

# UML Components

John Daniels  
Syntropy Limited  
john@syntropy.co.uk

© Syntropy Limited 2001 All rights reserved

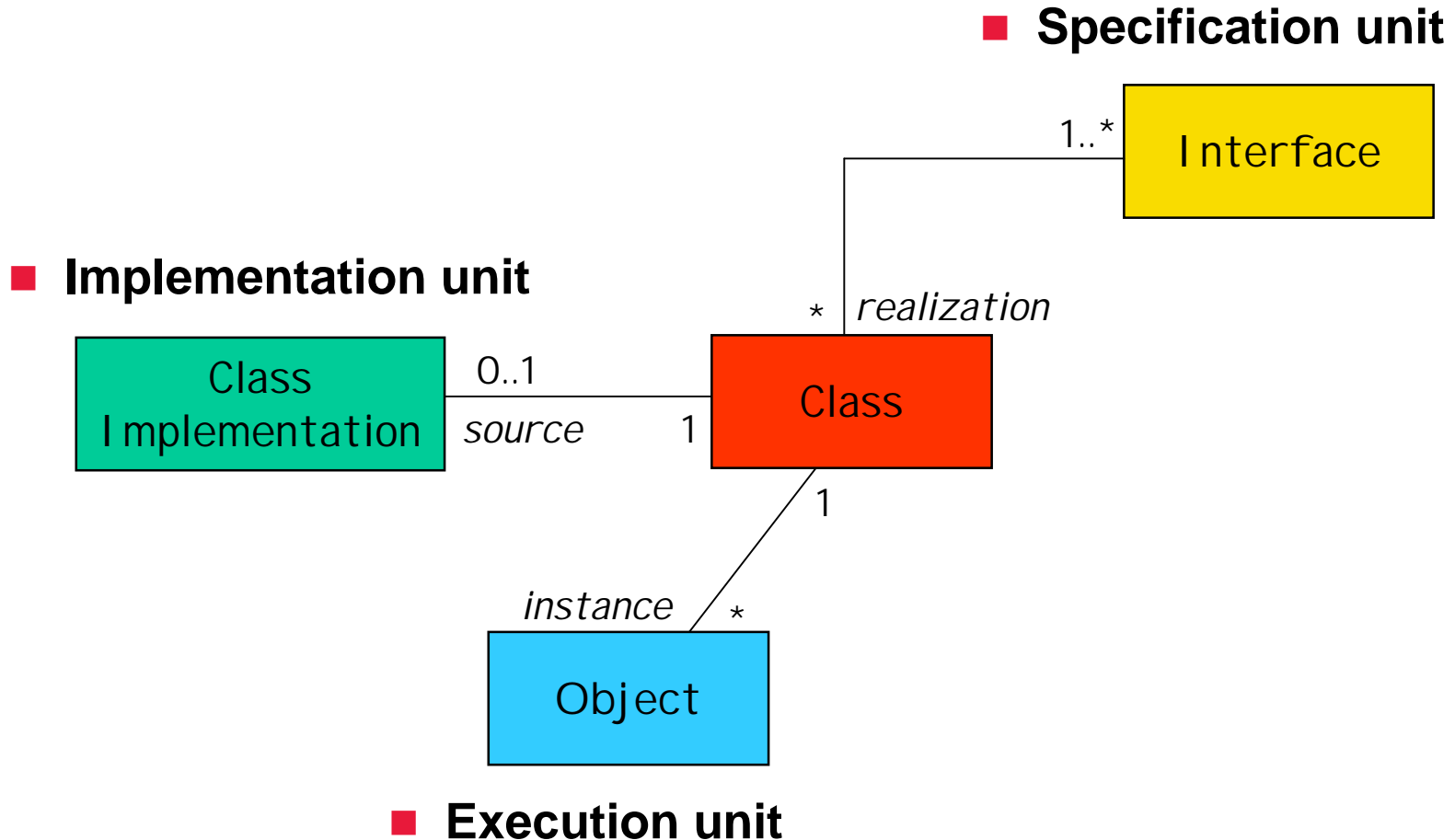
- Aspects of a component
- A process for component specification
- Implications for the UML

# Aspects of a component

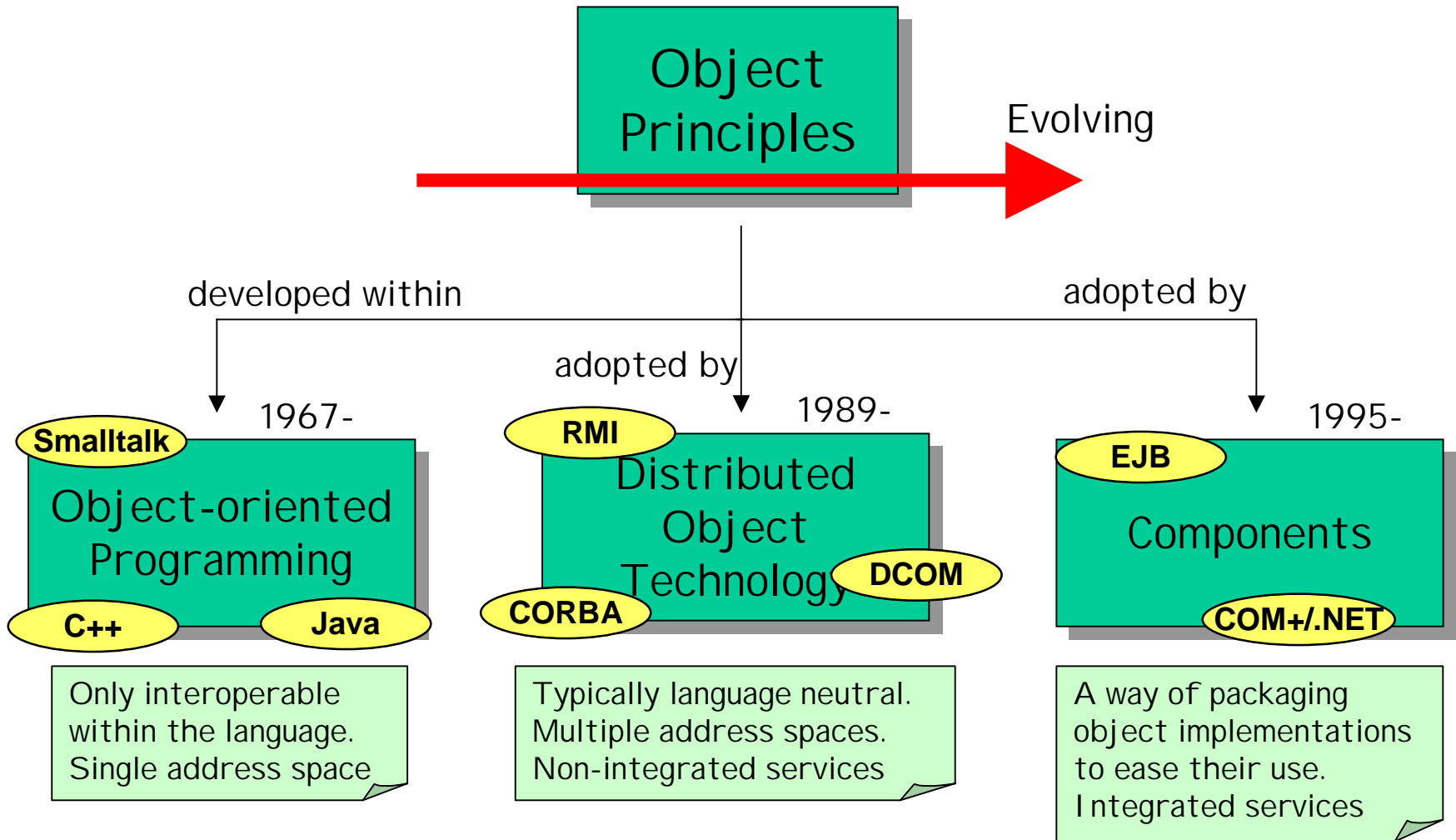
## Unified Modeling Language

- The UML is a standardised language for describing the structure and behaviour of things
- UML emerged from the world of object-oriented programming
- UML has a set of notations, mostly graphical
- There are tools that support some parts of the UML

# Aspects of an Object



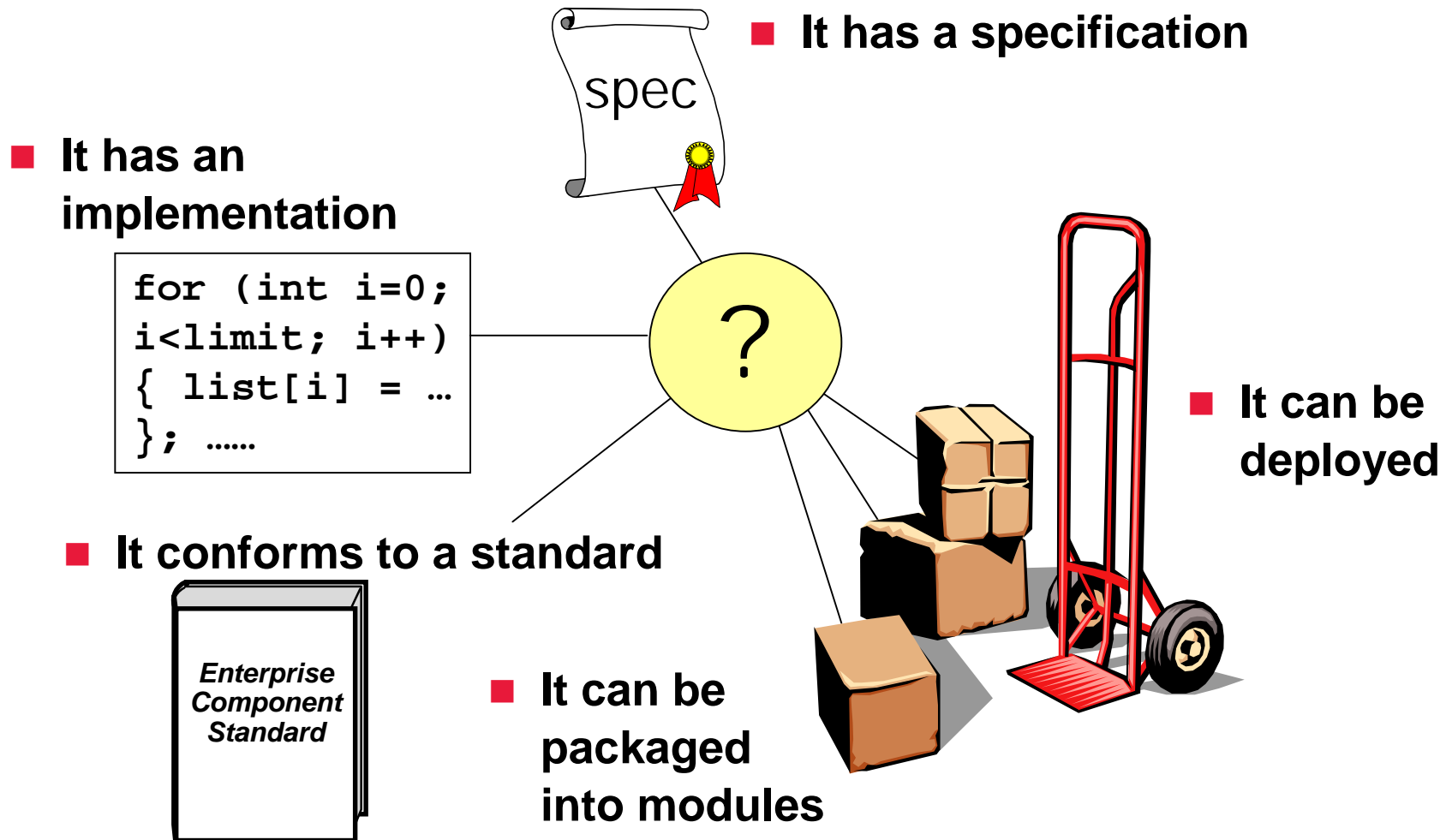
# Components in context



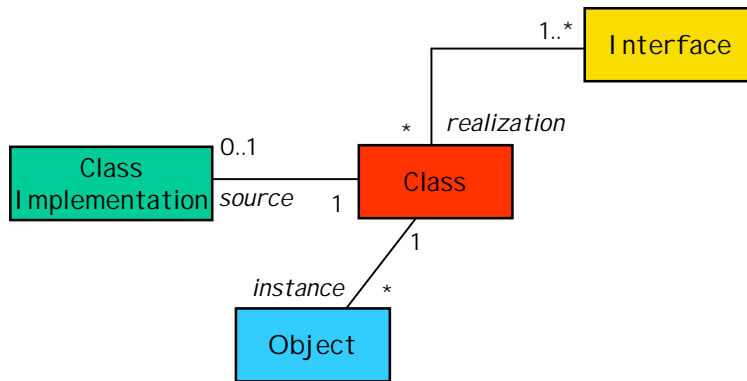
# Component standard features

- **Component Model:**
  - defined set of services that support the software
  - set of rules that must be obeyed in order to take advantage of the services
- Simple programming model, no need to design/know about the infrastructure
- Services include:
  - remote access, transactions, persistent storage, security
  - typically use services by configuring not programming

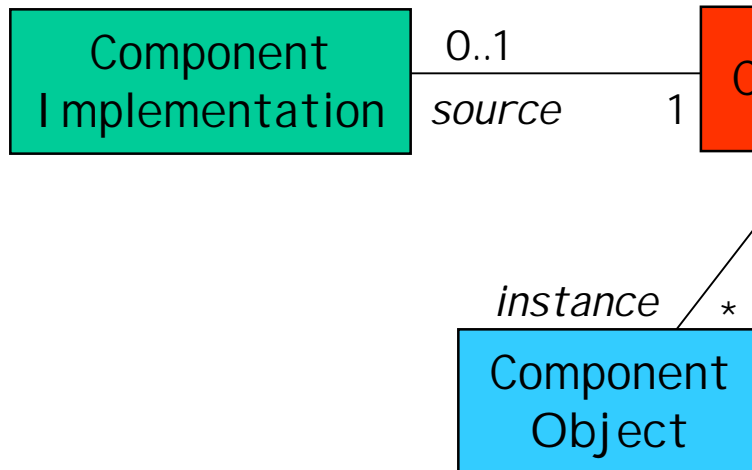
# Aspects of a component



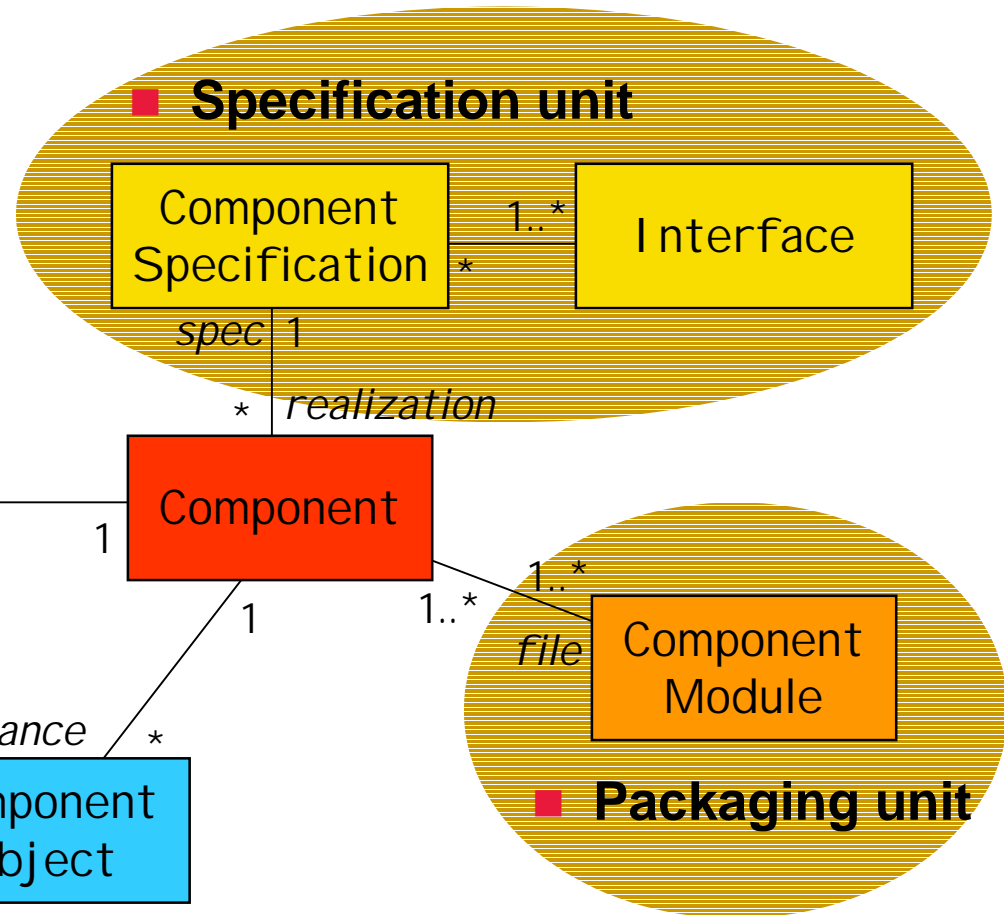
# Component forms



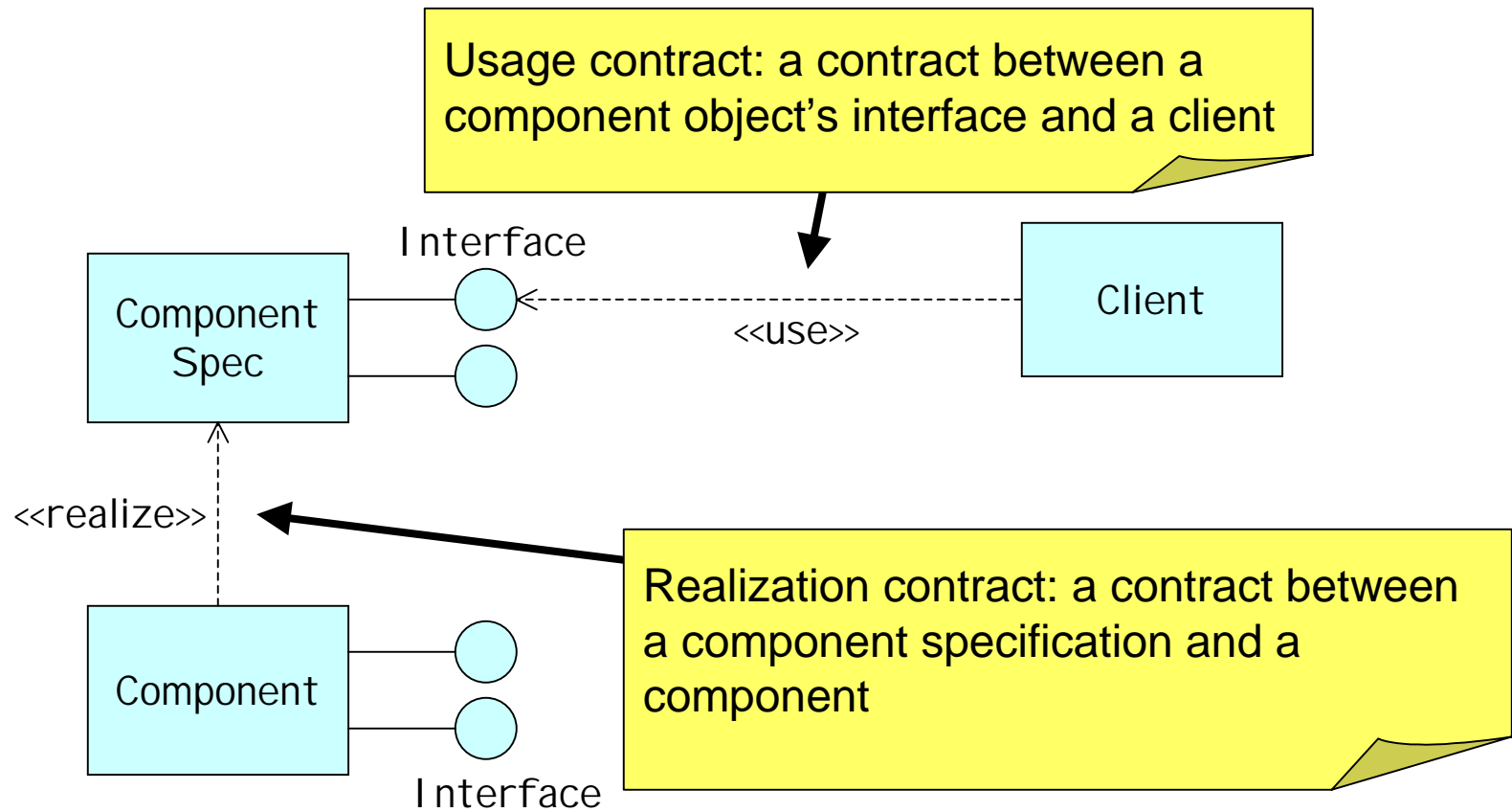
## ■ Implementation unit



## ■ Execution unit



# Two distinct contracts



# Interface specification

I OrderMgt
placeOrder(custNum, prodNum, quan) numOfOrders(custNum): Integer

I ProductMgt
reserveStock(prodNum, quan) availableStock(prodNum): Integer

We could specify placeOrder() like this:

“The number of orders for the customer is increased by one and a reserveStock message is sent to the component supporting the I ProductMgt interface”

# Separation of specification concerns

The client cares about this - it affects the subsequent result of `numOfOrders()`. Therefore it is part of the usage contract

“The number of orders for the customer is increased by one and a `reserveStock` message is sent to the component supporting the `IProductMgt` interface”

The `IOrderMgt` client does not care about this - but the implementer does. Therefore it is part of the realization contract

# Interfaces versus Component Specs

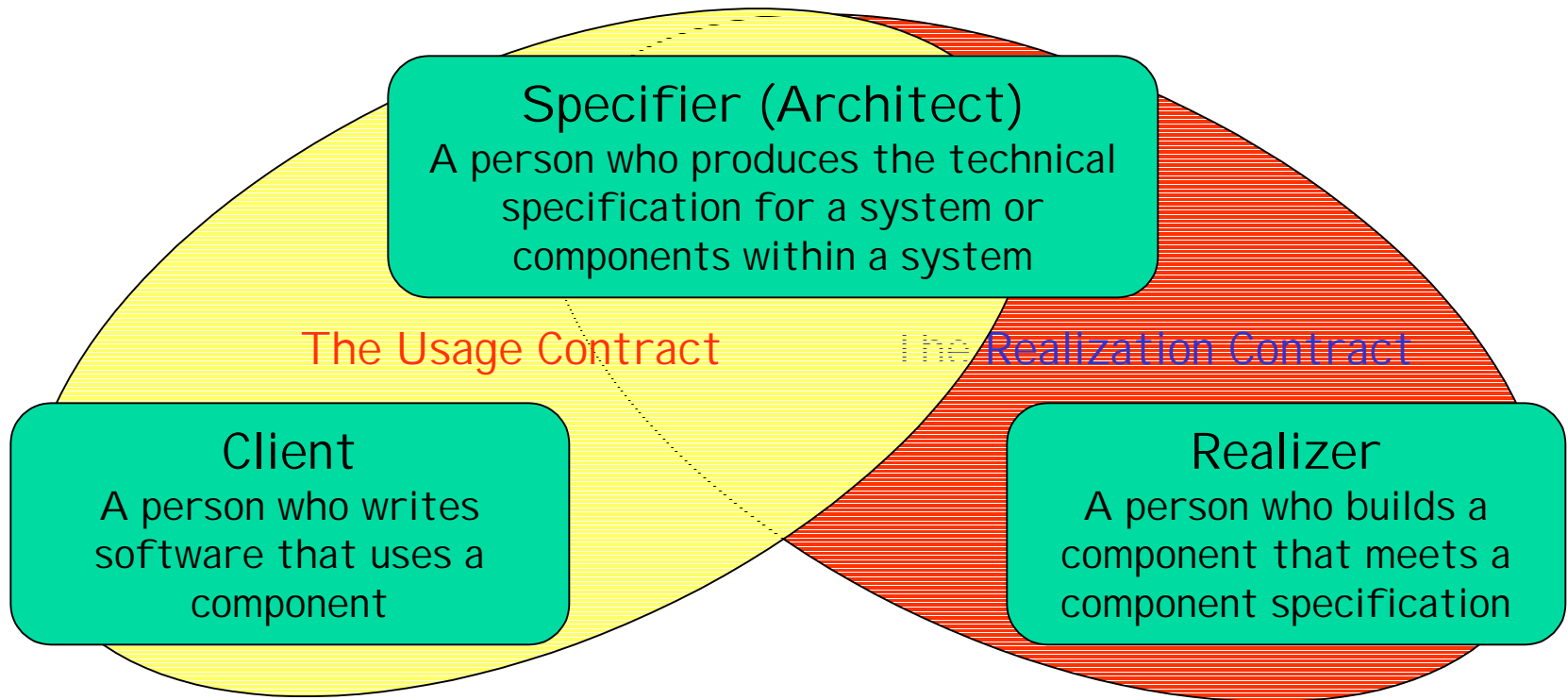
## Component Interface

- Represents the **usage** contract
- Provides a list of operations
- Defines an underlying logical information model specific to the interface
- Specifies how operations affect or rely on the information model
- Describes local effects only

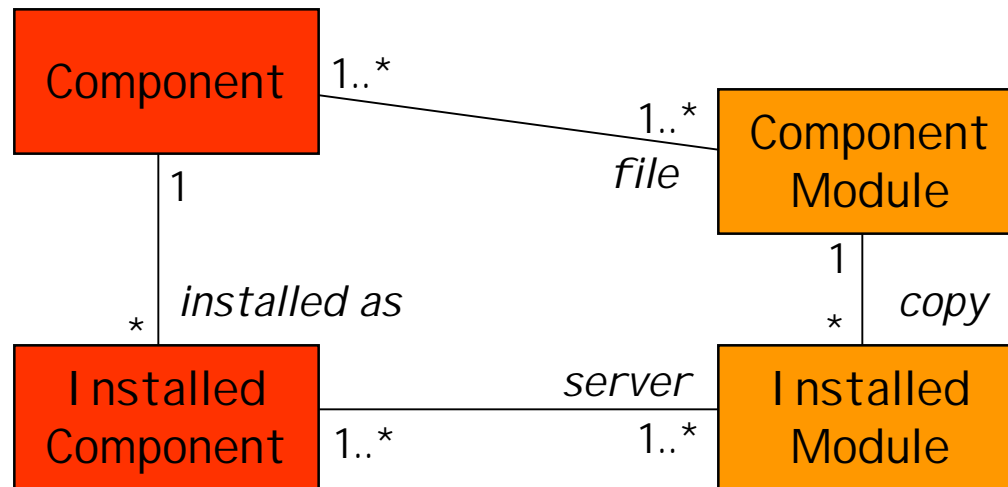
## Component Specification

- Represents the **realization** contract
- Provides a list of supported interfaces
- Defines the run-time unit
- Defines the relationships between the information models of different interfaces
- Specifies how operations should be implemented in terms of usage of other interfaces

# Contracts and roles



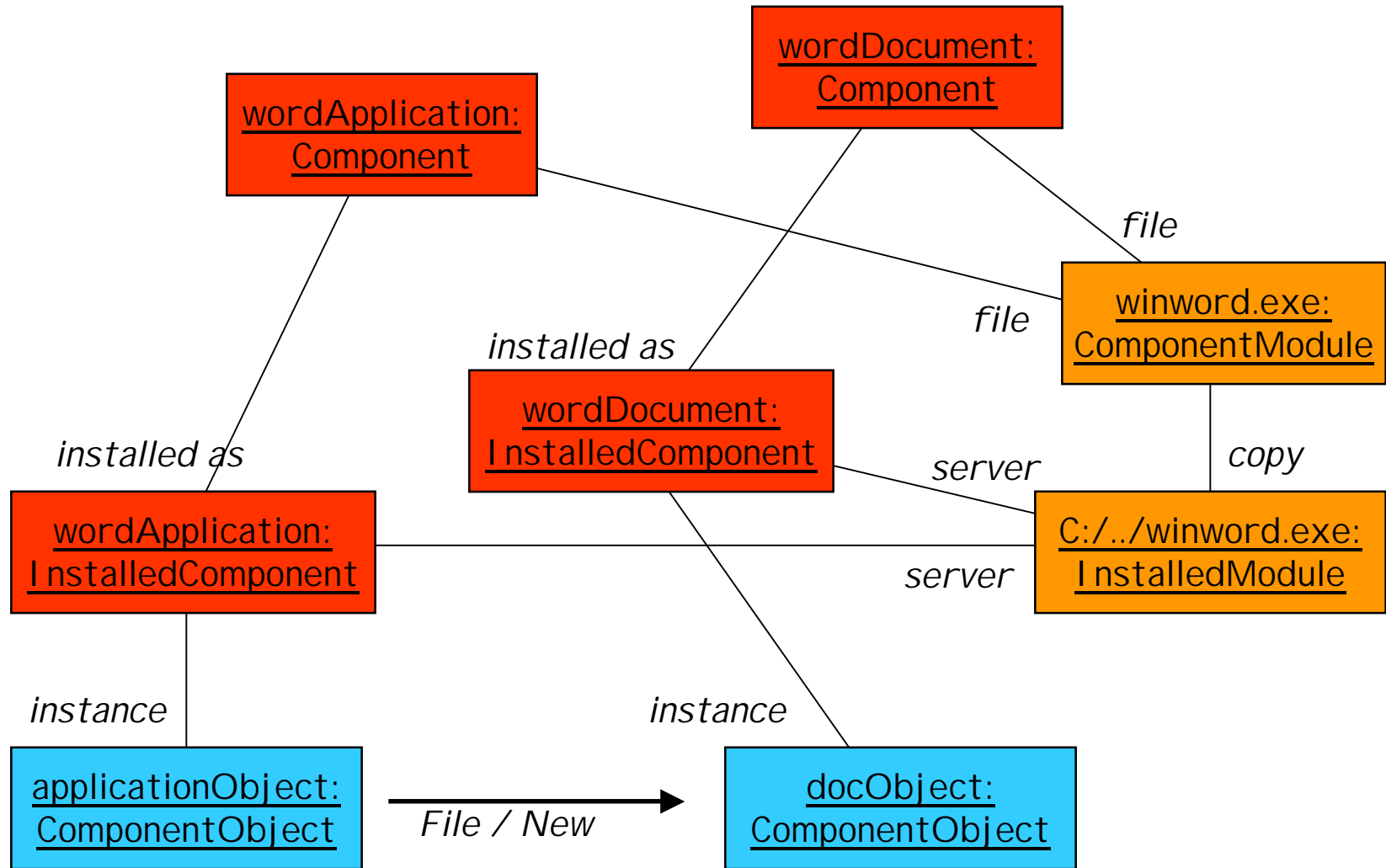
# Component deployment



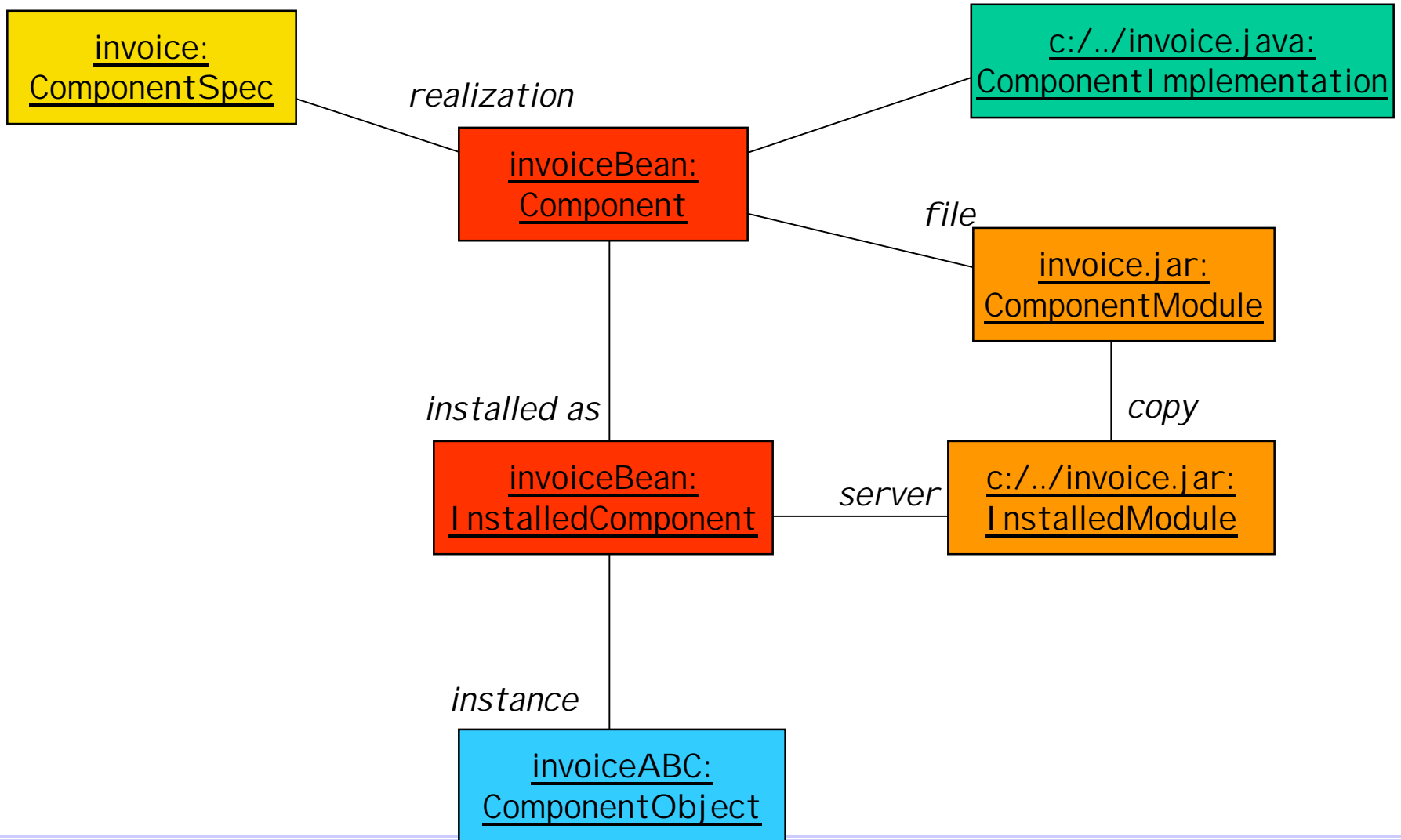
■ Registration unit

■ Installation unit

# Example - Microsoft Word™

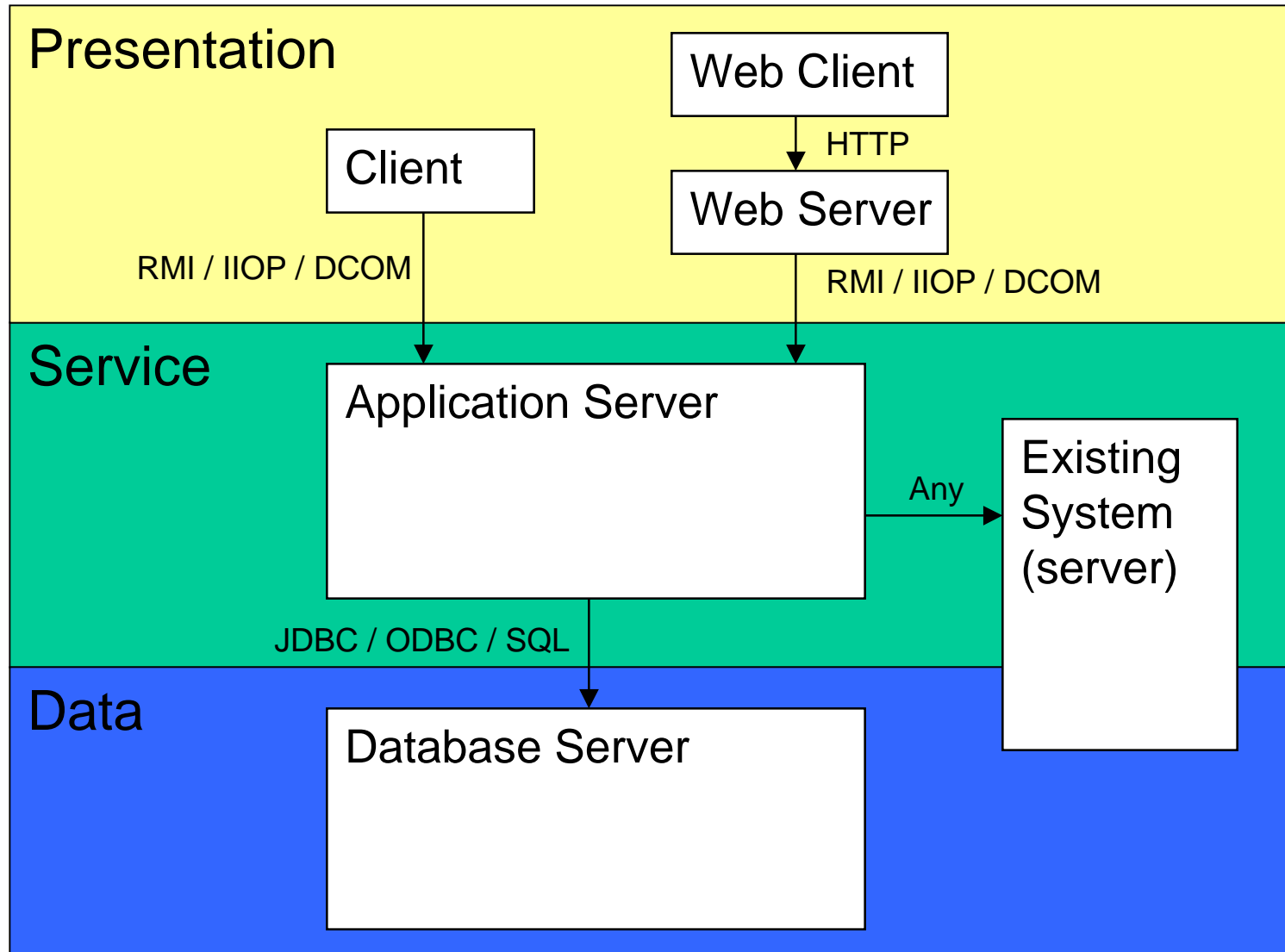


# Example - Enterprise Java Beans

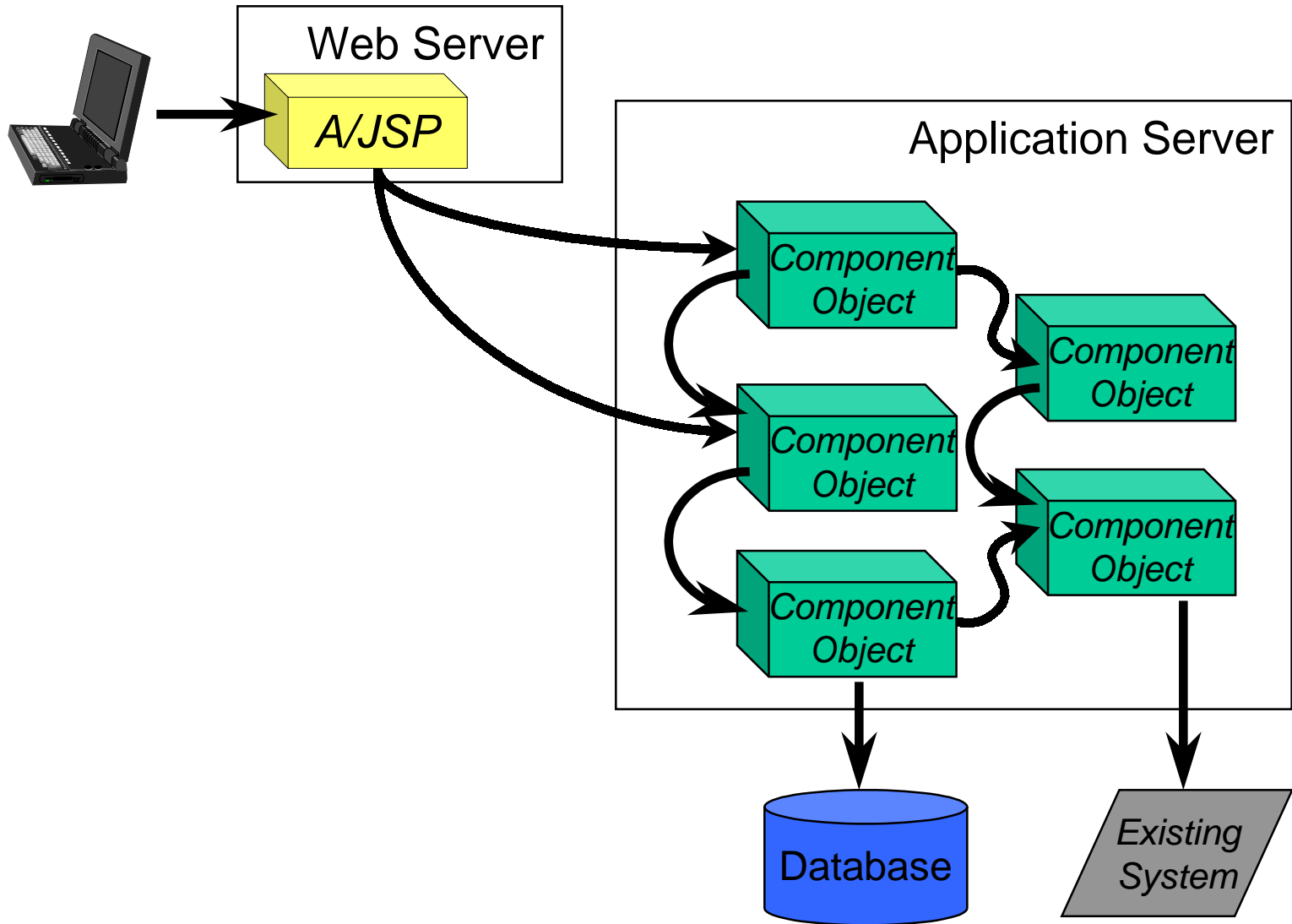


# A process for component specification

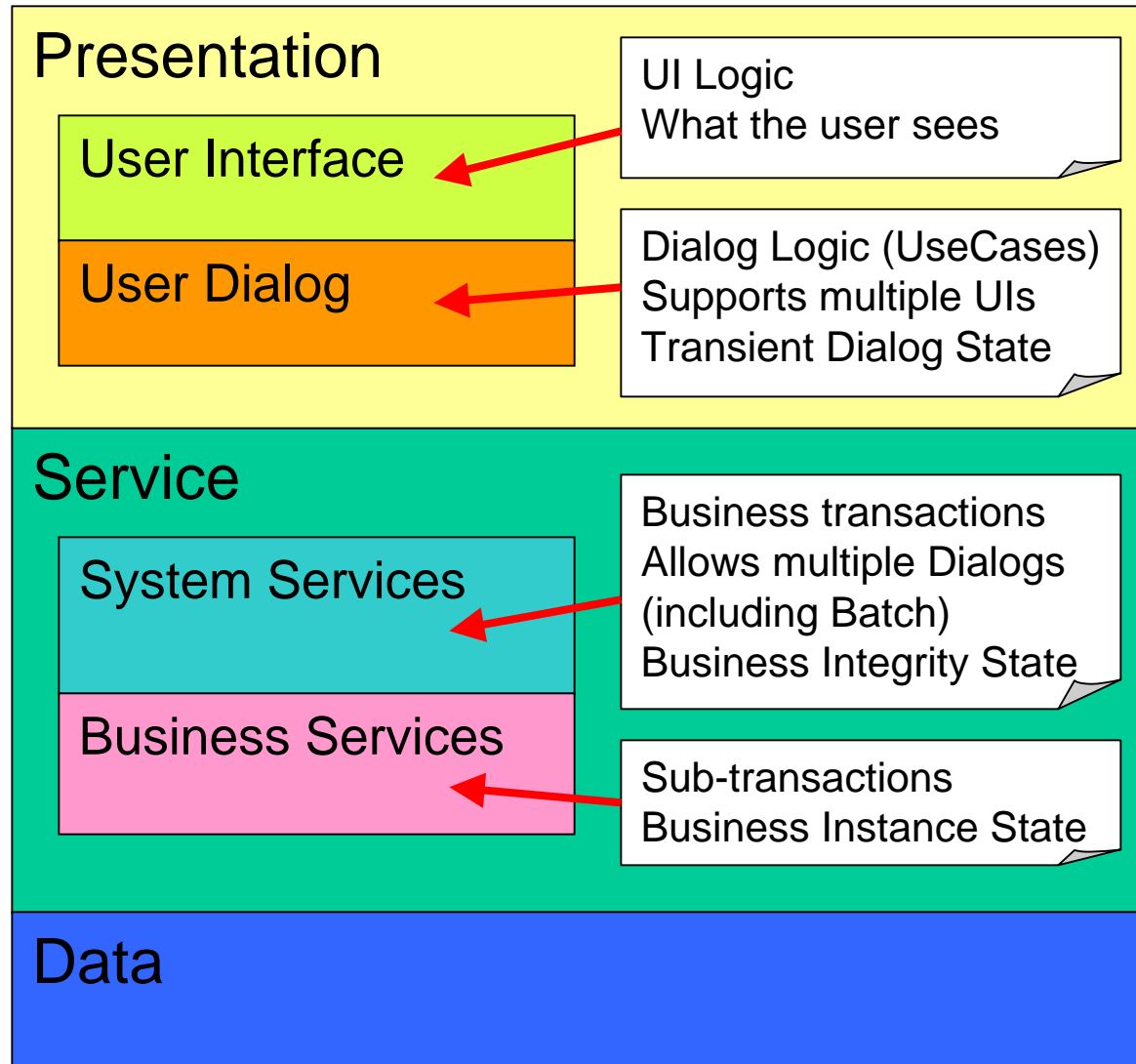
# Application Architecture Layers



# Application Blueprint



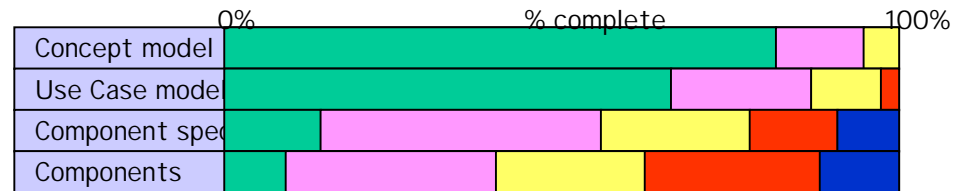
# Finer-Grain Application Layers



# Management and Development Processes

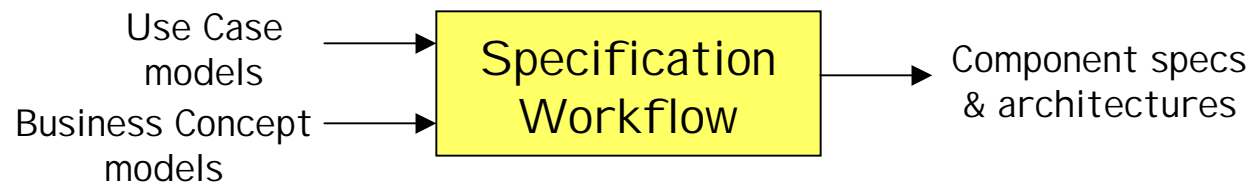
- Management Processes

- Schedule work and plan deliveries
- Allocate resources
- Monitor progress
- Control risk

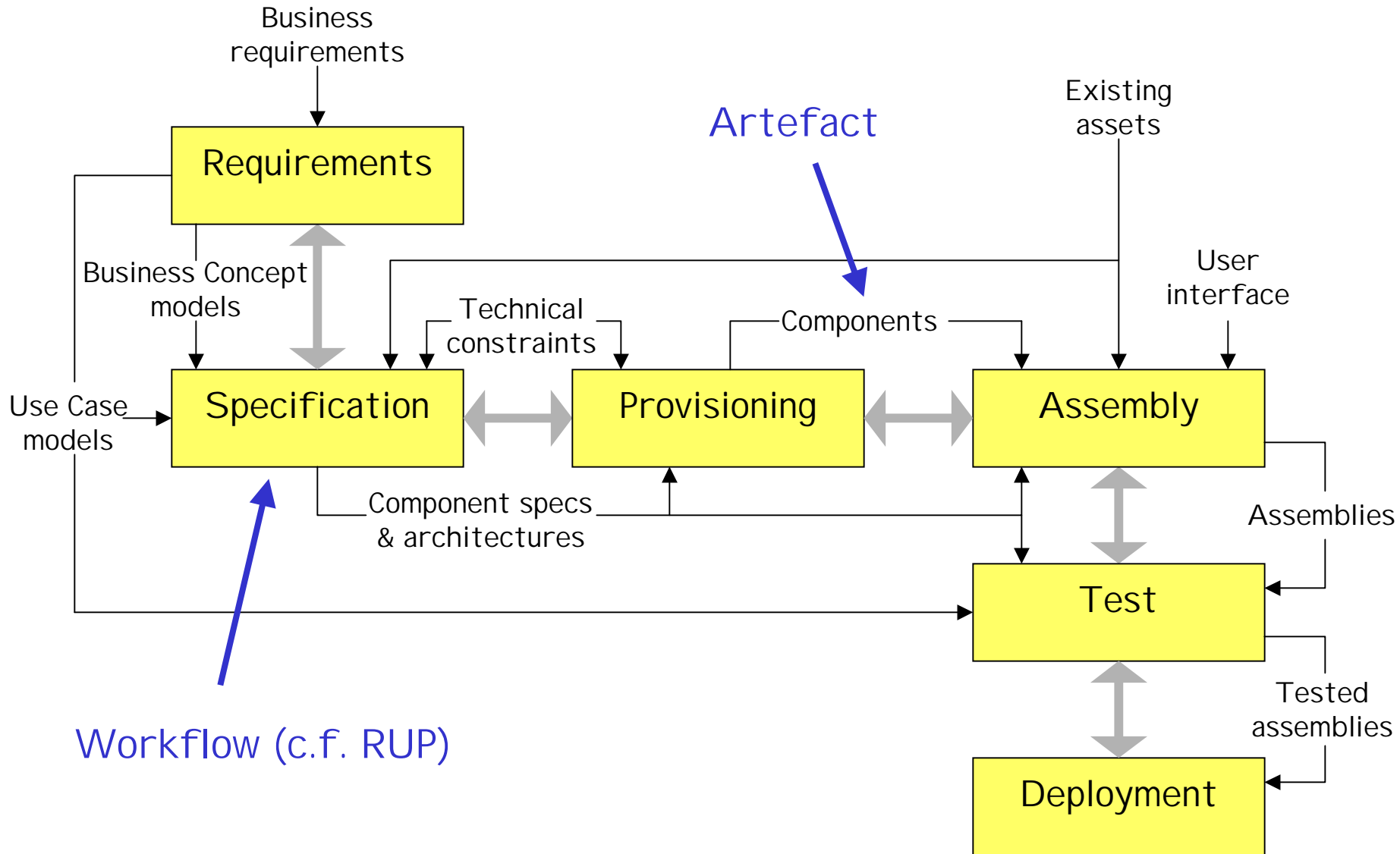


- Development Processes

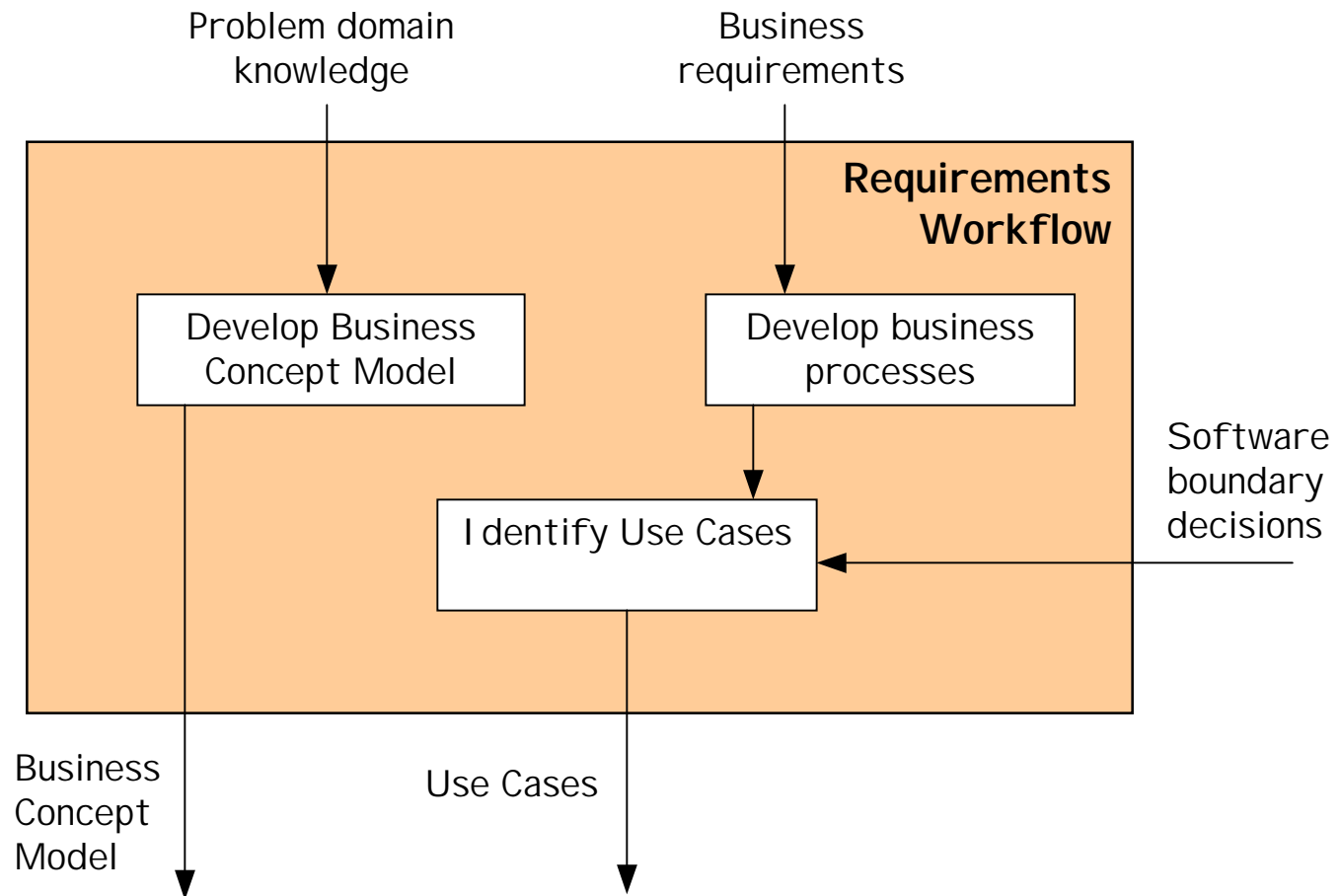
- Create working software from requirements
- Focus on software development artifacts
- Described independently of the management process
- Defines ordering constraints and dependencies
- Organized into Workflows



# Workflows in the development process

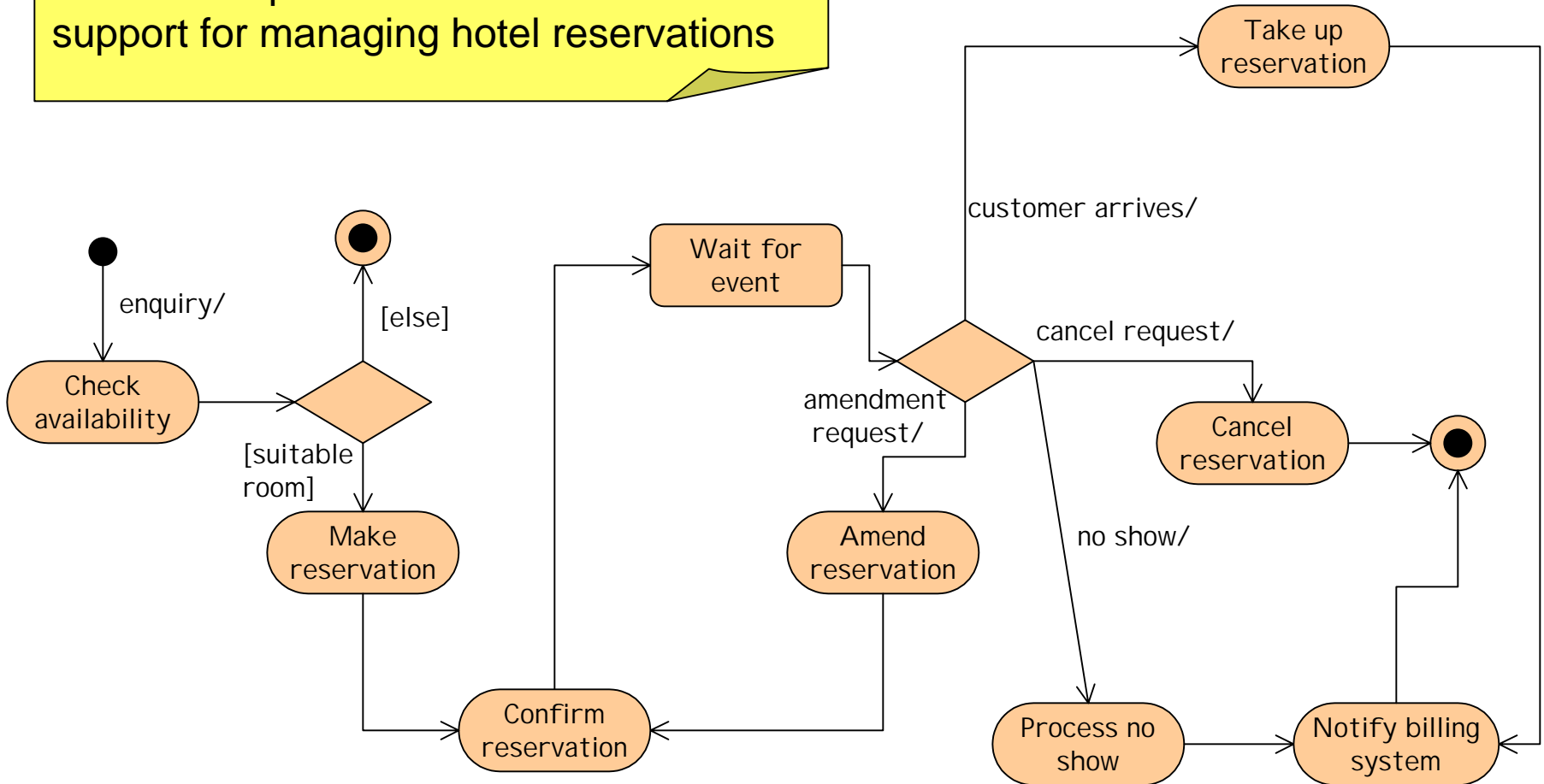


# The Requirements Workflow

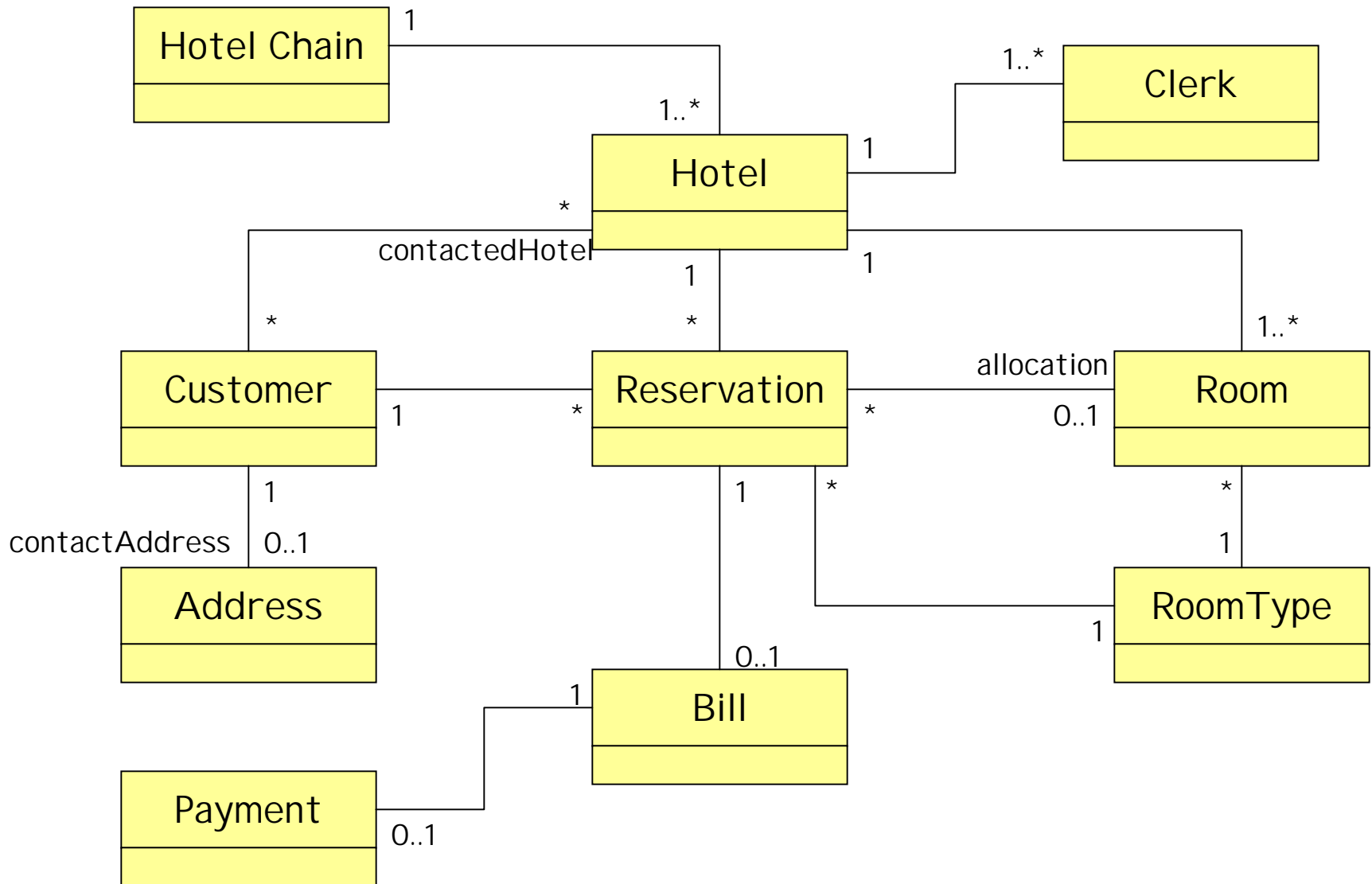


# Business process

We want to provide some automated support for managing hotel reservations

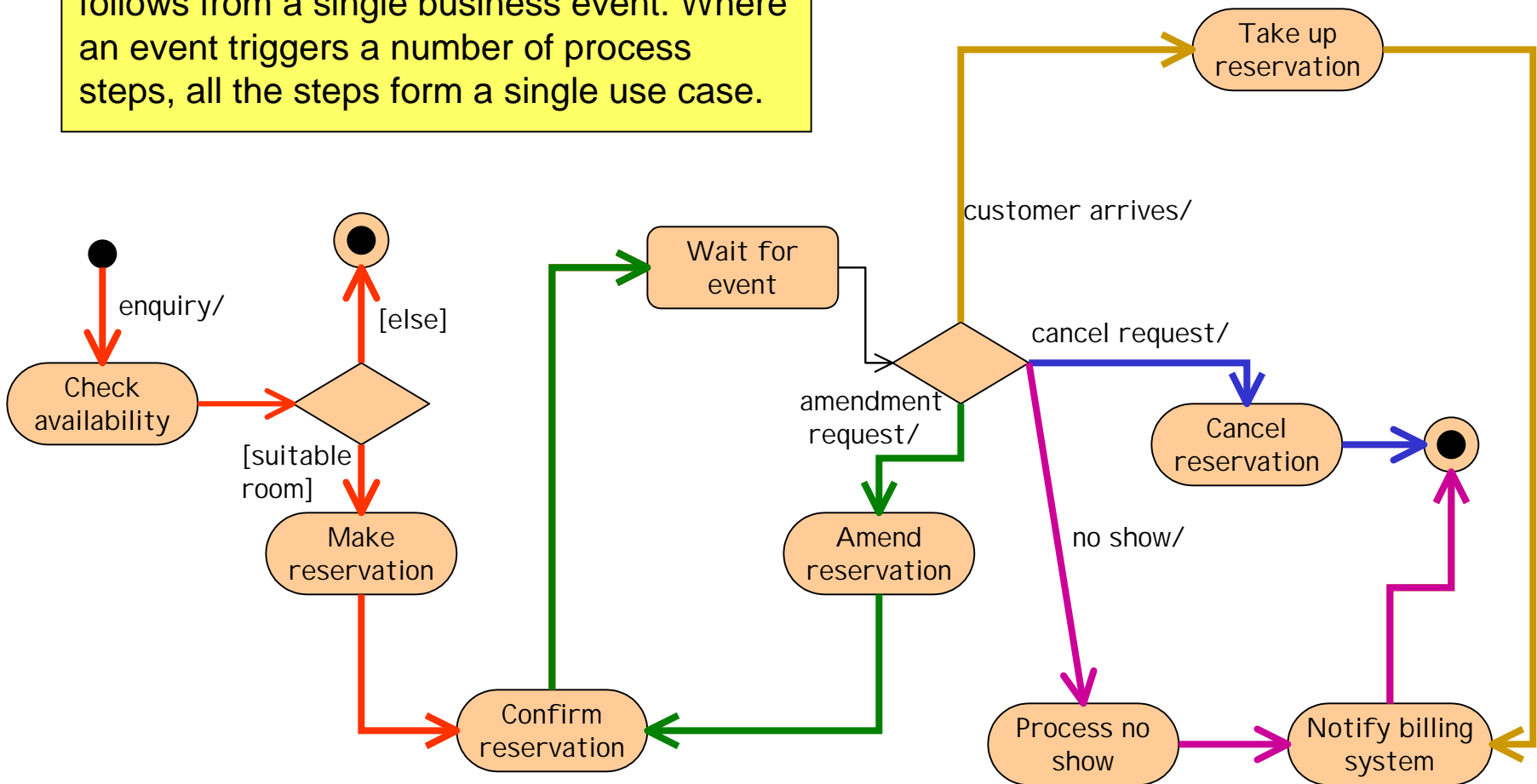


# Business Concept Model

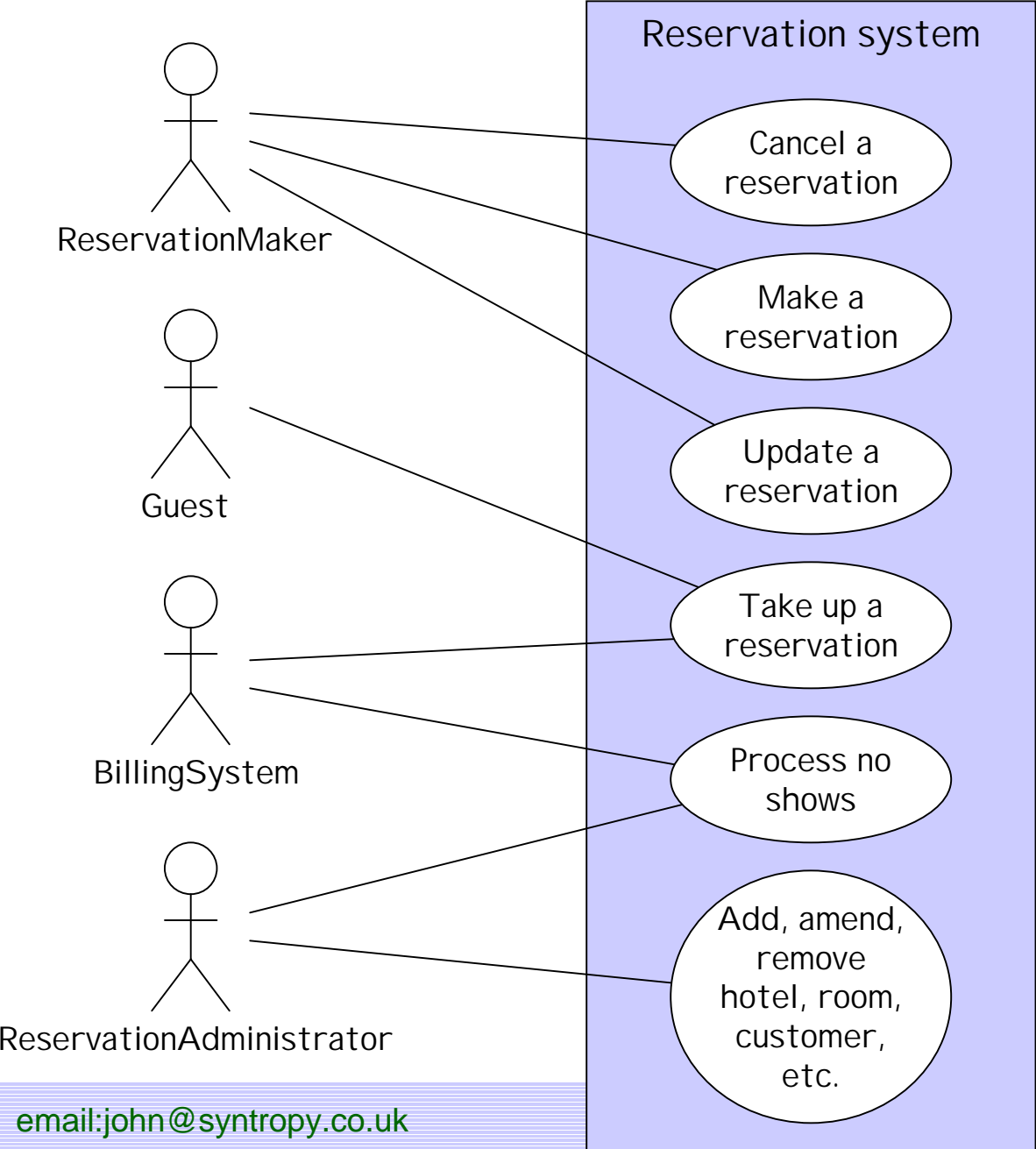


# Identify Use Cases

A use case describes the interaction that follows from a single business event. Where an event triggers a number of process steps, all the steps form a single use case.



# Use Case diagram



Name	Make a Reservation
Initiator	Reservation Maker
Goal	Reserve a room at a hotel

Steps

or  
Extension  
Points

### Main success scenario

1. Reservation Maker asks to make a reservation
2. Reservation Maker selects hotel, dates and room type
3. System provides availability and price
4. Reservation Maker agrees to proceed
5. Reservation Maker provides name and postcode
6. Reservation Maker provides contact email address
7. System makes reservation and gives it a tag
8. System reveals tag to Reservation Maker
9. System creates and sends confirmation by email

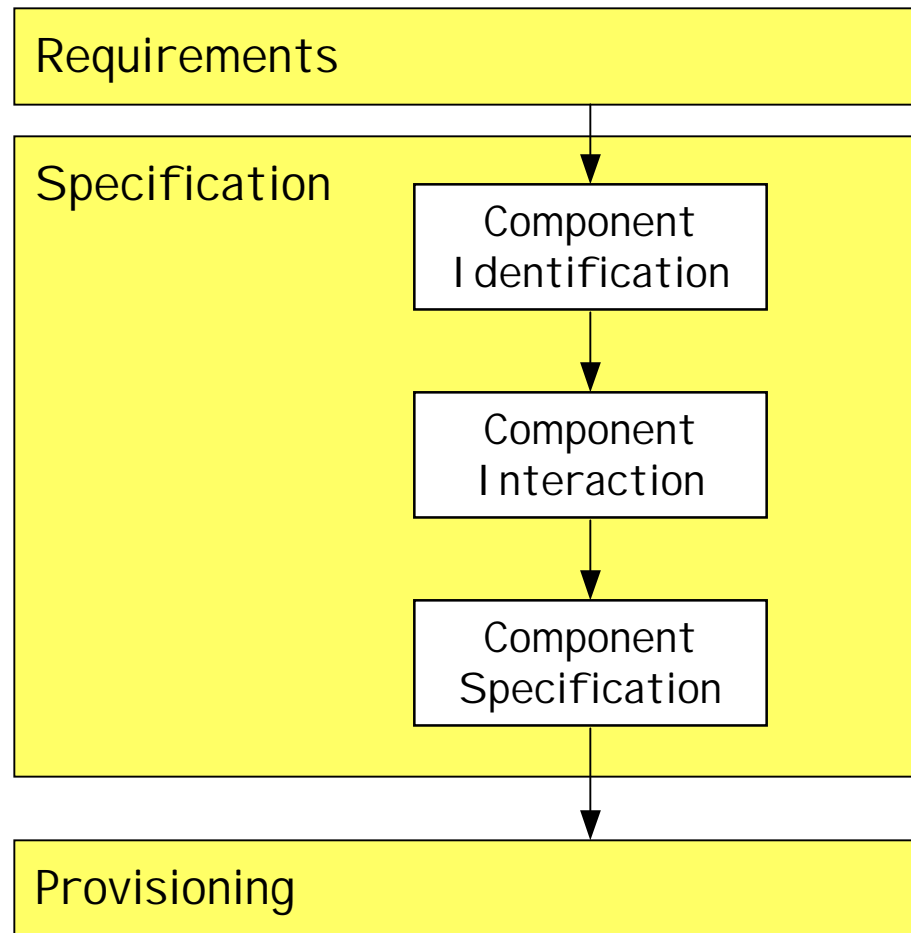
### Extensions

3. Room Not Available
  - a) System offers alternative dates and room types
  - b) Reservation Maker selects from alternatives
6. Customer already on file
  - a) Resume 7

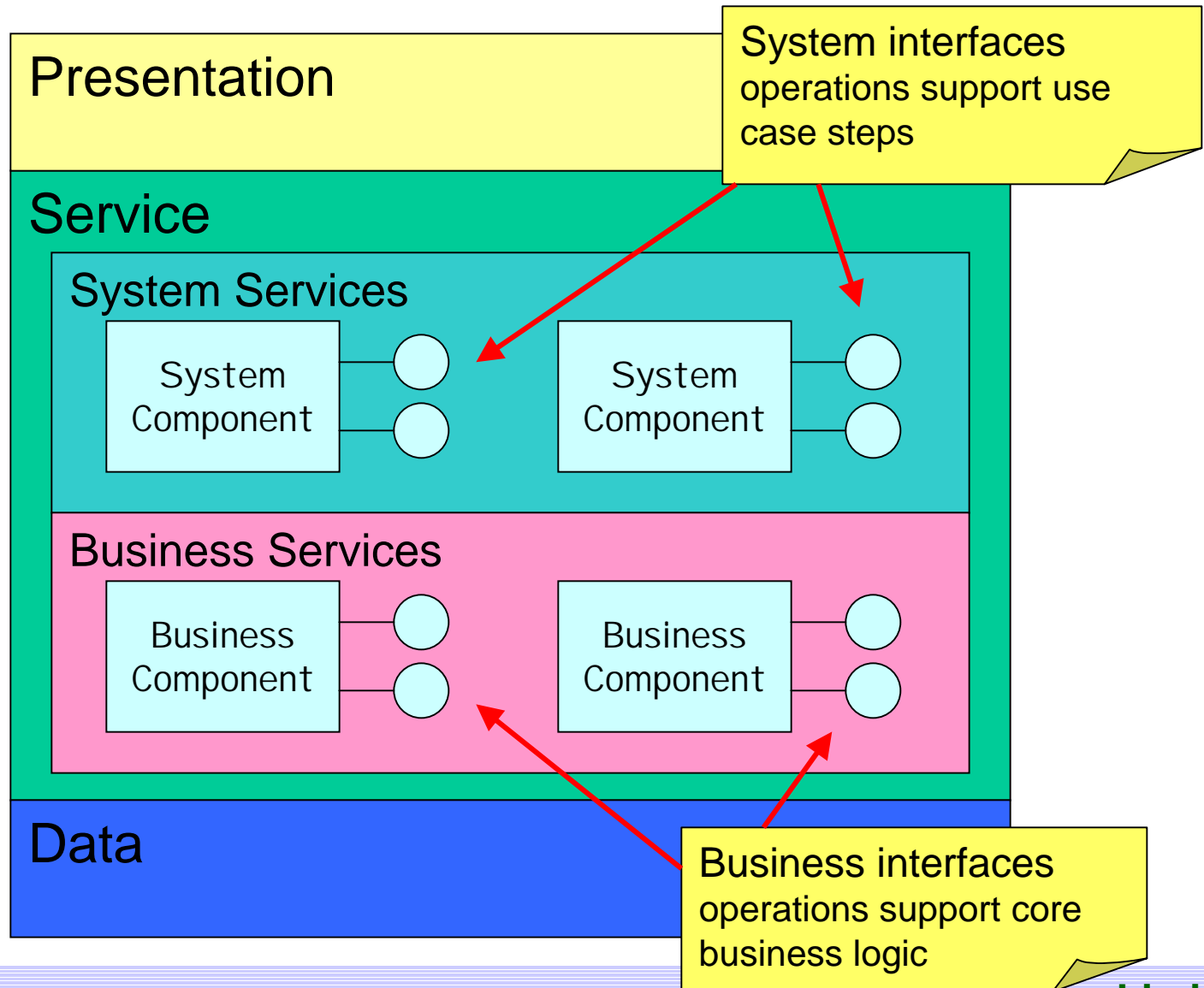
### Alternatives

Use an informal "Alternatives" section if you don't want to specify the detail required for an extension

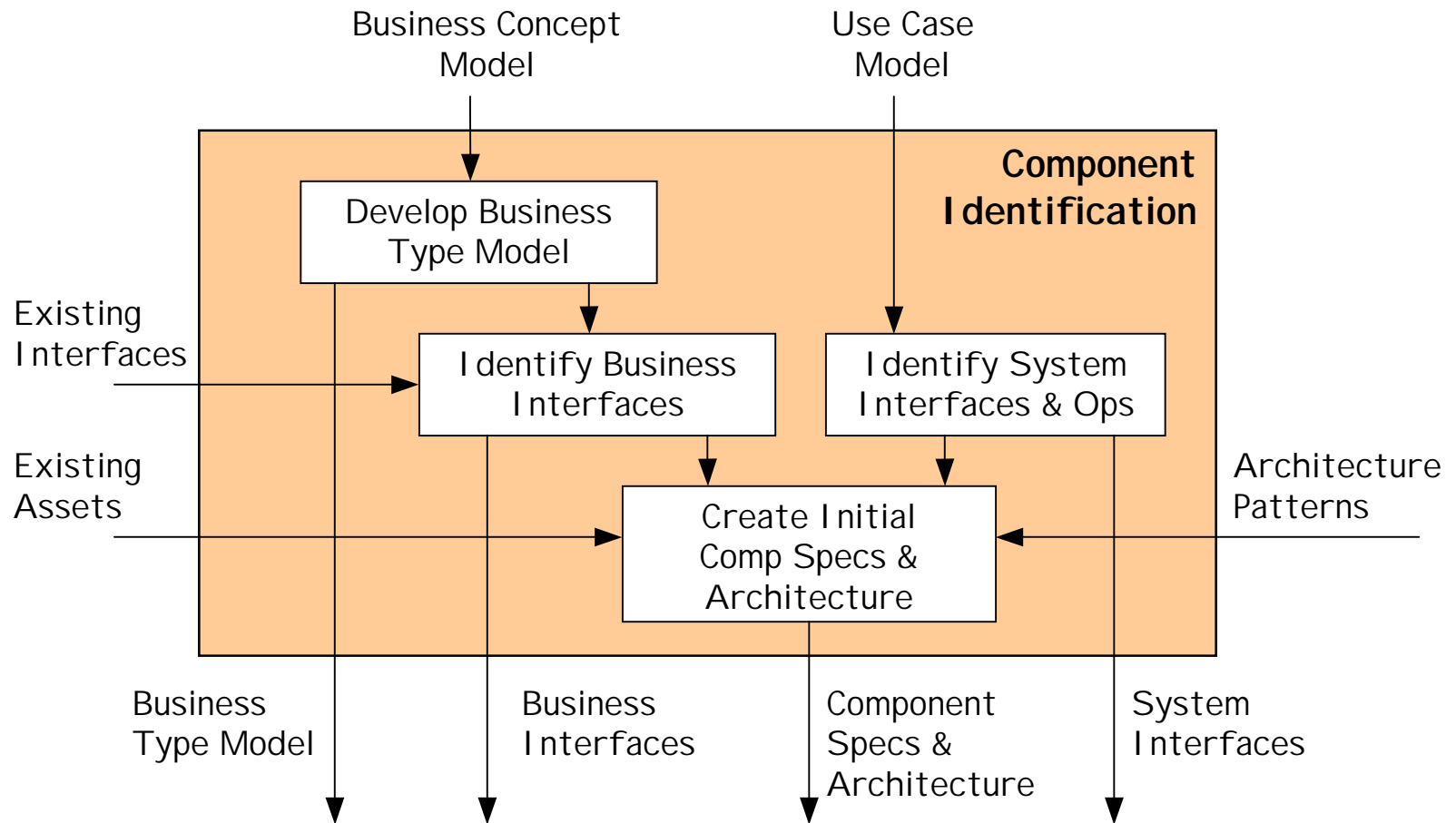
# The Specification Workflow



# Components in the service layers

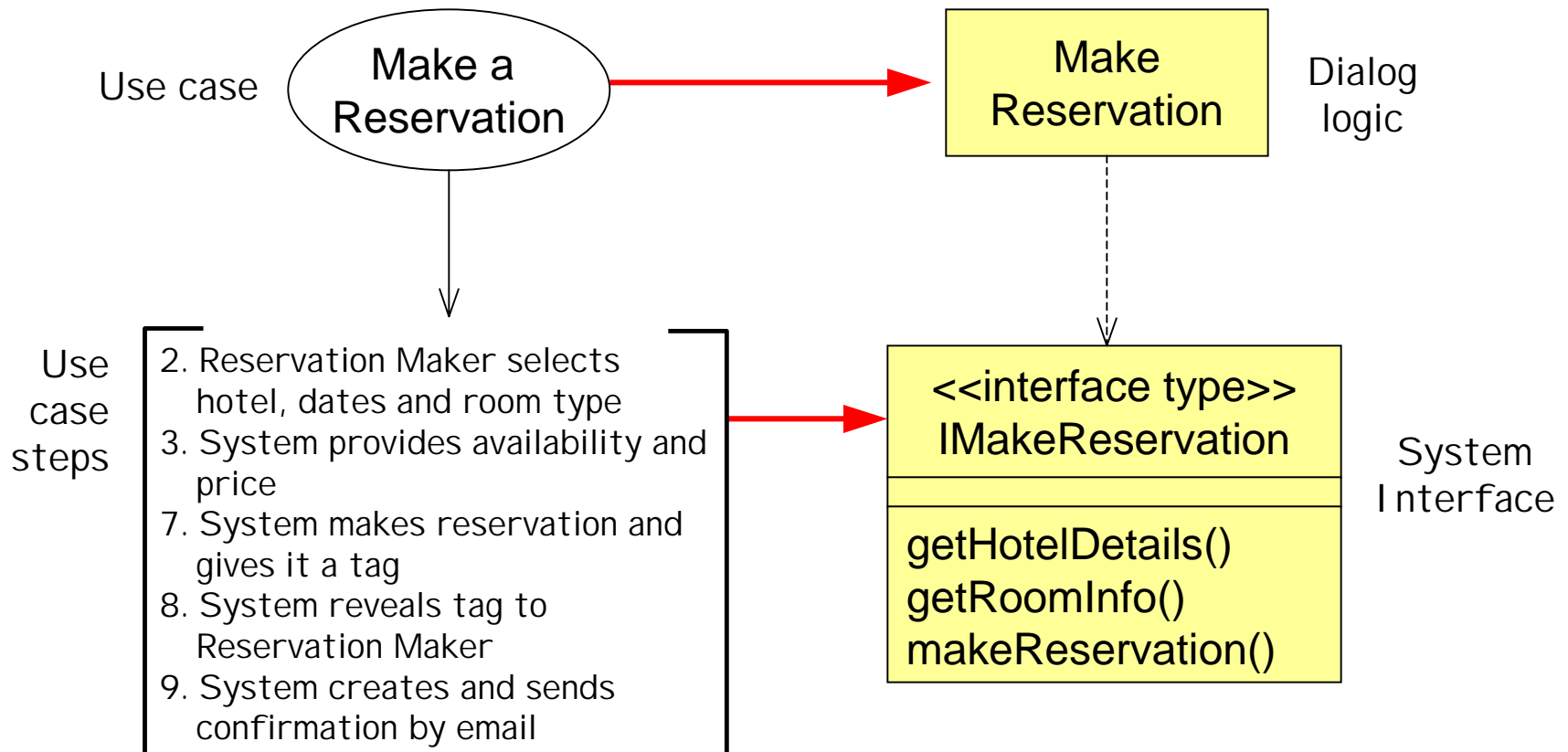


# Component Identification

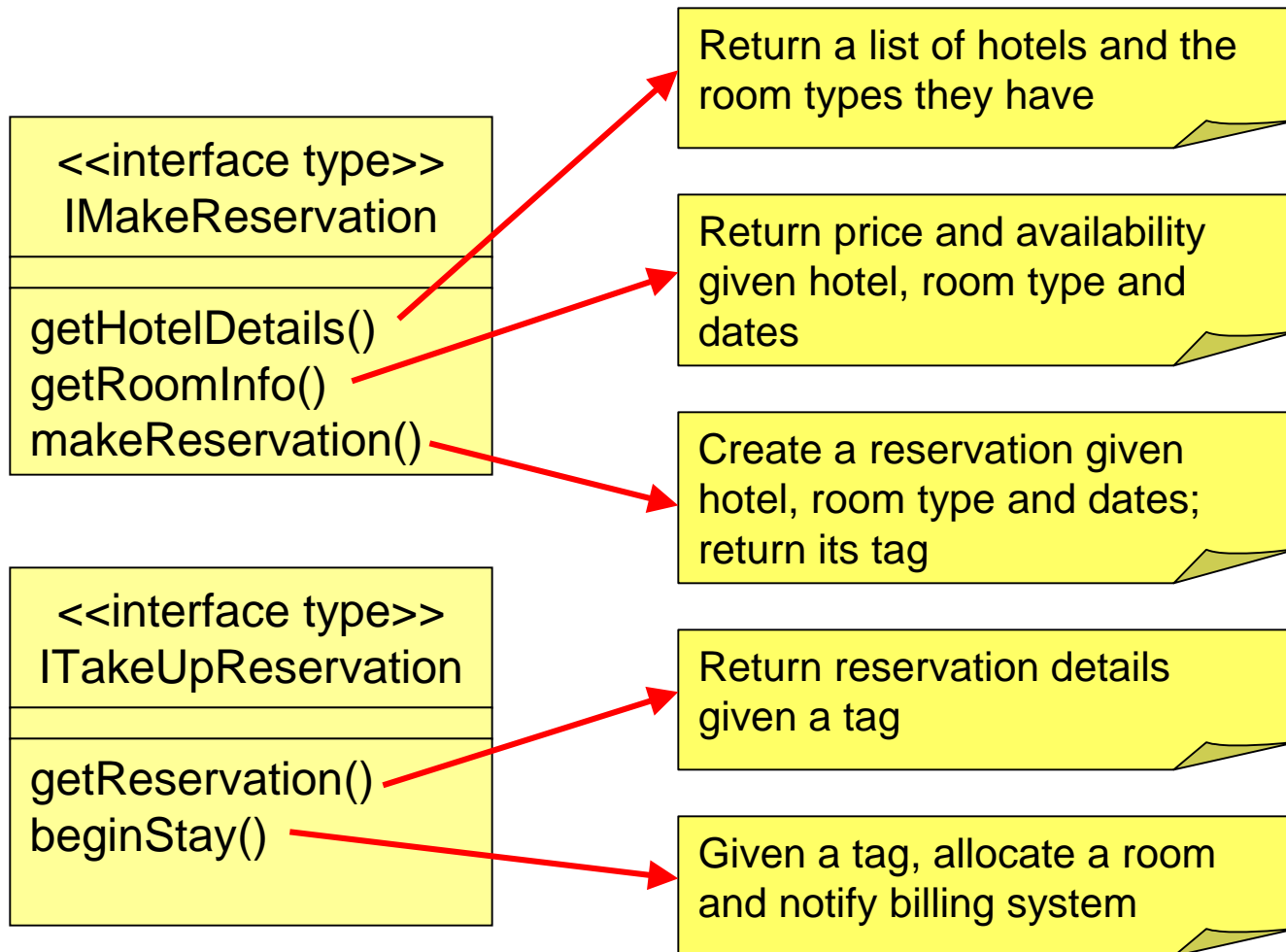


# Identify System Interfaces and operations

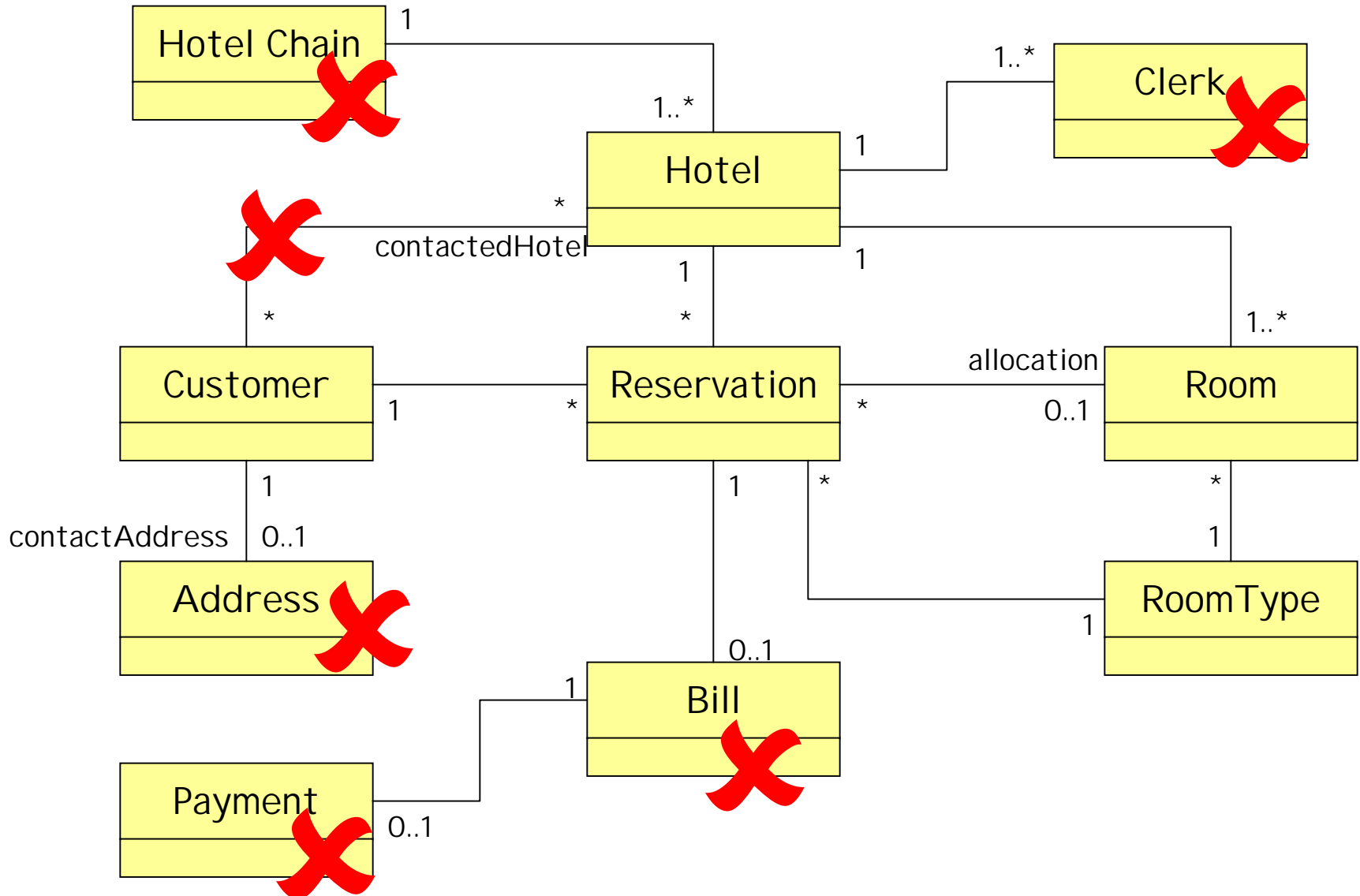
System interfaces act as facades - they are the point of contact for the UI and other external agents. They are supported by components in the system services layer.  
Start with one interface per use case, then refactor as necessary.



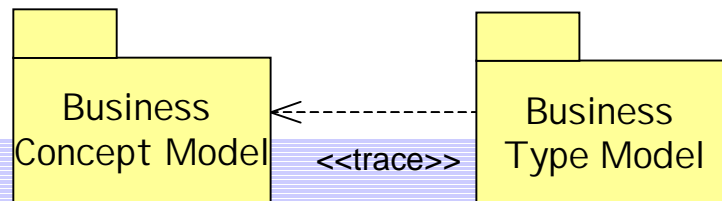
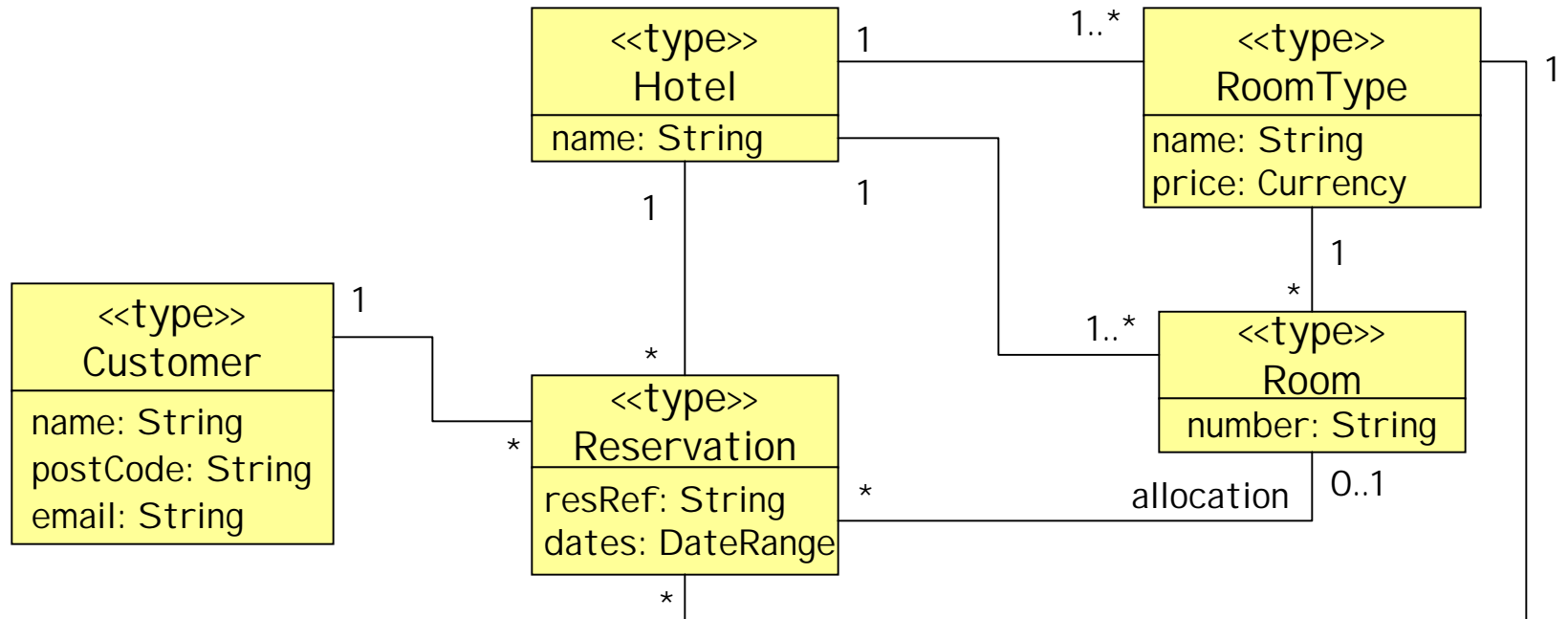
# Use case step operations



# Develop the Business Type Model



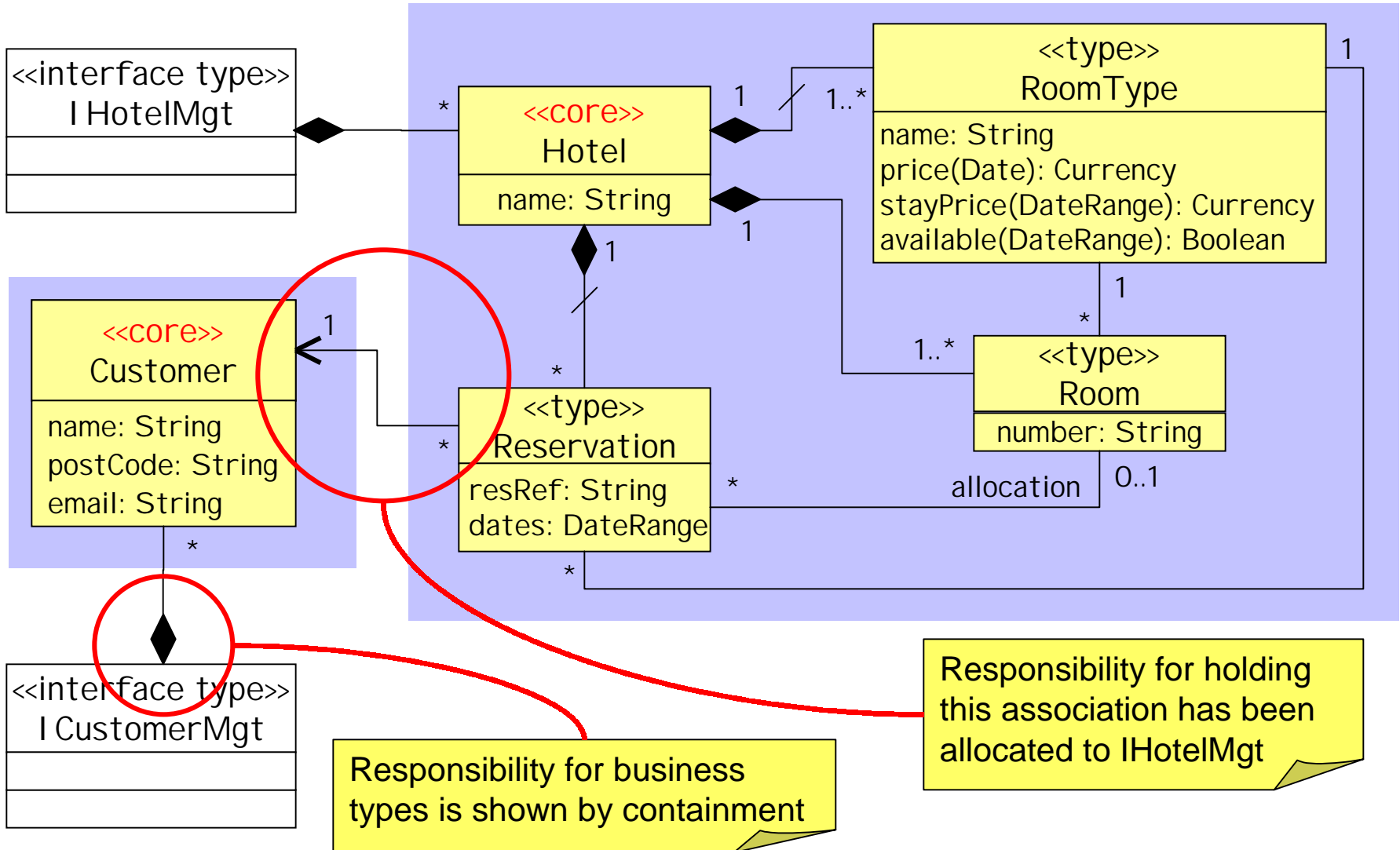
# Initial Business Type Diagram



# Identify Core types

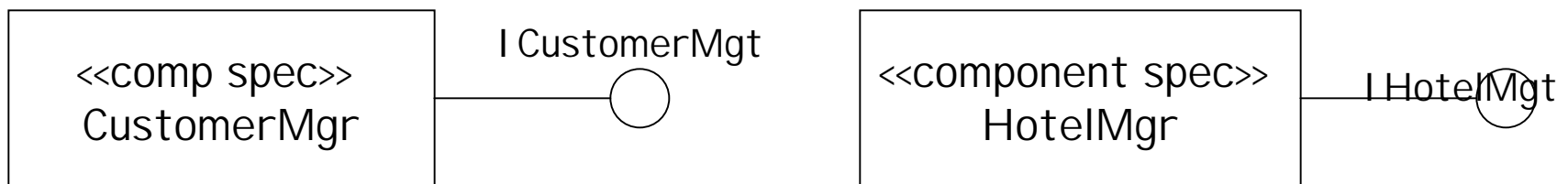
- Core types represent the primary business information that the system must manage
- Each core type will correspond directly to a business interface
- A core type has:
  - a business identifier, usually independent of other identifiers
  - independent existence – no mandatory associations (multiplicity equal to 1), except to a categorizing type
- In our case study:
  - Customer YES. Has id (name) and no mandatory assocs.
  - Hotel YES. Has id (name) and no mandatory assocs.
  - Reservation NO. Has mandatory assocs.
  - Room NO. Has mandatory assoc to Hotel
  - RoomType NO. Has mandatory assoc to Hotel

# Identify business interfaces



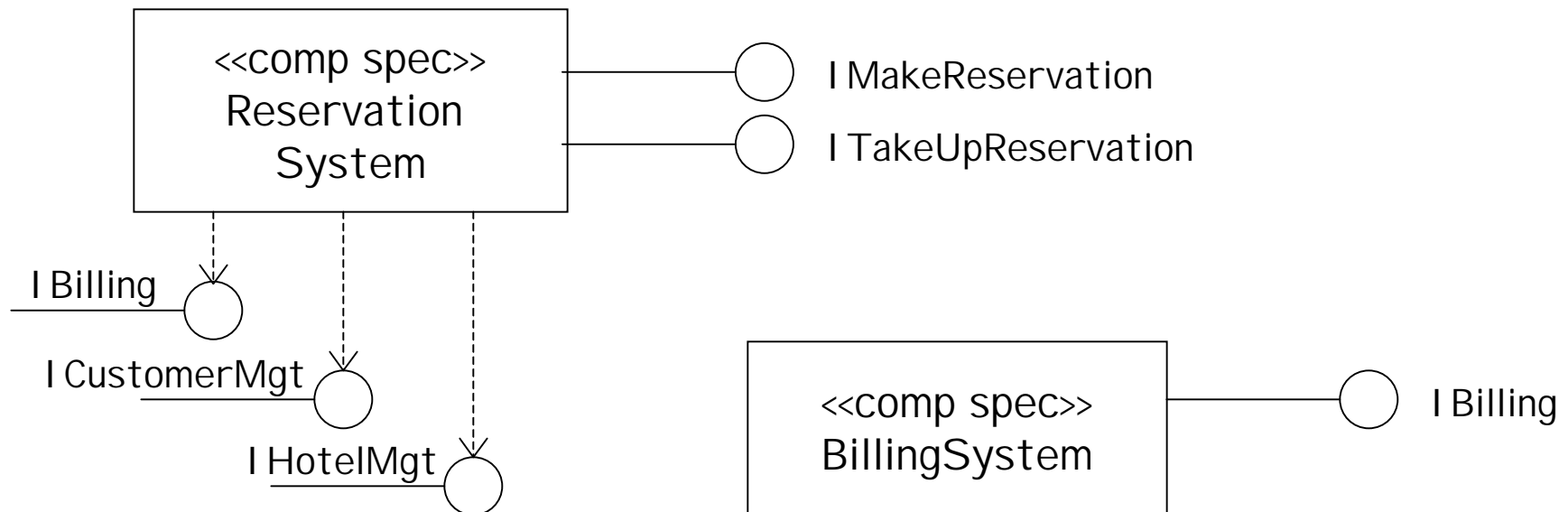
# Component Specifications

- We need to decide what components we want, and which interfaces they will support
- These are fundamental architectural decisions
- Business components:
  - they support the business interfaces
  - remember: components define the unit of development and deployment
- The starting assumption is one component spec per business interface

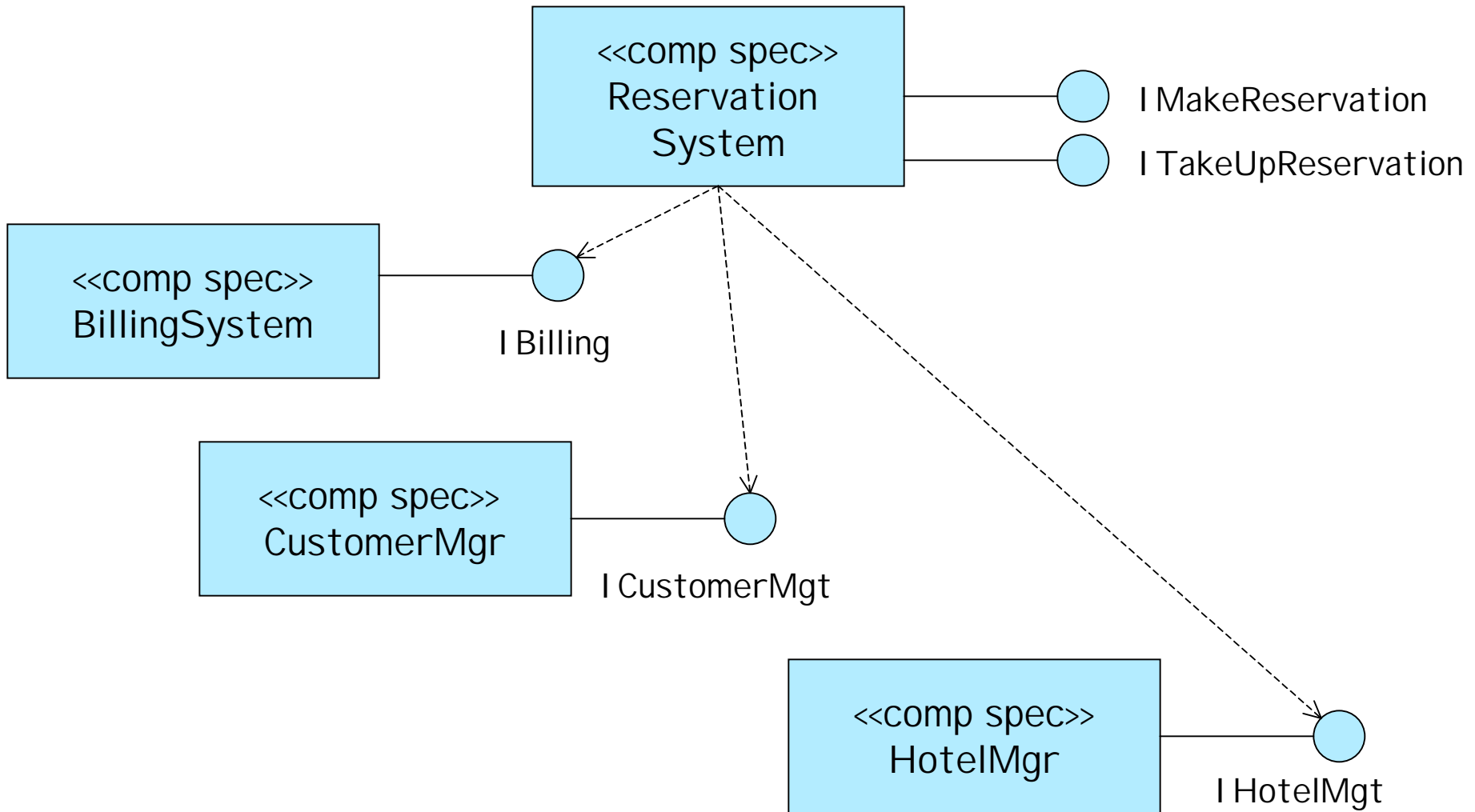


# System components

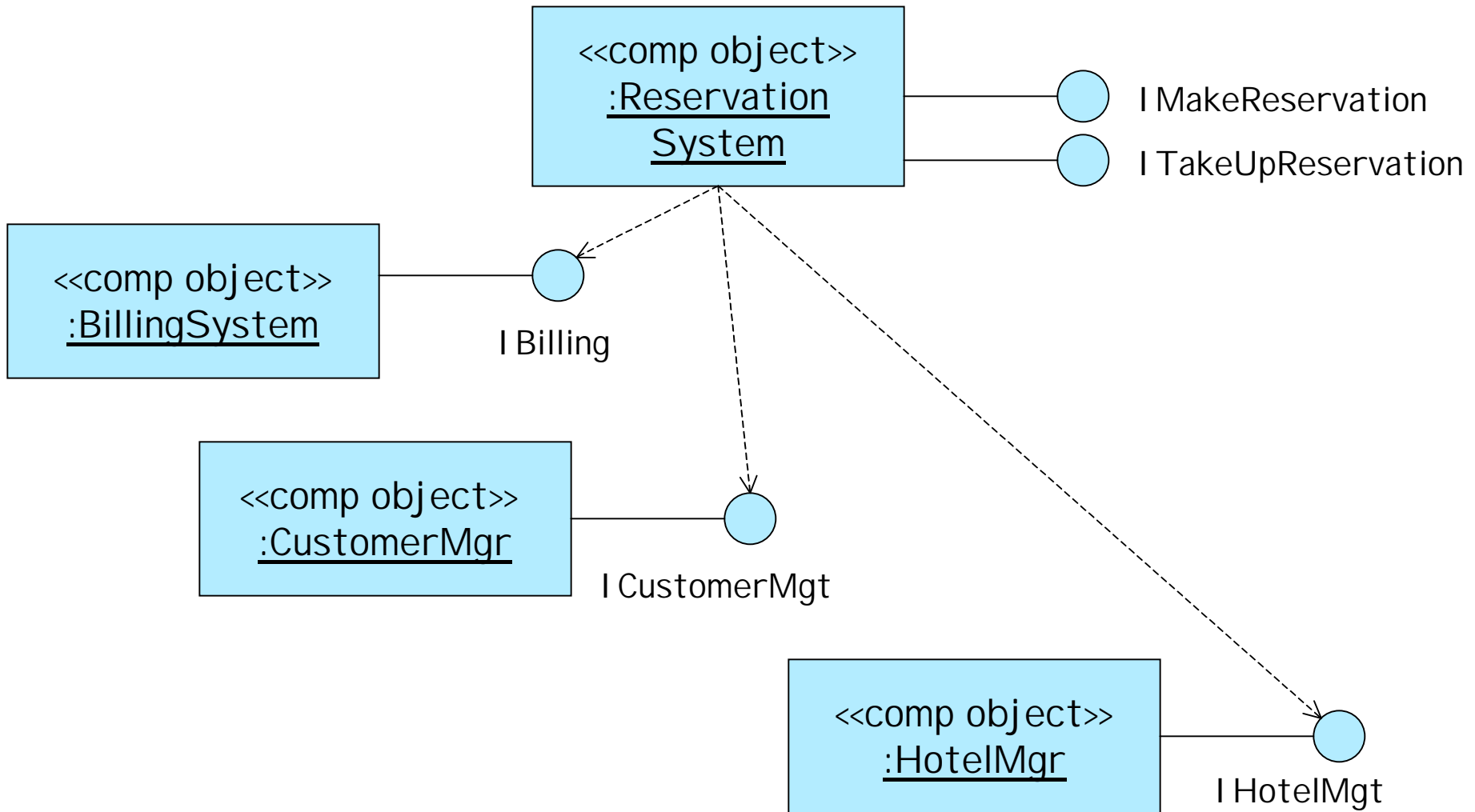
- We will define a single system component spec that supports all the use case system interfaces
  - Alternatives: one component per use case, support system interfaces on the business components
- Use a separate component spec for billing system wrapper



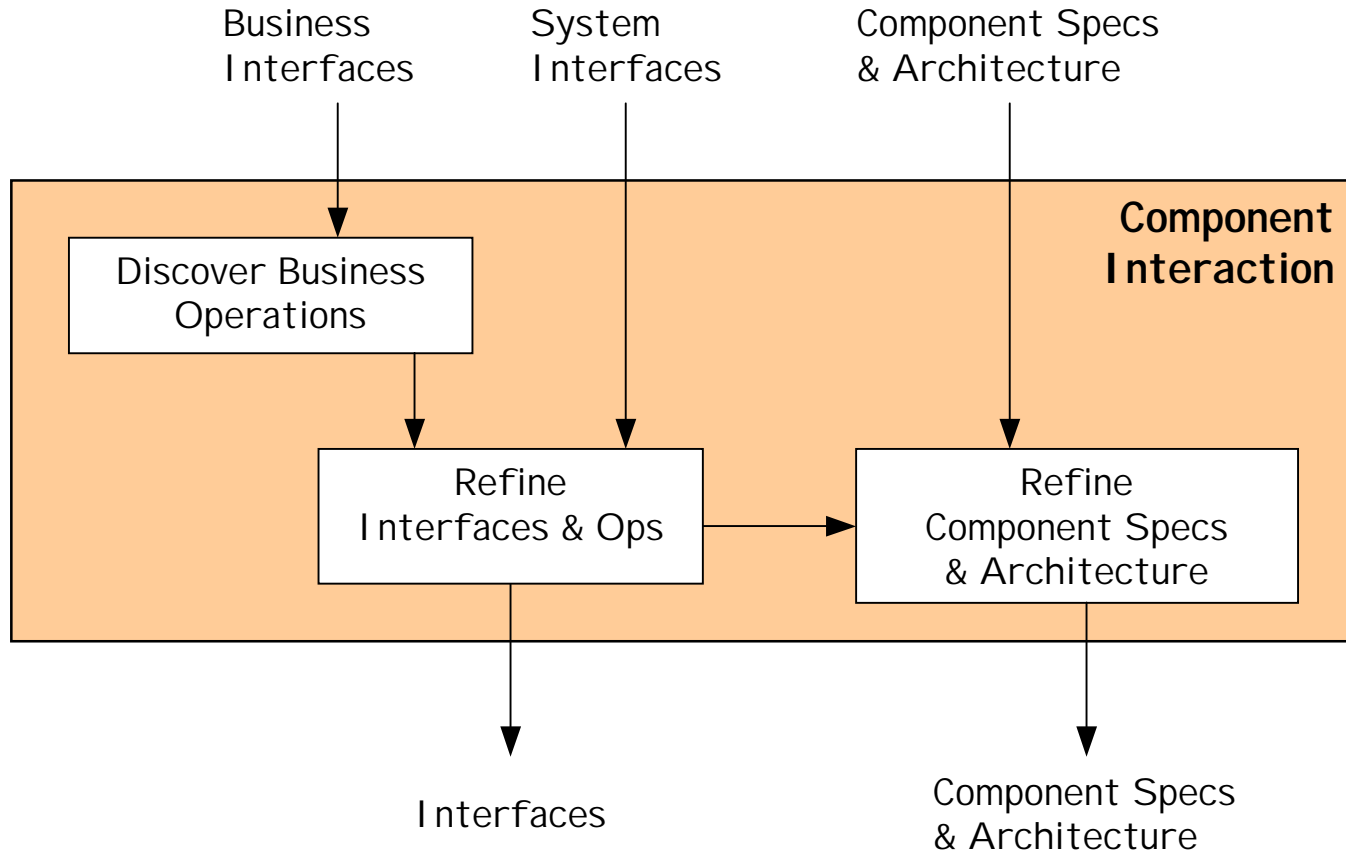
# Component architecture



# Minimal component object architecture

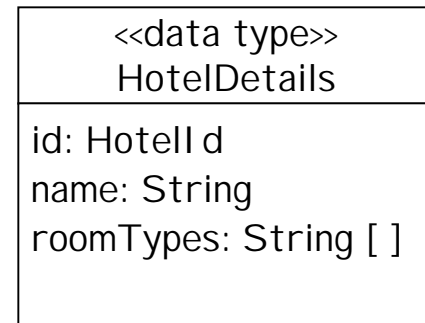
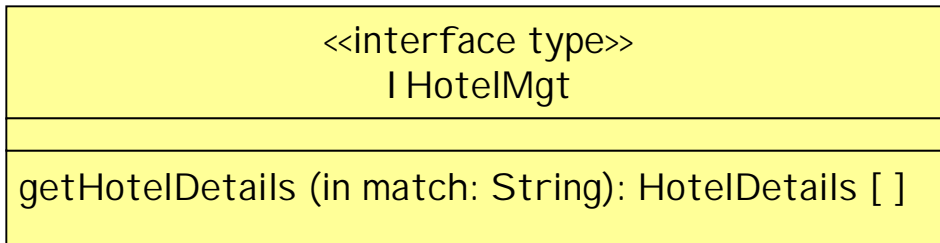
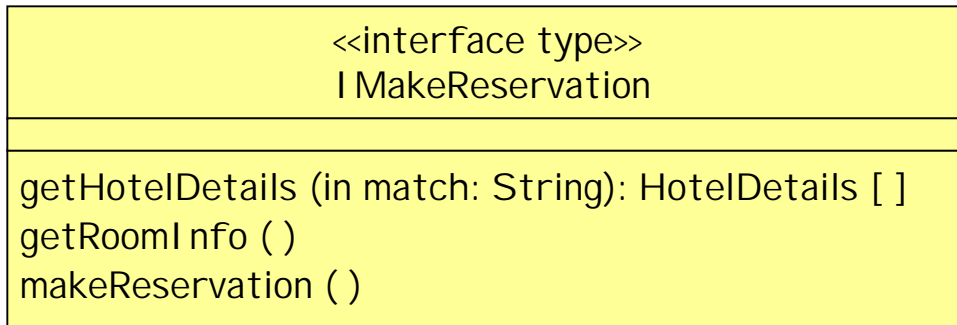
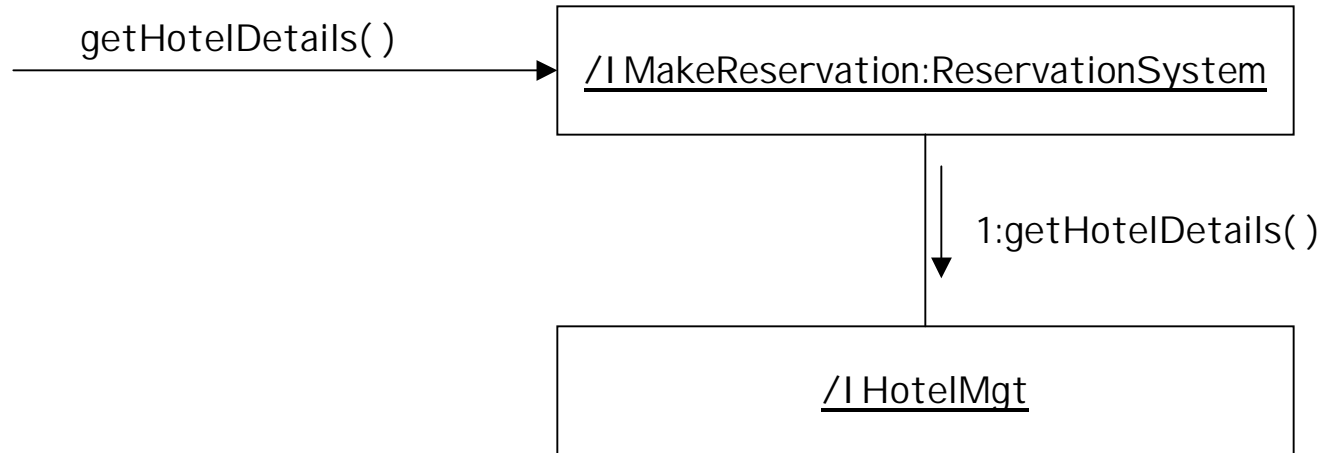


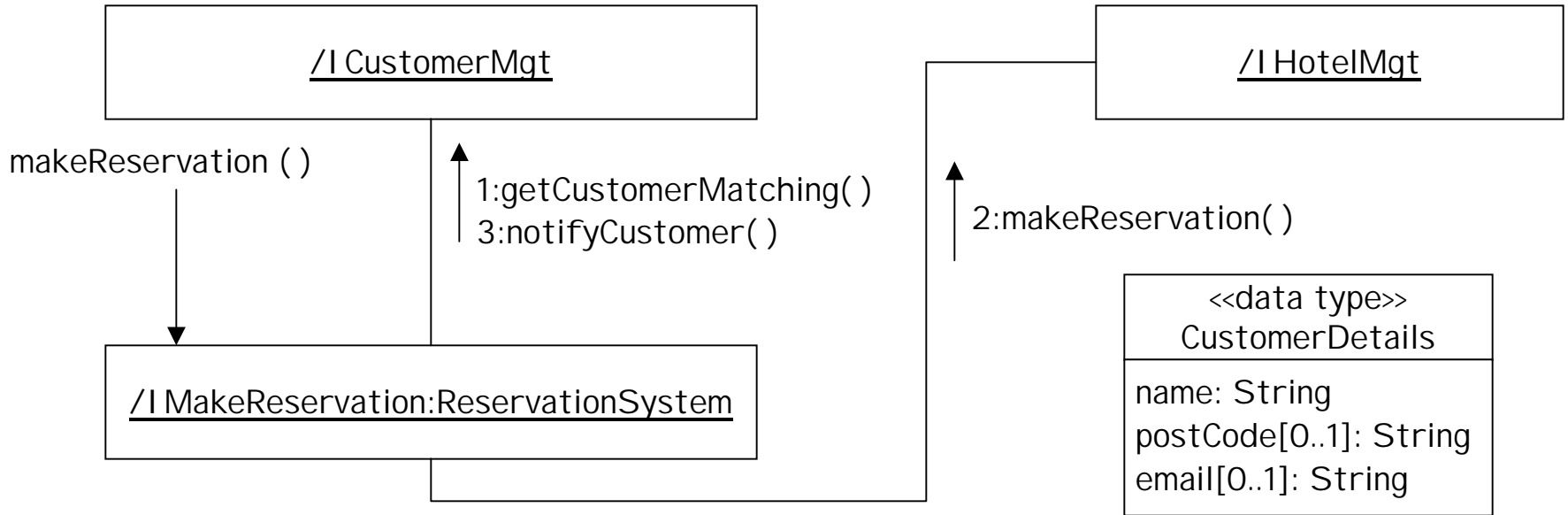
# Component Interaction



# Operation discovery

- Uses interaction diagrams (collaboration diagrams)
- The purpose is to discover operations on business interfaces that must be specified
  - not all operations will be discovered or specified
- Take each use case step operation in turn:
  - decide how the component offering it should interact with components offering the business interfaces
  - draw one or more collaboration diagram per operation
  - define signatures for all operations





<<interface type>>  
 I MakeReservation

---

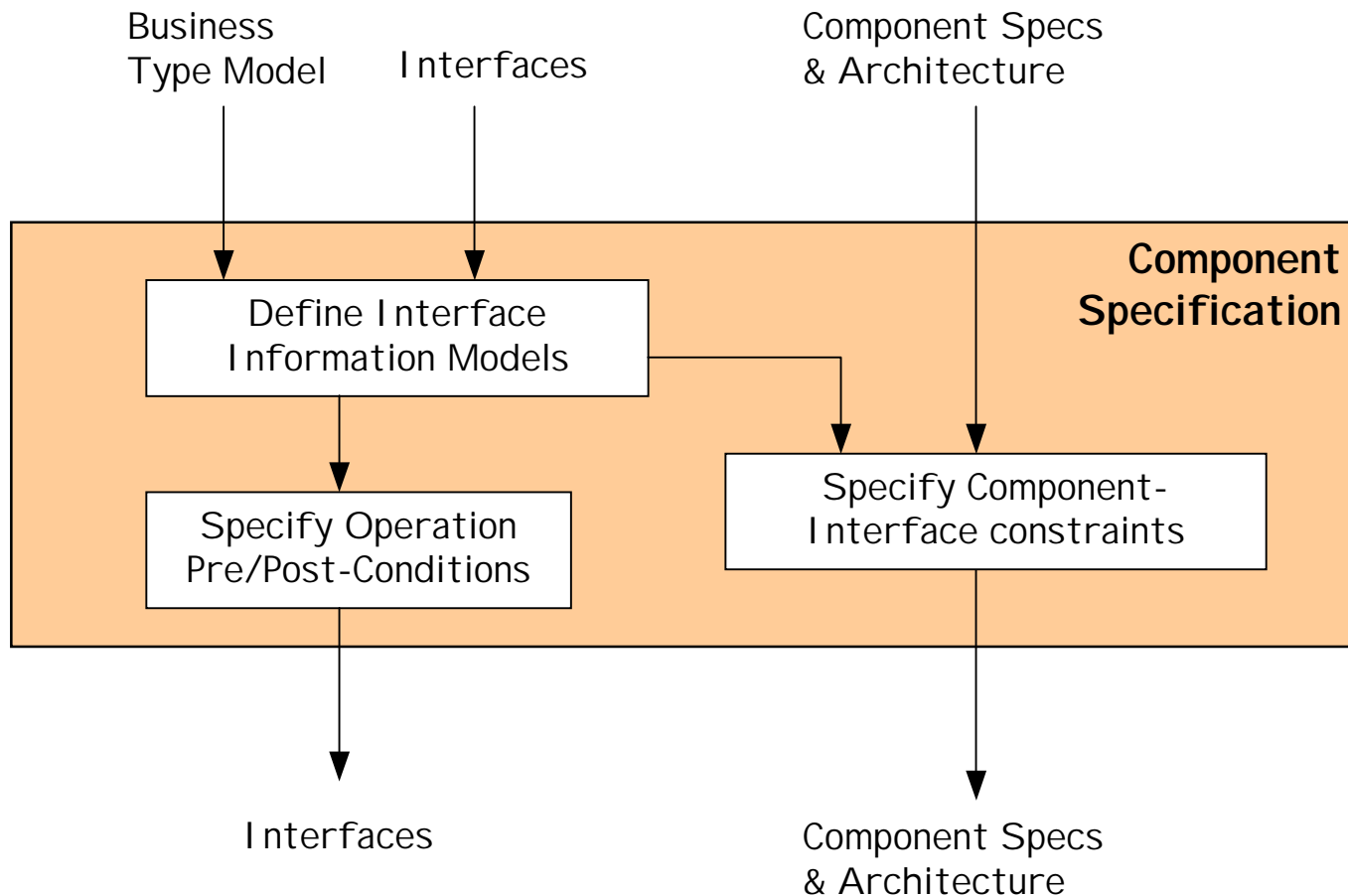
getHotelDetails (in match: String): HotelDetails [ ]  
 getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)  
 makeReservation (in res: ReservationDetails, in cus: CustomerDetails, out resRef: String): Integer

<<interface type>>  
 I HotelMgt

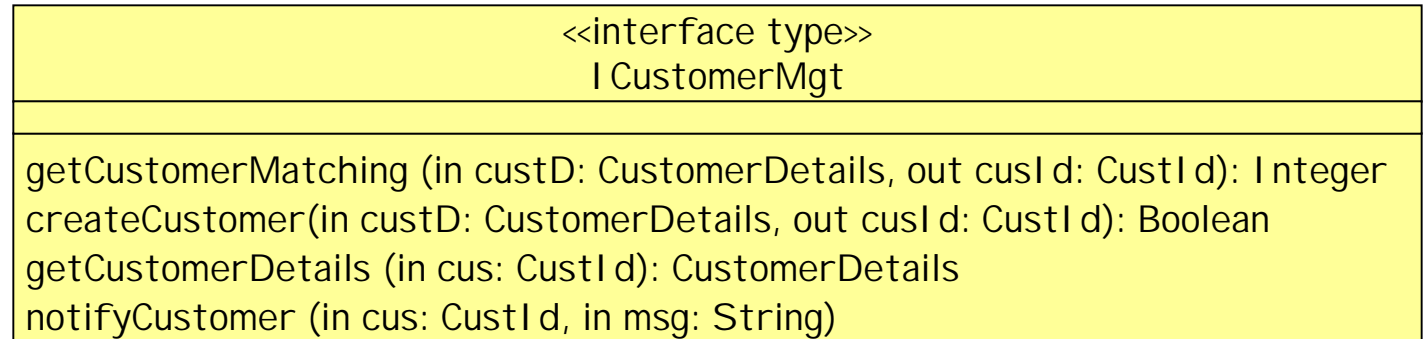
---

getHotelDetails (in match: String): HotelDetails [ ]  
 getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)  
 makeReservation (in res: ReservationDetails, in cus: CustId, out resRef: String): Boolean

# Component Specification



# Interface information model

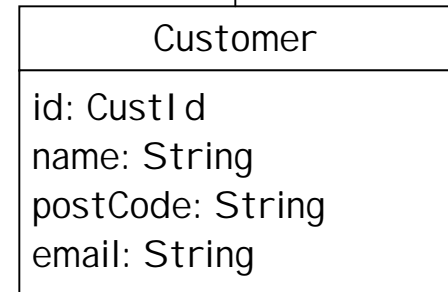


Defines the set of information assumed to be held by a component object offering the interface, **for the purposes of specification only**.

Implementations **do not** have to hold this information themselves, but they must be able to obtain it.

The model need only be sufficient to explain the effects of the operations.

The model can be derived from the Business Type Model.



# Pre- and post-conditions

- If the pre-condition is true, the post-condition must be true
- If the pre-condition is false, the post-condition doesn't apply
- A missing pre-condition is assumed 'true'
- Pre- and post-conditions can be written in natural language or in a formal language such as OCL

```
context | CustomerMgt::getCustomerDetails (in cus: CustId): CustomerDetails
```

```
pre:
```

```
-- cus is valid  
customer->exists(c | c.id = cus)
```

```
post:
```

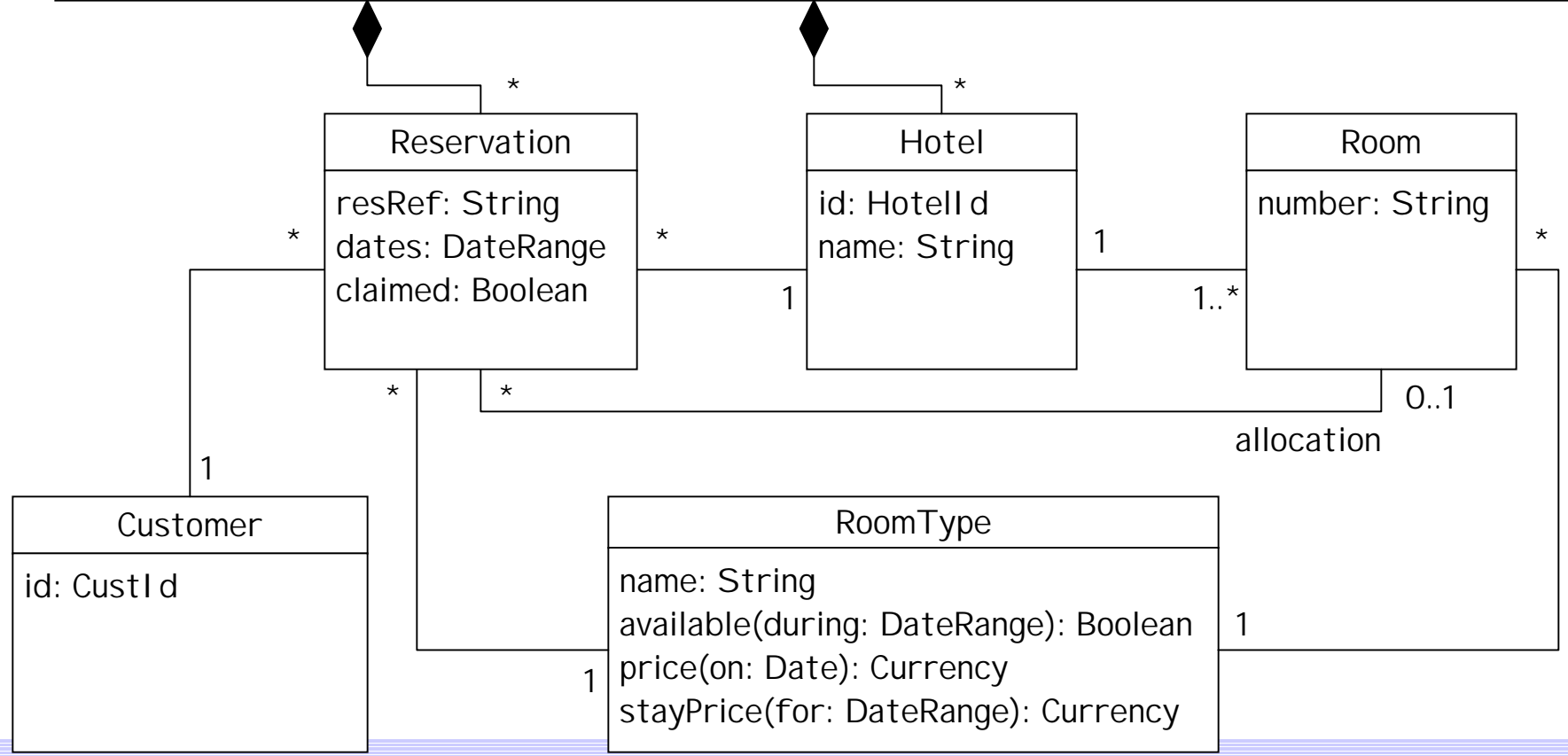
```
-- the details returned match those held for customer cus  
Let theCust = customer->select(c | c.id = cus) in  
result.name = theCust.name  
result.postCode = theCust.postCode  
result.email = theCust.email
```

```

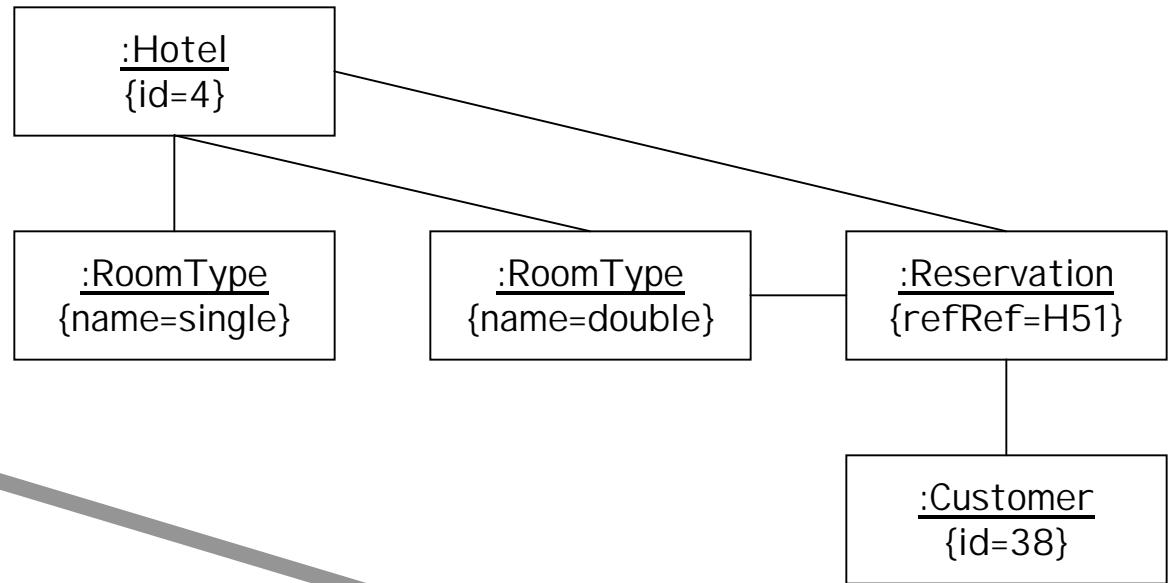
<<interface type>>
I HotelMgt

getHotelDetails (in match: String): HotelDetails [ ]
getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)
makeReservation (in res: ReservationDetails, in cus: CustId, out resRef: String): Boolean
getReservation(in resRef: String, out rd ReservationDetails, out cusId: CustId): Boolean
beginStay (resRef: String , out roomNumber: String): Boolean

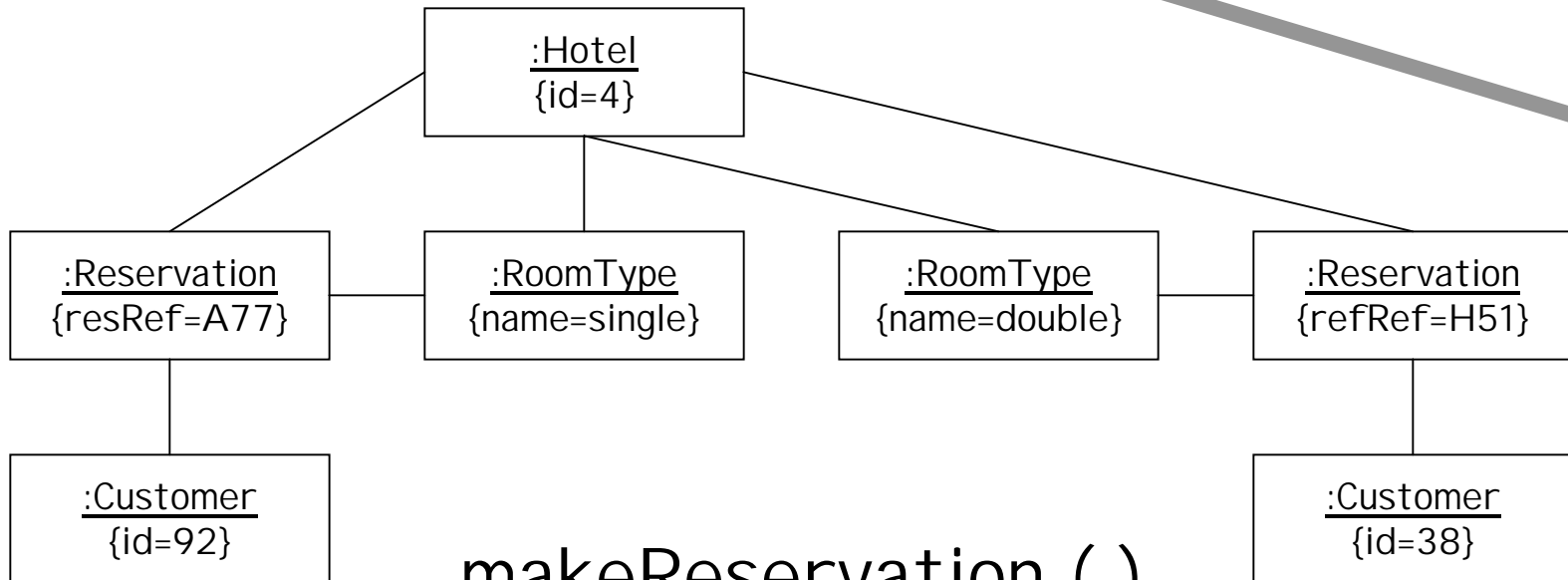
```



before



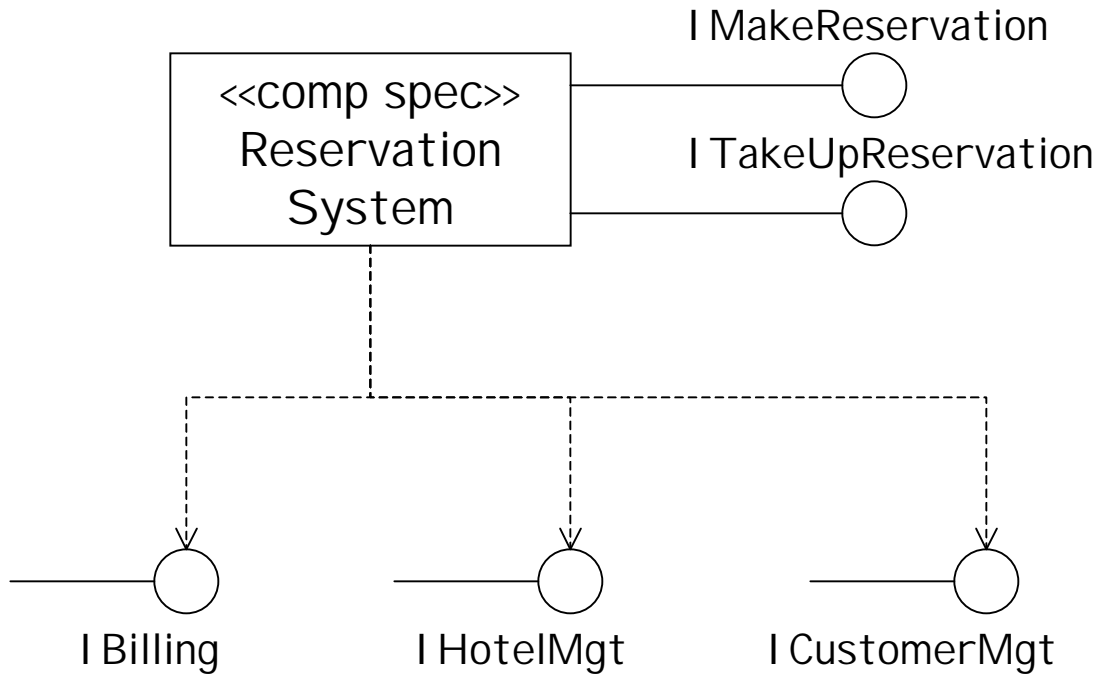
after



makeReservation ()

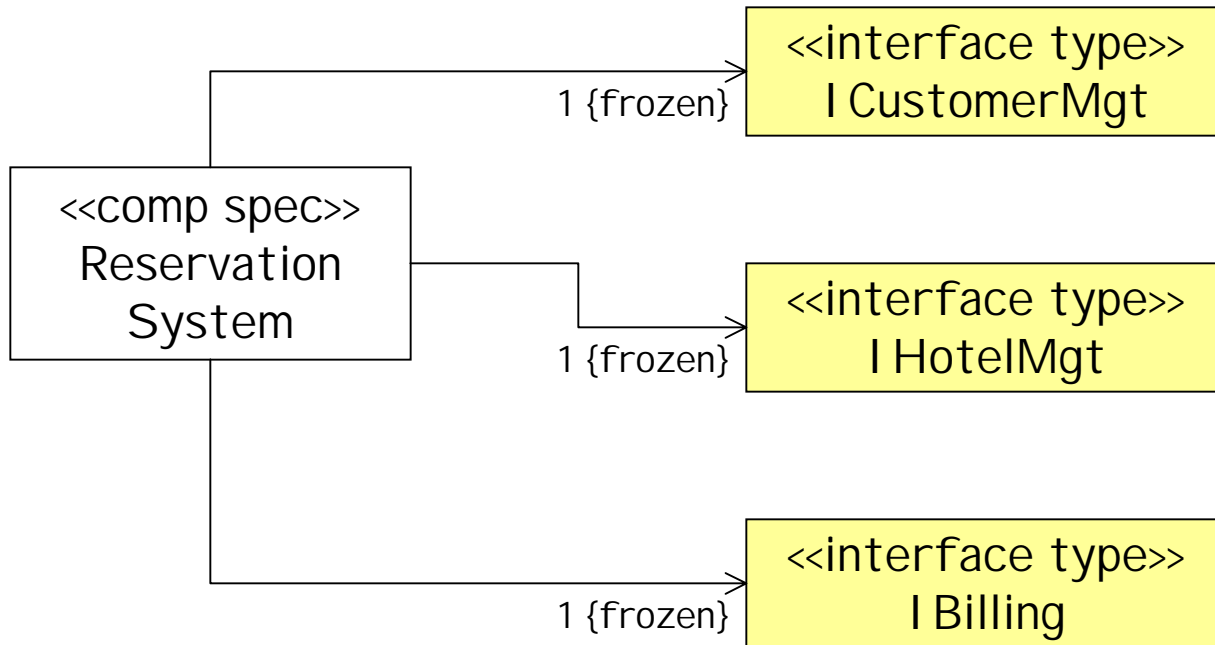
```
context | HotelMgt::makeReservation (  
    in res: ReservationDetails, in cus: CustId, out resRef: String): Boolean  
  
pre:  
    -- the hotel id and room type are valid  
    hotel->exists(h | h.id = res.hotel and h.room.roomType.name->includes(res.roomType))  
  
post:  
    result implies  
        -- a reservation was created  
        -- identify the hotel  
        Let h = hotel->select(x | x.id = res.hotel)->asSequence->first in  
            -- only one more reservation now than before  
            (h.reservation - h.reservation@pre)->size = 1 and  
            -- identify the reservation  
            Let r = (h.reservation - h.reservation@pre)->asSequence->first in  
                -- return number is number of the new reservation  
                r.resRef = resRef and  
                -- other attributes match  
                r.dates = res.dateRange and  
                r.roomType.name = res.roomType and not r.claimed and  
                r.customer.id = cus
```

# Specifying a component (1)



Specification of interfaces offered and used  
(part of the realization contract)

# Specifying a component (2)



Specification of the *component object* architecture.  
This tells us how many objects offering  
the used interfaces are involved

# Specifying a component (3)

**Context** ReservationSystem

-- between offered interfaces

I MakeReservation::hotel = I TakeUpReservation::hotel

I MakeReservation::reservation = I TakeUpReservation::reservation

I MakeReservation::customer = I TakeUpReservation::customer

-- between offered interfaces and used interfaces

I MakeReservation::hotel = iHotelMgt.hotel

I MakeReservation::reservation = iHotelMgt.reservation

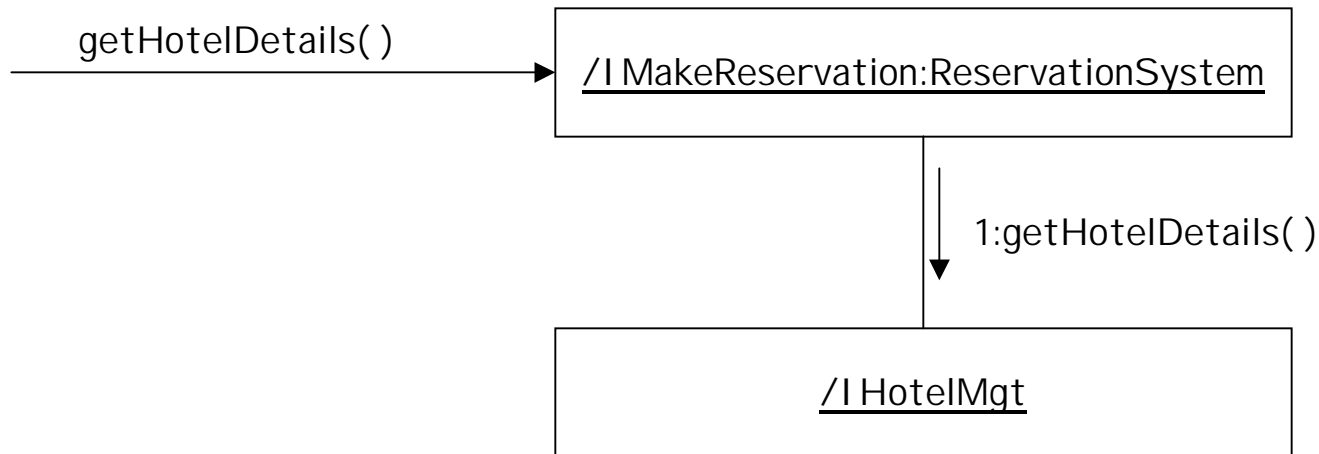
I MakeReservation::customer = iCustomerMgt.customer

**Specification of the Component Spec-Interface constraints.**

The top set of constraints tell the realizer the required relationships between elements of different offered interfaces.

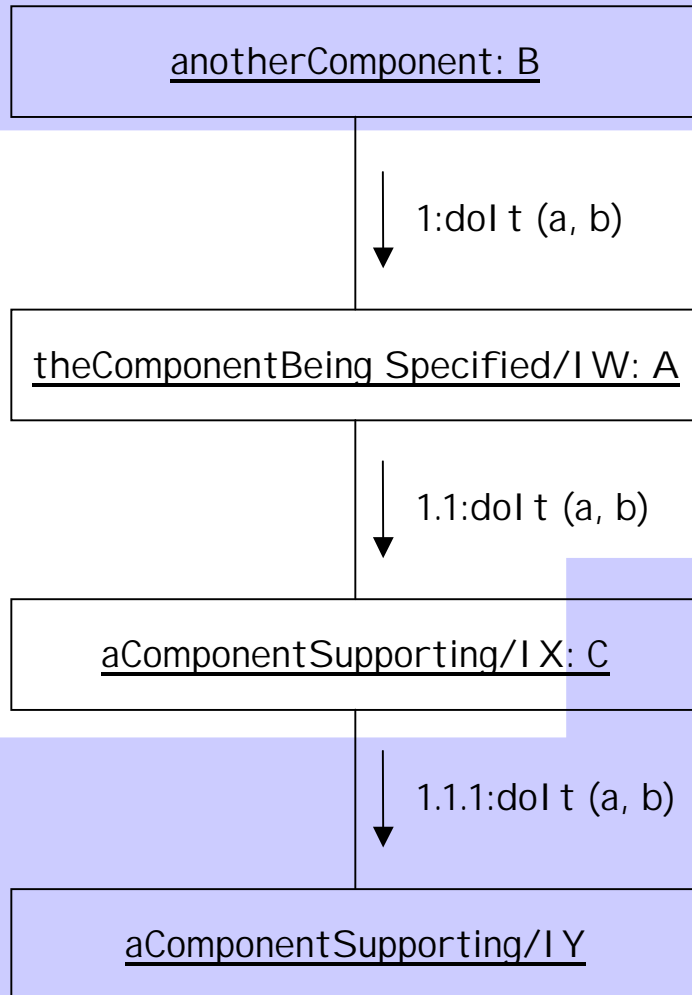
The bottom set tell the realizer the relationships between elements of offered interfaces and used interfaces that must be maintained.

# Interactions as specification?



- Is every implementation of `ReservationSystem` *required* to invoke `getHotelDetails()` in this situation?
- If so, drawing the collaboration diagram is an act of specification...
- If not, then we are using this technique simply as a way of discovering useful operations

# Specifying a component (4)



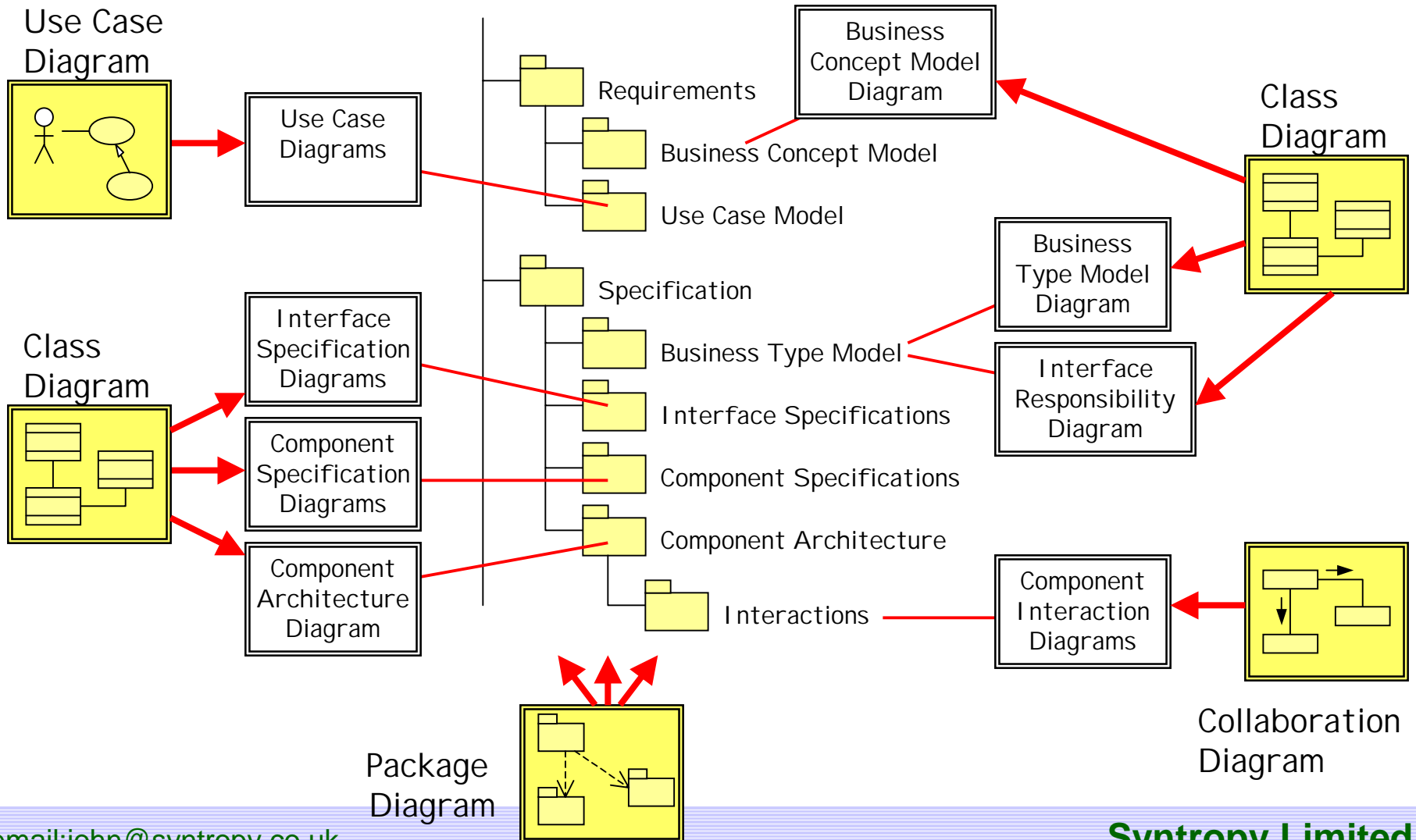
If we want to provide a more detailed specification we can use interaction diagram fragments.

These are pieces of the diagrams we drew earlier, for operation discovery, that focus on the component being specified.

Each fragment specifies how a particular operation is to be implemented in terms of interaction with other components.

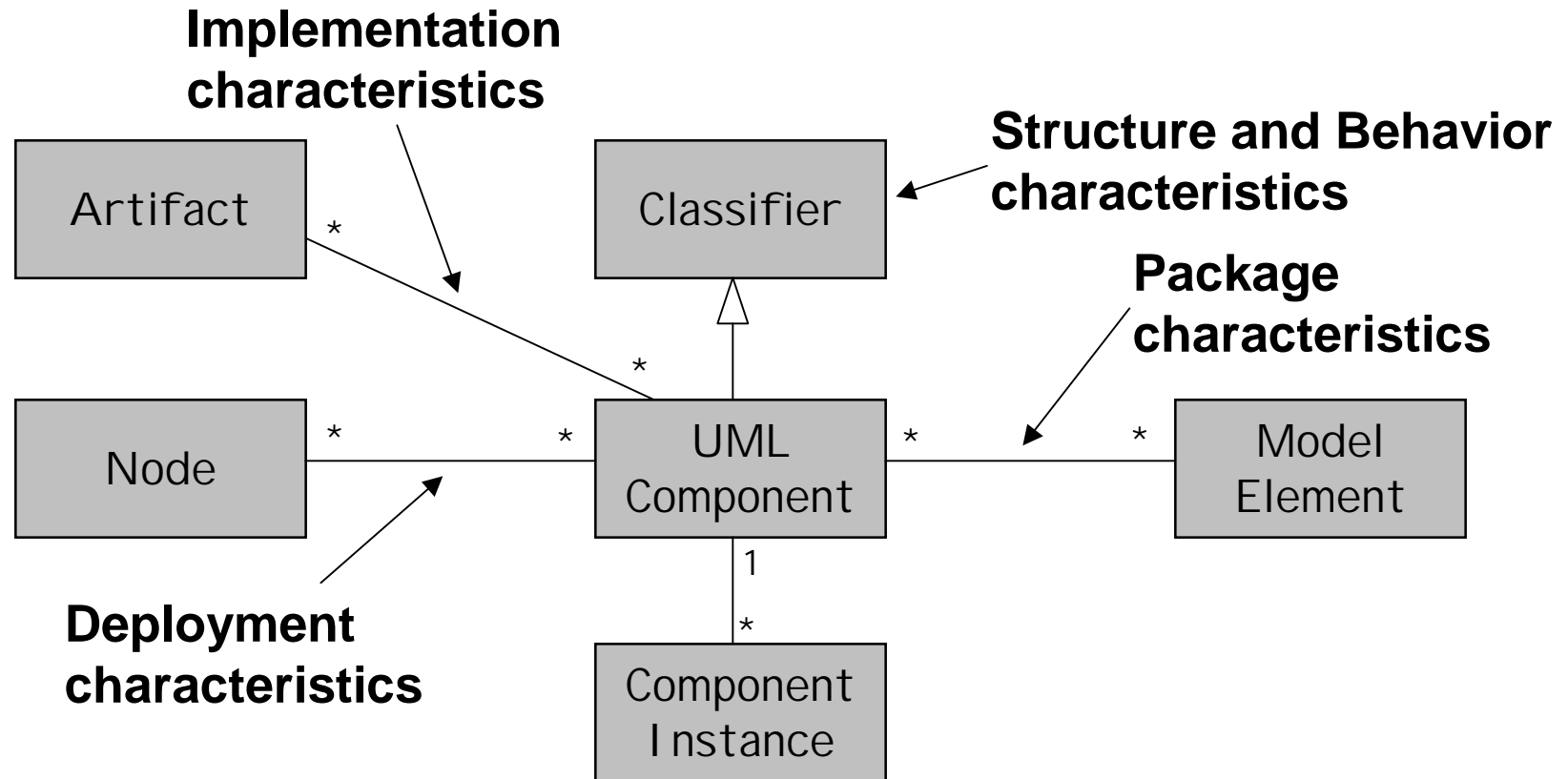
Warning: in some cases this will be over-specification.

# UML diagrams used in the process



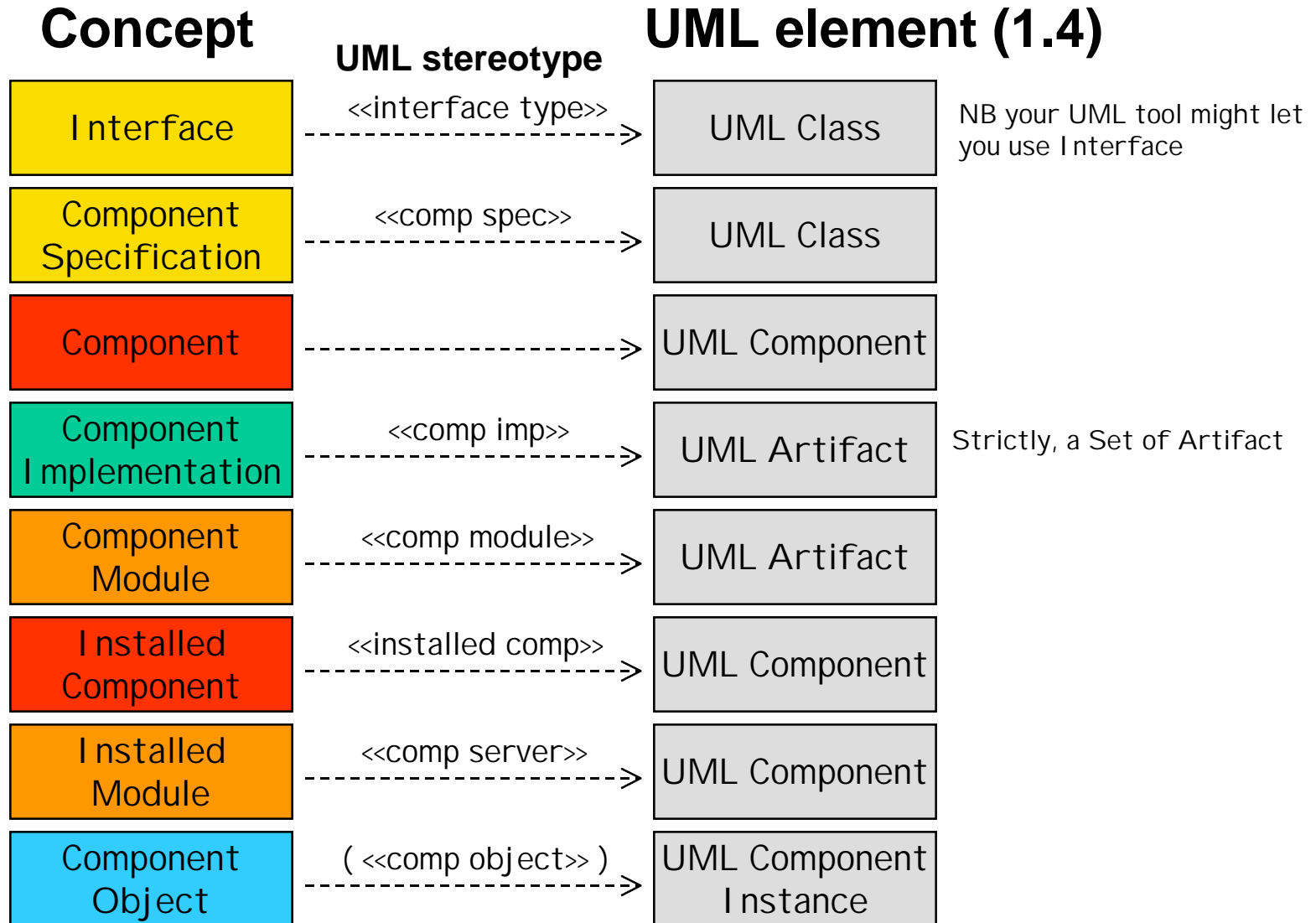
# Implications for the UML

# UML Component (v1.4)

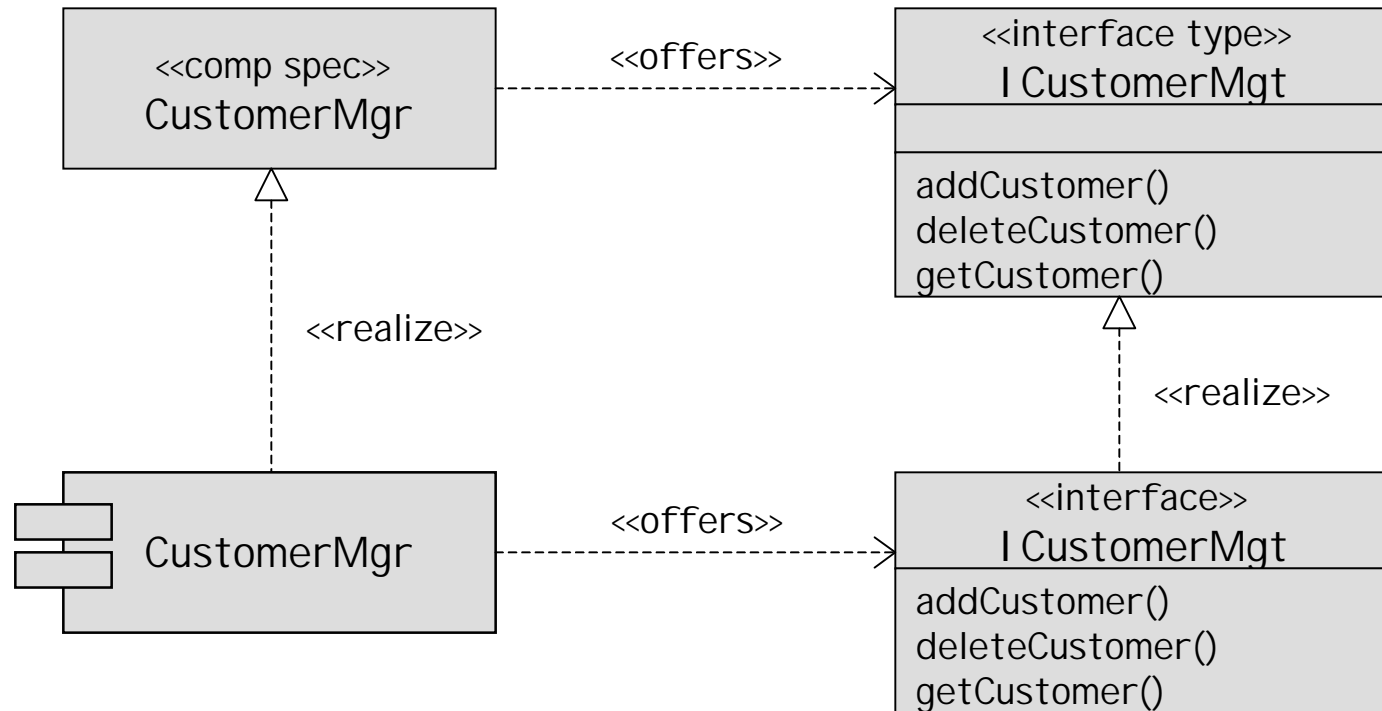


- UML Glossary: “a physical, replaceable part [...] that packages implementation and [...] provides the realization of a set of interfaces”

# Mapping to UML

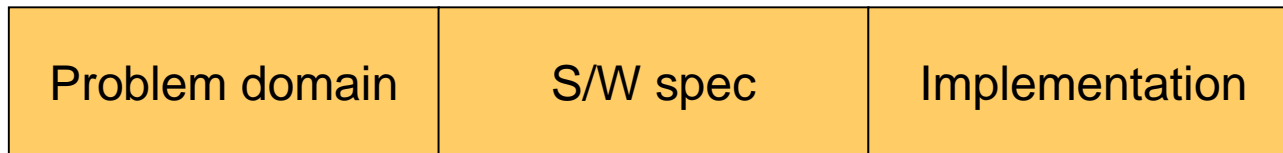


# Realization mappings



# Model “perspectives”

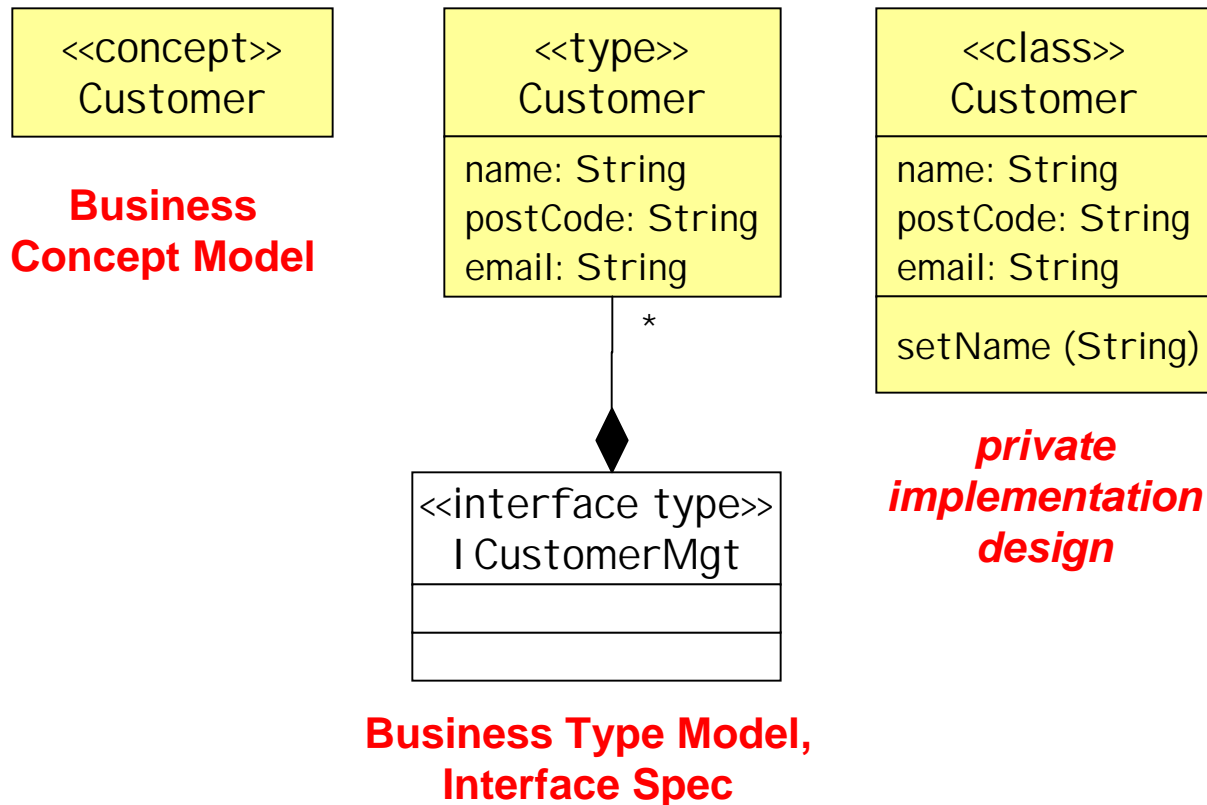
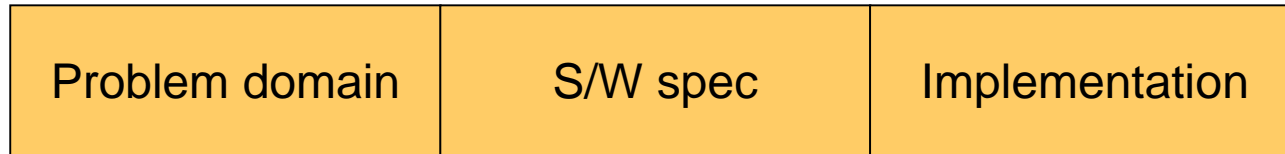
- UML is a language for describing models
- What is the purpose of your model?
  - Models that describe the problem domain
    - nothing to do with software
  - Models that specify software
    - ranging from the whole system to one small part
  - Models that describe the implementation of software



# Typical usage of UML notations

	Problem domain	S/W spec	Implementation
Use case		boundary interactions	
Class diagram	information models	component structures	component structures
Seq/collab diagram		required object interactions	designed object interactions
Activity diagram	business processes		algorithms
Statechart		object lifecycles	object lifecycles

# Same name, different purpose



# Want to know more?

- UML Components by John Cheesman and John Daniels, Addison-Wesley
- <http://www.umlcomponents.com>

