Luca Cabibbo

Architettura dei Sistemi Software

Una metodologia per la specifica di software a componenti

dispensa asw475

marzo 2021

How best to read this book.
Start at page 1 and keep going.
When you reach a page that's thicker than the rest and shiny on one side, you're done.

John Cheesman and John Daniels

- Riferimenti

- Cheesman, J. and Daniels, J. **UML Components**: A Simple Process for Specifying Component-Based Software. Addison-Wesley, 2000.
- Cheesman, J. and Daniels, J. **UML Components**: Un semplice processo per la specifica di software basato su componenti. Addison-Wesley, 2002.
  - http://www.umlcomponents.com/
- Daniels, J. UML Components (tutorial)
  - http://www.omg.org/news/meetings/workshops/presentations/uml2001_presentations/02-2_Daniels_Tutorial-UML_Components.pdf
- Eeles, P. and Cripps, P. **The Process of Software Architecting**. Addison-Wesley, 2010.
  - http://www.processofsoftwarearchitecting.com/

- Obiettivi e argomenti

- Obiettivi
  - presentare una semplice metodologia per la specifica di software basato su componenti
- Argomenti
  - introduzione
  - una metodologia per la specifica di software a componenti
  - discussione

* Introduzione

- Un *componente*
  - è un'entità software runtime
  - implementa un insieme di funzionalità
  - offre i suoi servizi mediante un insieme di interfacce con nome (interfacce fornite)
  - può richiedere servizi ad altri componenti, sempre sulla base di interfacce con nome (interfacce richieste)
- Un *sistema software basato su componenti*
  - è un'applicazione formata da un insieme di componenti, che vengono composti (al momento del rilascio) sulla base delle loro interfacce

Introduzione

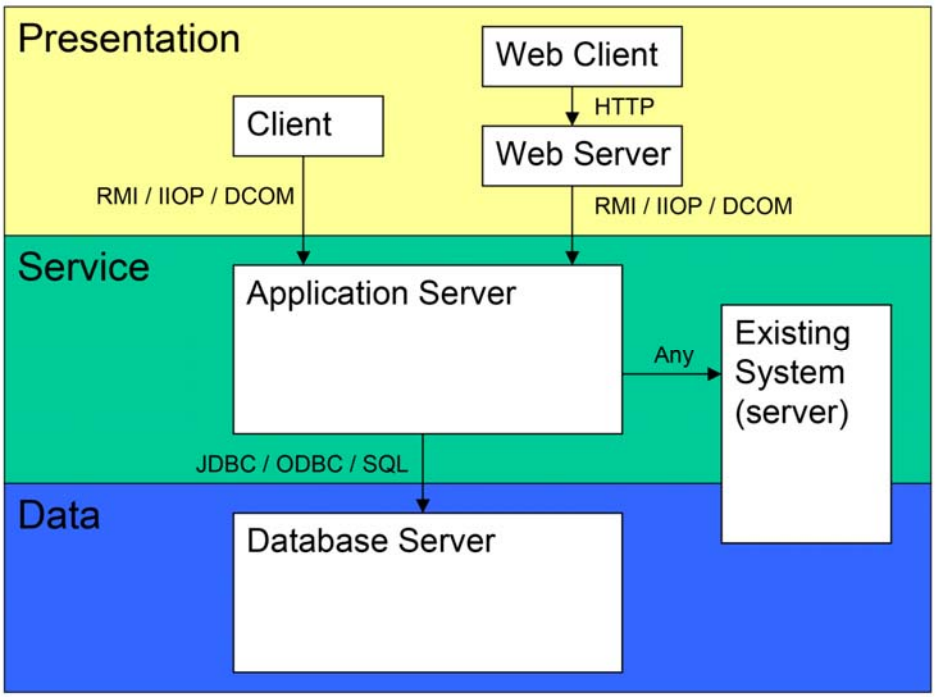
- La progettazione di un sistema software basato su componenti
  - in sede di *specifica*, è necessario scegliere
    - quanti e quali componenti (tipi di componenti) servono
    - quali le loro interfacce fornite e richieste – ciascuna interfaccia con la specifica di un insieme di operazioni
    - quali le interazioni tra i componenti scelti – in termini di collegamenti/dipendenze tra componenti, sia di natura statiche che dinamica, con riferimento alle loro interfacce fornite/richieste
  - in sede di *deployment*, è anche necessario scegliere, per ciascuno dei componenti
    - quanti e quali componenti oggetto servono
    - la particolare implementazione prescelta per ciascun componente

* Una metodologia per la specifica di software a componenti

- La specifica di un sistema software basato su componenti è un'attività complessa
  - viene ora presentato un "semplice processo" introdotto in UML Components per la progettazione di applicazioni di tipo enterprise
  - un processo più completo è invece descritto da [Eeles and Cripps] – ma è fuori dalla portata di questa dispensa
- Le intuizioni alla base della metodologia di UML Components
  - applicazione di Layers – con un'architettura a quattro strati, simile a quella di DDD
  - applicazione di Domain Model – usando come modelli di dominio il modello dei casi d'uso e il modello delle informazioni
  - applicazione di tecniche di modellazione del software di tipo statico e di tipo dinamico

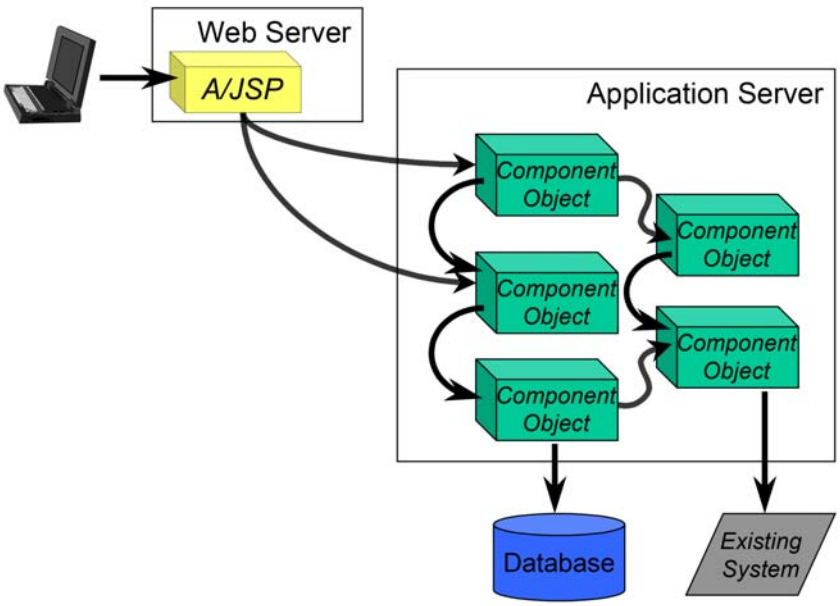
- Architettura a strati

- Architettura a strati dell'applicazione



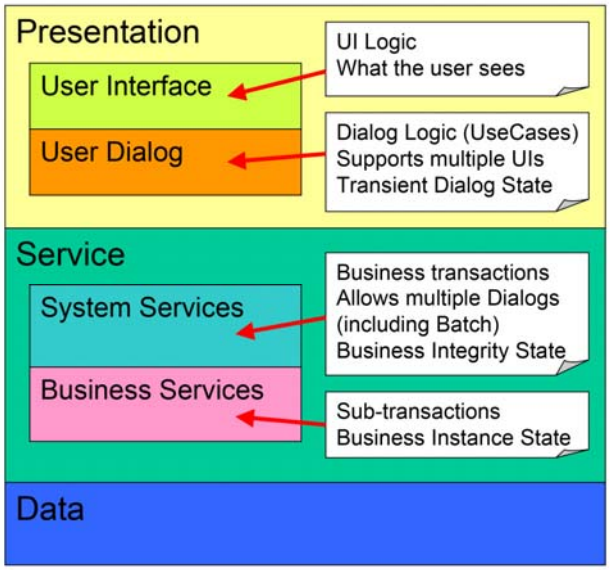
Architettura a strati

- Interazione tra componenti



Architettura a strati

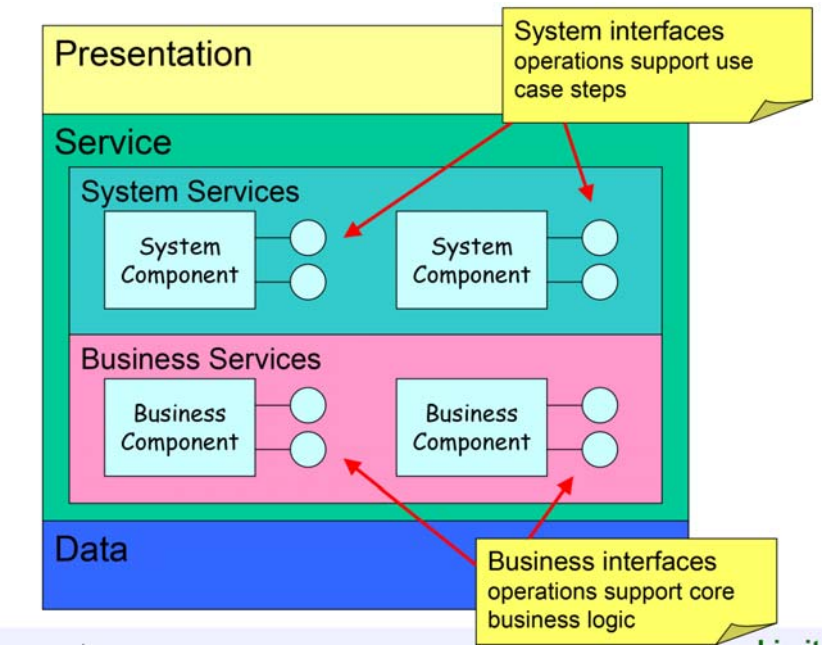
- Architettura a strati – maggior dettaglio



  - il metodo si concentra sulla specifica dei componenti nello strato dei servizi

- Tipologie di componenti

- Nello strato dei servizi, due sotto-strati – e, in corrispondenza, due tipi di componenti

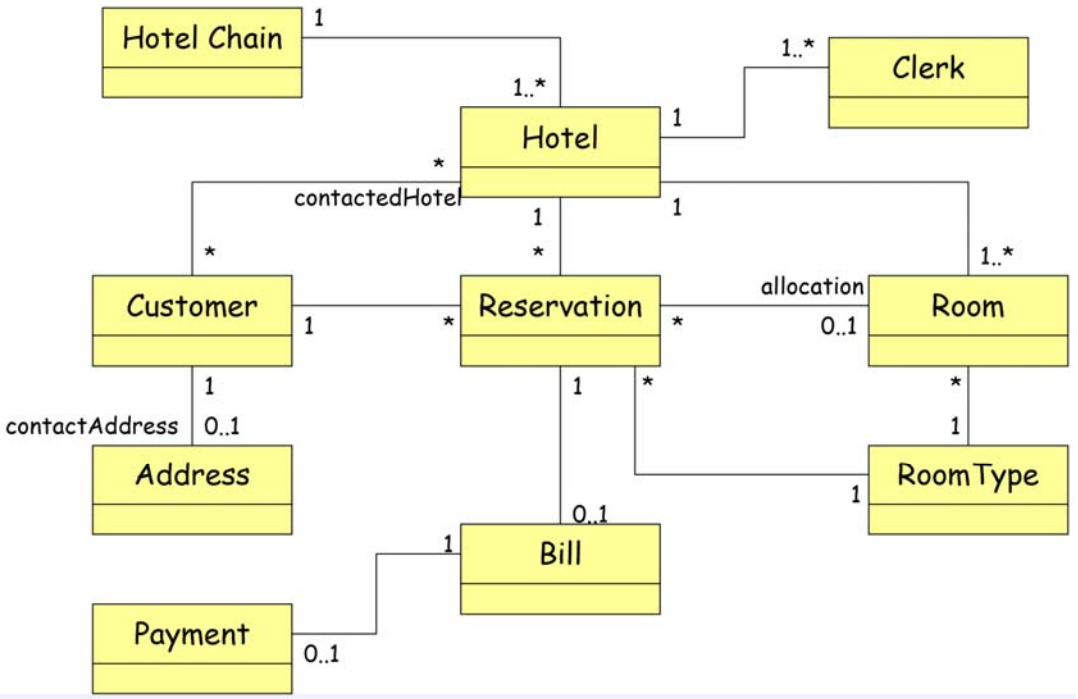


Tipologie di componenti

- Due tipi di componenti nello strato dei servizi
  - *componenti di sistema*
    - rappresentano casi d'uso
    - le loro interfacce – interfacce di sistema – definiscono e sostengono i passi dei casi d'uso ("operazioni di sistema")
  - *componenti di business*
    - implementano le funzionalità principali di business/ di dominio
    - gestiscono le informazioni di business
- Questa scelta è un'applicazione di Domain Model
  - motiva il tipo di modellazione di dominio utile

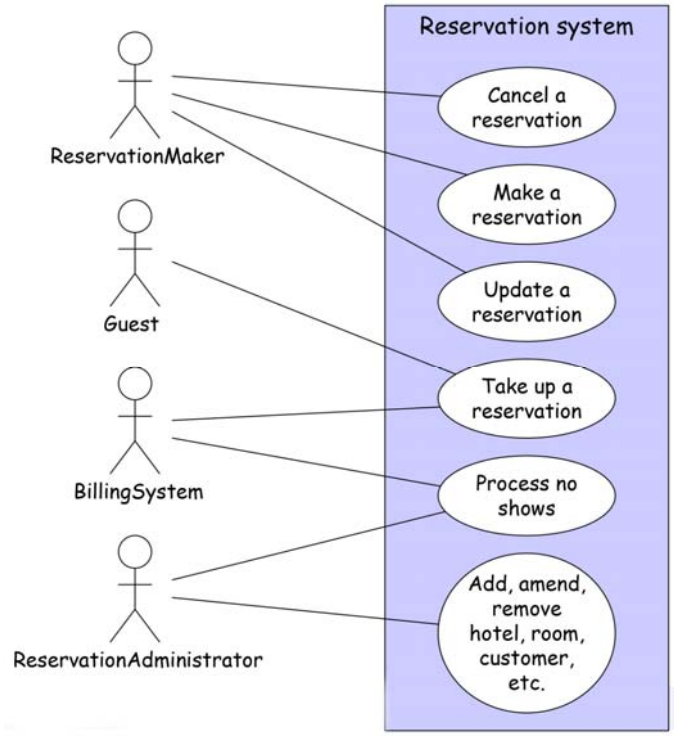
- Modellazione di dominio

- *Business concept model* – un modello concettuale delle informazioni che caratterizzano il dominio



Modellazione di dominio

- *Modello dei casi d'uso* – un modello delle funzionalità del sistema



Modellazione di dominio

- *Modello dei casi d'uso* – un modello delle funzionalità del sistema

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

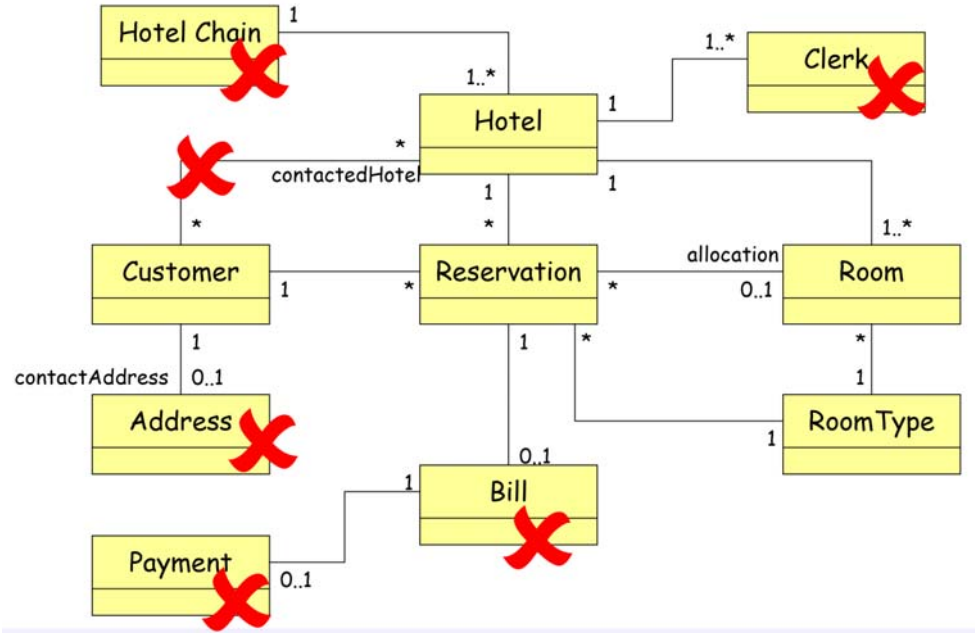
Extensions
3. Room Not Available
   a) System offers alternative dates and room types
   b) Reservation Maker selects from alternatives

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

- Scelta dei componenti di business

- *Business type model* – un modello delle informazioni che devono essere gestite *direttamente* dal sistema

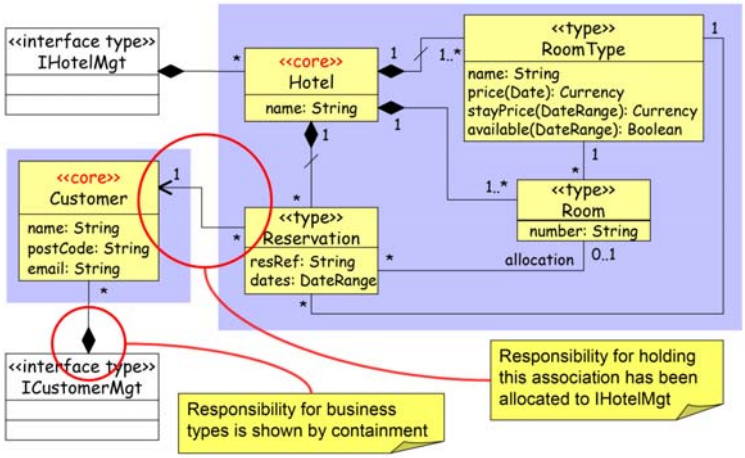


Scelta dei componenti di business

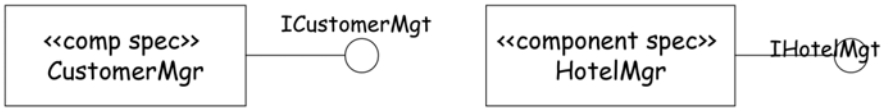
- Ciascun componente di business serve a gestire un tipo di dato "principale" nel sistema
  - un tipo di dato principale ("core type") è di solito caratterizzato da un'esistenza autonoma e da un'identificazione autonoma
  - questi tipi possono essere identificati dal modello dei tipi di business
  - i componenti di business sono di solito stateless
- In corrispondenza a questi core type, definisci
  - un'*interfaccia di business* di "gestione"
  - un *componente di business* "gestore"

Scelta dei componenti di business

- Scelta delle interfacce di business
  - le operazioni di queste interfacce verranno definite in seguito

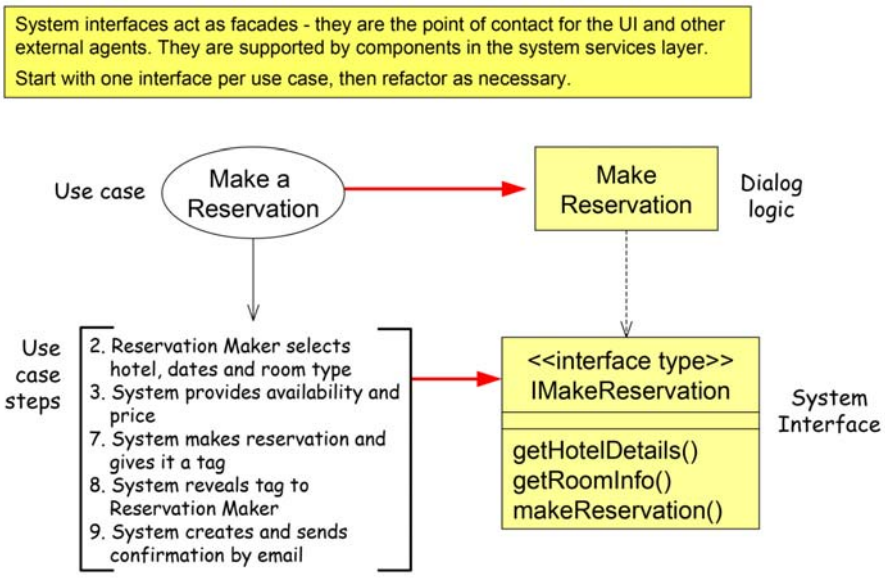


- Scelta dei componenti di business



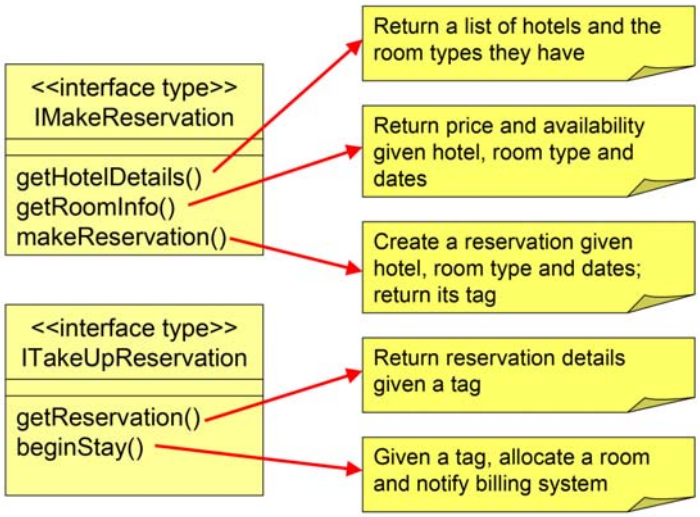
- Scelta delle interfacce di sistema

- Definisci un'*interfaccia di sistema* per ciascuno dei casi d'uso identificati
  - l'interfaccia di sistema per un caso d'uso dichiara le operazioni di sistema corrispondenti ai passi di quel caso d'uso



Scelta delle interfacce di sistema

- Definisci un'*interfaccia di sistema* per ciascuno dei casi d'uso identificati
  - l'interfaccia di sistema per un caso d'uso dichiara le operazioni di sistema corrispondenti ai passi di quel caso d'uso

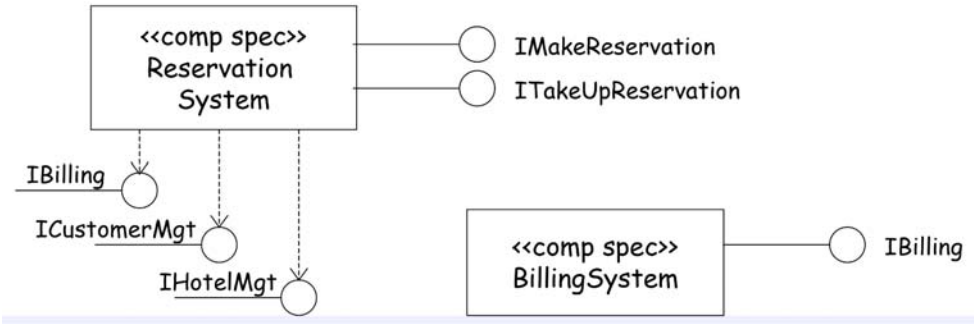


- Scelta dei componenti di sistema

- Per i *componenti di sistema* sono possibili diverse alternative
  - un unico componente di sistema per tutti i casi d'uso – che implementa (fornisce) tutte le interfacce di sistema
  - un componente di sistema per ciascun caso d'uso – ciascuno implementa (fornisce) l'interfaccia di sistema corrispondente
  - un componente di sistema per ciascun gruppo coeso di casi d'uso – ad es., per tutti i casi d'uso per un certo attore
  - questi componenti sono di solito stateful
  - questi componenti richiedono, in prima approssimazione, le interfacce fornite dai componenti di business
- Ulteriori *componenti di sistema*
  - è utile definire un componente di sistema per ciascuna sottosistema pre-esistente – con un'interfaccia di sistema corrispondente a quella del sottosistema pre-esistente

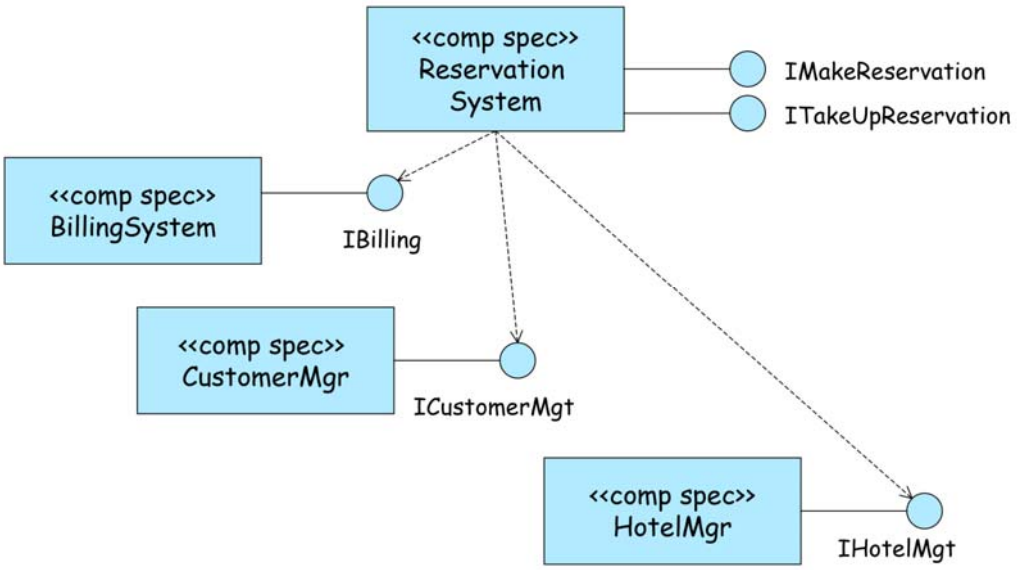
Scelta dei componenti di sistema

- Scelta dei componenti di sistema



- Architettura a componenti iniziale

- Architettura a componenti iniziale



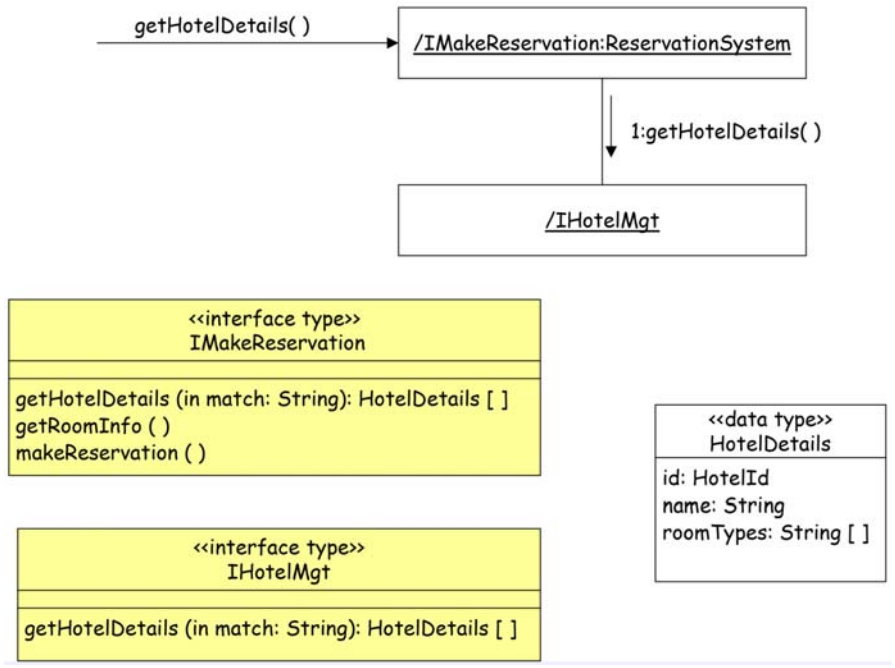
A che punto siamo?

- A che punto siamo con la specifica del nostro sistema software a componenti?
  - abbiamo identificati i nomi delle interfacce di sistema e di business
  - abbiamo identificato i componenti di sistema e di business necessari – con le loro interfacce fornite e richieste (almeno, in prima approssimazione)
  - abbiamo identificato le operazioni delle interfacce di sistema – ma non le operazioni delle interfacce di business
- Dunque, è ancora necessario
  - identificare le operazioni delle interfacce di business
  - raffinare le relazioni (e le interazioni) tra componenti di sistema e componenti di business
  - si può applicare una tecnica di progettazione dinamica

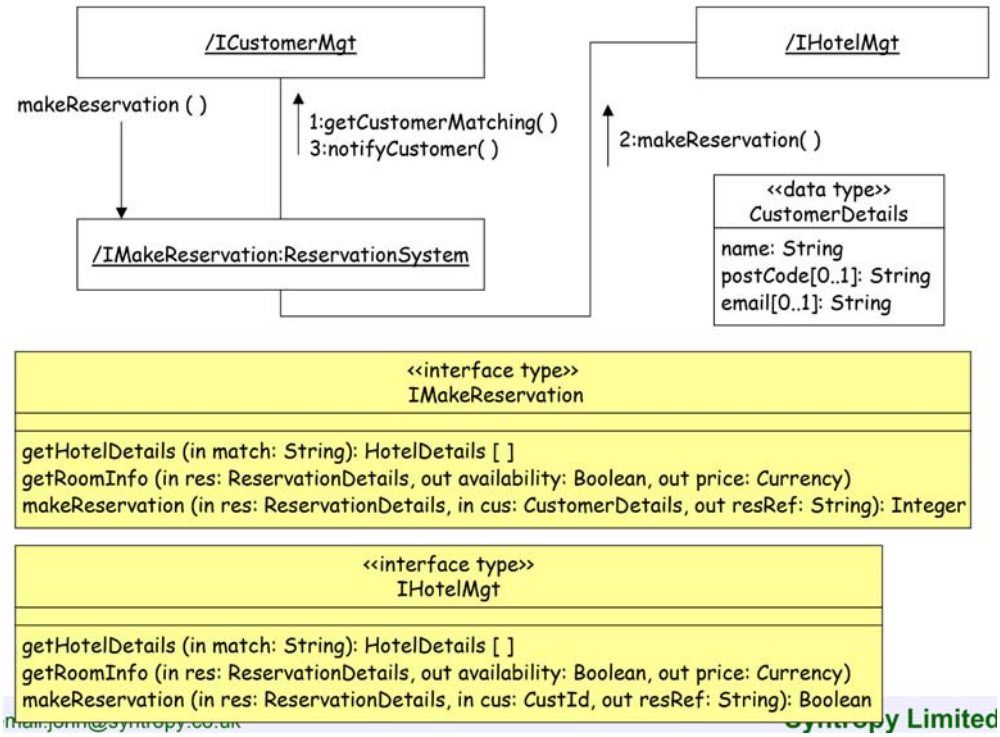
- Raffinamento del progetto

- Per identificare le operazioni delle interfacce di business e raffinare le relazioni (e le interazioni) tra componenti di sistema e componenti di business si può applicare una tecnica di progettazione dinamica
  - per ciascun caso d'uso e passo (operazione di sistema) di caso d'uso, ci si chiede "come fa il sistema (incarnato nei suoi componenti) a sostenere questa operazione di sistema?"
  - in corrispondenza, si identificano
    - le interazioni tra componenti di sistema e componenti di business
    - le operazioni delle interfacce di business

Progettazione dinamica

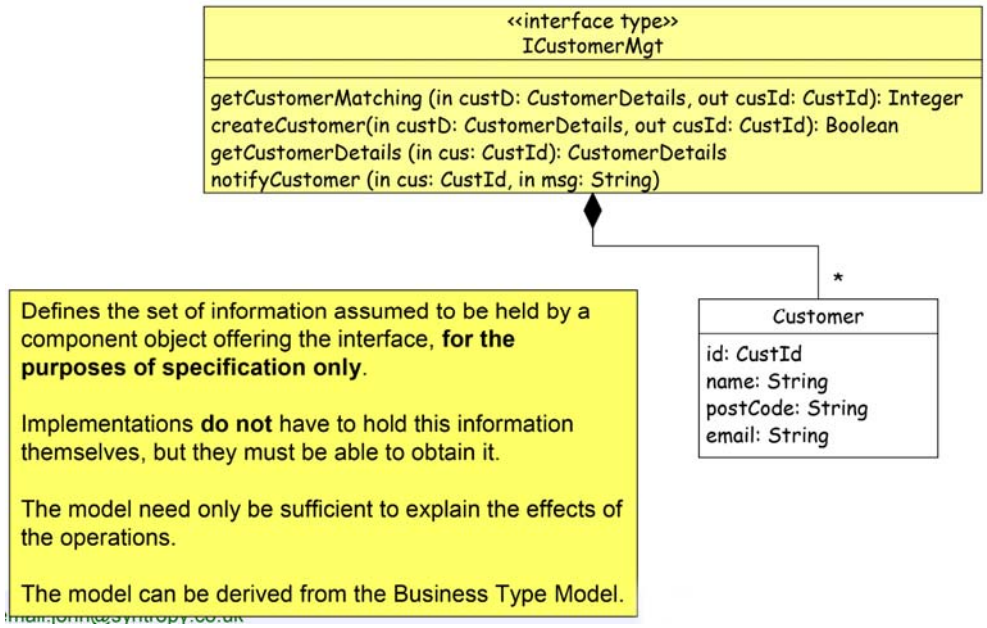


Progettazione dinamica



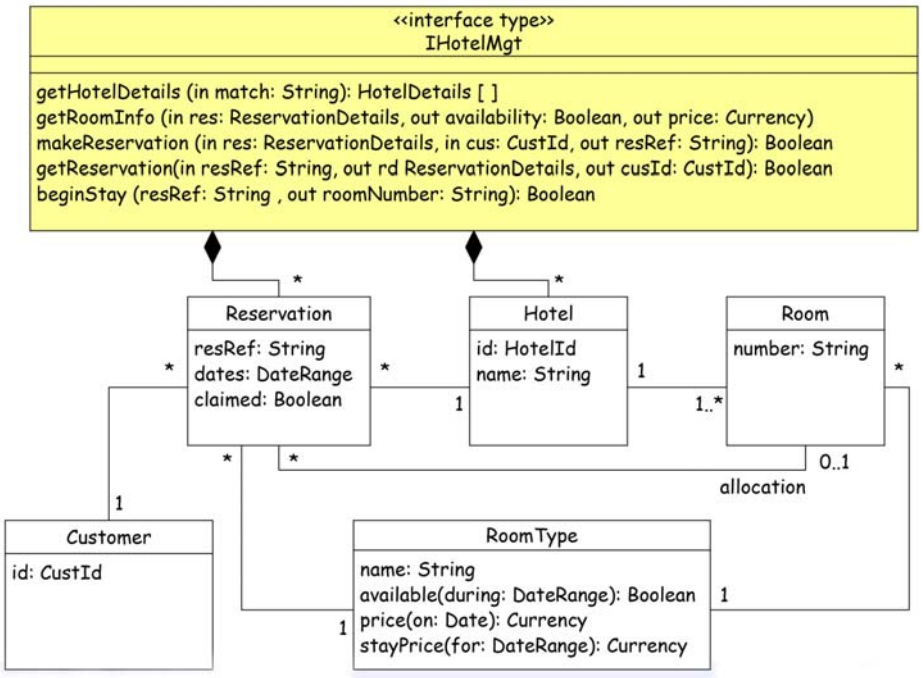
Raffinamento del progetto

- Ai fini della specifica, bisogna anche descrivere, per ciascuna interfaccia, quali sono i tipi di dati associati all'interfaccia



Raffinamento del progetto

- Ai fini della specifica, bisogna anche descrivere, per ciascuna interfaccia, quali sono i tipi di dati associati all'interfaccia



Raffinamento del progetto

- La specifica può anche essere dettagliata con i contratti delle operazioni delle interfacce – contratti definiti in termini di pre- e post-condizioni, riferite ai tipi associati all'interfaccia

```
context IHotelMgt::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
```


* Discussione

- Questa dispensa ha illustrato brevemente una metodologia per la specifica di un sistema software basato su componenti – per applicazioni di tipo enterprise
  - questa metodologia si basa sull'applicazione dei seguenti stili architetturali e tecniche di progettazione del software
    - Layers – con un'architettura a quattro strati, simile a quella di DDD
    - Domain Model – usando come modelli di dominio il modello dei casi d'uso e il modello delle informazioni
    - tecniche di progettazione del software basata sulla modellazione statica (dei componenti) e la modellazione dinamica (delle interazioni)