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
- Execution unit

Diagram:

- Interface
  - 1..*" realization
  - Class
    - 0..1 source
    - 1 instance
    - Object

Diagram notes:

- "realization" relationship between Interface and Class
- "source" relationship between Class and Implementation unit
- "instance" relationship between Class and Object

email: john@syntropy.co.uk
Syntropy Limited
Components in context

Object Principles

Only interoperable within the language. Single address space

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

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

Developed within

1967-

Smalltalk

Object-oriented Programming

C++

Java

1989-

RMI

Distributed Object Technology

CORBA

DCOM

1995-

EJB

Components

COM+/.NET

Adopted by

1967-

Smalltalk

Adopted by

1989-

RMI

Adopted by

1995-

EJB

Evolving

email: john@syntropy.co.uk

Syntropy Limited
• **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

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

email: john@syntropy.co.uk

Syntropy Limited
Component forms

- **Specification unit**
  - Component Specification
  - Interface
  - Component Module
  - Packaging unit

- **Implementation unit**
  - Component Implementation
  - Class
    - source
    - 0..1
    - realization
    - 1
  - Object
    - instance
    - *
  - Component
    - source
    - 0..1
    - 1
    - realization
    - *
    - instance
    - *
  - Implementation
    - source
    - 0..1
    - 1

- **Execution unit**
  - Component Object
  - file
  - 1..*

email: john@syntropy.co.uk

Syntropy Limited
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
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”

<table>
<thead>
<tr>
<th>IOrderMgt</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>placeOrder(custNum, prodNum, quan)</code></td>
</tr>
<tr>
<td><code>numOfOrders(custNum): Integer</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IProductMgt</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>reserveStock(prodNum, quan)</code></td>
</tr>
<tr>
<td><code>availableStock(prodNum): Integer</code></td>
</tr>
</tbody>
</table>
Separation of specification concerns

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.
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

Diagram showing relationships between Component, Installed Component, Component Module, and Installed Module with edges labeled as follows:
- Component has a cardinality of 1..* for Component Module
- Installed Component has a cardinality of * for installed as
- Component Module has a file cardinality of 1..*
- Installed Module has a cardinallity of 1..* for copy and server
Example - Microsoft Word™

- wordApplication: Component
  - wordDocument: InstalledComponent
    - installed as
      - file
        - copy
          - winword.exe: ComponentModule
            - C:../winword.exe: InstalledModule
    - server
      - file
        - installed as
          - wordApplication: InstalledComponent
            - instance
              - applicationObject: ComponentObject
              - instance
                - File / New
                  - docObject: ComponentObject

Example - Enterprise Java Beans

invoice: ComponentSpec

invoiceBean: Component

invoiceBean: InstalledComponent

invoiceABC: ComponentObject

c:/../invoice.jar: InstalledModule

invoice.jar: ComponentModule

c:/../invoice.java: ComponentImplementation

file

realization

installed as

copy

server
A process for component specification
Application Architecture Layers

### Presentation
- **Client**
  - RMI / IIOP / DCOM
- **Web Client**
  - HTTP
  - RMI / IIOP / DCOM

### Service
- **Application Server**
- **Web Server**
  - RMI / IIOP / DCOM
  - JDBC / ODBC / SQL
- **Existing System (server)**
  - Any

### Data
- **Database Server**

---

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

---

Use Case models → **Specification Workflow** → Component specs & architectures

Business Concept models
Workflows in the development process

Business requirements

Requirements

Business Concept models

Use Case models

Technical constraints

Component specs & architectures

Provisioning

Artefact

Existing assets

User interface

Assemblies

Test

Deployment

Workflow (c.f. RUP)
The Requirements Workflow

- Problem domain knowledge
- Business requirements

Develop Business Concept Model → Identify Use Cases → Develop business processes

Software boundary decisions

Business Concept Model → Use Cases
We want to provide some automated support for managing hotel reservations.
Business Concept Model

- Hotel Chain
  - Hotel
    - Customer
      - Address
    - Reservation
      - Room
        - RoomType
    - Bill
  - Clerk
  - Payment

Relationships:
- 1 Hotel Chain
- 1..* Hotel
- 1..* Contacted Hotel
- 1 Reservation
- 1..* Room
- 0..1 Contact Address
- 0..1 Allocation
- 0..1 Contact Address
- 0..1 Payment
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.
Reservation system

- Cancel a reservation
- Make a reservation
- Update a reservation
- Take up a reservation
- Process no shows
- Add, amend, remove hotel, room, customer, etc.

Use Case diagram
**Name** | **Make a Reservation**  
---|---  
**Initiator** | **Reservation Maker**  
**Goal** | **Reserve a room at a hotel**  

**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

**Alternatives**
Use an informal “Alternatives” section if you don’t want to specify the detail required for an extension
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

email:john@syntropy.co.uk
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.

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.

Use case

Make a Reservation

Dialog logic

Make Reservation

System Interface

IMakeReservation

getHotelDetails()
getRoomInfo()
makeReservation()

Use case steps

2. Reservation Maker selects hotel, dates and room type
3. System provides availability and price
7. System makes reservation and gives it a tag
8. System reveals tag to Reservation Maker
9. System creates and sends confirmation by email
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 Diagram

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

<<type>> Reservation
  resRef: String
  dates: DateRange

<<type>> Hotel
  name: String

<<type>> RoomType
  name: String
  price: Currency

<<type>> Room
  number: String
  allocation 0..1

<<type>> Room

Business Concept Model

<<trace>>

Business Type Model

Syntropy Limited
```
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
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.

Diagram:

```
<<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
Component architecture

- **Reservation System**
  - IMakeReservation
  - ITakeUpReservation

- **BillingSystem**
  - IBilling

- **CustomerMgr**
  - ICustomerMgt

- **HotelMgr**
  - IHotelMgt
Minimal component object architecture

```
<<comp object>> :Reservation System

<<comp object>> :BillingSystem
IBilling

<<comp object>> :CustomerMgr
ICustomerMgt

<<comp object>> :HotelMgr
IHotelMgt

IMakeReservation
ITakeUpReservation
```
Component Interaction

Discover Business Operations

Refine Interfaces & Ops

Component Interaction

Refine Component Specs & Architecture

Business Interfaces

System Interfaces

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
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

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
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
```
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
  name: String

Room
  number: String

Customer
  id: CustId

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

allocation
  0..1

Syntropy Limited
before

makeReservation ( )

after
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.
Specifying a component (3)

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.
Interactions as specification?

- 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
Specifying a component (4)

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

Class Diagram

Package Diagram

Business Concept Model Diagram

Business Type Model Diagram

Interface Responsibility Diagram

Component Interaction Diagrams

Collaboration Diagram

email: john@syntropy.co.uk

Syntropy Limited
Implications for the UML
UML Component (v1.4)

- **Implementation characteristics**
  - Artifact
  - Classifier
  - Node

- **Deployment characteristics**
  - UML Component
    - Component Instance
    - 1

- **Structure and Behavior characteristics**
  - Package characteristics
    - Model Element

- **Package characteristics**
  - Classifier
  - Node

- **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>&lt;&lt;interface type&gt;&gt;</td>
<td>UML Class</td>
</tr>
<tr>
<td>Component Specification</td>
<td>&lt;&lt;comp spec&gt;&gt;</td>
<td>UML Class</td>
</tr>
<tr>
<td>Component</td>
<td></td>
<td>UML Component</td>
</tr>
<tr>
<td>Component Implementation</td>
<td>&lt;&lt;comp imp&gt;&gt;</td>
<td>UML Artifact</td>
</tr>
<tr>
<td>Component Module</td>
<td>&lt;&lt;comp module&gt;&gt;</td>
<td>UML Artifact</td>
</tr>
<tr>
<td>Installed Component</td>
<td>&lt;&lt;installed comp&gt;&gt;</td>
<td>UML Component</td>
</tr>
<tr>
<td>Installed Module</td>
<td>&lt;&lt;comp server&gt;&gt;</td>
<td>UML Component</td>
</tr>
<tr>
<td>Component Object</td>
<td>(&lt;&lt;comp object&gt;&gt;)</td>
<td>UML Component Instance</td>
</tr>
</tbody>
</table>

NB your UML tool might let you use Interface

Strictly, a Set of Artifact

email: john@syntropy.co.uk

Syntropy Limited
Realization mappings

<<comp spec>>
CustomerMgr

<<realize>>
CustomerMgr

<<interface type>>
ICustomerMgt

<<offers>>
addCustomer()
deleteCustomer()
getCustomer()

<<realize>>
CustomerMgr

<<interface>>
ICustomerMgt

<<offers>>
addCustomer()
deleteCustomer()  
getCustomer()
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
### Typical usage of UML notations

<table>
<thead>
<tr>
<th></th>
<th>Problem domain</th>
<th>S/W spec</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use case</td>
<td></td>
<td>boundary interactions</td>
<td></td>
</tr>
<tr>
<td>Class diagram</td>
<td>information models</td>
<td>component structures</td>
<td>component structures</td>
</tr>
<tr>
<td>Seq/collab diagram</td>
<td></td>
<td>required object interactions</td>
<td>designed object interactions</td>
</tr>
<tr>
<td>Activity diagram</td>
<td>business processes</td>
<td></td>
<td>algorithms</td>
</tr>
<tr>
<td>Statechart</td>
<td></td>
<td>object lifecycles</td>
<td>object lifecycles</td>
</tr>
</tbody>
</table>
Same name, different purpose

Business Concept Model

<table>
<thead>
<tr>
<th>Problem domain</th>
<th>S/W spec</th>
<th>Implementation</th>
</tr>
</thead>
</table>

```
<<concept>>
Customer
```

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

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

<<interface type>>
ICustomerMgt

Business Type Model, Interface Spec

private implementation design
Want to know more?

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