Software Development Methodologies

Lecturer: **Raman Ramsin**

Lecture 5

Integrated Object-Oriented Methodologies: OPM and Catalysis

Object Process Methodology (OPM)

- Introduced by Dori in 1995
- Primarily intended as a novel approach to analysis modeling, combining the classic process-oriented modeling approach with object-oriented modeling techniques
- Has evolved into a full-lifecycle methodology
- Only one type of diagram is used for modeling the structure, function and behaviour of the system.
- Single-model approach avoids the problems of model multiplicity, but the model produced can be complex and hard to grasp.
- OPM process is little more than an abstract framework, and resembles the generic software development process.

OPM: Process

- Consists of three high-level subprocesses:
  - *Initiating:* preliminary analysis of the system, determining the scope of the system, the required resources, and the high-level requirements
  - *Developing:* with the focus on detailed analysis, design and implementation of the system
  - *Deploying:* Introduction of the system into the user environment, and subsequent maintenance activities performed during the operational life of the system

OPM: Initiating

- *Identifying:* the needs and/or opportunities justifying the development of the system are determined.
- *Conceiving:* the system is “conceived” through determining its scope and ensuring that the resources necessary for the development effort are available.
- *Initializing:* the high-level requirements of the system are determined.

OPM: Developing

- *Analyzing:* typically involves:
  - eliciting the requirements
  - modeling the problem domain and the system in Object Process Diagrams (OPD) and their Object Process Language (OPL) equivalents
  - selecting a skeletal architecture for the system
- *Designing:* typically involves:
  - adding implementation-specific details to the models
  - refining the architecture of the system by determining its hardware, middleware and software components
  - designing the software components by detailing the process logic, the database organization, and the user interface
- *Implementing:* constructing the components of the system and linking them together; typically involves:
  - coding and testing the software components
  - setting up the hardware architecture
  - installing the software platform (including the middleware)

OPM: Deploying

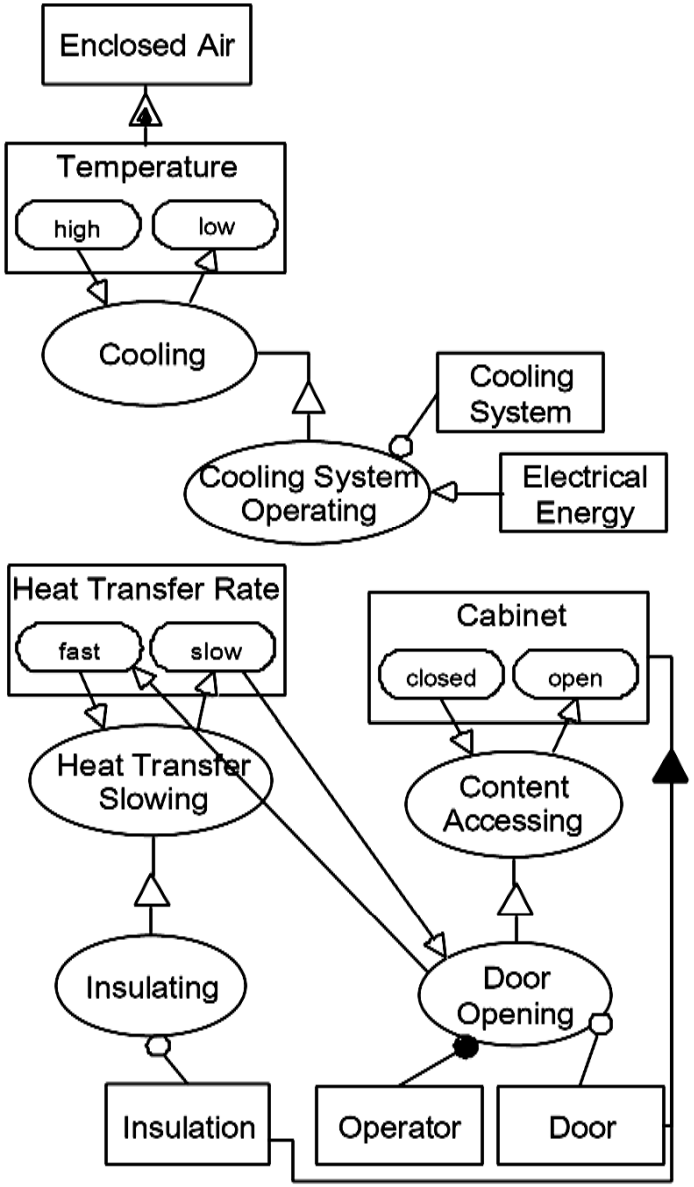
- *Assimilating:* introducing the implemented system into the user environment, mainly involving:
  - Training
  - generation of appropriate documents
  - data and system conversion
  - acceptance testing.
- *Using and Maintaining*
- *Evaluating Functionality:* [typically performed during the Using-and-Maintaining activity] checking that the current system possesses the functionality needed to satisfy the requirements
- *Terminating:*
  - declaring the current system as dead
  - applying the usual post-mortem procedures
  - prompting the generation of a new system

Object Process Diagram (OPD)

- Uses elements of types *object* and *process* to model the structural, functional and behavioural aspects
- Notation was later expanded to also include elements of type *state*, which were particularly useful in modeling real-time systems
- Every OPD can also be expressed in textual form, using a constrained natural language called the OPL (Object-Process Language)
- A set of OPDs is built for the system being developed, typically forming a hierarchy
- Multi-dimensional nature makes it difficult to focus on a particular aspect of the system without being distracted by other aspects.
- Some important behavioural aspects (such as object interactions, especially with regard to message sequencing) cannot be adequately captured

Object Process Diagram

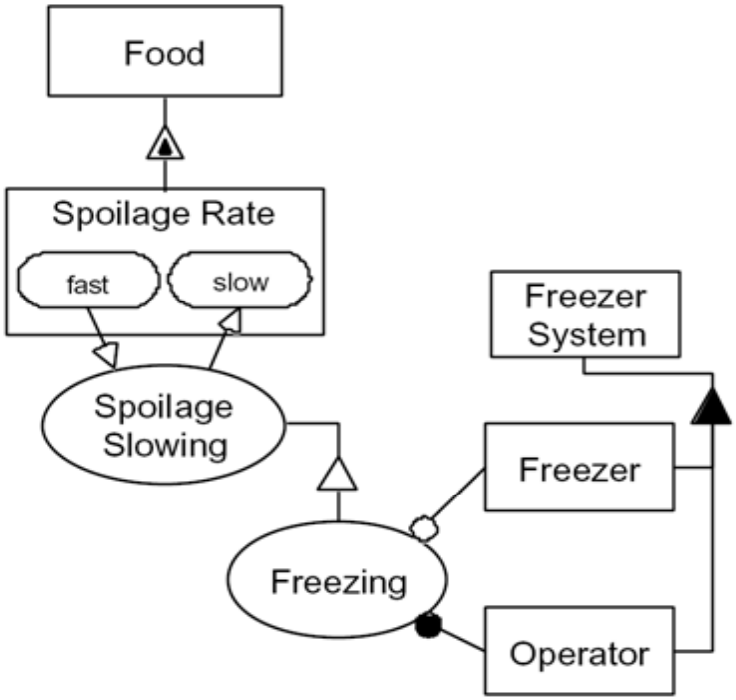
OPD



[Dori 2002]

Object Process Language

OPD and OPL



Food exhibits **Spoilage Rate,** which can be **fast** or **Slow.**

Spoilage Slowing changes **Spoilage Rate** from **fast** to **Slow.**

Freezing is **Spoilage Slowing.**

Freezing System consists of **Freezer** and **Operator**.

Freezing requires **Freezer**.

Operator handles **Freezing**.

[Dori 2002]

OPM: Strengths and Weaknesses

- ***Strengths***
  - Simplicity of process
  - Some degree of seamless development and traceability to requirements due to the singularity of the model type used (disrupted, though, because of OPD's limited modeling capacity)
  - Innovative structural and functional modeling in a single type of diagram (OPD)
  - Strong structural modeling at the inter-object level

OPM: Strengths and Weaknesses

- ***Weaknesses***
  - Process is defined at a shallow level, with ambiguities and inadequate attention to detail
  - Seamlessness and traceability are disrupted due to lack of behavioural models (especially at the inter-object and intra-object levels, directly affecting the identification and design of class operations)
  - No basis in system-level behaviour and usage scenarios
  - Poor behavioural modeling
  - No formalism
  - Poor intra-object structural modeling
  - Models are prone to over-complexity
  - No modeling of physical configuration

Catalysis

- Introduced by D'Souza and Wills in 1995, originally as a component-based formalization of OMT deeply influenced by Fusion, Objectory, Booch and Syntropy
- A UML-based, refined version of the methodology appeared in 1998
- Based on *fractal modeling* and *gradual refinement*
- Proposes a specific process for developing business systems
- Proposes a set of process patterns to be selected and applied according to the characteristics of the project in hand

Catalysis: Business Systems Development Process

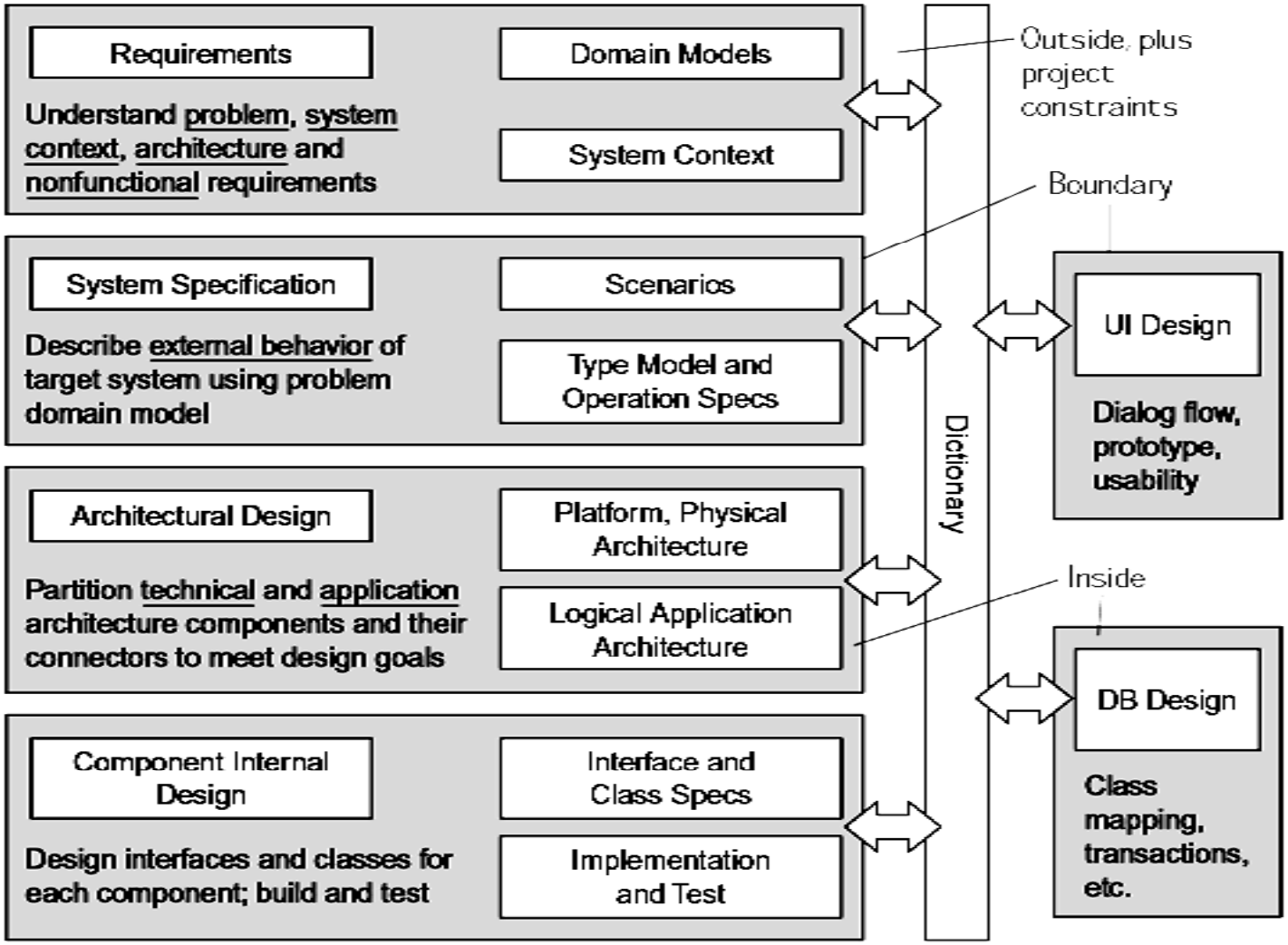
1. *Identify and Model the Requirements:* exploration and modeling of the problem domain and the requirements of the system

2. *Develop the System Specification:* identifying and modeling the functionality and high-level class-structure of the system. Designing the User Interface (UI) usually overlaps with this activity.

3. *Develop the Architectural Design:*
    1. designing the internal component (logical) architecture of the system
    2. designing the technical (physical) architecture defining the domain-independent parts of the system, such as the hardware and software platforms
    3. designing of the database architecture should also start at this stage

4. *Develop the Component Internal Design:* designing the internal detail of the components, which are then implemented and tested

Catalysis: Business Systems Development Process



[D'Souza and Wills1998]

Catalysis: Business Systems Development Process

1. Identify and Model the Requirements

- *Explore the problem domain and construct the Business Model,* which contains:
  - *class diagrams* depicting the *object-types* (analogous to classes) in the problem domain
  - special *collaboration diagrams* showing the *actions* that problem domain objects perform
  - *sequence diagrams* showing the sequence of the actions
  - a *glossary*, listing the terms used to define the problem domain
- *Identify and model the functional requirements of the system:* using a *System Context Diagram* showing the system as an object in the problem domain interacting with other objects. Actions on the system are use cases, and scenarios of interaction are expressed by sequence diagrams
- *Identify non-functional requirements:* e.g. performance, reliability, and reuse
- *Identify and model the known platform or architectural constraints:* machines, operating systems, middleware, legacy systems, and interoperability requirements are identified and modeled as package diagrams. Interactions are captured in collaboration diagrams and sequence diagrams
- *Identify the project and planning constraints*

Catalysis: Business Systems Development Process

2. Develop the System Specification

- The *system specification* mainly consists of:
  - A class (*type*) diagram showing the system as a type, emphasizing its attributes (internal types) and its associations with other types in the problem domain
  - A set of operations, depicting the actions that the system performs (functionality), usually captured in statecharts

Catalysis: Business Systems Development Process

3. Develop the Architectural Design

- *Identify the components comprising the system and their architecture:*
  - Component (*Application*) Architecture is usually described with package diagrams.
  - Specification types (system attributes) identified during the previous activity are split across different components.
  - Interaction among components is modeled through collaboration diagrams.

- *Identify the architecture of the domain-independent parts of the system:* modeled in the *Technical Architecture.* using package diagrams and collaboration diagrams. Components include:
  - hardware and software platforms
  - infrastructure components (such as middleware and databases)
  - utilities for logging/exception-handling/start-up/shutdown
  - design standards and tools
  - the choice of component architecture (such as JavaBeans or COM)

Catalysis: Business Systems Development Process

4. Develop the Component Internal Design

- Each and every component is designed, implemented and tested.
  - Design is done by identifying the programming language interfaces and classes, or pre-existing components, that constitute the component.
  - The architecture of these parts inside each component is modeled using a package diagram.
  - Interactions are shown by sequence and collaboration diagrams.

Catalysis: Process Patterns

- *Object Development from Scratch:* for when there is no existing system
- *Reengineering:* for when the objective is to improve an existing system
- *Business Process Improvement:* for applying object technology to organizations and systems other than software
- *Separate Middleware from Business Components:* for handling legacy systems as well as for insulating a system from certain changes in technology

Catalysis: Strengths and Weaknesses

- ***Strengths***
  - Based on requirements identified and modeled as system functionality and behaviour in the context of the problem domain: the system is modeled as a class – *type* – among other classes in the problem domain
  - Seamless development through uniform approach to modeling at different levels
  - Traceability to requirements via usage scenarios and use-case-based testing
  - Gradual refinement from problem domain to the system boundary, then to the component architecture of the system, and finally to the class architecture of the components

Catalysis: Strengths and Weaknesses

- ***Strengths*** **(Contd.)**
  - Process patterns identified for different kinds of projects
  - Special attention to non-functional requirements
  - Adequate complexity management
  - Special attention to physical configuration of the system early in the process
  - Smooth transition from logical to physical aspects
  - Component based approach
  - Fractal modeling
  - Rich structural and behavioural modeling at all levels. Functional modeling limited to UML's capabilities

Catalysis: Strengths and Weaknesses

- ***Weaknesses***
  - Heavy process; fractal modeling and process patterns help, but are not enough
  - Focus mostly confined to business systems, more or less limiting the applicability of the process

References

- Dori, D., *Object-Process Methodology: A Holistic Systems Paradigm*. Springer, 2002.
- D'Souza, D. F., and Wills, A. C., *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 1998.