On Software Frameworks

Introduction

These course notes intend to give a brief overview over the emerging technology of framework provision and usage. The material presented here is based on material listed in section References at the end of these notes.

Some Definitions

"A framework is a collection of classes that provide a set of services *for a particular domain*; a framework thus exports a number of individual classes and mechanisms which clients can use or adopt."

— *Grady Booch*

"A framework is a set of prefabricated software building blocks that programmers can use, extend, or customize for specific computing solutions."

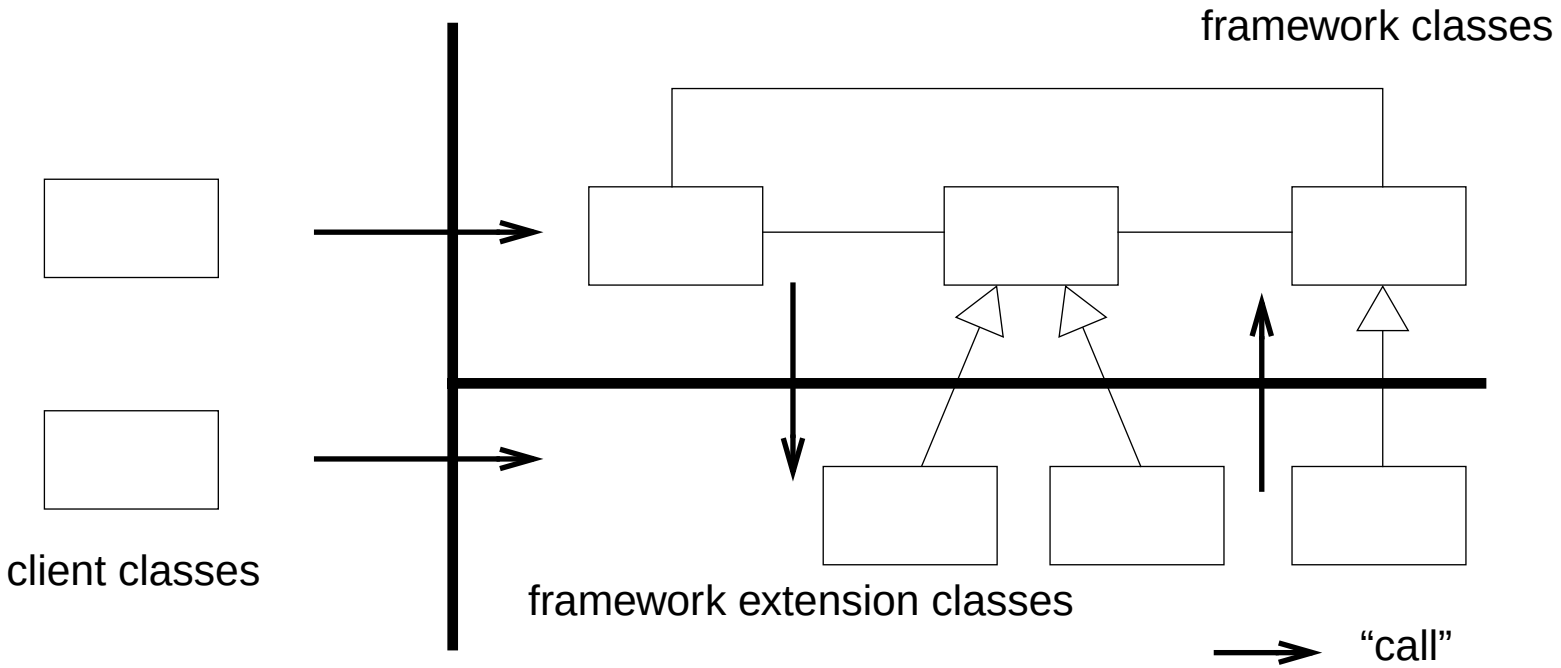
— *Taligent*

"A framework is more than a class hierarchy. It is a class hierarchy plus a model of interaction among the objects instantiated from the framework."

— *Ted Lewis*

Concepts

The next figure illustrates a framework-based application:



Client Classes

Programmers building application by using a framework write so-called *client classes*.

Framework Extension Classes

In many cases, frameworks must be specialized to meet the requirements of an application. These classes are the so-called *framework extension classes*. These classes fit into the framework through predefined extension points (also referred to as *hooks*).

Framework Classes

The classes the framework is made of.

Collaboration of Objects

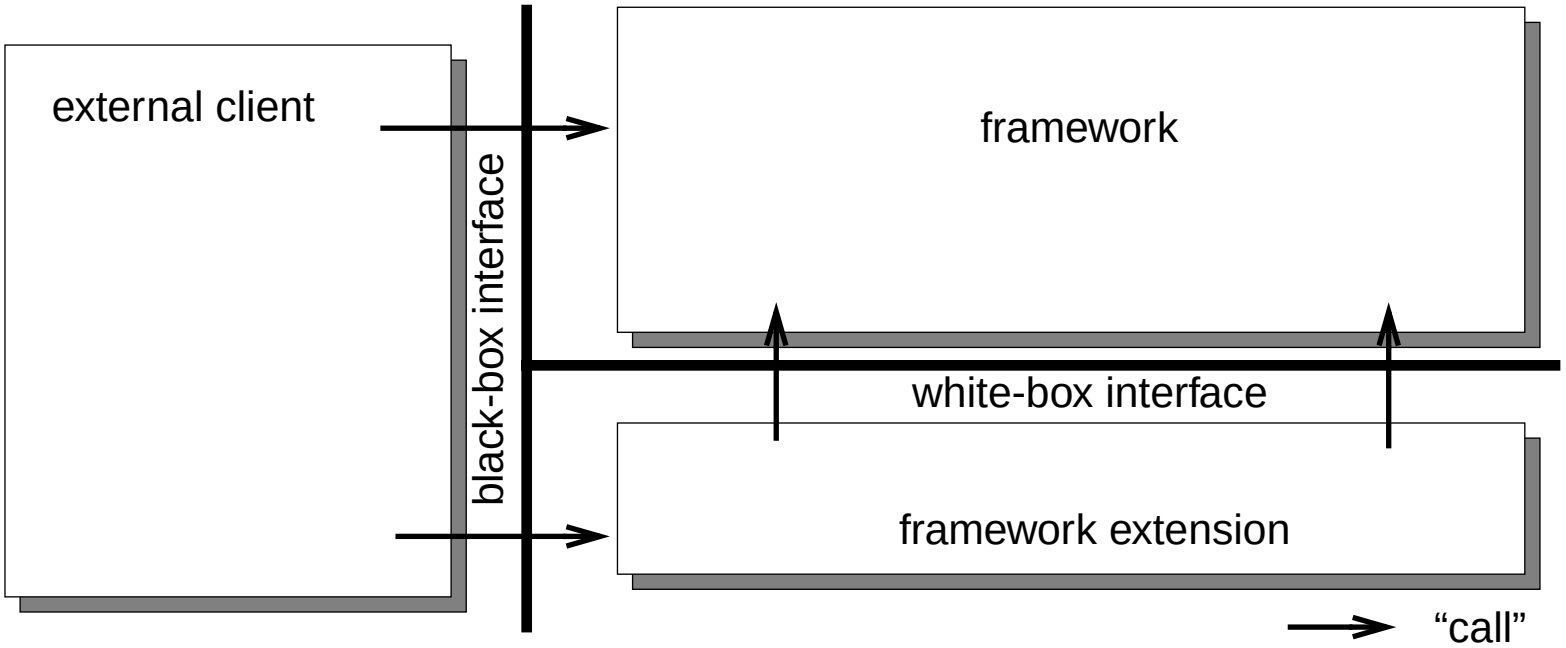
Frameworks provide a rich set of prefabricate classes and semantics for how *instances* of these classes (or subclass thereof) *interact*. A framework has some predefined *model of interaction*.

Hollywood Principle

The model of interaction defines the control of invocation of client objects. Methods of client objects are invoked according to the *“Don’t call us, we call you”* principle.

Framework Interfaces

A framework user faces the following programming interfaces:

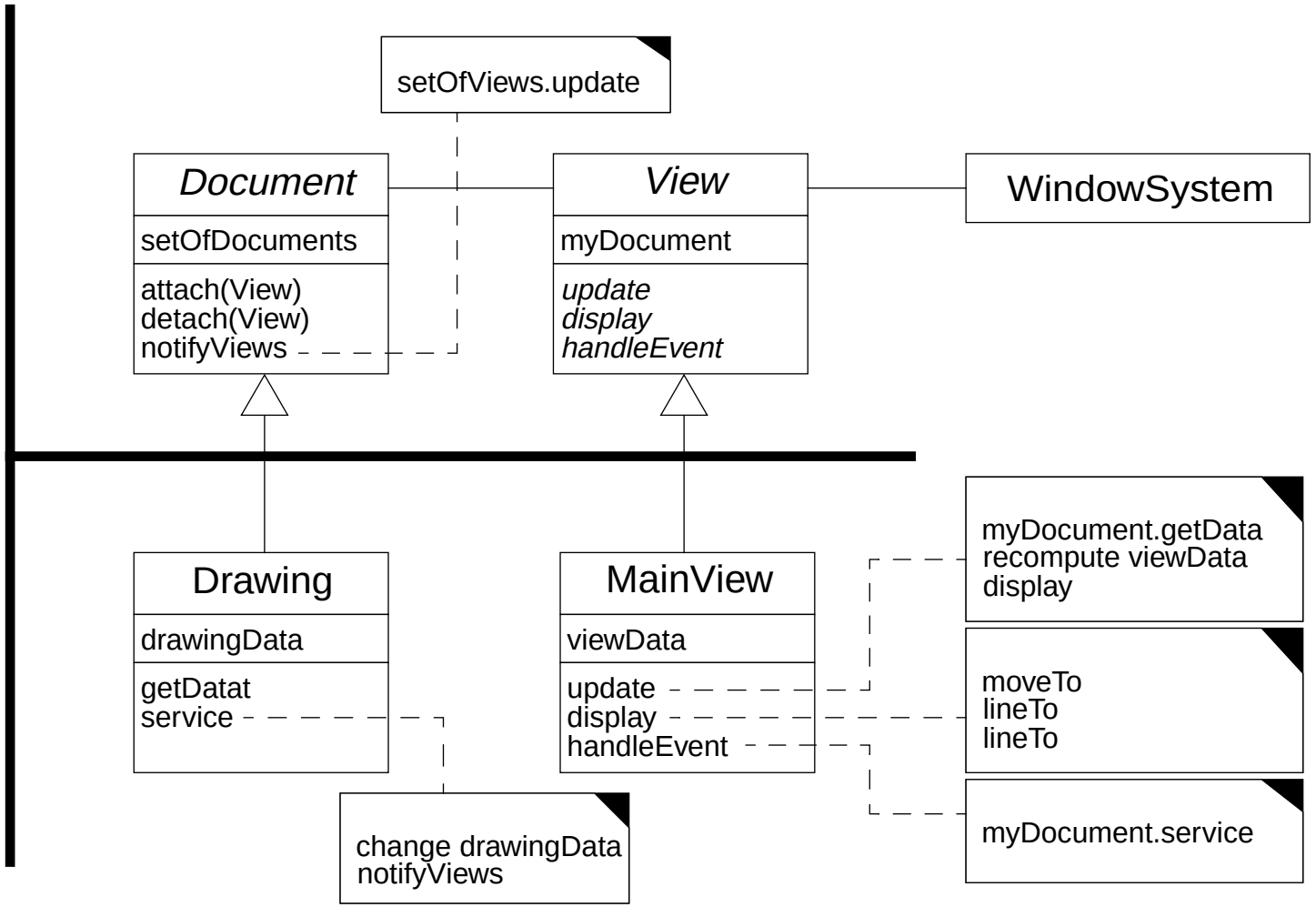


Rule of Thumb:

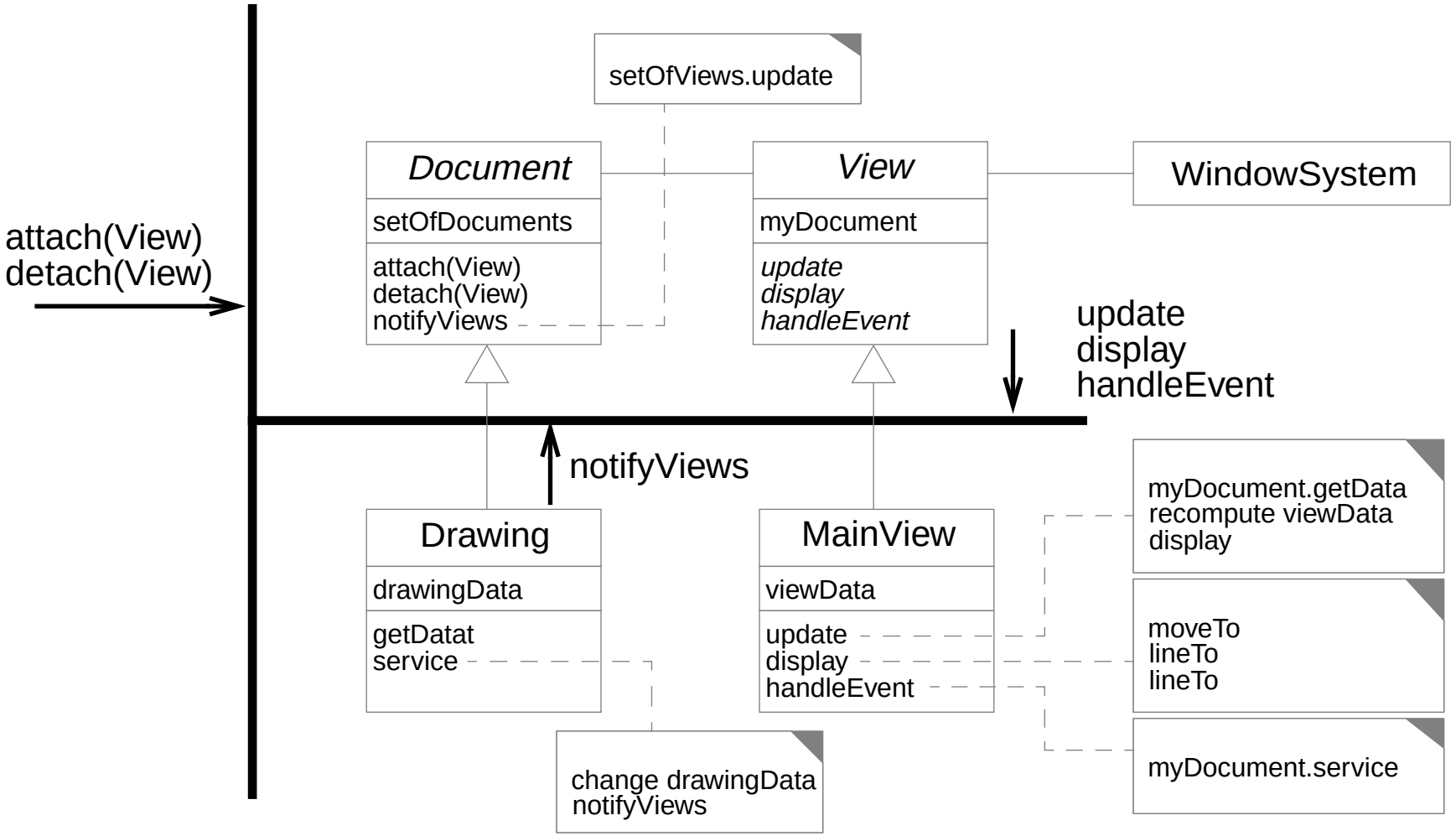
- black-box interface: typically *public* methods
- white-box interface: typically *protected* methods

A Framework Example

Classes:



Call Directions:



Class Libraries vs. Frameworks

Class Library:

- A *class library* is a collection of classes designed for general (e.g., `java.util.*`) or more specific (`java.io.*`) purposes. It's up to the application programmer to decide how to use objects of these classes.

Framework:

- Frameworks, on the other hand, consists of a prefabricated software architecture, using a predefined collaboration model, into which the application programmer must *plug in implementations at specific locations*.

- Frameworks themselves consists of a series of patterns. Patterns that suit well for the provision of hooks are for example:
  - template method
  - factory method
  - abstract factory
  - strategy
  - ...

Types of Frameworks

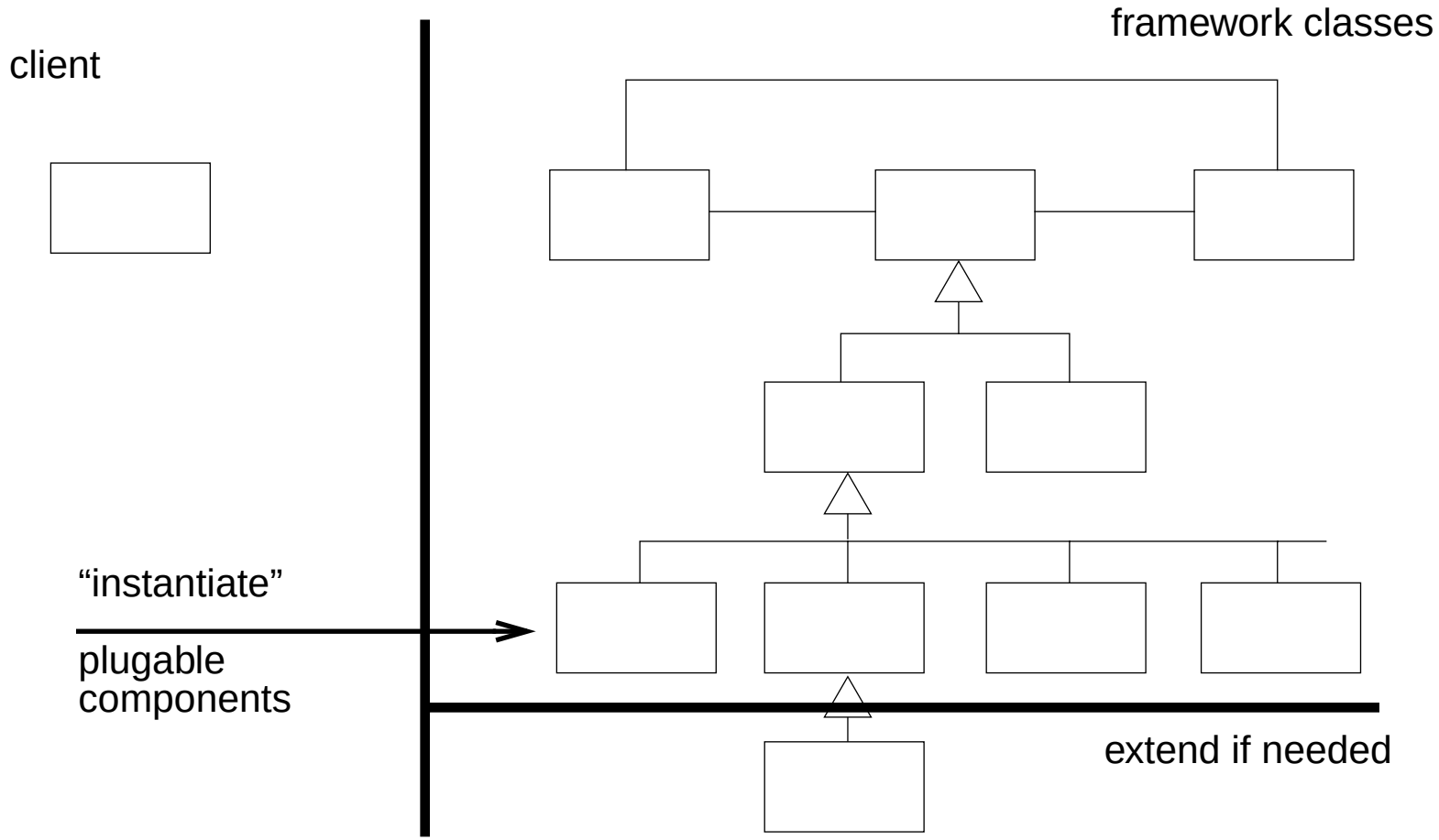
The granularity of frameworks can vary. There are frameworks addressing specific programming problems (e.g., object persistence, provision of naming services). Others directly address problem domains (e.g., book-keeping, inventory).

Thus, one can distinguish among several types of frameworks. For example:

- domain frameworks (e.g. for financial services)
- application frameworks (e.g. for GUI applications)
- utility frameworks (e.g. server skeletons, object persistence)

Framework with Pre-Built Components

Often, a framework has “pre-built” components. Framework users instantiate and use them. Alternatively, a pre-built component can even be more specialized via extension, if needed:



Benefit of Frameworks

Problem decomposition and finding *good, flexible structures* that solve the given problem, are the most challenging aspect in system design.

With the provision of a framework, it is this structural aspect that has already been addressed in such a way that the framework not only addresses the problem well, but facilitates *reusability*, *extensibility*, and *scalability*.

Framework Design Issues

When designing and implementing frameworks, the framework designer must consider the following aspects:

Abstract Classes

Provide reusable, meaningful abstractions. Determine anticipated behavior (i.e., the common semantic) in terms of abstract or concrete methods. Subclasses should be true specializations in terms of IS-A relationship.

Separation of Interfaces from Implementations

Separate types of objects from their concrete implementations. Allow the provision of alternate implementations.

Delegate Responsibilities

Delegate non-business related responsibilities of domain objects away from the domain classes.

Use of Design Patterns

Various design patterns help you to achieve flexible and extensible framework architectures.

- **Factory Method**
  Using the same creation process allows subclasses of an abstraction to provide their own object instantiations.
- **Abstract Factory**
  An abstraction whose subclasses implement a common protocol for creating families of related objects.
- **Bridge**
  Decouples the implementation of an abstraction from the hierarchy in which the abstraction resides, thereby allowing the two to vary independently.
- **Builder**
  Decouples the representation of complex objects from its construction process.
- **Observer/MVC/Java's event model**
  Defines uniform ways how objects that change state can inform interested parties.
- **Template Method**
  Defines the skeleton of an algorithm in a method, deferring some steps to subclasses provided by framework client programmers.
- **Strategy**
  Defines a family of algorithms, and let the algorithms vary independently from the framework that use it.

More on Framework Design Principles

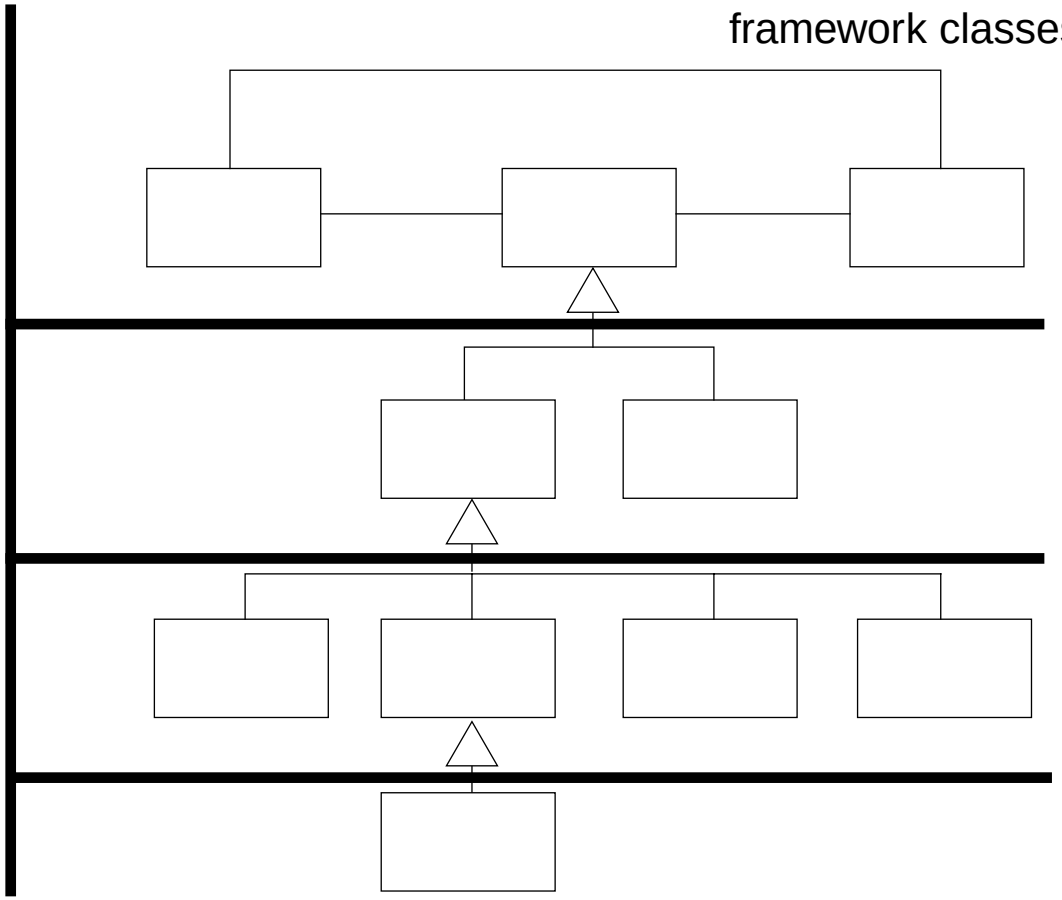
Process of Abstraction

"An abstraction is usually discovered by generalizing from a number of concrete examples." [Johnson & Foote, 1988]

"Useful abstractions are usually created by programmers with an obsession for simplicity, who are willing to rewrite code several times to produce easy-to-understand and easy-to-specialize classes." [Kent Beck, 1986]

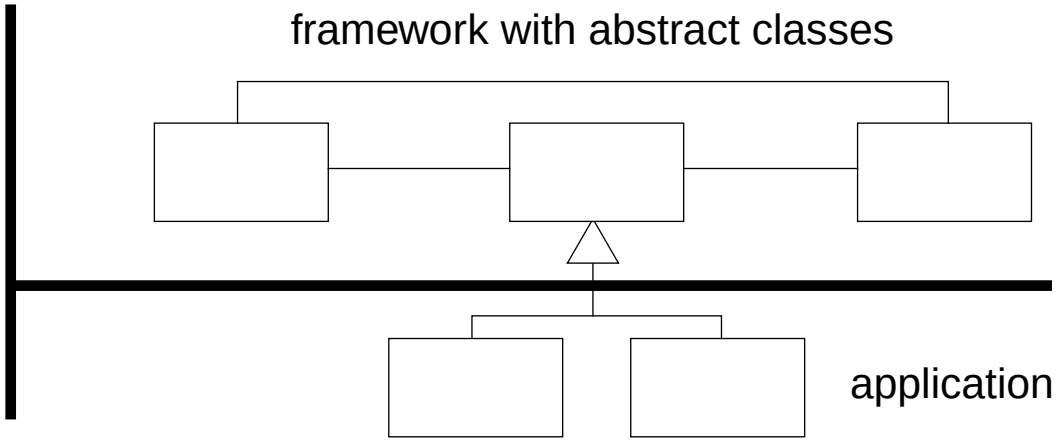
Layered Frameworks

client

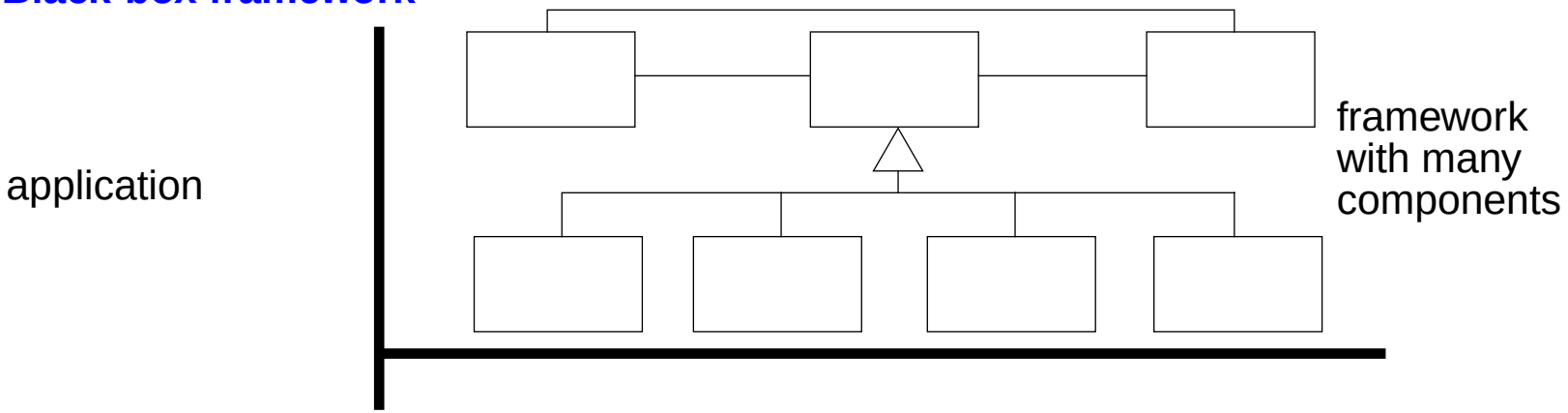


White-Box vs. Black-Box Frameworks

White-box framework



Black-box framework



Language Support for Reusable Code

Separate:

- Specialization (inheritance) hierarchy
- Call-compatibility (interface) mechanism

Solution (Java):

- Protocol = Interface =
  List of method names with signatures (argument types and result types)

- Allow protocols to be arranged in a (multiple-) inheritance hierarchy
  – problems with multiple inheritance (name clashes, duplicate inheritance) are avoided

Abstract Classes

A class not intended for instantiation

- usually incomplete: specifications of operations, but no implementation
- can provide default implementations
  (which may or may not be overwritten in subclasses)

Interface Slogans

- “Program to an interface, not an implementation” [GoF]
- “[Implementation] Inheritance is not subtyping” [Cook and others]

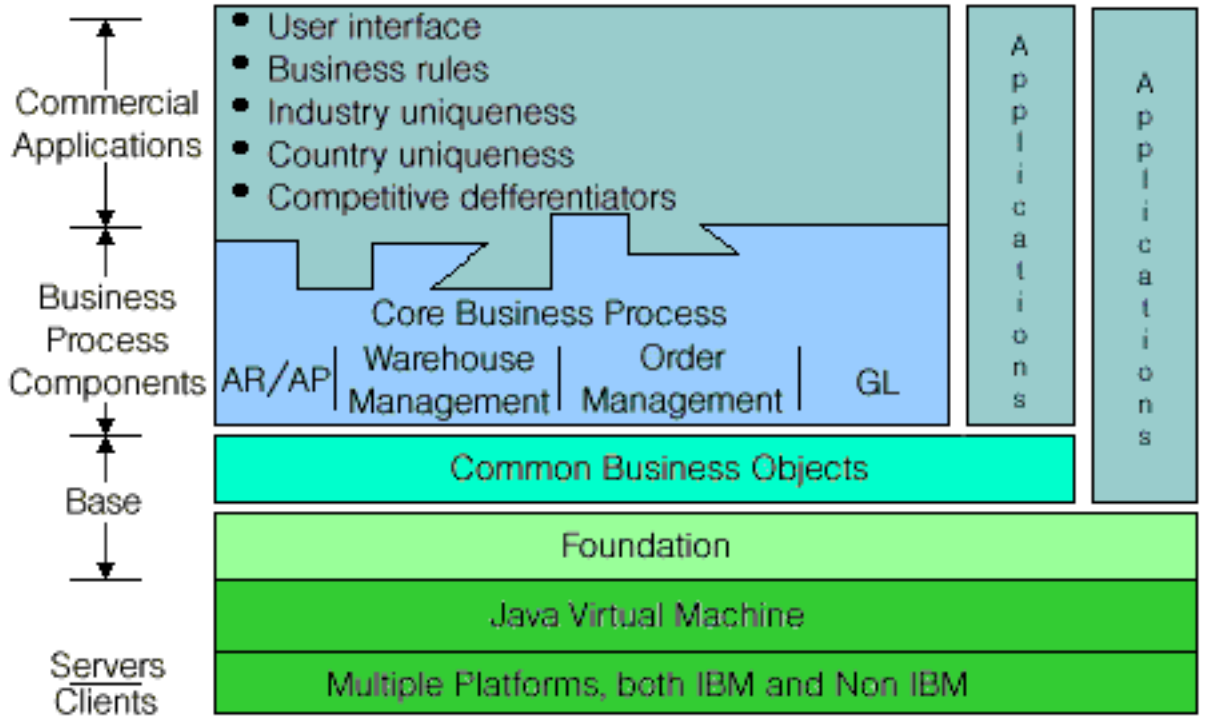
Some General Guidelines for OO Design

- Don’t put too much into a class. A class should represent a well-understood abstraction.
- Don’t overuse implementation (class) inheritance. Composition is usually more flexible.
- Implementation (class) inheritance should be used for specialization, not code reuse
- Program to an interface, not an implementation

Framework Examples

Domain Frameworks

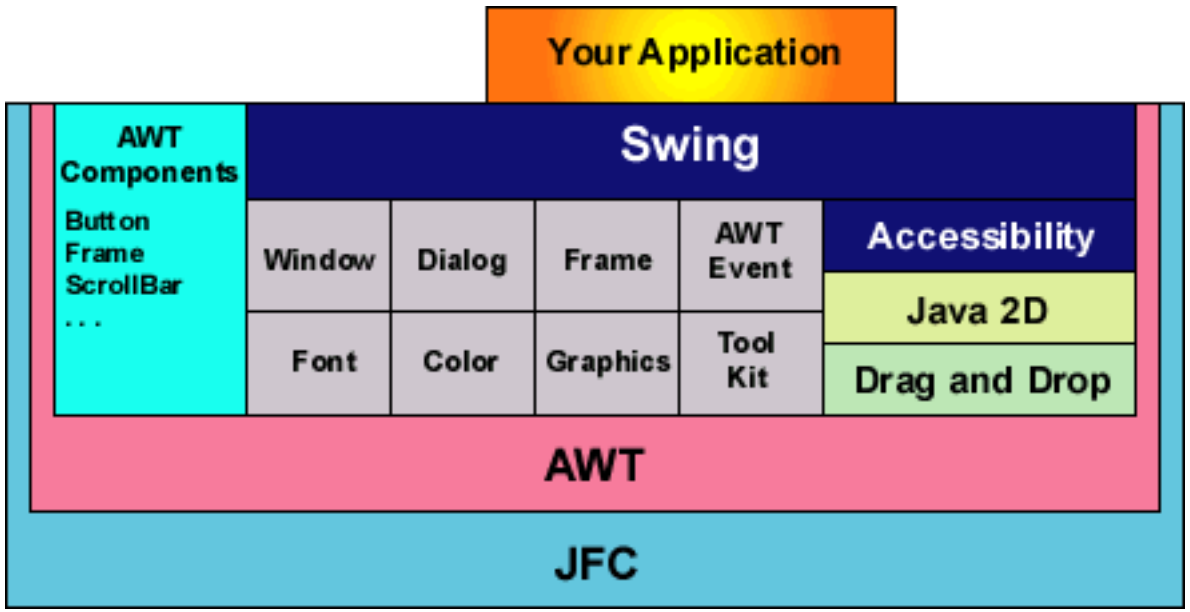
- San Francisco



From: http://www-4.ibm.com/software/ad/sanfrancisco/prd_summary.html

Application Frameworks

- Swing



http://java.sun.com/products/jfc/tsc/articles/getting_started/index.html

- InfoBus

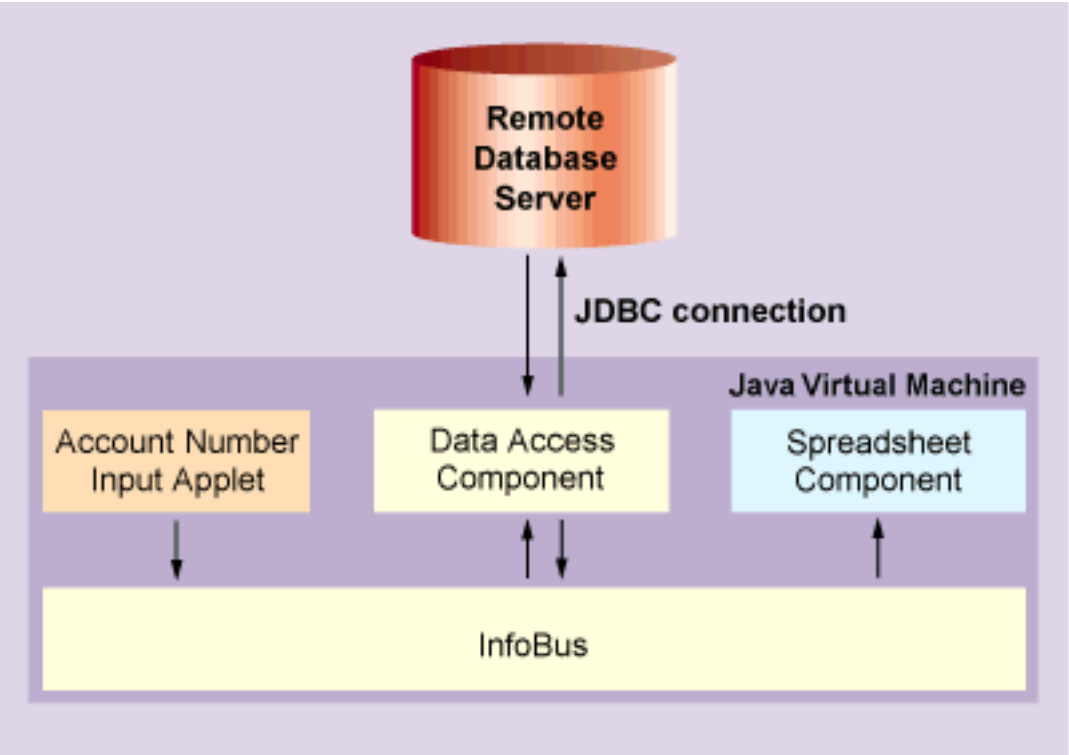
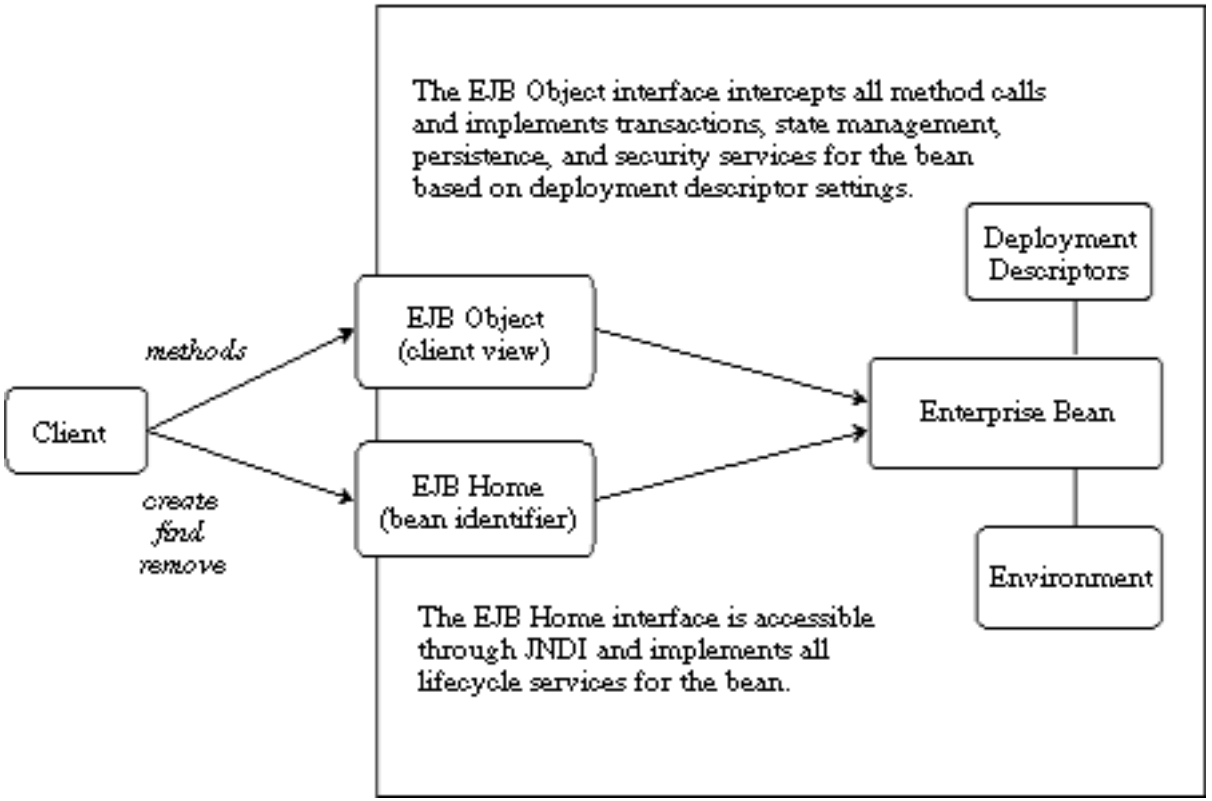


Figure 2. InfoBus Handles Data Flows. From a logical data flow perspective, our composite application consists of a series of three applets. The applets are connected according to HTML statements that specify, for each applet, the name(s) of the data items handled.

From: http://www.java-pro.com/archives/1998/jp_febmar_98/mc0298/mc0298.htm

- Enterprise JavaBeans



From: http://java.sun.com/products/ejb/white/white_paper.html

- CORBA Framework:

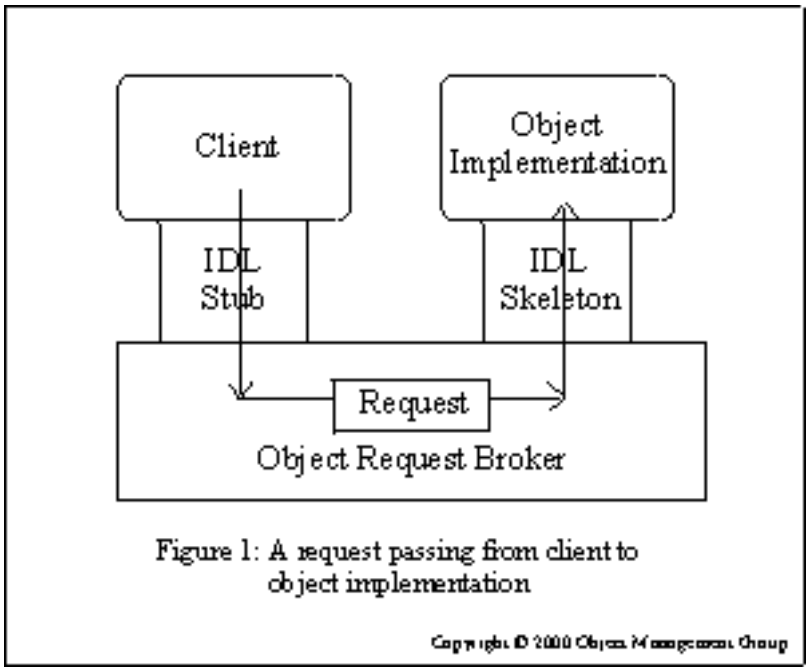
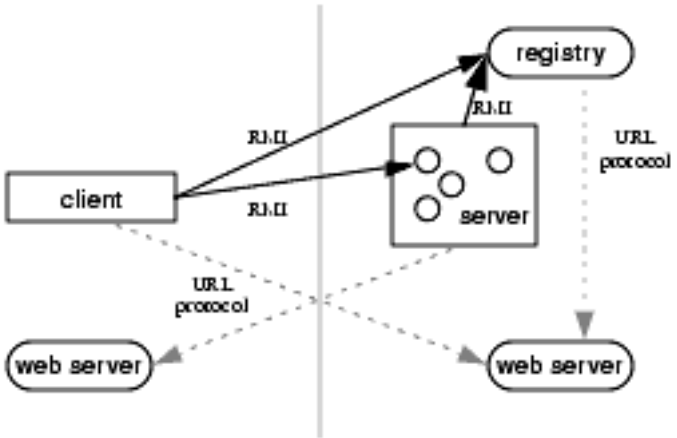


Figure 1: A request passing from client to object implementation

From: http://sisyphus.omg.org/gettingstarted/corbafaq.htm

- RMI Framework:



From: http://java.sun.com/j2se/1.3/docs/guide/rmi/spec/rmi-objmodel2.html

Utility Frameworks

- JUnit

References

1. D. Govani, “Java Application Frameworks”, Wiley 1999, ISBN 0-471-32930-4.
2. M.E. Fayad et al., “Implementing Application Frameworks”, Wiley, ISBN 0-471-25201-8.
3. K. Beck et al., “JUnit: A Cook’s Tour”, http://junit.sourceforge.net/doc/cookstour/cookstour.htm.