A Method of Building Executable Platform-Independent Application Models

Janusz Dobrowolski @ StateSoft.org

Dr. Jerzy Kolodziej @ StateSoft.org

StateSoft Inc

Prof. Bogdan Korel @ iit.edu

Illinois Institute of Technology (IIT)

Contents

- PIM Platform Independence Definition
- VeUML Modeling Goals
- VeUML Behavior Model Simplification
- Classes which should have behavior represented as State Machine models in VeUML.
- Conversion Between Behavior models in VeUML
- Projects Experience
- Independent iteration in Logical (PIM) and Physical (PSM) Architectures

VeUML Modeling Goals

- Maximizing the amount of automatically generated executable
- Maximize the amount of automatic validation
- Syntactically simplest models for complex systems
- Substantial complex system maintenance cost reduction

VeUML as the ITU Profile Candidate

International Telecommunication Union (ITU-T) SG17 in March 2004 decided to include VeUML as a candidate profile for the Telecommunication platforms, services and protocols for the study period 2004 – 2007.

VeUML systems modeling goals were considered as a factor in the decision making process.

Things hard or impossible to achieve without MDA

- Automatic Protocol, Platform and Services Generation
- Automatic Validation
- Services Feature Interactions Tracing
- Construction of Toolkits aware of precise domain knowledge

Renaming the problem is not a Solution

- Control
- Behavior
- Business Logic
- Preconditions, Postconditions
- Platform Independent Service Logic
- Policies
- Complex Decision Tables

VeUML provides solution to the above problems

VeUML Key Approach to Conquer Behavior Complexity:

Separation of Domains:

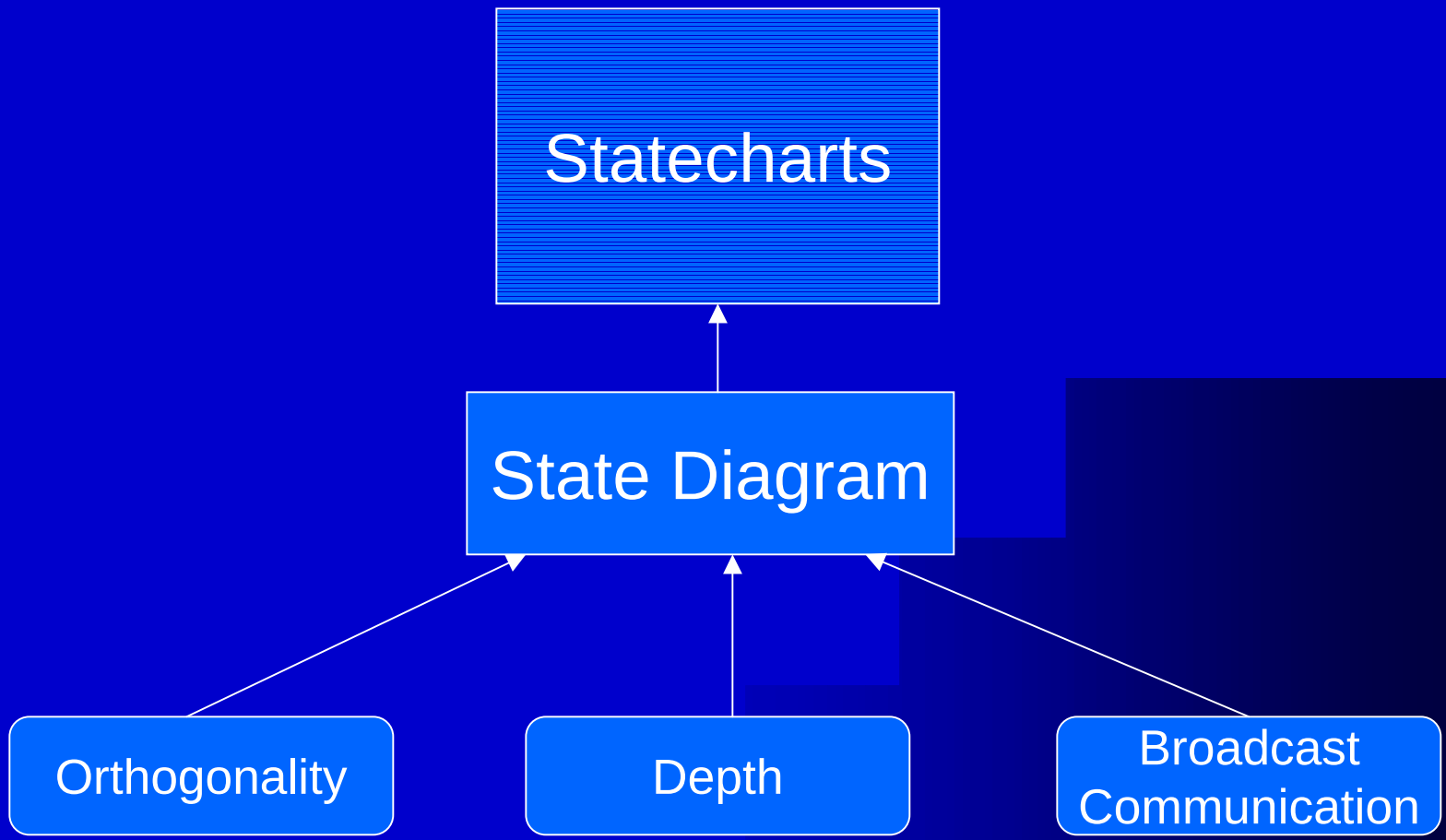
- **Service and Protocol Logic from Platform**
- **Class Data *from* Class Behavior**
- **Behavior *from* Concurrency -**
  (Generalized Objects Factory Pattern)
- **Events *from* Data**
- **Actions *from* Data**
- **Class behavior *from* the PSM Programming Language *(e.g., C++, Java)***

PIM Model Platform Independence Definition

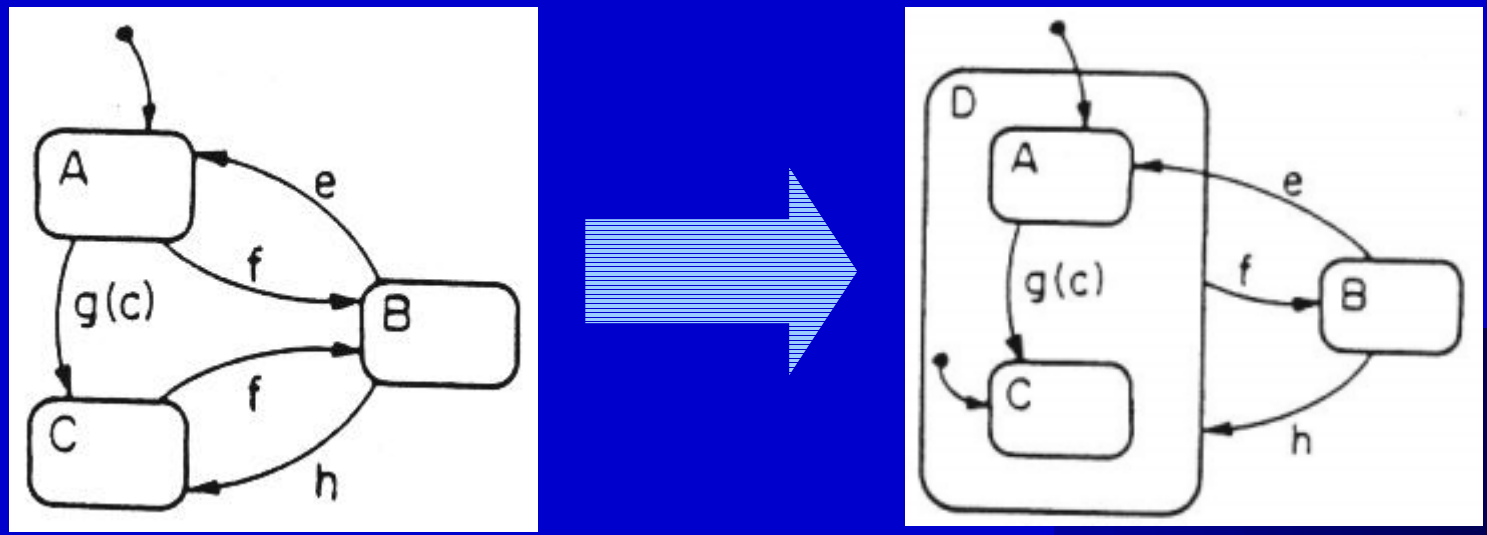
- Independence from PSM programming language
- Independence from PSM Operating System
- Independence from PSM Data Base

Behavior Models Simplification

Statecharts



Statecharts Notion of Depth



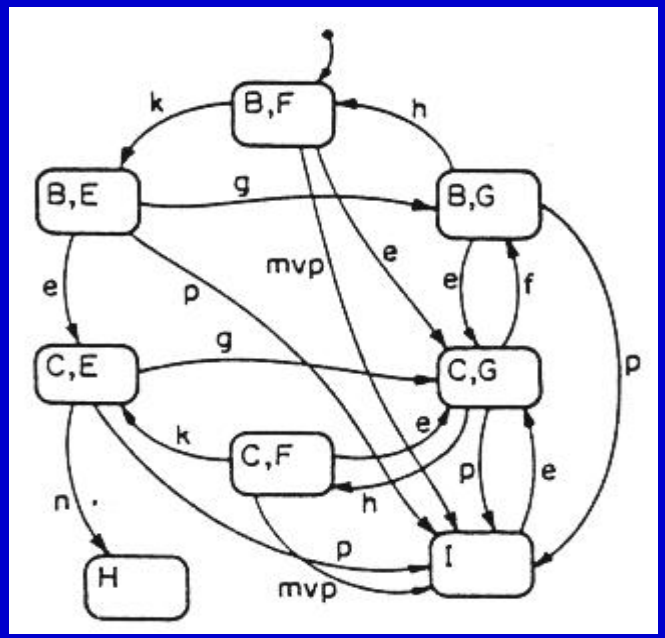
- **VeUML: Depth is OK on post-synthesis only; during the modeling Depth leads to an observer rather than an implementer viewpoint.**

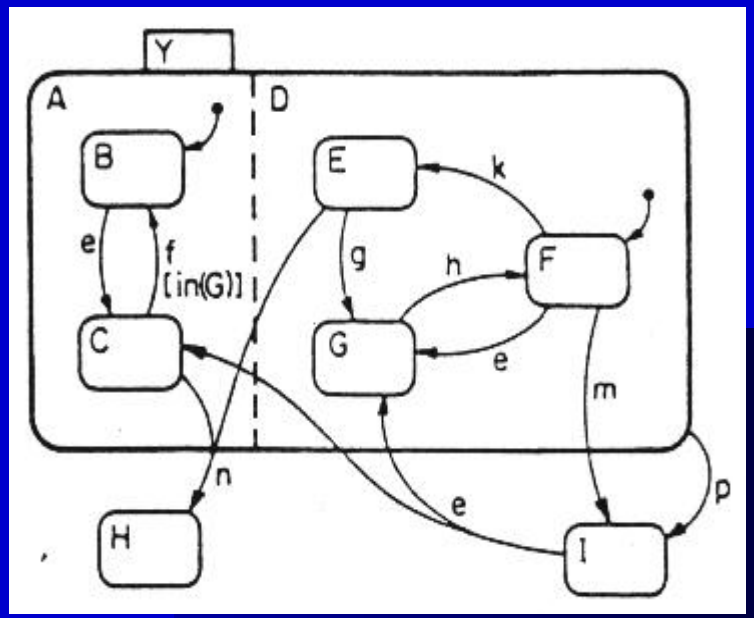
VeUML: Hierarchy of Classes

not Hierarchy of State Machines

Source of Drawing : Statecharts Definition

Orthogonality – the biggest Statecharts Fallacy

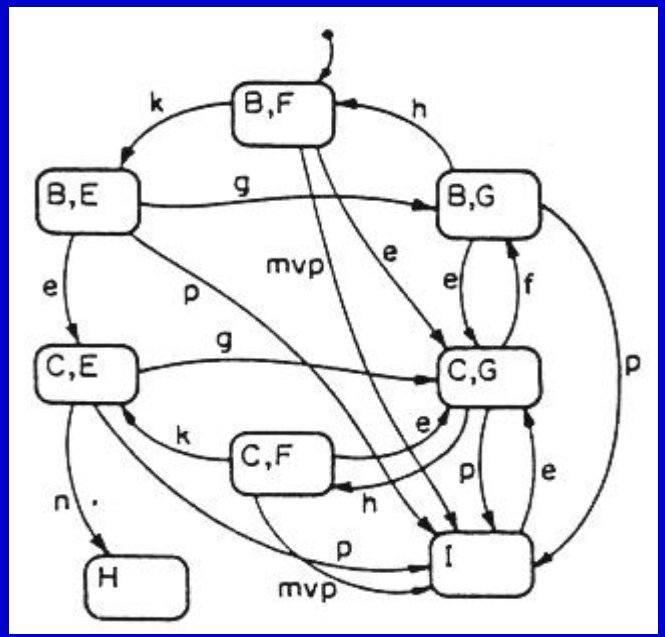


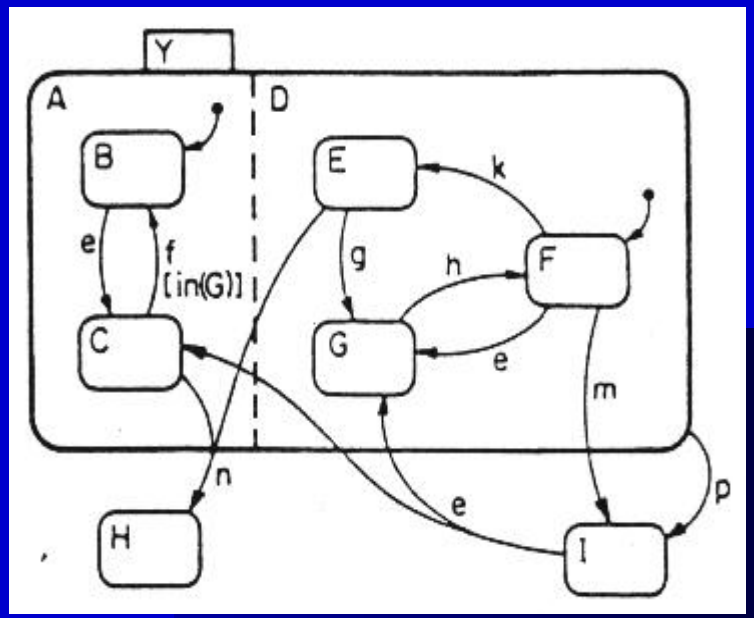


- **VeUML and XUML view – Concurrency within a class is a proof of an improper OOA/OOD architectural decomposition**

Source of Drawing : Statecharts Definition

Orthogonality – the biggest Statecharts Fallacy

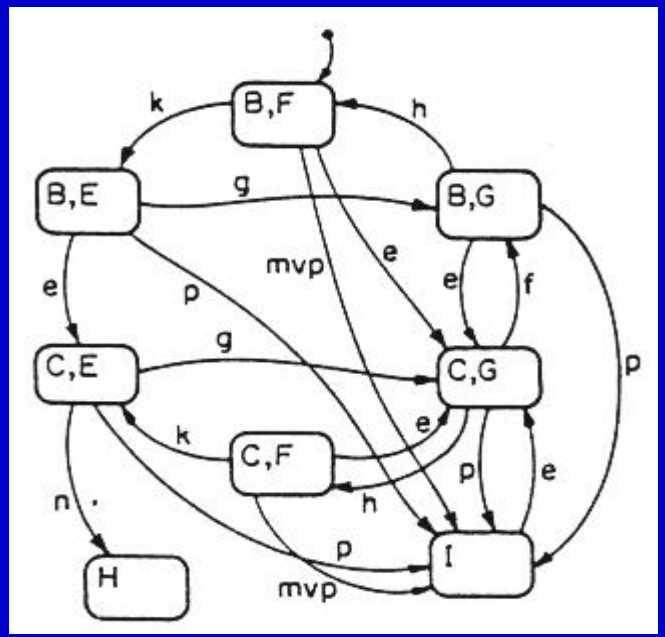


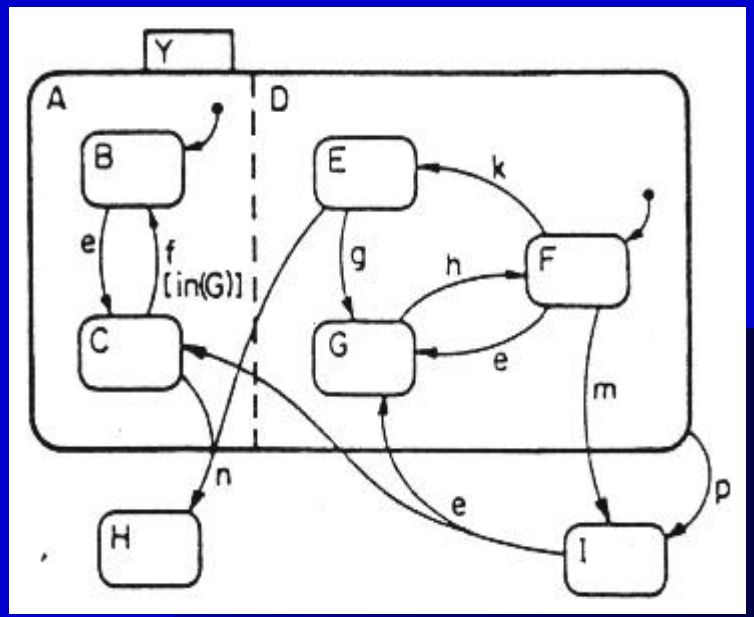


- **VeUML and XUML view – There might had been never a reason to combine independent State Machines as shown on the left side figure.**

Source of Drawing : Statecharts Definition

Orthogonality – the biggest Statecharts Fallacy

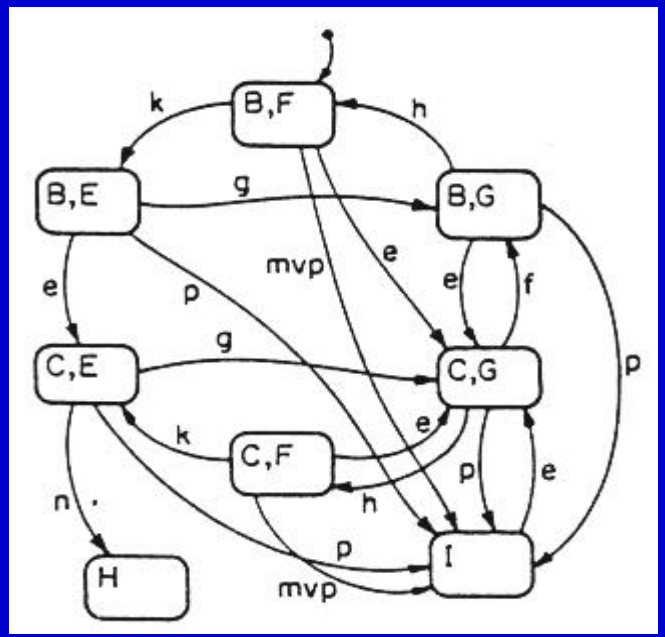


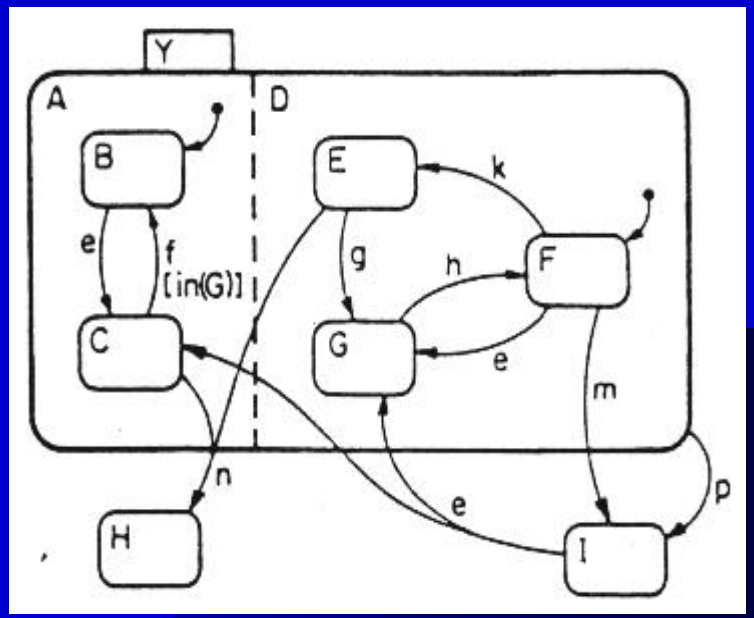


- **VeUML and XUML view – If one split State Machines without splitting associated data one is not doing OOA/OOD but rather SA/SD**

Source of Drawing : Statecharts Definition

Orthogonality – the biggest Statecharts Fallacy





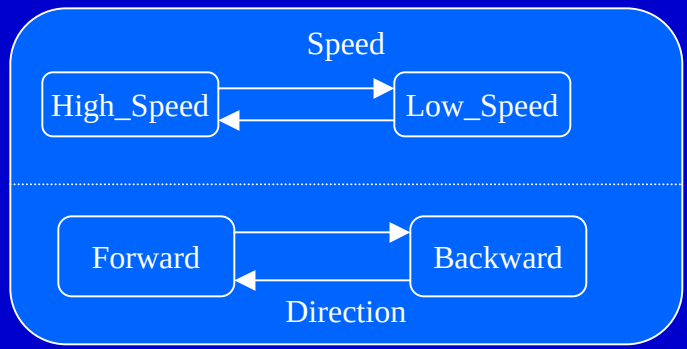
VeUML view – Concurrency and behavior are two independent system domains / aspects.

Example: States "A" and "D" belong to separate Classes. For instance: Bank's Procedures (application logic) is the same on Blade Servers and a Mainframe implementation.

StateSoft

Source of Drawing : Statecharts Definition

Orthogonality – the biggest Statecharts Fallacy

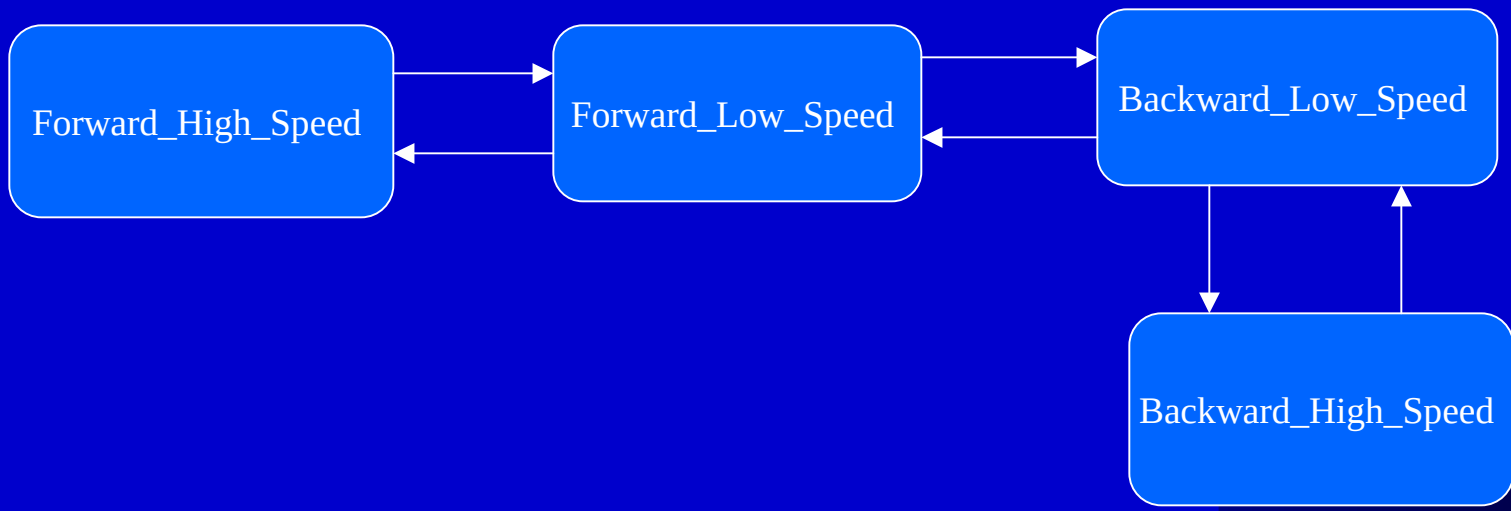


VCR Control: Statechart with AND-states - Speed and Direction.

One familiar with the tape control will realize that reversing the direction of the tape movement at the full speed creates a risk of breaking the tape. A race and nondeterminism are problems associated with this model

Source of Drawing : A text book example

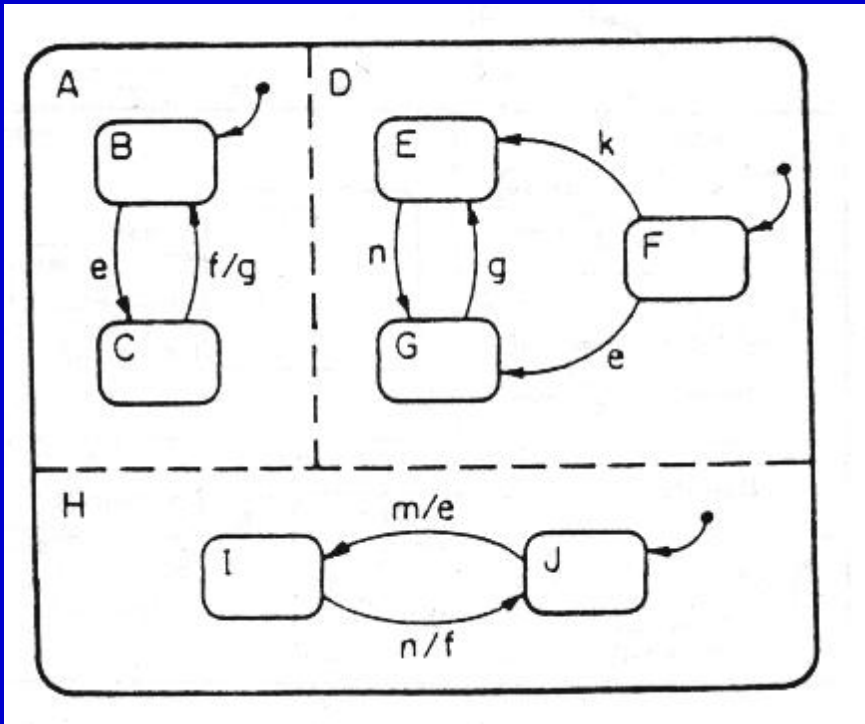
Simplicity of executable models – the biggest VeUML/Vcharts advantage



VCR Control without AND-states – Vchart

This is an Implementation Domain Model that can be translated to code automatically and will be free of the race and nondeterminism problem

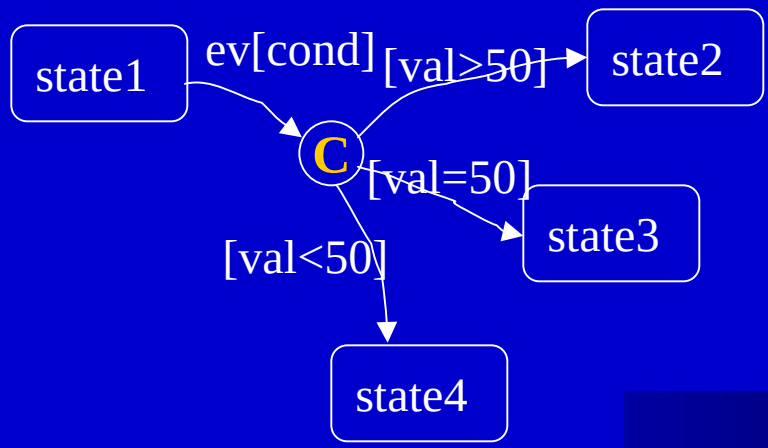
Statecharts Broadcast Communication



- **VeUML: Eliminated as a non-deterministic mechanism**

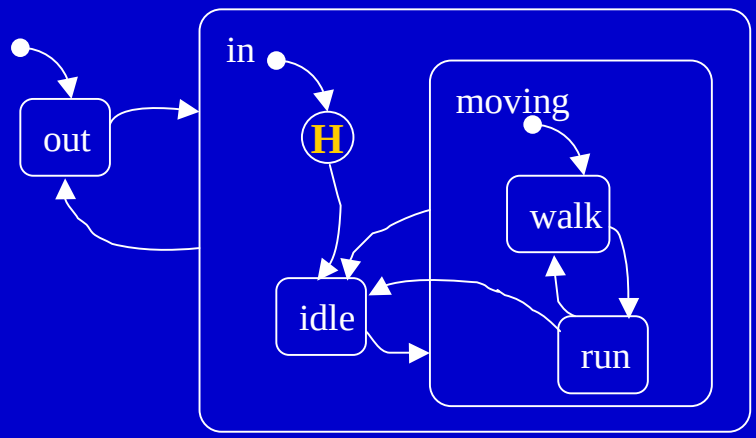
Source of Drawing : Statecharts Definition

Statecharts: Condition Connector “C”



- **VeUML view: ”C” is an unnecessary and confusing *Pseudostate***

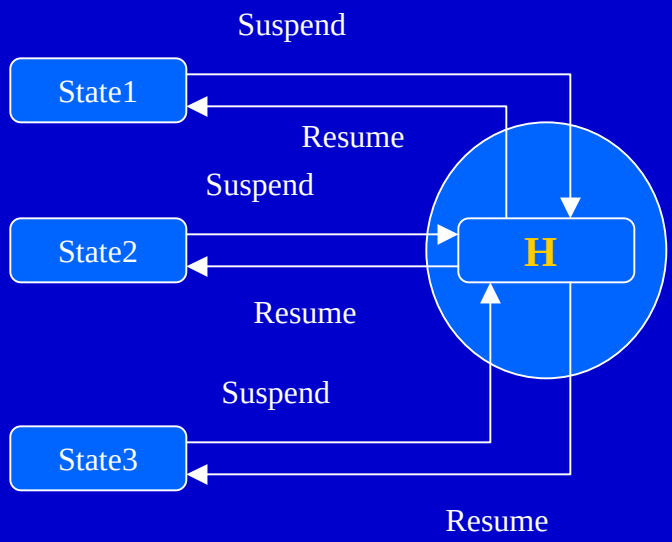
Statecharts: History Connector



VeUML view: History state reflects an external observer view of the behavior and NOT the implementer's view of behavior.

A leading source of design errors.

Statecharts: History Connector



In an actual systems implementation it is a very rare case that one transit from several states to common state (H) and than return to "most currently visited state" without performing some transition specific actions on transition.

History state had been eliminated in VeUML

State Specification in VeUML

```
State1

entry / Action1, Action2;

input: / event1 & event2  | event3 /Action3,  Action4;

exit /  Action5,  Action6;
```

Simplicity of the State Specification in VeUML greatly reduces chances for a modeling error.

VeUML Virtual Event Specification

- **Virtual Events have a form of pure names**

- **UML Guard and Event are translated to two Virtual Events in VeUML**

  Note: Comparators often generate Virtual Events

  oil_temp_exceeded & engine_running / Turn_fan1_On

Condition Specification

- Expression in the form of the &(and)'s and |(or)'s of Virtual Events
- Parameter values(true/false) are also translated to Virtual Events
- Condition Specification is the same for transitions as well as input actions within a state

VeUML Action Specification

- VeUML Actions have a form of pure names only e.g Action1, Action2, Action3;
- Entry and Exit actions are unconditional
- Transition Actions are conditional
- Input Actions (within a state) are conditional as well

Condition / Actions Examples

UML

- *Tm*(*en*(a), 3)[!overweight & doorClosed]/*tr*!(movable)
- reached[levelGap<30]/openDoor

VeUML

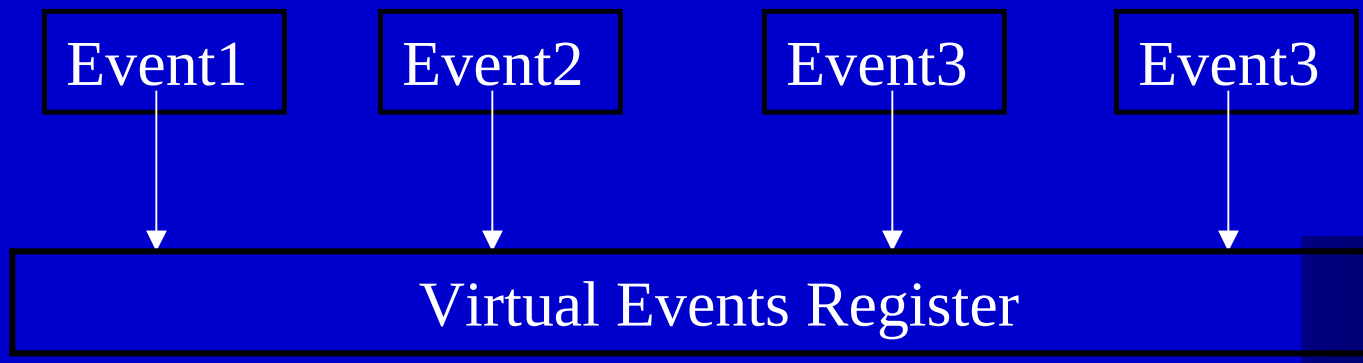
- *Temp_in_range* & not_overweight & doorClosed / *do_Not_move*
- Level_reached & levelGap_less_than_minimum / openDoor

Class Behavior Representation in VeUML

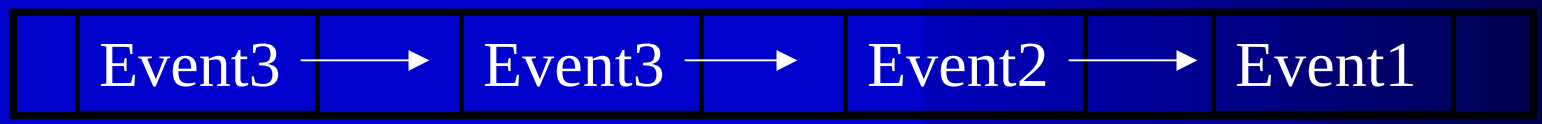
In VeUML all Statefull and Stateless Classes have behavior represented as Virtual Finite State Machines.

Events Processing

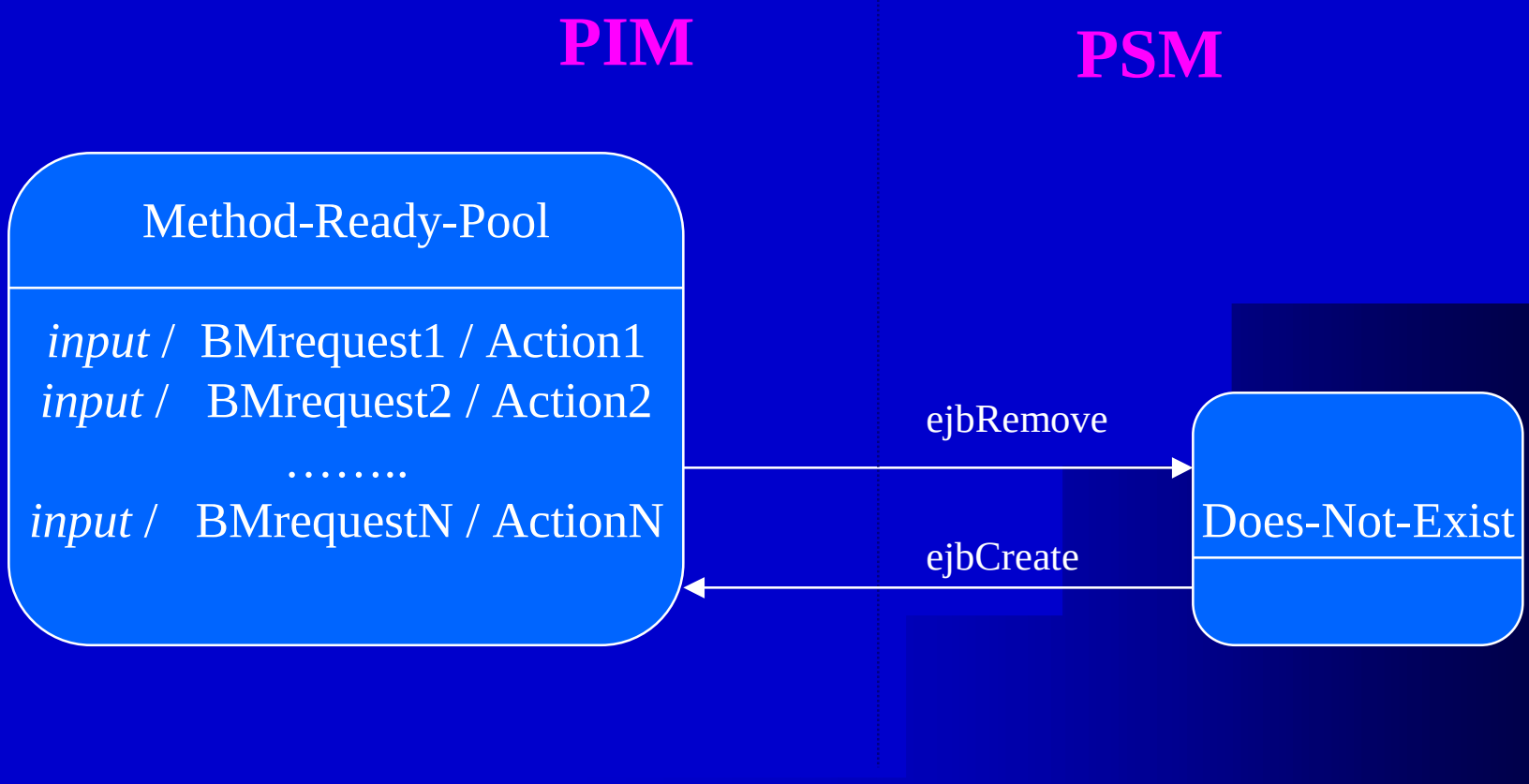
VeUML Parallel Events Processing



UML Serial Events Processing

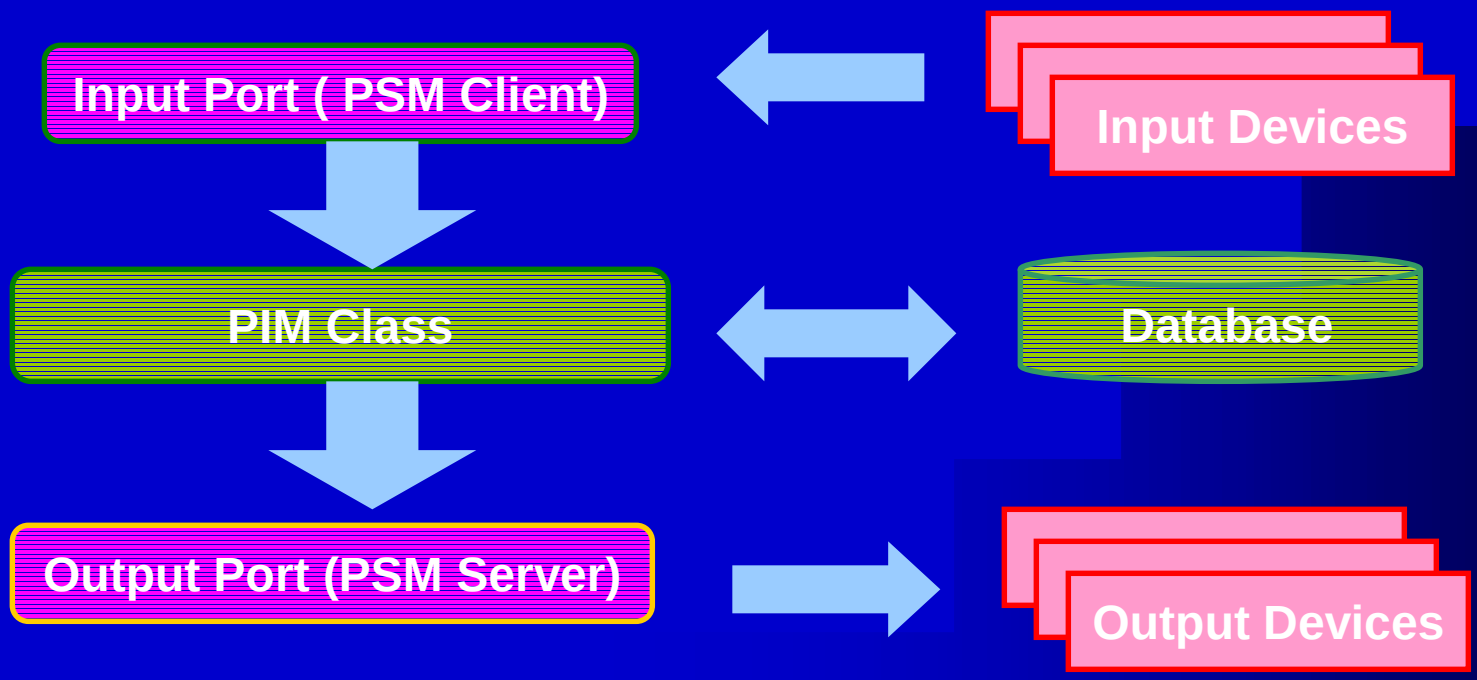


EJB Stateless Session Bean

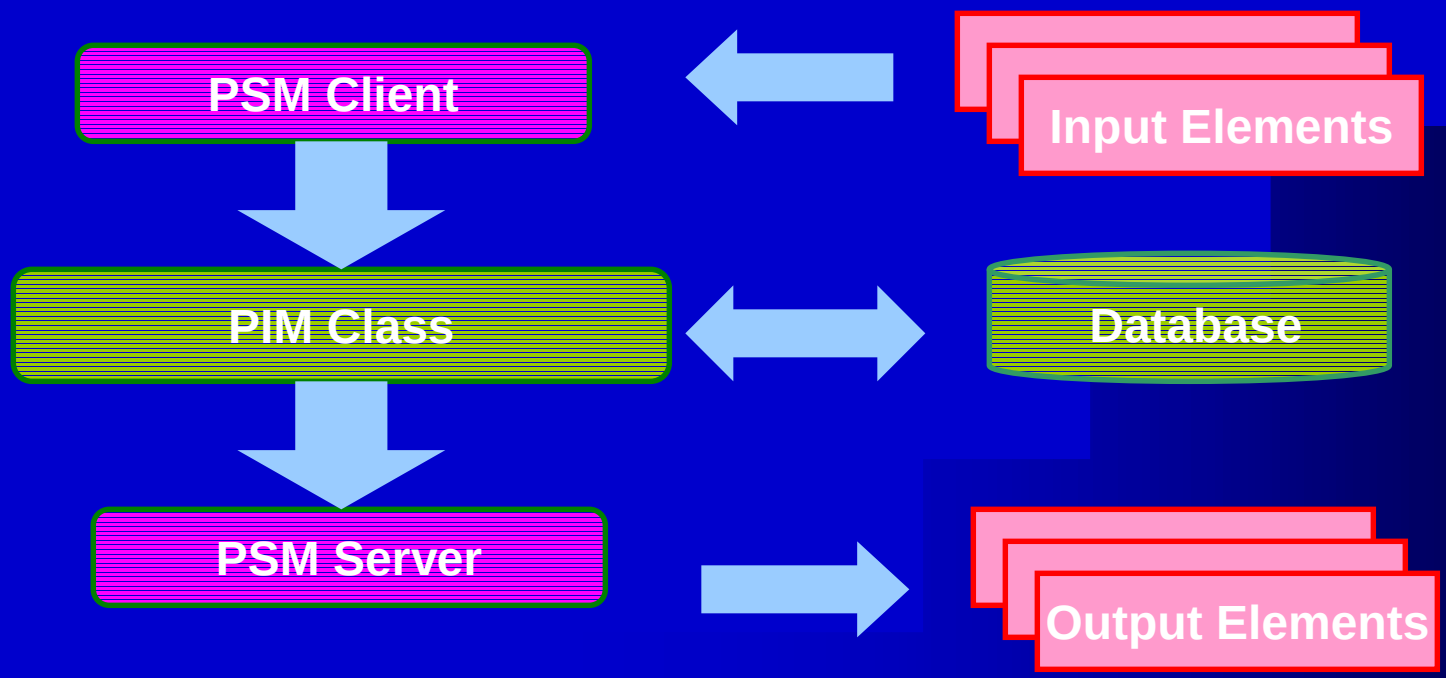


VeUML Notation (Vchart)

VeUML PIM Class Environment in Real Time Application



VeUML PIM Class Environment in Enterprise Application



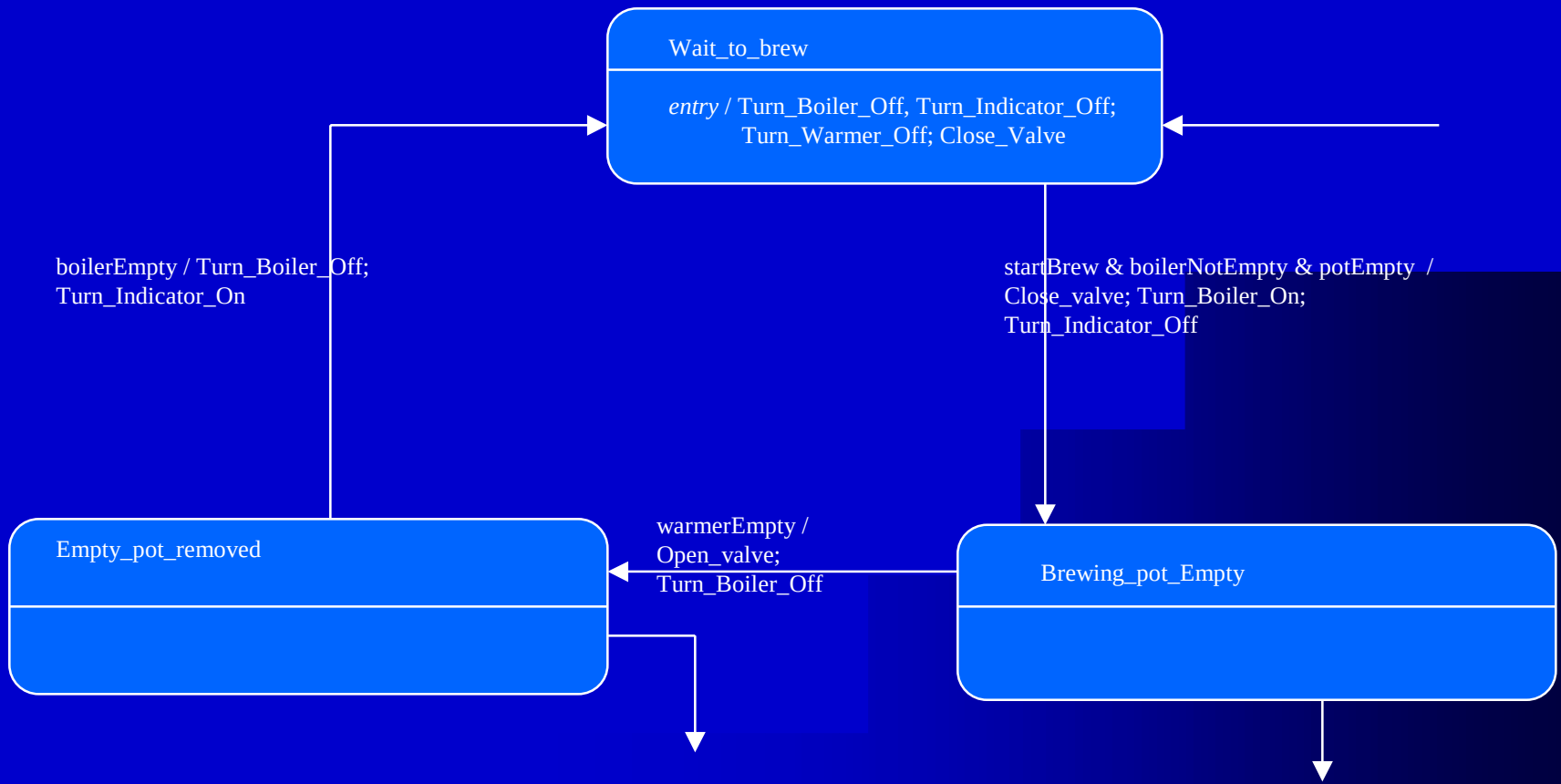
Conversion between Behavior Models in VeUML

VeUML extensions: VeSTD, VeXML, VeMSC,VeSDL:

* Immediate automatic representation conversion always possible:
  VeSTD <-> VeMSC <-> VeSDL <-> VeXML

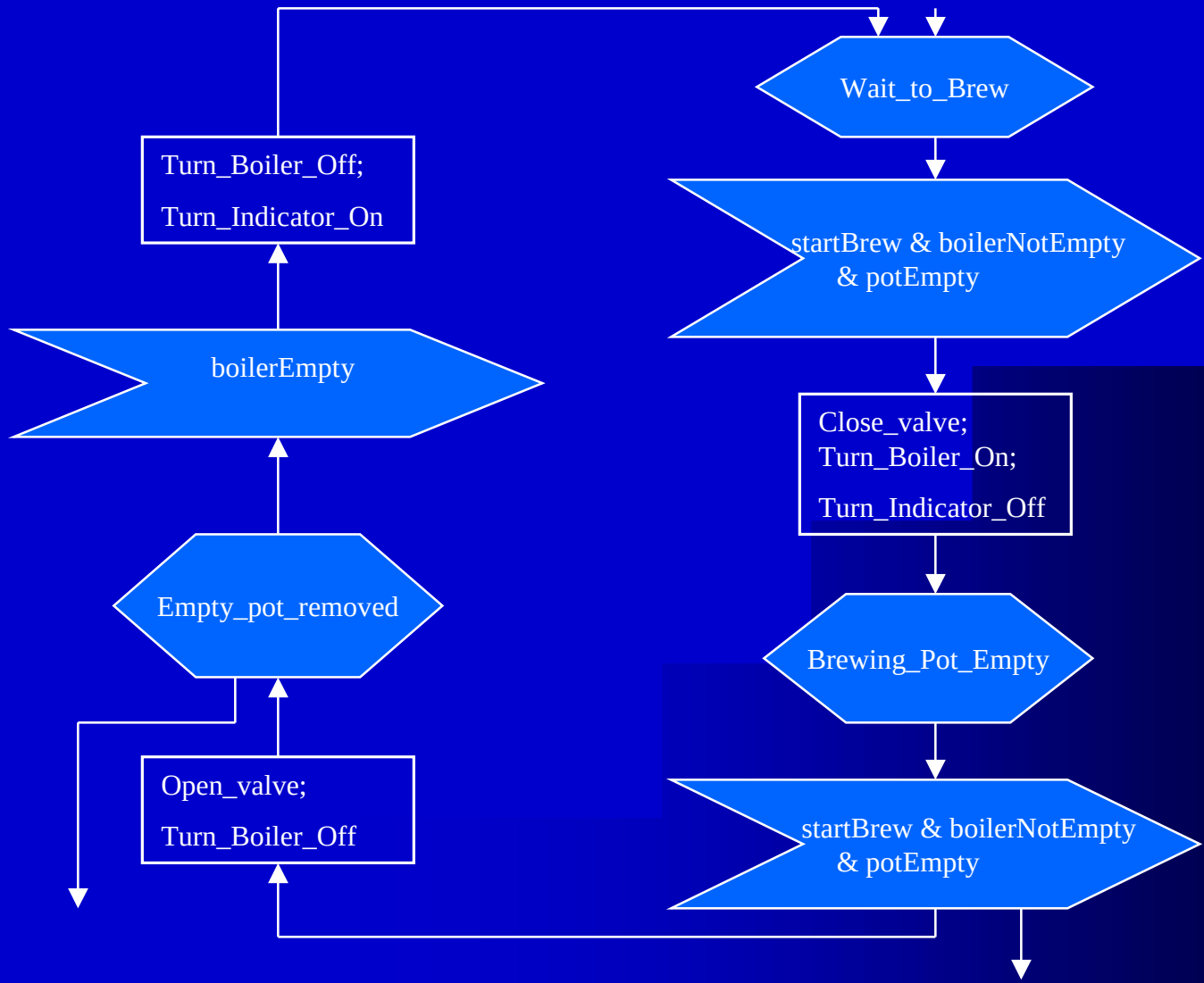
* **According to the OOA/OOD consistency principle Architect/Designer/Implementer view of a system should be the same. In VeUML the MDA behavioral model is the only target executable representation Architect/Designer/Implementer work with.**

VeSTD Example – Partial diagram



This Coffeemaker example is based in part on the book by
Robert C. Martin, "Object-Oriented C++ Applications Using The Booch Method", Prentice Hall, 1995.

VeSDL Example - Partial Diagram



StateSoft

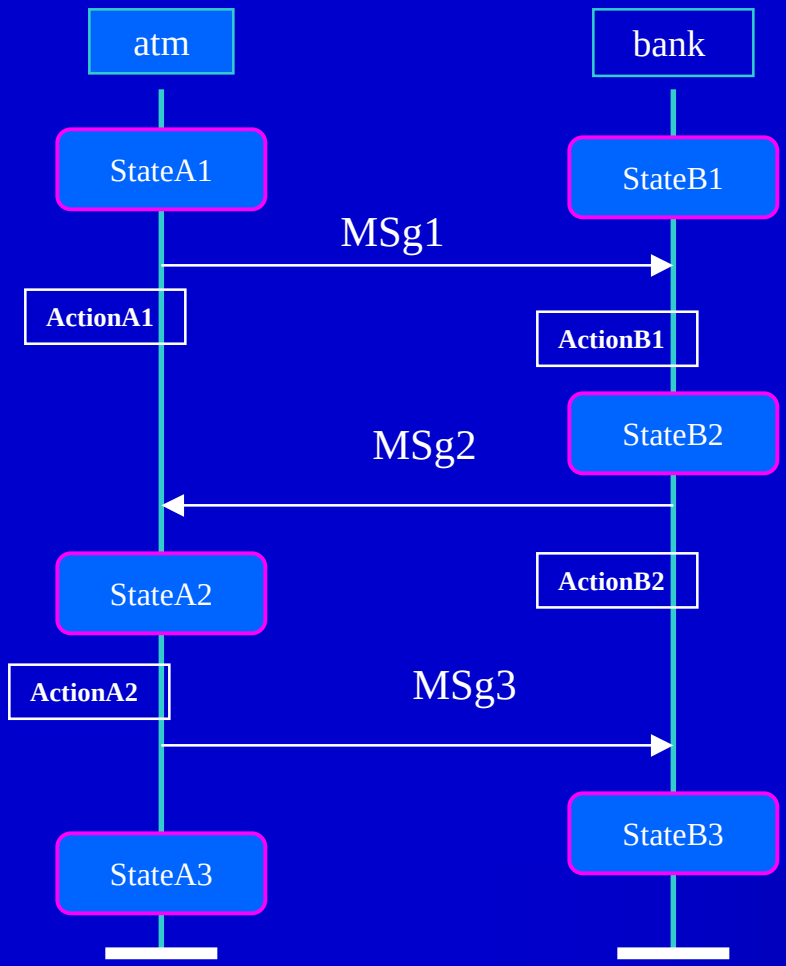
VeXML Example - Partial Listing

```
<?xml version="2.0"?>
<!-- (c) 2002 StateSoft, Inc.  All rights reserved. -->
<XML>
  <STATE>
    <TRANSITION>
      <CONDITION> "potNotEmpty / "
        <ACTION>
          "Close_valve; Turn_Boiler_On; Turn_warmer_On"
        </ACTION>
            <NEXTSTATE>
              "Empty_Pot_removed"
            </NEXTSTATE>

      </CONDITION>
    <TRANSITION>
  </STATE>
</XML>
```

StateSoft

VeUML Classes Behavior Generation from System Model MSC



In VeUML target executable model is synthesize automatically from elementary representations.

Other approaches often use elementary representation for high level simulation only.

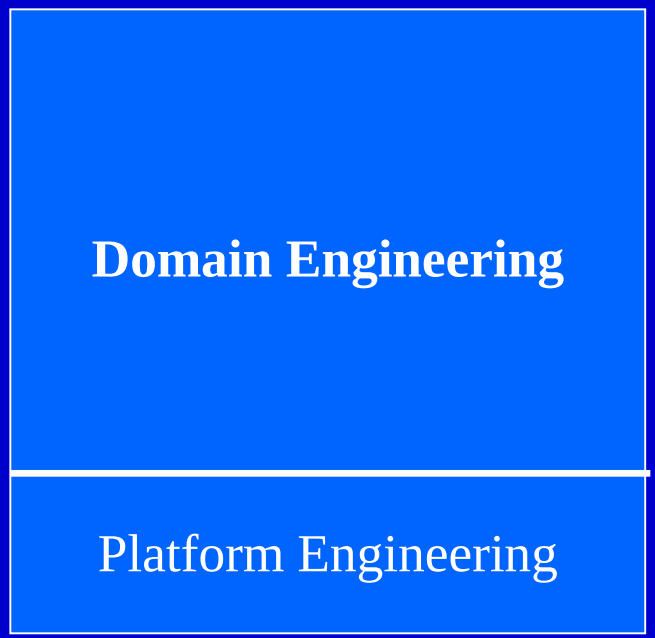
Project Experience

Projects Results

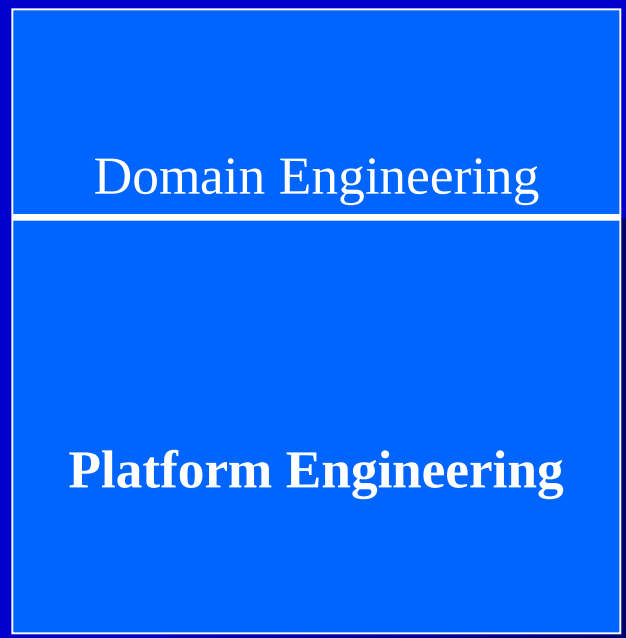
Problem	VeUML Solution
Complexity	*Separation of concerns* – data from behavior; events from data, behavior from concurrency, actions from data etc..
Cost	*Automatic executable generation*, automatic validation and simplified test. Resulted savings 45% *(AT&T/Lucent Data)*
Portability to different platforms	*Service/Business Logic represents Interlingua* - model and automatically generated executable are platform independent
Maintenance	*Substantially reduced due to single Domain/Concern Models*
Quality	*Late defects reduction by* 40% by exhaustive validation and early detection *(AT&T/Lucent Data)*
Performance	*Optimized Predictable* performance of VFSM executor

VeUML Advantage in Domain Engineering

MDA/VeUML - Models are Programming Language and Metalanguage Independent

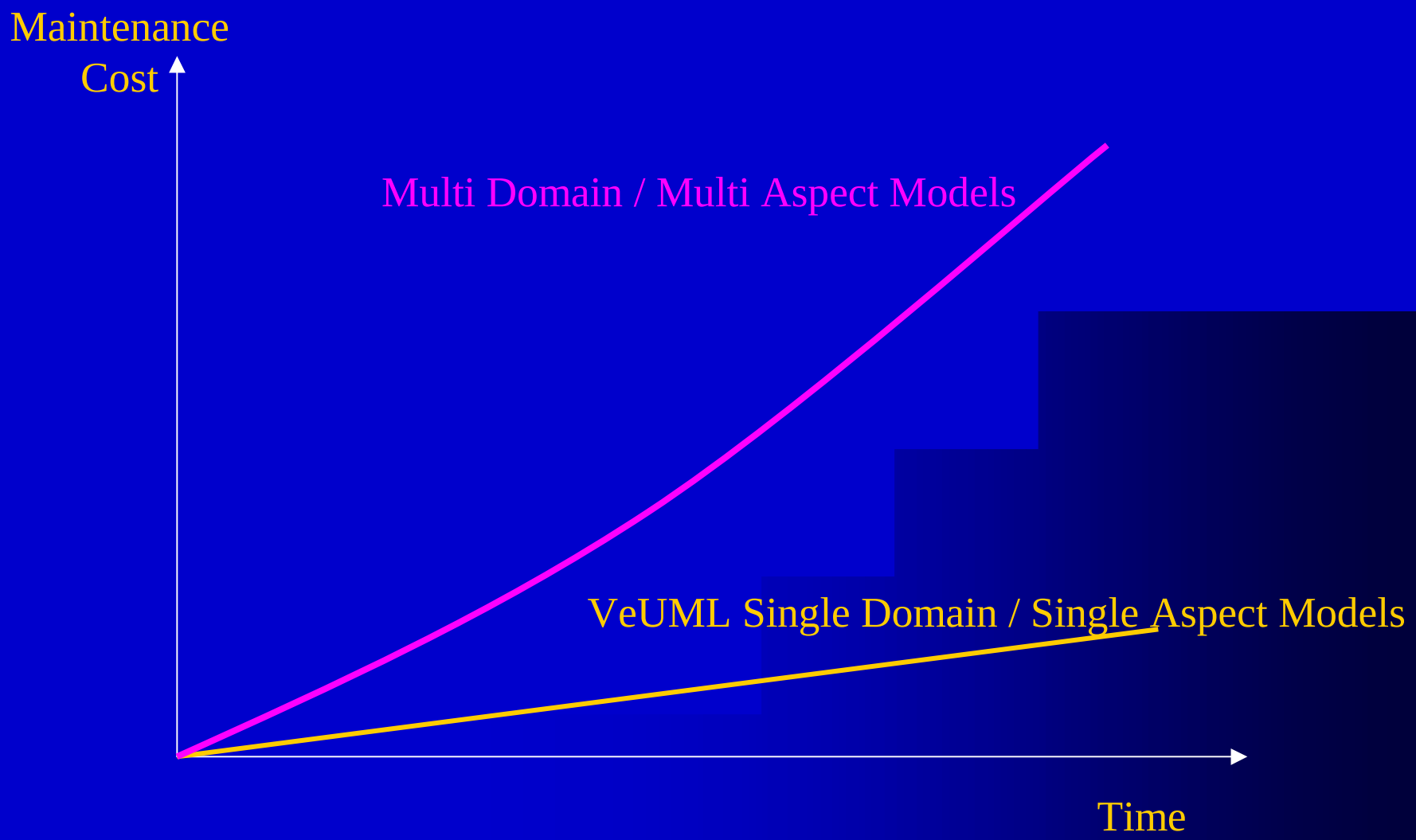


Alternative - Models are Programming Language or Metalanguage Dependent



- **Domain analysis is "the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain" [CMU/SEI-90-TR-21].**

VeUML - Maintenance Cost Reduction



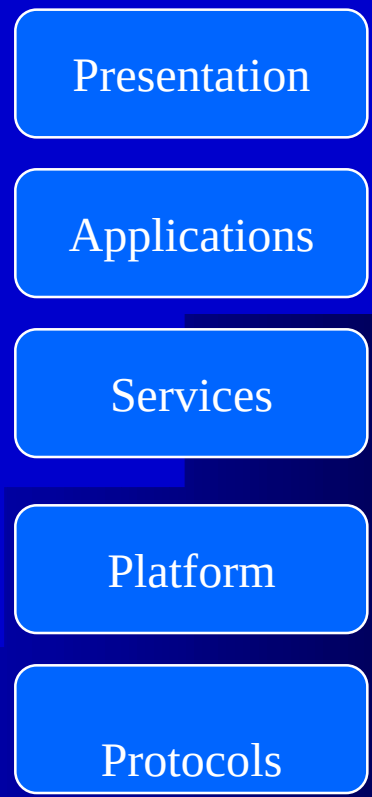
StateSoft

Independent Iteration in Logical (PIM) and Physical (PSM) Architectures

Current Systems Architecture

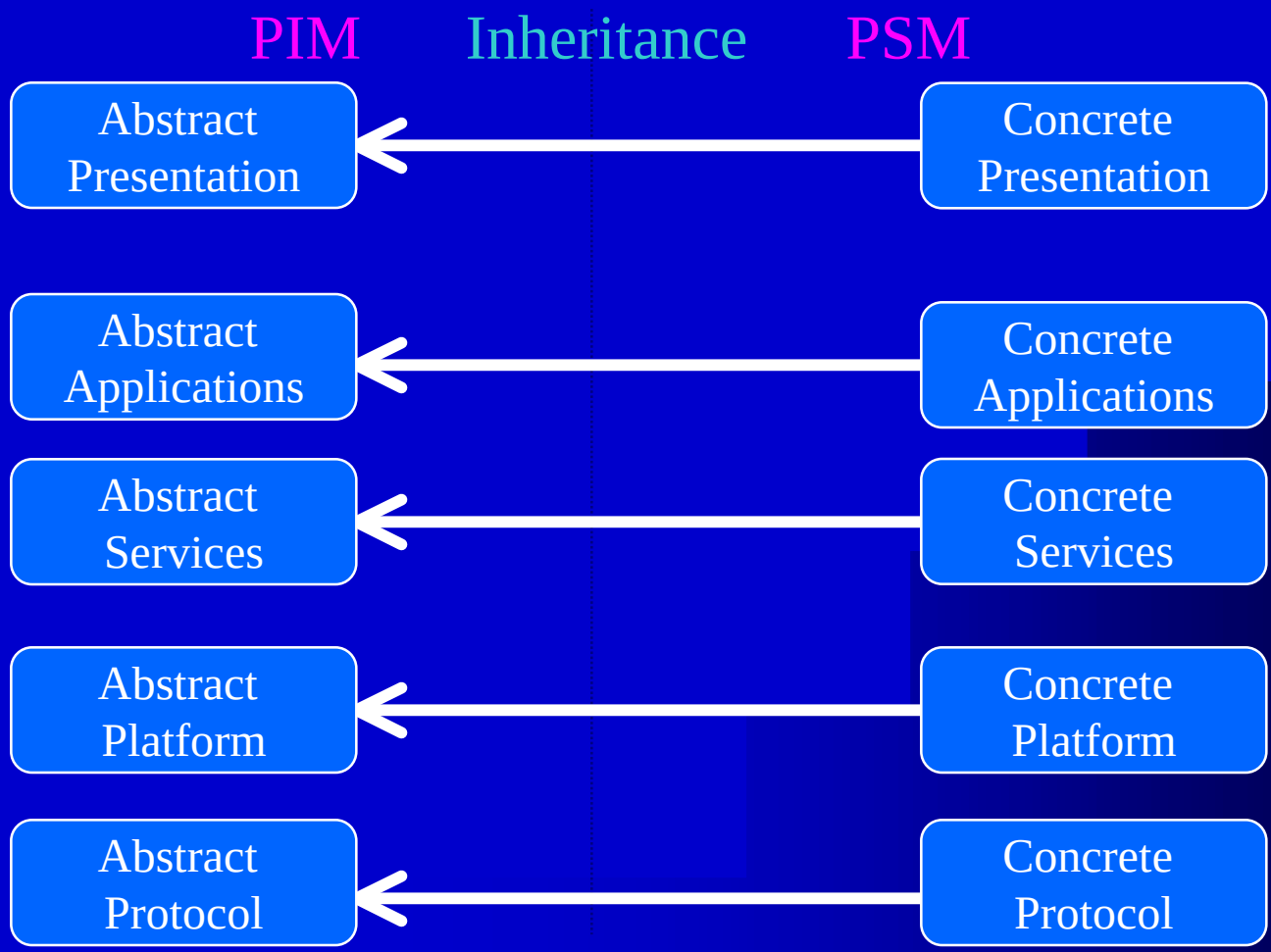
Platform Independent Model (PIM)

Platform Specific Model (PSM)



Limited potential for Reuse, Automatic Validation and Features Interaction Traceability

VeUML- Based Systems Architectures



Optimized for Reuse and an Automatic Validation

StateSoft

VeUML Modeling Summary

- Automatically generated executable
- Automatic validation
- Simpler models for complex systems
- Complex systems maintenance cost reduction
- Reuse of models