Real-Time UML Powered by SDL

- Software engineering challenges
- Language support: UML & SDL
- UML2

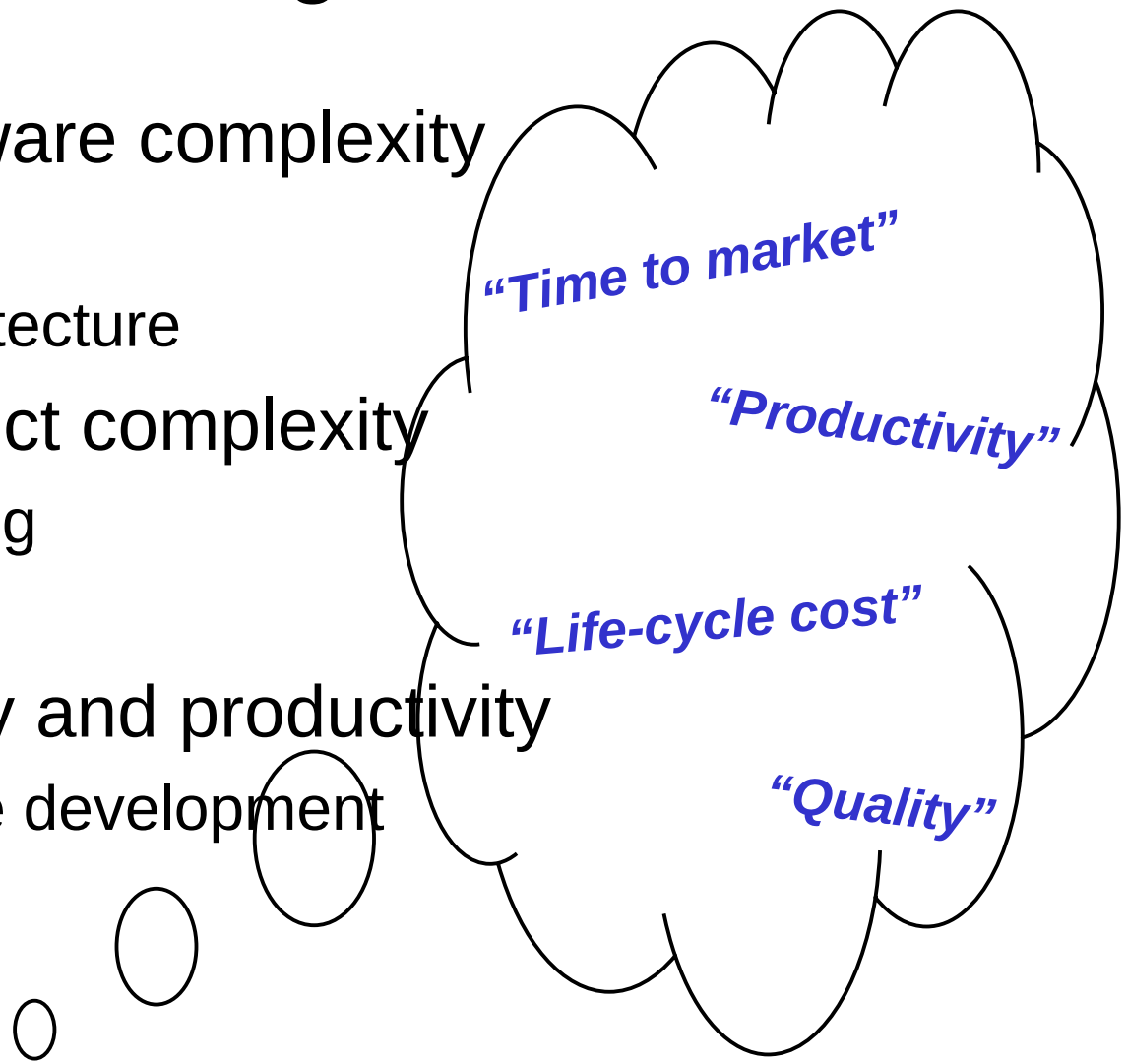
Anders Ek (anders.ek@telelogic.com)

Application Domain Characteristics

- Soft real-time systems
  - Reactive, event driven systems
  - Asynchronous communication
  - High degree of concurrency
  - Distributed
- Application areas
  - Telecom
  - Aerospace
  - Automotive

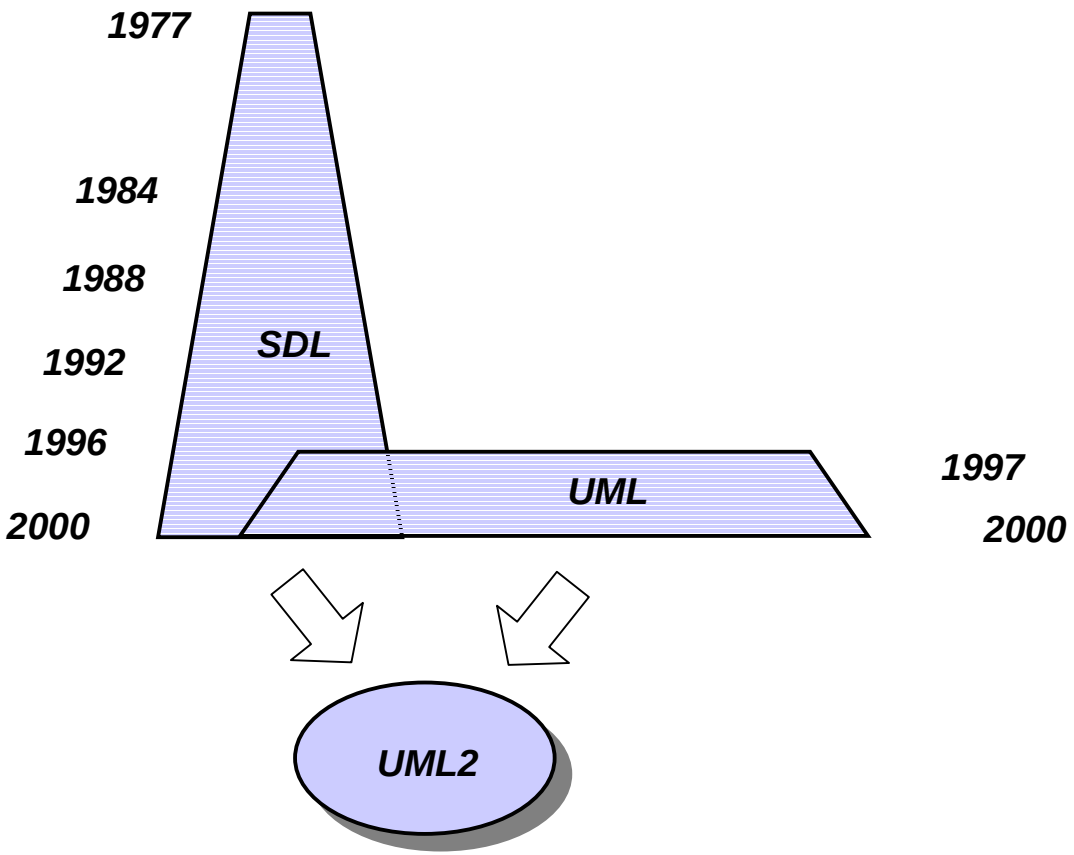
Challenges

- Managing software complexity
  - Viewpoints
  - Run-time architecture
- Managing project complexity
  - Work structuring
  - Contracts
- Enabling quality and productivity
  - Visual software development



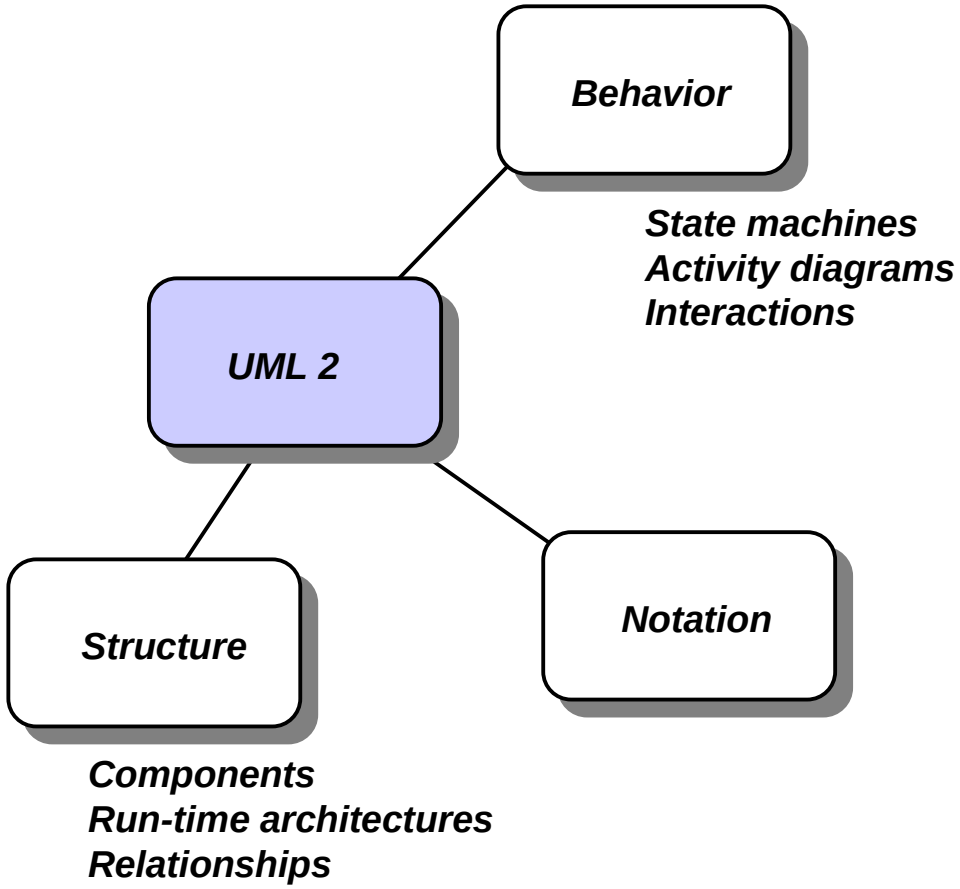
Languages

- Visual software development
  - UML
  - SDL
- SDL-2000 & Z.109
- UML 2.0



UML 2.0

- UML 2.0 RFIs
  - Dec 99
- UML 2.0 RFPs:
  - UML 2.0 Infrastructure
  - UML 2.0 Superstructure
  - UML 2.0 OCL
  - (UML 2.0 Interchange)
- Submissions
  - initial: August 2001
  - revised: February 2002



Massive Support for UML 2.0

LOI List:

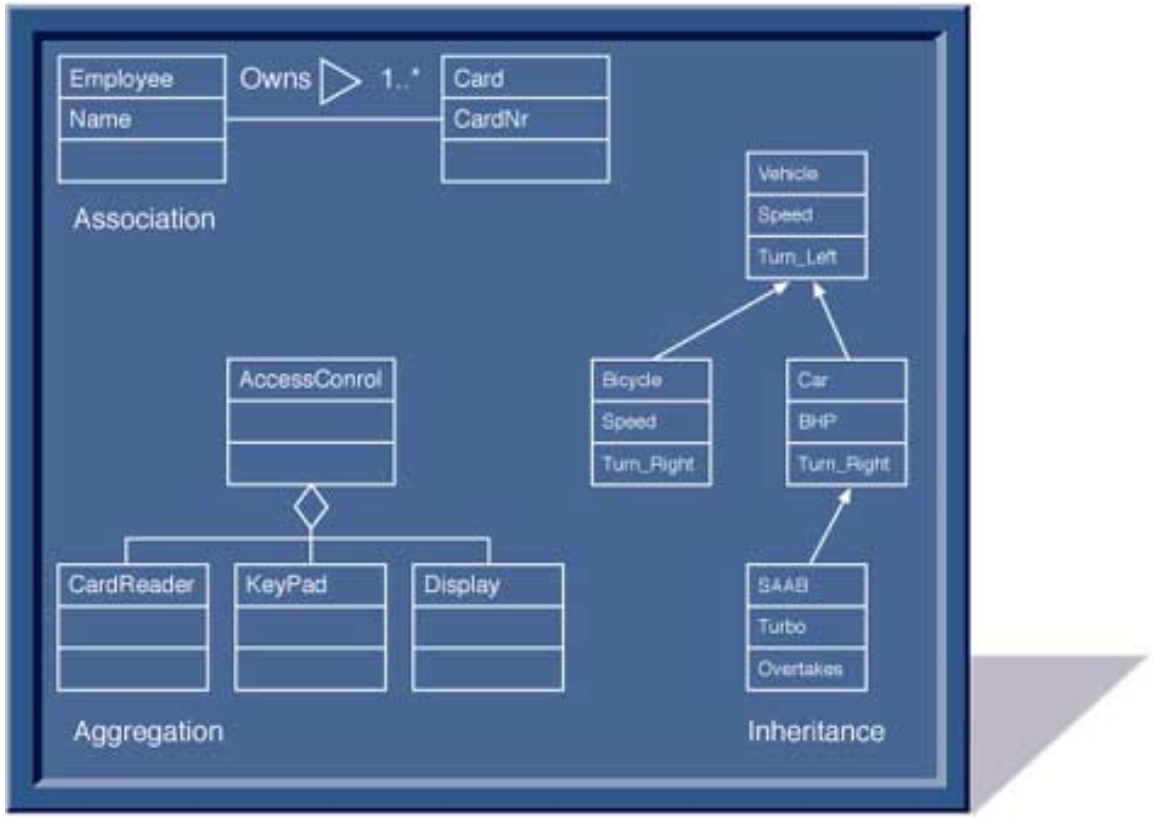
Company	Date	Notes
ARTISAN Software Tools	October 16, 2000	
Alcatel	November 9, 2000	
DSTC	October 24, 2000	
Data Access	September 21, 2000	
Ericsson	November 9, 2000	
Fujitsu	November 9, 2000	
Hewlett-Packard	November 1, 2000	
IONA	November 9, 2000	
International Business Machines	November 6, 2000	
Jaczone AB	November 10, 2000	
Kabira Technologies, Inc.	October 16, 2000	
MEGA International	November 10, 2000	
Mercury Computer Systems	October 19, 2000	
Motorola	November 6, 2000	
Rational Software	November 1, 2000	
Siemens AG	November 6, 2000	
Sight Code, Inc.	November 1, 2000	
Softeam	October 12, 2000	
Telelogic AB	November 1, 2000	

Challenges

- **Managing software complexity**
  - **Viewpoints**
  - **Run-time Architecture**
- Managing project complexity
  - Work structuring
  - Contracts
- Enabling quality and productivity
  - Visual software development

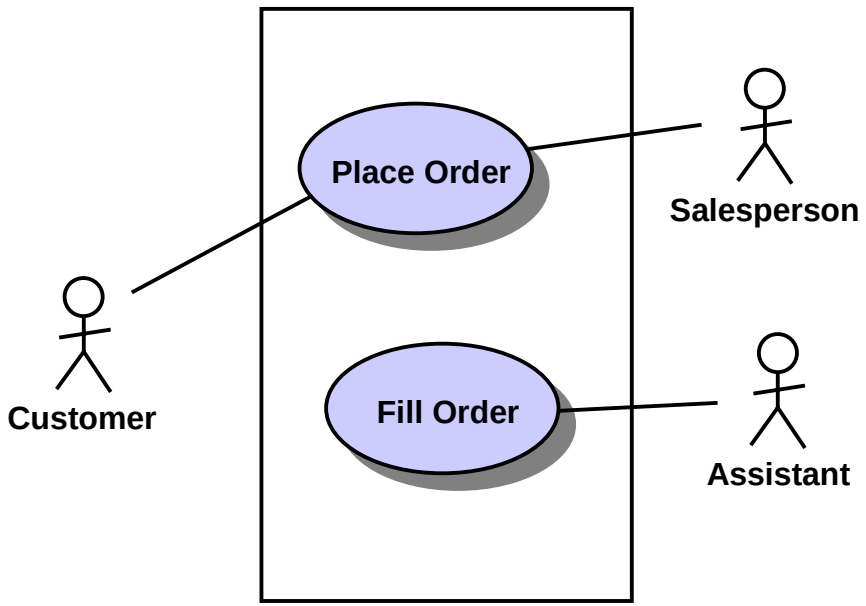
Static Structure

- Class diagrams
  - Classes
    - attributes
    - operations
  - Associations
    - aggregate
    - composite
  - Inheritance



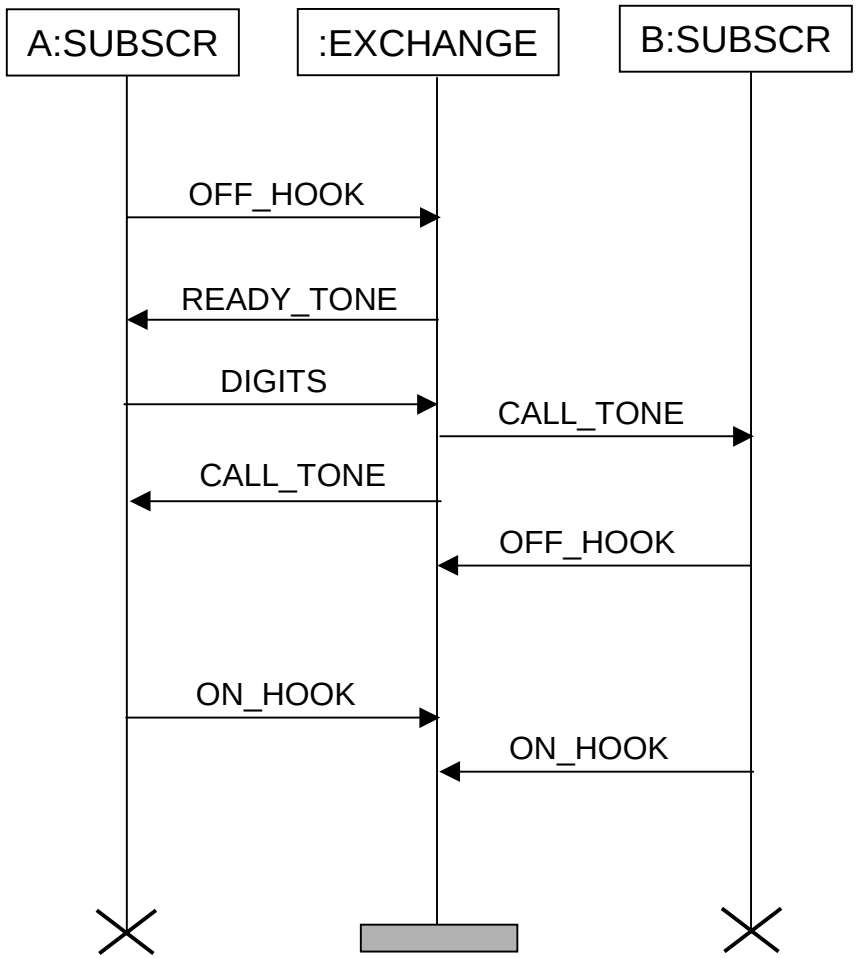
Use Cases

- Describe how external actors interact with the system
  - focus on functionality
- Communicate the primary functionality of the system to non-technicians
- Act as roadmaps for system usage scenarios
  - Close relationship to sequence diagrams



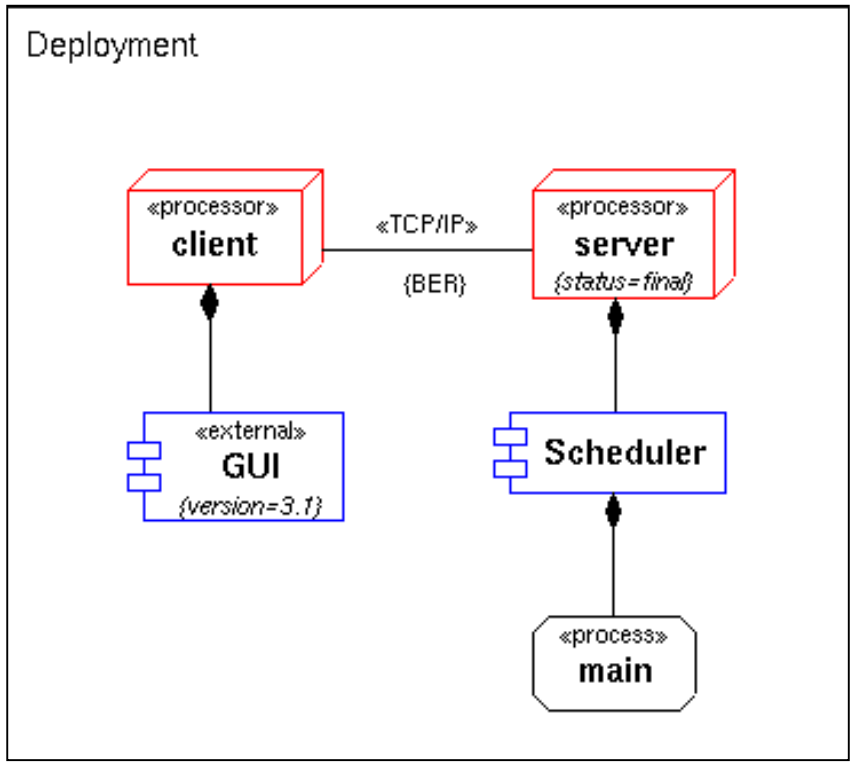
Sequence Diagrams

- Interactions between instances
  - communication
    - asynchronous
    - synchronous
  - life-cycle
    - creation
    - termination
  - intuitive
    - work well with use cases
- Requirements
- Testing



Implementation Views

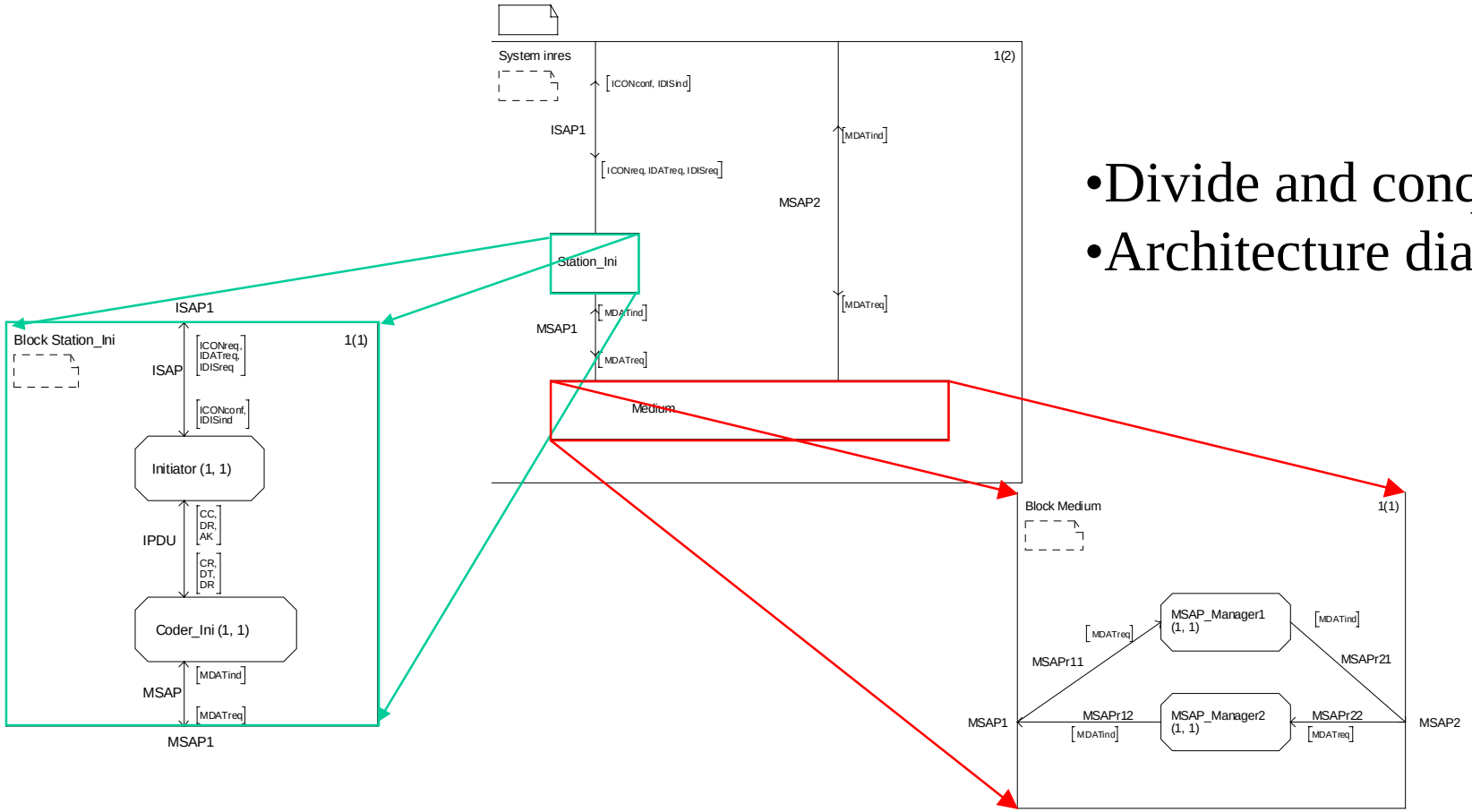
- Focus on application creation
  - code generation
  - compilation
  - “make”
- Configuration
  - target language
  - scheduling
- Deployment
  - distribution
  - message transport



Challenges

- **Managing software complexity**
  - **Viewpoints**
  - **Run-time Architecture**
- Managing project complexity
  - Work structuring
  - Contracts
- Enabling quality and productivity
  - Visual software development

Run-time Architecture



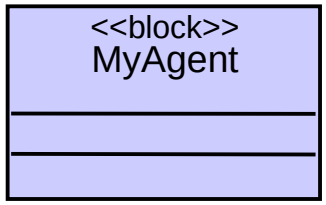
- Divide and conquer
- Architecture diagrams

Run-time Architectures (1)

"Proposals shall support the modeling of the internal structure of a classifier in terms of its hierarchical decomposition. The internal structure shall be allowed to contain instances of classifiers and links between these instances, without affecting the usage of these classifiers elsewhere. The connections between instances shall, at a minimum, specify possible communication."

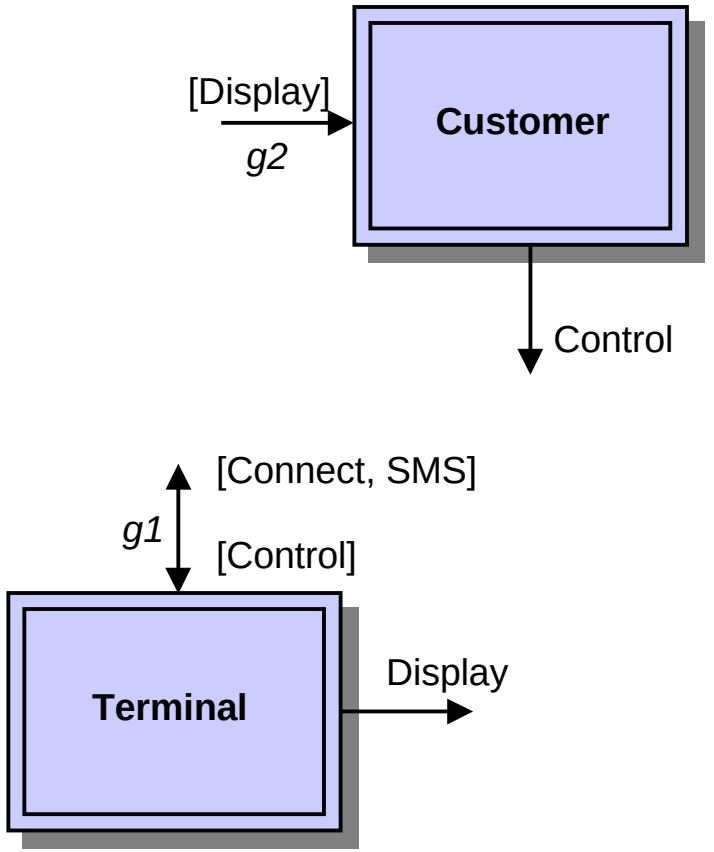
The Agent

- Agent is the fundamental modeling construct in SDL-2000
  - Hierarchical decomposition
  - Active object
- It can be considered a logical component
  - provides encapsulation
    - a “black box”
    - has interfaces
  - has internal structure
  - has behavior
    - through state machines



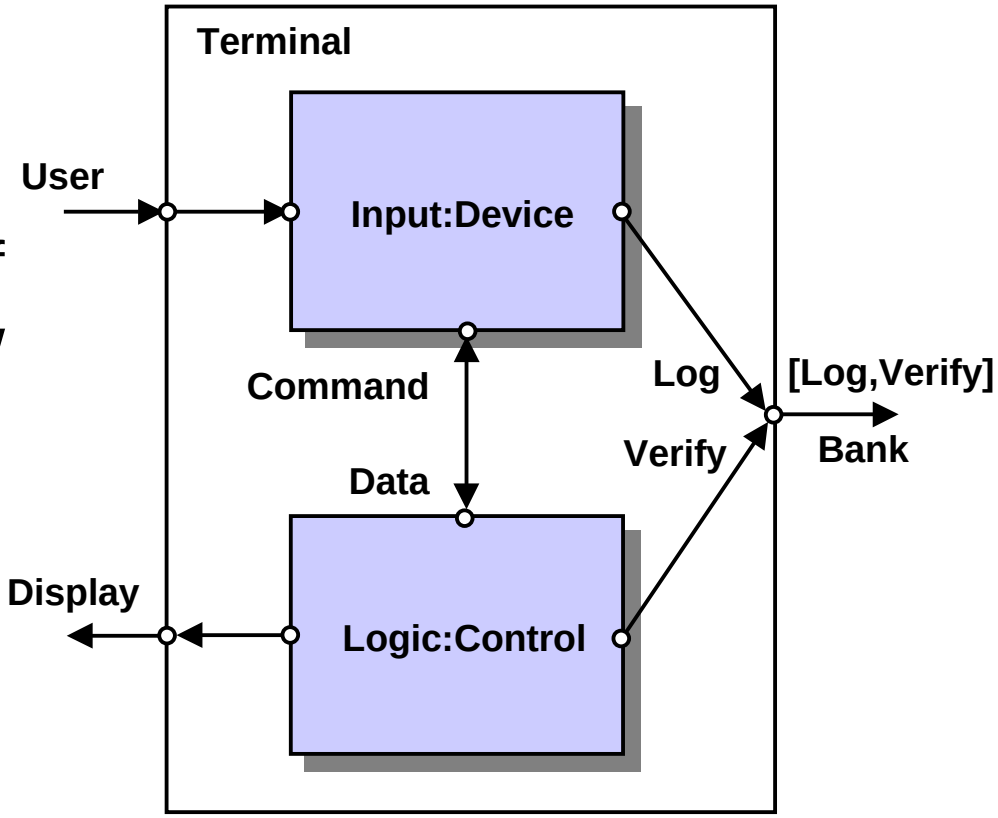
Encapsulation

- An agent has bi-directional gates that describe its
  - implemented interfaces
    - services that are realized by the agent
  - required interfaces
    - services that others must implement
- The interfaces are used to specify contracts between agents



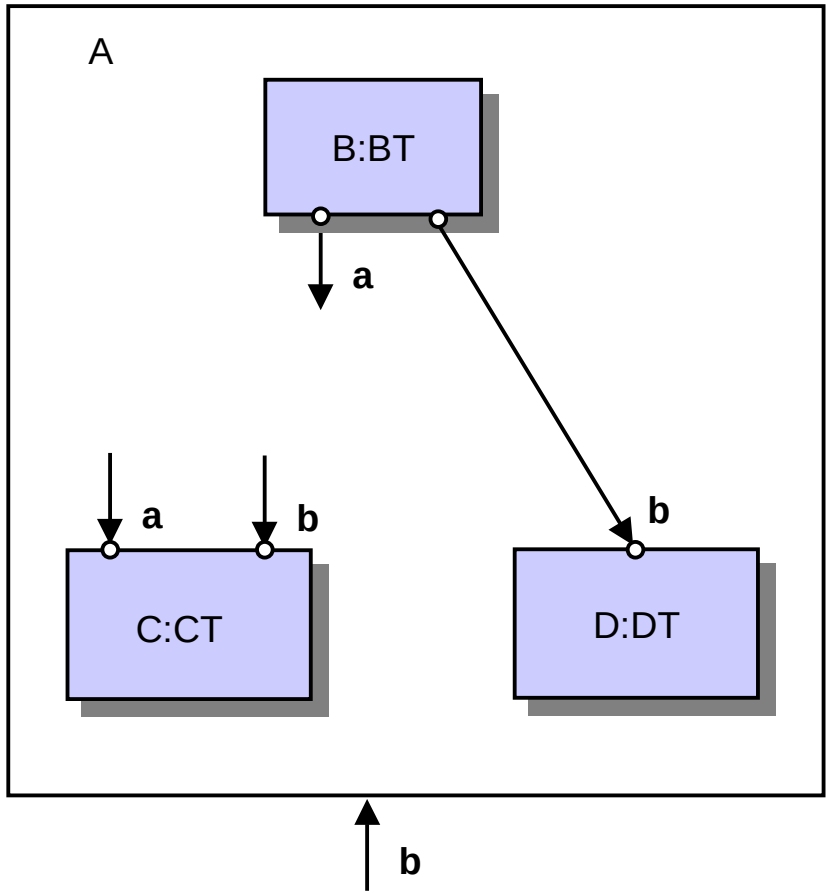
Interconnected Agents

- Agents are structured hierarchically
  - composition relation
- The internal structure of an agent describes how the contained agents interact
- Channels: Connection paths between agents
  - context dependent ‘associations’



Explicit and Implicit Channels

- Only agents with matching required and implemented interfaces may be connected
  - explicitly
    - channels are explicitly drawn between agents
  - implicitly
    - channels are not shown
    - the connections are derived through interface matching

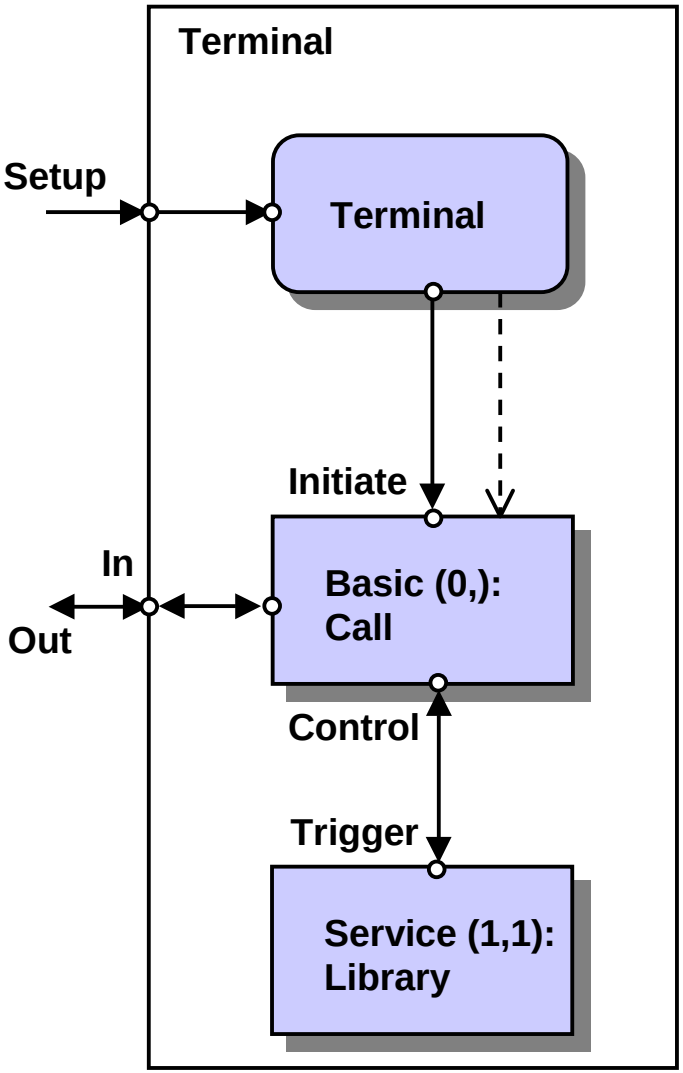


Run-time architectures (2)

“Proposals shall support the specification of the dynamic behavior of the internal structure of a classifier, including its connection to the state machine of the classifier, if any, its initial instantiation, as well as the dynamic addition and removal of parts and connections to/from the internal structure.”

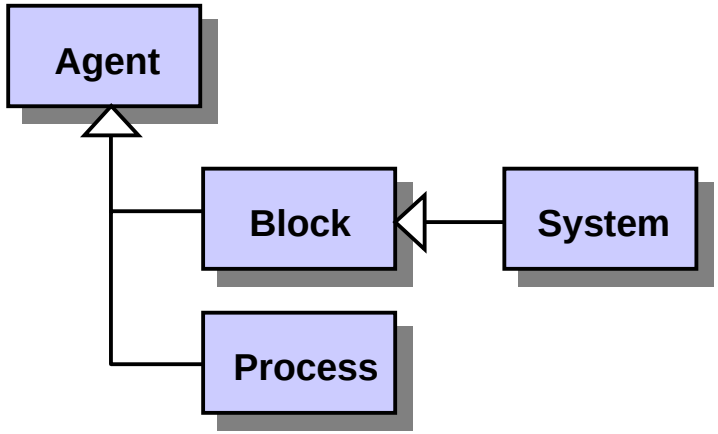
Dynamic Aspects of an Agent

- The internal structure of an agent also describe dynamic aspects of the agent
  - agent initialization
  - multiplicity requirements
  - state machine interaction
  - agent creation
  - life-cycle dependencies



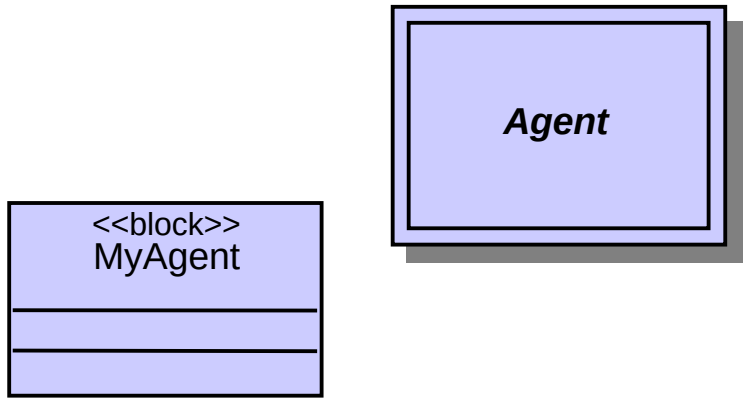
Agent Characterization

- We distinguish between different kinds of agents:
  - block
    - execute concurrently
    - distributed systems
  - process
    - execute alternatingly
    - shared memory
  - system (a top-level block)



Run-time Architecture

- UML 2 RFP asks for run-time architecture
- Hierarchical decomposition of structure
- Agent is the run-time architecture construct in SDL
  - provides encapsulation
    - a “black box” with required/implemented interfaces
  - has internal structure
    - subagents & channels
  - has behavior
    - through state machines

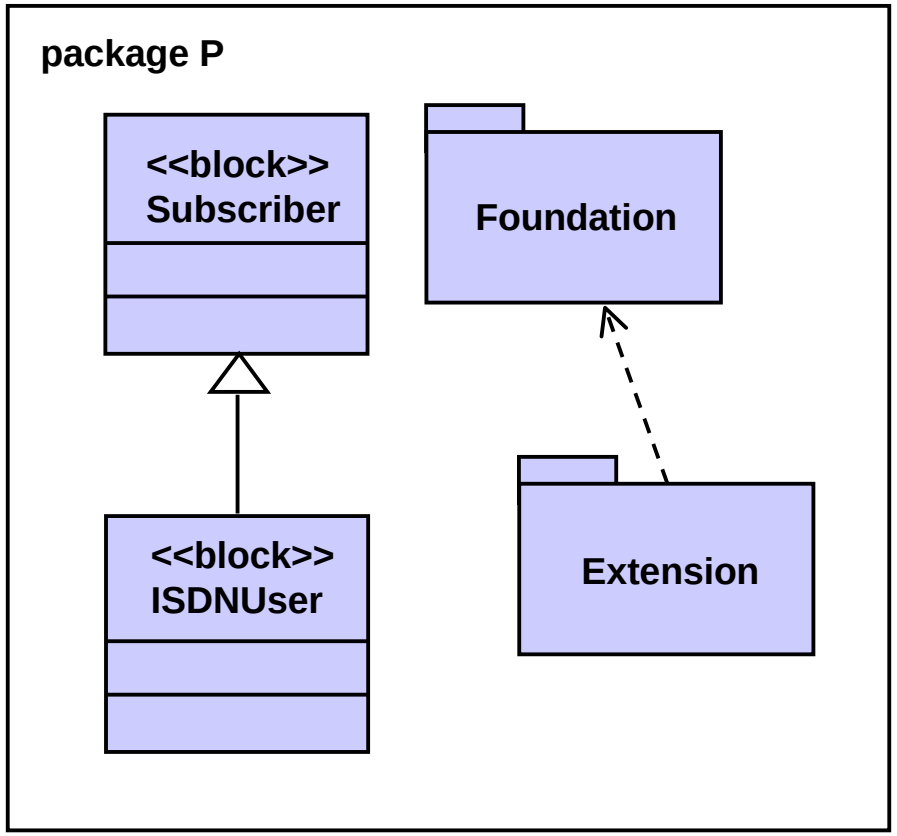


Challenges

- Managing software complexity
  - Viewpoints
  - Run-time Architecture
- **Managing project complexity**
  - **Work structuring**
  - **Contracts**
- Enabling quality and productivity
  - Visual software development

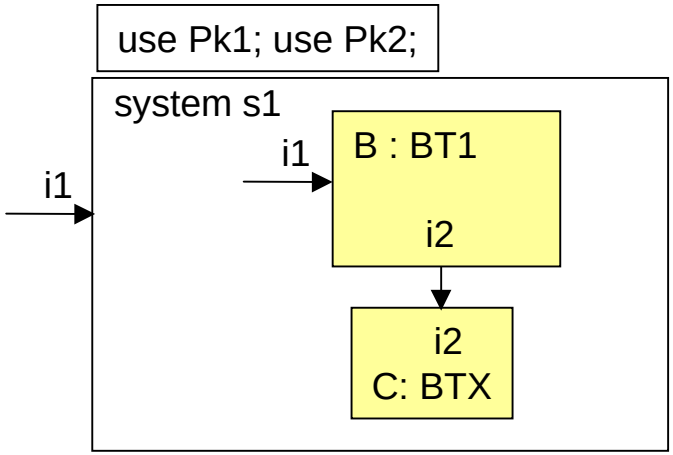
Packages

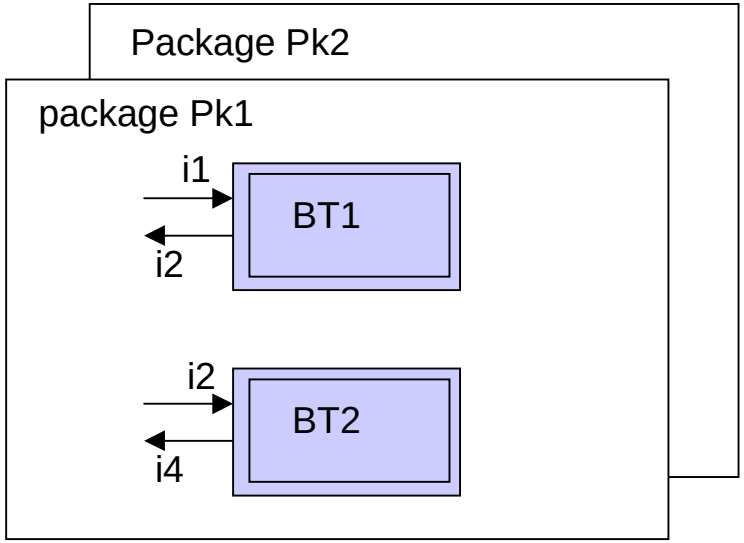
- Packages are used to organize a system into manageable and reusable work units
  - Name scopes
  - Usage dependencies
- Purpose is to allow team work.

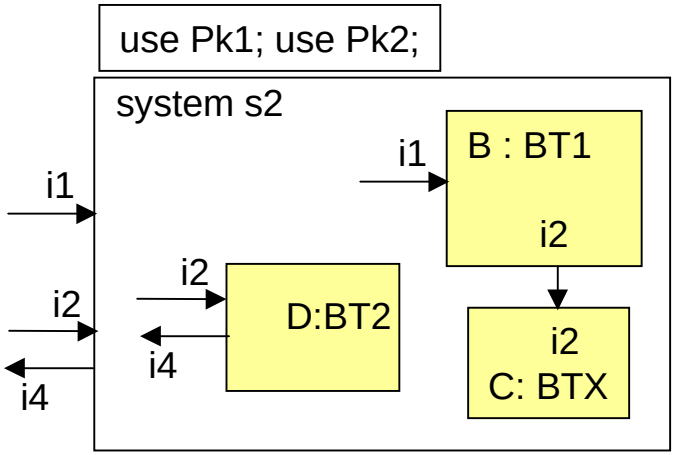


Packages, Systems and Agents

- Packages as libraries of reusable agents
- Systems are applications that instantiate agents

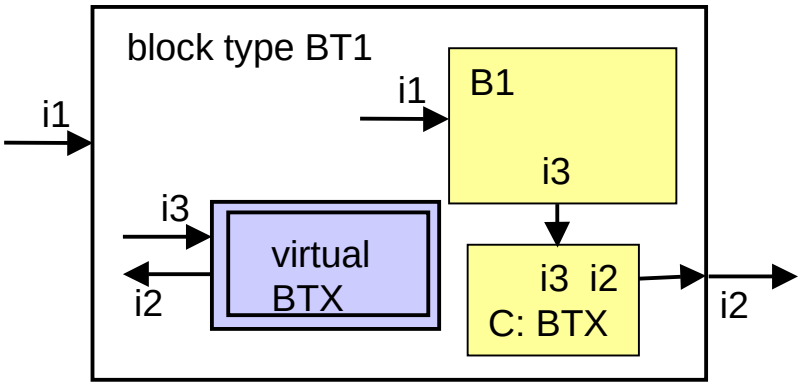


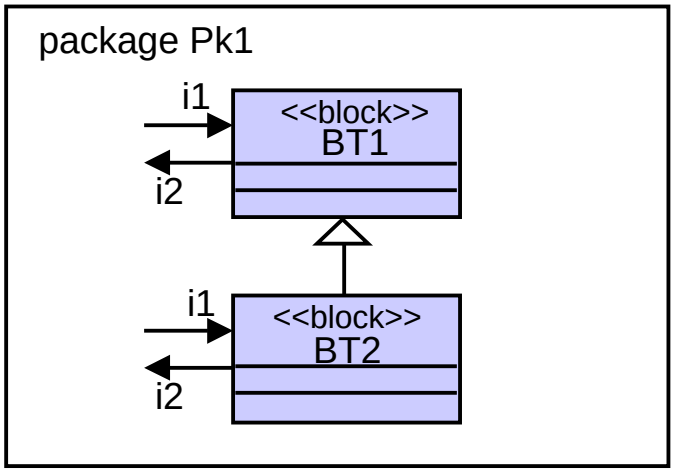


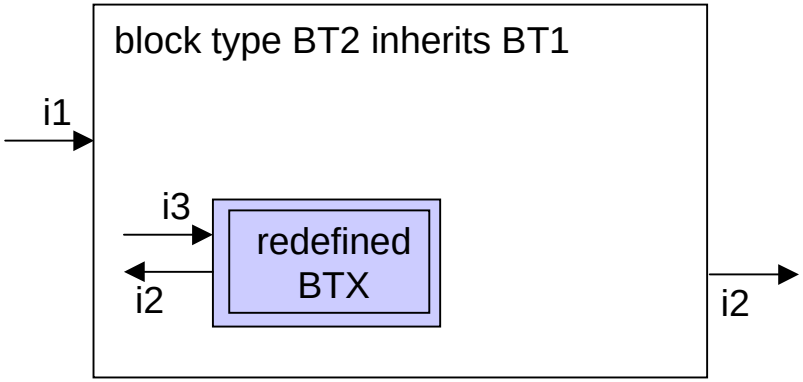


Frameworks & Specialization

- Specialization of architecture is the foundation of framework design





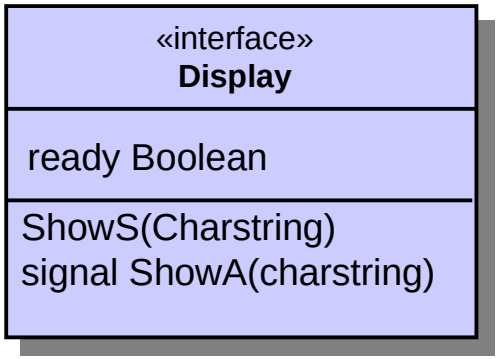


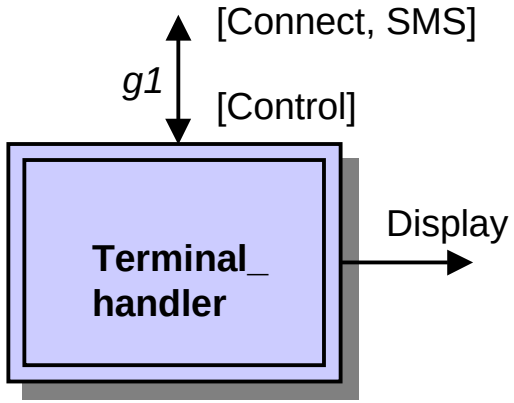
Challenges

- Managing software complexity
  - Run-time Architecture
  - Viewpoints
- **Managing project complexity**
  - **Work structuring**
  - **Contracts**
- Enabling quality and productivity
  - Visual software development

Static Interfaces

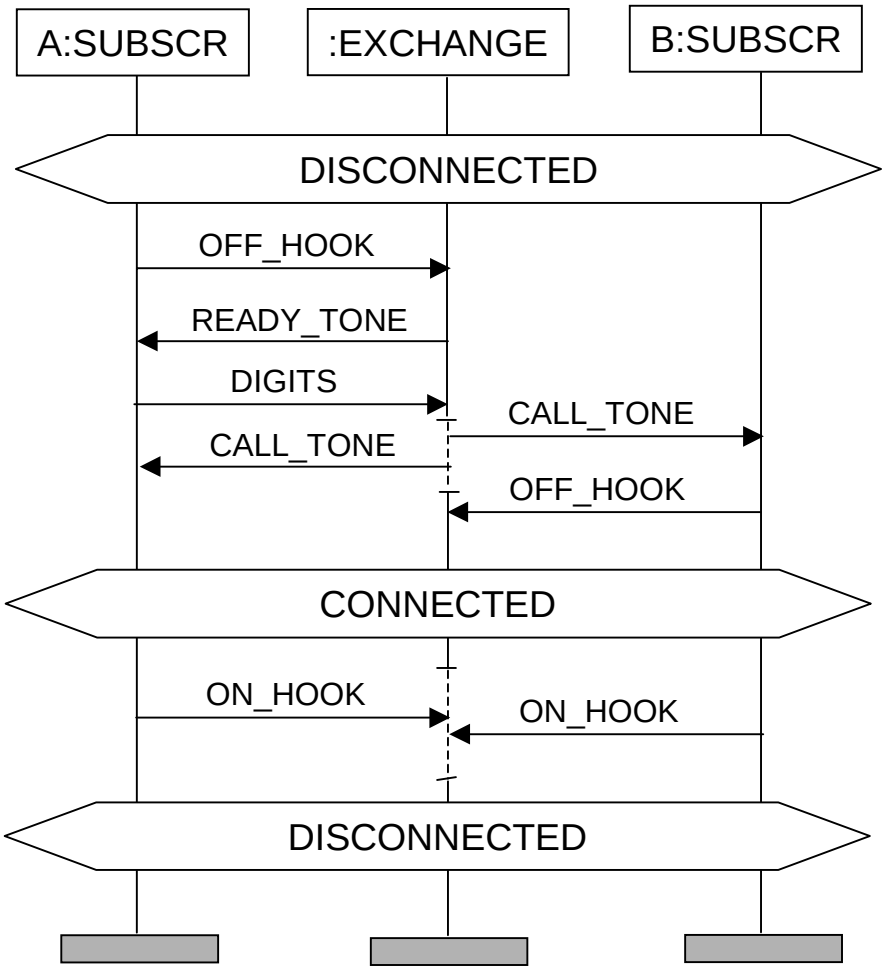
- An interface defines
  - attributes
  - operations
  - signals
- An agent has bi-directional gates that describe its
  - implemented interfaces
  - required interfaces
- The interfaces are used to specify contracts between agents & teams





Dynamic Interfaces

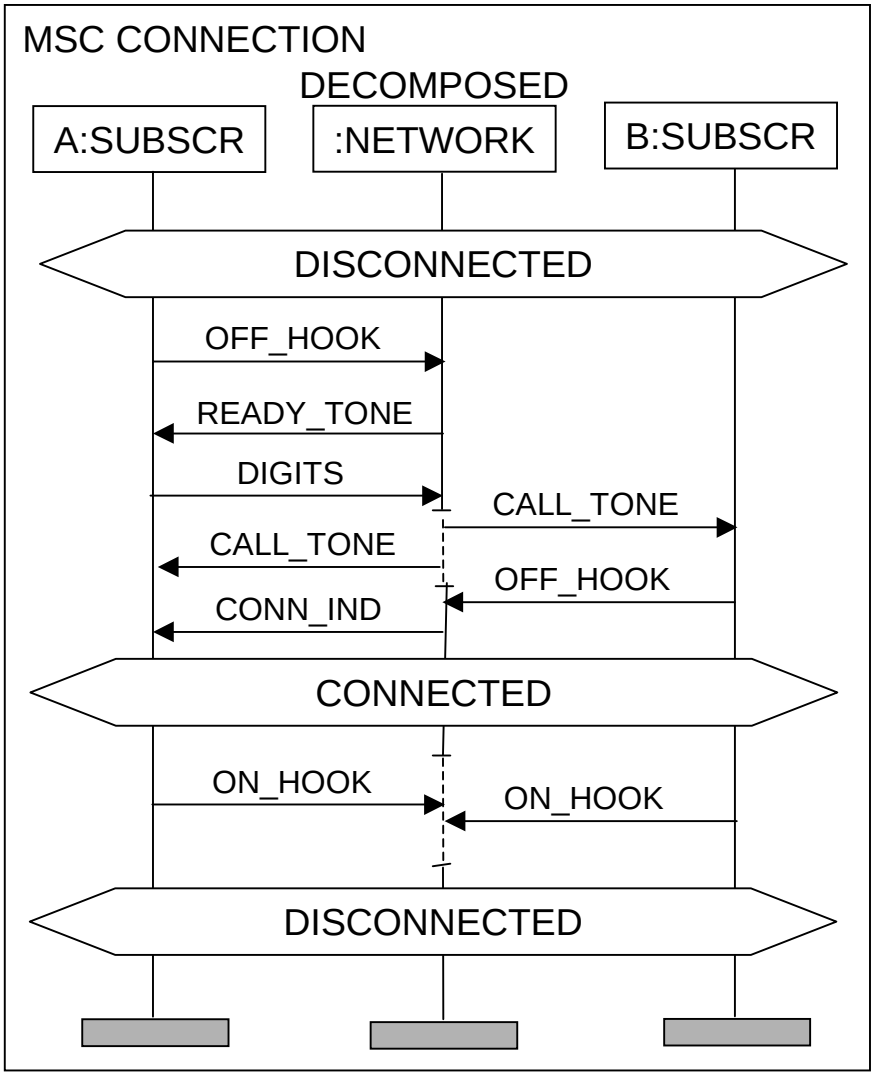
- Sequence diagrams / Message Sequence Charts
- Well-known, intuitive
- Formal trace semantics
- Requirements handling?
  - Scalability
  - Hierarchy

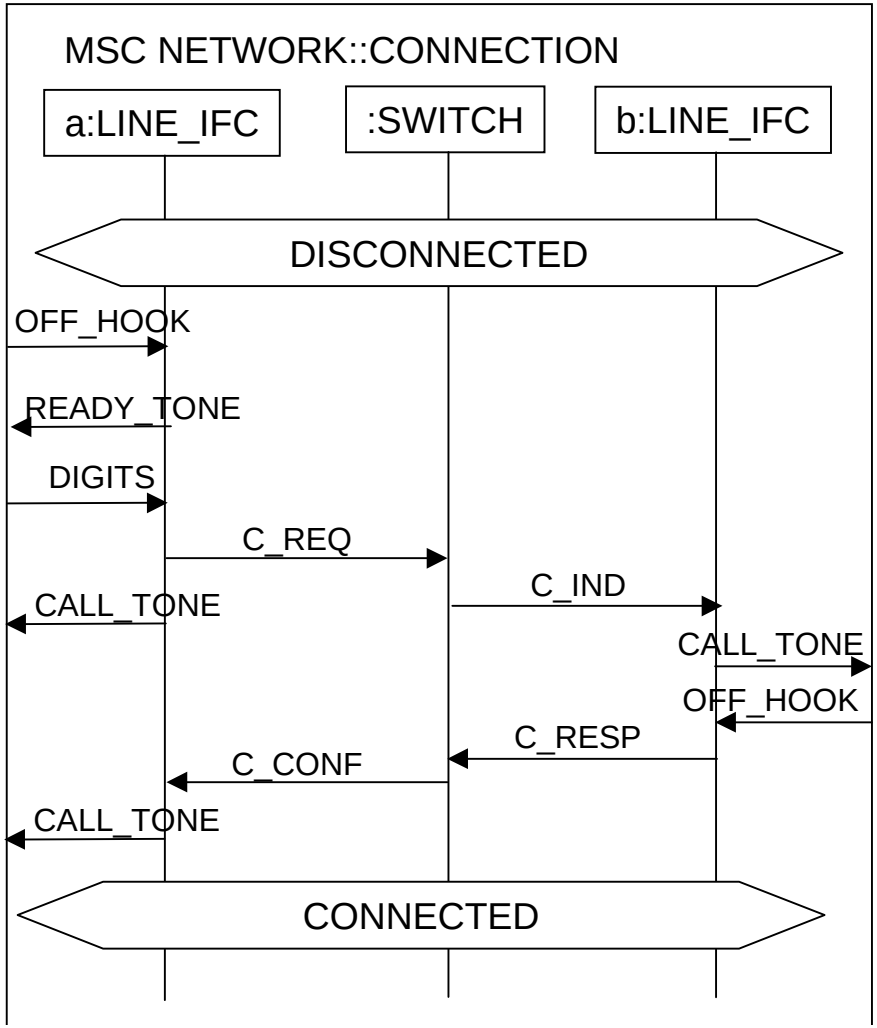


Interactions (1)

"Proposals shall define mechanisms to describe the decomposition of a role in an interaction into an interaction of its constituent parts."

Decomposition of MSCs





Interactions (2)

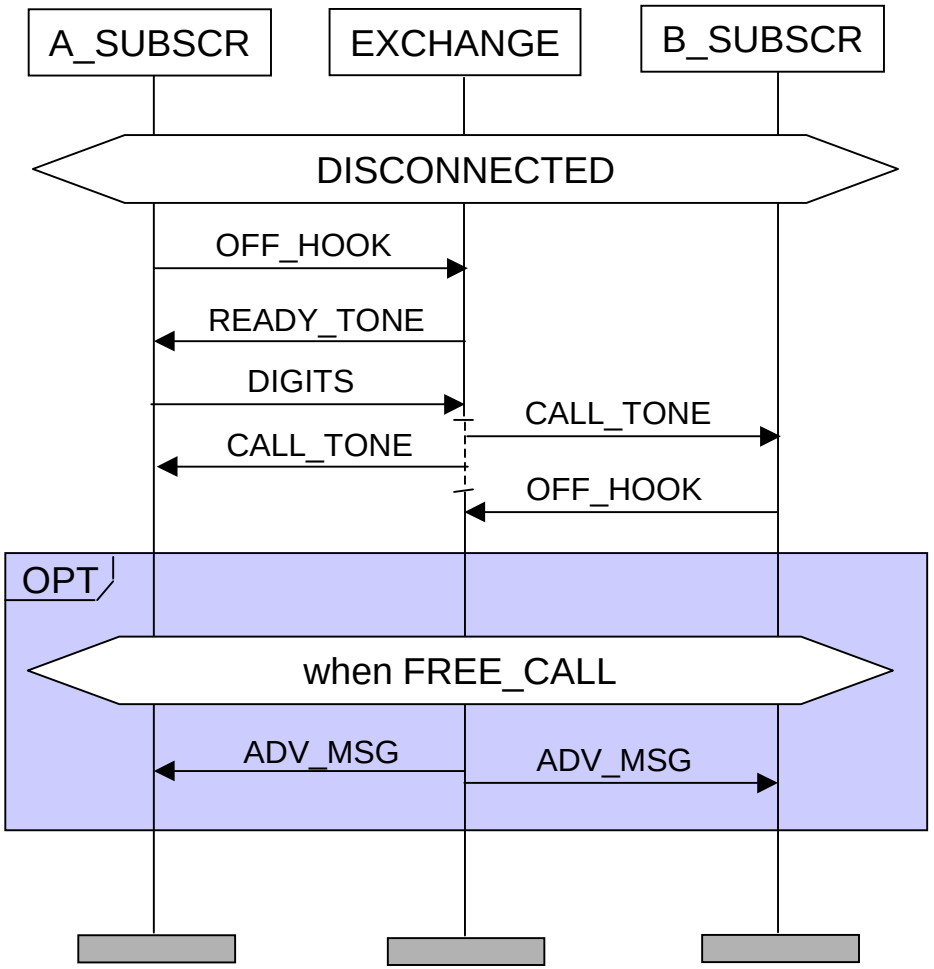
"The current sequence diagram notation offers little help to structure specifications using sequence diagrams …

•Modelers have limited ability to define variability within a single sequence diagram."

Inline Expressions

Compact notation for handling of minor variations

- alternatives
- optional parts
- repetitions
- exceptions

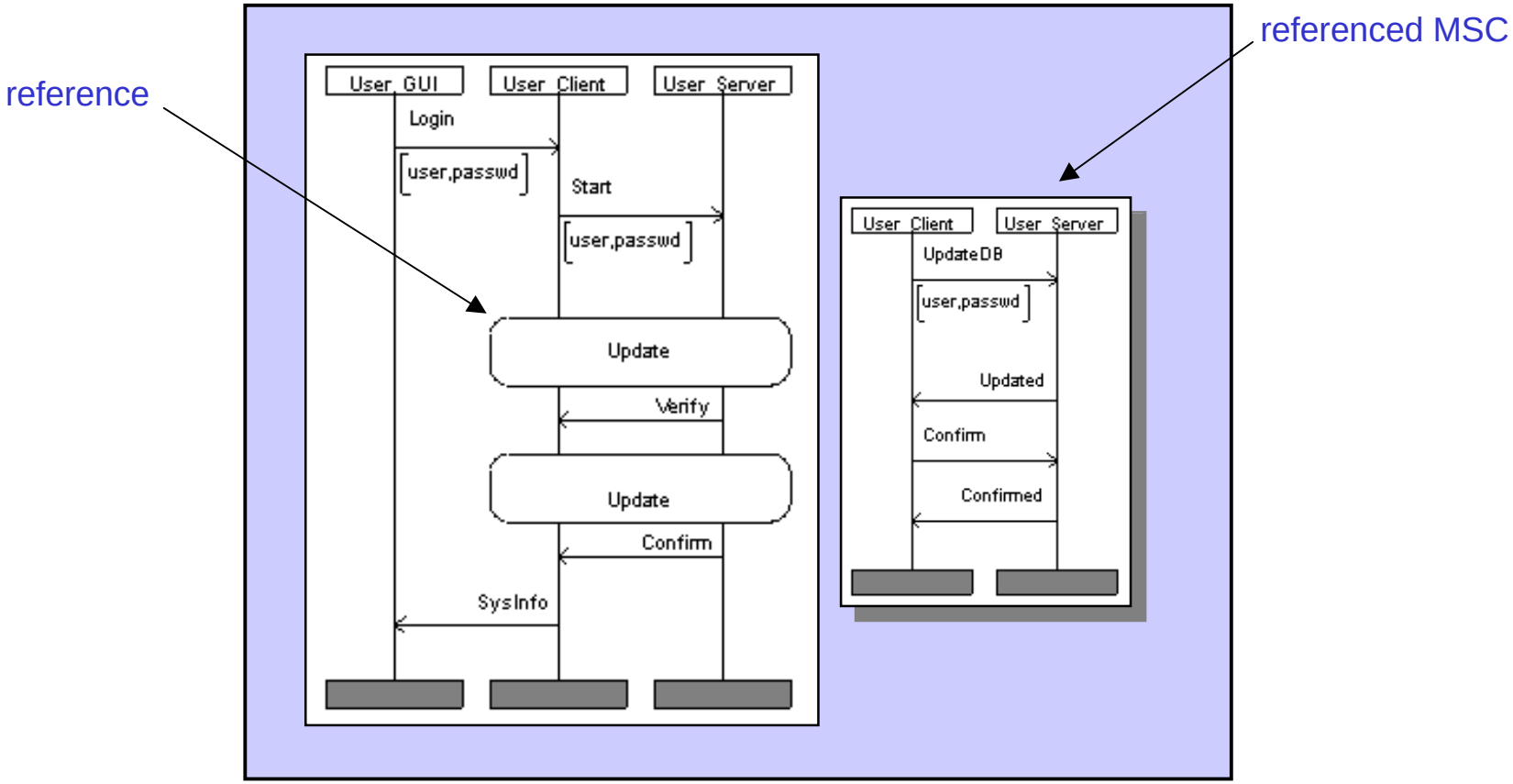


Interactions (3)

"The current sequence diagram notation offers little help to structure specifications using sequence diagrams …

•Modelers cannot reference one sequence diagram from another."

Sequence Diagrams & Reuse: MSC References

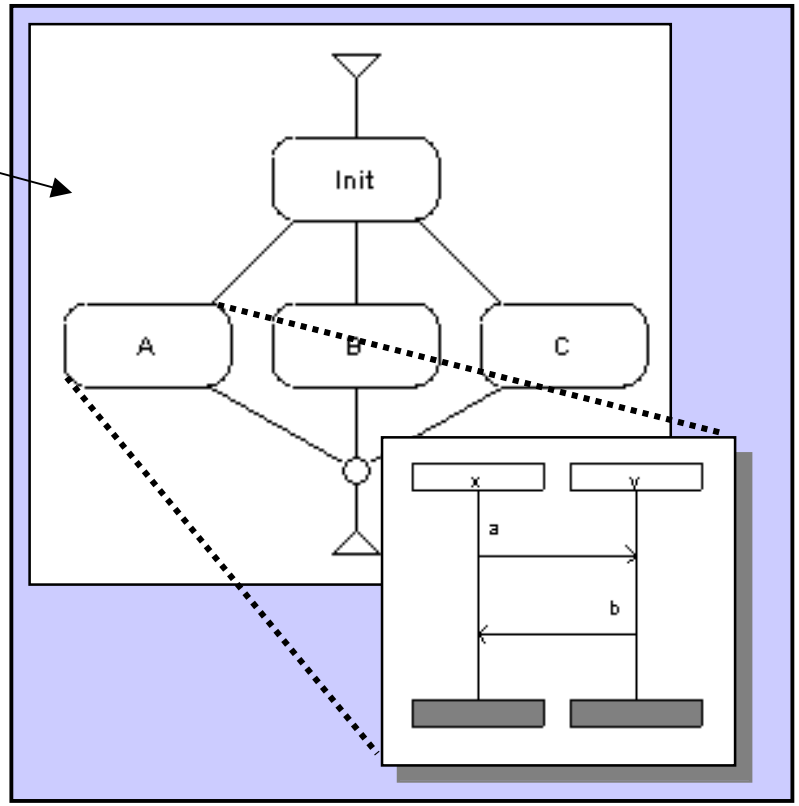


Interactions (2)

"The current sequence diagram notation offers little help to structure specifications using sequence diagrams nor does it provide an overview of how these sequence diagrams are related to each other."

Organizing Sequence Diagrams: High-level MSCs

structuring of MSCs



a compact way of describing several MSCs

extremely useful for large requirements specifications

Managing Complex Projects

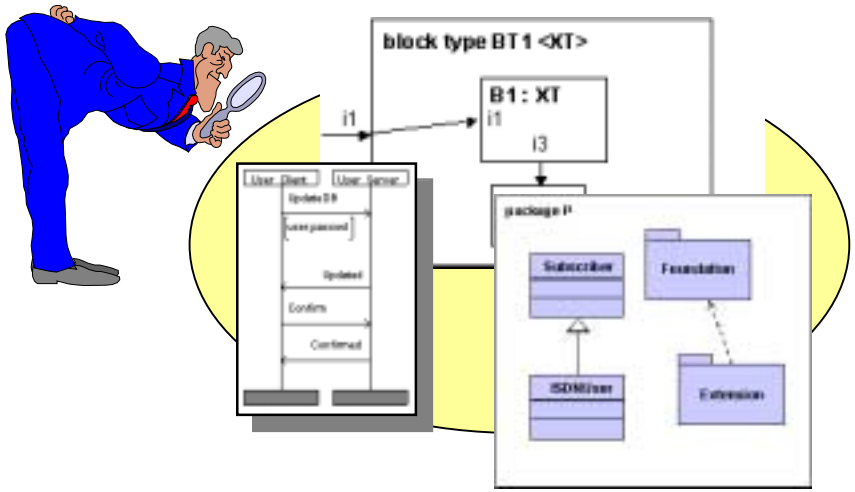
- Establish team responsibilities
  - packages
  - reusable agents, frameworks
- Establish team interfaces
  - static: interfaces
  - dynamic: sequence diagrams
    - high-level MSCs
    - MSC references
    - inline expressions
    - decomposition

Challenges

- Managing software complexity
  - Run-time Architecture
  - Viewpoints
- Managing project complexity
  - Work structuring
  - Contracts
- **Enabling quality and productivity**
  - **Visual software development**

Visual Modelling

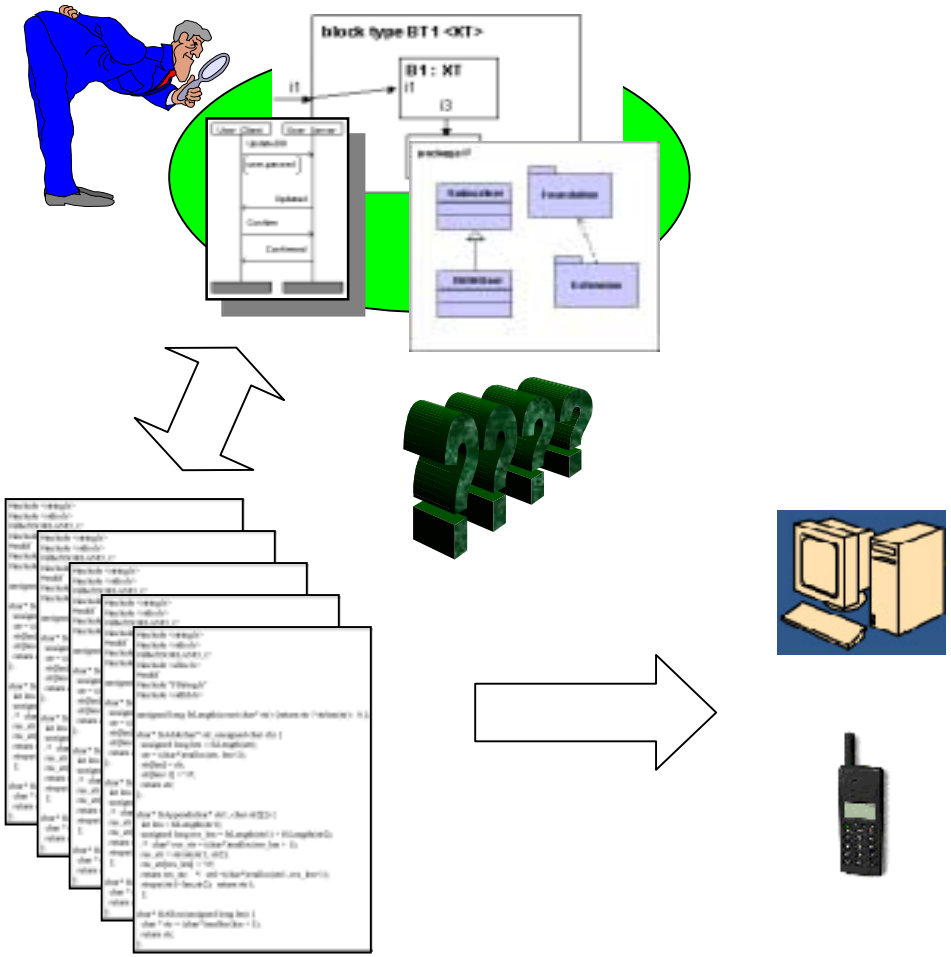
- Requirements
  - MSCs
  - Use case diagrams
- Application architecture
  - Agents
- Static Structure
  - Class diagrams
- Work structure
  - Package diagrams
- Physical architecture
  - Deployment/Component diagrams



Enforces quality in the product
and skill in the development teams

Visual Modeling - The Problem

- Make sure that what you see is what you get!



Visual Software Development

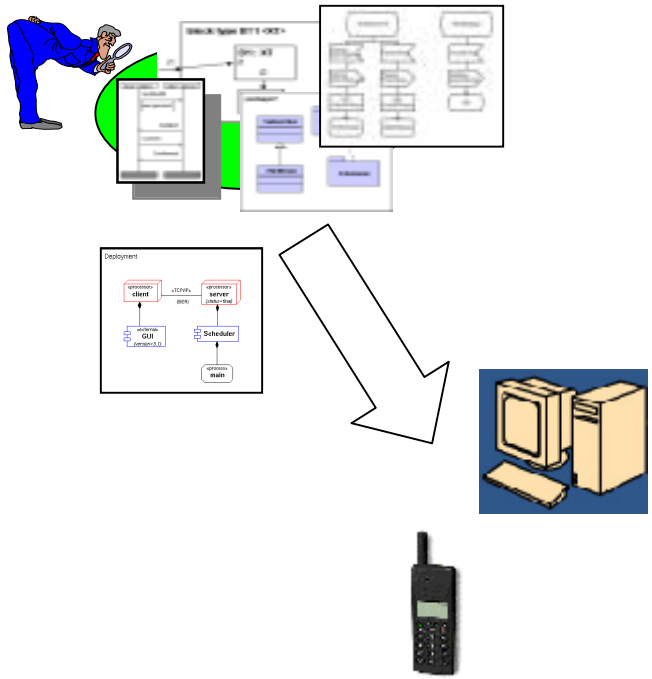
Visual software development

= Visual modeling

+ Behavioral modeling

- action notation
- data model

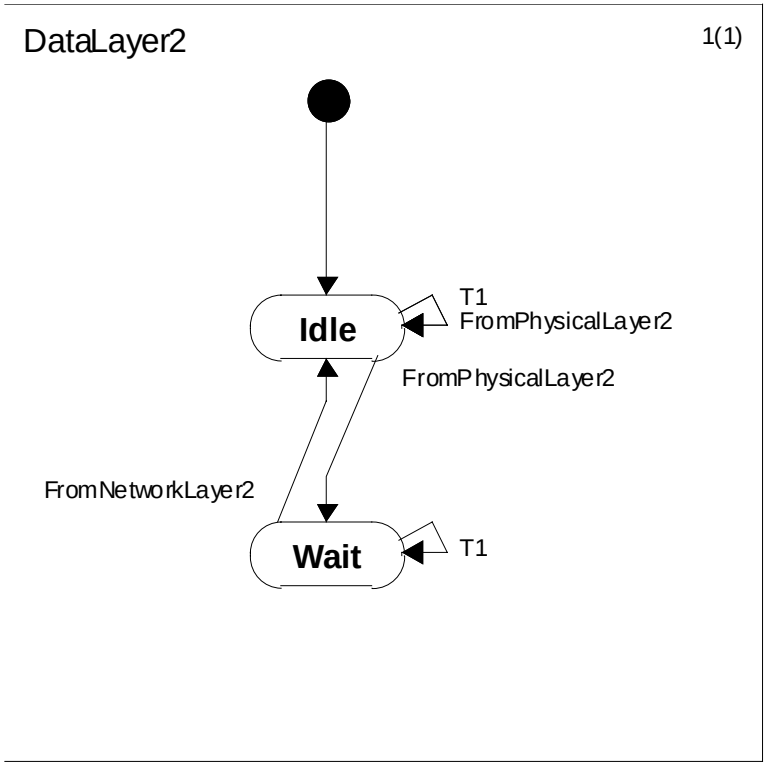
+ Run-time semantics



- Consequences:
  - High-level application specific modeling
  - Automated application generation
  - The diagrams are both documentation and source code

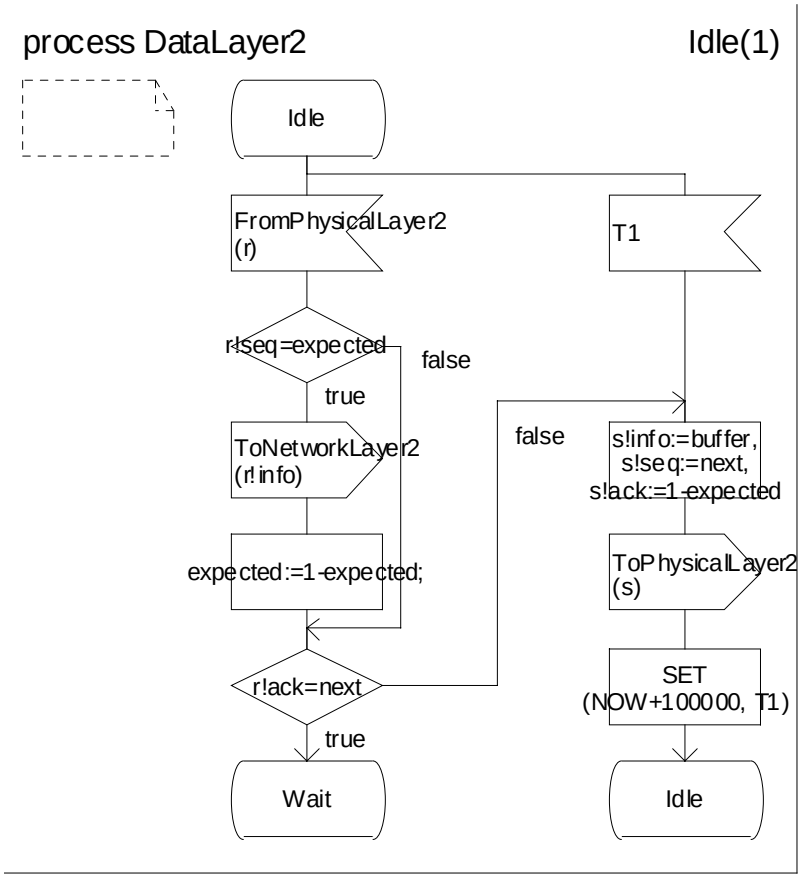
State-Centric State Machines

- State charts / state overview
- Provide overview
- Focus on states
  - transitions between states



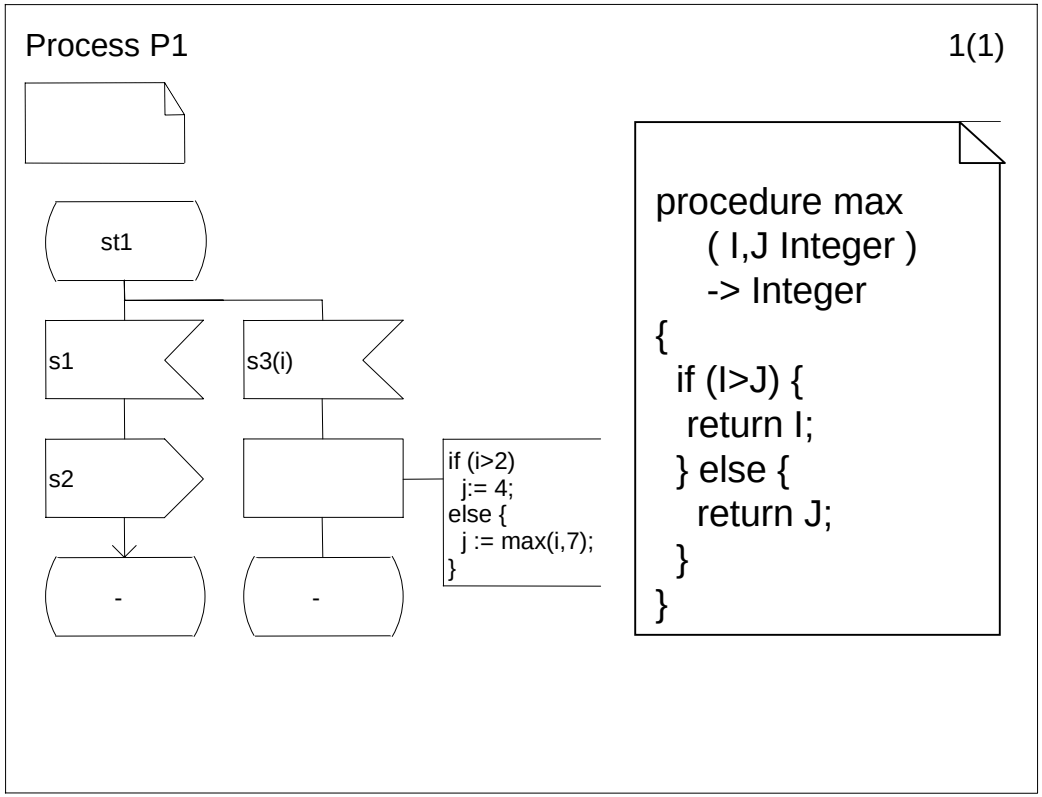
Transition-Centric State Machines

- Activity diagrams / State machines
- Provide detailed view
- Focus on transitions
  - state to state behavior
  - actions
  - control flow
  - communication



Textual Algorithms

- Focus on data manipulation and algorithms



Model Execution

- Behaviour + run time semantics

= Model execution

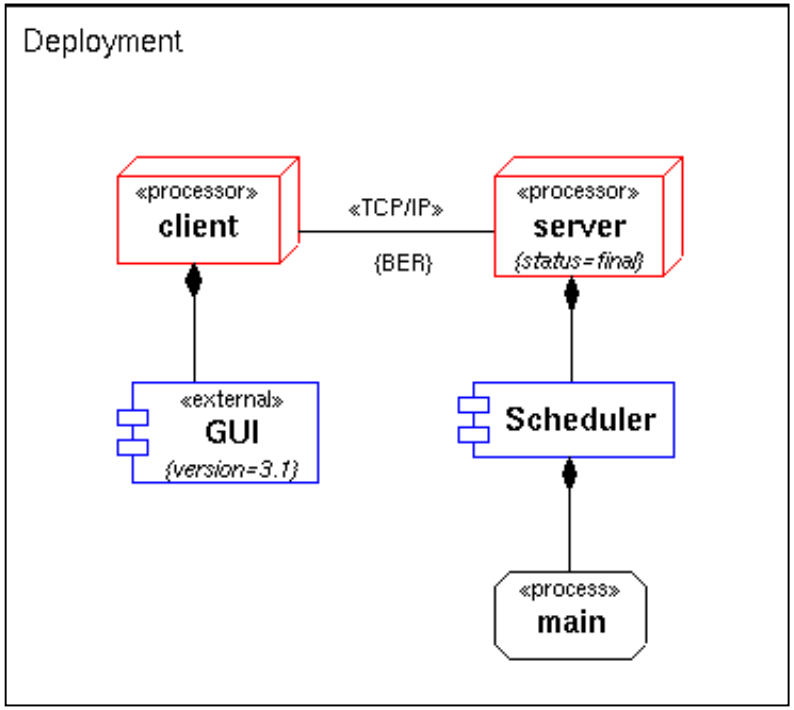
enables

- Execution based refinement
- Early verification
- Feedback directly in model
- Testing of incomplete systems

Focus on logical design

From Model to Application: Implementation Diagrams

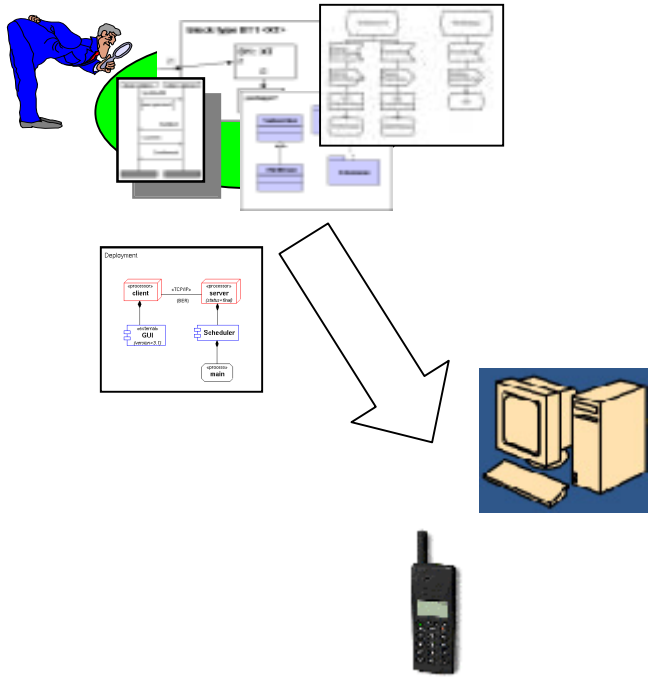
- Nodes / Components / Connections
- Physical component vs. agent mapping
- Thread policies …
- Defines build process



Focus on physical design

Visual Software Development

- Quality: Focus on understanding
  - Visual modeling
  - Model execution
  - Documentation is source code
- Productivity: Automation & specialization
  - High-level application specific modeling
  - Automated application generation



Challenges

- Managing software complexity
  - Viewpoints
  - Run-time architecture
- Managing project complexity
  - Work structuring
  - Contracts
- Enabling quality and productivity
  - Visual software development