TOOLS Europe 2000

Component-Based Design: A Complete Worked Example

John Daniels
Syntropy Ltd, UK

John@Syntropy.co.uk

Introduction

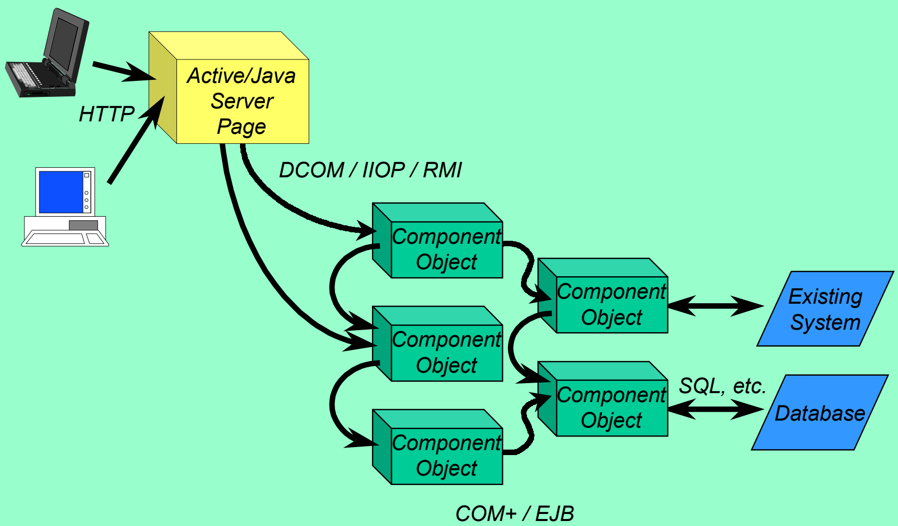
- Goal: follow a small example from requirements through to code-ready specification
- Component-based: assume that the target technology will be COM+, EJB or similar
- Process-centric: follow a well-defined design process
- Specification-oriented: most of the tutorial will be concerned with specifying the system and its components
- UML: use standard UML to describe the specifications

Tutorial Map

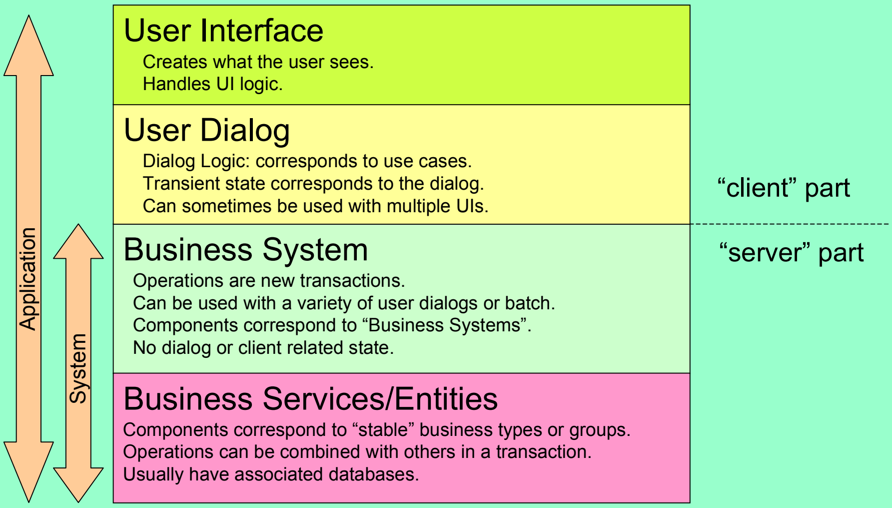
- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

Break

Blueprint for the systems being considered in this tutorial



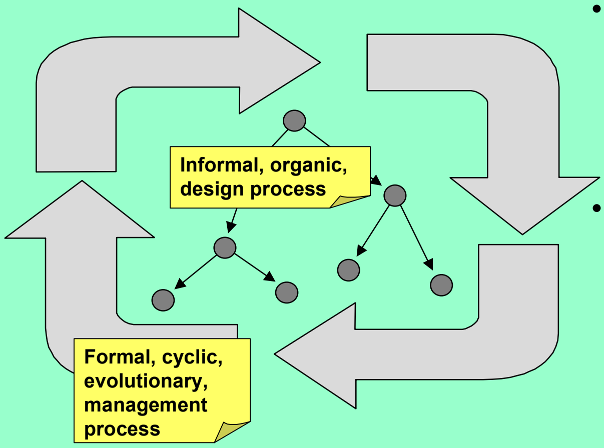
System architecture layers



Tutorial Map

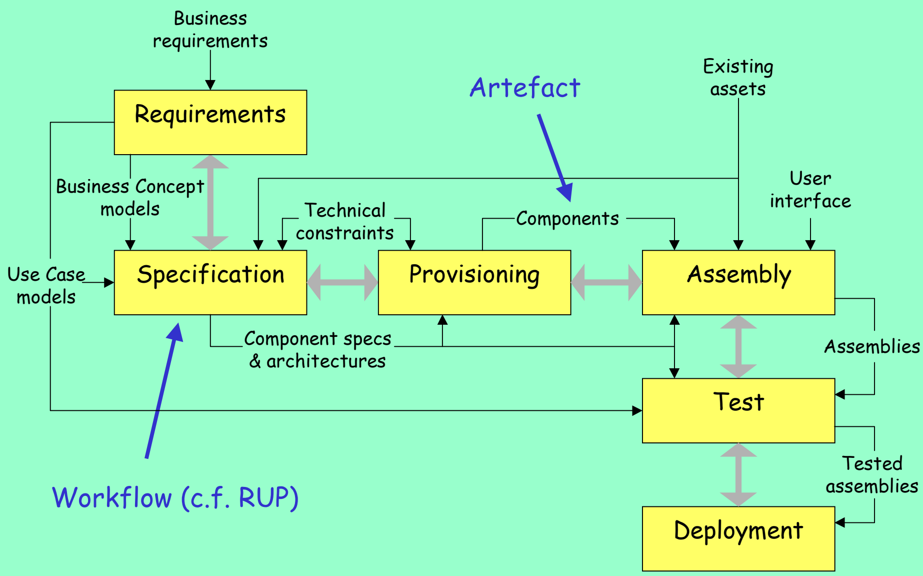
- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

The Design Process

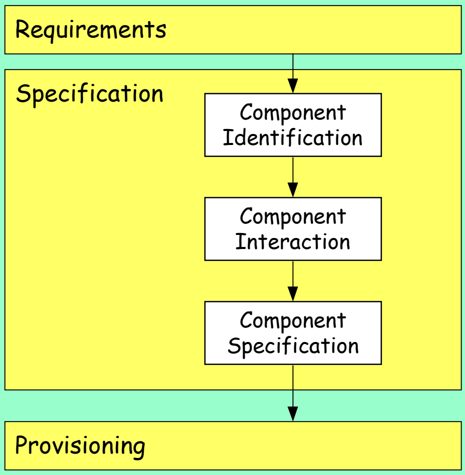


- The design process is organised around the production of artefacts
- The management process is organised around the delivery of working software

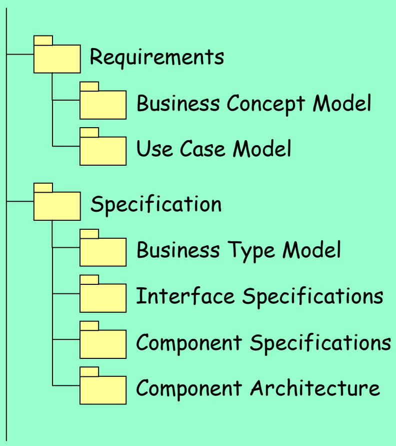
Workflows in the design process



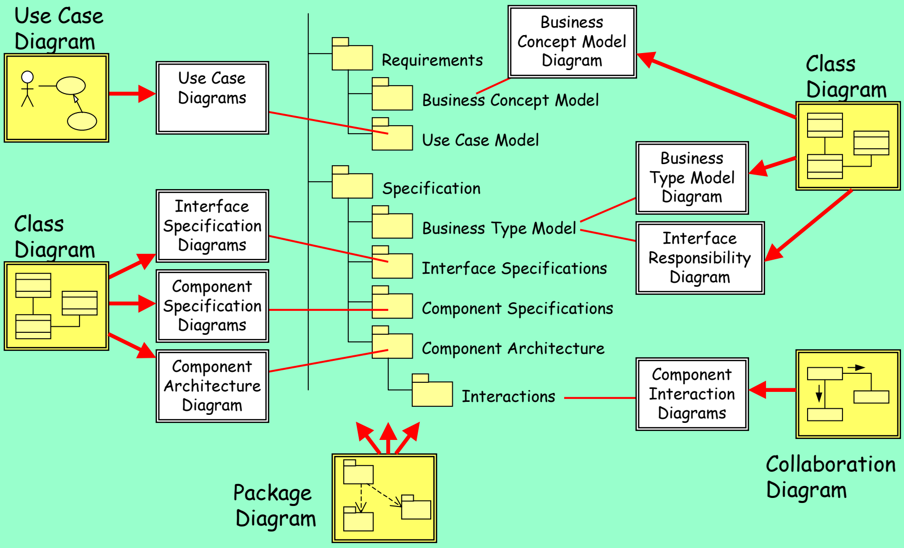
The scope of this tutorial



Organising the artefacts in tool packages

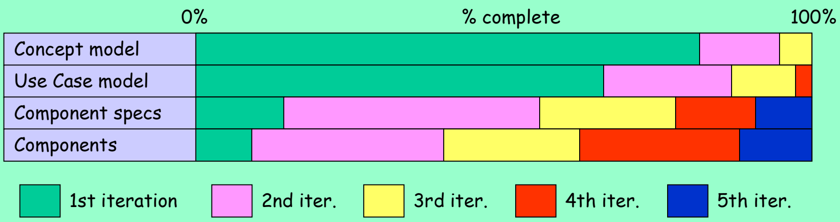


UML diagrams



Evolutionary delivery

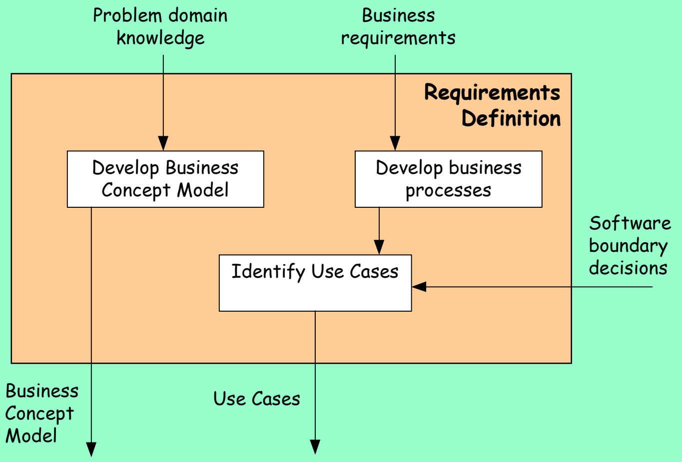
- All the artefacts evolve at each iteration
- Focus is on delivery to the user



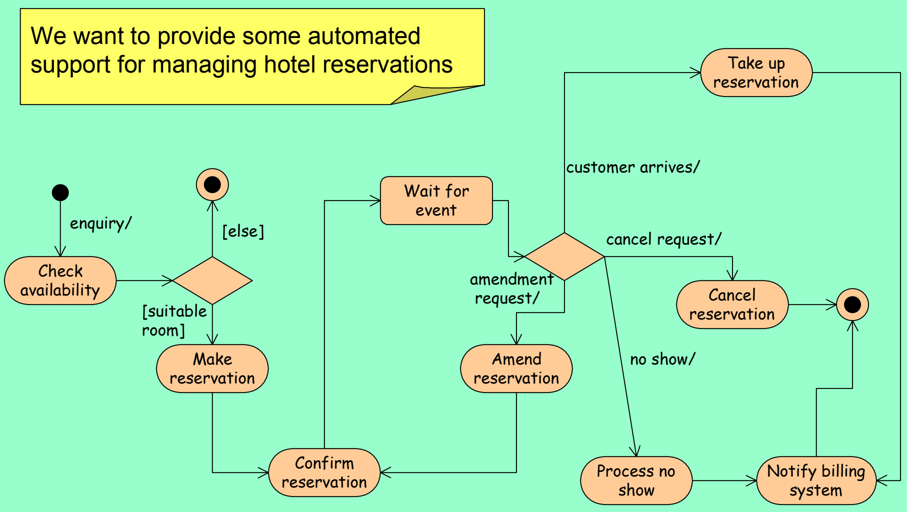
Tutorial Map

- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

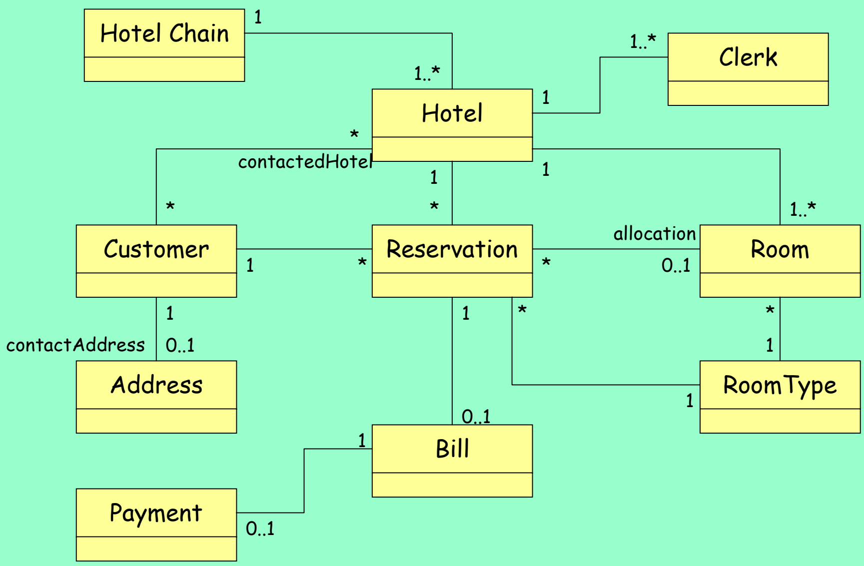
Requirements Definition



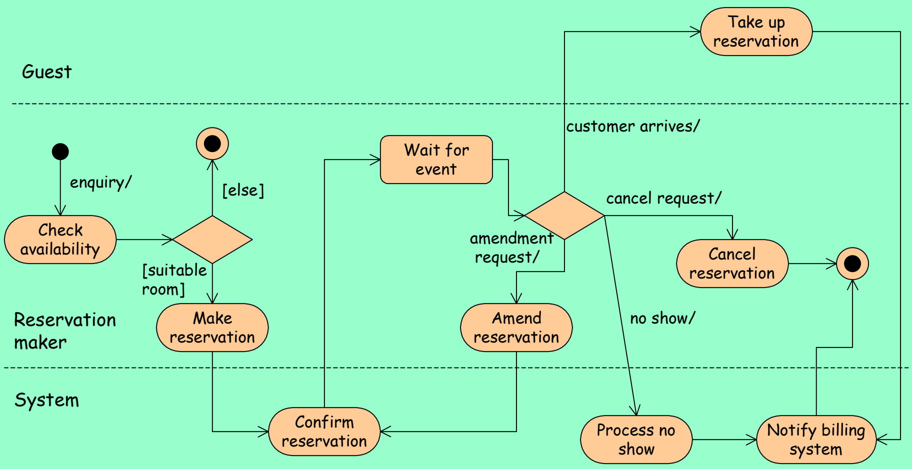
Business process



Business Concept Model

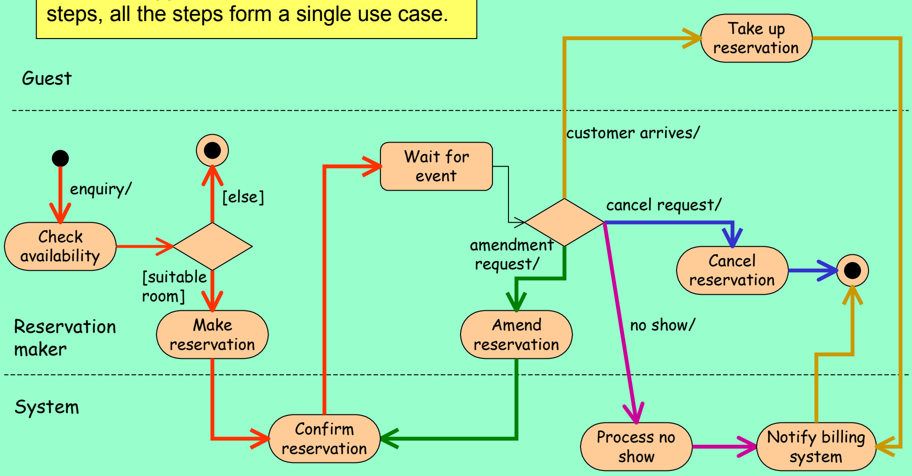


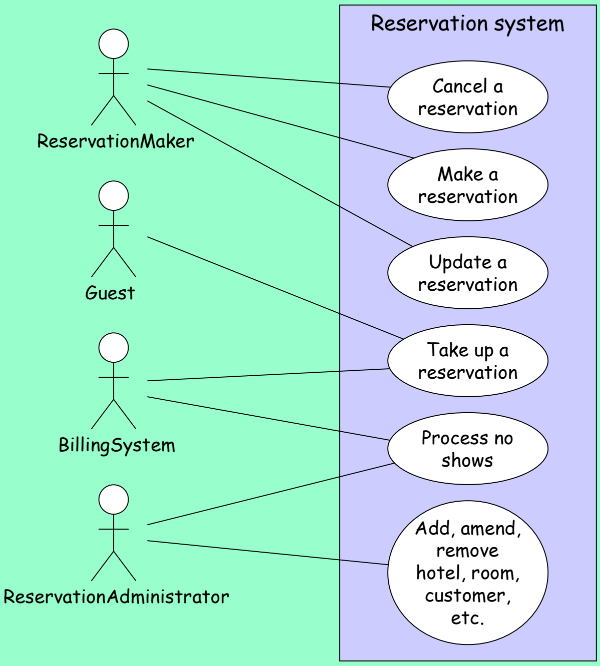
Assign responsibilities



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

Exercise 1

- Complete the use case on the next slide

Name	Take up a Reservation
Initiator	Guest
Goal	Claim a reservation

Main success scenario
1. Guest arrives at hotel to claim a room
2. Guest provides reservation tag to system
3.

Extensions

Name	Take up a Reservation
Initiator	Guest
Goal	Claim a reservation

Main success scenario

1. Guest arrives at hotel to claim a room
2. Guest provides reservation tag to system
3. System displays reservation details
4. Guest confirms details
5. System allocates a room
6. System notifies billing system that a stay is starting

Extensions

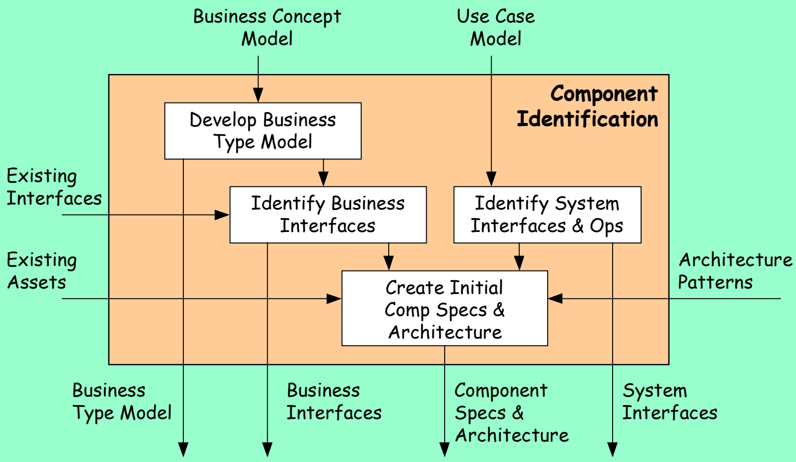
3. Tag not recognised
    1. Fail

etc.

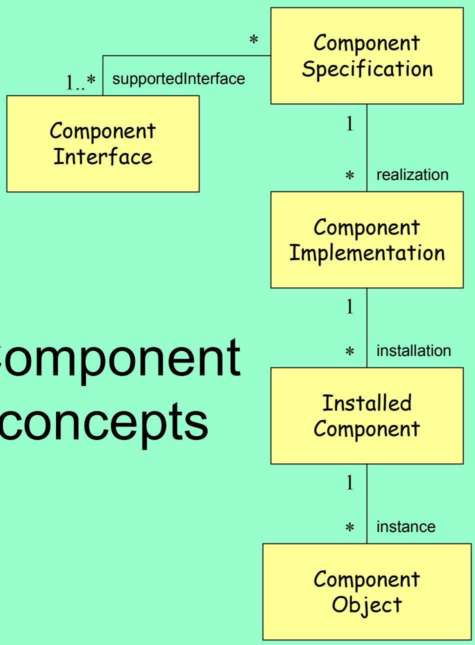
Tutorial Map

- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

Component Identification

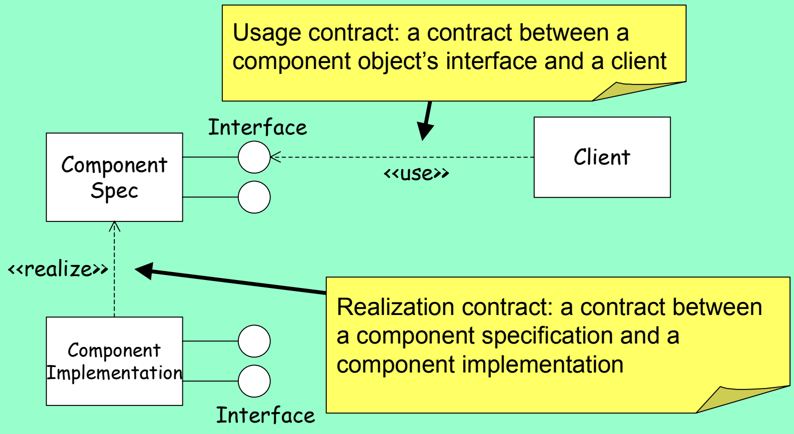


Component concepts

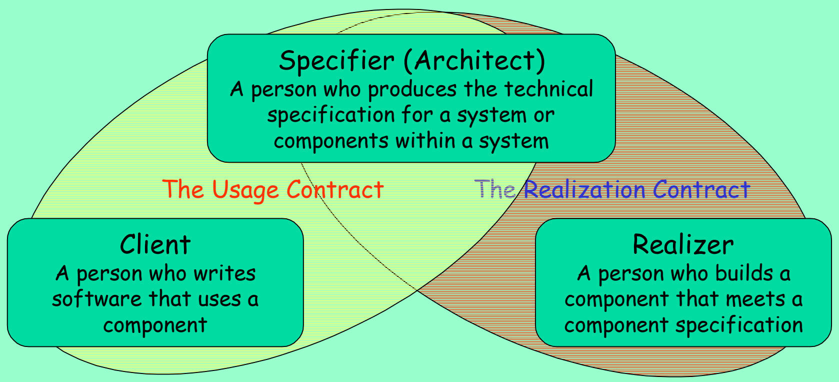


- Component Specification
  - The specification of a unit of software that describes the behaviour of a set of objects, and defines a unit of implementation and deployment
- Component Interface
  - A definition of a set of behaviours that can be offered by a Component Object
- Component Implementation
  - A realization of a Component Specification
- Installed Component
  - An installed (or deployed) copy of a Component Implementation
- Component Object
  - An instance of an Installed Component. A run-time concept

Two distinct contracts



Contracts and roles



Interfaces vs Component Specs

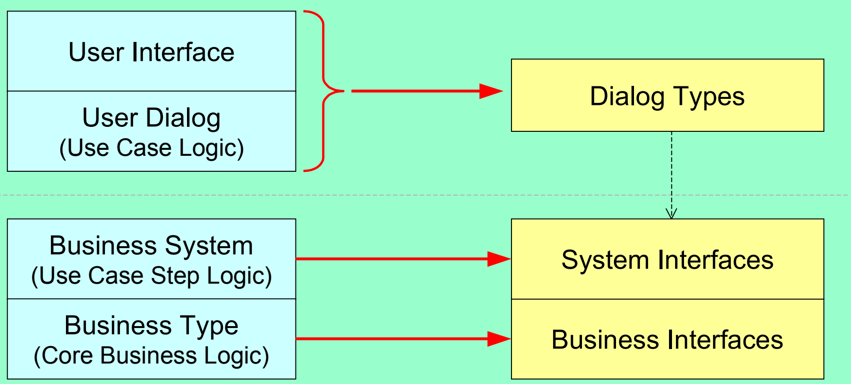
Component Interface

- 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

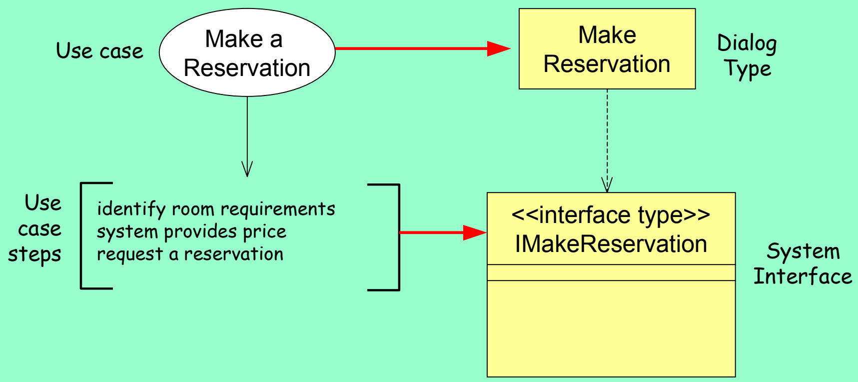
Component Specification

- 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

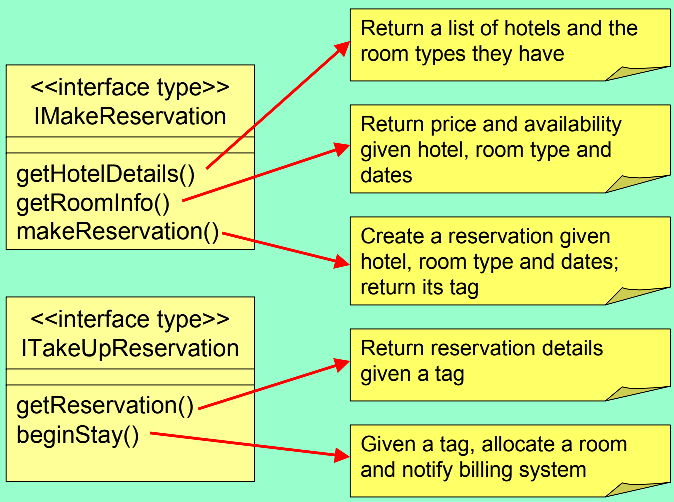
Mapping the architecture layers to software specifications



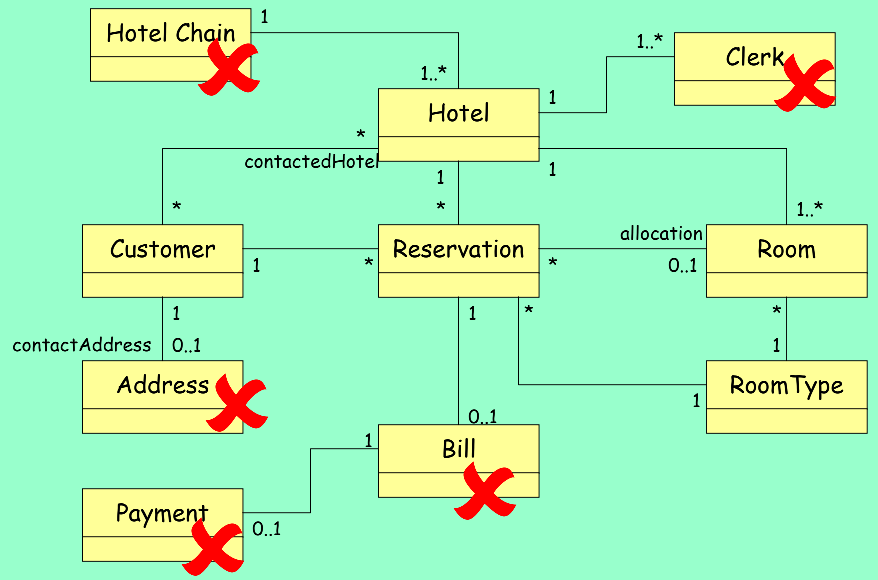
Identify System Interfaces and Operations



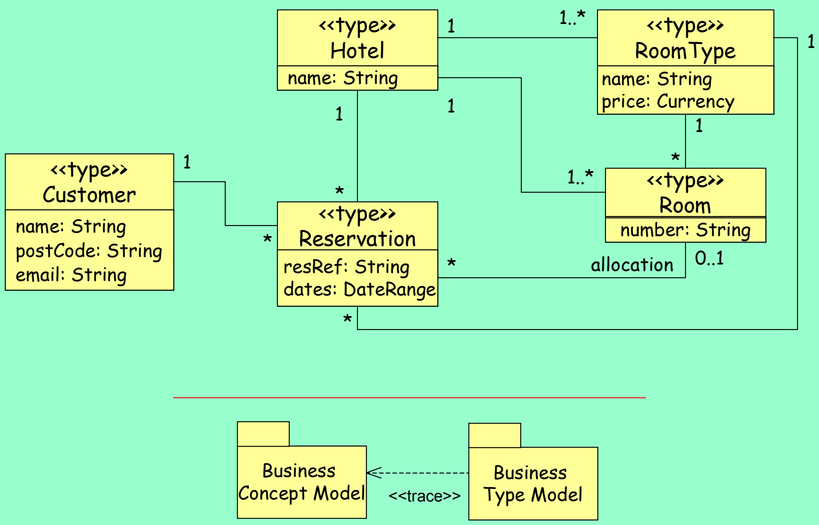
Use case step operations



Develop the Business Type Model



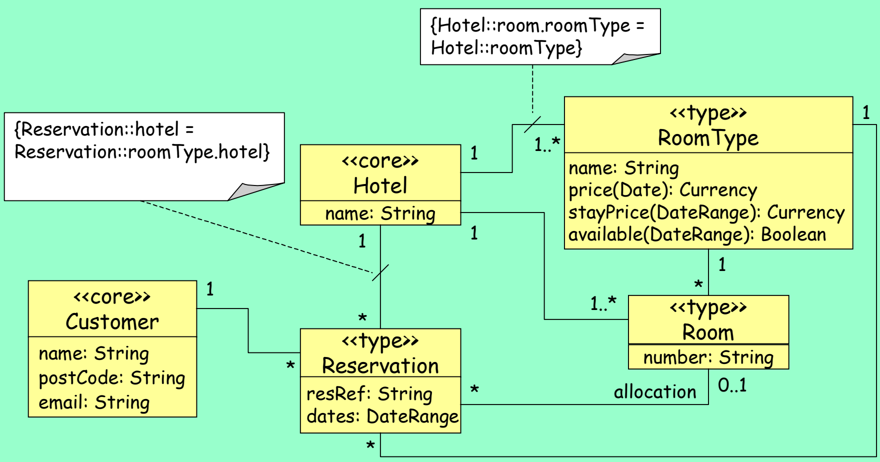
Initial Business Type Diagram



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

BTM with core types and constraints



Business rules in the BTM

context RoomType

-- AVAILABILITY RULES

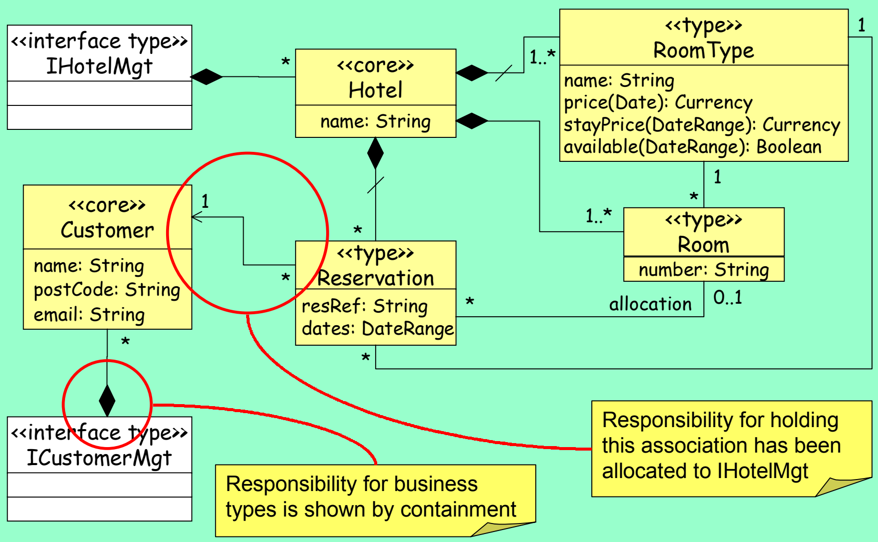
-- a room is available if the number of rooms reserved on all dates
-- in the range is less than the number of rooms
available(dr) = dr.asSet->collect(d | reservation->select(r | r.allocation->isEmpty and
    r.dates.includes(d))->size)->max < room->size)

-- can never have more reservations for a date than rooms (no overbooking)
Date->forAll(d | reservation->select(r | not r.allocation->isEmpty and
    r.dates.includes(d))->size) <= room->size

-- PRICING RULES

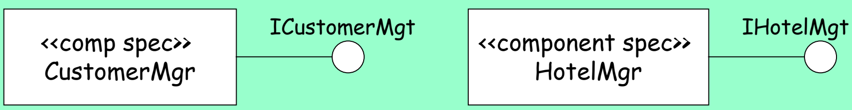
-- the price of a room for a stay is the sum of the prices for the days in the stay
stayPrice(dr) = dr.asSet->collect(d | price(d))->sum

Identify business interfaces: The Interface Responsibility Diagram



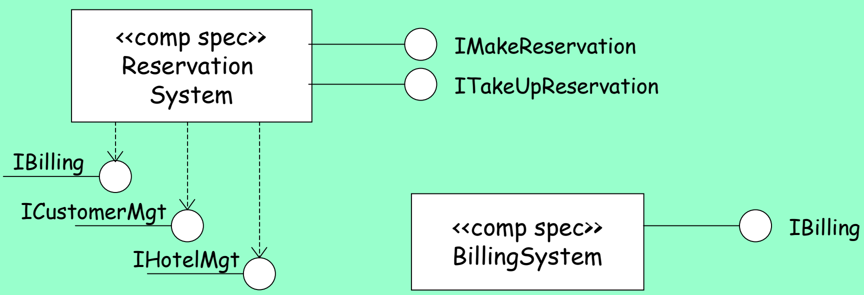
Component Specifications

- 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

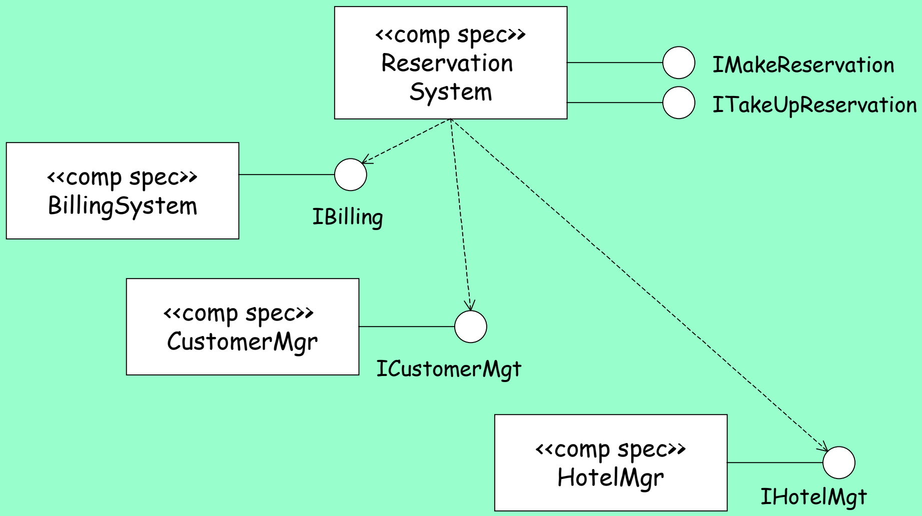


System components

- 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
- Separate component spec for billing system wrapper



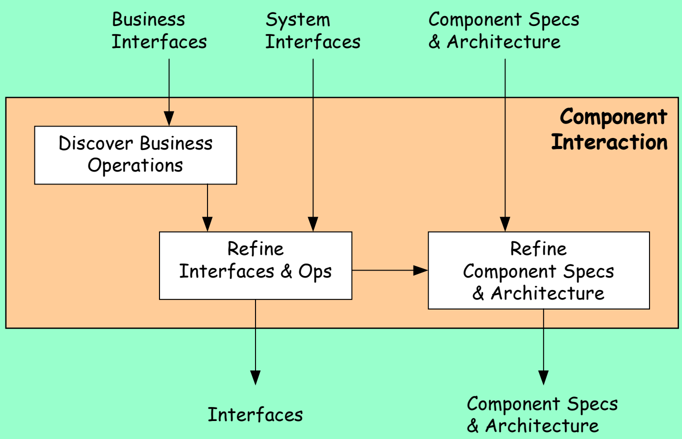
Initial component architecture



Tutorial Map

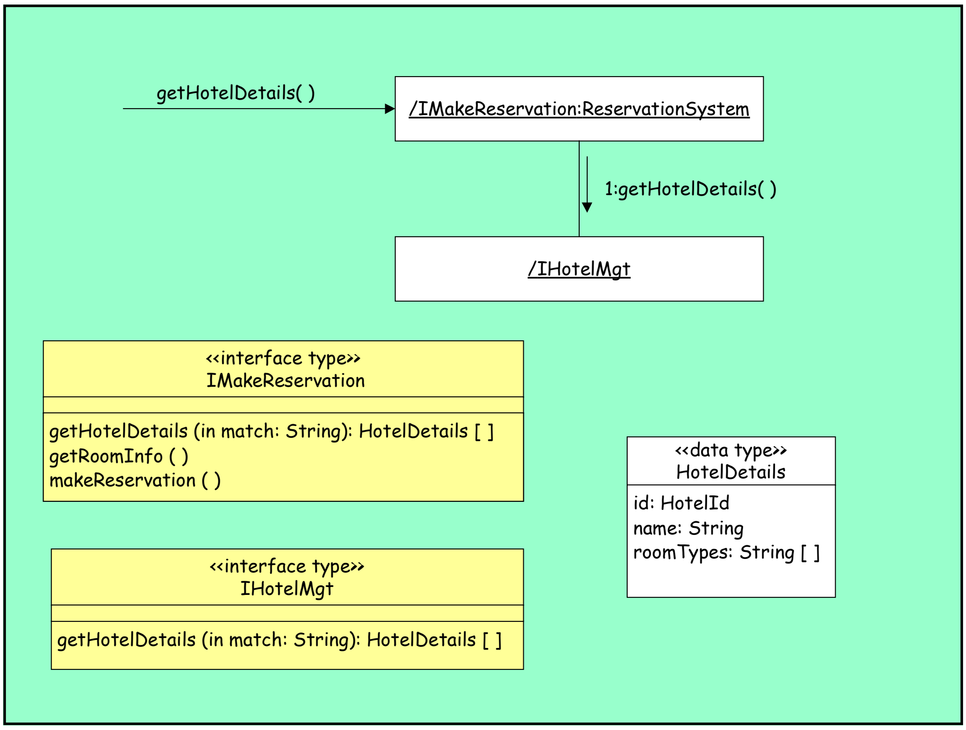
- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

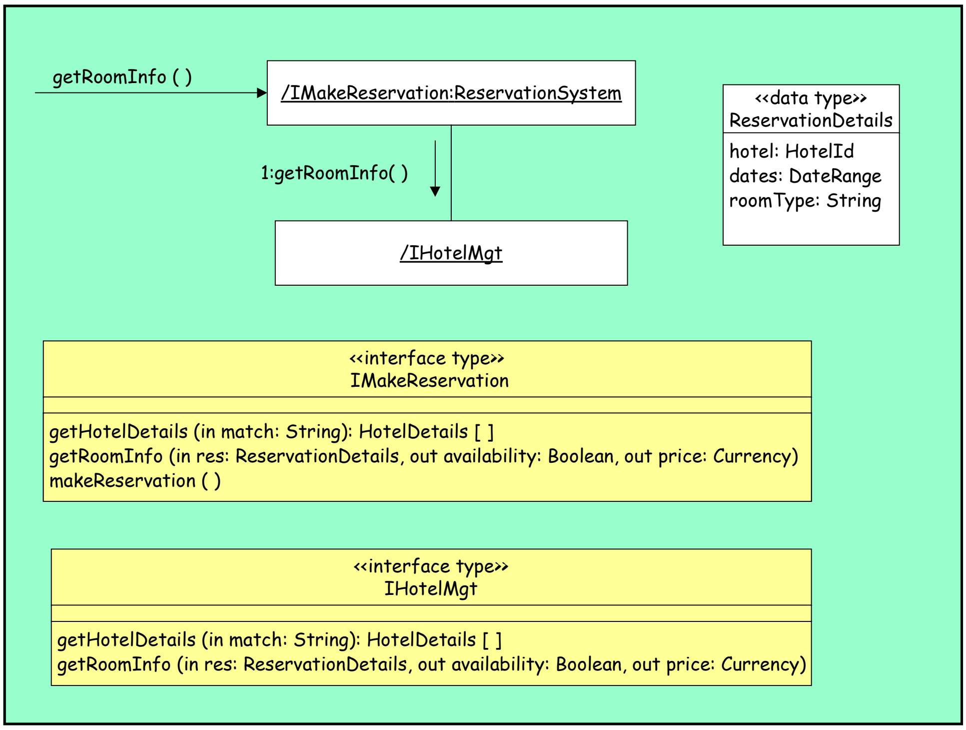
Component Interaction

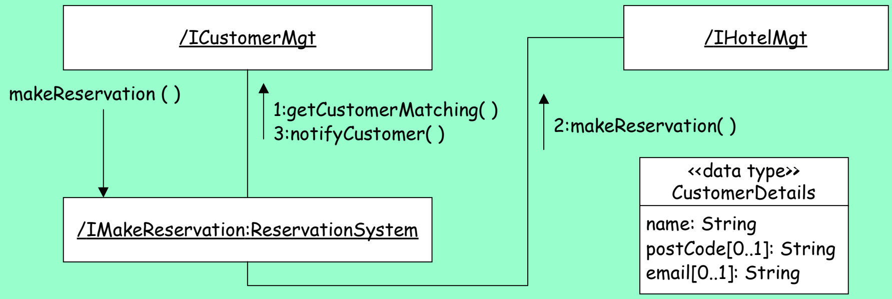


Operation discovery

- 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


getHotelDetails( )
/IMakeReservation:ReservationSystem
1:getHotelDetails( )
/IHotelMgt
<<interface type>>
IMakeReservation
getHotelDetails (in match: String): HotelDetails [ ]
getRoomInfo ( )
makeReservation ( )
<<data type>>
HotelDetails
id: HotelId
name: String
roomTypes: String [ ]
<<interface type>>
IHotelMgt
getHotelDetails (in match: String): HotelDetails [ ]


getRoomInfo ( )
/IMakeReservation:ReservationSystem
<<data type>>
ReservationDetails
hotel: HotelId
dates: DateRange
roomType: String
1:getRoomInfo( )
/IHotelMgt
<<interface type>>
IMakeReservation
getHotelDetails (in match: String): HotelDetails [ ]
getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)
makeReservation ( )
<<interface type>>
IHotelMgt
getHotelDetails (in match: String): HotelDetails [ ]
getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)



<<interface type>>
IMakeReservation

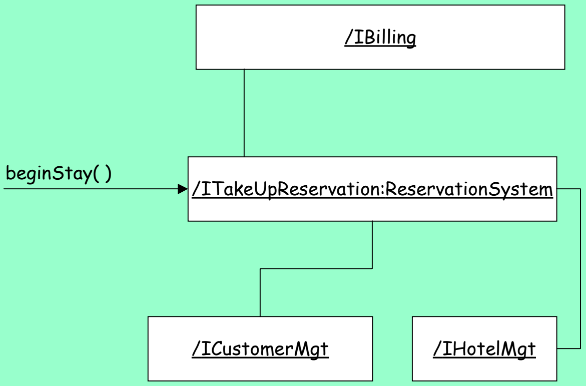
getHotelDetails (in match: String): HotelDetails [ ]
getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)
makeReservation (in res: ReservationDetails, in cus: CustomerDetails, out resRef: String): Integer

<<interface type>>
IHotelMgt

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

Exercise 2

- Specify the interaction for beginStay( )
- Complete the collaboration diagram on the next slide
- Add operations to the interface types on the slide after



<<interface type>>
ITakeUpReservation

beginStay (in resRef: String, out roomNumber: String): Boolean
-- result is true if room allocated successfully

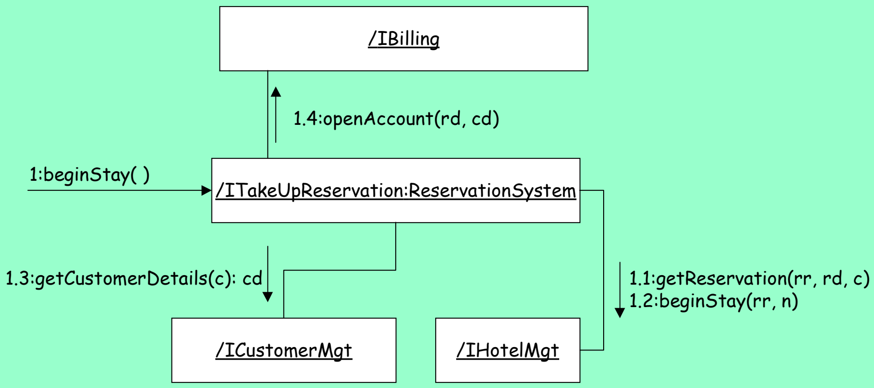
<<interface type>>
IHotelMgt

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

<<interface type>>
ICustomerMgt

getCustomerMatching (in custD: CustomerDetails, out cusId: CustId): Integer
createCustomer(in custD: CustomerDetails, out cusId: CustId): Boolean
notifyCustomer (in cus: CustId, in msg: String)

<<interface type>>
IBilling



<<interface type>>
ITakeUpReservation

beginStay (in resRef: String, out roomNumber: String): Boolean
-- result is true if room allocated successfully

<<interface type>>
IHotelMgt

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

<<interface type>>
ICustomerMgt

getCustomerMatching (in custD: CustomerDetails, out cusId: CustId): Integer
createCustomer(in custD: CustomerDetails, out cusId: CustId): Boolean
notifyCustomer (in cus: CustId, in msg: String)
getCustomerDetails (in cus: CustId): CustomerDetails

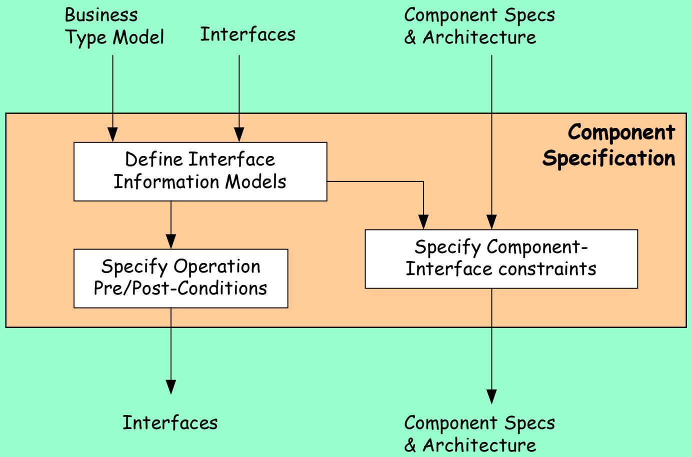
<<interface type>>
IBilling

openAccount (in res: ReservationDetails, in cus: CustomerDetails)

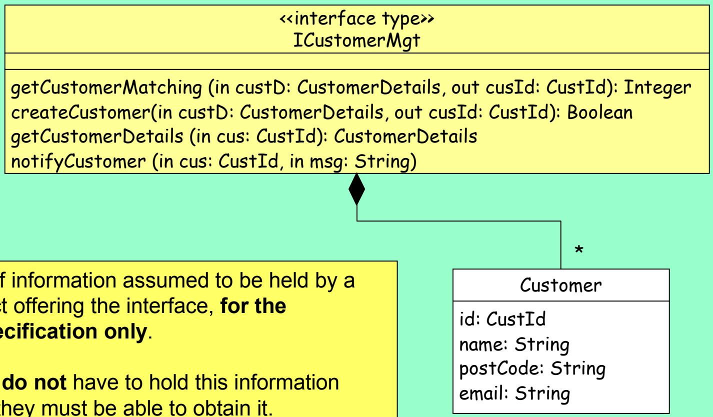
Tutorial Map

- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

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.

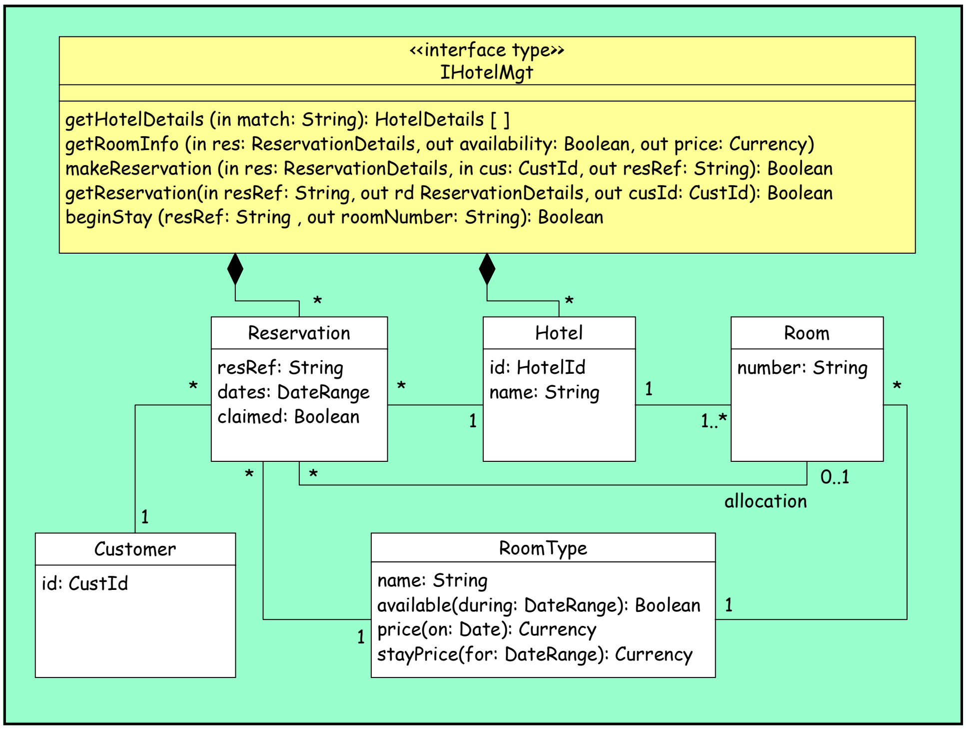
The model can be derived from the Business Type Model.

Pre- and post-conditions

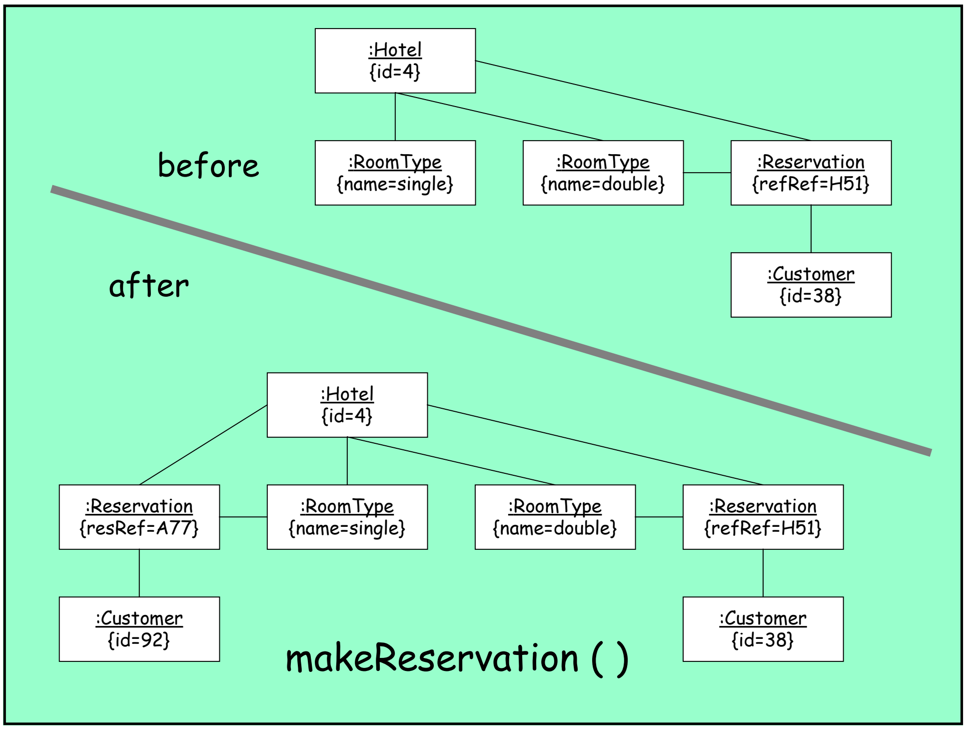
- If the pre-condition is true, the post-condition must be true
- If the pre-condition is false, the post-condition doesn't apply
- A missing pre-condition is assumed 'true'
- Pre- and post-conditions can be written in natural language or in a formal language such as OCL

```
context ICustomerMgt::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
```

```
context ICustomerMgt::createCustomer (in custD: CustomerDetails, out cusId: CustId): Boolean

pre:
    -- post code and email address must be provided
    custD.postCode->notEmpty and custD.email->notEmpty

post:
    result implies
        -- new customer (with name not previously known) created
        (not customer@pre->exists(c | c.name = custD.name)) and
        (customer - customer@pre)->size = 1 and
            Let c = (customer - customer@pre) in
                c.name = custD.name and c.postCode = custD.postCode and
                c.email = custD.email and c.id = cusId
```

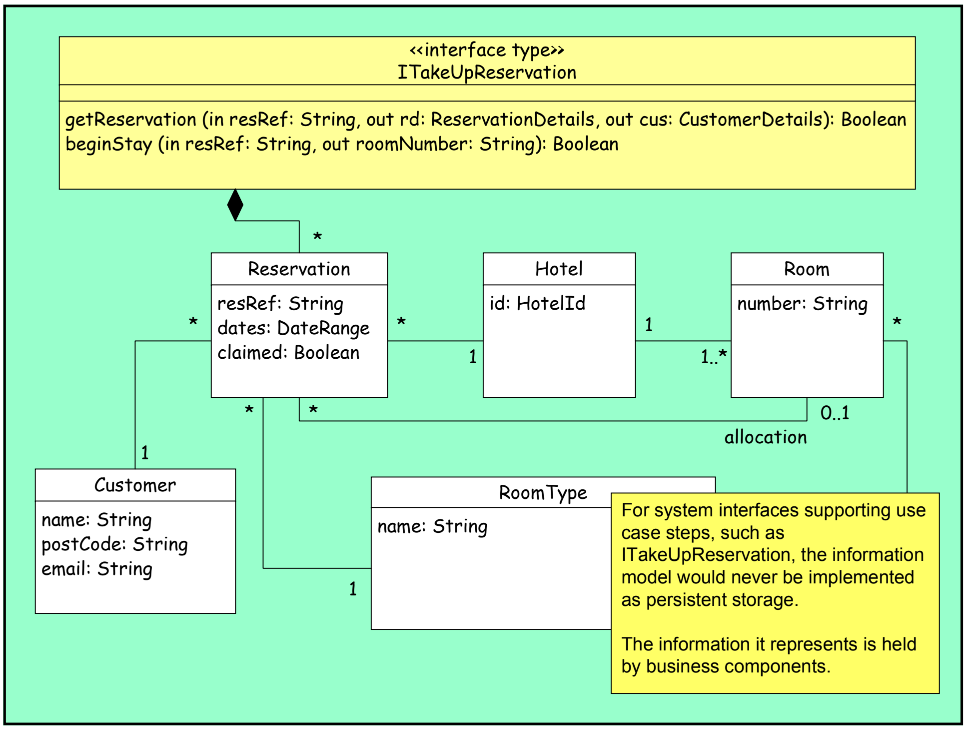


John Daniels - Component-based Design



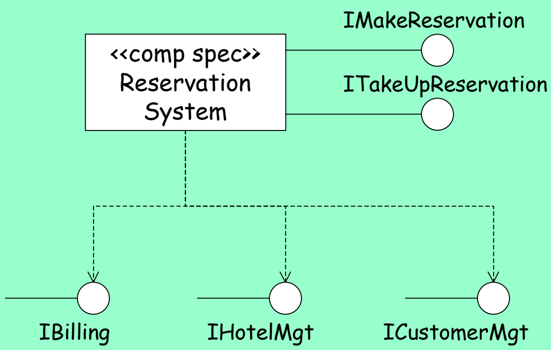

```
context makeReservation (in res: ReservationDetails, in cus: CustId, out resRef: String): Boolean

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
```

```
context IHotelMgt::beginStay (in resRef: String, out roomNumber: String): Boolean

pre:
    -- resRef is valid
    reservation->exists (r | r.resRef = resRef) and
    -- not already claimed
    not reservation->exists (r | r.resRef = resRef and r.claimed)

post:
    Let res = reservation->select (r | r.resRef = resRef) in
    result implies
        -- the reservation is now claimed
        res.claimed and
        roomNumber = res.allocation.number
        -- nb room allocation policy not defined
```



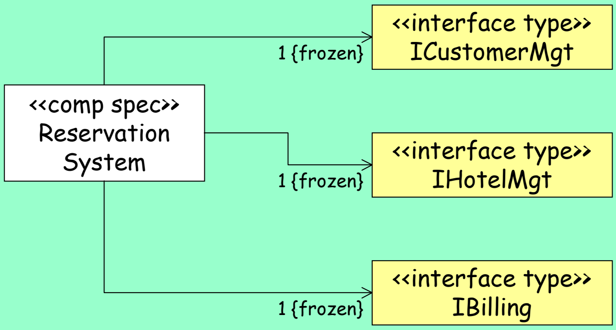
```
context ITakeUpReservation::beginStay (in resRef: String, out roomNumber: String): Boolean

pre:
    -- resRef is valid
    reservation->exists (r | r.resRef = resRef) and
    -- not already claimed
    not reservation->exists (r | r.resRef = resRef and r.claimed)

post:
    Let res = reservation->select (r | r.resRef = resRef) in
    result implies
        -- the reservation is now claimed
        res.claimed and
        roomNumber = res.allocation.number
        -- nb room allocation policy not defined here
```

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
IMakeReservation::hotel corresponds to ITakeUpReservation::hotel
IMakeReservation::reservation corresponds to ITakeUpReservation:: reservation
IMakeReservation::customer corresponds to ITakeUpReservation::customer

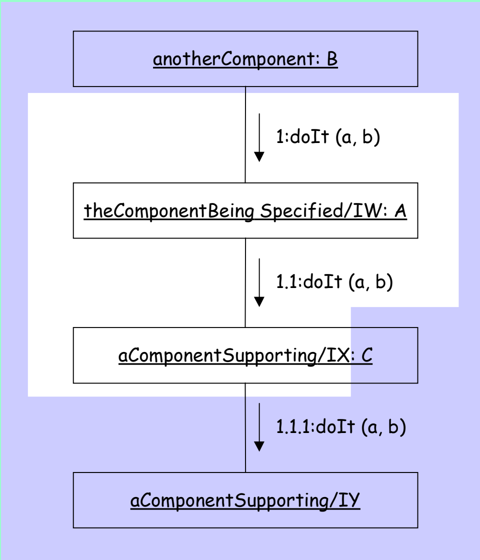
-- between offered interfaces and used interfaces
IMakeReservation::hotel corresponds to iHotelMgt.hotel
IMakeReservation::reservation corresponds to iHotelMgt.reservation
IMakeReservation::customer corresponds to iCustomerMgt.customer

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.

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.

Tutorial Map

- Introduction
- Design Process
- Requirements Definition
- Component Identification
- Component Interaction
- Component Specification
- Provisioning

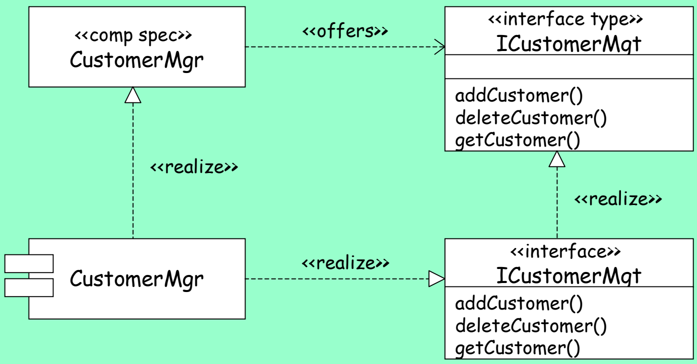
Provisioning

- Target technology

Component Standard	Platform Dependencies	Language Dependencies
Microsoft COM+	Windows 2000	None
Enterprise Java Beans	None	Java

- CORBA Component Model?

Realization mappings and restrictions



- Operation parameters
- Error handling
- Interface inheritance and support
- Architecture mappings

Operation parameters

- All parameters are either:
  - passed by value, or
  - references to component objects
- In EJB:
  - All parameters must be “in”
  - The parameters must obey RMI rules (base or serializable)
- For COM+:
  - If using COM automation the parameters must be VBA types

Error handling

- COM+ uses standard result structure
- EJB uses Java exceptions
  - Need to establish a policy:
    - exceptions correspond to defined result states, or
    - exceptions correspond to undefined results
  - If exceptions imply defined states:
    - use multiple post-conditions (Soundarajan & Fridella, UML99)

```
context addItem(p: IProduct, quantity: integer): void
pre: quantity >=0
post: not orderLine@pre->exists(o | o.product = p) and
          (orderLine - orderLine@pre)-> size = 1 and
          (orderLine - orderLine@pre)->exists(o |
                    o.product = p and o.quantity = quantity)
bi:BadItem.post: orderLine@pre->exists(o | o.product = p) and
          bi.originator = self and bi.errorString = “item already in order”
```

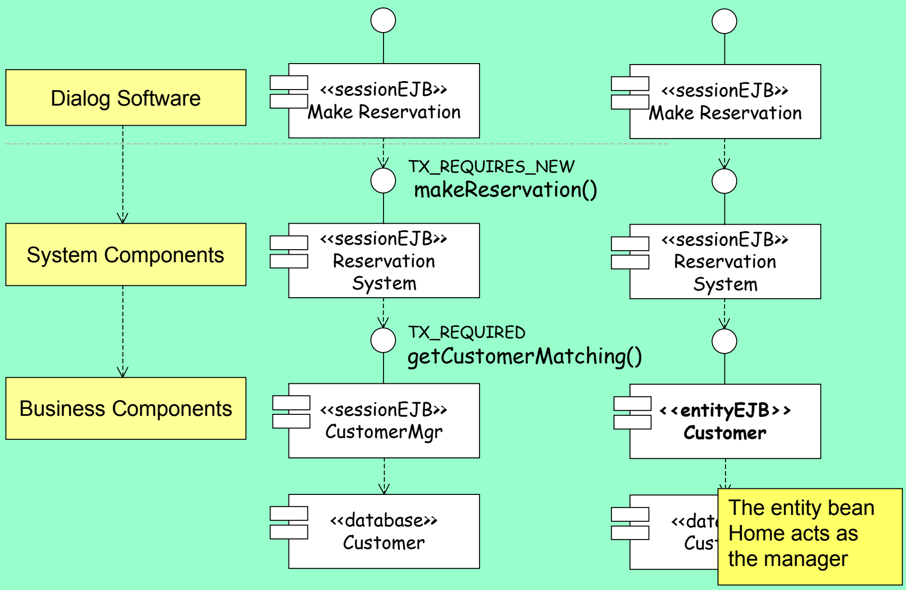
Normal post-condition, no previous line for this product

Post-condition that applies when BadItem exception is raised

Interface support and inheritance

- EJB:
  - each component offers one interface
  - interfaces can have multiple inheritance
  - therefore: use inheritance to offer multiple interfaces
  - beware clashes!
- COM+
  - each component can offer many interfaces
  - interfaces can have single inheritance

Architecture mappings



Want to know more?

- Forthcoming book by John Cheesman and John Daniels, Addison-Wesley, October 2000