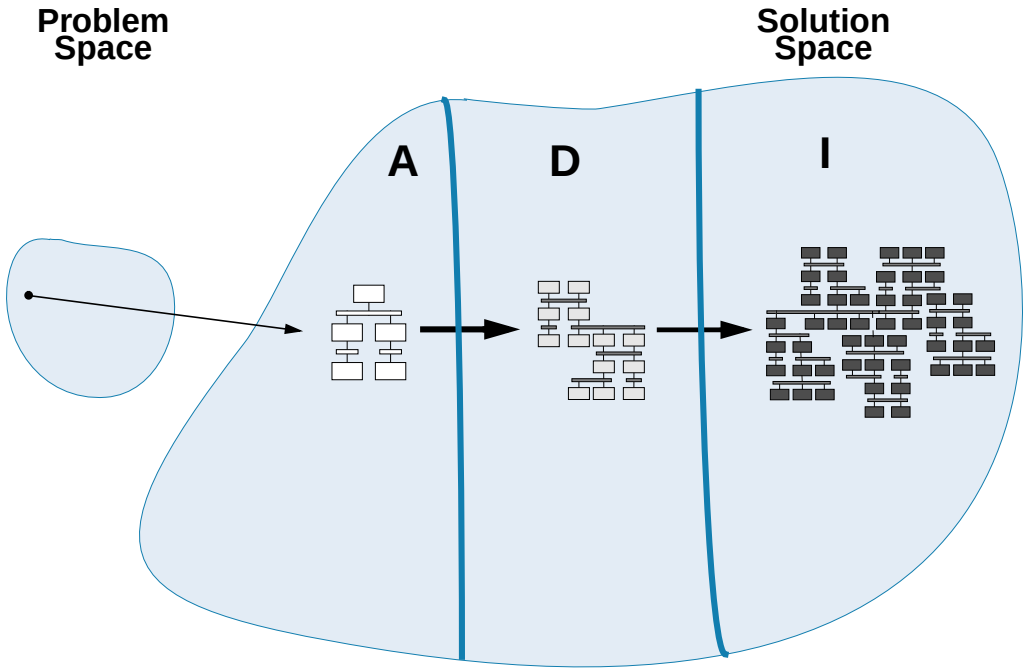
# From Architecture to Design








# Using UML to Model Architectures

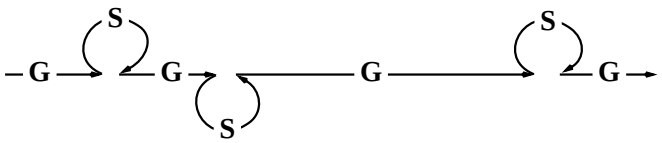
- Community fragmentation

| Academic approach to architectures | Industrial approach to architectures |
|---|---|
| focus on analytic evaluation of architectural models | focus on wide-range of development issues |
| individual models | families of models |
| rigorous modeling notations | practicality over rigor |
| powerful analysis techniques | architecture as the “big picture” in development |
| depth over breadth | breadth over depth |
| special-purpose solutions | general-purpose solutions |

# Standardization

- Provides an economy of scale
  - more and better tools
  - improved tool interoperability
  - more skilled developers
  - lower training costs

→ *Combine the benefits of powerful, specialized notations with those of widely adopted, general notations*




- Specific solution: integrate ADLs with UML
  - three integration approaches
  - multiple ADLs to date





# Unified Modeling Language

- Large, useful set of predefined constructs
- Extensible
- Semi-formal definition of syntax and semantics
- Potential for
  - wide adoption
  - standardization
  - substantial tool support
- Basis in experience with mainstream development methods

## Overview of UML

- A UML model comprises several partial models addressing
  - classes with attributes, operations, and relationships
  - states and behavior of individual classes
  - packages of classes and their dependencies
  - example scenarios of system usage
  - object instances with actual attributes in a scenario
  - actual behavior of interacting instances in a scenario
  - distributed component deployment and communication
- UML syntax and semantics are defined via
  - a meta-model
  - descriptive text
  - constraints





## Extensibility in UML

- New constructs may be added to address new development issues
- Three extensibility mechanisms
  - constraints — semantic restrictions on design elements
  - tagged values — allow addition of new attributes to elements
  - stereotypes — named grouping of constraints and tagged values

**Stereotype** Person for instances of meta-class Class
[1] A Person can be either female or male
`personGender : enum { female, male }`

- The metamodel may also be extended
  - results in a new notation
  - may be incompatible with UML-compliant tools

## Integration Strategies

- Strategy #1
  - use UML “as is”
  - enables direct comparison of UML and an ADL
  - clarifies the relationship between architecture and design
- Strategy #2
  - use UML’s built-in extension mechanisms on meta-classes
  - allows automated conformance checking
  - → select a meta-class semantically close to an ADL construct
  - → define a stereotype and apply it to meta-class instances
  - → class semantics are constrained to that of the ADL
- Strategy #3
  - Extend UML to directly support architectural concerns

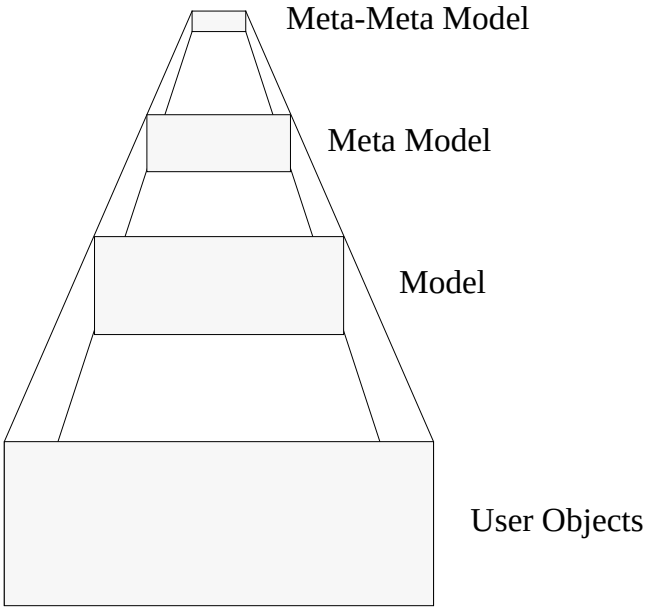




## Strategy #1

- Simultaneous consideration of architecture composition rules and UML notational constructs
- Develop a UML domain model
- Develop an (informal) architectural diagram
- Map domain classes to architectural components
- Design class (component) interfaces
- Provide constructs for modeling connectors
  - connectors add no functionality at the domain model level
- Model architectural structure in class diagrams

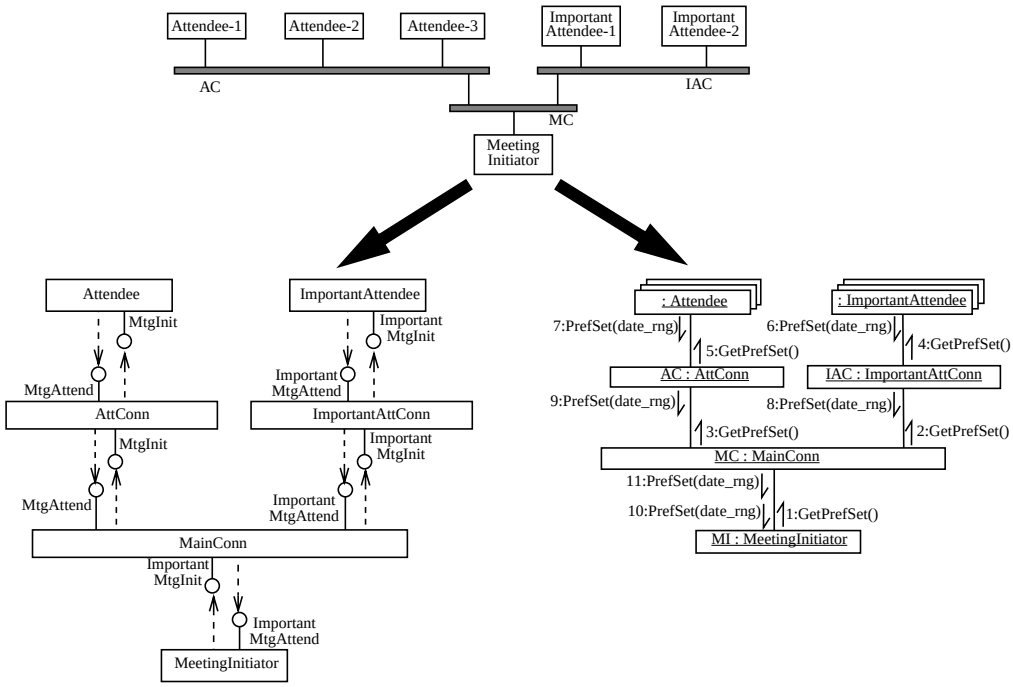
# UML Metamodeling Architecture View








# Strategy #1 Example



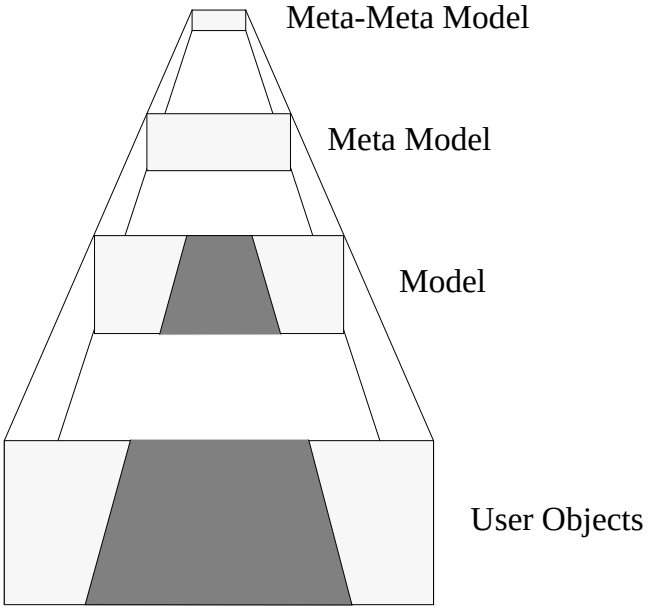
## Strategy #2

- Identify UML constructs similar to each major architectural construct
  - operation
  - message
  - event
  - port
  - component
  - connector
  - architecture
  - behavior
- Constrain those UML constructs via stereotypes
  - use stereotypes to model structural aspects of an architecture
- Describe semantics using UML statechart diagrams

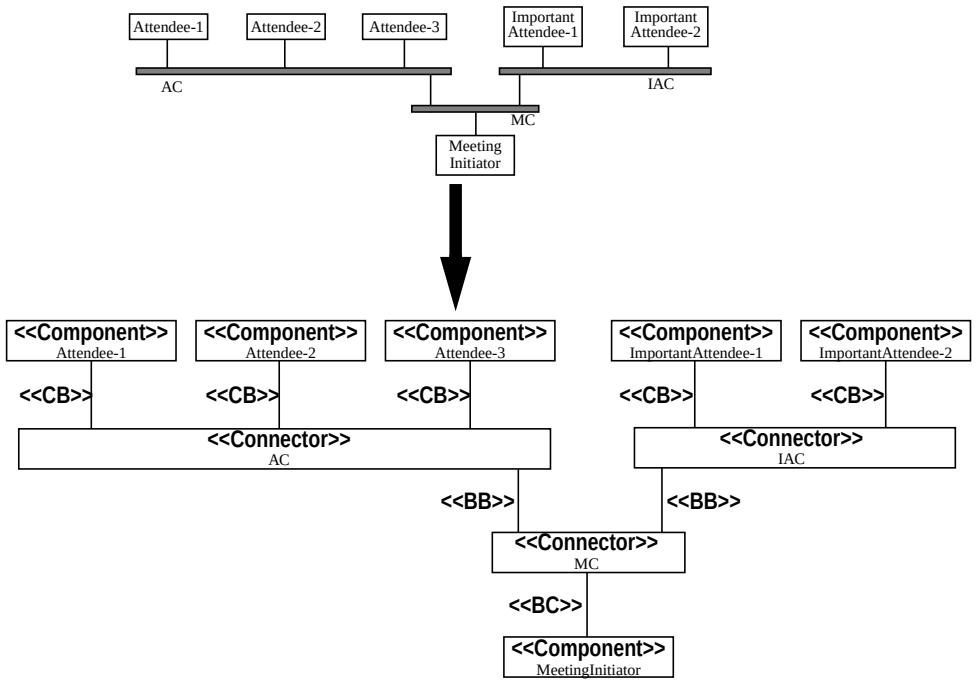




## UML Metamodeling Architecture View



# Strategy #2 Example



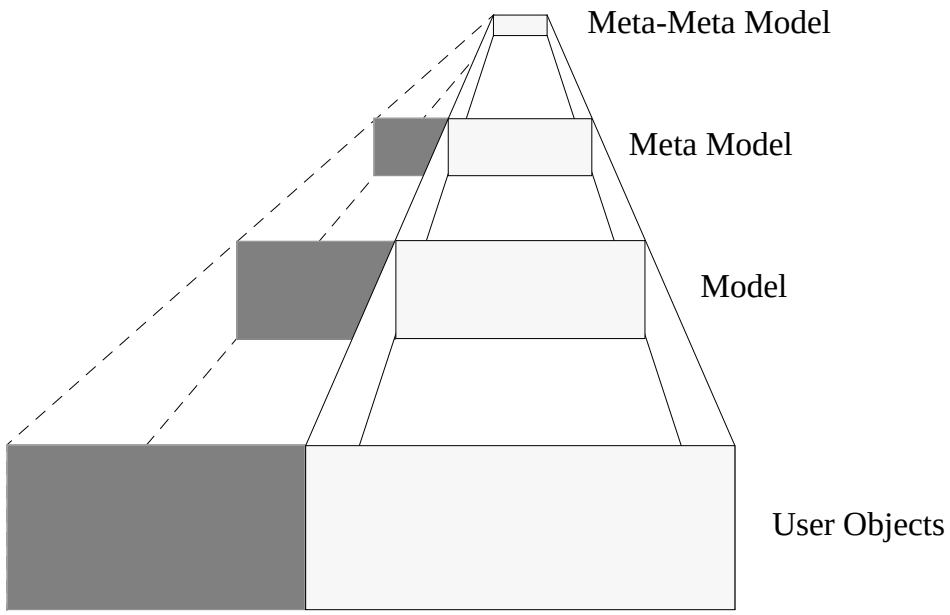





# Strategy #3

- Introduce *explicit* architectural constructs and constraints in UML
- Introduce additional notations for modeling architectural semantics
- Follow an approach similar to strategy #1 to model specific architectures
- Follow an approach similar to strategy #2 to model specific architectural styles

# UML Metamodeling Architecture View








# Discussion

- All three approaches have merits and shortcomings
- “Straight” UML
  - understandable architectures
  - manipulable by standard tools
  - architectural constraint violations
- “Constrained” UML
  - ensures architectural constraints
  - requires complete style specifications
  - requires OCL-compliant tools
- “Extended” UML
  - provides “native” support for architectures
  - requires backward tool compatibility
  - may result in incompatible UML versions

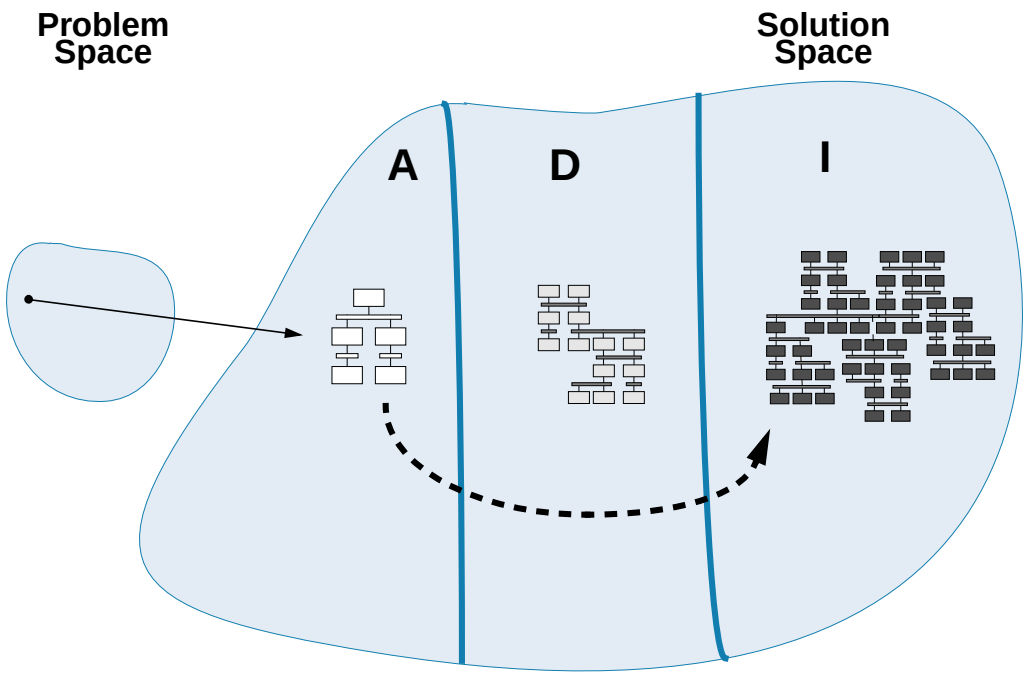
# ADLs and UML Can Complement Each Other

- Similar software modeling philosophies
  - neither constrains system implementation
  - ADL behavior modeling notations are expressible as sequence, collaboration, statechart, or activity diagrams
- Different assumptions
  - UML's intended usage is not architecture modeling
  - UML may make assumptions non-existent in ADLs
- Problem domain modeling
  - including relevant information early is critical
  - domain model as a useful architectural perspective
- Architectural abstractions
  - the goal is to match an architect's mental model
- Architectural style
  - requires constraining the UML meta-model



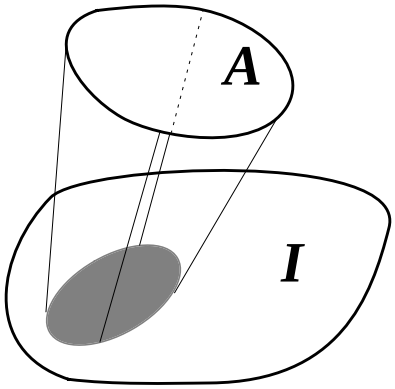


# From Architecture to Implementation



## Mapping an Architecture to an Implementation

- Infeasible in general
  - → reduces to transformational programming
- Possible by limiting the target space
  - middleware platforms
  - software bus technologies








## What is Middleware?

- Infrastructure that supports (distributed) component-based application development
  - a.k.a. distributed component platforms
  - mechanisms to enable component communication
  - mechanisms to hide distribution information
  - (large) set of predefined components
- Standard for constructing and interconnecting components
  - interchange
  - upgrade
  - adaptation
  - aggregation

## Elements of Middleware

- Software components
  - component interfaces
    - properties
    - methods
    - events

    } component's callable API
- Containers
  - shared context of interaction with other components
  - provide access to system-level services
- Metadata
  - self-descriptive information used by a component to *flexibly* communicate with others
- Integrated development environment
  - e.g., VisualCafe for Java





## Distribution Support in Middleware

- Hidden from application programmers
- Five distributed services required
  - remote communication protocols
    - e.g., RPC, message passing
  - directory services
    - for accessing shared, globally-available services
  - security services
    - protection of shared resources via authentication
  - transaction services
    - for concurrent data access/update
  - system management services
    - service monitoring, management, and administration

## Relating Architectures and Middleware

- Architecture-based development is top-down
  - decomposition
- Component-based development is bottom-up
  - composition
- Middleware imposes certain architectural constraints
  - usage of middleware and predefined components can influence the architecture
- Architecture can impose constraints on middleware
  - specifying the architecture restricts the number of middleware options
- Architecture ≈ Framework + Framework Methodology

→ *Finding suitable middleware can aid in the implementation of an architecture*



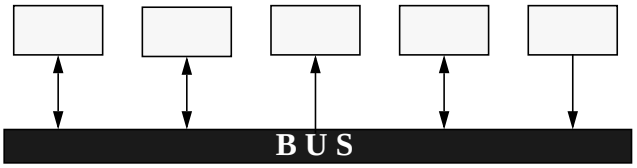


## Existing Middleware Platforms

- OLE
- OpenDoc

- CORBA
- DCOM
- JavaBeans

- Jedi
- C2 Implementation Framework

# Software Buses

- Middleware mostly focuses on components
  - components are homogeneous
  - intra-middleware OTS reuse
  - connectors are hidden in the middleware infrastructure
- Software bus technologies focus on component interactions
  - components may be heterogeneous
  - wider reuse possible
- Buses still require an architectural approach to application development