Agenda

• Aspects of a component
• A process for component specification
• Implications for the UML
Aspects of a component
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
Aspects of an Object

- **Specification unit**

- **Implementation unit**
  - Class Implementation
  - Class
  - Object
  - Source: 0..1
  - Realization: 1..*
  - Instance: *

- **Execution unit**
Components in context

Object Principles

Only interoperable within the language. Single address space.

Developed within 1967-
Smalltalk
Object-oriented Programming

Typically language neutral. Multiple address spaces. Non-integrated services.

Adopted by 1989-
RMI
Distributed Object Technology

A way of packaging object implementations to ease their use. Integrated services.

Adopted by 1995-
EJB
Components

1967-
Smalltalk
Object-oriented Programming

1989-
RMI
Distributed Object Technology

1995-
EJB
Components

email: john@syntropy.co.uk

Syntropy Limited
Component standard features

- **Component Model:**
  - defined set of services that support the software
  - set of rules that must be obeyed in order to take advantage of the services

- **Simple programming model, no need to design/know about the infrastructure**

- **Services include:**
  - remote access, transactions, persistent storage, security
  - typically use services by configuring not programming
Aspects of a component

- It has a specification
- It can be deployed
- It can be packaged into modules
- It conforms to a standard
- It has an implementation

```
for (int i=0; i<limit; i++)
{
    list[i] = ...
}
```

email: john@syntropy.co.uk
Component forms

- Specification unit
  - Component Specification
    - Interface
      - spec
      - realization

- Implementation unit
  - Component Implementation
    - Component
      - source
      - instance
    - Object
      - Component Object
      - file

- Packaging unit
  - Module

- Execution unit
Two distinct contracts

Usage contract: a contract between a component object’s interface and a client

Realization contract: a contract between a component specification and a component

Component Spec

Component

Client

Interface

<<use>>

<<realize>>
Interface specification

We could specify `placeOrder()` like this:

“The number of orders for the customer is increased by one and a `reserveStock` message is sent to the component supporting the `IProductMgt` interface”
The client cares about this - it affects the subsequent result of `numOfOrders()`. Therefore it is part of the usage contract.

“The number of orders for the customer is increased by one and a `reserveStock` message is sent to the component supporting the `IProductMgt` interface”

The `IOrderMgt` client does not care about this - but the implementer does. Therefore it is part of the realization contract.

email: john@syntropy.co.uk
Interfaces versus Component Specs

**Component Interface**

- Represents the **usage** contract
- Provides a list of operations
- Defines an underlying logical information model specific to the interface
- Specifies how operations affect or rely on the information model
- Describes local effects only

**Component Specification**

- Represents the **realization** contract
- Provides a list of supported interfaces
- Defines the run-time unit
- Defines the relationships between the information models of different interfaces
- Specifies how operations should be implemented in terms of usage of other interfaces
Contracts and roles

Specifier (Architect)
A person who produces the technical specification for a system or components within a system

Realizer
A person who builds a component that meets a component specification

Client
A person who writes software that uses a component

The Usage Contract

Realization Contract
Component deployment

- Registration unit
- Installation unit
Example - Microsoft Word™

- `wordApplication`: Component
  - `wordDocument`: Component
    - `wordDocument`: InstalledComponent
      - `winword.exe`: ComponentModule
        - `C:\..\winword.exe`: InstalledModule
          - `File / New`:
            - `applicationObject`: ComponentObject
              - `docObject`: ComponentObject
                - `instance` of `wordApplication`:
                  - `instance` of `wordDocument`:
                    - `installed as` server:
                      - `installed as` server:
                        - `file`:
                          - `copy`:
                            - `file`:
Example - Enterprise Java Beans

invoice: ComponentSpec

realization

invoiceBean: Component

installed as

invoiceBean: InstalledComponent

server

c:/../invoice.jar: ComponentModule

copy

c:/../invoice.jar: InstalledModule

file

invoice.jar: ComponentModule

realization

invoiceABC: ComponentObject

InstalledModule

ComponentImplementation
A process for component specification
Application Architecture Layers

Presentation

Web Client

HTTP

RMI / IIOP / DCOM

Client

Web Server

RMI / IIOP / DCOM

Service

Application Server

JDBC / ODBC / SQL

Data

Database Server

Existing System (server)

Any
Web Server

A/JSP

Application Server

Component Object

Component Object

Component Object

Component Object

Database

Existing System

email: john@syntropy.co.uk

Syntropy Limited
Finer-Grain Application Layers

Presentation

User Interface
- UI Logic
- What the user sees

User Dialog
- Dialog Logic (UseCases)
- Supports multiple UIs
- Transient Dialog State

Service

System Services
- Business transactions
- Allows multiple Dialogs (including Batch)
- Business Integrity State

Business Services
- Sub-transactions
- Business Instance State

Data
Management and Development Processes

- **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
Workflows in the development process

- Business requirements
- Business Concept models
- Use Case models
- Component specs & architectures
- Technical constraints

Requirements → Specification → Provisioning → Assembly → Test → Deployment

Artefact

Existing assets

User interface

Workflows in the development process (c.f. RUP)

email: john@syntropy.co.uk

Syntropy Limited
The Requirements Workflow

1. Problem domain knowledge
2. Business requirements
3. Develop Business Concept Model
4. Identify Use Cases
5. Develop business processes
6. Use Cases

Software boundary decisions
We want to provide some automated support for managing hotel reservations.

Business process diagram:
- Check availability
- Make reservation
- Confirm reservation
- Wait for event
  - amendment request/
  - cancel request/
- Amend reservation
- Process no show
- Notify billing system
- Take up reservation
- Cancel reservation

Events:
- enquiry/
- [suitable room]
- [else]
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.
**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
The Specification Workflow

Requirements

Specification
- Component Identification
- Component Interaction
- Component Specification

Provisioning
Components in the service layers

Presentation

Service

System Services

- System Component
- System Component

Business Services

- Business Component
- Business Component

Data

System interfaces operations support use case steps

Business interfaces operations support core business logic
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.
Use case step operations

<<interface type>> IMakeReservation
getHotelDetails()
getRoomInfo()
makeReservation()

Return a list of hotels and the room types they have
Return price and availability given hotel, room type and dates
Create a reservation given hotel, room type and dates; return its tag

<<interface type>> ITakeUpReservation
getReservation()
beginStay()

Return reservation details given a tag
Given a tag, allocate a room and notify billing system
Develop the Business Type Model
Initial Business Type Model

- **Customer**
  - name: String
  - postCode: String
  - email: String

- **Hotel**
  - name: String

- **RoomType**
  - name: String
  - price: Currency

- **Room**
  - number: String

- **Reservation**
  - resRef: String
  - dates: DateRange

- **Allocation**
  - allocation: 0..1

---

Business Concept Model <<trace>> Business Type Model

email: john@syntropy.co.uk

Syntropy Limited
Identify Core types

- Core types represent the primary business information that the system must manage.
- Each core type will correspond directly to a business interface.
- A core type has:
  - a business identifier, usually independent of other identifiers
  - independent existence – no mandatory associations (multiplicity equal to 1), except to a categorizing type
- In our case study:
  - Customer  YES. Has id (name) and no mandatory assocs.
  - Hotel  YES. Has id (name) and no mandatory assocs.
  - Reservation  NO. Has mandatory assocs.
  - Room  NO. Has mandatory assoc to Hotel
  - RoomType  NO. Has mandatory assoc to Hotel
Identify business interfaces

Responsibility for business types is shown by containment

Responsibility for holding this association has been allocated to IHotelMgt

Syntropy Limited
Component Specifications

- We need to decide what components we want, and which interfaces they will support.
- These are fundamental architectural decisions.
- Business components:
  - They support the business interfaces.
  - Remember: components define the unit of development and deployment.
- The starting assumption is one component spec per business interface.

```
<<comp spec>>
CustomerMgr

ICustomerMgt

<<component spec>>
HotelMgr

IHotelMgt
```
System components

• We will define a single system component spec that supports all the use case system interfaces
  – Alternatives: one component per use case, support system interfaces on the business components

• Use a separate component spec for billing system wrapper

```
<<comp spec>>
Reservation System

IBilling
ICustomerMgt
IHotelMgt

IMakeReservation
ITakeUpReservation

<<comp spec>>
BillingSystem

IBilling
```
Minimal component object architecture

- Reservation System
- Billing System
- Customer Mgr
- Hotel Mgr

Interfaces:
- IMakeReservation
- ITakeUpReservation
- IBilling
- ICustomerMgt
- IHotelMgt
Component Interaction

- Business Interfaces
- System Interfaces
- Component Specs & Architecture

- Discover Business Operations
- Refine Interfaces &Ops
- Refine Component Specs & Architecture

Interfaces
Component Specs & Architecture
Operation discovery

- Uses interaction diagrams (collaboration diagrams)
- The purpose is to discover operations on business interfaces that must be specified
  - not all operations will be discovered or specified
- Take each use case step operation in turn:
  - decide how the component offering it should interact with components offering the business interfaces
  - draw one or more collaboration diagram per operation
  - define signatures for all operations
```
// ICustomerMgt
makeReservation()

1: getCustomerMatching()
3: notifyCustomer()

/IMakeReservation:ReservationSystem

// IHotelMgt
makeReservation()
2: makeReservation()

<<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>>
IHotelMgt

getHotelDetails (in match: String): HotelDetails[]
getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)
makeReservation (in res: ReservationDetails, in cus: CustId, out resRef: String): Boolean
```

<<data type>>
CustomerDetails

name: String
postCode[0..1]: String
email[0..1]: String
Component Specification

- Define Interface Information Models
- Specify Operation Pre/Post-Conditions
- Specify Component-Interface constraints

Business Type Model Interfaces

Component Specs & Architecture

Interfaces Component Specs & Architecture
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.
• 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
```
getHotelDetails (in match: String): HotelDetails []
getRoomInfo (in res: ReservationDetails, out availability: Boolean, out price: Currency)
makeReservation (in res: ReservationDetails, in cus: CustId, out resRef: String): Boolean
getReservation(in resRef: String, out rd ReservationDetails, out cusId: CustId): Boolean
beginStay (resRef: String , out roomNumber: String): Boolean

<<interface type>>
IHotelMgt

Reservation
resRef: String
dates: DateRange
claimed: Boolean

Hotel
id: HotelId
ame: String

Room
number: String

RoomType
name: String
available(during: DateRange): Boolean
price(on: Date): Currency
stayPrice(for: DateRange): Currency

Customer
id: CustId

allocation 0..1
before

after

makeReservation ()
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
Specifying a component (1)

Specification of interfaces offered and used
(part of the realization contract)
Specifying a component (2)

Specification of the *component object* architecture. This tells us how many objects offering the used interfaces are involved.
**Context** ReservationSystem

-- between offered interfaces
IMakeReservation::hotel = ITakeUpReservation::hotel
IMakeReservation::reservation = ITakeUpReservation::reservation
IMakeReservation::customer = ITakeUpReservation::customer

-- between offered interfaces and used interfaces
IMakeReservation::hotel = iHotelMgt.hotel
IMakeReservation::reservation = iHotelMgt.reservation
IMakeReservation::customer = iCustomerMgt.customer


The top set of constraints tell the realizer the required relationships between elements of different offered interfaces.

The bottom set tell the realizer the relationships between elements of offered interfaces and used interfaces that must be maintained.
Is every implementation of ReservationSystem required to invoke `getHotelDetails()` in this situation?
If so, drawing the collaboration diagram is an act of specification...
If not, then we are using this technique simply as a way of discovering useful operations.
If we want to provide a more detailed specification we can use interaction diagram fragments.

These are pieces of the diagrams we drew earlier, for operation discovery, that focus on the component being specified.

Each fragment specifies how a particular operation is to be implemented in terms of interaction with other components.

Warning: in some cases this will be over-specification.
UML diagrams used in the process

Use Case Diagram
- Use Case Diagrams
  - Requirements
    - Business Concept Model Diagram
  - Use Case Model

Class Diagram
- Business Concept Model
- Use Case Model
- Interface Specifications
  - Component Specifications
  - Component Architecture
  - Interactions
- Business Type Model
- Interface Responsibility Diagram

Package Diagram

Component Interaction Diagrams

Collaboration Diagram

email: john@syntropy.co.uk
Implications for the UML
- UML Glossary: “a physical, replaceable part [...] that packages implementation and [...] provides the realization of a set of interfaces”
**Mapping to UML**

<table>
<thead>
<tr>
<th>Concept</th>
<th>UML stereotype</th>
<th>UML element (1.4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>«interface type»</td>
<td>UML Class</td>
</tr>
<tr>
<td>Component Specification</td>
<td>«comp spec»</td>
<td>UML Class</td>
</tr>
<tr>
<td>Component</td>
<td>«comp imp»</td>
<td>UML Component</td>
</tr>
<tr>
<td>Component Module</td>
<td>«comp module»</td>
<td>UML Artifact</td>
</tr>
<tr>
<td>Installed Component</td>
<td>«installed comp»</td>
<td>UML Component</td>
</tr>
<tr>
<td>Installed Module</td>
<td>«comp server»</td>
<td>UML Component</td>
</tr>
<tr>
<td>Component Object</td>
<td>«comp object»</td>
<td>UML Component Instance</td>
</tr>
</tbody>
</table>

**NB your UML tool might let you use Interface**

**Strictly, a Set of Artifact**
Realization mappings

<<comp spec>>
CustomerMgr

<<realize>>

CustomerMgr

<<offers>>

<<interface type>>
ICustomerMgr

addCustomer() deleteCustomer()
getCustomer()

<<realize>>

<<interface>>
ICustomerMgr

addCustomer() deleteCustomer() getCustomer()
• 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
<table>
<thead>
<tr>
<th></th>
<th>Problem domain</th>
<th>S/W spec</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use case</strong></td>
<td></td>
<td>boundary</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>interactions</td>
<td></td>
</tr>
<tr>
<td><strong>Class diagram</strong></td>
<td>information</td>
<td>component</td>
<td></td>
</tr>
<tr>
<td></td>
<td>models</td>
<td>structures</td>
<td>structures</td>
</tr>
<tr>
<td><strong>Seq/collab diagram</strong></td>
<td>required</td>
<td>designed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>object</td>
<td>object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>interactions</td>
<td>interactions</td>
<td></td>
</tr>
<tr>
<td><strong>Activity diagram</strong></td>
<td>business</td>
<td></td>
<td>algorithms</td>
</tr>
<tr>
<td></td>
<td>processes</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Statechart</strong></td>
<td></td>
<td>object lifecycles</td>
<td>object lifecycles</td>
</tr>
<tr>
<td>Problem domain</td>
<td>S/W spec</td>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>----------------</td>
<td></td>
</tr>
</tbody>
</table>

**Business Concept Model**

- <<concept>> Customer

**Business Type Model, Interface Spec**

- <<type>> Customer
  - name: String
  - postCode: String
  - email: String

- <<interface type>> ICustomerMgt

- <<class>> Customer
  - name: String
  - postCode: String
  - email: String
  - setName (String)

*private implementation design*
Want to know more?

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