Componentes con UML

Una forma simple de especificar software basado en componentes

M.I. Capel

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

DSBCS

Máster Universitario en Ingeniería Informática

21 de octubre de 2015

Índice

Índice

Índice

Índice

Especificación de SBCs

Contratos en Sistemas Basados en Componentes

- Contrato de utilización
- Contrato de realización

Especificación de una interfaz

Determinar qué partes la componen y describirlas sin ambigüedad.

Especificación de un componente

Se trata fundamentalmente de agrupar las interfaces que implementará e indicar las restricciones que afectan a la realización del propio componente.

Interfaces provistas

Una interfaz de este tipo cumple:

- Es implementada por el propio componente o
- Es implementada por uno de los objetos del componente o
- Es ofrecida por un puerto del componente

Figura: El componente `Servicios Meteorológicos` implementa la interfaz `Pronóstico Tiempo`

Interfaces requeridas

Una interfaz de este tipo cumple:

- Es una dependencia de uso del propio componente
- Es una dependencia de uso de uno de los objetos del componente
- Es necesitada por un *puerto público* del componente

Figura: El componente `Servicios Usuario` requiere la interfaz `ServiciosIOrder`

Proceso de Especificación de Interfaces

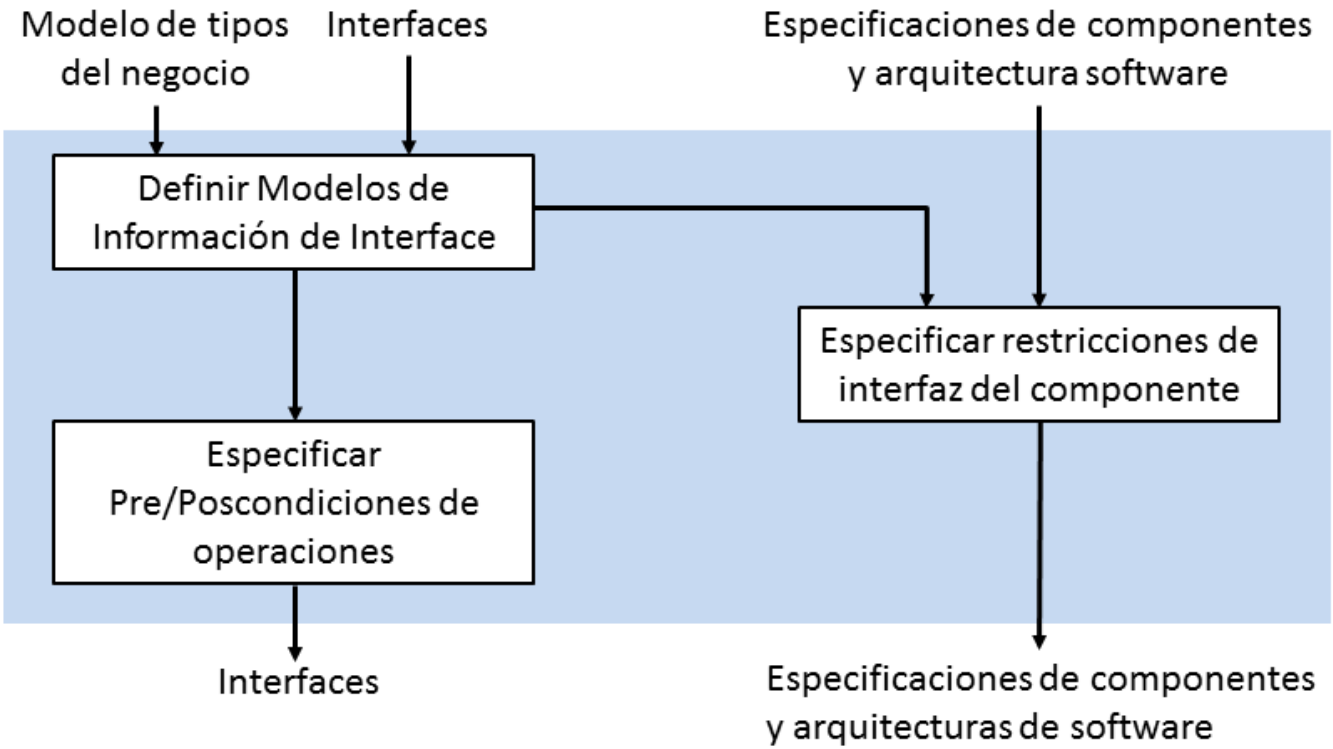

Figura: Especificación de un componente dentro del flujo de trabajos completo en esta etapa

Especificación de interfaces vs. operaciones

Especificación de operaciones de componentes

- Carecen de cualquier información estructural acerca de un componente
- Nivel de descripción inadecuado respecto de la gestión de dependencias

Especificación de interfaces de componentes

- Agrupación de operaciones relacionadas
- Dicha agrupación es revisitada en la tarea de *factorización*
- Introducción de subtipado en las interfaces
- Pueden incluir sólo 1 operación *autocontenida* y no dependiente

Especificación de operaciones

¿Qué tiene que incluir y qué no la especificación de una operación?

- Descripción de la relación entre las entradas, salidas y el estado del objeto componente
- El efecto que tiene la llamada sobre la anterior relación

¿Qué tiene que garantizar?

- Transparencia de las relaciones del objeto componente con otros

Especificación de operaciones II

Elementos de la especificación:

- Los parámetros de entrada y salida
- Las restricciones que sean de aplicación
- Cualquier cambio de estado resultante en el componente

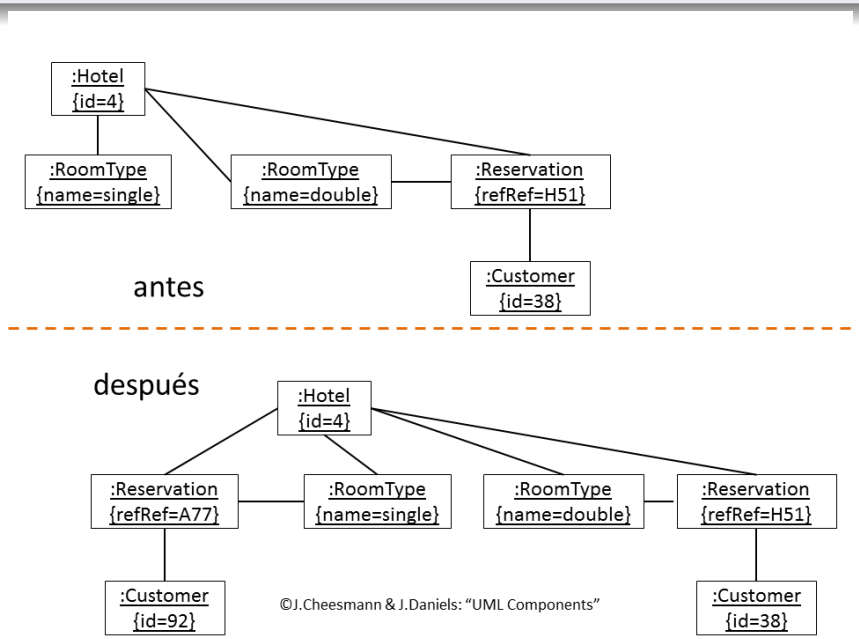

Figura: Efecto de la operación `IHotelMgt::makeReservation()` en parte del estado del componente

Especificación de interfaces I

Características del modelo:

- Suficiente para permitir la especificación de las operaciones de la interfaz
- Así como restricciones y efectos sobre el estado del componente
- Descripción de cambios del estado como resultado de las operaciones
- Se construirá incrementalmente conforme se elabora la especificación de las operaciones especificaciones de las operaciones, añadiendo tipos, atributos, etc.

Especificación de interfaces II

Condiciones que respetará el modelo de especificación:

- Las interfaces sólo son asociadas a *información tipada*
- Sólo contendrán información del conjunto de estados propios de ese componente
- Nunca pueden dar información sobre implementación estado del componente
- Tampoco sobre su persistencia

Modelo de Especificación de Interfaz

Representación del estado del *objeto componente* del que depende la interfaz

- Necesidad de un Modelo de Información de Interfaz (MII)

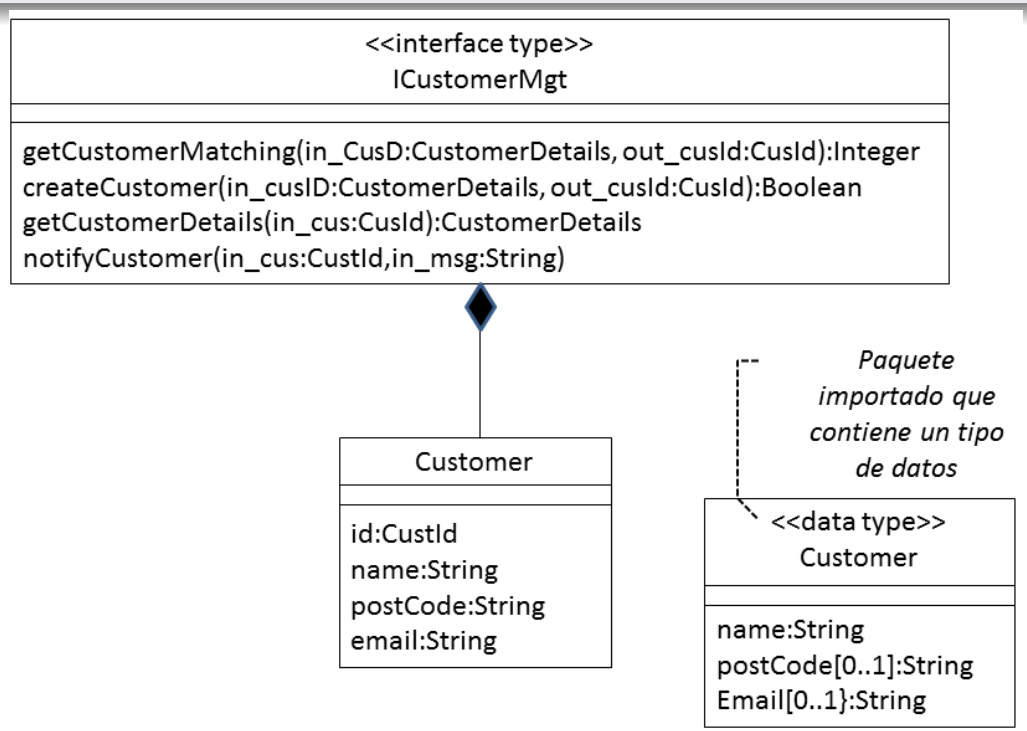

Figura: Diagrama previo al modelo de especificación de la interfaz `ICustomerMgt`

Modelo de Información de Interfaz (MII)

Discusión del ejemplo `ICustomerMgt`

- `Customer` es una *Información con Tipo*
- Los tipos de una interfaz no pueden mantener asociaciones con nada fuera del modelo
  - Ubicación en el mismo paquete que la interfaz
  - Salvo *subtipos* heredados entre interfaces, los tipos no se comparten pero se pueden importar

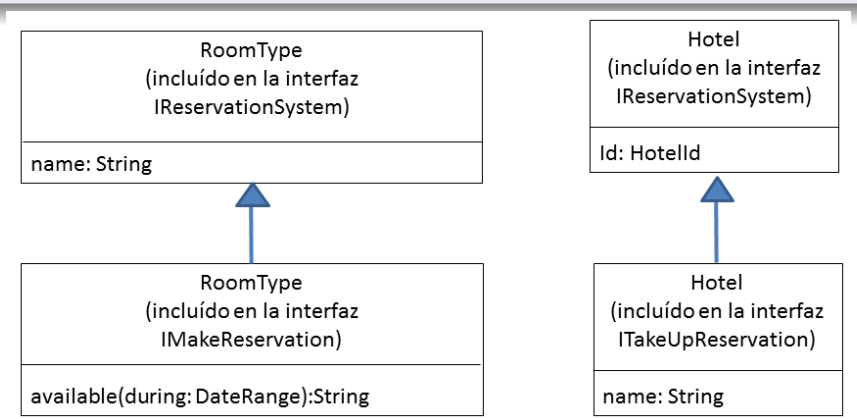

Figura: Ejemplo de herencia de tipos después de una factorización de interfaces

Pre y postcondiciones

- Cada operación posee una *pre* y *poscondición* asociada, que especifican con detalle qué hará la operación
- No proporcionan información algorítmica o de implementación
- Actúan como la *letra pequeña* de un contrato con el cliente
- Precondición: la condición que garantiza que la ejecución de la operación hará cierta la poscondición
- La llamada de la operación aludida es totalmente independiente del valor de certeza de su precondición
- Cualquier suposición respecto de la ejecución de las operaciones es responsabilidad del cliente
- Las garantías contractuales (poscondición) son responsabilidad del proveedor de la operación

OCL

Definición

Object Constraint Language es un lenguaje declarativo que permite construir expresiones lógicas y sirve, por ejemplo, para especificar condiciones contractuales en la especificación de interfaces de componentes software

Expresiones OCL de pre y poscondiciones semánticamente correctas

- Se pueden referir a los parámetros, resultado de las operaciones y al estado del objeto componente
- No se pueden referir a elementos de otras interfaces
- La especificación de interfaces sólo afectan localmente

Expresiones con OCL

Especificación OCL de operación de cambio de nombre

```
1  context ICustomerMgt::changeCustomerName(in cus:CustId, in
       newName:String)
2
3  pre:
4     --cus es un identificador valido de cliente
5     customer->exists(c | c.id = cus)
6  post:
7     --el nombre del cliente cuyo identificador es 'cus' se
          convierte en 'newName'
8     customer->exists(c | c.id = cus and c.name = newName)
```

'customer' se refiere al conjunto de clientes asociados a un objeto-componente de soporte: `ICustomerMgt`

Expresiones con OCL (II)

Especificación de operación para obtener detalles de un cliente

```
1  context ICustomerMgt::getCustomerDetails(in cus: CustId):
       CustomerDetails
2  pre: --cus es un identificador valido de cliente
3     customer->exists(c | c.id = cus)
4  post: --los detalles devueltos tras la ejecucion coinciden con
       del cliente cuyo identificador es 'cus'
5        --encontrar al cliente
6     Let elCliente = customer->select(c | c.id = cus) in
7        --especificar el resultado
8        result.nombre= elCliente.nombre and
9        result.codigoPostal = elCliente.codigoPostal and
10       result.email = elCliente.email
11    --la devolucion es implicita con la asignacion de la variable
         'result'; no hay cambio de estado del componente
```

Esta operación no cambia el estado del objeto que la realiza: la poscondición sólo especifica el resultado a devolver.

OCL (II)

Condiciones en poscondiciones

- Las expresiones de poscondición pueden referirse tanto al estado antes de la ejecución de la operación (`@pre` de OCL) como a su estado posterior
- Permiten escribir expresiones que especifican cómo cambian los atributos o las asociaciones en el MII como resultado de la ejecución de una operación

Creación del Modelo de Información de una Interfaz

Generalidades del MII de reservas en hoteles

- La interfaz `IHotelMgt` se preocupa de organizar reservas en hoteles asignando habitaciones
- `ICustomerMgt` se preocupa de gestionar los clientes
- Tipos que son responsabilidad de `IHotelMgt`:
  - `Hotel`
  - `RoomType`
  - `Room`
  - `Reservation`
- Tipos que son responsabilidad de `ICustomerMgt`:
  - `Customer`

Creación del Modelo de Información de una Interfaz II

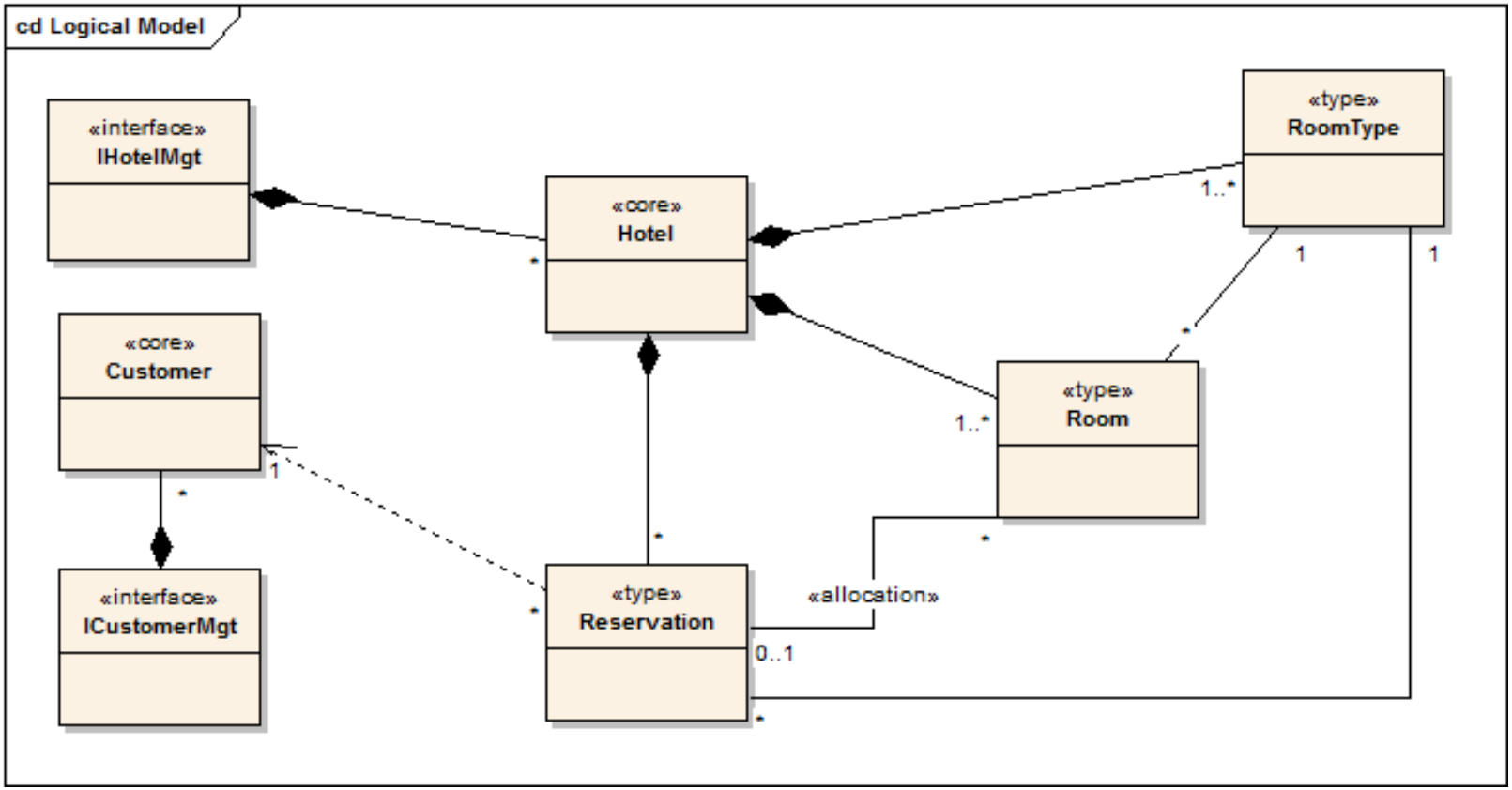

Figura: Diagrama de Responsabilidades de la Interfaz `IHotelMgt`

Creación del Modelo de Información de una Interfaz III

Adaptaciones en el DRI para obtener el MII

- Los tipos: `Hotel`, `RoomType` y `Reservation` han de incluirse en el MII de `IHotelMgt`
- La asociación entre reservas y clientes no tiene por qué integrarse en el MII
- Pueden transformarse las asociaciones en el DRI:
  - Inclusión de asociación directa: `IHotelMgt -> Reservation`
  - De asociación *derivada* a *directa*: `Hotel->Reservation`
- Se pueden eliminar asociaciones del DRI:
  - Asociación derivada `Hotel -> RoomType`
- Se pueden añadir atributos: atributo `claimed` a `Reservation`

Creación del Modelo de Información de una Interfaz IV

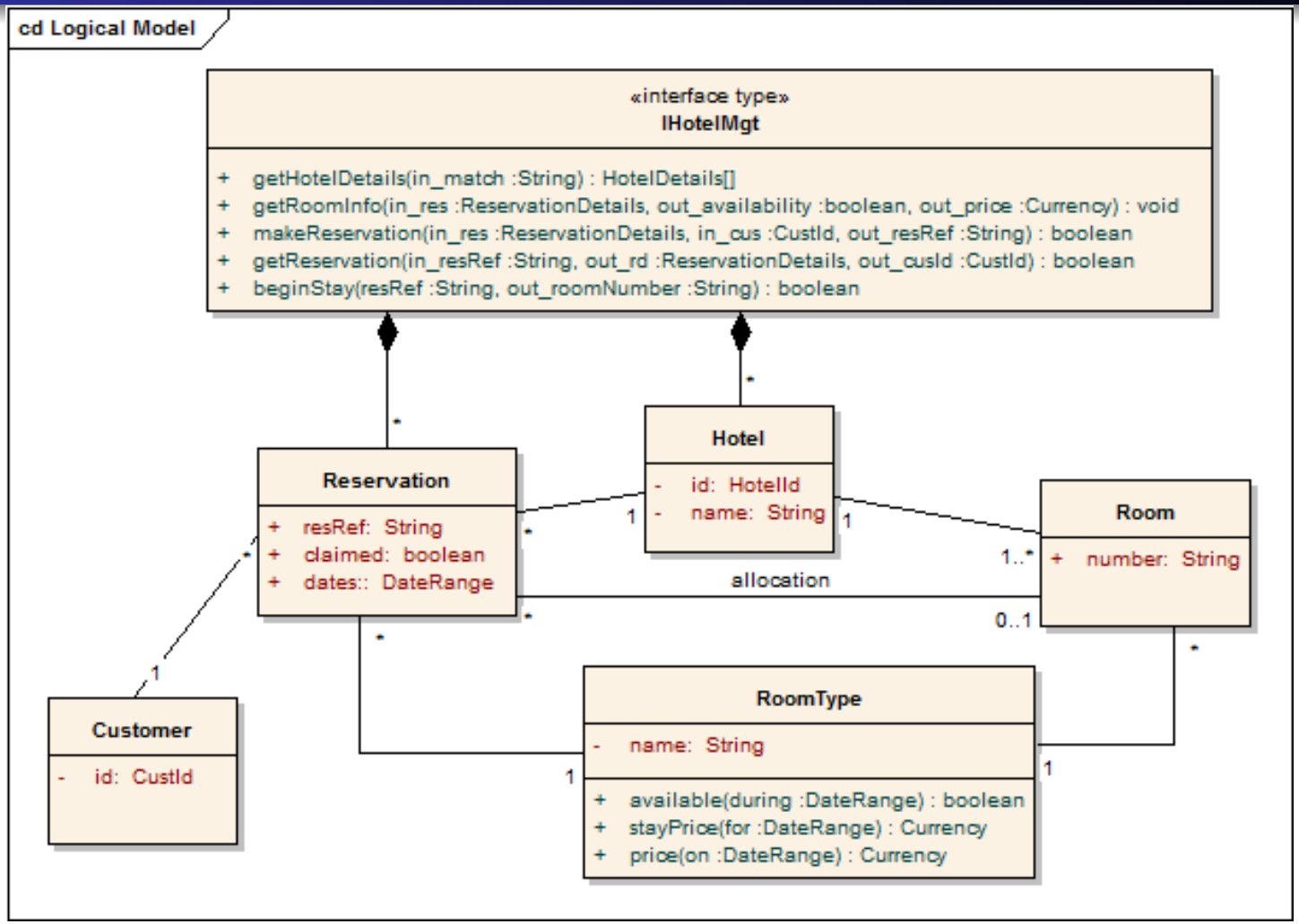

Figura: Diagrama de Especificación de Interfaz de `IHotelMgt`

Creación del Modelo de Información de una Interfaz V

Invariantes

- *Invariante*: restricción asociada a un tipo que debe mantenerse cierta para todas las instancias del mismo
- Los invariantes pueden expresarse gráficamente con UML
- Los invariantes se pueden escribir con expresiones OCL:

```
1 context r: Reservation inv:
2   --una reserva esta confirmada (claimed) si tiene ya una
        habitacion asignada
3   r.claimed = r.allocation->noEmpty
```

- A partir de la definición anterior se podría utilizar “claimed” como forma abreviada de la relación
- Un invariante conecta diferentes partes de la información contenida en una especificación

Especificación de operaciones con OCL

Ejercicio propuesto

Especificar completamente la operación `IHotelMgt::makeReservation (...)` con OCL; utilizando los operadores: `exists`, `select` y `asSequence->first`

Especificación de Interfaces del Sistema

Interfaces del Sistema

- El Modelo de Información de una Interfaz de Sistema (MIIS) es un subconjunto del *modelo del tipo de negocio*
- Este modelo de interfaz pretende agrupar las funciones que realiza el sistema sin asistencia, automáticamente
- Diferencias respecto de la elaboración de los MII de negocio:
  - MIIS no tiene porqué contener todos los tipos del modelo de negocio
  - La elaboración de *diagramas de especificación de responsabilidades* no proporciona tanta información como lo hacen en el caso de la elaboración del MII
  - No se tenga claro todos tipos del modelo de negocio que hay que incluir hasta la programación de las operaciones

Modelos de Información de Interfaces del Sistema

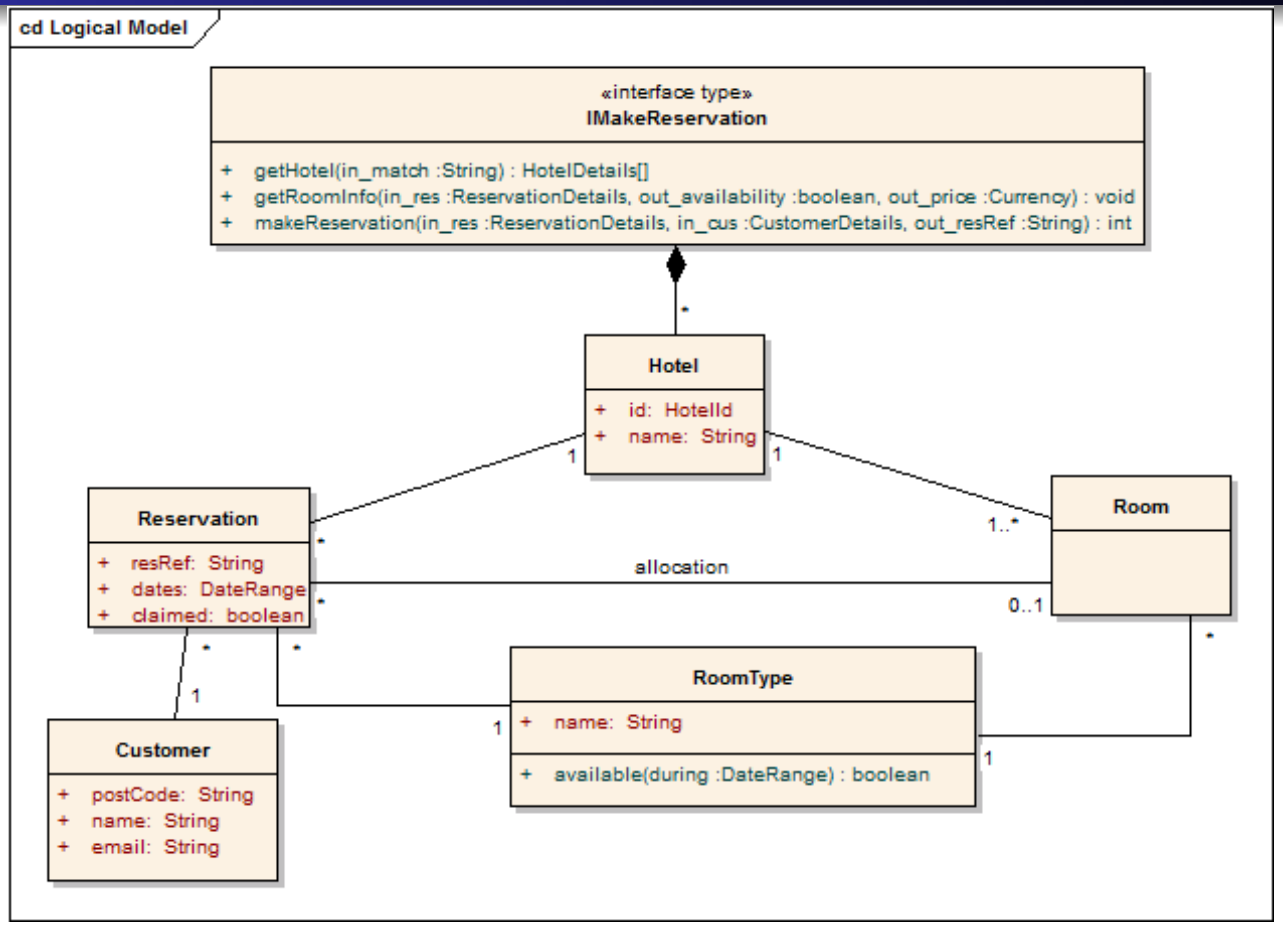

Figura: Diagrama de Especificación de Interfaz de Sistema `IMakeReservation`

Este modelo no necesita el atributo `number` de la clase `Room` del MII de negocio

Modelos de Información Interfaces Sistema II

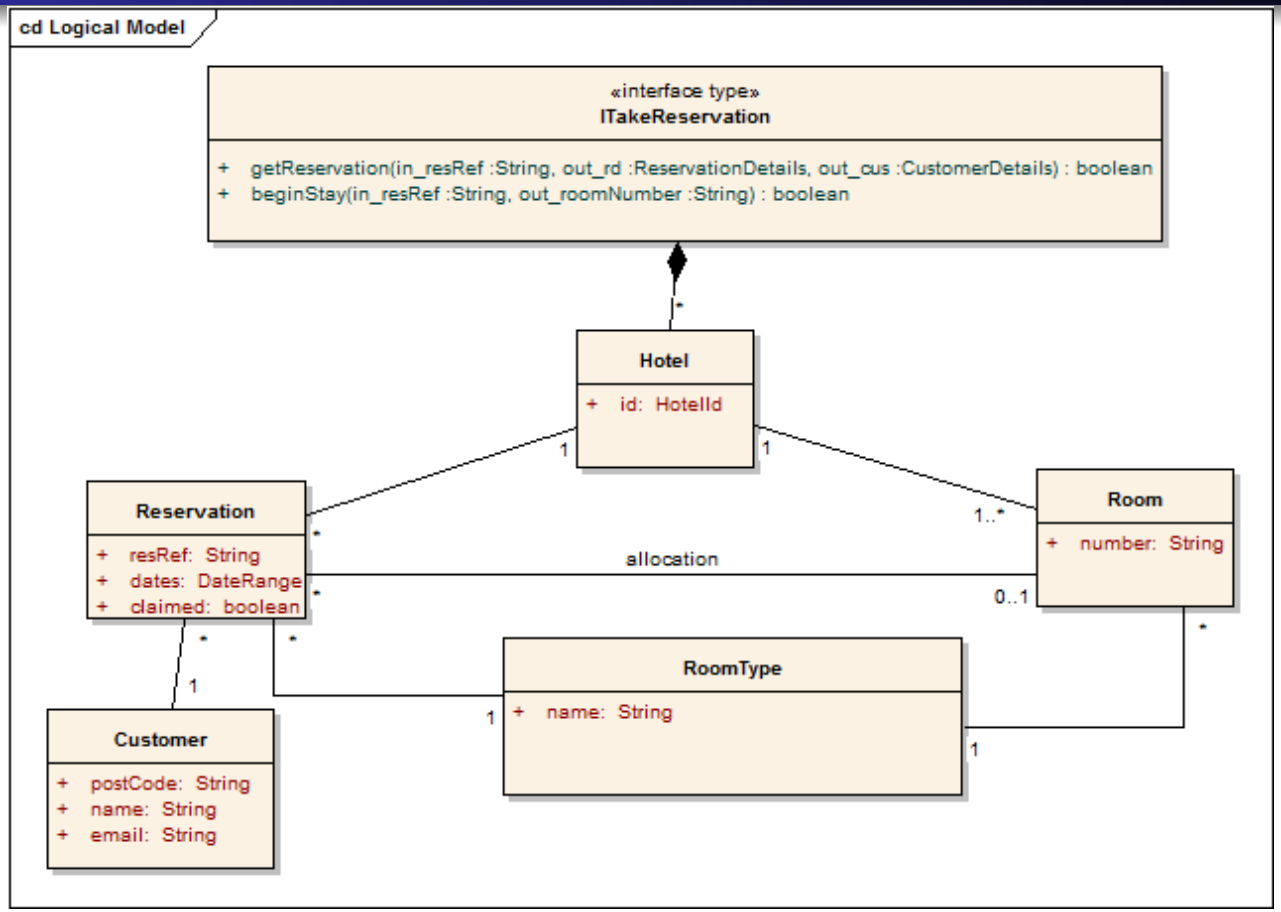

Figura: Diagrama de Especificación de Interfaz de Sistema `ITakeUpReservation`

- Necesita ahora el atributo `number` de la clase `Room`
- No necesita ni el atributo `name` de Hotel ni `available(during)` de `RoomType`

Especificación de componentes

Diferencias con la especificación de otras interfaces

- Las interfaces MII de negocio y MIIS se refieren al *contrato de utilización*
- Ahora nos preocuparemos más por el *contrato de realización*
- Los más importante es describir las dependencias entre un componente y otras interfaces
- Incluye las restricciones de realización y combinación de componentes

Especificación de componentes II

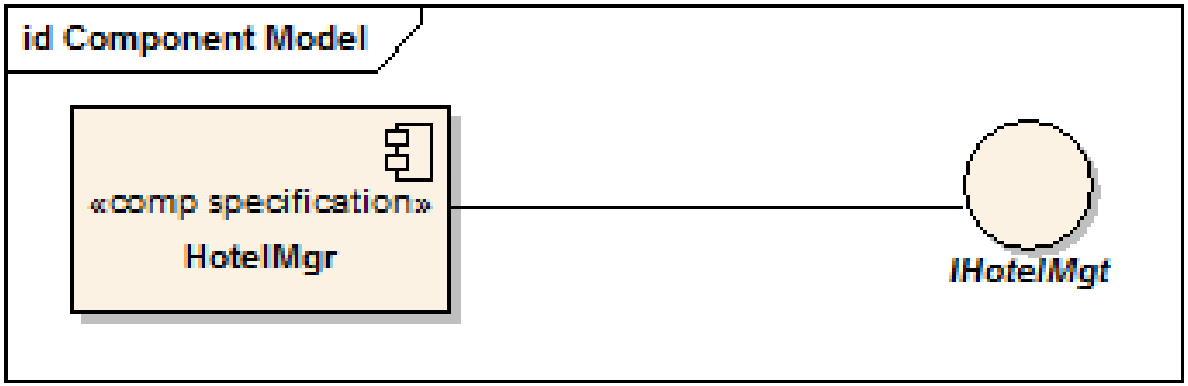

Figura: Diagrama de Especificación de Componente de `HotelMgr`

El componente debe ofrecer la interfaz `IHotelMgt` y no se le impide utilizar otras interfaces

Especificación de componentes III

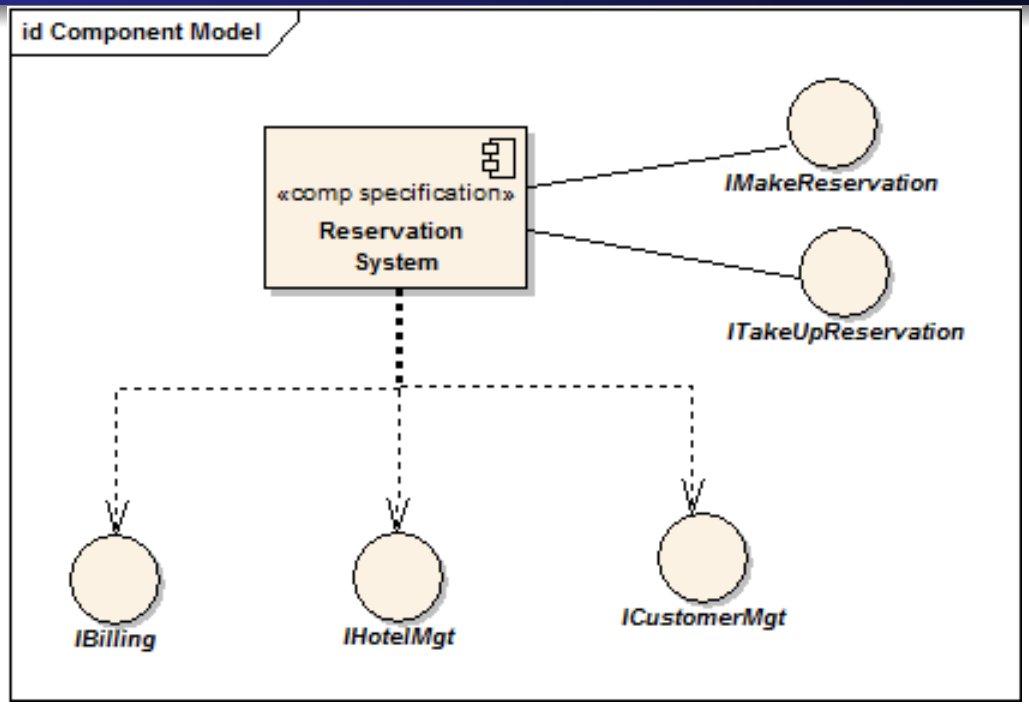

Figura: Diagrama de Especificación de Componente de `HotelMgr`

- El componente debe ofrecer 2 interfaces de sistema y ha de utilizar 3 interfaces de negocio adicionales
- No dice cómo van a utilizarse estas interfaces en las implementaciones del componente

Especificación de componentes IV

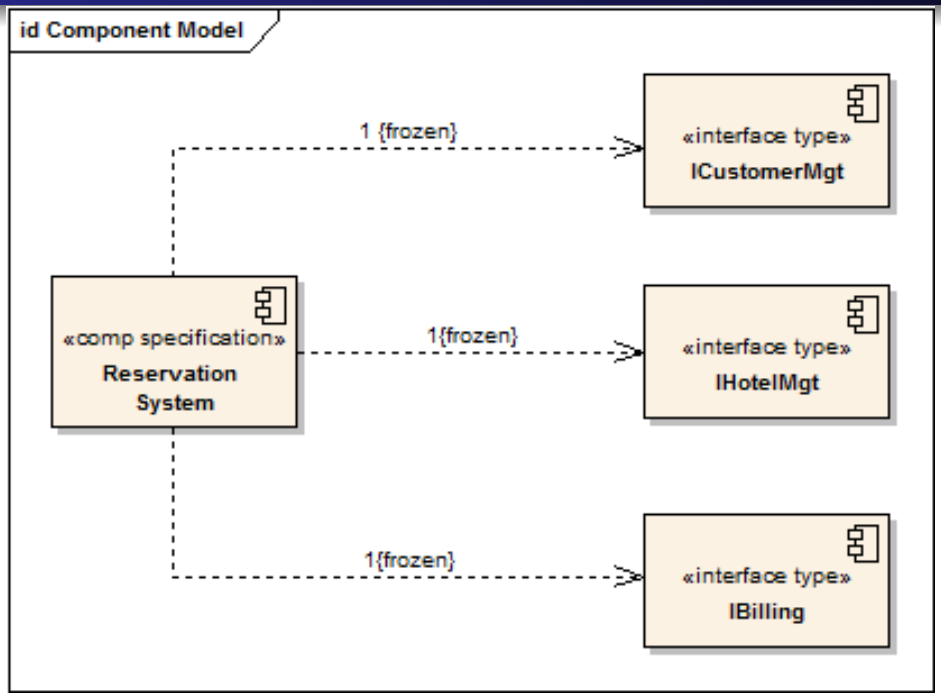

Figura: Diagrama de Especificación de Componente de `ReservationSystem`

- Todas las implementaciones del componente debe utilizar los mismos objetos que ofrecen cada una de las interfaces
- Restricción `frozen`: siempre los mismos objetos a lo largo de todo el tiempo de vida del objeto-componente

Restriciones entre interfaces

Completar las especificaciones de los componentes

- ¿Cómo se relacionan entre sí las interfaces *provistas* por un componente?
- ¿Cómo se relacionan con las interfaces *requeridas* en el componente?

Interfaces provistas

Restricciones que aplican sobre estas interfaces

Se pretende dejar claro que los tipos del mismo nombre ofrecidos en 2 ó más interfaces se refieren al mismo concepto

```
1 context ReservationSystem
2 --restricciones entre interfaces provistas
3 IMakeReservation::hotel = ITakeUpReservation::hotel
4 IMakeReservation::reservation = ITakeUpReservation::reservation
5 IMakeReservation::customer = ITakeUpReservation::customer
```

Las instancias del tipo del MII `IMakeReservation` son *lógicamente* las mismas (=) que las instancias del MII `ITakeUpReservation`

Interfaces provistas y requeridas

Relaciones entre todas las interfaces

- Las implementaciones de las interfaces *provistas* obtienen la información que necesitan de los *componentes de negocio*: no reimplementan tipos comunes
- Tampoco hay que especificar los protocolos de mensajes que se establecen entre una interfaz *provista* y las interfaces *requeridas*
- Describir sólo las restricciones en OCL que hacen que los modelos de información de todas las interfaces *casen*

```
1 context ReservationSystem
2 --restricciones entre interfaces provistas y requeridas
3 IMakeReservation::hotel = IHotelMgt::hotel
4 IMakeReservation::reservation = IHotelMgt::reservation
5 IMakeReservation::customer = ICustomerMgt::customer
```

Factorización de interfaces

Motivación

Cada interfaz ha de tener un modelo de información diferente, pero a veces sólo difieren en pequeños cambios; por consiguiente, se produce mucha redundancia.

Pasos

- Introducir interfaces *abstractas* nuevas que actúen como *super tipos* de otras interfaces que compartan información
- La interfaz abstracta mantiene los elementos coincidentes de varios MIIs y también operaciones comunes
- Puede ser indicado cuando los *modelos de casos de uso* de los que proceden las interfaces comparten actores

Factorización de interfaces II

Ejercicio propuesto

1) Factorizar los elementos comunes de los modelos de información de las interfaces `IMakeReservation` y `ITakeUpReservation` y ubicarlos en una nueva interfaz `IReservationSystem` de las que las 2 interfaces aludidas heredan.
2) Elaborar los diagramas de clases de las interfaces `IReservationSystem` y rehacer el de `IMakeReservation`

Bibliografía Fundamental


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**.