IBM Software Group

Rational. software

Software Models, the Unified Modeling Language, and Real-Time UML

Bran Selic
IBM Distinguished Engineer
IBM Rational Software Canada
bselic@ca.ibm.com

@business on demand software

The State of the Art?

- Deeply embedded in the software for controlling long-distance phone traffic routing sat the following (kind of) code:

```
                    …
switch (caseIndex) {
case‘A’:        route = routeA;
                    …
                    break;
                    …
case‘M’:        route = routeM;

case‘N’:        route = routeN;
                    …
                    break;
              …}
```

Missing “break” statement!

- Consequence: Loss of long-distance phone service in NE USA
- Total cost ≈ $800 M (1990)

Accidental Complexity

- Fred Brooks: *The Mythical Man-Month*
- *Essential complexity:* inherent in the problem and cannot be eliminated by technological or methodological means
  - E.g., making airplanes fly
- *Accidental complexity:* unnecessary complexity introduced by a technology or method
  - E.g., building construction without using power tools
  - …or, translating designs (models) into programs without the help of computers

Today's Recipe

- 2 messages:
  - Describe advanced methods for developing embedded software systems
  - Describe recent OMG standards that support model-driven engineering of embedded software systems
- In 3 parts:
  - Part I: The role of modeling in software development
  - Part II: The semantic foundations: UML (2.0)
  - Part III: Model-driven engineering in real-time systems using UML

Part I:
Models, Software Models, and Model-Driven Development

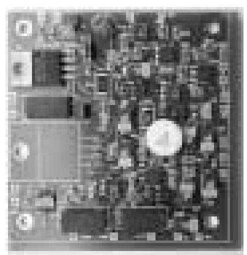
Models in Traditional Engineering

- Probably as old as engineering (e.g., Vitruvius)

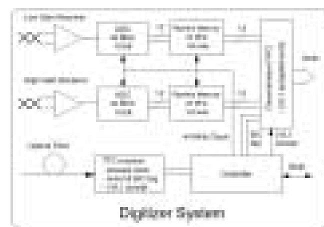
Engineering Models

- Engineering model:

  A reduced representation of some system that highlights the properties of interest from a given viewpoint



Modeled system



Functional Model

- We don't see everything at once
- We use a representation (notation) that is easily understood for the purpose on hand

How Engineering Models are Used

1. To help us understand complex systems
   - Useful for both *requirements* and *designs*
   - Minimize risk by detecting errors and omissions early in the design cycle (at low cost)
     - Through analysis and experimentation
     - Investigate and compare alternative solutions
   - To communicate understanding
     - Stakeholders: Clients, users, implementers, testers, documenters, etc.
2. To drive implementation
   - The model as a blueprint for construction

Characteristics of Useful Models

- Abstract
  - Emphasize important aspects while removing irrelevant ones
- Understandable
  - Expressed in a form that is readily understood by observers
- Accurate
  - Faithfully represents the modeled system
- Predictive
  - Can be used to answer questions about the modeled system
- Inexpensive
  - Much cheaper to construct and study than the modeled system

> *To be useful, engineering models must satisfy <u>all</u> of these characteristics!*

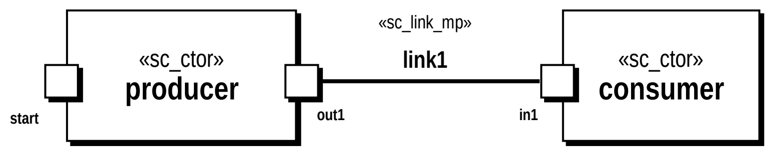
A Bit of Modern Software…

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
```

```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```

Can you spot the architecture?

…and its UML Model

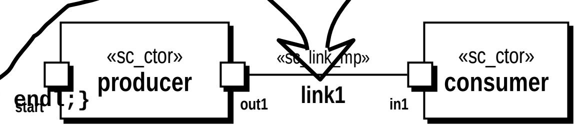


Can you spot the architecture?

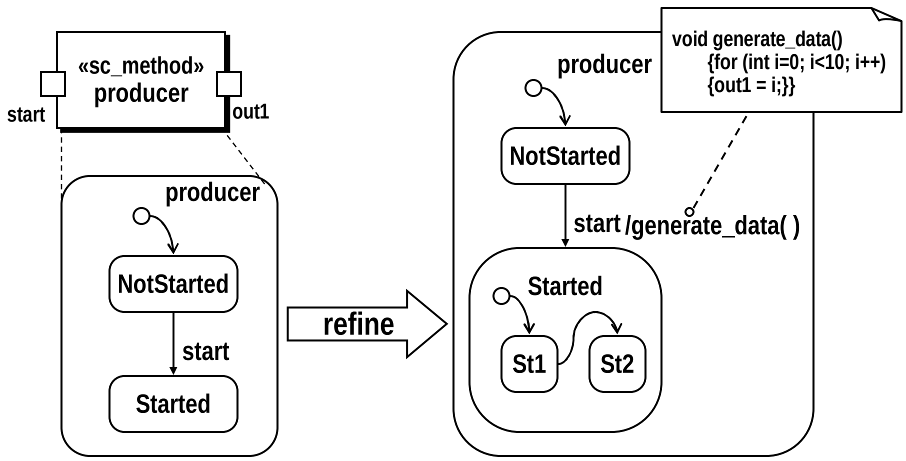
The Software and Its Model

```
SC_MODULE(producer)
{
sc_outmaster<int> out1;
sc_in<bool> start; // kick-start
void generate_data ()
{
for(int i =0; i <10; i++) {
out1 =i ; //to invoke slave;}
}
SC_CTOR(producer)
{
SC_METHOD(generate_data);
sensitive << start;}};
SC_MODULE(consumer)
{
sc_inslave<int> in1;
int sum; // state variable
void accumulate (){
sum += in1;
cout << "Sum = " << sum << endl;}
```

```
SC_CTOR(consumer)
{
SC_SLAVE(accumulate, in1);
sum = 0; // initialize
};
SC_MODULE(top) // container
{
producer *A1;
consumer *B1;
sc_link_mp<int> link1;
SC_CTOR(top)
{
A1 = new producer("A1");
A1.out1(link1);
B1 = new consumer("B1");
B1.in1(link1);}};
```



Model Evolution: Refinement



- Models can be refined continuously until the specification is complete

The Remarkable Thing About Software

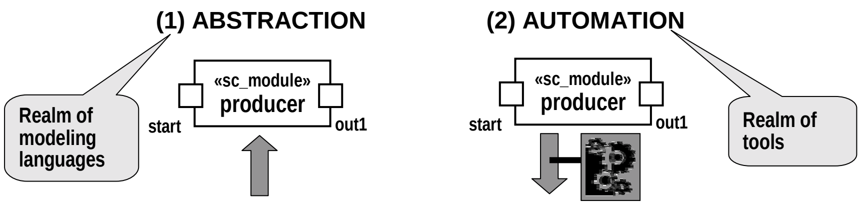
> *Software has the rare property that it allows us to directly evolve models into full-fledged implementations without changing the engineering medium, tools, or methods!*

⇒ This ensures perfect accuracy of software models; since the model and the system that it models are the same thing

The model evolves into the system it was modeling

Model-Driven Style of Development (MDD)

- An approach to software development in which the focus and primary artifacts of development are models (as opposed to programs)
- Based on two time-proven methods

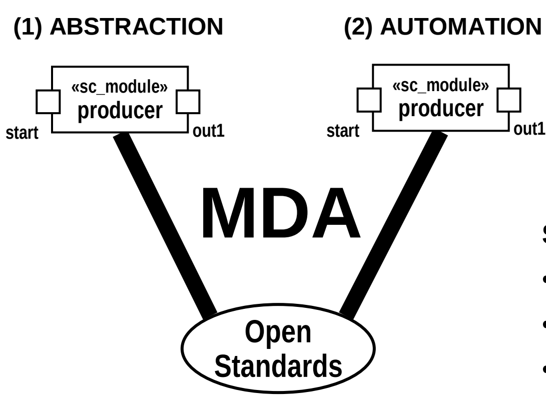



```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " <<
sum << endl;}
```

```
SC_MODULE(producer)
{sc_inslave<int> in1;
int sum; //
void accumulate (){
sum += in1;
cout << "Sum = " <<
sum << endl;}
```


OMG's Model-Driven Architecture (MDA)

- An OMG initiative
  - A framework for a set of <u>*open*</u> standards in support of MDD



Standards for:

•Modeling languages

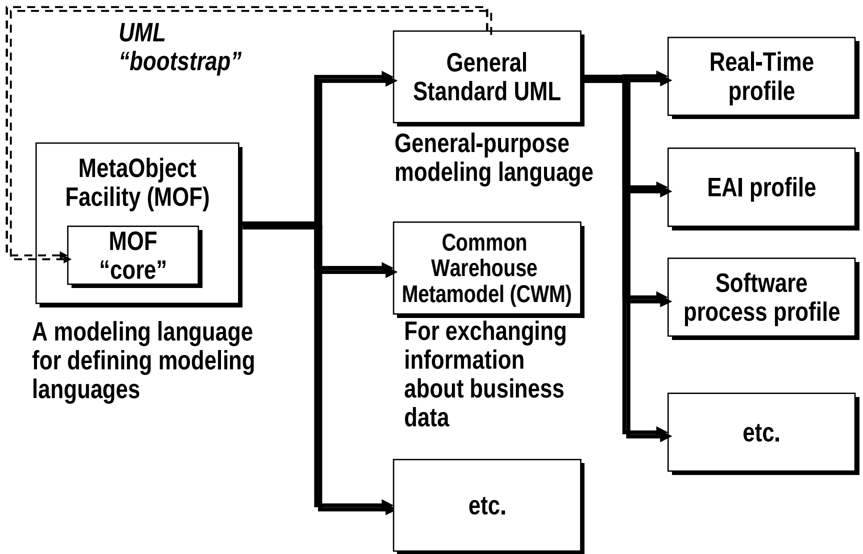
•Model transformations

•Software processes

•Model interchange…

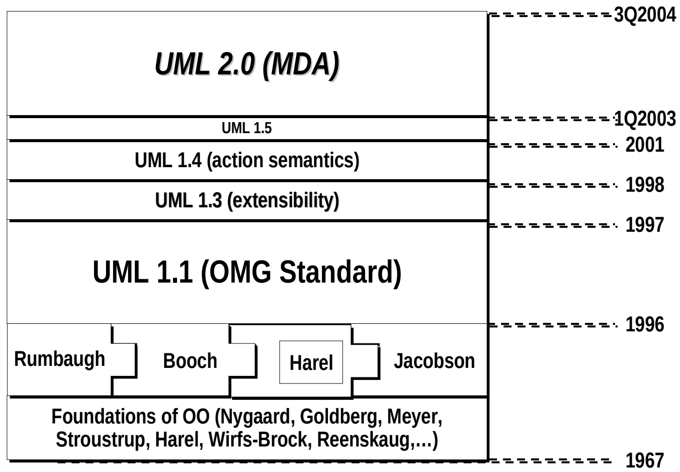
The Languages of MDA

- Set of modeling languages for specific purposes



Part II:
OMG Modeling Language Standard
– UML 2.0

UML: The Foundation of MDA

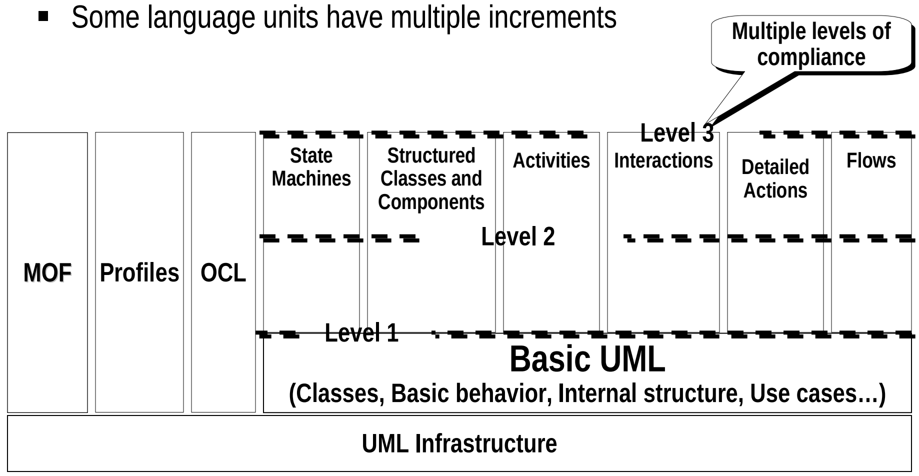


Contents

- **Foundations**
- Actions
- Activities
- Interactions
- Structures
- Summary

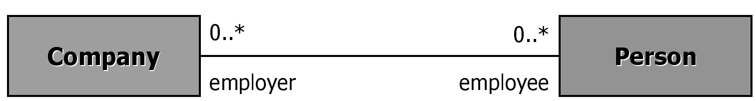
Language Structure

- A core language + a set of optional "language units"
  - Some language units have multiple increments

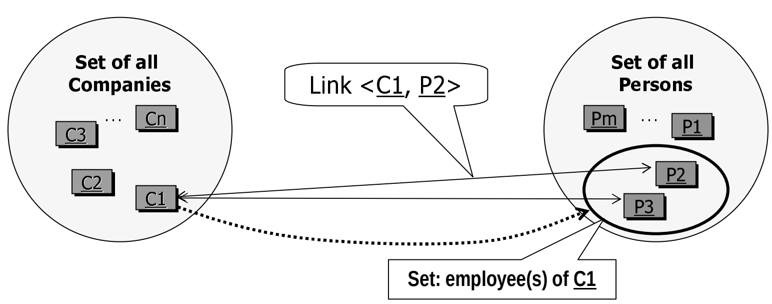


Class Diagram Semantics

- Represent relationships between *instances* of classes

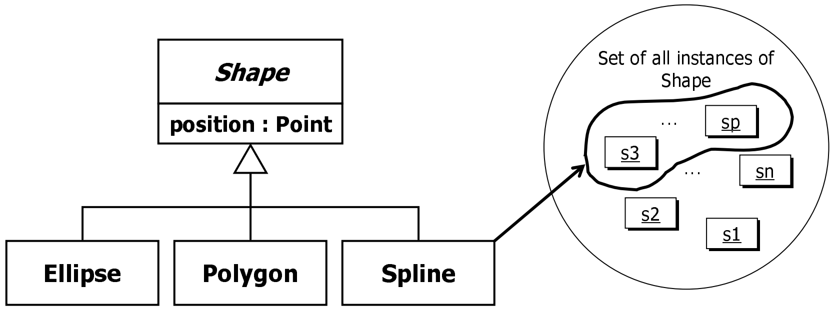


Formal (set theoretic) interpretation:



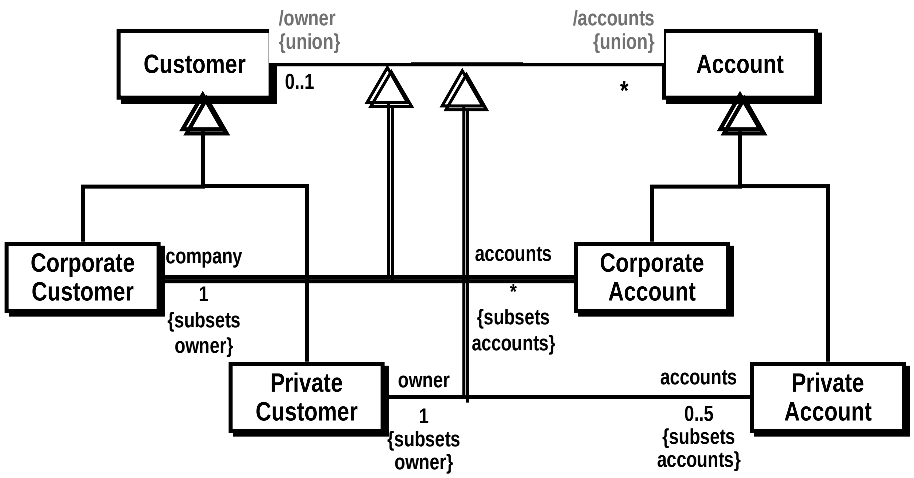
Generalization Semantics

- Subclasses = specialized subsets of parent
- Subclass inherits all features of the parent and may add its own
- Relationship between classes



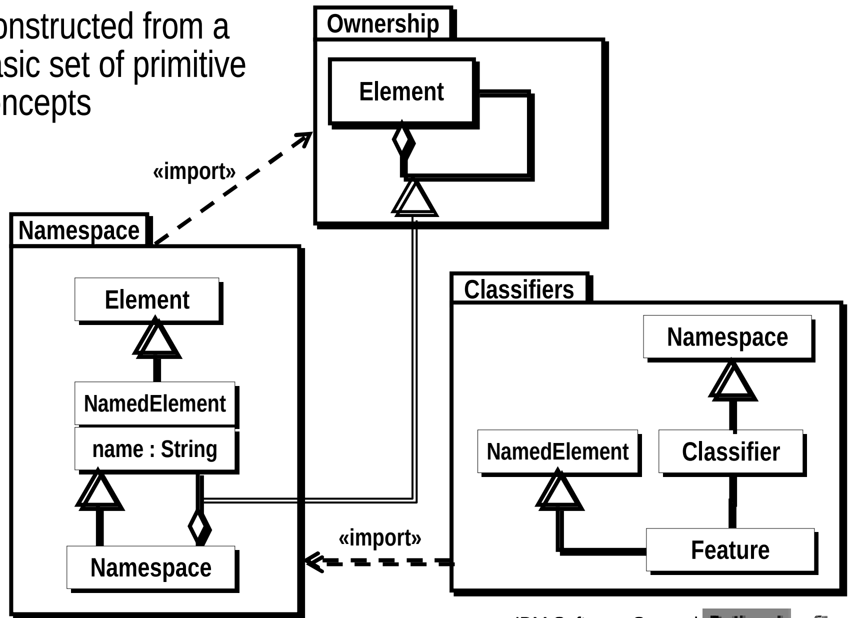
Association Specialization

- Also used widely in the definition of the UML metamodel
  - Avoids covariance problems

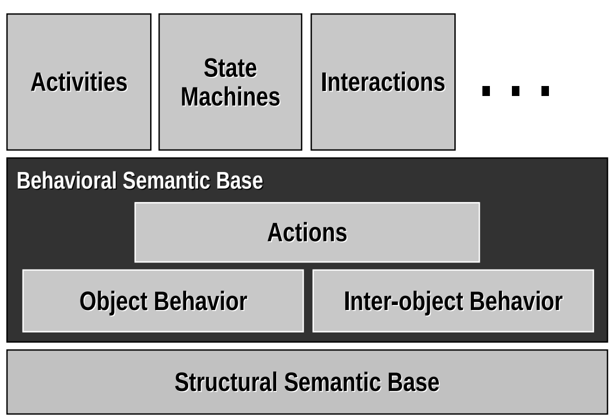


Example: Classifier Definition

- Constructed from a basic set of primitive concepts

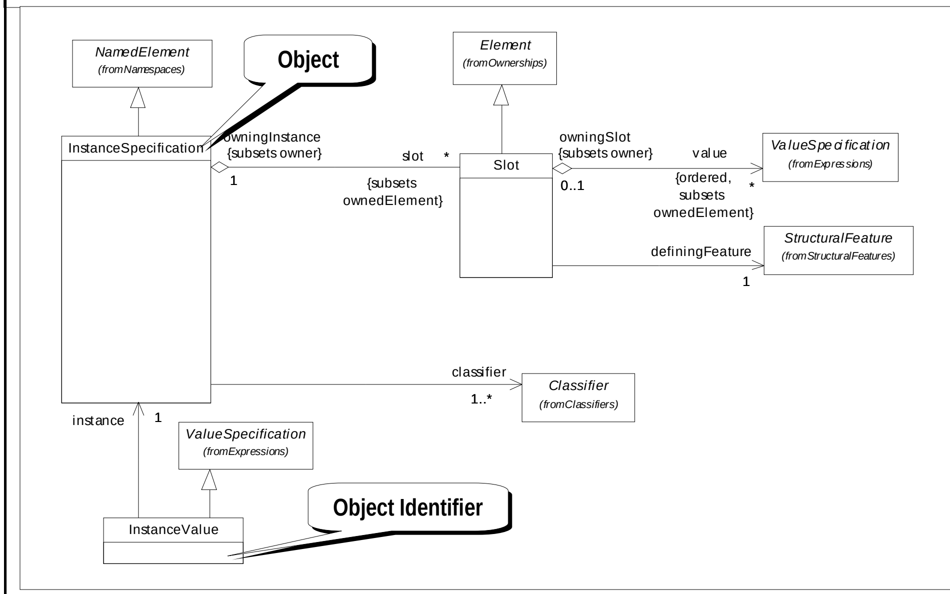


UML 2.0: Run-Time Semantics



Metamodel Description of Objects

IBM



27 IBM Software Group | Rational. software

Basic Structural Elements

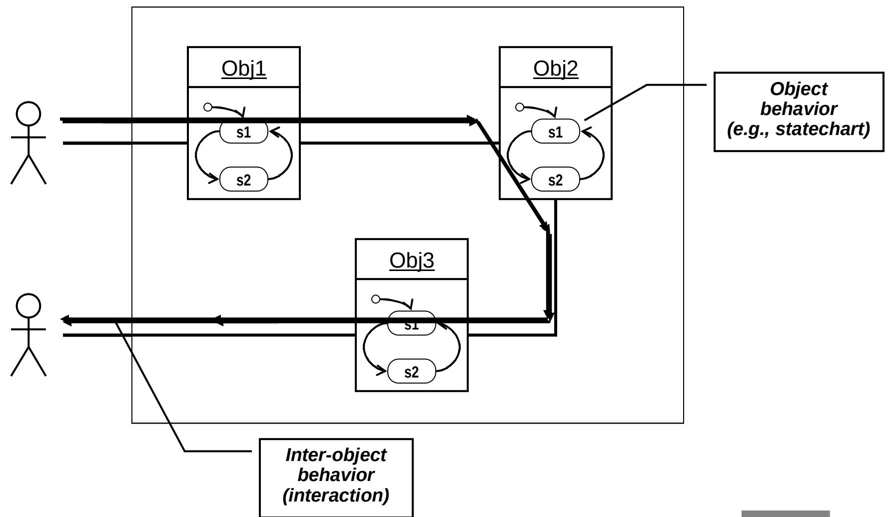
IBM

- Values
  - Universal, unique, constant
  - E.g. Numbers, characters, object identifiers ("instance value")
- "Cells" (Slots/Variables)
  - Container for values or objects
  - Can be created and destroyed dynamically
  - Constrained by a type
  - Have identity (independent of contents)
- Objects (Instances)
  - Containers of slots (corresponding to structural features)
  - Just a special kind of cell
- Links
  - Tuples of object identifiers
  - May have identity (i.e., some links are objects)
  - Can be created and destroyed dynamically

28 IBM Software Group | Rational. software

How Things Happen in UML

- In UML, all behavior results from the actions of (active) objects



How Things Happen in UML

- An action is executed by an object
  - May change the contents of one or more variables or slots
  - If it is a communication (“messaging”) action, it may:
    - Invoke an operation on another object
    - Send a signal to another object
    - Either one will eventually cause the execution of a procedure on the target object…
    - …which will cause other actions to be executed, etc.
  - Successor actions are executed
    - Determined either by control flow or data flow

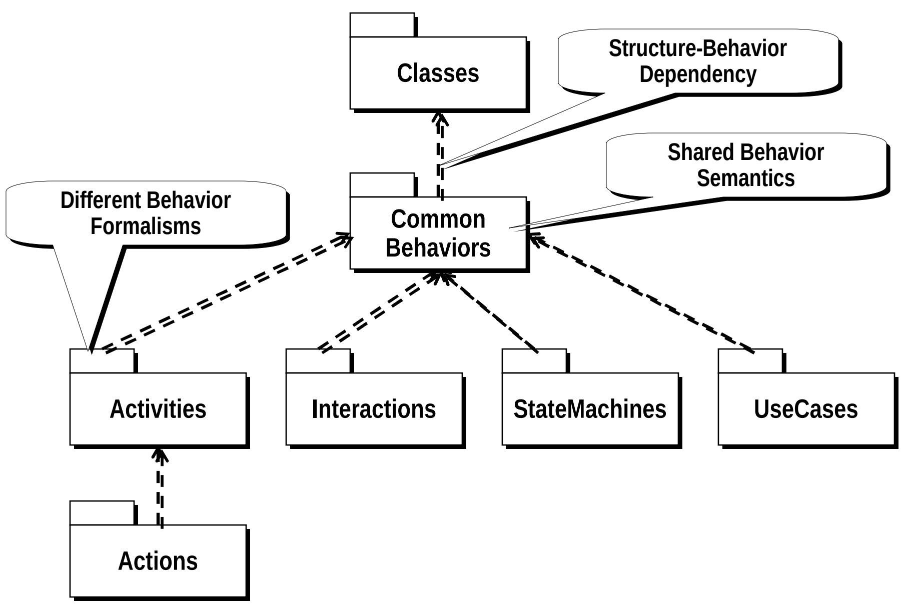
Active Object Definition

- From the spec:

An active object is an object that, as a direct consequence of its creation, [eventually] commences to execute its classifier behavior [specification], and does not cease until either the complete behavior is executed or the object is terminated by some external object.

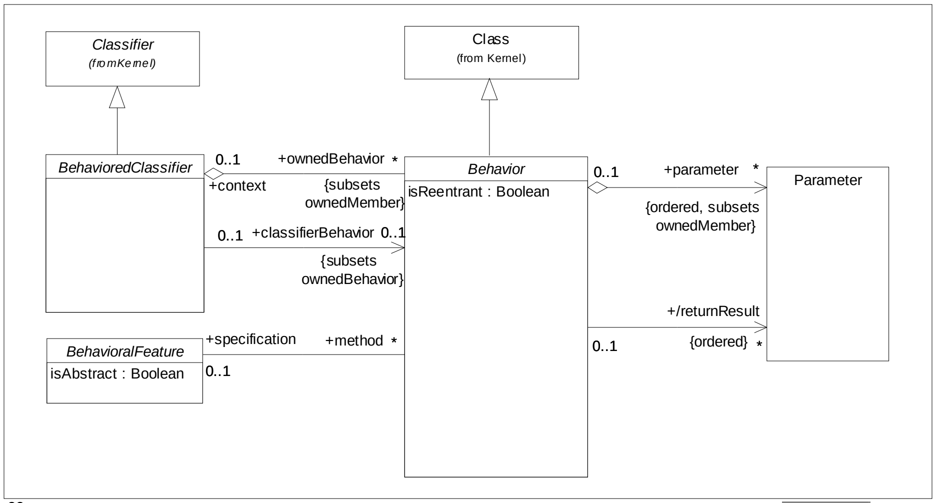
The points at which an active object responds to [messages received] from other objects is determined solely by the behavior specification of the active object...

Metamodel Structure



Common Behavior Metamodel

- The "classifier behavior" of a composite classifier is distinct from the behavior of its parts (i.e., it is NOT a resultant behavior)



Contents

- Foundations
- **Actions**
- Activities
- Interactions
- Structures
- Summary

Actions in UML

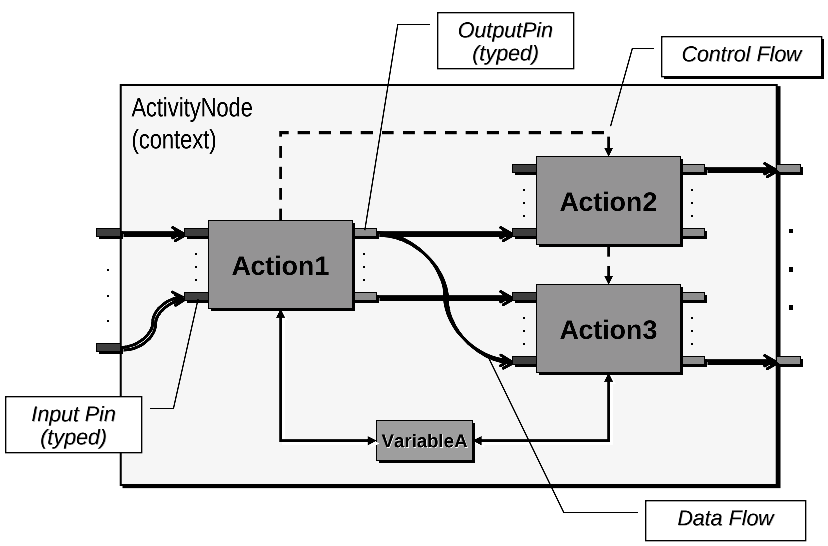
- Action = fundamental unit of behavior
  - for modeling fine-grained behavior
  - Level of traditional programming languages
- UML defines:
  - A set of action types
  - A <u>semantics</u> for those actions
    - i.e. what happens when the actions are executed
    - Pre- and post-condition specifications (using OCL)
  - No concrete syntax for individual kinds of actions (notation)
    - Flexibility: can be realized using different concrete languages
- In UML 2, the metamodel of actions was consolidated
  - Shared semantics between actions and activities (Basic Actions)

35

IBM Software Group | Rational. software

Object Behavior Basics

- Support for multiple computational paradigms



36

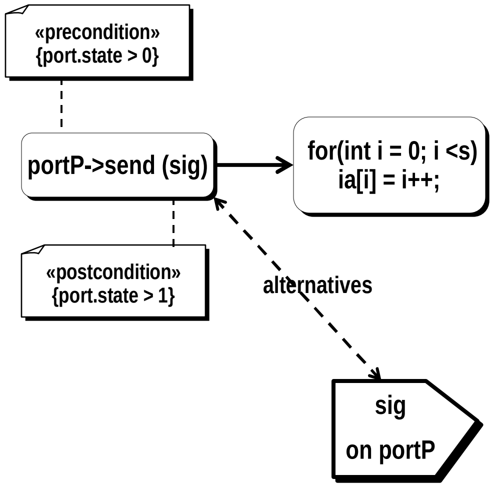
IBM Software Group | Rational. software

Categories of Actions

- Communication actions (send, call, receive,…)
- Primitive function action
- Object actions (create, destroy, reclassify,start,…)
- Structural feature actions (read, write, clear,…)
- Link actions (create, destroy, read, write,…)
- Variable actions (read, write, clear,…)
- Exception action (raise)

General Notation for Actions

- No specific symbols (some exceptions)

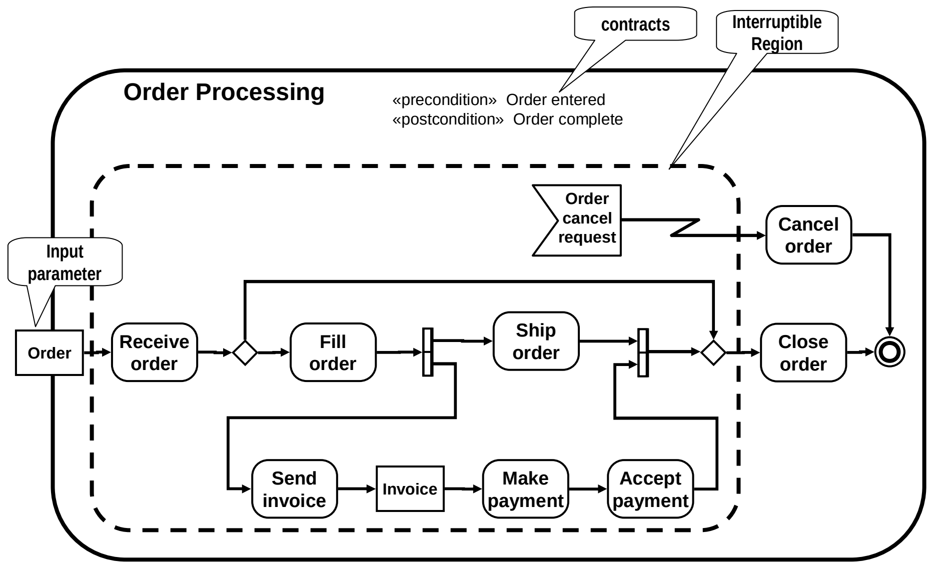


Contents

Activities

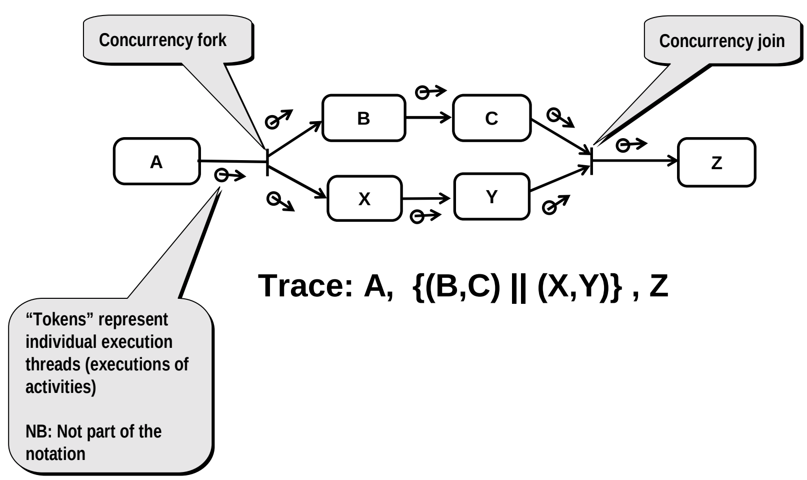
- Significantly enriched in UML 2.0 (relative to UML 1.x activities)
  - More flexible semantics for greater modeling power (e.g., rich concurrency model based on Petri Nets)
  - Many new features
- Major influences for UML 2.0 activity semantics
  - Business Process Execution Language for Web Services (BPEL4WS) – a de facto standard supported by key industry players (Microsoft, IBM, etc.)
  - Functional modeling from the systems engineering community (INCOSE)

Activity Graph Example



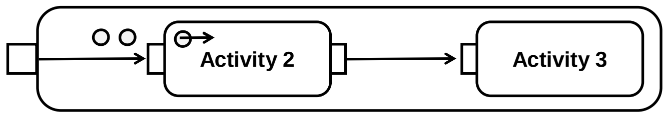
Extended Concurrency Model

- Fully independent concurrent streams (“tokens”)



Activities: Token Queuing Capabilities

- Tokens can
  - queue up in “in/out” pins.
  - backup in network.
  - prevent upstream behaviors from taking new inputs.



- …or, they can flow through continuously
  - taken as input while behavior is executing
  - given as output while behavior is executing
  - identified by a {stream} adornment on a pin or object node

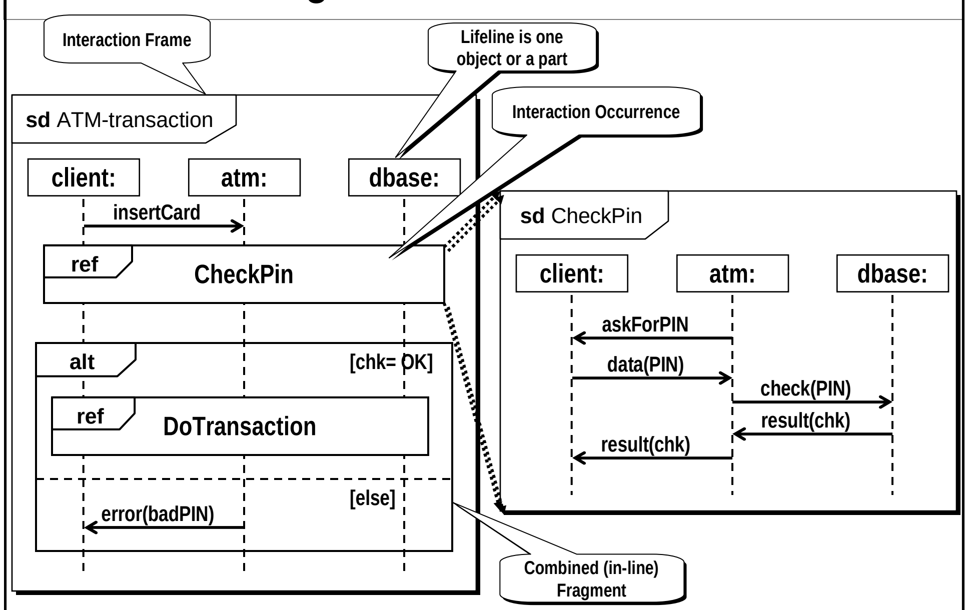
Contents

- Foundations
- Actions
- Activities
- **Interactions**
- Structures
- Summary

Overview of New Features

- Interactions focus on the communications between collaborating instances communicating via messages
  - Both synchronous (operation invocation) and asynchronous (signal sending) models supported
- Multiple concrete notational forms:
  - sequence diagram (based on ITU Standard Z.120 – MSC-2000)
  - communication diagram
  - interaction overview diagram
  - timing diagram
  - interaction table

Interaction Diagrams



Combined Fragment Types (1 of 2)

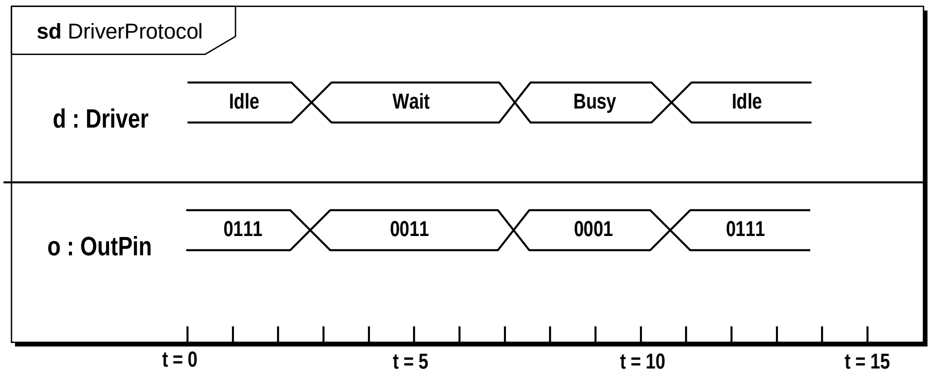
- Alternatives (**alt**)
  - choice of behaviors – at most one will execute
  - depends on the value of the guard (“else” guard supported)
- Option (**opt**)
  - Special case of alternative
- Break (**break**)
  - Represents an alternative that is executed instead of the remainder of the fragment (like a break in a loop)
- Parallel (**par**)
  - Concurrent (interleaved) sub-scenarios
- Negative (**neg**)
  - Identifies sequences that must not occur

Combined Fragment Types (2 of 2)

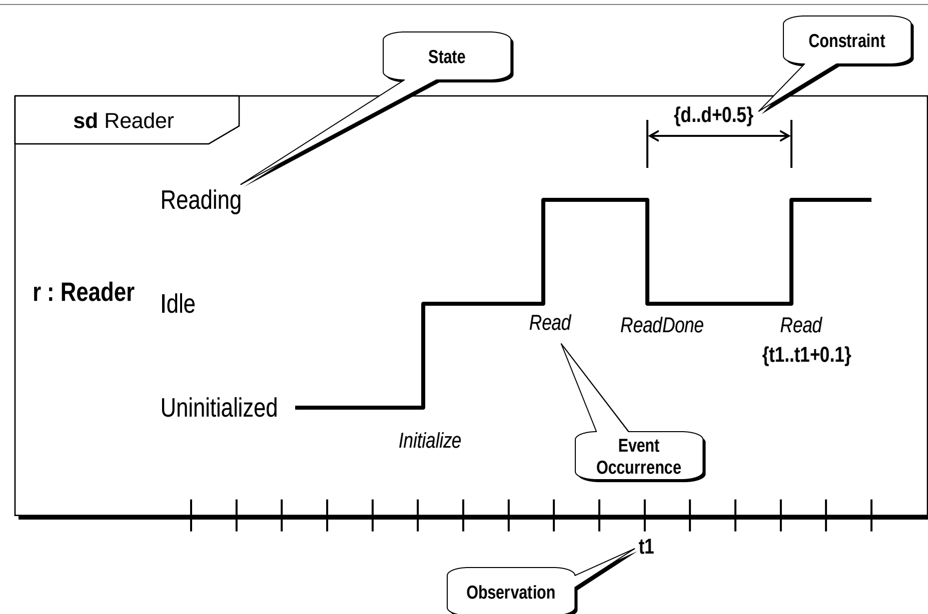
- Critical Region (**region**)
  - Traces cannot be interleaved with events on any of the participating lifelines
- Assertion (**assert**)
  - Only valid continuation
- Loop (**loop**)
  - Optional guard: [<min>, <max>, <Boolean-expression>]
  - No guard means no specified limit

Timing Diagrams

- Can be used to specify time-dependent interactions
  - Based on a simplified model of time (use standard "real-time" profile for more complex models of time)



Timing Diagrams (cont.)



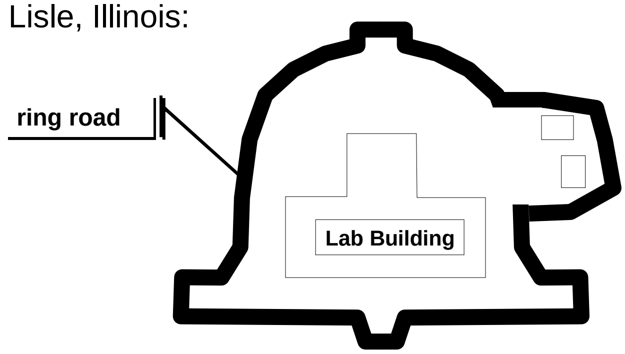
Contents

51

IBM Software Group | Rational. software

A Parable

In Lisle, Illinois:



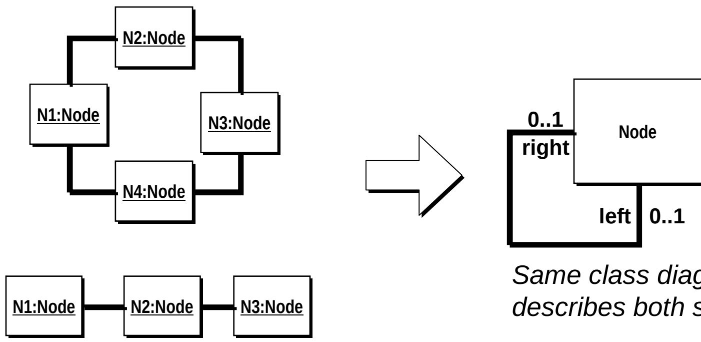
- “Architectural decay”, caused by:
  - Lack of high-level (architectural) view of the code
  - Difficulties in formally enforcing architectural decisions

52

IBM Software Group | Rational. software

Aren’t Class Diagrams Sufficient?

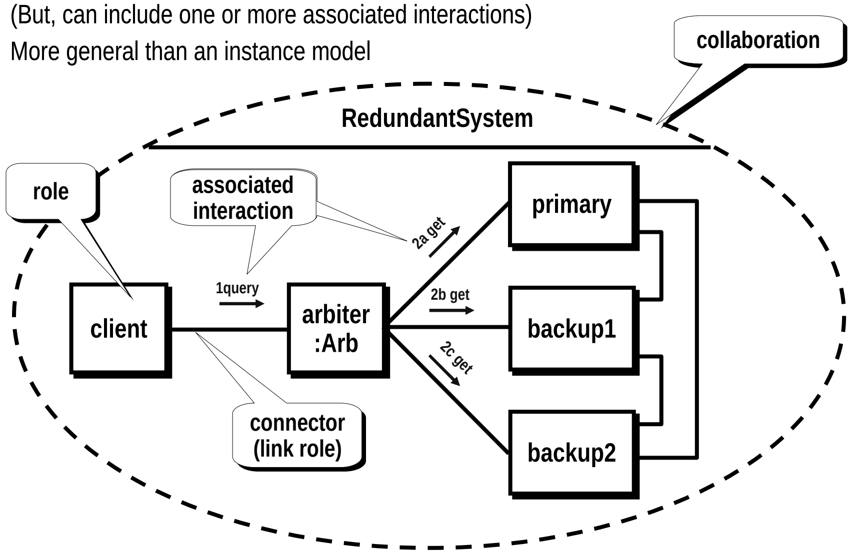
- No!
  - Because they abstract out certain specifics, class diagrams are not suitable for performance analysis
- Need to model structure at the instance level



Same class diagram describes both systems!

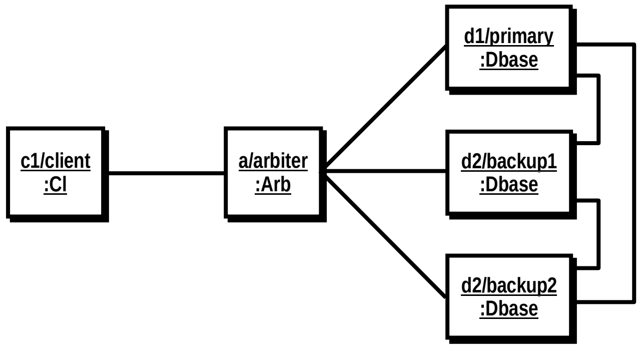
Collaborations

- In UML 2.0 a collaboration is a *purely structural concept*
  - (But, can include one or more associated interactions)
  - More general than an instance model

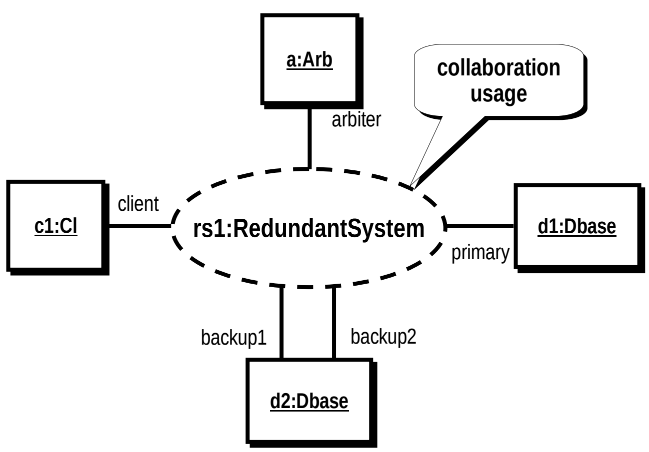


Roles and Instances

- Specific object instances playing specific the roles in a collaboration

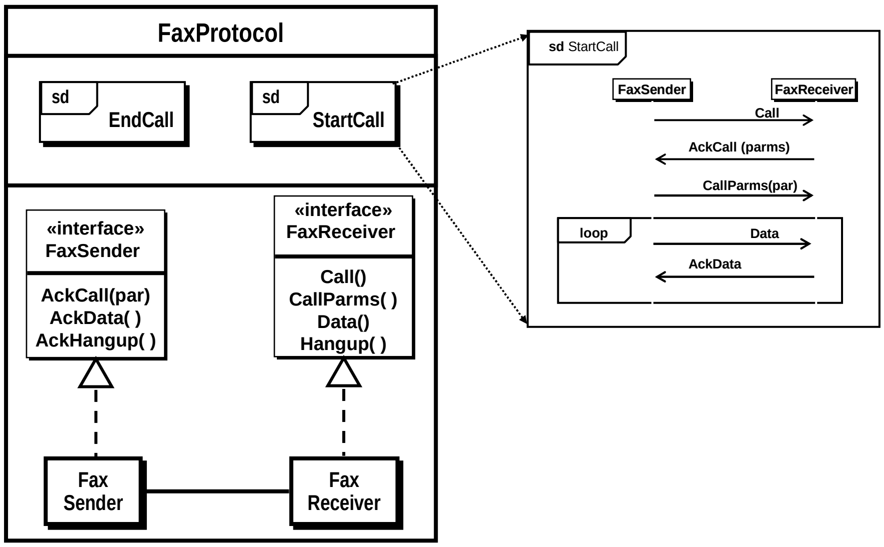


Alternative Notation



Collaboration Protocols

- For dynamic relationships between interfaces

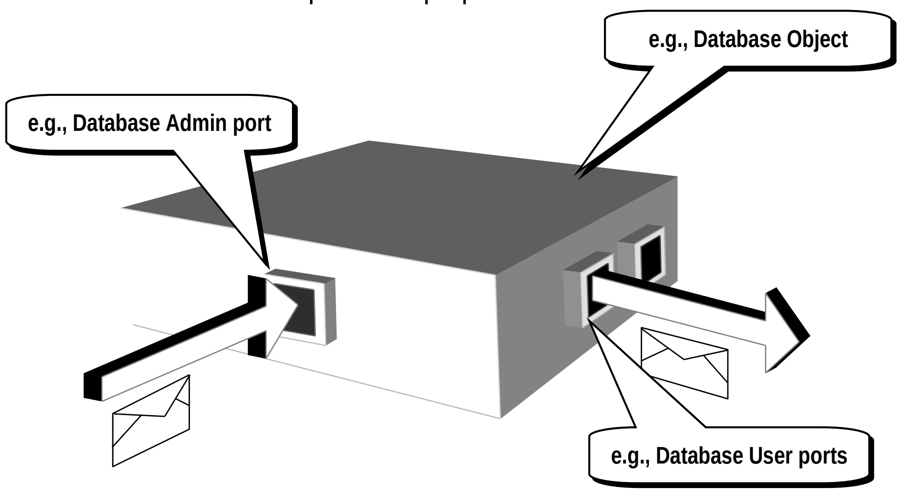


Structured Classes

- Classes with
  - Internal (collaboration) structure
  - Ports (optional)
- Primarily intended for architectural modeling
- Heritage: architectural description languages (ADLs)
  - UML-RT profile: Selic and Rumbaugh (1998)
  - ACME: Garlan et al.
  - SDL (ITU-T standard Z.100)

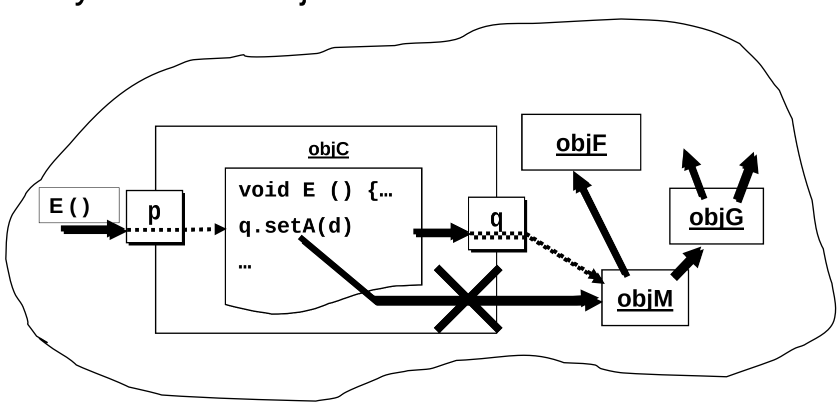
Structured Objects: Ports

- Multiple points of interaction
  - Each dedicated to a particular purpose



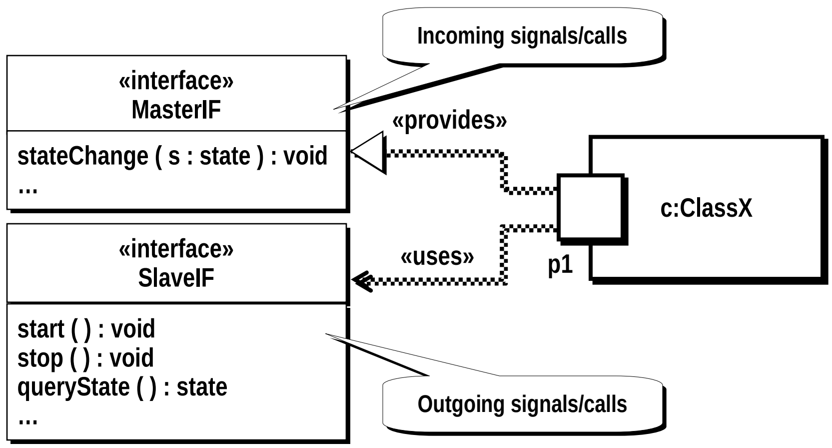
New Feature: Ports

- Used to distinguish between multiple collaborators
  - Based on port through which interaction is occurring
- Fully isolate an object's internals from its environment



Port Semantics

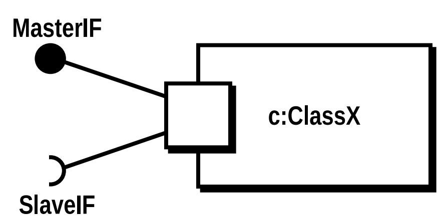
- A port can support multiple interface specifications
  - Provided interfaces (what the object can do)
  - Required interfaces (what the object needs to do its job)



61 IBM Software Group | Rational. software

Ports: Alternative Notation

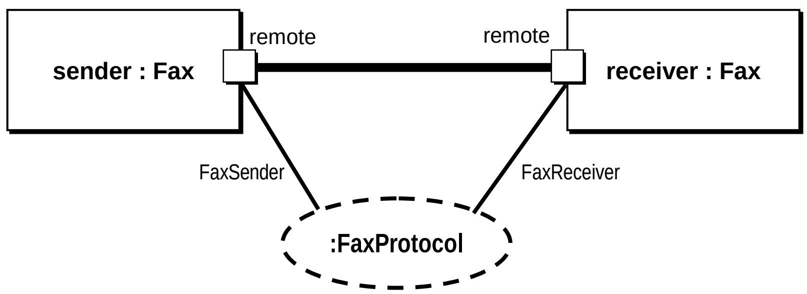
- Shorthand "lollipop" notation with 1.x backward compatibility



62 IBM Software Group | Rational. software

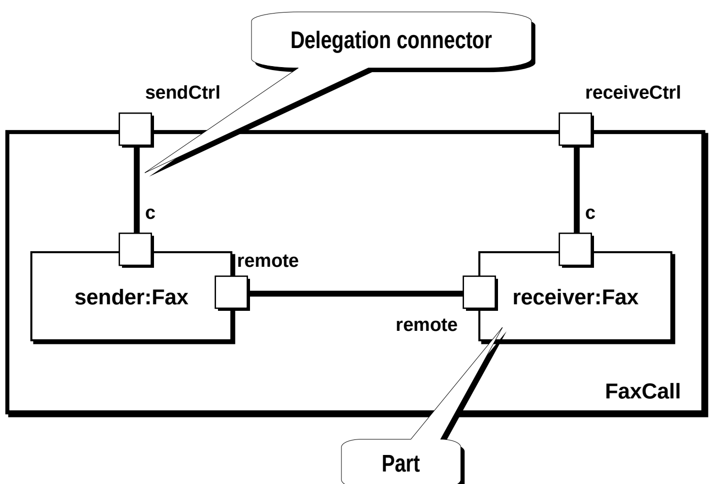
Assembling Structured Objects

- Ports can be joined by connectors
- These connections can be constrained to a protocol
  - Static checks for dynamic type violations are possible
  - Eliminates "integration" (architectural) errors



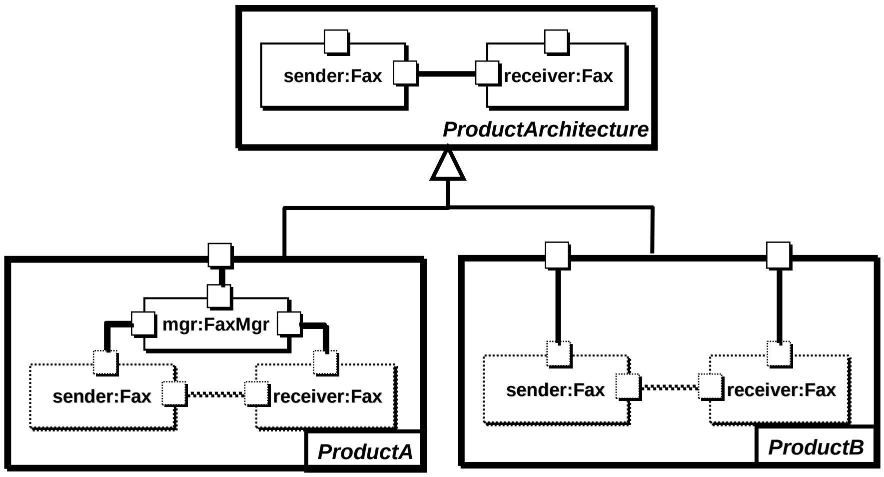
Structured Classes: Internal Structure

- Structured classes may have an internal structure of (structured class) parts and connectors



Structure Refinement Through Inheritance

- Using standard inheritance mechanism (design by difference)



Summary: UML 2.0

- First major revision of UML
- Original standard had to be adjusted to deal with
  - MDD requirements (precision, code generation, executability)
- UML 2.0 characterized by
  - Small number of new features + consolidation of existing ones
  - Scaleable to large software systems (architectural modeling capabilities)
  - Modular structure for easier adoption (core + optional specialized sub-languages)
  - Increased semantic precision and conceptual clarity
  - Suitable foundation for MDA (executable models, full code generation)

Part III:
The Real-Time UML Profile

Contents

A Giant Speaks…

Edsger Wybe Dijkstra (1930 – 2002)

- *"[The interrupt] was a great invention, but also a Pandora's Box. ….essentially, for the sake of efficiency, concurrency [became] visible… and then, all hell broke loose"* (EWD 1303)
- *"I see no meaningful difference between programming methodology and mathematical methodology"* (EWD 1209)

69 IBM Software Group | Rational. software

"Because [programs] are put together in the context of a set of information requirements, they observe no natural limits other than those imposed by those requirements. Unlike the world of engineering, there are no immutable laws to violate."

\- Wei-Lung Wang
Comm. of the ACM (45, 5)
May 2002

"All machinery is derived from nature, and is founded on the teaching and instruction of the revolution of the firmament."

\- Vitruvius
On Architecture, Book X
1st Century BC

70 IBM Software Group | Rational. software

What is Engineering?

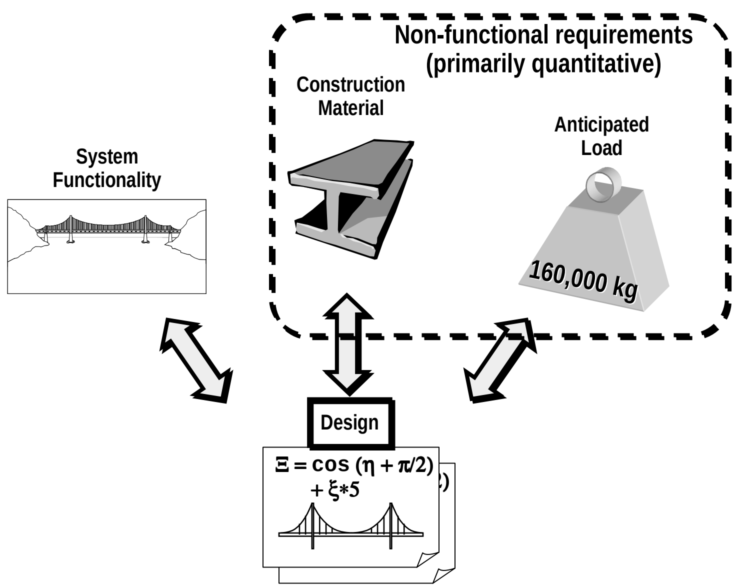
- *Merriam-Webster Collegiate Dictionary:*

  engineering: the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people

- What does this have to do with software design?
  - *"…no natural limits…no immutable laws to violate"*

71 IBM Software Group | Rational. software

The Classical Engineering Design Problem

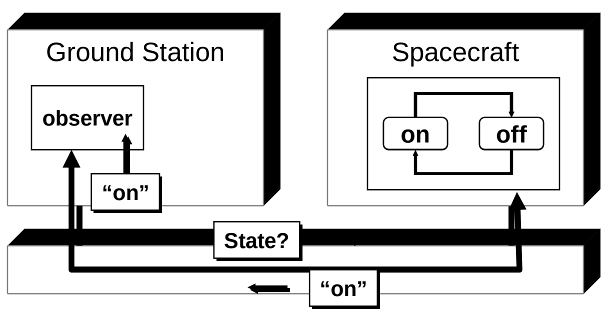


72 IBM Software Group | Rational. software

What is Software Made of?

The Case of Distributed Systems

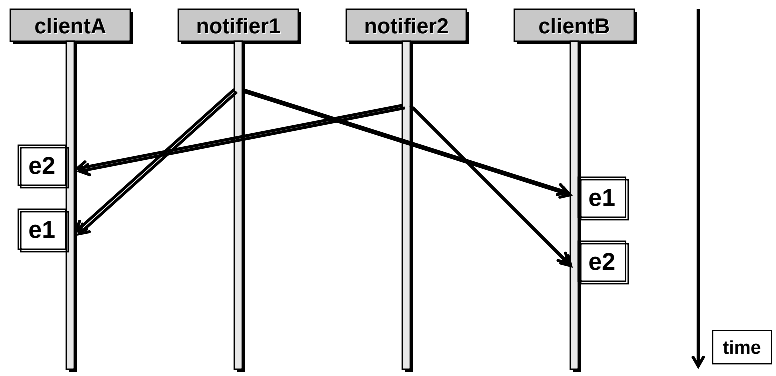
- Possibility of out of date status information due to transmission delays



- Quantity can change quality…

The Effect of Communication Media

- Inconsistent views of system state:
  - different observers see different event orderings



- Can we not hide this by adding “fault transparency” layers?

Impossibility Result

It is not possible to guarantee that agreement can be reached in finite time over an asynchronous communication medium, if the medium is lossy or one of the distributed sites can fail

- Fischer, M., N. Lynch, and M. Paterson, “Impossibility of Distributed Consensus with One Faulty Process” *Journal of the ACM*, (32, 2) April 1985.

Impossibility Result No.2

Even when communication is fully reliable, it is not possible to guarantee common knowledge if communication delays are unbounded

- Halpern, J.Y, and Moses, Y., “Knowledge and common knowledge in a distributed environment” *Journal of the ACM*, (37, 3) 1990.

- *In distributed situations, all failure transparency mechanisms require distributed agreement*
- *Conclusion: No matter how hard we try to paper this over with “transparency” layers, we cannot hide all failures*

77

IBM Software Group | Rational. software

The “End-To-End” Argument

- *The end-to-end argument* [Saltzer et al.]:
  - if transparency cannot be guaranteed to a sufficient degree, the application can never be fully protected from the effects of distribution

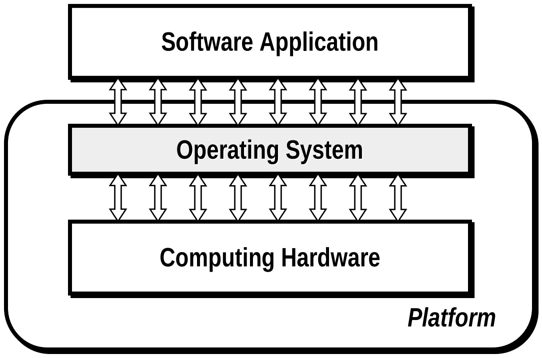
  ⇒ *the overhead of using transparency mechanisms may not always be justified by the benefits obtained*

  ⇒ *The choice is an engineering decision based primarily on a quantitative analysis of the physical properties of the environment in which the software is running*

78

IBM Software Group | Rational. software

What Software is Made of



- The raw material of software is its hardware/software platform
  - The platform always bottoms out in hardware
  - Software inherits the physical limitations (speed, reliability, capacity, etc.) of its platform

79 IBM Software Group | Rational. software

"Physical Programming"

- Computer System = Software + Computer Platform
- The (physical) limitations of the platform must be a first-order concern software design
  - Even for many applications that are not deemed "real time"
  - More and more critical applications will have stringent requirements on availability and responsiveness
- The bad news:
  Murphy's Law: the physical world is inherently complex

80 IBM Software Group | Rational. software

Contents

Requirements for a Real-Time UML

- "*UML profile for scheduling performance and time*"
  - Adopted as an official OMG standard (ptc/2004-02-01)
- Defines standard methods for using UML to model:
  - Physical time
  - Timing specifications
  - Timing services and mechanisms
  - Modeling resources (logical and physical)
  - Concurrency and scheduling
  - Software and hardware infrastructure and their mapping
  - ..including specific notations for the above

RT Profile: Design Principles

- Ability to specify quantitative information directly in UML models
  - key to quantitative analysis and predictive modeling
- Flexibility:
  - users can model their RT systems using modeling approaches and styles of their own choosing
  - open to existing and new analysis techniques
- Facilitate the use of (quantitative) analysis methods
  - eliminate the need for a deep understanding of analysis methods
  - as much as possible, automate the generation of analysis models and the analysis process itself

83 IBM Software Group | Rational. software

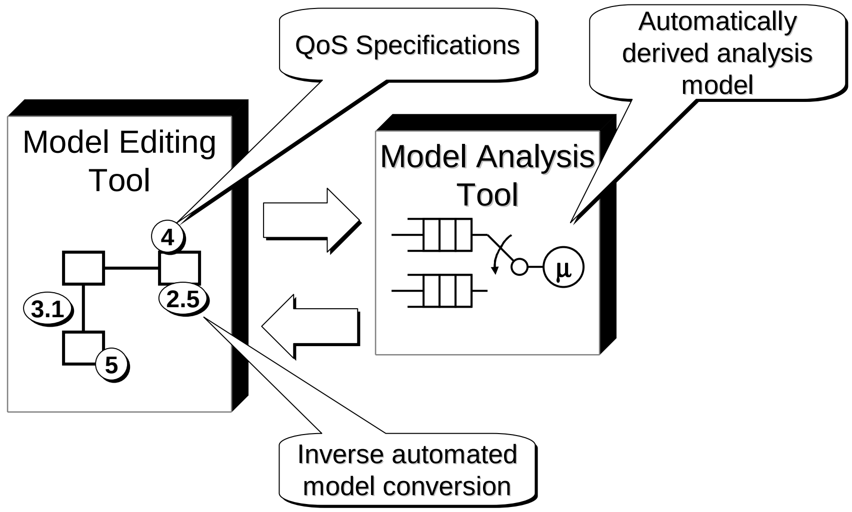
Main Quantitative Methods for RT Systems

- Schedulability analysis
  - *will the system meet all of its deadlines?*
- Performance analysis
  - *what kind of response will the system have under load?*

84 IBM Software Group | Rational. software

Automating Analysis

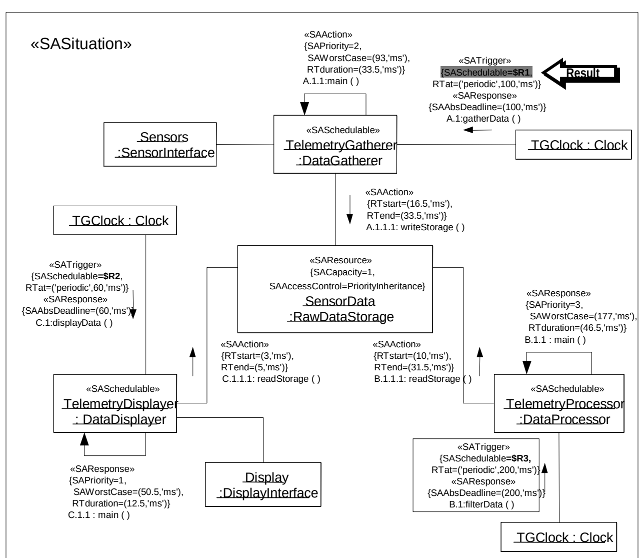
- Inter-working of specialized tools via shared standards



85

IBM Software Group | Rational. software

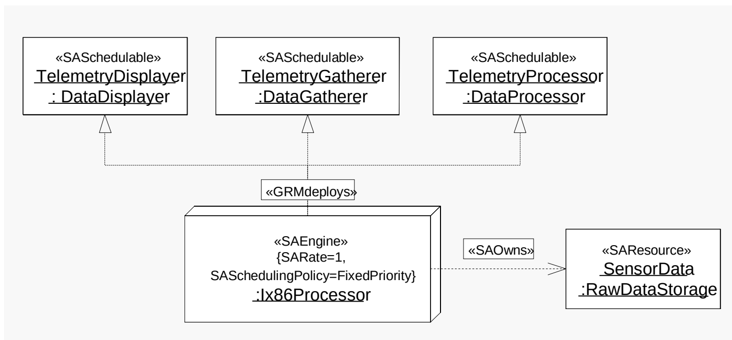
Example: Schedulability Annotations



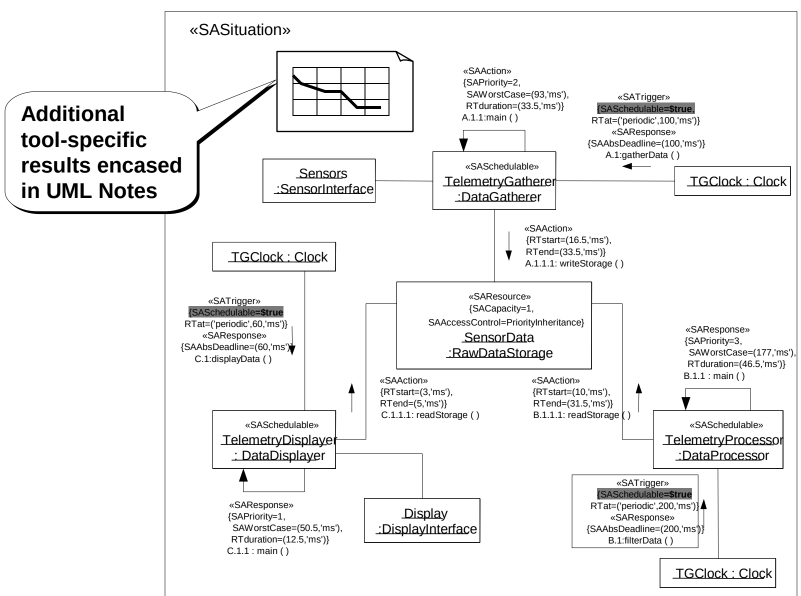
86

IBM Software Group | Rational. software

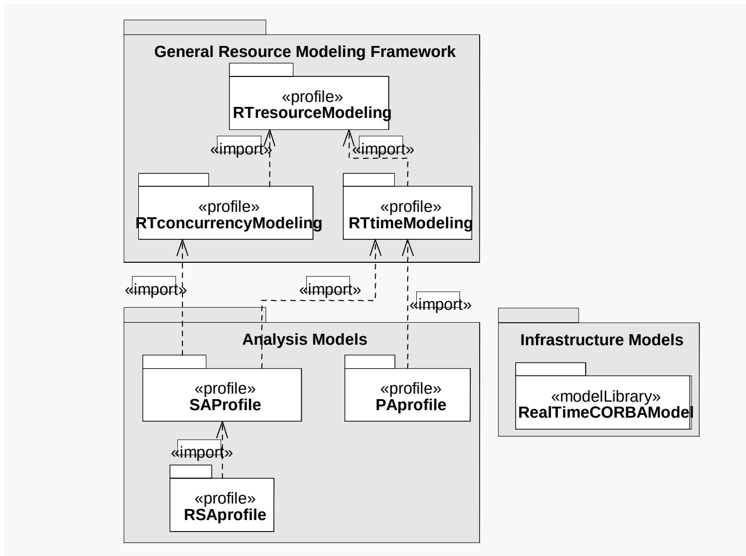
Example: Deployment Specification



Example: Analysis Results

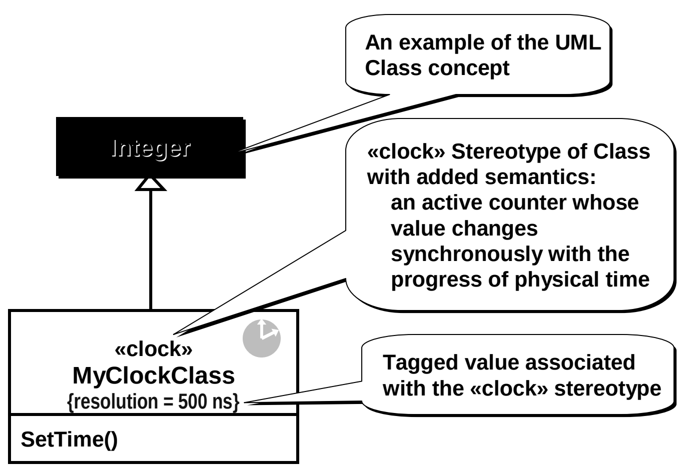


UML Real-Time Profile Structure

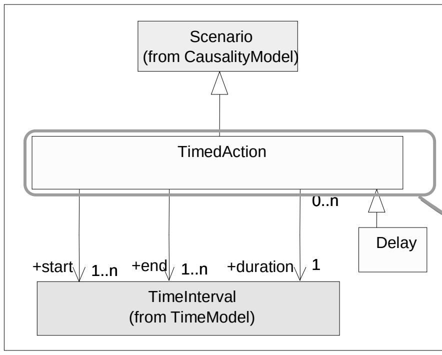


Specializing UML: Stereotypes

- We can add semantics to any standard UML concept
  - …but, must not violate standard UML semantics



Format: Domain Model and Extensions



Stereotype	Base Class	Tags
«RTaction»	Action	RTstart RTend RTduration
	ActionExecution	
	Message	
	Stimulus	
	Method	
	ActionSequence	
	ActionState	
	SubactivityState	
	Transition	
	State	

Contents

1. On software physics
2. An introduction to the standard real-time UML profile
3. **Modeling resources and quality of service**
4. Modeling time in UML
5. Modeling platforms in UML
6. Engineering-oriented MDD

Quality of Service

- The physical characteristics of software can be specified using the general notion of *Quality of Service (QoS):*

  a specification of how well a service can or should be performed

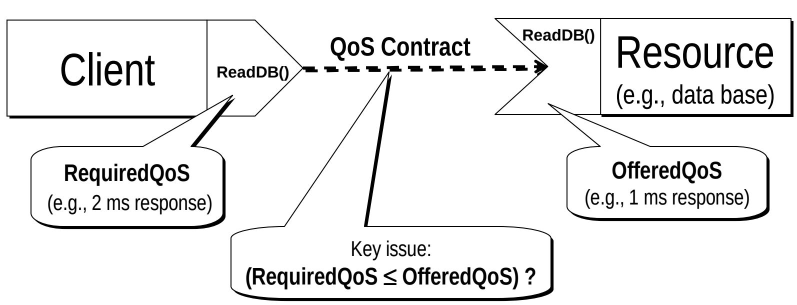
  - throughput, latency, capacity, response time, availability, security...
  - usually a quantitative measure
- QoS concerns have two sides:
  - *offered QoS:* the QoS that is available (supply)
  - *required QoS:* the QoS that is required to do a job (demand)

93 IBM Software Group | Rational. software

Resources and QoS Contracts

- *Resource*:

  an element whose ability or capacity is limited, directly or indirectly, by the finite capacities of the underlying physical platform
- The relationship between resources and resource users



94 IBM Software Group | Rational. software

Verifying QoS Contracts

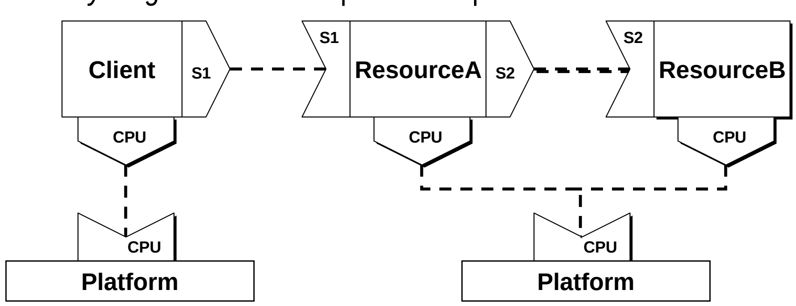
- Can QoS contracts be statically checked by a compiler?
  - The good news: Yes (in most cases)
  - The bad news: it can be hard
- Some issues:
  - In most cases QoS verification cannot be done incrementally – the full system context is required
  - Each type of QoS (e.g., bandwidth, CPU performance) combines differently – no general theory for QoS analysis
- Fortunately, much of this can be automated

95

IBM Software Group | Rational. software

Offered vs. Required QoS

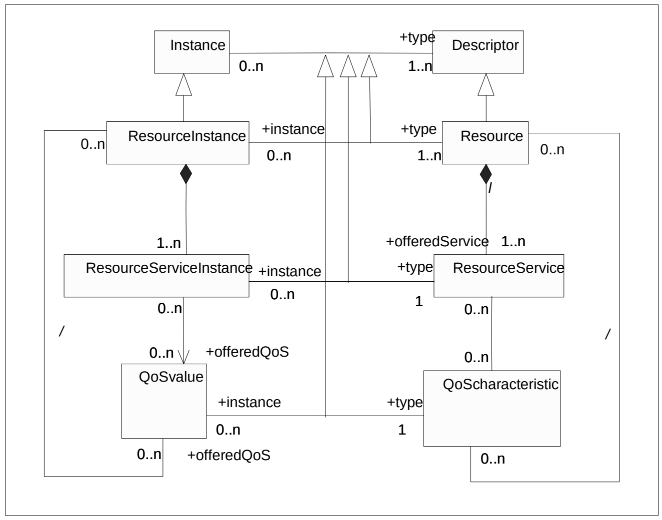
- Like all guarantees, the offered QoS is *conditional* on the resource itself getting what it needs to do its job
- This extends in two dimensions:
  - the *peer* dimension
  - the *layering* dimension: for platform dependencies



96

IBM Software Group | Rational. software

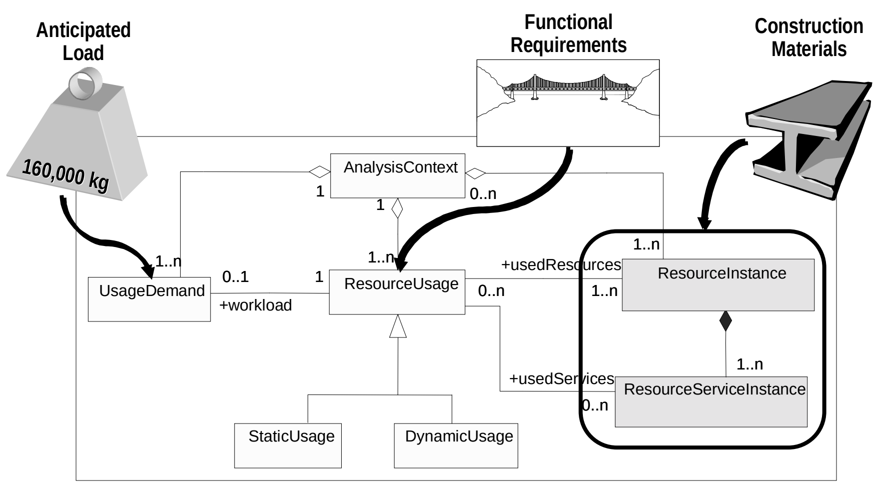
Core Resource Model (Domain Model)



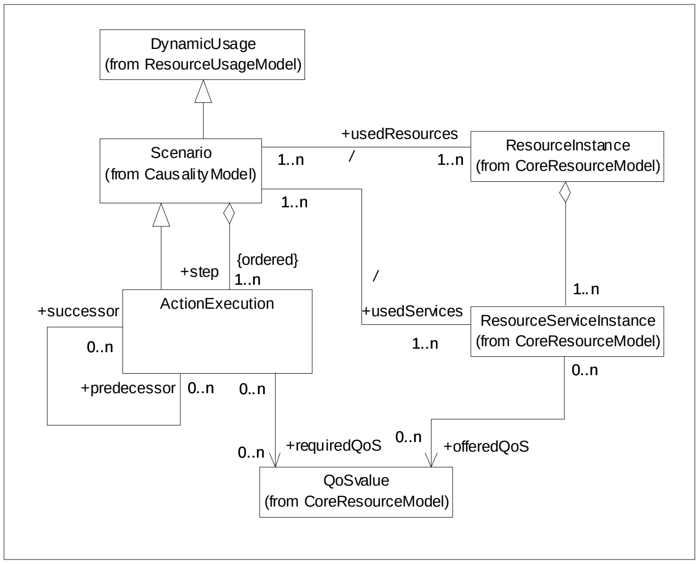
- *NB: This is just a model of the concepts! (domain model)*

Basic Resource Usage Model

- Models a particular situation that needs to be analyzed for some time-related property (e.g., response time)



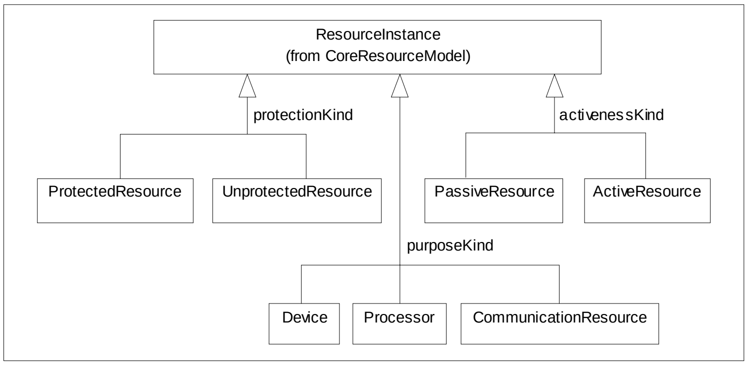
Dynamic Usage Model



99

IBM Software Group | Rational. software

Resource Categorizations



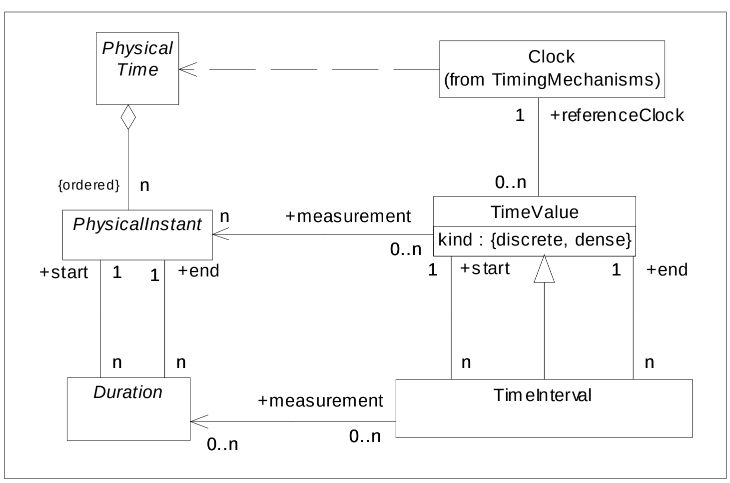
100

IBM Software Group | Rational. software

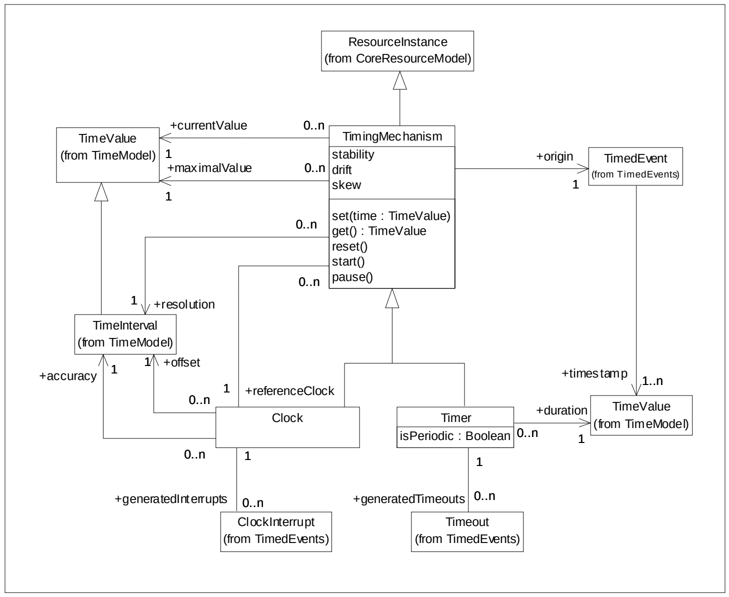
Contents

1. On software physics
2. An introduction to the standard real-time UML profile
3. Modeling resources and quality of service
4. Modeling time in UML
5. Modeling platforms in UML
6. Engineering-oriented MDD

Physical and Measured Time



Timing Mechanisms Model

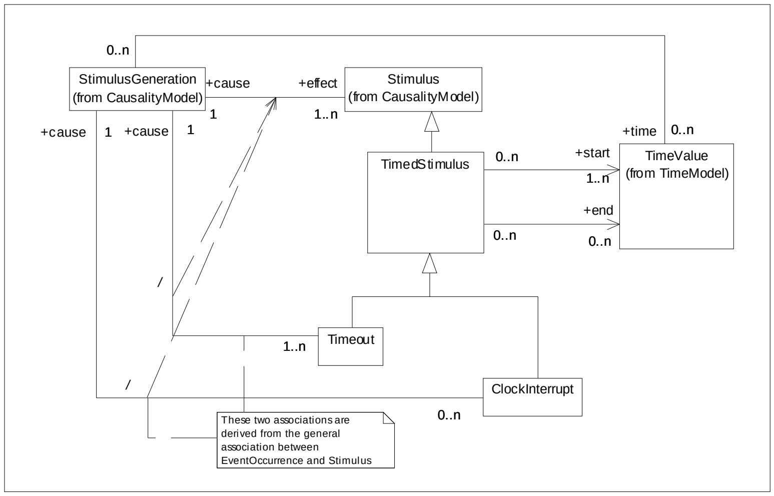


Example Timing Stereotype

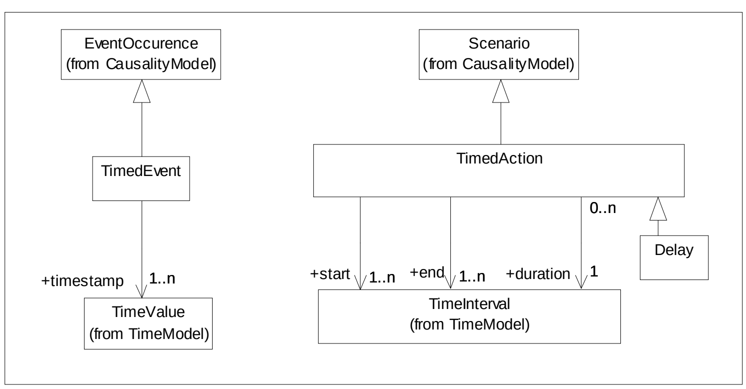
Stereotype	Base Class	Tags
«RTaction»	Action	RTstart RTend RTduration
	ActionExecution	
	Message	
	Stimulus	
	Method	
	ActionSequence	
	ActionState	
	SubactivityState	
	Transition	
	State	

Tag	Tag Type	Multiplicity	Domain Concept
RTstart	RTtimeValue	[0..1]	TimedAction::start
RTend	RTtimeValue	[0..1]	TimedAction::end
RTduration	RTtimeValue	[0..1]	TimedAction::duration

Timed Stimuli

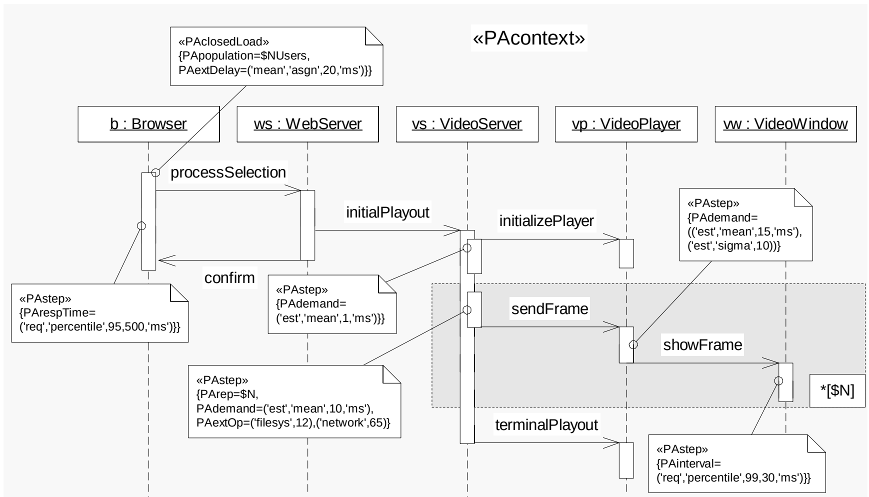


Timed Events and Timed Actions



Time Annotations – Example 1

- In various behavioral diagrams (sequence, activity, etc.)

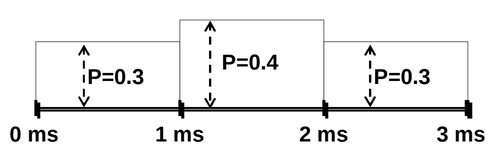


107

IBM Software Group | Rational. software

Specifying Time Values

- Time values can be represented by a special stereotype of Value («RTtimeValue») in different formats; e.g.
  - 12:04 (time of day)
  - 5.3, 'ms' (time interval)
  - 2000/10/27 (date)
  - Wed (day of week)
  - $param, 'ms' (parameterized value)
  - 'poisson', 5.4, 'sec' (time value with a Poisson distribution)
  - 'histogram' 0, 0.3 1, 0.4 2, 0.3, 3, 'ms'



108

IBM Software Group | Rational. software

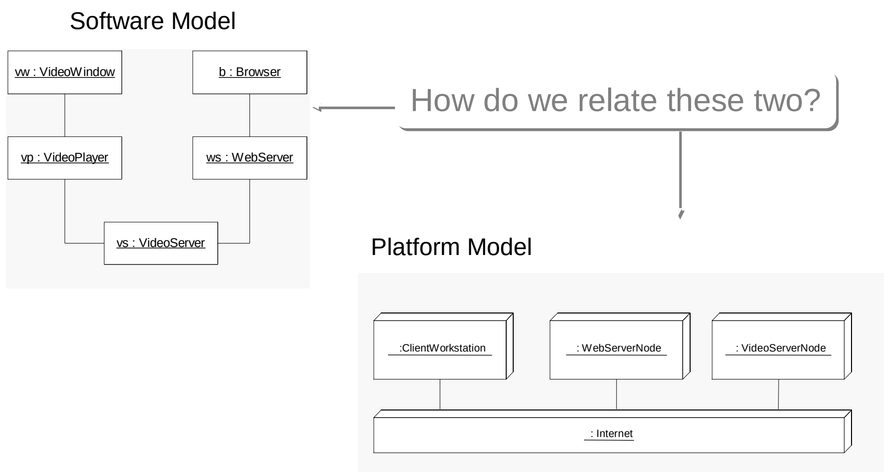
Specifying Arrival Patterns

- Method for specifying standard arrival pattern values
  - Bounded: *‘bounded’, <min-interval>, <max-interval>*
  - Bursty: *‘bursty’, <burst-interval> <max.no.events>*
  - Irregular: *‘irregular’, <interarrival-time>, [<interarrival-time>]**
  - Periodic: *‘periodic’, <period> [, <max-deviation>]*
  - Unbounded: *‘unbounded’, <probability-distribution>*
- Probability distributions supported:
  - Bernoulli, Binomial, Exponential, Gamma, Geometric, Histogram, Normal, Poisson, Uniform

Contents

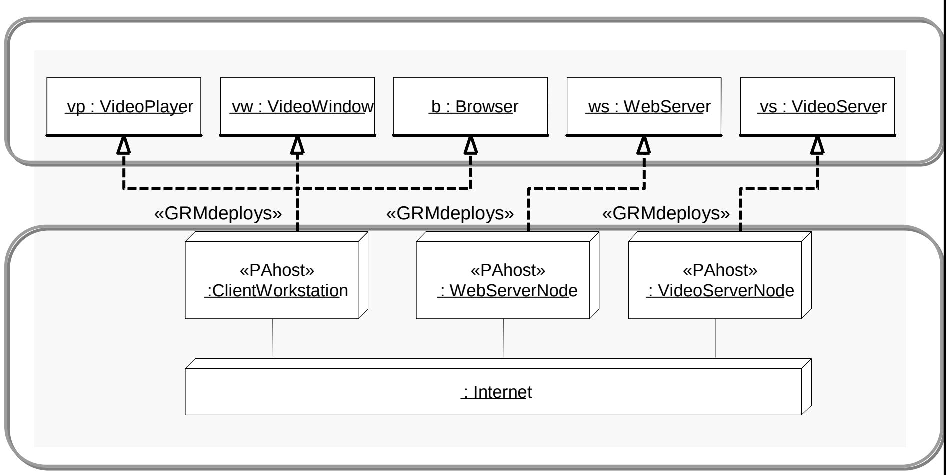
1. On software physics
2. An introduction to the standard real-time UML profile
3. Modeling resources and quality of service
4. Modeling time in UML
5. Modeling platforms in UML
6. Engineering-oriented MDD

Software and Platform Models



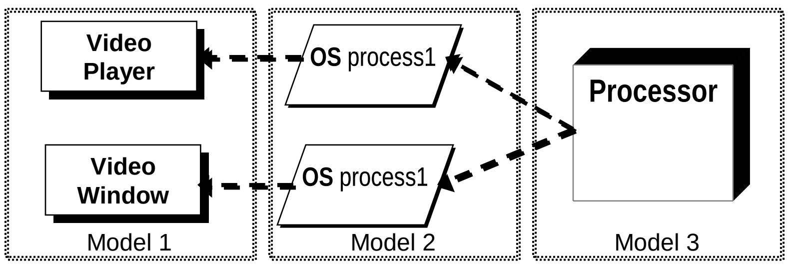
Realization Mappings

- A mapping between two models:



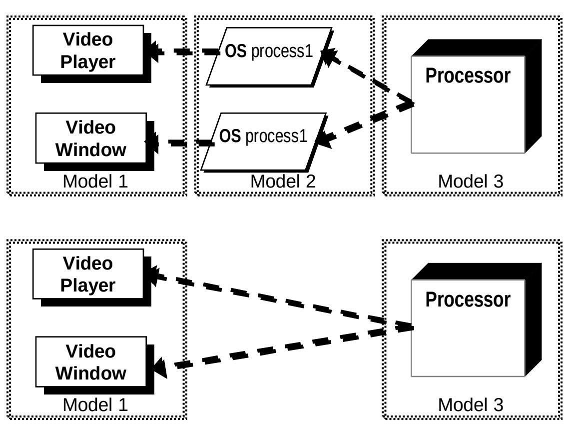
Realization Mappings Semantics

- Semantics: the logical elements are *realized* by the corresponding engineering model elements
  - logical elements can be viewed as being deployed on the corresponding engineering elements



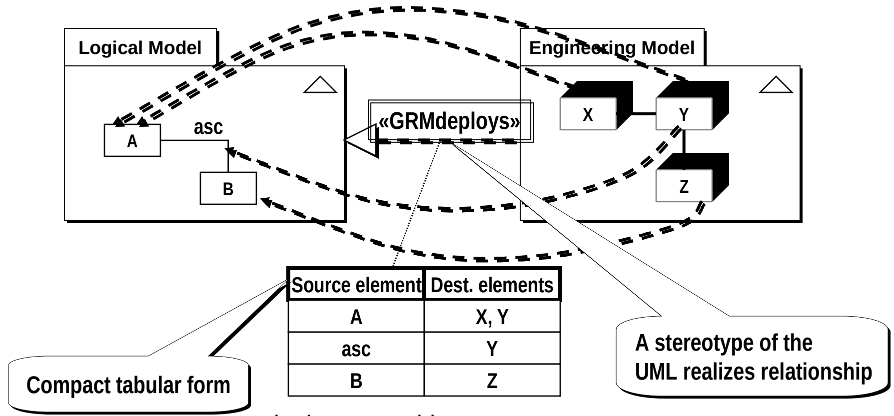
Choice of Level of Abstraction

- Intermediate levels may be abstracted out
  - depends on the desired granularity of modeling
  - affects the semantics of the realization relationship



Modeling Realization in UML

- An association between models with explicit realization mappings between model elements



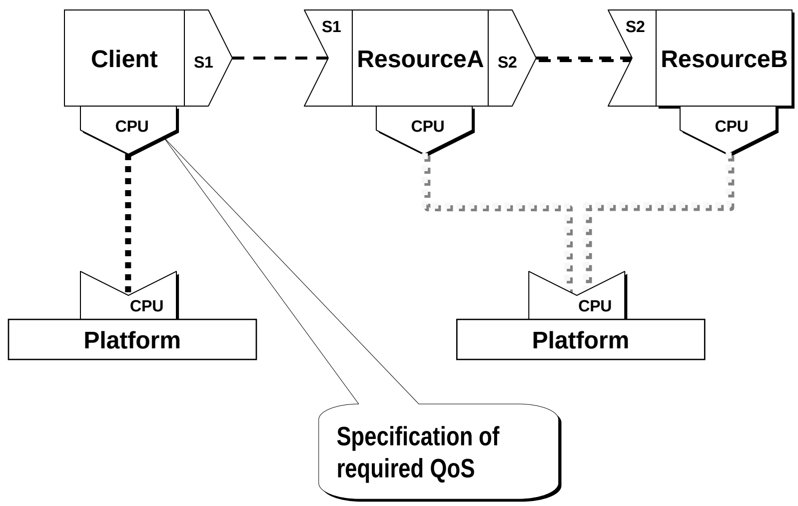
Source element	Dest. elements
A	X, Y
asc	Y
B	Z

deployment table

Contents

1. On software physics
2. An introduction to the standard real-time UML profile
3. Modeling resources and quality of service
4. Modeling time in UML
5. Modeling platforms in UML
6. Engineering-oriented MDD

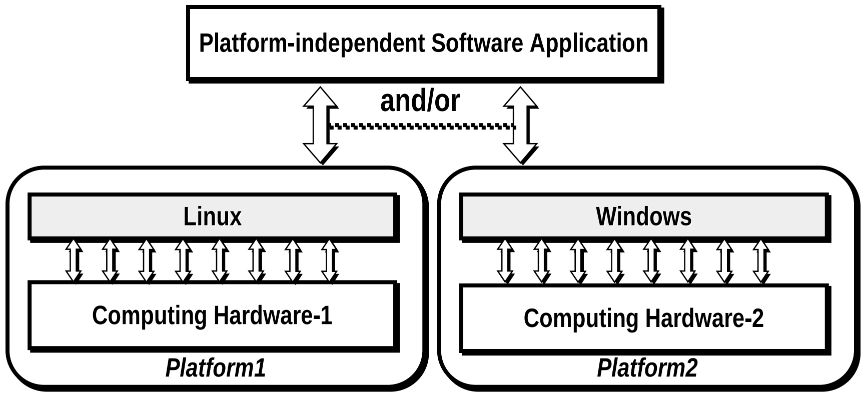
Reminder: The Layered Interpretation



Platform QoS

- Example platform QoS characteristics
  - Maximum acceptable context switching times
  - Minimum CPU execution speeds
  - Minimal memory requirements
  - Maximum acceptable communication delay
  - Minimal communication throughput
- Unfortunately, most software today is not explicit about its platform QoS requirements
  - Makes porting difficult

The Blessings of Platform Independence



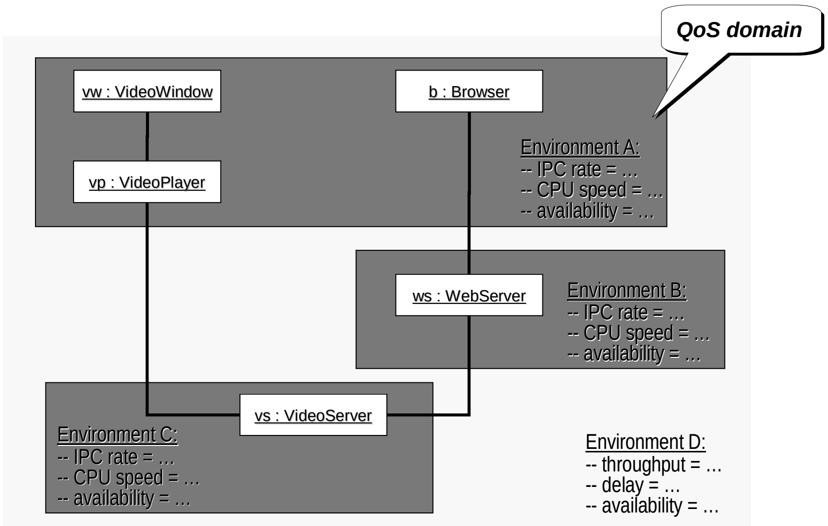
- Portability
- Protection from technology change
- Separation of concerns

Achieving Platform Independence

- Dilemma: *How can we achieve platform independence if our application has to be aware of platform characteristics?*
- Solution: *Include a technology-independent specification of the required QoS as part of the application*
  - Defines the envelope of acceptable platforms for the application *independently of specific technologies*

Required Environment Partitions

- Example: an Internet-based video application



121 IBM Software Group | Rational. software

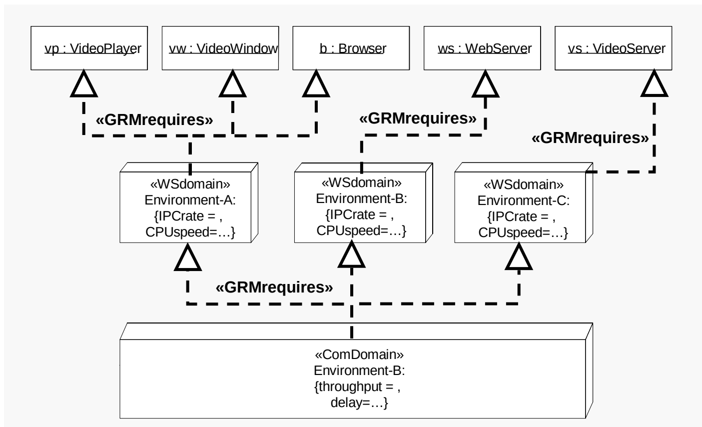
QoS Domains

- A domain in which certain QoS values apply uniformly:
  - CPU performance
  - communications characteristics (delay, throughput, capacity)
  - failure characteristics (e.g., availability, reliability)
  - etc.
- The QoS values of a domain can be compared against those of any concrete platform to determine its suitability

122 IBM Software Group | Rational. software

Modeling QoS Domains in UML

- «GRMrequires» = stereotype of «GRMdeploys»
- Defines a reference platform for an application



Model-Driven Engineering

- *Design of software based on use of:*
  - *Models (i.e., model-driven development)*
  - *QoS specifications (accounting for physical properties)*
  - *Quantitative and qualitative analysis techniques and computer simulation*
- Advantages:
  - Higher reliability (simplification due to use of models)
  - Early detection of design flaws and inadequacies => increased productivity
  - Platform independence

Conclusion

- With the availability of:
  - A ***standard*** language for specifying qualitative aspects
  - A ***standard*** means of specifying non-functional aspects
  - Mature computer-based analysis and design tools

  …we can perhaps, at long last, raise the level of reliability of software engineering to that which we have come to expect in traditional engineering disciplines