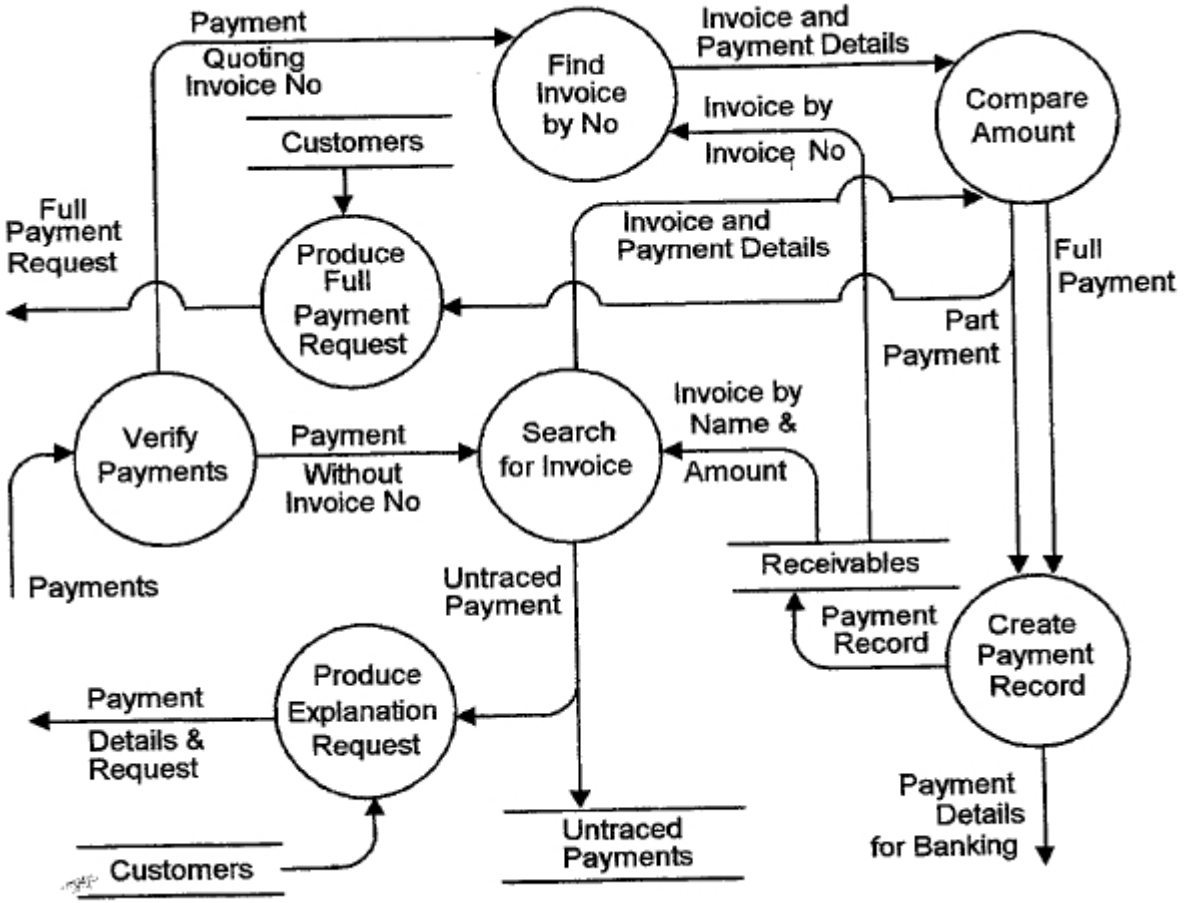


DFDs model

- collection of functions
- sources and sinks of data
- data dependencies

Data Flow Diagrams (DFDs)

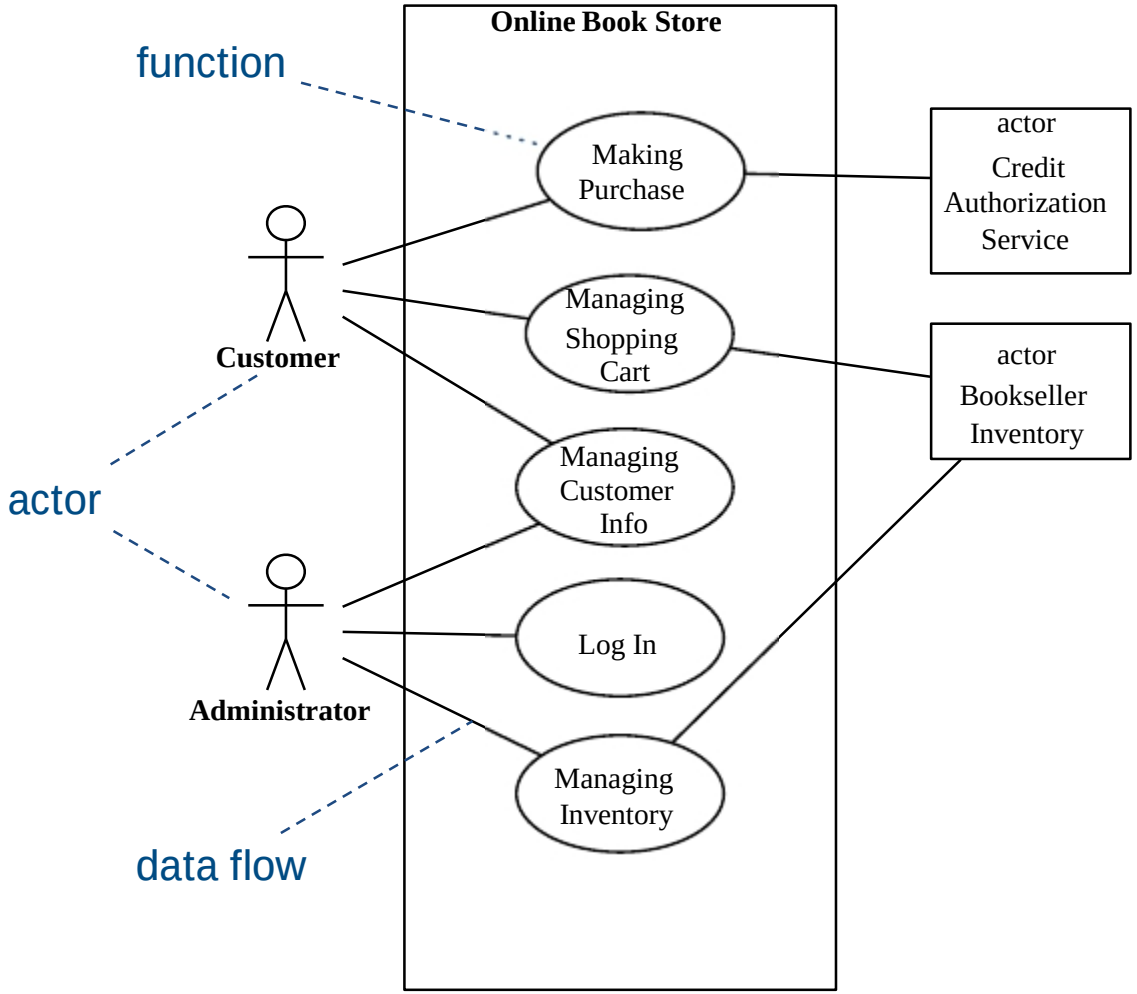
Although DFDs are good for communicating the big picture, they are inherently incomplete, undetailed, and ambiguous.



[Figure from M. Jackson, 1995]

UML Use Case Diagrams

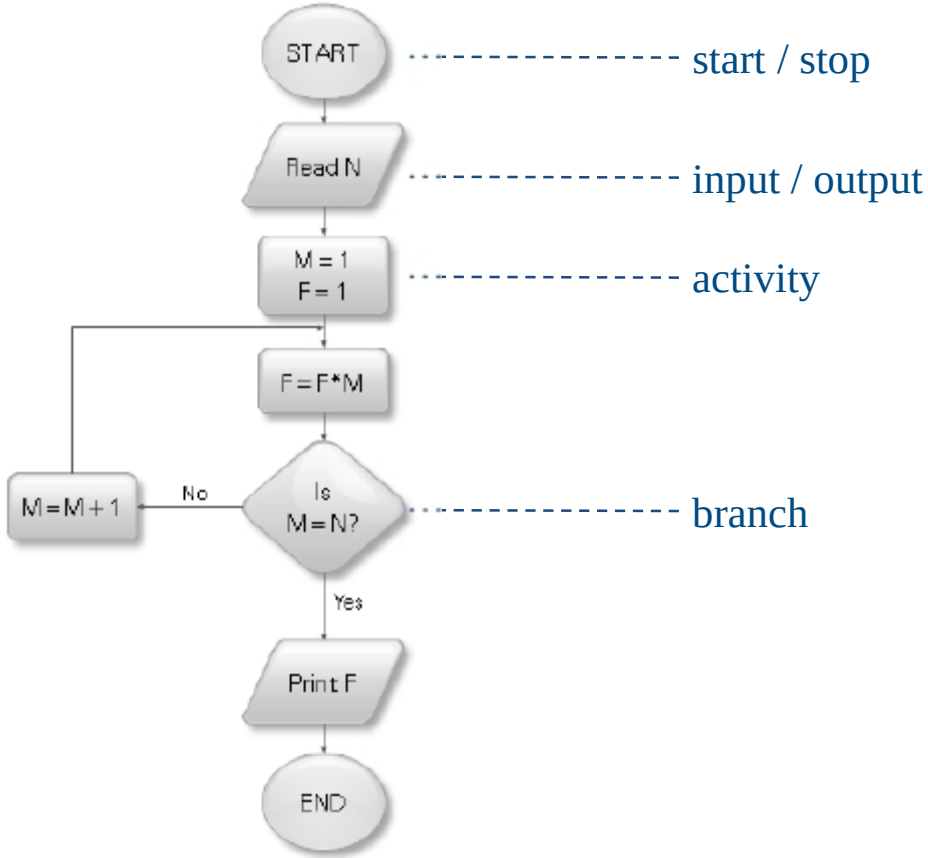
Use Case Diagrams are a very high-level data-flow diagram.



Flowcharts

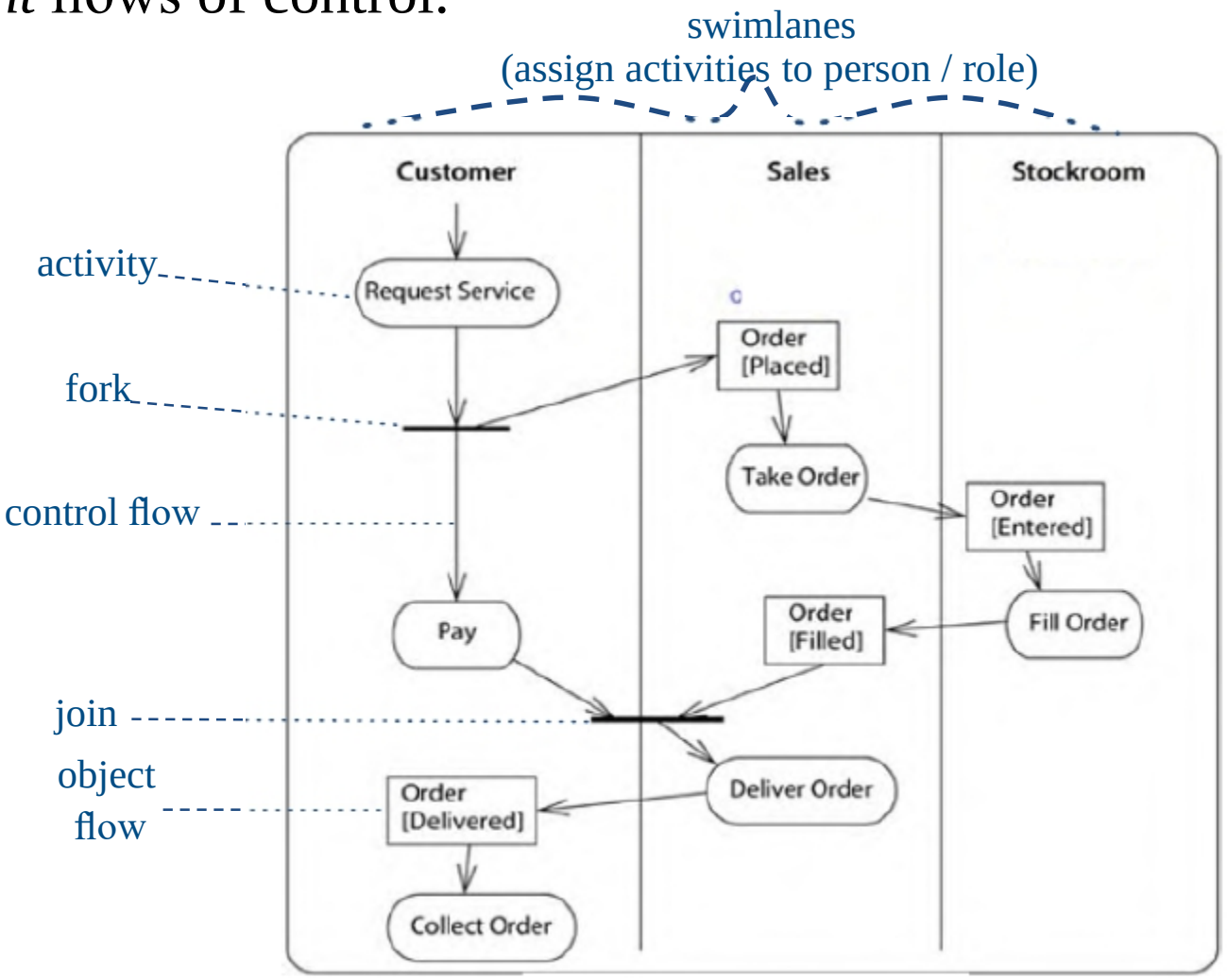
Flowcharts are an ancient modelling notation, for representing *behaviour* in terms of steps of an algorithm.

Depict control flow rather than data flow



UMLActivity Charts

UMLActivity Charts are a variation on flowcharts that support *concurrent* flows of control.

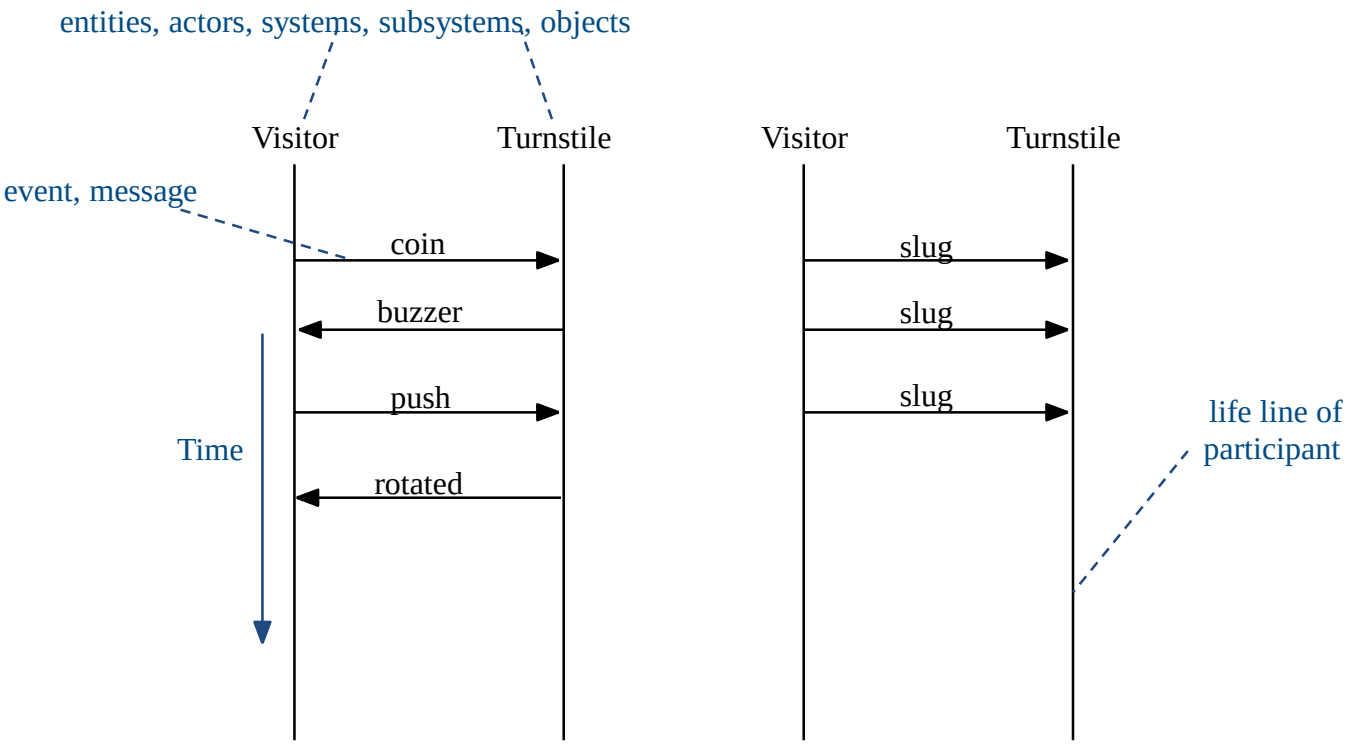


[Figure from M. Blaha, J. Rumbaugh, 2005]

Event Traces

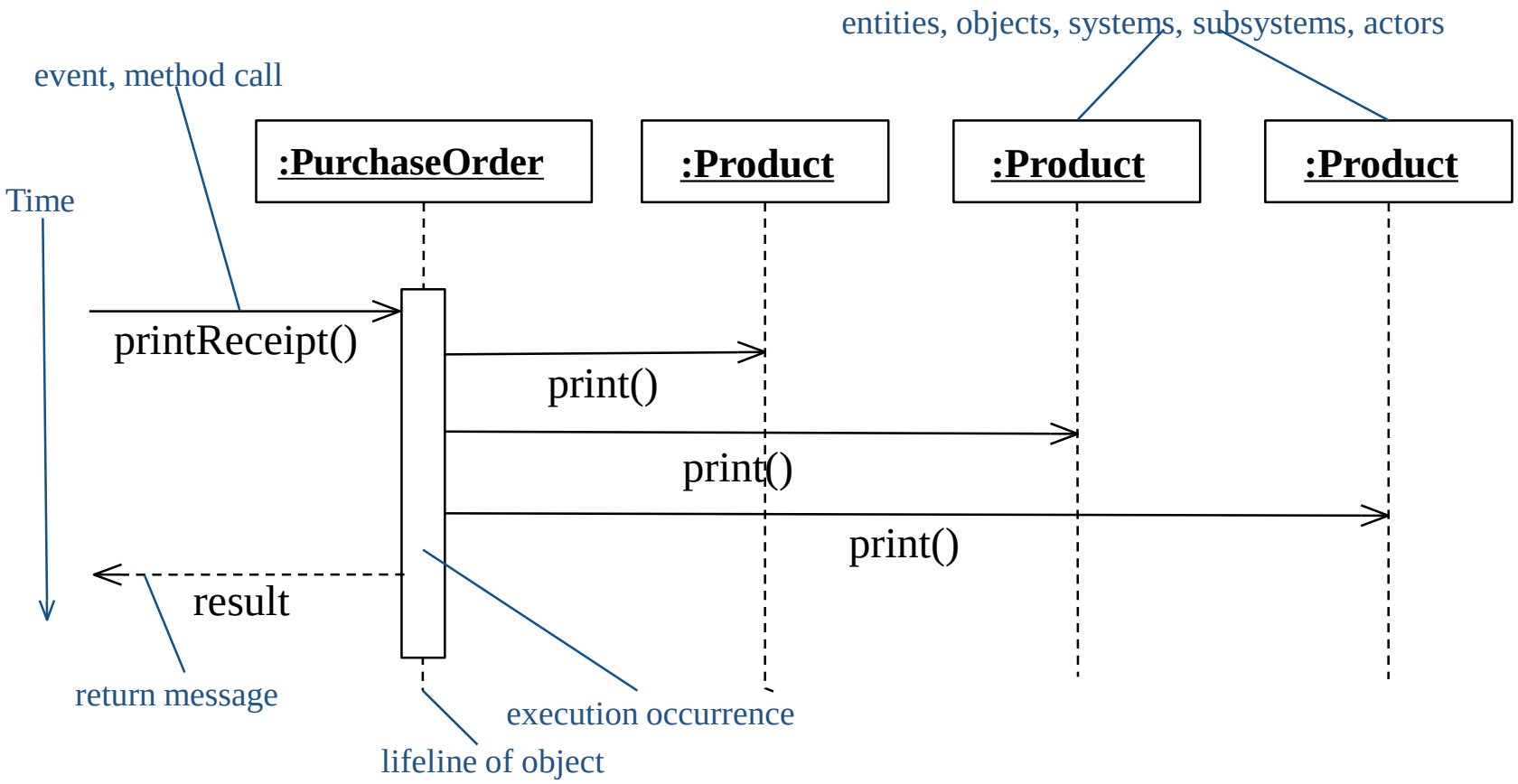
Dynamic model of *behaviour* showing communication among entities in one scenario (execution trace).

Shows a slice of behaviour, not complete behaviour.



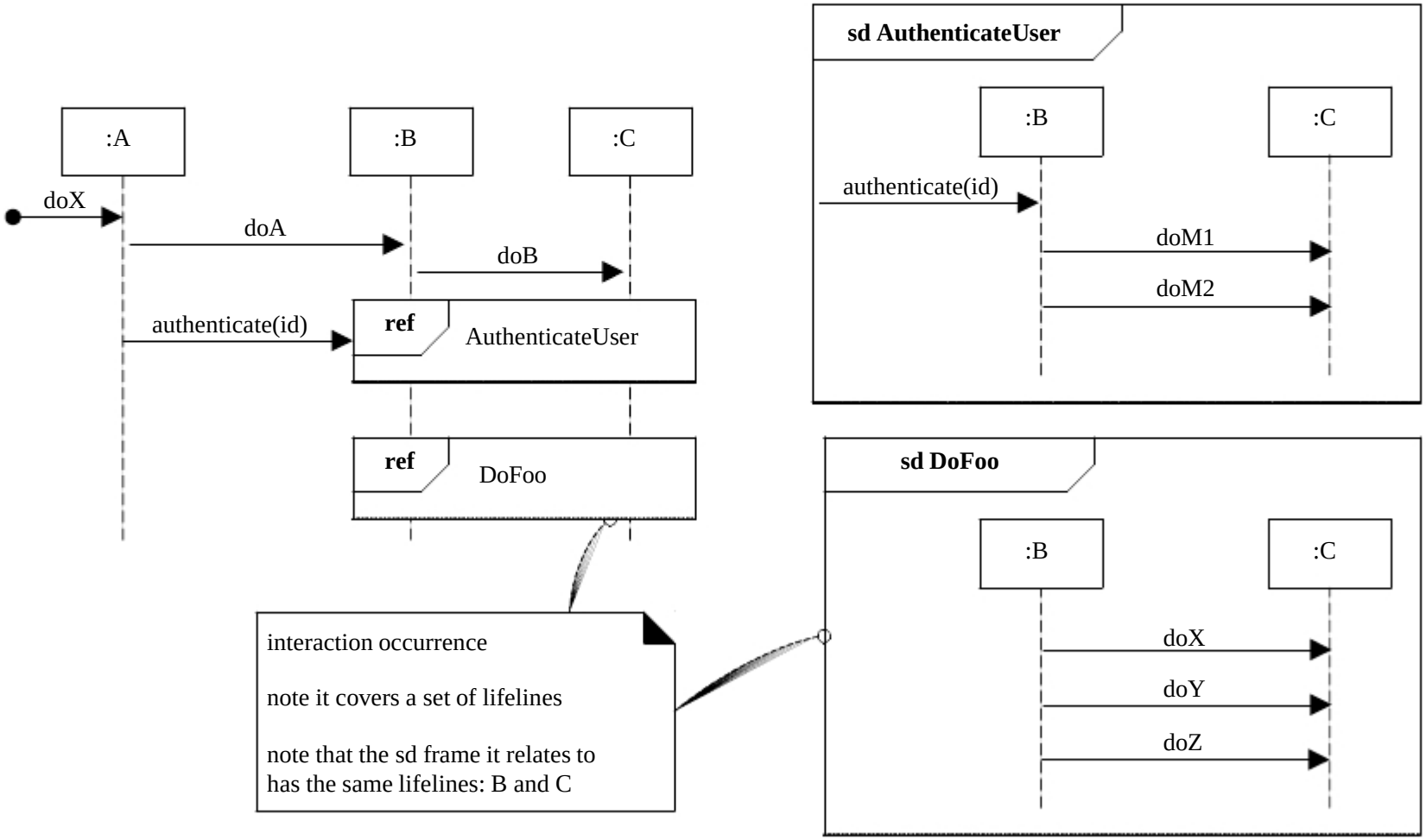
UML Sequence Diagrams

UML Sequence Diagrams are elaborate event traces....



UML Sequence Diagrams

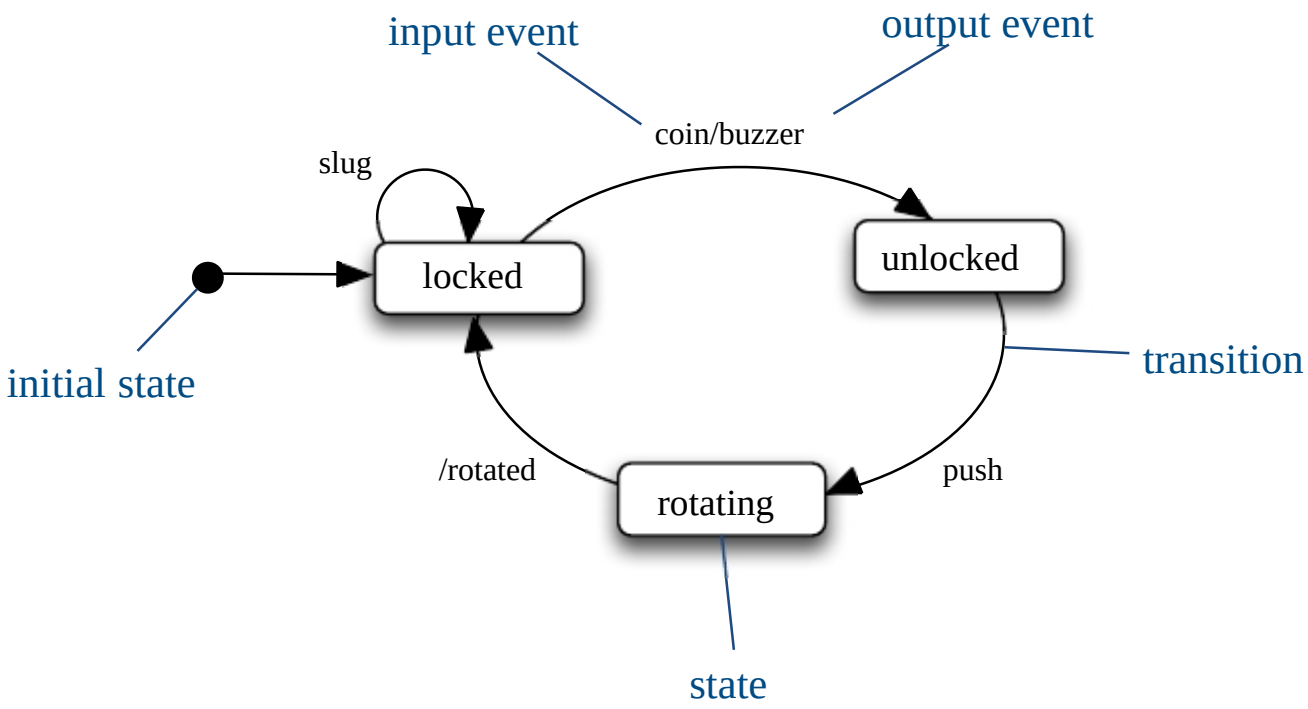
... including branches, loops, concurrency, optional subsequences, references to other sequence diagrams.



Larman, *Appliying UML and Patterns, 3ed*

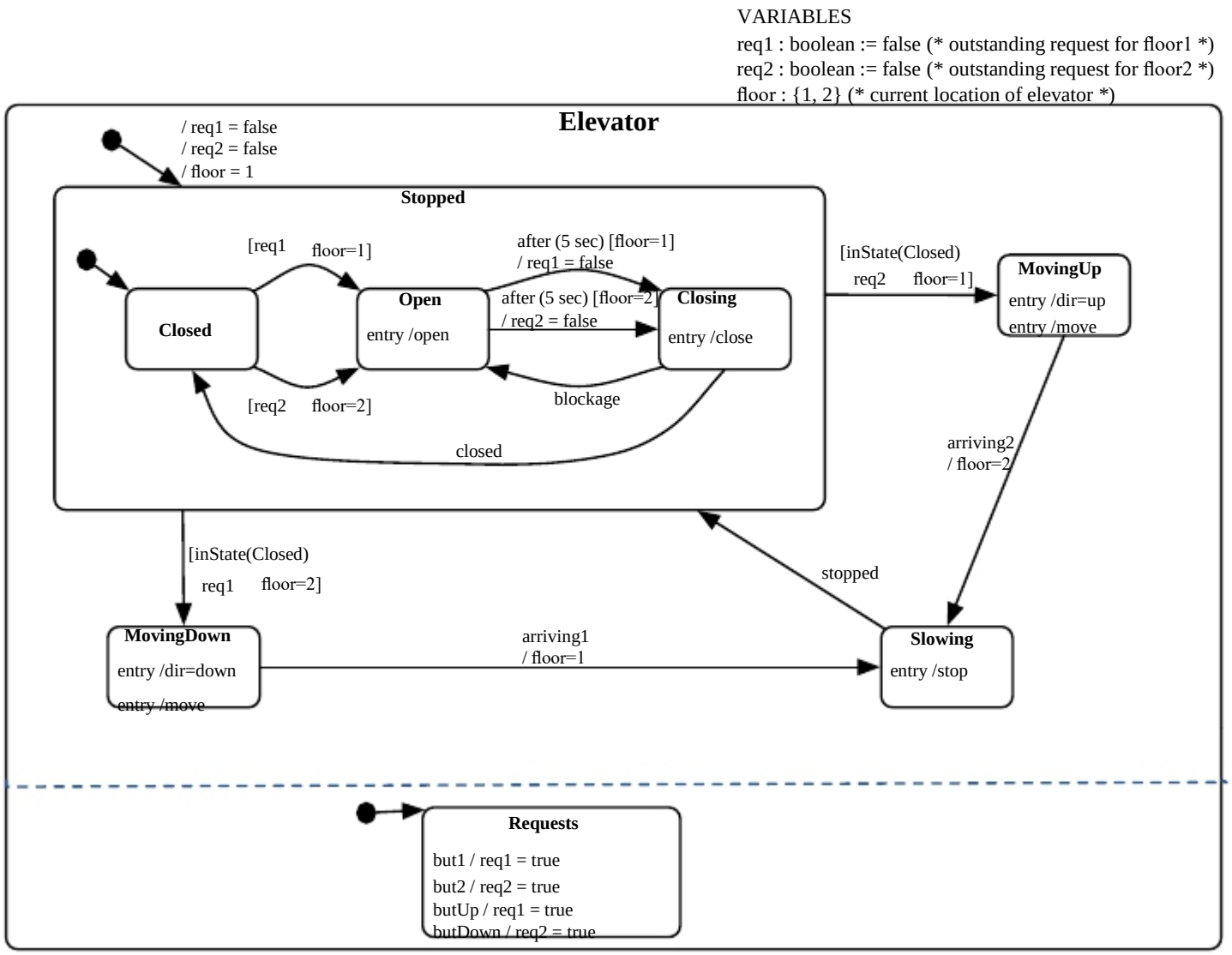
State Machines

Compact representation of all event traces.



UML StateMachine Diagrams

UML StateMachine Diagrams borrow heavily from David Harel's statecharts.



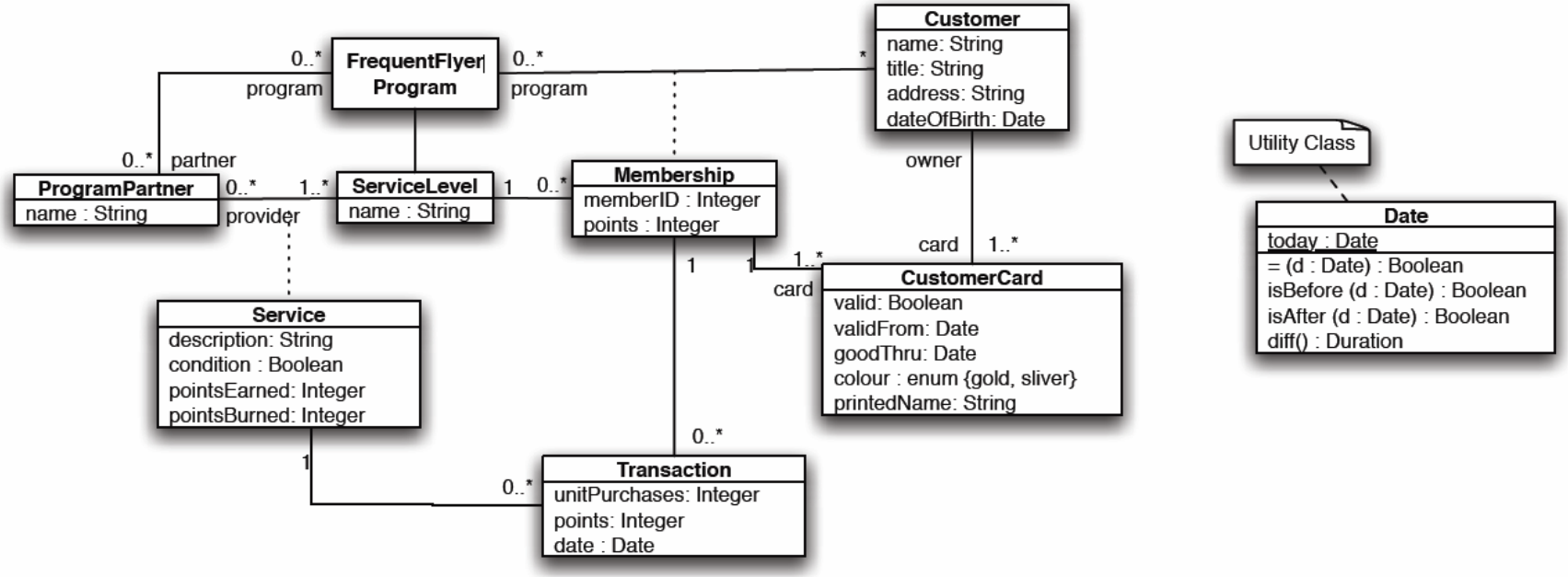
Logic

Logic is the basis for a number of languages that express constraints on allowable interpretations of other models.

- allowable instantiations of data models
- invariants among attribute values in data models
- pre/post conditions of functions
- event conditions in event traces
- guard conditions in state machines

Object Constraint Language

OCL was designed for expressing constraints on UML diagrams.



context CustomerCard **inv**
self.printedName = (self.owner.title.concat(self.ownername))

context CustomerCard **inv**
(self.valid and self.colour=gold) implied self.membership.serviceLevel = "gold"

UML: 13 Different Diagram Types

Structural

- Class Diagrams
- Object Diagrams
- Composite Structure Diagrams
- Component Diagrams
- Package Diagrams
- Deployment Diagrams

Behavioural

- Interaction Overview Diagrams
- Activity Diagrams
- Sequence Diagrams
- Communication Diagrams
- State Machine Diagrams
- Timing Diagrams

Functional

- Use Case Diagrams

OCL is a separate language that was invented for writing constraints on UML models

What else can be modeled?

- Assurance cases
- Feature diagrams
- Environment and controller
- Performance (quantitative models)
- Multiple products
- …..

Next up: Part 2

- Meta-modeling
- Mappings between models
- DSMLs / generic modeling languages
- Introduction to model transformations and their analysis

- Sources: Sahar Kokaly's lecture in CAS756 (McMaster, 2015), Rich Paige's lectures in York University, UK

References

[BlRu05] M. Blaha, J. Rumbaugh, *Object-Oriented Modeling and Design with UML*, 2ed, Prentice hall, 2005.

[BRJ05] G. Booch, J. Rumbaugh, I. Jacobson. *UML User Guide. 2nd Edition*. Addison Wesley. 2005.

[EJ09] S. Easterbrook, T. Johns, "Engineering the Software for Understanding Climate Change," *Computing in Science and Engineering*, pp. 65-74, November/December, 2009

[Jac95] M. Jackson, *Software Requirements and Specifications*, ACM Press, 1995.

[Kra07] J. Kramer. "Is Abstraction the key to Computing?" *Communications of the ACM*. April 2007/Vol. 50, No. 4.

[KSLB03] G. Karsai, J. Sztipanovits, A. Ledeczi, T. Bapty. "Model-Integrated Development of Embedded Software.", *Proc. IEEE*, Jan 2003, pp 145-164.

References

[KT08] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley 2008.

[Lud04] Ludewig, J. 2004. "Models in software engineering--an introduction". *Software and Systems Modeling* 2, 5-14.

[PfAt09] S. Pfleeger, J. Atlee, *Software Engineering: Theory and Practice*, Prentice Hall, 2009.

[RJB05] Rumbaugh, Jacobson, Booch, *The Unified Modeling Language Reference Manual*, 2nd ed., Addison-Wesley, 2005.

[Sch06] D.C. Schmidt. "Model-Driven Engineering." *IEEE Computer*, vol. 39, no. 2, Feb. 2006. Page(s):25 - 31

[Sei03] Ed Seidewitz. "What Models Mean." *IEEE Software*, vol. 20, no. 5, pp. 26-32, Sep./Oct. 2003.

[Sel03] Bran Selic, "The Pragmatics of Model-Driven Development," *IEEE Software*, vol. 20, no. 5, pp. 19-25, Sep./Oct. 2003.

[Sta73] H. Stachowiak, *Allgemeine Modelltheorie*, Springer-Verlag, 1973.