Presentation for OMG Conference Florida, May 2003

Advanced PIM Development

Petter Graff - Vice President
Vladimir Bacvanski - Vice President
InferData Ltd.
www.inferdata.com
pgraff@inferdata.com
vbacvanski@inferdata.com

Software Development Issues

Covered in this section:

- Some of the main problems and goals of software development
- MDA from 30.000 feet

The World is Changing

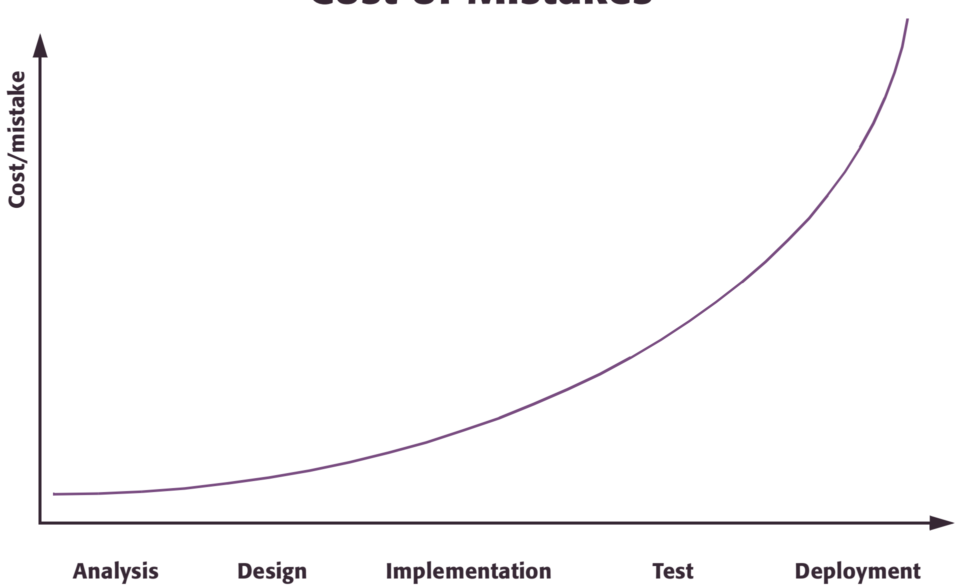
"I think there is a world market for maybe five computers."
Thomas Watson, chairman of IBM, 1943

"There is no reason anyone would want a computer in their home."
Ken Olson, president/founder of Digital Equipment Corp., 1977

"640K ought to be enough for anybody."
Bill Gates, 1981

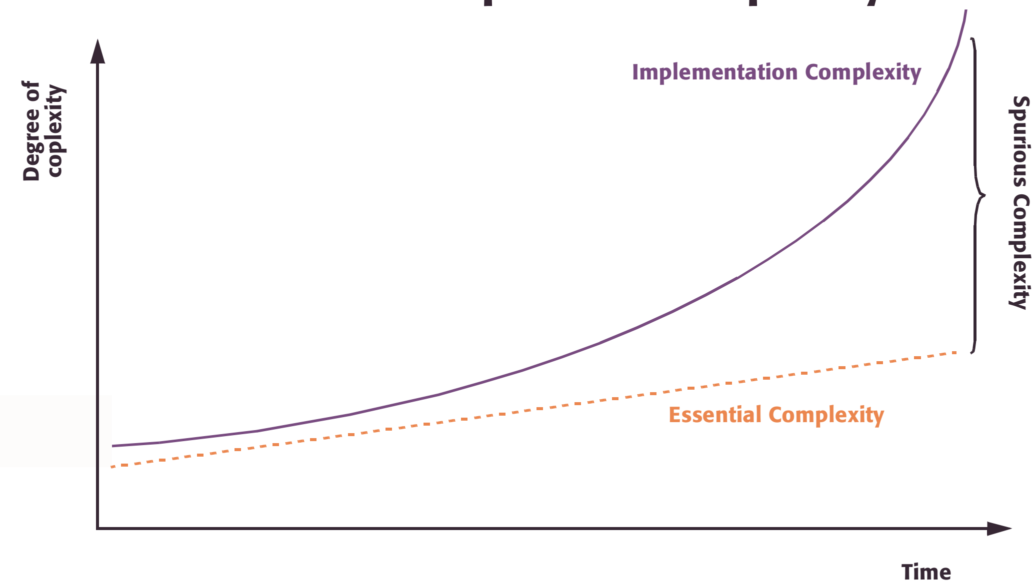
- We are building more and more complex systems
- We need more sophisticated tools and methods to handle the increased complexity

Cost of Mistakes



- We need to identify issues as early as possible

Essential vs. Spurious Complexity



- Over time, complexity of implementations grow exponentially
- However, the requested complexity typically grows linearly

The Object-Oriented Advantage: Continuity

Problem Domain:

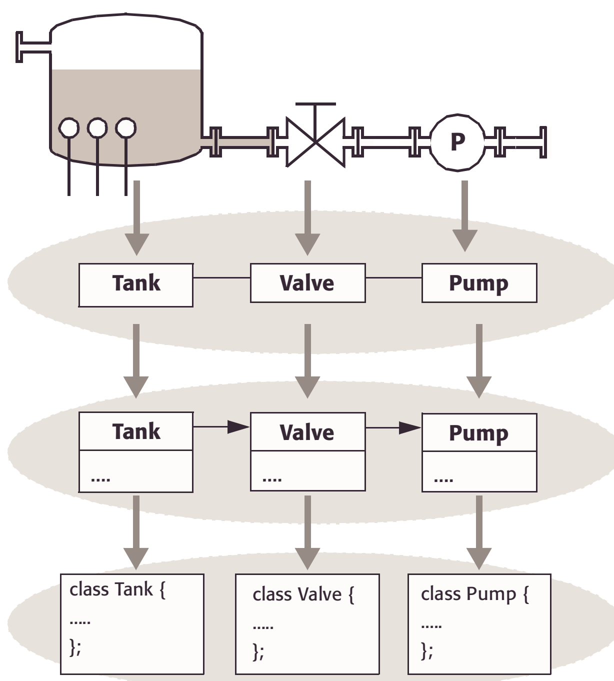
Example:
Tank Control System

System Analysis:
What are we going to build?

System Design:

How are we going to build it?

System Implementation



The Promise of Object-Oriented Technology

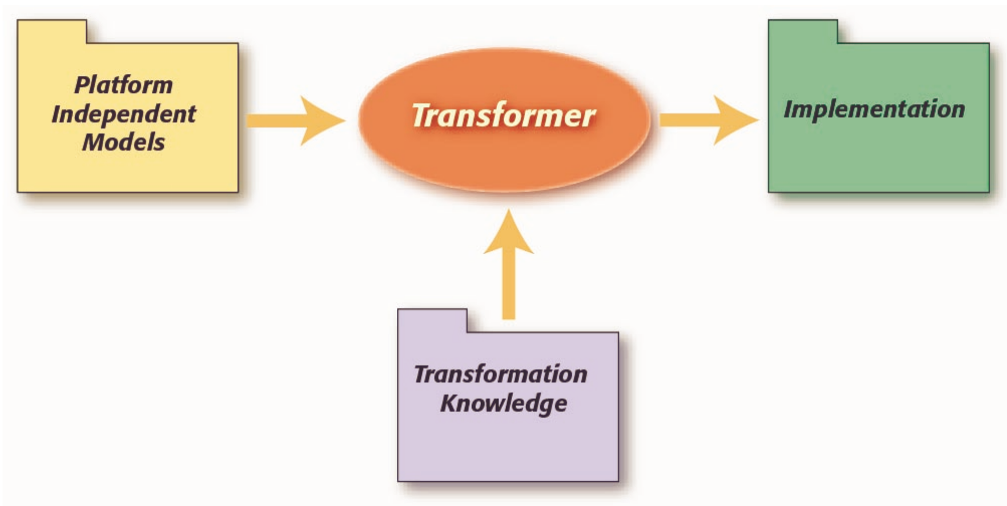
Object technology can help deliver a number of desirable characteristics (adapted and extended from Fusion, by Coleman et. al):

- **Correct** (does it deliver the right answers?)
- **Reliable** (does it deliver correct answers hour after hour, day after day?)
- **Testable** (can we test it easily?)
- **Debuggable** (can we locate and identify bugs?)
- **Correctable** (can we fix any bugs we discover?)
- **Flexible** (can we change the way it works?)
- **Extensible** (can we extend/enhance it to deliver new functionality?)
- **Adaptable** (can we use it in new contexts/environments?)
- **Interoperable** (can we connect it to other systems?)
- **Portable** (can we move it to new platforms?)
- **Reusable** (can we reuse parts of it in new systems?)
- **Reused** (are parts of it reused from other systems, libraries, frameworks?)
- **Tunable** (can we improve performance bottlenecks?)

Has Object-Orientation Kept Its Promises?

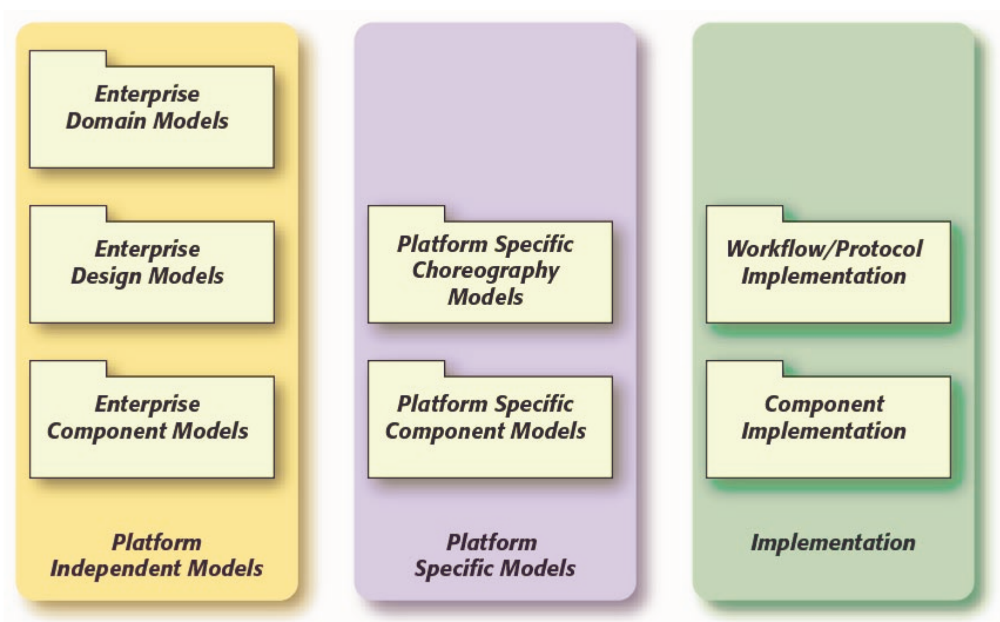
- Object oriented technology has brought us great advantages reducing spurious complexity and allowing us to reason resolve issues early
- Continuity minimize spurious complexity
- Formalization of models through requirements models helps us identify issues early
- We still have a long way to go
  - Reuse not significantly improved
  - Still expensive to build software

Model Driven Architecture (MDA)



- Use of platform independent models (PIM) as specification
- Transformation into platform specific models (PSM) using tools

PIMs and PSMs



Benefits of MDA

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit
- Speed of development
  - Most of the implementation is generated
- Quality of implementation
  - Experts provide transformation templates
- Continuity
  - 100% continuity from specification to implementation

Goals of This Tutorial

- Outline of a software development process that:
  - Provides a path from fuzzy requirements to system specifications appropriate as input to MDA tools
  - Support precise (but abstract) models at various levels useful for enterprise software development
- Show how we may apply MDA to generate implementations
  - Use of the InferData's MDA tool MCC to transform platform independent models into J2EE implementations
  - Discuss extensions required to standard UML models to be able to generate high quality implementations
- High level outline:
  1. Outline a UML based software methodology
  2. Create a PIM from scratch
  3. Generate J2EE implementations based on the PIM model

Software Processes

Covered in this section:

- Definition of software processes
- High-level overview of the essential software process phases

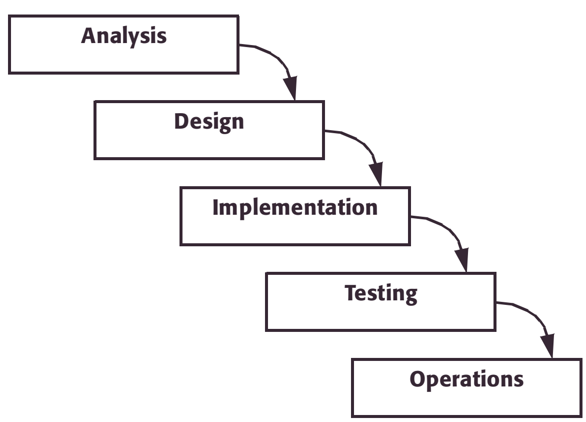
What is a Process?

- A process defines answers to three questions:
- ***Who*** is doing ***what***?
- ***When*** to do it?
- ***How*** to reach a goal?

Approaches to Software Development

- There are several models for software development
- Often, several approaches can be combined on a project
- Important approaches:
  - Hacking
  - Serial Development
  - Iterative Development
  - Incremental Development
  - Lightweight Approaches

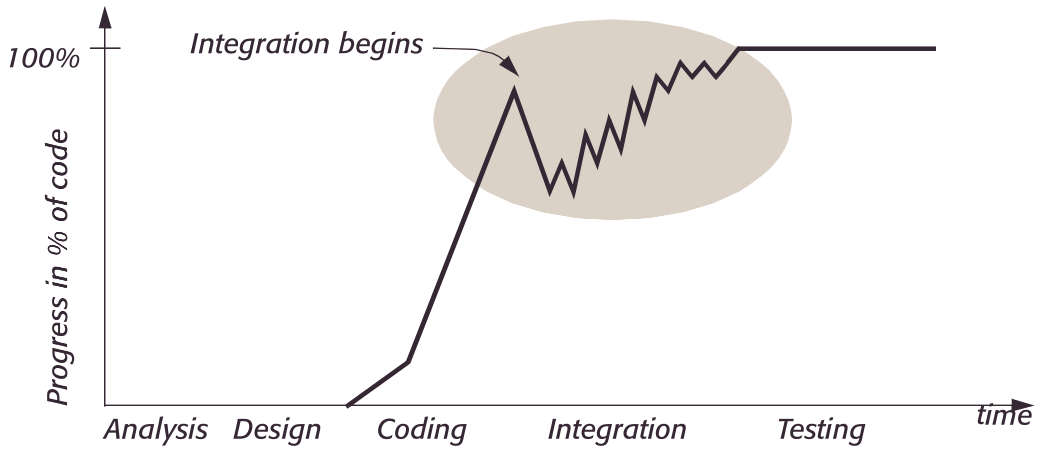
Serial Development: The Waterfall Model



- Introduced in 1970
- Popular for large projects
- Advantage: Improvement over previous practices
- Problem: it does not reflect the reality of software development
- Problem: does not support changes very well

- Advantage: deliverables and milestones are well defined

Waterfall Model Progress Profile



- Problems: coding is postponed until to late
- Integration and testing in the waterfall model can cost 40% of all expenses in development [Royce 98]

Risk and Requirements Management

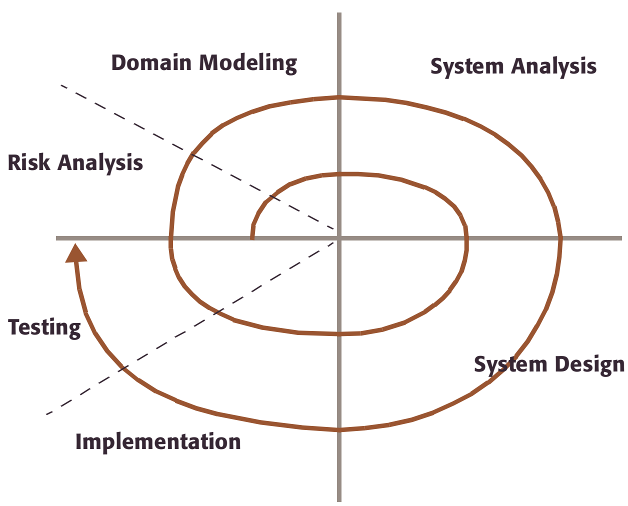
- In the waterfall model, coding is postponed
- Risk during analysis and design remains very high
- Risks are resolved in the coding and especially integration phase
- Design faults are exposed late
- Waterfall models often treats all requirements as equally important
- Only a small portion of requirements actually is a decision driver for the architecture of the system
- Requirements are often stated in a functional form

Iterative Development

- Focuses on reduction of risk
  - The goal: encountering risks as soon as possible
- An initial version of the system is produced, focusing on the high-risk areas
- Development proceeds by building on the core architecture
- Iterations are adding new functionality or new levels of detail
- Integration is continuous
- Iterations are planned, but plans can be changed

InferData Corporation

Iterative Development Model as a Spiral

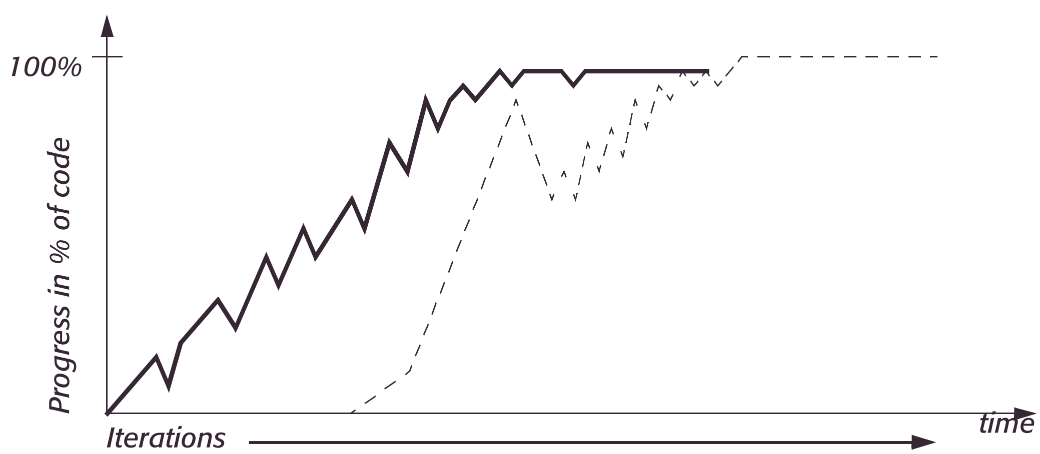


- The iterations are frequent and planned
- We test during and after every iteration
- The last iteration ends with a system release.

Incremental Development

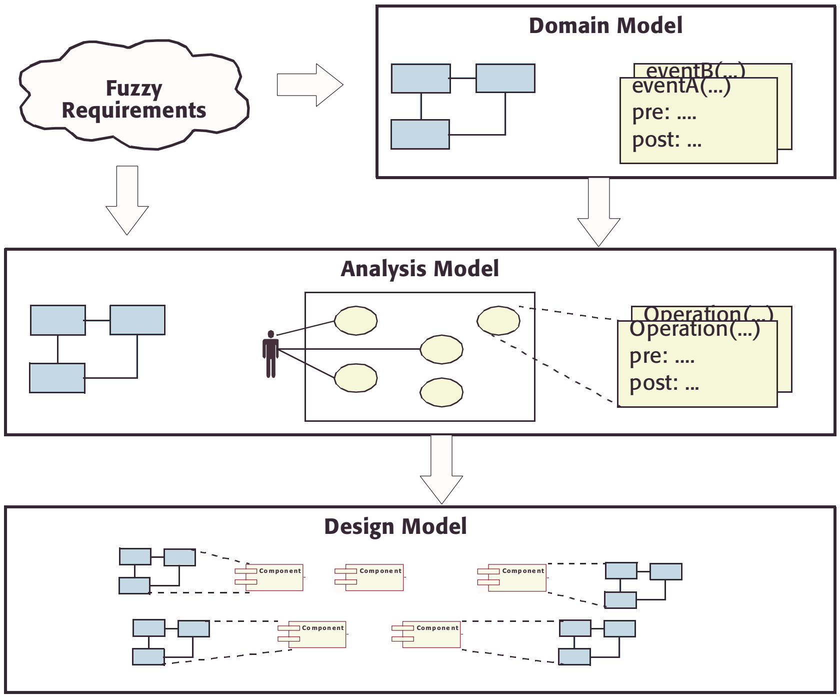
- The application is delivered in small releases
- Implement the key issues first
- Iterative development is typically combined with the iterative model:
  - Some iterations resolve technical risks
  - Some iterations are adding functionality
- Problem with incremental development is that it is not suitable for all application
  - Air traffic control, life support medical systems, ...

Iterative and Incremental Model Progress Profile



- Software is continuously integrated: significant reduction of risk
- Performance, scaleability, fault tolerance issues are exposed early in the development

High Level Process Overview



Conceptual Modeling

Covered in this section:

- The process of building conceptual models
- The artifacts produced to describe a conceptual model

Fuzzy Requirements

- Most requirements start out fuzzy

"We need to build a system to better serve our customers. The customers should be able to make reservation and check availability over the internet. The system should keep real-time status of the various hotels in the chain.

The system must also help facilitate the daily operation of the hotels, including room allocation, charge management and payment.

The system shall be easy to use and be built according to best practices in the industry..."

- Before we can build a system we have to:
  - Define the domain
  - Define what to automate
  - Define the software architecture (or foundations) to use

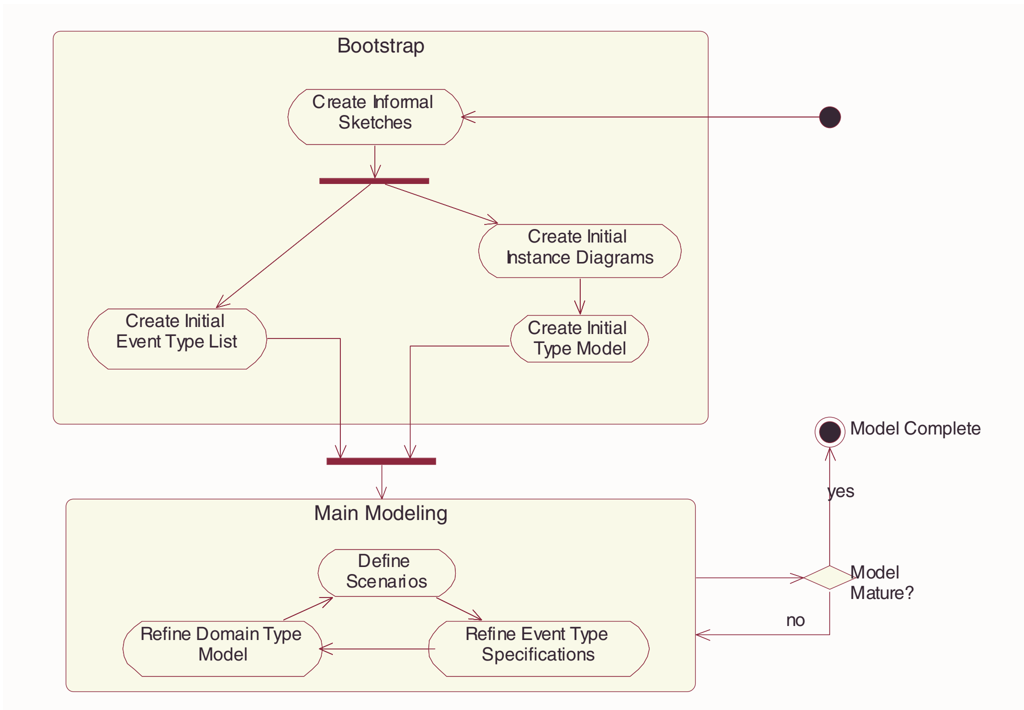
InferData Corporation

Conceptual Models

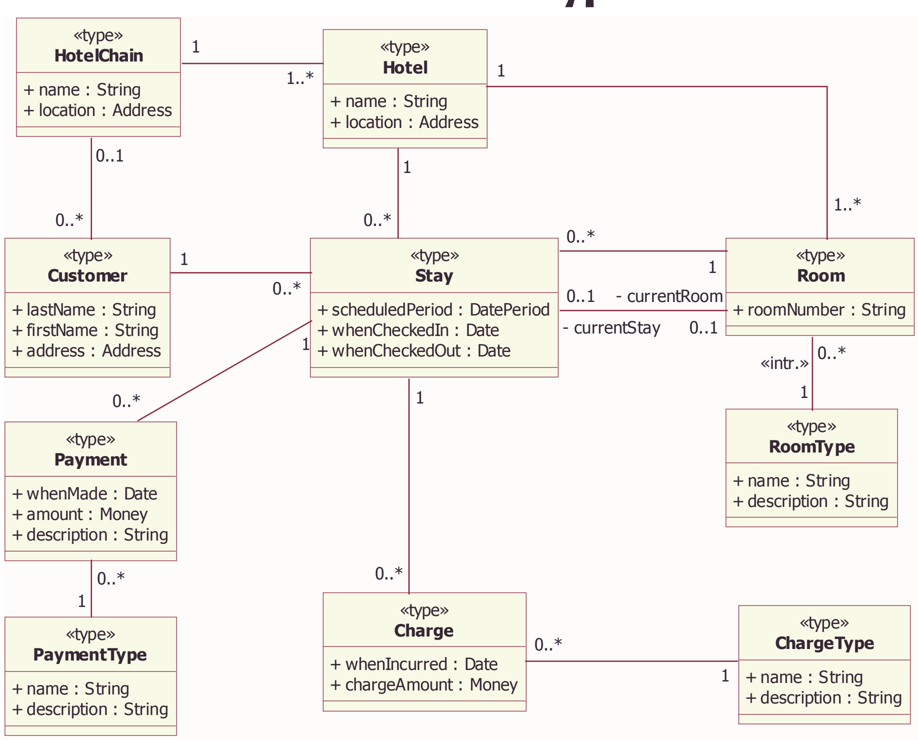
- A conceptual model describes the structure and behavior of a domain
- An object-oriented domain model describes the structure and behavior using objects
- A conceptual model can be informal or formal
- It can describe the world "as-is" or the world "to-be"
  - Sometimes it's useful to model the "as-is" and then the "to-be"
- The goals of a conceptual model are to:
  - Increase our understanding of the domain
  - Standardize the terms used to describe the domain
  - Serve as a starting point for building the system specification
- The notation for models is the Unified Modeling Language (UML)

Copyright © InferData 2003

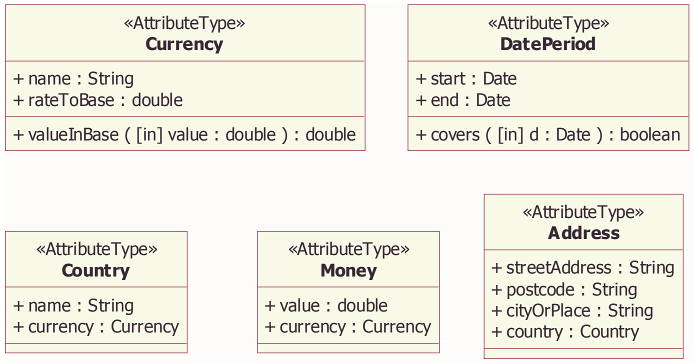
Workflow View



Domain Structure - Type Model



Attribute Types



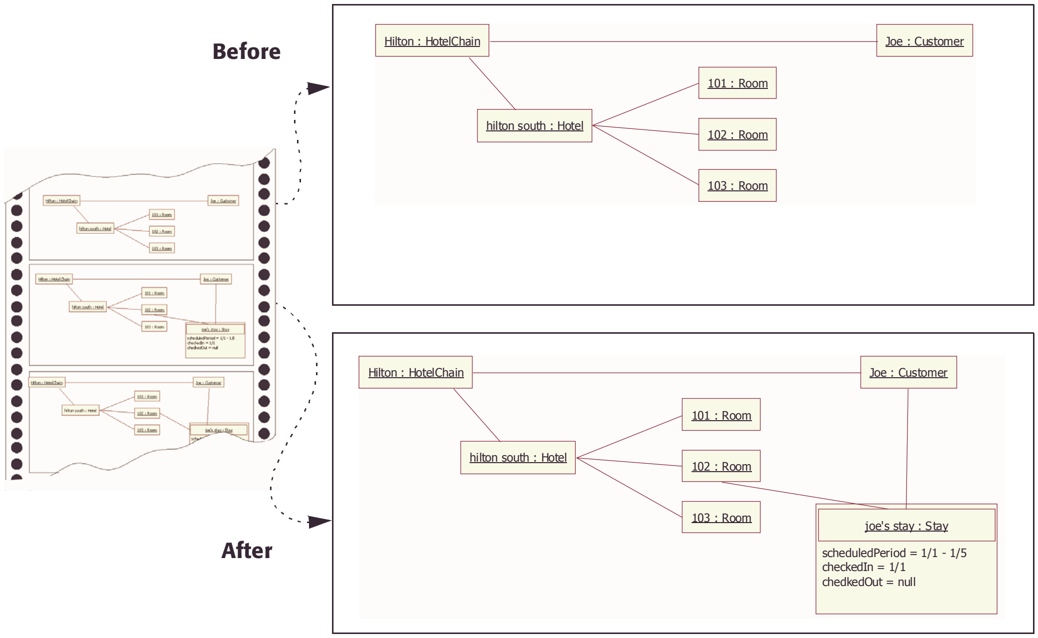
- Some types are defined but not modelled as first class type
- Model preferences determine if they become first class types or remain as attribute types

Domain Behavior - Event Type Specifications

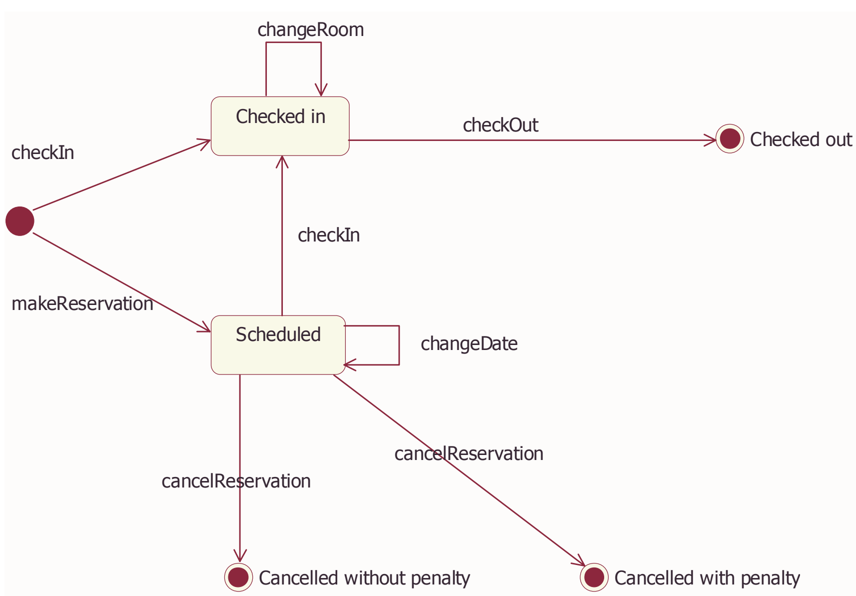
```
makeReservation( hotel: Hotel customer: Customer, period: DatePeriod, roomType: RoomType )

Preconditions:
  -- There is a room available in the requested date period
  hotel.rooms->exists( r |
       r.roomType = roomType AND
       r.stays->notExists( s | s.sheduledPeriod.overlaps( period ) )
  )

Postconditions:
  -- A new stay has been scheduled for a room of the room type requested
  hotel.stay->exists( s |
     s.scheduledPeriod = period AND
     s.isNew AND
     s.roomType = roomType)
```

- The behavior of the domain is captured in event type specifications
- The event type specifications are optionally formalized using OCL

Domain Scenarios



State Models



- Optionally, we enhance the models with state perspectives

Enterprise Design

Covered in this section:

- What is enterprise design?
- Axes of refinement
- Platform independent choreography

Critical Design Decisions

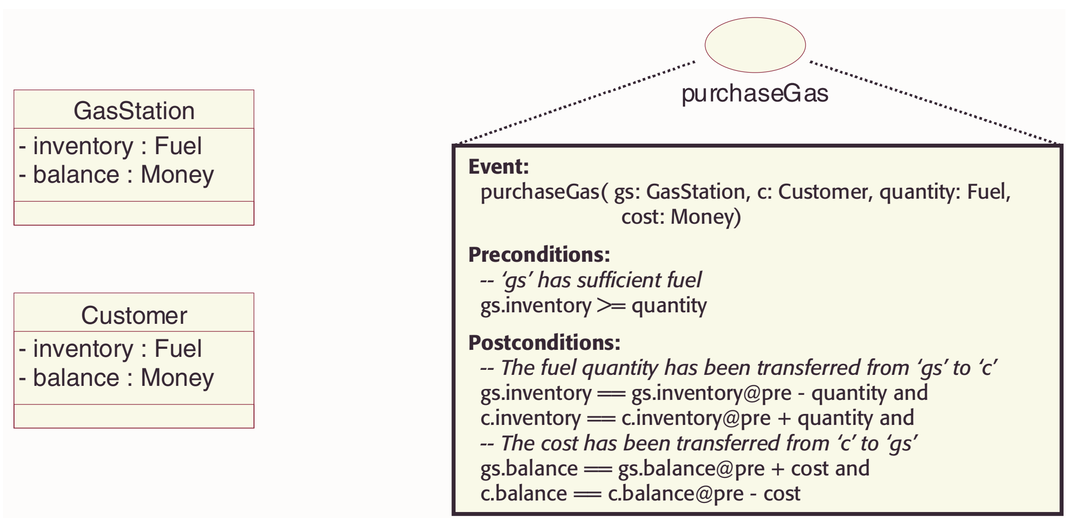
- Where and how are the data stored?
  - Who owns the data?
  - How do we improve performance through caching?
- Where are rules of the enterprise implemented?
  - The impact of locality on performance
- How is the system security achieved?

Axes of Refinement

- Enterprise design is a result of a series of refinements of the domain model
- Two possible dimensions of refinement:
  - Refinement in space
  - Refinement in time

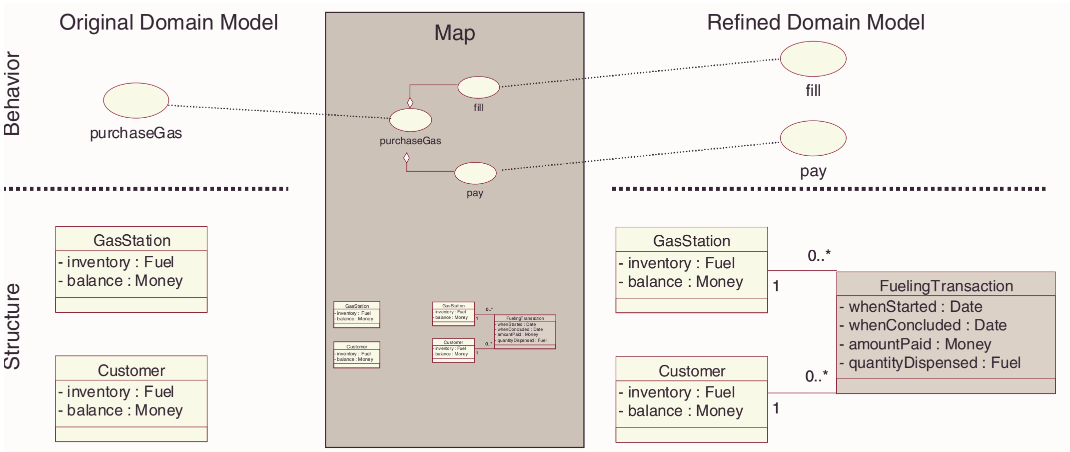
InferData Corporation

Gas Station Domain Model



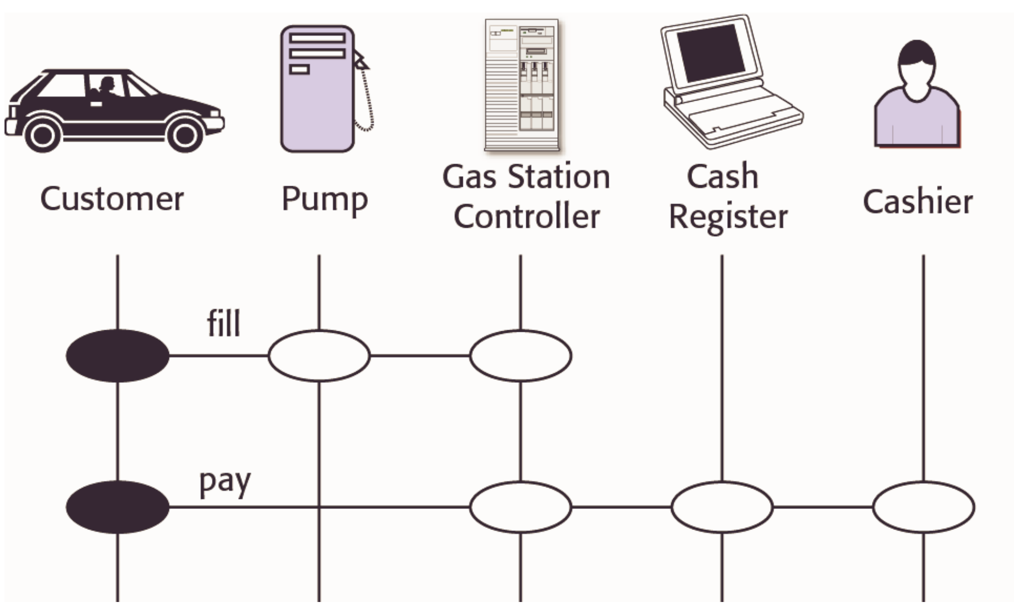
- Highest possible level of abstraction
- No matter what business process or systems are in place, it must be possible to purchase gas!

Refinement in Time



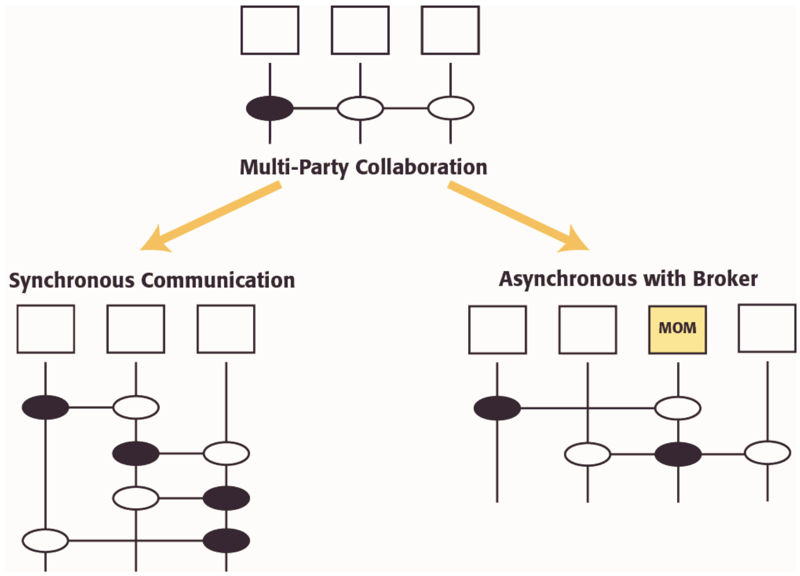
- Purchase gas refined in time
  - purchaseGas := fill + pay
- Refinement requires statespace to support temporal invariants
  - Introduction of the fueling transaction to remember that we filled but have not yet paid or visa versa

Refinement in Space



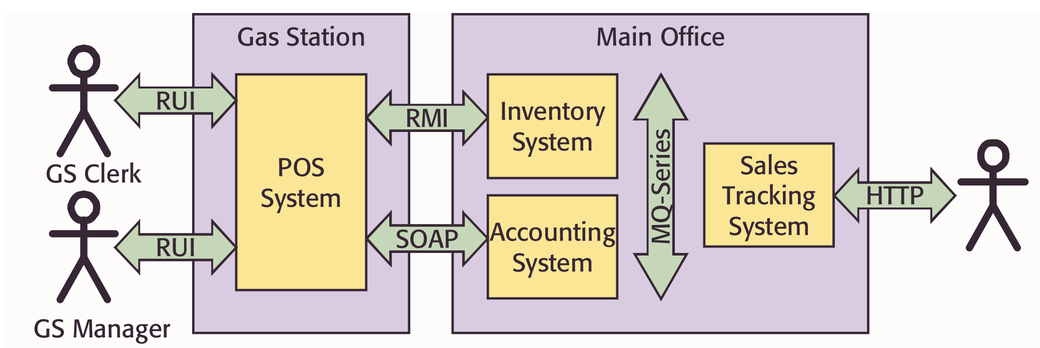
- Refinement in space introduces enterprise components with responsibilities
- Each component potentially requires separate models

Environmental Constraints for Enterprise Design



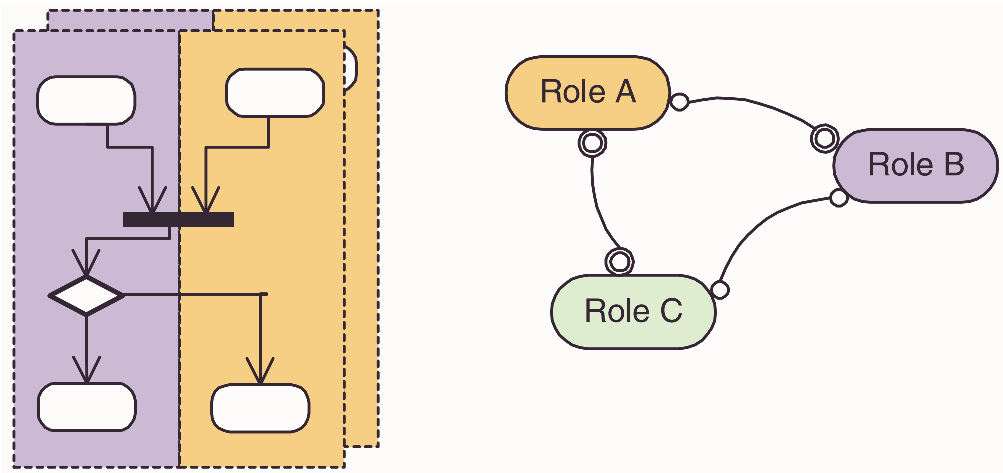
- Environment may impose restrictions on business process

Platform Specific Enterprise Architecture



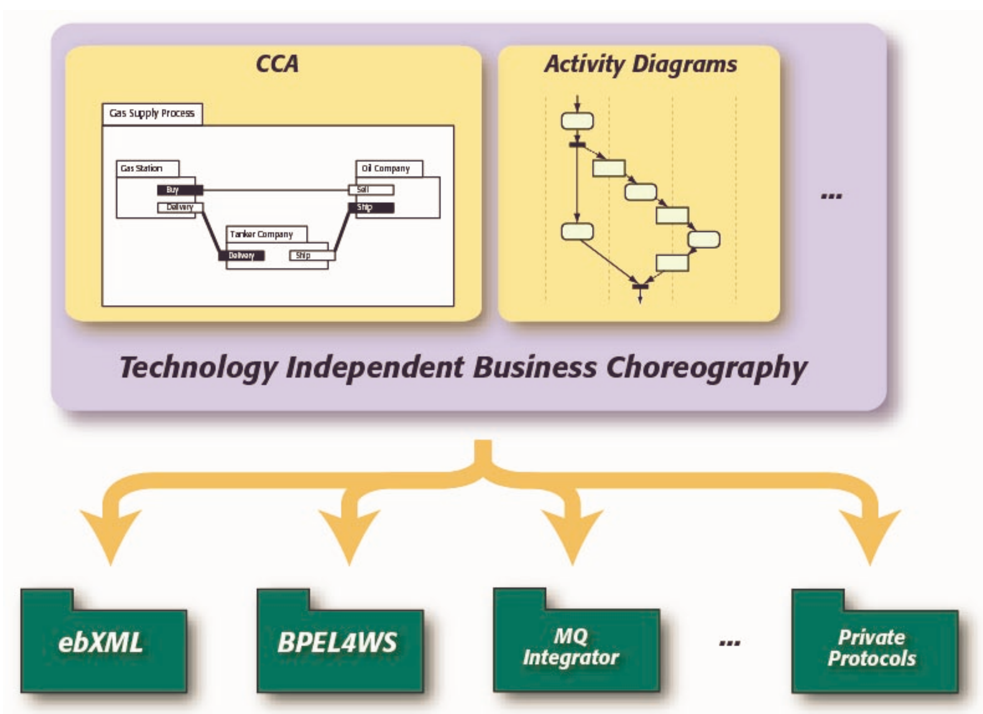
- The enterprise has many actors
  - Many systems
  - Many different kinds of users
- Potentially, each actor is connected through different technologies
  - Web Services
  - RMI/IIOP
  - etc.

Platform Independent Choreography

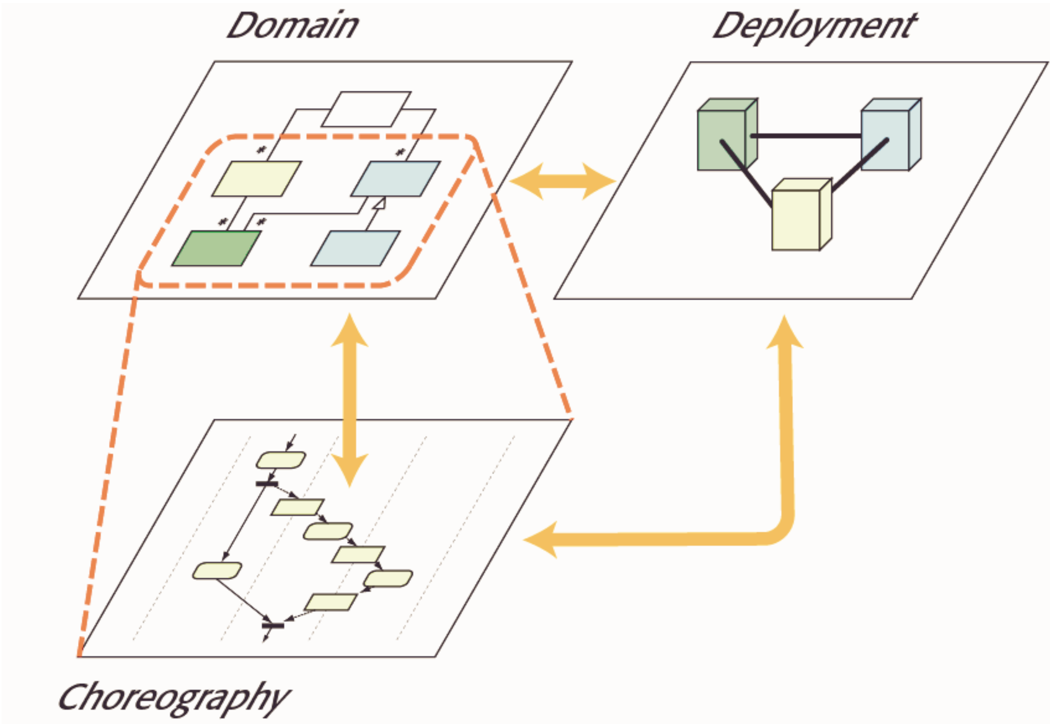


- Potentially, we could express choreography in a platform independent fashion:
  - UML Activity diagrams
  - OORAM models
  - EDOC

PIM to PSM Choreography



Continuity and Mappings



Choreography in This Tutorial...

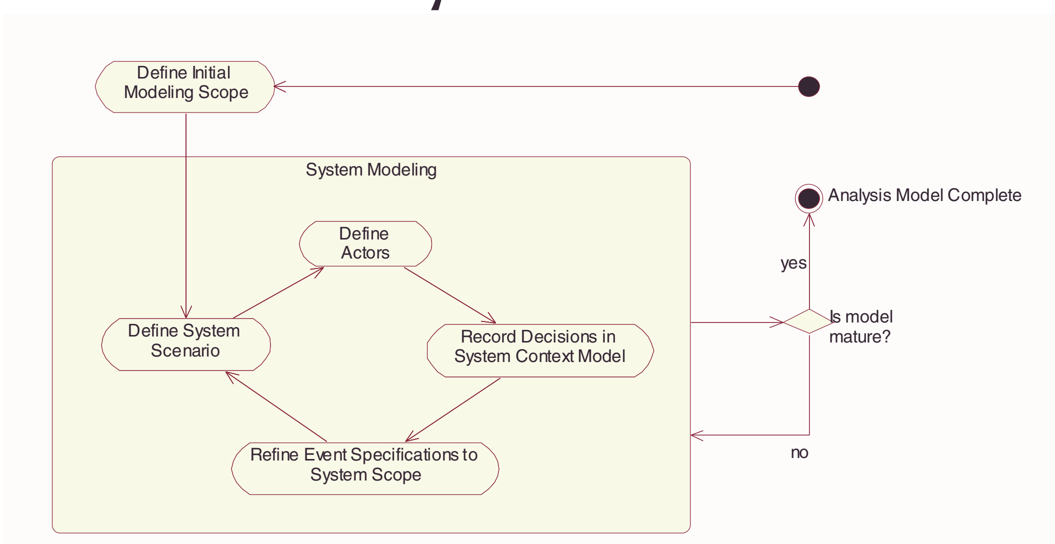
- We'll not focus on the choreography in this tutorial
- We'll rather focus on how ONE single actor, a system, may be implemented using MDA

Specification Modeling

Covered in this section:

- High level overview of the specification process
- A description of the artifacts produced to specify a system using UML

Analysis Workflow



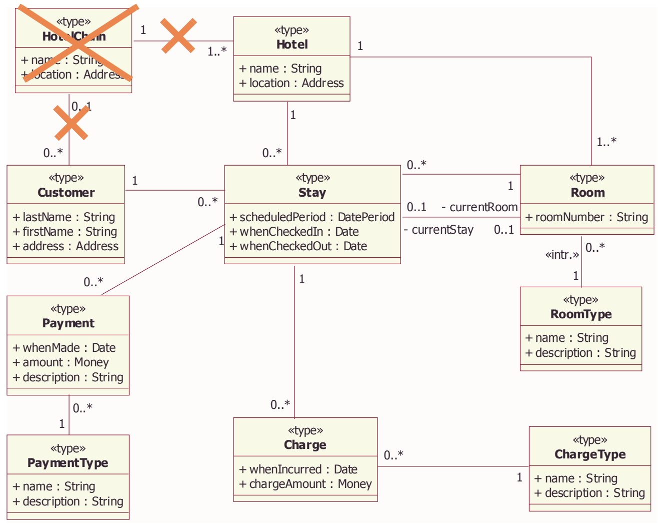
Artifacts

- To claim a complete specification/analysis model, we must produce
  - Analysis type model
  - System context model
  - System operation specifications
  - A selective set of scenarios
- Optionally we also produce
  - State model for key types
  - Activity diagrams describing the business design

Analysis Type Models

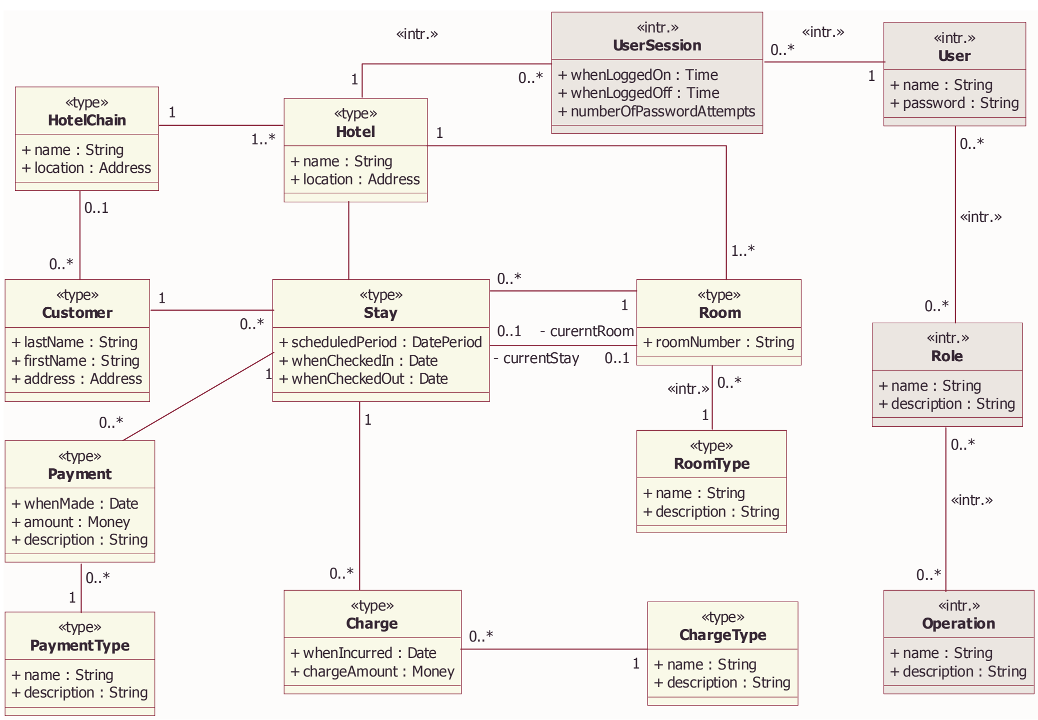
- The analysis type model uses the same notational constructs as the domain type model
- It defines the information types that the system is envisioned to persist
- The analysis type model may be a subset of the domain type model
  - If the domain model covered a greater area than the system involvement
- The analysis type model may introduce types not found in the domain type model
  - Types to handle the interaction between actors and the system
- Goal of the analysis type model
  - Maintain continuity to the domain type model
  - Provide vocabulary for all system operations

Analysis Type Model Subset Example



- We may for instance decide to create a system for individual hotels
- No need for hotel chains

Expansion Caused by Interaction



Use Cases / System Operations

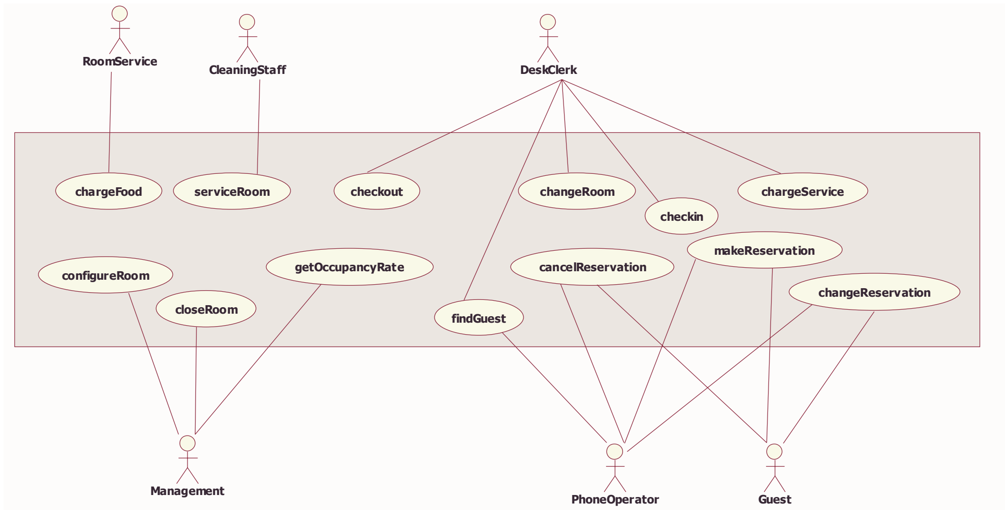
- The system operations describe some unit of behavior that the system is responsible for
- The system operations are most often refinements of the domain event types using the same notational constructs
- A bit simplified:
  - *"The system describes how the system is informed or detects a domain event and the responsibilities the system has when the event type occurs in the domain"*
- The system operation is often a direct copy of the domain event type, however...
- ... new operations may be required to support the interaction between the actors and the system
- Example:
  - Operations to validate the external actors. E.g. Logon, Logoff
  - Operations to configure the external actors. E.g. addUser, removeUser
- We may also refine a domain event type into finer grained system interactions

System Operations

```
makeReservation( hotel: Hotel customer: Customer, period: DatePeriod, roomType: RoomType )

Actor(s):
  Desk Clerk
  Internet Customer

Preconditions:
  -- There is a room available in the requested date period
  hotel.rooms->exists( r |
       r.roomType = roomType AND
       r.stays->notExists( s | s.sheduledPeriod.overlaps( period ) )
  )

Postconditions:
  -- A new stay has been scheduled for a room of the room type requested
  hotel.stay->exists( s |
     s.scheduledPeriod = period AND
     s.isNew AND
     s.roomType = roomType)
```

- The system operations must define who performs the operations (Actors)

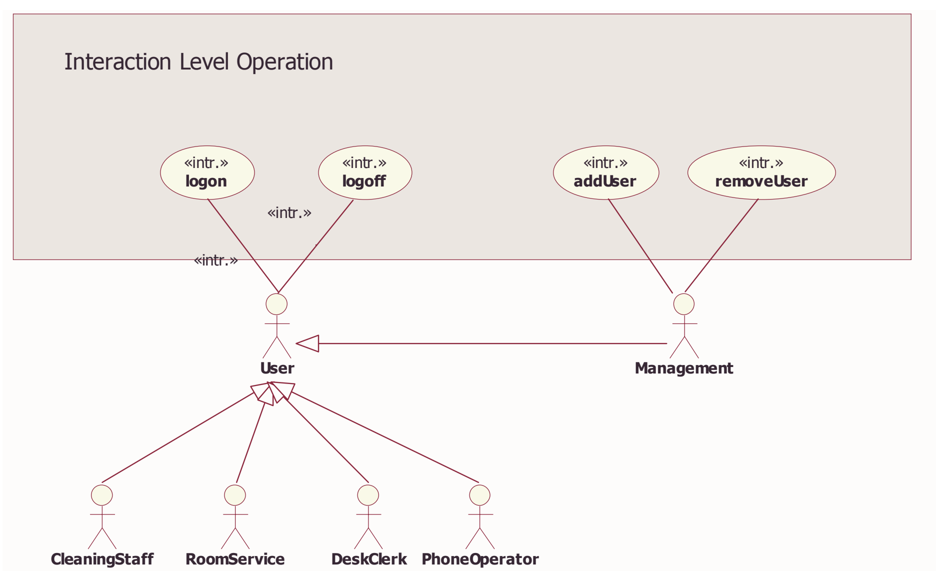
System Context Model - Domain Behavior



- We document the context of the system operations in *System Context Models*
- The diagram above shows the system operations derived from the domain event types

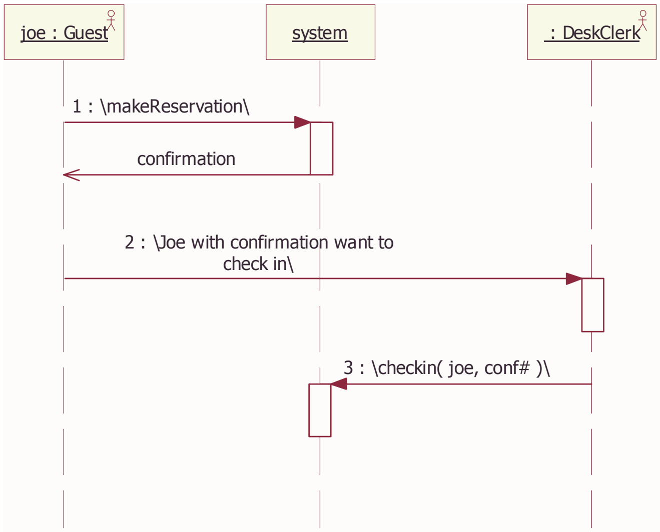
InferData Corporation

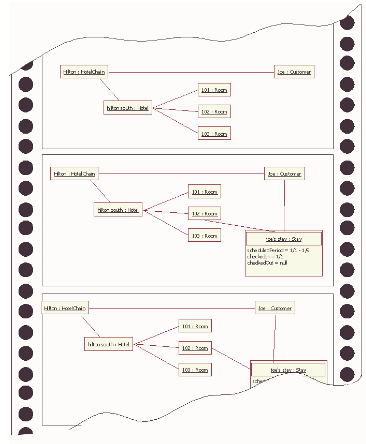
System Context Model - Interaction Behavior



- The system context model should also include the interaction level system operations
- It is recommended to keep these operations in separate diagrams

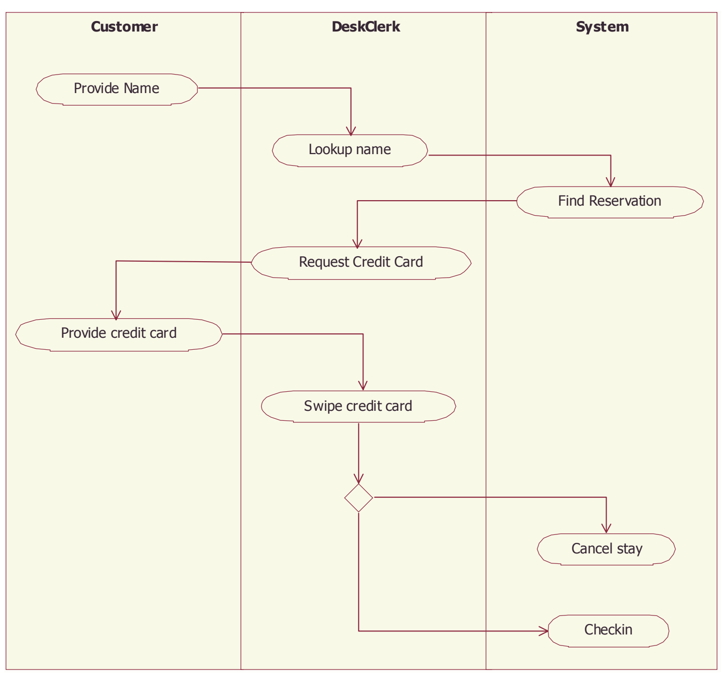
System Level Scenarios





- The system level scenarios describe how domain scenarios are to be realized when the system has been built
- We can reuse the domain scenarios with added operational context

Activity Diagrams



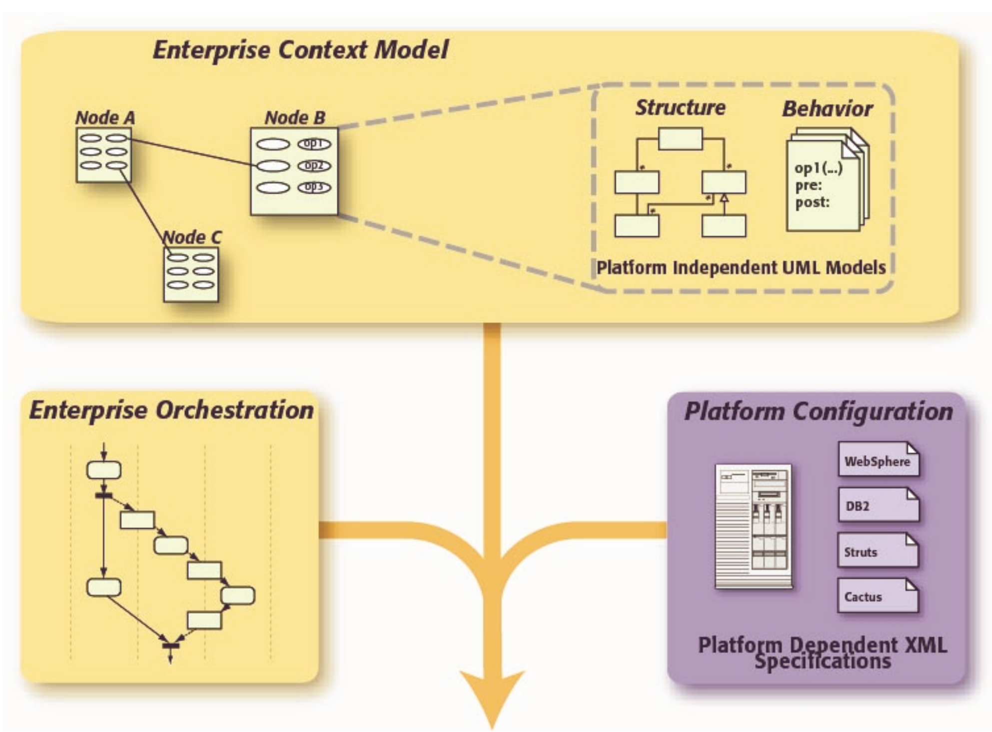
- Business processes can be described further using activity diagrams

Model Driven Architecture

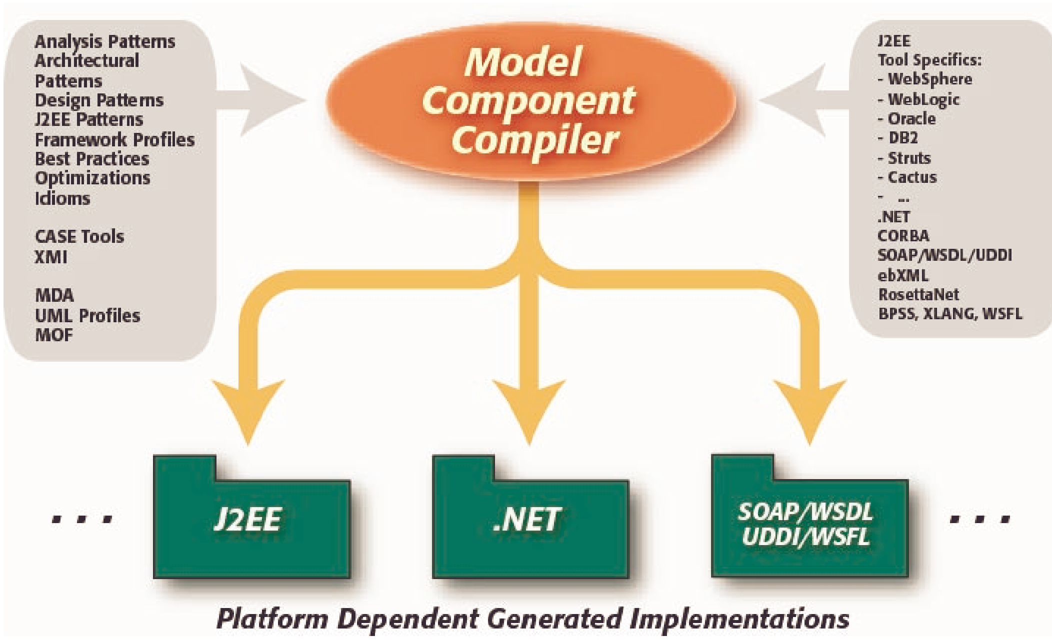
Covered in this section:

- Artifacts and architecture for an MDA tool

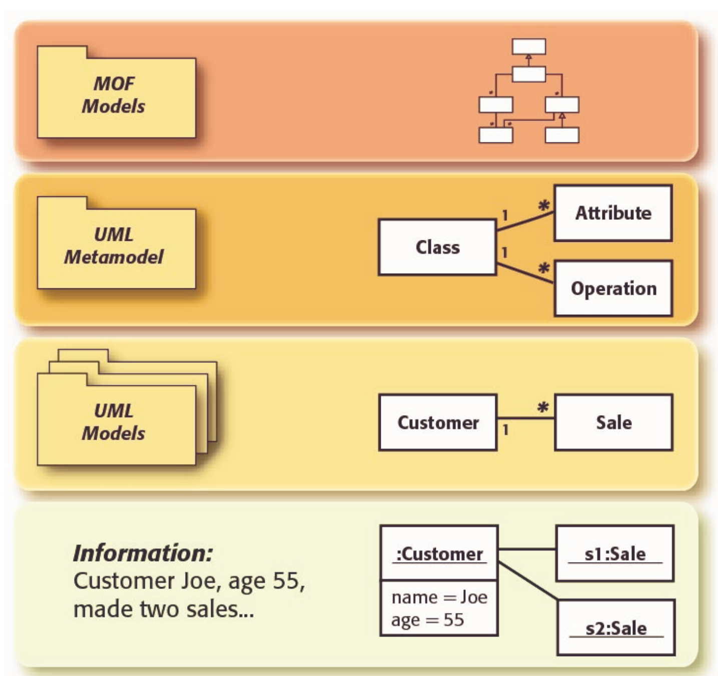
MCC Input Models



Transformation and Output



Meta Object Facilities (MOF)



CASE STUDY!!!

InferData Corporation

Copyright © InferData 2003