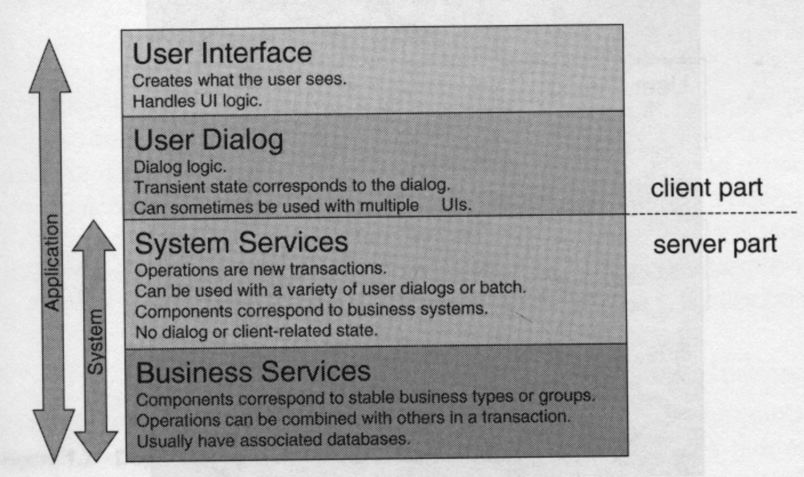
A Process for Specifying Component-Based Software

by Cheesman and Daniels (2001)

Motivation

- Major challenge in software engineering today: **manage change**
- For Cheesman and Daniels, the objective of component reuse is of less importance.
- Aim: provide advice, guidance, and examples for modeling enterprise-scale component systems.

Architectural layers



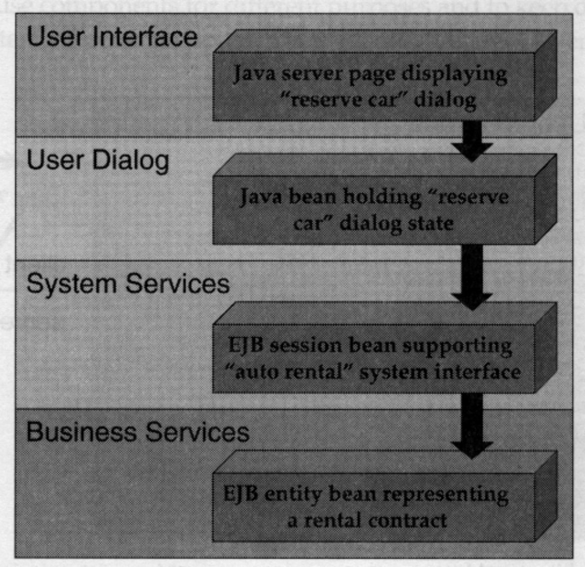
User Interface
Creates what the user sees.
Handles UI logic.

User Dialog
Dialog logic.
Transient state corresponds to the dialog.
Can sometimes be used with multiple UIs.

System Services
Operations are new transactions.
Can be used with a variety of user dialogs or batch.
Components correspond to business systems.
No dialog or client-related state.

Business Services
Components correspond to stable business types or groups.
Operations can be combined with others in a transaction.
Usually have associated databases.

Example components in the layers



User Interface

Java server page displaying "reserve car" dialog

User Dialog

Java bean holding "reserve car" dialog state

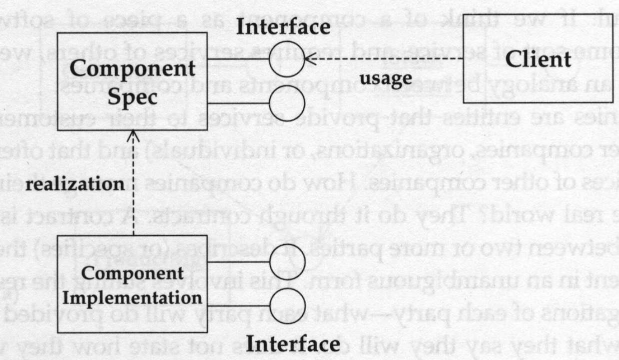
System Services

EJB session bean supporting "auto rental" system interface

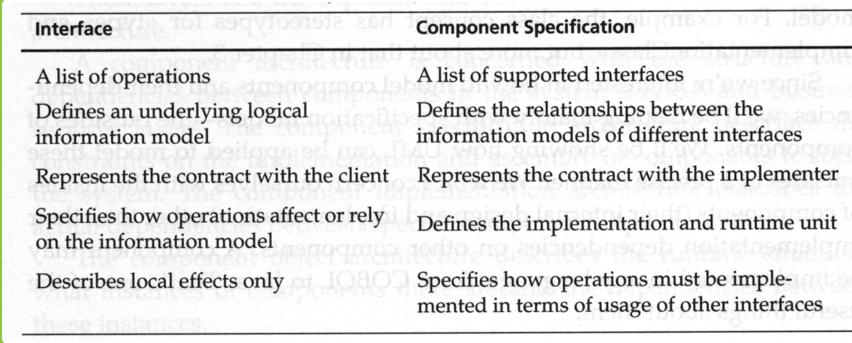
Business Services

EJB entity bean representing a rental contract

Realization vs. usage contracts

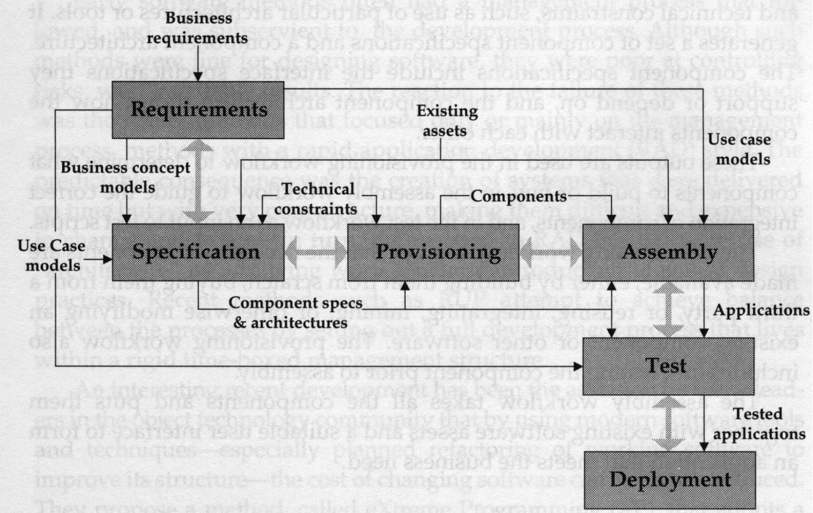


Interface- vs. component specification

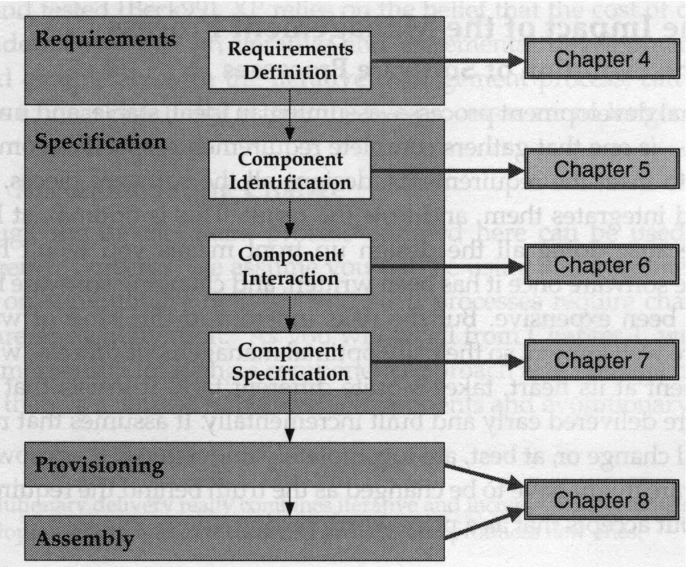


Interface	Component Specification
A list of operations	A list of supported interfaces
Defines an underlying logical information model	Defines the relationships between the information models of different interfaces
Represents the contract with the client	Represents the contract with the implementer
Specifies how operations affect or rely on the information model	Defines the implementation and runtime unit
Describes local effects only	Specifies how operations must be implemented in terms of usage of other interfaces

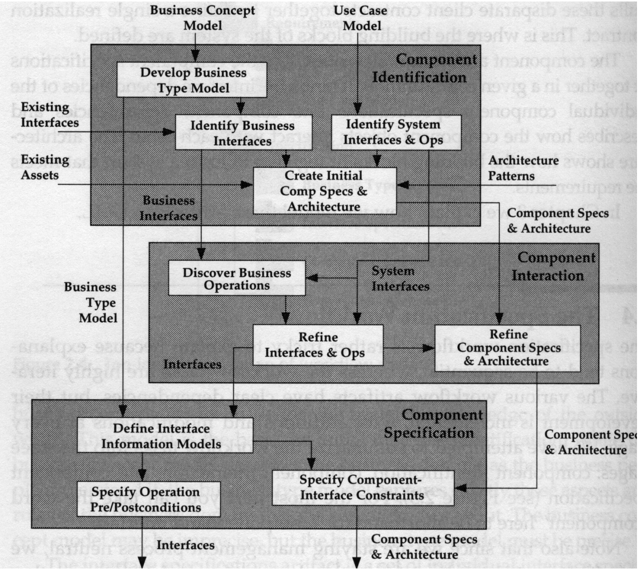
Workflow of the overall development process



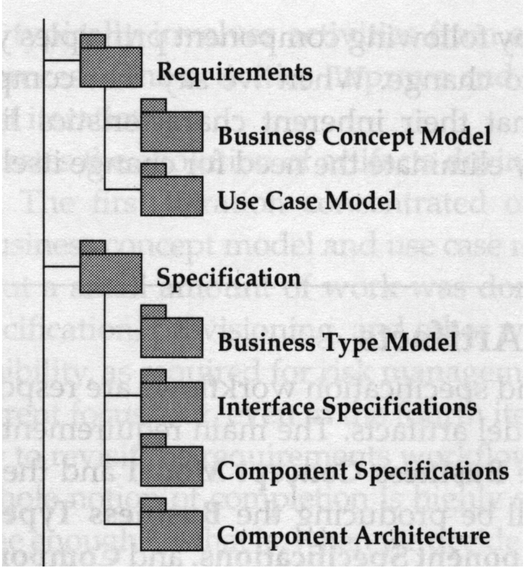
Stages of the process



Stages of the specification workflow

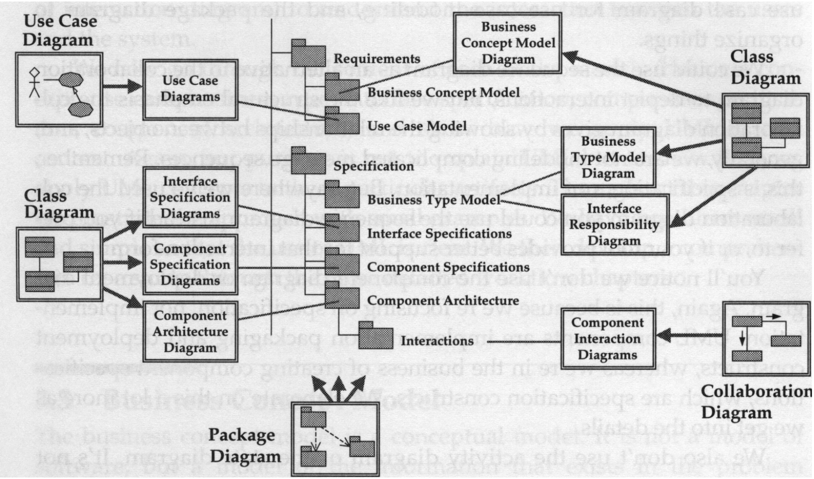

Business Concept Model
Use Case Model
Component Identification
Develop Business Type Model
Existing Interfaces
Identify Business Interfaces
Identify System Interfaces & Ops
Existing Assets
Create Initial Comp Specs & Architecture
Architecture Patterns
Business Interfaces
Component Specs & Architecture
Component Interaction
Discover Business Operations
System Interfaces
Business Type Model
Refine Interfaces & Ops
Refine Component Specs & Architecture
Interfaces
Component Specification
Define Interface Information Models
Component Spe
& Architecture
Specify Component-Interface Constraints
Specify Operation Pre/Postconditions
Interfaces
Component Specs & Architecture

Models to be produced



- Requirements
  - Business Concept Model
  - Use Case Model
- Specification
  - Business Type Model
  - Interface Specifications
  - Component Specifications
  - Component Architecture

Notations used



Use Case Diagram

Use Case Diagrams

Class Diagram

Interface Specification Diagrams

Component Specification Diagrams

Component Architecture Diagram

Package Diagram

Requirements

Business Concept Model

Use Case Model

Specification

Business Type Model

Interface Specifications

Component Specifications

Component Architecture

Interactions

Business Concept Model Diagram

Class Diagram

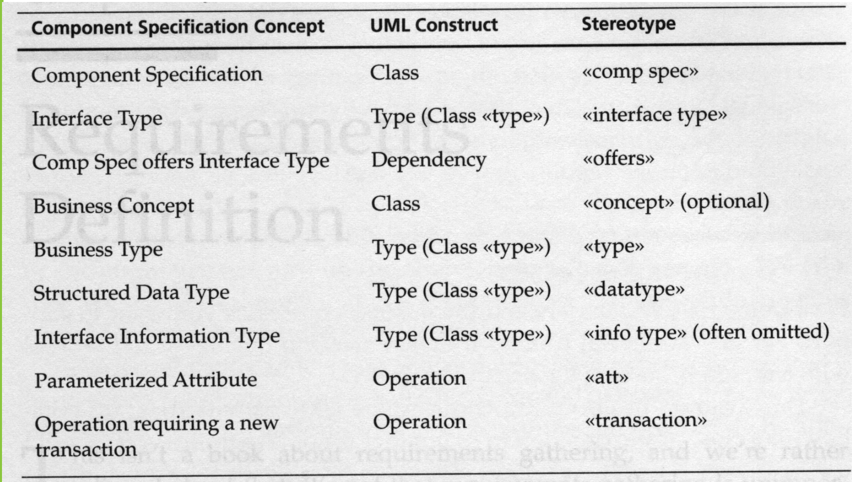
Business Type Model Diagram

Interface Responsibility Diagram

Component Interaction Diagrams

Collaboration Diagram

Summary of UML extensions



Component Specification Concept	UML Construct	Stereotype
Component Specification	Class	«comp spec»
Interface Type	Type (Class «type»)	«interface type»
Comp Spec offers Interface Type	Dependency	«offers»
Business Concept	Class	«concept» (optional)
Business Type	Type (Class «type»)	«type»
Structured Data Type	Type (Class «type»)	«datatype»
Interface Information Type	Type (Class «type»)	«info type» (often omitted)
Parameterized Attribute	Operation	«att»
Operation requiring a new transaction	Operation	«transaction»

Requirements Definition

Requirements definition: overview

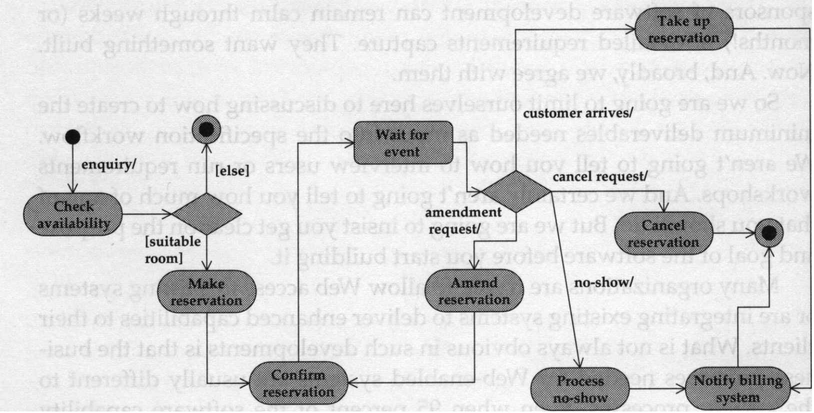
1. Business process
2. Business concept model
3. System envisioning
4. Use cases
   4.1. Actors and roles
   4.2. Use case identification
   4.3. Use case descriptions
   4.4. Quality of service

Business process

- Business process to be supported must be understood
- Its description is **not** a statement of the requirements for the IT system (software)
- Notation: e.g., UML activity diagrams

Example of a business process

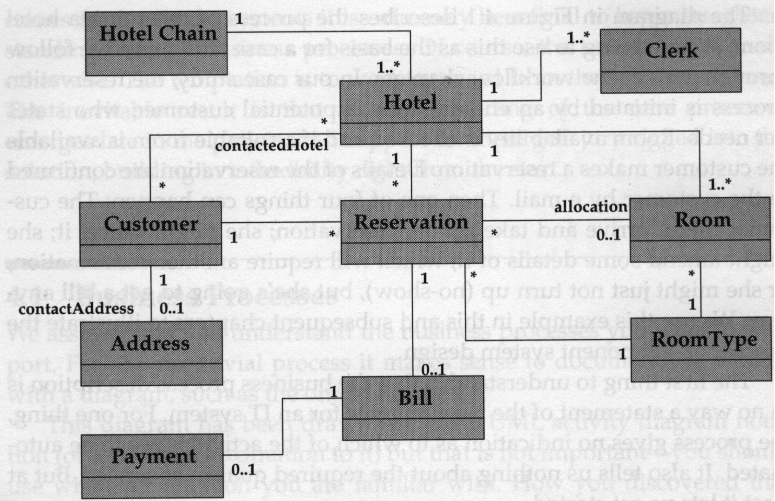
Running example: hotel reservation



Business concept model

- Expresses domain knowledge about the application domain; thus, it is not related to software.
- Does not need to be tightly scoped to the problem
- Notation: UML class diagrams

Example of a business concept model



System envisioning I

Define the software boundary; make clear which functions are the responsibility of the software.

Example:

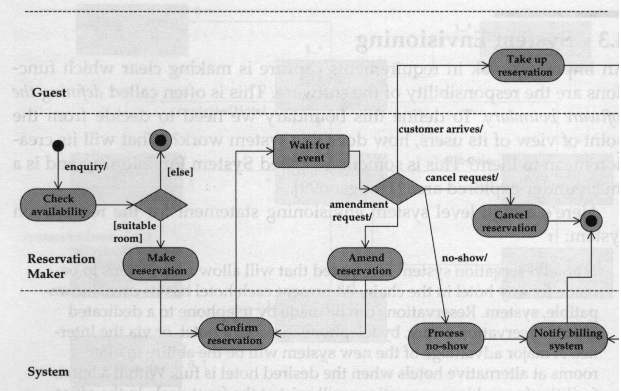
> *A hotel reservation system is required that will allow reservations to be made for any hotel in the chain. At present each hotel has its own, incompatible, system. Reservations can be made by telephone to a dedicated central reservation center, by telephone direct to a hotel, or via the Internet. A major advantage of the new system will be the ability to offer rooms at alternative hotels when the desired hotel is full.*

System envisioning II

Within a hotel, facilities for making reservations will exist at the front desk, in the office, and at the concierge's desk. Each hotel has a reservation administrator who is responsible for controlling reservations at the hotel, but any authorized user may make a reservation. The target time for making a reservation by telephone or in person is three minutes. To speed up the process, details of previous customers will be stored and made available.

Use cases

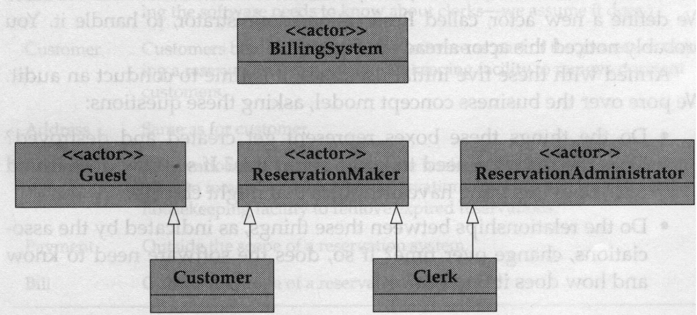
Allocate responsibility for the business process steps. Notation: swim lanes.



Note: responsibility decisions have a profound effect on the shape of the resulting software. They are often taken too quickly.

Actors and roles

- Actors are **roles** that initiate and control the steps assigned to them, even though the software may play a part in these steps.
- To be flexible, generalization relations can be introduced.



Use case identification I

Use cases

- describe the interaction of actors with the software
- are a functional specification of the software
- define the boundary between the software and its environment
- **describe the interaction that follows from a single business event**

Use case identification II

Hotel example: five events (corresponding to five use cases)

1. Make Reservation (covering Check Availability, Make Reservation, and Confirm Reservation steps)
2. Cancel Reservation
3. Amend Reservation (covering Amend Reservation and Confirm Reservation)
4. Take Up Reservation (covering Take Up Reservation and Notify Billing System)
5. Process No-Show (covering Process No-Show and Notify Billing System)

Use case identification III

Discussion

- Not turning up is a bit of a noevent. A business rule must define when the no-show event is generated, e.g. no arrival until 8 p.m.
- The processing of no-shows can either be triggered by a clock and be performed automatically, or be initiated by a user (which is chosen here).
- Therefore, the use case is renamed Process No-Shows, because it deals with all reservations that meet the no-show business rule.
- But who is the corresponding actor?
  Introduce ReservationAdministrator (see above)

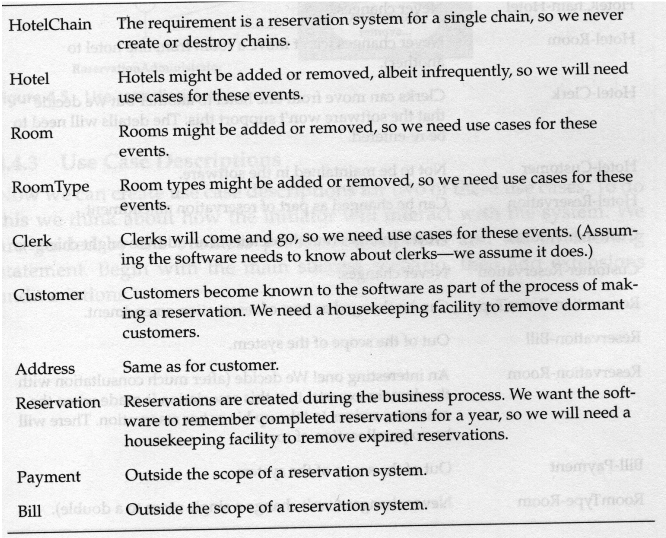
Use case identification IV

Audit

Considering the business concept model, answer the following questions:

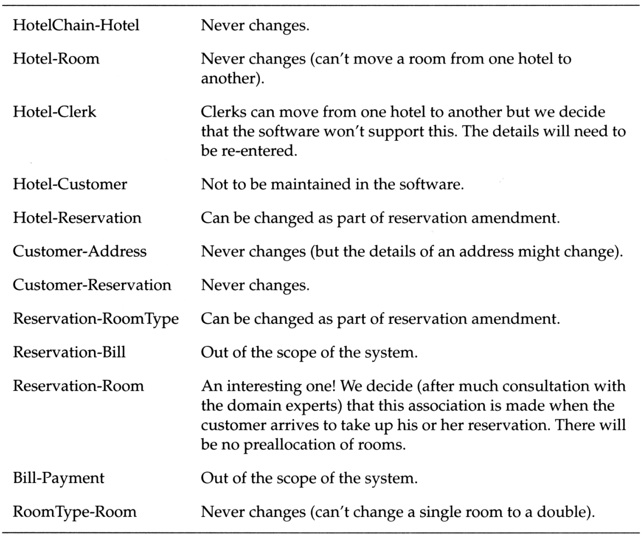
- about the classes
  - Do the things these boxes represent get created and destroyed?
  - Does the software need to know about this?
  - If so, how does it find out?
  - Does this thing have attributes that might change?
- about the associations
  - Do the relationships between these things change over time?
  - If so, does the software need to know and how does it find out?

Use case identification V



HotelChain	The requirement is a reservation system for a single chain, so we never create or destroy chains.
Hotel	Hotels might be added or removed, albeit infrequently, so we will need use cases for these events.
Room	Rooms might be added or removed, so we need use cases for these events.
RoomType	Room types might be added or removed, so we need use cases for these events.
Clerk	Clerks will come and go, so we need use cases for these events. (Assuming the software needs to know about clerks—we assume it does.)
Customer	Customers become known to the software as part of the process of making a reservation. We need a housekeeping facility to remove dormant customers.
Address	Same as for customer.
Reservation	Reservations are created during the business process. We want the software to remember completed reservations for a year, so we will need a housekeeping facility to remove expired reservations.
Payment	Outside the scope of a reservation system.
Bill	Outside the scope of a reservation system.

Use case identification VI



HotelChain-Hotel	Never changes.
Hotel-Room	Never changes (can't move a room from one hotel to another).
Hotel-Clerk	Clerks can move from one hotel to another but we decide that the software won't support this. The details will need to be re-entered.
Hotel-Customer	Not to be maintained in the software.
Hotel-Reservation	Can be changed as part of reservation amendment.
Customer-Address	Never changes (but the details of an address might change).
Customer-Reservation	Never changes.
Reservation-RoomType	Can be changed as part of reservation amendment.
Reservation-Bill	Out of the scope of the system.
Reservation-Room	An interesting one! We decide (after much consultation with the domain experts) that this association is made when the customer arrives to take up his or her reservation. There will be no preallocation of rooms.
Bill-Payment	Out of the scope of the system.
RoomType-Room	Never changes (can't change a single room to a double).

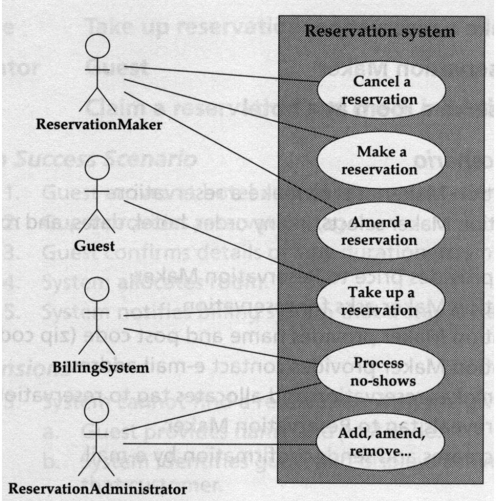
Use case identification VII

We assume that all the things in our model might have attributes that can change, so the full list of uses cases, so far as we know now, is as follows:

- Make Reservation
- Cancel Reservation
- Amend Reservation
- Take Up Reservation
- Process No-Shows
- Add/Amend/Remove Hotel
- Add/Amend/Remove Room
- Add/Amend/Remove Room Type
- Add/Amend/Remove Clerk
- Amend Customer
- Remove Dormant Customers
- Amend Address
- Remove Old Reservations

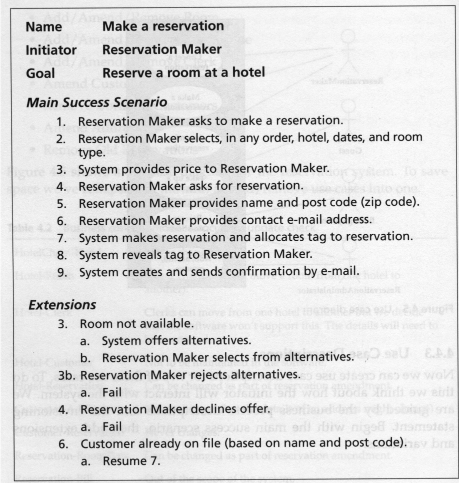
Use case identification VIII

Resulting use case diagram



Use case descriptions I

For each use case, describe main success scenario, then add extensions and variations.



Name **Make a reservation**

Initiator **Reservation Maker**

Goal **Reserve a room at a hotel**

Main Success Scenario

1. Reservation Maker asks to make a reservation.
2. Reservation Maker selects, in any order, hotel, dates, and room type.
3. System provides price to Reservation Maker.
4. Reservation Maker asks for reservation.
5. Reservation Maker provides name and post code (zip code).
6. Reservation Maker provides contact e-mail address.
7. System makes reservation and allocates tag to reservation.
8. System reveals tag to Reservation Maker.
9. System creates and sends confirmation by e-mail.

Extensions

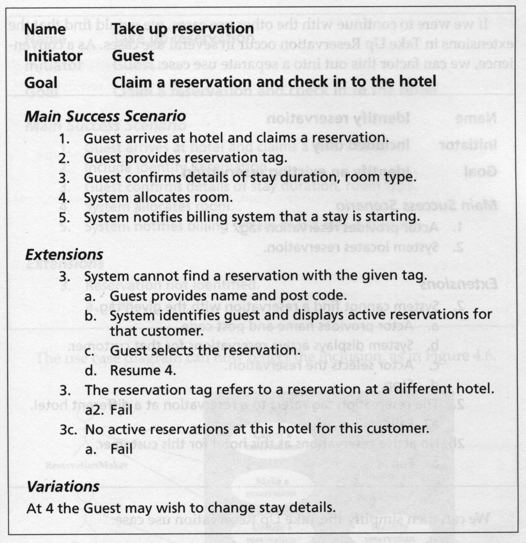
3. Room not available.
   a. System offers alternatives.
   b. Reservation Maker selects from alternatives.

3b. Reservation Maker rejects alternatives.
   a. Fail

4. Reservation Maker declines offer.
   a. Fail

6. Customer already on file (based on name and post code).
   a. Resume 7.

Use case descriptions II



Name	**Take up reservation**
Initiator	**Guest**
Goal	**Claim a reservation and check in to the hotel**

Main Success Scenario

1. Guest arrives at hotel and claims a reservation.
2. Guest provides reservation tag.
3. Guest confirms details of stay duration, room type.
4. System allocates room.
5. System notifies billing system that a stay is starting.

Extensions

3. System cannot find a reservation with the given tag.
   a. Guest provides name and post code.
   b. System identifies guest and displays active reservations for that customer.
   c. Guest selects the reservation.
   d. Resume 4.

3. The reservation tag refers to a reservation at a different hotel.
   a2. Fail

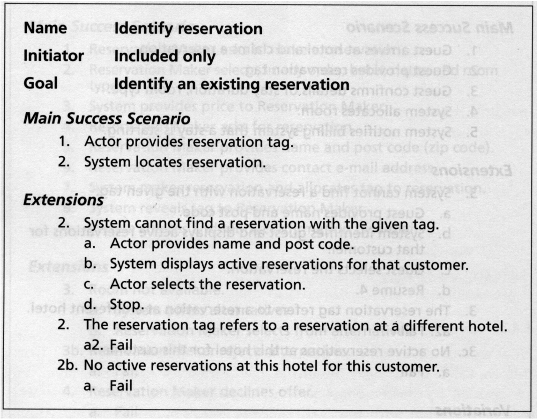
3c. No active reservations at this hotel for this customer.
   a. Fail

Variations

At 4 the Guest may wish to change stay details.

Use case descriptions III

If we were to continue with the other uses cases, we would find that the extensions in Take Up Reservation occur in several use cases. As a convenience, we can factor this out into a separate use case:



Name **Identify reservation**

Initiator **Included only**

Goal **Identify an existing reservation**

Main Success Scenario

1. Actor provides reservation tag.
2. System locates reservation.

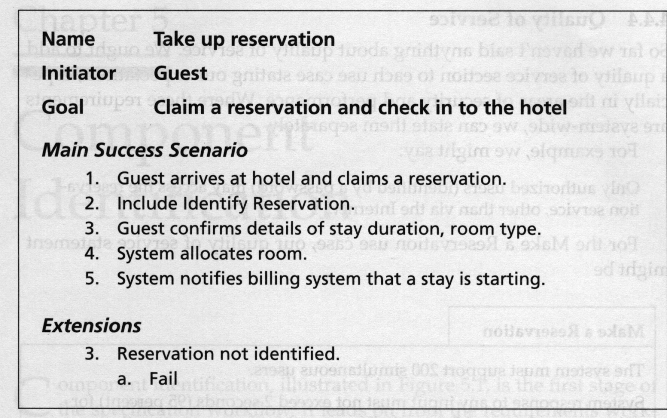
Extensions

2. System cannot find a reservation with the given tag.
   a. Actor provides name and post code.
   b. System displays active reservations for that customer.
   c. Actor selects the reservation.
   d. Stop.
2. The reservation tag refers to a reservation at a different hotel.
   a2. Fail

2b. No active reservations at this hotel for this customer.
   a. Fail

Use case descriptions IV

We can then simplify the Take Up Reservation use case:



Name **Take up reservation**

Initiator **Guest**

Goal **Claim a reservation and check in to the hotel**

Main Success Scenario

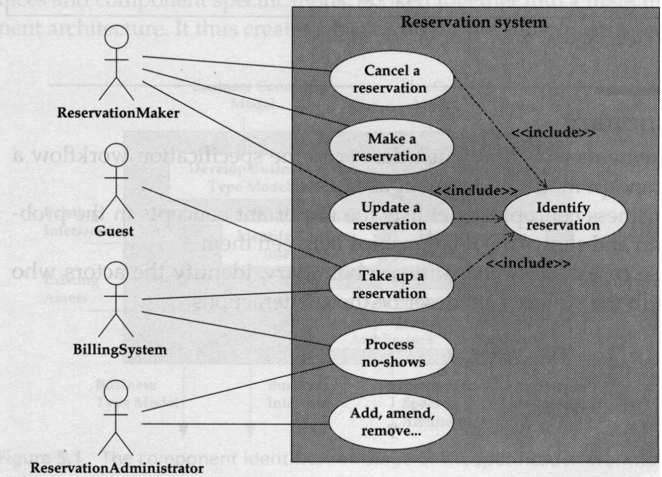
1. Guest arrives at hotel and claims a reservation.
2. Include Identify Reservation.
3. Guest confirms details of stay duration, room type.
4. System allocates room.
5. System notifies billing system that a stay is starting.

Extensions

3. Reservation not identified.
   a. Fail

Use case descriptions V

Final use case diagram:



Quality of service I

- We ought to add a quality of service section to each use case, stating our expectations, especially in the areas of security and performance.
- Where these requirements are system-wide, we can state them separately.
- For example, we might say:

> *Only authorized users (identified by a password) may access the reservation service, other than via the Internet.*

Quality of service II

- For the Make a Reservation use case, our quality of service statement might be

  > *The system must support 200 simultaneous users.*
  >
  > *System response to any input must not exceed 2 seconds (95 percent) for direct connections and 5 seconds (90 percent) for Internet connections.*
  >
  > *The system must support (total number of rooms) * 10 active reservations, and assume 100 percent hotel occupancy.*

Summary of requirements definition

- The requirements workflow must deliver to the specification workflow a business concept model and a set of use cases.
- The business concept model lists the important concepts in the problem domain and shows the relationships between them.
- The use cases clarify the software boundary, identify the actors who interact with the software, and describe those interactions.

Component identification stage of the specification workflow I

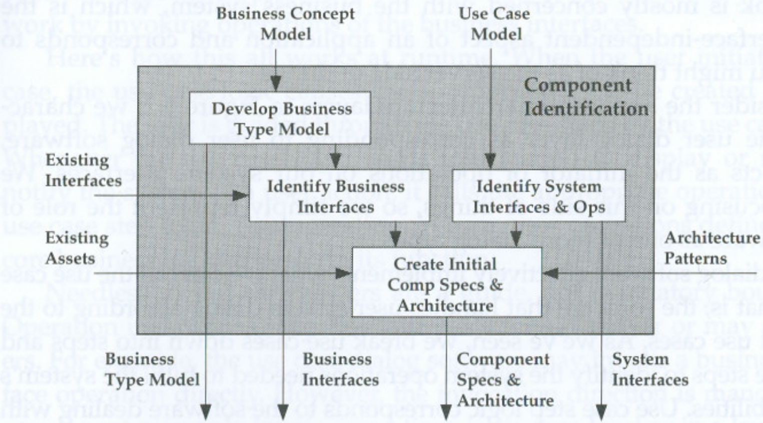


Figure 5.1 The component identification stage of the specification workflow

Component identification stage of the specification workflow II

- Goal: create an initial set of interfaces and component specifications, hooked together into a first-cut component architecture.
- Emphasis: discovery
  - What information needs to be managed?
  - What interfaces are needed to manage it?
  - What components are needed to provide that functionality?
  - How will they fit together?
- Identify the system interfaces and system components in the system services layer.
- Identify the business interfaces and business type components in the business services layer.
- Take into account existing interfaces, databases, or components that need to be interfaced with and that may need adapting.
- Try to apply architectural patterns.

Focus of interface identification

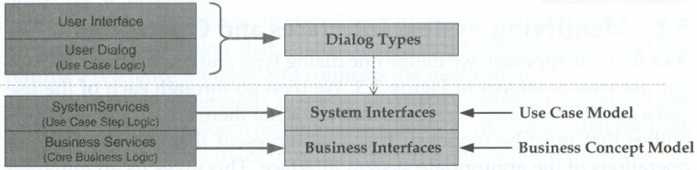


Figure 5.2 Interface inputs and correspondence to application architecture layers

- Process is concerned with the UI-independent aspects of an application, corresponding to the server side of things.
- Refine business concept model (representing human's eye view) into business type model (representing software's eye view).
- Use business type model to develop business interfaces.
- The implementations of components supporting these interfaces form the core business logic.

Runtime behavior

- When a user initiates a use case, the use case logic causes the appropriate UI to be created and displayed.
- The user is guided through the use case steps by the use case logic.
- Whenever the use case logic needs information to display or needs to notify the system of a user action, it calls the appropriate operation in the use case step logic.
- This operation, in turn, uses operations defined in the core business logic to perform its function.

Note: A component may only invoke operations on its own level or in a level below itself.

Identifying system interfaces and operations

- Identify one dialog type and one system interface per use case.
- Then go through each use case and for each step consider whether or not there are any system responsibilities that must be modeled.
- If so, represent them as one or more operations on the appropriate system interface.

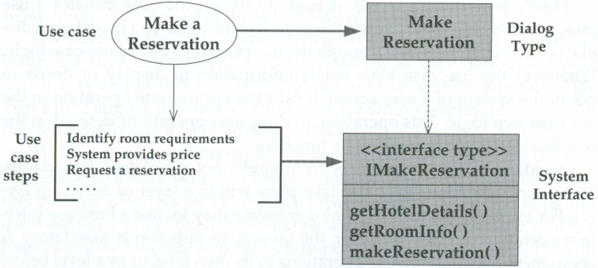
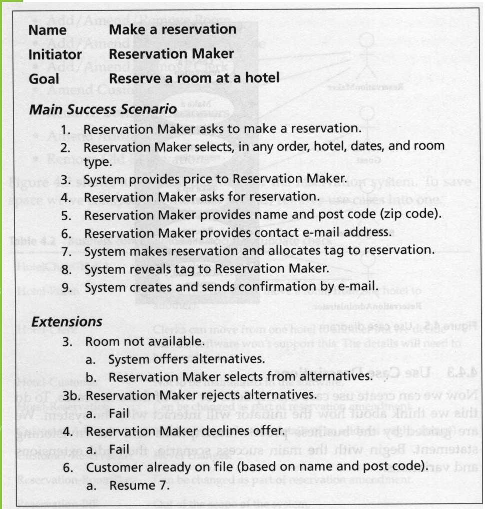


Figure 5.3 Use cases map to system interfaces

Example: Make a reservation



Name **Make a reservation**

Initiator **Reservation Maker**

Goal **Reserve a room at a hotel**

Main Success Scenario

1. Reservation Maker asks to make a reservation.
2. Reservation Maker selects, in any order, hotel, dates, and room type.
3. System provides price to Reservation Maker.
4. Reservation Maker asks for reservation.
5. Reservation Maker provides name and post code (zip code).
6. Reservation Maker provides contact e-mail address.
7. System makes reservation and allocates tag to reservation.
8. System reveals tag to Reservation Maker.
9. System creates and sends confirmation by e-mail.

Extensions

3. Room not available.
   a. System offers alternatives.
   b. Reservation Maker selects from alternatives.

3b. Reservation Maker rejects alternatives.
   a. Fail

4. Reservation Maker declines offer.
   a. Fail

6. Customer already on file (based on name and post code).
   a. Resume 7.

- Define initial system interface called IMakeReservation.
- Step 2: system must allow to get details of different hotels (getHotelDetails()).
- Step 3: Price and availability for a given request must be provided (getRoomInfo()).
- Step 7: operation makeReservation() needed that creates a reservation, returns a reference number, and confirms the reservation.

Results of identifying system interfaces and operations

- Parameters of the operations are defined later when considering the component interactions.
- The interfaces we have defined at system level are specific to that system and will not typically be reusable by different systems.
- Reuse of interfaces across systems is the purpose of the business interfaces, to be discussed next.

Identifying business interfaces

The business interfaces are abstractions of the information that must be managed by the system. The process for identifying them is as follows:

1. Produce a scoped copy of the business concept model as the business type model.
2. Refine the business type model and specify any additional business rules with constraints.
3. Identify **Core Business Types**.
4. Create business interfaces for core types and then add them to the business type model.
5. Refine the business type model to indicate business interface responsibilities.
6. Check that the defined interfaces align with any overriding policies, such as those defined in a corporate component architecture.

Create the business type model

- The business type model is represented by a UML class diagram, like the concept model, but its purpose is different.
- Whereas the concept model is simply a map of the information of interest in the problem domain, the business type model contains the specific business information that must be held by the system being specified.
- The business type model is initially created by copying the concept model and adding or removing elements until its scope is correct.

Note: The business type model must be a precise model, because it is the base from which the business interfaces will emerge.

Example I

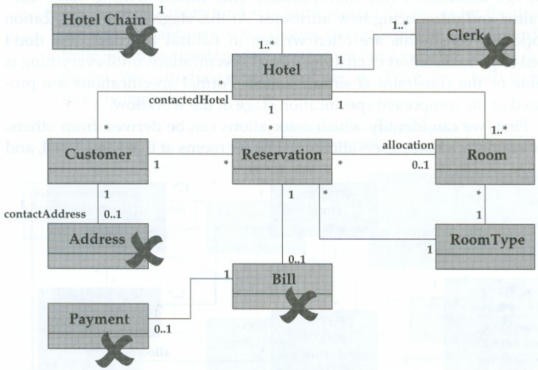


Figure 5.6 Scoping the business type model

Example II

- Eliminate the HotelChain type because the system shall support only a single chain of hotels.
- Eliminate the Hotel-Customer association (see use case definition phase).
- Eliminate Payment and Bill because they are the domain of a separate billing system.
- Eliminate Clerk and Address to keep the example simpler.

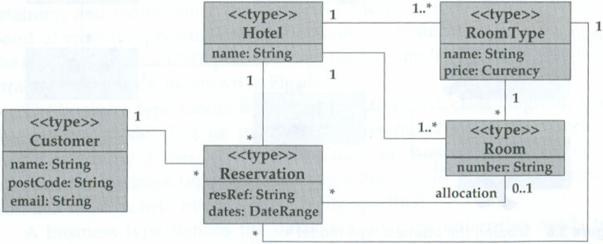


Figure 5.7 Initial business type

Define business rules I

- Add any additional required business rules to the simple ones captured directly through association role multiplicities.
- This means writing some constraints and introducing new attributes.

Example:

- Identify which associations can be derived from others:
  - A hotel reservation must be for rooms at that same hotel, and the type of room specified must be available at that same hotel.

Define business rules II

- Availability rules:
  - A room is available if the number of rooms reserved at all dates in the requested range is less than the number of rooms.
    Introduce new parameterized attribute available(DateRange) for RoomType, on which to hang this rule.
  - You can never have more reservations for a date than rooms (no overbooking).
- Pricing rules
  - The price of a room for a stay is the sum of the prices for the days in the stay.
    Change price attribute on RoomType to be parameterized by date.
    Introduce new attribute stayPrice, on which to hang this rule.

Define business rules III

Adding the extra attributes allows us to write these rules in OCL.

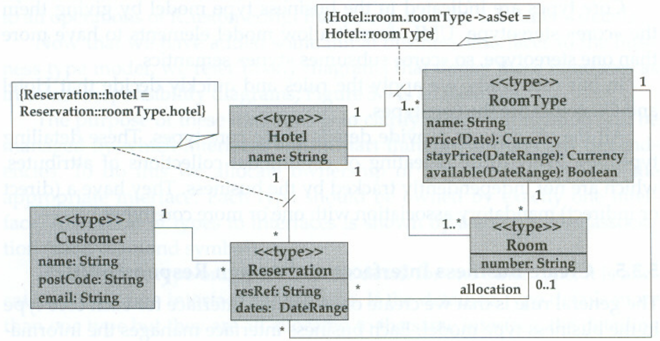


Figure 5.8 Business type model

Identify core types

- The purpose of identifying core types is to start thinking about which information is dependent on which other information, and which information can stand alone.
- A core business type is a business type that has independent existence within the business.
- Example: core types are Hotel and Customer.

Create business interfaces and assign responsibilities I

- General rule: create one business interface for each core type of the business type model.
- Each business interface manages the information represented by the core type and its detailing types.
- Naming convention: IxxxMgt

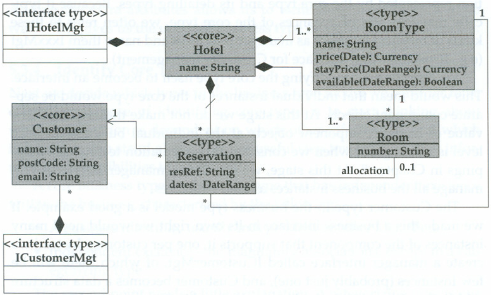


Figure 5.9 Interface responsibility diagram of the business type model

Create business interfaces and assign responsibilities II

- Each type should be owned by exactly one interface (composition relation).
- Where to allocate Reservation (provides details to both Hotel and Customer)?
- Decision: allocate Reservation to Hotel; mark association between Reservation and Customer to be navigable only toward customer.

Allocating responsibility for associations I

- When an association exists between types managed by different interfaces, this is an *inter-interface association*.
- The association between Reservation and Customer is such an association.
- A decision has to be made where this information will be recorded.
- Inter-interface associations are a specific form of dependency, which contradict the high-level goal to reduce dependencies.
- Therefore: try to avoid two-way references between interfaces.

Allocating responsibility for associations II

- Decision: Reservation references Customer, and Customer is independent of Reservation.
- Association is navigable in only one direction.

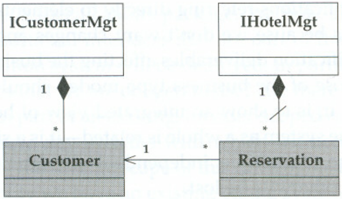


Figure 5.10 Assigning reference direction

[1] How this is achieved in the implementation is, of course, a totally separate issue.

Creating initial interface specifications I

- The system interfaces that we created earlier, which are not part of the business type model, form an initial set of interface specifications that subsequent stages will refine directly.
- The business type model and the business interfaces are internal workflow artefacts.
- Once we are happy with the interface responsibility diagram, we create another set of business interfaces in the interface specifications package, corresponding to the business interfaces we created in the business type model.
- We will further work on those interfaces in the component specification phase.

Creating initial interface specifications II

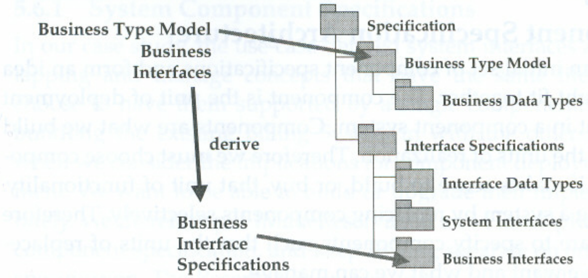


Figure 5.11 Package structure detail

Existing interfaces and systems

- Add to the interface specifications package any additional interfaces that are part of the environment into which the software will be deployed.
- In particular, are there any existing interfaces that we are obliged to use?
- Are there any systems with which we need to interface, but which are outside the specific scope of the given development project?
- Example: billing system. Its interfaces are added to the set of system interfaces.

Component Specification Architecture

- We now create an initial set of component specifications and form an idea of how they might fit together.
- We must choose components in such a way that it makes sense to build or to buy that unit of functionality.
- In most cases, we will create a separate component specification for each interface specification that we have identified.
- Multiple interfaces on one component can be considered if
  - The concepts represented by the different interfaces have the same lifetime.
  - The interactions between the interfaces are complex, frequent, or involve large amounts of data.
  - We want to keep component granularity at a reasonable size.

System component specifications I

- In our case study, the use-case-driven system interfaces are strongly overlapping and manage concepts that have the same lifetimes.
- We therefore put IMakeReservation and ITakeUpReservation on one component.
- However, IBilling is kept separate.
- The reservation system makes use of IBilling, so we add the dependency between them.
- We also add interface dependencies on ICustomerMgt and IHotelMgt, although we don't know if these really exist at this stage.
- We will validate these when we study the component interactions.

System component specifications II

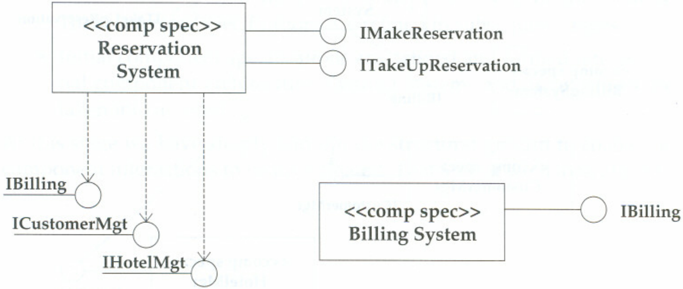
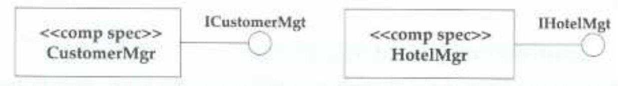


Figure 5.12 System component specifications

Business component specifications

- For the business interfaces, our starting point is one component per interface.
- Since the manager interfaces were created to manage instances of core business types and their associations, they are concerned with information that is managed independently.
- Result:



An initial architecture I

- Now we have an initial set of component specifications, including their supported interfaces and their interface dependencies.
- Since we don't have any interfaces being offered by more than one component specification in our example, we can bind the interface dependencies of the component specifications directly onto their corresponding component specification interfaces.

An initial architecture II

Result:

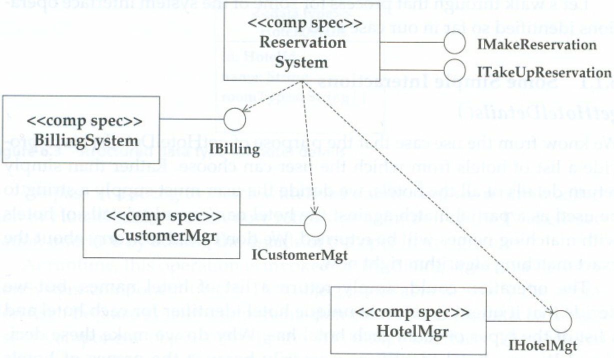


Figure 6.2 Initial component architecture

Summary of component identification I

Main principles:

- The system interfaces correspond to use cases, and their operations are derived from use case steps.
- A business type model is developed representing the system's eye view of the business concept model. Business rules are captured on the business type model as constraints. The business type model is an internal workflow artifact, which is useful to maintain.
- Business interfaces are discovered by identifying core types in the business type model and creating interfaces to manage them and their details.

Summary of component identification II

- Initial business interface specifications are created by copying the business type model interfaces. These interfaces are refined in subsequent stages.
- Initial component specifications are defined and formed into an initial component architecture. Existing systems and architectures are taken into account.