On Model-Driven Software Engineering

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

Thesis

1. Software engineering is in much greater need of engineering methods than many of its practitioners assume

2. The current situation can be greatly improved through the systematic application of model-driven design and development methods

A Giant Speaks...

Edsger Wybe Dijkstra (1930 – 2002)

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

"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

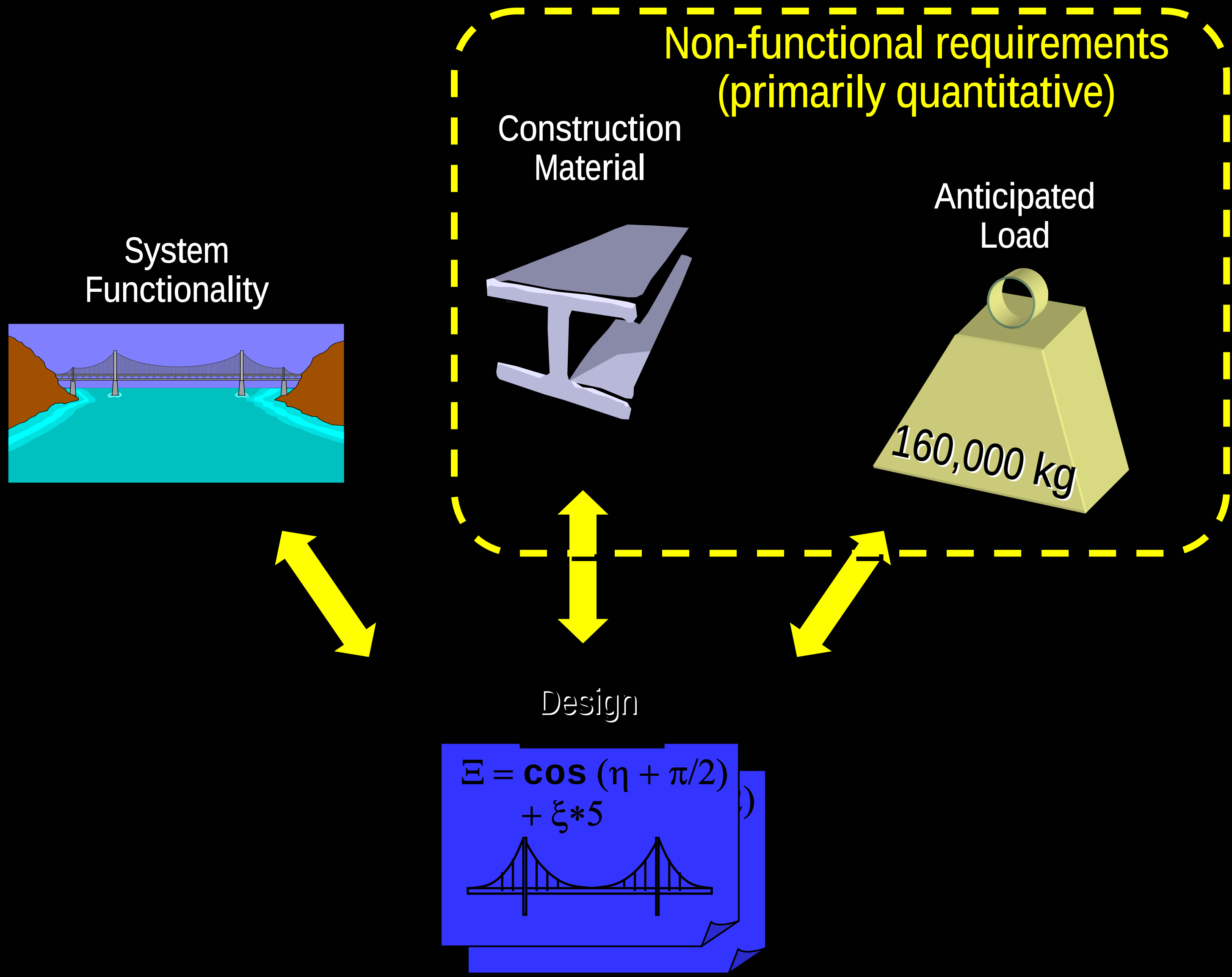
Engineering

- *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"*

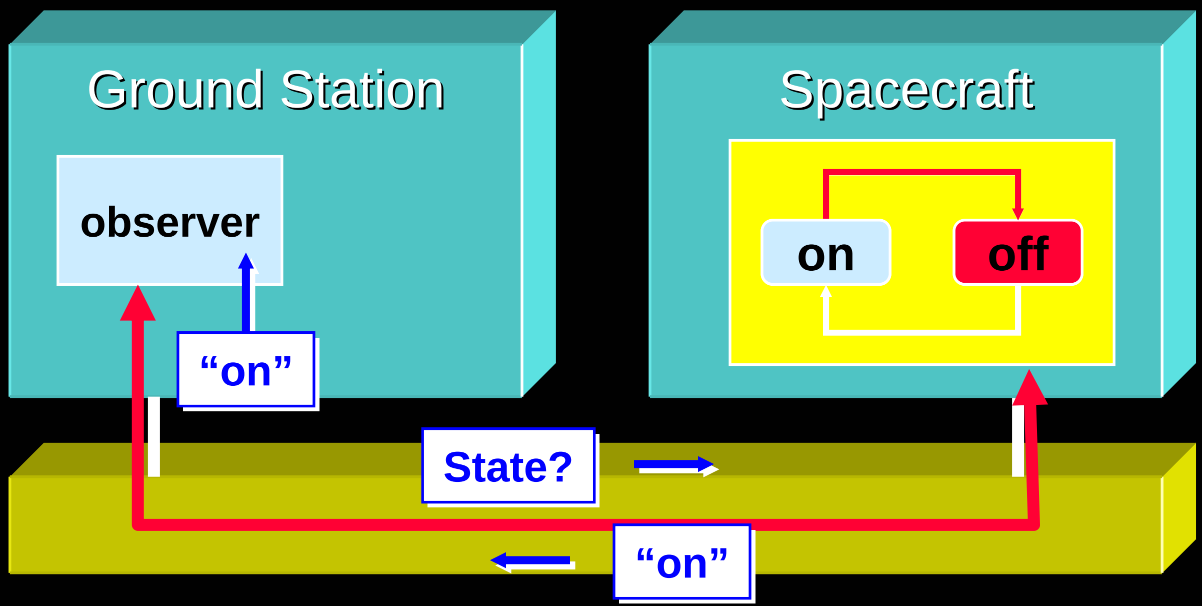
The Classical Engineering Design Problem



What is Software Made of?

The Case of Distributed Systems

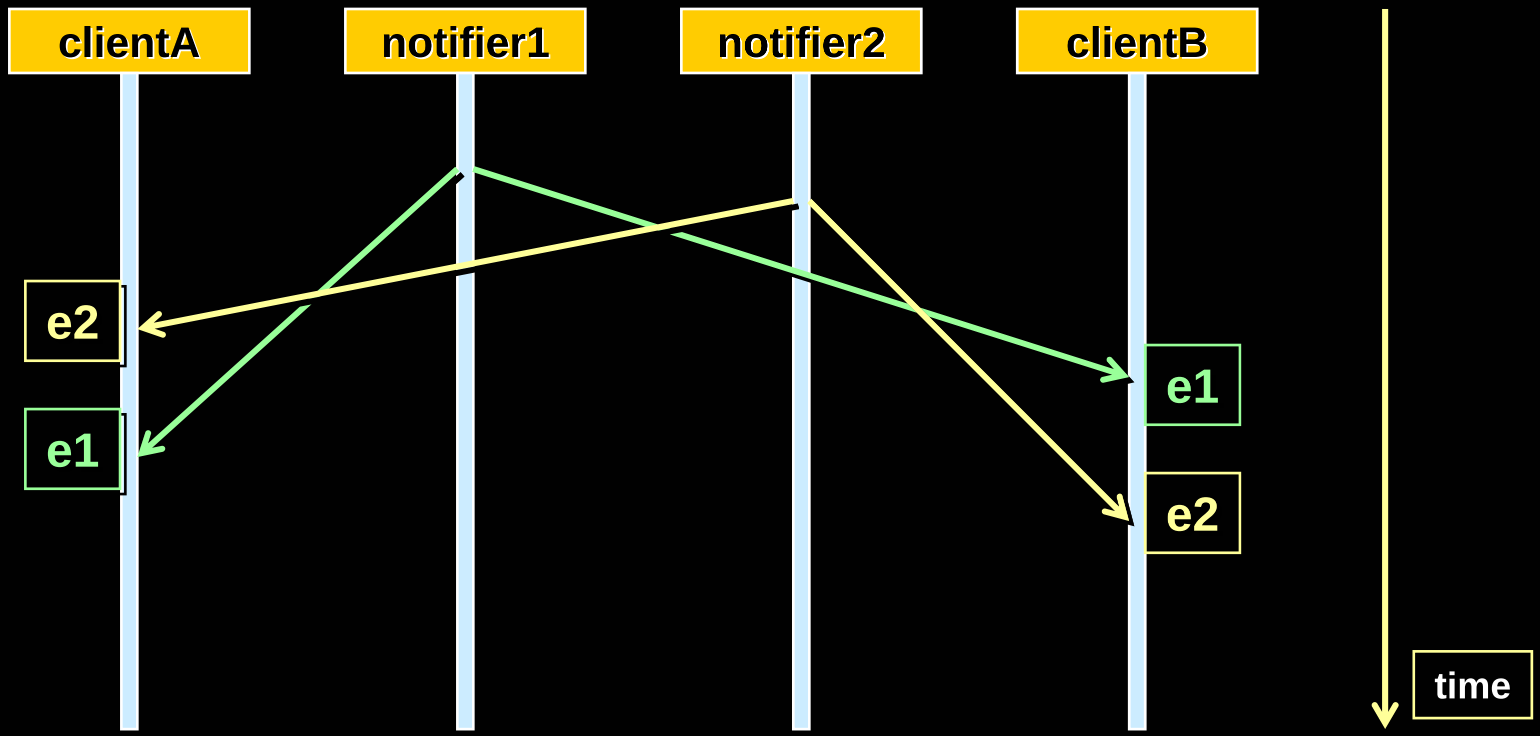
- Possibility of out of date status information due to transmission delays



- The physical characteristics of the computing environment affect the control logic

The Effect of Communication Media

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



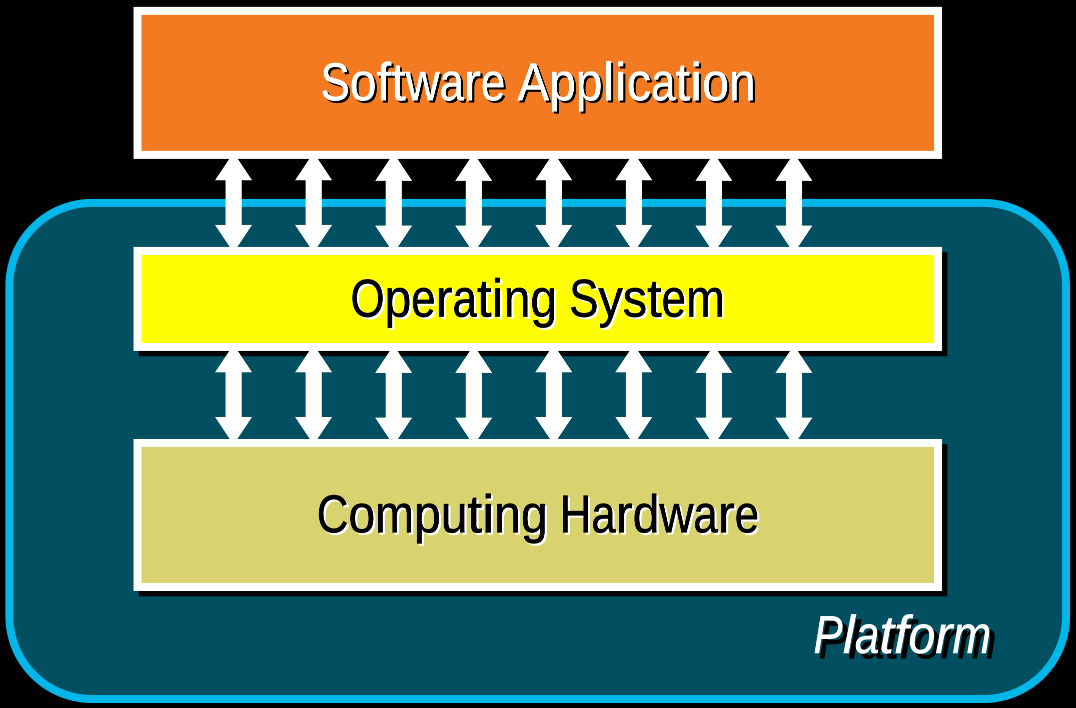
- Can we not hide this by adding “fault transparency” layers?

A Fundamental Theoretical 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.

What Software is Made of



- The raw material of software is its platform
  - Combination of software and hardware
  - Always bottoms out with the hardware
  - The hardware imposes its physical properties to the software (speed, reliability, capacity, etc.)

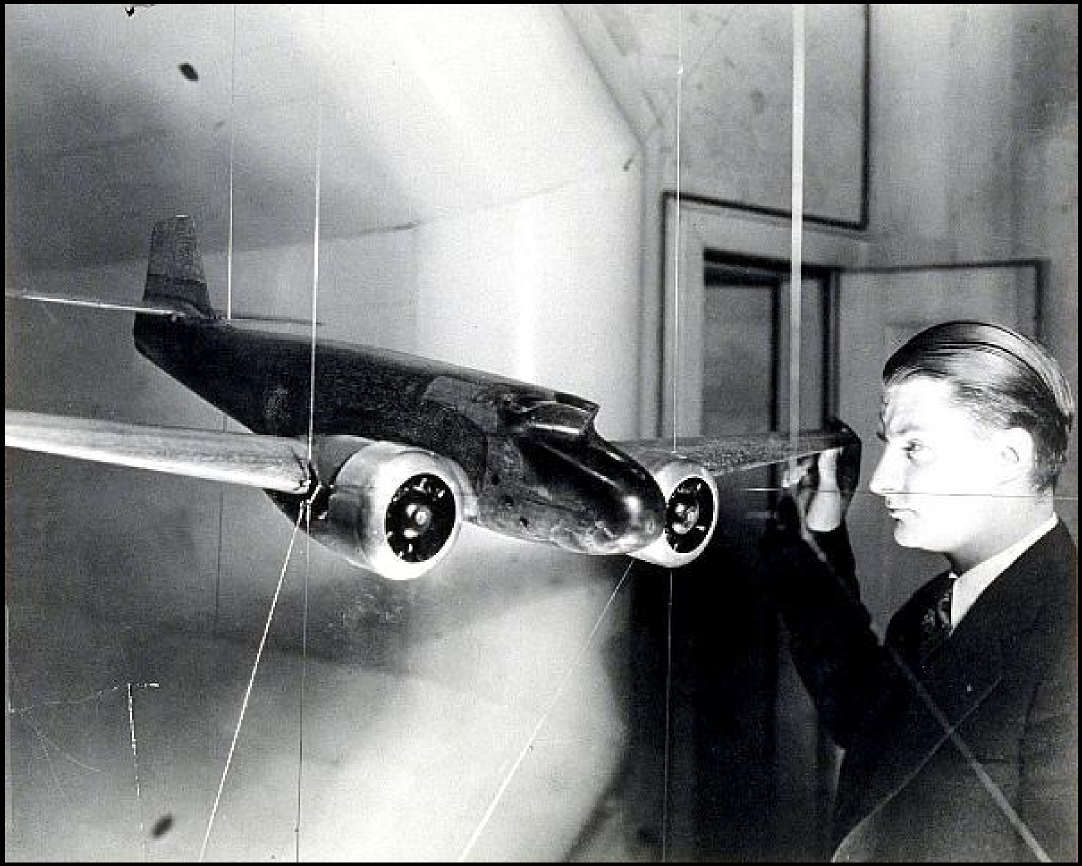
“Physical Programming”

- Computer System = Software + Hardware
- Claim: *the physical characteristics of the platform should, in many cases, be a first-order concern when designing software*
  - More applicable by the day as our demands for availability and reliability of software grow and as the levels of distribution increase
- The bad news: the physical world is inherently complex
- …and, what about platform independence?

Models in Engineering and Software

Models in Traditional Engineering

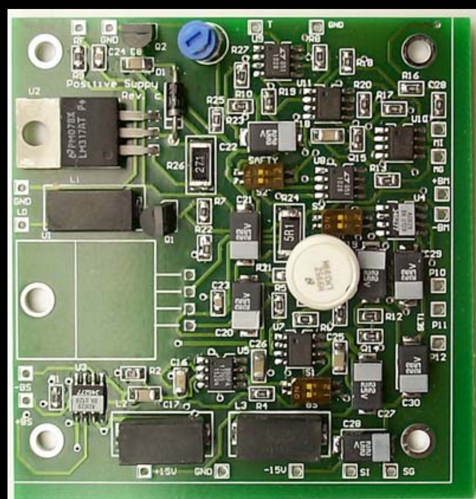
- Probably as old as engineering (e.g., Vitruvius)



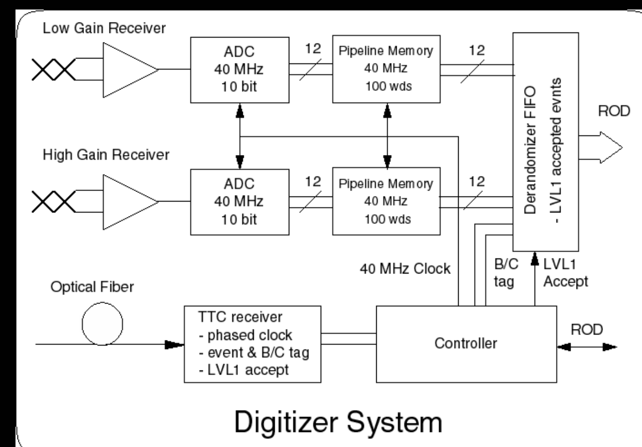
Engineering Models

- Engineering model:

  A <u>reduced representation</u> of some system that highlights the properties of interest <u>from a given viewpoint</u>



Modeled system



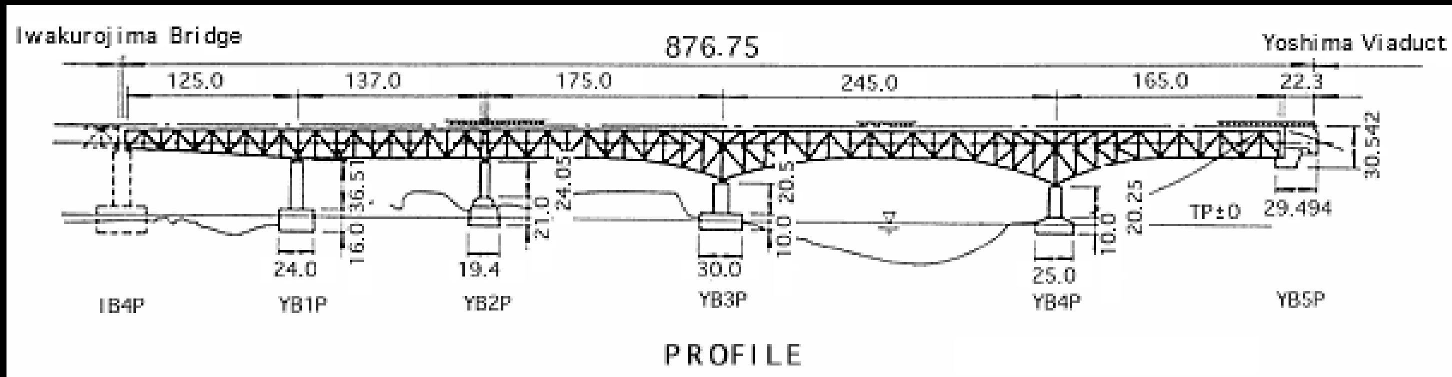
Functional Model

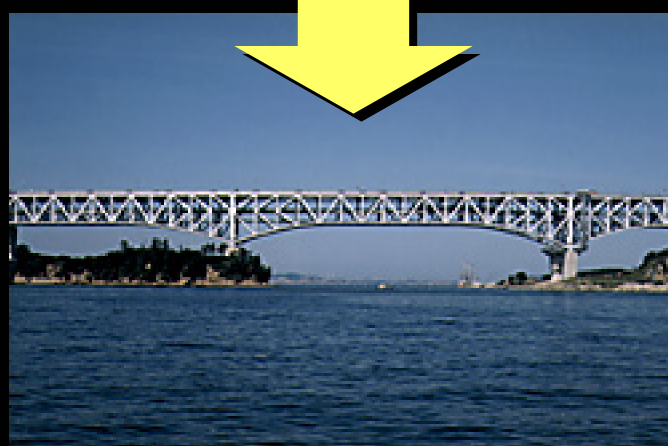
- We don't see everything at once
- We use a representation (notation) that is easily understood for the purpose on hand

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

Models versus Systems





Differences due to:

- Idiosyncrasies of actual construction materials
- Construction methods
- Scaling-up effects
- Skill sets/technologies
- Misunderstandings

Can lead to serious errors and discrepancies in the realization

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 all 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?

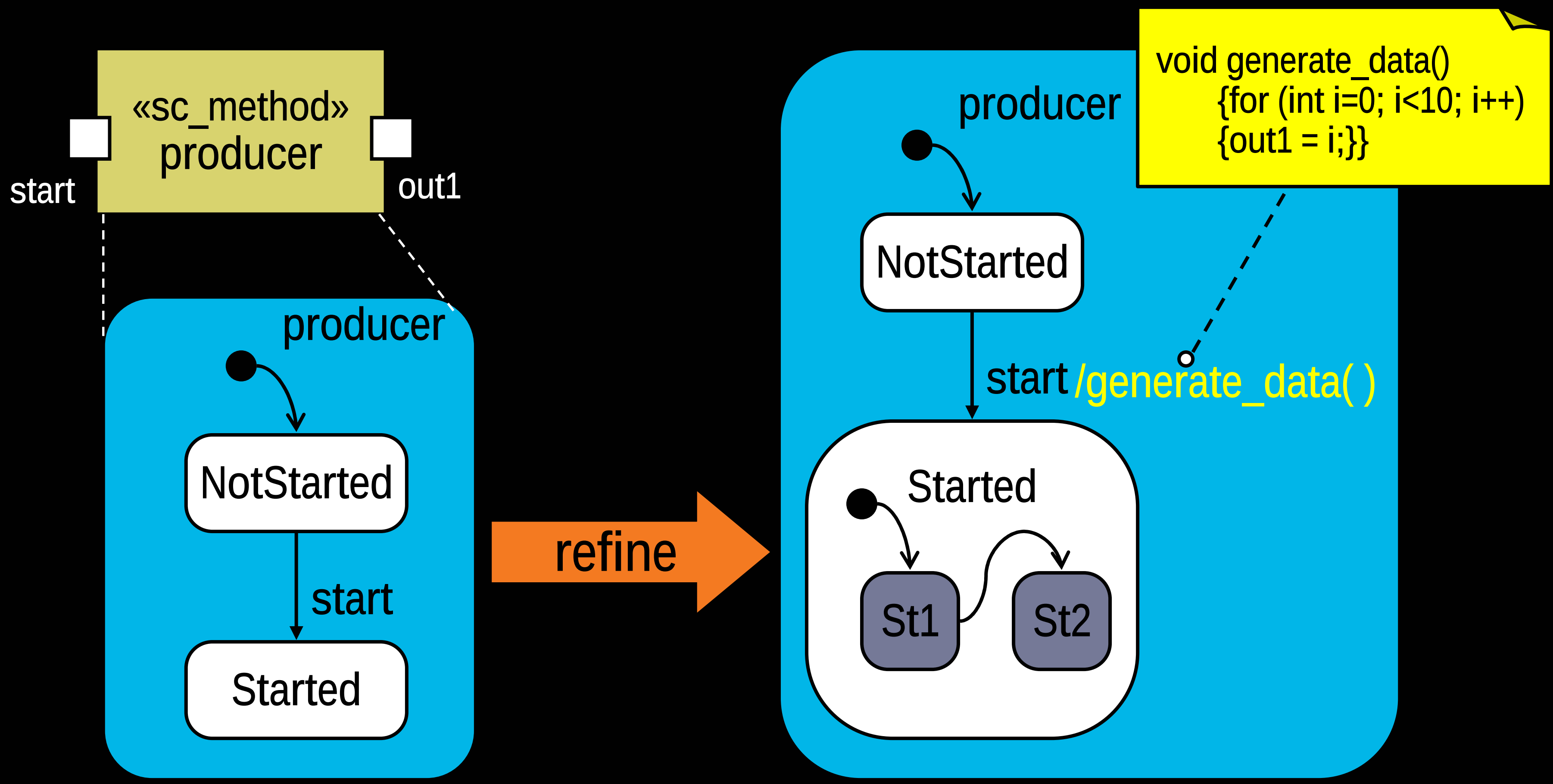
...and its UML Model

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

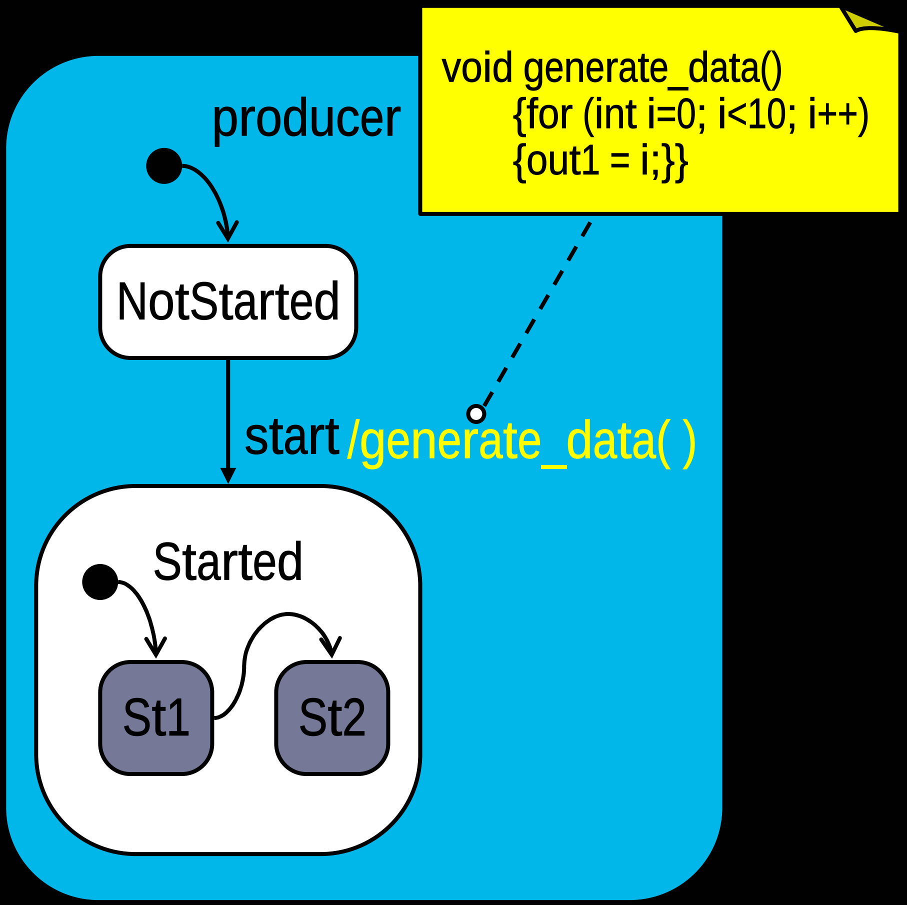
The Remarkable Thing About Software

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

How Computers Help

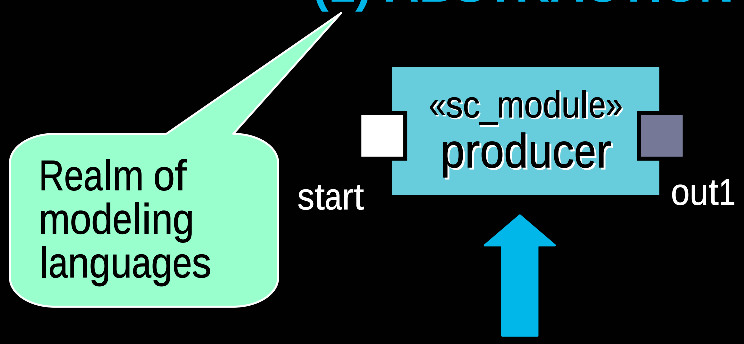


- We hide detail by selecting a view and letting the computer do the rest

Model-Driven Style of Development (MDD)

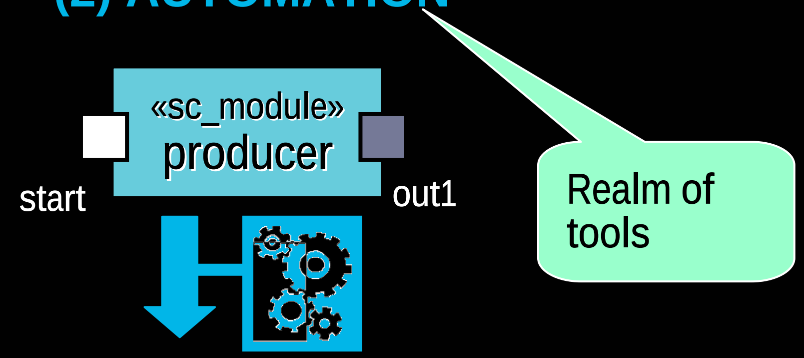
- An approach to software development in which the focus and primary artifacts of development are models (as opposed to programs)
- Based on two time-proven methods

(1) ABSTRACTION




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

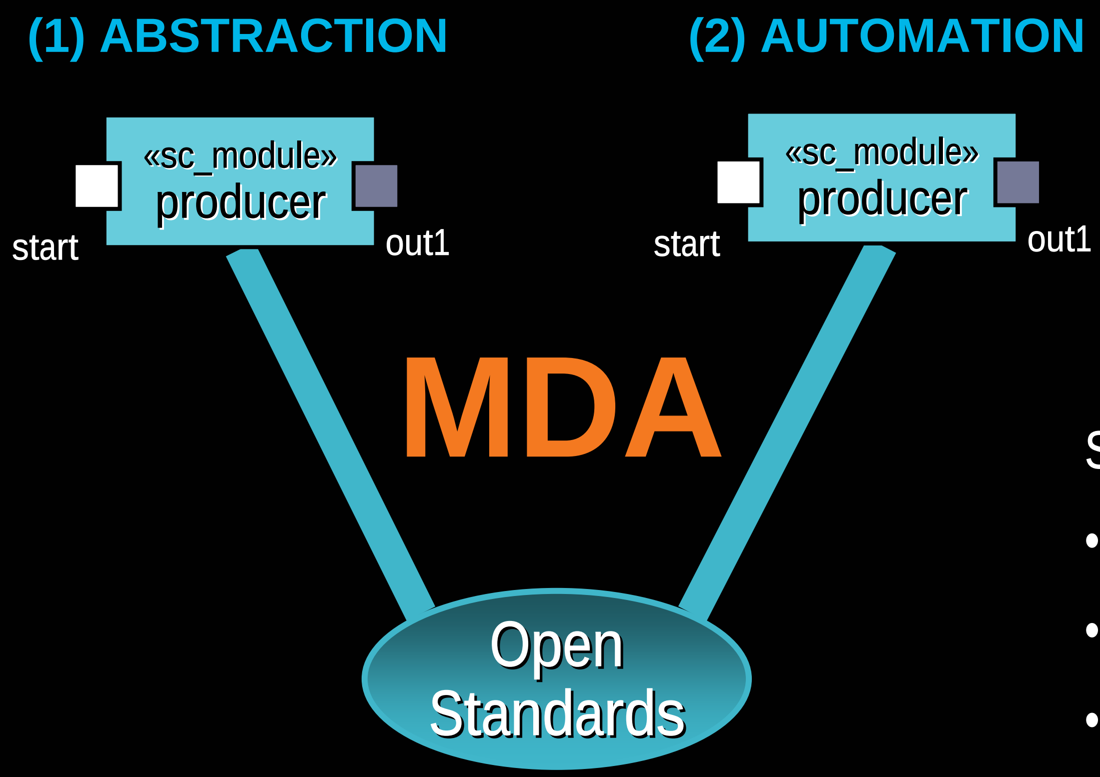
(2) AUTOMATION




```
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 *open* standards in support of MDD



Standards for:

•Modeling languages

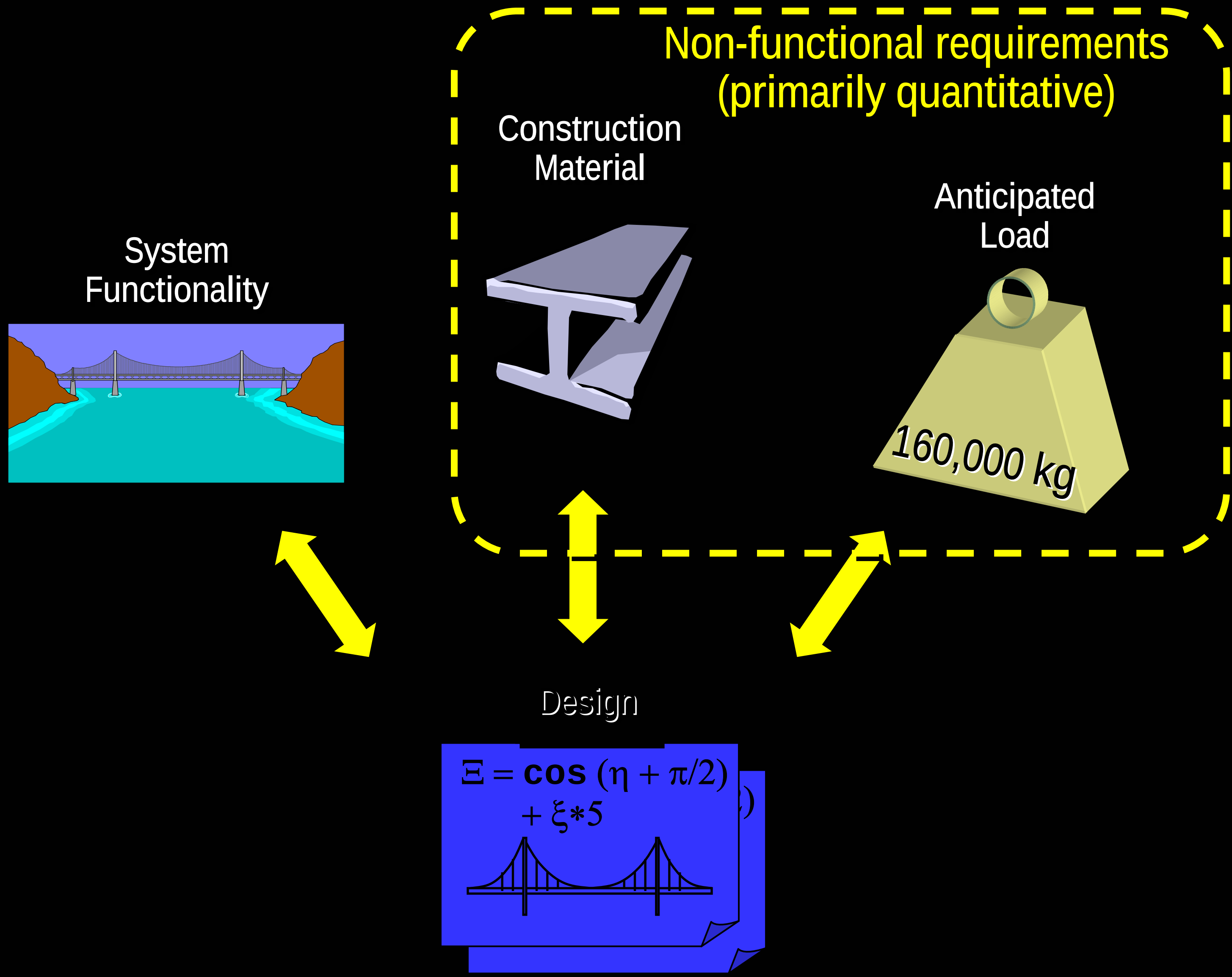
•Model transformations

•Software processes

•Model interchange...

The Engineering of Software

The Classical Engineering Design Problem



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

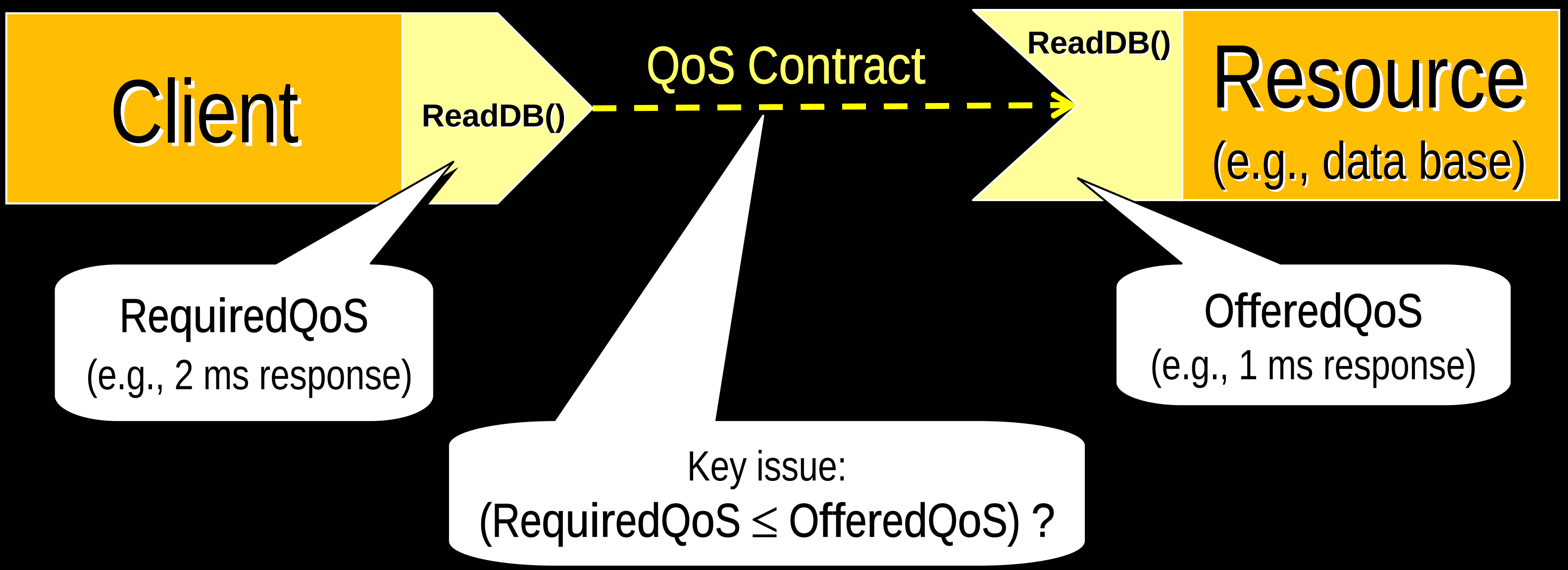
  - throughput, latency, capacity, response time, availability, security...
  - usually a quantitative measure
- QoS concerns have two sides:
  - *offered QoS:* the QoS that is available (supply)
  - *required QoS:* the QoS that is required to do a job (demand)

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

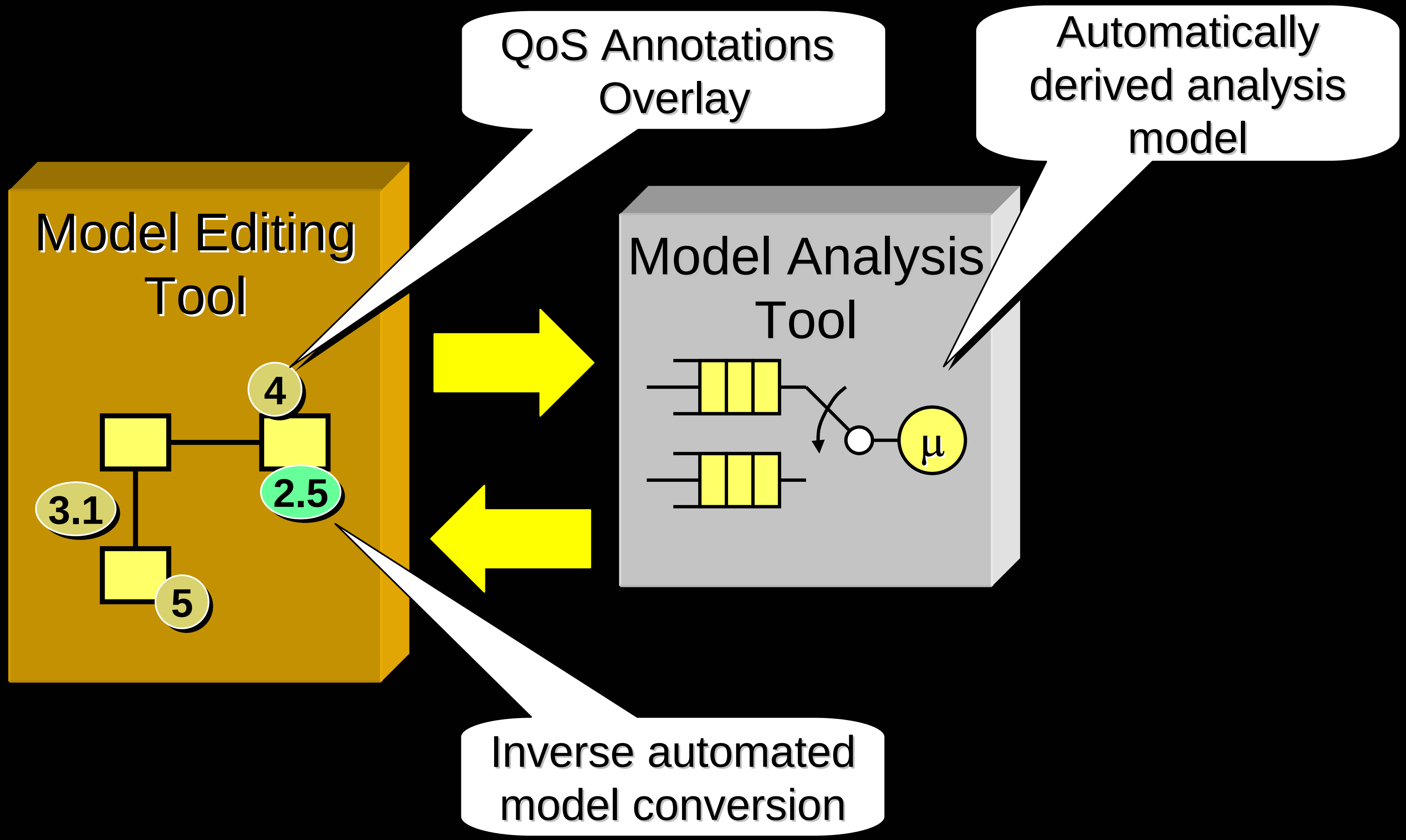


Verifying QoS Contracts

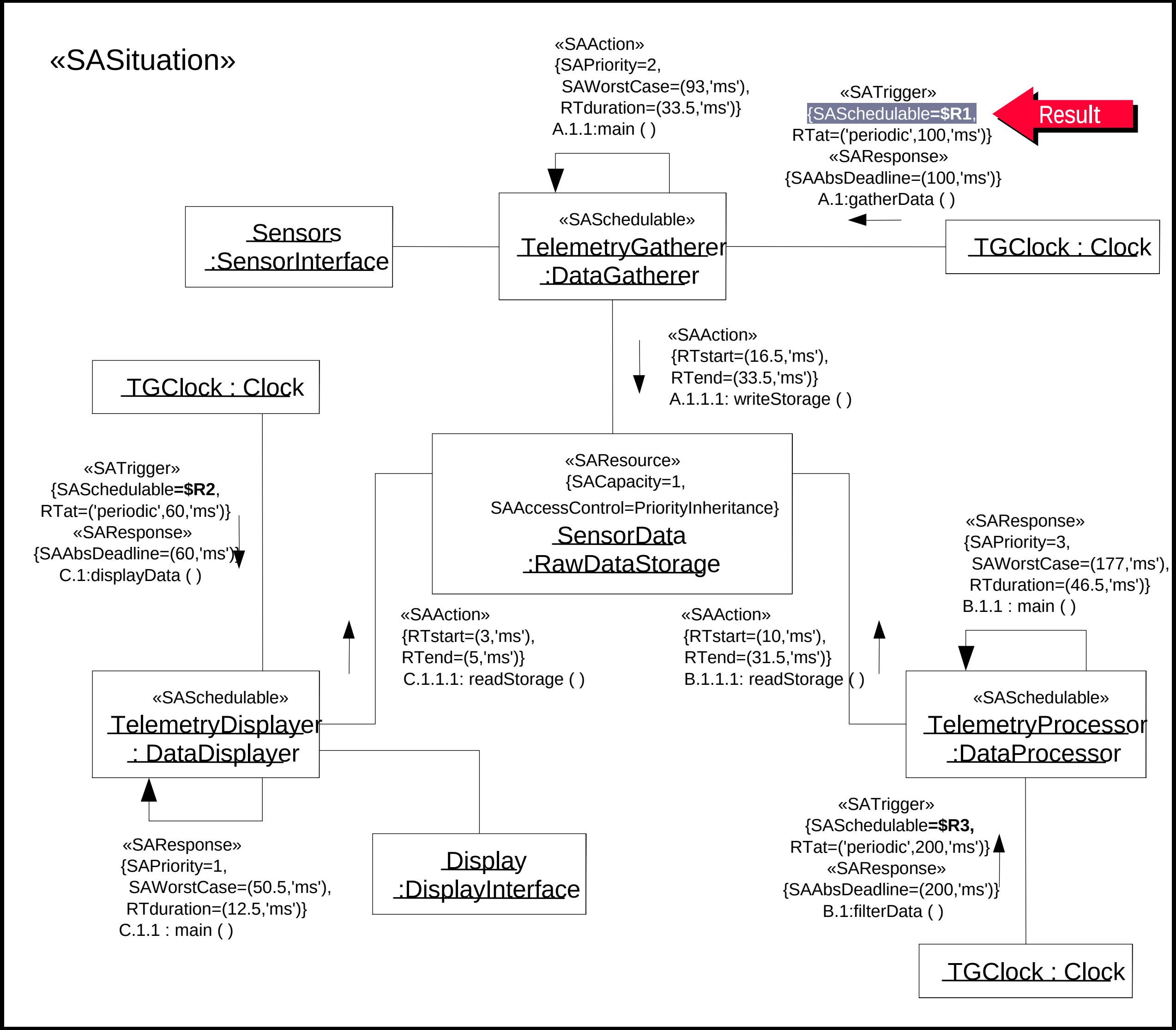
- Can QoS contracts be statically checked by a compiler?
  - The good news: Yes (in most cases)
  - The bad news: it is usually hard
- Some issues:
  - In most cases QoS verification cannot be done incrementally – the full system context is required
  - Each type of QoS (e.g., bandwidth, CPU performance)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

Automating Engineering Analysis

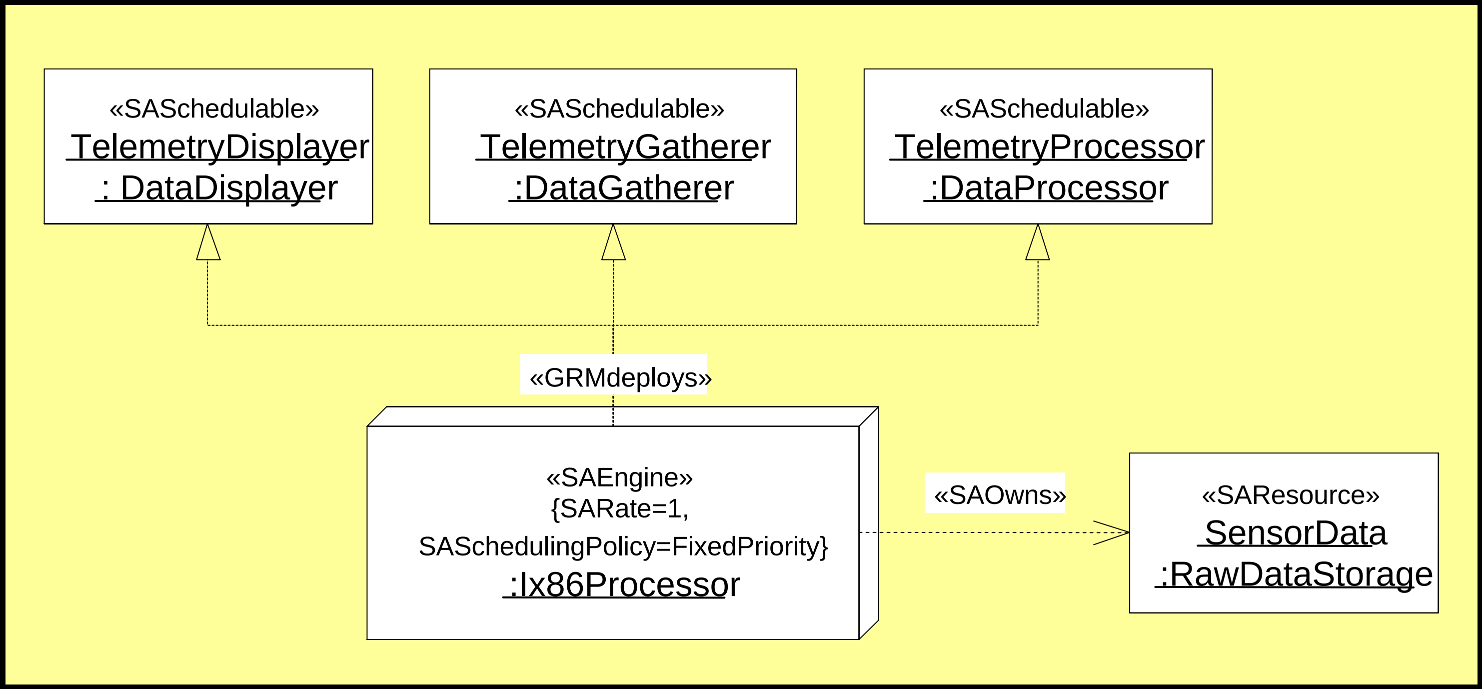
- Inter-working of specialized tools via shared standards



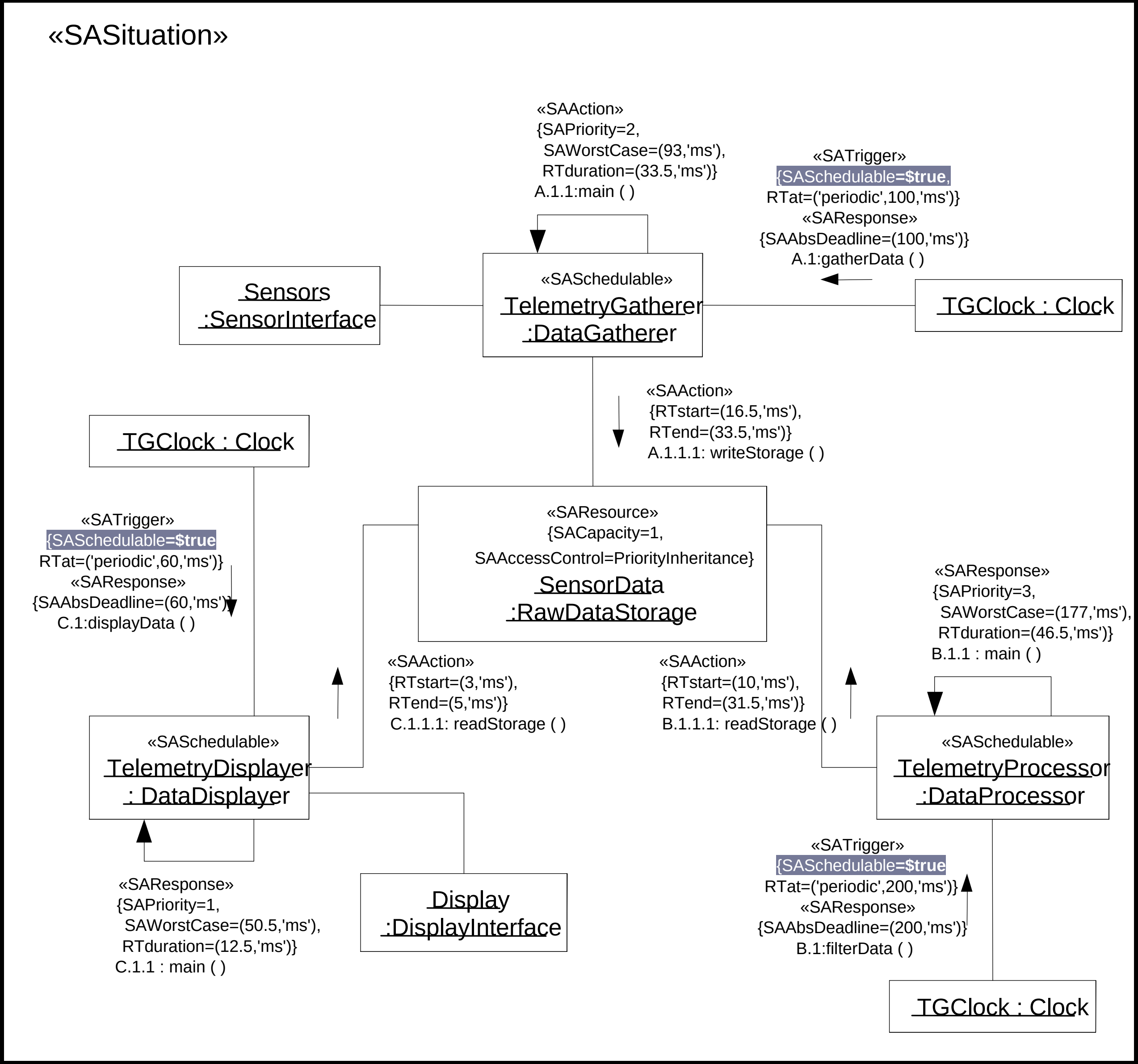
Example: Schedulability Annotations



Example: Deployment Specification

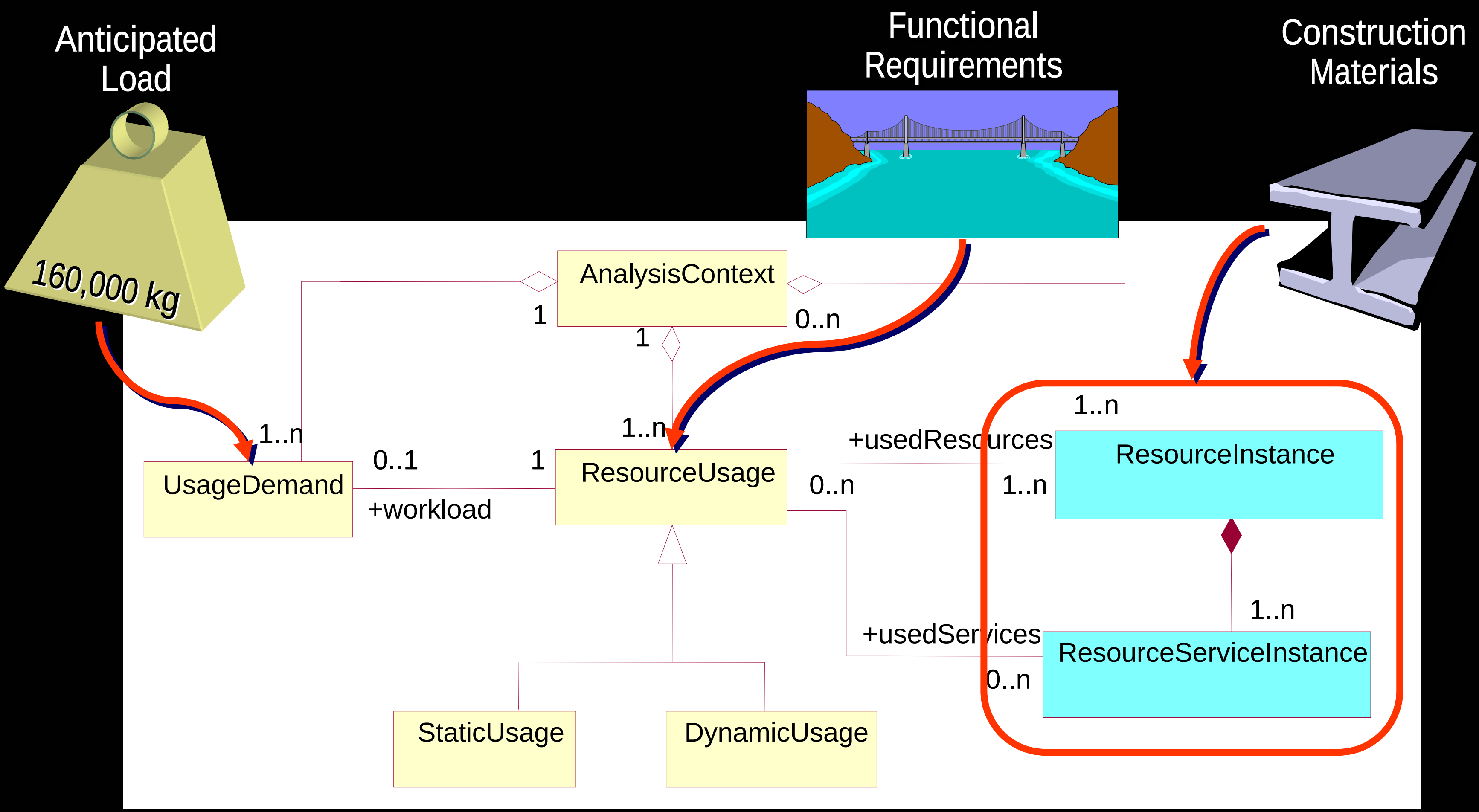


Example: Analysis Results



Basic Resource Usage Model

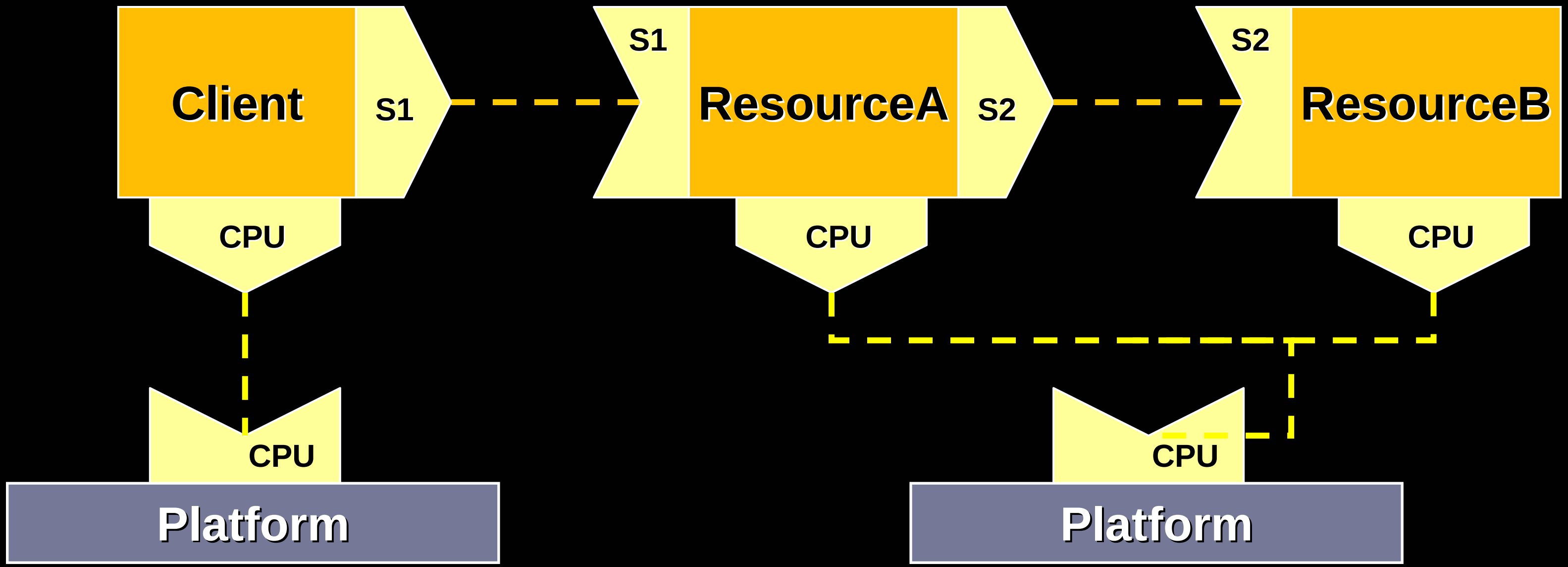
- Models a particular situation that needs to be analyzed for some time-related property (e.g., response time)



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

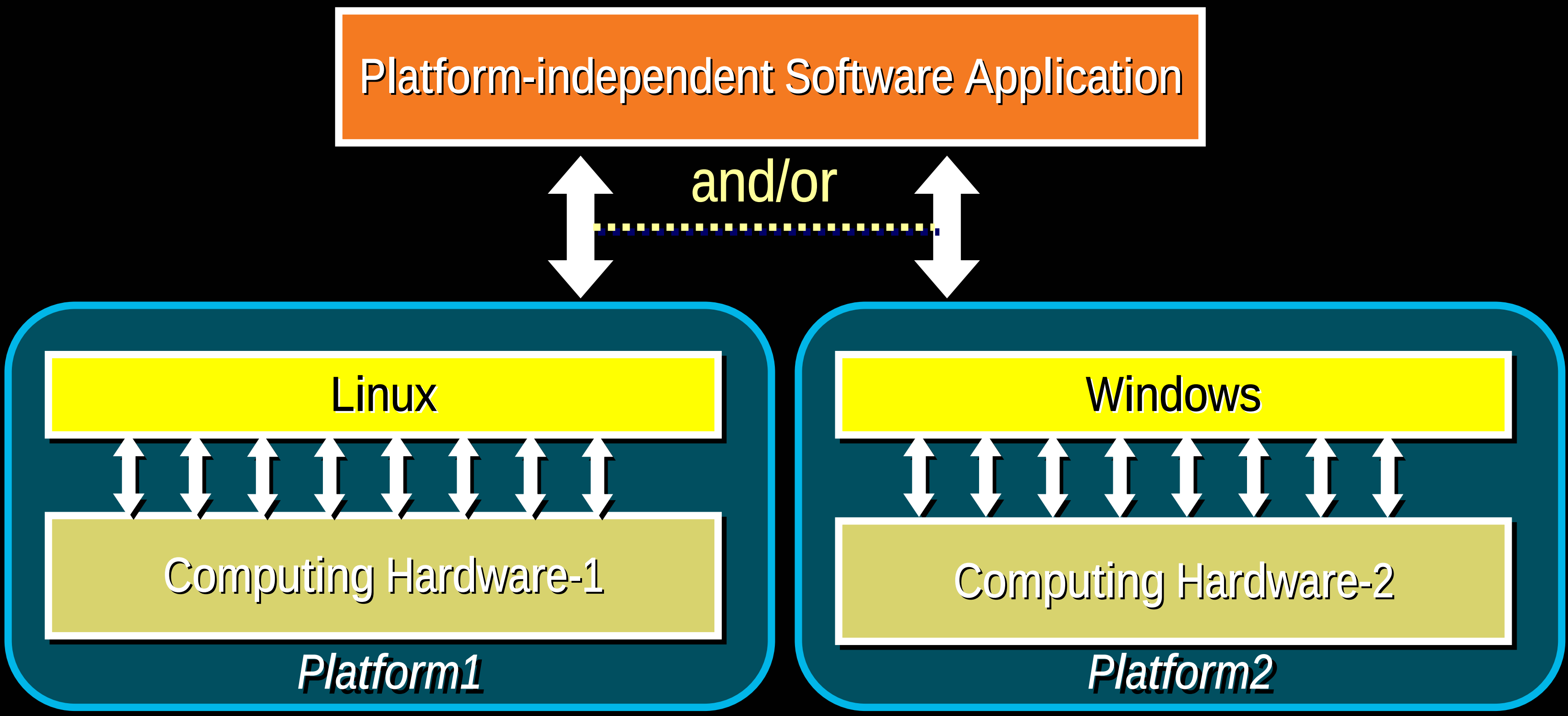


Platform QoS

- 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

IBM



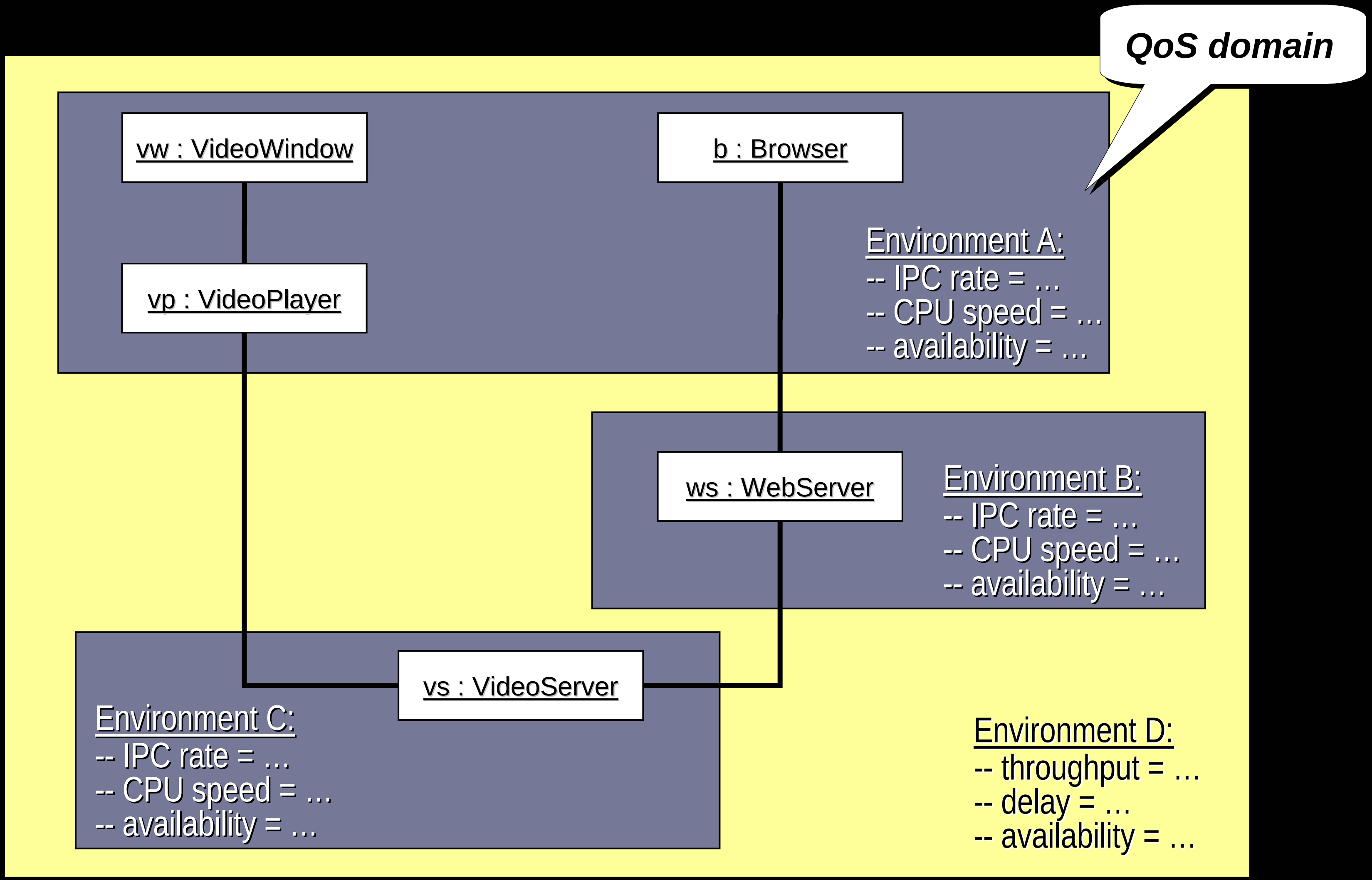
- Portability
- Protection from technology change
- Separation of concerns

Achieving Platform Independence

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

Required Environment Partitions

- Example: an Internet-based video application

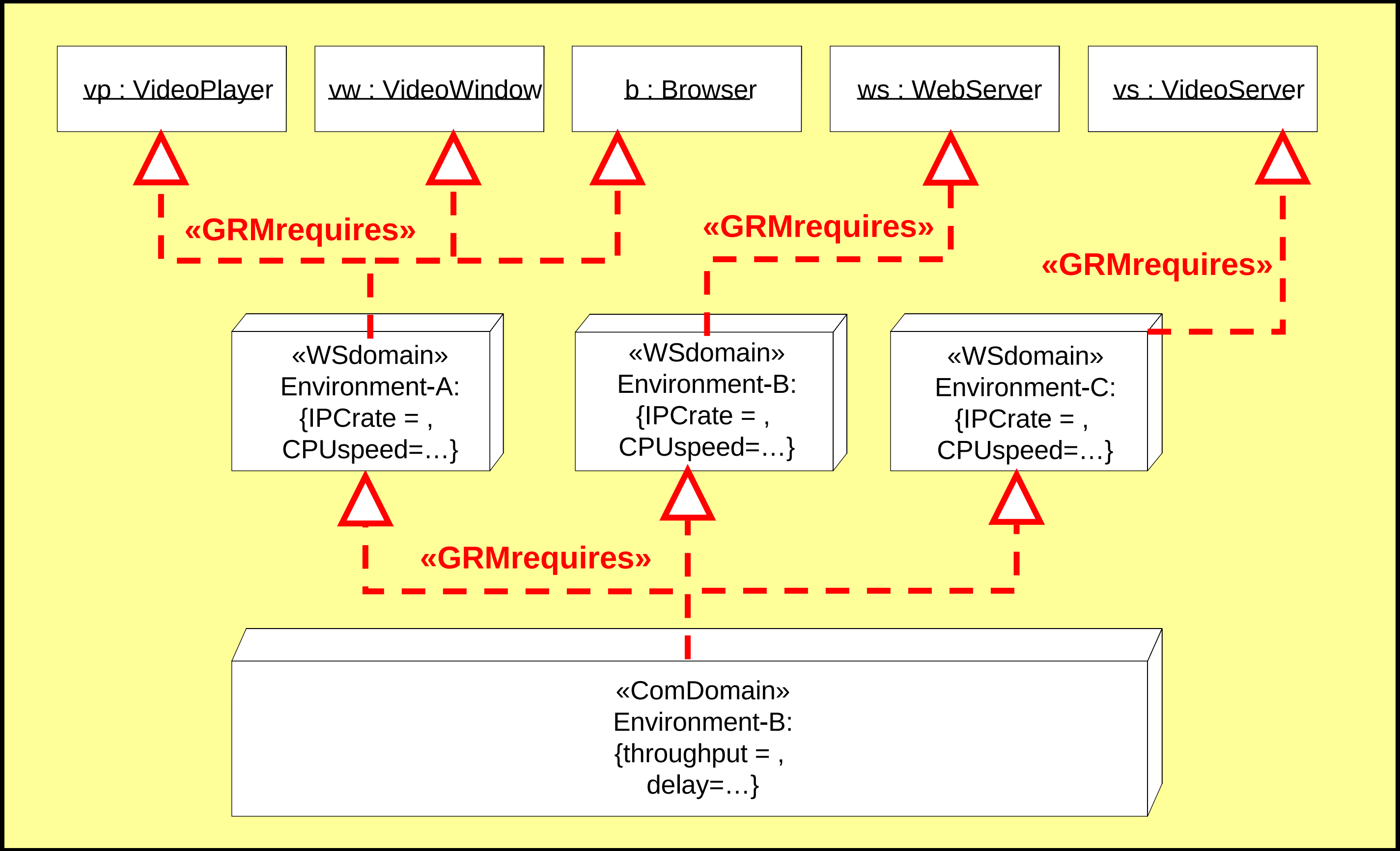


QoS Domains

- 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

Modeling QoS Domains in UML

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



Conclusions

IBM

- *Software design can be much more than applied logic:* *The design of software is, in many cases, strongly dependent on the physical characteristics of the underlying platform*
- *Model-driven software engineering, characterized by:*
  - *Models of both application software and platforms*
  - *Qualitative and quantitative analysis techniques (including computer-based model execution)*
- Resulting in increased productivity and product reliability:
  - Early detection of design flaws
  - Increased levels of abstraction
  - Increased levels of automation

QUESTIONS?

(bselic@ca.ibm.com)