

Interação dos Componentes

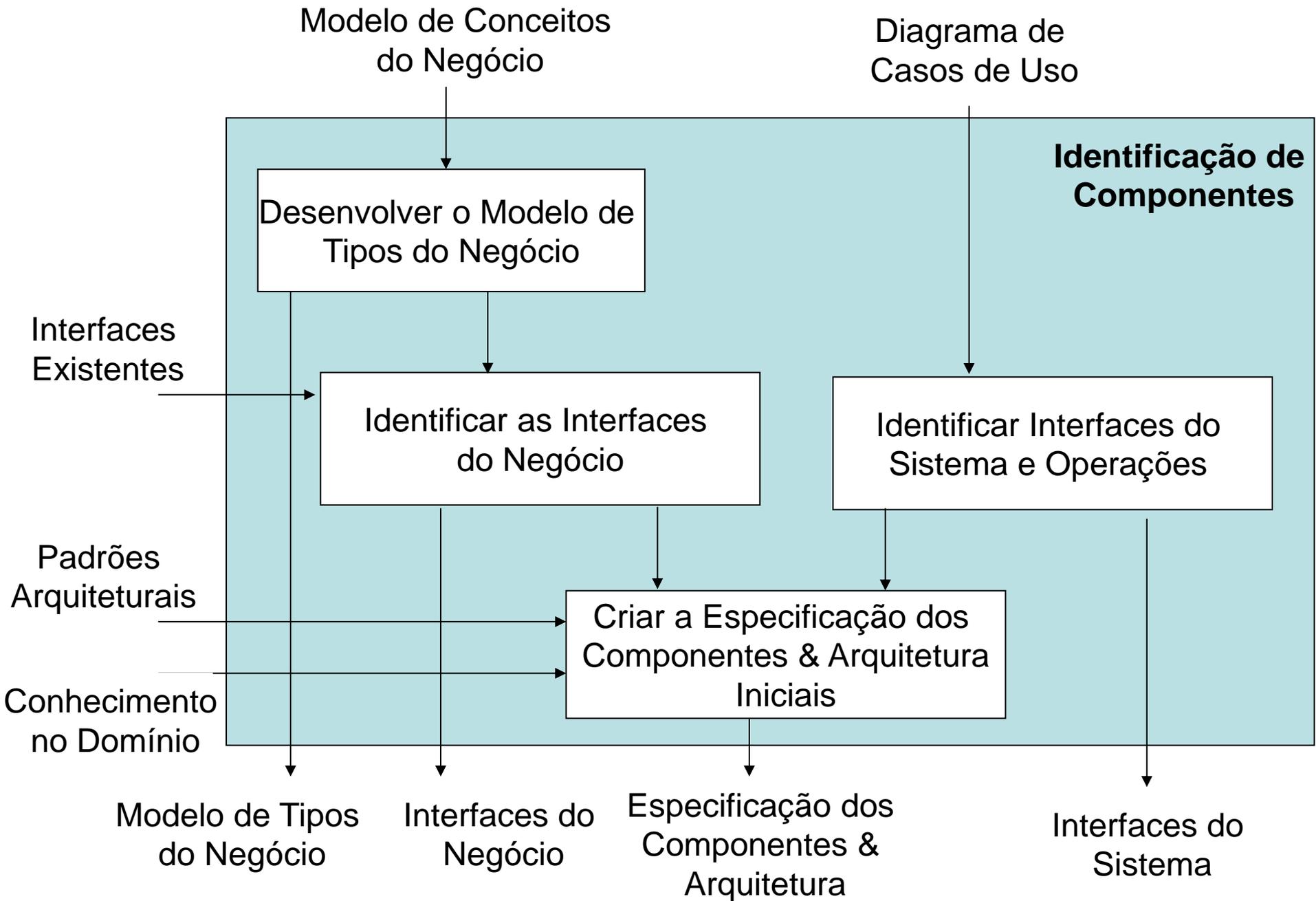
Cap. 6

Interações dos Componentes

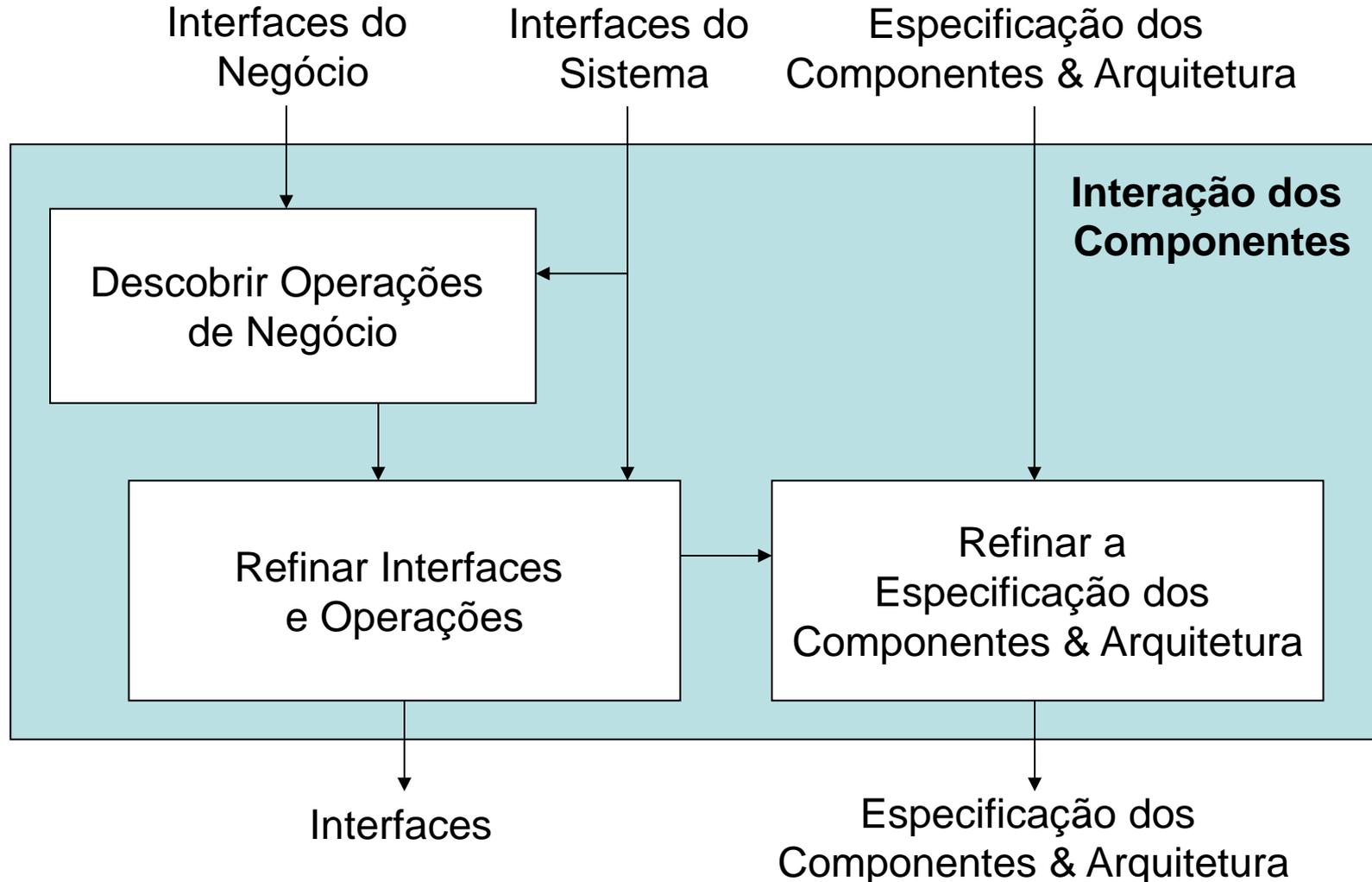
- A fase anterior forneceu o conjunto inicial dos componentes e interfaces.
- Mostra como os componentes trabalham juntos para oferecer as funcionalidades.
- A modelagem de interações é uma técnica genérica para modelagem de comportamento

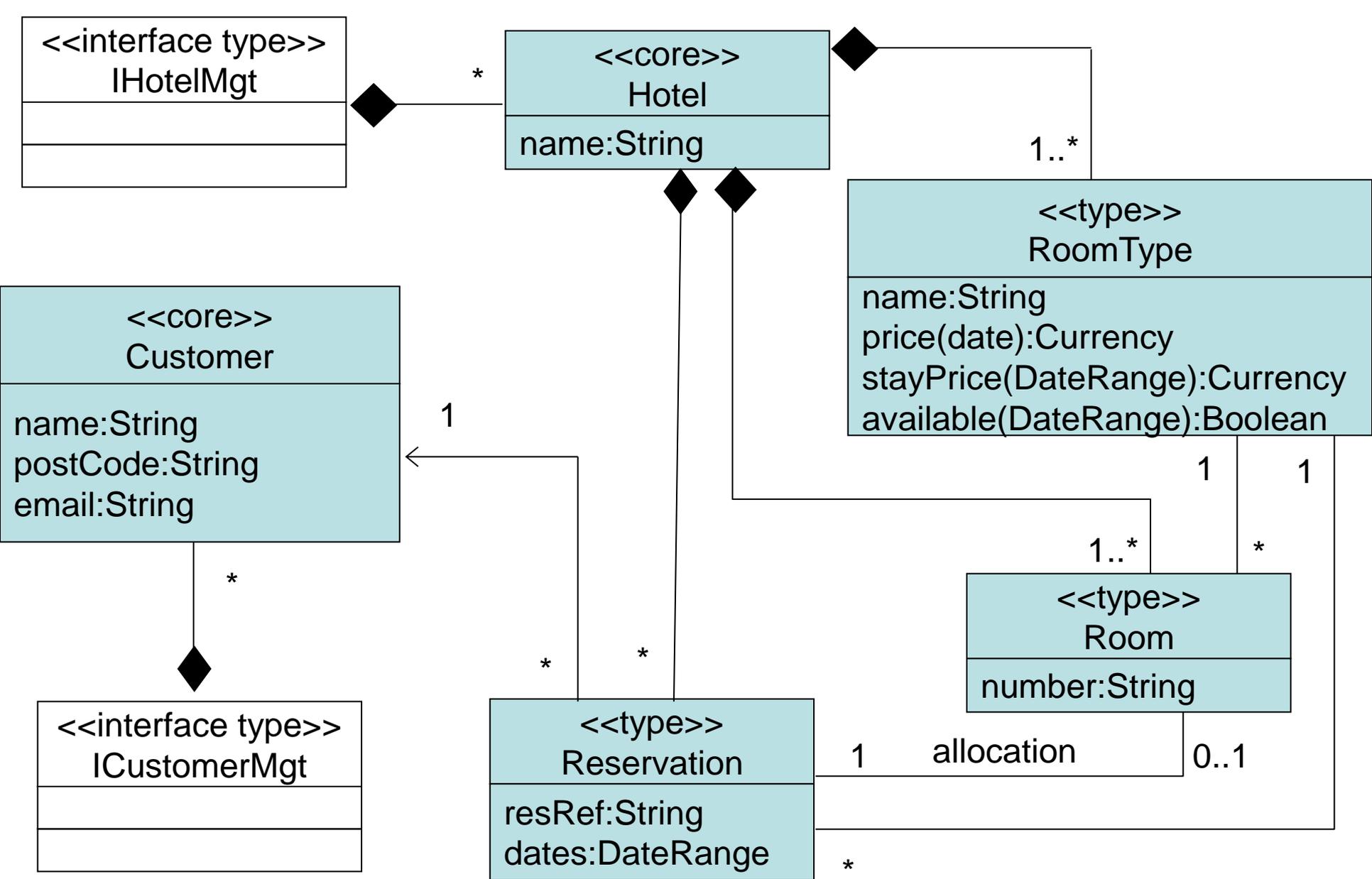
Interações dos Componentes (cont.)

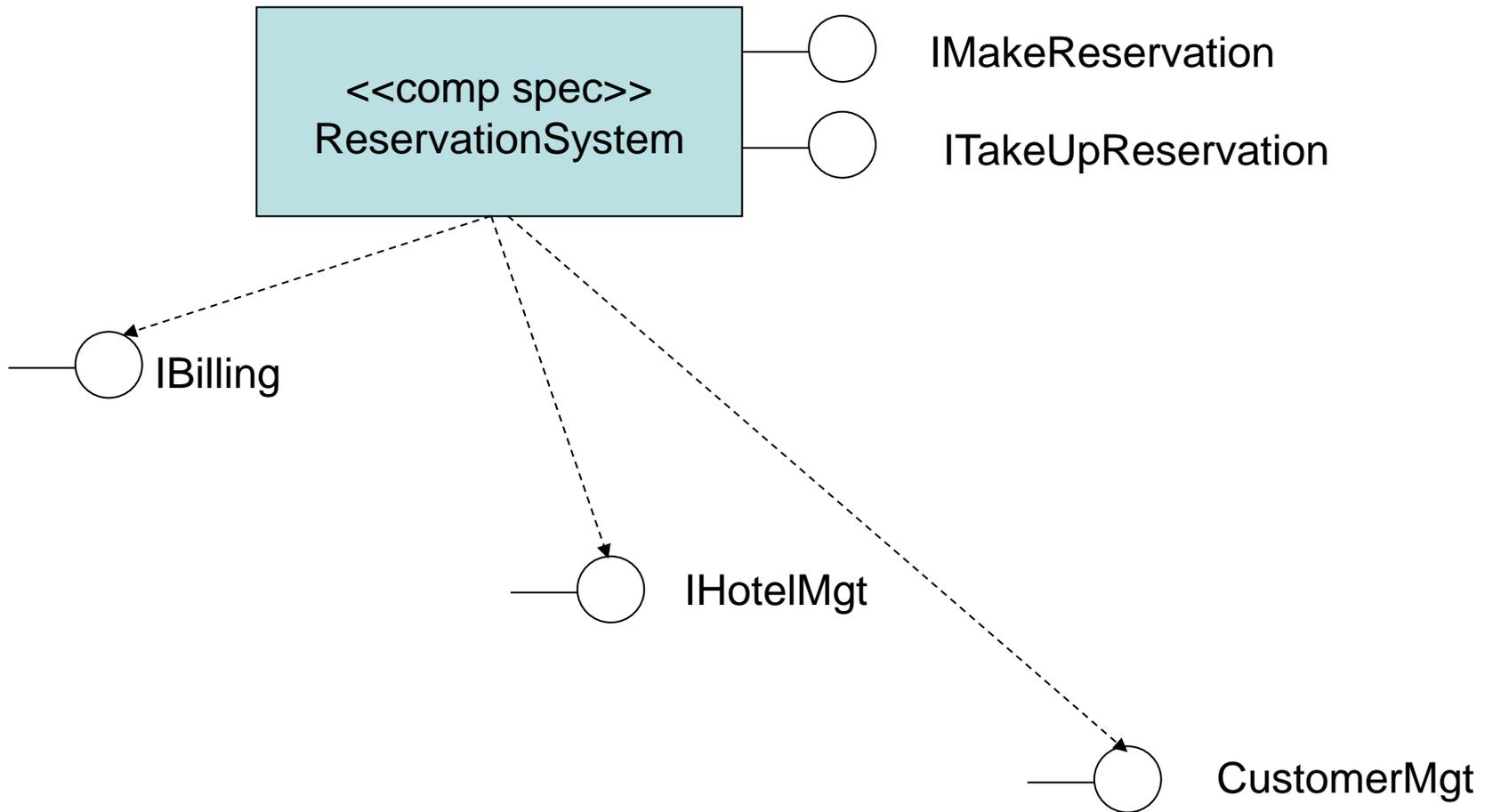
- Tem o objetivo de:
 - Definir as interações que ocorrem dentro do sistema.
 - Refinar as definições de interfaces.
 - As assinaturas das operações das interfaces do sistema não são conhecidas ainda.
 - Identificar o uso das interfaces.
 - Descobrir novas interfaces e operações.
 - As operações das interfaces de negócio não foram identificadas ainda.



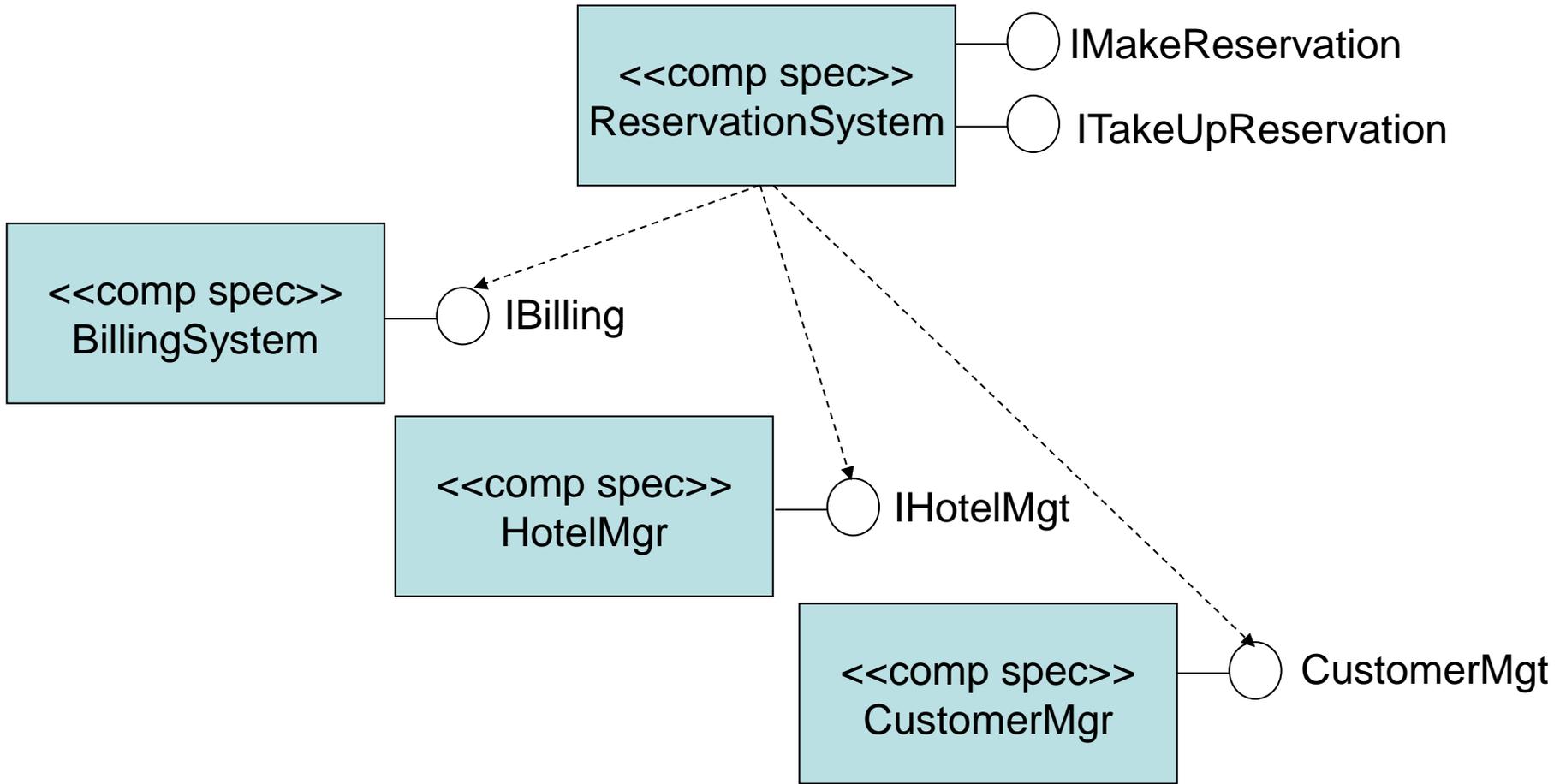
Interações dos Componentes





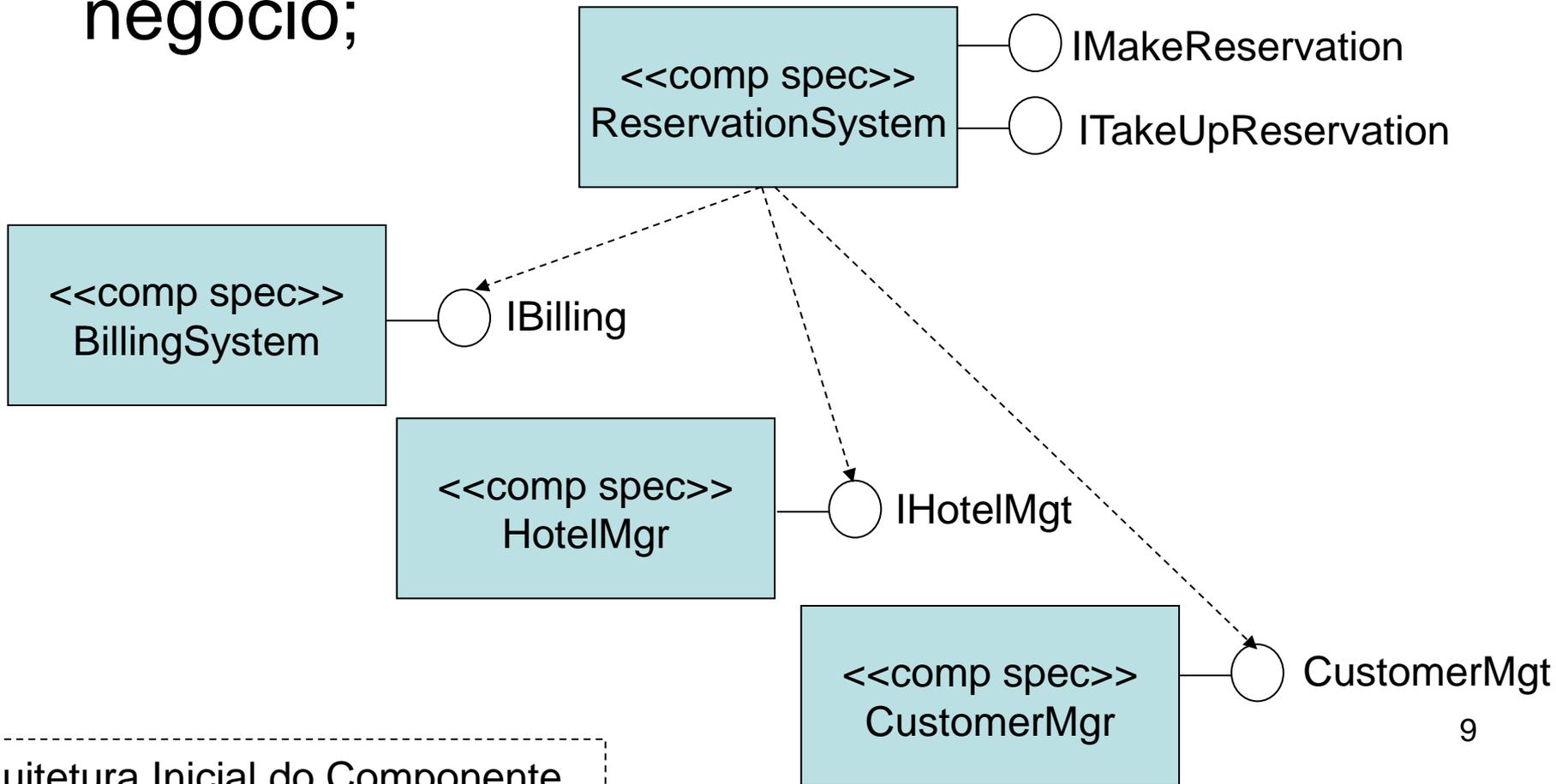


Uma Arquitetura Inicial

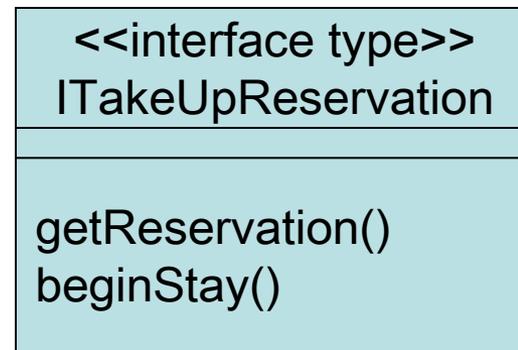
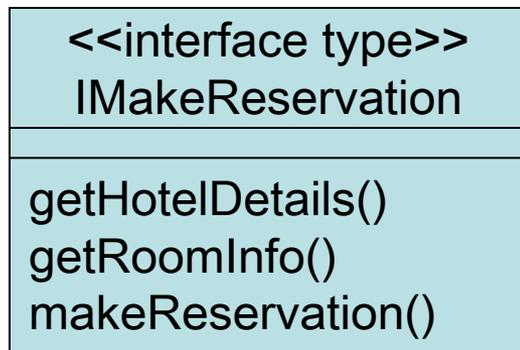


Descobrir Operações de Negócio

- O foco é descobrir as operações de negócio;



Descobrir Operações de Negócio (cont)



Conjunto inicial de operações das interfaces

Descobrir Operações de Negócio (cont.)

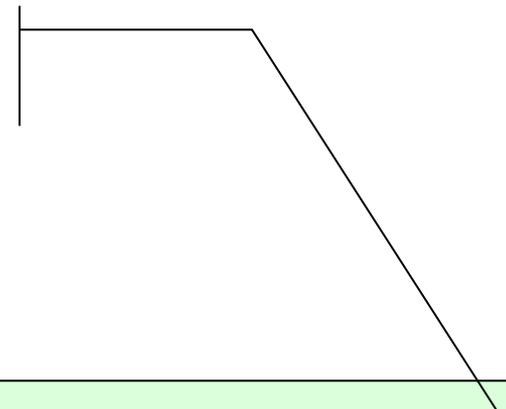
- Diagramas de colaboração são usados para descobrir e modelar cada possível fluxo de execução;
- Todos os fluxos alternativos importantes devem ser modelados;
- Deve-se encontrar e estabelecer as restrições ao fluxo de execução decorrente das chamadas as operações;

Exemplo: *getHotelDetails()*

- A operação:
 - Recebe como argumento uma string com o nome do hotel;
 - Retorna uma lista com um identificador para cada hotel e os tipos de acomodações;
- Um “tipo” é definido para retornar as informações:



Retorna uma coleção de estruturas HotelDetails



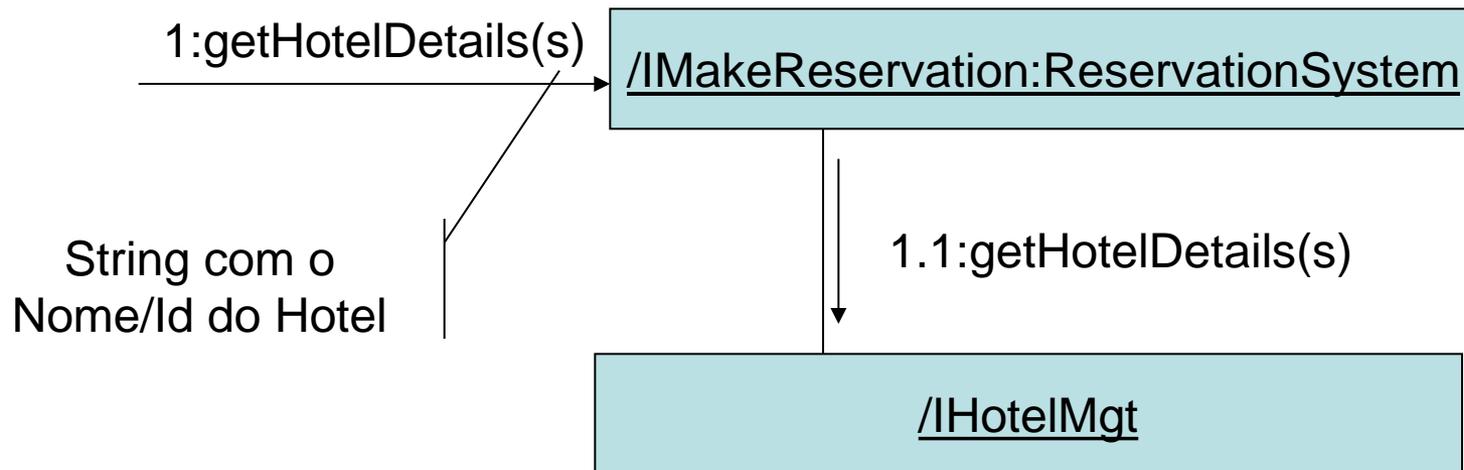
```
IMakeReservation::getHotelDetails(in match:string):HotelDetails[]
```

Exemplo: *getHotelDetails()* (cont)

- A operação é invocada dinamicamente pela camada de tipo de diálogo;
- O objeto não pode satisfazer a operação porque **componentes de sistema não armazenam dados de negócio**;
- A operação é delegada à interface `IHotelMgt`;

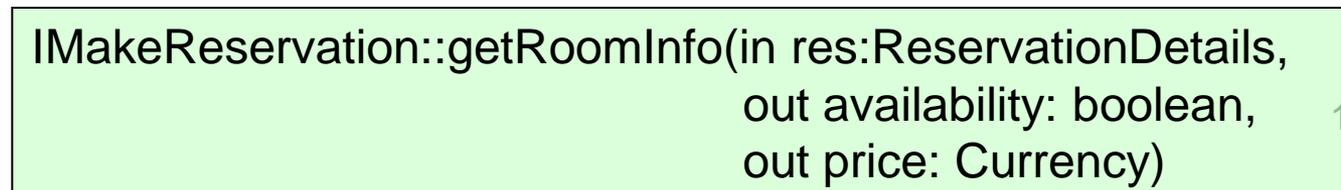
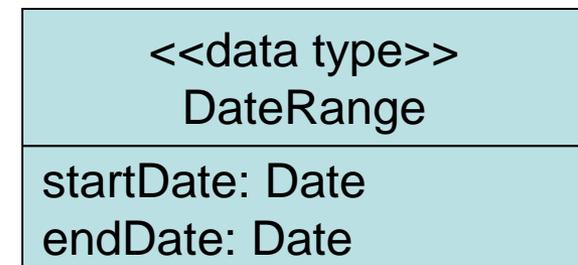
Exemplo: *getHotelDetails()* (cont)

- Com essa investigação a primeira operação de negócio é descoberta na interface `IHotelMgt`;



Exemplo: *getRoomInfo()*

- A operação:
 - Recebe como argumento um identificador do hotel, a data da estada e o tipo de acomodação;
 - Retorna a disponibilidade do tipo de acomodação e o preço;
- Também delega a operação para a *IHotelMgt*;



Exemplo: *makeReservation()*

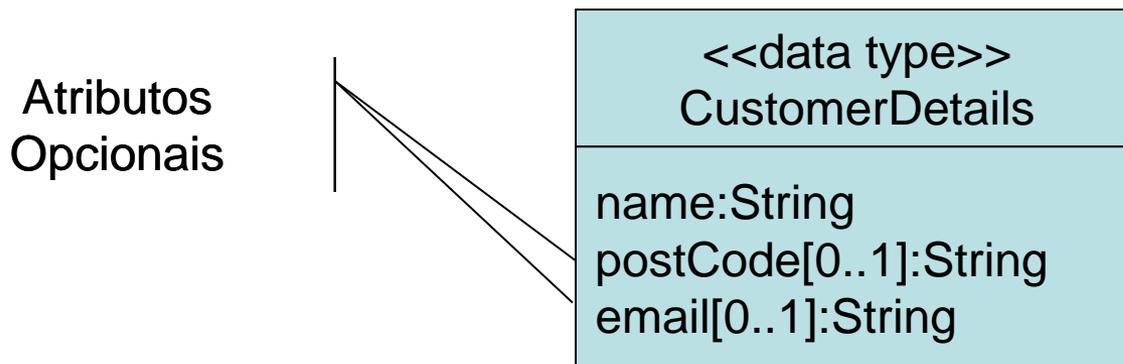
- A operação:
 - Cria a reserva e avisa o cliente por e-mail
 - Recebe como argumento os detalhes da reserva e os detalhes do cliente;
 - Retorna a referência para uma nova reserva

```
IMakeReservation::makeReservation(in res:ReservationDetails,  
                                  in cus: CustomerDetails,  
                                  out resRef: String): Integer
```

Código de
Retorno da
operação

Exemplo: *makeReservation()* (*Cont.*)

- Se for um novo cliente, os três atributos são requeridos (**name**, **postCode**, **email**)
- Se for um cliente existente, o postCode é requerido apenas se **name** não for único.



Código de Retorno da Operação *makeReservation()*

```
IMakeReservation::makeReservation(in res:ReservationDetails,  
                                  in cus: CustomerDetails,  
                                  out resRef: String): Integer
```

- É usado para retornar a situação da chamada e execução da operação;
- Os valores são:
 - 0: Sucesso;
 - 1: É um novo usuário e os atributos postCode e email não foram fornecidos;
 - 2: Nome não é único e o atributo postCode não foi fornecido;

Uma nova Operação

- O sistema referencia internamente um cliente por um código, custId;
- Esse código, diferentemente do hotelId, foi mantido como um conceito interno;
- Uma operação na interface ICustomerMgt deve encontrar o custId a partir de CustomerDetails;

```
ICustomerMgt::getCustomerMatching(in custD: CustomerDetails,  
                                   out cusId: CustId): Integer
```

Código de
Retorno da
operação

Código de Retorno da Operação *getCustomerMatching()*

```
ICustomerMgt::getCustomerMatching(in custD: CustomerDetails,  
                                   out cusId: CustId): Integer
```

- Os valores são:
 - 0: Sucesso;
 - 1: Cliente não existente;
 - 2: Nome não é único e o atributo postCode não foi fornecido;

Outra Operação

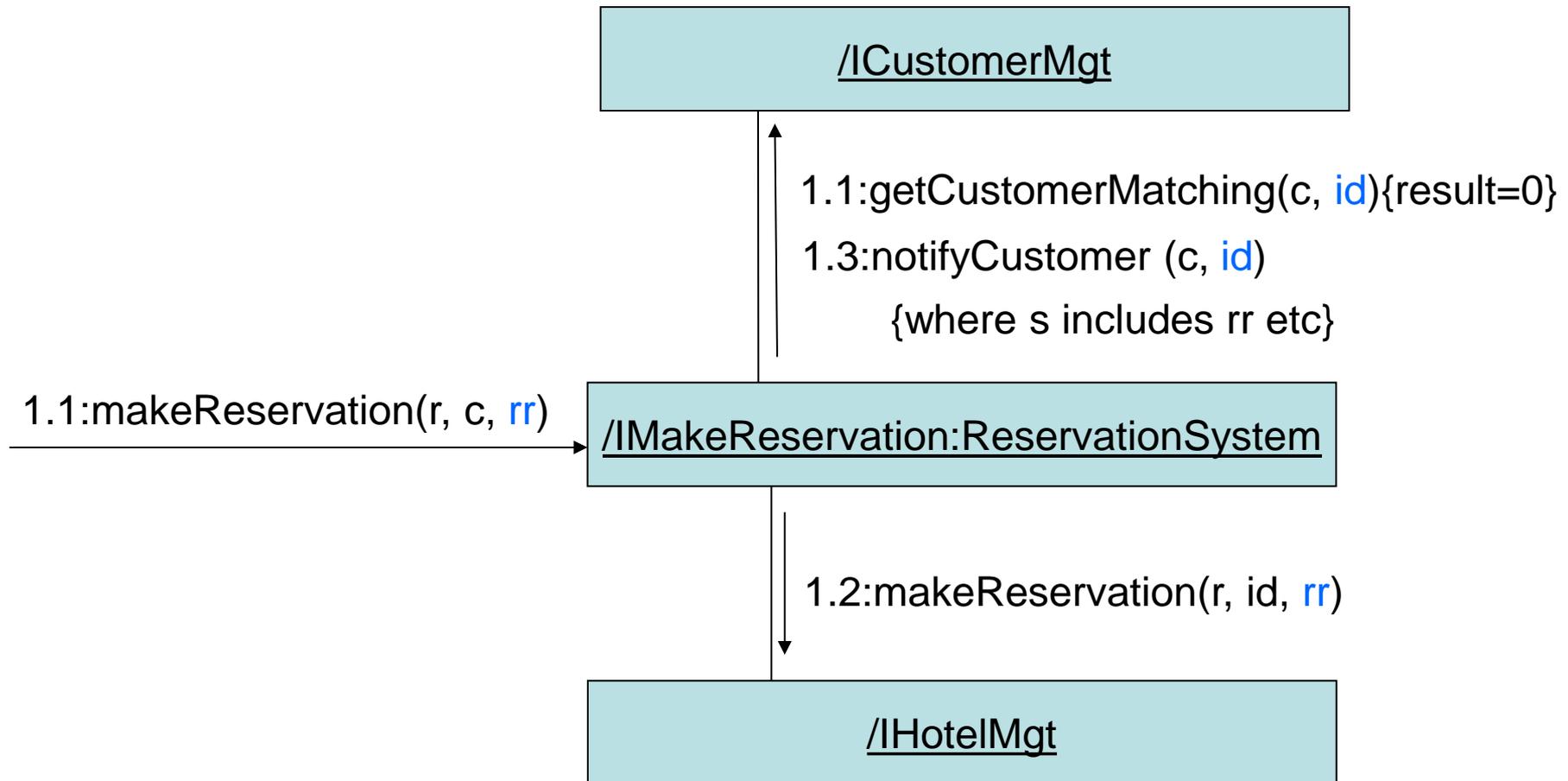
- notifyCustomer tem como entrada a identificação do cliente e a mensagem a ser enviada.

```
ICustomerMgt::notifyCustomer( in cus: CustId,  
                               in: msg:String)
```

Quebra de Dependência dos Componentes

- O componente HotelMgr é responsável por armazenar a associação entre reservas e clientes;
- O componente CustomerMgr é independente do componente HotelMgr;
- O componente ReservationSystem não pode delegar a operação a HotelMgt e deixar que este faça a chamada à operação getCustomerMatching();

Quebra de Dependência dos Componentes

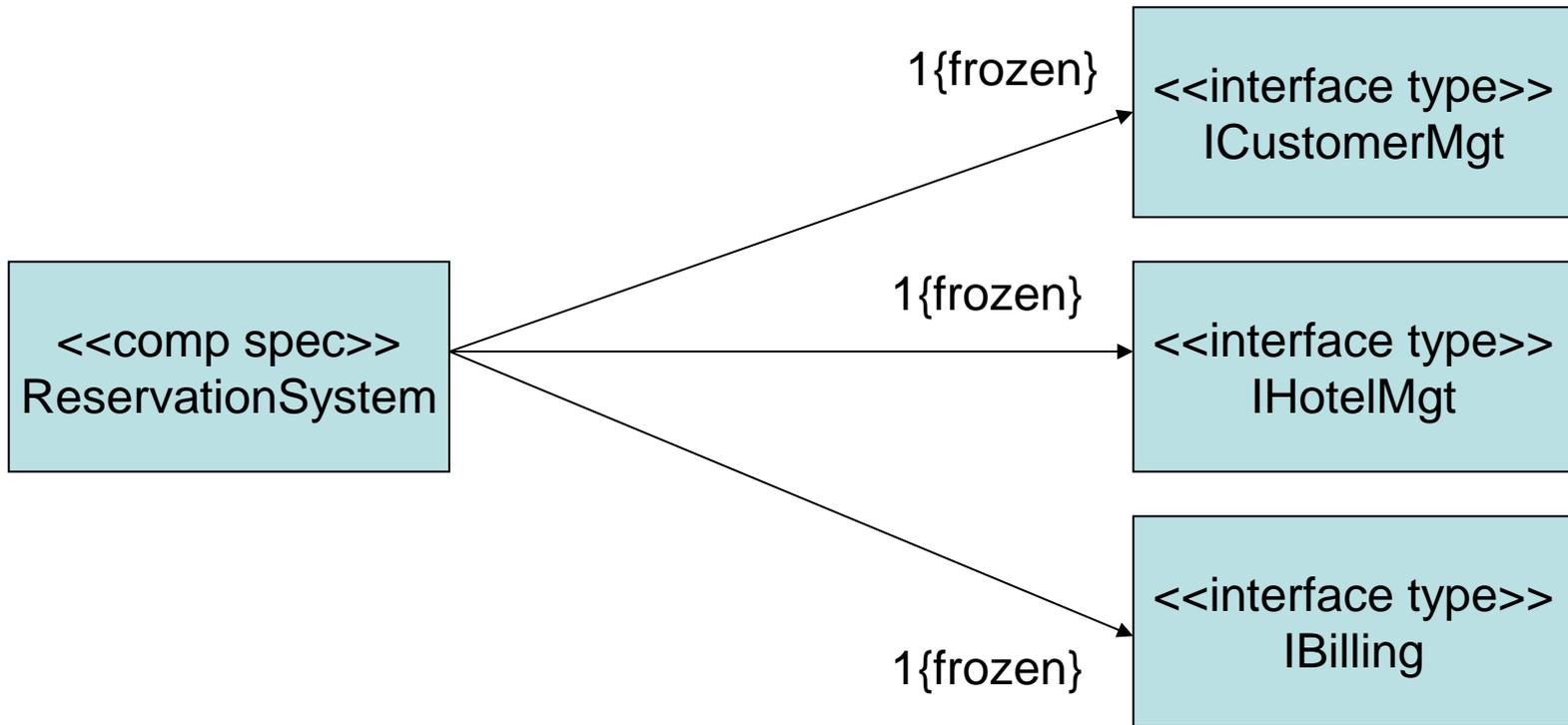


Interações de *makeReservation()* (cliente existente)

Manter a Integridade Referencial

Arquitetura dos objetos componentes

- A arquitetura da especificação de componentes mostra apenas quais componentes existem mas não quantos de cada tipo em tempo de execução;
- É necessário mostrar claramente que ReservationSystem usa sempre os mesmos objetos componentes de negócio;



Controlar as Referências Intercomponentes

- Um problema que surge da integridade referencial intercomponentes é: como garantir a validade dos identificadores?
- HotelMgr mantém identificadores de Customer:
 - Como assegurar que essas referências são válidas?
 - O que acontece se removermos (delete) um cliente?

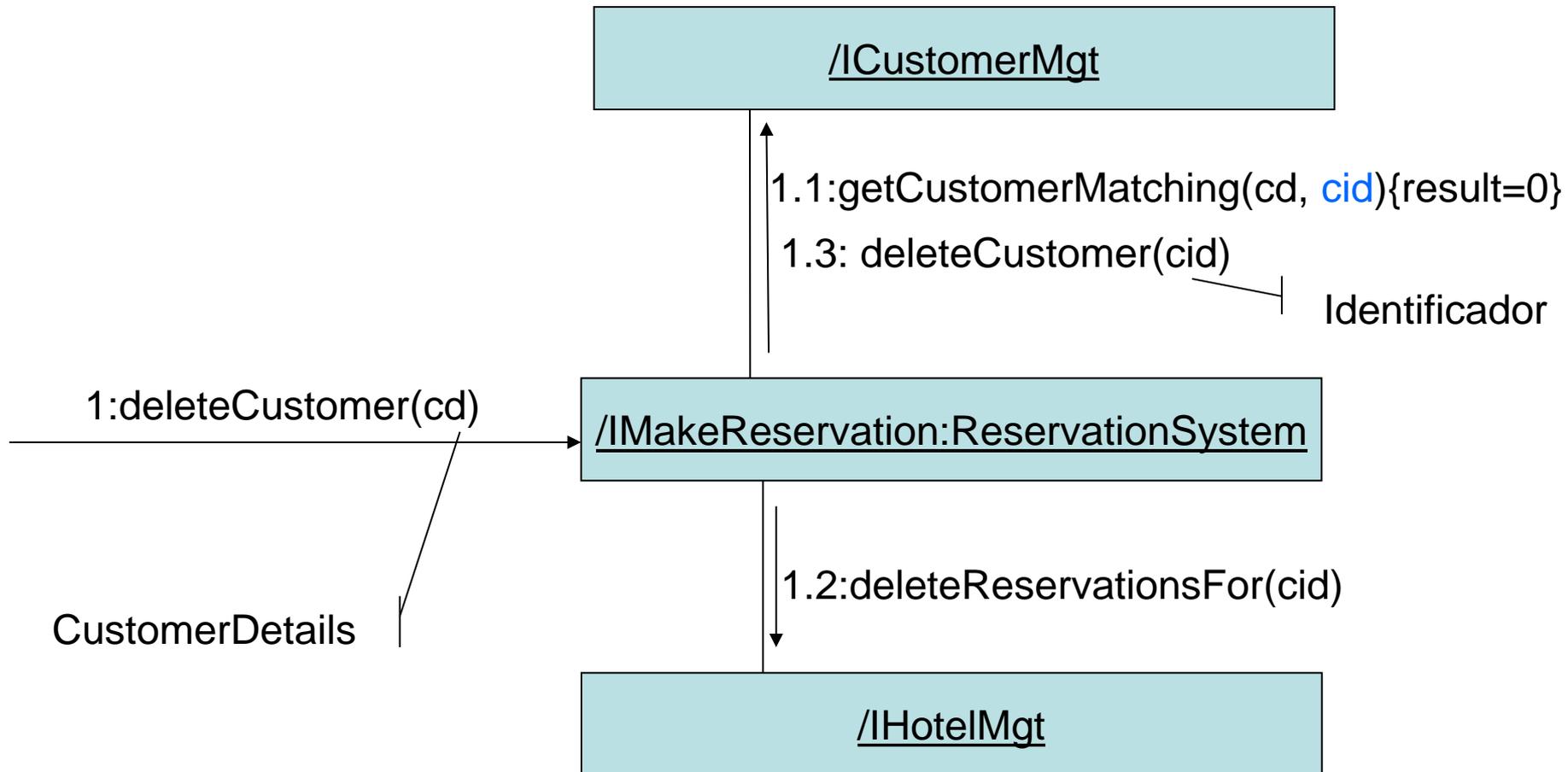
Controlar as Referências Intercomponentes

- Algumas das possíveis soluções:
 1. A responsabilidade é do componente que armazena a referência;
 2. A responsabilidade é do componente alvo da referência;
 3. A responsabilidade é de um terceiro componente que faz a mediação;
 4. Permitir e tolerar referencias inválidas;
 5. Não permitir a exclusão de informações;

Análise

- A opção 1 é descartada, porque isso obrigaria que HotelMgr se comunicasse diretamente com CustomerMgr
- Como queremos referências sempre válidas, a opção 4 também é descartada.
- Vamos supor que pelas funcionalidades descritas nos casos de uso, a alternativa 5 não é possível.

Controlar as Referências Intercomponentes (opção 3)



Interações *deleteCustomer()* (mantendo a integridade referencial)

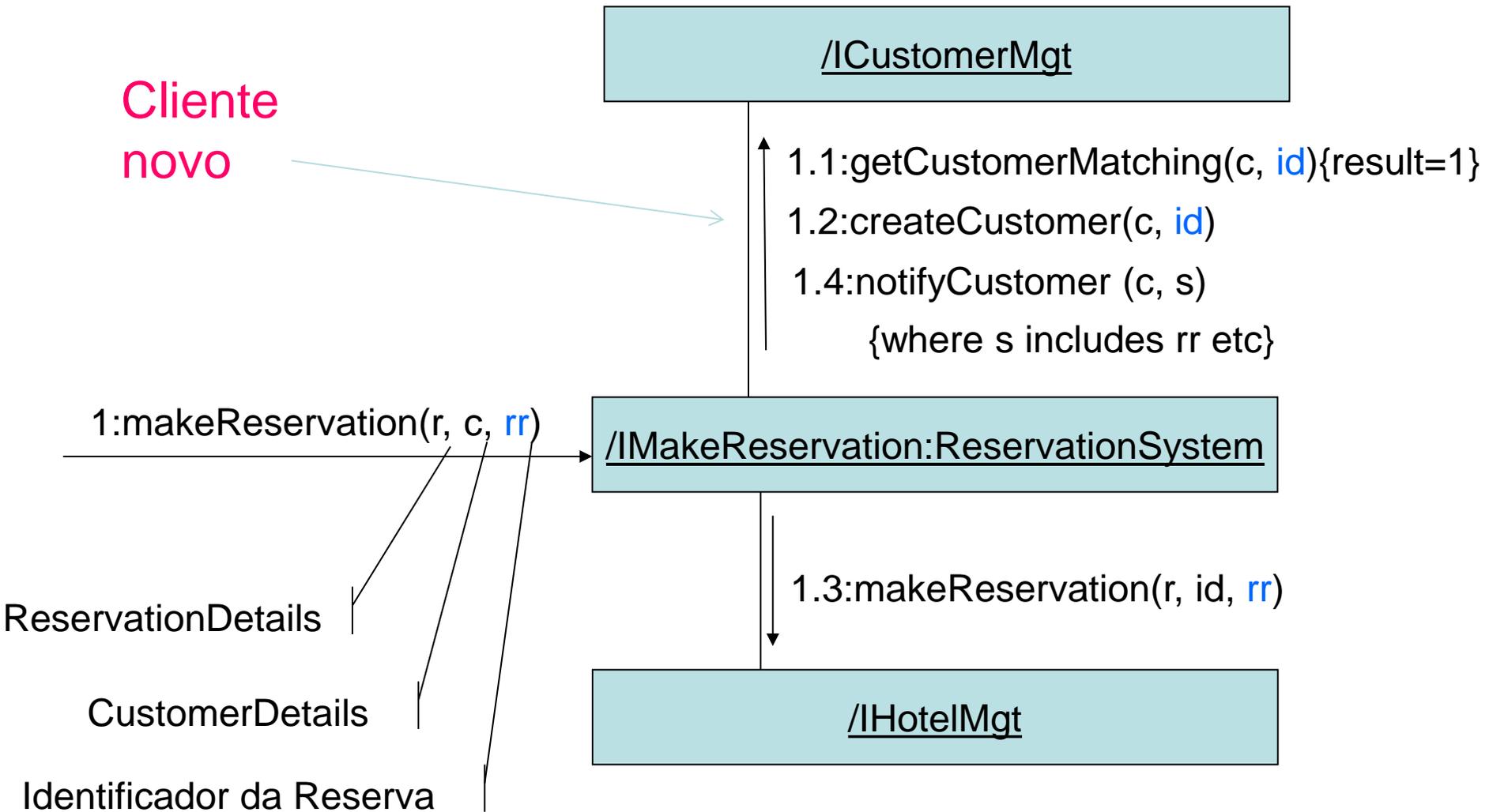
Análise (cont.)

- A opção 3 é mais simples, mas tem um problema: ela assume que o componente CustomerMgr é exclusivo do sistema de reserva. Isto é, assume que deleteCustomer() só é chamada pelo sistema de reservas.
- Isso torna o componente menos reusável.

Análise (cont.)

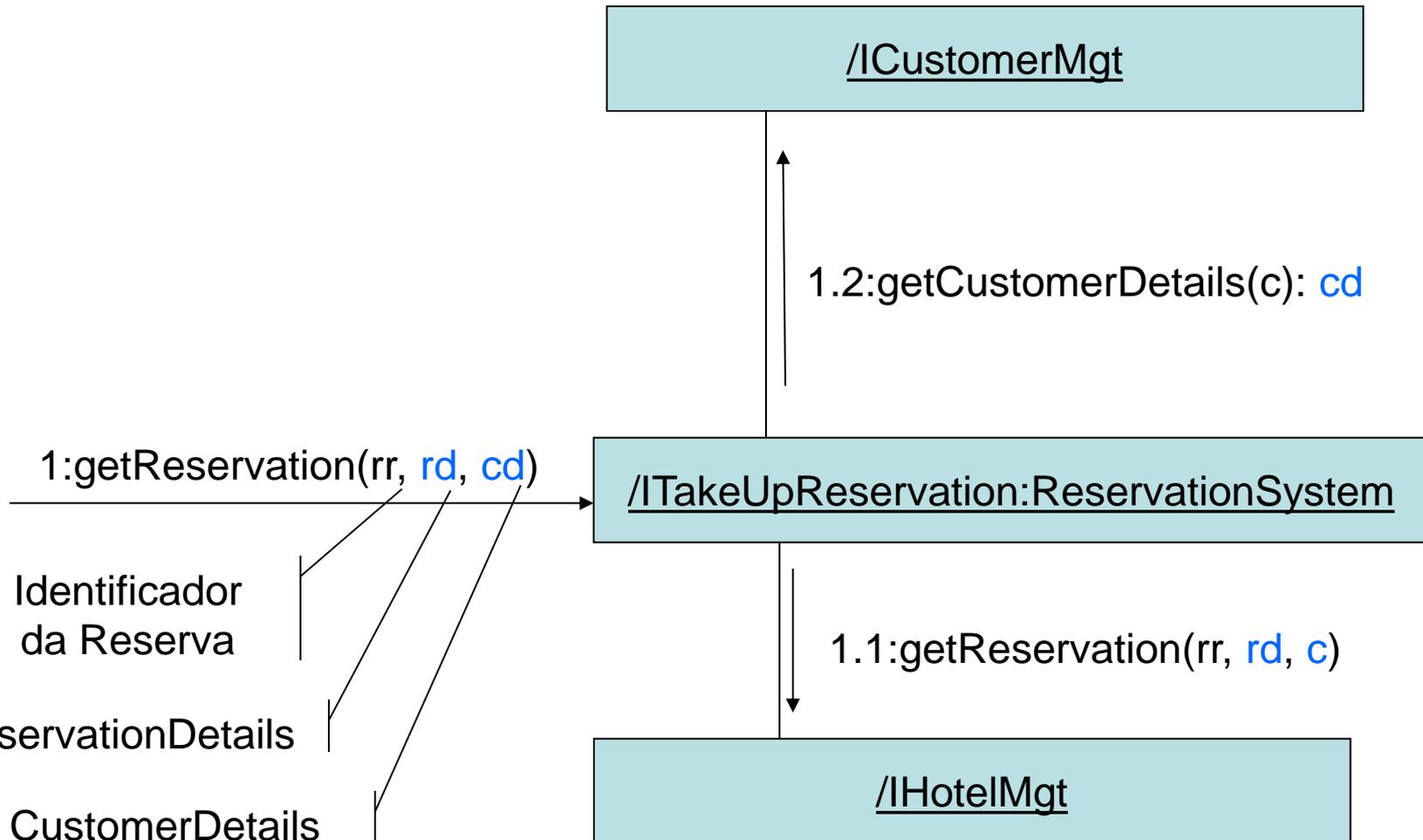
- Se não é possível assumir acesso exclusivo a CustomerMgt, então devemos usar a opção 2.
- Esta solução é mais complexa:
 - Todo objeto componente que tem uma referência para Customer deve notificar CustomerMgr.
 - CustomerMgr terá que manter uma lista de todas as depêndencias para notificar os outros componentes quando remover um Customer → Observer

Completando a operação makeReservation



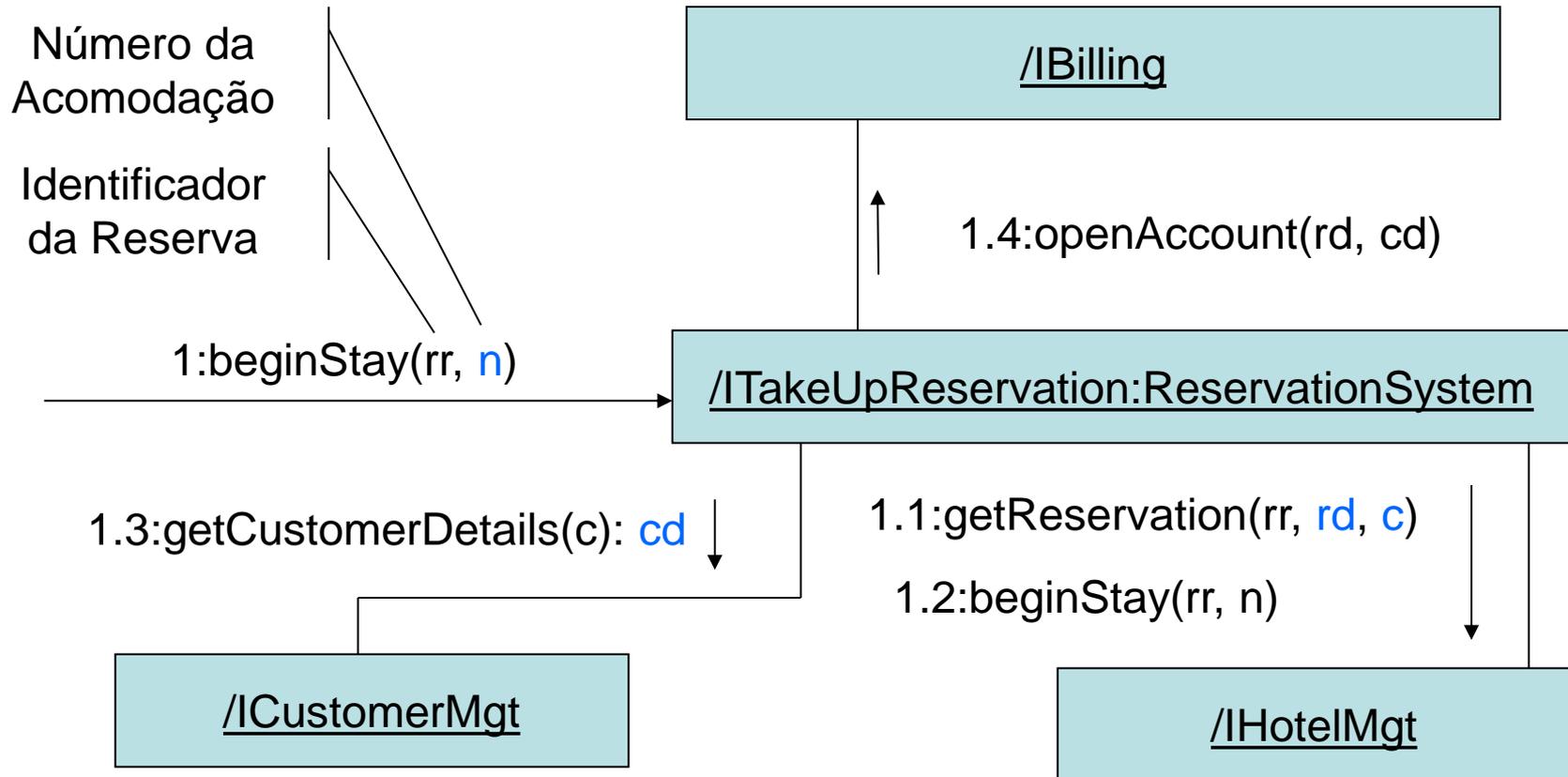
Interações de *makeReservation()* (novo cliente)

Outras Operações



Interações de `getReservation()`

Demais Operações



Interações de *beginStay()*

Interfaces do Sistema

<<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>>
ITakeUpReservation

getReservation(in resRef:String, out rd:ReservationDetails, out cus:CustomerDetails):Boolean
beginStay(in resRef:String, out roomNumber:String):Boolean

<<interface type>>
IBilling

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

Interfaces de Negócio

<<interface type>>
IHotelMgt

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
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
getCustomerDetails(in cus:CustId):CustomerDetails
notifyCustomer(in cus:custId, in msg:String)

Refinar as Operações e Interfaces

- Após a descoberta e definições de operações é necessário refinar as operações e interfaces;
- Pode-se fatorar as interfaces com generalizações ou divisão em mais interfaces;
- As operações também podem ser fatoradas para se tornar mais genéricas;

Refinar Interfaces (cont)

- Não se deve antecipar funcionalidades e requisitos;
- Muita da flexibilidade dos sistemas baseados em componentes se devem ao fato de que componentes podem oferecer múltiplas interfaces;
- Novos requisitos podem ser acomodados usando-se novas interfaces e gerenciando a dependência;

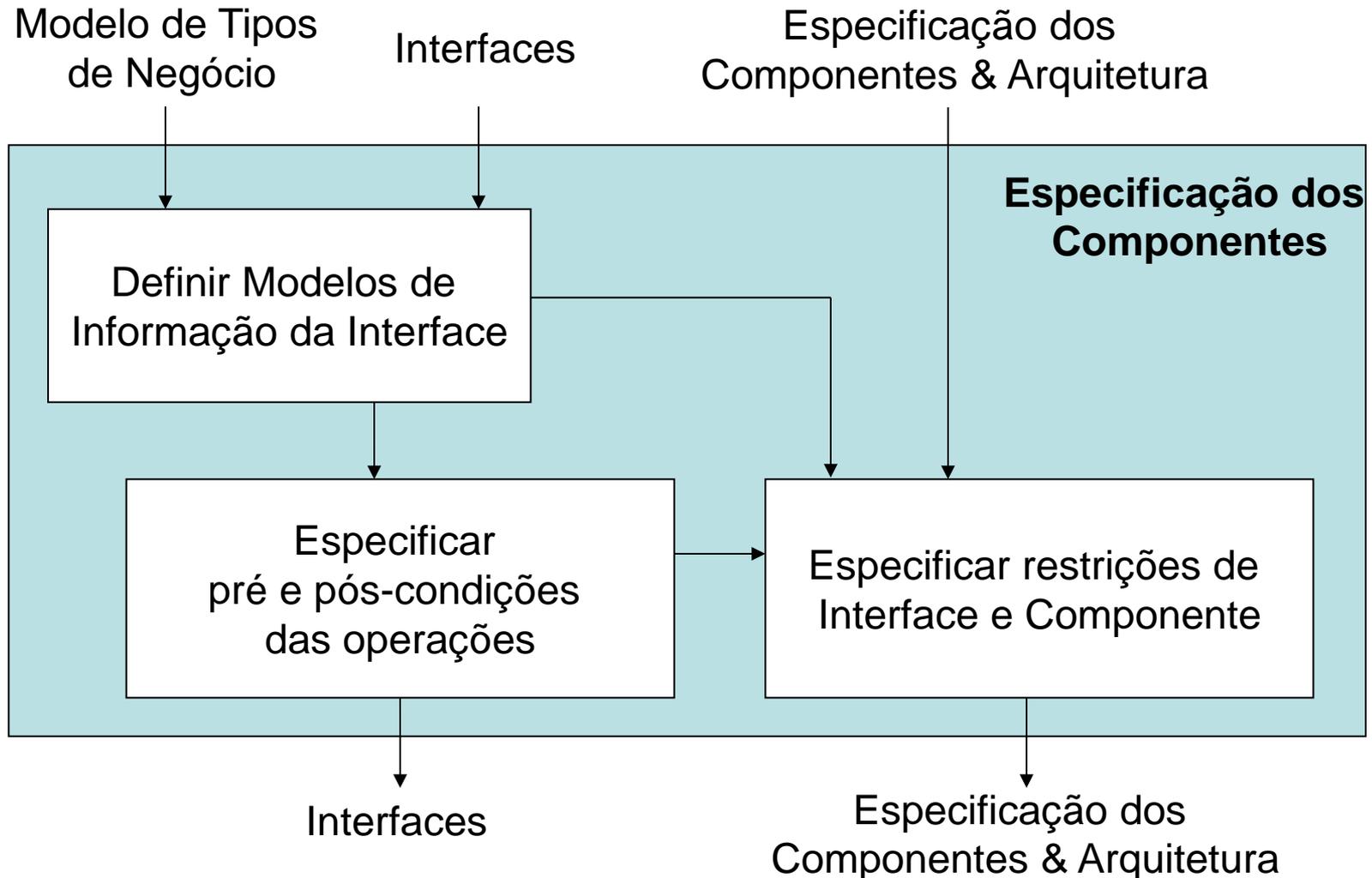
Especificação de Componentes

Cap. 7

Especificação de Componentes

- O objetivo é especificar os contratos de uso e os contratos de realização;
- O contrato de uso é definido pela especificação de interface;
- O contrato de realização é definido pela especificação de componente;

Especificação de Componentes



Especificação de Interfaces

- Uma interface é um conjunto de operações que definem, cada uma, serviços e funções do objeto componente;
- As interfaces definem como gerenciar as dependências e, portanto, são unidades de contrato dos componentes;
- A primeira tarefa é descrever o estado dos objetos componente;

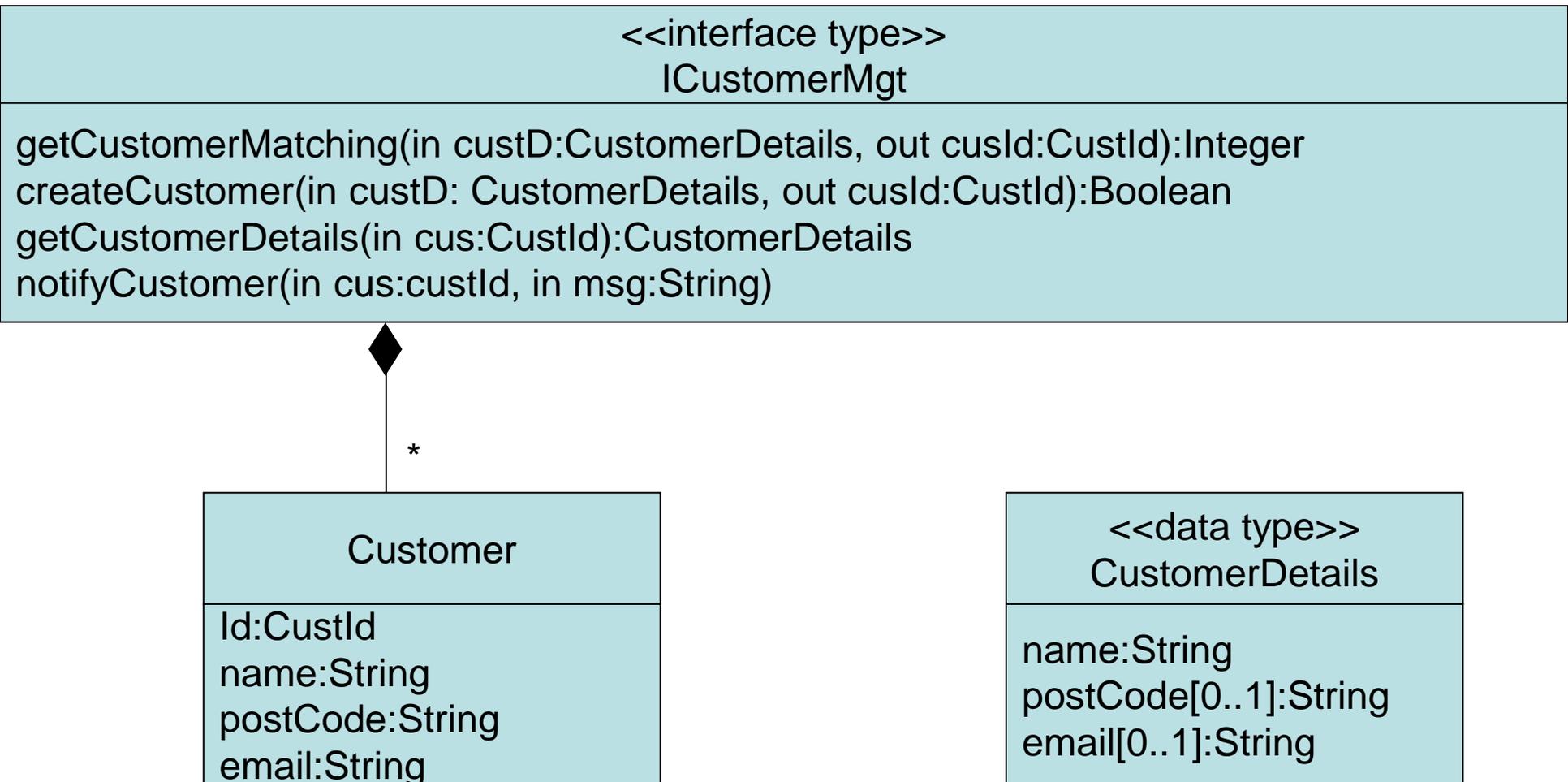
Modelos de Informação da Interface

- Cada interface deve possuir um modelo de informação que especifica as operações das interfaces;
- É um modelo dos possíveis estados de um objeto componente;
- As mudanças de estado dos objetos componente, causados pelas operações, podem ser descritas por esse modelo;

Modelos de Informação da Interface: Especificação das Operações

- Deve mostrar:
 - os parâmetros de entrada;
 - os parâmetros de saída;
 - os resultados da mudança de estado dos objetos componentes;
 - As restrições aplicáveis.

Exemplo de Especificação de Interface



Pré-condições e Pós-condições

- Cada operação tem uma pré-condição e uma pós-condição;
- A pós-condição especifica o efeito da operação se a pré-condição for verdadeira;
- A pré-condição não é uma condição para que uma operação seja invocada. Ela é uma condição de garantia para que a pós-condição seja verdadeira;

Pré-condições e Pós-condições (cont.)

- OCL é usada para especificar as pré- e pós-condições;
- Uma operação para alterar o nome do cliente:

```
context ICustomerMgt::  
    changeCustomerName(in cus:CustId,newName:String)  
pre:  
    -- Cus é um identificador de cliente valido  
    customer->exists(c | c.id = cus)  
  
post:  
    -- o nome do cliente de identificar cus é newName  
    customer->exists(c | c.id = cus and c.name = newName)
```

Exemplo: *getCustomerDetails()*

```
context ICustomerMgt::
    getCustomerDetails(in cus:CustId): CustomerDetails
pre:
    -- Cus é um identificador de cliente valido
    customer->exists(c | c.id = cus)

post:
    -- os detalhes retornados são iguais ao do cliente cujo
    -- identificador é cus
    -- encontra o cliente
    Let theCust = customer->select(c | c.id = cus) in
        -- especifica o resultado
        result.name = theCust.name and
        result.postCode = theCust.postCode and
        result.email = theCust.email
```

Regras para criar expressões OCL

- Podem se referir aos parâmetros, resultado das operações e ao estado do objeto componente (conforme definido na interface do modelo de informação da interface)
- Não podem se referir a mais nada
- Pré-condições podem se referir ao estado anterior (@pre) e posterior à execução da operação.

Processo para a construção de Modelos de Informação da Interface

- Pode-se derivar o Modelo de Informação de Interface a partir do Modelo de Tipos de Negócio;
- Os tipos que pertencem a duas interfaces devem aparecer em todos os modelos de informação de interface, mas são mostrados apenas os atributos relevantes.

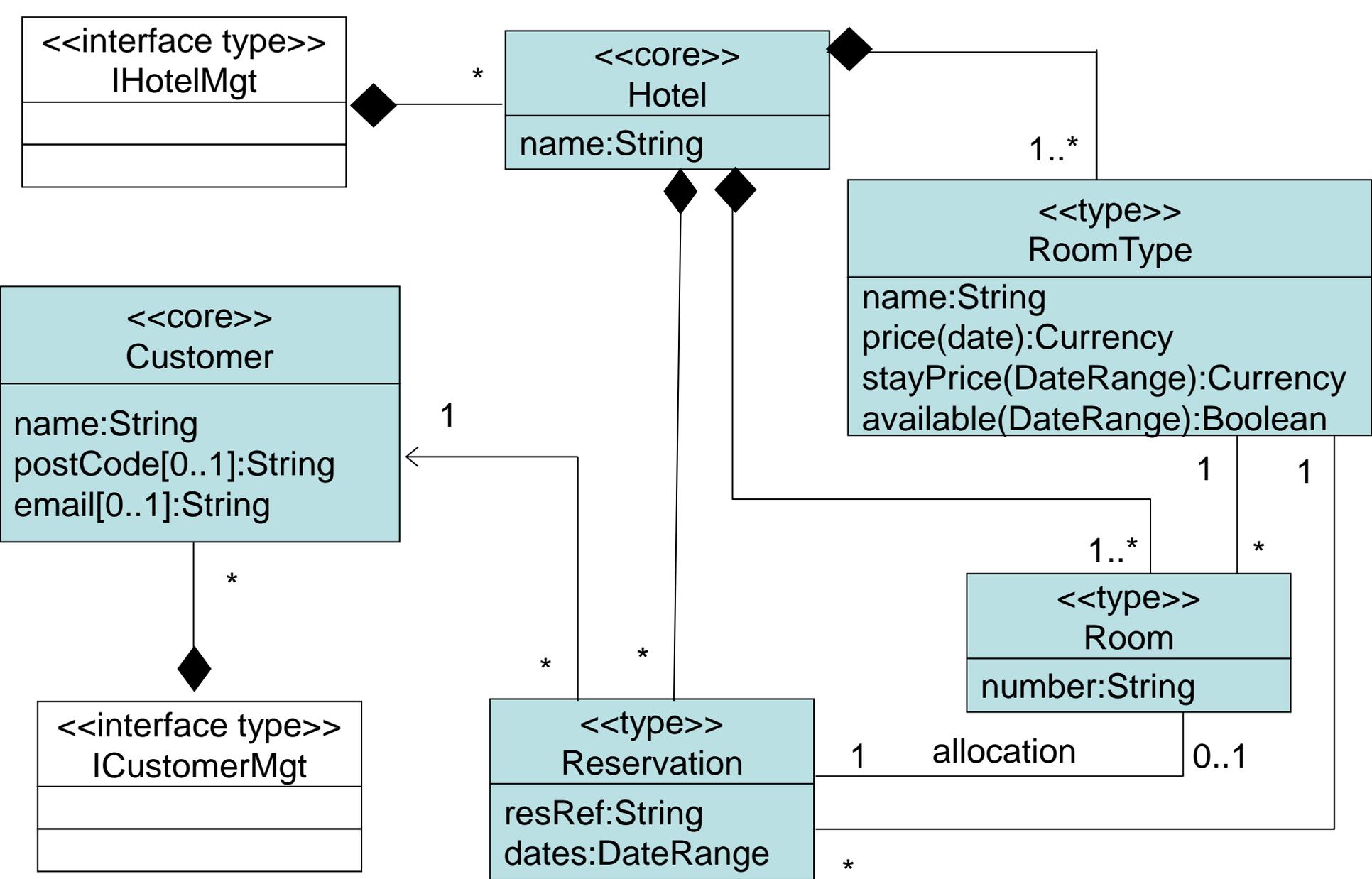
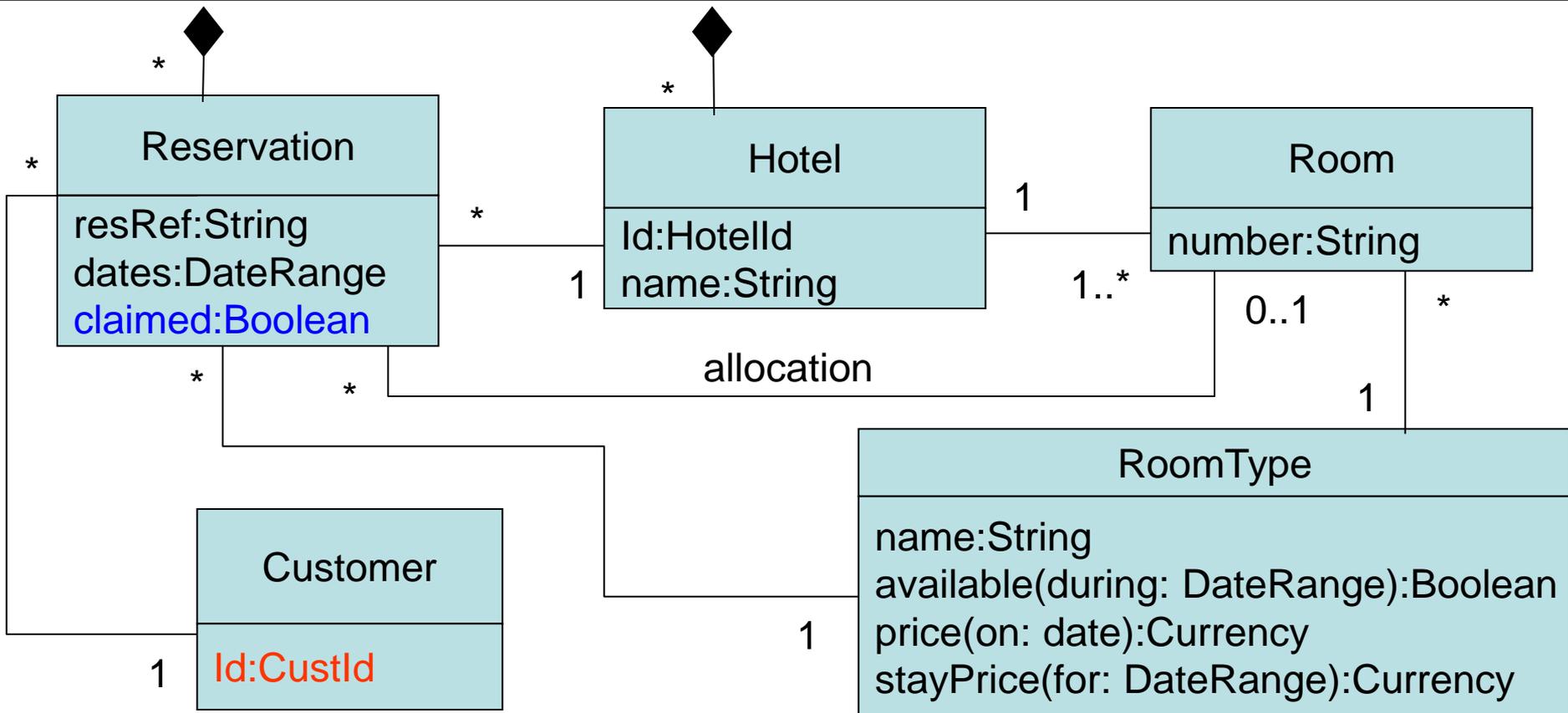


Diagrama de Responsabilidade das Interfaces

<<interface type>>
IHotelMgt

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
getReservation(in resRef:String, out rd:ReservationDetails, out cusId:CustId):Boolean
beginStay(resRef:String, out roomNumber:string:Boolean



Invariantes

- Um invariante é uma restrição ligada ao tipo que deve ser verdadeira para todas as instâncias do tipo;
- Ex: o seguinte invariante relaciona o atributo requerido de uma reserva com a associação entre Reservation e Room;

```
Context r: reservation inv:  
  -- a reserva é ocupada se existe uma alocação de  
  -- acomodação  
  r.claimed = r.allocation->notEmpty
```

Instantâneos

- IHotelMgt::makeReservation(
 in res: ReservationDetails,
 in cus: CustId,
 out cusRef: String):Boolean
- O que esperamos que aconteça se a operação for chamada com os seguintes valores:
 - Hotel Id =4, customer Id =92, room type = single ?
- Estes diagramas podem ajudar a definir as pré- e pós-condições.

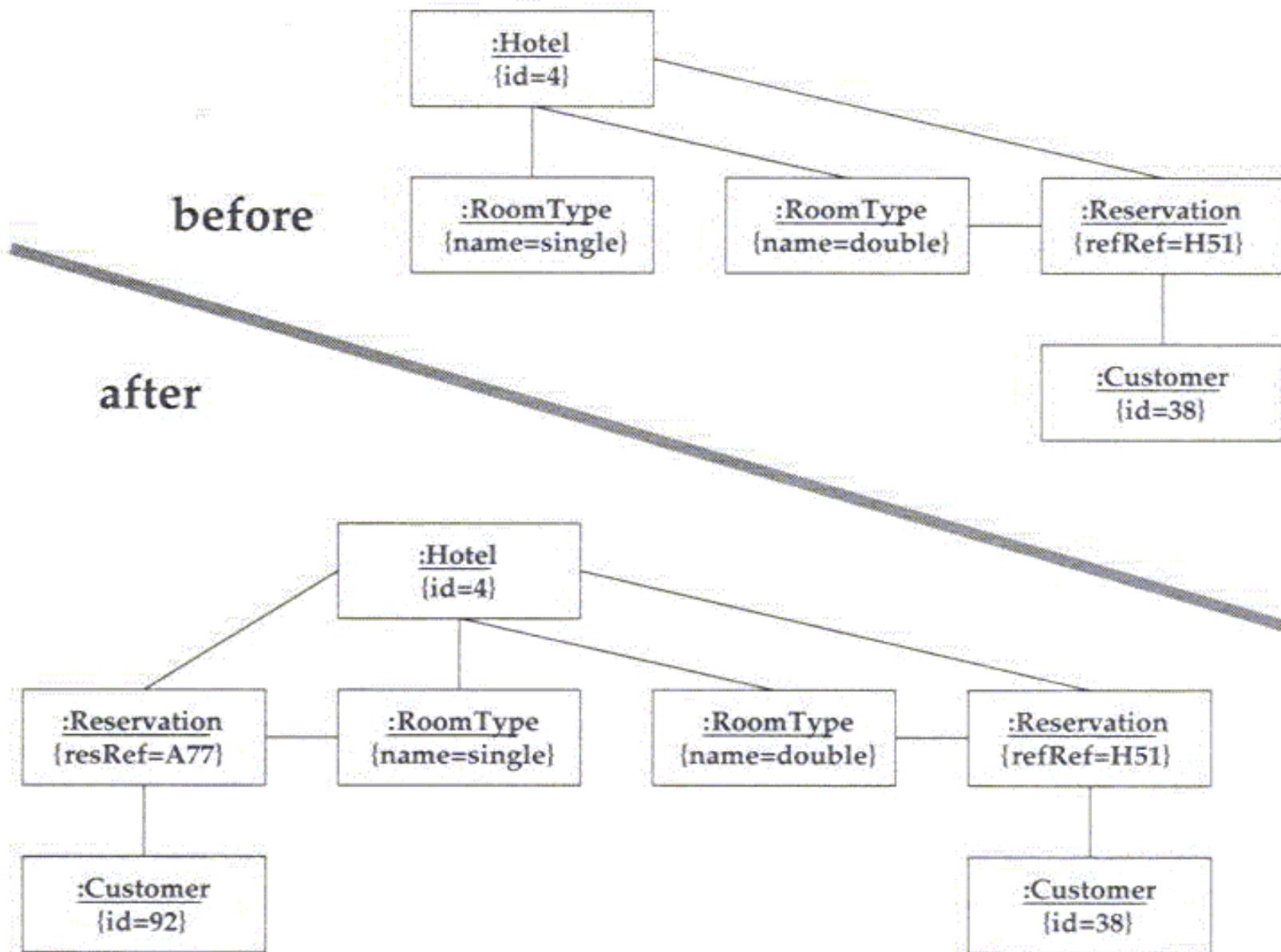


Figure 7.5 "Before" and "after" snapshot instance diagrams for IHotelMgt::makeReservation()

context IHotelMgt::

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

pre:

```
-- hotel e tipo de acomodações válidos  
hotel->exists(h | h.id = res.hotel and  
             h.room.roomType.name->includes(res.roomType))
```

post:

```
-- retornar verdadeiro para sucesso da operação  
-- o resultado implica em:  
-- a reserva foi criada  
-- identificar o hotel  
Let h = hotel->select(x |  
                    x.id=res.hotel)->asSequence->first in  
-- deve haver uma reserva a mais que anteriormente  
(h.reservation - h.reservation@pre)->size=1 and  
-- identificar a reserva  
Let r=(h.reservation-h.reservation@pre)->asSequence->first in  
-- a ref retornada é da nova reserva  
r.resRef=resRef and  
-- verificar a igualdade dos outros atributos  
r.dates=res.dateRange and  
r.roomType.name=res.roomType and not r.claimed and  
r.customer.id=cus
```

Especificação das Interfaces do Sistema

- As interfaces do sistema também devem ser especificadas;
- Os invariantes podem ser especificados tanto nas interfaces do sistema como nas interfaces de negócio;

<<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

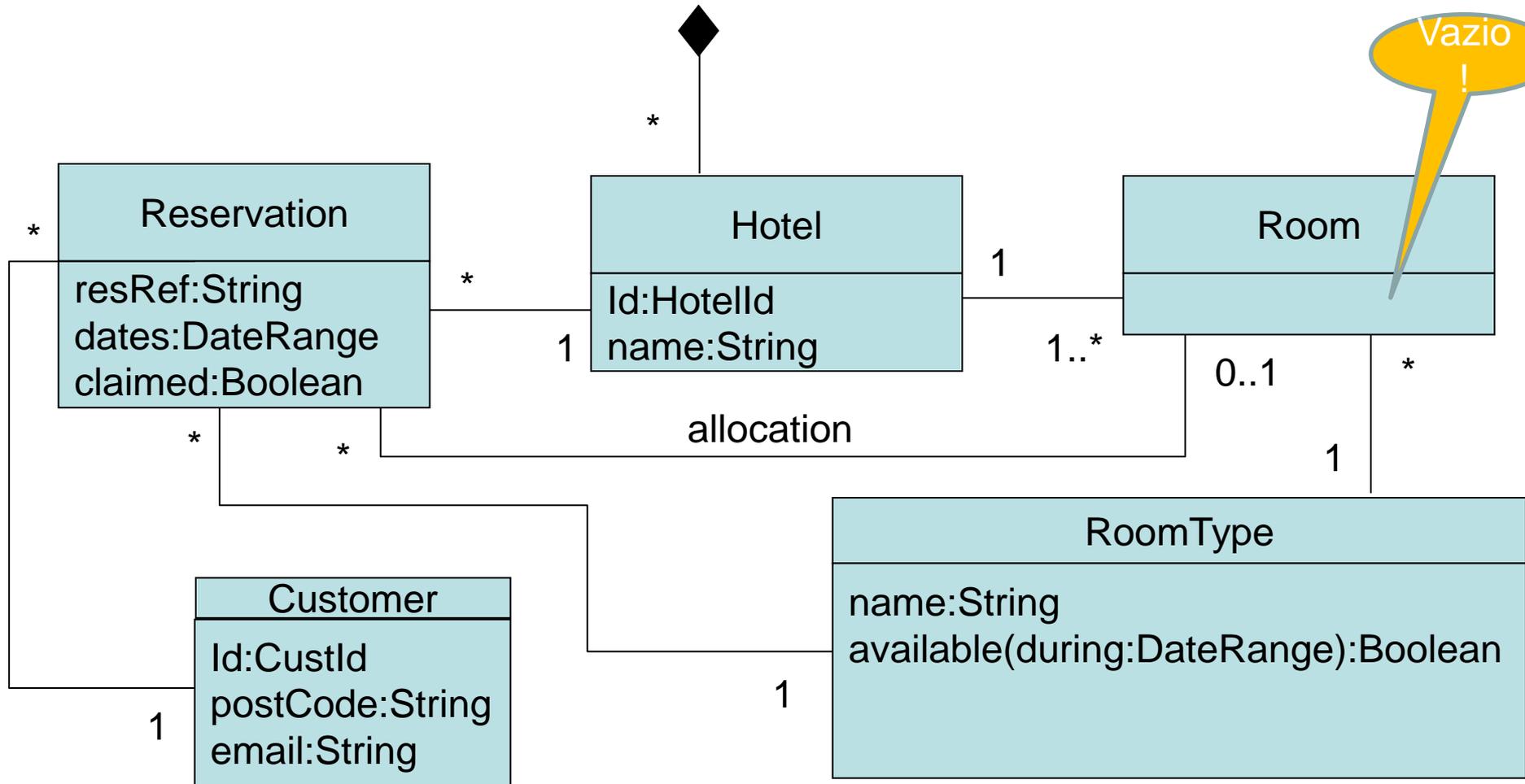


Diagrama de especificação de interface para IMakeReservation

<<interface type>>
ITakeUpReservation

getReservation(in resRef:String, out rd:ReservationDetails, out cus:CustomerDetails):Boolean
beginStay(in resRef:String, out roomNumber:String):Boolean

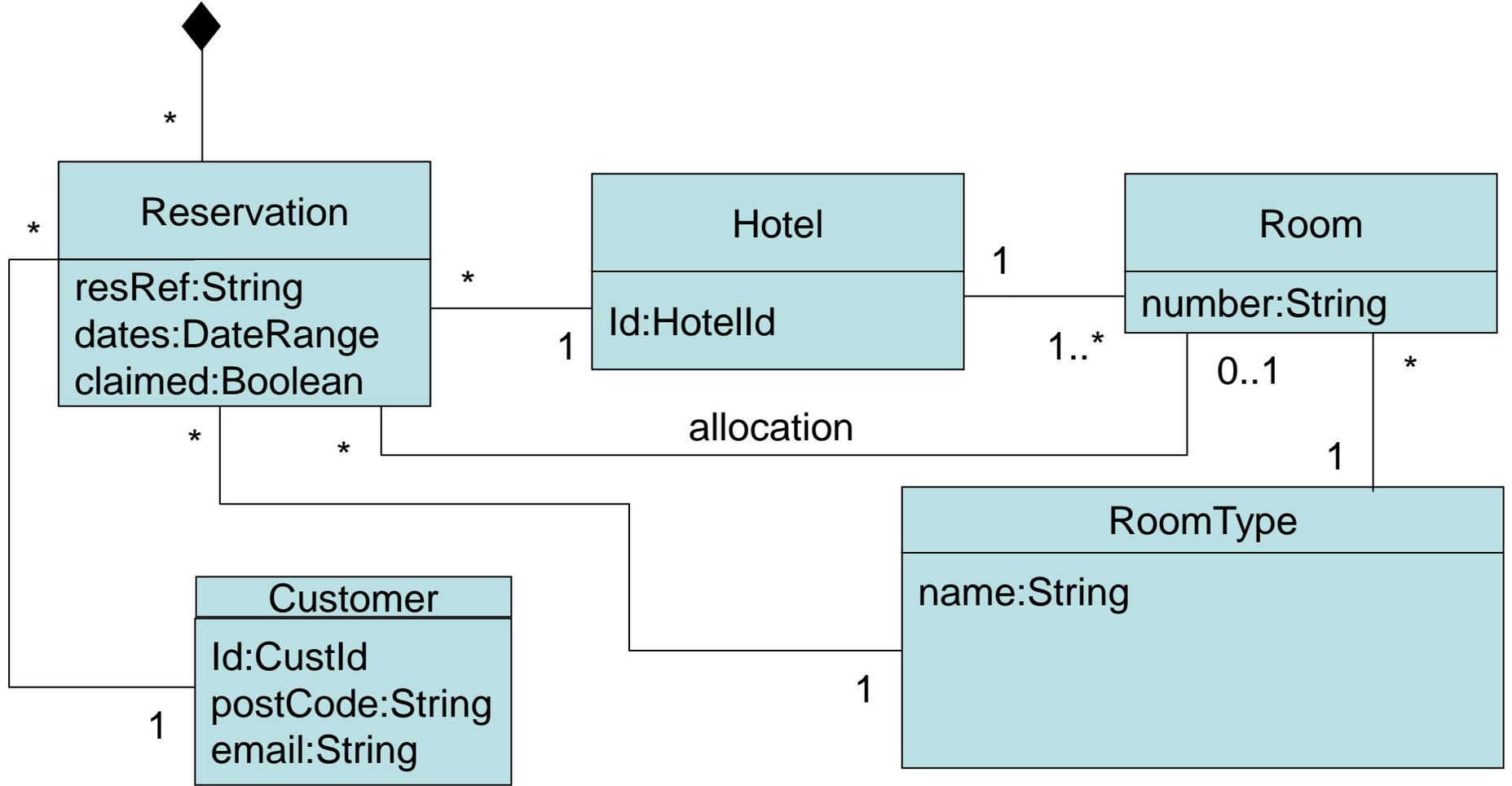


Diagrama de especificação de interface para ITakeUpReservation

Especificação dos Componentes

- Para cada especificação de componente é necessário estabelecer a quais interfaces sua realização dá suporte;
- Um diagrama de especificação de componentes mostra a dependência entre as interfaces do objeto componente e as interfaces oferecidas;

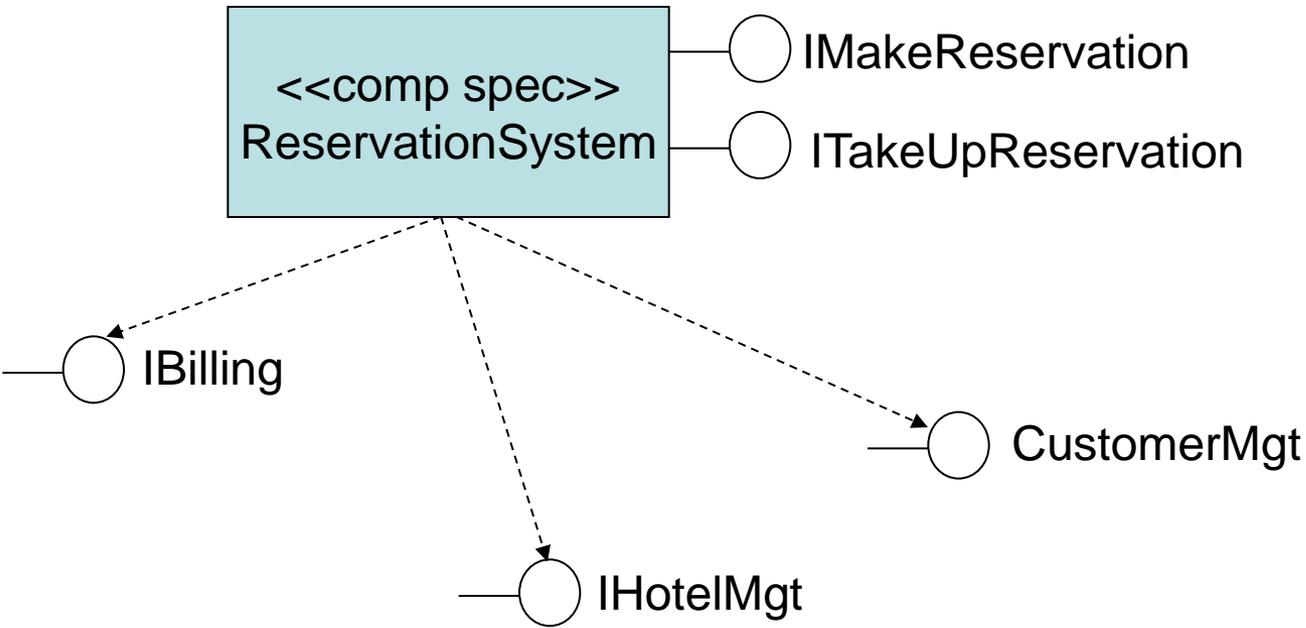


Diagrama de Especificação de Componentes para ReservationSystem

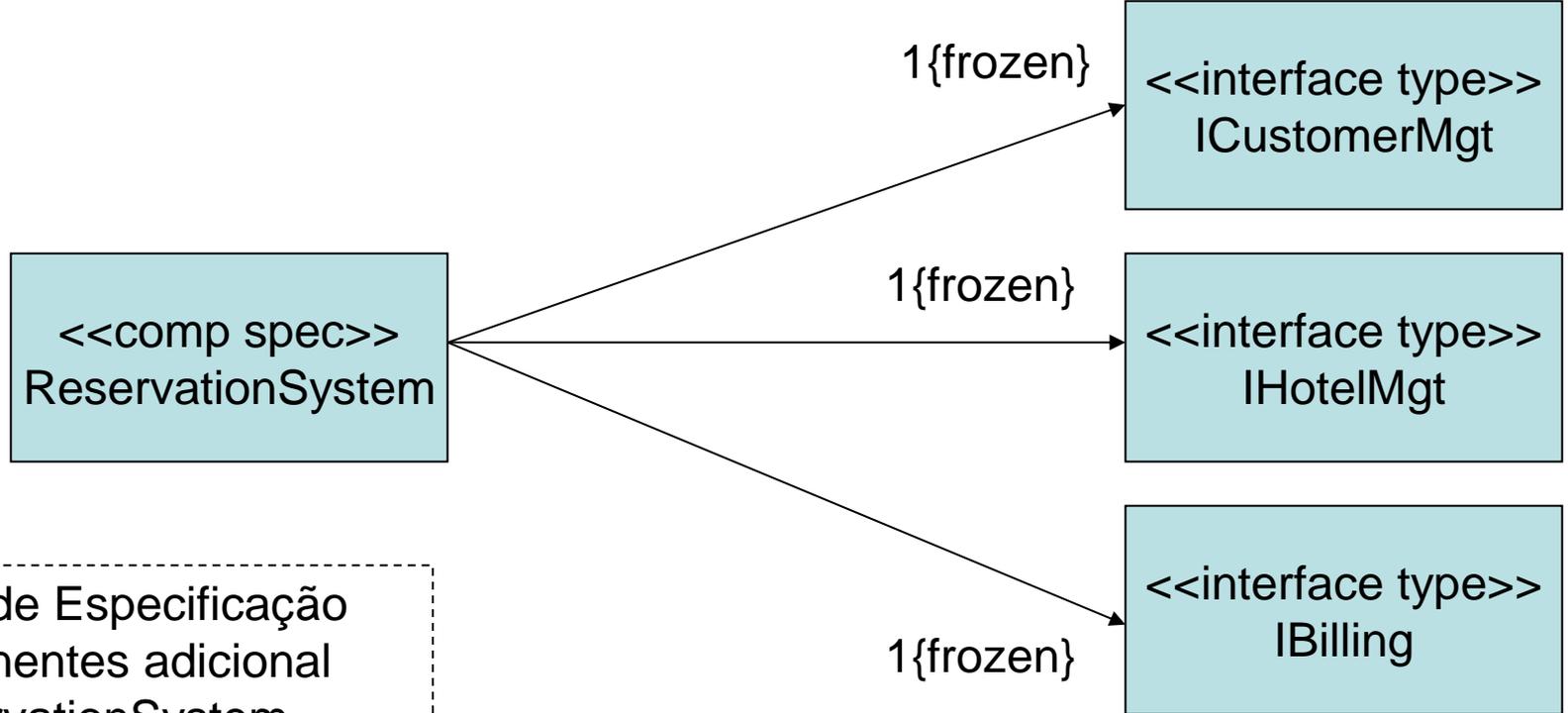


Diagrama de Especificação de Componentes adicional para ReservationSystem

Restrições Inter-Interface

- A análise das restrições inter-interface trata dos relacionamentos entre os modelos de informação de interface:
 - como as interfaces oferecidas se relacionam;
 - como as interfaces oferecidas se relacionam com as interfaces usadas;

Interfaces Oferecidas

- Diferentes interfaces de uma especificação de componente podem fazer referência a um mesmo tipo;
- Deve-se definir explicitamente quais tipos são usados por quais interfaces;

como as interfaces
oferecidas se relacionam

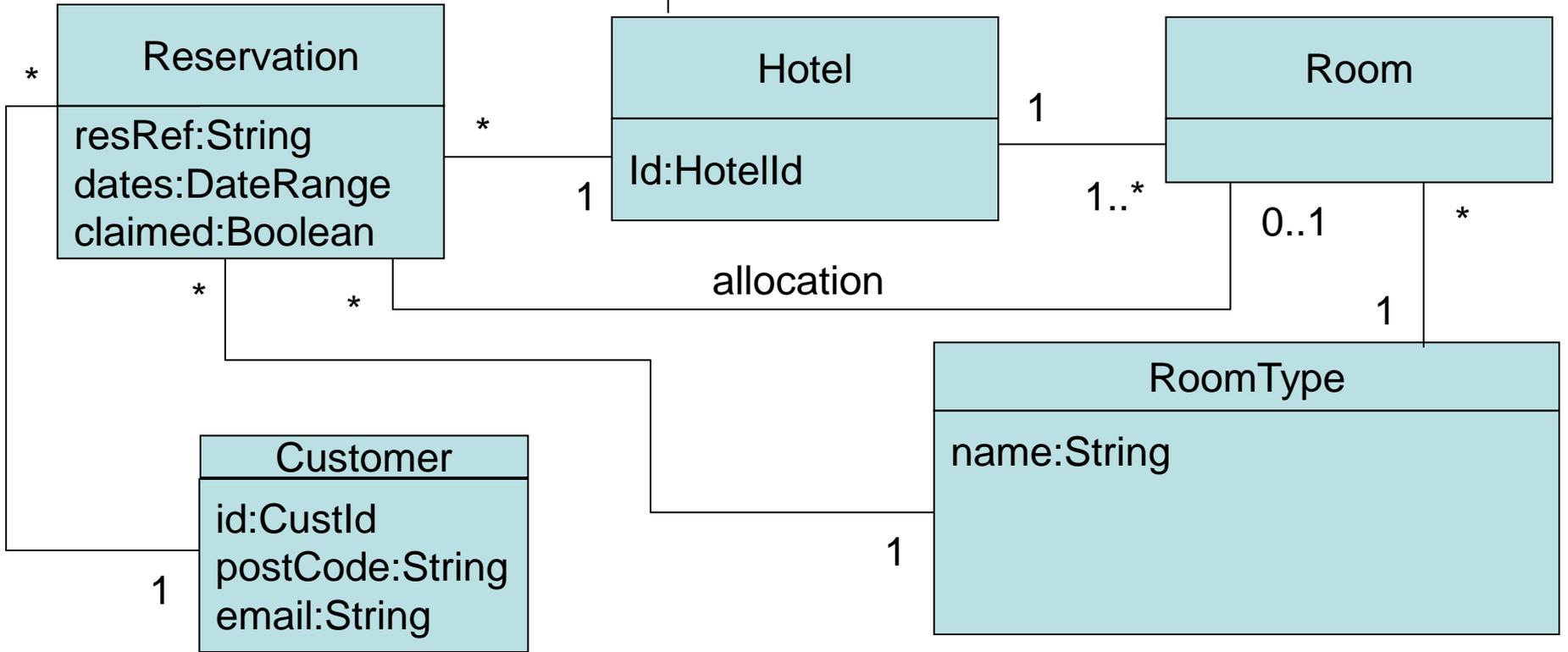
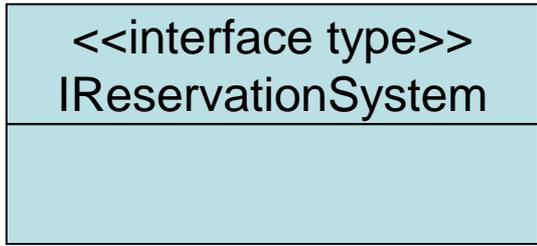
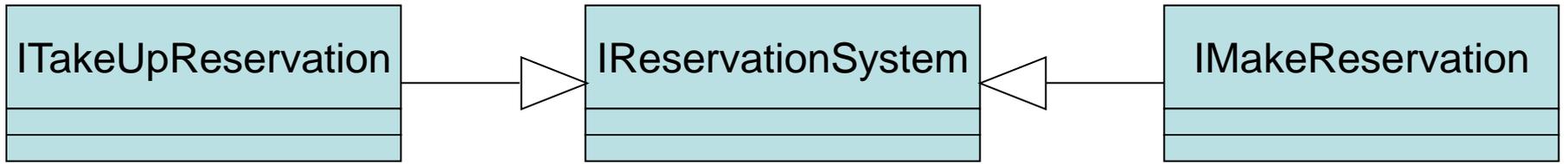
```
context ReservationSystem
-- restrições entre interfaces oferecidas
IMakeReservation::hotel = ITakeUpReservation::hotel
IMakeReservation::reservation = ITakeUpReservation::reservation
IMakeReservation::customer = ITakeUpReservation::customer
```

como as interfaces oferecidas
se relacionam com as usadas

```
context ReservationSystem
-- restrições entre interfaces usadas e interfaces oferecidas
IMakeReservation::hotel = IHotelMgt::hotel
IMakeReservation::reservation = IHotelMgt::reservation
IMakeReservation::customer = ICustomerMgt::customer
```

Fatorando Interfaces

- Para um sistema grande, criar todas as interfaces pode ser trabalhoso.
- As interfaces podem ser fatoradas em um supertipo.
- Em alguns casos pode ser prático juntar interfaces e não se preocupar com subtipos.



<<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

