Domain Engineering – Using Domain Concepts to Guide Software Design

Iris Reinhartz-Berger

Department of Management Infromation Systems
University of Haifa, Israel

Arnon Sturm

Department of Information Systems Engineering
Ben-Gurion University of the Negev, Israel

Yair Wand

Sauder School of Business
University of British Columbia, Canada

Tutorial 3

Agenda (1)

- **Part 1: Domain Engineering**
  - The Landscape of Reuse
  - The Landscape of Knowledge Engineering & Management
  - The Landscape of Validation & Verification
  - Domain Engineering as an Intersection Point
    - Domain Definitions
    - Domain Engineering Definitions
    - Domain Engineering Activities
    - Domain Engineering & Application Engineering
  - Domain Analysis Techniques
    - Feature-Oriented Techniques: FODA and PLUS
    - Metamodeling Techniques: GME and metaEdit+
  - Domain Engineering Advantages & Limitations

Agenda (2)

- **Part 2: The Application-based Domain Modeling (ADOM) Approach**
  - Motivation
  - General Architecture
  - ADOM and UML through Examples
  - Experimental results with ADOM
  - The Current State of ADOM
  - ADOM Advantages
  - ADOM Limitations & Future Work

Part 1:
Domain Engineering

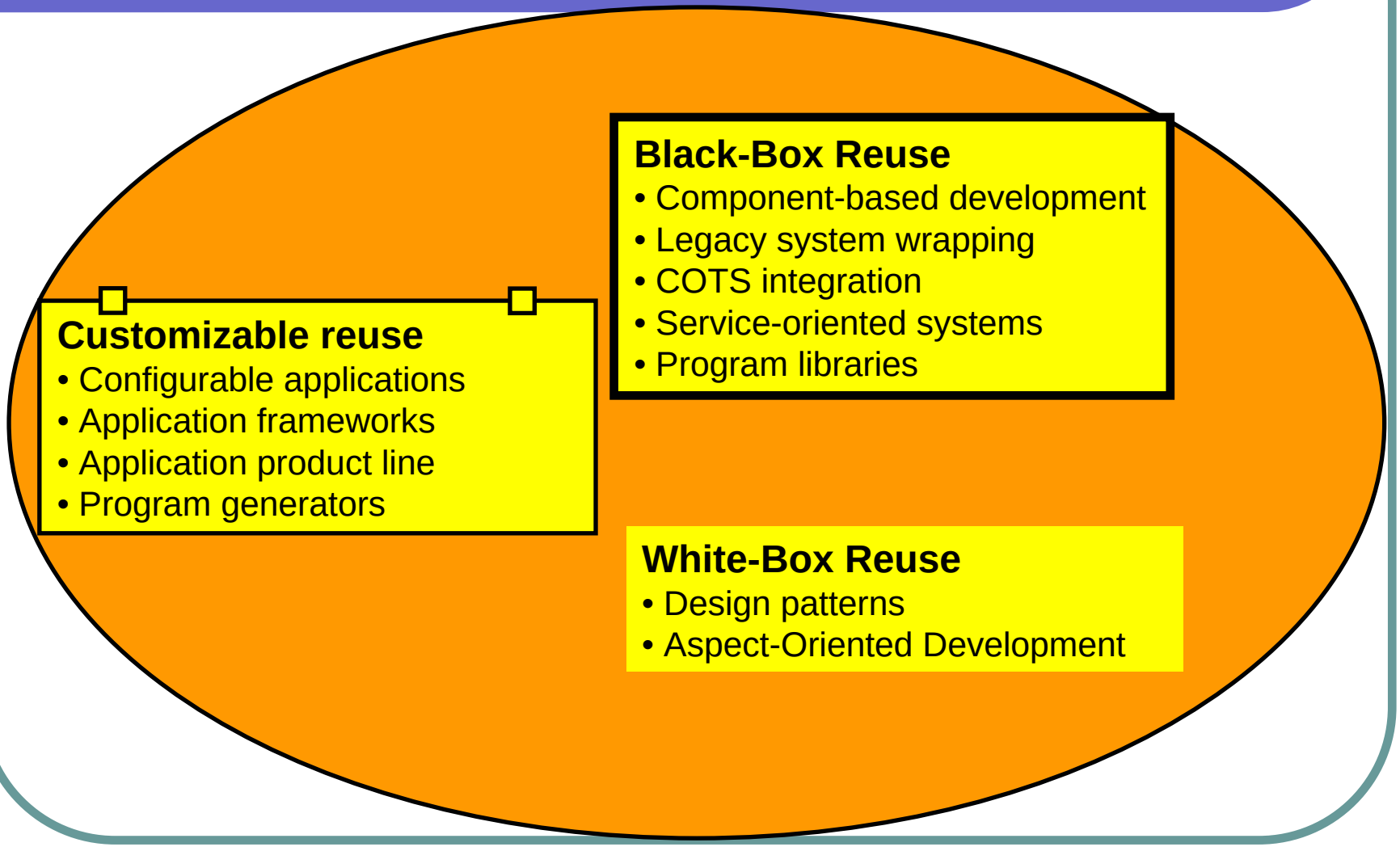
Tutorial 3

The Landscape of Reuse

Wikipedia

- ***Reuse*** is the activity of using a segment of an artifact (code, design, requirements, etc.) again in different context.
- The benefits of reuse include:
  - Reducing development time
  - Eliminating the likelihood of bugs and errors
  - Localizing modifications when a change in implementation is required
  - Standartization
- Desired features for software artifacts intended to be reused include:
  - Adaptable, Brief (small size), Flexible, Parameterization, Generic, Fast, Simple (low complexity), Localization of volatile (changeable) design assumptions, Modularity

The Landscape of Reuse

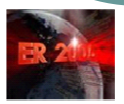


A Comparison of Reuse Approaches

	CBD	Legacy system wrapping	Service-oriented	COTS systems	Program libraries
Basic Elements	Component, independent executable entity	An interface	Service, unit of work done by a service provider to achieve desired end results for a service consumer	Commercial, Off-The-Shelf products	Classes and functions
Features	General, structural& functional	Tailored interfaces	General-purpose, functional, business-oriented	General, sometimes parameterized	General purpose
Size	Varies from simple functions to entire applications	Varies from simple functions to entire applications	Simple or complex functions	Complete systems	Usually does not make an application
Way of reuse	Through interfaces	Through interfaces	Through interfaces	Activation, through API	Utilize existing functions

A Comparison of Reuse Approaches

	Design Patterns	Aspect-Oriented	Config. App.	App. frameworks	App. product lines	Program generators
Basic Elements	Pattern, description of a problem and the essence of its solution	Aspect, concern	Configuration mechanism	Source code	Any software artifact	Transformation rules
Features	Very abstract and general, functional	Functional, cut-across system structure	Specific	Abstract and general, object-oriented	Varies	Domain specific
Size	Very small	Varies	Varies from simple to entire applications	Varies from simple functions to entire applications	Varies from simple functions to entire applications	Varies from simple functions to application skeletons
Way of reuse	Matching the specific problem and then apply its solution	Through weavers	Change of configuration	Usually through inheritance and polymorphism	Customize according to specific needs	Create automatic code

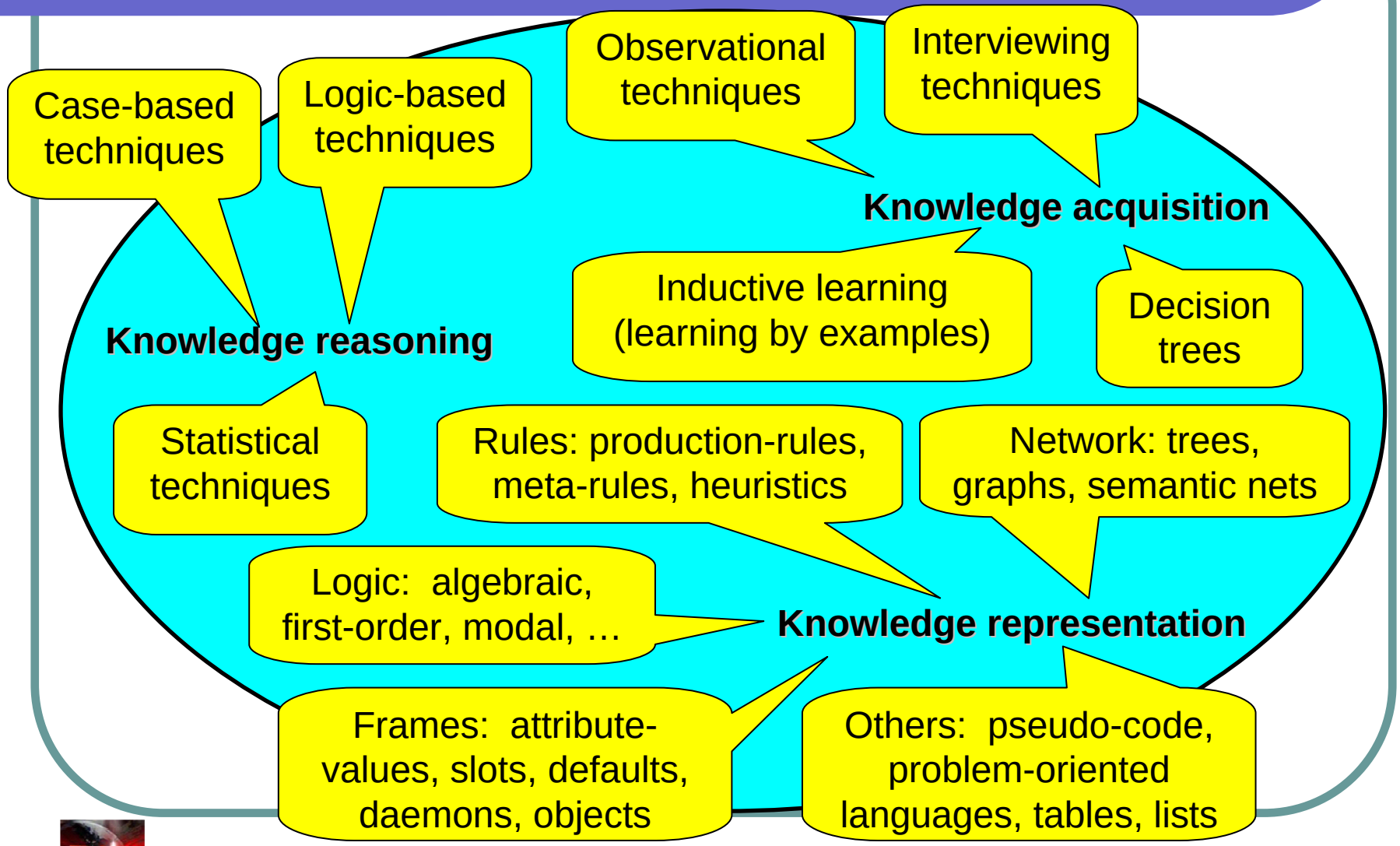


The Landscape of Knowledge Engineering

Knowledge-Based System Analysis And Design – A Kads Developer's Handbook by Tansley & Hayball, Prentice Hall, 1993

- ***Knowledge*** is a rich form of information, often stored by humans as expertise in some restricted domain.
- ***Knowledge engineering*** covers the activities involved with obtaining knowledge and creating formal representation of the obtained knowledge. It includes:
  - ***Knowledge acquisition/elicitation*** – the activity of extracting or filtering relevant domain knowledge from an expert or user
  - ***Knowledge representation*** – the activity that uses a notation or formalism for coding the knowledge to be stored
  - ***Knowledge reasoning*** – the activity of inferring new knowledge from existing information and knowledge

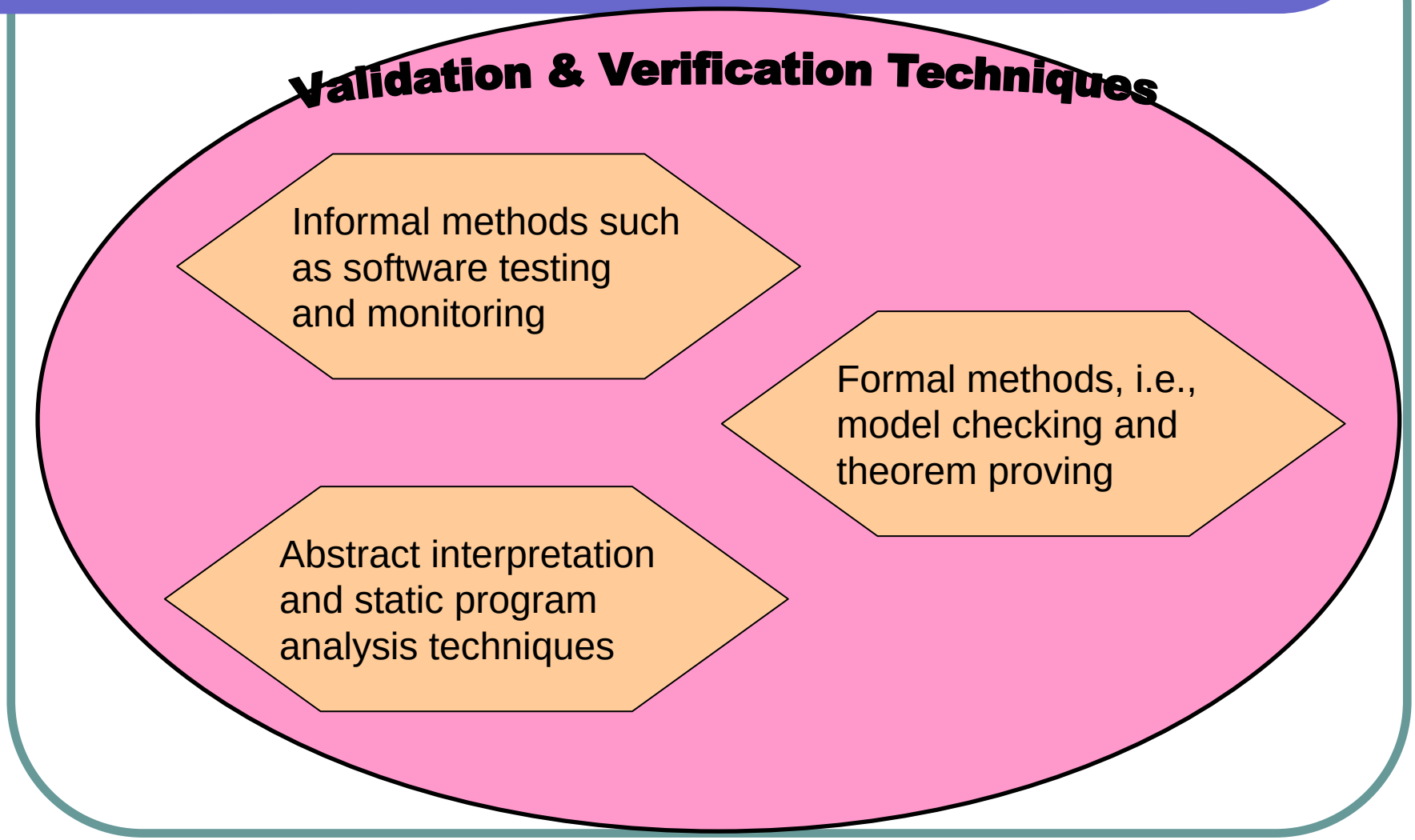
The Landscape of Knowledge Engineering



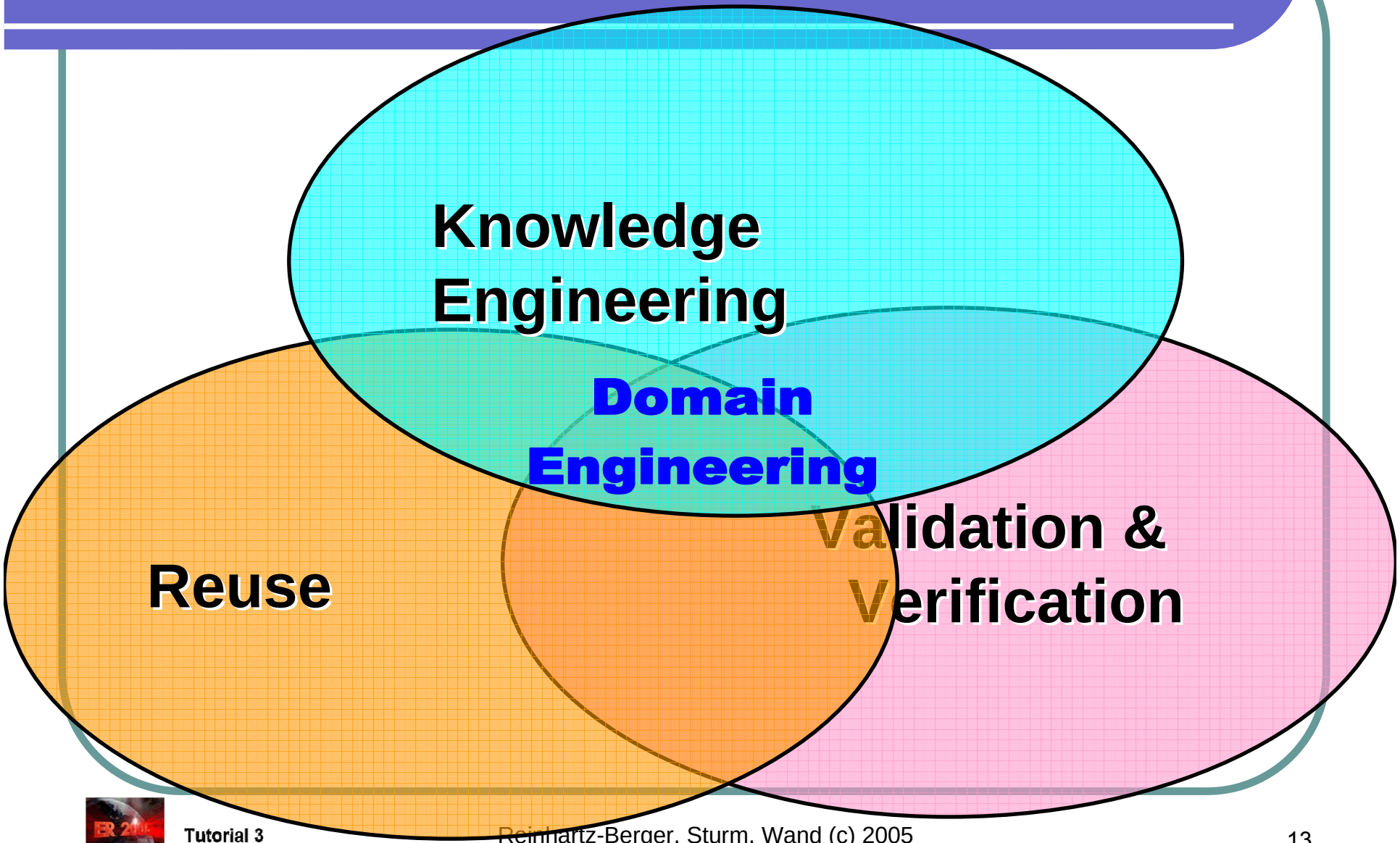
The Landscape of Validation & Verification

- ***Validation*** is the process of determining the degree to which a model is an accurate representation of the real world from the perspective of the intended uses of the model (AIAA G-077-1998).
  - The purpose of validation is building the right system
  - It is partially done by getting client (and other stakeholders) feedback
- ***Verification*** is the process of determining that a model implementation accurately represents the developer's conceptual description of the model and the solution to the model (AIAA G-077-1998).
  - The purpose of verification is developing the right building
  - It is partially done by monitoring the development process

The Landscape of Validation & Verification



Domain Engineering as an Intersection Point



Examples for the use of the word Domain

The Free Dictionary by Farlex

1. In a LAN, a sub-network made up of a group of clients and servers under the control of one security database. Dividing LANs into domains improves performance and security.
2. In a communications network, all resources under the control of a single computer system.
3. On the Internet, a registration category (domain name).
4. In database management, all possible values contained in a particular field for every record in the file.
5. A group of end points (phones or gateways) in a SIP telephony environment.
6. In magnetic storage devices, a group of molecules that makes up one bit.
7. In a hierarchy, a named group that has control over the groups under it, which may be domains themselves.

Our Definition of Domain

- A ***domain*** is an area of knowledge characterized by a set of concepts, their relationships, and constraints accepted by practitioners in that area.
  - a set of applications that use a common jargon for describing the concepts, problems, and solutions
  - a class of similar systems that share common features and operations
- Other names for the term ‘domain’: ‘product line’, ‘product family’, …
- Domain attributes: mature, stable, economically viable
- Domain examples: Telephone switches, Insurance portals, Customer Relation Management, Online banking

Domain Engineering Definitions

- ***Domain engineering*** is the development and evolution of domain specific knowledge and artifacts to support the development and evolution of systems in the domain.
  - The purpose of domain engineering is to identify, model, construct, catalog, and disseminate the *commonalities* and *differences* of particular domain applications
  - Domain engineering includes engineering of domain models, components, methods and tools
  - Domain engineering includes three main activities: domain analysis, domain design, and domain implementation

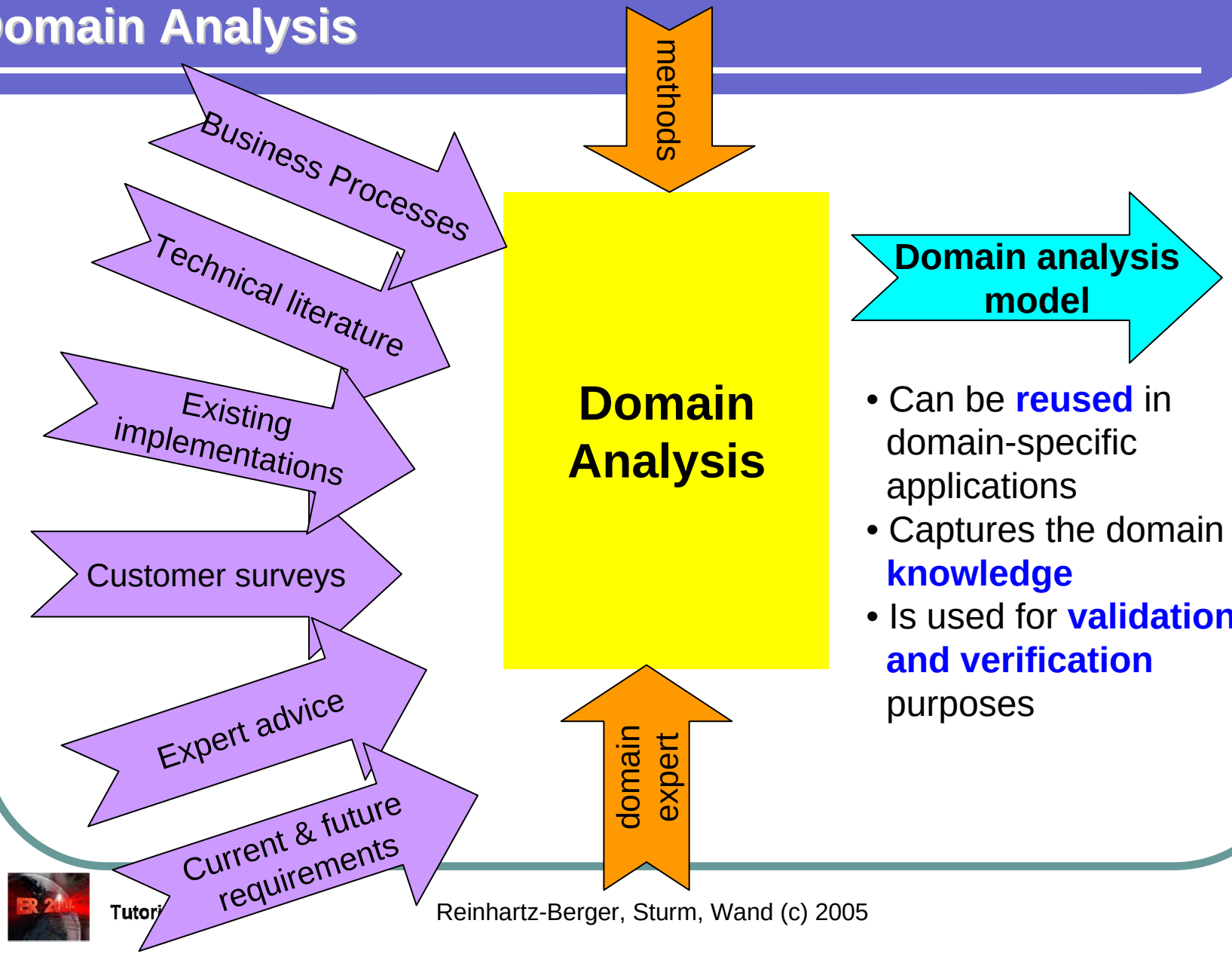
Domain Engineering Activities
Domain Analysis

- ***Domain analysis*** (The Free Dictionary by Farlex)
  - Is the domain engineering activity in which domain knowledge is studied and formalized as a domain definition and a domain specification
    - Relates to non-implementation issues
  - Is the process of identifying, collecting, organizing, analyzing and representing a domain model and software architecture from the study of existing systems, underlying theory, emerging technology and development histories within the domain of interest
  - Is the analysis of systems within a domain to discover commonalities and differences among them

Domain Engineering Activities
Domain Analysis

- Domain analysis includes:
  - Set objectives: identify stakeholders and their objectives
  - Scope domain: define selection criteria
  - Define domain: identify boundary conditions, examples, counter examples, main features
  - Define relations: define relations to other domains, divide the domain into sub-domains
  - Acquire domain information from experts, legacy systems, literature, prototyping
  - Describe domain terminology: lexicon of terms, commonality & variability, features
  - Refine domain: build overall domain, analyze trade offs, innovative feature combinations
- A common way for carrying out domain analysis is modeling

Domain Engineering Activities
Domain Analysis

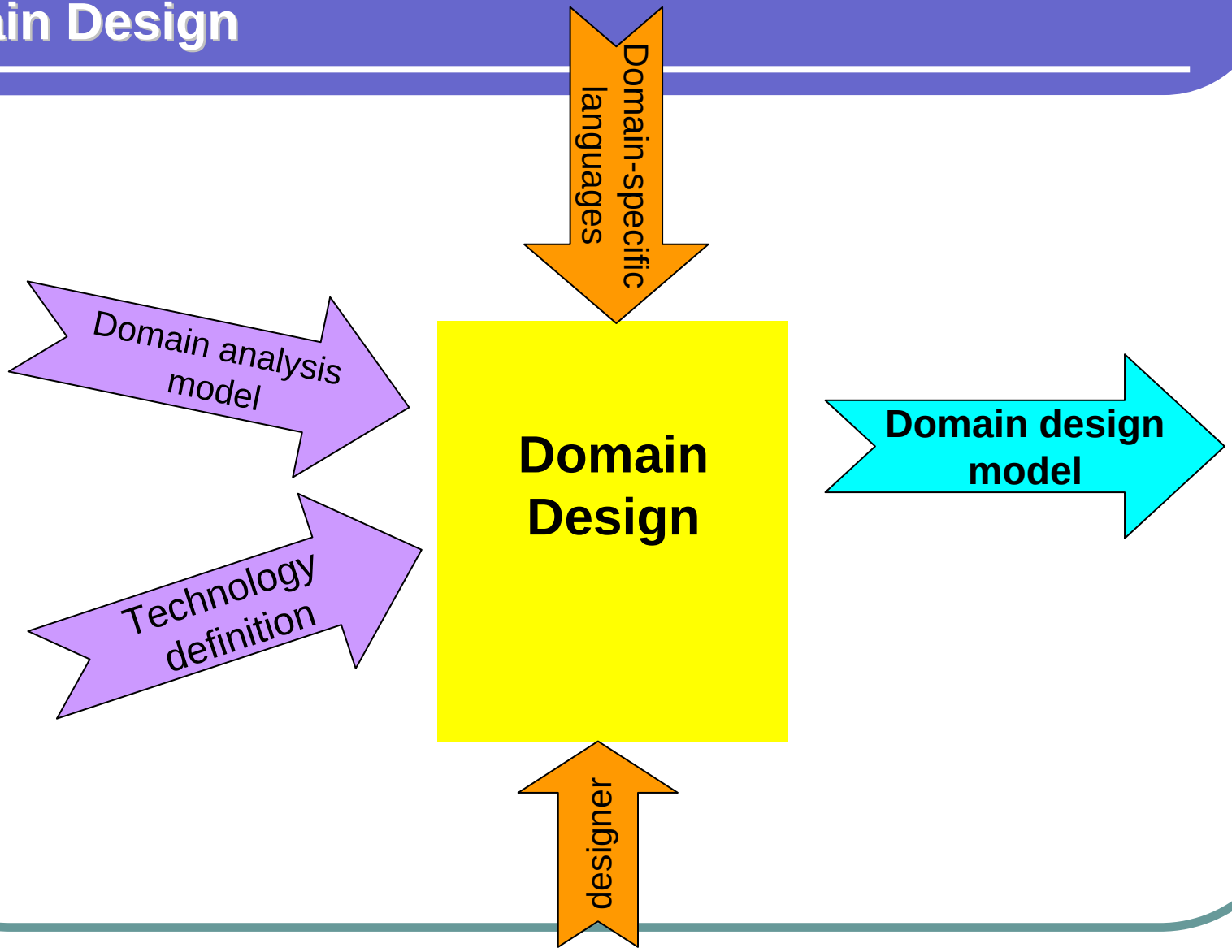


- Can be **reused** in domain-specific applications
- Captures the domain **knowledge**
- Is used for **validation and verification** purposes

Domain Engineering Activities
Domain Design

- ***Domain design*** the activity that takes the results of domain analysis to identify and generalize solutions for those common requirements in the form of a Domain-Specific Software Architecture (DSSA) Carnegie-Mellon Software Engineering Institute.
  - It focuses on the problem space, not just on a particular system's requirements, to design a solution (solution space)
  - It supports developing a common architecture for the system in the domain and devising a production plan
  - The domain design model should represent a generic architecture developed for supporting activities in the analyzed domain and provide the framework for the development of reusable components in domain implementation

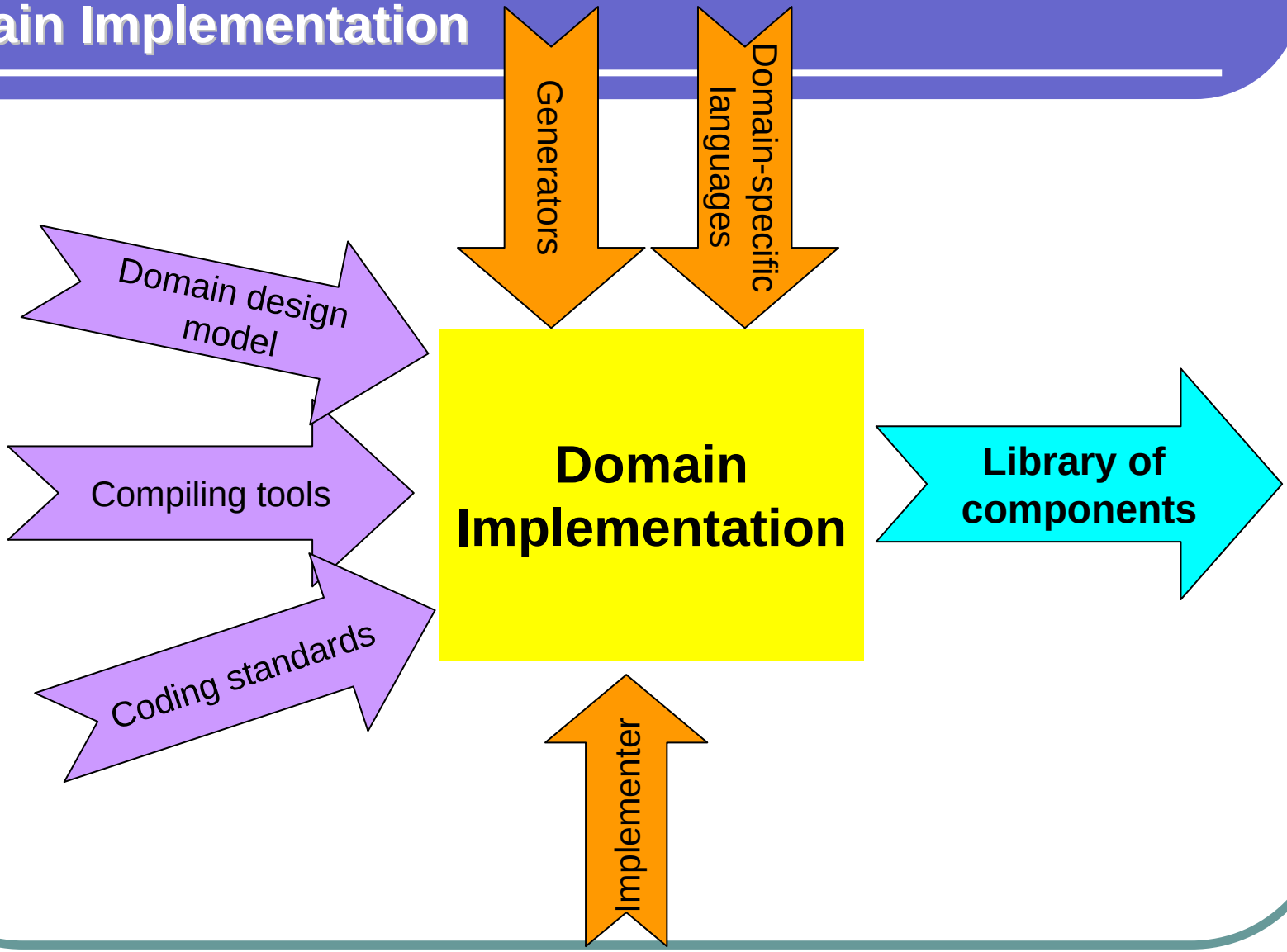
Domain Engineering Activities
Domain Design



Domain Engineering Activities
Domain Implementation

- ***Domain implementation*** is the process of identifying ***reusable components*** based on the domain model and generic architecture Carnegie-Mellon Software Engineering Institute.

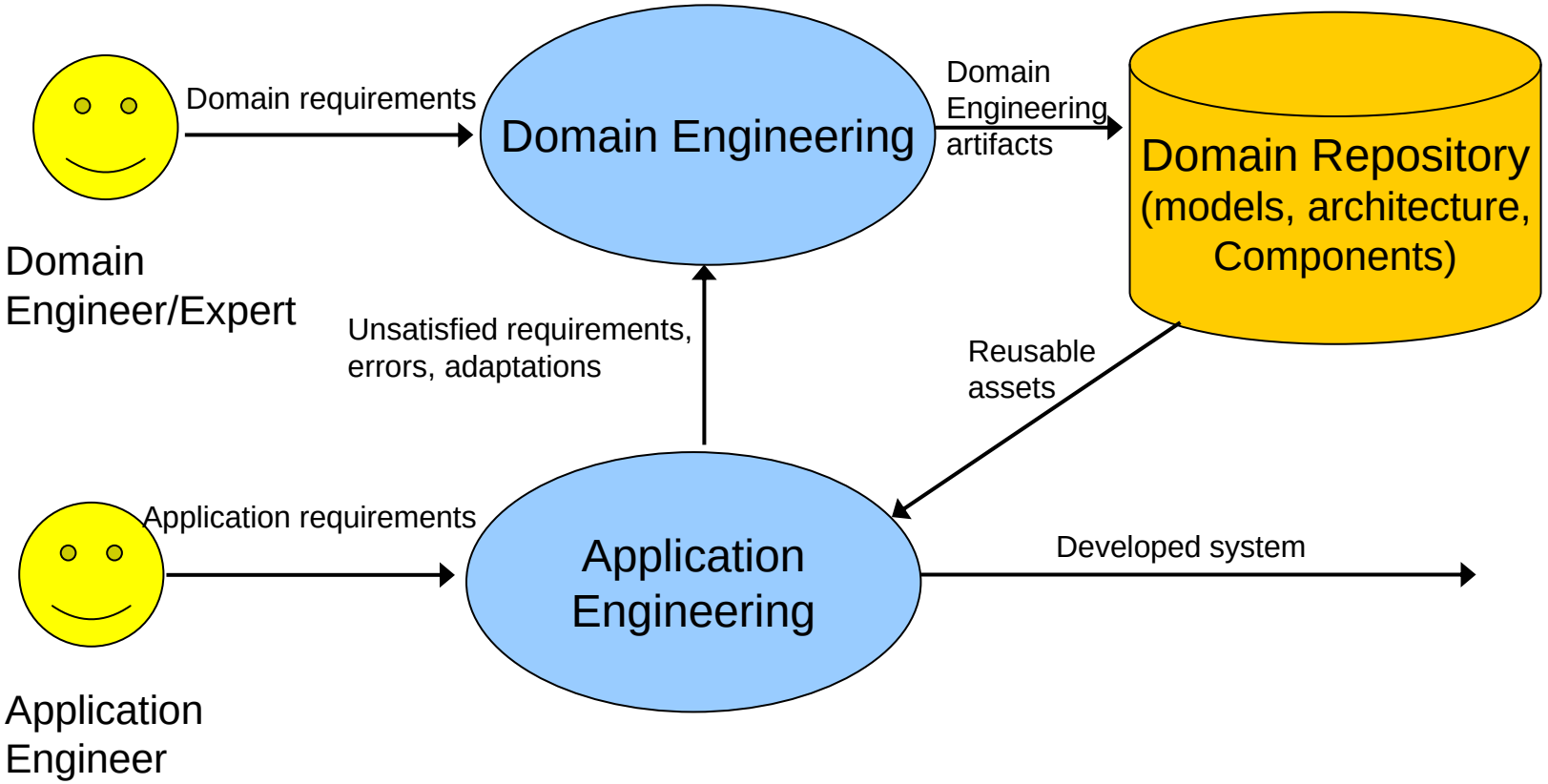
Domain Engineering Activities
Domain Implementation



Domain Engineering & Application Engineering

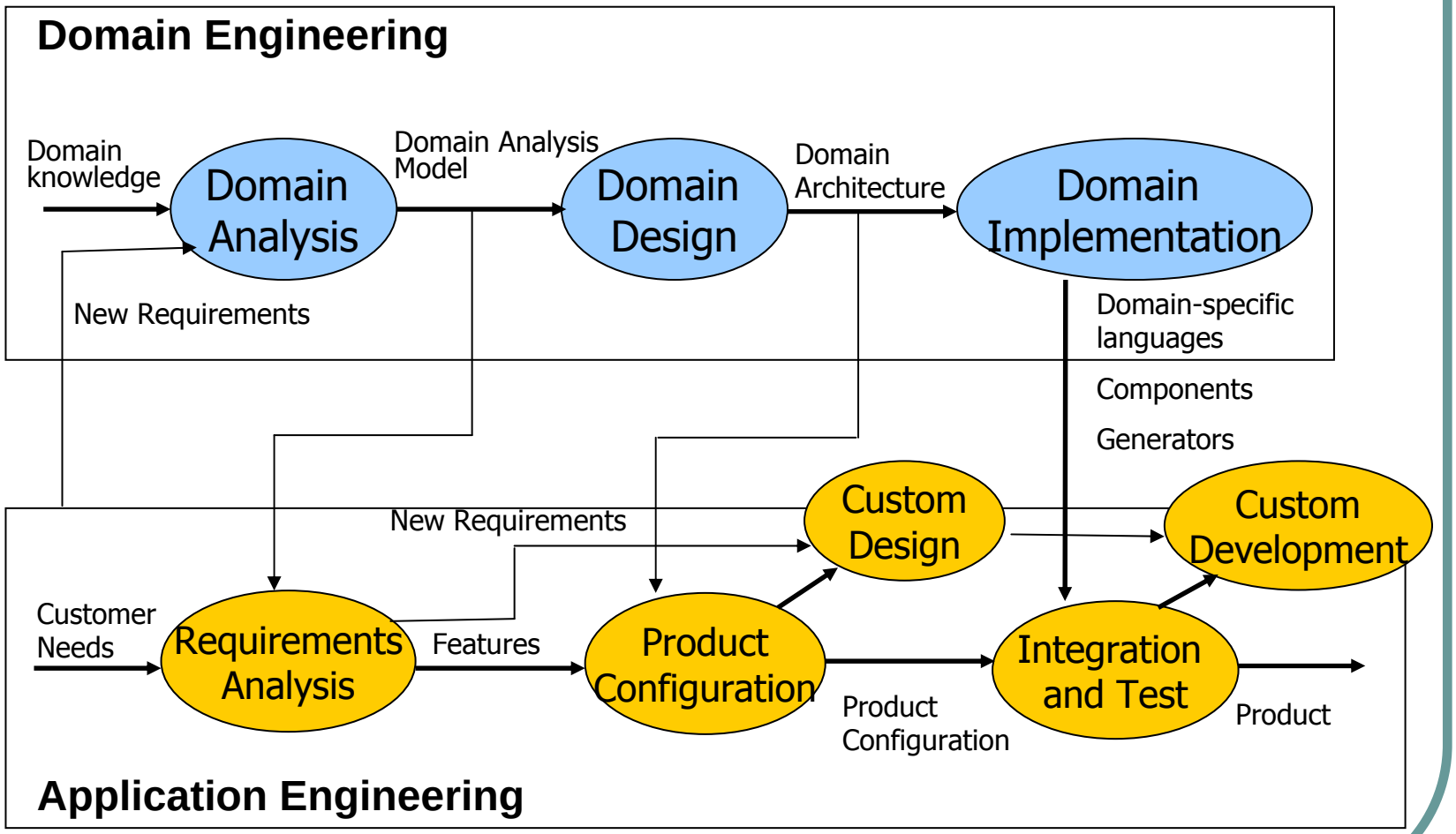
Domain Engineering	Application Engineering
a systematic approach to construct reusable assets in a given problem domain	uses the assets to build specialized software systems in the given domain
Define and scope domain	Do delta analysis and design relative to a domain model and architecture
Analyze examples, needs, trends	
Develop domain model and architecture	Use component systems as starting point
Structure commonality and variability	Find, specialize, and integrate components
Engineer reusable component systems, languages, and tools	Exploit variability mechanisms, languages, generators, ….

Domain Engineering & Application Engineering



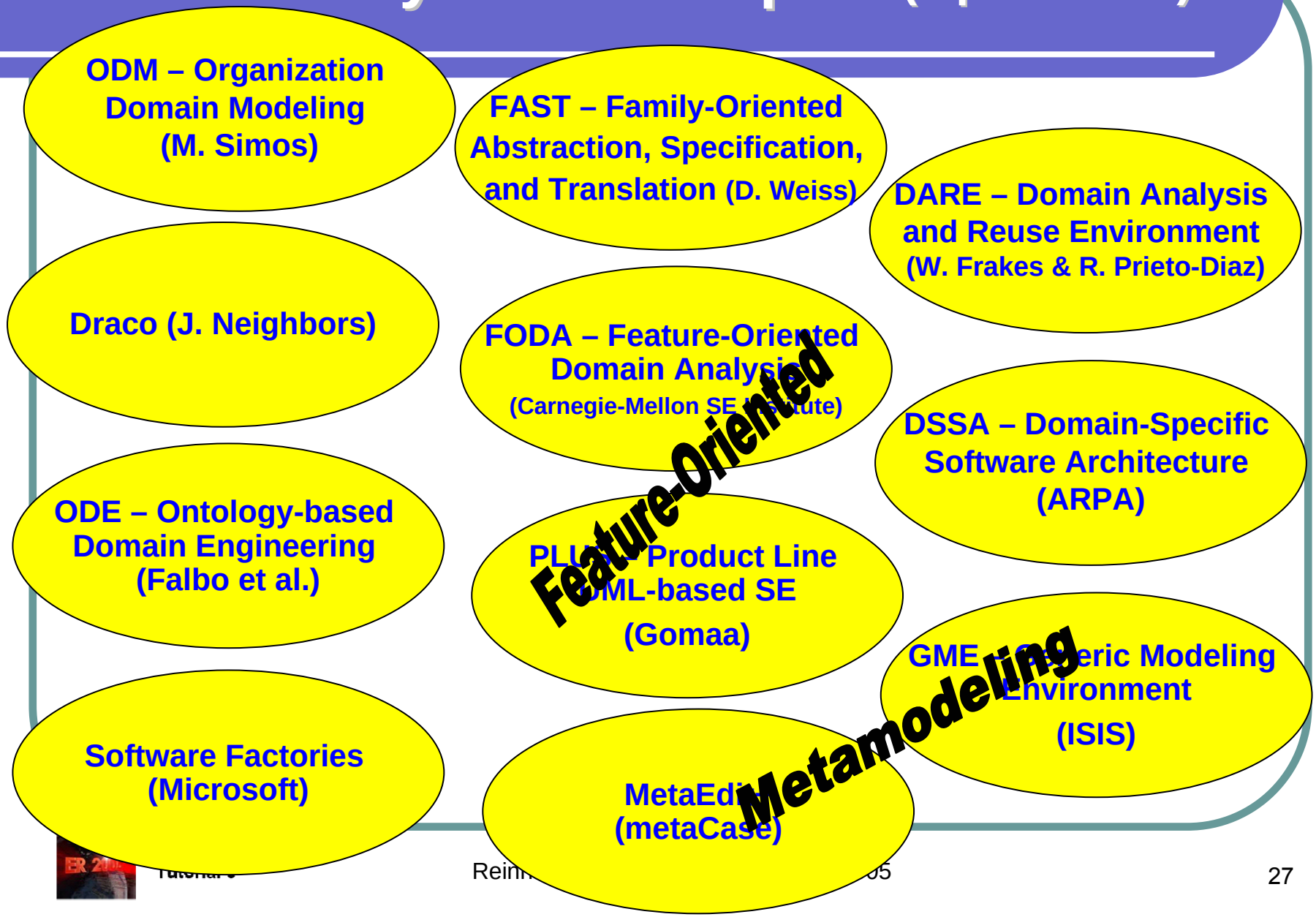
Adapted from: Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004

Domain Engineering & Application Engineering



Adapted from: the Software Engineering Institute at Carnegie Mellon

Domain Analysis Techniques (a partial list)



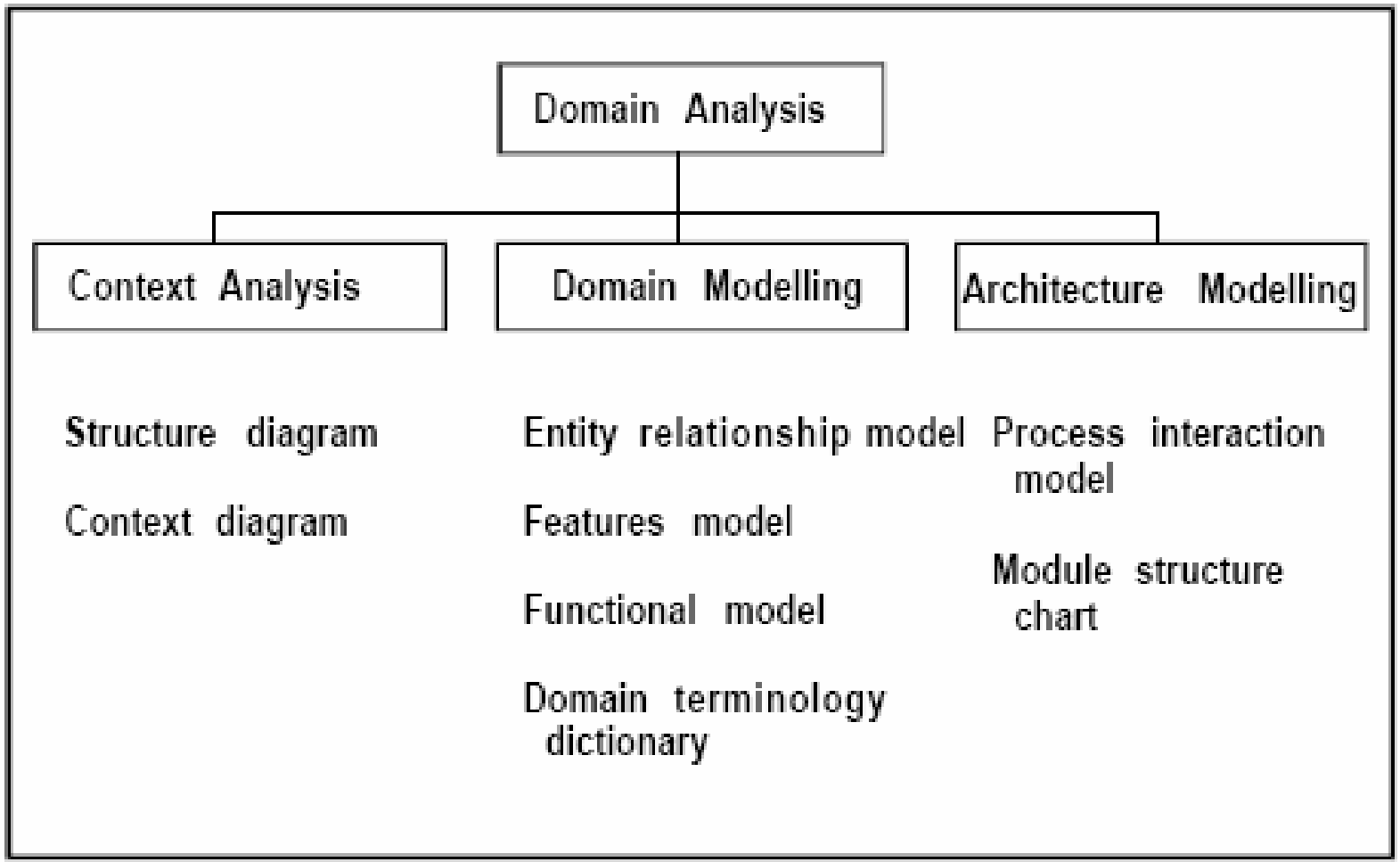
Domain Analysis Techniques
Feature-Oriented Approaches

- Successful software reuse requires the systematic discovery and exploitation of commonality across related software systems.
- ***Feature Oriented Software Engineering*** is an emerging discipline in which feature models are used to model the commonality and variability in a product family.
  - A feature is a requirement or characteristic that is provided by one or more members of the software product line
  - Examples of such approaches:
    - Feature-Oriented Domain Analysis (FODA)
    - Product Line UML-based Software engineering (PLUS)

Domain Analysis Techniques
Feature-Oriented Domain Analysis (FODA)

- The ***Feature-Oriented Domain Analysis (FODA)*** method supports reuse at the functional and architectural levels.
  - Developed by the Software Engineering Institute at Carnegie Mellon
- The basic observations of FODA:
  - The general knowledge of a domain can be achieved by abstracting away “factors” that make one application different from other applications
  - To develop applications from the generic products, those factors that have been abstracted away must be made specific and reintroduced during refinement

Domain Analysis Techniques
Feature-Oriented Domain Analysis (FODA)



Domain Analysis Techniques
Feature-Oriented Domain Analysis (FODA)

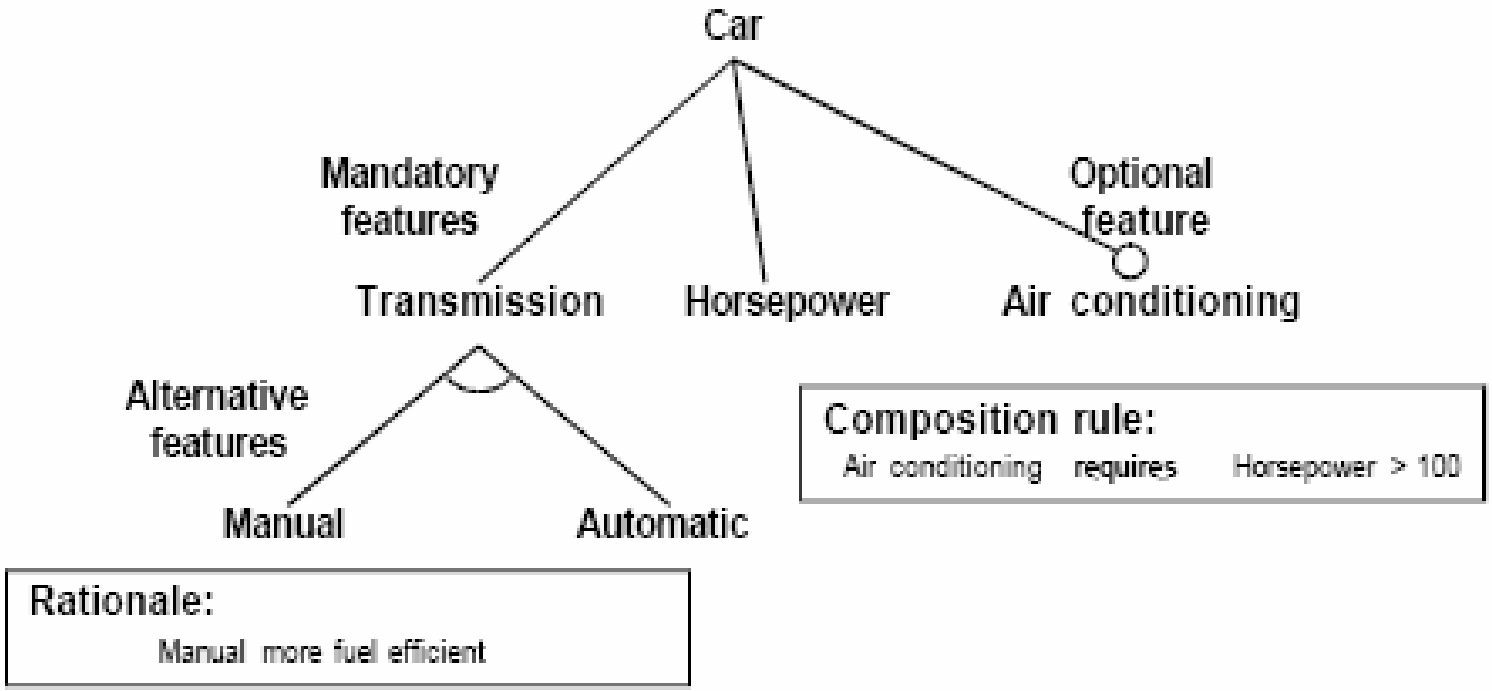
- ***Context Analysis*** aims at defining the scope of a domain that is likely to yield exploitable domain products.
- ***Domain Modeling*** aims at analyzing the commonalities and differences within applications in the domain. It includes:
  - Feature Analysis
  - Entity-Relationship modeling
  - Functional analysis (using state and activity charts)
- ***Architecture Modeling*** aims at providing a software "solution" to the problems defined in the domain modeling phase.

Domain Analysis Techniques
Feature-Oriented Domain Analysis (FODA)

- A basic diagram in FODA is the ***feature diagram***, which represents a hierarchical decomposition of features and their kinds (mandatory, alternative, optional feature).
- Composition rules for features describe which combinations are valid/invalid:
  - Requires rules: capture implications between features
  - Mutually-exclusive rules: Model constraints on feature combinations
- Rationale includes reasons for choosing a feature.
- Feature diagrams and their instantiations:
  - Feature diagrams concisely describe all possible configurations (called instances) of a system, focusing on the features that may differ in each of the configuration
  - An instance of a feature diagram consists of an actual choice of features matching the requirements imposed by the diagram

Domain Analysis Techniques
Feature-Oriented Domain Analysis (FODA)

An example of a Feature Diagram



Domain Analysis Techniques
Product Line UML-based Software engineering (PLUS)

- The ***Product Line UML-based Software engineering (PLUS)*** method extends UML-based modeling methods in order to address software product lines
  - Developed by Hassan Gomaa (“Designing Software Product Lines with UML”)
- PLUS includes:
  - **Requirements modeling:**
    - **Product line scoping:** determining the functionality, the degree of commonality and variability, and the likely number of product line members
    - **Use case modeling:** the functional requirements of the product line are specified in terms of use cases and actors
    - **Feature modeling:** Features, which are the primary vehicle for describing commonality and variability in software product lines, are determined and organized into sets

Domain Analysis Techniques
Product Line UML-based Software engineering (PLUS)

- PLUS includes:
  - **Analysis modeling:**
    - **Static modeling:** A problem-specific static model is defined
    - **Object structuring:** The objects—kernel, optional, and alternative—that participate in each use case are determined
    - **Dynamic modeling:** The use cases from the use case model are realized to show the interaction among the objects participating in each kernel, optional, and alternative use case
    - **Finite state machine modeling:** The state-dependent aspects of the product line are defined by means of hierarchical statecharts
    - **Feature/class dependency analysis:** This step is used to determine what classes from the analysis model are needed to realize the features from the feature model

Domain Analysis Techniques
Product Line UML-based Software engineering (PLUS)

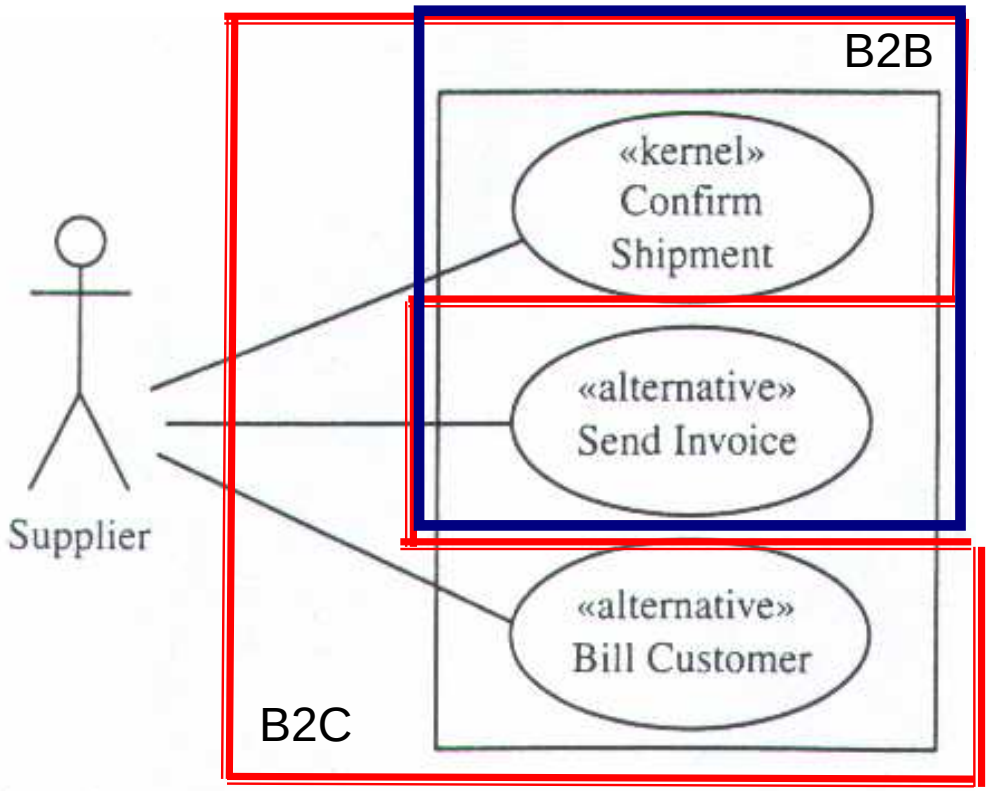
- PLUS includes:
  - **Design modeling:**
    - **Software architectural pattern–based design:** It is necessary to understand both the structural and communication patterns that can be used for designing the product line software architecture
    - **Software product line architectural design:** The design of the product line architecture is approached from the viewpoint of structuring a distributed application into software components and their interconnections

Domain Analysis Techniques
Product Line UML-based Software engineering (PLUS)

- There are two approaches for developing the domain model of the software product line
  - **The Kernel First Approach**
    - The kernel of the application domain is determined first
    - The common aspects of the domain are determined before the variability
  - **The View Integration Approach**
    - It is most applicable when systems that can be analyzed exists
    - Each system is considered a view of the application domain
    - The different views are integrated

Domain Analysis Techniques
Product Line UML-based Software engineering (PLUS)

An Example of Use Case Modeling in PLUS



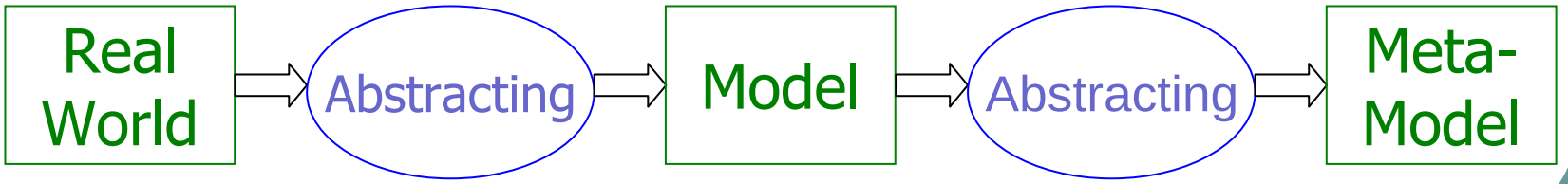
A more detailed example
(slides 103-106)

Figure 4.6 *Example of kernel and alternative use cases*

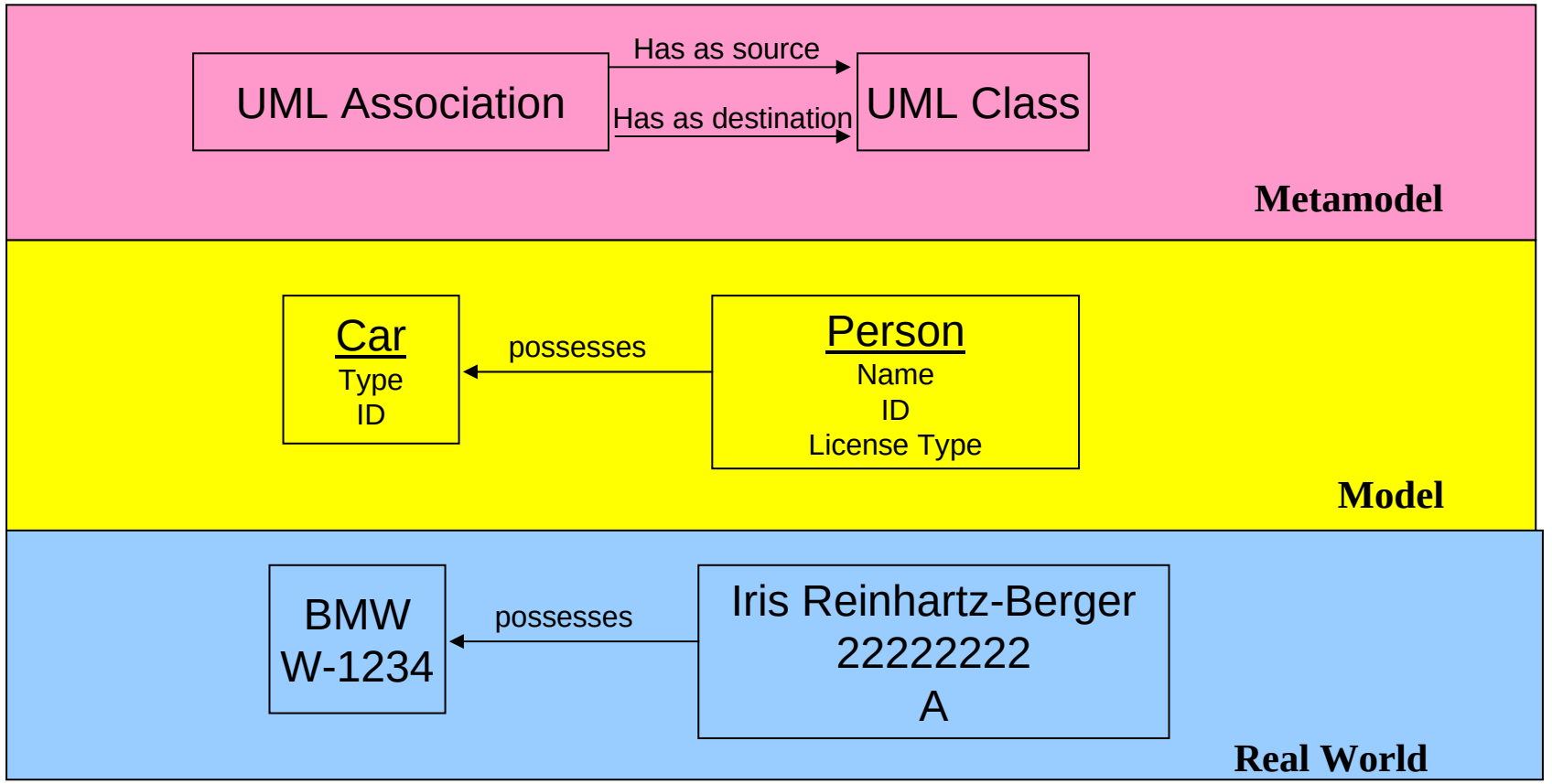
From: Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004

Domain Analysis Techniques
Metamodeling Techniques

- System analysis and design activities can be divided into three types with increasing abstraction levels:
  - The *real world* is what system analysts perceive as reality
  - A *model* is an abstraction of this perceived reality that enables its representation using some approach, language, or methodology
  - A *metamodel* is a model of a model, namely the generic constructs available for modeling a modeling language, technique, or method, and their relationships



Domain Analysis Techniques
Metamodeling Techniques



Domain Analysis Techniques
Metamodeling Techniques

- **Some metamodels usage:**
  - Metamodels are the foundation for integration in software development
  - Metamodels help abstracting low level integration and interoperability details and facilitate partitioning problems into orthogonal sub-problems
  - Metamodels enable checking and verifying the completeness and expressiveness of a model
  - Metamodels enable generating implementation artifacts
- **Examples of metamodels:**
  - UML version 1.3 (slides 107-109)

Domain Analysis Techniques
Generic Modeling Environment (GME)

- The ***Generic Modeling Environment (GME)*** is a domain-specific, model-integrated program synthesis tool for creating and evolving domain-specific, multi-aspect models of large-scale engineering systems.
  - Developed by the institute for software integrated systems at Vanderbilt University
- The modeling paradigm contains all the syntactic, semantic, and presentation information regarding the domain
  - Which concepts will be used to construct models?
  - What relationships may exist among those concepts?
  - How the concepts may be organized and viewed by the modeler?
  - What are the rules governing the construction of models?

Domain Analysis Techniques
Generic Modeling Environment (GME)

- GME modeling concepts include:
  - A ***Project*** contains a set of Folders
  - ***Folders*** are containers that help organize models
  - Models, Atoms, References, Connections and Sets are all **first class objects (FCO)**
  - ***Atoms*** are the elementary modeling objects that do not have internal structure (i.e. they do not contain other objects)
    - Each kind of Atom is associated with an icon
    - Each kind of Atom can have a predefined set of attributes, whose values are user changeable

Domain Analysis Techniques
Generic Modeling Environment (GME)

- GME modeling concepts include:
  - ***Models*** are compound objects that can have parts and inner structure
    - A part in a container Model always has a ***Role***
    - The modeling paradigm determines what *kind* of parts are allowed in Models acting in which Roles
    - E.g., if we want to model digital circuits below the gate level, then we would have to use Models for gates (instead of Atoms) that would contain transistor Atoms
  - The containment relationship creates the hierarchical decomposition of Models
    - Any object must have at most one parent, which is a Model
    - At least one Model does not have a parent; it is called a *root Model*

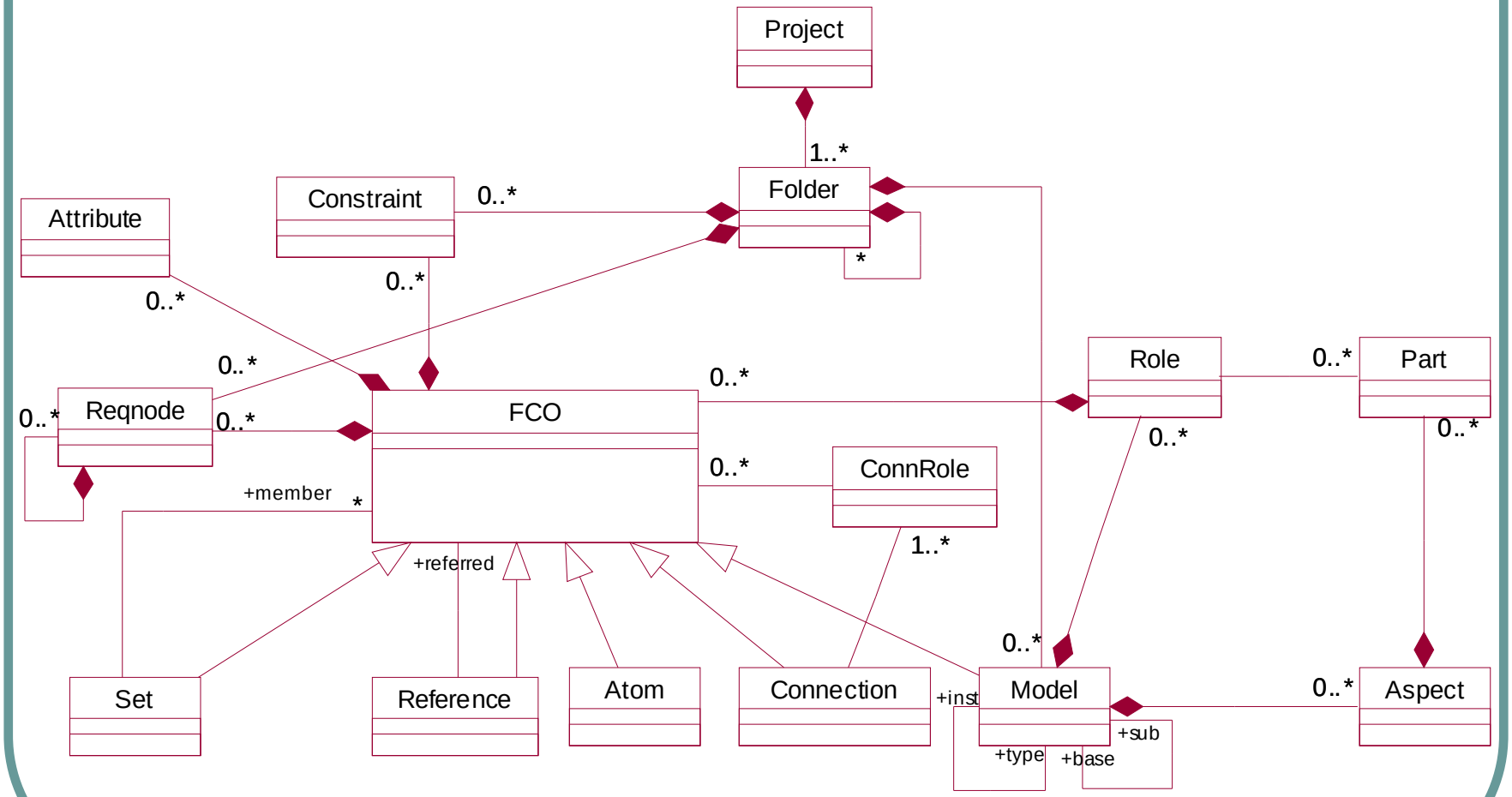
Domain Analysis Techniques
Generic Modeling Environment (GME)

- GME modeling concepts include:
  - ***Aspects*** provide primarily visibility control
    - Every Model has a predefined set of Aspects
    - Each part can be visible or hidden in an Aspect
    - Every part has a set of *primary aspects* where it can be created or deleted
  - A ***connection*** is a line that connects two parts of a model
    - Connections have at least two attributes: appearance and directionality
    - In order to make a Connection between two objects they must have the same parent in the containment hierarchy (and they also must be visible in the same Aspect, i.e. one of the primary Aspects of the Connection)
    - Connections can further be restricted by explicit Constraints specifying their multiplicity, for instance

Domain Analysis Techniques
Generic Modeling Environment (GME)

- GME modeling concepts include:
  - ***References*** are parts that are similar in concept to pointers found in various programming languages
    - Any first class object (FCO), except for a Connection, can be referred to (even references themselves)
    - References can be connected just like regular model objects
    - A reference always refers to exactly one object, while a single object can be referred to by multiple References
  - ***Sets*** can be used to specify a relationship among a group of objects
    - Connections and References are binary relationships
    - The only restriction is that all the members of a Set must have the same container (parent) and be visible in the same Aspect

Domain Analysis Techniques
Generic Modeling Environment (GME)



To a GME example (slides 110-113)

Domain Analysis Techniques
metaEdit+

- ***MetaEdit+*** (by metaCase) provides:
  - A domain specific language comprising six building blocks (meta-meta model elements)
  - A scripting language for defining translation rules
  - An application specification environment that is based on the domain specific language and enforces its constraints
  - A code generator that is based on the application model and the translation rules
- Using MetaEdit+ includes two main steps:
  - An experienced developer first defines the metamodel concepts, rules, and their mapping to code
  - Other developers then make models with the concepts guided by the rules, and code is automatically generated

Domain Analysis Techniques
metaEdit+

- Form-based metamodeling
  - Fast to enter concepts
  - Scales better than graphical metamodels
- Metamodel and model in same tool
  - Can test metamodel as it is being built or changed
- Reuse of metamodel concepts
  - Within one metamodel, across different metamodels

Domain Analysis Techniques
metaEdit+

- MetaEdit+ modeling concepts include:
  - A ***graph*** is a collection of objects, relationships, roles, and bindings of these to show which objects a relationship connects via which roles
  - An ***object*** is a thing that exists on its own
  - A ***property*** is a describing or qualifying characteristic associated with the other types
  - A ***port*** is a part of an object to which a role can connect
  - A ***relationship*** is an explicit connection between a group of objects. Relationships attach to objects via roles
  - A ***role*** specifies how an object participates in a relationship

To a metaEdit+ example (slides 114-118)

Domain Analysis Techniques
Advantages

- Provide means for gathering and organizing domain related information and **knowledge**.
- Provide libraries of assets to be **reused** in particular applications.
- Provide **validation and verification** templates for particular applications in order to avoid semantic errors in early development stages.

Domain Analysis Techniques
Limitations

- Deal with too broad areas (domains) which are usually understood only during the development process.
- Deal mainly with static and structural constraints.
- Deal with two different abstraction levels: the domain and the application.
  - Each level is accompanied by different notions and notations
  - The transition between the levels remains sometimes vague

Part 2:
The ADOM Approach

Tutorial 3

The ADOM Approach
Motivation

- Application and domain models are **similar**:
  - They both define structure
  - They both exhibit functionality (behavior)
  - They both introduce structural and behavioral constraints
- However they also **differ** in their:
  - Abstraction level
  - Flexibility level
- ADOM treats domains with regular software engineering tools, methods, and techniques.

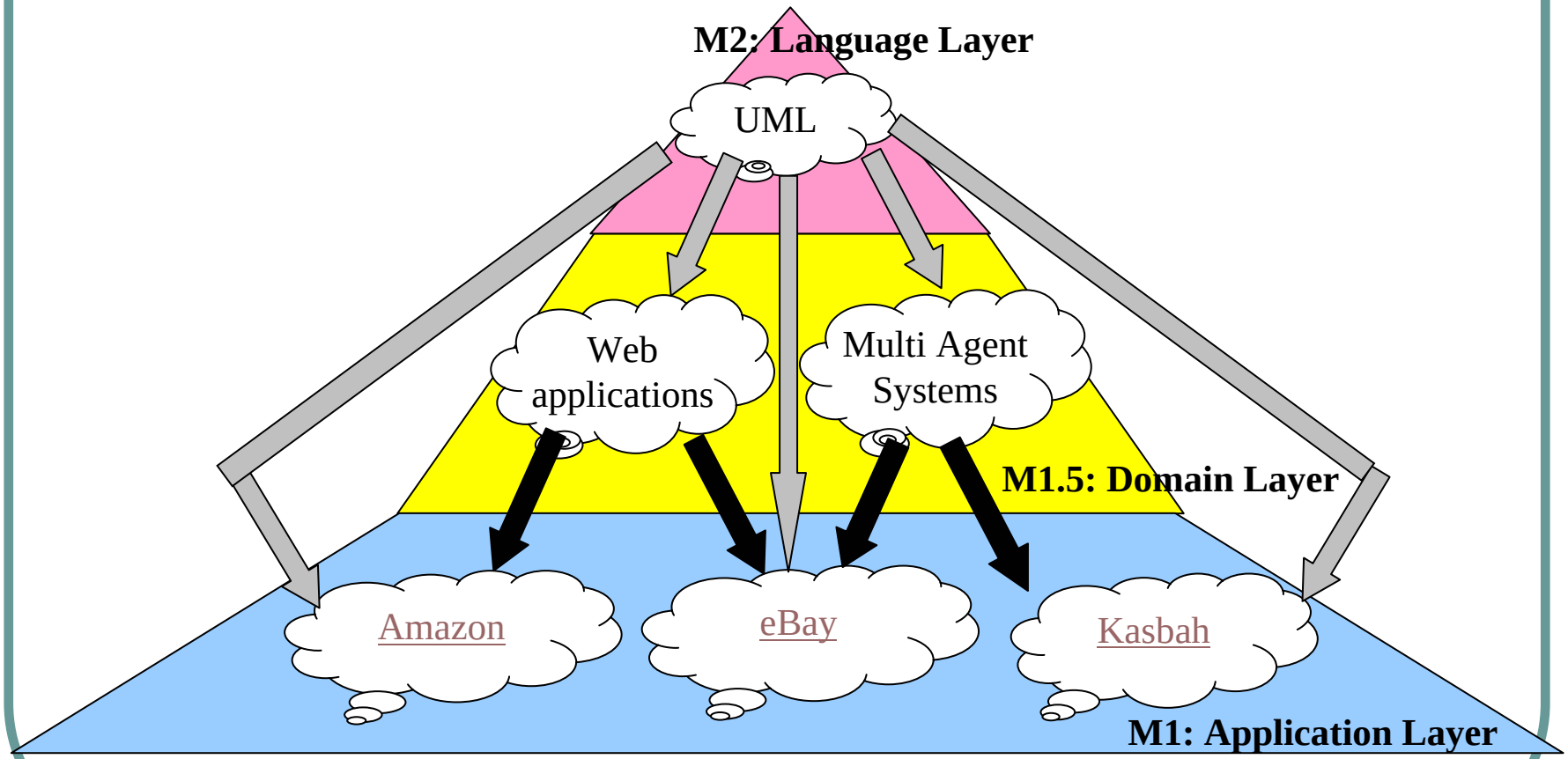
The ADOM Approach
Motivation

- When modeling a domain, there are two policies:
  - **Modeling whatever is allowed**
    - Everything that is allowed in an application model should be presented in the domain model.
    - This policy emphasizes the variability of domain applications.
  - **Constraining whatever is needed**
    - The domain model should include constraints that should be satisfied by all the application models in that domain.
    - The application models can include additional information, as long as it does not violate one of the domain constraints.
    - This policy emphasizes the commonalities of the applications in a particular domain.
- **The ADOM approach adapts the "constraining whatever is needed" policy.**

The ADOM Approach
General Architecture

- ADOM defines three layers of abstraction:
  - The **application layer**, which consists of domain-specific systems
  - The **domain layer**, which captures sets of domain concepts and relations
  - The **modeling language layer**, which consists of modeling language metamodels
- In all layers the same languages (notions and notations) are applied.
- Each layer defines constraints on the more concrete layers.

The ADOM Approach
General Architecture



The ADOM Approach
General Architecture

- In the **domain layer** the structural and behavioral elements of a specific domain are specified.
  - This can be done, for example, by using UML structural and behavioral views
- An **application model** uses a domain model as a validation template and a reusable knowledge source:
  - Any element in the application model is classified according to the elements declared in the domain model
  - All the constraints enforced by the domain model should be applied in any application model of that domain
  - This can be done, for example, by using UML built-in stereotype mechanism

ADOM & UML through Examples
The Domain of Process Control Systems

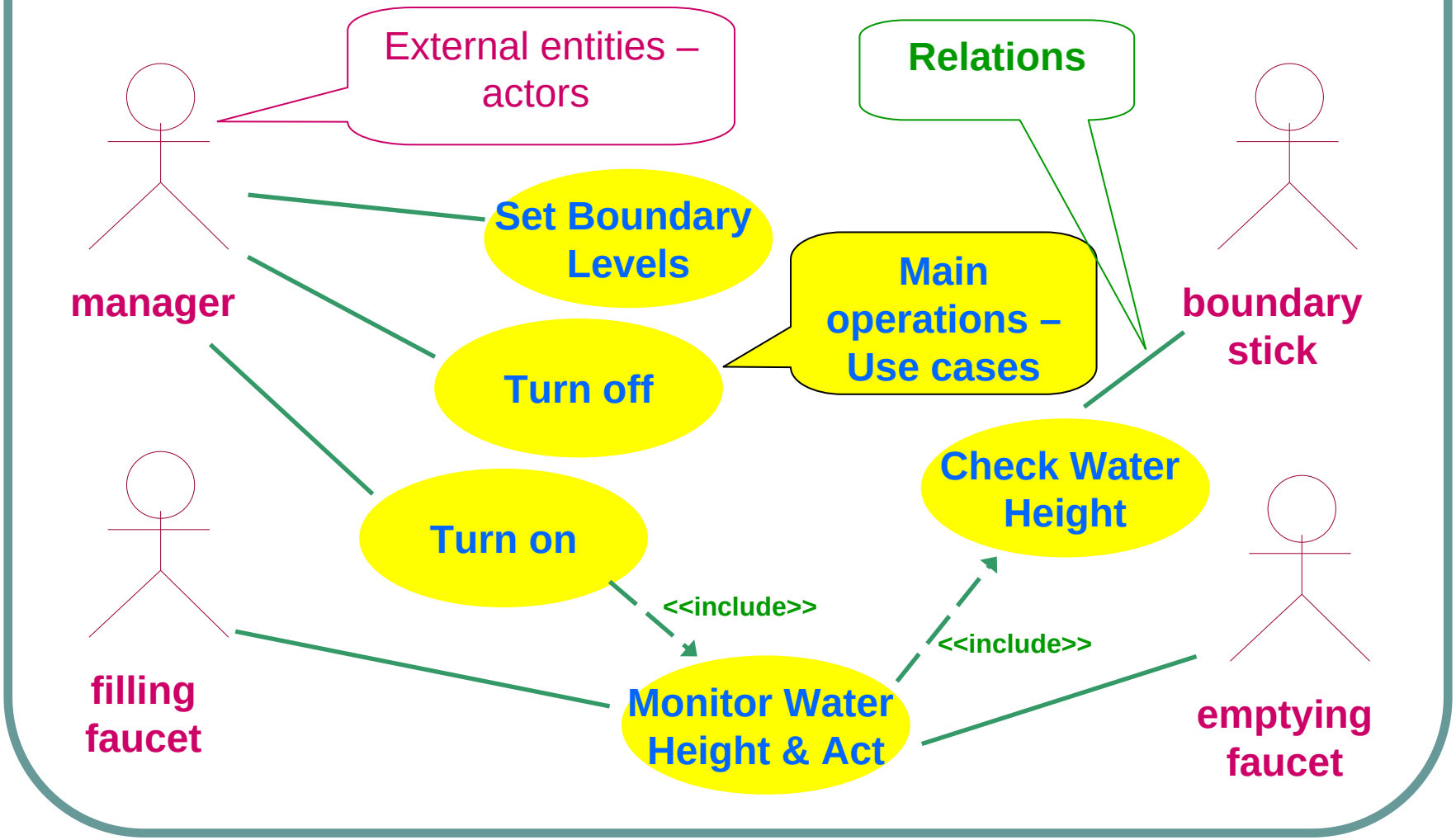
- Applications in the domain monitor and control the values of certain variables through a set of components that work together to achieve a common objective or purpose.
- Examples for application areas within this domain include engineering and industrial control systems, control systems in the human body, and financial derivation-tracking products.
- Two specific examples in the domain:
  - A **water level control system (WLC)** for monitoring and controlling the water level in tanks
  - A **home climate control system (HCC)** for monitoring and controlling room temperature and humidity

ADOM & UML through Examples
The Water Level Control System

- The Water Level Control (WLC) system monitors and controls the water levels in tanks
- Each tank has several boundary limits according to different manger preferences
- The system is expected to satisfy all these requests, ensuring that the actual water level is always in the closed range [Low, High]
- The actual level of the water in the tank is measured by a boundary stick
- Emptying and filling faucets are installed in each tank, enabling changing the water level at will

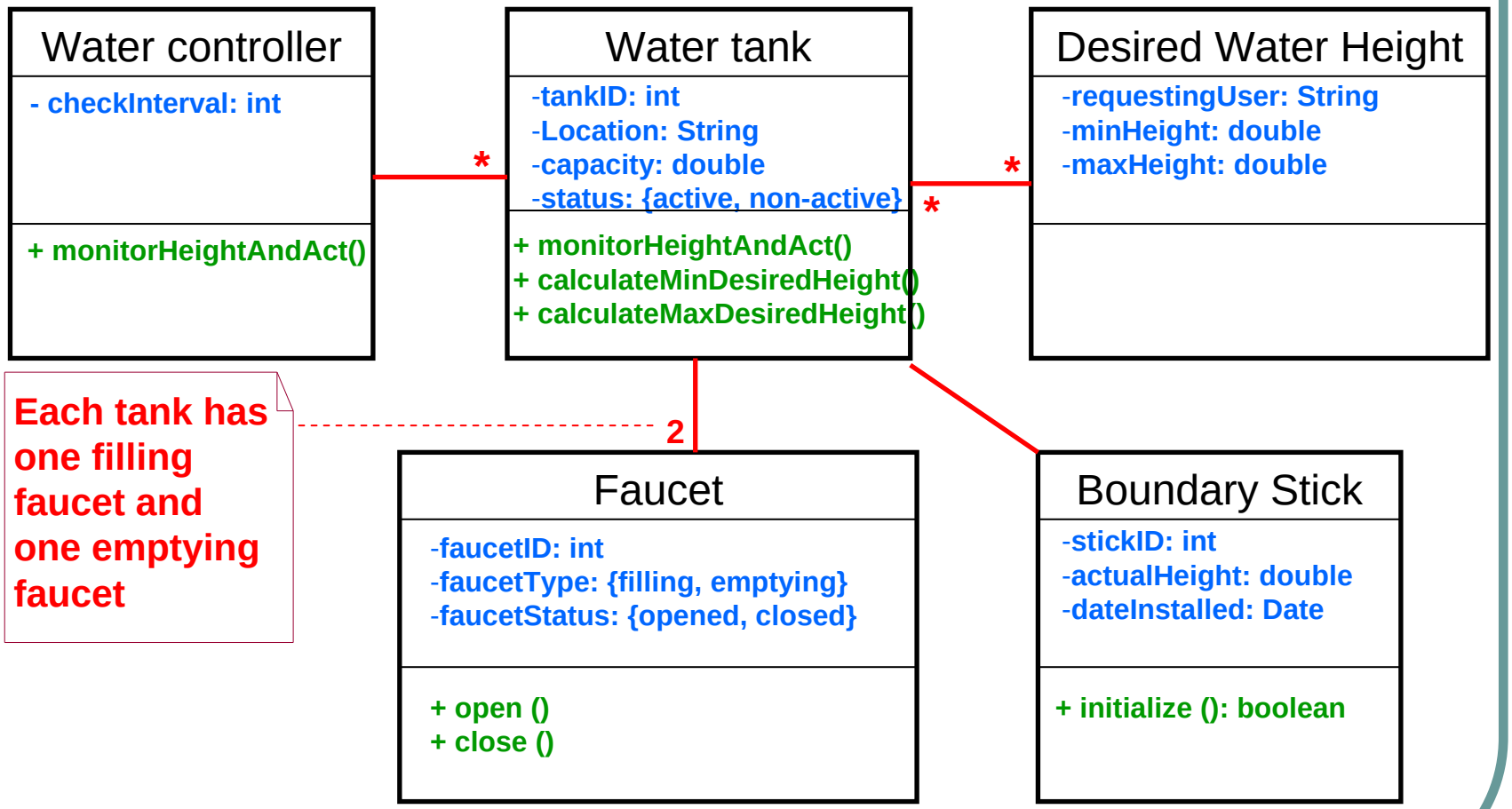
ADOM & UML through Examples

The Water Level Control System – A UC Diagram



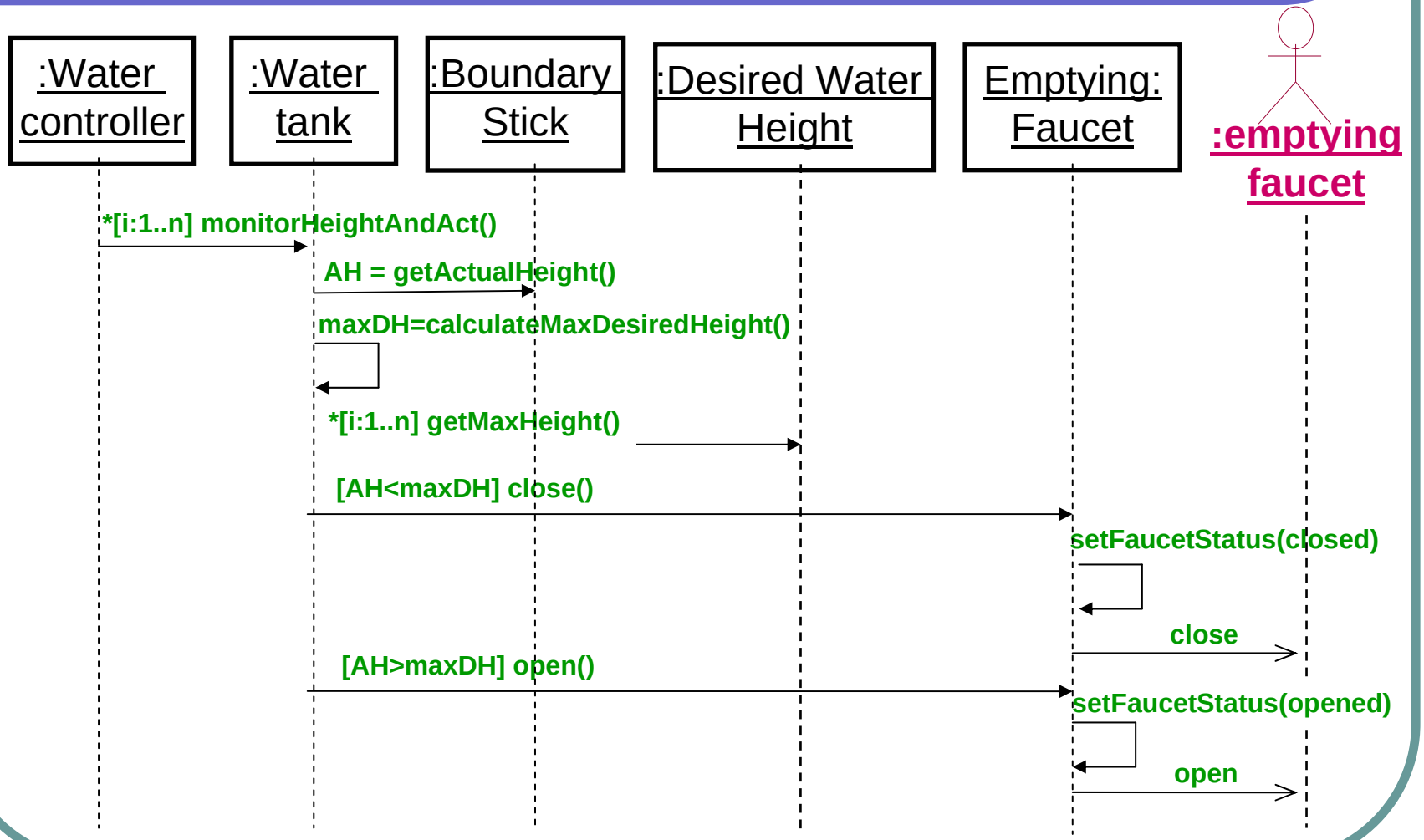
ADOM & UML through Examples

The Water Level Control System – A Class Diagram



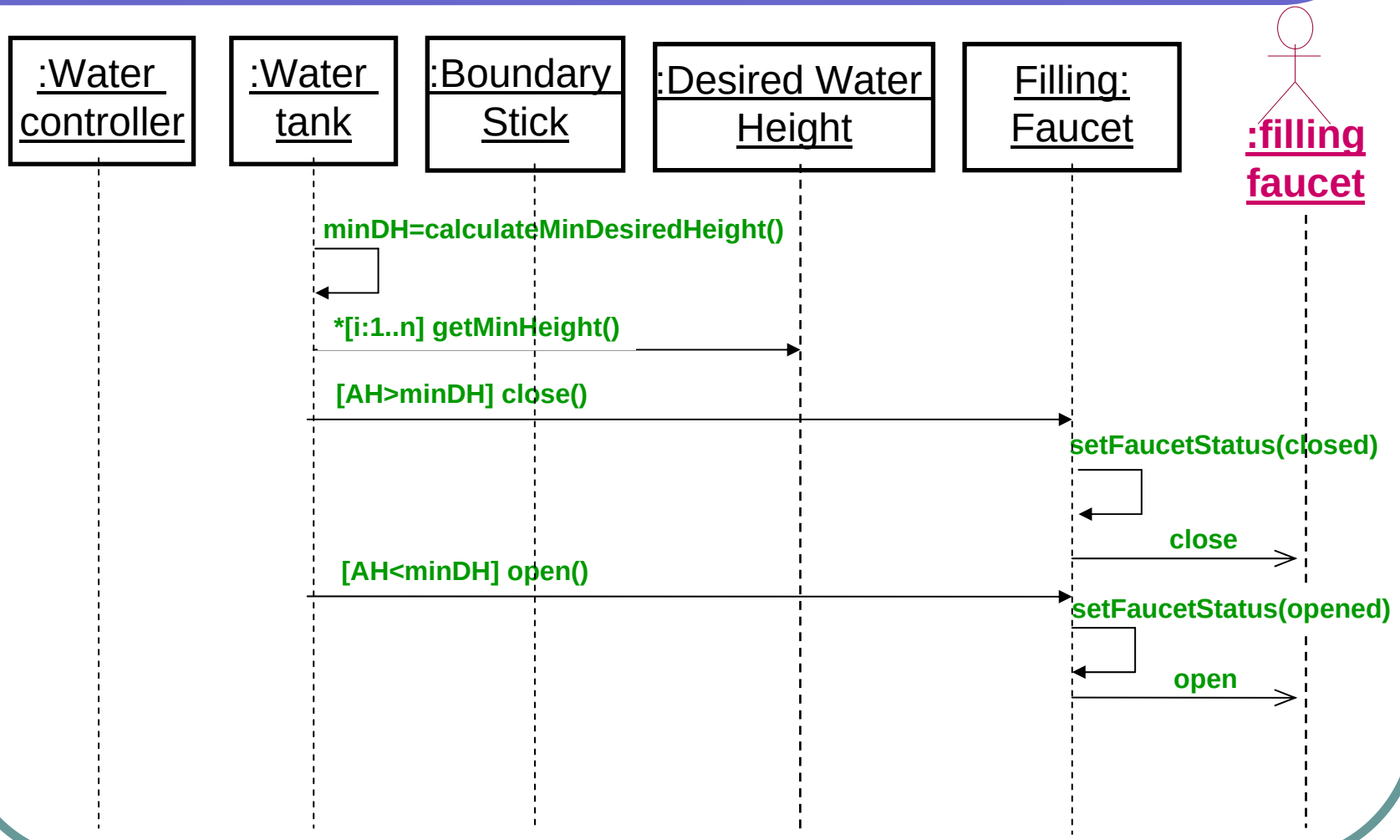
ADOM & UML through Examples

The Water Level Control System – A Sequence Diagram



ADOM & UML through Examples

The Water Level Control System – A Sequence Diagram

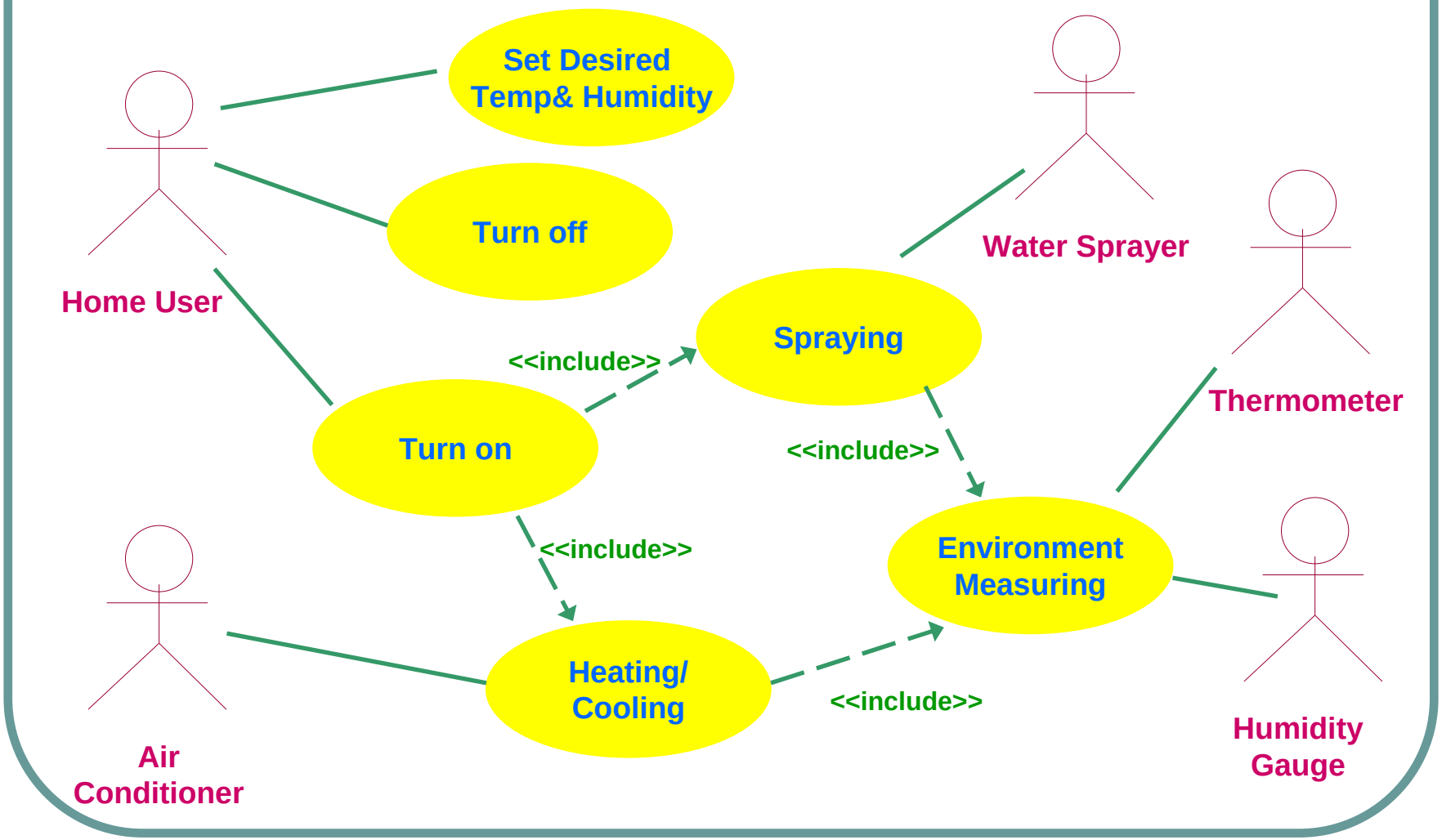


ADOM & UML through Examples
The Home Climate Control System

- The Home Climate Control (HCC) application ensures that the temperature in a room of a house remains in the closed range [LT, HT], while the humidity in this room remains lower than the maximum defined humidity (<MH)
- The system controls several rooms, while each room has its own limit values (LT, HT, MH) which can be configured by the user
- The actual levels of temperature and humidity are measured by thermometers and humidity gauges, respectively
- An air conditioner and a water sprayer are installed in each room, enabling changing the temperature and humidity at will

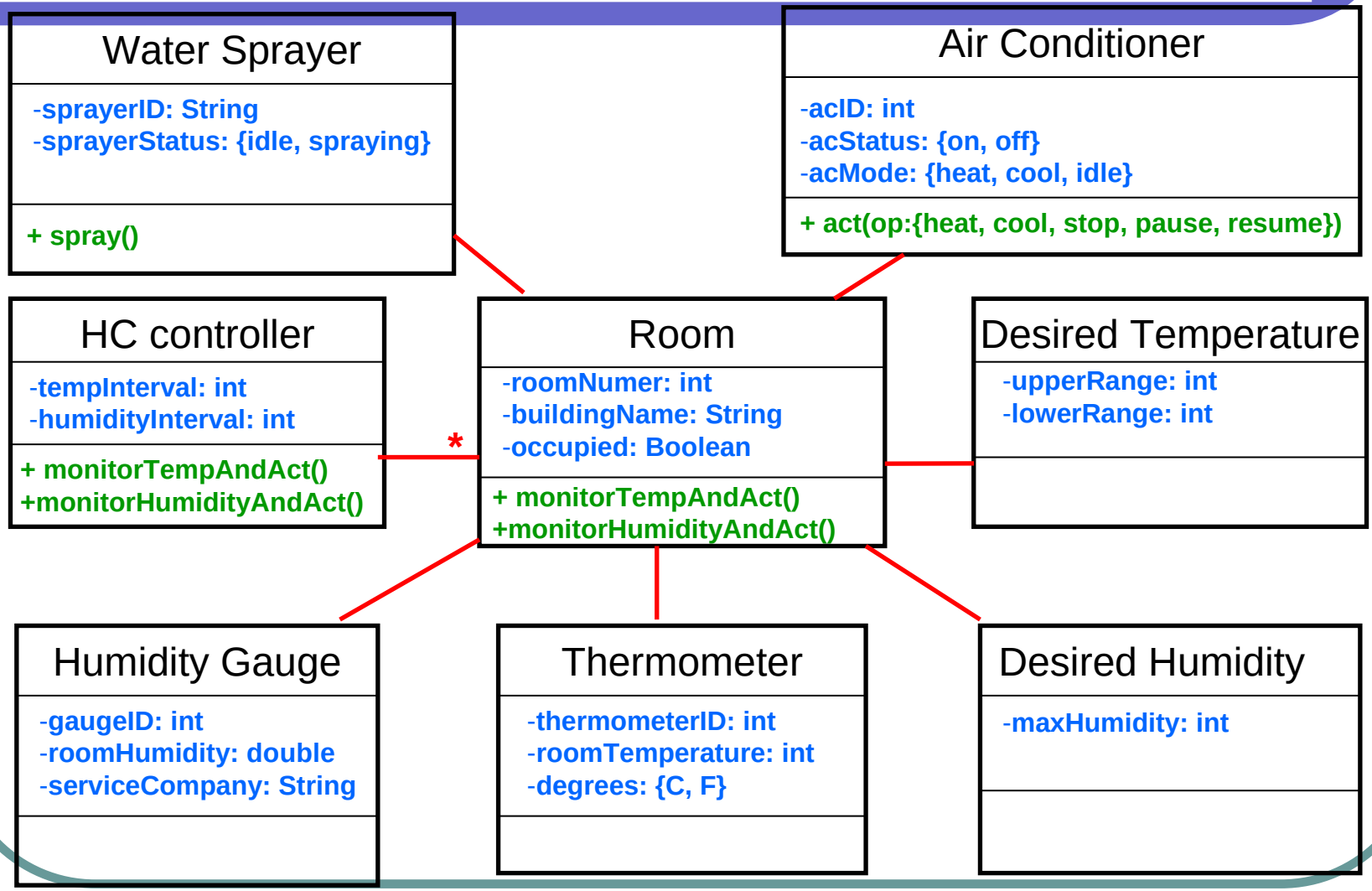
ADOM & UML through Examples

The Home Climate Control System – A UC Diagram



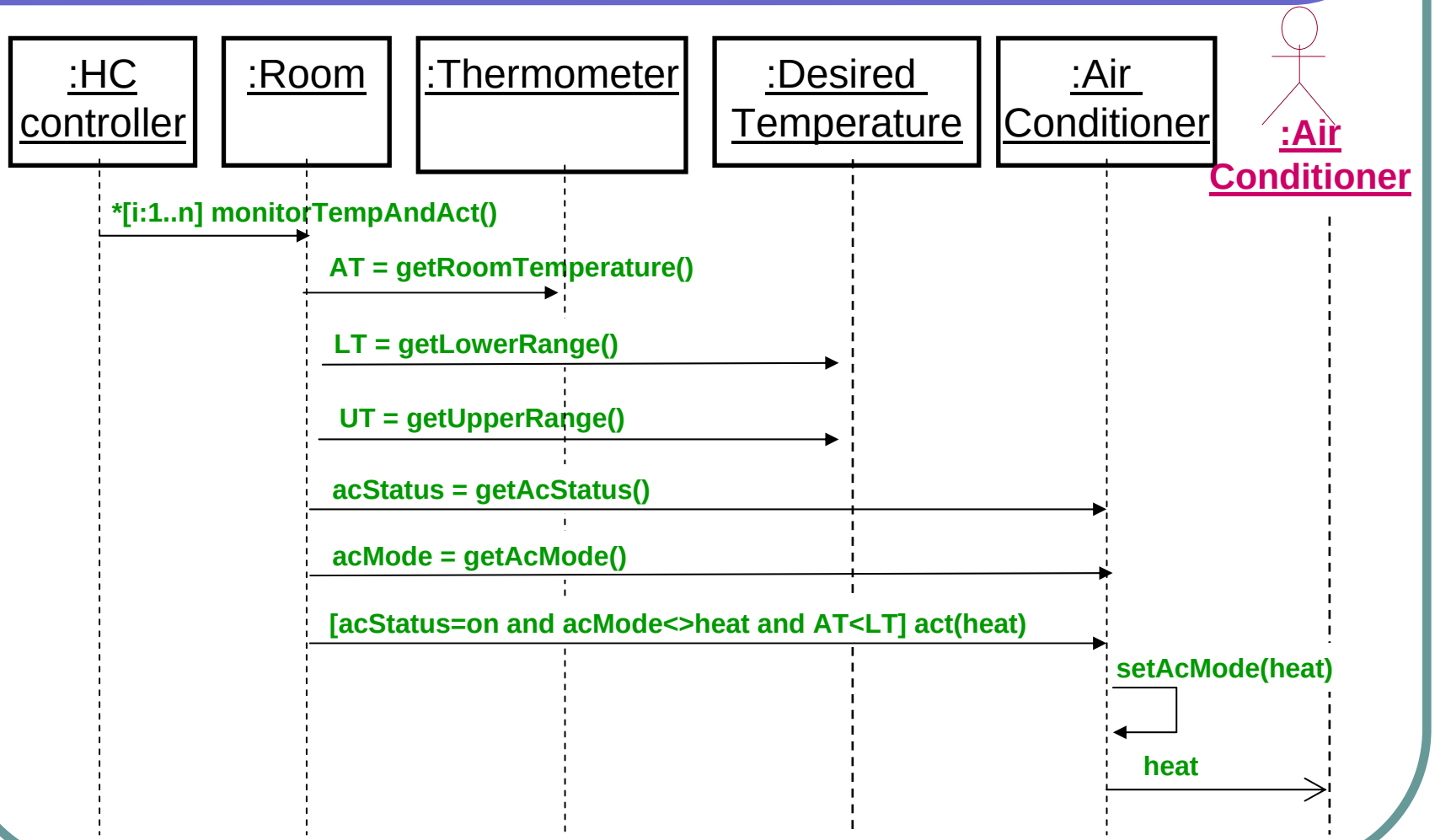
ADOM & UML through Examples

The Home Climate Control System – A Class Diagram



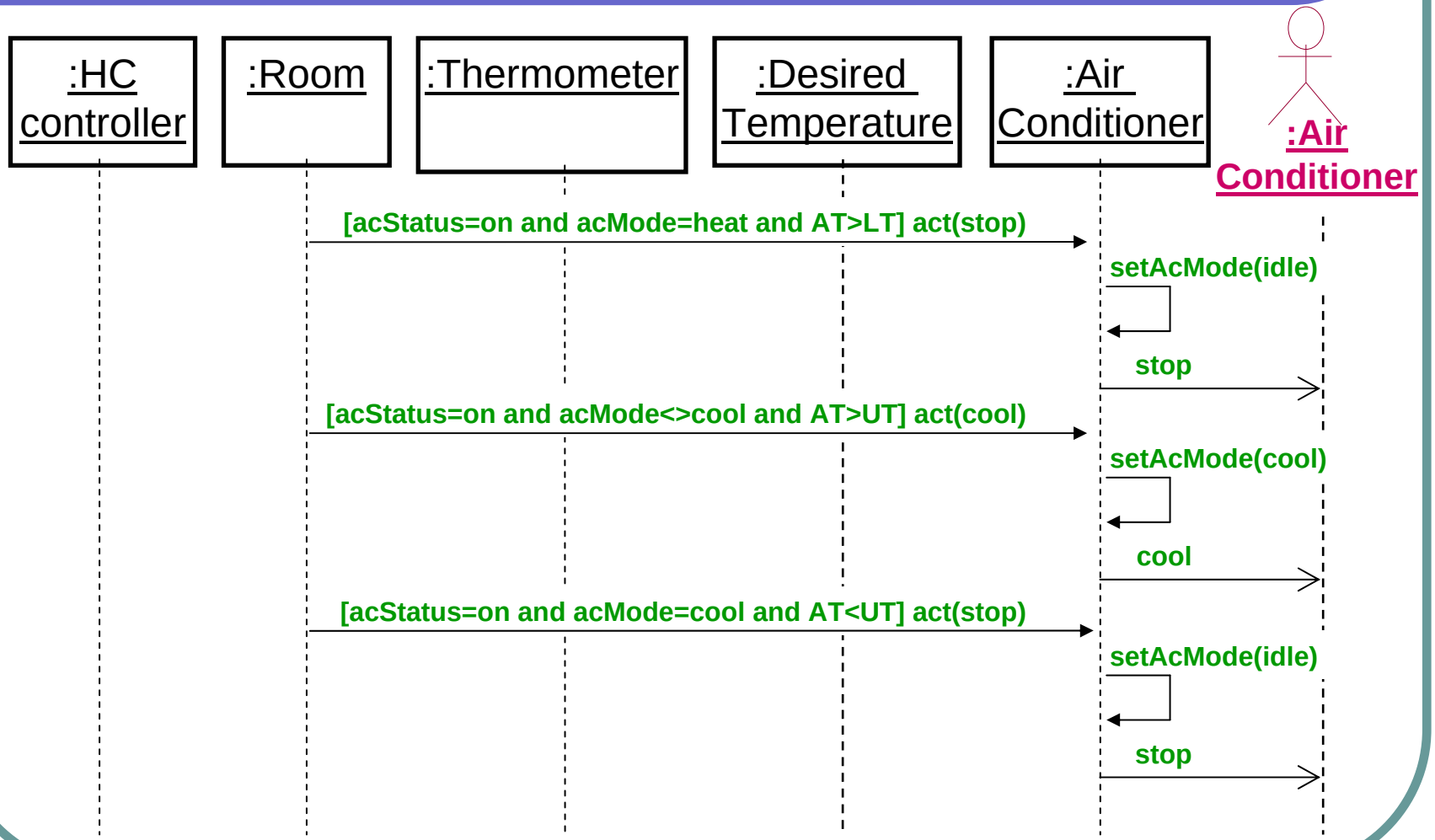
ADOM & UML through Examples

The Home Climate Control System – A Sequence Diagram



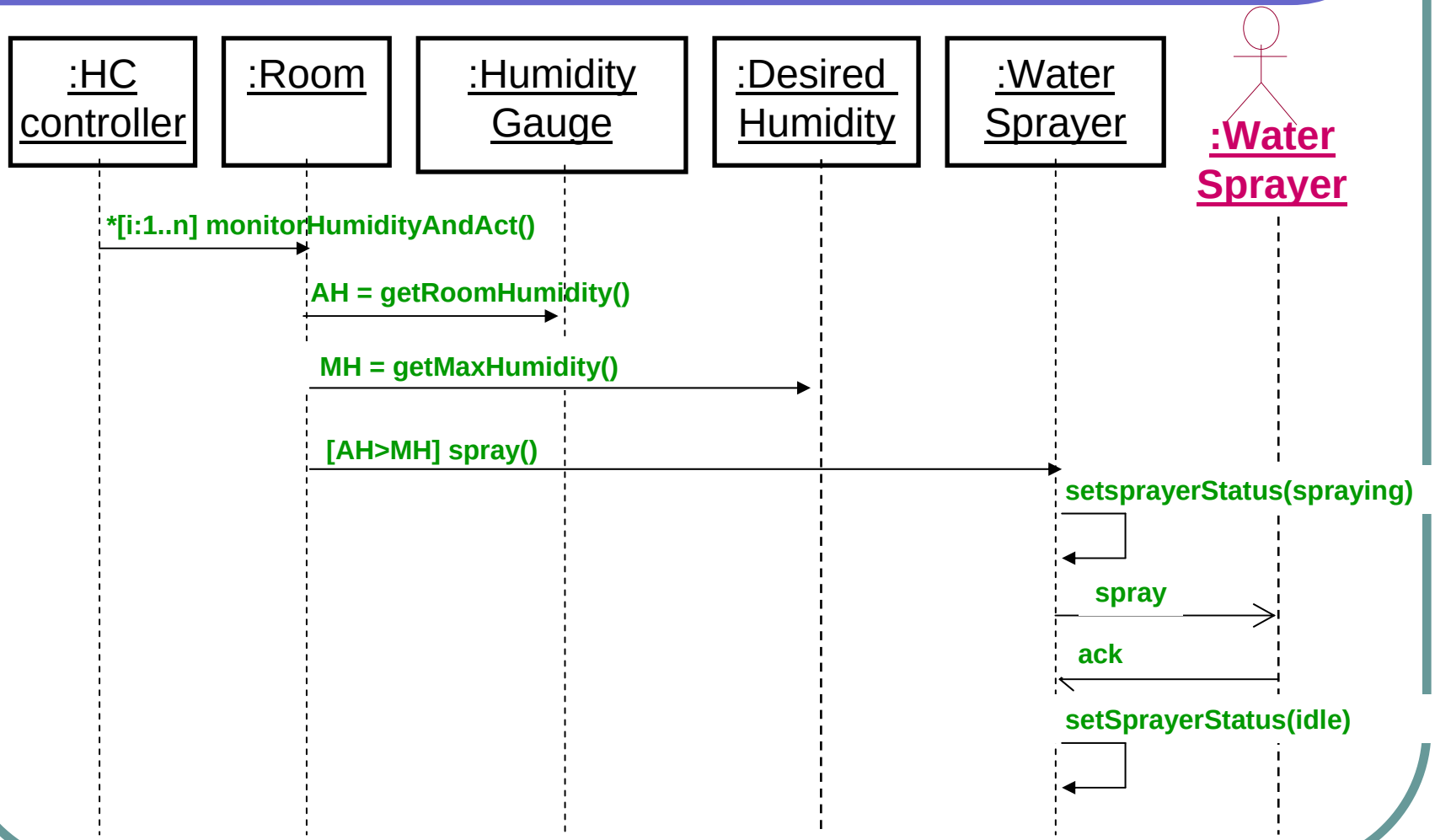
ADOM & UML through Examples

The Home Climate Control System – A Sequence Diagram



ADOM & UML through Examples

The Home Climate Control System – A Sequence Diagram



ADOM & UML through Examples
The Domain of Process Control Systems

- Basic concepts in the PCS domain:
  - **Controller**
  - **Controlled Device** – e.g., emptying/filling faucet, air-conditioner, water sprayer, etc.
  - **Controlled Element** – e.g., tank, room, etc.
  - **Controlled Value** – e.g., water height, temperature, humidity, etc.
  - **Sensor** – e.g., boundary stick, thermometer, humidity gauge, etc.
  - **Operator** – e.g., user, maintainer, etc.

ADOM & UML through Examples
The Domain of Process Control Systems

- Relations between PCS concepts:
  - A **Controller** controls **Controlled Elements**
    - A water controller controls water tanks
    - A climate controller controls rooms
  - The **Controlled Values** in a **Controlled Element** are constrained
    - The water height in a water tank is constrained
    - The temperature and humidity in a room are constrained
  - A **Sensor** measures **Controlled Values**
    - A boundary stick measures water height
    - A thermometer measures temperature
    - A humidity gauge measures humidity

ADOM & UML through Examples
The Domain of Process Control Systems

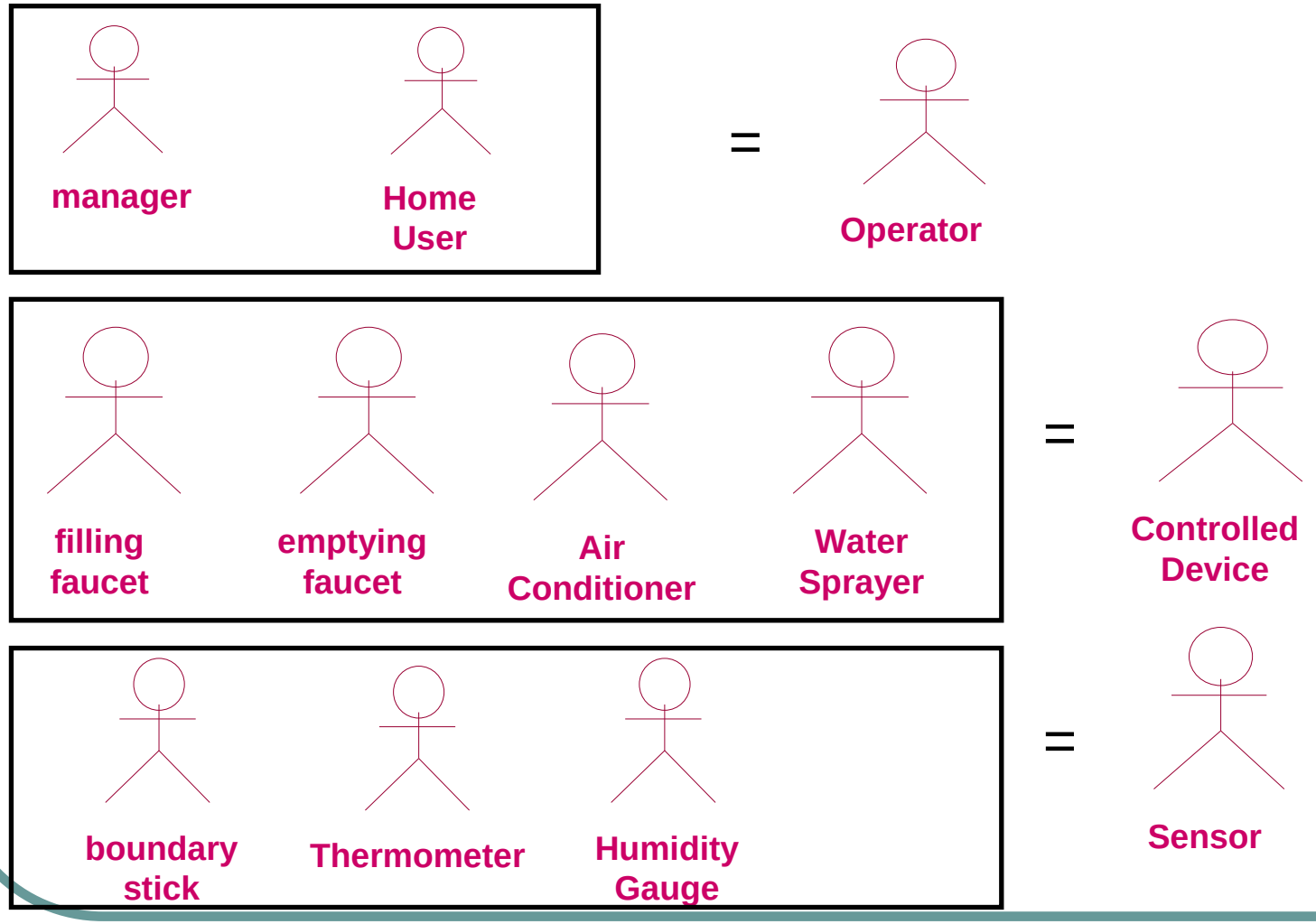
- Relations between PCS concepts:
  - A **Controlled Device** is installed in a **Controlled Element**
    - An emptying/filling faucet is installed in a tank
    - An air-conditioner is installed in a room
    - A water sprayer is installed in a room
  - An **Operator** activates the system
    - A manager activates the WLC system
    - A home user activates the HCC system
  - An **Operator** might configure the system
    - A manager configures the water height range in a tank
    - A home user configures the temperature and humidity levels in a room

ADOM & UML through Examples
The Domain of Process Control Systems

- The behavior in a PCS system:
  - **Configuring**
    - The water height range
    - The temperature and humidity levels
  - **Monitoring & Checking the actual values**
    - The water height
    - The temperature
    - The humidity
  - **Acting**
    - Filling/emptying
    - Heating/cooling
    - Spraying water

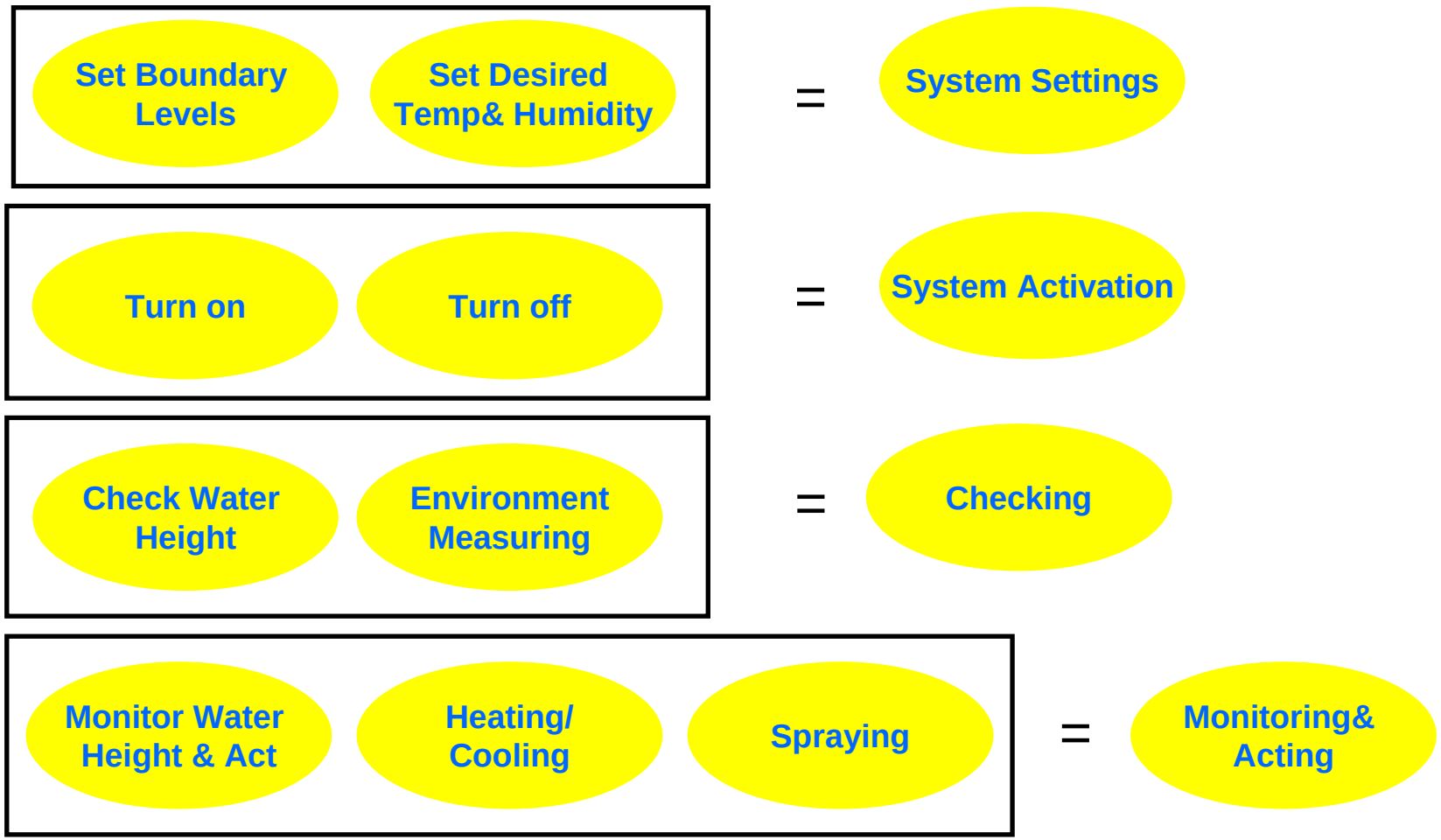
ADOM & UML through Examples

The Process Control Systems – Similarities



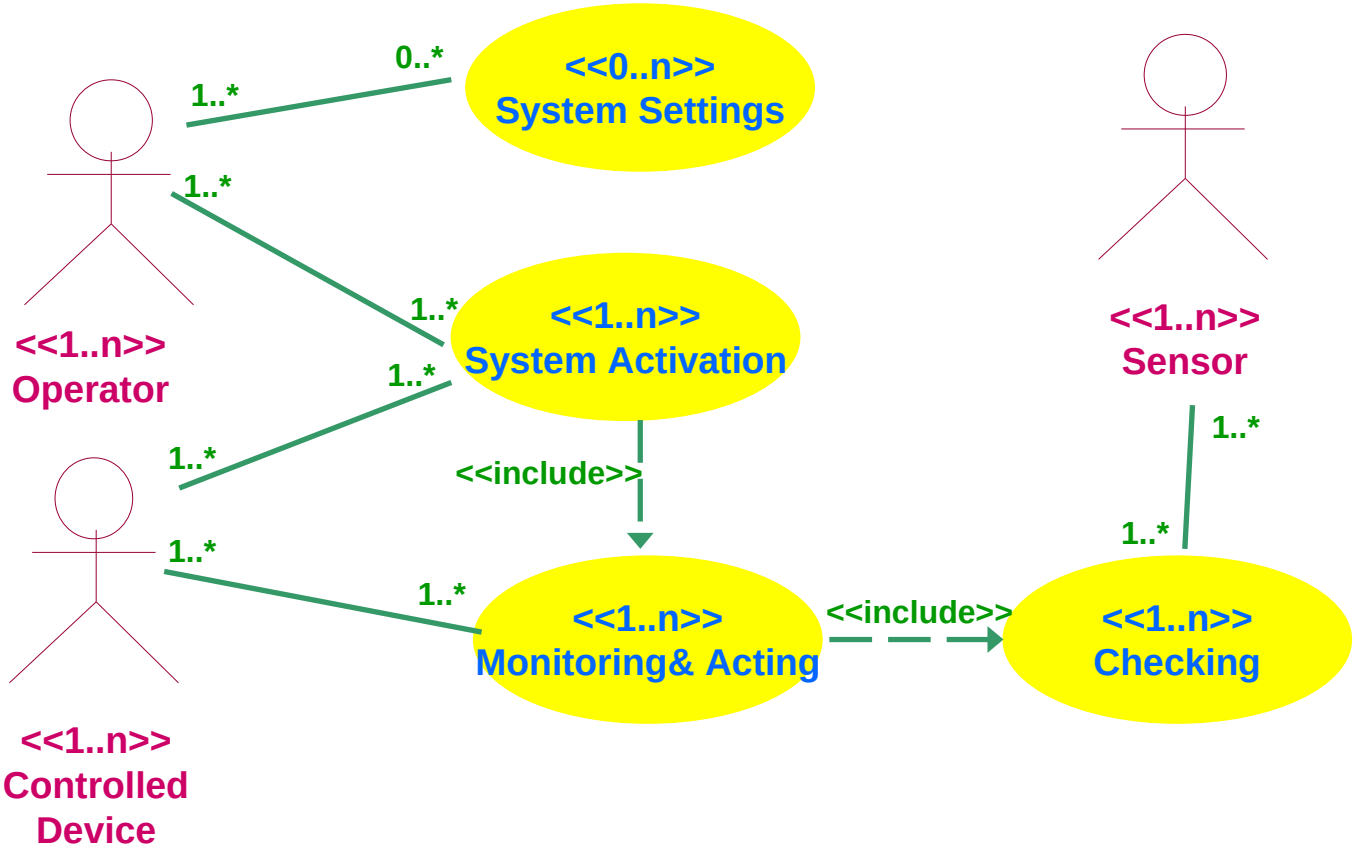
ADOM & UML through Examples

The Process Control Systems – Similarities



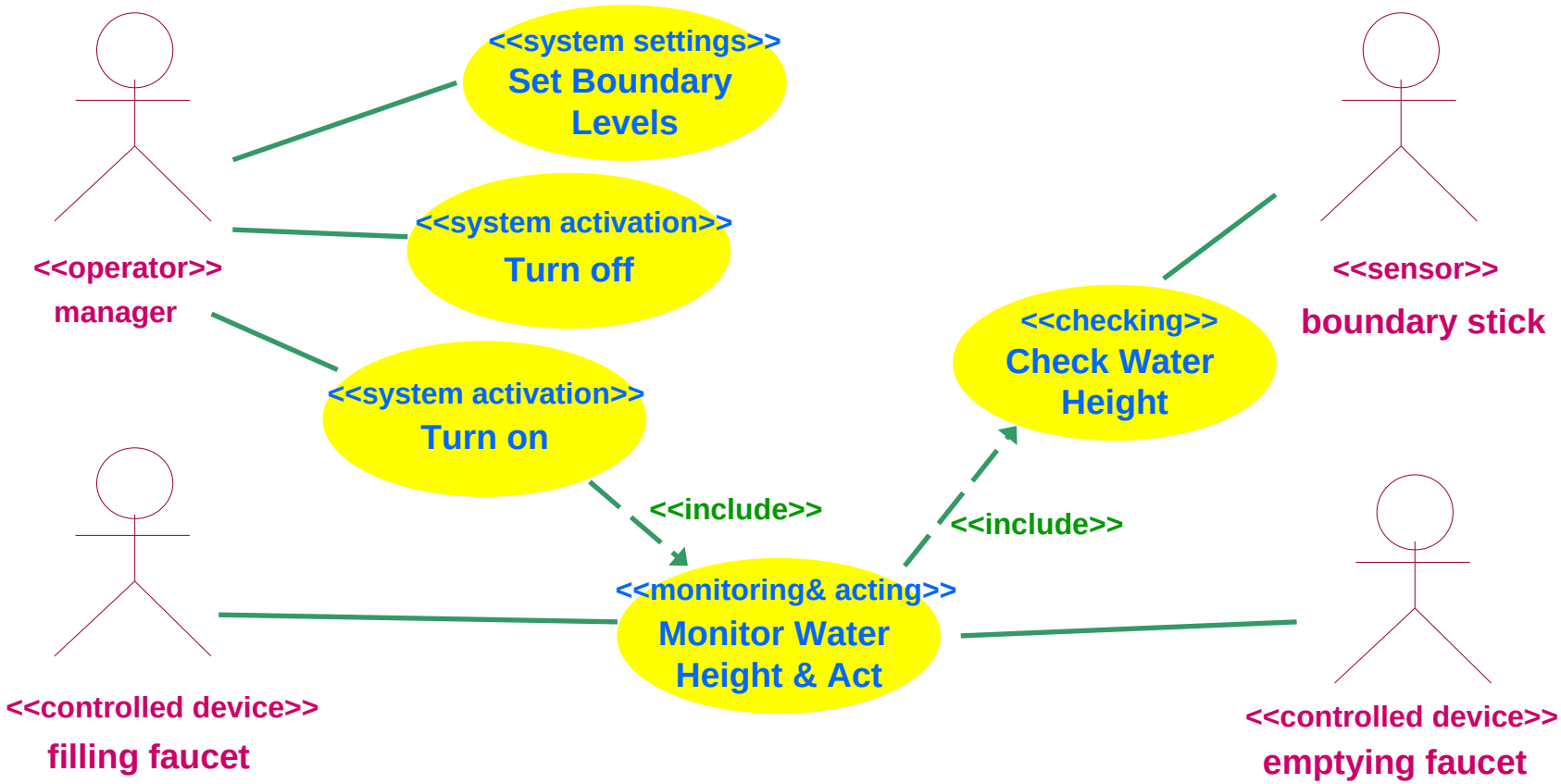
ADOM & UML through Examples

The Process Control Systems Use Case Diagram



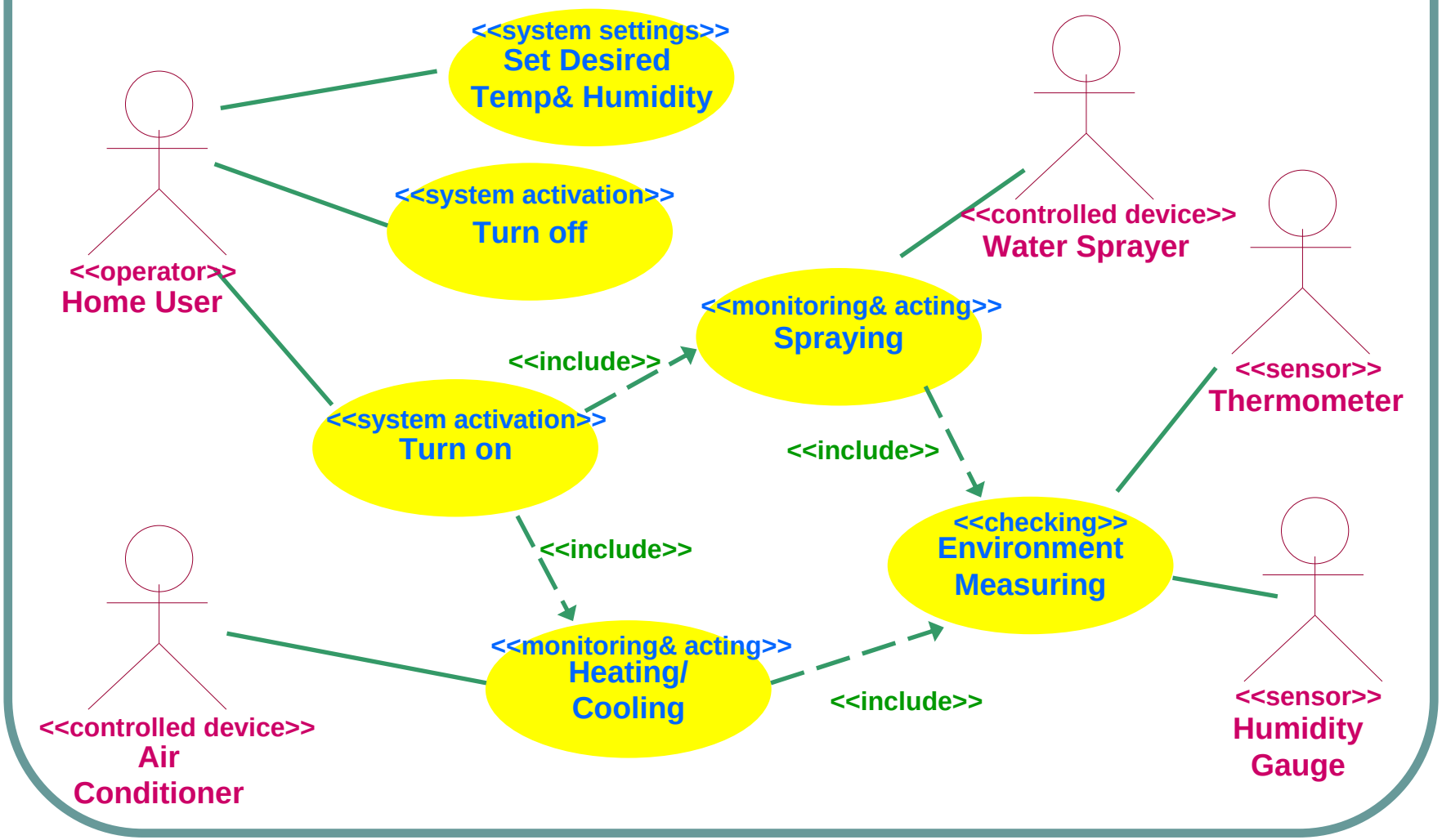
ADOM & UML through Examples

The Water Level Control Use Case Diagram



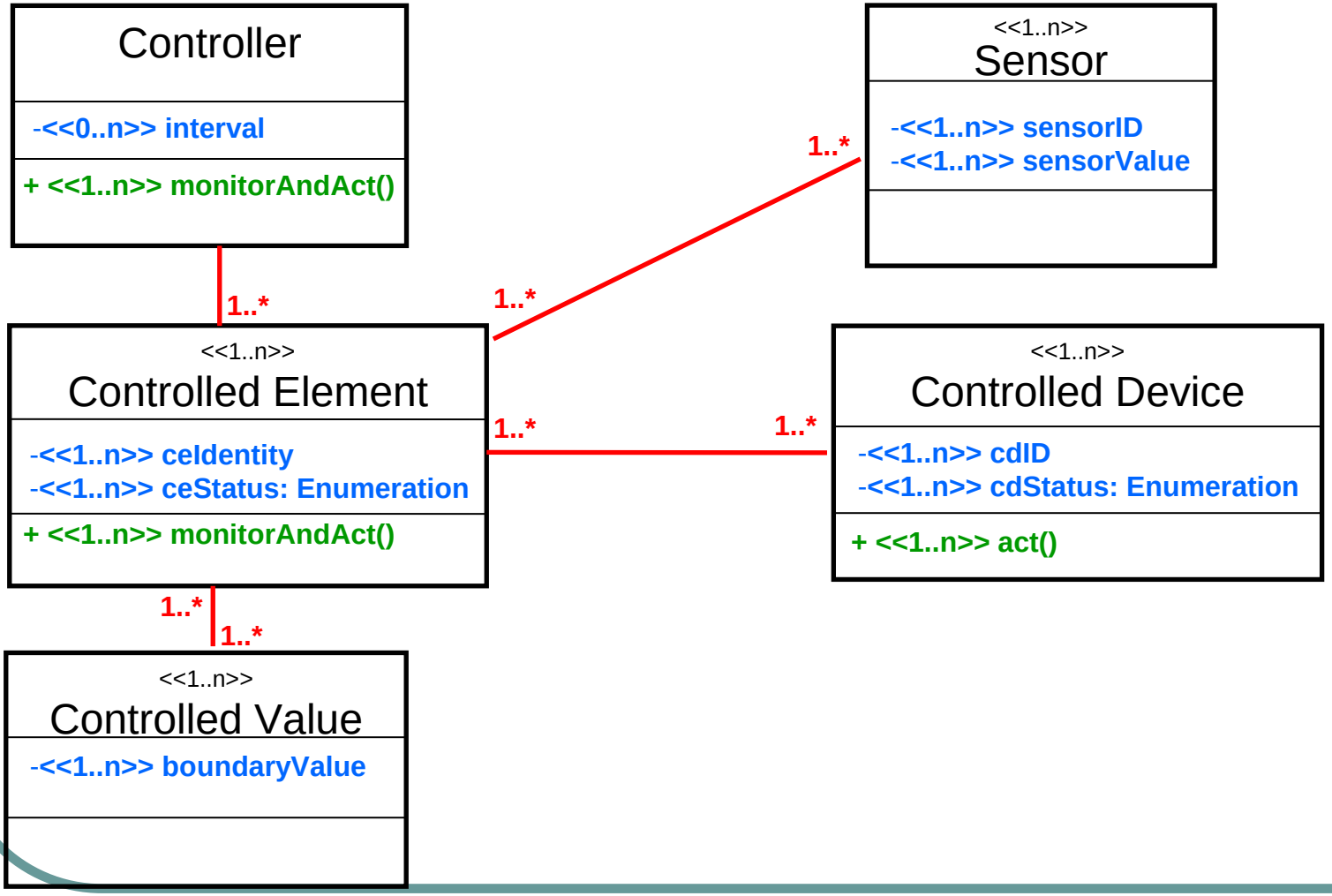
ADOM & UML through Examples

The Home Climate Control Use Case Diagram



ADOM & UML through Examples

The Process Control Systems Class Diagram



ADOM & UML through Examples
Class Diagram Comments – Attribute Signatures

- The general signature of an attribute in a domain model is:

<<attrMin..attrMax>> attrScope attrCName: attrType

  - <<*attrMin..attrMax*>> defines how many attributes of the particular application class can be classifies as the attribute classification name (*attrCName*)
  - The attribute scope (*attrScope*) is one of public, package, protected, or private. A scope of a domain model element is the least restricted scope that this element can get in an application model of that domain.
  - The attribute type (*attrType*) can be any atomic or composite type in UML (e.g., integer, Boolean, float, String, Date, etc.) or a set of those.
- Examples:
  - <<1..n>> private cdStatus: Enumeration ⇔Each Controlled Device has at least one private attribute of an enumeration type indicating the device status.

ADOM & UML through Examples
Class Diagram Comments – Operation Signatures

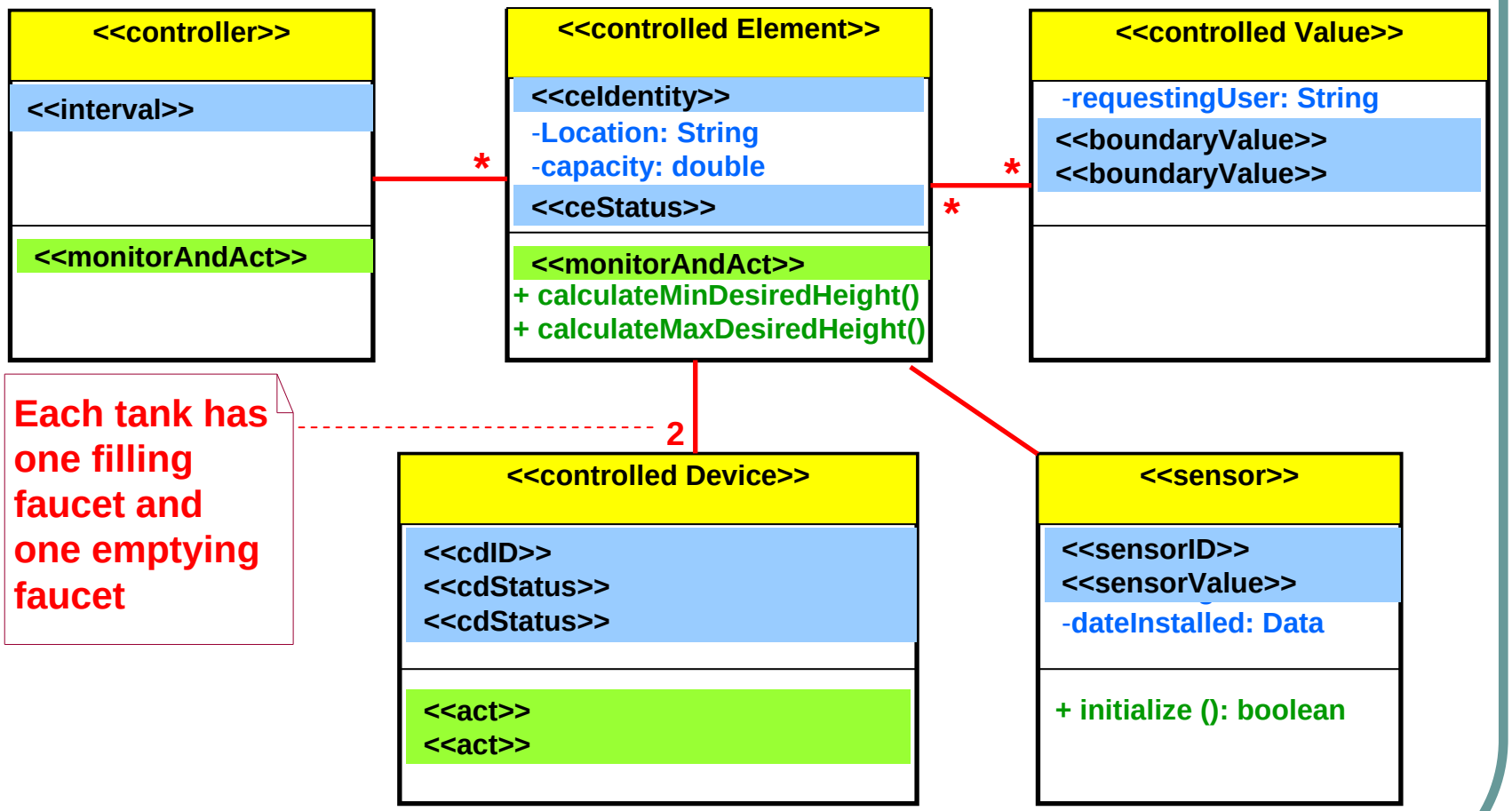
- The general signature of an operation in a domain model is:

<<opMin..opMax>> opScope opCName
(<<$param_1$Min..$param_1$Max>> $param_1$CName: $param_1$Type, …,
<<$param_n$Min..$param_n$Max>> $param_n$CName: $param_n$Type):
reType

  - <<*opMin..opMax*>> and <<*$param_i$Min..$param_i$Max*>> are multiplicity constraints
  - *opScope* is the scope constraint
  - *$param_i$Type* and *reType* are the type constraints
- Examples:
  - <<1..n>> public monitorAndAct(): Boolean ⇔Each Controller has at least one operation classified as monitorAndAct which returns a Boolean type. The number of parameters, their types, and the operation scope are not restricted in the domain level.

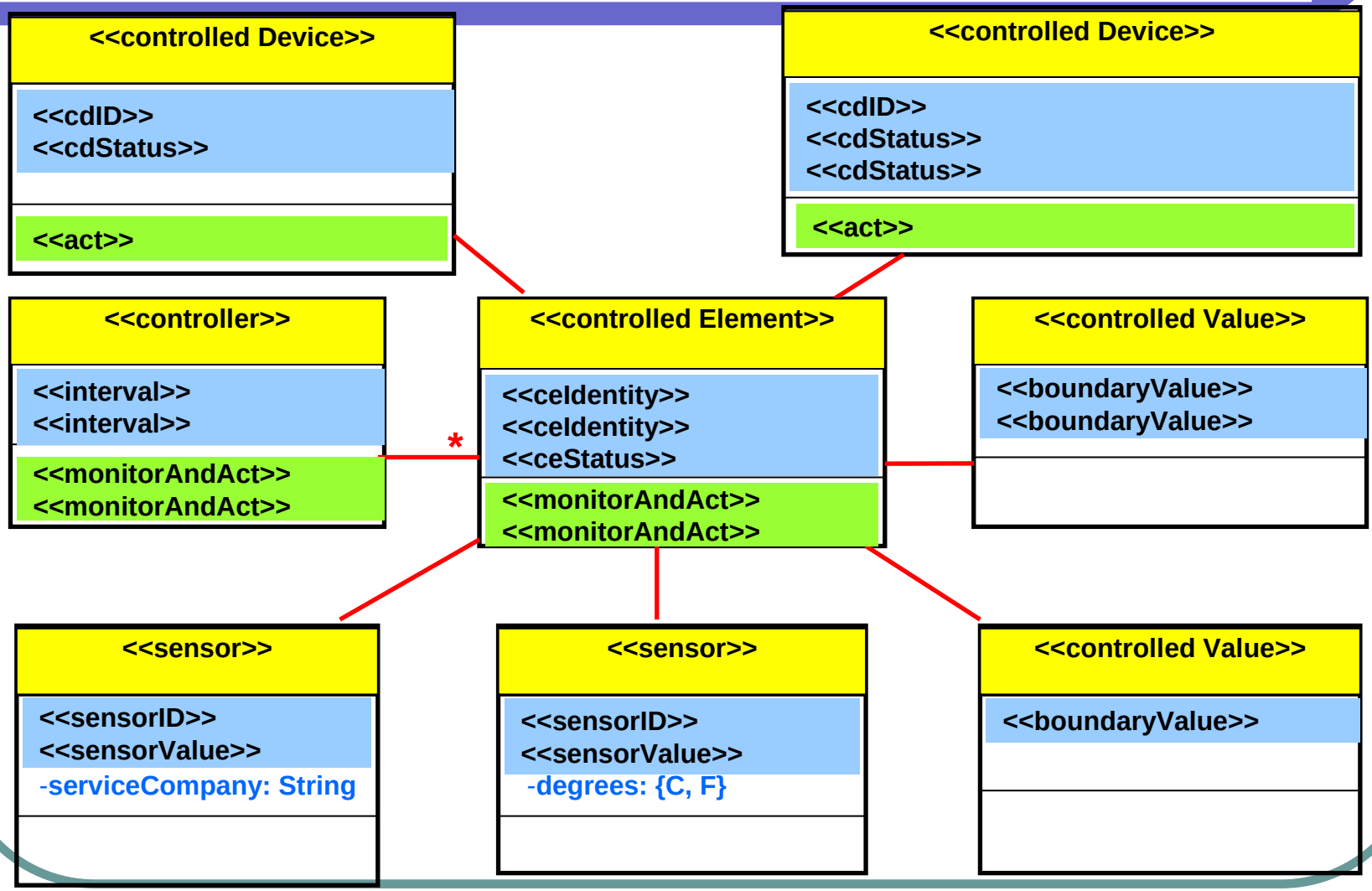
ADOM & UML through Examples

The Water Level Control Class Diagram



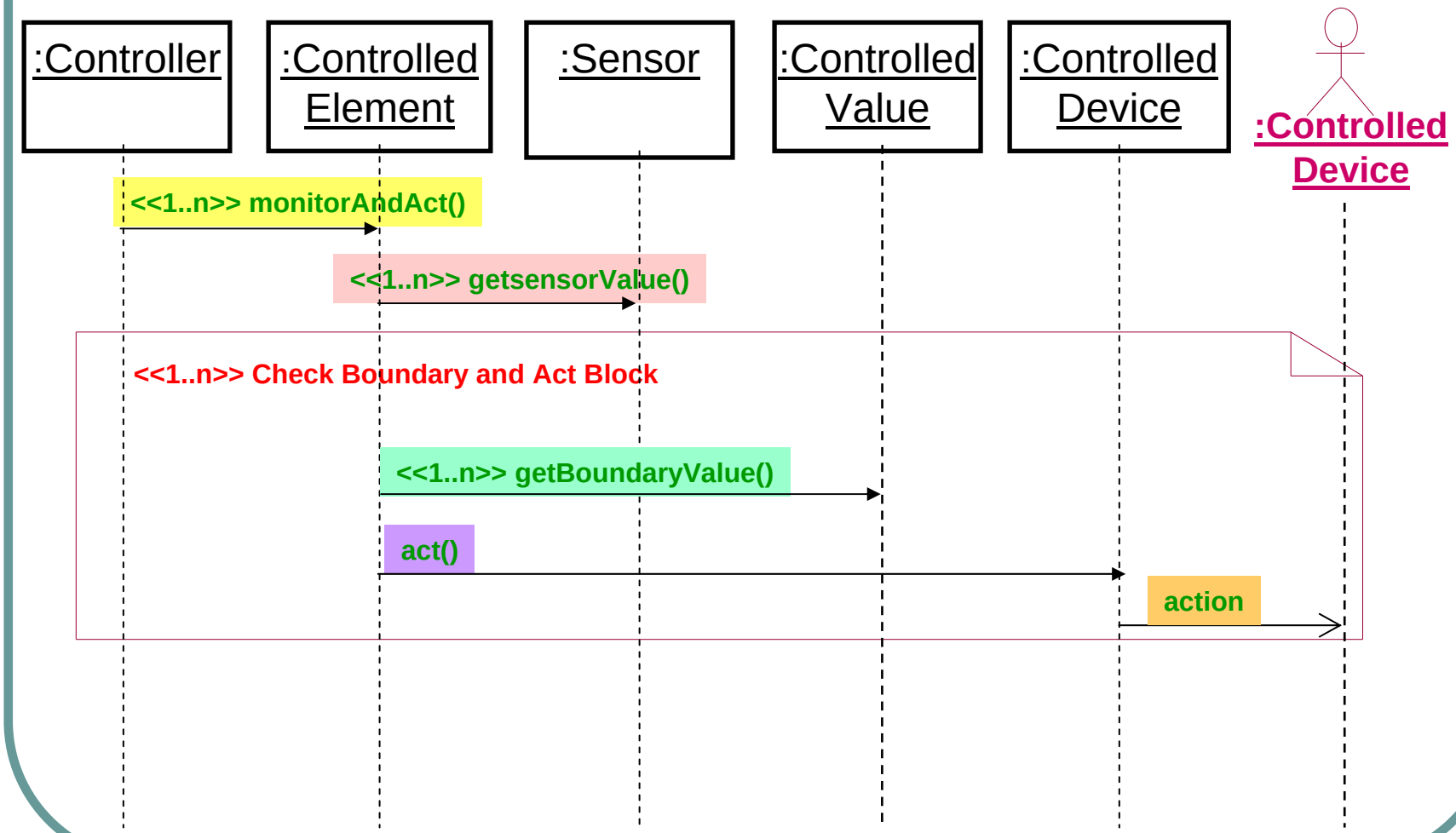
ADOM & UML through Examples

The Home Climate Control Class Diagram



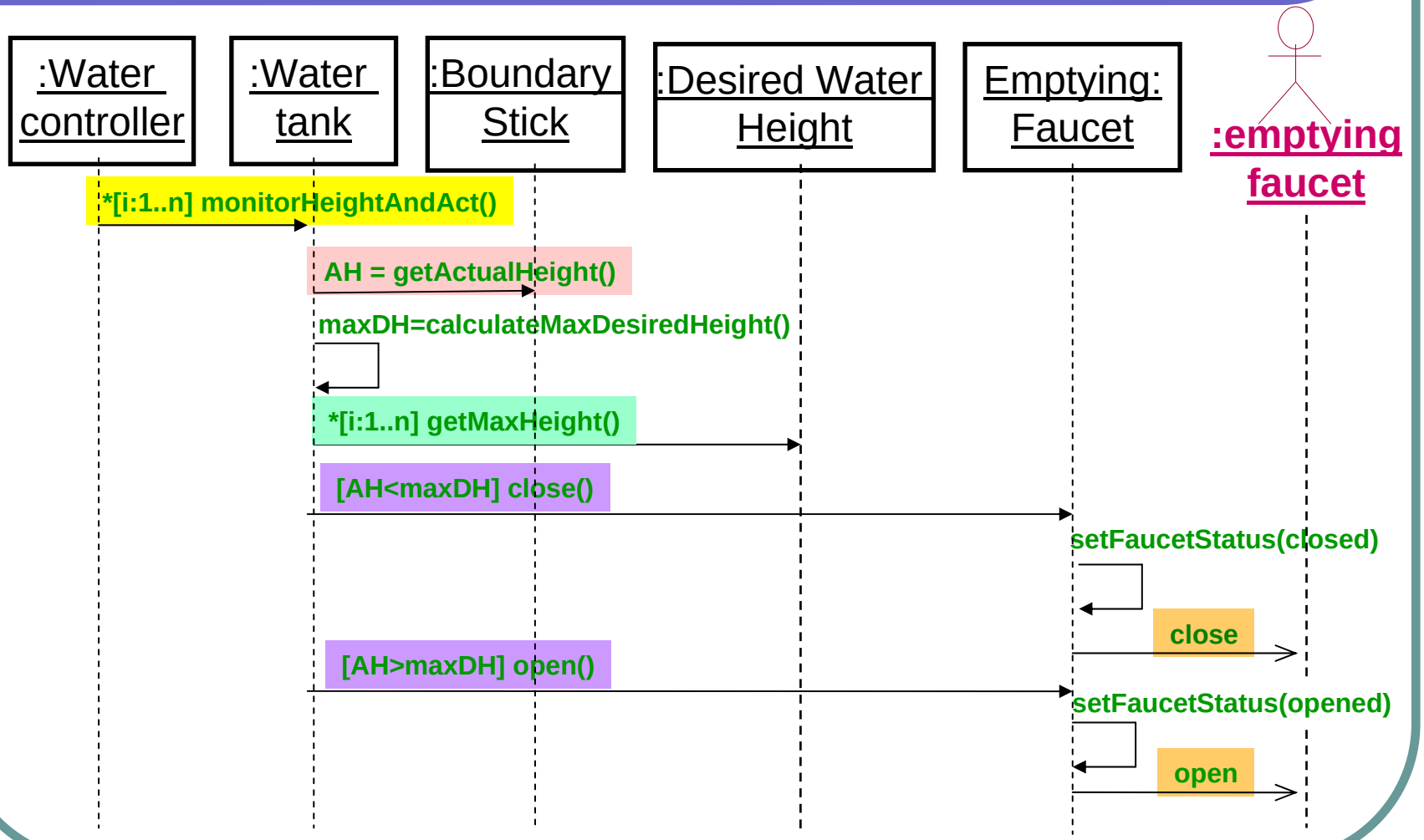
ADOM & UML through Examples

The Process Control Systems Sequence Diagram



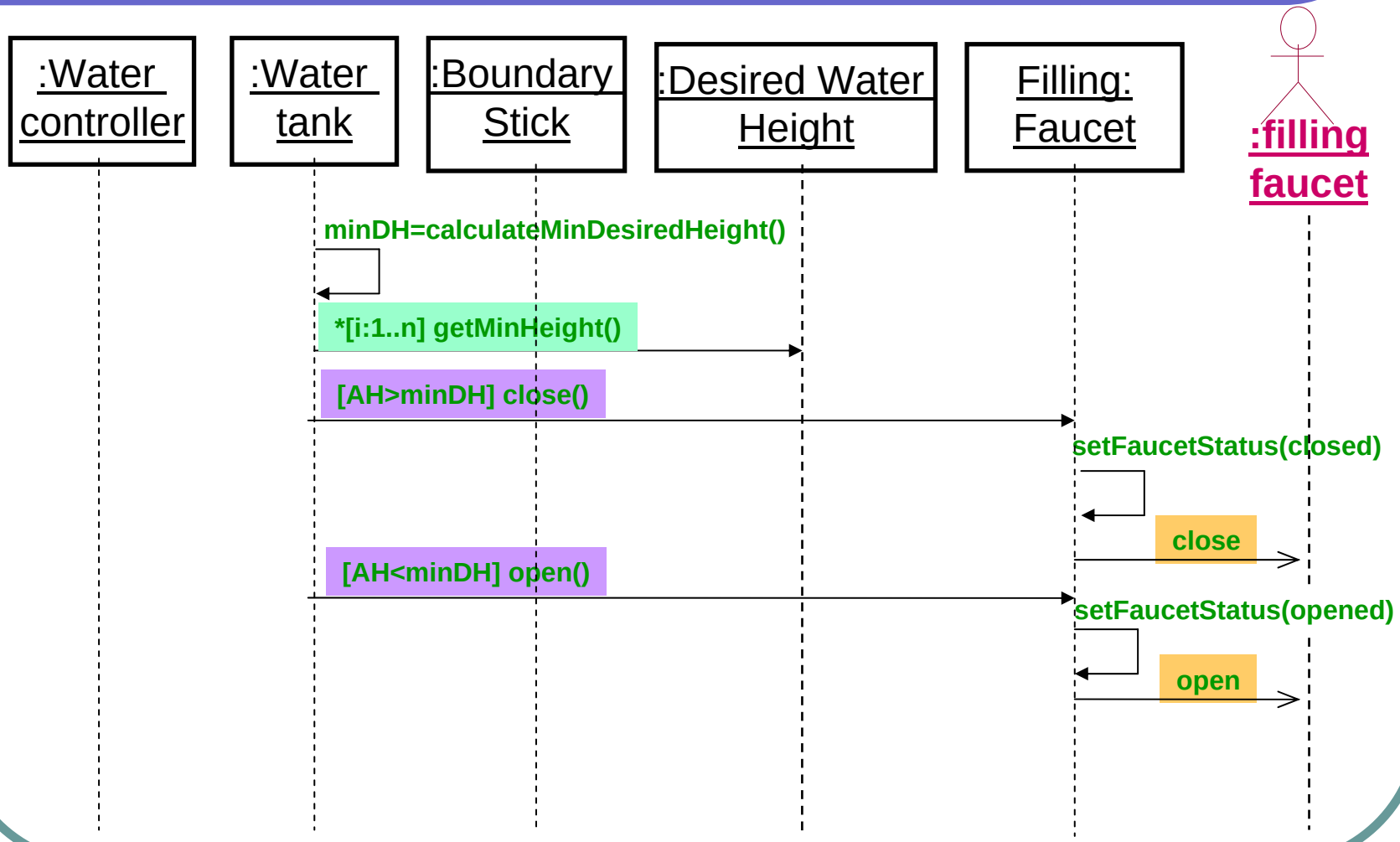
ADOM & UML through Examples

The Water Level Control Sequence Diagram



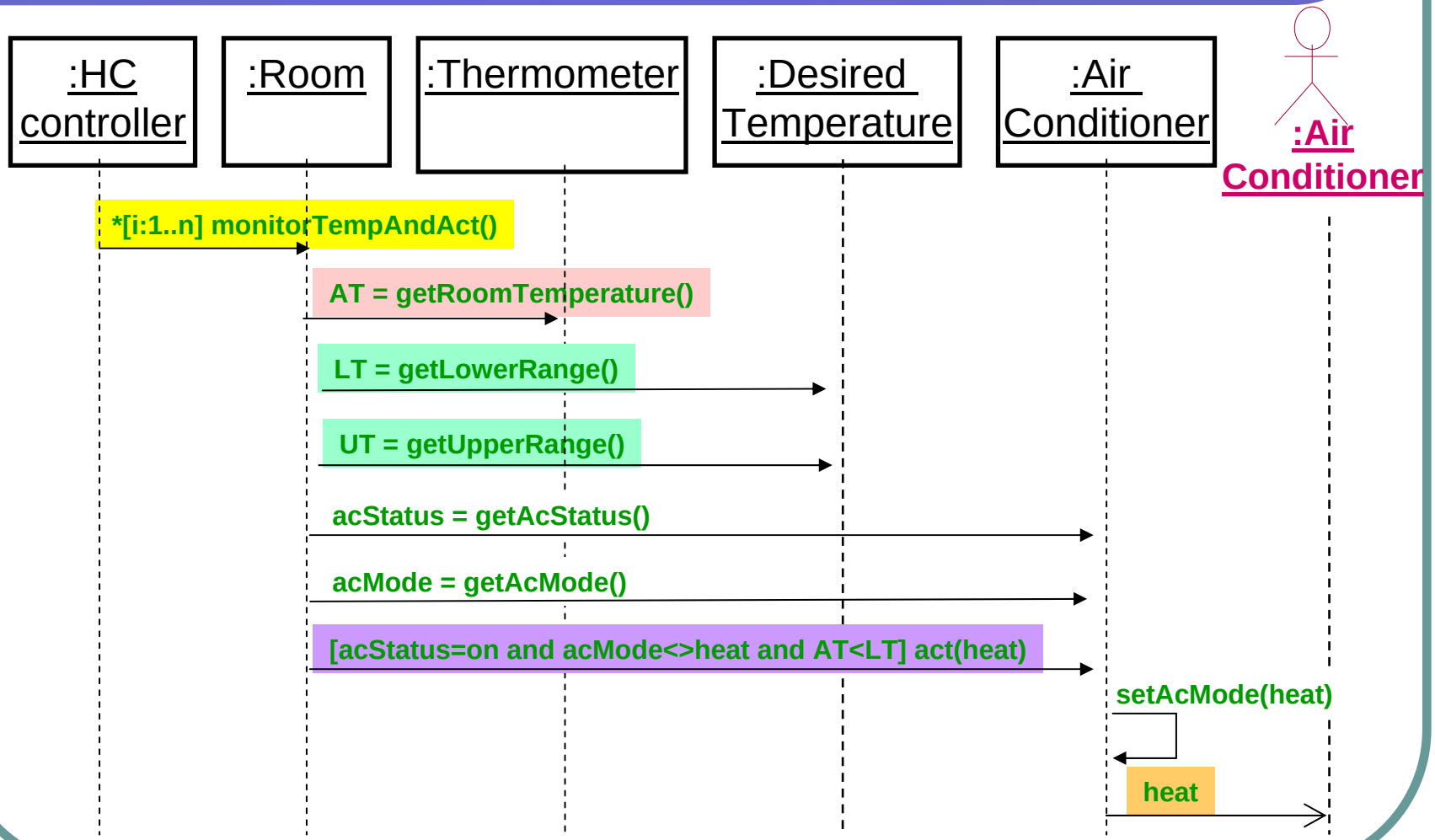
ADOM & UML through Examples

The Water Level Control Sequence Diagram



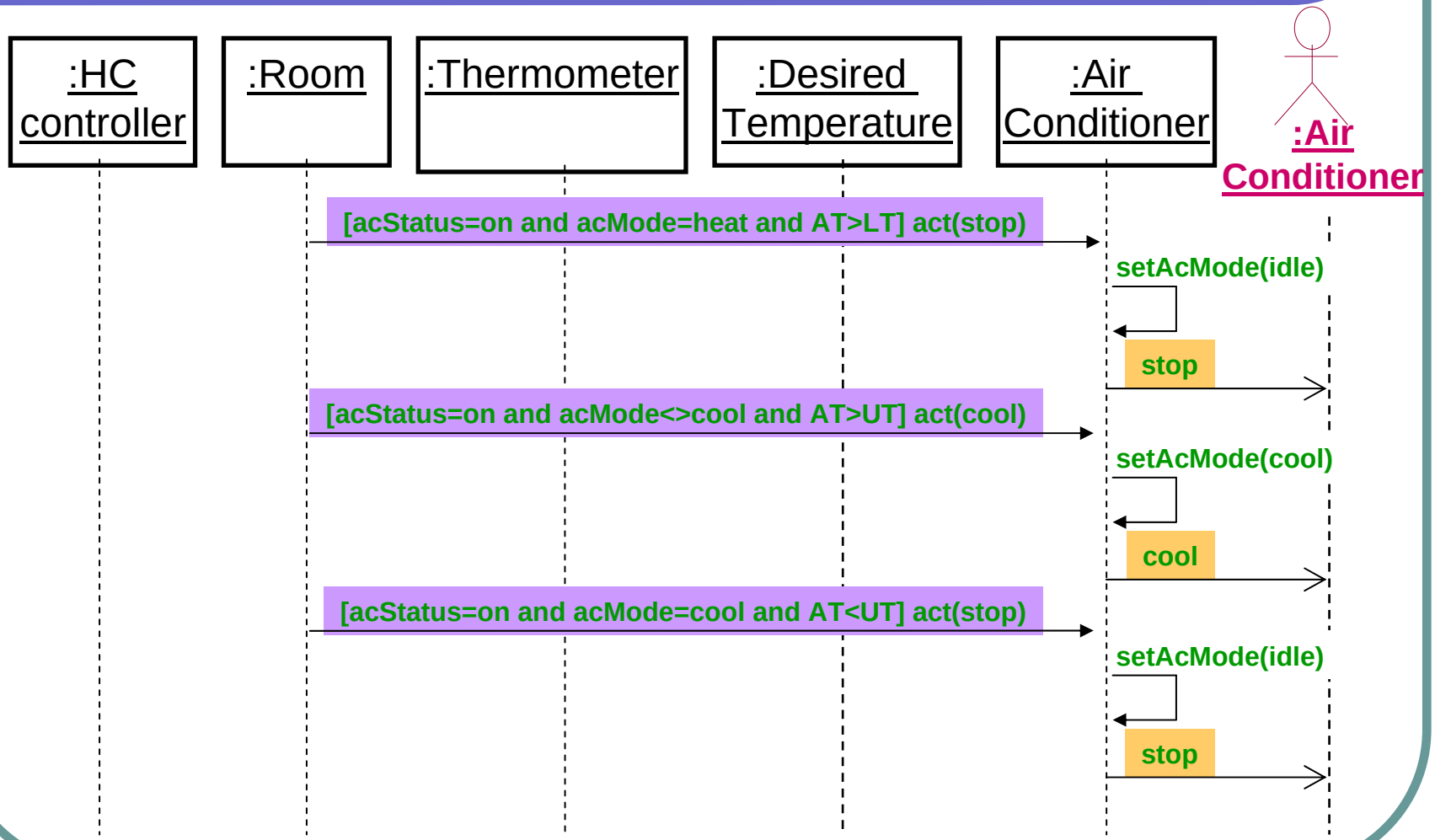
ADOM & UML through Examples

The Home Climate Control Sequence Diagram



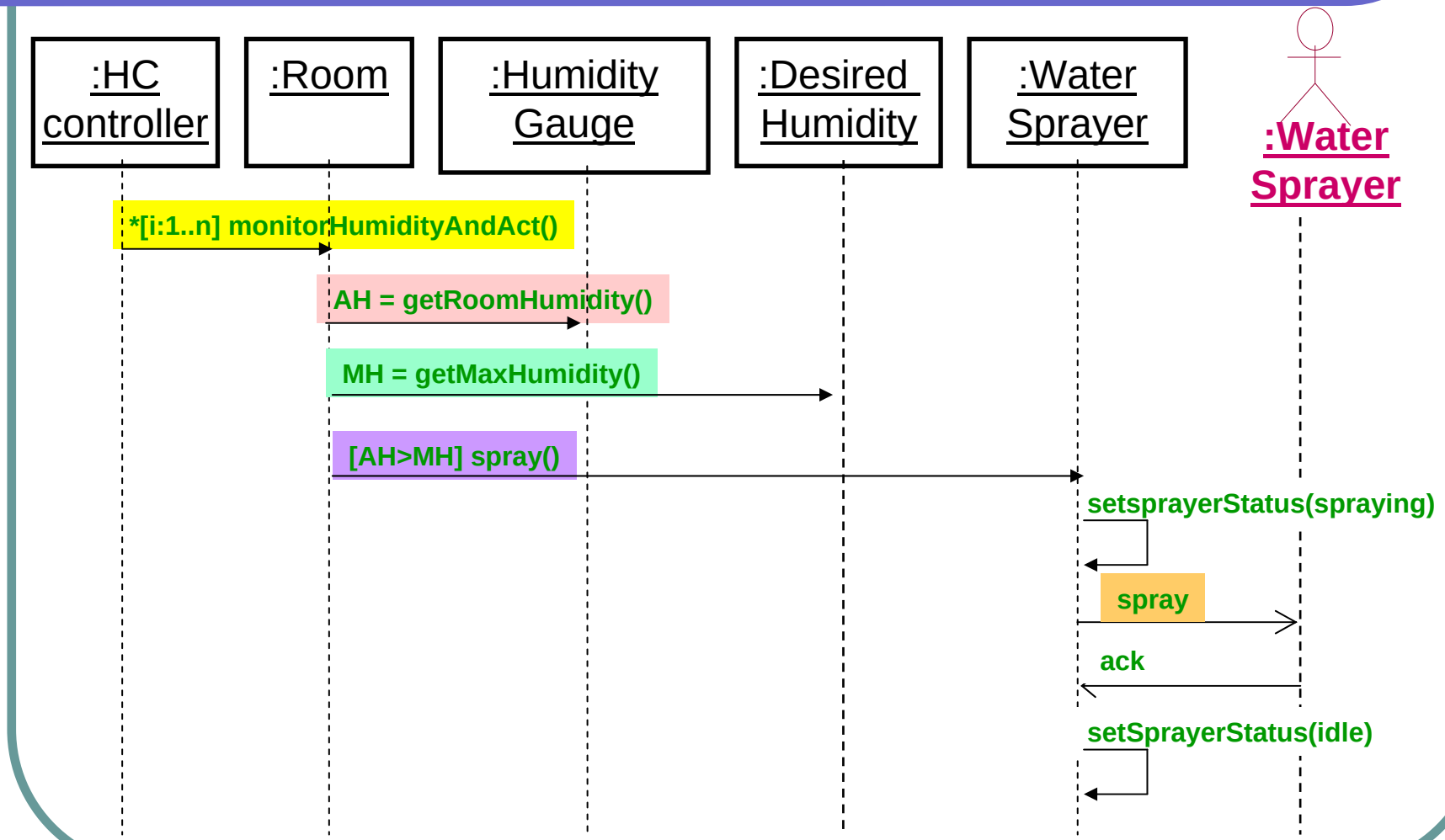
ADOM & UML through Examples

The Home Climate Control Sequence Diagram



ADOM & UML through Examples

The Home Climate Control Sequence Diagram



The ADOM Approach
Experimental Results

- **The experiments research questions:**
  - Does adding a domain model ease an individual's application comprehension?
  - Does adding a domain model help in creating more correct application models?
- **The subjects:**
  - Third year students in a four-year engineering B.Sc. program at the Ben-Gurion University of the Negev, Israel
  - All of them were students of the Information Systems Engineering program
  - They had no previous knowledge or experience in system modeling and specification
  - They took the course "Object-Oriented Analysis and Design" at the winter semester of the 2004-5 academic year

The ADOM Approach
Experimental Results

- **The experimental tasks:**
  - In the first task the students were asked to respond to nine true/false comprehension questions about the Home Climate Control (HCC) application. Examples of questions:
    - There are two types of devices that are controlled by the system
    - The system checks its sensor data, the thermometer and the humidity gauge, only through the Heating/Cooling use case
    - In each situation when the air-conditioner is on and the room temperature is lower than the lowest bound of the desired temperature, the heating operation of the air-conditioner is activated
  - In the second task the students were asked to build a model of the Water Level Control (WLC) application based on a set of requirements. They were asked to provide:
    - The system use case diagram
    - The system class diagram
    - A sequence diagram of tank filling
    - A statechart of a water faucet

The ADOM Approach
Experimental Results

- **Experimental Settings:**
  - The students were divided arbitrarily into two groups of 34 and 36 students
  - Each group got a different test form type:
    - The regular UML forms included only the HCC model: a use case diagram, a class diagram, a sequence diagram describing a heating/cooling scenario, and state diagrams describing the functionality of a water sprayer and an air-conditioner
    - The ADOM-UML forms included the domain model, as well as the HCC model
  - Checking the differences between the two groups according to the average grades of the students in their studies, no significant difference was found ($t = 0.32$, $p \sim 0.75$).

The ADOM Approach
Experimental Results

- Experiment Results

	Regular UML	ADOM-UML	t	p
Comprehension (out of 18)	12.42 (69.00%)	13.74 (76.33%)	1.68	< 0.01
Modeling (out of 32)	24.88 (77.75%)	28.51 (89.09%)	3.52	< 0.001

Remark: Although it can be argued that having more information or knowledge about the domain is the reason for the differences in the student achievements, having more diagrams in different abstraction level (application vs. domain) to consult with in exam conditions somehow balances the asymmetry between the two groups.

The ADOM Approach
Current State

- The ADOM approach was already applied to:
  - UML:
    - I. Reinhartz-Berger and A. Sturm, Behavioral Domain Analysis – The Application-based Domain Modeling Approach, the 7th International Conference on the Unified Modeling Language (UML'2004), Lecture Notes in Computer Science 3273, pp. 410-424, 2004.
    - A. Sturm and I. Reinhartz-Berger, Applying the Application-based Domain Modeling Approach to UML Structural Views, the 23rd International Conference on Conceptual Modeling (ER'2004), Lecture Notes in Computer Science 3288, pp. 766-779, 2004.
  - Activity Diagrams for enterprise modeling
    - I. Reinhartz-Berger, P. Soffer, and A. Sturm, A Domain Engineering Approach to Specifying and Applying Reference Models, accepted to Enterprise Modeling and Information Systems Architectures (EMISA'05), 2005.
  - Object-Process Methodology (OPM)

The ADOM Approach
Perspectives

- **The reuse perspective**
  - ADOM supports open reuse by incorporating domain elements into application models
    - Basic elements: the modeling language elements
    - Features: structural and behavioral views and constraints
    - Size: varies
    - Way of reuse: utilizing domain elements
- **The knowledge engineering perspective**
  - ADOM provides a means for knowledge representation
- **The validation and verification perspective**
  - ADOM provides a means for specifying validation templates (i.e., domain models) for application modeling

The ADOM Approach
Advantages

- Treating domains similarly to applications enables the specification of behavioral constraints and not just structural ones.
- ADOM can be used for validating system models against their domain models in order to detect semantic errors in early development stages.
  - Validating an application model throughout gradual system development stages reduces the development cost as errors are detected in early stages
  - These errors cannot be automatically found when using syntactic modeling languages alone
- Treating domains in a separate layer (and not in the metamodel layer) enables adjustment of ADOM to different modeling languages.
- The usage of the same modeling language for both the application and domain layers can reduce the ontological gap and the communication problems between the different stakeholders in the system development process.

The ADOM Approach
Advantages

- Applying ADOM specifically to UML, the standard object-oriented modeling language, also benefits from the maturity of the UML environment, including its CASE tools.
- The combination of ADOM and UML establishes a formal framework for defining and constraining stereotypes in UML.
  - To the best of our knowledge the only way to define stereotypes and UML extensions (till now) was via a natural language
  - While using natural languages is more comprehensible to humans, it lacks the needed precision and formality for defining domain elements, constraints, and usage contexts

The ADOM Approach
Limitations & Future Work

- ADOM has no supporting tools (yet).
  - Short term: Developing a UML-based CASE tool that supports ADOM
  - Long term: Developing an ADOM guiding tool that can be plugged into various CASE tools
- The completeness and expressiveness of the constraints in ADOM have not been checked (yet).
  - Short term: Using OCL in order to support additional types of constraints, such as requiring that a class operation will get no parameters in any application model of the domain
  - Long term: Defining the set of constraints needed for any domain engineering method and refining it to ADOM and particular languages

The ADOM Approach
Limitations & Future Work

- As many domain engineering techniques and methods, the ADOM approach can be criticized as dealing with too broad areas (domains) which are usually understood only during the development process.
  - Most of these problems can be solved by taking a special care of the declaration of domain scopes and the way domain engineering is woven into software engineering
  - We plan to check these problems on different large domains, such as e-commerce applications and training simulators

The End...

Questions can be addressed to: iris@mis.hevra.haifa.ac.il

Tutorial 3

Appendices

Tutorial 3

Domain Analysis Techniques
An Example of Feature Modeling in PLUS

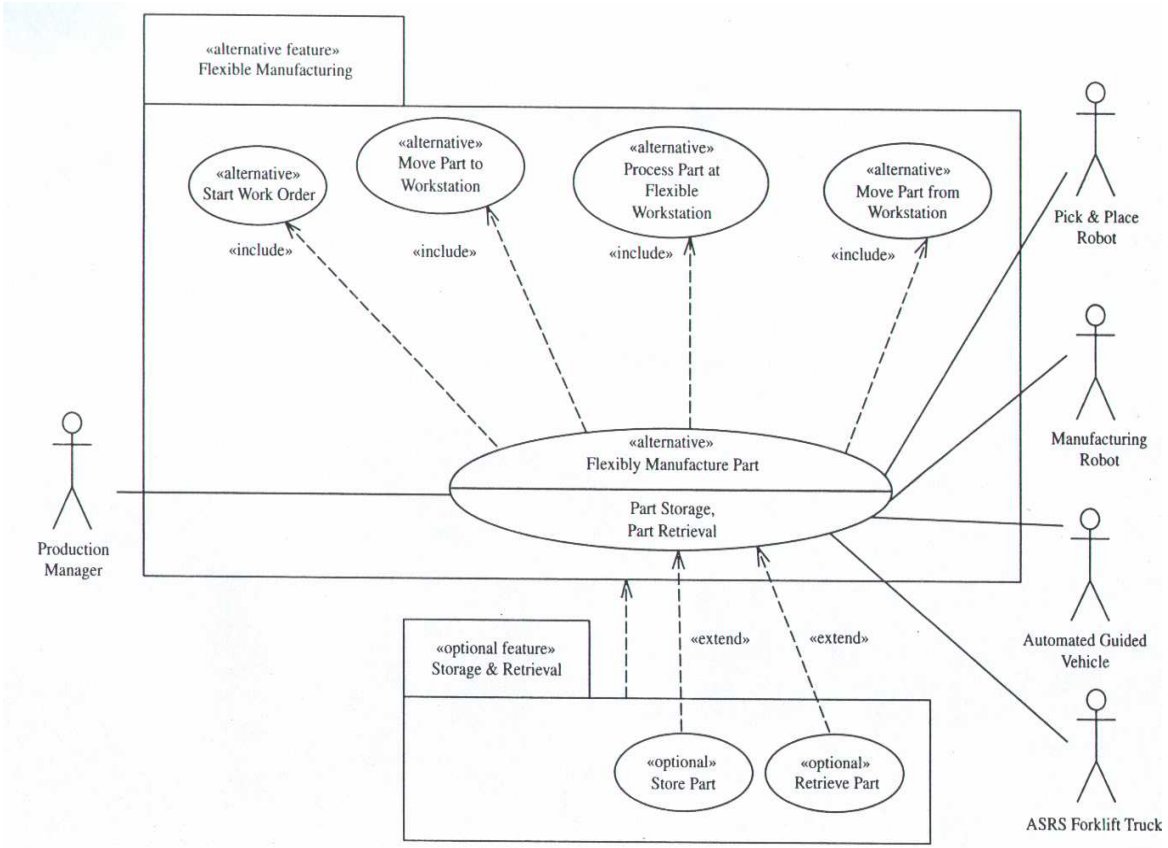


Figure 5.3 `Flexible Manufacturing` *use case and feature dependencies*

From: Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004

Domain Analysis Techniques
An Example of Static Modeling in PLUS

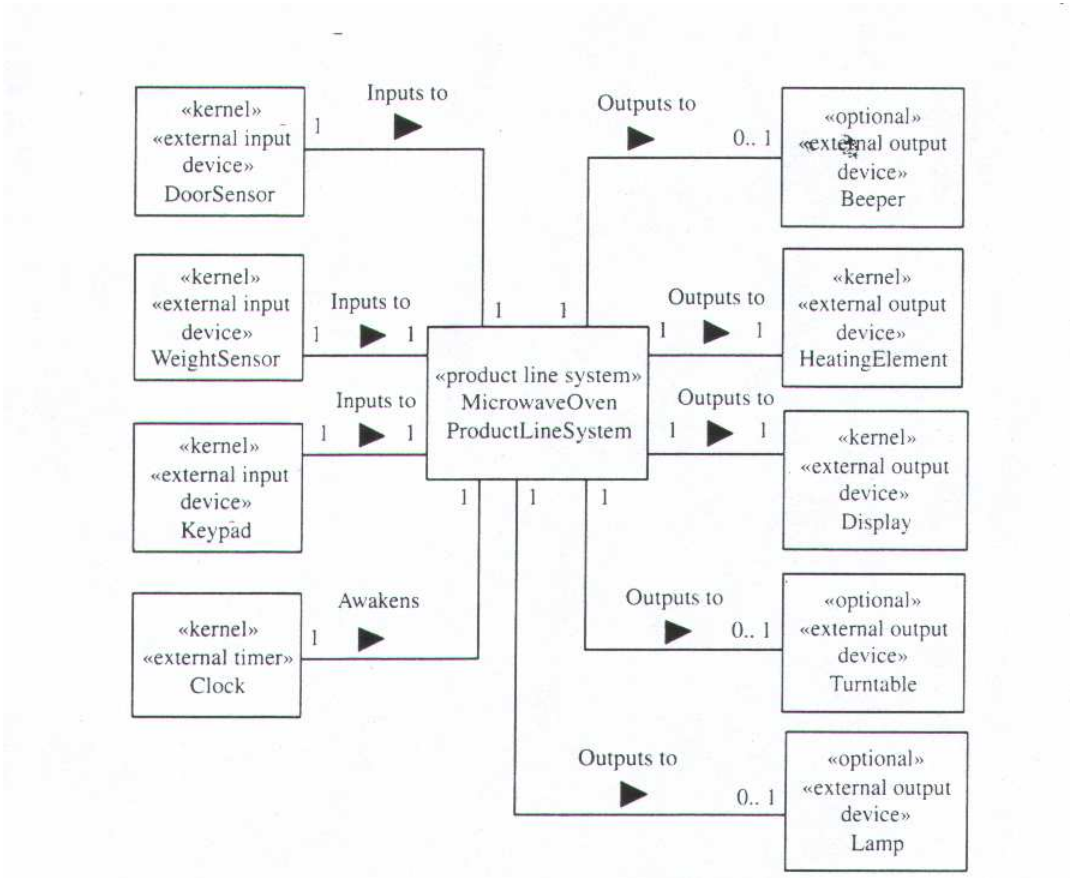


Figure 6.6 *Microwave oven product line context class diagram*

From: Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004

Domain Analysis Techniques
An Example of Dynamic Modeling in PLUS

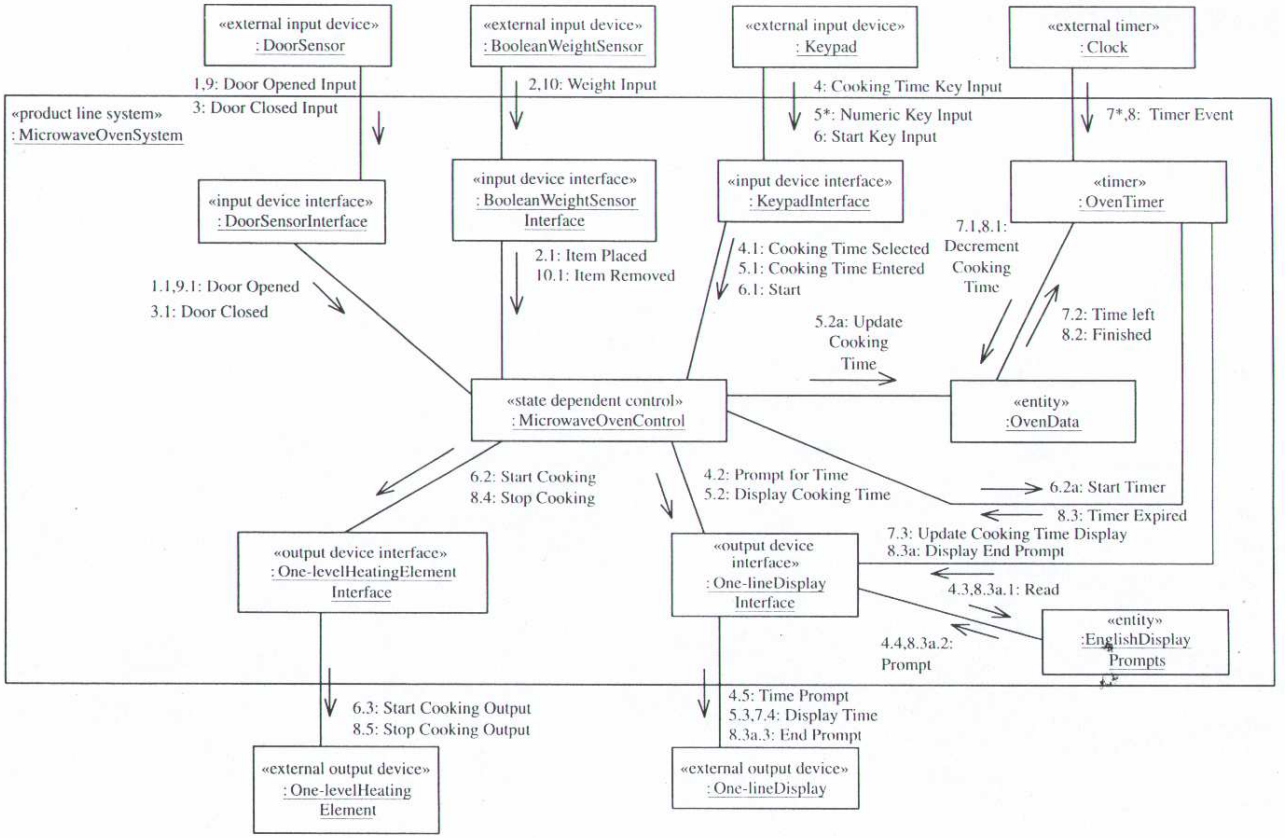


Figure 7.6 *Communication diagram for a kernel use case:* Cook Food

From: Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004

Domain Analysis Techniques
An Example of Dynamic Modeling in PLUS

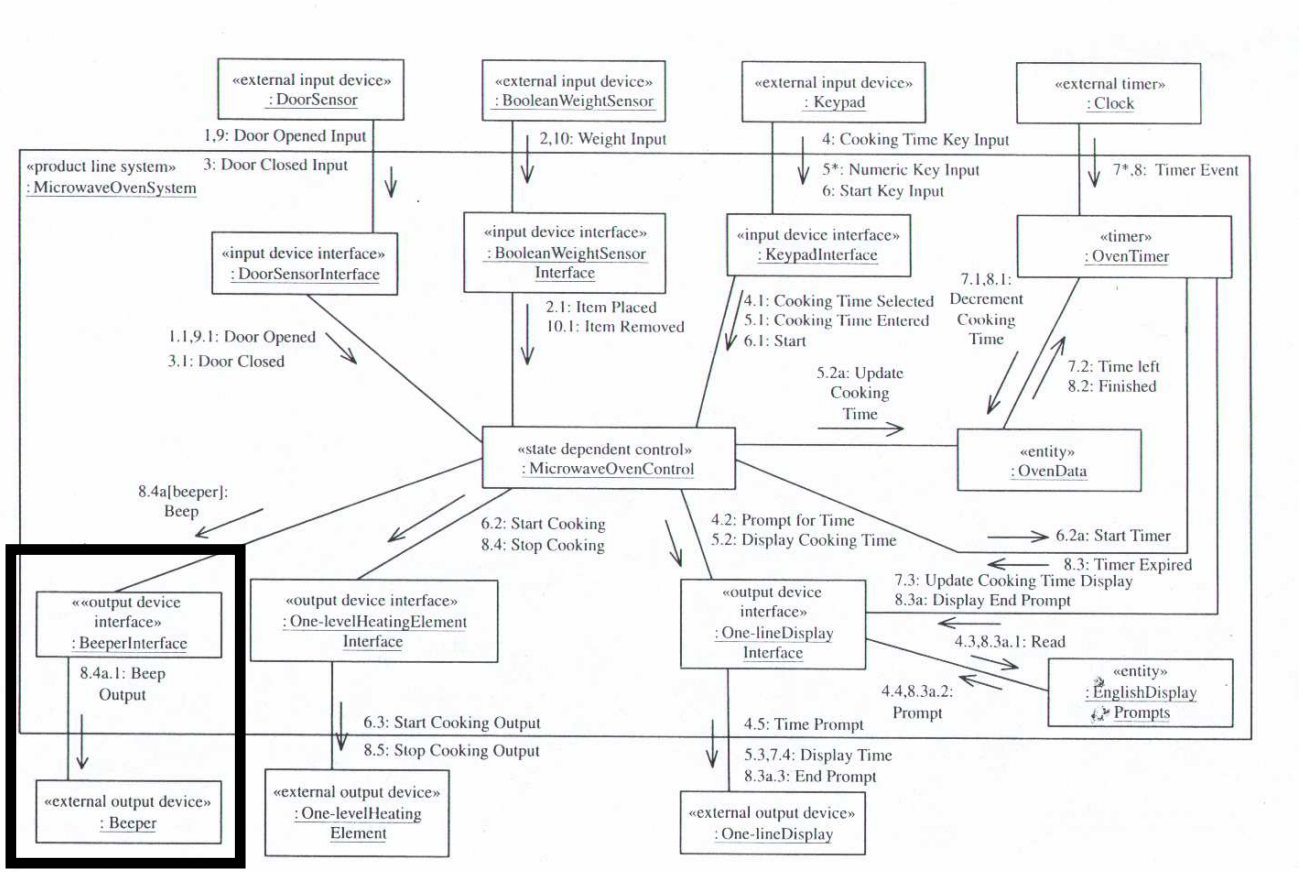
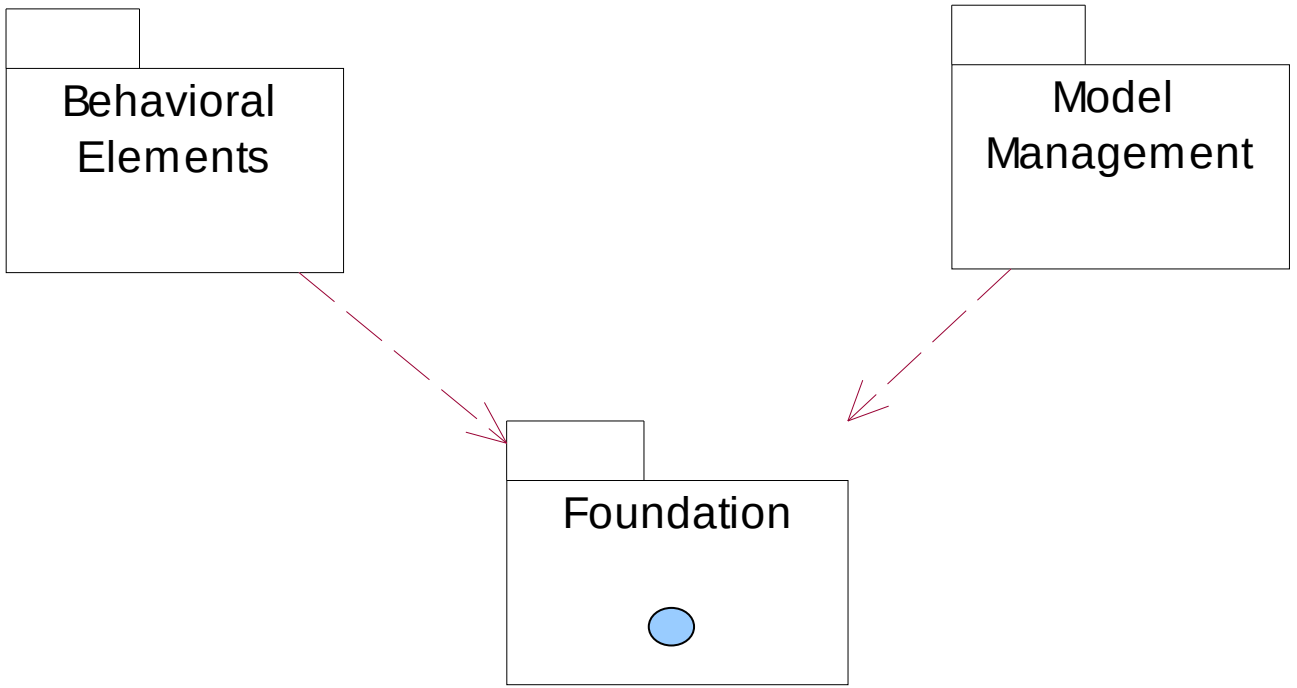


Figure 7.7 *Variant communication diagram depicting the impact of the* `Beeper` *variation point and optional feature*

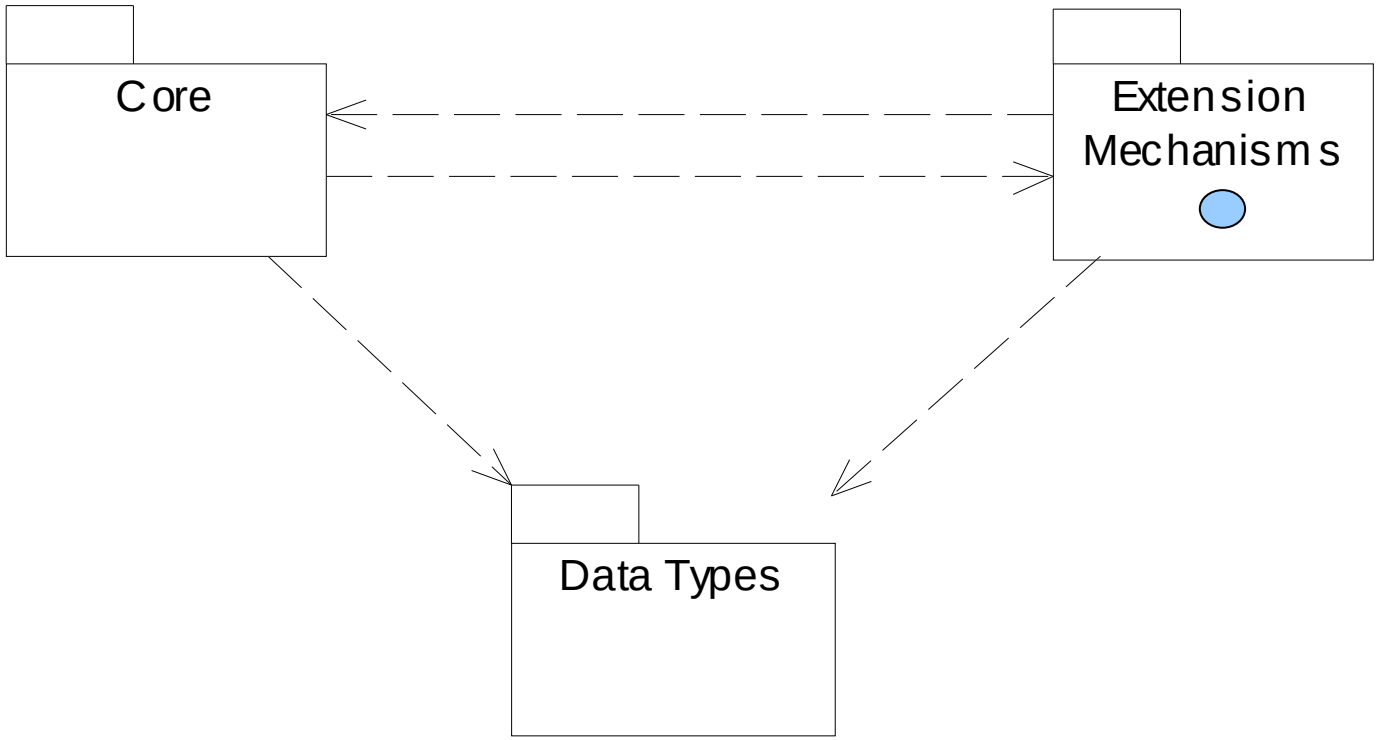
From: Hassan Gomaa, Designing Software Product Lines with UML, Addison Wesley, 2004

Domain Analysis Techniques
The UML Metamodel – Version 1.3

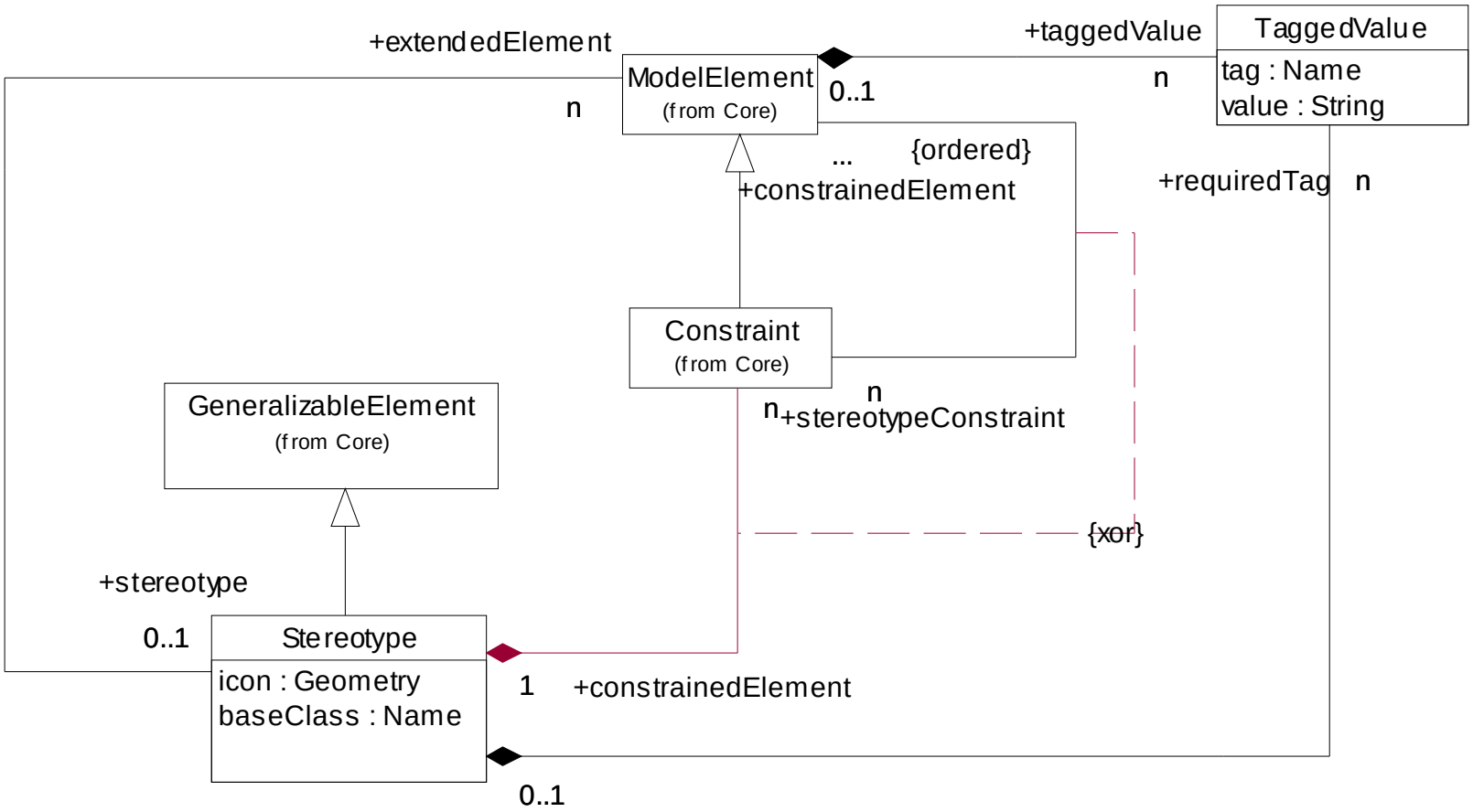
UML Metamodel v. 1.3 R20: Top-Level Packages



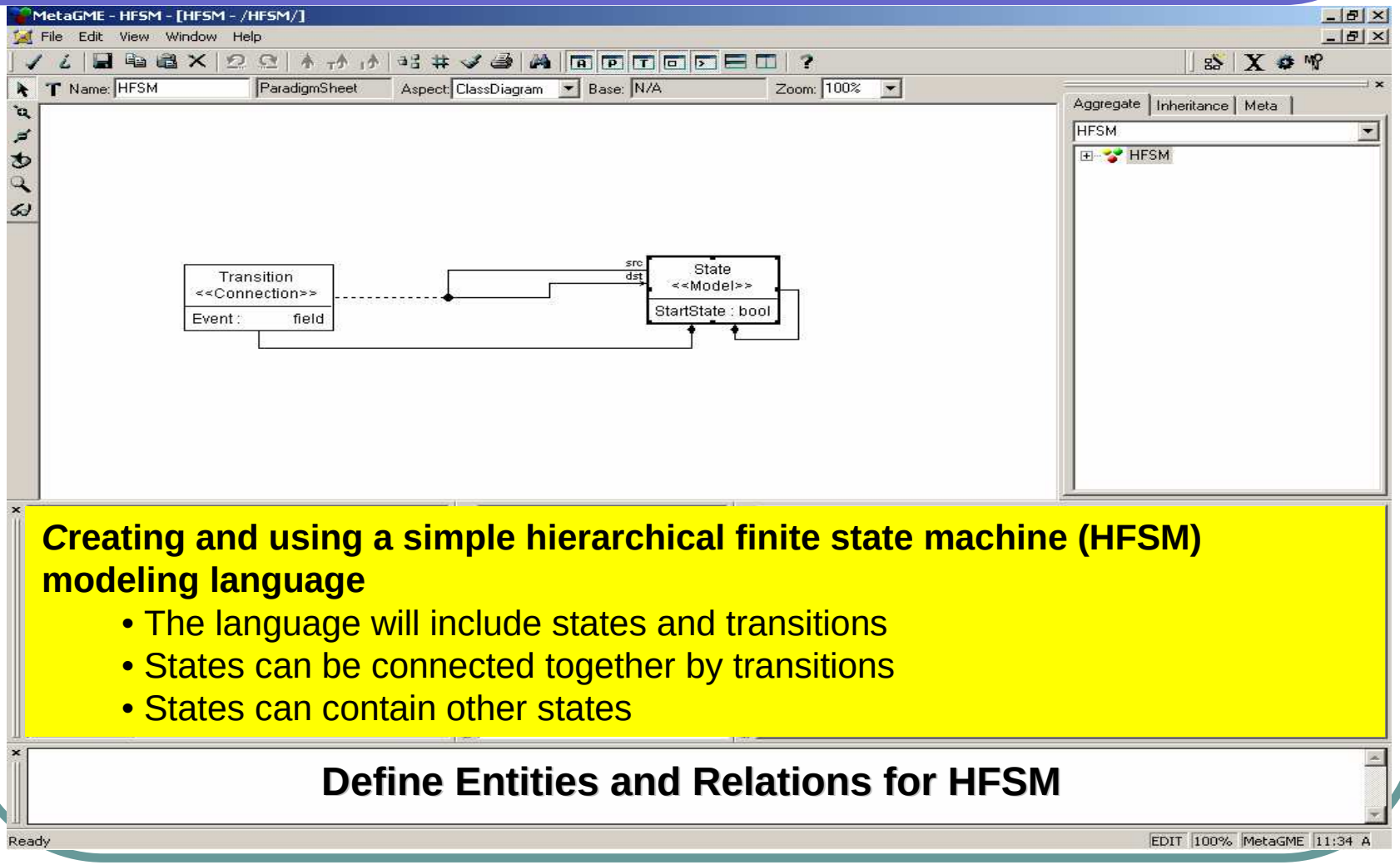
Domain Analysis Techniques
The UML Metamodel – Version 1.3



Domain Analysis Techniques
The UML Metamodel – Version 1.3



Domain Analysis Techniques
Generic Modeling Environment (GME)

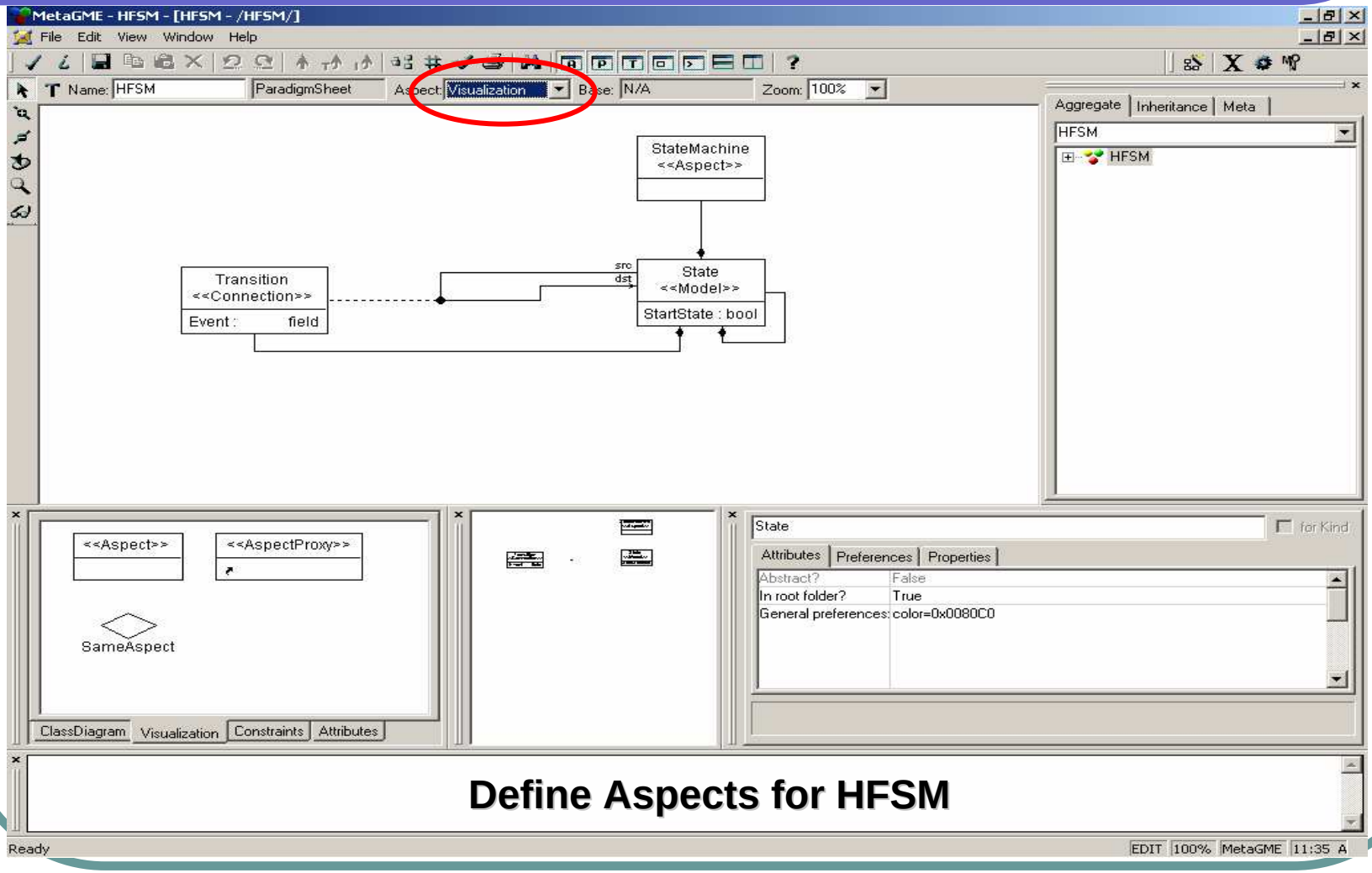


Creating and using a simple hierarchical finite state machine (HFSM) modeling language

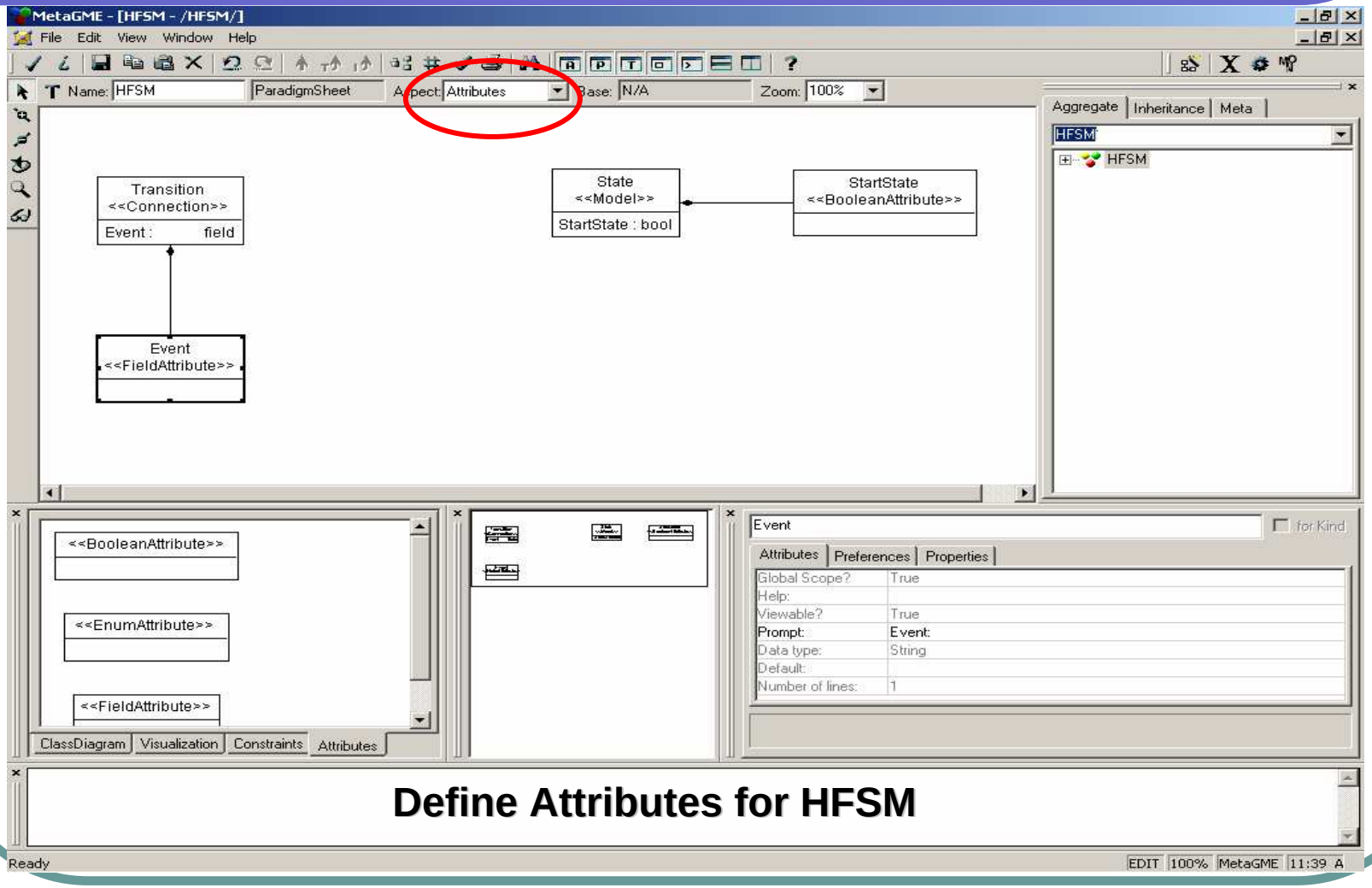
- The language will include states and transitions
- States can be connected together by transitions
- States can contain other states

Define Entities and Relations for HFSM

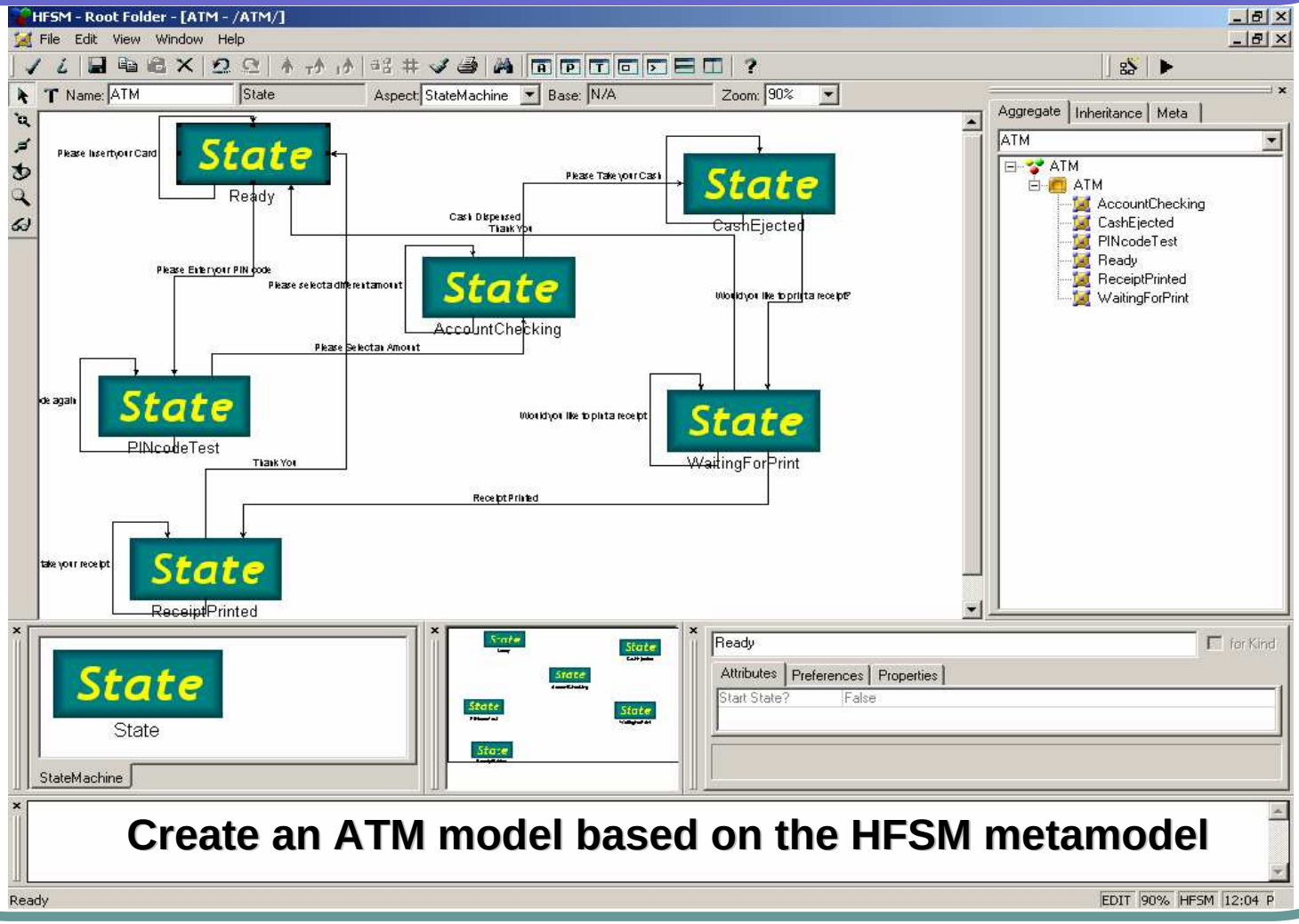
Domain Analysis Techniques
Generic Modeling Environment (GME)



Domain Analysis Techniques
Generic Modeling Environment (GME)

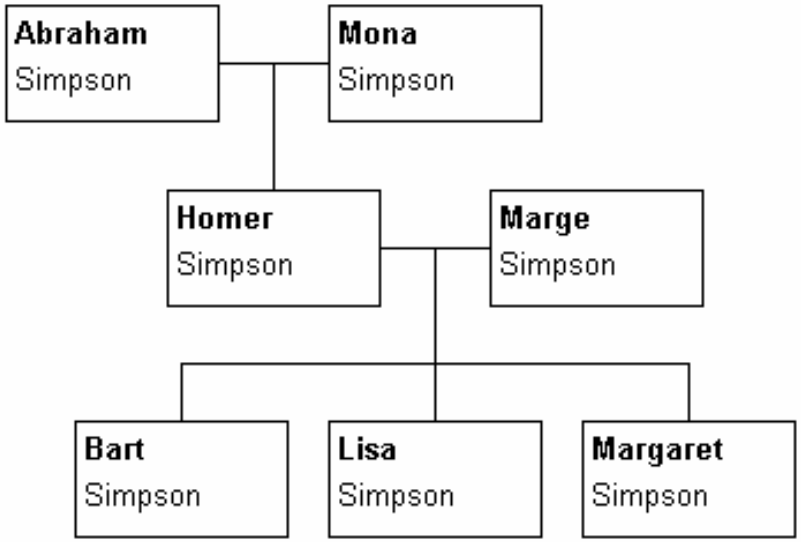


Domain Analysis Techniques
Generic Modeling Environment (GME)

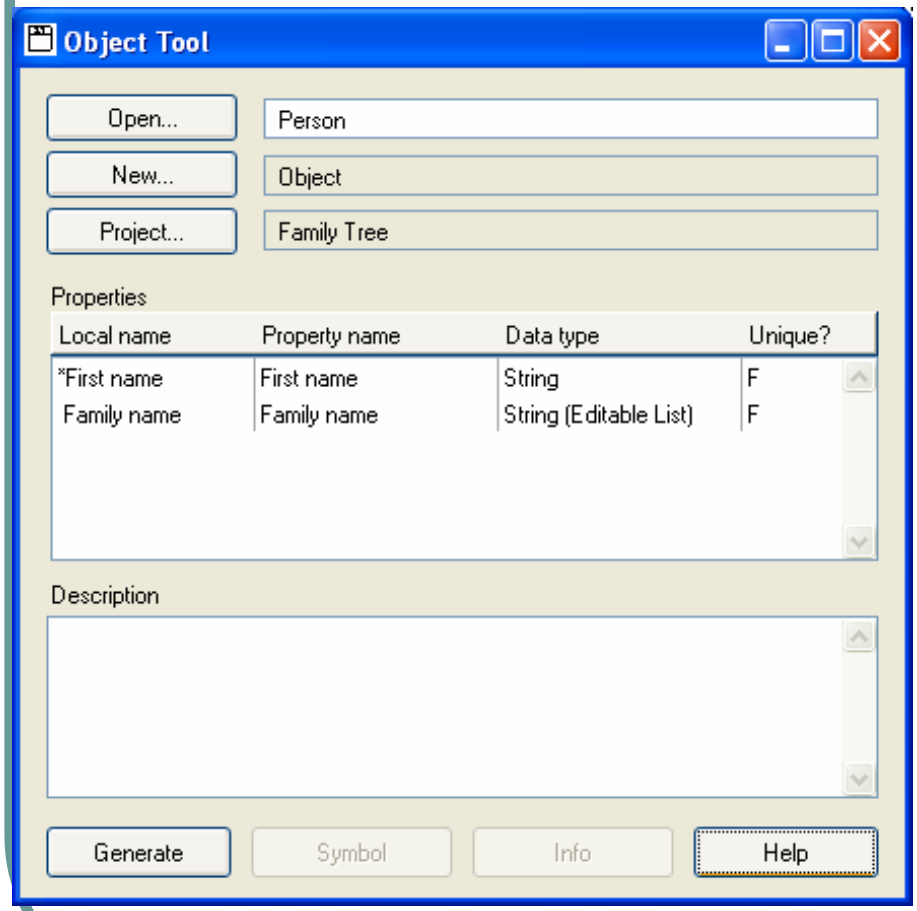


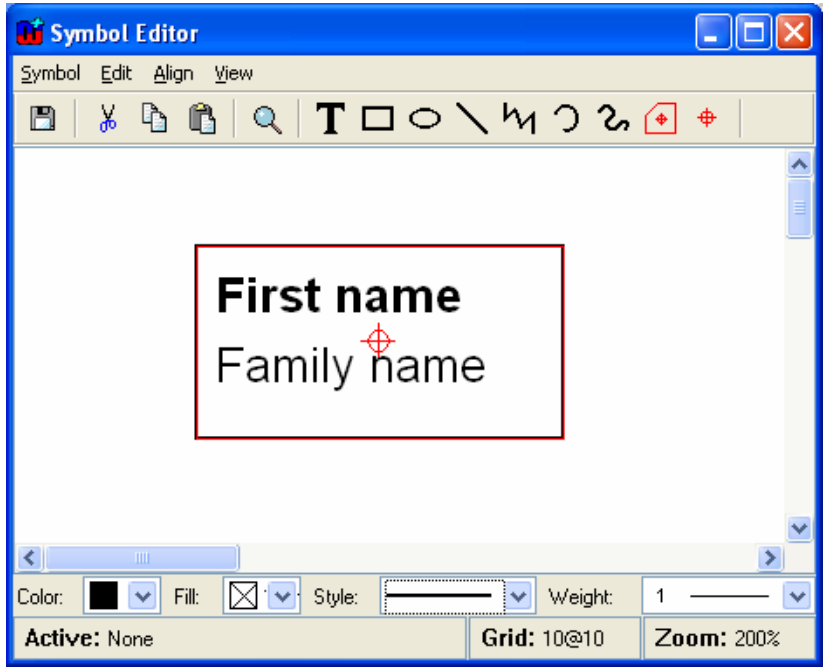
Domain Analysis Techniques
metaEdit+

- A Family Tree Language
  - There has to be a concept of *Person*
  - Each *Person* must have two other *Persons* as *Parents*
  - A *Person* can be a *Parent* to his or her *Children*
  - *Parents* and *Children* together form a *Family* relationship

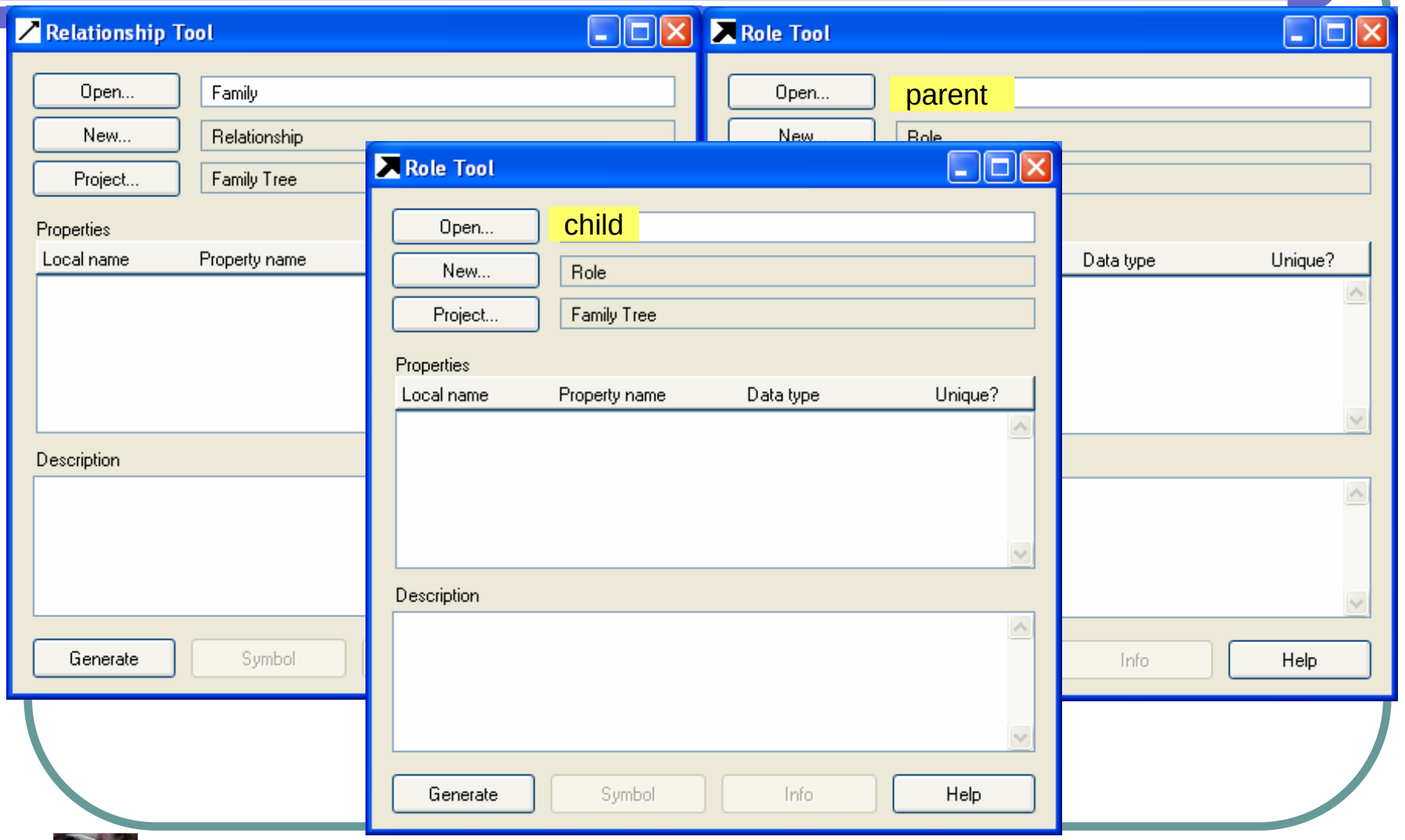


Domain Analysis Techniques
metaEdit+

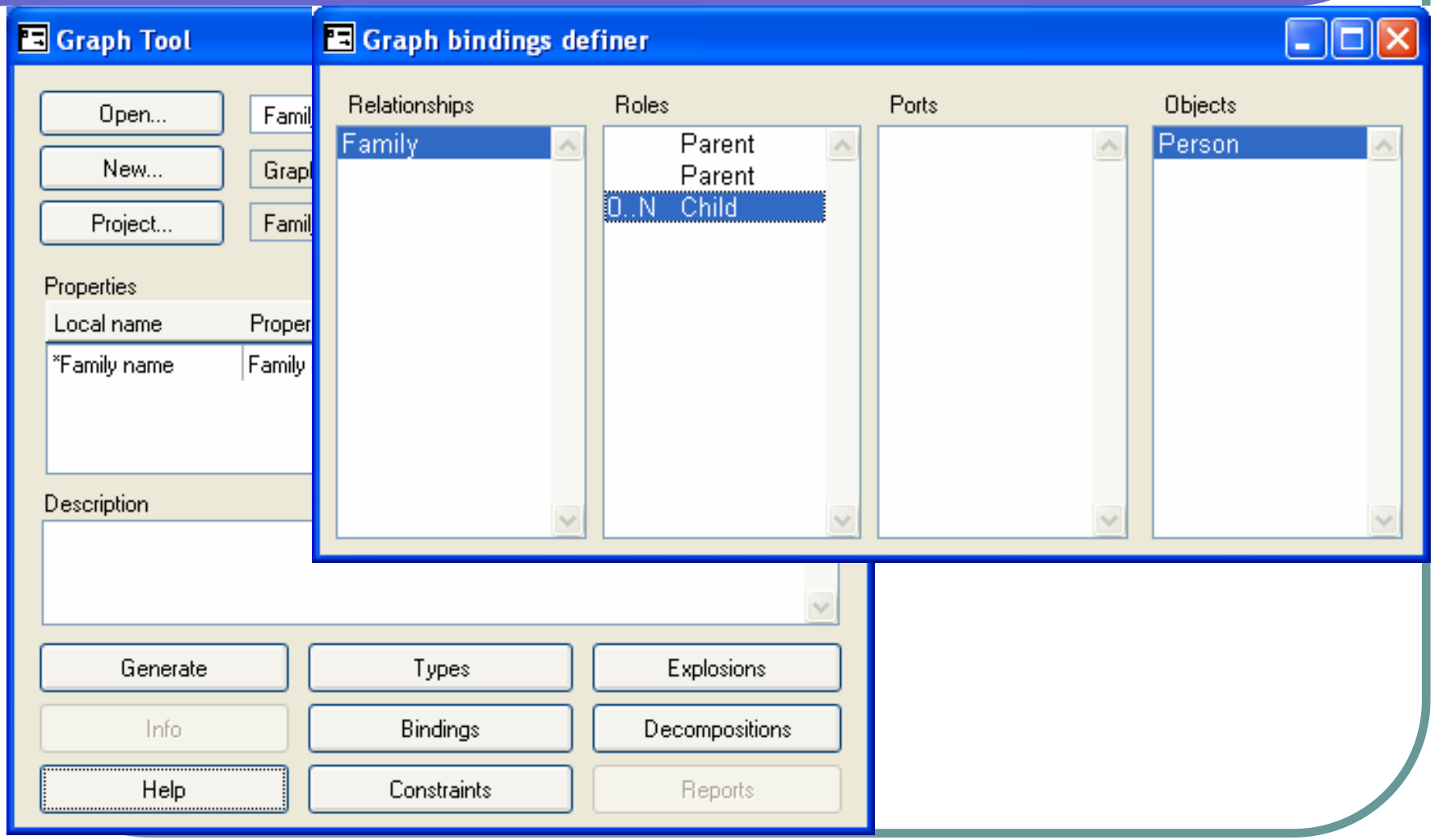




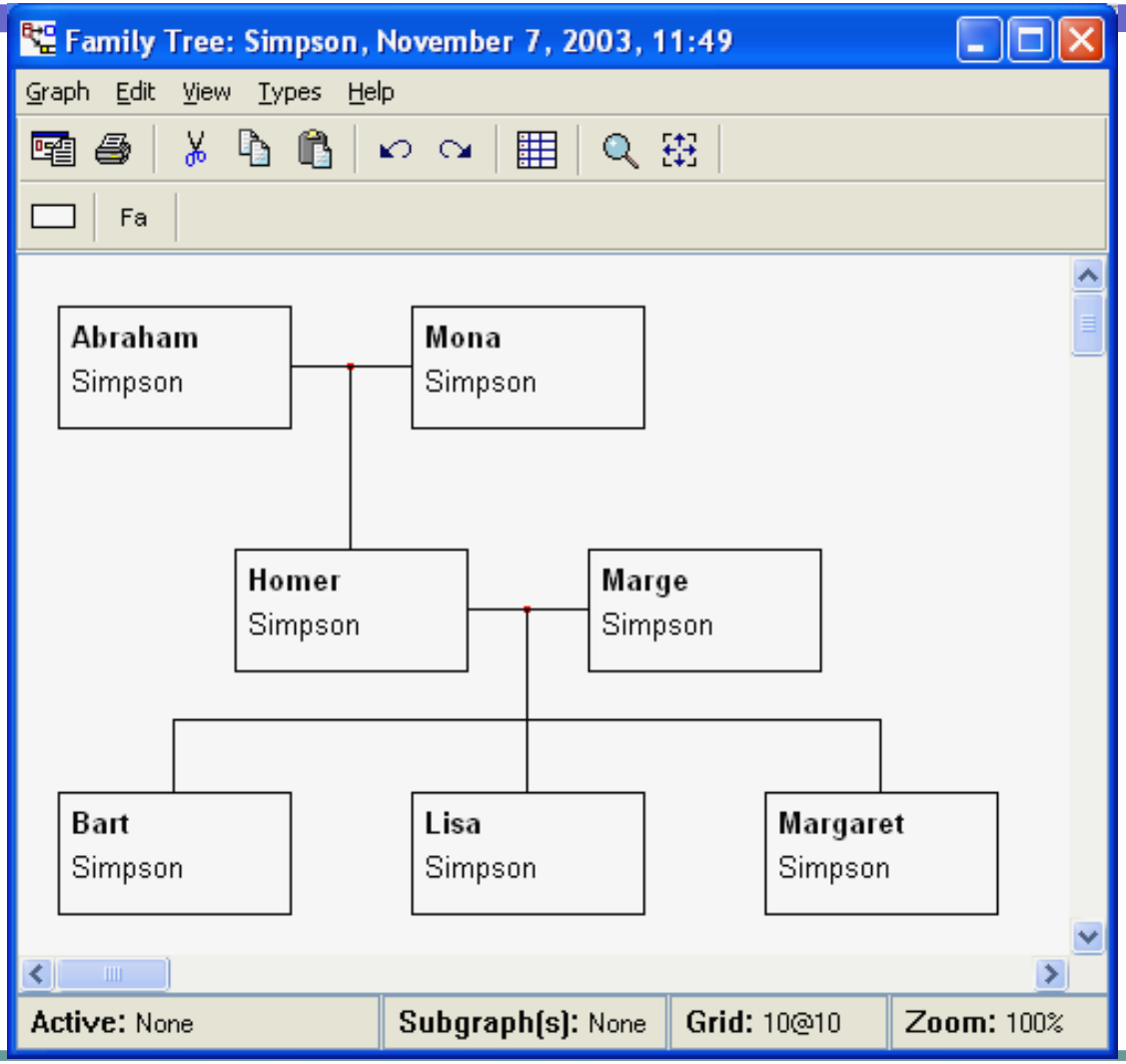
Domain Analysis Techniques
metaEdit+



Domain Analysis Techniques
metaEdit+



Domain Analysis Techniques
metaEdit+



Domain Analysis Techniques
References (1)

- Arango, G. “Domain analysis: from art form to engineering discipline”, Proceedings of the Fifth International Workshop on Software Specification and Design, p.152-159, 1989.
- Carnegie, M. “Domain Engineering: A Model-Based Approach”, Software Engineering Institute, http://www.sei.cmu.edu/domain-engineering/, 2002.
- Champeaux, D. de, Lea, D., and Faure, P. Object-Oriented System Development, Addison Wesley, 1993.
- Cleaveland, C. “Domain Engineering”, http://craigc.com/cs/de.html, 2002.
- Clauss, M. "Generic Modeling using UML extensions for variability", Workshop on Domain Specific Visual Languages, Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01), 2001.
- Davis, J. “Model Integrated Computing: A Framework for Creating Domain Specific Design Environments”, The Sixth World Multiconference on Systems, Cybernetics, and Informatics (SCI), 2002.
- Gomma, H., Designing Software Product Lines with UML, 2004.

Domain Analysis Techniques
References (2)

- Gomma, H. and Eonsuk-Shin, M. "Multiple-View Meta-Modeling of Software Product Lines", Proceedings of the Eighth IEEE International Confrerence on Engineering of Complex Computer Systems, 2002.
- Gomaa, E. and Kerschberg, L. "Domain Modeling for Software Reuse and Evolution", Proceedings of Computer Assisted Software Engineering Workshop (CASE 95), 1995.
- Harel, D. Statecharts: a Visual Formalism for Complex Systems. Science of Computer Programming 8: 231-274, 1987.
- Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A.,"Feature-Oriented Domain Analysis (FODA) Feasibility Study", CMU/SEI-90-TR-021 ADA235785, 1990.
- Morisio, M., Travassos, G. H., and Stark, M. "Extending UML to Support Domain Analysis", Proceedings of the Fifth IEEE International Conference on Automated Software Engineering, pp. 321-324, 2000.
- Nordstrom, G., Sztipanovits, J., Karsai, G., and Ledeczi, A. "Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments", Proceedings of the IEEE Sixth Symposium on Engineering Computer-Based Systems (ECBS), pp. 68-74, 1999.
- Petro, J. J., Peterson, A. S., and Ruby, W. F. "In-Transit Visibility Modernization Domain Modeling Report Comprehensive Approach to Reusable Defense Software" (STARS-VC-H002a/001/00).