Applied UML

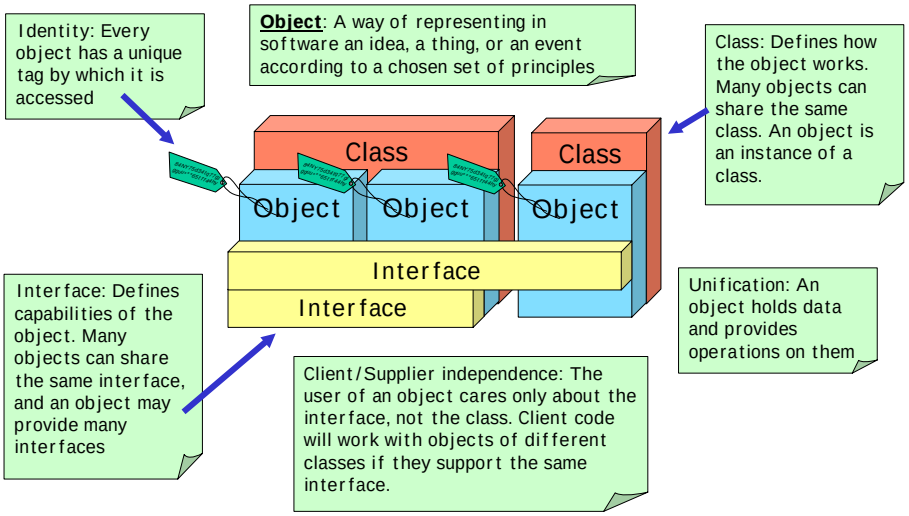
John Daniels
Syntropy Limited
john@syntropy.co.uk
www.syntropy.co.uk

What is UML?

Unified Modeling Language

- The UML is a standardised language for describing the structure and behaviour of things
- UML emerged from the world of object-oriented programming
- UML has a set of notations, mostly graphical
- There are tools that support some parts of the UML

Object principles



Applying UML

- Understand your architecture
- Understand your process
- Decide what you need to represent
- Choose the best UML notation to fit

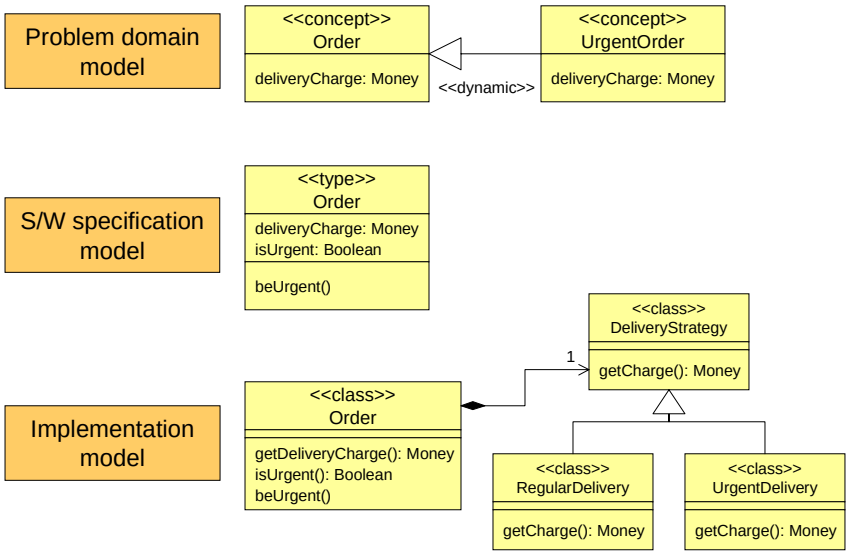
Model perspectives

- UML is a language for describing models
- What is the purpose of your model?
  - Models that describe the problem domain
    - nothing to do with software
  - Models that specify software
    - ranging from the whole system to one small part
  - Models that describe the implementation of software
- Some UML notations can be used for all three
- Understand which you are doing!

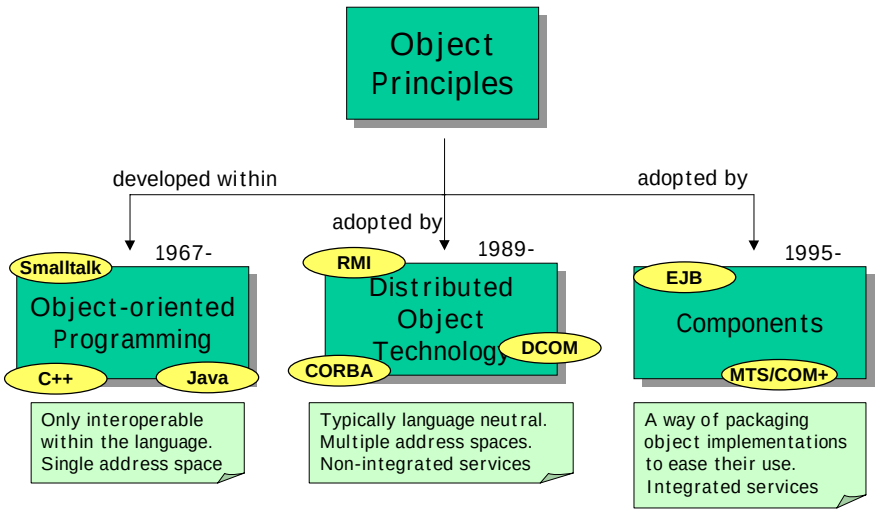
Usage of UML notations

	Problem domain	S/W spec	Implementation
Use case		boundary interactions	
Class diagram	information models	class/component structures	class/component structures
Seq/collab diagram		required object interactions	designed object interactions
Activity diagram	business processes		algorithms
Statechart		object lifecycles	object lifecycles

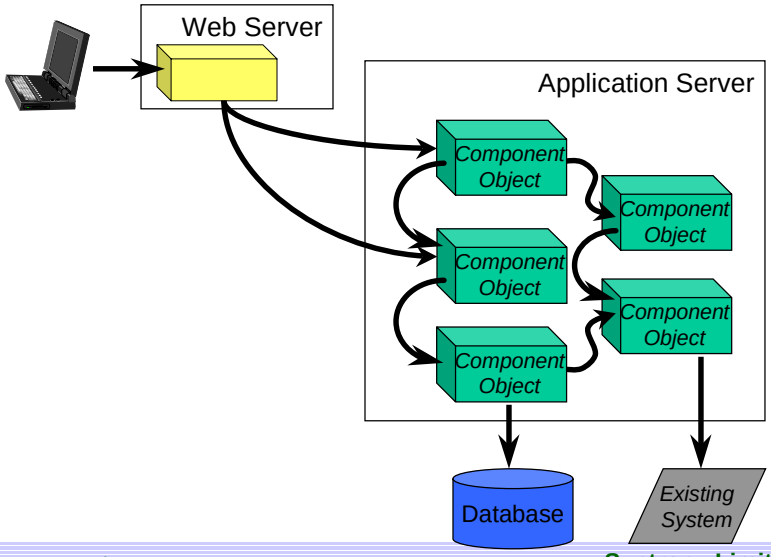
Same name, different purpose



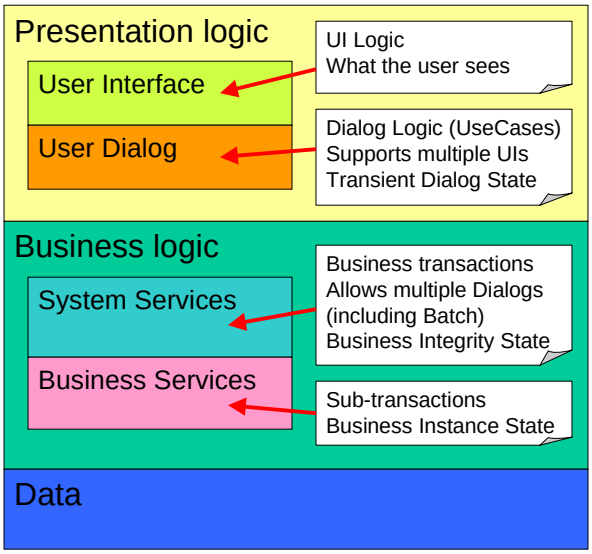
Components and objects



Architecture: Application Blueprint

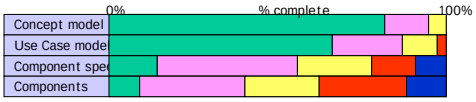


Architecture: Application Layers

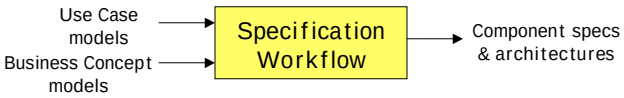


Process: Management vs. Development

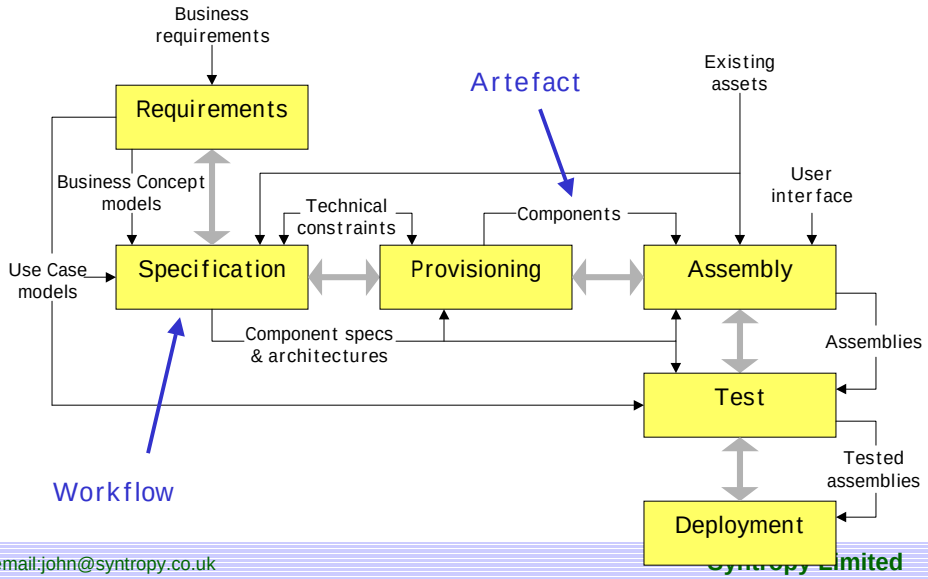
- Management Processes
  - Schedule work and plan deliveries
  - Allocate resources
  - Monitor progress
  - Control risk



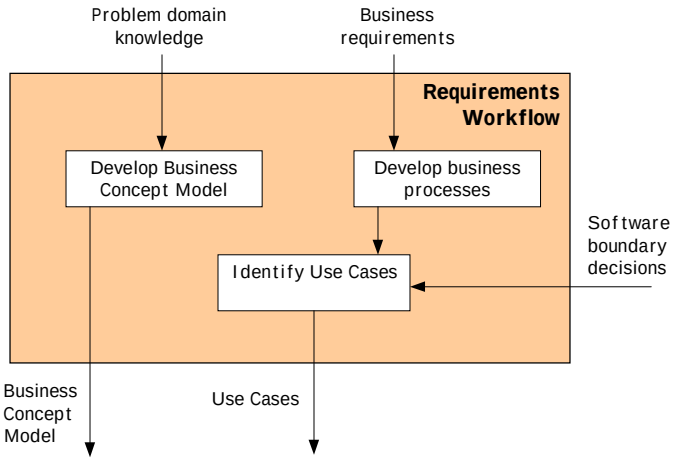
- Development Processes
  - Create working software from requirements
  - Focus on software development artifacts
  - Described independently of the management process
  - Defines ordering constraints and dependencies
  - Organized into Workflows



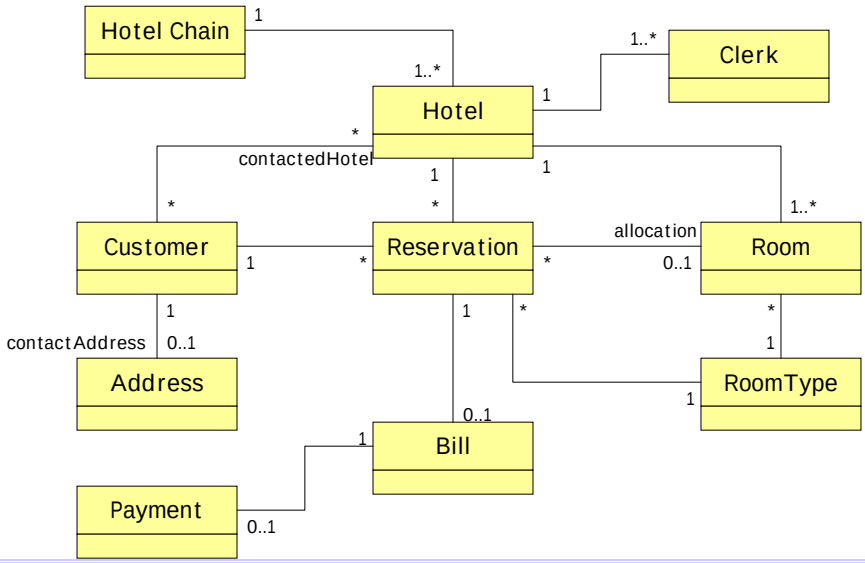
Process: Workflows in the development process



Process: The Requirements Workflow

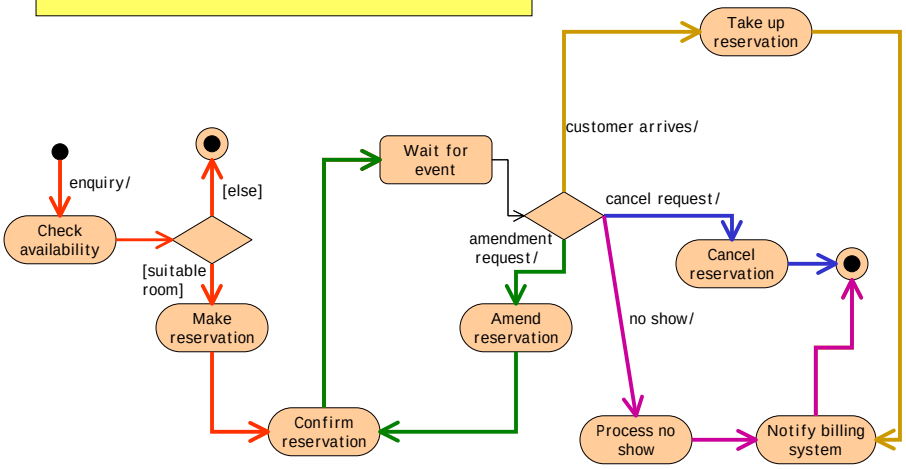


Business Concept Model



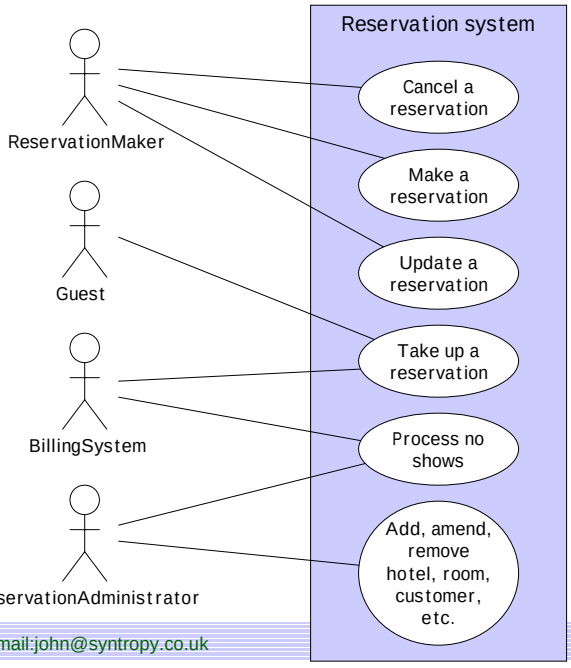
Identify Use Cases

A use case describes the interaction that follows from a single business event. Where an event triggers a number of process steps, all the steps form a single use case.



email:john@syntropy.co.uk

Use Case diagram



email:john@syntropy.co.uk

Name	Make a Reservation
Initiator	Reservation Maker
Goal	Reserve a room at a hotel

Steps

or
Extension
Points

Main success scenario

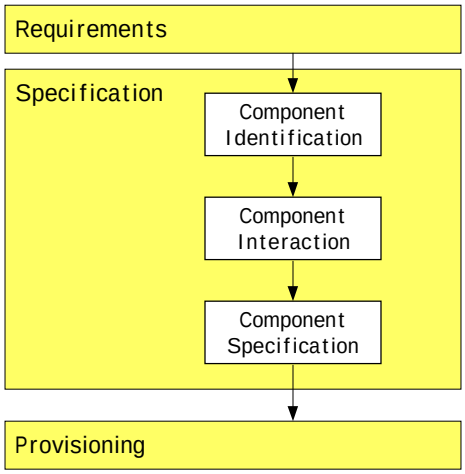
1. Reservation Maker asks to make a reservation
2. Reservation Maker selects hotel, dates and room type
3. System provides availability and price
4. Reservation Maker agrees to proceed
5. Reservation Maker provides name and postcode
6. Reservation Maker provides contact email address
7. System makes reservation and gives it a tag
8. System reveals tag to Reservation Maker
9. System creates and sends confirmation by email

Extensions

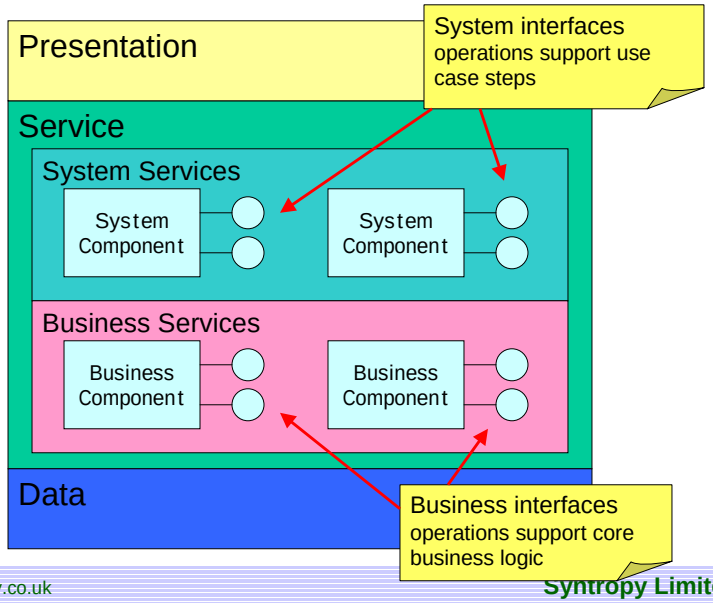
3. Room Not Available
   a) System offers alternative dates and room types
   b) Reservation Maker selects from alternatives

6. Customer already on file
   a) Resume 7

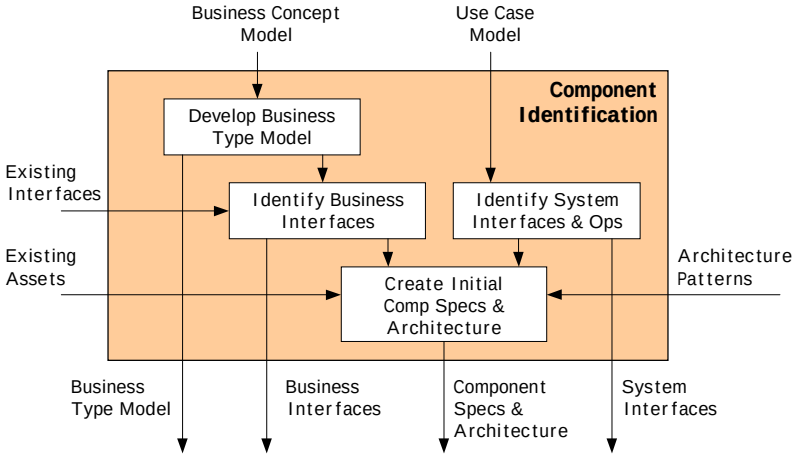
Process: The Specification Workflow



Components in the service layers



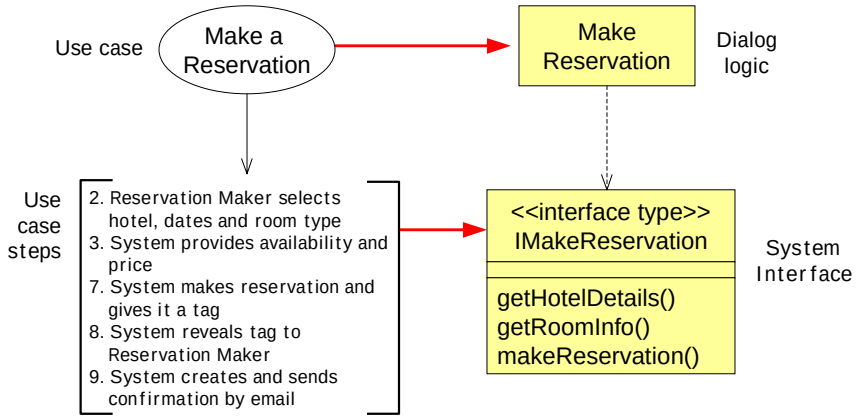
Process: Component Identification



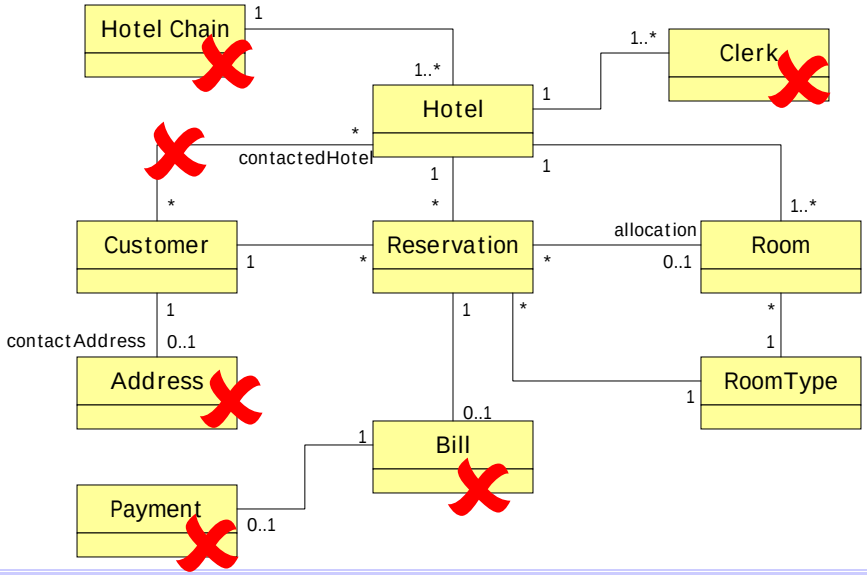
Identify System Interfaces and operations

System interfaces act as facades - they are the point of contact for the UI and other external agents. They are supported by components in the system services layer.

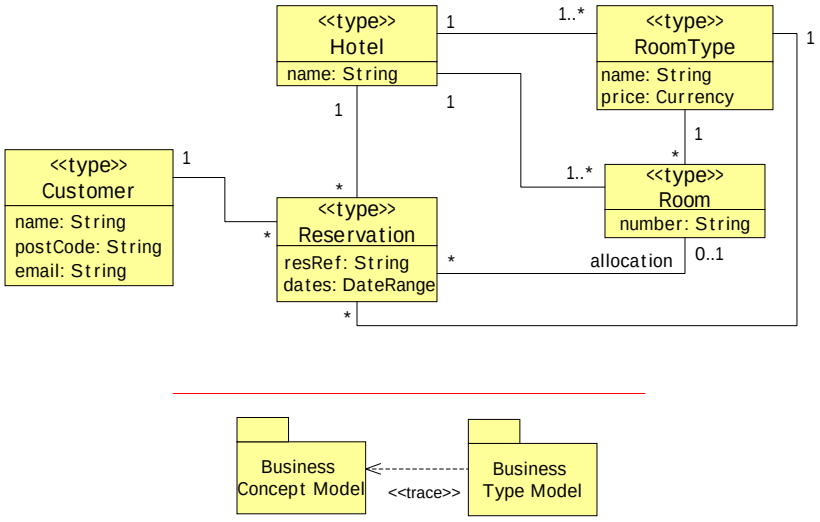
Start with one interface per use case, then refactor as necessary.



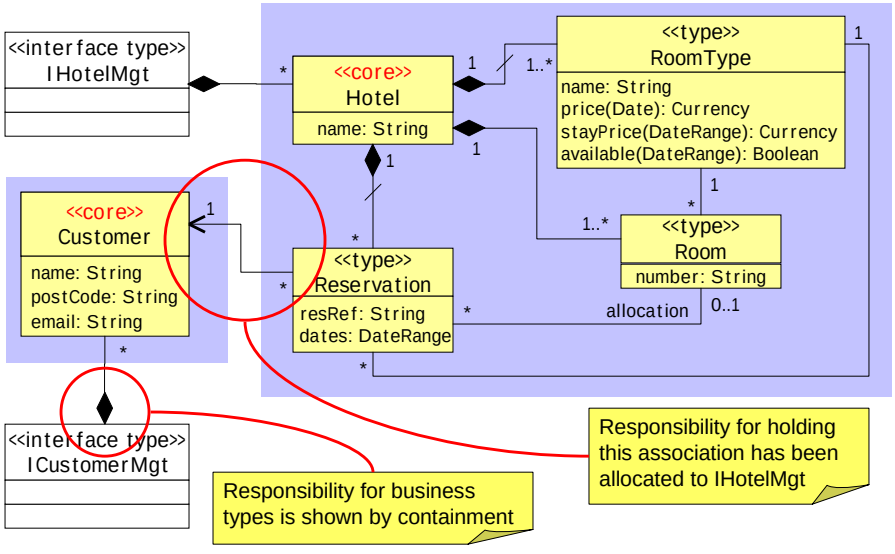
Develop the Business Type Model



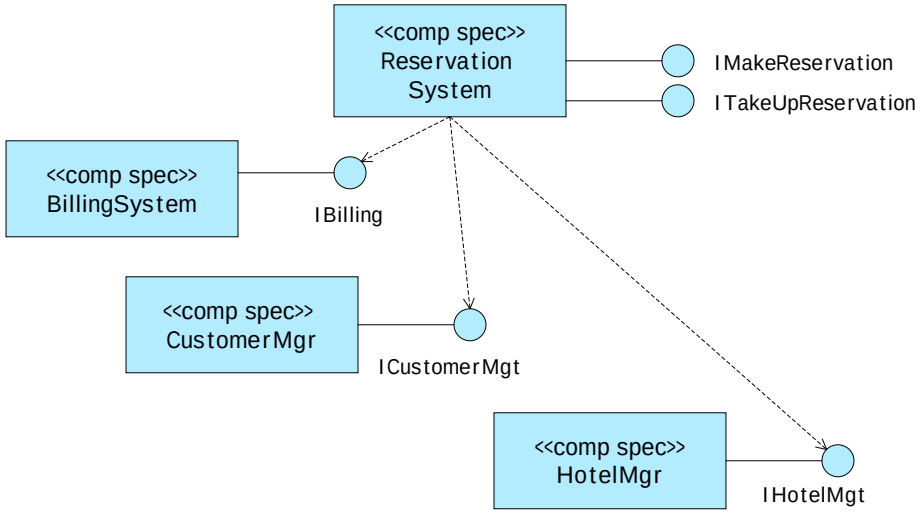
Initial Business Type Diagram



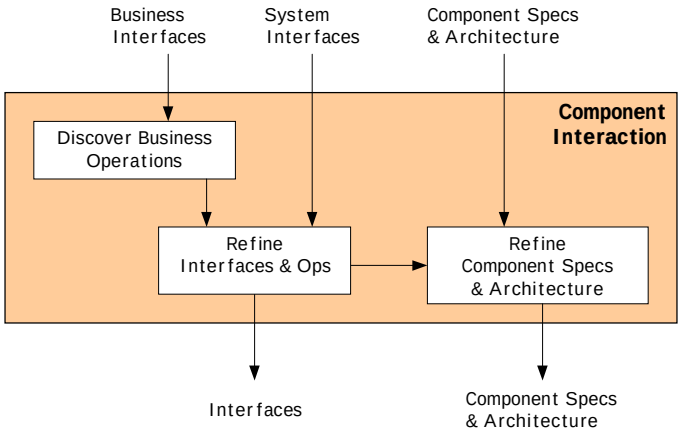
Identify business interfaces

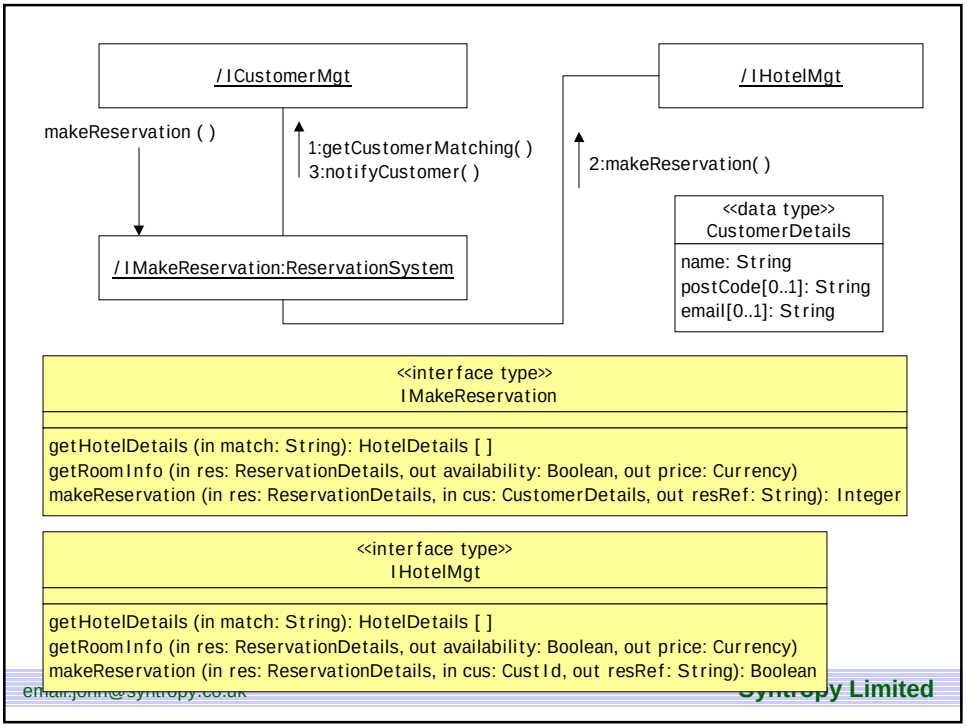


Component architecture

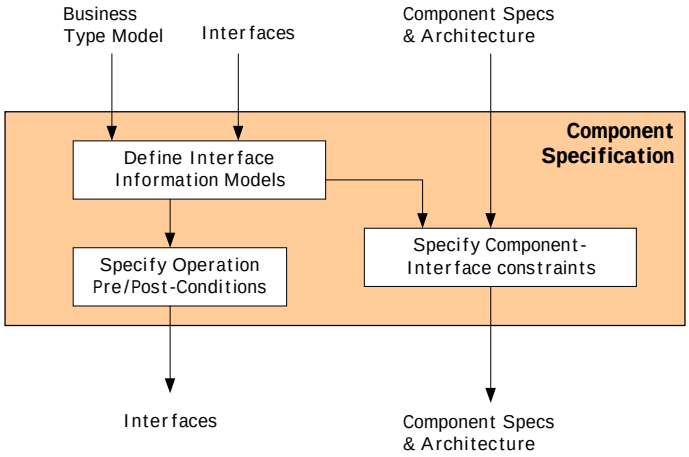


Component Interaction

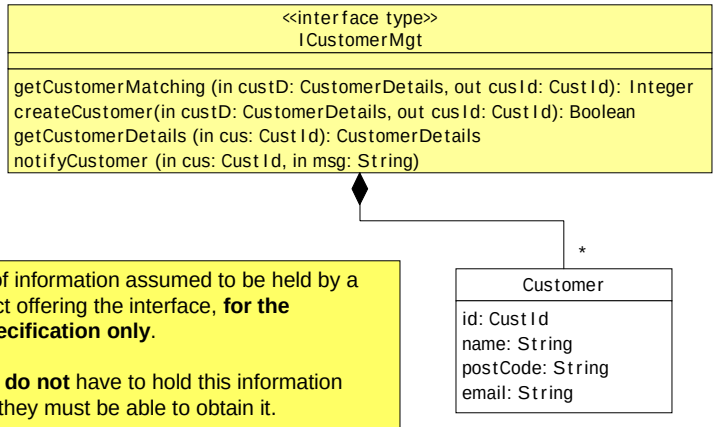




Component Specification



Interface information model



Defines the set of information assumed to be held by a component object offering the interface, **for the purposes of specification only**.

Implementations **do not** have to hold this information themselves, but they must be able to obtain it.

The model need only be sufficient to explain the effects of the operations.

The model can be derived from the Business Type Model.

Pre- and post-conditions

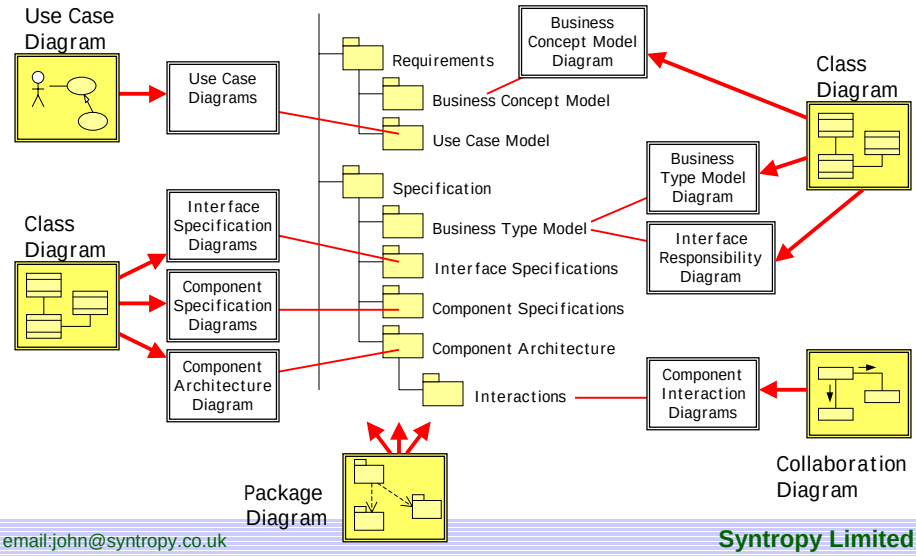
- If the pre-condition is true, the post-condition must be true
- If the pre-condition is false, the post-condition doesn't apply
- A missing pre-condition is assumed 'true'
- Pre- and post-conditions can be written in natural language or in a formal language such as OCL

```
context ICustomerMgt::getCustomerDetails (in cus: CustId): CustomerDetails

pre:
      -- cus is valid
      customer->exists(c | c.id = cus)

post:
      -- the details returned match those held for customer cus
      Let theCust = customer->select(c | c.id = cus) in
      result.name = theCust.name
      result.postCode = theCust.postCode
      result.email = theCust.email
```

UML diagrams



Want to know more?

- UML Components by John Cheesman and John Daniels, Addison-Wesley
- http://www.umlcomponents.com