SOA: A Service for Products

MTH 9815: Software Engineering For Finance

Breman Thuraisingham, Morgan Stanley

Introduction

- We now introduce a Service Oriented Architecture (SOA) with a simple example
- We then illustrate the power of SOA with a ProductService over bonds and interest rate swaps
- We will introduce the beginnings of a trading system with various services in an SOA architecture needed to build the platform
- We start with a ProductService and then move to pricing, trade booking, execution, and market data in the next class
- SOA can thus be used for distributed systems (and is the next level of such a system) – very common in financial software platforms

Shared Memory (from last class)

- Middleware is best for distributed computing systems where it does not matter where components are deployed
- Performance can degrade when components are placed in different datacenters, particularly when those datacenters are physically located far apart from each other
- The best performance is where components are co-located on the same machine
- For such co-location, shared memory can be used to communicate between components
- Shared memory is where processes share the same physical memory on the machine (as if they were one program)
- If the server writes to a data structure that the client reads, you have essentially communicated data between client and server as if it were via a network connection, but now it's much faster since it's a memory read/write
- Beware of contention: if one processor is reading from a location that another processor is writing to, both processors cannot cache the memory and must go back to physical memory

Shared Memory: Boost.Interprocess (from last class)

- Use the Boost.Interprocess C++ library for sharing memory across processes
- Use class `boost::interprocess::shared_memory_object`
- Must set the size of the shared memory object in bytes using the `truncate()` method
- Memory must be explicitly destroyed after use by using the `remove()` method

Memory Mapped Files

- A memory mapped file is a part of memory (typically virtual) that can be accessed across processes
- Can be a file in virtual memory, shared memory, etc
- Generally mapped to the kernel's file cache for fast access
- C++ supports memory mapped files in Boost
- Java supports memory mapped files in a `FileChannel` class

Service Oriented Architecture

- A Service Oriented Architecture (SOA) specifies the definition of services each with specific functions and responsible for a set of data that coordinate amongst themselves in a distributed system
- SOA is a refinement of distributed computing
- Services perform business logic and are self-contained
- They encapsulate the data and functions they are responsible for, hiding the details of how they perform these functions or store this data
- SOA architecture takes the concepts above for a client/server to the next level, with the servers being able to both perform functions and vend out data
- But instead of multiple well-defined clients communicating via a single well-defined server component, we have many servers (or services) responsible for different operations and data
- And each service can itself be a “client” to another server

Service Oriented Architecture (Continued)

- Thus the line between client and server is blurred in a SOA architecture
- You still have client-only components though in SOA that go to multiple services for the data they need and the operations they need performed

Defining a Service

- Let’s now put together a simple example of SOA
- Consider the following definition of a Service:
  - template<typename K, typename T>
  - class Service
  - {
  - public:
  -   virtual T& GetData(K key) = 0;
  - };
- This defines a Service with keys of type K and values of type V
- Services can get more sophisticated where we can add listeners to be notified for data changes in the Service – we explore this in more detail in upcoming classes where we use Services

Service Example: DirectoryService

- We illustrate with a simple example
- Let’s consider a `Person` class
- We define a `DirectoryService` which returns objects of type `Person`
- We lookup `Person` instances with a name
- See example in test_soa.cpp

Accessing Service Data

- We can use middleware on the network to access data from a Service
- We can use RPC to retrieve data in a Service from a separate process
- We can also use a pub/sub mechanism to access service data
- Updates to the data can be streamed via pub/sub
- Tibco RV provides a powerful way to use middleware to access Service data via middleware (with a last value cache)

Representing a Basic Product

- We will model bonds and interest rate swaps as C++ classes
- First let's model a base class `Product` for any type of financial product
- A product consists of an identifier and a product type (bond, interest rate swap, etc)
- We model this `Product` class in products.hpp

Representing a Bond

- Bonds are the most fundamental type of fixed income product
- It is a Cash product and can be government debt, agency debt, corporate debt, etc
- Now we model an actual bond
- See the `Bond` class in products.hpp
- A bond consists of:
  - An identifier
  - An identifier type
  - A product type BOND
  - A ticker
  - A coupon
  - A maturity date
- These attributes are part of our `Bond` class

Representing a Bond (Continued)

- Many more attributes that we can also model!
- A bond is keyed on its product identifier, which is well defined (CUSIP or ISIN generally) and is used to lookup products later in our product service

Representing an Interest Rate Swap

- An interest rate swap is our most typical kind of fixed income derivative product
- It is a swap of a fixed rate leg and a floating rate leg
- Buy and sell are called pay and receive from the perspective of the payer or receiver of the fixed leg
- Can be used as a hedge against changes in interest rates
- Swaps are hedged with EuroDollar futures and OTR Treasuries

Interest Rate Swap Attributes

- Attributes of a swap are:
  - Fixed and floating leg day count convention (30/360, Act/360)
  - Fixed leg payment frequency (quarterly, semi-annual, annual)
  - Floating rate index (LIBOR, EURIBOR)
  - Floating index tenor (1m, 3m, 6m, 12m)
  - Effective date
  - Termination date
  - Currency (USD, EUR, GBP)
  - Term (in years)
  - Swap type (Standard, Forward, IMM, MAC, Basis)
  - Swap leg type (Outright, Curve, Fly)
- We model `IRSwap` in products.hpp
- Many more attributes that we could add here!
- Swaps are keyed on a product identifier which is not well-defined and generally describes the swap

BondProductService

- Now we create a `BondProductService`
- This class stores objects of type `Bond`
- See code in productservice.hpp
- This is the building block of a bond trading system where we need to access Cash products
- We can also add utility methods to search by specific attributes
- Hint: fleshing out such helper methods will be part of the next homework!

IRSwapProductService

- Now we create a `IRSwapProductService`
- This class stores objects of type `IRSwap`
- See code in productservice.hpp
- This is the building block of a swaps trading system where we need to access swaps products
- We can add rich utility methods to search by specific types of attributes, e.g. search for all MACs, Standards, Outrights, etc

Service Listeners

- We can create listeners on a service to listen to data changes in products
- Listeners are part of the observer pattern and are an essential part of SOA for clients to be notified upon changes to service data
- See the example illustrated
- We explore service listeners in a trading system in more detail in the next class
- Again, middleware can be used to propagate data updates to interested listeners in other processes – a fundamental part of a distributed SOA platform!

The Power of SOA!

- In this class we have explored the power of SOA
- SOA defines how components are put together in a distributed platform
- Service processes should contain actual services – generally multiple services to break down data and operations
- Services operate together in a cohesive unit to create a platform
- In the next class, we discuss how we bring several services together to construct a trading system:
  - Product Service
  - Pricing Service
  - Trade Booking Service
  - Execution Service
  - Market Data Service
  - Algo Trading Service
- As a quant, think of how you can use SOA in your platform!