MDS System Model

Cliff Kettemborough

Agenda

- Overview: What is a System Model
- Walkthrough of the system model structure
- Demo of the MDS system model

Overview: What is a System Model

System Model Definition

"[A system model] is an organized, internally consistent set of abstractions that collaborate to achieve system description at a desired level of detail and maturity." — Bruce Douglass

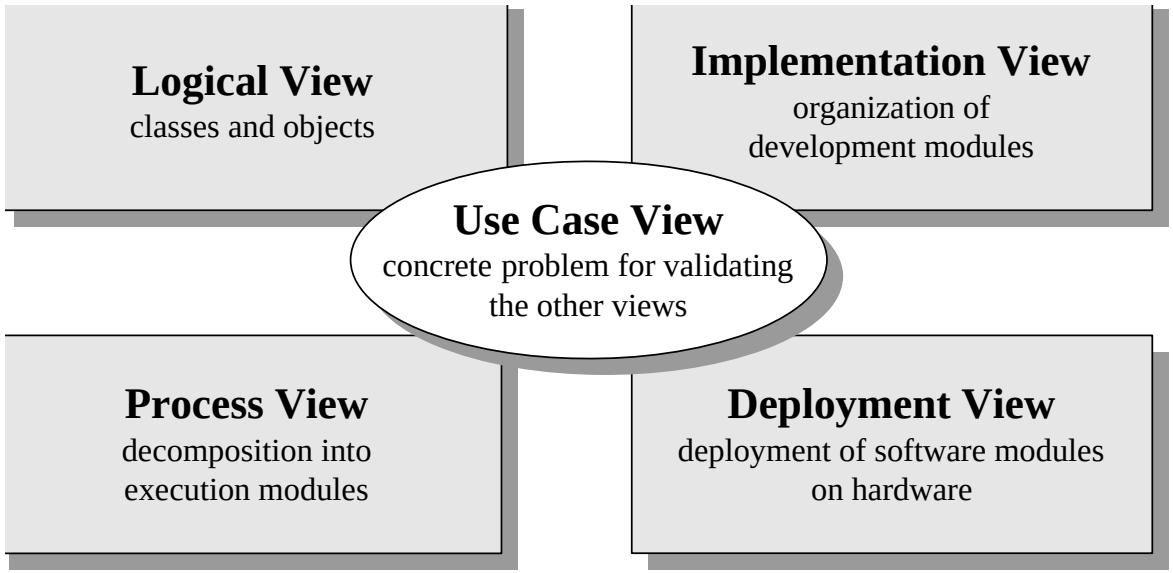
- This "internally consistent" model includes:
  - Functional requirements
  - Quality of service requirements
  - Logical organization
  - Runtime organization
  - Deployment
  - Physical organization of source
  - Components
  - Design patterns
  - Test scenarios

"It is important to understand that the analysis model, design model, translation (source code) model, and testing model are not different models that are somehow linked. They are different views of the same system model" — Bruce Douglass

- The system model is perceived from multiple views.

Krutchen "4+1 View" Model

Five views of a system model intended to capture the system architecture



Logical view	classes, objects, collaborations, interactions
Implementation view	modules, subroutines, subsystems, packages
Process view	tasks, threads, processes
Deployment view	response time, bandwidth, geographic constraints, power requirements, resistance to failures for each node, module or program
Use case view	actors, use cases, classes, collaborations

Describing the System Model in UML

- UML offers a set of diagrammatic and notational conventions for specifying, visualizing, constructing and documenting the system model
- UML includes 9 diagrams for different views of a system model
  - Defines a common meta-model that is independent of a particular programming language
  - Facilitates an Object-oriented Analysis and Design (OOAD) process
  - Encourages use-case driven, architecture-centric, approach that encourages an iterative and incremental development process
- UML is gaining industry support
  - Products from iLogix, Rational, Artisan, Visual Object Modelers, Object International, etc.
  - UML training and reference materials are widely available
  - UML model interchange is under development by OMG

The 9 UML Diagrams

Each diagram captures a different view of the model

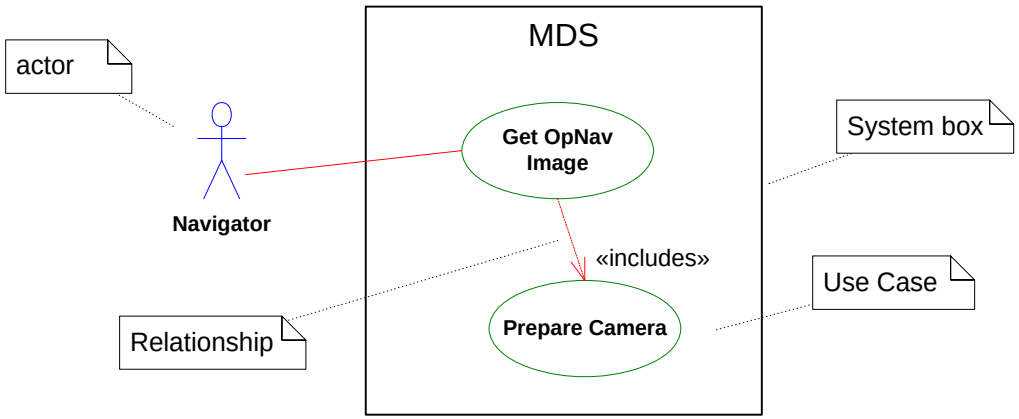
Use Case	expresses user requirements and capabilities required from the model
Sequence	captures message flows in a scenario
Collaboration	captures the message flows between collaborating objects in a scenario
Statechart	captures the dynamic behavior in a scenario
Activity	captures the concurrent activities in a scenario
Class	expresses the static structure of the model
Object-Model	expresses relationship between instantiated classes in model
Component	captures the work units and development dependencies in the model
Deployment	captures process allocation of the model

Sample diagrams can be viewed in the 2/19/99 Process Peer Review Package
http://mds.jpl.nasa.gov/teams/ProcessEngineering/ProcessPeerReview/process6.ppt

Walkthrough of the system model structure

Use Case View of Requirements

"*A use case is a system function that returns an observable result to an actor without revealing internal system structure*" — Bruce Douglass



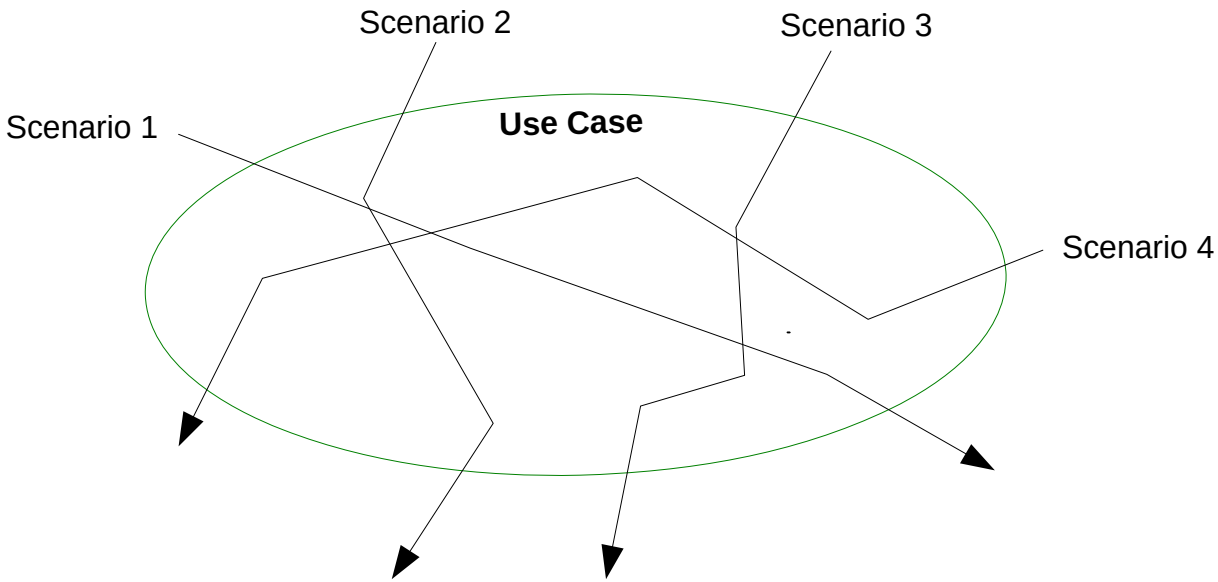
Intent of a Use case**:**

- **Describe how a user interacts with the system.**
- **Express requirements in ordinary language.**
- **Unify a set of behaviors under a single idea.**

A use case does not describe implementation in any form.

Scenarios Instantiate Use Cases

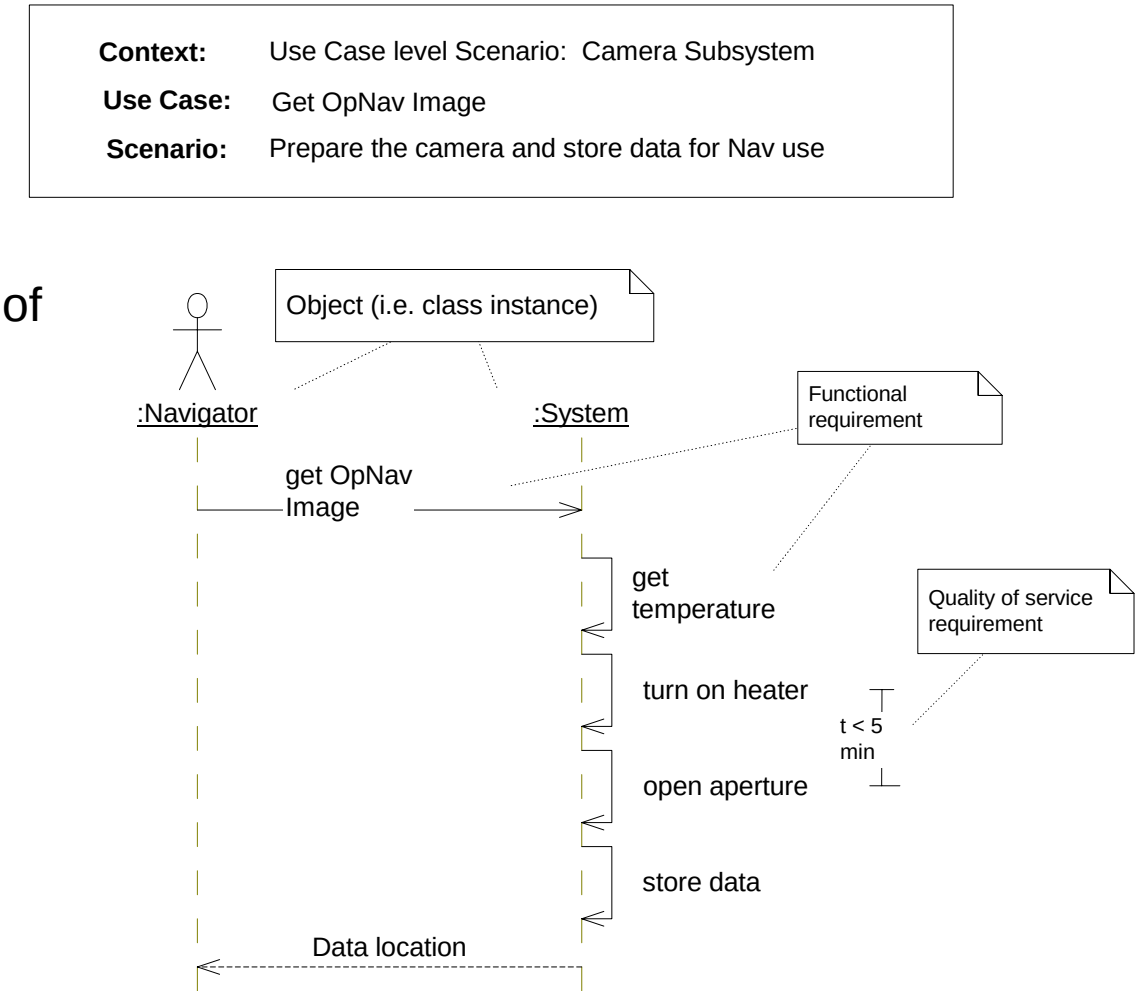
A scenario is a particular path through the abstract and general description provided by the use case. — Pierre-Alain Muller



A **use case** describes an abstract function of the system.
A **scenario** describes a specific implementation of a use case.

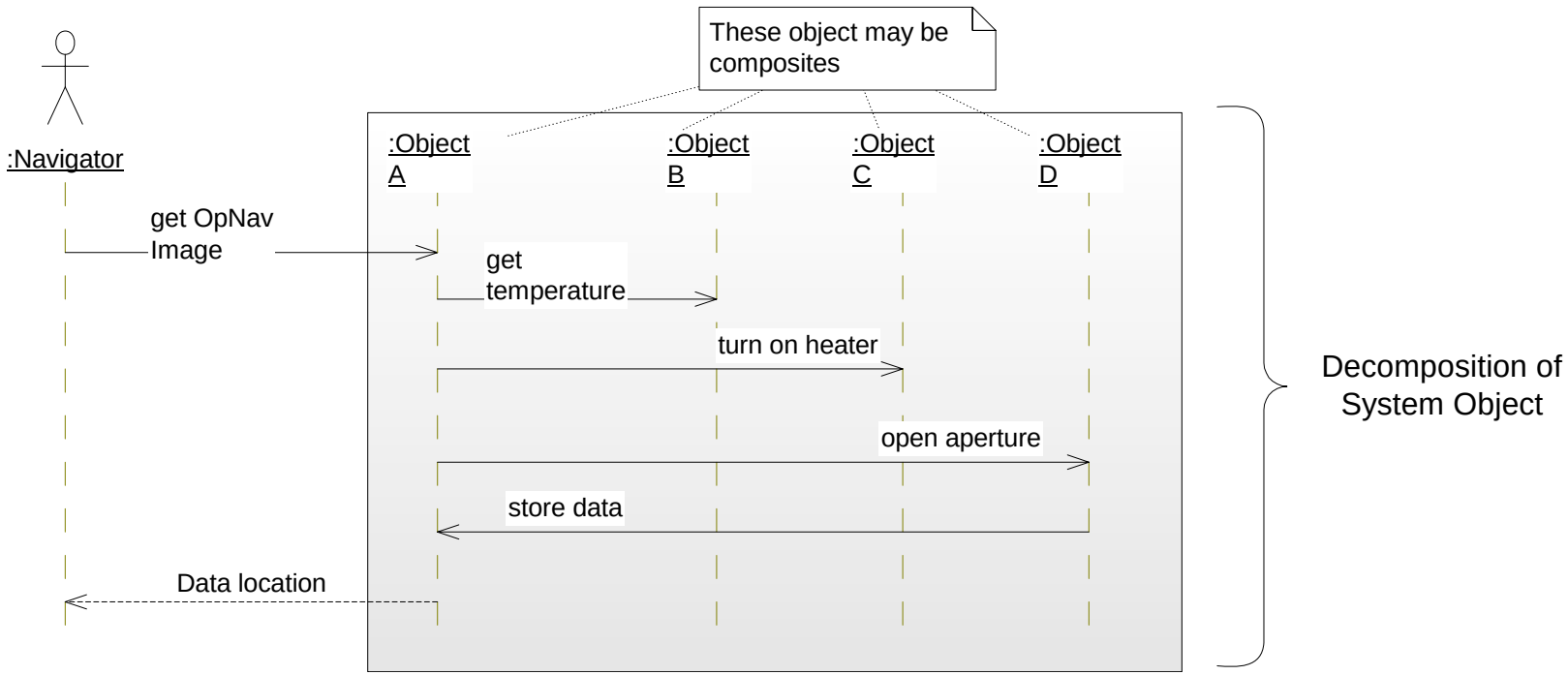
Use-Case-Level Scenario

- Purpose
  - Used to capture functional requirements
  - Used to capture quality of service requirements
- Defines system object that represents a composite of collaborating classes.
- Defines a high-level interface required by clients



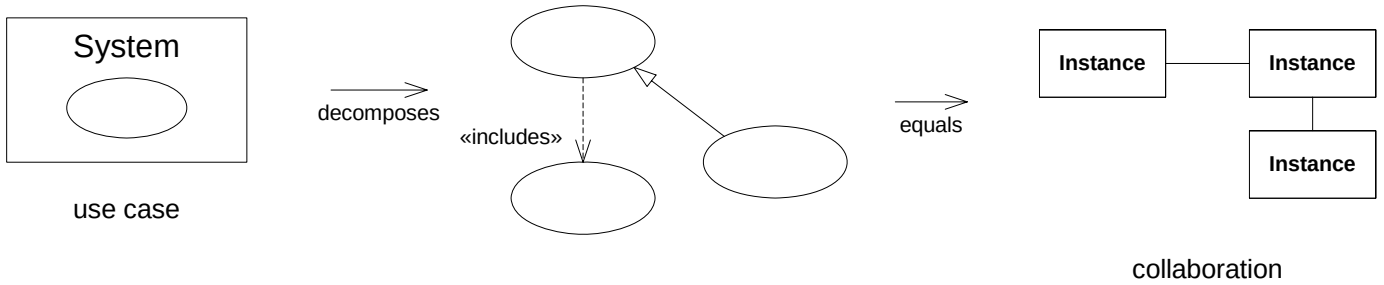
What's Inside the System Object?

- The system object can be decomposed into a collaboration of class instances.
  - Classes are selected from the logical domains
  - Each instance may also be a collaboration

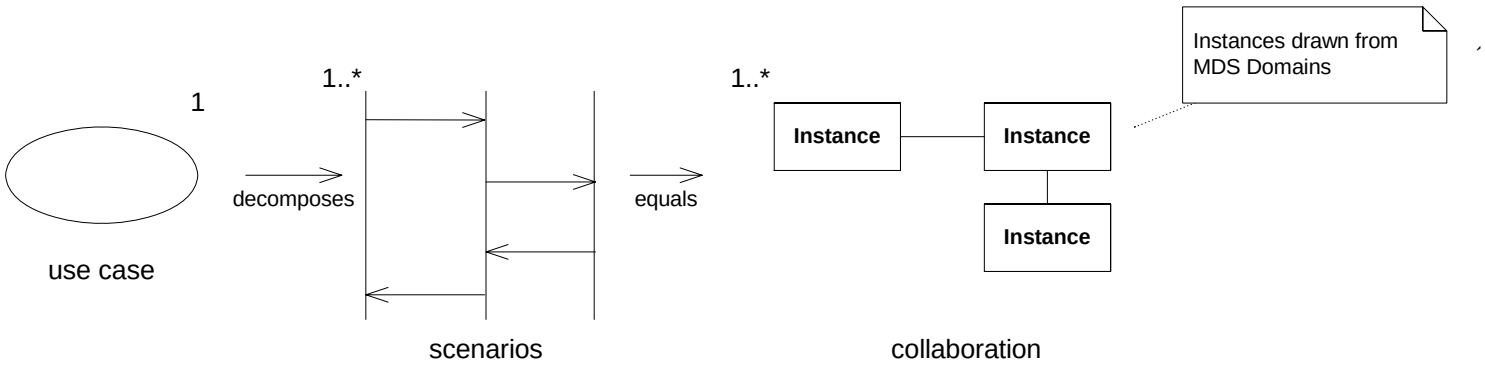


Behavioral Relationships

A system can be decomposed into use cases and/or collaboration.



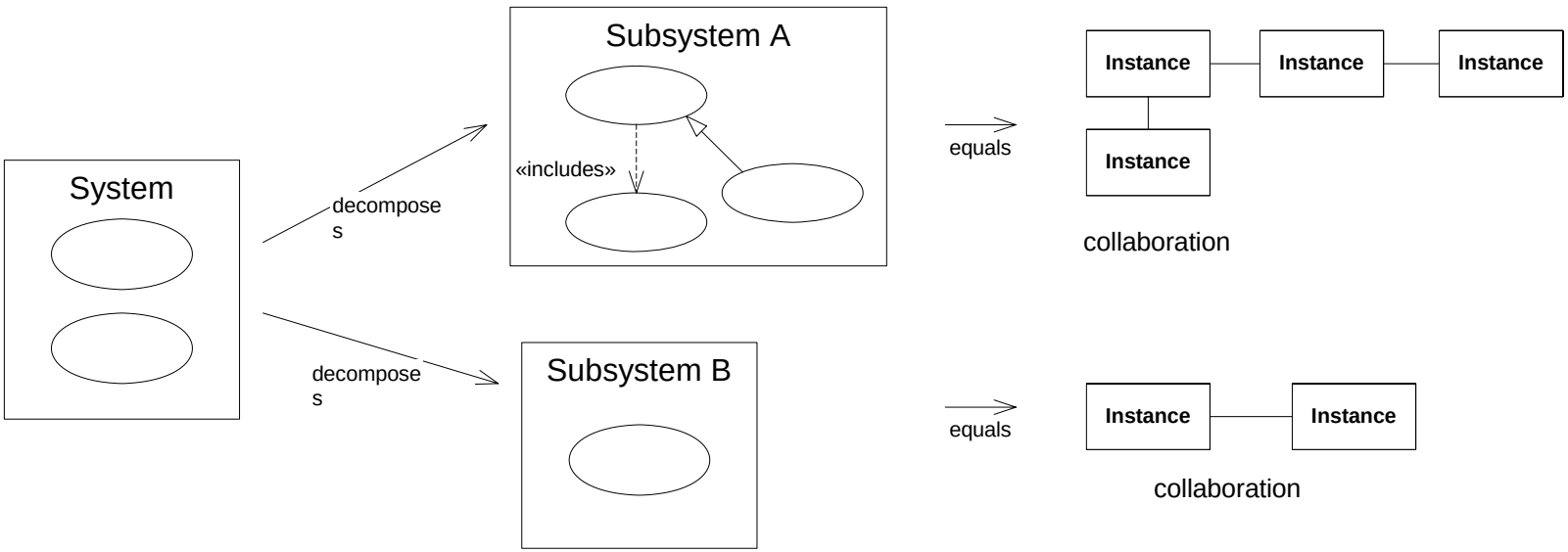
A use case can be decomposed into a collaboration.



continued

Behavioral Relationships *(cont'd)*

A system can be decomposed into subsystems and subsystems can be decomposed into collaborations.

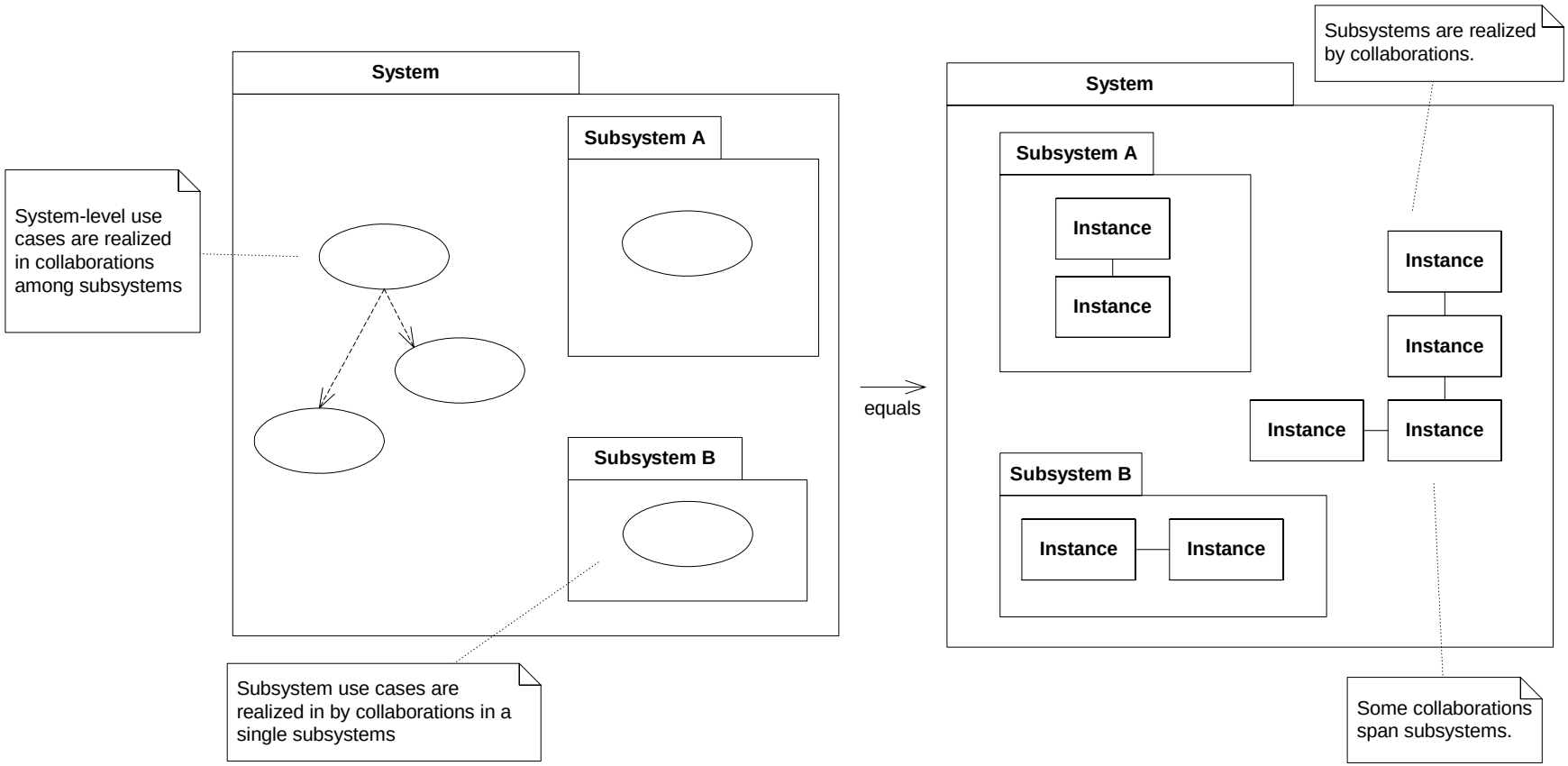


Domains and Subsystems

- System-level use cases are used to partition the system with “system analysis” and “object analysis.”
  - System analysis partitions the system behaviorally
    - Leads to an instantiated view that shows how the system will produce a required behavior, i.e. a functional requirement.
    - Partitions the system into large-scale functional units
    - Artifacts: subsystems and architectural patterns
  - Object analysis partitions the system logically
    - Leads to a static view that shows how the system is divided into areas of responsibility.
    - Partitions system in class hierarchies
    - Artifacts: domains, frameworks, class hierarchies
- System analysis and object analysis are interdependent.
  - Behavior derives from an instantiated a logical elements.
  - Logical elements are developed to meet a behavioral need.

Example Subsytem View

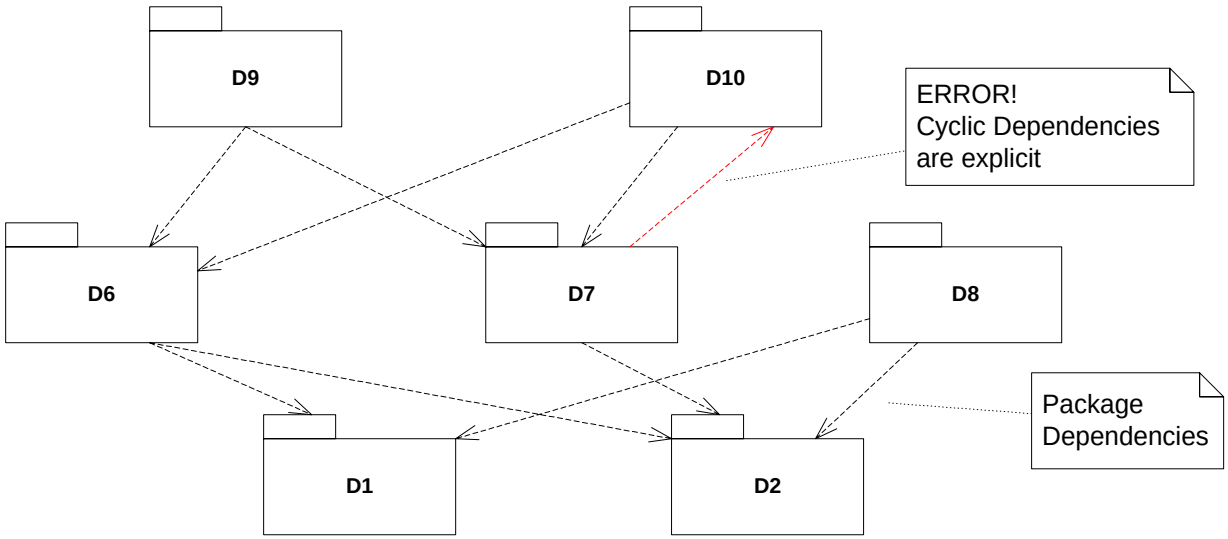
Since use cases are realized by collaborating objects, a collaboration is an alternative view of a use case.



Note: Use case diagrams use system boxes that imply no structure. Subsystem views use packages to reveal behavioral structures.

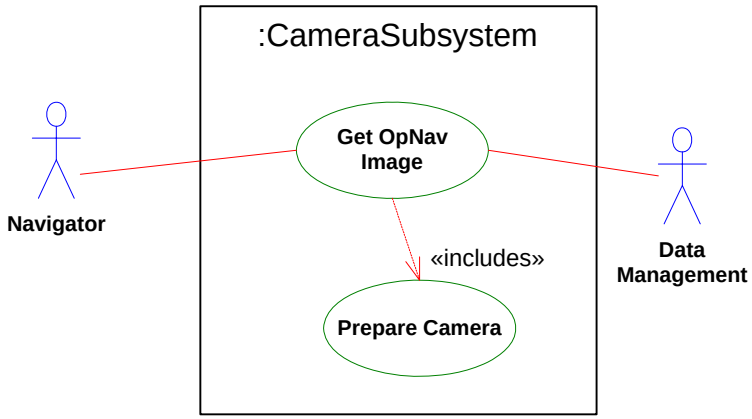
Example Domain View

- The framework is partitioned by subject into packages. Each subject package is a domain.
- Domains unite the logical and physical partitioning of the code.
  - Class hierarchies divide along domain boundaries
  - Code is physically organized by domains
  - Structural dependencies are exposed in the domain view



Decomposing a Subsystem Use Case

- Re-examine the first example in a subsystem context

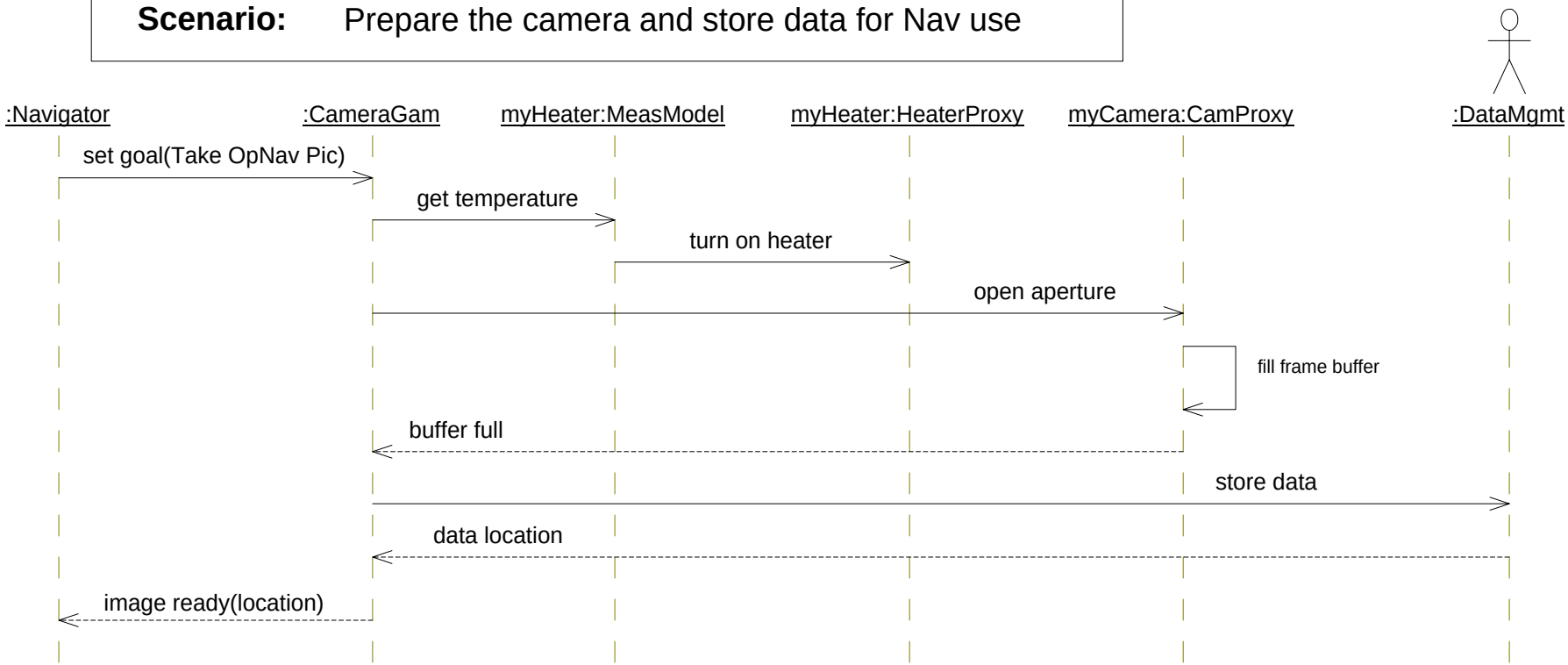


- In the initial use case diagram, data management was part of the system (i.e. mds). In this context, data management is outside the system.
- Sample Scenario: Prepare the camera and store the data for Nav use
  - Scenario includes the functionality called out in the “prepare camera” use case.
  - Happy day scenario. Other scenarios would capture more complex behavior.

Decomposition into Collaboration

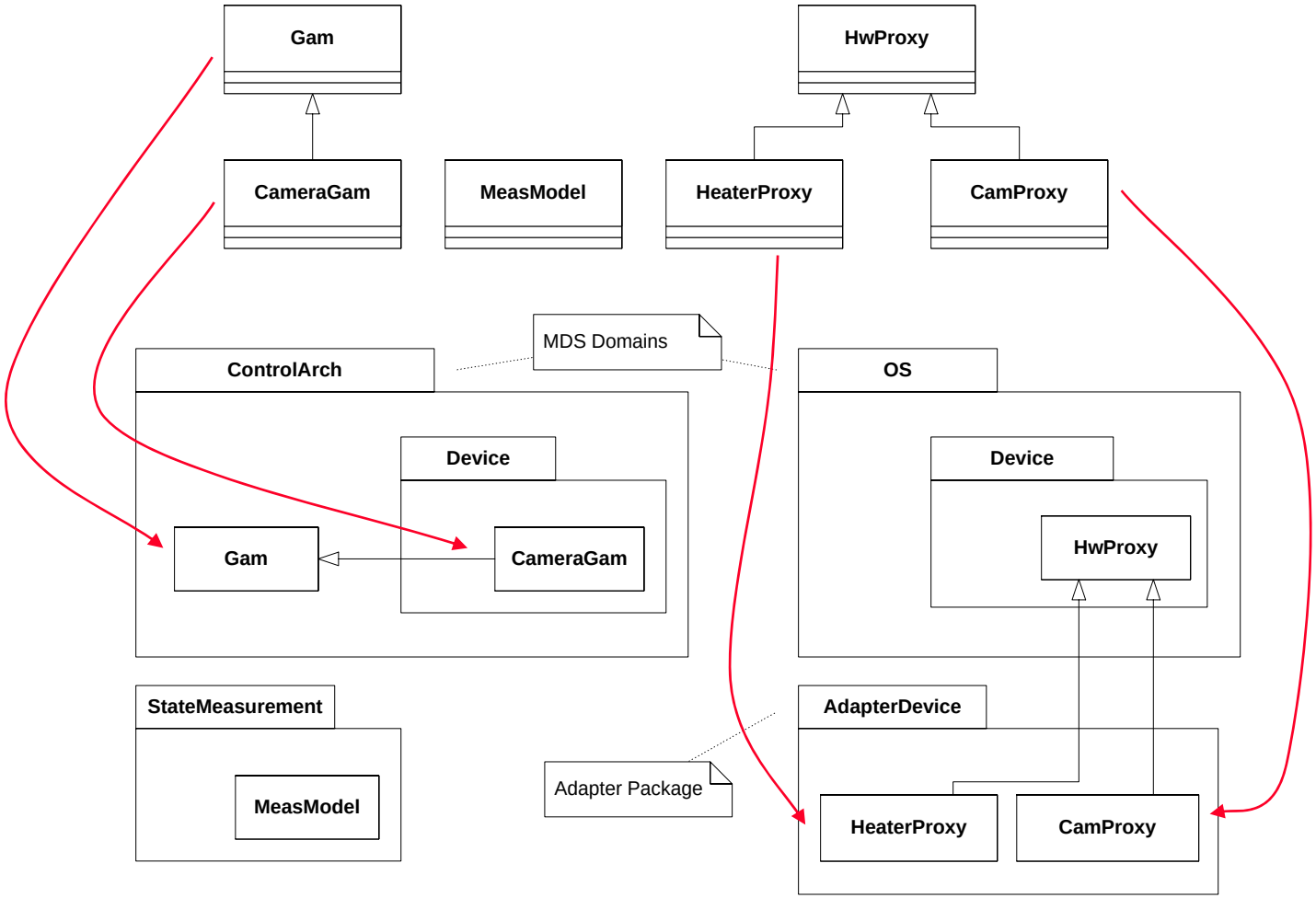
Example: Object analysis of scenario from use case in camera subsystem

Context:	Object Analysis: Camera Subsystem
Use Case:	Get OpNav Image
Scenario:	Prepare the camera and store data for Nav use



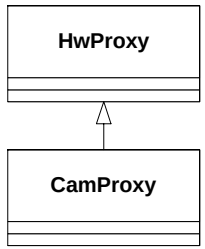
Logical Decomposition

The underlying framework of classes is decomposed into domain hierarchies. The frameworks are *not* decomposed by function but by responsibility.

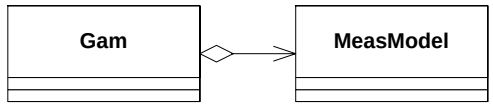


Adapting the model

Logical adaptation

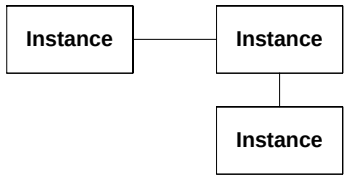


Inheritance

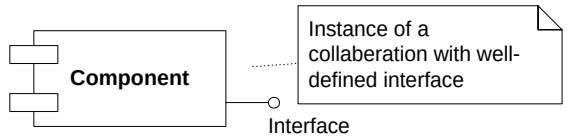


aggregation

Behavioral adaptation

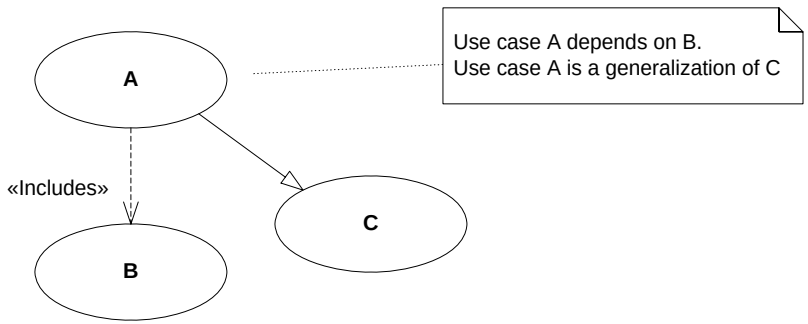


Design Patterns



Component Architecture

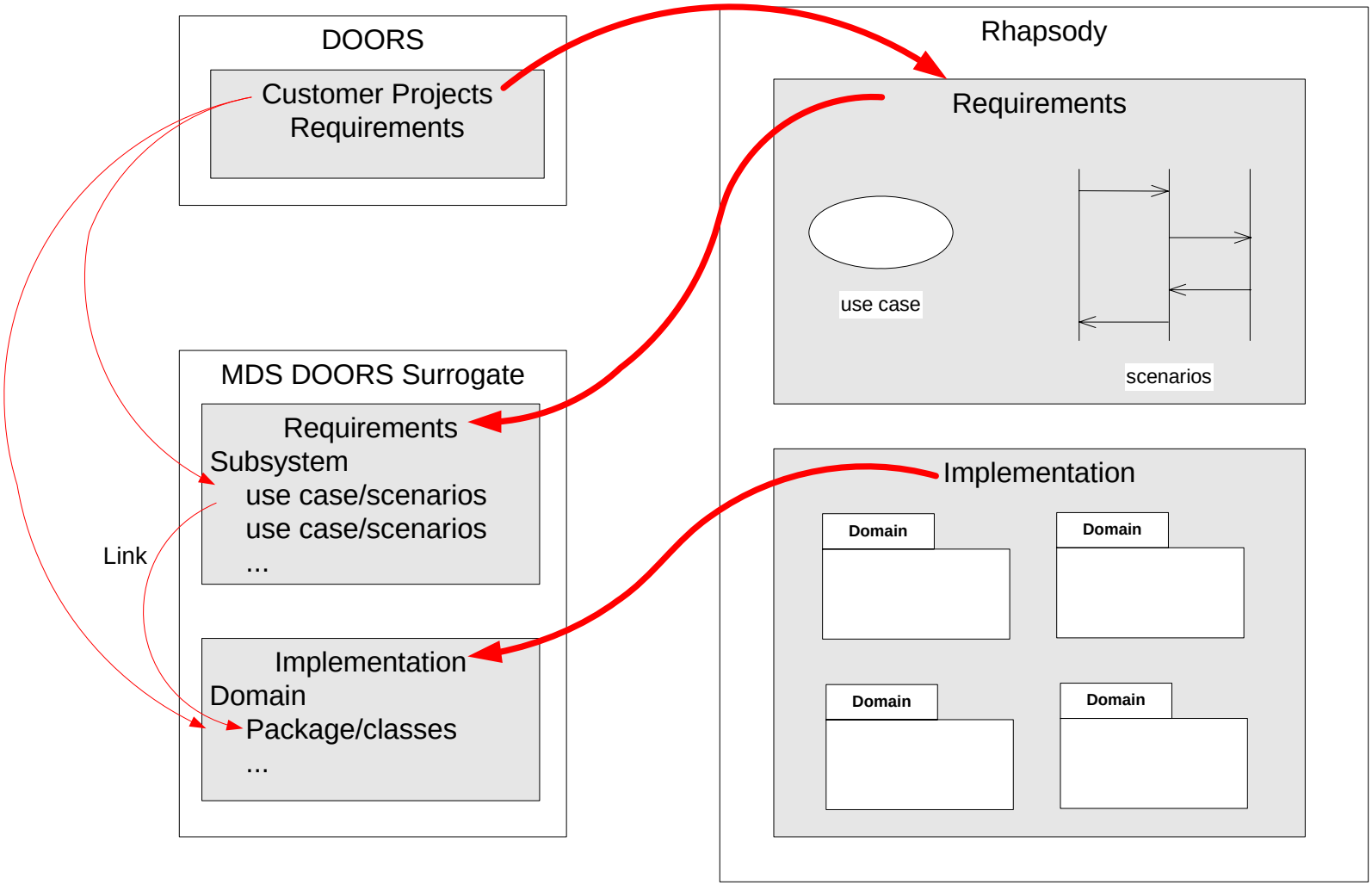
Requirement adaptation



Use Case reuse

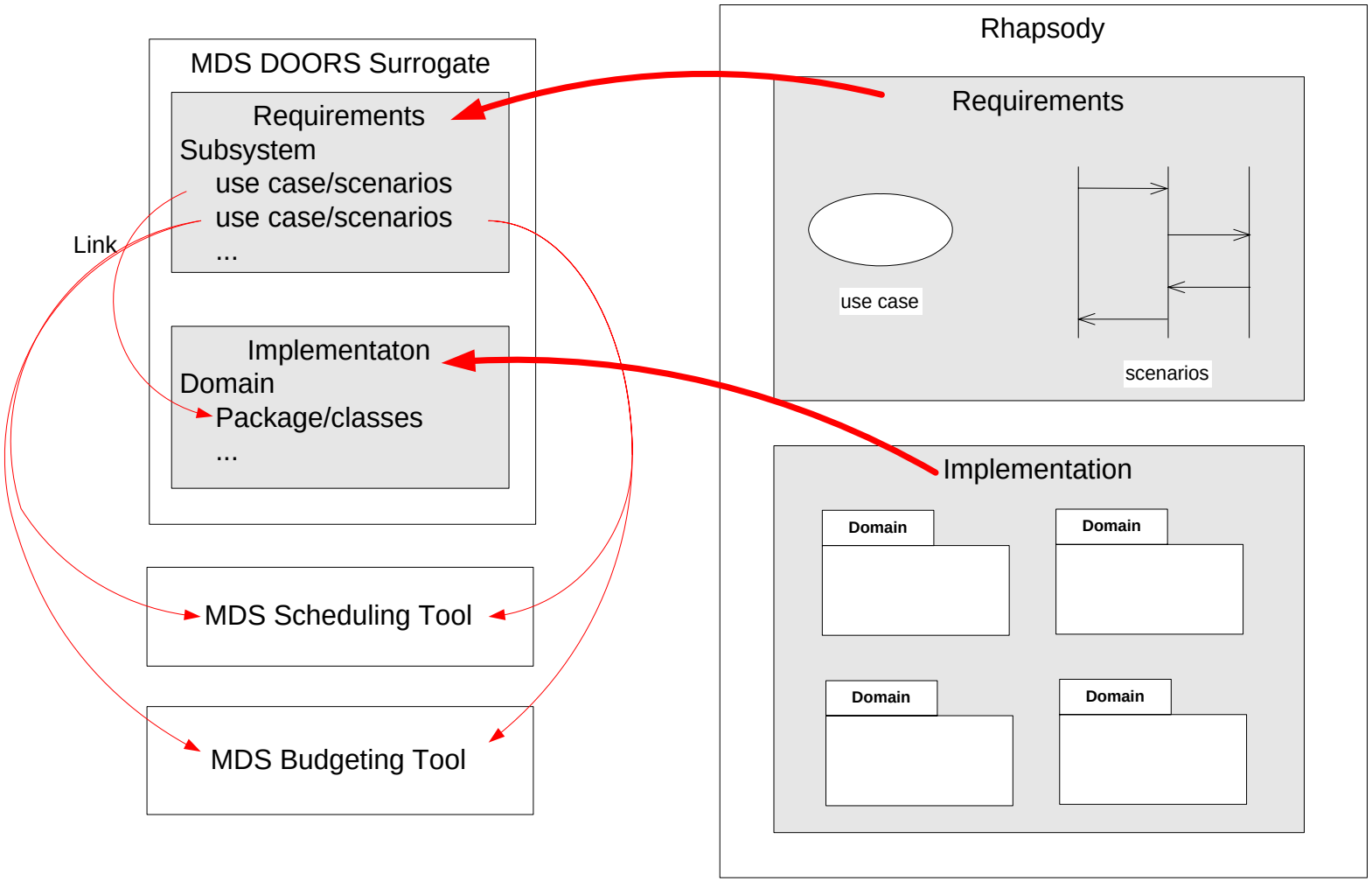
Rhapsody Doors Interface

Customers can use DOORS to map to MDS requirements and implementation



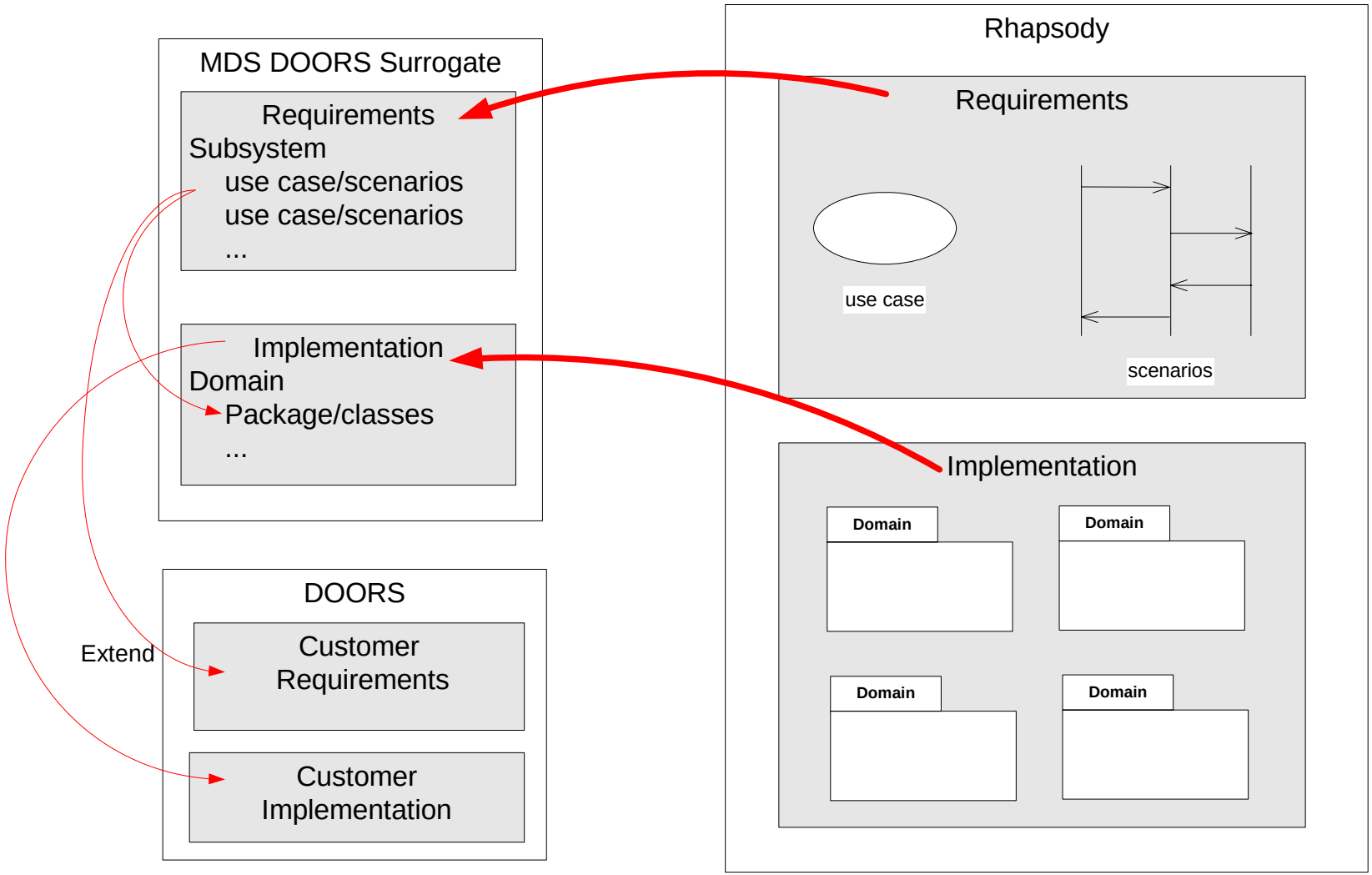
Rhapsody Doors Interface *(cont'd)*

MDS will use DOORS to map implementation to budget and schedule



Rhapsody Doors Interface *(cont'd)*

Future customers can use DOORS or Rhapsody to extend MDS requirements and implementation



Summary

- The system model is divided into views.
- Use cases and scenarios capture requirements.
  - Use cases capture black-box functionality
  - Scenarios capture a specific instantiation of a use case
  - Collaborations of objects satisfy requirements
- The various views reflect different logical and behavioral partitions of the system.
  - Domains reflect a static logical view of the system
  - Subsystems reflect a dynamic (run-time) behavioral view of the system
- The model can be adapted to develop many systems