软件设计模式(一)

潘爱民

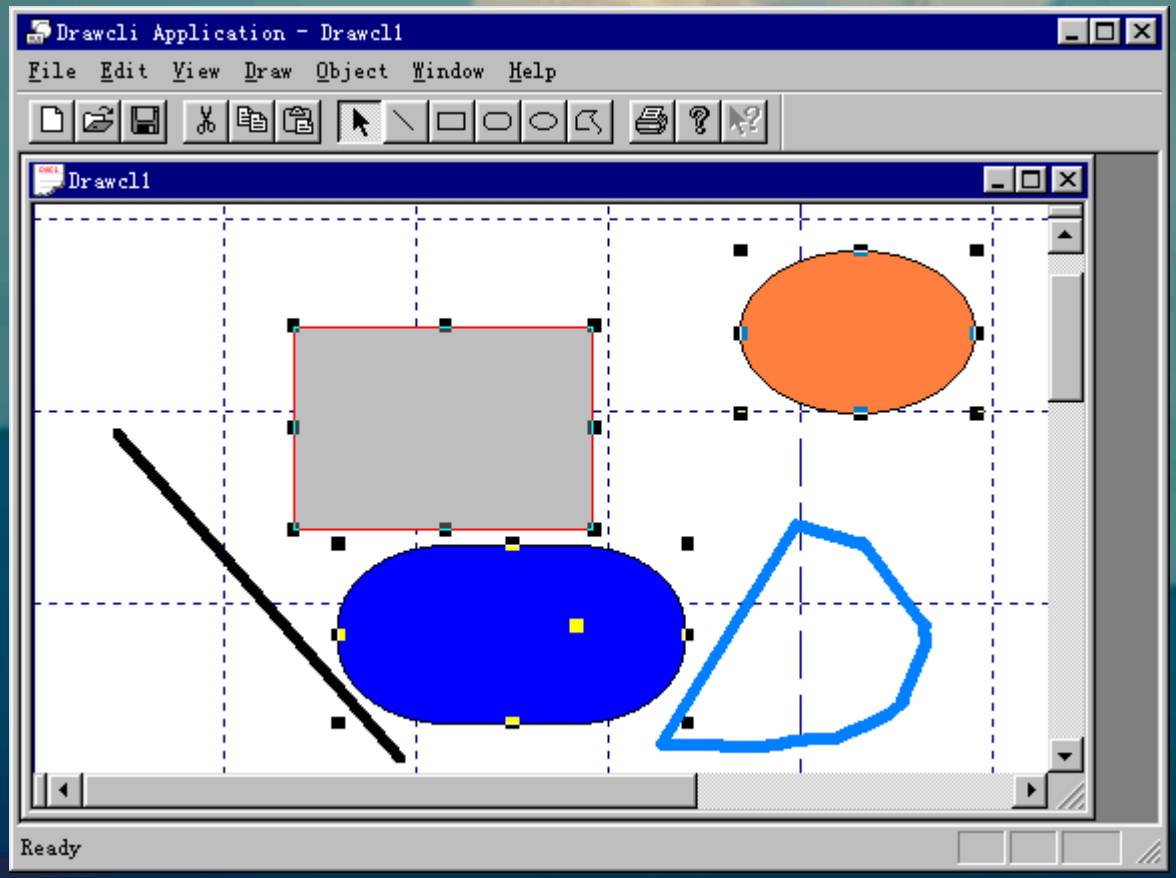
http://www.icst.pku.edu.cn/compcourse

内容

- 从一个例子看模式
- 关于模式的研究情况
- 介绍一些重要的模式(部分)

一个设计例子

- VC/Samples/MFC/OLE/DrawCli



DrawCli的基础

- MFC提供的基础
  - Doc/View结构
    - Doc提供了强大的数据管理功能
    - View提供了强大的显示功能
    - splitwnd功能
  - CWinApp/CMainFrame
    - 提供了一套命令处理流程
  - 对OLE的封装
    - Active Container
    - OLE Clipboard
    - Property Page
  - 数据结构管理功能
    - CObject/CObList

我们的设计焦点

- 用C++对象来表示每一个图元
  - 定义图元的公共接口
- 如何处理用户的操作
  - 鼠标的动作
- 图元对象的创建和管理

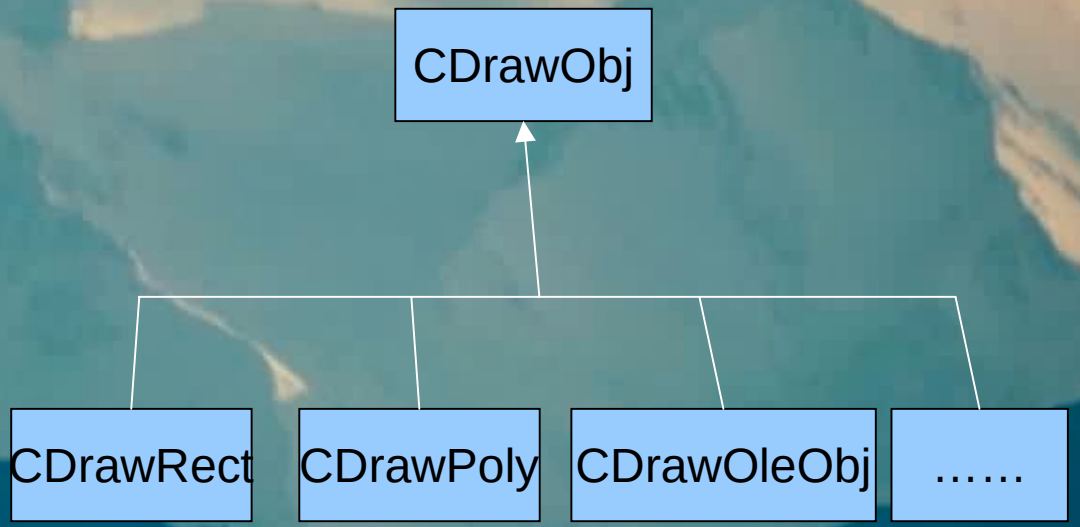
图元基类

```
class CDrawObj : public CObject
{
// Attributes
        CDrawDoc* m_pDocument;       // owner

        virtual int GetHandleCount();
        virtual CPoint GetHandle(int nHandle);
        virtual HCURSOR GetHandleCursor(int nHandle);
        virtual void SetLineColor(COLORREF color);
        virtual void SetFillColor(COLORREF color);
// Operations
        virtual void Draw(CDC* pDC);
        virtual void DrawTracker(CDC* pDC, TrackerState state);
        virtual void MoveTo(const CRect& positon, CDrawView* pView = NULL);
        virtual int HitTest(CPoint point, CDrawView* pView, BOOL bSelected);
        virtual BOOL Intersects(const CRect& rect);
        virtual void MoveHandleTo(int nHandle, CPoint point, CDrawView* pView = NULL);
        virtual void OnOpen(CDrawView* pView);
        virtual void OnEditProperties();
        virtual CDrawObj* Clone(CDrawDoc* pDoc = NULL);
        virtual void Remove();
        virtual void Serialize(CArchive& ar);
// ……
};
```

图元层次结构

- CDrawObj
- CDrawRect
- CDrawPoly
- CDrawOleObj
- ……



图元的创建

- 永久支持：
  - CDocument::Serialize->CObList::Serialize ->CObject::Serialize
- CDrawObj::Clone
- DrawTool：创建图元对象

交互操作

- 键盘和菜单命令
  - MFC内部机制
- 鼠标操作
  - 在CDrawView的鼠标处理函数中处理
  - 工具箱：工具链，有一个当前活动工具
  - 转交给当前活动工具来处理

工具箱和CDrawTool

```
class CDrawTool
{
// Constructors
public:
        CDrawTool(DrawShape nDrawShape);

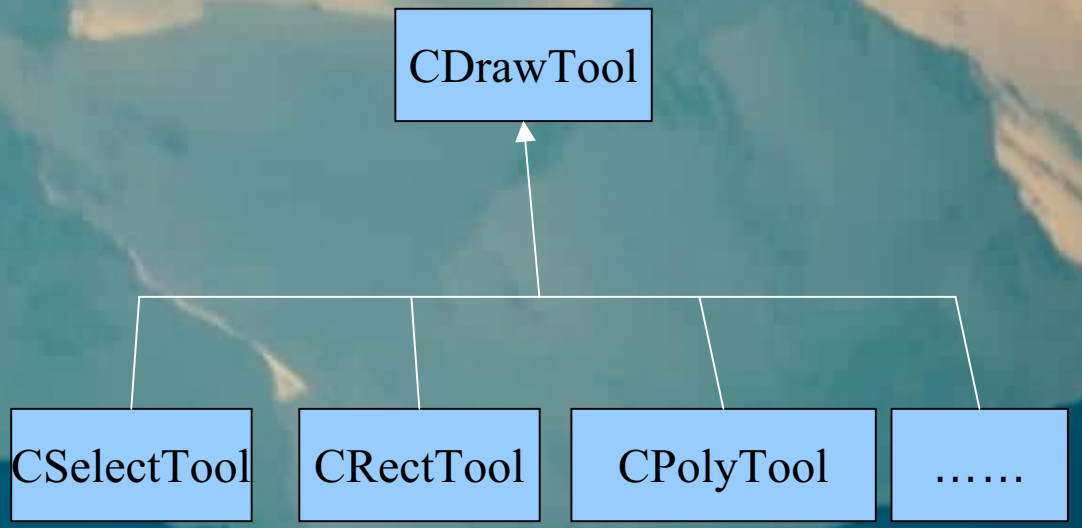
// Overridables
        virtual void OnLButtonDown(CDrawView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonDblClk(CDrawView* pView, UINT nFlags, const CPoint& point);
        virtual void OnLButtonUp(CDrawView* pView, UINT nFlags, const CPoint& point);
        virtual void OnMouseMove(CDrawView* pView, UINT nFlags, const CPoint& point);
        virtual void OnEditProperties(CDrawView* pView);
        virtual void OnCancel();

// Attributes
        DrawShape m_drawShape;

        static CDrawTool* FindTool(DrawShape drawShape);
        static CPtrList c_tools;
        static DrawShape c_drawShape;
// ……
};
```

DrawTool层次结构

- CDrawTool
- CSelectTool
- CRectTool
- CPolyTool
- ……



例子中的模式

- CDrawObj和CDrawTool合起来构成了Factory Method模式
- CDrawTool::Clone用到了原型创建模式
- 每一个CDrawTool都是一个singleton
- Adapter模式：把OLE对象封装成CDrawObj
- * 可以增加Composite模式
- facade模式：通过CDrawDoc/View与MFC通信
- chain of responsibility：如鼠标处理工作
- * 用Command模式增加undo/redo功能

patterns

- 定义：
  - A physical arrangement of elements
  - Repeating;with some degree of correspondence in successive trials or observations
  - 典范、范例，事物的标准样式
  - In the book “Design Patterns”, the design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context
  - Design patterns represent solutions to problems that arise when developing software within a particular context

- 几个近义词
  - idiom、technique、paradigm

pattern与framework

- Patterns支持软件结构和设计的重用
  - 抓住了特定领域中问题的成功解决方案中的静态、动态结构和相互之间的协作关系
  - patterns与开发语言无关，但是建立在一定的环境基础上
  - 例如：经典的MVC、Factory Method
- frameworks支持细节设计和代码的重用
  - framework是一组组件的综合，这些组件相互协作，为一族相关应用提供了一个可重用的框架结构
  - 例如：MMC、MS Script Engine
- 两者结合起来， design patterns and frameworks有助于提高软件的质量
  - 比如：重用性，扩展性，性能，可维护性

Design pattern与framework(续)

- a framework supplies the infrastructure and mechanisms that execute a policy for interaction between abstract components with open implementations.
- frameworks are often said to abide by the Hollywood Principle ("Don't call us, we'll call you.")
- 比较：
  - Design patterns are more abstract than frameworks
  - Design patterns are smaller architectural elements than frameworks
  - Design patterns are less specialized than frameworks
- framework与class library

Pattern的研究情况

- 关于pattern研究的历史
  - A Pattern Language，Christopher Alexander，1977
  - “Advanced C++：Programming Styles and Idioms”，James Coplien，1992
  - “Design Patterns: Elements of Reusable Object-Oriented Software”，GOF，1995
  - “Pattern-Oriented Software Architecture: A System of Patterns” (简称为“POSA”)，GoV，1996
  - ……

Pattern的研究现状

- pattern与Java
- pattern与CORBA
- pattern与系统结构
- pattern与generic programming结合
- 其他

POSA中的模式分类

- Architectural Patterns
  - 表达了软件系统的基本结构组织形式或者结构方案。
  - 它包含一组预定义的子系统，规定了这些子系统的责任，同时还提供了用于组织和管理这些子系统的规则和向导。
- Design Patterns
  - 为软件系统的子系统、组件或者组件之间的关系提供一个精炼之后的解决方案。
  - 它描述了在特定环境下，用于解决通用软件设计问题的组件以及这些组件相互通信时的可重现结构。
- Idioms
  - 是一个与编程语言相关的低级模式。
  - 它描述了如何实现组件的某些功能，或者利用编程语言的特性来实现组件内部要素之间的通信功能。

POSA: Architectural Patterns(1)

- Architectural Patterns are very high-level structural patterns.
- Also called “Conceptual Patterns ”
- From Mud to Structure: Organize components.
  - Layers: Organize components into layers where layer i's services are only used by layer i+1.
  - Pipes and Filters: Divide the task into several sequential processing steps -- the output of task i is the input of task i+1.
  - Blackboard: Several independent programs work cooperatively on a common data structure.
- Distributed Systems :Handle distributed computation.
  - Broker: Introduce a broker component to to achieve better decoupling of clients and servers -- brokers accept requests from clients and forward the requests to servers, then return the results back to the clients.

POSA: Architectural Patterns(2)

- Interactive Systems :Keep a program's functional core independent of the user interface.
  - Model - View - Controller: Divides the application into processing, output, and input. View and controller parts are usually observers of the model via the observer pattern.
  - Presentation - Abstract - Control: Divides the application up to heirarchies or MVC-like components. Each component is dependent upon and and provides functionality for the a higher-level component. There is only one top-level component.
- Adaptable Systems : Design for change.
  - Microkernel Encapsulate the fundamental services of the application.
  - Reflection Divide the application into a meta-level and a base level to make the application "self-aware". The meta level encapsulates knowledge of the system; the base level encapsulates knowledge about the problem domain.

POSA: Design Patterns(1)

- Structure Decomposition: Decompose subsystems and complex components into cooperating parts.
  - Whole - Part: Define a component that encapsulates smaller objects. Prevent clients from directly accessing the contained objects, but provide a interface for the aggregate.
- Organization of Work : Components collaborate to solve complex problems.
  - Master - Slave The master divides a task among identical (but independent) slaves, the combines the slave's partial results to arrive a solution.
- Access Control : Guard and control access to services and components.
  - Proxy: Clients communicate with a representative (proxy) rather than the target object itself. The proxy can perform pre- and post-processing to provide validation checking, access control, remote object access, extra computation, etc. See also Gamma et al's proxy pattern

POSA: Design Patterns(2)

- Management: Handle homogenous collections of objects, services and components in their entirety.
  - Command Processor: Extends Gamma et al's command pattern by adding an explicit command processor
  - View Handler: Separate the the management of views from the code required to present or control specific views. Similar to Gamma et al's Abstract Factory and Mediator.
- Communication :Organize communication between components.
  - Forward - Receiver: Contain all system-specific communication functionality in separate components so distributed peers can communicate without loosing portability
  - Client - Dispatcher - Server: A *dispatcher* acts as an intermediate layer between clients and servers. The dispatcher provides the communication channel and a name service to hide physical locations.
  - Publisher - Subscriber: Same as Gamma et al's Observer pattern.

Idioms

- Also called “Programming Patterns”
- Idioms are low-level patterns specific to a programming language.
  - Counted Pointer: Simplifies memory management of shared objects providing reference counting.
- 其他
  - Virtual constructor
  - smart pointer
  - handle/body

关于“Design Pattern”

- 对已有模式的整理、分类
- 一套描述模式的词汇，可用于交流和文档化
- 为软件设计总结了宝贵的经验，这些设计经验可以被重用，但不是简单的代码重用
- 分类：
  - Creational Patterns
  - Structural Patterns
  - Behavioral Patterns
- 在软件设计模式领域，起到先驱的作用

重提：指导模式设计的三个概念

- 重用(reuse)：是目标
- 接口与实现分离
- decouple

如何描述一个模式

- 关键要素
  - Design pattern name，Aliases or Also Known As
  - Problem，Intent or Goal
  - Forces，Constraints，Motivation
  - Context， Applicability
  - Solution
    - Structure
    - Participants
    - Collaboration
    - Implementation
  - Evaluation，Resulting Context，Consequences
  - Related Patterns
  - Examples，Known uses

creational patterns

- Abstract Factory(kit)
- Builder
- Factory Method(virtual constructor)
- Prototype
- Singleton
- * Finder

模式 1：Factory Method (一)

- Aliases：virtual constructor
- Intent
  - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Motivation
  - Frameworks use abstract classes to define and maintain relationships between objects. A framework is often responsible for creating these objects as well.

Factory Method模式(二)

- Applicability： Use the Factory Method pattern when
  - a class can't anticipate the class of objects it must create.
  - a class wants its subclasses to specify the objects it creates.
  - classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate.

插：virtual constructor

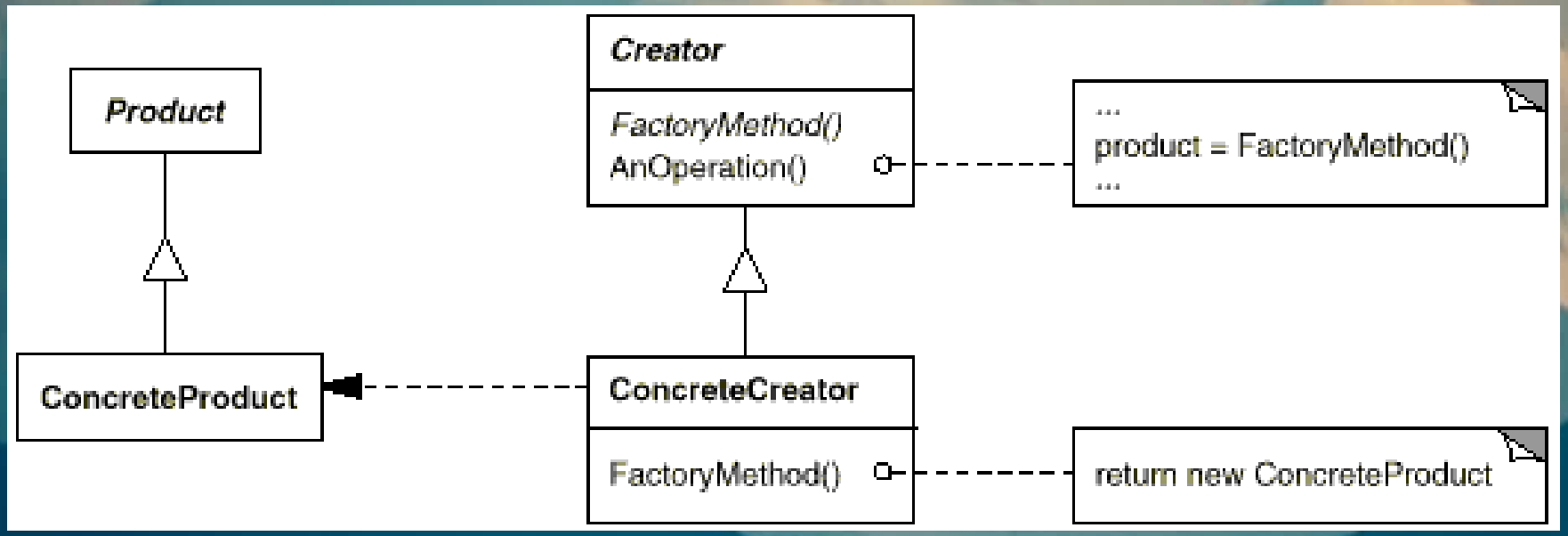
- intent：
  - 在一个类层次中，客户在runtime要创建一个对象，对象的子类型未确定，根据客户的需要和环境情况，确定对象的类型
- problem：
  - 只知道对象的一般类型，不知道确切类型(需要从环境中获取类型信息)
- Forces ：
  - 隐藏对象的类型层次，只发布基接口
  - 如何确定最合适的派生类
  - 客户必须有办法使用派生类的服务

插：virtual constructor(续)

- solution：
  - 使用Envelope/Letter或者Handle/Body pattern
  - 由envelope或者handle根据环境信息选择适当的派生类型
- 例子：
  - 1 根据stream动态创建对象
  - 2 COM对象

Factory Method模式(三)

- struct



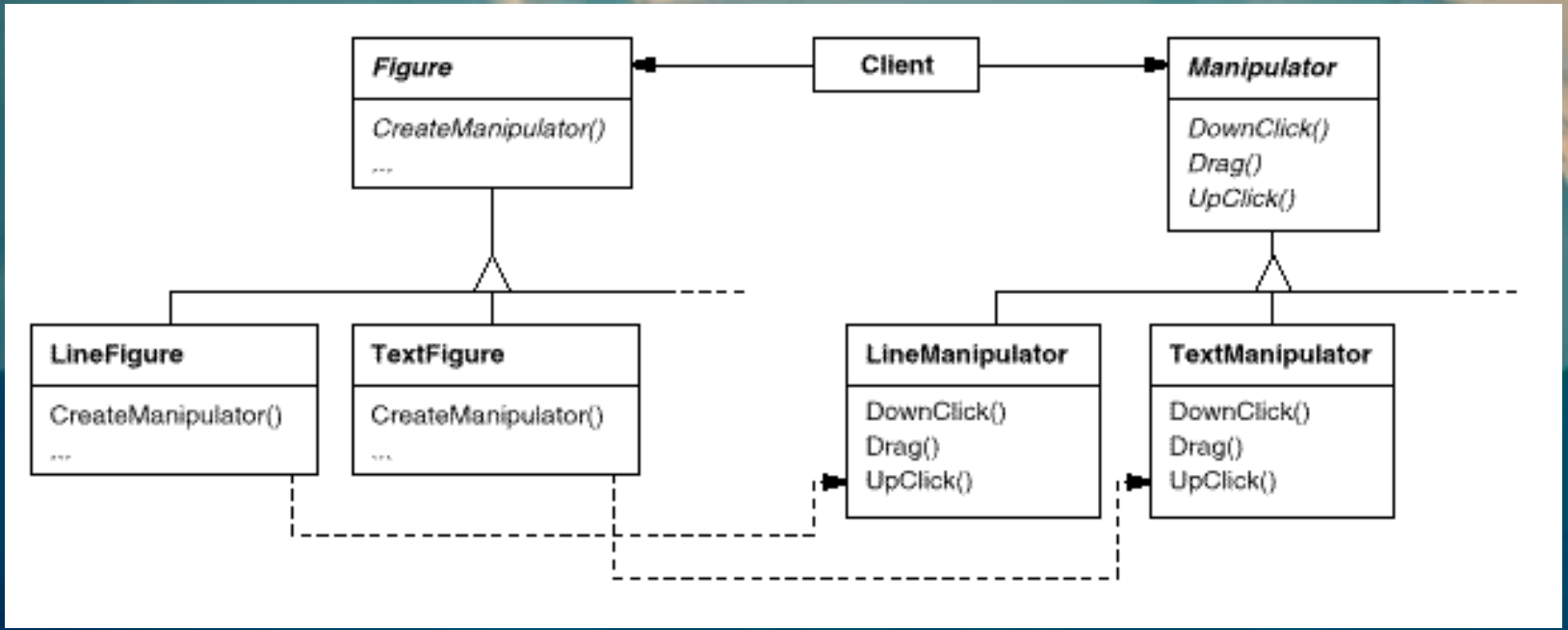
- Participants
  - Product、ConcreteProduct、Creator、ConcreteCreator
- Collaborations

Factory Method模式(四)

- Evaluation
  - 多态性：客户代码可以做到与特定应用无关，适用于任何实体类
  - 缺点：需要Creator和相应的子类作为factory method的载体，如果应用模型确实需要creator和子类存在，则很好；否则的话，需要增加一个类层次
  - 优点：
    - (1) Provides hooks for subclasses。基类为factory method提供缺省实现，子类可以重载新的实现，也可以继承父类的实现。体现了：加一层间接性，增加了灵活性
    - (2) Connects parallel class hierarchies

Factory Method模式(五)

- Connects parallel class hierarchies

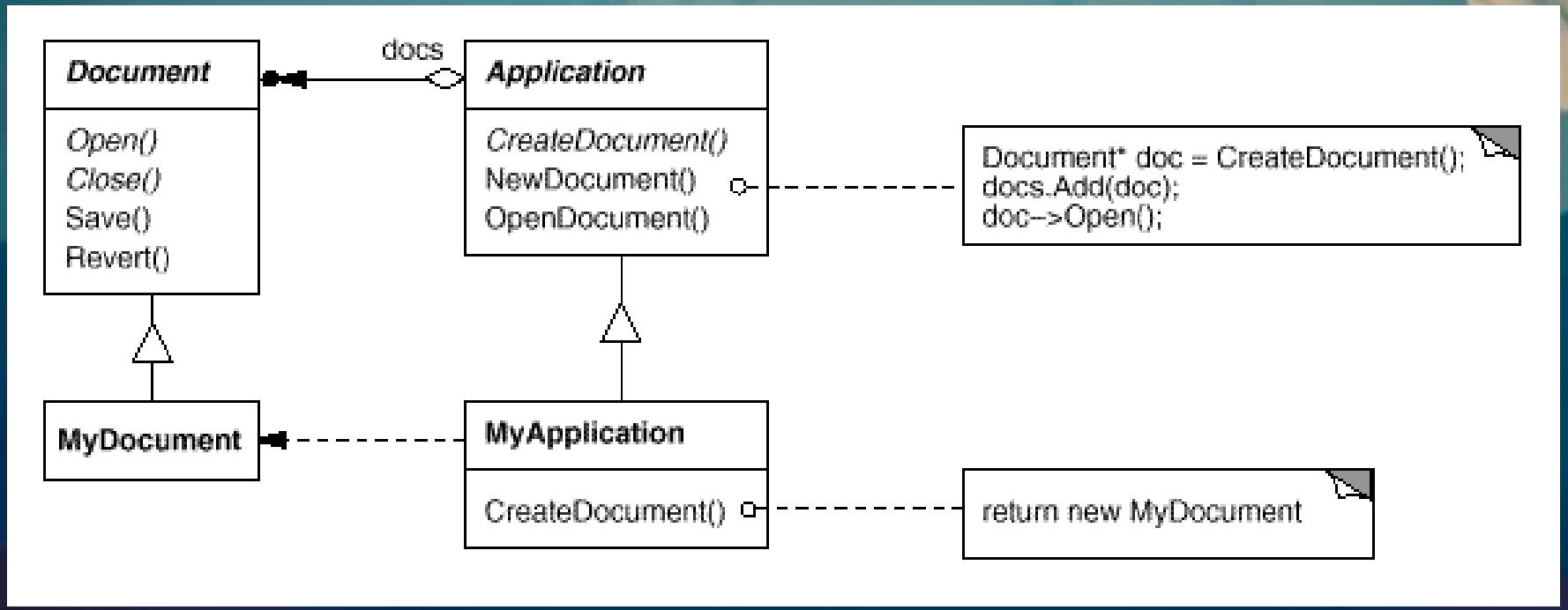


Factory Method模式(六)

- Implementation
  - (1) 父类是否提供缺省的实现
  - (2) factory method的参数
  - (3) Language-specific variants and issues
    - SmallTalk，使用类型
    - C++，使用lazy initialization技术
  - (4) Using templates to avoid subclassing

Factory Method模式(七)

- Related Patterns
  - Abstract factory
  - Prototype
- Examples



模式 2： Abstract Factory(一)

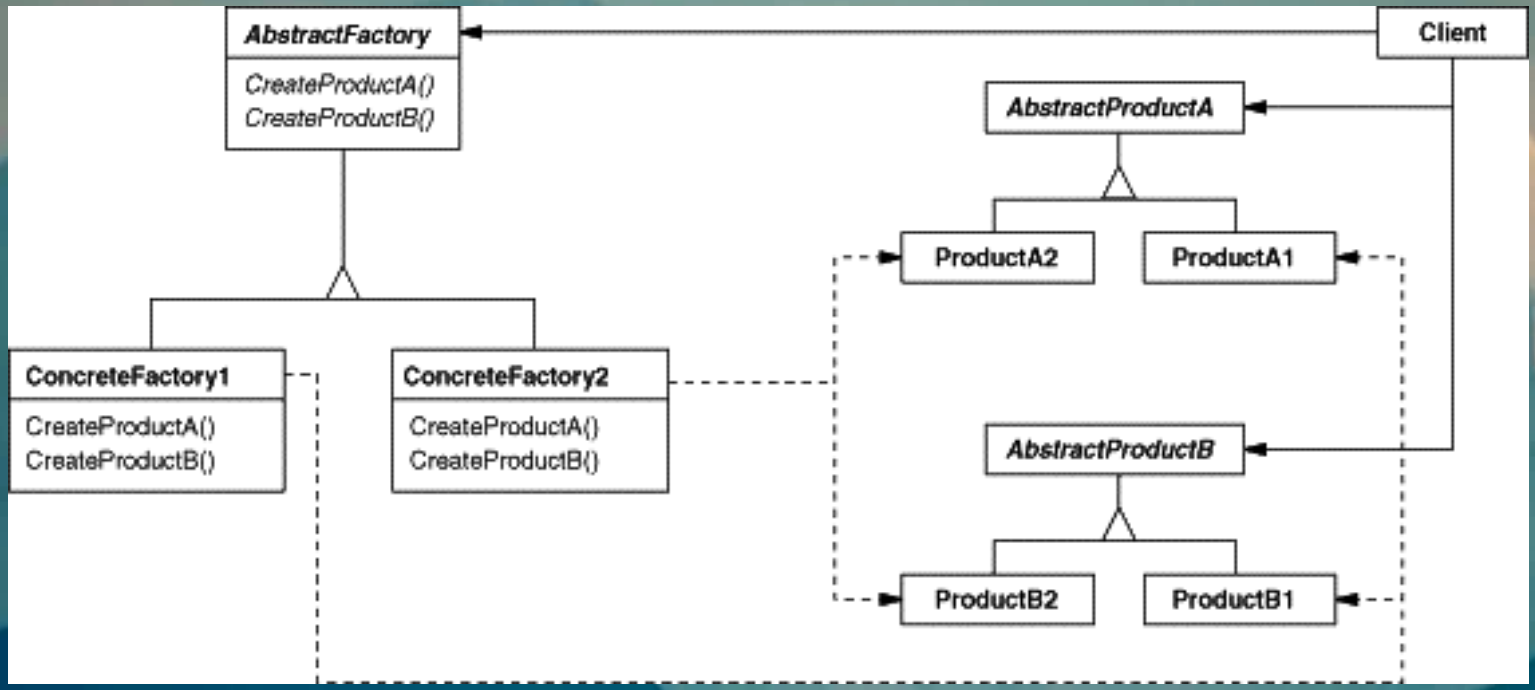
- Aliases：Kit
- Intent
  - Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- Motivation
  - 为了解决一族相关或者相依对象的创建工作，专门定义一个用于创建这些对象的接口(基类)。客户只需与这个基接口打交道，不必考虑实体类的类型。

Abstract Factory(二)

- Applicability，Use the Abstract Factory pattern when
  - a system should be independent of how its products are created, composed, and represented.
  - a system should be configured with one of multiple families of products.
  - a family of related product objects is designed to be used together, and you need to enforce this constraint.
  - you want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.

Abstract Factory(三)

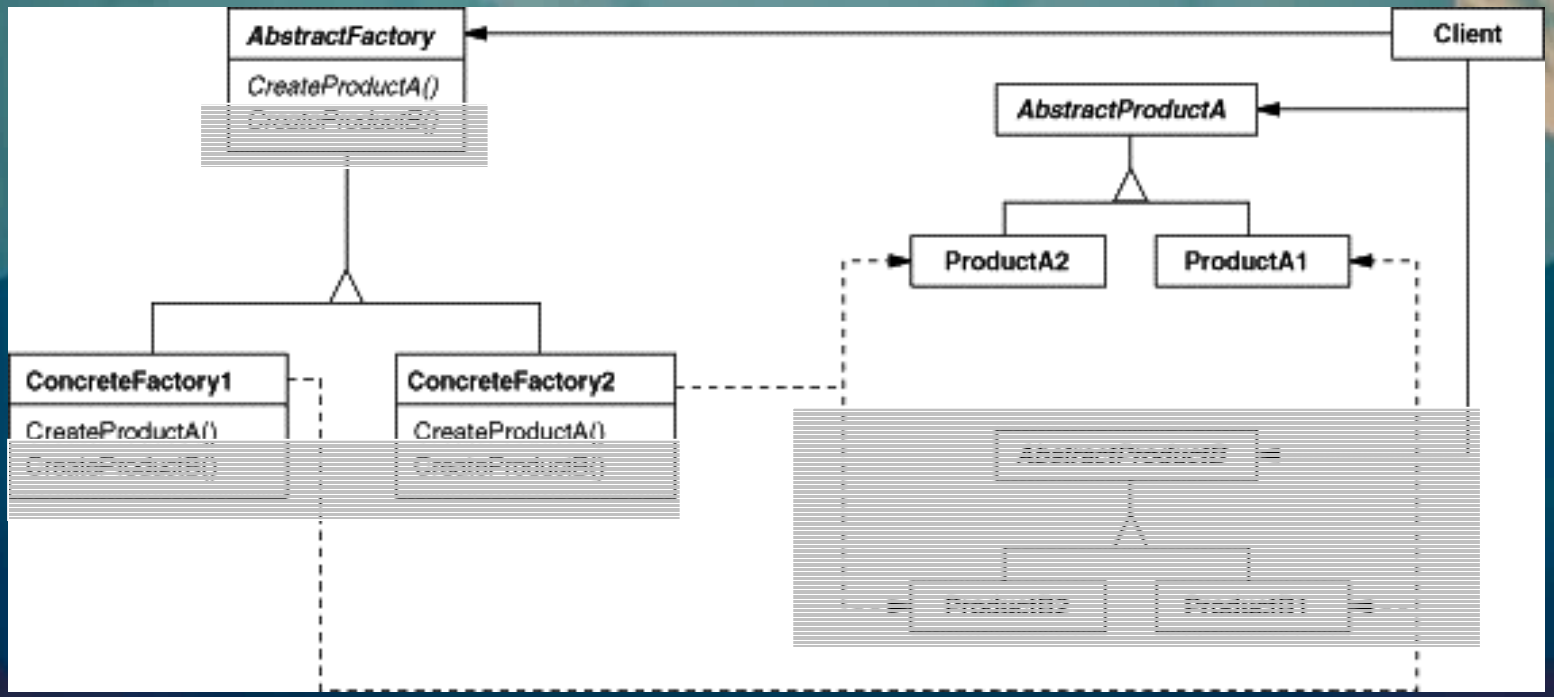
- Struct



- Participants：
  - Client、AbstractFactory、ConcreteFactory、AbstractProduct、ConcreteProduct
- Collaborations

Abstract Factory(四)

- Evaluation
  - 与factory method的关系
    - 多个factory method合在一起
    - factory method一定是virtual的



Abstract Factory(五)

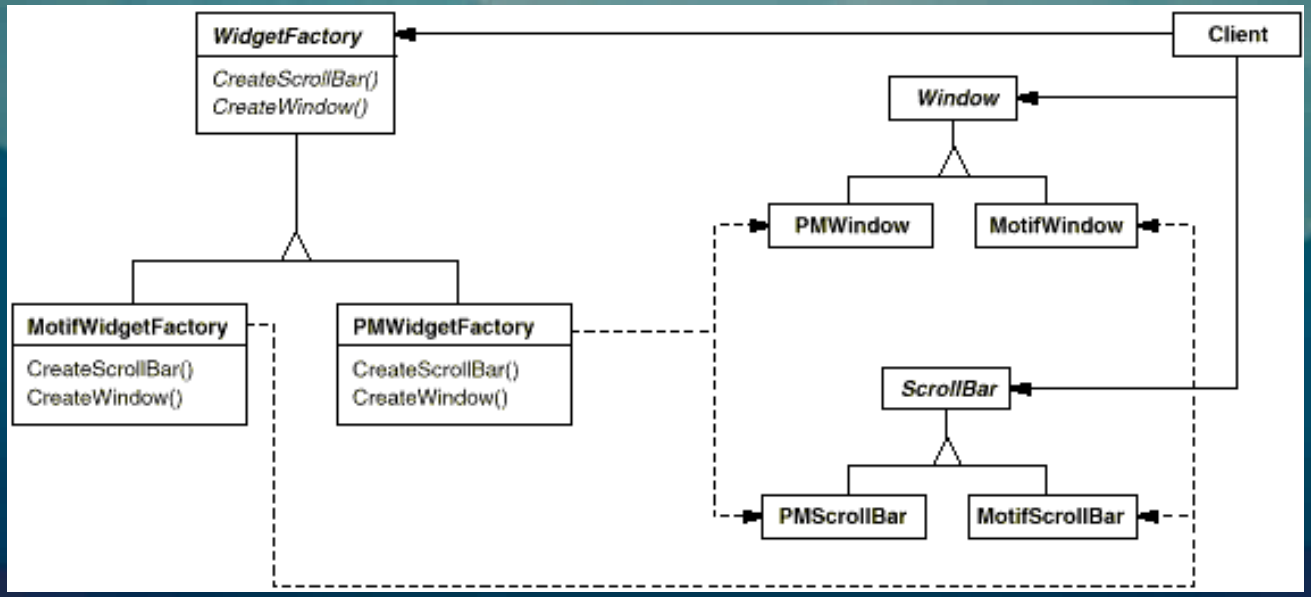
- Evaluation(续)
  - 优点：
    - factory把product的类型封装起来，分离了具体的类
    - 易于变换product族
    - 保证不同族之间的product相互不会碰撞，即保证products的一致性
  - 缺点：
    - factory对象的方法数目对应product数目，增加新的product种类比较困难，要影响到factory的基类，进而影响到所有的子类

Abstract Factory(六)

- Implementation
  - Factories as singletons, 每个product族往往只需要一个factory对象就可以了
  - Creating the products, 对于product族比较多的情况，可以使用prototype模式来实现这些factories，而不必对于每一个具有细微差别的product族都使用一个concrete factory class
  - Defining extensible factories，针对Evaluation中提到的缺点，通过参数化技术提高factory的适应能力和扩展性

Abstract Factory(七)

- Related Patterns
  - Factory Method、Prototype 、Singleton
- Examples
  - WidgetFactory



插：COM中的class factory

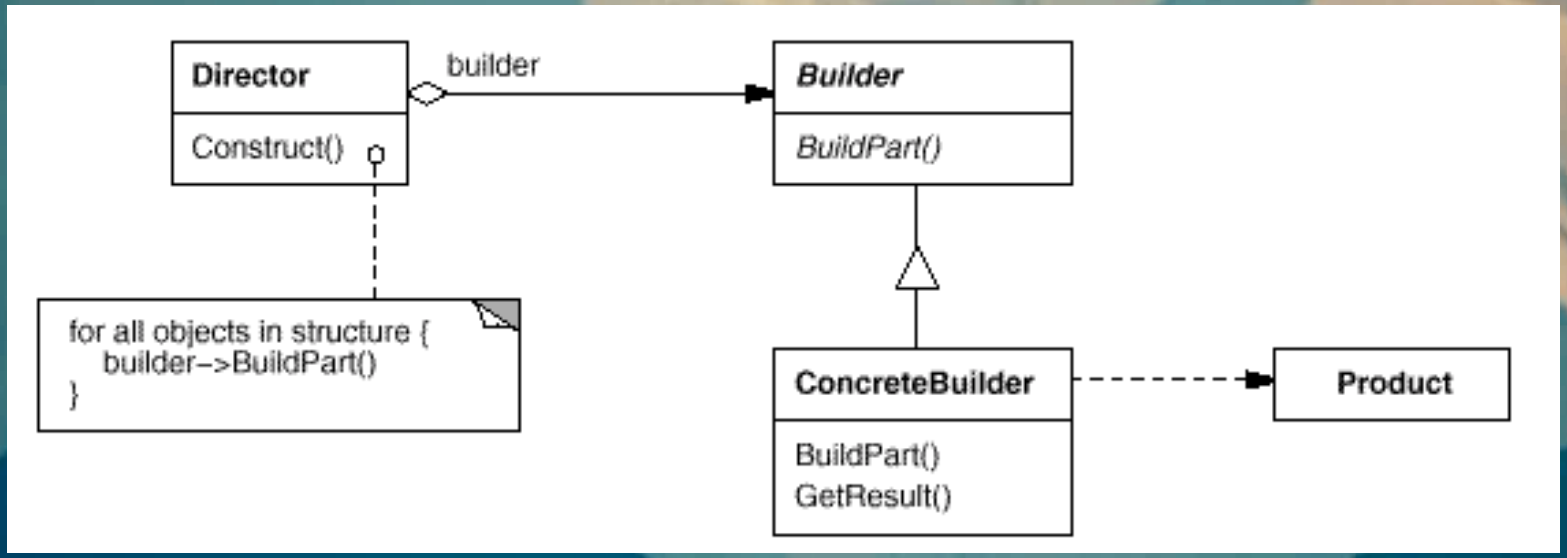
- 兼有两种模式：factory method和abstract factory
  - IClassFactory是abstract factory的接口
  - CreateInstance是factory method
  - 对于每一个coclass，class object就是Concrete Factory，
  - 每一个产品的抽象接口为IUnknown，COM对象是真正的concrete product
  - IPSFactoryBuffer也是abstract factory的接口
- 通过factory method创建对象
  - 比客户直接创建对象，要灵活
  - ConcreteFactory也是一个concrete product，所以可以重用底层的许多机制，如套间机制、跨进程机制等

模式三：Builder (一)

- Intent
  - Separate the construction of a complex object from its representation so that the same construction process can create different representations
- Motivation
  - 在复杂对象的构造过程中，允许同样的构造过程能够加入新的被构造元素
  - “结构化构造过程”
- Applicability, Use the Builder pattern when
  - the algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.
  - the construction process must allow different representations for the object that's constructed.

Builder (二)

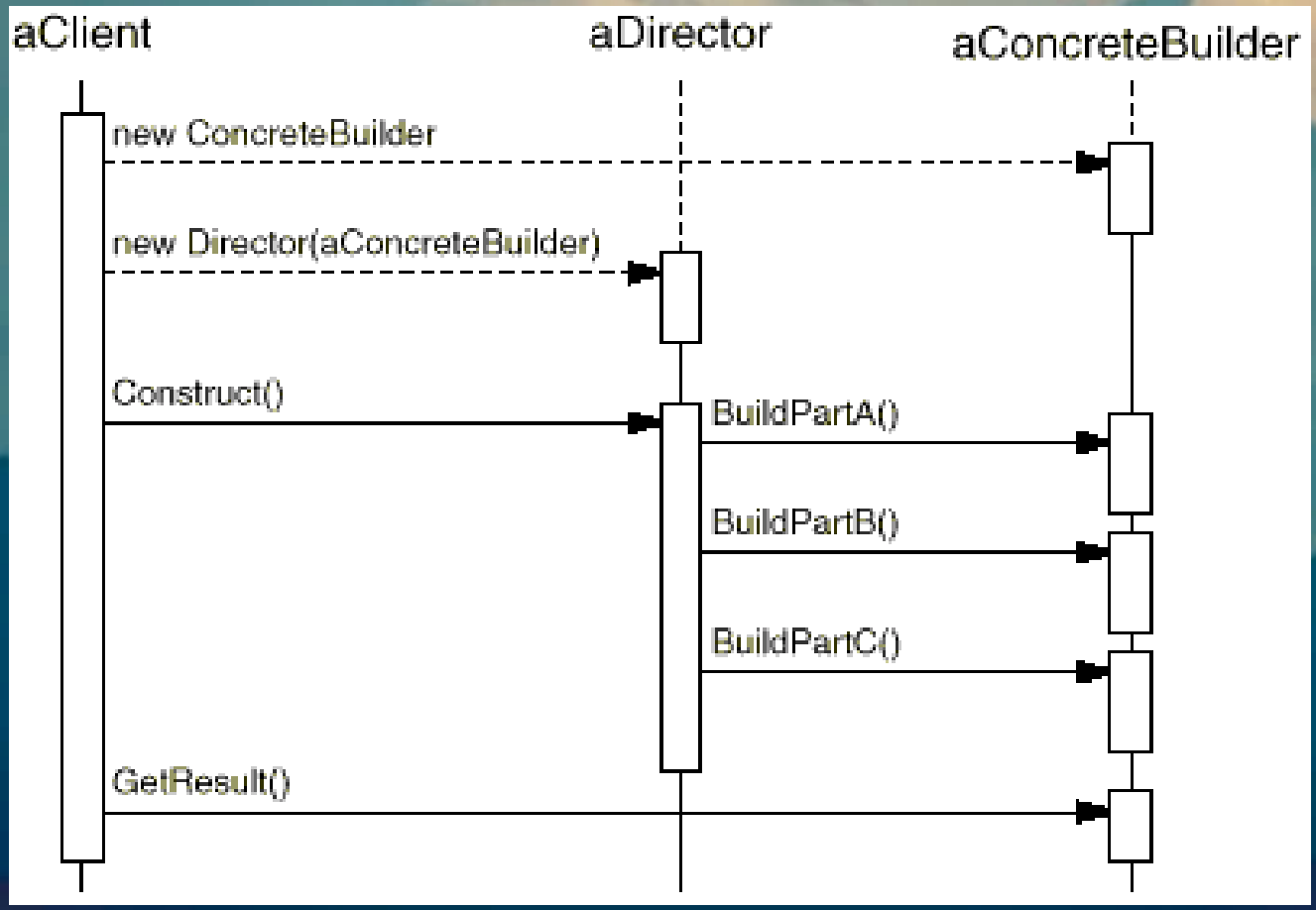
- Structure



- Participants
  - Director、 Builder、ConcreteBuilder、 Product

Builder (三)

- Collaborations

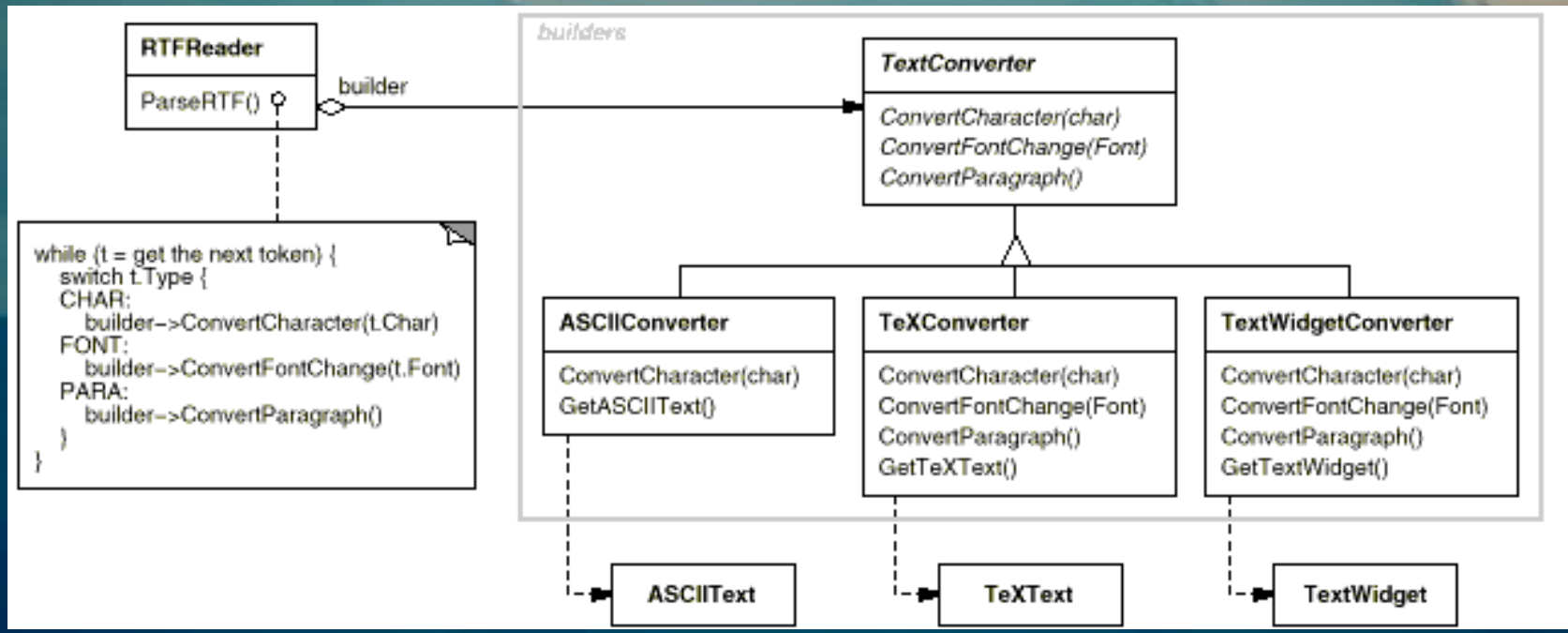


Builder (四)

- Evaluation
  - It lets you vary a product's internal representation
  - It isolates code for construction and representation
  - It gives you finer control over the construction process
- Implementation
  - Builder interface(Assembly and construction)
  - Why no abstract class for products?
  - Empty methods as default in Builder.
- Related patterns
  - Abstract Factory
    - 区别：(1) builder重在构造过程，最后一步返回结果;
      (2) builder构造许多复杂对象

Builder (五)

- Examples
  - readers、parsers、converters
  - the persistence of OLE documents

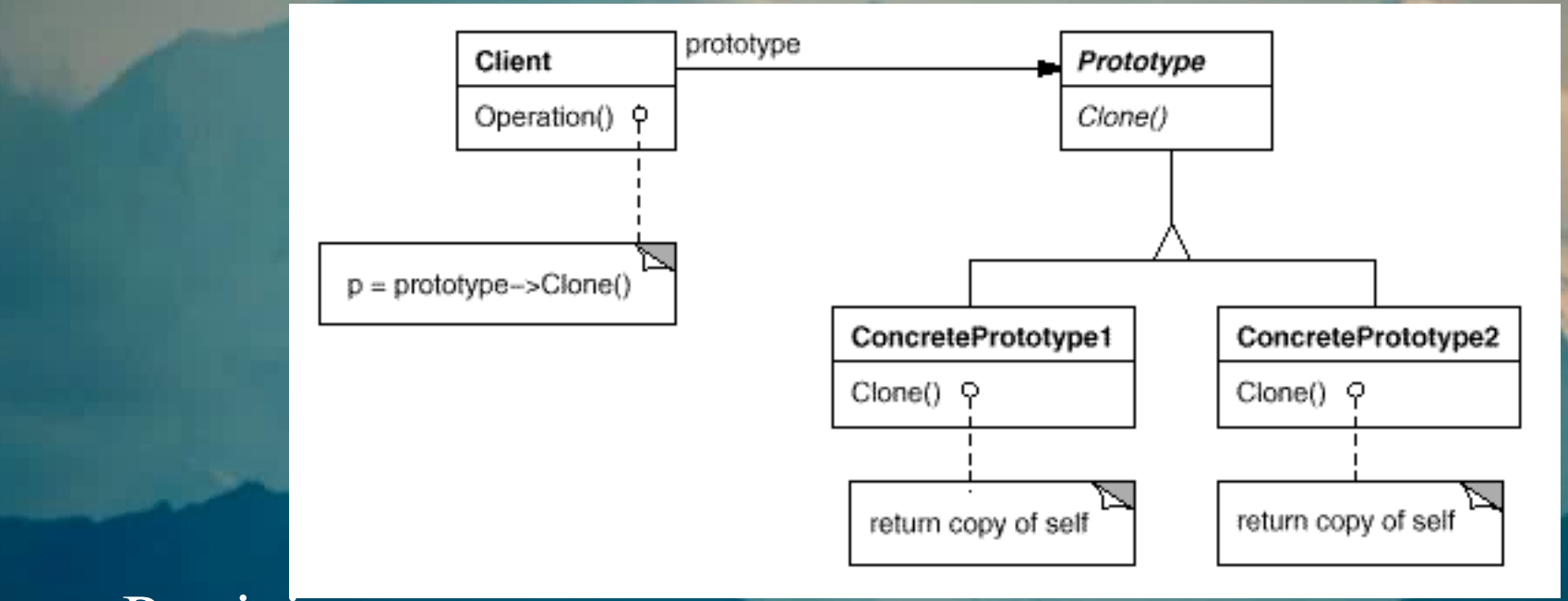


模式四：Prototype(一)

- Intent
  - Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- Motivation
  - 以一个已有的对象作为原型，通过它来创建新的对象。在增加新的对象的时候，新对象的细节创建工作由自己来负责，从而使新对象的创建过程与框架隔离开来
- Applicability
  - 当产品的创建过程要独立于系统时
  - 当产品的类型是在runtime时被指定的情况下
  - 避免创建一个与product层次平行的factory层次时

Prototype(二)

- Structure



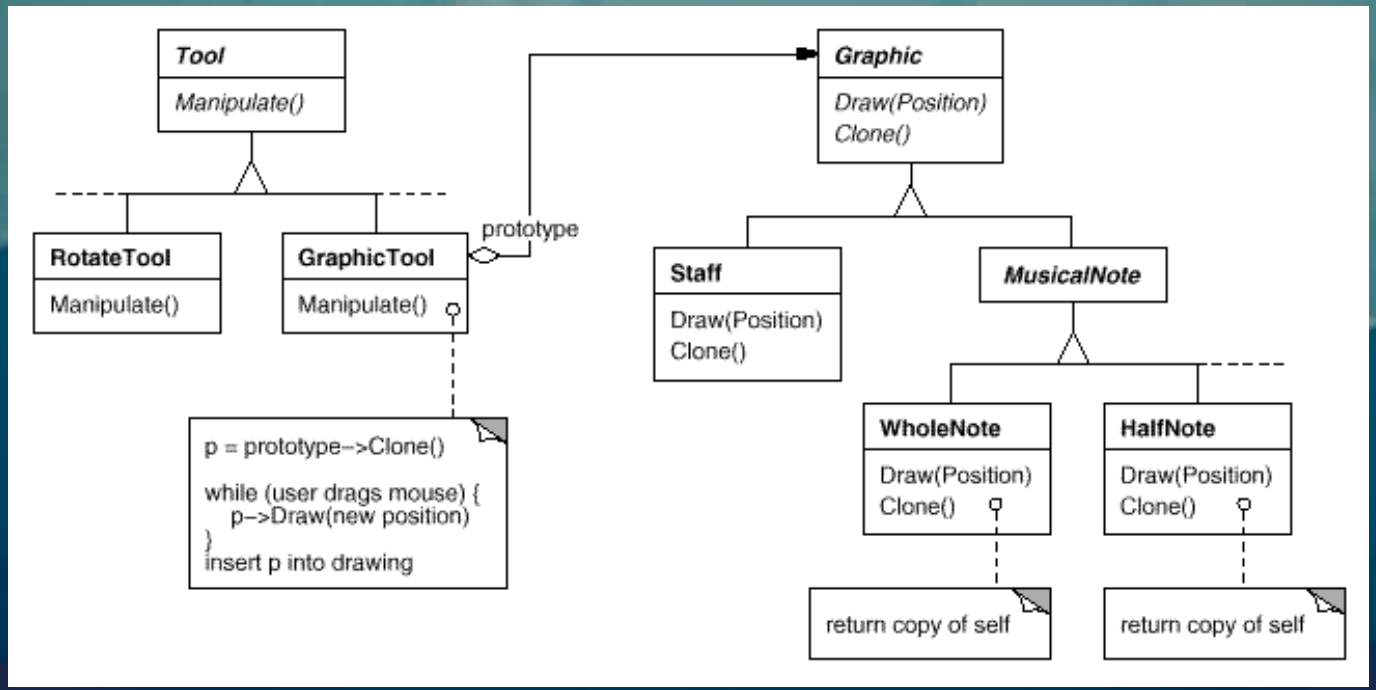
- Participants
  - Prototype、ConcretePrototype、Client
- Collaborations

Prototype(三)

- Evaluation
  - Adding and removing products at run-time
  - Specifying new objects by varying values，降低系统中类的数目
  - Configuring an application with classes dynamically
  - 要求：每一个product类都必须实现Clone操作
  - 对于C++语言特别有意义：C++的class不是first-class objects
- Implementation
  - Using a prototype manager
  - Implementing the Clone operation
    - shallow copy versus deep copy
    - Save & Load
  - Initializing clones
    - 两阶段构造

Prototype(四)

- Related patterns
  - Prototype与Abstract Factory往往是相互竞争的
  - factory method
- Examples
  - DrawCli(不完全)，music editor

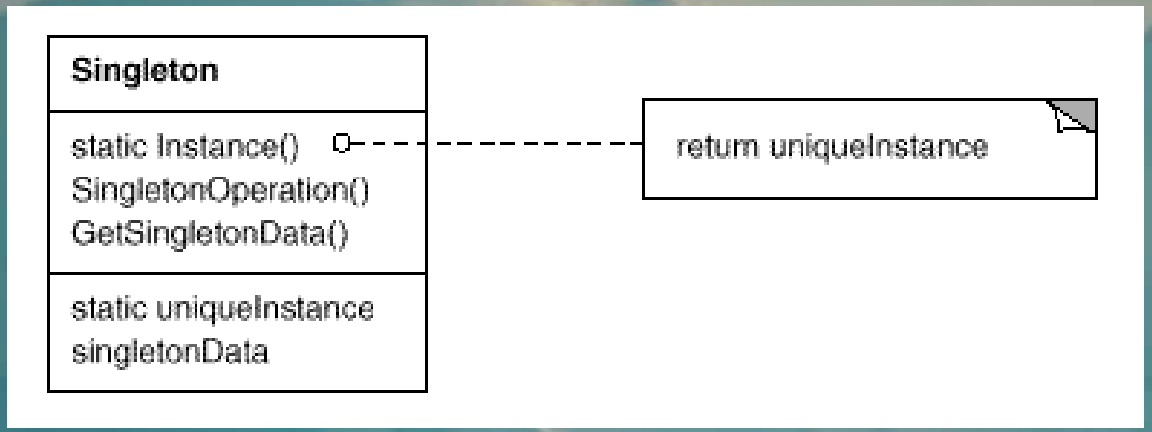


模式五：Singleton(一)

- Intent
  - Ensure a class only has one instance, and provide a global point of access to it.
- Motivation
  - It's important for some classes to have exactly one instance.
- Applicability, Use the Singleton pattern when
  - there must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
  - when the sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

Singleton(二)

- Structure



- Participants
  - Singleton
- Collaborations
  - Clients access a Singleton instance solely through Singleton's Instance operation.

Singleton(三)

- Evaluation
  - Controlled access to sole instance
  - Reduced name space
  - Permits refinement of operations and representation， 允许子类化
  - Permits a variable number of instances
  - More flexible than class operations (static member functions in C++)
  - 这种思想比较适用于Object-Based中的许多情形
- Implementation
  - Ensuring a unique instance
    - 考虑使用lazy initialize
    - 使用global/static object的缺点
  - Subclassing the Singleton class

Singleton(四)

- Related patterns
  - Singleton与其他创建型模式并不矛盾，可以用singleton来实现其他模式中的对象。包括Abstract Factory、Builder、Prototype等。
    多个实例对于构造过程往往并无意义，所以在许多情况下singleton模式比较符合应用背景

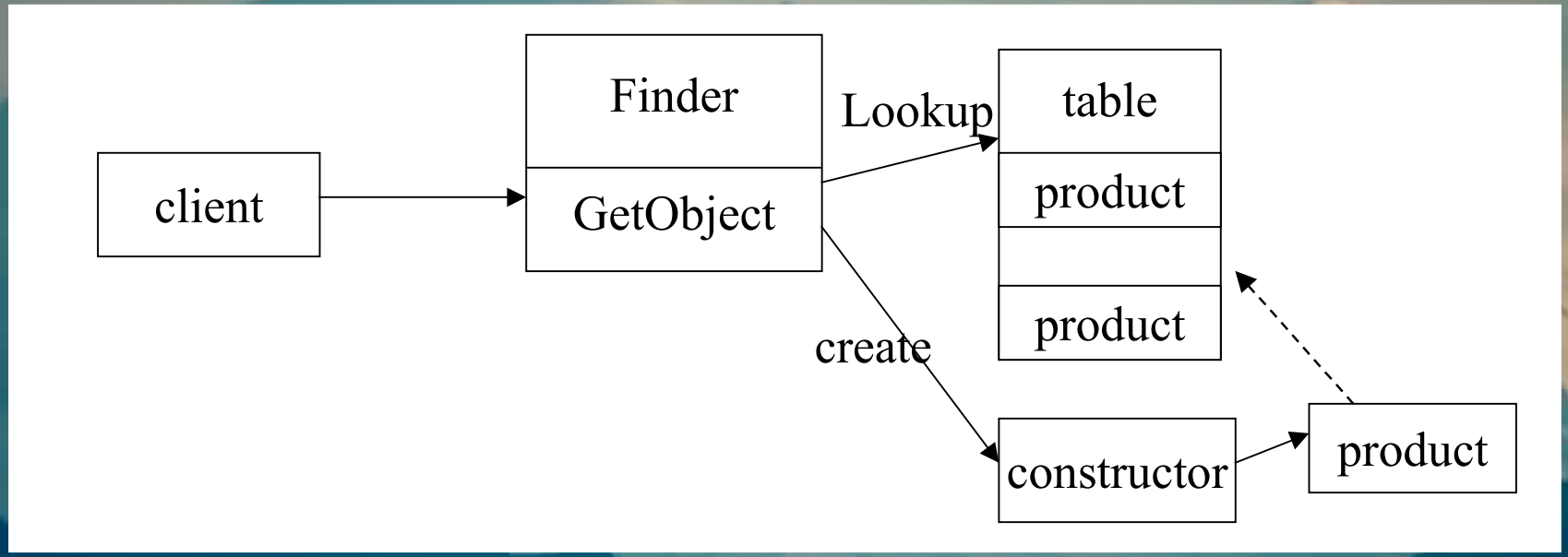
- Examples
  - MFC中的CWinApp派生类实例theApp
  - ……

增加模式六：Finder(一)

- Intent
  - 利用环境信息，根据客户的请求，找到已有的、符合要求的对象，返回给客户
- Alias
  - Object-retriever
- Motivation
  - 在有些情况下，客户希望连接到一个对象上，它提供一些状态标识信息，由Finder返回已经被创建的对象，或者重新创建新的对象(如果当前不存在满足条件的对象)
- Applicability, Use the Finder pattern when
  - 当需要在软件不同部分之间建立Client/Object连接时
  - 把获取对象的过程隐藏起来
  - view finder：The application demands user customizability of the actions taken when a particular file format is encountered in the browser.

Finder(二)

- Structure



- Participants
  - client、finder、product-table、constructor、product
- Collaborations

Finder(三)

- Evaluation
  - 避免同样的对象被实例化两次，从而提高资源利用率，避免发生资源竞争
  - 把连接对象的过程与客户隔离开
  - 带来的问题：多个客户共享同样的资源，如何有效管理对象的所有权？
- Implementation
  - 实现product table以及相应的管理设施，保证查找过程的有效性
  - 每一个product的生命周期管理
  - 如何标识product的类型，客户如何多态地提供状态标识信息
  - Finder对象本身可以是一个singleton

Finder(四)

- Related patterns
  - 与Singleton的区别：singleton是一个类的单个实例；而Finder是避免相同的对象(通常是类型和状态信息都相同)被创建两次。
  - 与Prototype的区别
  - 在创建product子步骤中，需要与其他创建型模式结合使用
- Examples
  - moniker in COM
  - 在Netscape浏览器中，根据MIME类型，找到插件，然后创建view

creational patterns小结

- 了解每一种模式的实质
  - 具体实现的时候可能会有变化情况，或者扩展，或者退化
- factory method是基础，abstract factory是它的扩展
- factory method、abstract factory、prototype都涉及到类层次结构中对象的创建过程，有所取舍
  - prototype需要prototype manager
  - factory method需要依附一个creator类
  - abstract factory需要一个平行的类层次
  - 根据应用的其他需求，以及语言提供的便利来决定使用哪种模式

creational patterns小结(续)

- builder往往适合于特定的结构需要，它所针对的product比较复杂
- singleton有比较强烈的物理意义，可以用在许多细微的地方，不一定与类层次关联
- finder模式需要有一定范围内的对象管理功能
- 这些patterns都很常见，有时需要结合两种或者多种模式完成系统中对象的构造过程