Application Programming Interfaces

CMPU4023 - Enterprise Application Development

What is an API?

- The idea of an API is a familiar one in computer science and we find implementations of them everywhere in the enterprise
- Developers create and consume software-defined interfaces to some *external* functionality
- External could mean within the same *memory space* as the consumer but provided by a third-party library or in a different memory space from the consumer either on the same computer system or a remote system
- In the context of enterprise services, the API is the level of abstraction which describes the functionality and capabilities provided by the service and how they're expected to be used
- In the literature, an API is said to *export* its interfaces

Simple API Illustration

- Suppose there exists a microservice whose job is to calculate the V.A.T. for products being shipped to various destination countries around the globe
- A separate order fulfillment service needs to calculate the V.A.T. for a shipment to a particular customer in order to complete an invoice
- The fulfillment service calls the V.A.T. service's API, passing details of the products and destination country
- The V.A.T. service responds with the V.A.T. information (e.g. rates, amounts, etc)
- The fulfillment service is said to be a consumer of the V.A.T. service

Desirable Characteristics

- **Consistent**. The interface behaviour used to access one type of functionality is used consistently to access all functionality. This includes behaviour in success and failure mode, the kinds of return types and errors codes
- **Predictable**. If you already know how one part of the API works and behaves, you can easily guess how another part would work even if you've never used it before
- **Learnable**. There exists living documentation detailing how the API works and, where appropriate, the rationale as to why certain things were designed in certain ways to aid a deeper understanding by the consumer

Remote APIs

- For our purposes, we will focus on a particular kind of API widely found in the enterprise, APIs that are the remotely-accessed over networks
- Broadly, there are three paradigms for exposing such remote APIs

- **Message Passing** is lowest level abstraction where participants exchange messages of some structure to/from specified receivers and senders

- **Remote Procedure Calls (RPC)** is a mechanism, build upon message passing, that allows you to call methods on remote services as though they were being called on local objects (services)

- **Distributed Shared Memory (DSM)** offers the highest level of abstraction wherein distributed memory on autonomous computers has the appearance of being centrally managed

Message Passing Semantics

- **Blocking/non-blocking:** If blocking, then the sender and receiver wait indefinitely until the transmission is complete, otherwise the operations return immediately
- **Buffered/unbuffered:** If unbuffered, it means the message is delivered directly to the receiver or from the sender without any intermediate buffering of requests or receipts
- **Reliable/unreliable:** If reliable, the sender has some guarantees that the message has been delivered to the recipient using some protocol implementation-dependent mechanism

Remote Procedure Call Semantics

- RPC semantics are similar to those of local procedure calls, making transparent the fact that there may be a high latency, low reliability network connecting the participants
- Procedure arguments (request data) and results (response data) are automatically marshalled and transmitted in some network portable way (e.g. dealing with big-endian vs little-endian nodes)
- Some RPC systems assume the same language execution environment on both sides (e.g. Java RMI) whereas others are fully heterogenous and technology agnostic (e.g. CORBA)

Distributed Shared Memory

- In DSM, participants read and write to distributed shared memory variables through shared virtual addresses (handles)
- The granularity of the shared memory is a design tradeoff between efficiency and liveness and typically ranges from a few bytes to a page
- The key challenge for DSM systems is maintaining consistency between the multiple copies of the same memory on each autonomous node
- Implicit in the data access semantics of DSM is the synchronisation of read and write access to shared variables
- DSM can support heterogeneous hardware and software nodes with the added burden of implementing portability at some layer in the stack

Enterprise APIs

- Historically, RPC-style APIs have been popular within the enterprise intranet
- However RPC standards like Java RMI and CORBA did not adequately address security so recent years has seen a migration to message passing style APIs based on Web technologies
- The Web stack, with its Internet origins has a strong security story in terms of encrypted transport and various authentication and authorization solutions
- This module will focus on message passing APIs which are widely adopted in the enterprise today

Transport and Application Layer

- Remote APIs have to operate over some kind of network between participating nodes
- The de facto standard transport layer protocol in the enterprise today is TCP/IP which provide reliable delivery guarantees which is usually what is required in most cases
- The application layer protocol of choice is the **H**yper**T**ext **T**ransport **P**rotocol (HTTP), part of the Web suite of technologies
- These can be combined with VPNs (layer 2 or 3) or TLS (layer 4) for encryption and authentication

Summary

- APIs are the exported interfaces that allow inter-networked services to communicate and exchange information and service in the enterprise
- In the context of SOA and microservices, we are interested primarily in remote APIs
- Remote APIs can be implemented using message passing, RPC or DSM semantics
- In practice, most are now message based
- Web technologies dominate the implementations of APIs in the enterprise today because they are standards-based, ubiquitous and well supported and tested