Component Specification with UML

A simple form of specification of component-based software

M.I. Capel

ETS Ingenierías Informática y
Telecomunicación
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada
Email: manuelcapel@ugr.es
http://lsi.ugr.es/mcapel/

DSBCS

Máster Universitario en Ingeniería Informática

October, 16th 2017

1. Component Specification
2. Creation process of the Interface Information Model (IIM)
3. Creation process of the System IIM
4. Component factorization techniques

Component Based Systems (CBS) Specification

Contracts in CBS

- Contract of use
- Contract of realization

Specification of an interface

To determine which parts the interface is composed of and to describe these parts without ambiguity

Specification of a component

The interfaces that the component implements are grouped and the constraints that affect the component itself must be written

Provided interfaces

An interface of this type fulfills the following conditions:

- It is implemented by the component itself or
- It is implemented by one of the component's objects or
- It is yield by one component's port

Figure: Component `Weather services` implements the interface `Weather Forecast`

Required Interfaces

An interface of this type fulfills the following conditions:

- There is a dependency of use of the component itself
- There is a dependency of use of the component's objects
- There is needed by a component's *public port*

Figure: Component `User Services` requires the interface `ServiciosIOrder`

Specification process of interfaces

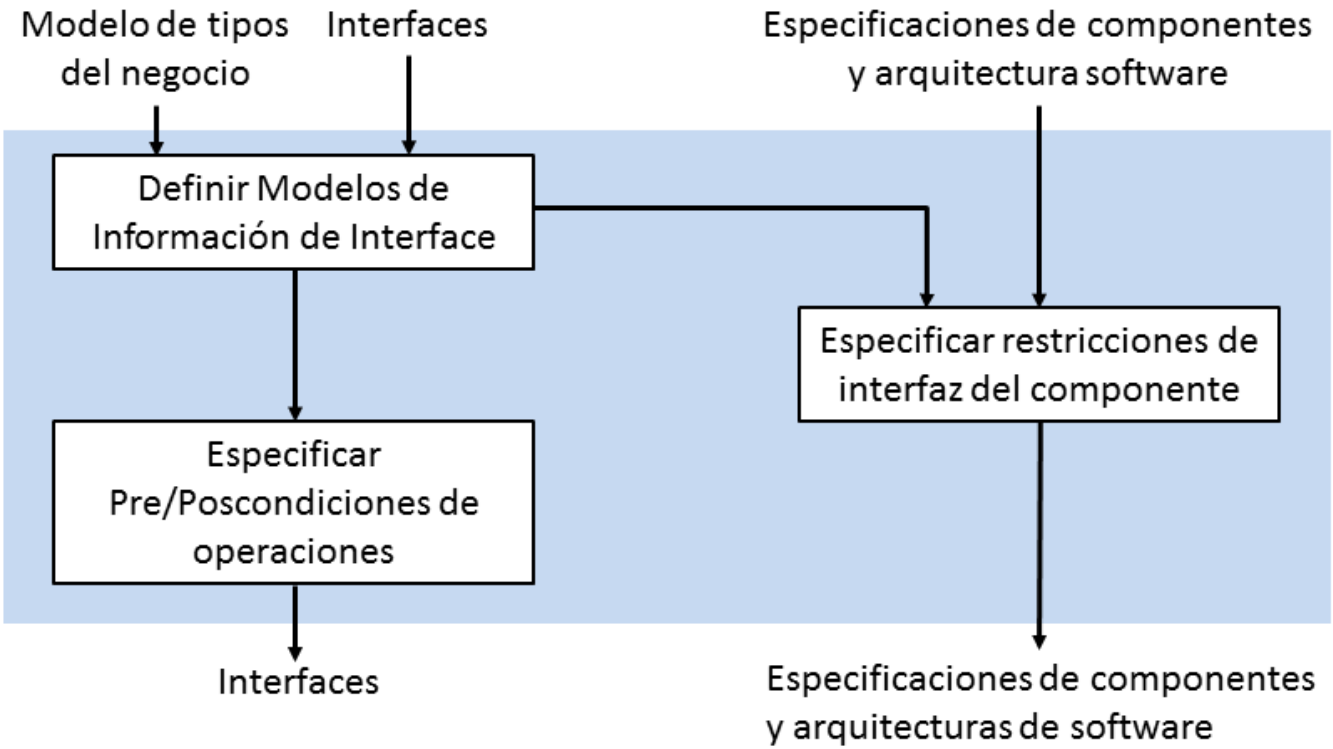

Figure: Specification of a component within the complete workflow at this stage

Interface specification vs. operation specification

Component operations specification:

- They lack of any structural information of the component
- Operations definition does not provide an appropiate description level of a component's dependencies

Component interfaces specification

- Grouping of related operations
- This type of operation grouping will be revisited in the *factorization* activity afterwards
- Interface subtyping introduction
- Can include only 1 *self-contained* and independent component

Operations specification

What does an operation specification need to include? and what does not?

- Relationship between operation inputs, outputs and component object state description
- To define the effect that one call will have on the input/output relation in an operation

What does it have to guarantee?

- Transparency of the relations between the component object and other objects

Operations specification II

Elements of the specification:

- Input and output parameters
- Constraints of application to the component object
- Any change of state that results in the component object

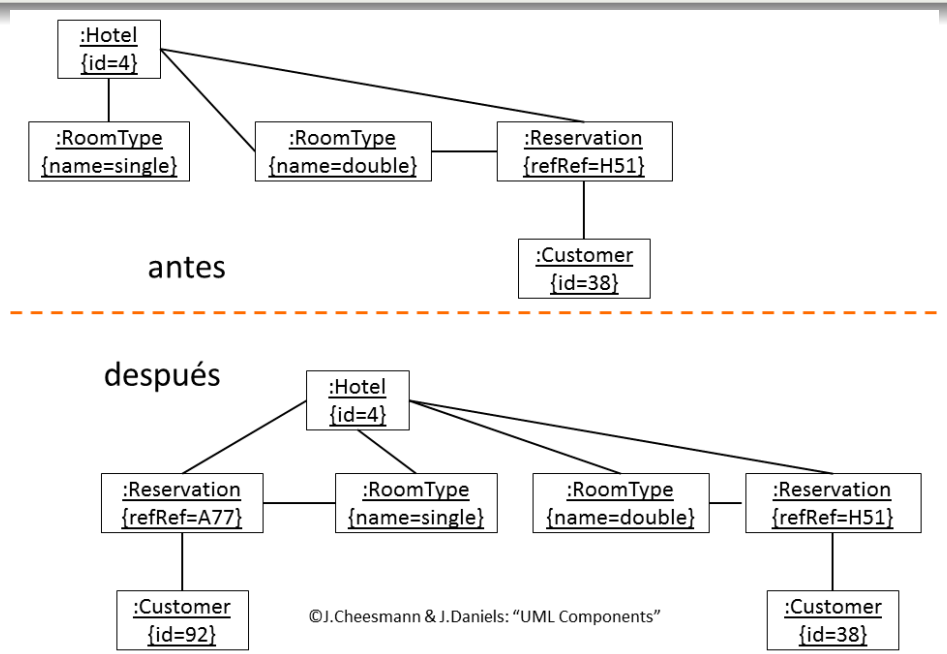

Figure: `IHotelMgt::makeReservation()` operation effect in part of the component object

Interface specification I

Model characteristics:

- The Interface Information MOdel (IIM) contains enough information to allow carrying out the operations specification as part of the interface
- As well as the effect and constraints that an operation execution has on the component's state
- Description of the state changes as result of operation execution
- By adding types, attributes, ..., an interface specification is incrementally built as the specification activity advances

Interface specification II

Conditions that the specification model must abide:

- Interfaces are only associated to *typed information*
- Only will include information on the set of states that a component owns
- Will never give information on the component internal-state implementation
- Neither do interfaces give information about persistence of the component

Interface Specification Model

Component object state representation of which the interfaces depends on

- An Interface Information Model (IIM) is needed

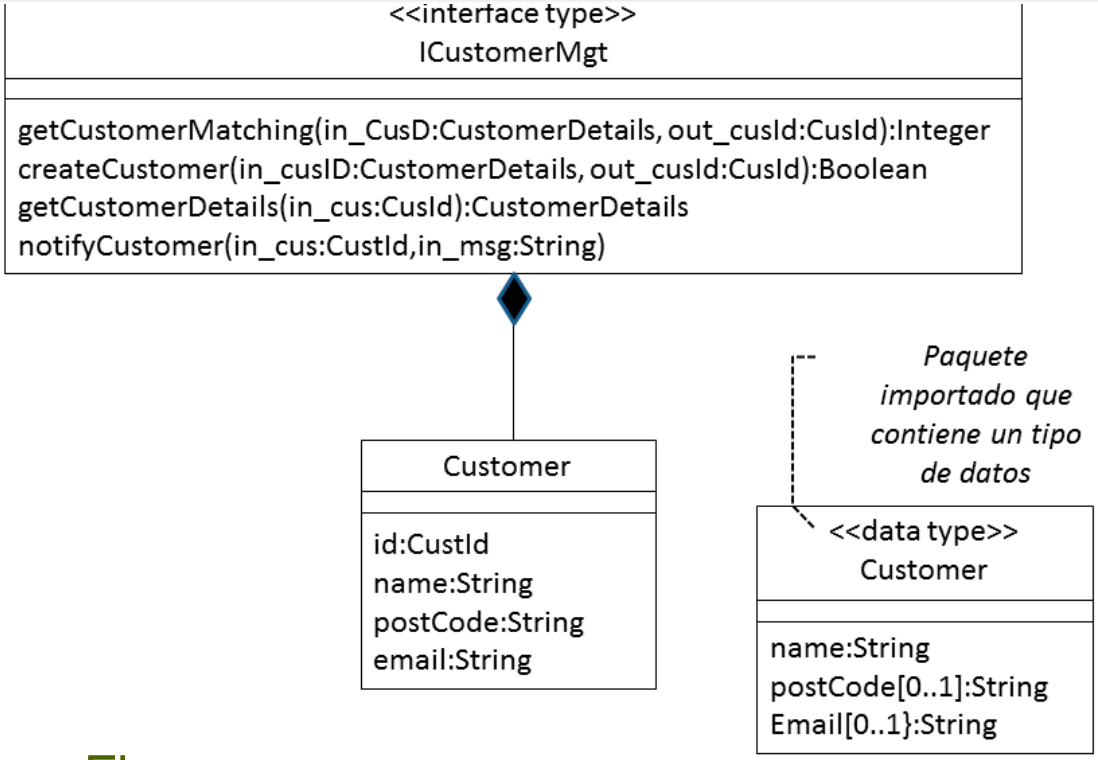

Figure: 1. Interface specification model for `ICustomerMgt`

Interface Information Model (IIM)

`ICustomerMgt` exemple discussion

- `Customer` is a *typed information*
- The interface types cannot maintain associations with any entity outside the model
  - Location in the same package that the interface
  - Exception made for inherited *subtypes* among interfaces; these types are not shared but can be imported

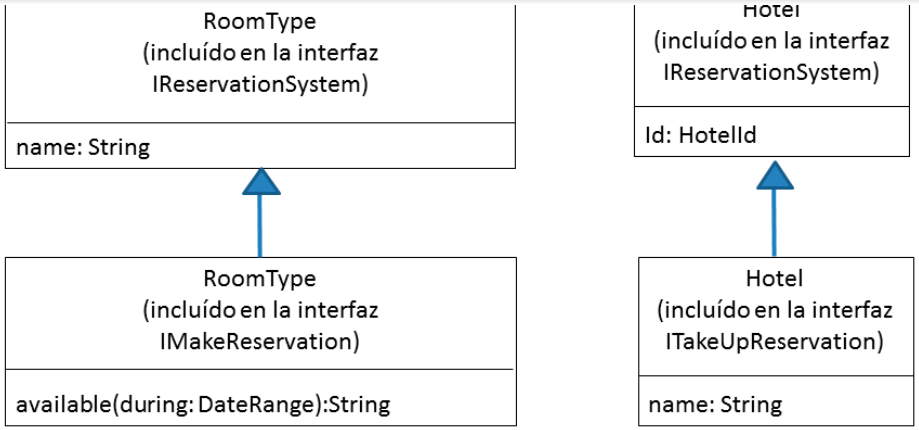

Figure: Type inheritance example after performing a factorization of interfaces

Pre and postconditions

- Each operation has got an associated *pre* and *poscondition*, which details what the operation will do
- Pre and postconditions do not provide any algorithmic or implementation-related information
- Act as the *fine print* of a contract with the client
- Precondition: it guarantees that the execution of the operation will make true the postcondition
- The mentioned operation call is completely independent of the certainty value of its precondition
- Any assumption about the operation execution is responsibility of the client
- Postdition: contractual guarantee that is responsability of the operation provider (e.g. the component *developer*)

OCL

Definition

Object Constraint Language is a declarative programming notation that allows buliding logical written expressions. OCL is of use for specifying contractual conditions (for instance) when it comes to develop the interface specification of software components

OCL expressions of sematically correct pre and postconditions

- Can refer to parameters, operation results and to the component's object state
- Cannot refer to other interfaces elements
- The interface specification only has a local effect (they only apply to the information model handled)

Expressions built with OCL

OCL specification of the name change operation

```
1  context ICustomerMgt::changeCustomerName(in cus:CustId,
2                in newName:String)
3
4  pre:
5     --cus  is a client valid identifier
6     customer->exists(c | c.id = cus)
7  post:
8     --the client name whose identifier is 'cus' is changed to '
      newName'
9     customer->exists(c | c.id = cus and c.name = newName)
```

'customer' means the set of clients associated to the supporting component-object: `ICustomerMgt`

OCL Expressions (II)

Operation for obtaining a client details specification

```
1  context ICustomerMgt::getCustomerDetails(in cus: CustId):
       CustomerDetails
2  pre: --cus is a client valid identifier
3     customer->exists(c | c.id = cus)
4  post: --the returned details after execution are identically
       equal to the client with identifier 'cus'
5        --to find the client
6     Let elCliente = customer->select(c | c.id = cus) in
7        --tro specify the result
8        result.nombre= elCliente.nombre and
9        result.codigoPostal = elCliente.codigoPostal and
10       result.email = elCliente.email
11    --the returned value is implicitly yield with the assignment
         of the variable 'result'; there is no state change
```

The operation does not change the executing object state, since a postcondition only specifies the returning result.

OCL (II)

Conditions in posconditions

- Expressions of a poscondition can refer either the state prior the operation execution (`@pre` of OCL) or to its subsequent state
- Allow writing expressions that specify how the IIM atributes and associations change as result of an operation execution

Interface Information Model creation example

General aspects of the IIM 'hotel reservations'

- The interface `IHotelMgt` addresses hotel booking management by carrying out assignment of rooms to client
- `ICustomerMgt` addresses client management
- `IHotelMgt` is responsible of the following types definition:
  - `Hotel`
  - `RoomType`
  - `Room`
  - `Reservation`
- `ICustomerMgt` is responsible of the following types definition:
  - `Customer`

Interface Information Model creation example - II

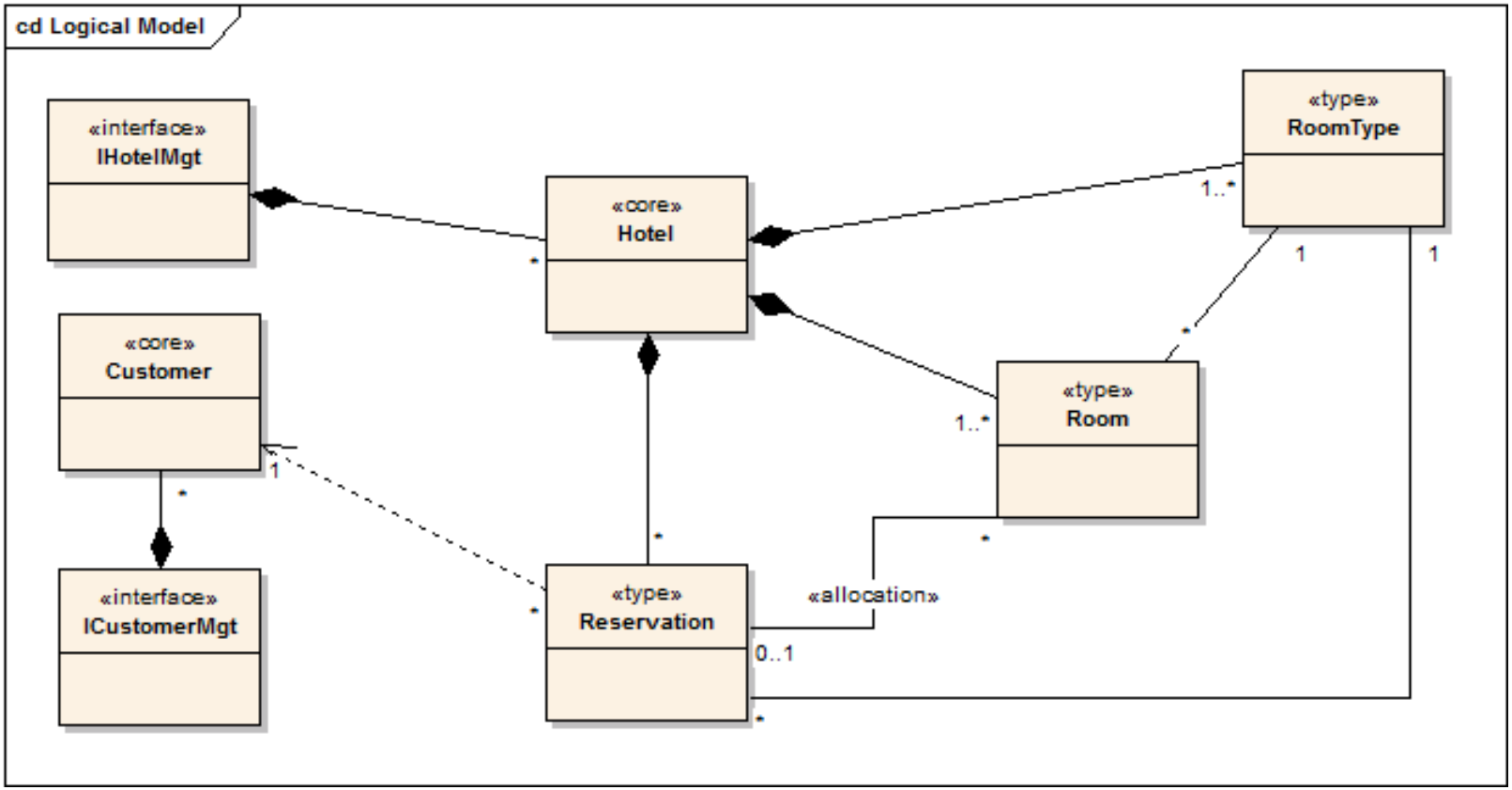

Figure: Responsability Diagram of the Interface (RDI) `IHotelMgt`

Interface Information Model creation example - III

DRI adaptations necessary to obtain the IIM

- Types `Hotel`, `RoomType` and `Reservation` must be included in the interface information model of `IHotelMgt`
- The association between reservations and customers does not need to be included in the IIM
- Associations in the RDI can be transformed:
  - Direct association inclusion: `IHotelMgt -> Reservation`
  - From *derived* association into *direct* association: `Hotel->Reservation`
- Some DRI associations can be deleted:
  - Derived association: `Hotel -> RoomType`
- Attribute inclusion to the DRI: attribute `claimed` into `Reservation`

Interface Information Model creation example IV

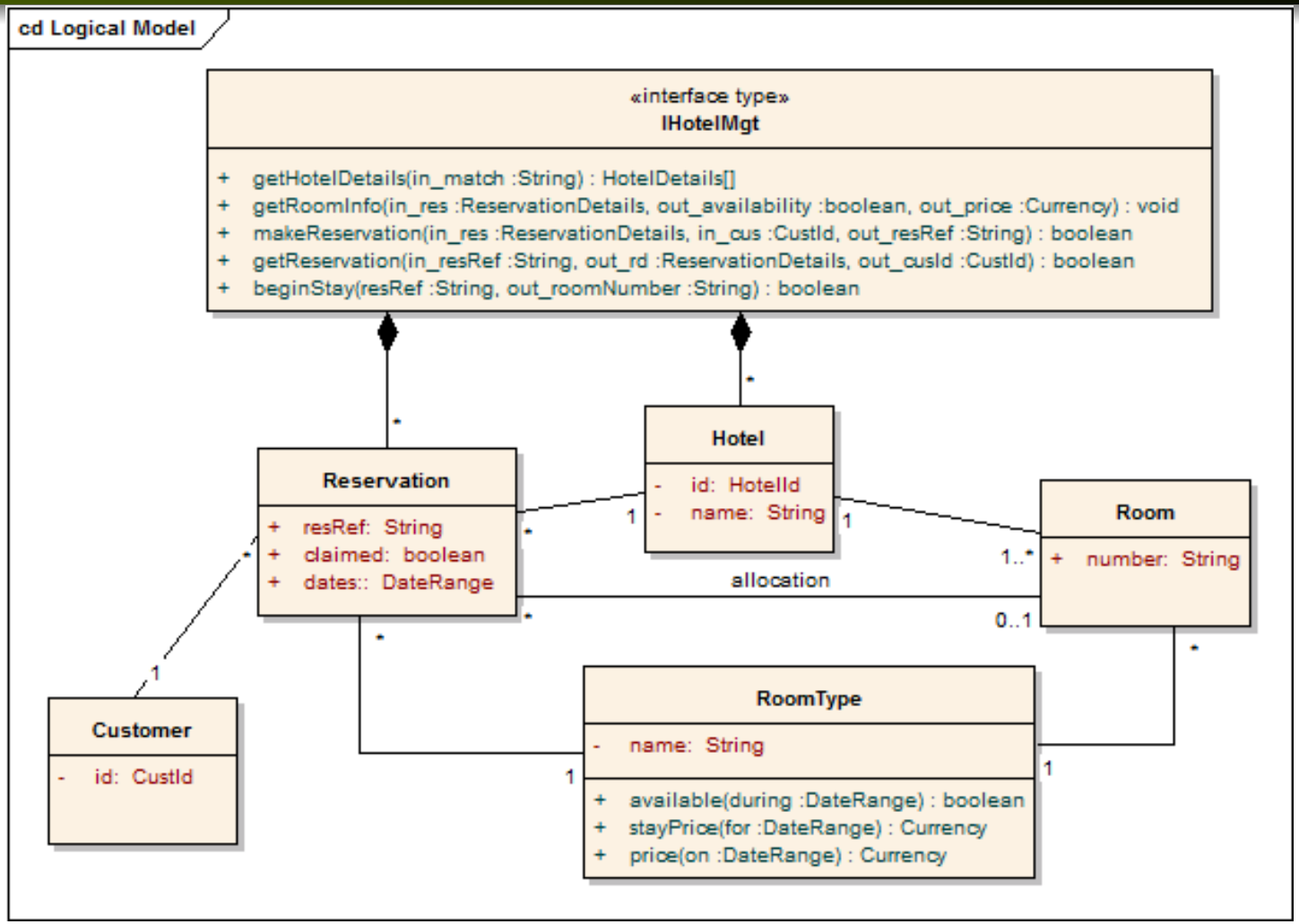

Figure: Interface Specification Diagram of `IHotelMgt`

Interface Information Model creation example V

Invariants

- *Invariant*: constraint associated to a type that can be maintained true in all instances of the type
- Invariants can be graphically expressed resorting to UML
- Invariants can be written as OCL expressions:

```
1 context r: Reservation inv:
2   --a reservation is claimed if there is already an
        assigned room
3   r.claimed = r.allocation->noEmpty
```

- From the definition above we can use “claimed” as an abbreviated form of a relation
- An invariant is capable of connecting different parts of the information included in a specification

OCL Operations Specification

Assignment

By using the operators: `exists`, `select` y `asSequence->first`, to specify the operation `IHotelMgt::makeReservation (...)` entirely with the OCL notation.

System Interfaces Specification

System Interfaces

- The so called System's Information Interface Model (SIIM) is a subset of the *business type model*
- This interface model tries to group any function automatically, i.e. without human assistance, performed by the system
- Differences between SIIM and IIM of a business model:
  - SIIM does not have to contain all the business model types
  - *Responsibility Specification Diagrams* preparation does not yield as much information on the business model types as in the case of IIM elaboration
  - Business model types that must be included in the IIM cannot be clearly identified until programming the operations

System Information Interfaces Models

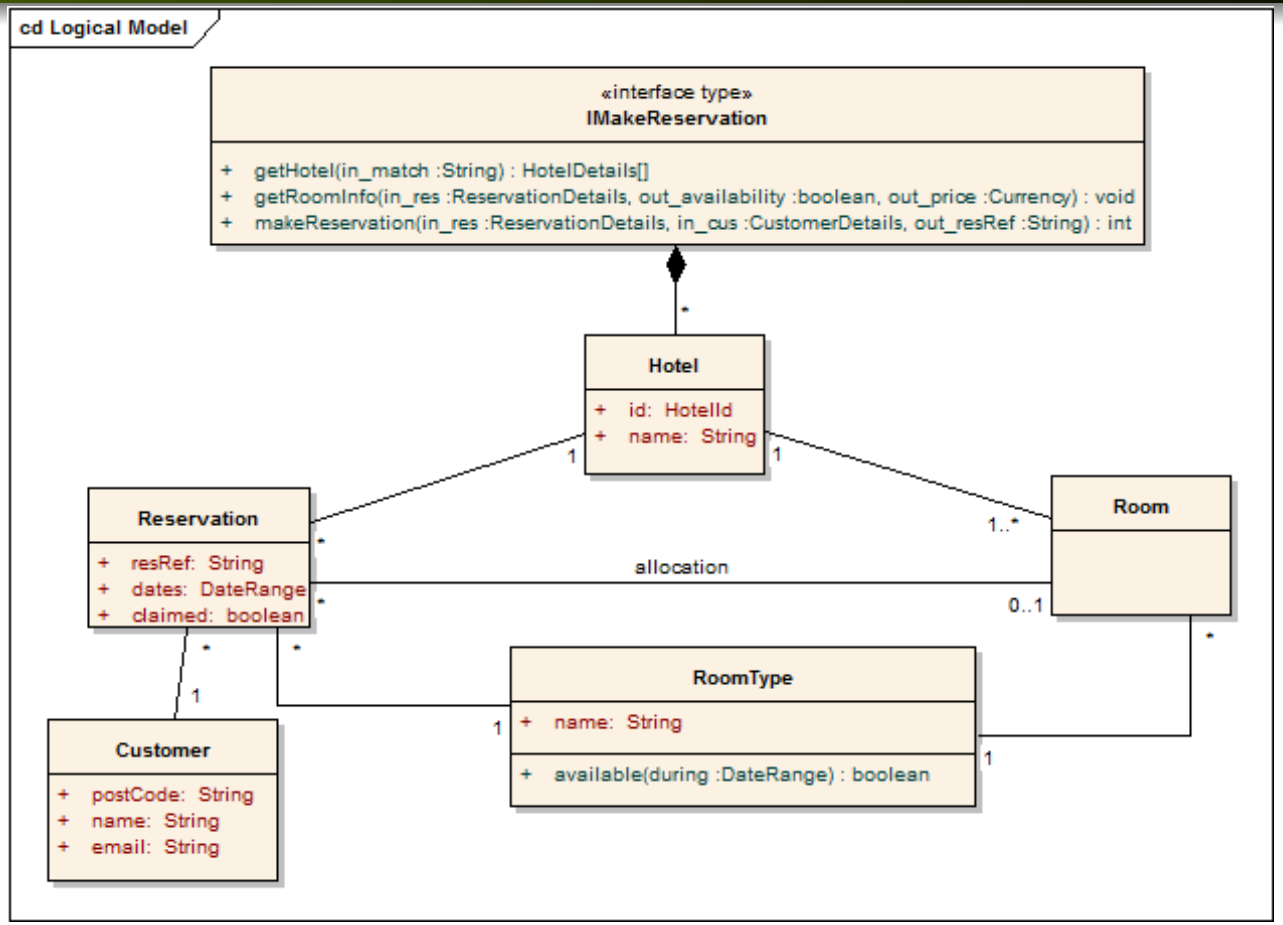

Figure: System Interface Specification Diagram `IMakeReservation`

This model does not need the business-IIM's `Room` class attribute `number`

System Information Interfaces Models II

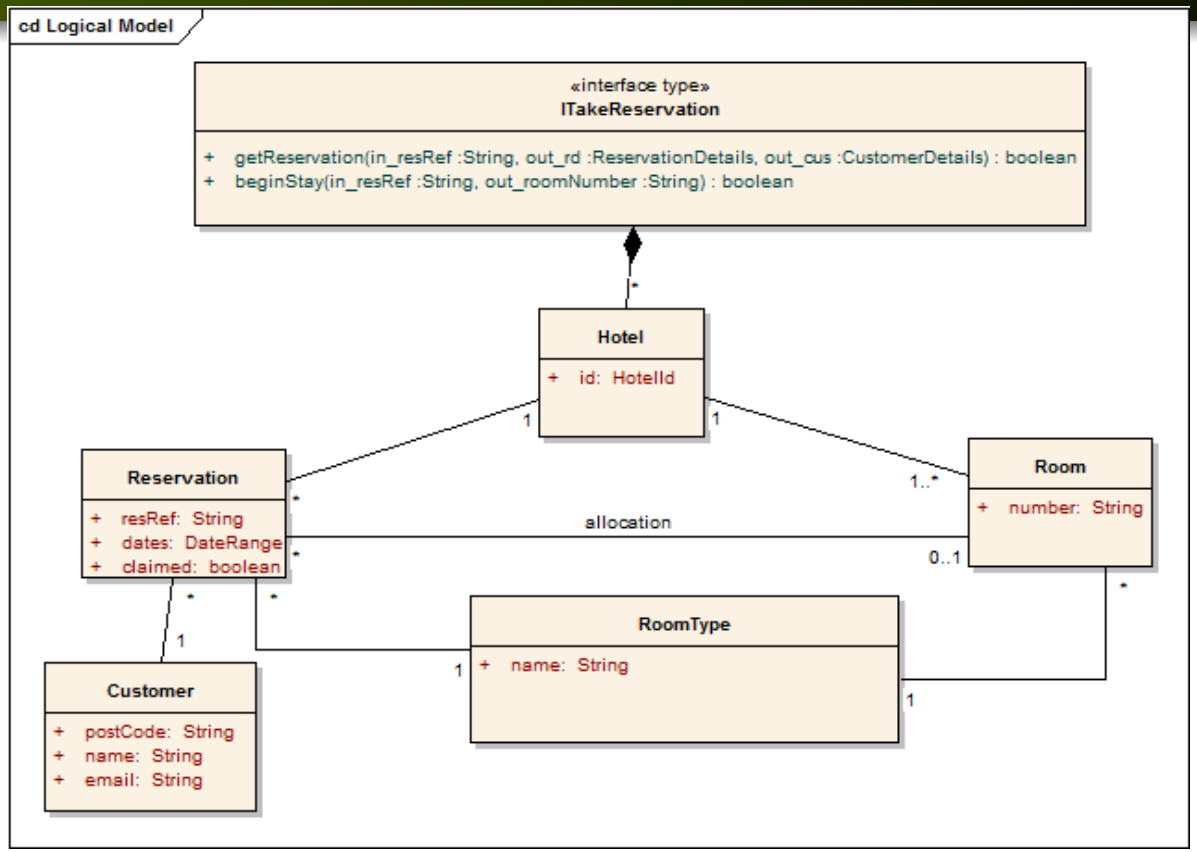

Figure: System Interface Specification Diagram `ITakeUpReservation`

- It needs now the attribute `number` of `Room`
- It does not need the attribute `name` of Hotel nor the attribute `available(during)` of `RoomType`

Component Specification

Differences with respect to the specification of other interfaces

- The business' IIM and SIIM refer to the *contract of use*
- We are now more interested in the *contract of realization* specification
- The most important thing here is to describe the dependencies between a component and other interfaces
- Includes the *realization constraints* and combination of components

Component Specification II

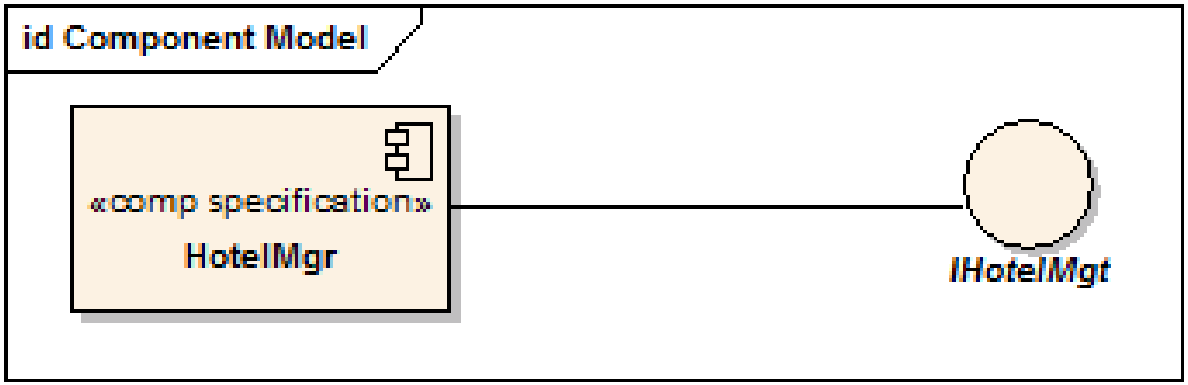

Figure: Component Specification Diagram of `HotelMgr`

The component must yield the interface `IHotelMgt` and it cannot be prevented from using other interfaces

Component Specification III

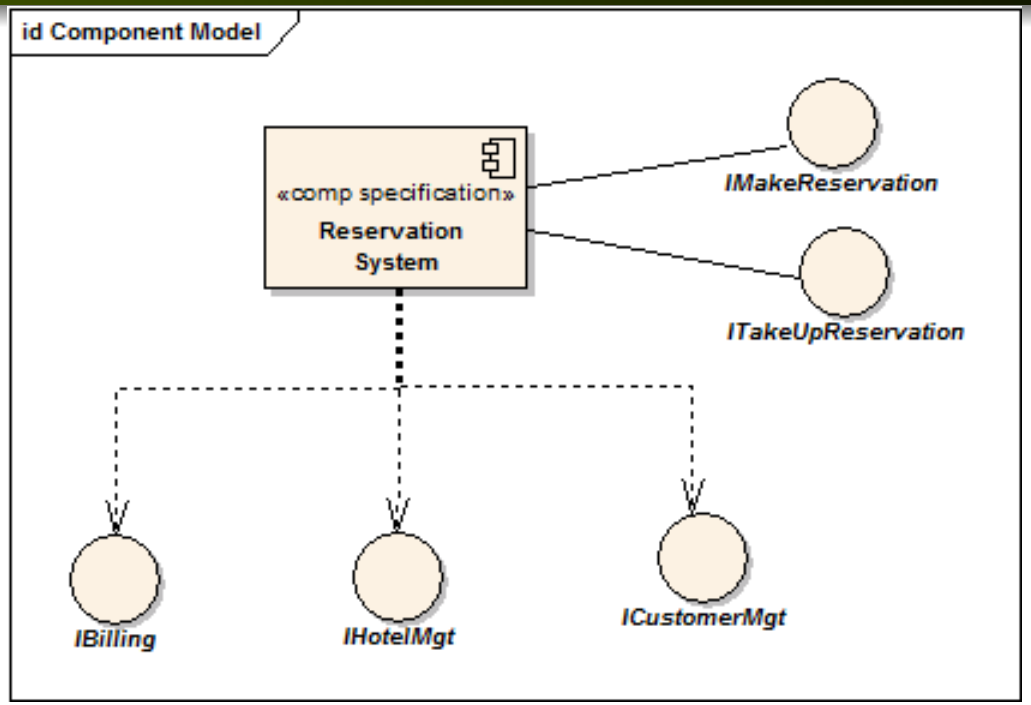

Figure: Component Specification Diagram of `HotelMgr`

- The component must yield 2 system interfaces and it has to use 3 more business interfaces
- It does not tell how these interfaces will be used during the component implementation activity

Component Specification IV

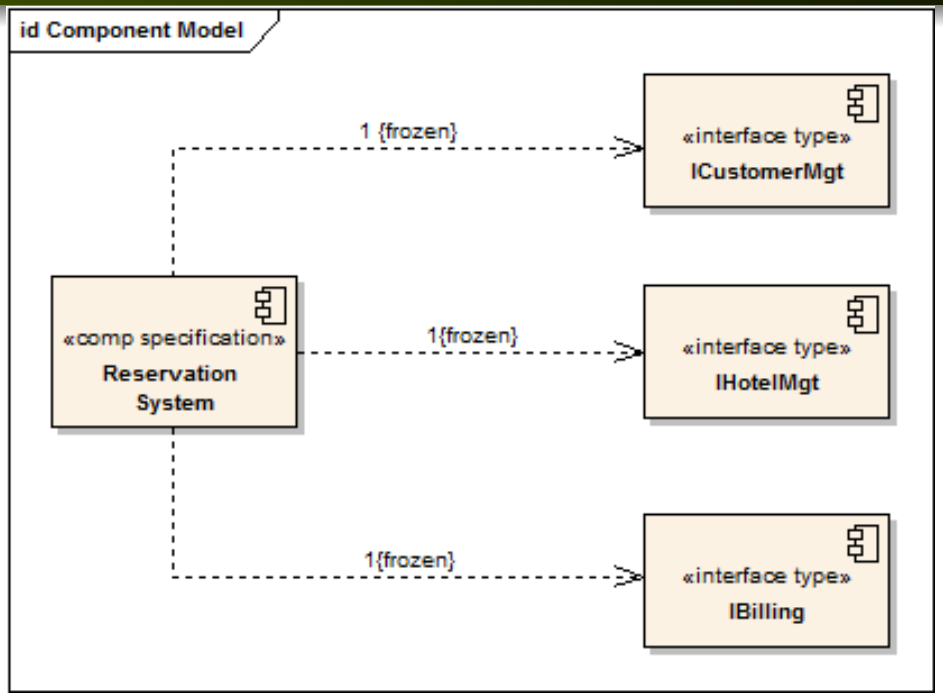

Figure: Component Specification Diagram of `ReservationSystem`

- All the component implementations have to use the same objects that each one its interfaces offer
- The `frozen` constraint: the component-object will have the same objects all its life-time long

Constraints between interfaces

To complete the specifications of the components

- How the interfaces *provided* by one component relate to each other?
- How the interfaces *required* in one component relate to each other?

Provided Interfaces

Constraints that apply over these interfaces

It must be made clear that types with the same name and yielded by 2 or more interfaces refer to the same concept

```
1 context ReservationSystem
2 --constraints between provided interfaces
3 IMakeReservation::hotel = ITakeUpReservation::hotel
4 IMakeReservation::reservation = ITakeUpReservation::reservation
5 IMakeReservation::customer = ITakeUpReservation::customer
```

Instances of an IIM type such as `IMakeReservation` are *logically* equal (=) that instances of `ITakeUpReservation` (IIM type as well)

Provided and required interfaces

Constraints among all the interfaces

- Implementations of *provided* interfaces obtain all the information they need from *business components*, i.e., these interfaces do not implement common types
- Neither do we need to specify message protocols established between a *provided* interface and a *required* interface
- We only need to describe those OCL constraints that make match information models of all the interfaces

```
1 context ReservationSystem
2 --constraints between provided and required interfaces
3 IMakeReservation::hotel = IHotelMgt::hotel
4 IMakeReservation::reservation = IHotelMgt::reservation
5 IMakeReservation::customer = ICustomerMgt::customer
```

Interface factorization

Motivation

Each interface has to have a different information model, but sometimes they only differ from each other by small changes, and thus a lot of redundance happens in the complete specification

Steps

- To include new *abstract* interfaces that act as *super types* of other interfaces that share information between them
- The abstract interface holds all the common elements and operations of several IIMs
- Definition of abstract interfaces could be useful when the *use case models* –of which the interfaces originate– share a set of *actors*

Interface factorization II

Assignment

1) Factorize the common information elements of interfaces: `IMakeReservation` and `ITakeUpReservation` and put them in a new interface: `IReservationSystem`. Then, the interfaces `IMakeReservation` and `ITakeUpReservation` inherit from `IReservationSystem`
2) Make the class diagrams of the interfaces `IReservationSystem` and rebuild the `IMakeReservation` one

Fundamental references

Cheesman, J. and Daniels, J. (2001).

UML Components: A Simple Process for Specifying Component–based Software.

Component Software Serirs. Addison-Wesley, first edition.

Eden, A., Hirshfeld, Y., and Kazman, R. (2006).

Abstraction classes in software design.

IEE Software, 153(4):163–182.

Exposito, D. and Saltarello, A. (2009).

Architecting Microsoft .NET solutions for the enterprise.

Microsoft Press, Redmond, Washington.

Szyperski, C. (1998).

Component Software. Beyond Object-Oriented Programming.

Addison–Wesley, **Básica**.