Implementation, Process, and Deployment Views

1. Motivation
2. Process View
3. Implementation View
4. Deployment View
5. ATM Example
1. Motivation

- Complex software systems involve a wide range of functionality, deployed on independent processing nodes, involving a wide variety of languages, platforms, and technologies.
- **Example: (a complex web-based system)**

The process, implementation, and deployment views capture this complexity by:
- Describing runtime entities: the threads and processes that form the system’s concurrency and synchronization.
- Describing source and executable components, their organization, and their dependencies.
- Describing hardware topology and mapping software components to processing nodes.
- Describing build procedures.
2. Process View

Overview

- Consists of the *processes* and *threads* that form the system’s *concurrency* and *synchronization* mechanisms, as well as their interactions.

- Addresses issues such as:
  - Concurrency and parallelism (e.g. synchronization, deadlocks etc.)
  - Fault tolerance (e.g. isolation of functions and faults, reliability)
  - System startup and shutdown
  - Object and data distribution
  - Performance (response time, throughput) and scalability

- Is captured using *class, interaction and statechart diagrams* with a *focus on active classes and objects*. Derives from the Logical view the concurrency and synchronization mechanisms underlying the software product.
**Processes and Threads**

- **Process**: a heavyweight flow of control that can execute independently and concurrently with other processes.
- **Thread**: a lightweight flow that can execute independently and concurrently with other threads within the same process.

Independent flows of control such as **threads** and **processes** are modeled as *active objects*. An active object is an instance of an *active class*. You may specify a process using the stereotype *process* and a thread using the stereotype *thread*.

<table>
<thead>
<tr>
<th>&lt;&lt;process&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReservationAgent</td>
</tr>
<tr>
<td>location</td>
</tr>
</tbody>
</table>

An **Active Object** is an object that owns a process or thread and can initiate control activity.

- Graphically an Active Class is represented as a class with thick lines.

Plain classes are called **passive** because they cannot independently initiate control.
Communication

• You model interprocess communication using interaction diagrams:
  - Synchronous communication
  - Asynchronous communication

• Two approaches: RPC (synchronous) and message passing (asynchronous)

Synchronization

• Modeled by adding constraints to the operations; there are three kinds of synchronization:
  - Sequential
  - Guarded
  - Concurrent

```
<<thread>>
Buffer

size: Integer

add() {concurrent}
remove() {concurrent}
```
sequential
• Callers must coordinate so that only one call to an *Instance* (on any sequential *Operation*) may be outstanding at once. If simultaneous calls occur, then the semantics and integrity of the system cannot be guaranteed.

guarded
• Multiple calls from concurrent threads may occur simultaneously to one *Instance* (on any guarded *Operation*), but only one is allowed to commence; the others are blocked until the performance of the first *Operation* is complete.

• It is the responsibility of the system designer to ensure that deadlocks do not occur due to simultaneous blocks.

concurrent
• Multiple calls from concurrent threads may occur simultaneously to one *Instance* (on any concurrent *Operations*). All of them may proceed concurrently with correct semantics.

**Note:** Java uses the *Synchronized* modifier, which maps to UML *Concurrent property.*
Example

Consider a trip planning service (e.g. expedia etc.) that is used by travelers to identify and book all at once the best deal in terms of flight, hotel, car rental etc. Model a basic scenario where a customer uses the system to book flight and hotel room by highlighting the concurrency and synchronization involved.
3. Implementation View

Overview

- Describes the *organization of static software modules* (source code, data files, executables, documentation etc.) in the development environment in terms of:
  
  - **Packaging and layering**
  - **Configuration management** (ownership, release strategy etc.)

- Are modeled using *UML Component Diagrams*.
  
  - UML components are physical and replaceable parts of a system that conform to and provide the realization of a set of interfaces

*Three kinds of components:*

- **Deployment components**: components necessary and sufficient to form an executable system, such as DLLs, executables etc.
- **Work product components**: residue of development process such as source code files, data files etc.
- **Execution components**: created as a consequence of executing system such as COM+ which is instantiated from a DLL.
UML Components

Notation

Standard Component Stereotypes
- **executable**: a component that may be executed on a node
- **library**: a static or dynamic object library
- **table**: a component that represents a database table
- **file**: a component that represents a document source code or data
- **document**: a component that represents a document
Components and Classes
• There are significant differences between components and classes:
  • *classes represent logical abstractions*
  • *components represent physical entities that live on nodes*
• A component is a physical element that provides the implementation of logical element such as classes (that is shown using a dependency relationship)
Component Interfaces

• An interface is a *collection of operations that are used to specify a service of a class or a component.*
• Interfaces provide the glue that binds components together
• A component may provide the implementation of an interface (realization) or may access its services (dependency).
Examples

• **Executable Release** (for a web-based application)
Source Code (showing different versions of the same program)
4. Deployment View

Overview
- Shows how the various executables and other runtime entities are mapped to the underlying platforms or computing nodes.

- Addresses issues such as:
  • Deployment
  • Installation
  • Maintenance

Concentrates on how the software is deployed into that somewhat important layer we call 'hardware'.

Exposes:
• System performance
• Object/data distribution
• Quality of Service (QoS)
• Maintenance frequency and effects on uptime
• Computing nodes within the system


**Deployment Diagram**

**Notation**

- A *node* is a *physical element representing a computational resource*, generally having some memory and processing capability.

- Nodes are used to model the topology of the hardware on which the system executes: processor or device on which components may be deployed.

- You may *organize nodes by specifying relationships* among them.
Nodes and Components

- Nodes are locations upon which components are deployed.
- A set of objects or components that are allocated to a node as a group is called a **distribution unit**.

You may also **specify attributes and operations** for them: *speed, memory*

```
S: server
processorSpeed=300mHz
memory=1Gb
Deploys
dbadmin.exe
tktmstr.exe
```

```
Deploys
pos.exe
contacts.exe
```

```
sales
- pos.exe
- contacts.exe
```
**Deployment Diagram**

- You use a deployment diagram to model the static deployment view of a system.

  - Example 1
- Example 2

- kiosk
- RAID farm
- console

<<10-T Ethernet>>

<<processor>>

<<RS-232>>
• **Distribution of Components**

![Diagram showing distribution of components]

- **:kiosk**
  - 

- **:RAID farm**
  - 

- **c:console**
  - 

- **S: server**
  - processorSpeed=300mHz
  - memory=1Gb
  - Deploys
    - dbadmin.exe
    - tktmstr.exe
5. ATM Example

Customer

- CardReader
- Display
- KeyPad
- Dispenser Feeder
- Dispenser Sensor
- Cash Counter

Client Manager

- Transaction Manager
- Withdrawal
- Account Manager

Persistent Class

Account
Process View

Class diagram

<table>
<thead>
<tr>
<th>Classes</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display, KeyPad, CardReader, ClientMgr, DispenserFeeder, DispenserSensor, CashCounter</td>
<td>ClientMgr</td>
</tr>
<tr>
<td>Transaction Mgr, Withdrawal</td>
<td>TransactionMgr</td>
</tr>
<tr>
<td>AccountMgr, Account, Persistent class</td>
<td>AccountMgr</td>
</tr>
</tbody>
</table>
**Implementation View**

- **Source Components**

<table>
<thead>
<tr>
<th>Classes</th>
<th>Source Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>CardReader, Display, KeyPad</td>
<td>Display.java</td>
</tr>
<tr>
<td>DispenserFeeder, DispenserSensor, CashCounter</td>
<td>DispenserFeeder.java</td>
</tr>
<tr>
<td>ClientMgr</td>
<td>ClientMgr.java</td>
</tr>
<tr>
<td>TransactionMgr</td>
<td>TransactionMgr.java</td>
</tr>
<tr>
<td>AccountMgr</td>
<td>AccountMgr.java</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>Withdrawal.java</td>
</tr>
<tr>
<td>Account, Persistent class</td>
<td>Account.java</td>
</tr>
</tbody>
</table>
-Executable Release

Executable Release

Executable
ClientMgr.exe

Binary
Display.class
DispenserFeeder.class
Account.class
Withdrawal.class
AccountMgr.class
TransactionMgr.class
Deployment View

- Deployment Diagram

Customer

ATM Client

SNA

ATM Application Server

intranet

ATM Data Server

- Deployment of Active Objects

:ATM Client

:ClientManager

:ATM Application Server

:TransactionManager

:ATM Data Server

:AccountManager