Domain Driven Modernization of Legacy Systems

Dr. Vladimir Bacvanski

Petter Graff

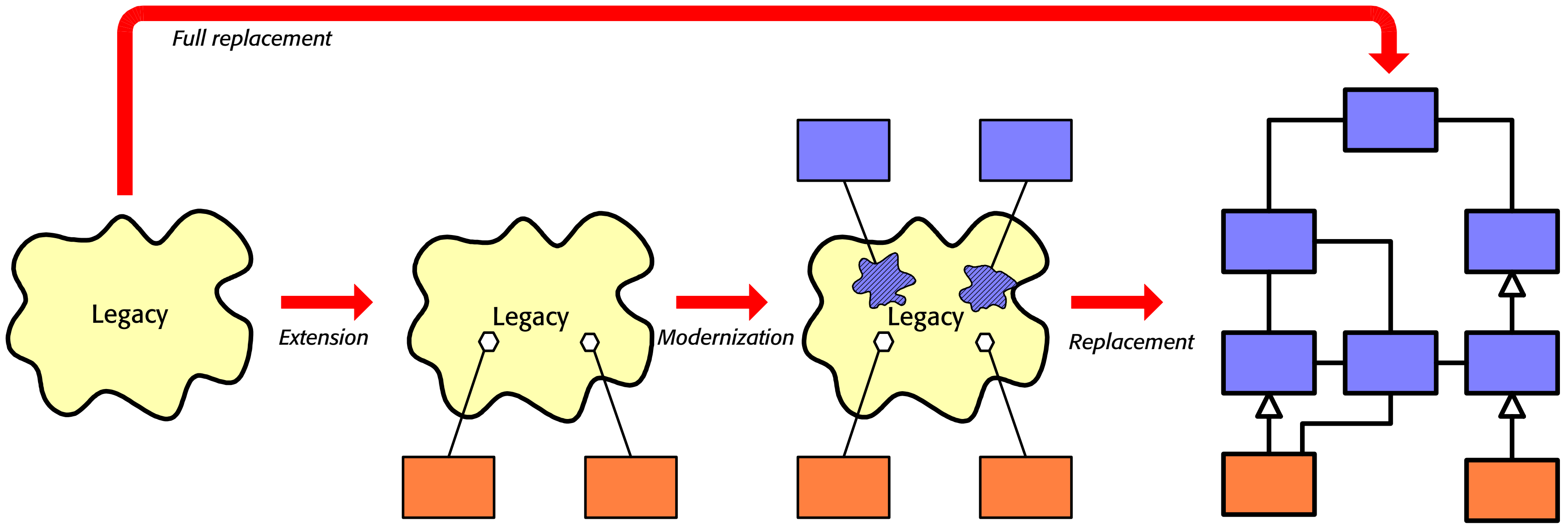
vladimir@inferdata.com

petter@inferdata.com

Overview of the Talk

- Modernization and Replacement Challenges
- Domain and its Systems
- Domain Driven Modernization Process
- Domain Models
- System Models
- Representing Legacy Functionality
- Use of MDA

Modernization and Replacement



Legacy with extensions implemented in new technology.

Legacy with extensions and replaced areas of functionality

Legacy system fully replaced and extended

🟧 Extension in new technology —○ Connection point in the legacy 🟦 Replacement in new technology

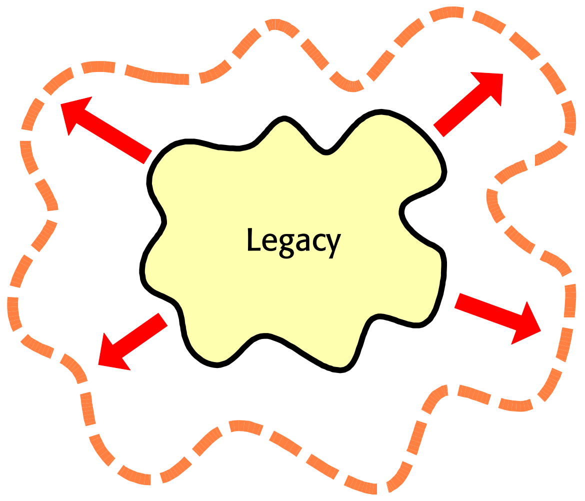
Key Challenges

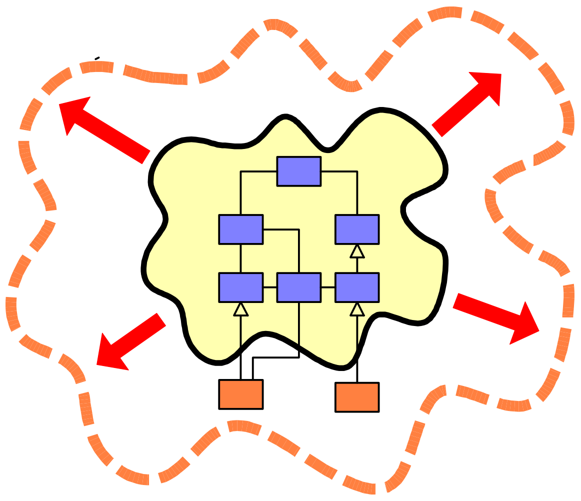
- Align modernized system with business goals
- Preserve the assets of the legacy systems
  - Extensions need to match the legacy system
- Eventually, replace the legacy system
  - Do not just implement the old system in the new technology
  - Migrate to new, component based architecture
  - The extensions should be reused

Disadvantages of Traditional Approaches

- Understanding the legacy system
  - If the source of information is the legacy system, there can be a mismatch between the implementation and the new business domain
- Modernization through reverse engineering:
  - The new system carries the paradigm of the legacy system
  - Often not aligned with the new business environment
- Extending the system
  - Extensions added in the paradigm of the legacy system may not be aligned with the business domain
  - May cause rework when the system is replaced

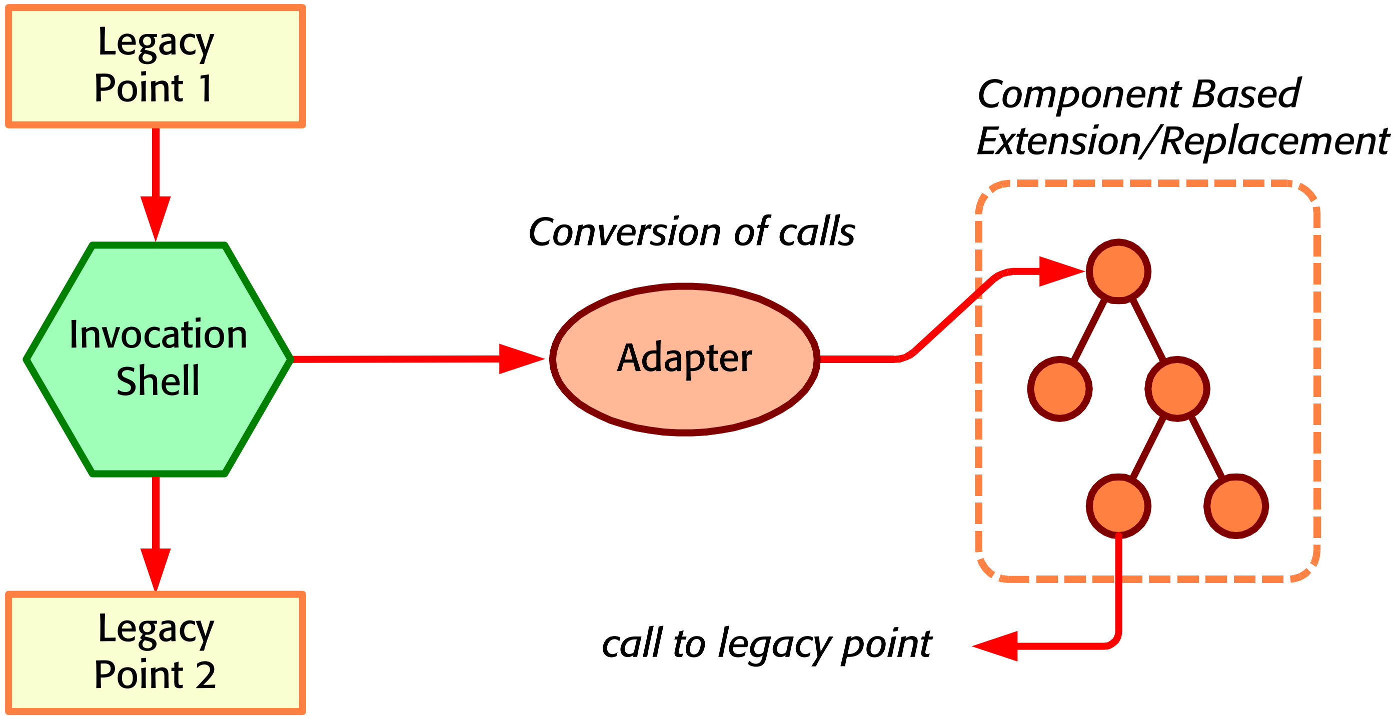
Extending Legacy Systems





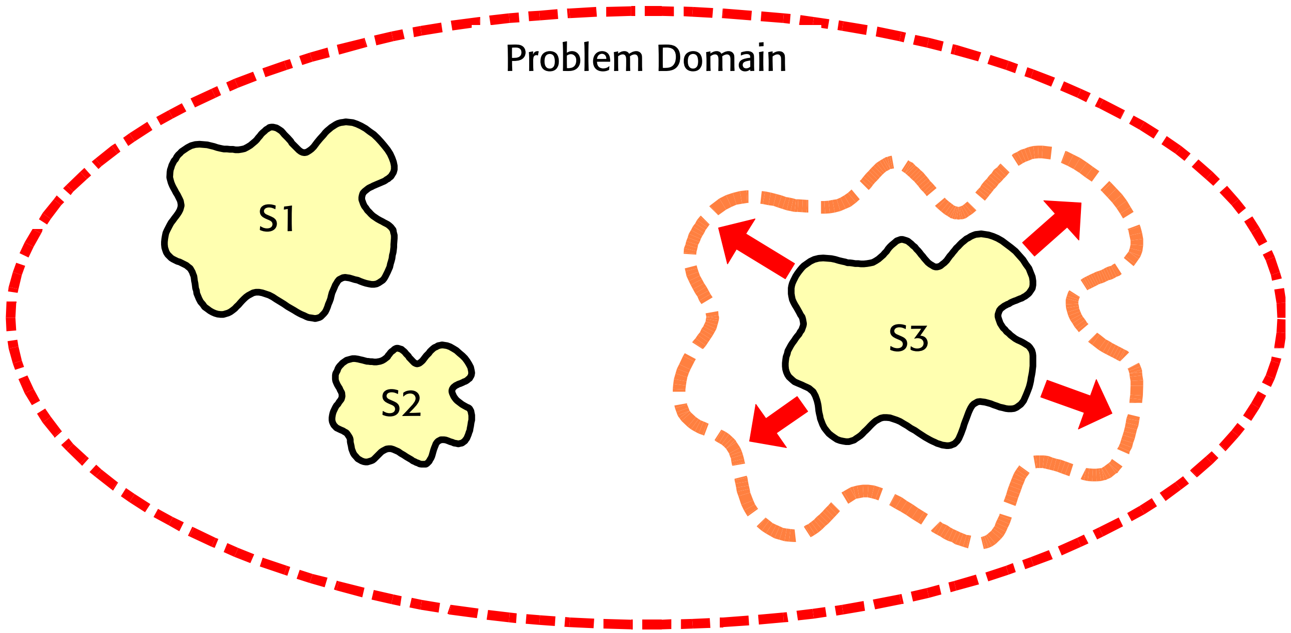
- Extending legacy:
  - Direct extensions follow the paradigm of the legacy system
  - Replacement will have to re-implement the extensions
- Extending modernized system:
  - Extensions need to match the legacy and the new components
  - The legacy system will eventually be replaced

Communication Through Adapters



- What is the information to be exchanged?
- What is the communication protocol?

Domain And The Systems

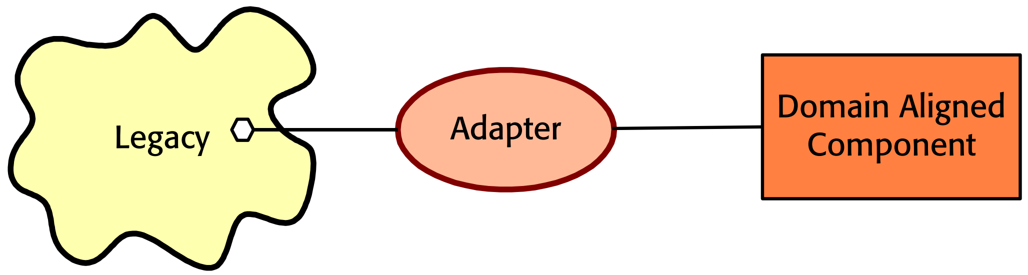


- A system covers an area of the domain
- Knowing the domain enables the identification of future extensions

Understanding Legacy Systems

- For reengineering and modernization efforts we can choose two starting points:
  - Start with the system itself
  - Start with the domain
- Domain provides the context for understanding of legacy systems
  - Concepts from the domain are mapped to the system and vice versa

Aligning Extensions With The Domain

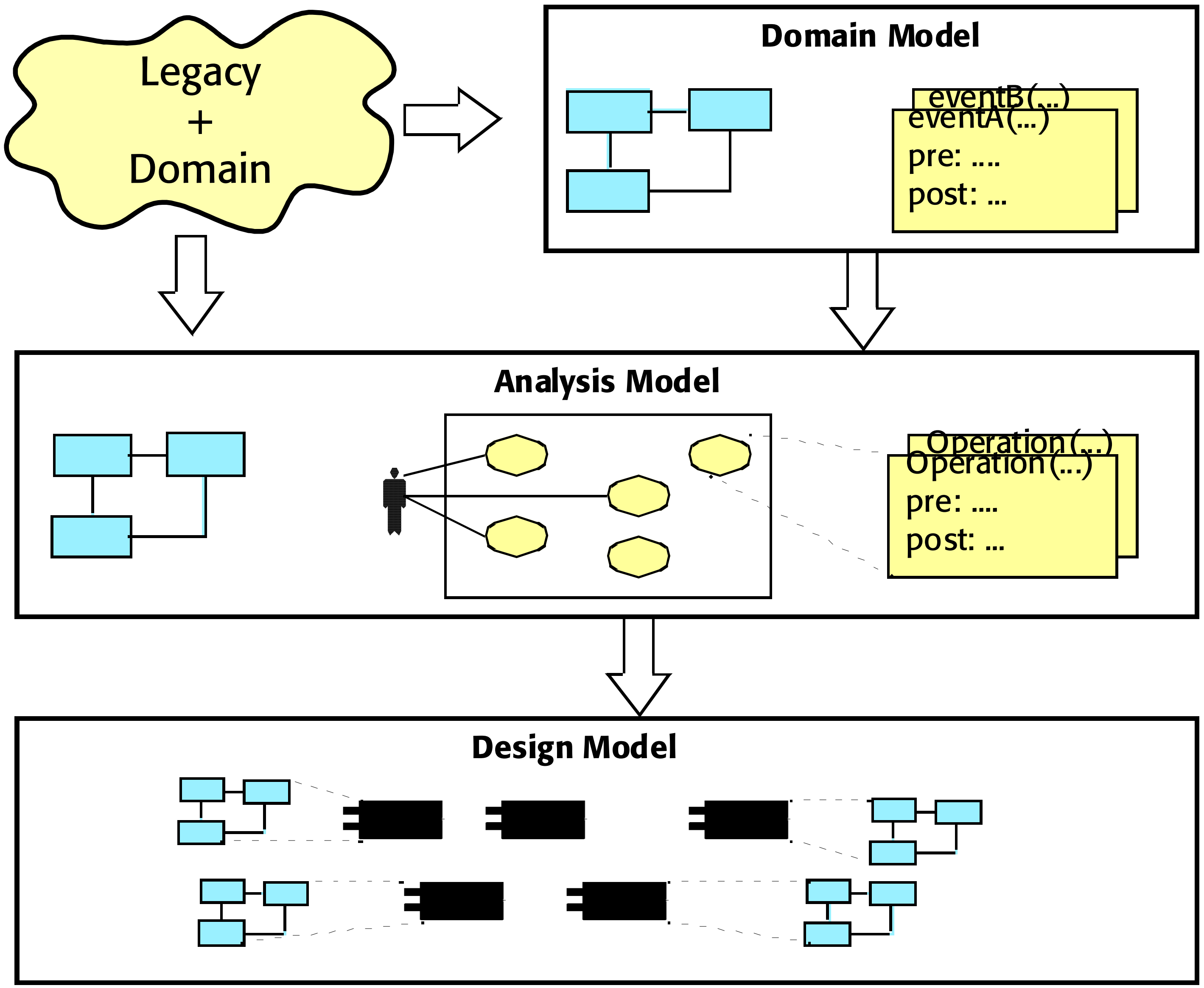


- Extensions are identified in the domain
- Extensions are integrated with the components that map to the concepts from the domain
  - Adapters between the legacy and the domain aligned components are needed
- Domain models are essential if the system needs to become the foundation of a product line

Domain Driven Modernization Process

1. Model the domain
2. Model the existing system
   - Develop model adapters between the system model and the domain
3. Model extensions if needed
4. Generate component interfaces
5. Develop adapters
6. Generate extensions

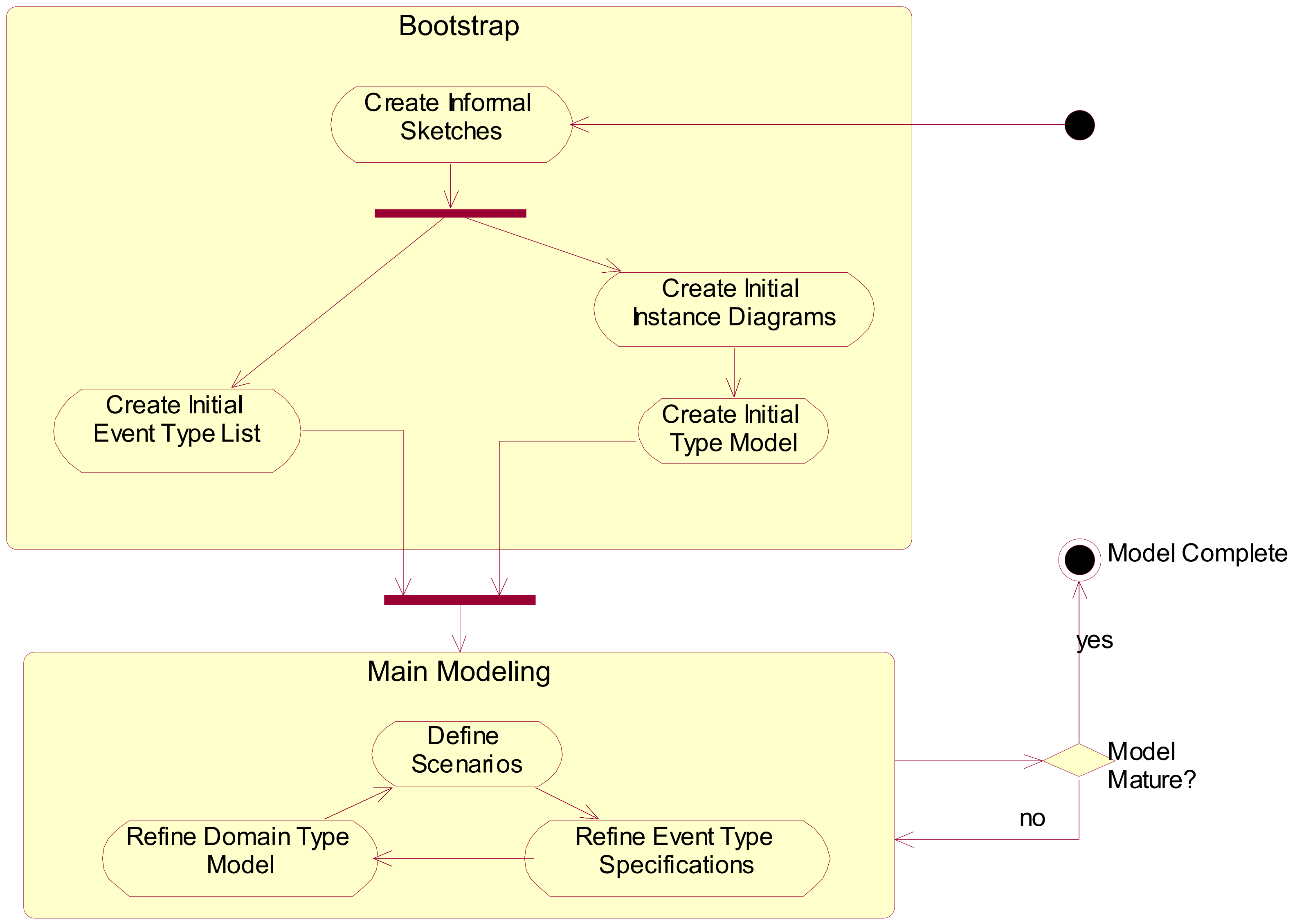
High Level Process Overview



Domain Models

- A domain model describes the structure and behavior of a domain
- An object-oriented domain model describes the structure and behavior using objects
- A domain 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 domain 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 UML

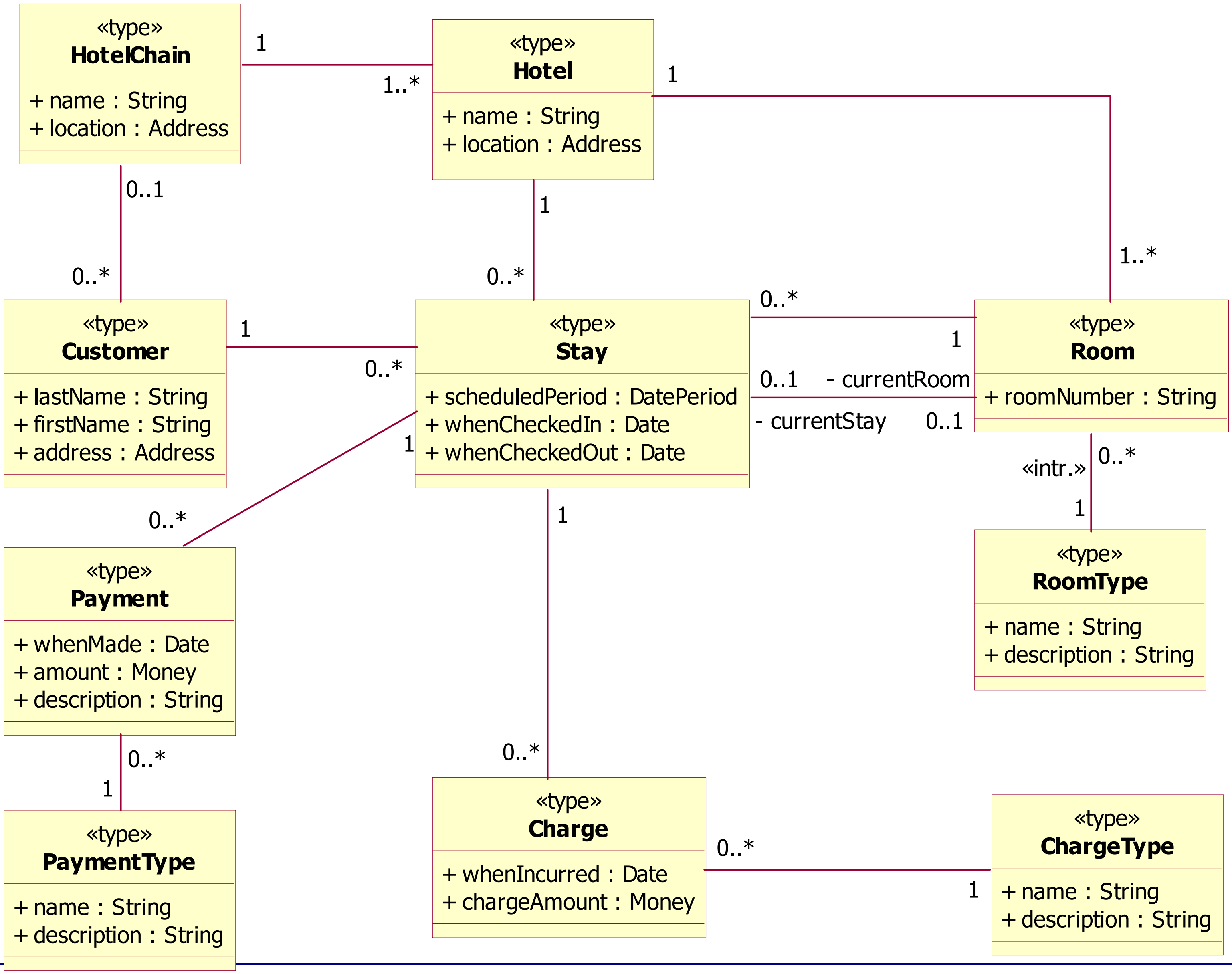
Domain Model: Workflow View


Bootstrap
Create Informal Sketches
Create Initial Instance Diagrams
Create Initial Event Type List
Create Initial Type Model
Model Complete
yes
Main Modeling
Define Scenarios
Model Mature?
no
Refine Domain Type Model
Refine Event Type Specifications

Example: A Hotel System

- A hotel system needs to be modernized and extended
- Eventually, the system is going to be replaced with a new one
- The extensions to the system need to be reusable in the new system

Conceptual Model: Type Model

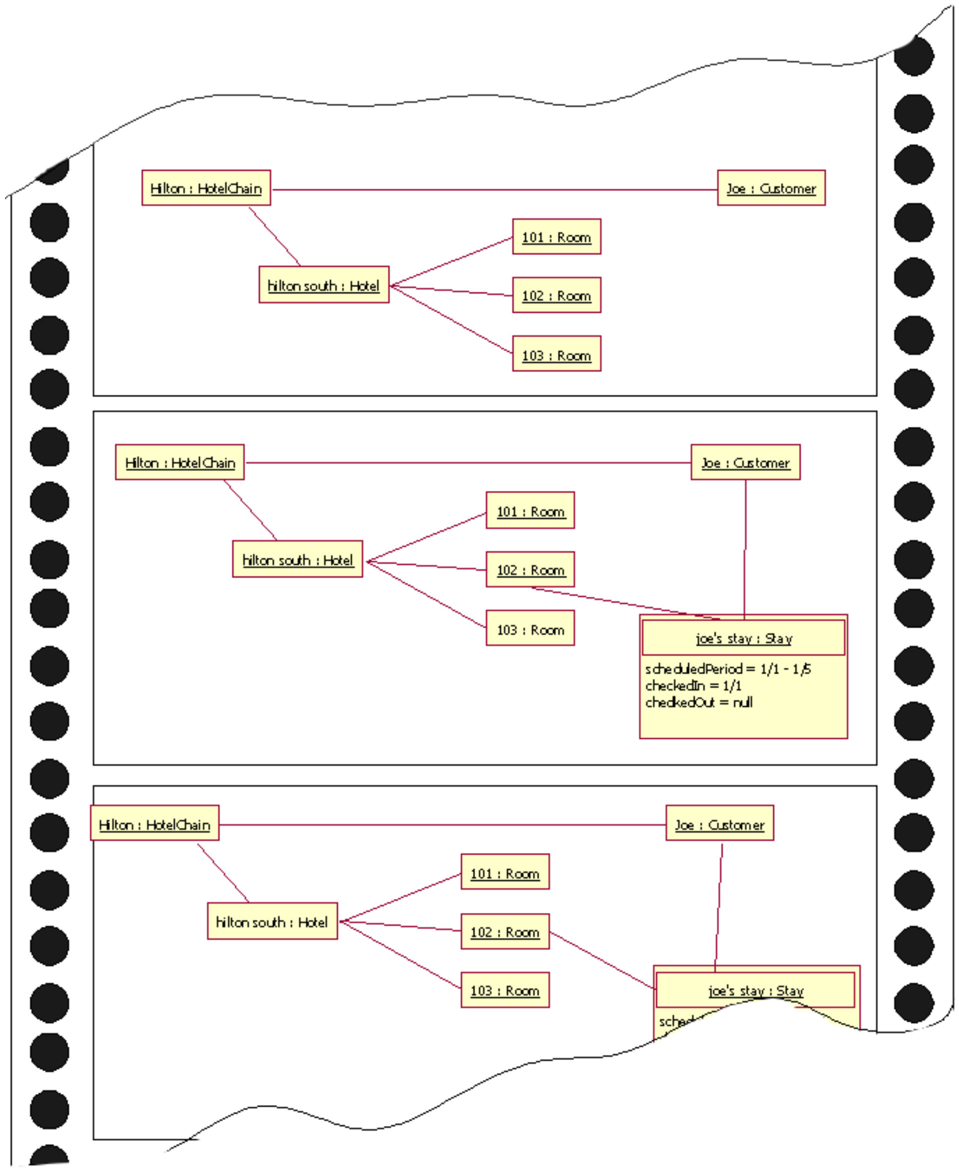


Domain Behavior - Event Type Specifications

```
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.scheduledPeriod.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)
```

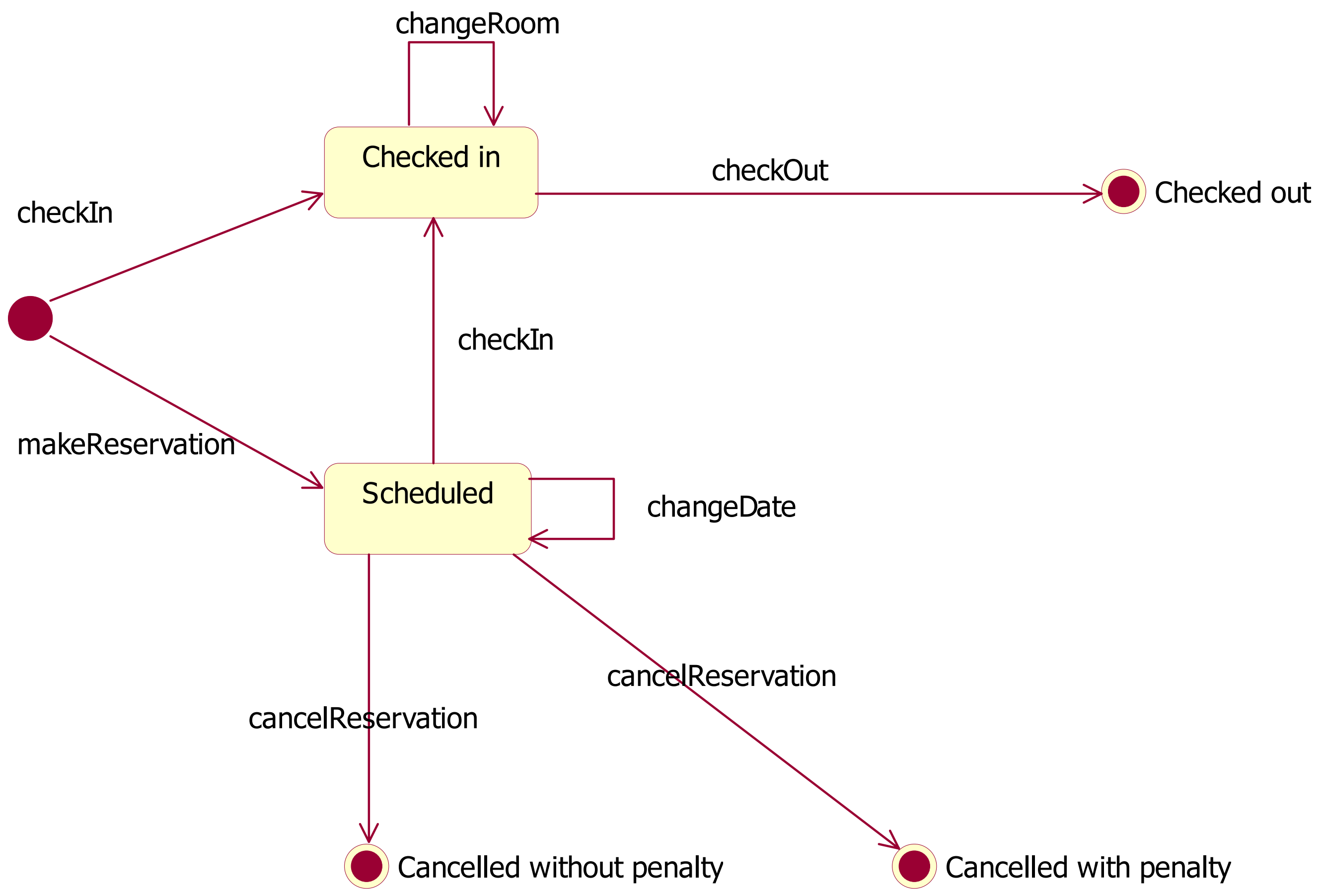
- The behavior of the domain is captured in event type specifications
- The event type specifications are optionally formalized using OCL

Domain Scenarios



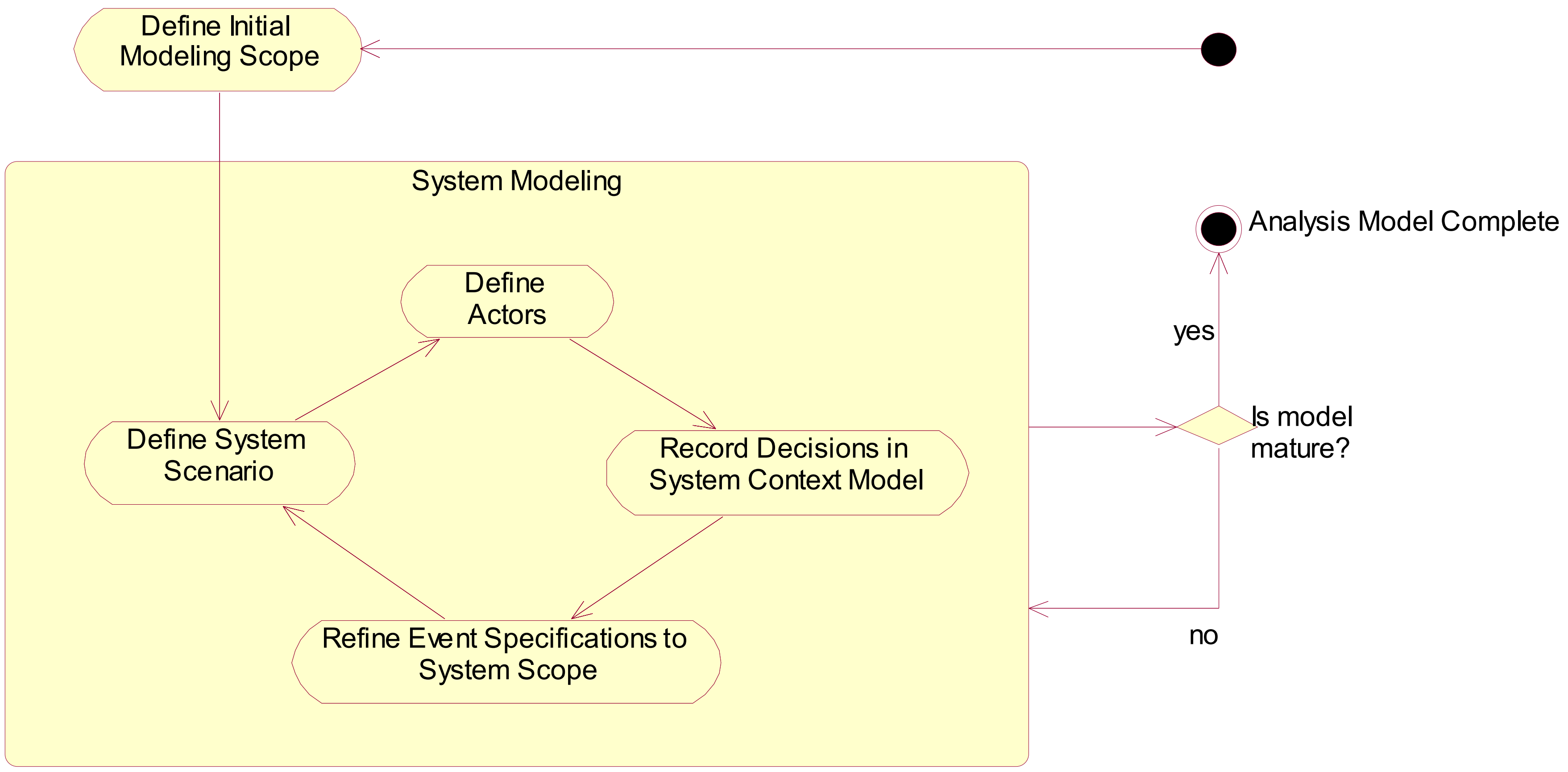
- Scenarios are useful to illustrate how the model works
- Increases the understanding of the model
- Crosschecking of the models

Use of State Models



- Optionally, we enhance the models with state perspectives
- Excellent crosschecking tool

System Analysis Modeling Workflow


Define Initial Modeling Scope
System Modeling
Analysis Model Complete
Define Actors
yes
Define System Scenario
Record Decisions in System Context Model
Is model mature?
no
Refine Event Specifications to System Scope

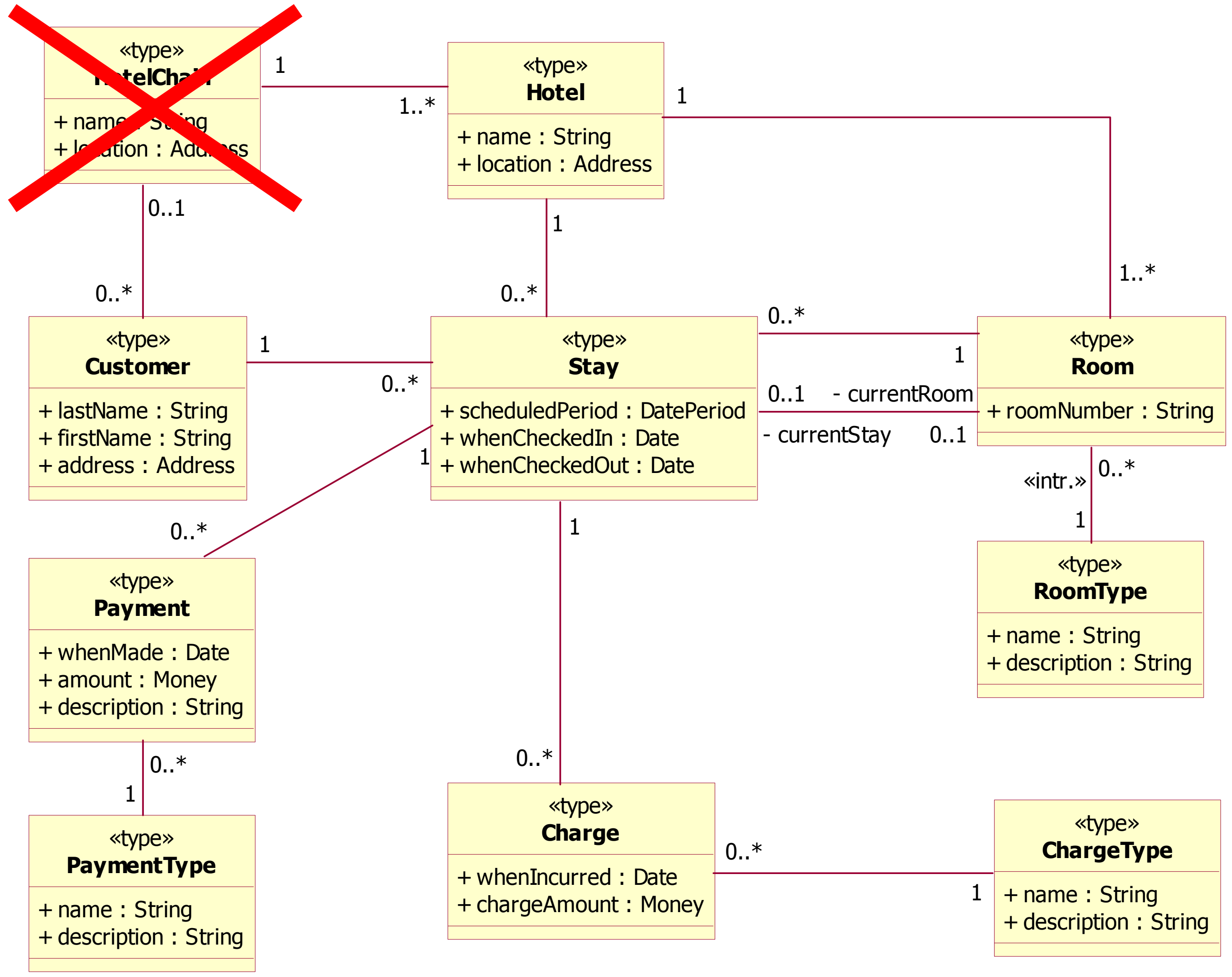
Artifacts Produced in Analysis

- 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

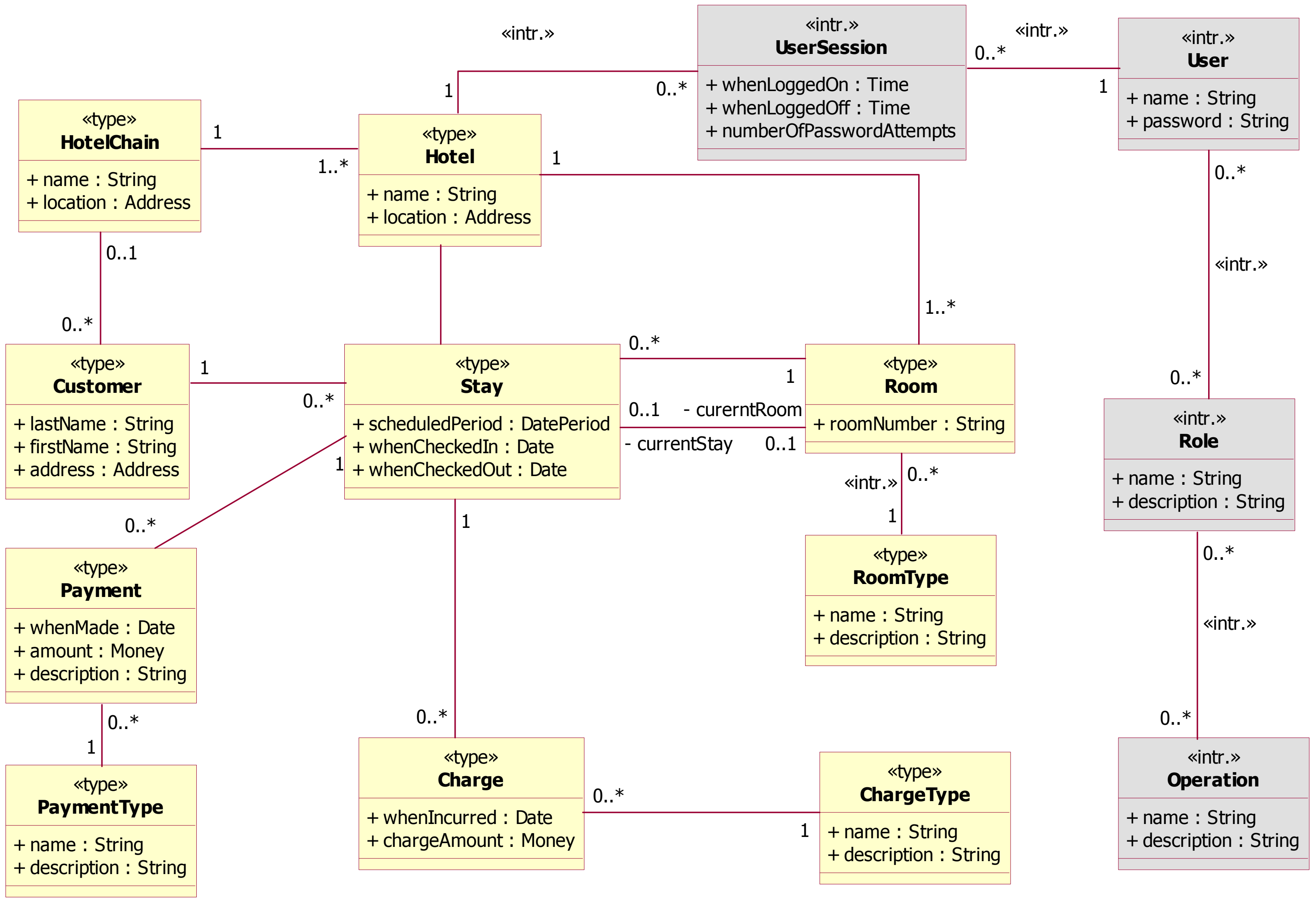
- The analysis type model uses the same notational constructs as the domain type model
- It defines the information types that the system is envisioned to persist
- The analysis type model may be a subset of the domain type model
  - If the domain model covered a greater area than the system involvement
- 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 chains)

Expansion Caused by Interaction / Legacy



System Operations or Use Cases

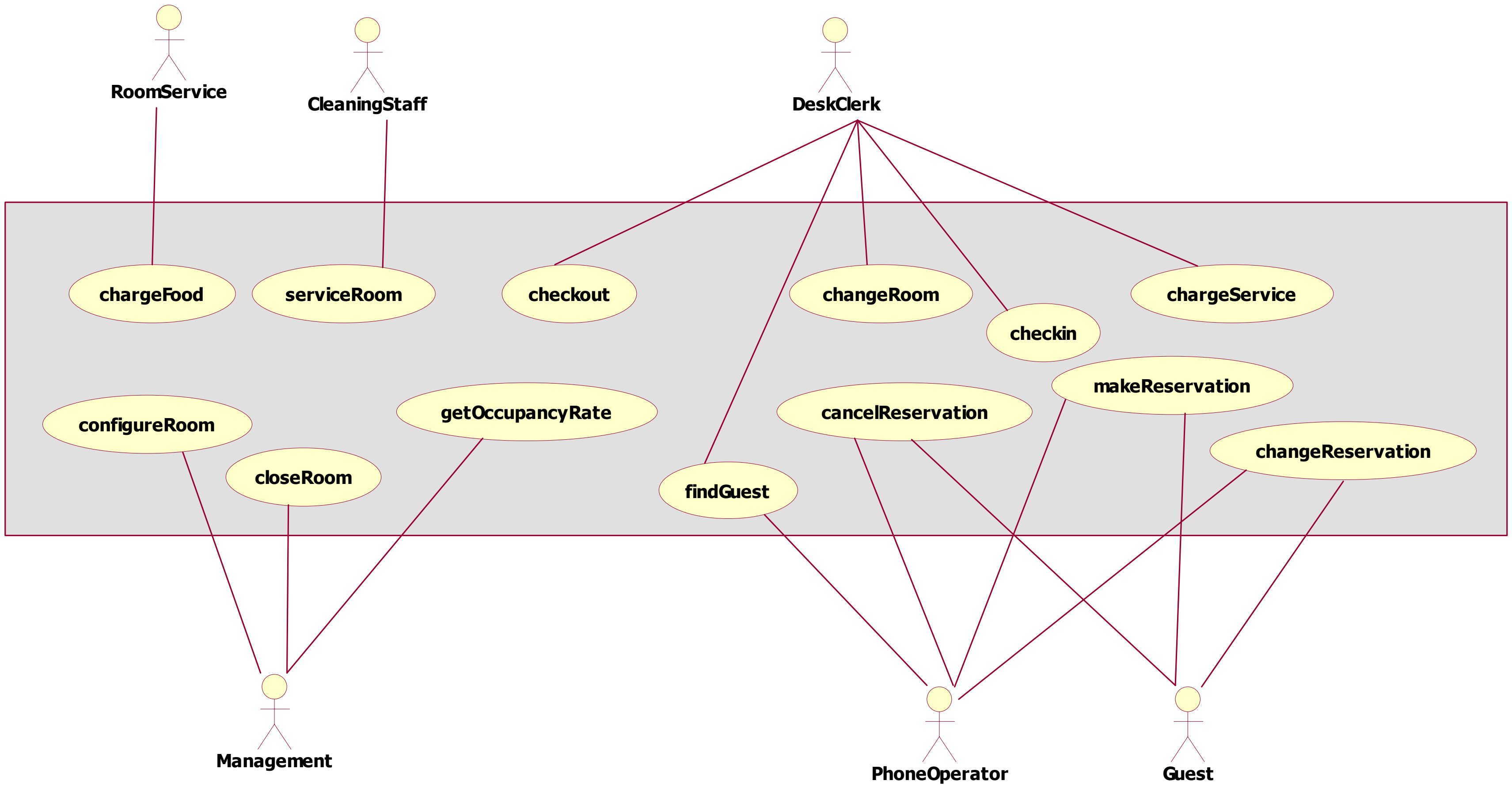
- The system operations describe some unit of behavior that the system is responsible for
- The system operations are most often refinements of the domain event types using the same notational constructs
- The system operation is often a direct copy of the domain event type, however…
  - … new operations 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.scheduledPeriod.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)
```

- System operations must define who performs the operations (Actors)

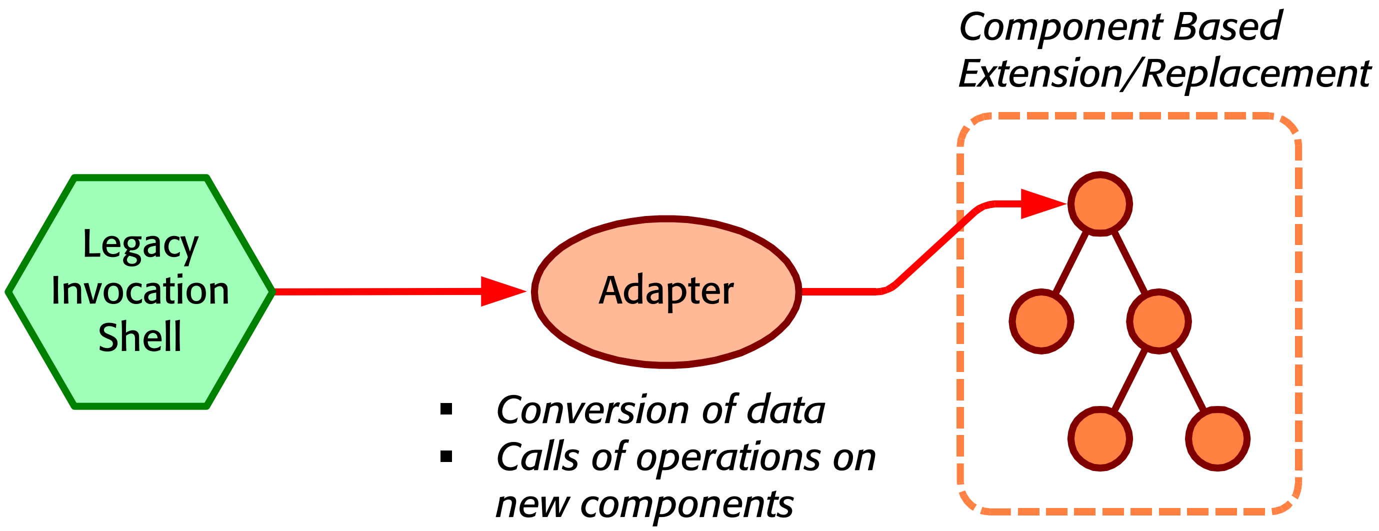
System Context Model for Domain Behavior



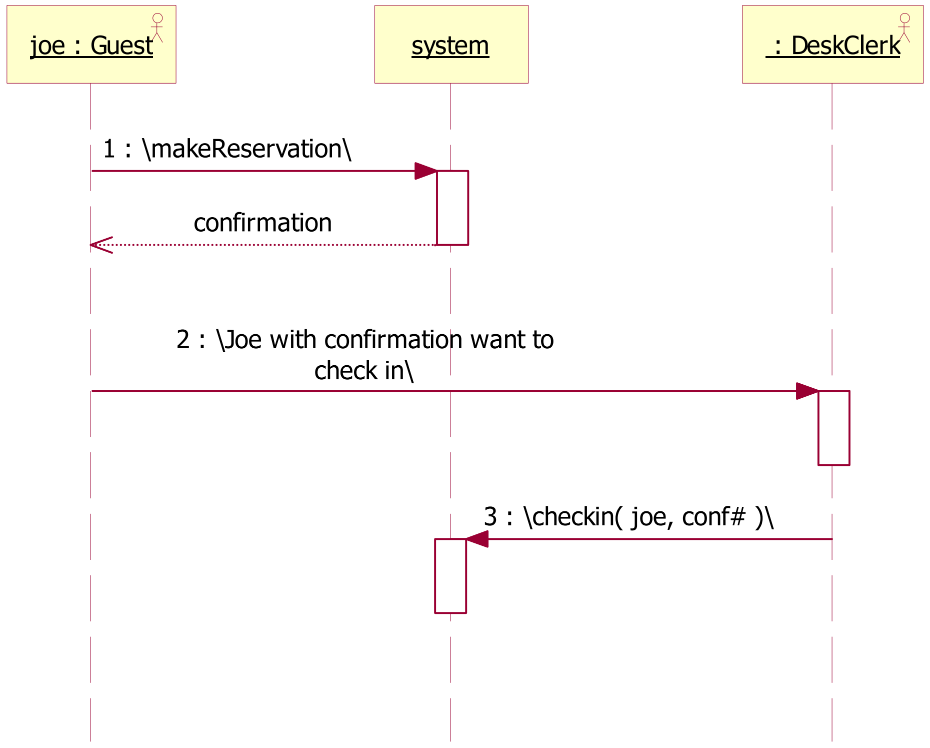
- We document the context of the system operations in System Context Models
- The diagram above shows the system operations derived from the domain event types and legacy functionality

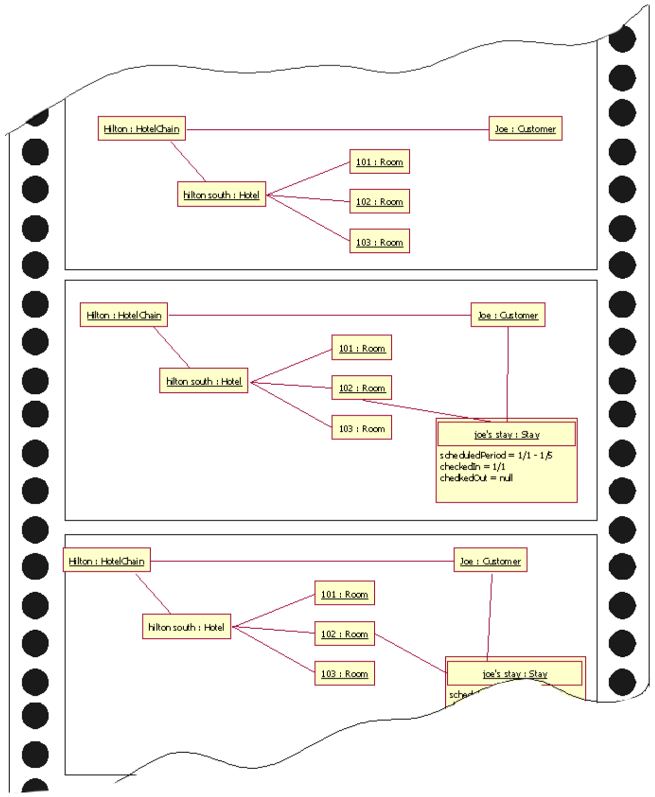
Representing Legacy Functionality

- Legacy functionality on the high level that remains reused is modeled through system operations
- The realization of system operations will act as an adaptor between the new and the legacy part of the system


Component Based Extension/Replacement
Legacy Invocation Shell
Adapter
Conversion of data
Calls of operations on new components

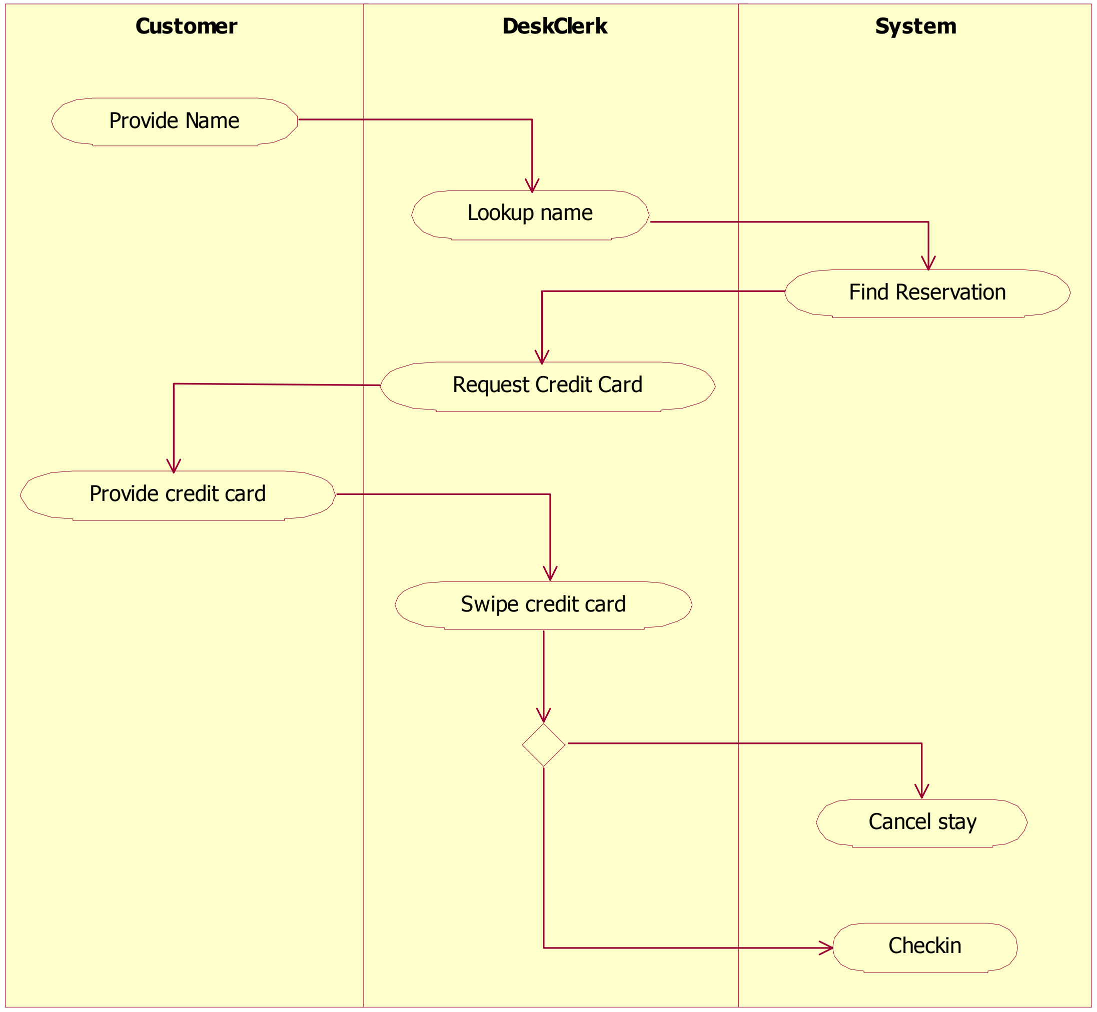
System Level Scenarios


joe : Guest
system
: DeskClerk
1 : \makeReservation\
confirmation
2 : \Joe with confirmation want to check in\
3 : \checkin(joe, conf#)\



- 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



- Activity diagrams can be used to enhance business context
- Useful for organization performance reviews

Specifying Models: HUTN++

- XML based model specification
- Inspired by OMG's HUTN
- Built for the MCC Meta Model for Node Implementations
- Simple schema that enables model creation through XML tools or simple editors

HUTN++ Example: Persistent Class

```
<persistentClass id=“Hotel" category="" name=“Hotel">
  <attributes>
    <attribute id=“Hotel_name" name="name" type="String">
      <description>The name represents……</description>
      <cardinality min="0" max="1"/>
    </attribute>
    <attribute id=“Hotel_address" name=“mailing" type=“Address">
      <description>The address is…</description>
      <cardinality min="0" max="1"/>
    </attribute>
  </attributes>
  <queries/>
</persistentClass>
```

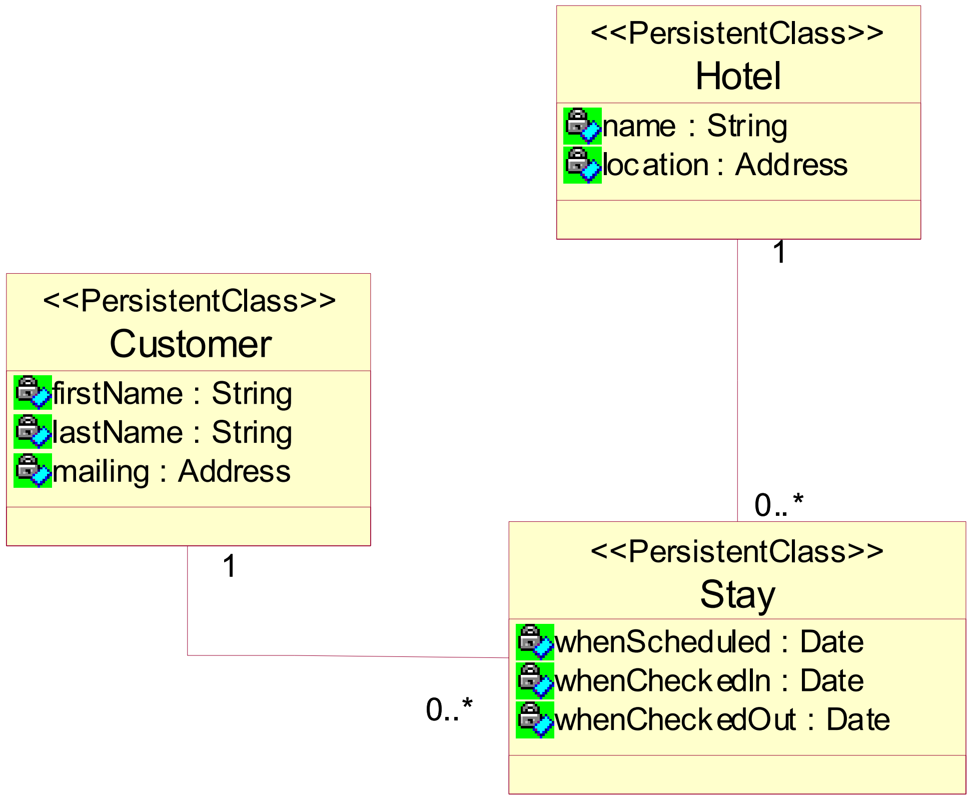
HUTN++ Example: Association

```
<association id=“Hotel2Stay" name=“Hotel2Stay">
    <roleA id=“Hotel2Stay_Stay" name=“stays" singularName=“stay”
      type=“Stay" navigable="true">
      <cardinality min="0" max="UNBOUNDED"/>
    </roleA>
    <roleB id=“Hotel2Stay_Hotel" name=“hotel" type=“Hotel“
      navigable="true">
      <cardinality min="1" max="1"/>
    </roleB>
 </association>
```

HUTN++ Example: Service

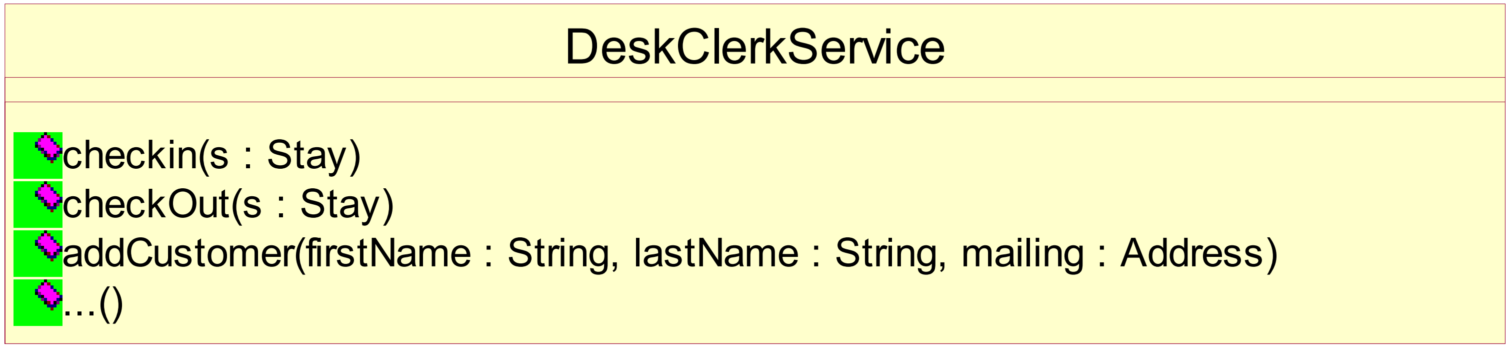
```
<service id=“DeskClerkService" name=“DeskClerkService" isStateful="false" isAbstract="false">
    <operation id="SupplierInterface_checkin" name=“checkin" isQuery="false">
      <arguments>
        <argument name=“stay" inputOutput="in" type=“Stay">
          <cardinality min="1" max="1"/>
        </argument>
     </arguments>
      <preCondition assumedCheckedByClient="false">
              -- The stay is not currently checked in
     </preCondition>
      <postCondition>
              -- The stay’s whenChecked In is set to …
     </postCondition>
    </operation>
  </service>
```

UML Tool Stereotype: Persistent Class



- To model the persistent types in a standard UML tool, we use stereo types
- <<Persistent Class>> tags a class to be a persistent type
- Associations are not stereotyped
- Primitive types are defined with the stereotype <<PrimitiveType>>

UML Tool Stereotype: Service Class


DeskClerkService
checkin(s : Stay)
checkOut(s : Stay)
addCustomer(firstName : String, lastName : String, mailing : Address)
...()

- To model the services in a standard UML tool, we use stereotype
- <<Service>> tags a class to be a service

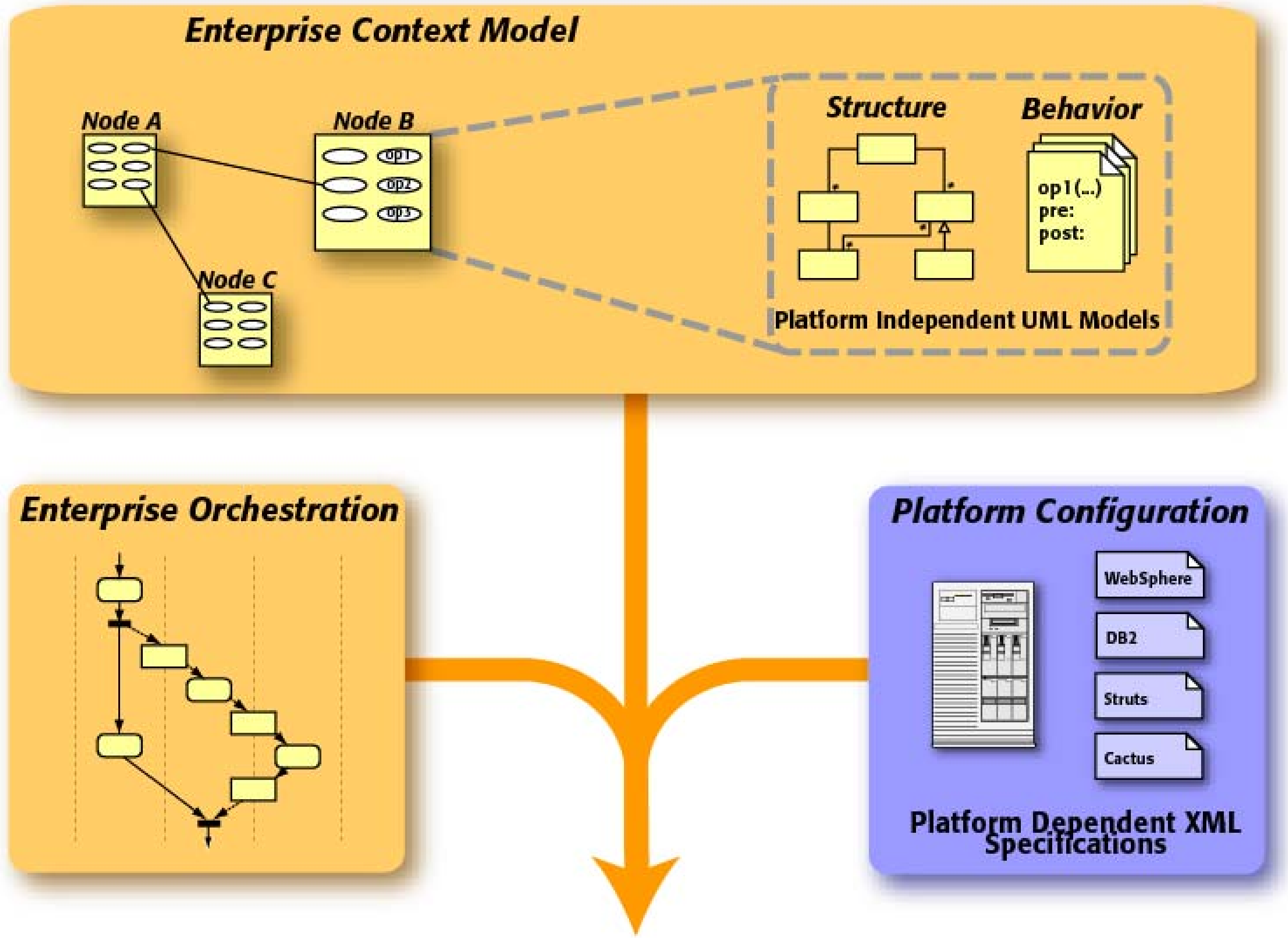
Platform Configuration Specification

- To optimize the PSM, we need to know how the application is intended to be deployed
- J2EE:
  - Which application server?
  - What database?
  - How many machines
  - Network configuration?
- .NET
  - How many servers?
  - How powerful are the servers?
- Etc.

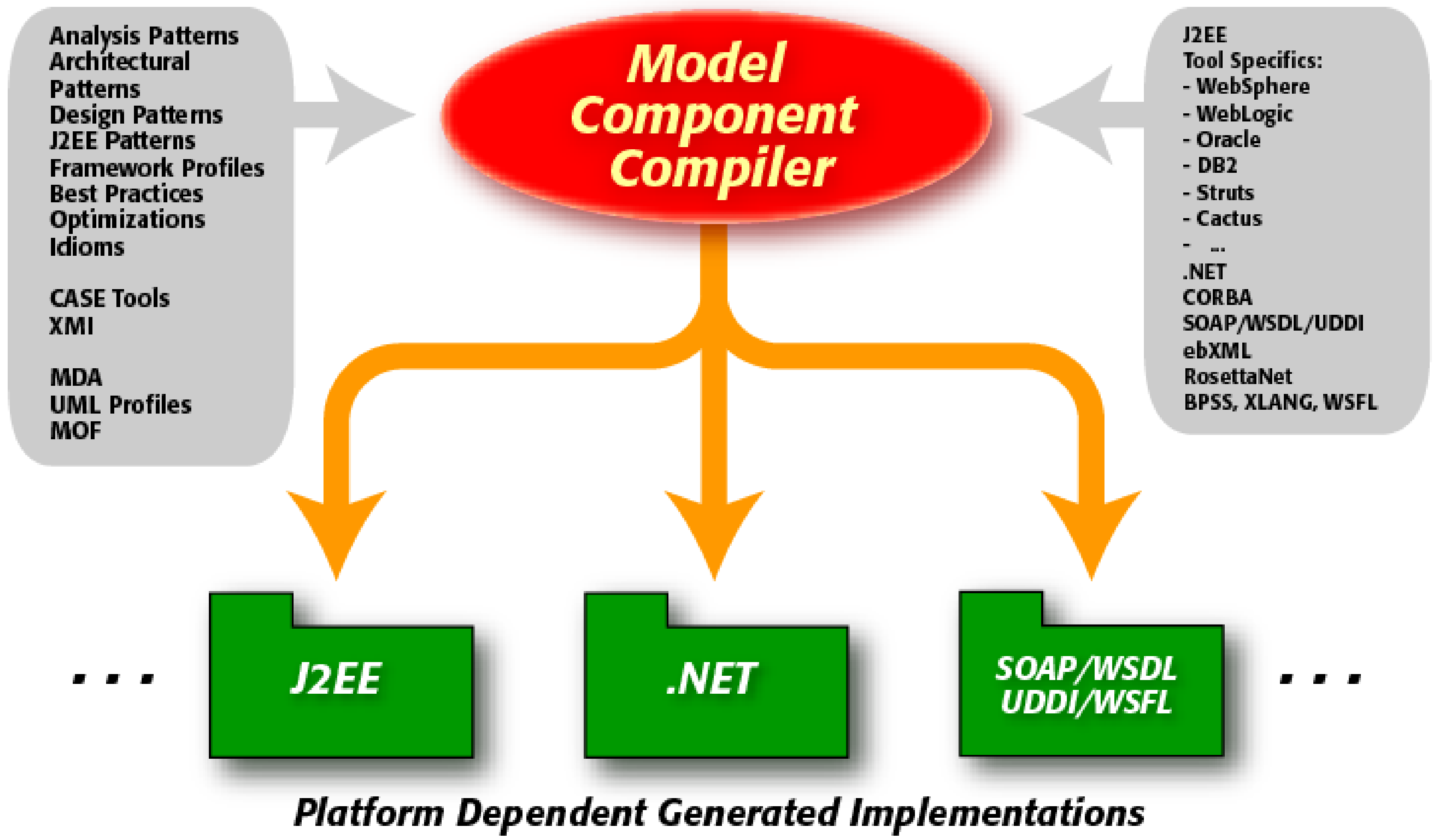
What is Generated?

- Persistence Model options
  - Entity beans?
  - Data Access Objects?
  - Purely a virtual model for queries?
  - …
- Services options
  - Session Beans?
  - Web Services?
  - Simple Java Beans?
  - …
- Queries
  - EJBQL?
  - SQL
  - …

MDA Tool Used: MCC And Its Input Models



MCC: Transformation and Output



Adapters

- Adapters are used in two flows of invocations:
  - From legacy to the new components
  - From new components to the legacy
- In both cases, the adaptors use the interfaces of new components
  - Interfaces generated by an MDA tool
  - Implementation to the legacy side done manually
- The adaptors can execute:
  - In the same address spaces
    - E.g. using JNI for C/Java interoperability
  - In different address spaces, decoupling the systems
    - E.g. using Web Services or MOM

Conclusion

- Domain models are excellent start for modernization of legacy systems
  - Enable alignment of the software architecture with the business
  - Provide context for understanding of legacy systems
  - Enable identification of extension points
  - Enable use of MDA tools for generation of extensions and replacement components
- Future work:
  - Integration of MDA with the reengineering tools