



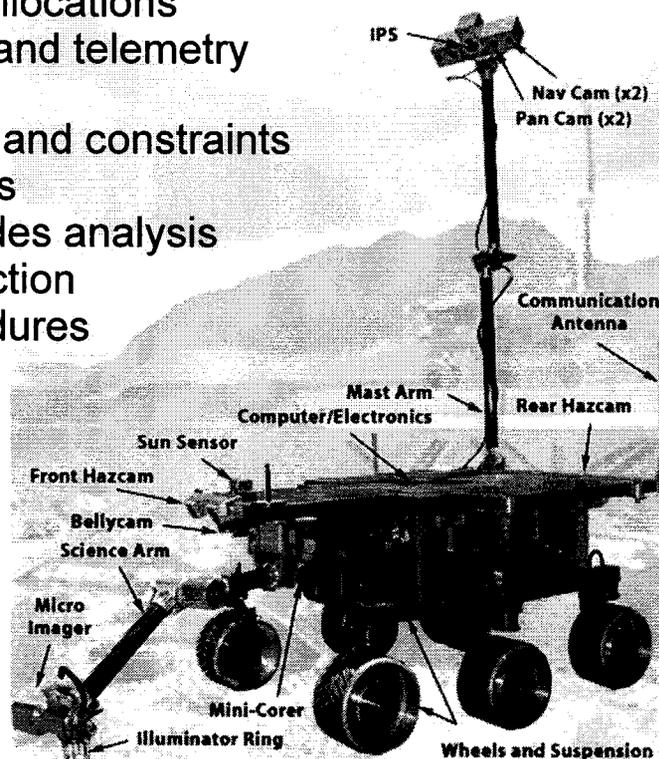
---

# The Mission Data System's Software Architecture Framework

Dr Nicolas Rouquette  
Principal Software Architect  
Jet Propulsion Laboratory  
<mailto:nicolas.rouquette@jpl.nasa.gov>

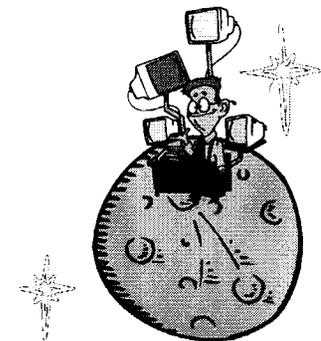
- Systems engineering is *outward* looking

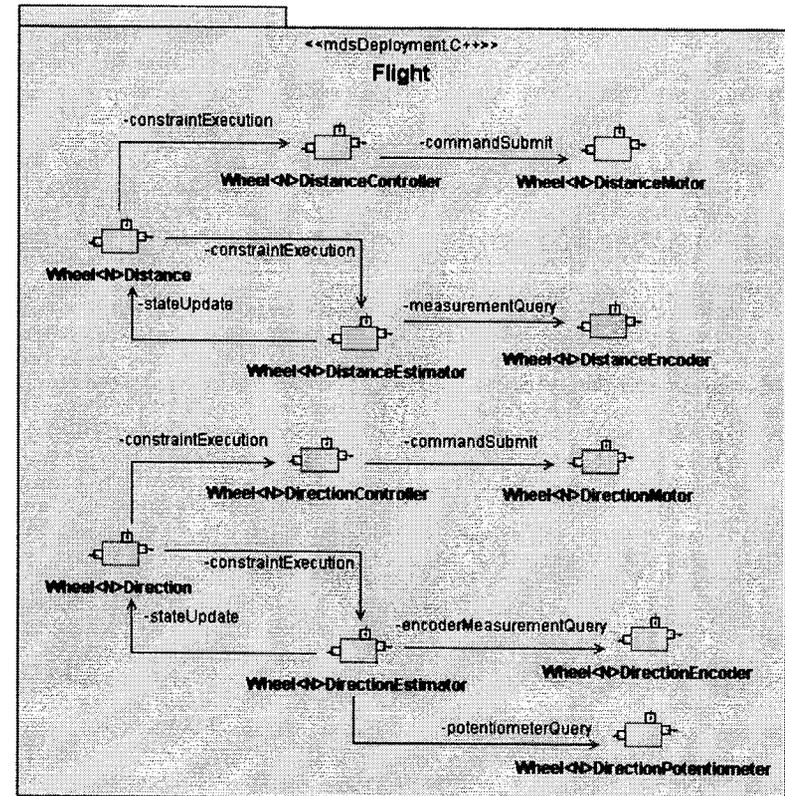
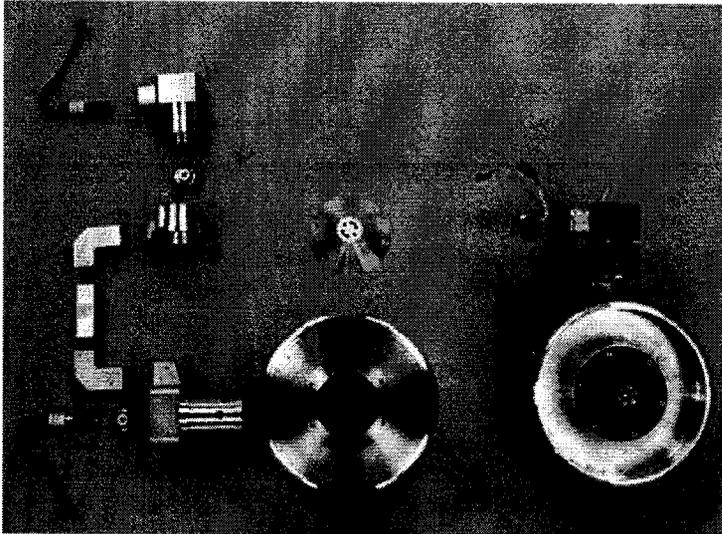
- Mission scenarios
- Functional decomposition
- System analysis
- Performance requirements
- Resource allocations
- Command and telemetry dictionaries
- Flight rules and constraints
- Control laws
- Failure modes analysis
- Fault protection
- Test procedures



- Software engineering is *inward* looking

- Languages, libraries, operating systems...
- Concurrent threads, processes, memory management...
- Real time execution
- Patterns, abstractions, general algorithms...
- Data representation, serialization...
- Interprocess communication
- Deadlocks, access violations, exceptions...



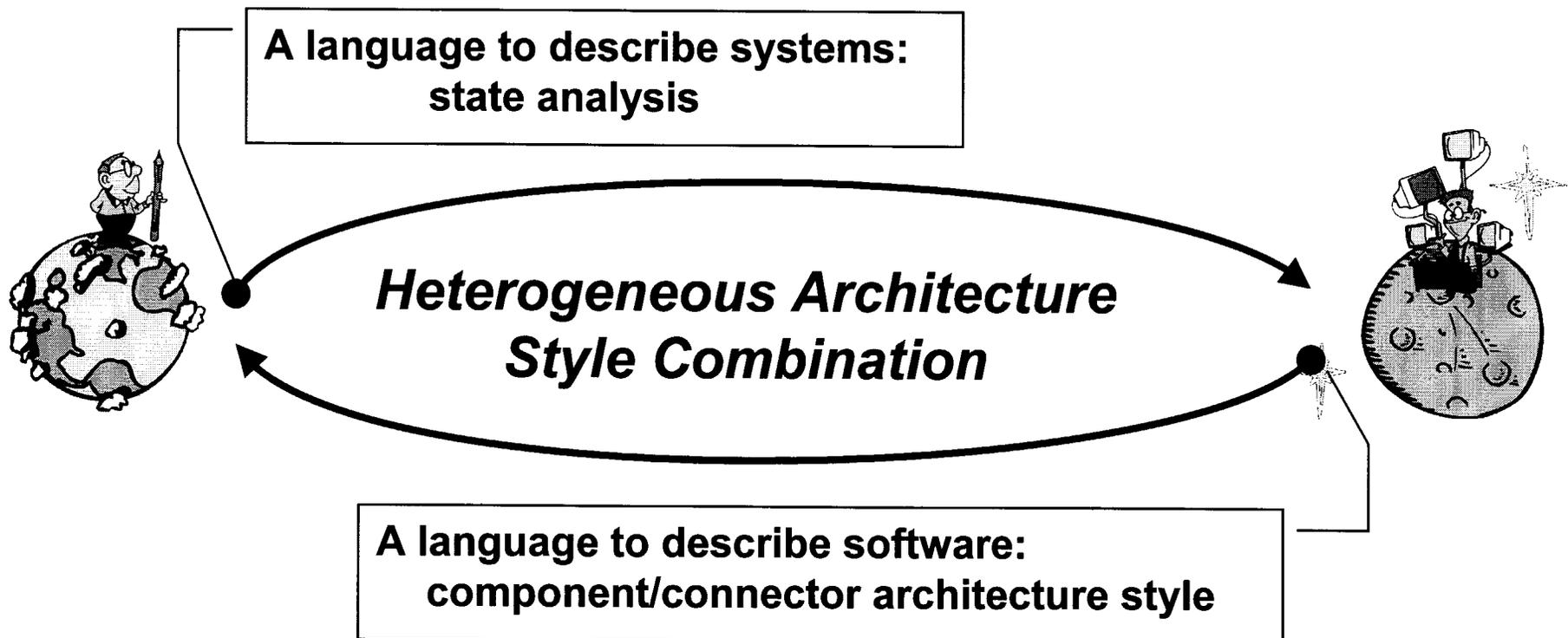


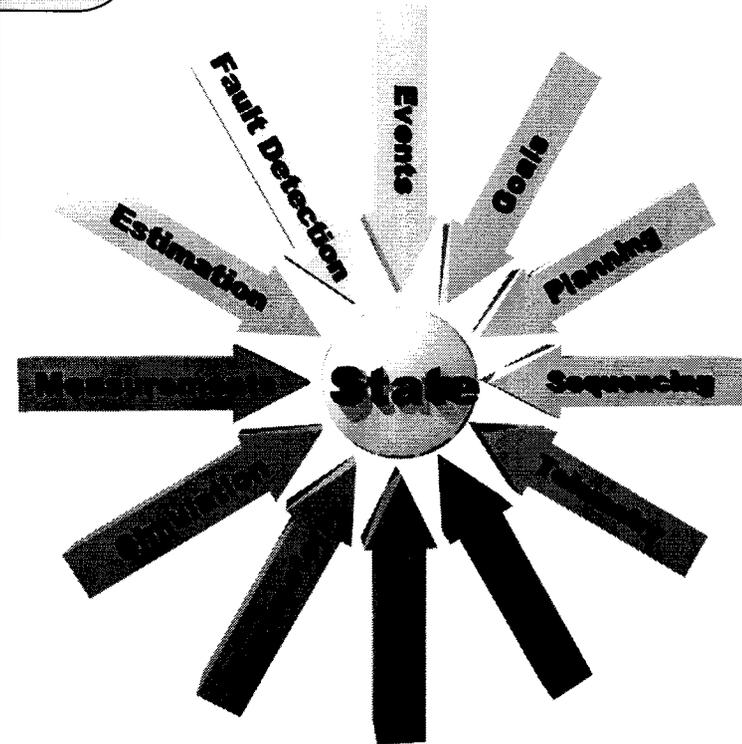
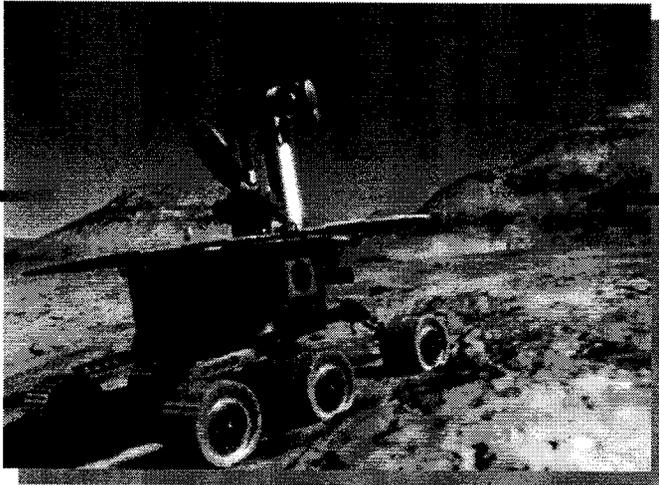
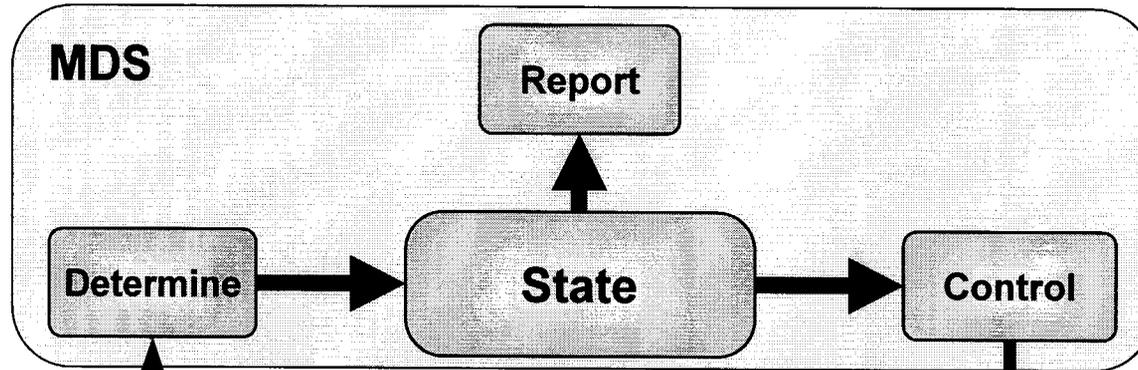
The principle risk to mission success is **miscommunication**

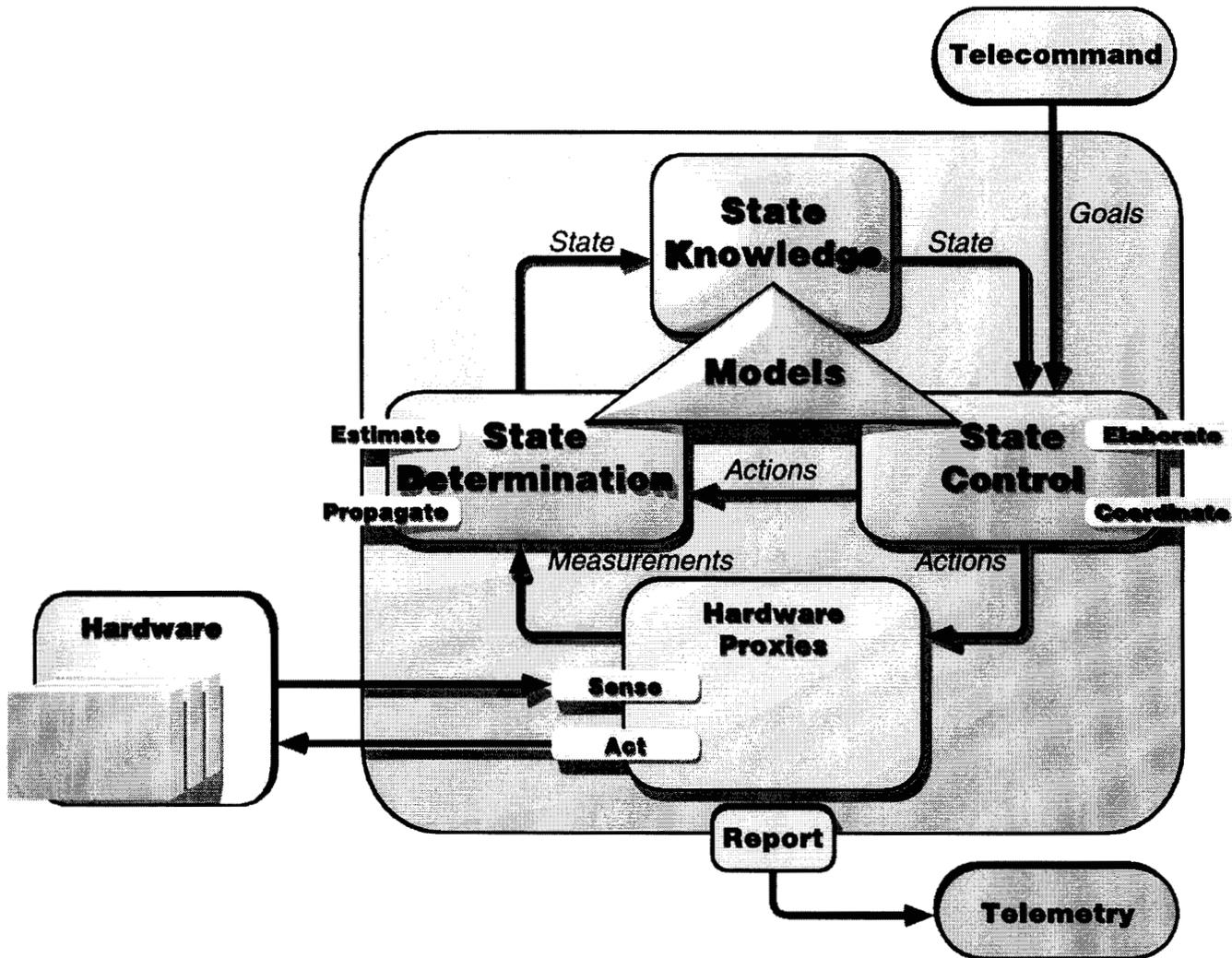
–What systems engineers want can be **hard to express**

–What software engineers build can be **hard to understand**

Abstract system and software engineering concerns at the level of architecture where they can be related & compared easily





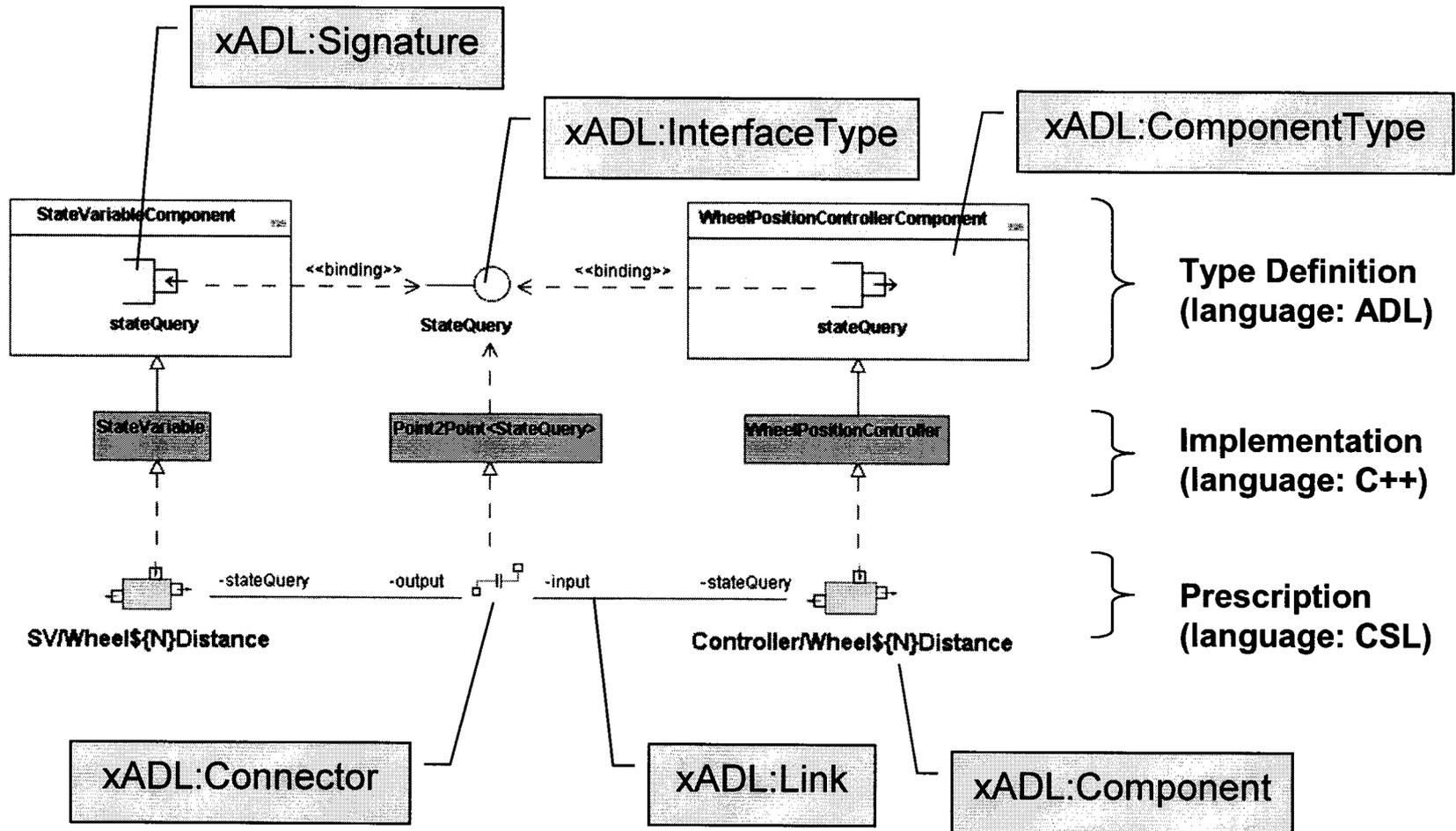




# The Component/Connector Architecture Style



- 1<sup>st</sup> Heritage: xArch ( <http://www.isr.uci.edu/projects/xarch/> )
  - DARPA-sponsored research
    - @ UCI (Dashofy, van der Hoek)
    - @ CMU (Garlan, Schmerl)
  - XML schemas for describing architecture instances
    - Key elements: Components, Connectors, Interfaces, Links
- 2<sup>nd</sup> Heritage: xADL2.0 ( <http://www.isr.uci.edu/projects/xarchuci/> )
  - Separates several kinds of architectures
    - At the type level (things that can be built)
    - At the instance level
      - Prescription (things that shall be built)
      - Description (things that are built)
- MDS
  - Extensions of xADL (parametric elements, prescription patterns)
  - A C++ runtime system (bootstrapped, extensible)
  - Architecture profiling (model-based transformation systems)

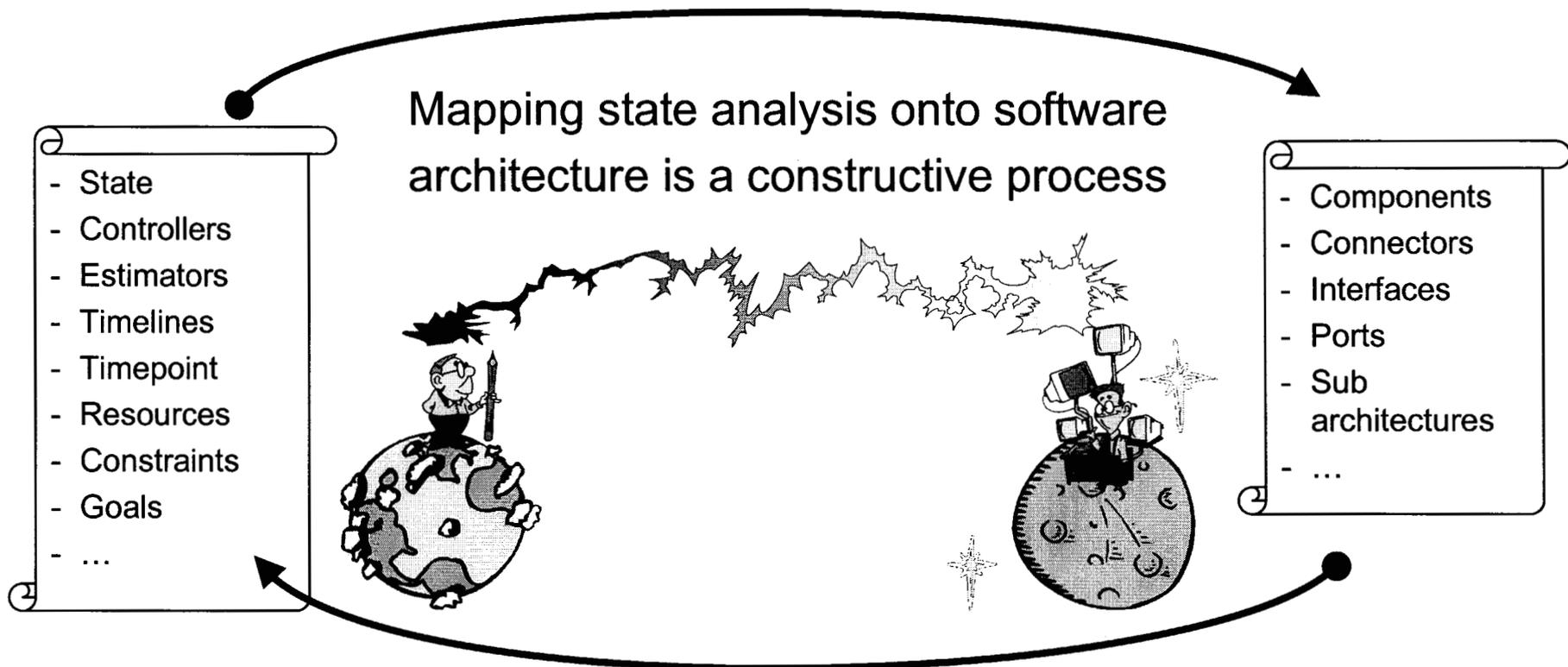




# MDS' focus on Software Architecture



- Central motivation
  - Bridging the gap between system and software engineering cultures
    - At the core, it is a communication mismatch problem
- Two syndromes
  - Resolving the communication mismatch at the software design level
  - “Class-A” flight software culture relies heavily on wizards and gurus
- Two approaches
  - Heterogeneous architecture style combination
    - State architecture style
      - Communicate about *any* system aspect in a consistent language
    - Extensible component/connector architecture style
      - Communicate about software aspects at a higher level of abstraction
  - Architecture Hoisting
    - Elevating software design decisions up to a higher level of visibility
      - The architectural level is an abstraction that has broad visibility
    - Increasing software dependability & reliability
      - Shift risk management from the implementation level (low visibility) to the architecture level (high visibility)



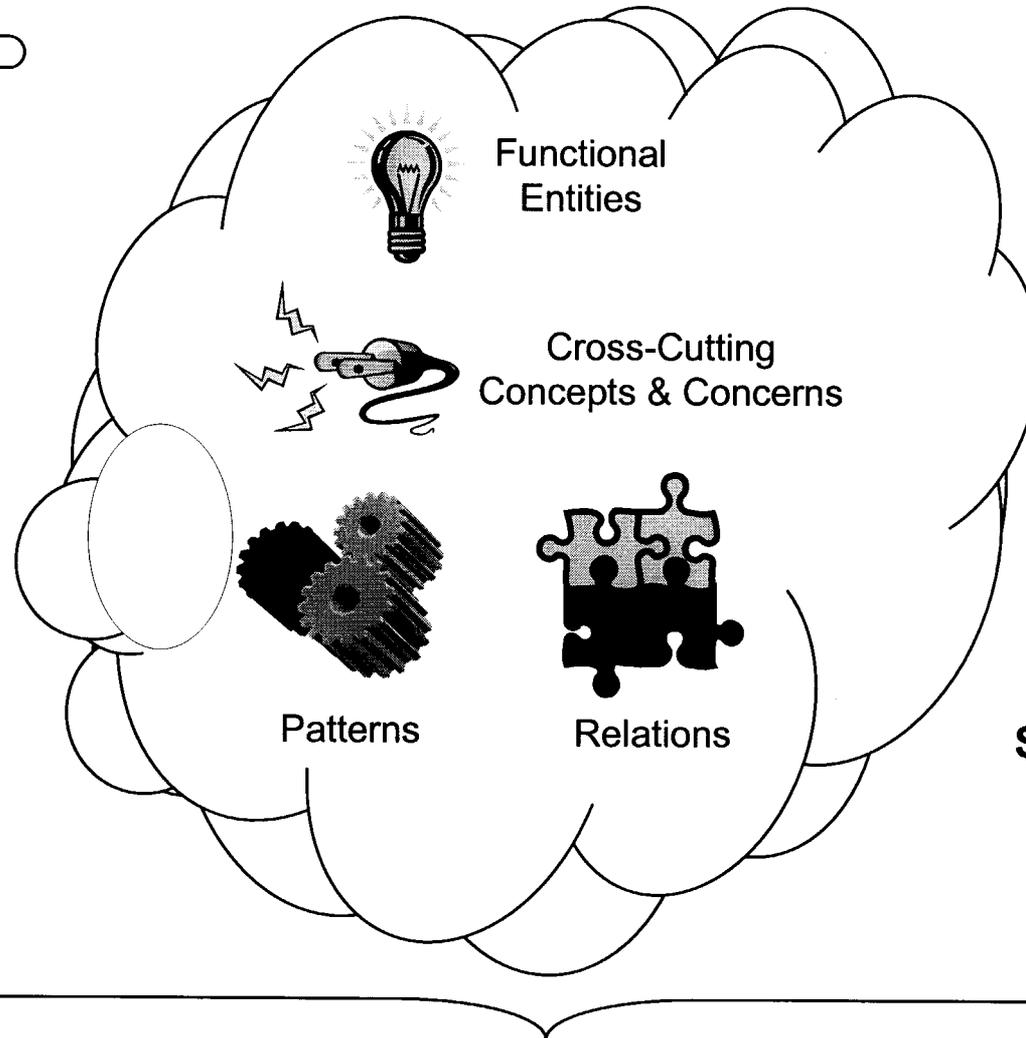
System engineers need feedback from the software

- distinguish between prescription => the system as required  
& description => the system as built
- compare predicted vs. actual properties (-ilities, performance, etc...)

# Heterogeneous Style Combination: Mapping State Elements into Software Architecture

- State
- Controllers
- Estimators
- Timelines
- Timepoint
- Resources
- Constraints
- Goals
- ...

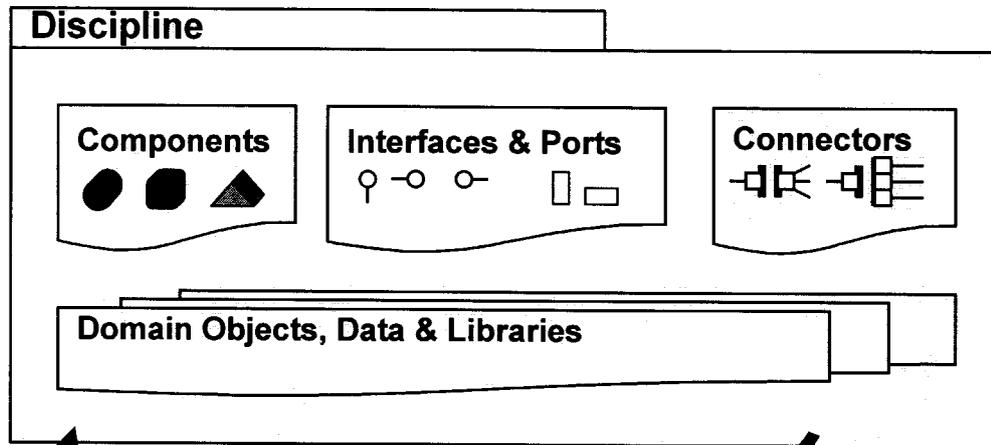
**Elements of  
State Analysis**



- Components
- Connectors
- Interfaces
- Ports
- Sub architectures
- ...

**Elements of  
Software Architecture**

**Framework Design & Architecture  
For Product-Line Engineering across Discipline Areas**



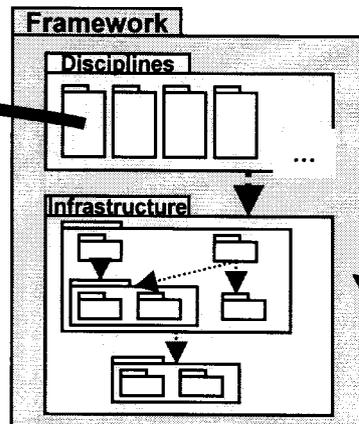
## Cross-Cutting Concerns

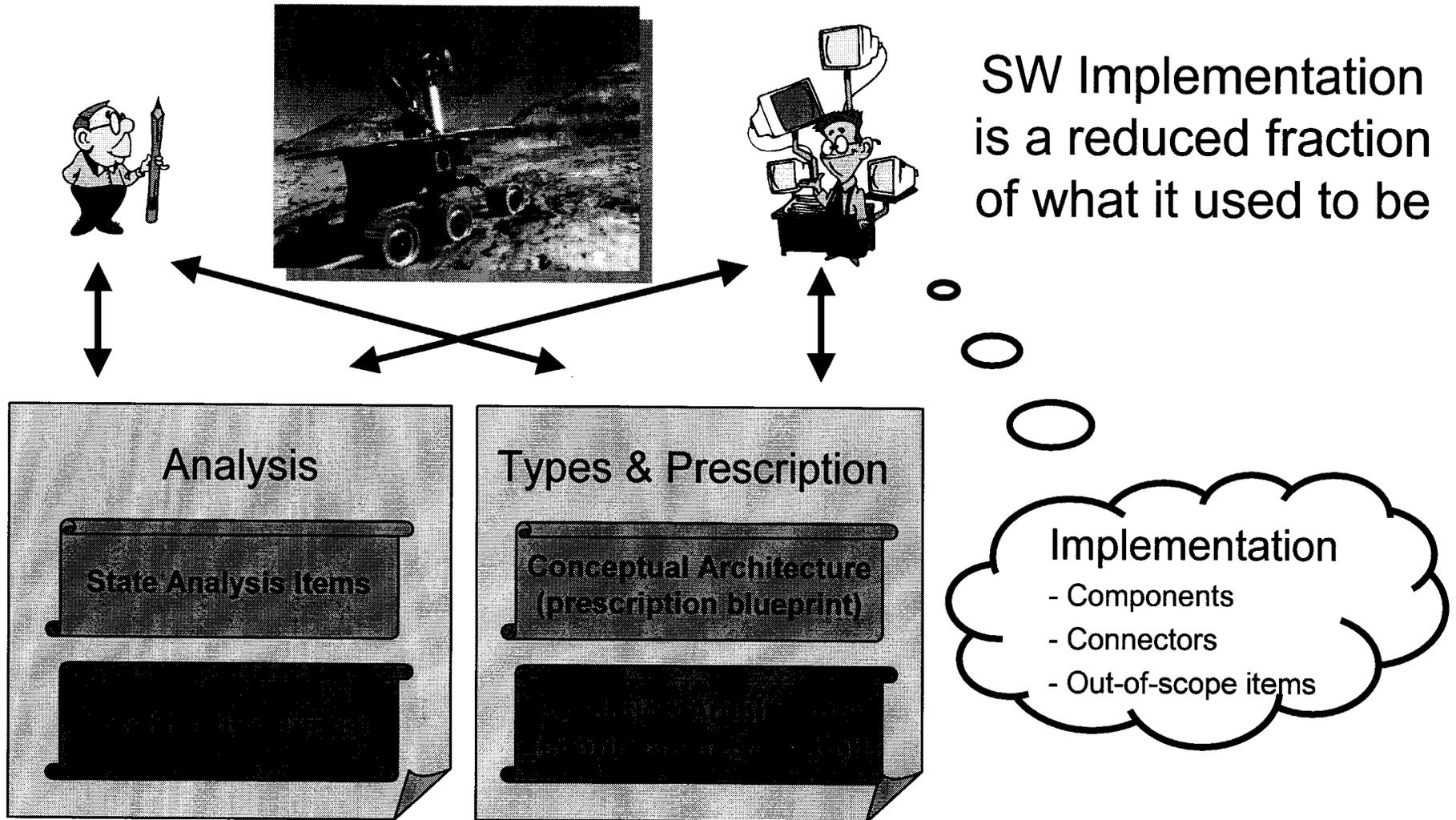
- Initialization
- Named Object Registry
- Embedded Web Client/Server
- Component Architecture
- Component Scheduler
- Virtual Clock
- Time Management
- Event Logging Facility
- MDS Directory Access Protocol
- Data Management & Transport
- ...

Each discipline provides reusable design architectural elements...

... disciplines face similar issues ...

... that have solutions in the MDS framework



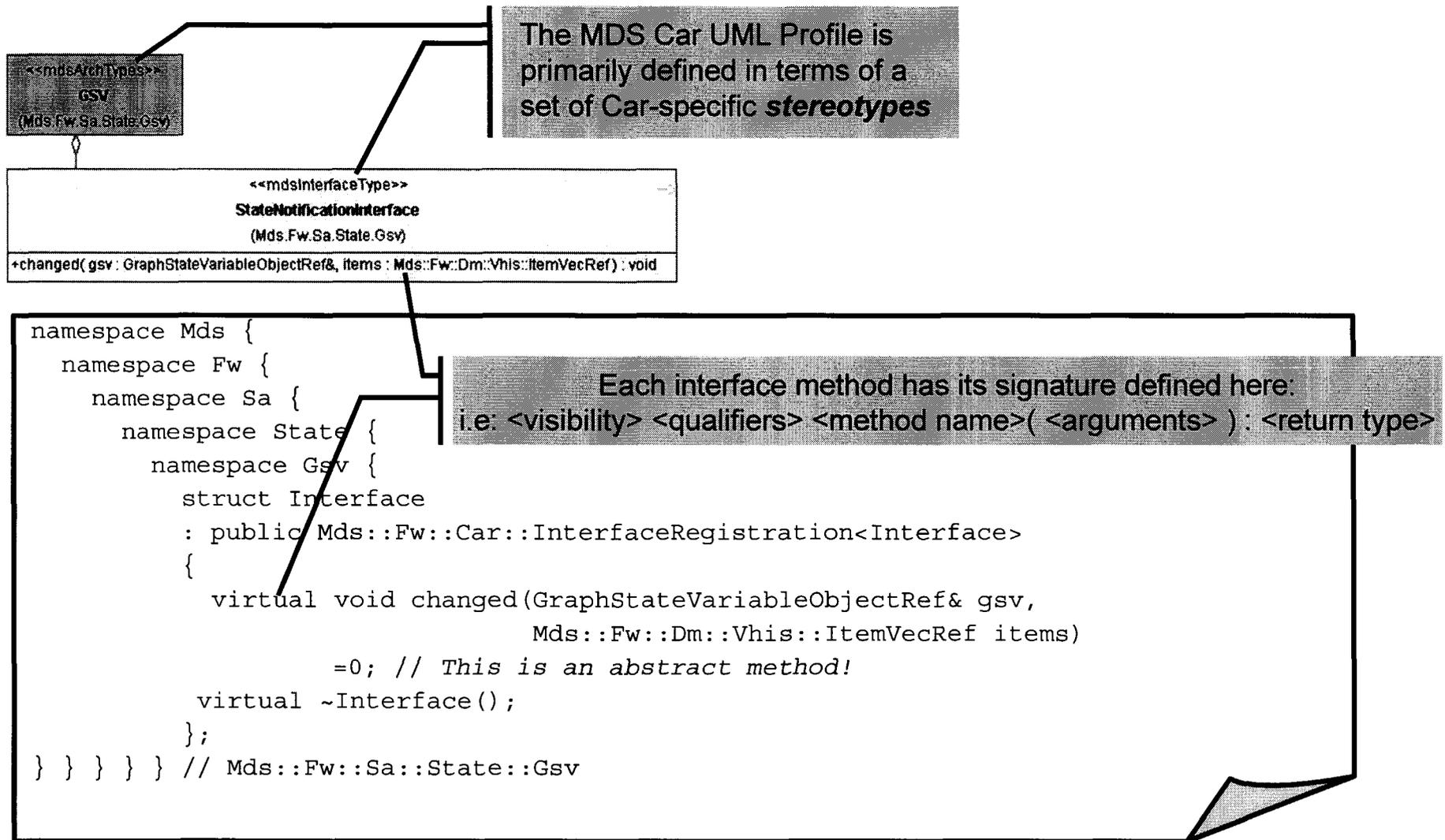


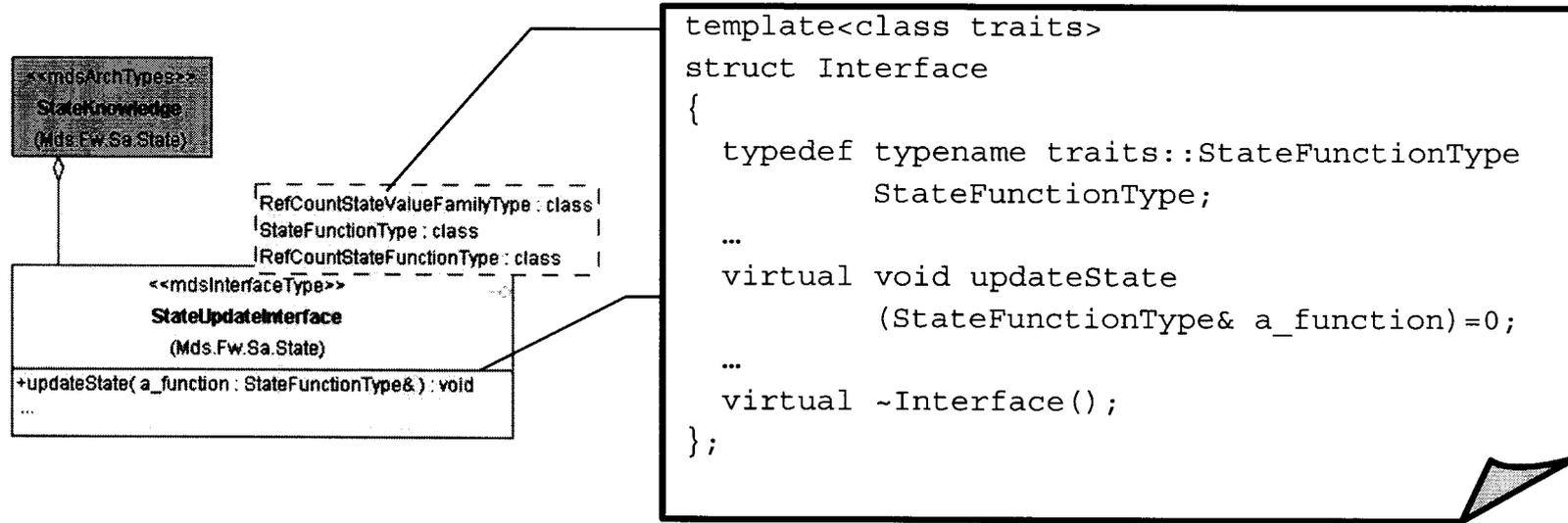


# Architecture Type Modeling -- InterfaceTypes



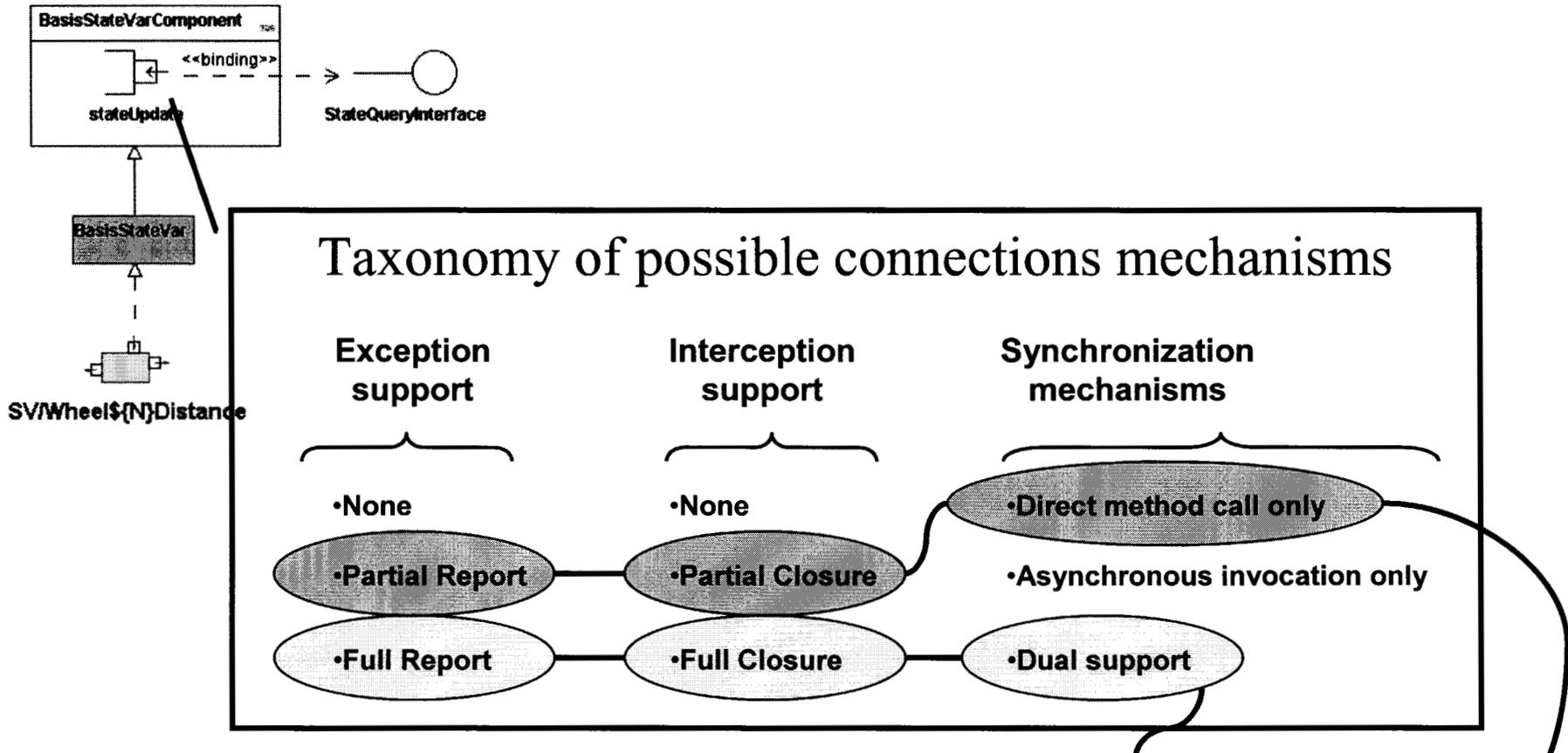
- 4 extensions
  - Interface definition with a set of methods each defining:
    - A return type
    - A set of method arguments (type/name/qualifiers)
    - A set of method qualifiers (e.g., constness)
  - Parametric modeling => maps to C++ templates
  - Interface Inheritance => maps to C++ interface inheritance
  - Communication Profiles
    - Idea borrowed from Mehta *et al's* connector taxonomy (ICSE 2000)
    - Each profile is a combination of policies
    - 3 policies defined (so far)
      - Synchronization asynchronous, synchronous, both
      - Interceptors none, partial (method name), full (closure)
      - Exception none, partial (method name), full (closure)
    - Optimizations
      - At code generation time => using profile-based transformations
      - At compile time => using policy-based class design





- An unfortunate C++ bias of UML
  - parametric data types modeled after C++ template classes
  - each trait parameter yields one template parameter
- CAR
  - *indirect* parametric data types
  - a single template parameter with one sub-type for each trait parameter





- A profile is a set of policy combinations, e.g: Validation Development
- The runtime system enforces mechanism compatibility across interfaces
- The generated code is optimized for the policies in effect

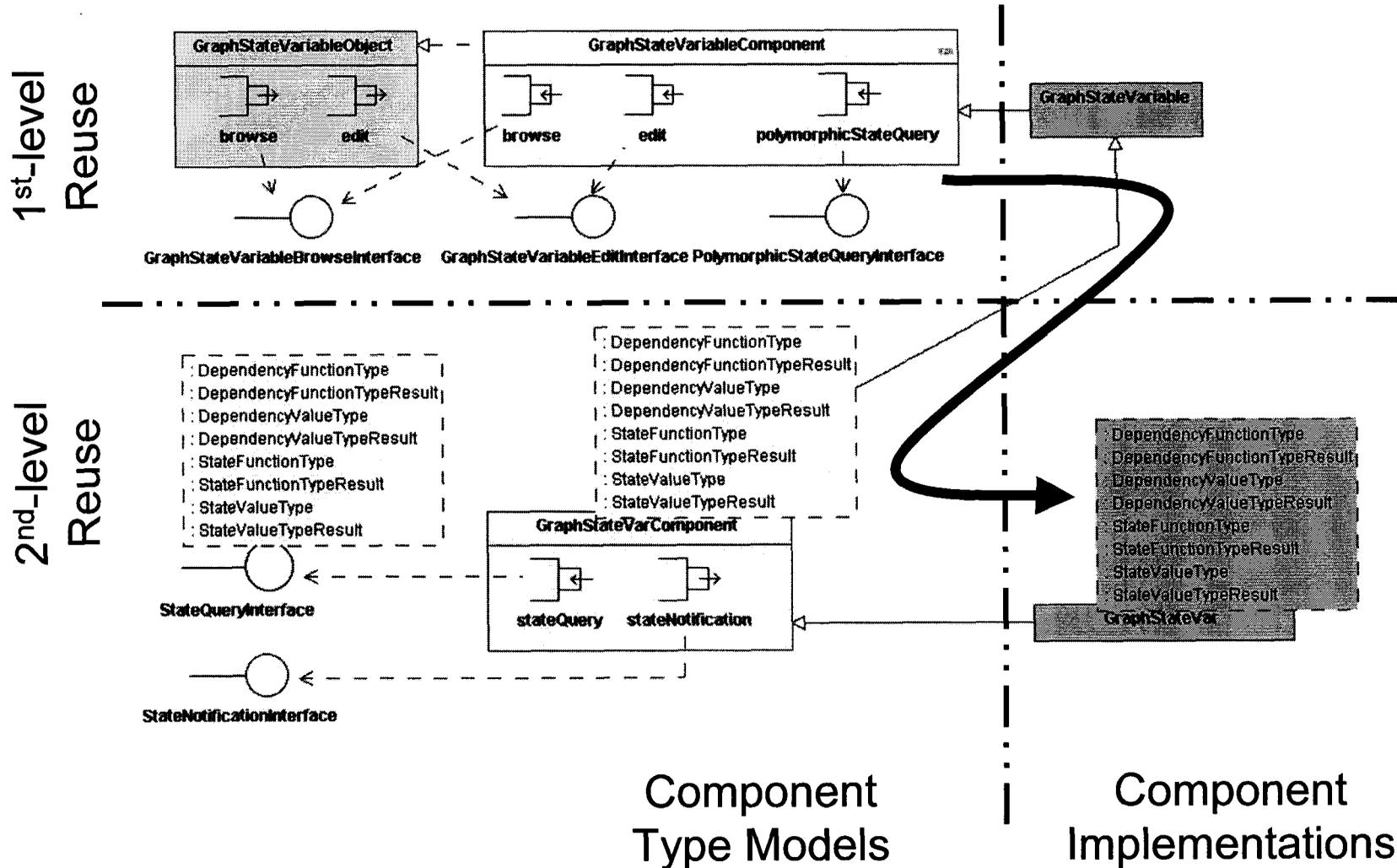


# Architecture Type Modeling – ComponentTypes



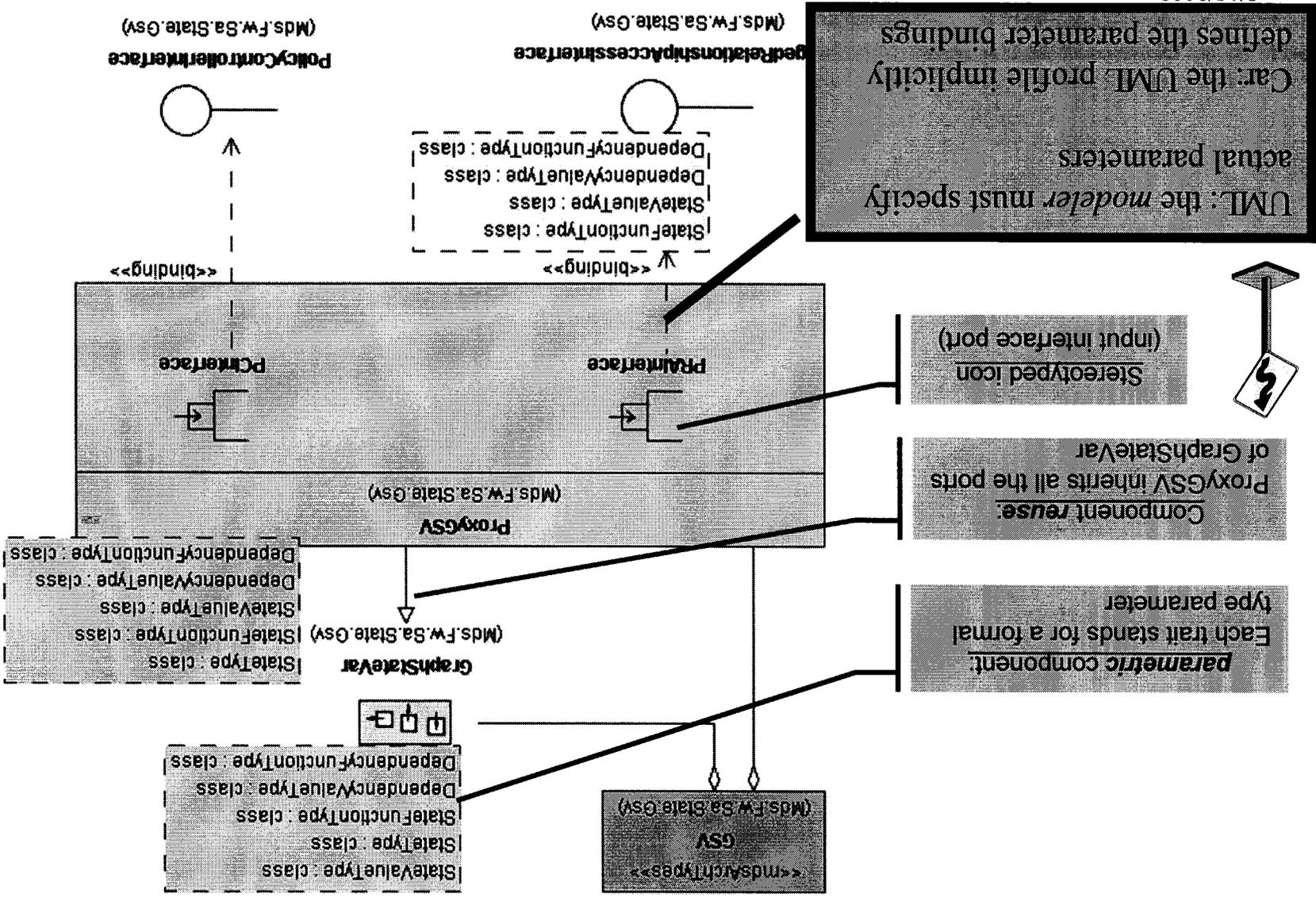
- 3 extensions
  - Roles
    - Idea borrowed from xACME's property & constraint extensions
    - A role is an enumerated value symbol (e.g.,: side = left)
    - Roles decorate component ports (signatures in xADL) and components
    - Roles serve two purposes
      - As a means to express simple constraints on valid prescriptions
      - As a basis for anonymous, role-based programming across connectors (e.g, “block the left wheel, turn the right wheel”)
  - The “Zorro” pattern of structural & implementation inheritance & specialization
    - Reuse the code from the architecture type definitions across implementations
    - Independently extend components with additional ports & behavior
  - Parametric modeling
    - Similar extension to that of parametric interfaces

# Example of the "Zorro" pattern





# Car Design: Modeling a parametric component

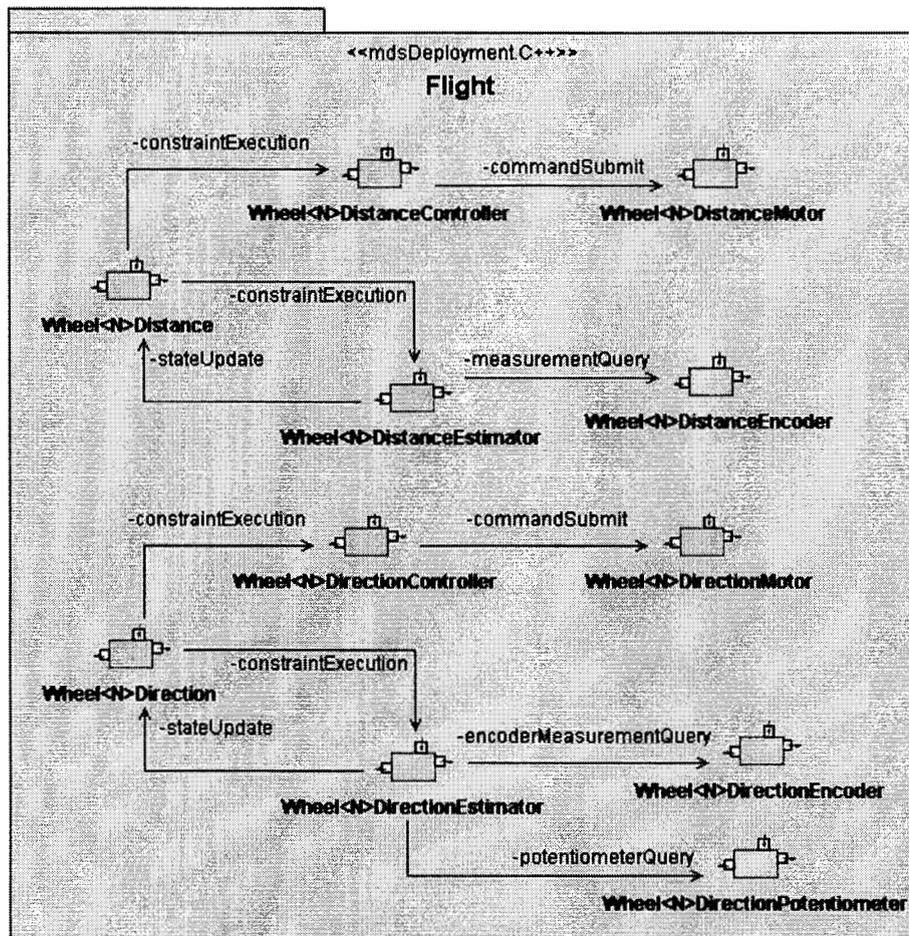




# Why so many extensions of xADL?

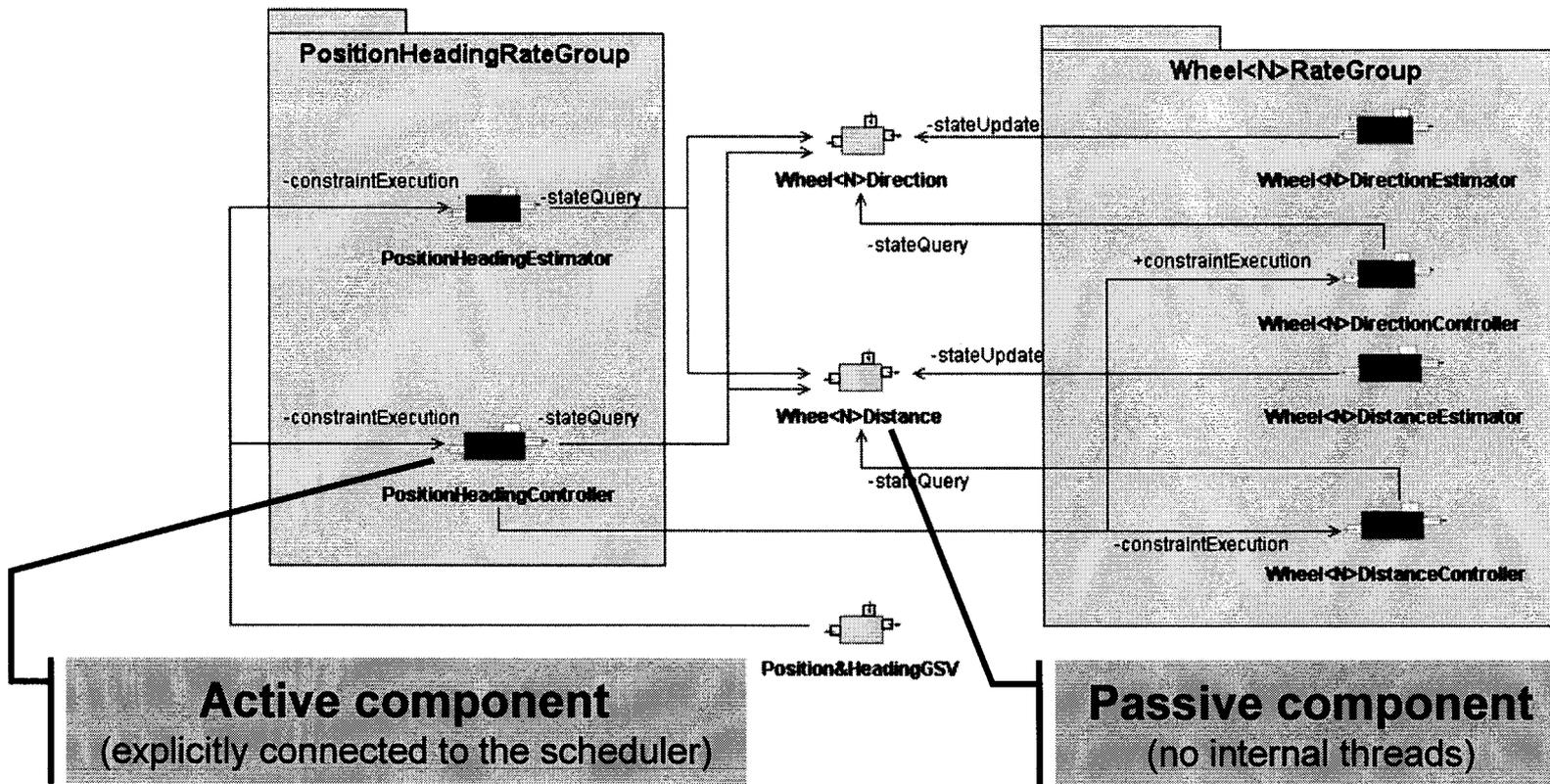


- Architecture hoisting:
  - Shifting concerns from software engineering to architecture engineering
- Example at the prescription level
  - Conceptual architecture                   => what the system should be
  - Execution architecture                   => how it should work
  - Goal:
    - Manage these two aspects of prescription separately & independently
    - Example
      - System engineer:                   => I need 6 controllers & 6 estimators
        - This is a conceptual architecture concern
      - Software engineer:                   => There can be only 3 threads
        - This is an execution architecture constraint
      - Resolving these two views involves:
        - Negotiating tradeoffs
        - Configuring resources for schedulability & performance

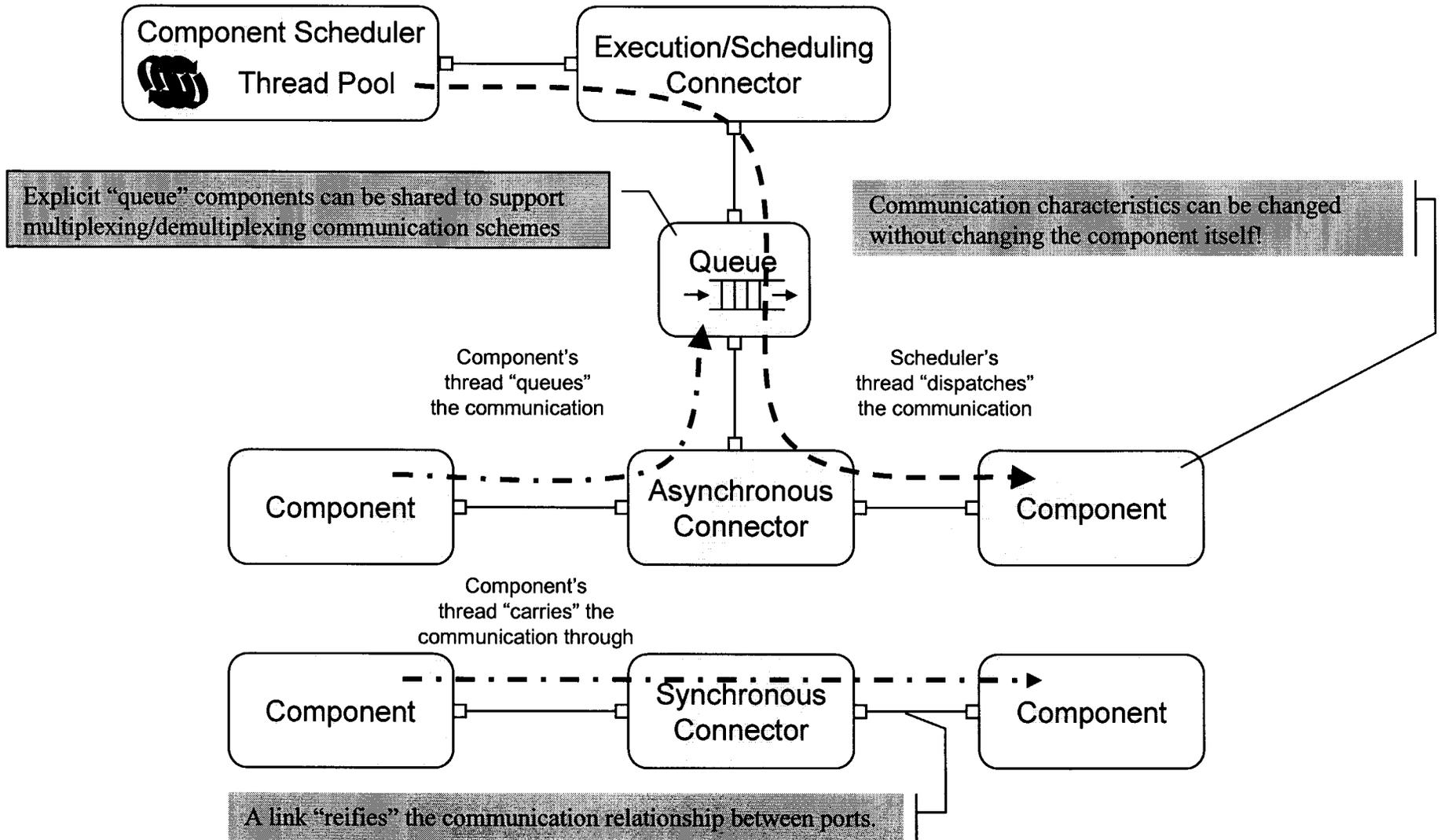


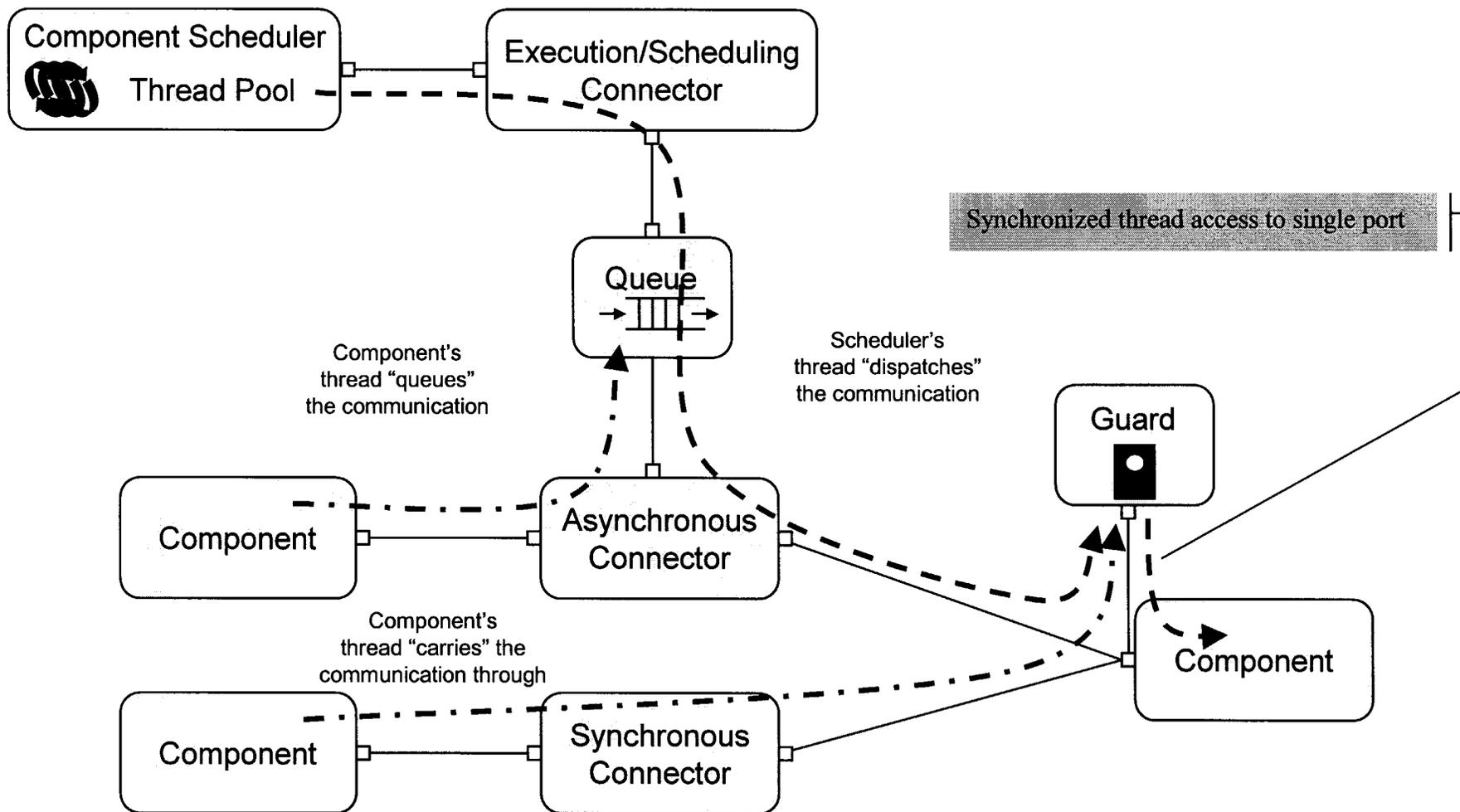
Prescription pattern for each of FIDO's 6 wheels

- This view describes information flow among components via interfaces & connectors
  - It is an item that should be under separate configuration management from other prescription aspects
  - The contents of this view is strongly tied to the identification of states & items in state analysis.
  - If the analysis does not change, this view should not change
- Issue:
  - Can we find a reasonable way to define a pattern like this?
  - Scripting languages make this easy

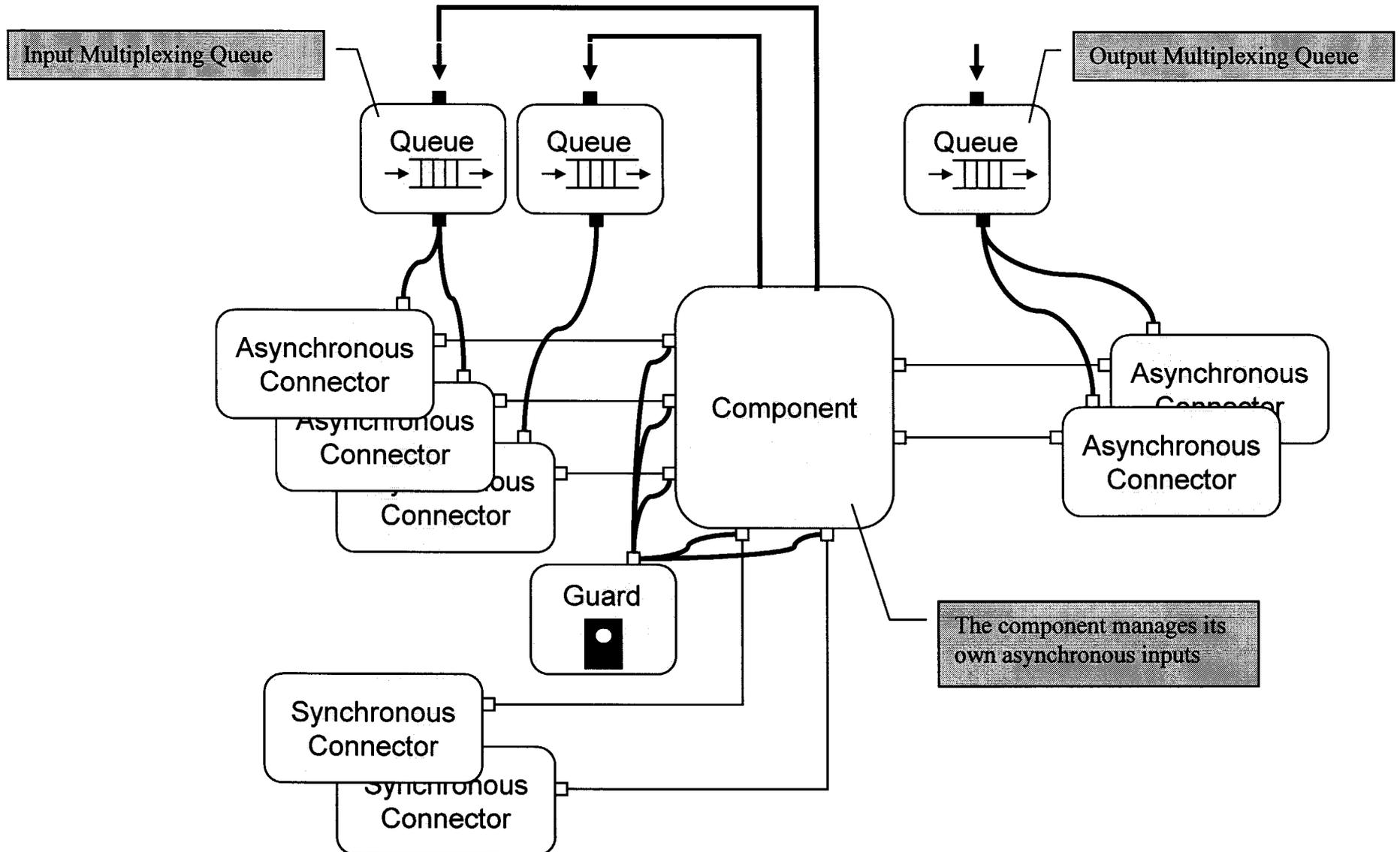


- Many things needed besides threads
  - Asynchronous communication
  - Synchronization guards
  - Scheduling information (rates, ordering, etc...)

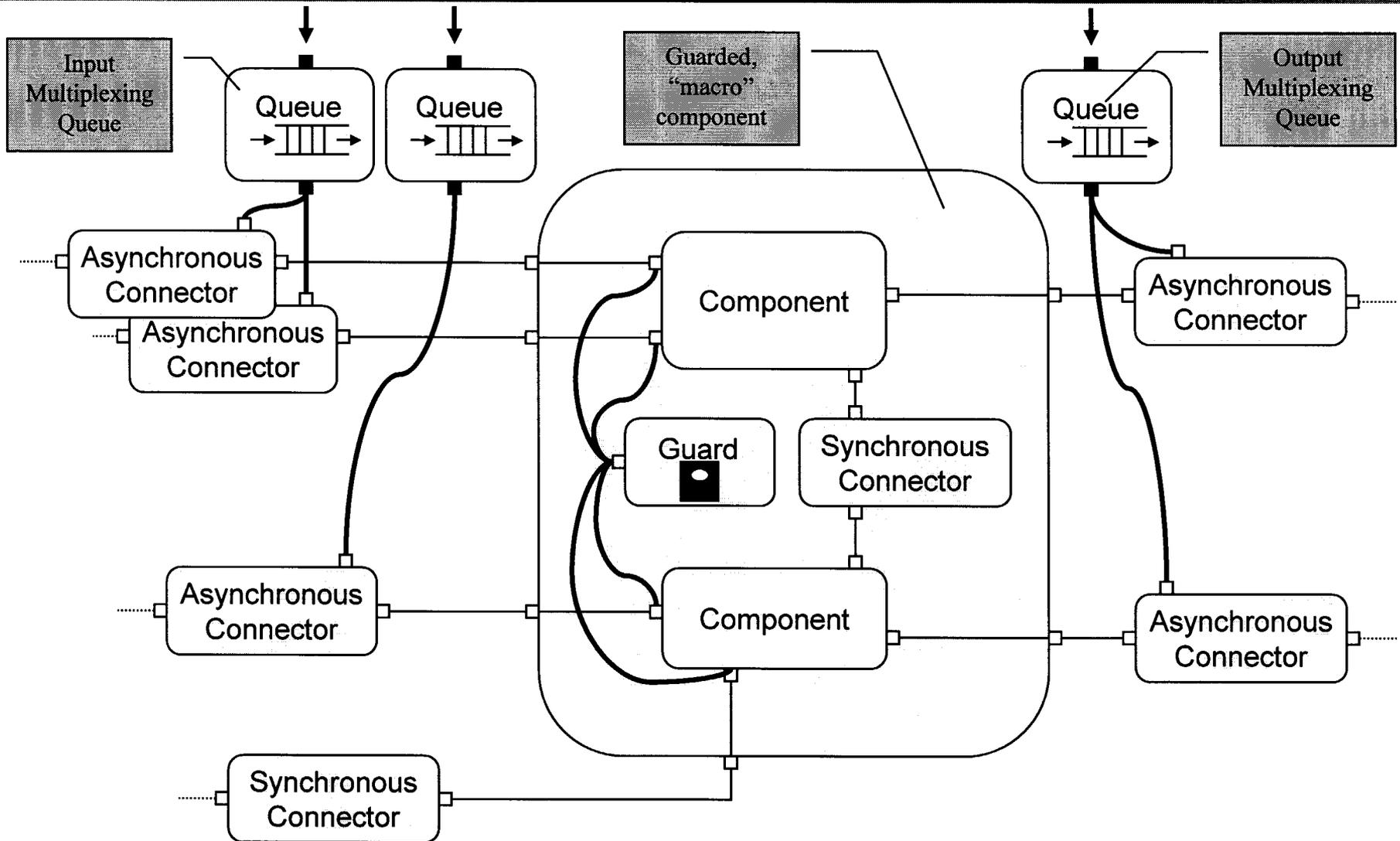


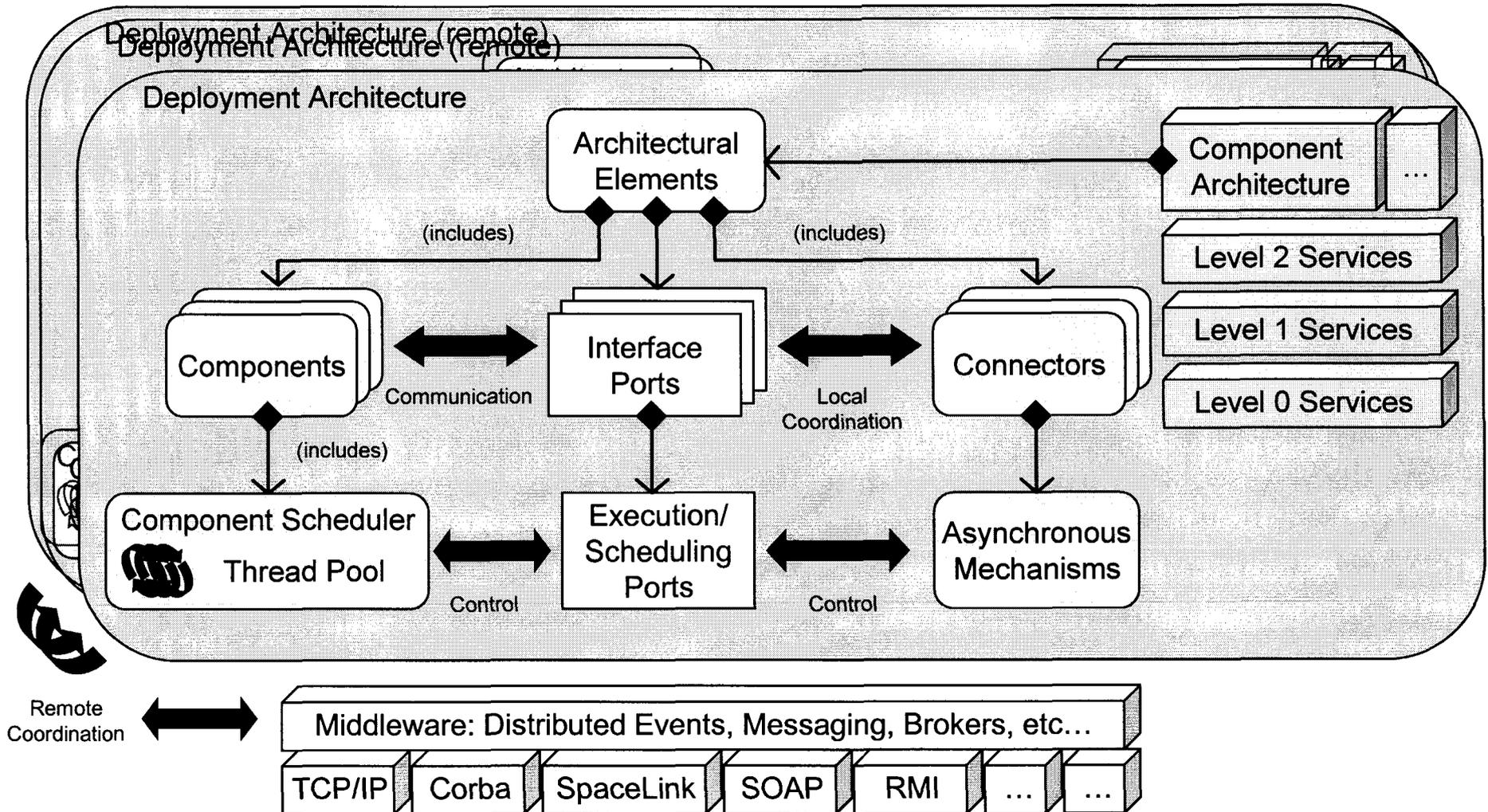


# Putting it all together in a configuration...

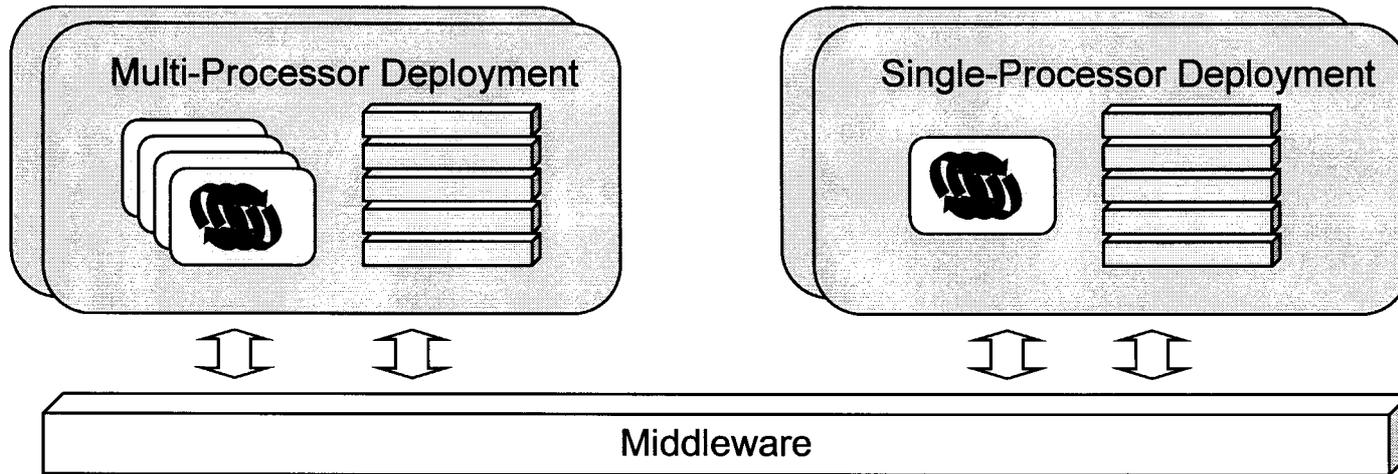


# Putting it all together in a configuration...

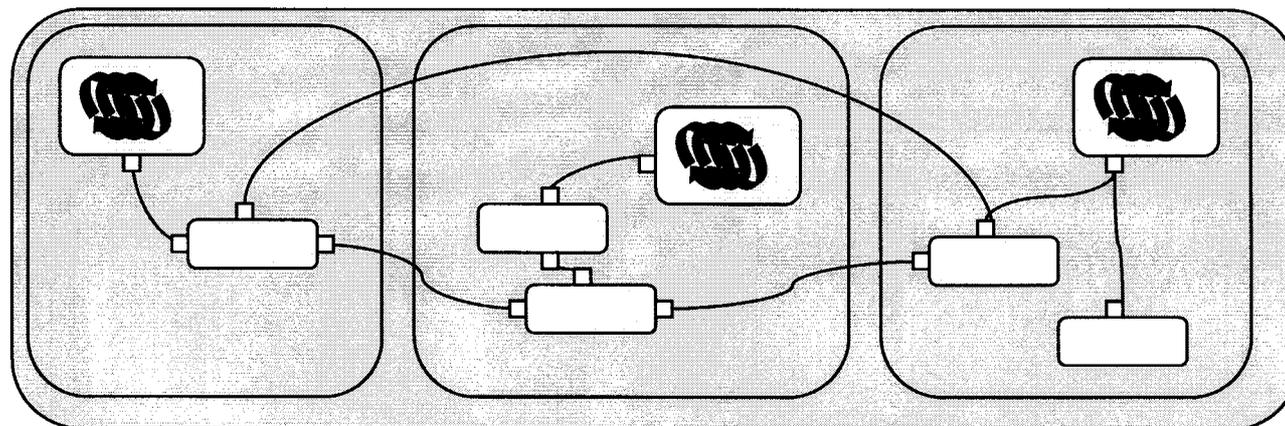




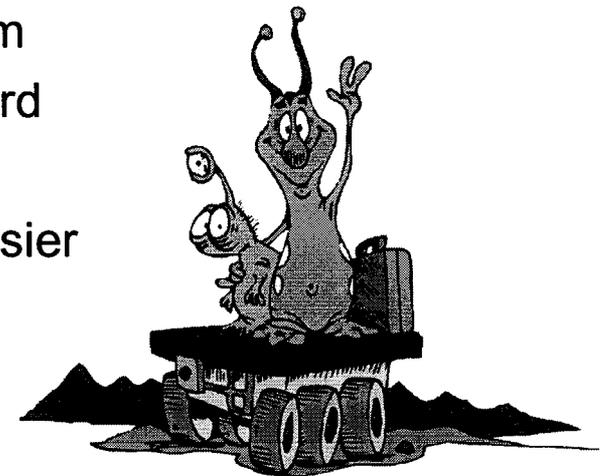
## Physical Architecture View

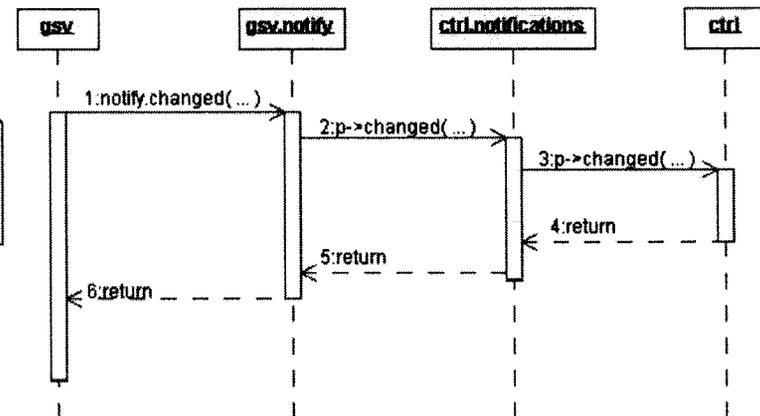
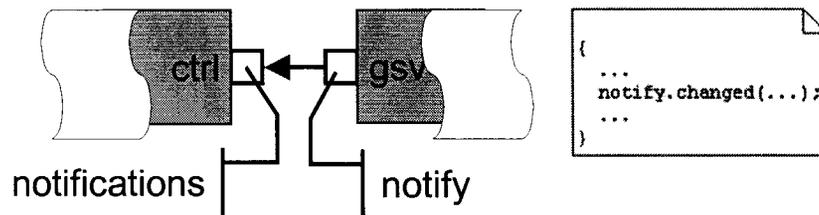
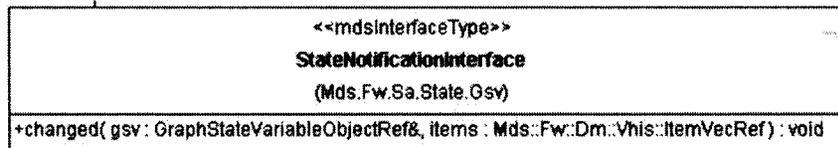


## Component/Connector Architecture View



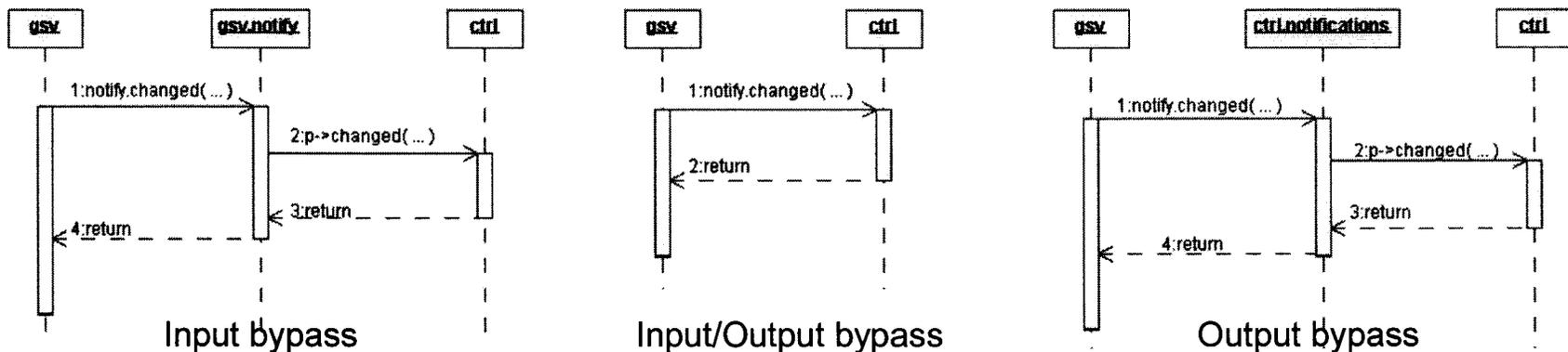
- System & Software Engineering
  - It's been a problem for the last 30 years of space exploration
  - Many forces drive us to re-examine and change this relationship
- Architecture Hoisting
  - It is more than just using different architecture views (e.g., Krutchen's 4+1)
  - It is a deliberate effort to shift engineering effort from
    - Software engineering where it is hard to
    - Architecture engineering where it is easier
- Extensible Architecture Description Languages
  - It's in hyper-growth phase
  - Tool availability & support can be a problem
    - At the modeling level (UML is woefully inadequate)
    - Model-based transformation systems are necessary to get full benefit
    - Programming languages are lagging (interfaces are still 3<sup>rd</sup>-class citizens)





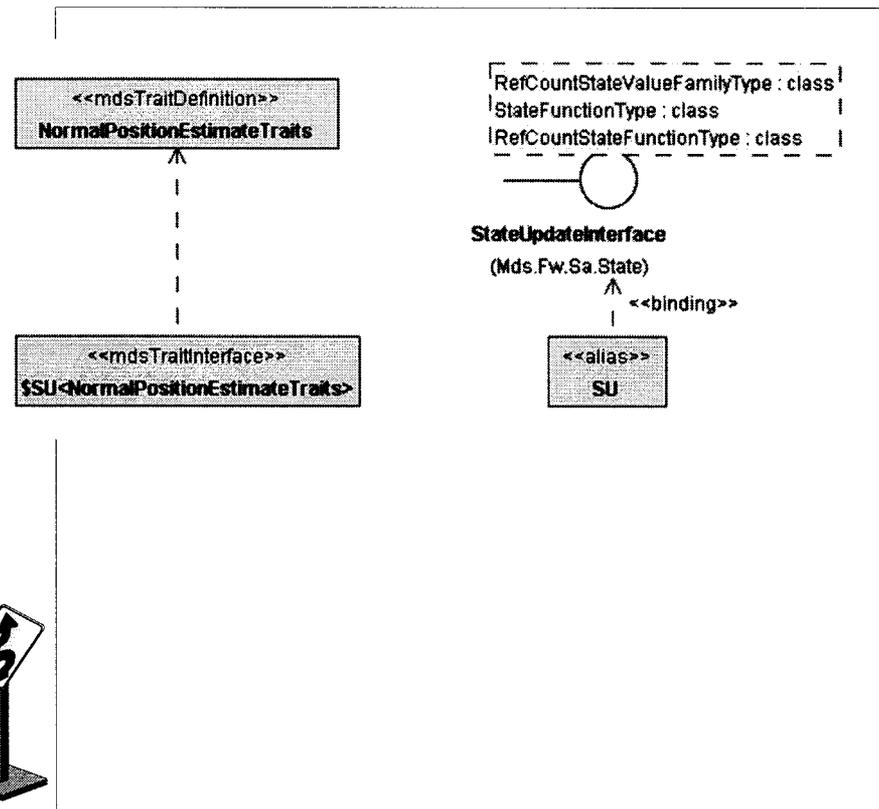
- Port-based communication
  - An output port has a pointer 'p' to the other port on the link
  - An input port has a pointer 'p' to the entity that implements the interface
- Taxonomy of port binding configurations
  - { method call, indirect invocation } x { method call, indirect invocation }

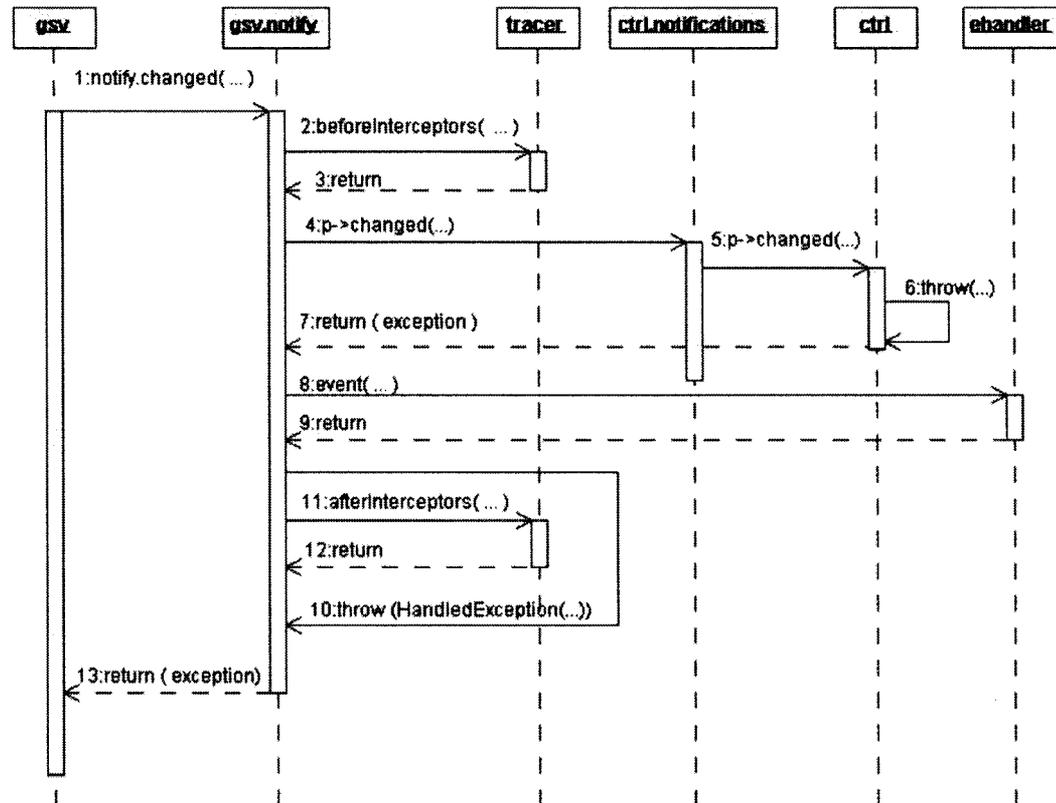
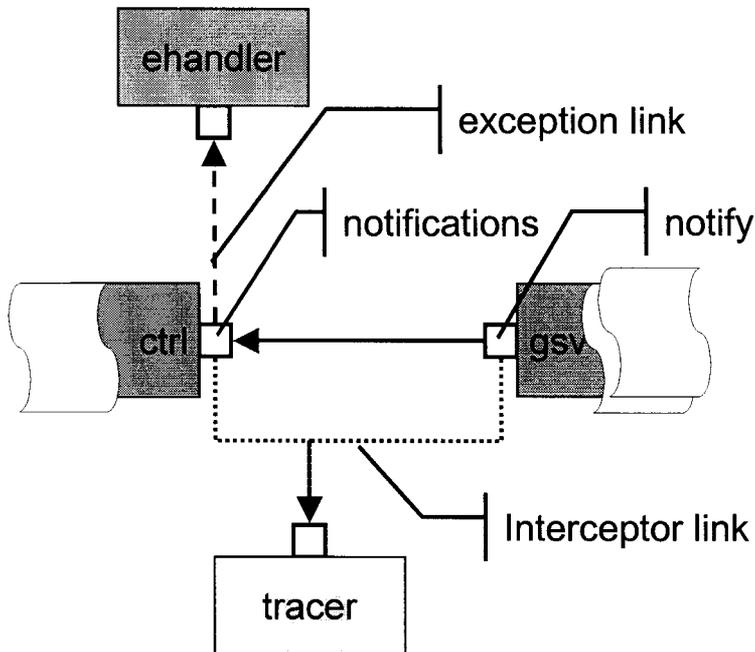
- Port bypass
  - Pass-through property
    - No interceptors
    - No exception handlers
    - Direct method call
  - Proxy exchange & port binding protocol
    - Each port on either side of a prescribed link creates a proxy of itself
    - Each port receives a proxy of the port on the other side of the link
    - Proxies & ports can be queried for pass-through property



- Separation of design concerns towards facilitating reuse
  - Communication protocol design      => one set of interface methods
  - Compile-time polymorphism & optimization      => several type specializations

- UML verbosity
  - standard notation for bindings leads to verbosity & clutter
- CAR
  - Minimal diagram clutter (eliminate duplication)
  - Caveat:  
Complex UML profile!  
Complex UML/xADL transform





- Variations

- Full interceptors & exception => provides access to method arguments
- Partial interceptors & exception => provides access to method name

- Benefits

- Exception handlers: Separate of exception detection vs. exception recovery
- Interceptors: infuse aspect-oriented style into the component/connector style