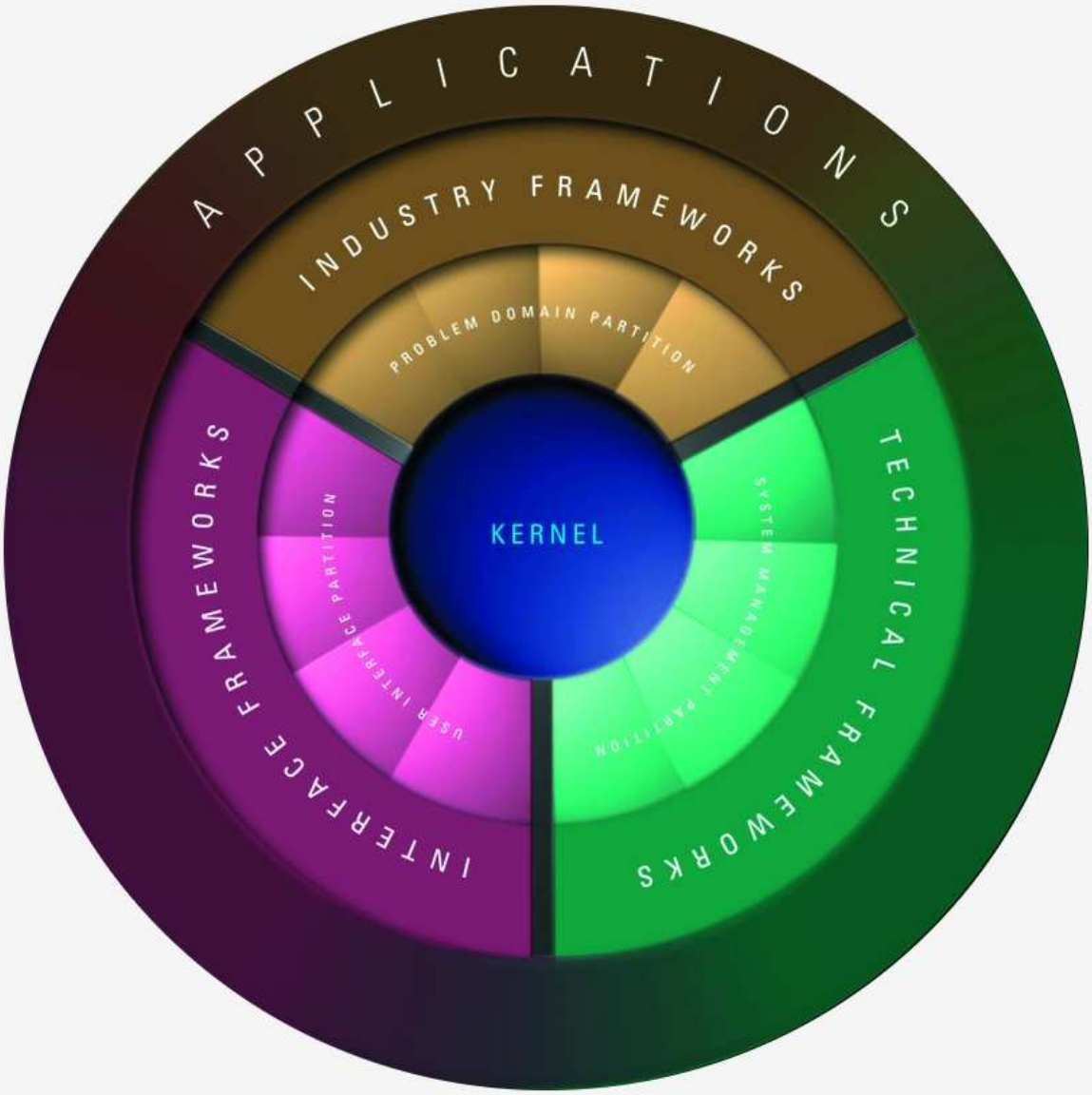
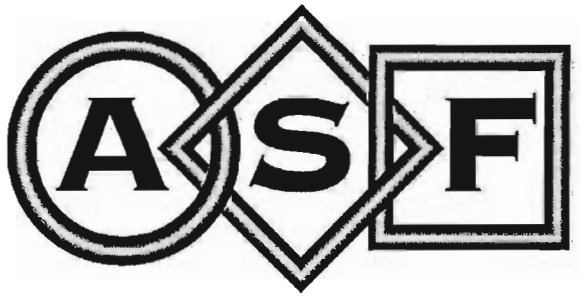
Architectures for Object-Oriented Development



ASF
AUSTIN SOFTWARE FOUNDRY



AUSTIN SOFTWARE FOUNDRY

Architectures for Object-Oriented Development

January 1998

Printed: January 1998

AUSTIN SOFTWARE FOUNDRY
500 CAPITAL OF TEXAS HIGHWAY NORTH
BUILDING 8, SUITE 250
AUSTIN, TX 78746
(512) 329 - 6697 PHONE
(512) 329-6698 FAX
WWW.FOUNDRY.COM

Table of Contents

Unit 1: Introduction

Objectives 1-1
Topics 1-1
The Big Picture 1-2
    What You Already Know 1-3
    Course Objectives 1-4
    Course Overview 1-5
Object-Oriented Development 1-7
Use Case Model 1-8
    Use Cases 1-9
    Class Diagram 1-10
    Interaction Diagrams 1-11
    Types of Objects 1-12
    Object-Oriented Development Goals 1-13
Objects Verses Components 1-14
Why Components? 1-16
Alternatives to the Confusion and Complexity 1-17
Summary 1-19

Unit 2: Architecture Basics

Objectives 2-1
Topics 2-1
Architecture 2-2
Information System Architecture 2-5
Software Architecture 2-9
    Software Architecture for Distributed Systems 2-11
    Software Architecture for Distributed Business Systems 2-12
Summary 2-16

Unit 3: Pattern Basics

Objectives 3-1
Topics 3-1
Patterns 3-2
    Object-Oriented Patterns 3-4
Software Pattern History 3-5
    Types of Object-Oriented Patterns 3-6
    Patterns and Frameworks 3-7
Pattern Organization 3-8
    Elements of a Pattern 3-8
    Pattern Catalogs and Systems 3-11
    Pattern Catalogs 3-11
    Pattern Systems 3-12
    Pattern Classification 3-13
Using Patterns 3-17
Summary 3-20

Unit 4: User Navigation

Objectives 4-1
Topics 4-1
Interface Patterns 4-2
MDI 4-3
Master/Detail 4-7
Summary 4-11

Unit 5: Solving Business Problems

Objectives 5-1
Topics 5-1
Domain Patterns 5-2
Item Description 5-3
Event Logging 5-5
Time Association 5-7
Player/Player Role 5-9
Summary 5-12

Unit 6: Applying Patterns to Business Process

Objectives 6-1
Topics 6-1
Business Architecture 6-2
Use Case Domain Pattern 6-3
Business Control/Business Entity Relationships 6-9
Use Case Domain Pattern Example 6-11
Summary 6-17

Unit 7: Structuring Applications: Partitioning

Objectives 7-1
Topics 7-1
Application Partitions 7-2
    Partitioned Application Architecture 7-3
    Problem Domain Partition 7-4
    User Interface Partition 7-6
    System Management Partition 7-8
    Benefits of Partitioning 7-10
Application Architecture 7-11
    Application Architecture Relationships 7-12
Summary 7-15

Unit 8: Structuring Applications: Layering

Objectives 8-1
Topics 8-1
Layer Architectural Pattern 8-2
    Application Architecture Layers 8-4
    Kernel Layer 8-5
    Class & Component Layer 8-6
    Framework Layer 8-7
    Application Layer 8-8
Extension Layer Pattern 8-10
    Extension Layer Example 8-13
Application Organization 8-14
    Application Base Classes 8-16
    Application Components 8-17
    Application Kits 8-18
Summary 8-19

Unit 9: Managing and Accessing Object Collections

Objectives 9-1
Topics 9-1
Management Design Patterns 9-2
Containers 9-4
    Container Pattern 9-9
    Example Using Containers 9-12
Collections 9-14
Summary 9-18

Unit 10: Controlling Component Access

Objectives 10-1
Topics 10-1
Access Control 10-2
Proxy 10-5
Facades 10-12
Summary 10-18

Unit 11: Creating Objects

Objectives 11-1
Topics 11-1
Creation Patterns 11-2
Object Factory 11-4
Singleton 11-9
Summary 11-13

Unit 12: Adapting to Other Components and Systems

Objectives 12-1
Topics 12-1
Adaptation Patterns 12-2
Adapter 12-4
Summary 12-11

Unit 13: Designing the Logical Business Architecture

Objectives 13-1
Topics 13-1
Use Case Domain Pattern Review 13-2
Use Case Pattern Example 13-5
Use Case Pattern/Conceptual Design 13-6
Context Design Pattern 13-7
Applying the Context Pattern 13-10
Designing PowerBuilder OO Applications 13-15
Option 1 - One Class Instance for Each Database Row 13-17
Option 2 - Single Instance with an Array of Attributes 13-19
Option 3 - Single Instance, Use DataWindow for Attributes 13-20
DataWindow Registering 13-22
Summary 13-25

Unit 14: Connecting the Partitions

Objectives 14-1
Topics 14-1
Interaction Patterns 14-2
Model-View-Controller 14-4
Implementing Model-View-Controller 14-11
Summary 14-16

Unit 15: Object Communication

Objectives 15-1
Topics 15-1
Communication Patterns 15-2
Command Processor 15-4
Publish-Subscribe 15-11
Implementing Publish-Subscribe within MVC 15-18
Summary 15-22

Appendix A: Discovering and Documenting Patterns

Patterns A-1
Documenting New Patterns A-2
    Pattern Reviews A-3
Integrating New Patterns into the Pattern System A-4
Adjusting Pattern Organization Schemes A-4
Removing Outdated Patterns A-5

Appendix B: Bibliography

References B-1

Unit 1: Introduction

Objectives

Upon completion of this unit, you should be able to:

- Describe *what* this course is about
- Review common *deliverables* used in object-oriented development
- Discuss objects as they relate to *component* development

Topics

- The Big Picture
- Object-Oriented Development
- Objects Verses Components

The Big Picture

Many organizations have realized that object-oriented development is key to developing reusable, flexible, robust, scaleable applications. They have trained developers in object-oriented development techniques and are using object-oriented development tools for application development. However, many organizations are realizing that creating good object-oriented designs that are flexible, loosely coupled, and truly reusable does not come automatically. Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. That is why many organizations are moving towards a **pattern-oriented software architecture.**

How do patterns help with the object-oriented design challenge? The use of patterns is an effort to build on the collective experience of skilled designers and software engineers that have already proven solutions to many recurring design problems. Patterns make it easier to reuse successful designs and architectures. This allows novices to act as if they were experts without having to gain many years of experience.

This course will introduce the basic concepts required to read and utilize published patterns, as well as a process for documenting newly discovered patterns. The overall goal of the course is to teach developers an architecture-driven approach to developing applications using proven domain, user interface, and design patterns. These patterns will be used in the development of an application. Developers will start with analysis and create a pattern-oriented architecture that will be used to construct a fully functional application.

The primary techniques taught in this course for accomplishing this goal are:

- Selection, review, and application of various types of patterns throughout the analysis and design process.
- Separation of three distinct groups of objects within every application:
  - User interface objects,
  - Business objects,
  - System management objects.
- Organization of an application's architecture using application layering techniques.

What You Already Know

This course combines many of the techniques and skills you have learned in earlier courses, or while developing object-oriented applications. You should already be familiar with the following concepts:

- **Object-oriented analysis techniques** - You should be conversant with object-oriented analysis concepts, such as object, class, association, aggregation, generalization, etc. You should be able to take a requirements definition and create a problem domain object model from it.
- **Object-oriented design techniques** - You should be familiar with object-oriented design techniques, such as polymorphism and delegation. This course does not focus on how to implement object-oriented design techniques using an object-oriented language.
- **Object modeling using Rational Rose** - You should be experienced in using Rational Rose to develop and modify object models. Several class lab exercises will be accomplished by using Rational Rose to alter object models.
- **Object-oriented language syntax** - You should be familiar with the syntax of an object-oriented language (PowerBuilder or Java). During the construction portion of this course, you will be asked to develop the application that you have designed using the development language chosen for the course. This course does not cover object-oriented features of the language.

Course Objectives

Upon completion of this course, you should be able to:

- **Explain why business and application architectures are key to developing flexible, reusable systems.** You should be able to explain to others what business and application architectures are and how they provide the foundation for sound application designs.
- **Create an architecture-driven partitioned application architecture.** You should be able to take a set of analysis and design specifications and create an application architecture that will effectively partition the user interface, business, and utility objects.
- **Analyze a design problem and select and apply a domain, design, or user-interface pattern to solve the problem.** You should be able to take the abstract definition of a domain, design, or user-interface pattern and apply it in a specific way to your application.
- **Develop object-oriented systems using patterns.** You should be able to assemble interface, business, and utility objects constructed in PowerBuilder into complete object-oriented applications based upon patterns.
- **Incorporate the process for mining and reviewing new patterns into your development process.** You should be able to define new patterns, develop a pattern language for describing patterns, and incorporate new patterns into a pattern system for your organization.

Course Overview

This course is broken into several parts. Units 1 through 3 are an introduction and present basic pattern and architecture concepts. Unit 4 discusses user-interface patterns that can be used for user navigation. Units 5 and 6 cover solving business problems and building business components using domain patterns. Units 7 and 8 introduces architecture patterns that are used to structure an application. Units 9 through 15 cover various aspects of application design including such topics as object creation, access, management and communication. Finally, in the optional units of the course, the design created during the first two parts of the course are implemented using object-oriented programming.

- **Unit 2: Architecture Basics** - presents basics of architecture including information systems architecture as well as business and application architecture.
- **Unit 3: Patterns Basics** - presents an overview of patterns with a review of the history of patterns, the leading figures and their works, and terminology related to pattern descriptions, pattern catalogs, and pattern systems.
- **Unit 4: Solving User Interface Problems** - introduces several commonly used user interface patterns and gives examples of their use.
- **Unit 5: Solving Business Problems** - introduces several patterns that can be used to solve common reoccurring domain problems such as event logging, time association, item description, and multiple roles.
- **Unit 6: Applying Patterns to Business Processes** - covers how to translate events and business responses from use case definitions into objects, attributes, methods, and relationships in an object model.
- **Unit 7: Structuring Applications: Partitioning** - covers how to design a partitioned application architecture that can easily be distributed using the separation of user-interface, problem domain, and system management components.
- **Unit 8: Structuring Applications: Layering** - covers how to design a layered application architecture ensuring that relationships between layers in an application promote extensibility and reusability.

- **Unit 9: Managing and Accessing Object Collections** - introduces patterns that can be used to create base classes such as containers to manage collections of objects that can occur in any partition in an application.

- **Unit 10: Controlling Component Access** - introduces patterns, such as proxy and facade, that can help guard and control access to services or components.

- **Unit 11: Creating Objects** - introduces patterns, such as object factory and singleton that help in the creation of objects, services, and components.

- **Unit 12: Adapting to Other Components and Systems** - covers how to design an adaptable architecture where components can be easily plugged into or swapped out of an application with minimal application changes.

- **Unit 13: Designing Business Servlets** - introduces a pattern to convert the logical creation of business components into a flexible design for business entities, rules, and contexts.

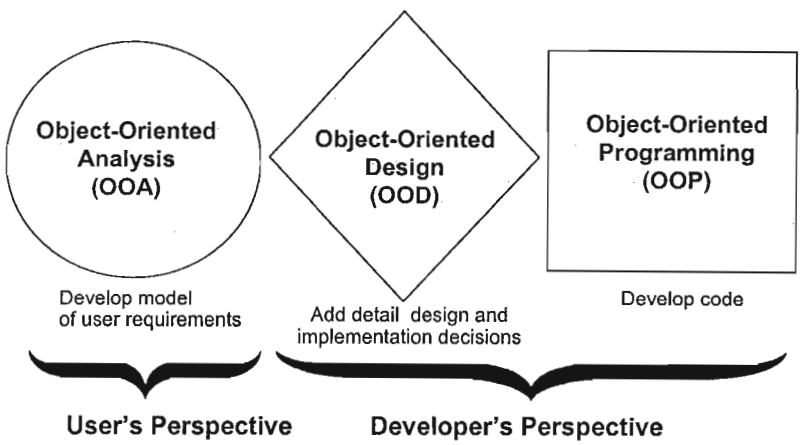
- **Unit 14: Connecting the Partitions** - covers the model-view-controller pattern that defines how the user interface partition should interact with the rest of the system.

- **Unit 15: Object Communication** - describes how patterns can be used to help organize the communication between components.

Object-Oriented Development

The primary goal of software development is to produce high quality software that is flexible and meets the needs of users. You have already realized that object-oriented techniques have the potential to help you achieve your goal of producing higher-quality software. You know that objects provide a stable, integrating framework that helps systems designers develop solutions using real business objects with attributes and behavior.

Object-oriented development methods focus on three major steps: object-oriented analysis (OOA), object-oriented design (OOD), and object-oriented programming (OOP). During object-oriented analysis, conceptual models are developed which focus on user requirements and views the system from a user's perspective. During object-oriented design, logical models are created where detailed design and implementation decisions are made. The logical models are physically constructed and implemented during object-oriented programming. Object-oriented design and programming is a view of the system from a developer's perspective.

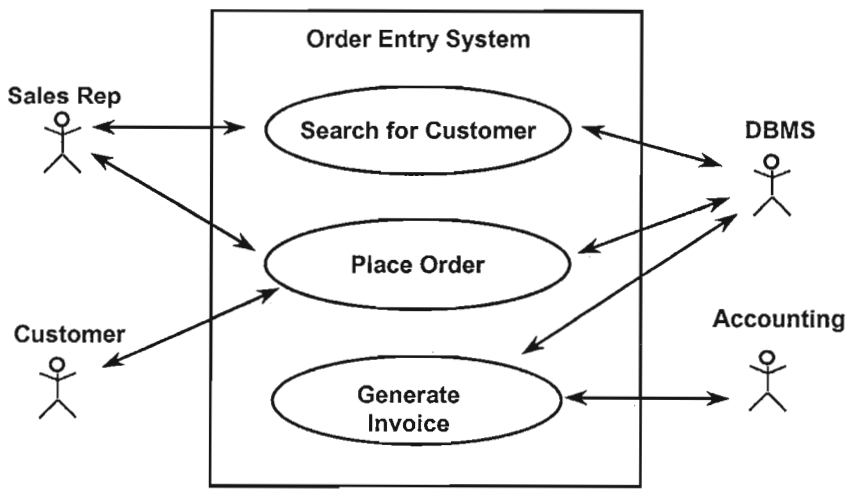


Let's review some of the key models used during object-oriented analysis and design to develop the objects in an application and their relationships.

- Use Case Diagram
- Class Diagram
- Behavior Diagram
- Activity Diagram

Use Case Model

The model used to describe the primary types of interactions between the system and its users is called the **use case model.** Each of these interactions is known as a **use case**, and a system comprises one or more use cases. Use case models can be employed to identify classes and objects in the system, or to divide the system up into manageable subsystems for development.



Use Cases

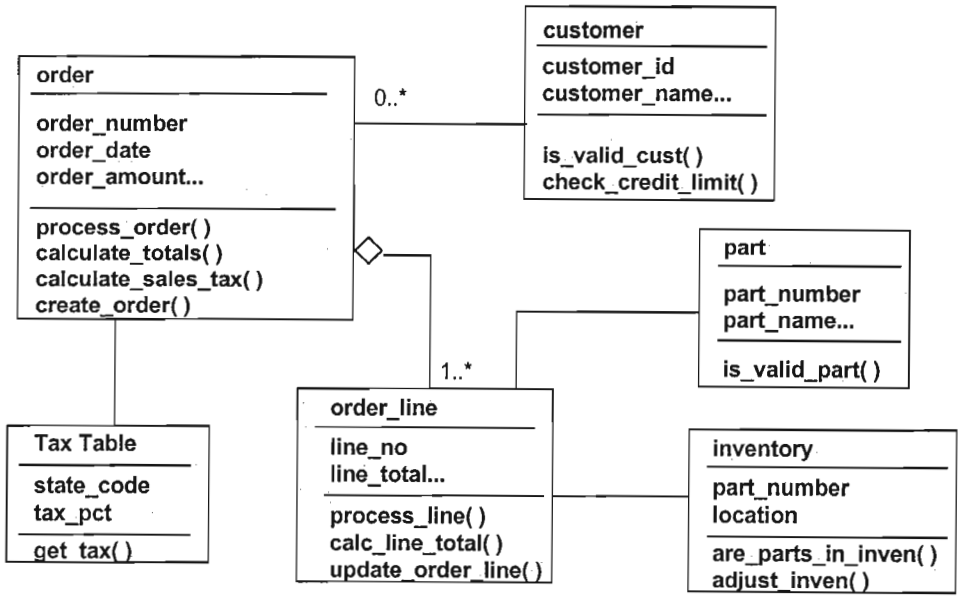
The **use case** can be used to define the interactions of a system in detail. The use case summary, actors, preconditions, postconditions, exceptions are all used to define the interaction of the system and the users. In addition, key events can be identified along with detail business responses to define the details of the interaction. Once the business responses have been identified, the business classes involved and constraints can be defined.

Use Case:	Place Order				
Summary:	A customer calls to place an order for some mail order items. The sales rep creates a new order, enters the appropriate customer information and checks to see if the items are available. If the items are available, the order is processed, calculating sales tax, shipping charges, and an order total, and the inventory of available items is decremented.				
Actors:	Customer, Sales Rep				
Preconditions:	A sales rep has access to the order entry system				
Key Event	**Detection Mechanism**	**Event Input Data**	**Business Response**	**Class Data**	**Data/Behavior Constraints**
Customer calls to place order	Sales Rep		Create new order Check for valid customer Retrieve customer data or create new customer Create order line items Check for valid items Are items in inventory? Calculate line item totals Calculate sales tax Calculate shipping charges Calculate order total	order, orderline, customer, inventory, parts	Determine sales tax calculation based on customer state

Exceptions:	***Invalid Customer:*** No customer exists with the information entered by the sales rep ***No Item Found:*** Item number entered does not match any items found, item is not valid ***Item discontinued:*** Item number entered is no longer available for purchase and cannot be ordered ***Item is out of stock***: Item is temporarily out of stock in inventory
Postconditions:	The sales rep has created a new order for the requested items

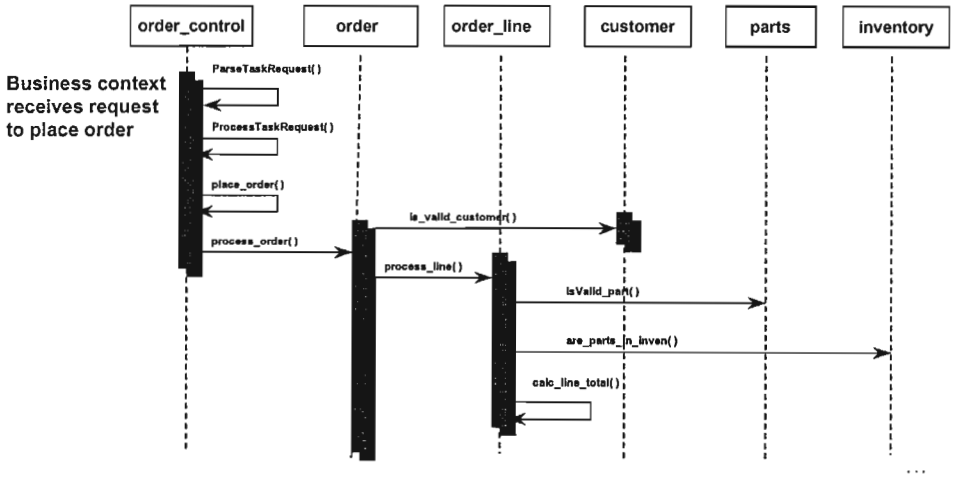
Class Diagram

The **class diagram** is a model that relates all the data and processes in a system to the entities within it. It also shows the relationships among the entities. The class diagram is the core of an object-oriented development method; it integrates processes, data, and entities.



Interaction Diagrams

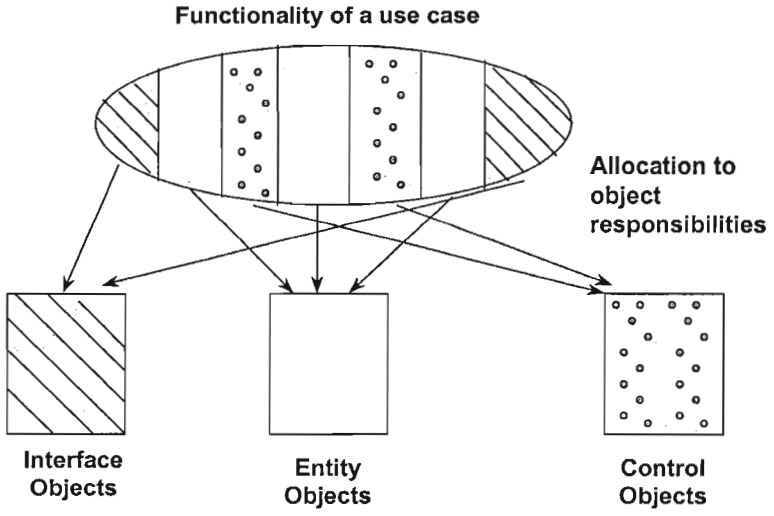
During object-oriented analysis and design, it is often helpful to model the specific interactions between classes and objects, especially when they become complex. These **interaction diagrams** let designers model the messages that objects send to and receive from each other.



Types of Objects

The analysis model in an object-oriented system represents the application's requirements through a conceptual structure that is independent of the actual implementation environment. Analysis of this structure results in the identification of three types of objects: **entity objects, interface objects** and **control objects.**

- Entity objects - model information in the system that should be held for a long time, and should typically survive a use case. All behavior naturally coupled to this information should also be placed in entity objects.
- Interface objects - model the behavior and information that is dependent on the interface to the system.
- Control objects - model the functionality that is not naturally tied to any other object.



The functionality of a use case can be translated into the three basic types of objects. As we will see as we get further into the course, these three types of objects form the basic structure from which the application architecture for the system is built.

Object-Oriented Development Goals

Object-oriented techniques are not visible to users. If you ask users whether or not a CommandButton should be inherited from a standard user object and well-encapsulated to function as desired, they don't have any idea nor should they care. They only care that the application demonstrates certain external qualities. Your goal as a developer is to make sure that your application embodies these qualities. The use of object-oriented principles and techniques can help you provide these characteristics in your applications:

- **Correctness** - The application should perform its tasks exactly as defined by the user. An integrated process of object-oriented analysis, design, and programming can help ensure correctness.

- **Robustness** - The application should continue to function properly even when used under abnormal conditions. An object-oriented system architecture can provide a stable foundation to help applications operate better under such conditions.

- **Extendibility** - The application should be easily adapted to changes in the user's specifications. An application built from modular pieces (classes and objects) can more easily be modified.

- **Reusability** - Because they often contain elements that are common across whole businesses or situations, applications should be able to be reused in whole or in part in other applications. An application built from parts that were each designed with a single, well defined purpose should be easier to divide and reuse.

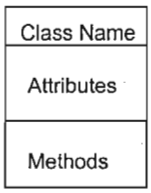
- **Compatibility** - Software products and elements should be easily combined with others to quickly solve similar business problems. Once again, parts designed with a single, well defined purpose and simple methods for connecting to each other should be more compatible.

Objects Verses Components

Object-oriented analysis, design, and programming techniques have the potential to improve the quality of software and allow development goals to be achieved. The hope of many development organizations is that the use of object technology will lead to reusable components. Component development will allow applications to be developed quickly by assembling predefined components together to build an application. It sounds good in theory but what is a component and how does it differ from objects? How can architecture-driven development and patterns be used to create components?

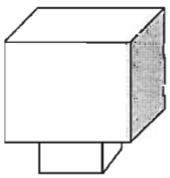
Objects encapsulate data and behavior for a single entity. Components, on the other hand, encapsulate or package one object or a combination of objects.

Object



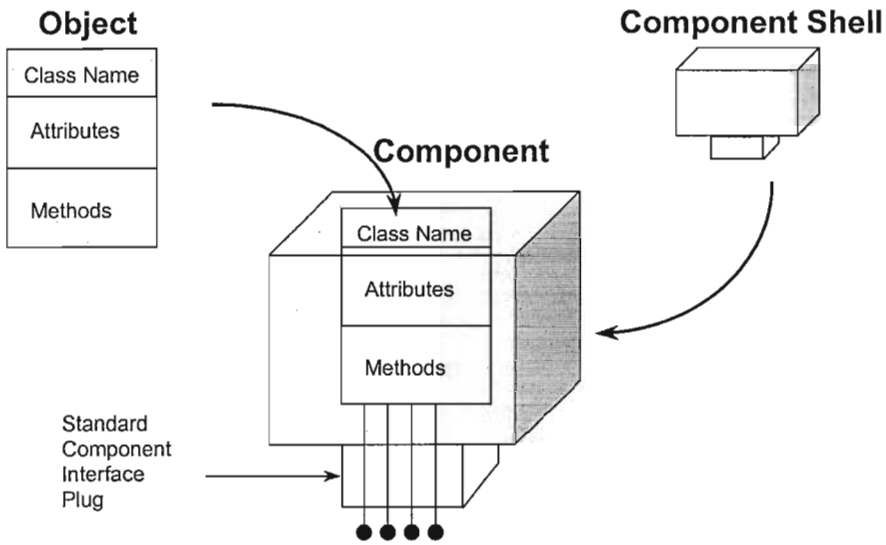
- **Encapsulates data and behavior for a single entity**
- **Are implemented using object-oriented programming languages, tools and techniques**

Component



- **Encapsulates, or packages, objects**
- **Are implemented, or manufactured, using interface definition and object binding tools**

A component is really a shell that serves as a container for one or more objects. A component has a standard interface plug that allows it to be "plugged-in" to an application.



Definition

A **component** is a reusable, self-contained piece of software that is independent of any application. It has the following characteristics:

- **Form**: binary, self-contained piece of software
- **Size**:
  - Fine: individual C++, PB, Java object
  - Medium: GUI control, output service
  - Large: applet or whole application
- **Usage**: combined and used in various ways locally or across networks
- **Interface**: manipulated only through its published interface
- **Extension**: can be extended through inheritance, aggregation, polymorphism

Why Components?

Object-Oriented Programming: Too Complex?

The move towards component development has been accelerated partly because of feelings by some that object-oriented programming is too complex. Some of the reasons that object-oriented programming is perceived to be complex include:

- Flow of control is decentralized into many collaborations, or messages, between many classes
- Fundamental building blocks are higher level abstractions
- Structural relationships can seem complicated, i.e. inheritance, aggregation, association
- Architectural relationships can be complex to implement, i.e. application partitioning
- Learning curve is long and steep
- "Any sufficiently advanced technology is indistinguishable from magic." -- *Arthur C. Clarke's 3rd Law of Technology, "Profiles of the Future: An Inquiry into the Limits of the Possible"*

Corporate Reuse Libraries: Too Complex?

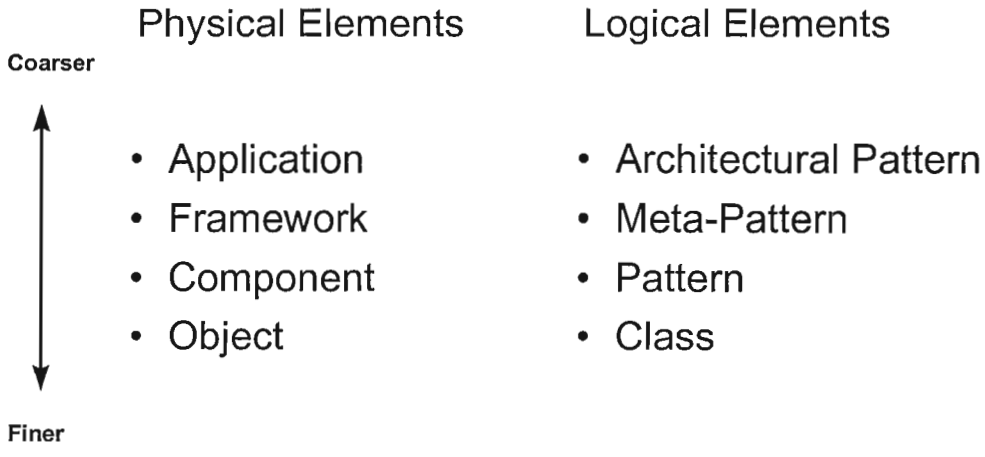
In the effort to do object-oriented development, many organizations have either developed or purchased application framework libraries. These libraries were intended to be corporate reuse libraries and provide the foundation for application development. However, some have found them difficult to use and unmanageable. Some of the reasons include:

- Requires high overhead to create it, and to maintain standards/quality
- Difficult to search through large, fine-grained libraries of classes
- Significant architectural design work still remains to combine individual classes from the library into applications
- Searches are usually done at coding time, and this is often too late to introduce reuse into a project
- Incentives often don't exist for contribution to, and reuse from the library

Alternatives to the Confusion and Complexity

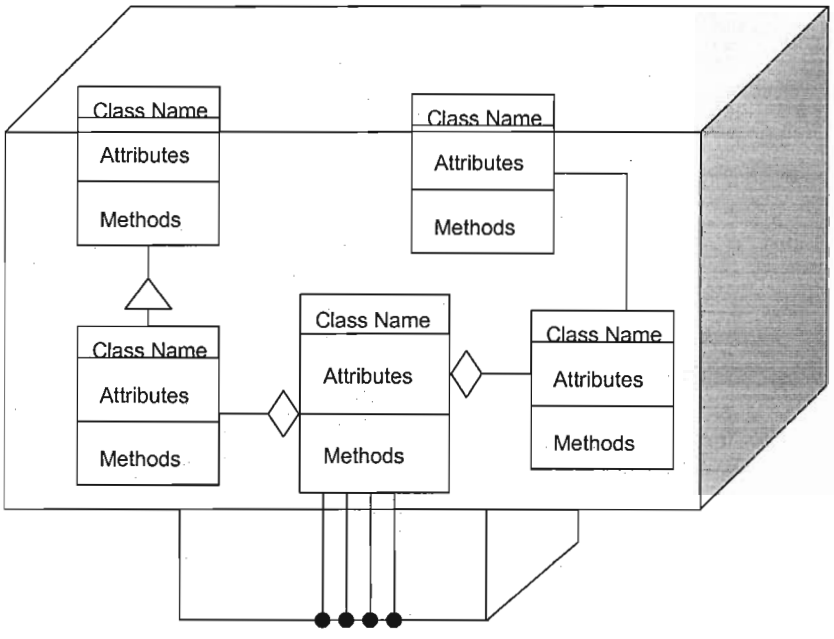
Simplify through Abstractions

One way to simplify the complexities of object-oriented development is to create larger grain reusable artifacts. The finest grained level of development is classes and objects. This is where most of the early focus was - creating reusable objects. Classes can be combined into logical combinations resulting in patterns. The physical implementation of these patterns of objects is components. Combinations of patterns are used to create meta-patterns which can result in frameworks. Finally architectural patterns can be used to build an application. Each of these levels represents a higher abstraction of an application and can simplify development.



Patterns and Components

Component development is key to the future of rapid object-oriented application development. Successful component development is dependent upon a sound foundation of objects and classes that have been created using good object-oriented analysis and design techniques. These objects can then be combined using object-oriented patterns to create components.



Summary

- The primary goal of development is to produce high-quality software that is flexible and meets the needs of users. Object-oriented techniques can improve the quality and maintainability of software.
- This course will teach you how to create an architecture-driven partitioned application.
- This course will also introduce the basic concepts required to read and utilize published patterns, as well as a process for documenting newly discovered patterns.
- Common object-oriented analysis and design deliverables include the development of use case models, class diagrams, and interaction diagrams.
- Every object in an application can be classified as either an entity object, an interface object, or a control object.
- A software component is a reusable, self-contained piece of software that is independent of any application.
- Software components are higher-level artifacts that consist of a pattern of objects. They can be used to simplify object-oriented systems.