CMPU4021 Distributed Systems

Revisions - 2

Remote Invocation

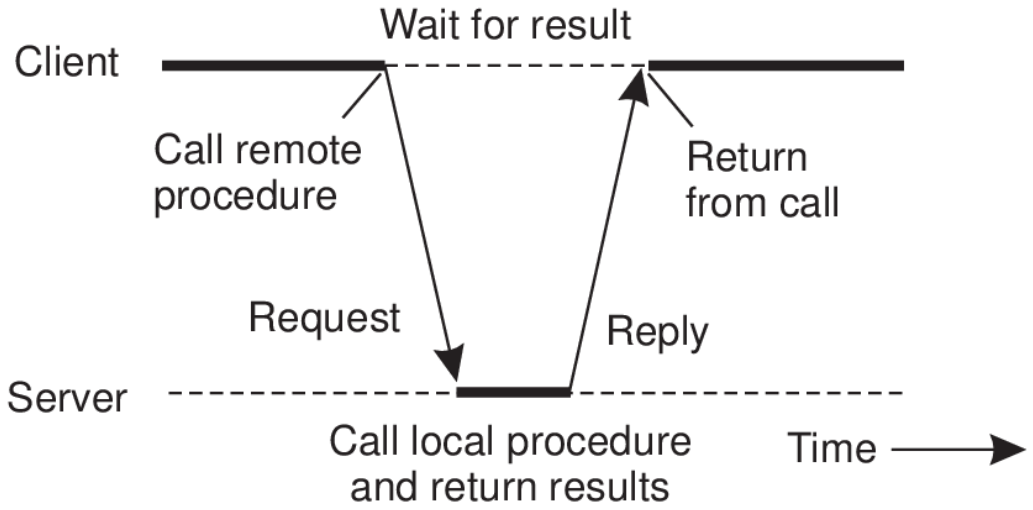
- Covers a range of techniques
- Based on a two-way exchange between communicating entities in a distributed system
  - resulting in the calling of a remote operation, procedure or method.

- Remote Procedure Calls (RPC)

- Remote method invocation (RMI)

Basic RPC operation

- Supports client-server computing with servers offering a set of operations through a service interface and clients calling these operations directly as if they were available locally.
- RPC systems offer (at a minimum)
  - access transparency
    - the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa.
  - location transparency



Remote method invocation (RMI)

- Strongly resembles remote procedure calls but in a world of distributed objects.
- A calling object can invoke a method in a remote object.
- Communication among distributed objects via RMI
  - Recipients of remote invocations are remote objects, which implement remote interfaces for communication
- Reliability
  - Either one or both the invoker and invoked can fail, and status of communication is supported by the interface. e.g.,
    - notification on failures,
    - reply generation,
    - parameter processing – marshalling/unmarshalling
- Local invocations target local objects, and remote invocations target remote objects

Interfaces

- Interfaces hide the details of modules providing the service; and access to module variables is only indirectly via ‘getter’ and ‘setter’ methods / mechanisms associated with the interfaces
  - e.g., call by value/reference for local calls through pointers vs. input, output, and input paradigms in RMI through message-data and objects

- *Service interfaces*
  - client-server model, specification of the procedures offered by a server
    - defining the types of input and output arguments

- *Remote interfaces*
  - distributed object model, specifies the methods of an object that are available for invocation by objects in other processes
    - defining the types of the input and output arguments of each of them.

Interface definition languages (IDLs)

- IDL is a language that is used to define the interface between a client and server process in a distributed system.

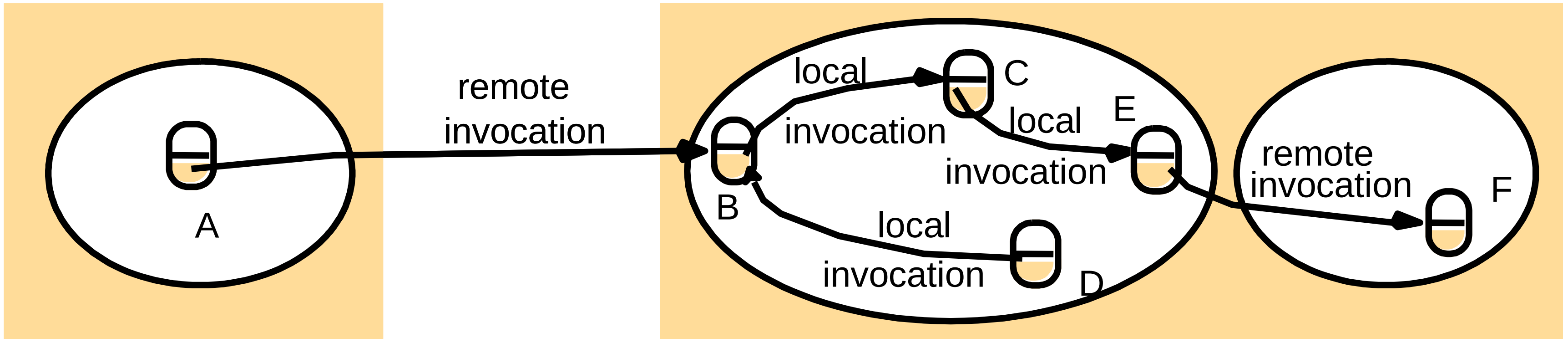
- Each interface definition language also has a set of associated IDL compilers
  - one per supported target language.

- An IDL compiler compiles the interface specifications, listed in an IDL input file, into source code (e.g., C/C++, Java) that implements the low-level communication details required to support the defined interfaces.

- Provides a notation for defining interfaces in which each of the parameters of a method may be described as for *input* or output in addition to having its type specified.

The distributed object model: Remote and local method invocations

- Objects receiving remote invocations (service objects) are remote objects, e.g., B and F
- Object references are required for invocation, e.g., C must have E’s reference or B must have A’s reference
- B and F must have remote interfaces (of their accessible methods)



Distributed Garbage Collection

- Where any process includes remote objects, then it is equipped wlth both
  - Local garbage collector
  - Distributed garbage collector
- For any remote object O
  - `O.holders` is a list of all the processes that have a remote reference to that object i.e. got a stub for it
- When a client C receives a remote reference for O it makes an `addRef` call to O's garbage collector
  - resulting in its being added to `O.holders`
- When C's local garbage collector attempts to delete the stub object for O, it calls `removeRe`f on O's garbage collector, resulting
  - it its being removed from `O.holders`
- When `O.holders` is empty, O can be deleted

Invocation Semantics

- Unreliable network
  - For all request –reply protocols, messages may get lost
- Solutions for lost / retransmitted messages
  - Retry request
  - Filter Duplicates
  - Retransmit results

Invocation Semantics

- In local OO system
  - all methods are invoked exactly once per request –guaranteed – unless whole process fails
- In distributed object system, we need to know what has happened if we do not hear result from remote object i.e. did the request go missing, did the response go missing
- 3 different types of guarantee (invocation semantics) may be provided
  - could be implemented in a middleware platform intended to support remote method invocations:
    - *Maybe*
    - *At-Least-Once*
    - *At-Most-Once*

Maybe Invocation Semantics

- If the invoker cannot tell whether a remote method has been invoked or not

- Very inexpensive, but only useful if the system can tolerate occasional failed invocations

At-Least-Once Invocation Semantics

- If the invoker receives a result, then it is guaranteed that the method was invoked at least once
- Achieved by resending requests to mask omission failure
- Only useful if the operations are idempotent (x = 10, rather than x = x + 10)
- Inexpensive on server

At-Most-Once Invocation Semantics

- If the invoker receives a result, then it is guaranteed that the method was invoked only once
- If no result is received, then the method was executed either never or once
- Achieved by resending requests, and storing and resending responses
- More expensive on a server / remote object, which must maintain results and recognise duplicate messages

Invocation semantics: failure model

Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Design Issues of RMI

- Local invocations have
  - at-most-once, or
  - exactly-once semantics

- Distributed RMI, the alternative invocation semantics are:
  - *Retry request message* – retransmit until reply is received or on server failure – at-least-once semantics;

  - *Duplicate message filtering* – discard duplicates at server (using seq #s or ReqID);

  - Buffer result messages at server for *retransmission* – avoids redo of requests (even for idempotent ops) – at-most-once semantics.
    - Idempotent operation – the one which can be performed repeatedly with the same effect as if it had been performed exactly once.

Security Goals

- Keep systems, programs, and data secure
- Three areas
  - Confidentiality
    - Keeping data & resources hidden
  - Integrity
    - Protecting against unauthorized changes to the data or resources
  - Availability
    - Ensuring that a system is accessible and capable of working to required performance specifications

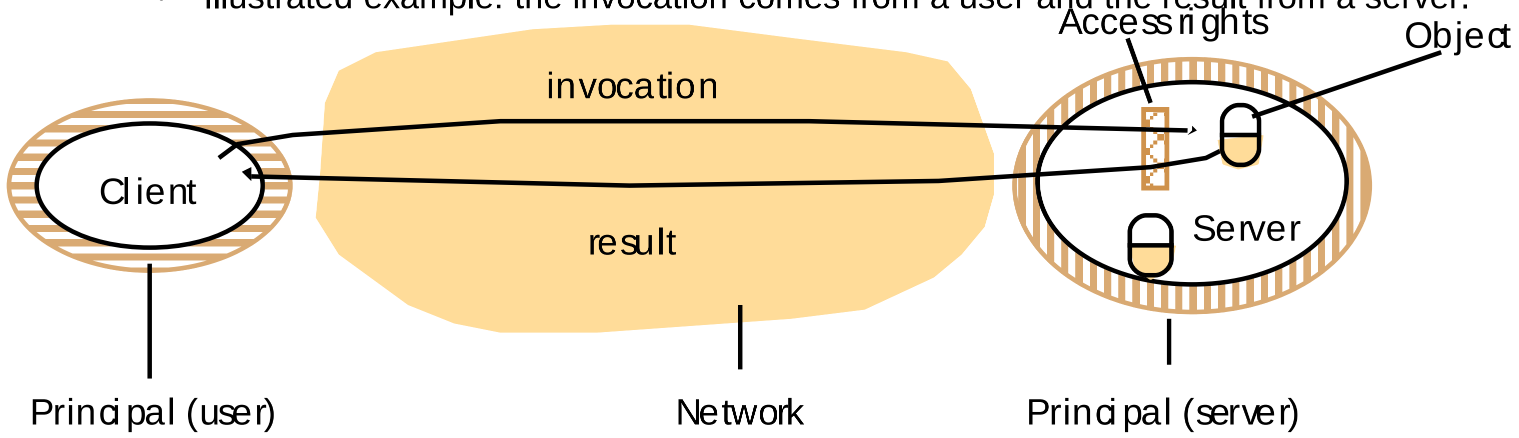
Security in distributed systems

- Two specific concerns that centralized systems do not have
  - use of a network
    - contents may be seen by other, possibly malicious, parties
  - use of servers
    - That authenticate the client and control access to services
    - physical access to the system and the security controls - unknown to the client

Security model

- Security model:
  - The security of a distributed system can be achieved by securing processes and communication channels, and
  - by protecting objects (or resources of all types) they encapsulate against unauthorized access.

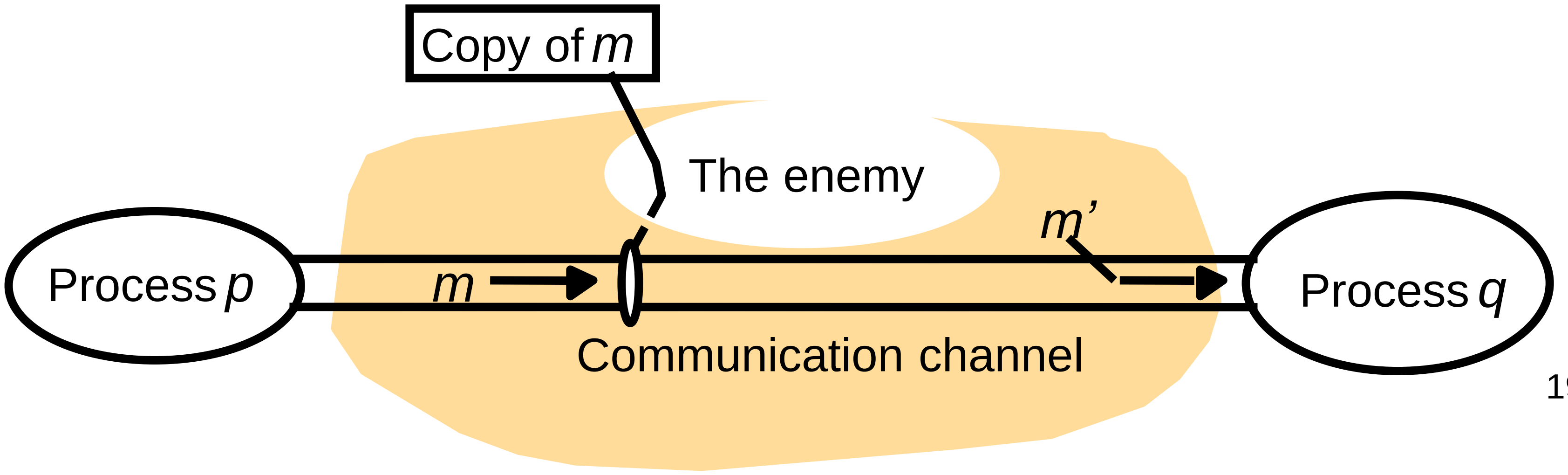
- Protecting objects/resources
  - Uses *access rights* - specify who is allowed to perform the operations of an object/resource
    - E.g., who is allowed to read or to write its state.
  - A principal (the authority issuing access rights) - may be a user or a process.
    - Illustrated example: the invocation comes from a user and the result from a server.



Security model: Securing processes and their interactions

- Enemy model:
  - Eavesdropping via message interception, repeated trials of network access.
  - Threats to processes:
    - inserting and forwarding incorrect IP addresses in message to servers or clients to confuse or disguise the enemy-sender.
  - Threats to communication channels:
    - copying, altering or inserting incorrect data into message streams to deceive or replicate unauthorized transactions (e.g. banking)

- These threats can be defeated by the use of *secure channels*
  - *which are based on cryptography and authentication*



Secure Channels

- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing.
  - This enables the server to protect its objects correctly and allows the client to be sure that is receiving results from a bona fide server.
- Ensure the privacy and integrity (protection against tampering) of the data transmitted across it.
- Each message includes a physical or logical time stamp to prevent messages from being replayed or reordered.
- Examples:
  - Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols which provide a secure channel between two communication peers.

Confidential group communication

- All group members share the **same** secret key
  - Used to encrypt and decrypt all messages transmitted between group members.
  - All members are trusted to indeed keep the key a secret
  - Vulnerable to attacks compared to two-party secure channels.

- Use a **separate** shared secret key between each pair of group members
  - As soon as one member turns out to be leaking information, the others can stop sending messages to that member, but still use the keys they were using to communicate with each other.

  - Instead of having to maintain one key, it is now necessary to maintain $N(N-1)/2$ keys, which may be a difficult problem by itself.

  - Using a **public-key cryptosystem** can improve matters
    - In that case, each member has its own *(public key, private key)*, pair, in which the public key can be used by all members for sending confidential messages. In this case, a total of *N* key pairs are needed. If one member ceases to be trustworthy, it is simply removed from the group without having been able to compromise the other keys.
  - Key management
    - Establishing and distributing keys is not a trivial matter.

Public key cryptography

- Does not require the parties to share a secret key.
- A message encrypted with your *private* key can be decrypted only with your *public* key.
  - Anyone can perform the decryption - you are the only one who could do the encryption
    - authentication
- A message encrypted with your *public* key can be decrypted only with your *private* key.
  - Anyone can do the encryption - you are the only one that is able to decrypt the message
    - confidentiality and secure communication.
- Public key encryption algorithms examples:
  - RSA (Rivest–Shamir–Adleman) and ECC (Elliptic Curve Cryptography).

Public Key Infrastructure

- Public Key Infrastructure (PKI) is a term used for a framework that enables secure exchange of information based on public key cryptography.

- It allows identities (of people, organizations, etc.) to be bound to digital certificates and provides a means of verifying the authenticity of certificates.

- PKI encompasses
  - Keys
  - Certificates
  - Public key encryption, and
  - trusted Certification Authorities (CAs) who generate and digitally sign certificates.

What Applications use Certificates?

- Web browsers
  - X.509 certificates with Transport Layer Security (TLS)
  - Code-signing schemes
    - signed Java ARchives, and Microsoft Authenticode.
  - Secure E-Mail standards
    - Privacy Enhanced Mail (PEM) and Secure/Multipurpose internet Mail Extensions (S/MIME).

Transport Layer Security (TLS)

- Provides authentication, integrity, and encrypted communication
- Secure communication without prior negotiation or help from 3rd parties
- Free choice of crypto algorithms by client and server
- Communication in each direction can be authenticated, encrypted or both
- Most widely used to secure HTTP interactions for use in Internet commerce and other security-sensitive applications.

TLS

Advantages

- Validates the authenticity of the server
  - if you trust the CA
- Protects integrity of communications
- Protects the privacy of communications

Disadvantages

- Latency for session setup
- Older protocols had weaknesses
- Attackers also use TLS

Indirect Communication

- The essence of indirect communication is to communicate through an intermediary

Usage

- In distributed systems where change is anticipated
  - in mobile environments where users may rapidly connect to and disconnect from the global network – and must be managed to provide more dependable services.

- For event dissemination in distributed systems where the receivers may be unknown and liable to change
  - E.g. in managing event feeds in financial systems

Indirect communication techniques

1. Group communication

   - in which communication is via a group abstraction with the sender unaware of the identity of the recipients

2. Publish-subscribe systems

   - a family of approaches that all share the common characteristic of disseminating events to multiple recipients through an intermediary

3. Message queue systems

   - messages are directed to the familiar abstraction of a queue with receivers extracting messages from such queues

4. Shared memory–based approaches

   -allows a processor to address a memory location at another computer as if it were local memory

Group Communication

- A message is sent to a group and then this message is delivered to all members of the group.
- The sender is not aware of the identities of the receivers.
- Represents an abstraction over multicast communication
- May be implemented over
  - IP multicast; or
  - An equivalent *overlay network* adding significant extra value in terms of
    - managing group membership,
    - detecting failures and
    - providing reliability and ordering guarantees.

Group Communication: Key areas of application

- An important building block for reliable distributed systems
- The reliable dissemination of information to potentially large numbers of clients,
  - E.g. in the financial industry, where institutions require accurate and up-to date access to a wide variety of information sources;
- Support for collaborative applications, where events must be disseminated to multiple users to preserve a common user view
  - E.g. in multiuser games;
- Support for a range of fault-tolerance strategies
  - E.g. the consistent update of replicated data, or
  - the implementation of highly available (replicated) servers;
- Support for system monitoring and management
  - E.g. load balancing strategies.

Group Communication: Programming Model

- The central concept is that of a group with associated group membership
  - processes may join or leave the group.
- Processes can then send a message to this group and have it propagated to all members of the group with certain guarantees in terms of reliability and ordering.
- Thus, group communication implements multicast communication
  - in which a message is sent to all the members of the group by a single operation.
- A process issues only one multicast operation to send a message to each of a group of processes

Publish-Subscribe Systems

- Publish-Subscribe Systems
  - Also known as *distributed event-based systems*

- A publish-subscribe system is a system where
  - *publishers* publish structured events to an event service and
  - *subscribers* express interest in particular events through
  - *subscriptions* which can be arbitrary patterns over the structured events.

- For example, a subscriber could express an interest in all events related to a book, such as the availability of a new edition or updates to the related web site.

- The task of the publish subscribe system is to match subscriptions against published events and ensure the correct delivery of *event notifications*.

- A given event will be delivered to potentially many subscribers, and hence publish-subscribe is fundamentally a one-to-many communications paradigm.

Publish-subscribe systems

- Used in a wide variety of application domains, particularly those related to the large-scale dissemination of events.

- Examples:
  - financial information systems;
  - other areas with live feeds of real-time data (including RSS feeds);
  - support for cooperative working, where a number of participants need to be informed of events of shared interest;
  - support for ubiquitous computing, including the management of events emanating from the ubiquitous infrastructure (for example, location events)
  - a broad set of monitoring applications, including network monitoring in the Internet.

- Publish-subscribe is also a key component of Google’s infrastructure, including for example the dissemination of events related to advertisements, such as ‘ad clicks’, to interested parties.

Publish-subscribe systems: The programming model

- Publishers disseminate an event *e* through a *publish(e)* operation and subscribers express an interest in a set of events through subscriptions.

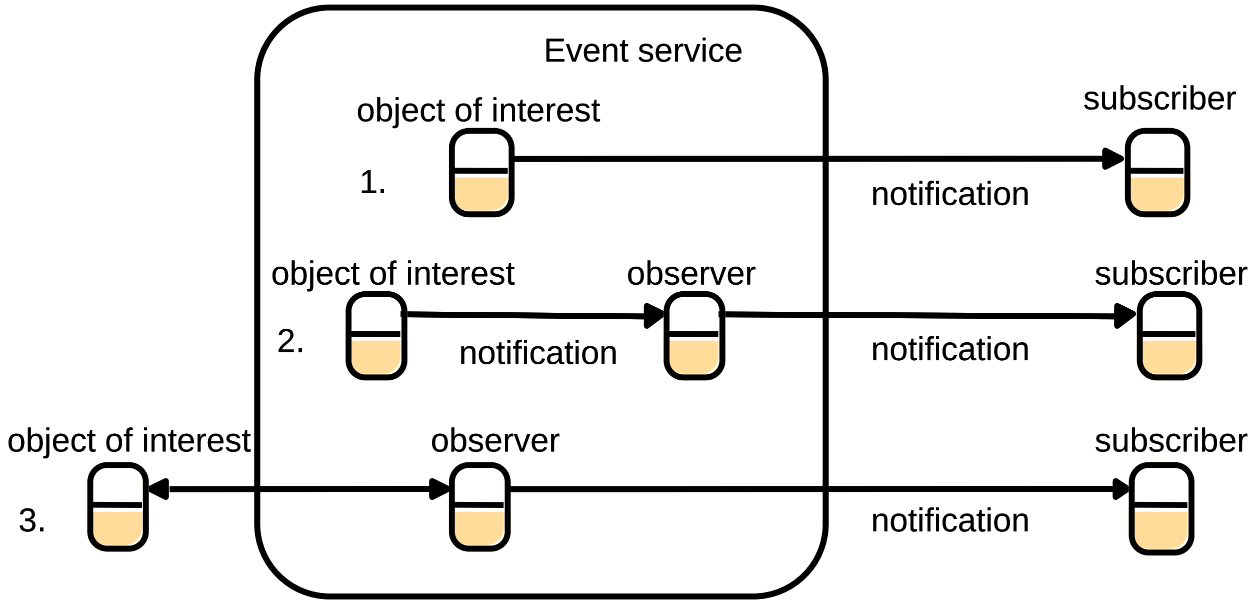
- They achieve this through a *subscribe(f)* operation where *f* refers to a filter
  - that is, a pattern defined over the set of all possible events.
  - The expressiveness of filters (and hence of subscriptions) is determined by the subscription model

- Subscribers can later revoke this interest through a corresponding *unsubscribe(f)* operation.

- When events arrive at a subscriber, the events are delivered using a *notify(e)* operation.

Architecture for distributed event notification

1. An object of interest inside the event service without an observer. It sends notifications directly to the subscribers.
2. An object of interest inside the event service with an observer. The object of interest sends notifications via the observer to the subscribers.
3. An object of interest outside the event service. In this case, an observer queries the object of interest in order to discover when events occur. The observer sends notifications to the subscribers.



Message queue systems

- They are point-to-point
  - the sender places the message into a queue, and it is then removed by a single process. Message queues are also referred to as Message-Oriented Middleware.
- A major class of commercial middleware with key implementations
  - IBM’s WebSphere MQ,
  - Microsoft’s MSMQ
  - Oracle’s Streams Advanced Queuing (AQ).
- The main use of such products is to achieve Enterprise Application Integration (EAI)
  - that is, integration between applications within a given enterprise – a goal that is achieved by the inherent loose coupling of message queues.
- They are also extensively used as the basis for commercial transaction processing systems because of their intrinsic support for transactions.

Clock synchronization

- In a centralized system, time is unambiguous.
- When a process wants to know the time, it simply makes a call to the operating system.
- If process A asks for the time, and then a little later process B asks for the time, the value that B gets will be higher than (or possibly equal to) the value A got.
  - It will not be lower.
- In a distributed system, achieving agreement on time is not trivial.

Clock synchronization

- Strongly related to communication between processes is the issue of how processes in distributed systems synchronize.
- Synchronization is all about doing the right thing at the right time.
- A problem in distributed systems, and computer networks in general, is that there is no notion of a globally shared clock
  - Processes on different machines have their own idea of what time it is.

Clock synchronization

- Physical clocks
  - Keep time of day
  - Consistent across systems, but
    - tend not to be in perfect agreement
    - clock drift, which means that they count time at different rates, and so diverge.

- Logical clocks
  - Keeps track of event ordering

Physical clocks

- Problem
  - Sometimes we simply need the exact time, not just an ordering
  - E.g. file compilation using `make`
    - If the source file `input.c` has time 2151 and the corresponding object file input.o has time 2150, `make` knows that input.c has been changed since `input.o` was created, and thus `input.c` must be recompiled.
    - If `output.c` has time 2144 and `output.o` has time 2145, no compilation is needed

- Solution
  - Universal Coordinated Time (UTC)
    - Based on the number of transitions per second of the cesium 133 atom
    - At present, the real time is taken as the average of some 50 cesium clocks around the world.

- UTC is broadcast through short-wave radio and satellite.
  - Satellites can give an accuracy of about +/- 0:5 ms
  - Satellite sources include the Global Positioning System (GPS).

Logical clocks

- What usually matters is not that all processes agree on exactly what time it is, but
  - that they agree on the order in which events occur
- Requires a notion of ordering
- A logical clock measures the passing of time in terms of logical operations, not the physical time

Logical Clocks

- Assign sequence numbers to messages
  - All cooperating processes agree on order of events
- Assumptions
  - No central time source
    - Each system maintains its own local clock
    - No total ordering of events
  - Multiple processes, each one of them:
    - Has unique IDs
    - Has its own incrementing counter

Logical Clocks

- Main types
  - Lamport’s logical clocks
  - Vector clocks

Logical clocks – Lamport logical clocks

- Lamport clocks
  - Allow processes to assign sequence numbers, so called *Lamport timestamps*, to messages and other events
    - all cooperating processes can agree on the order of related events.

- A monotonically increasing software counter
  - Whose value does not have particular relationship to any physical clock.

Vector clocks

- Vector clocks
  - Developed to overcome the shortcoming of Lamport’s clocks
- A vector is a logical clock that guarantees that
  - If two operations can be ordered by their logical timestamps, then one must have happened before the other.

- Implemented with an array of counters, one for each process in the system.

Vector Clocks

- A way of identifying which events are causally related
- Guaranteed to get the sequencing correct
- The problem:
  - The size of the vector increases with more actors
    - the entire vector must be stored with the data
  - Comparison takes more time than comparing two numbers
  - If messages are concurrent
    - Application will have to decide how to handle conflicts

Clock synchronisation

- The *happened-before* relation is a partial order on events that reflects a flow of information between them
  - Within a process, or via messages between processes.
- Lamport clocks are counters that are updated in accordance with the happened-before relationship between events.
  - Each event `e`, such as sending or receiving a message, is assigned a globally unique logical `timestamp C(e)` such that when event `a` happened before `b`, `C(a) < C(b).`
- Vector clocks are an improvement/extension on Lamport clocks.
  - If `C(a) < C(b)`, we even know that event `a` *causally* preceded `b`.

WEB SERVICES

Web Services

- Set of protocols by which services can be published, discovered, and used in a technology neutral form
  - Language & architecture independent
- Provide a basis where a client program in one organisation may interact with a server in another organisation without human supervision.
- Based on the ability to use an HTTP request to cause the execution of a program.
  - An result is produced by called program and then returned.

Web Services

- General principles
  - Payloads are text (XML or JSON)
    - Technology-neutral
  - HTTP used for transport
    - Use existing infrastructure: web servers, firewalls, load-balancers

- Web server
  - Provides a basic HTPP service

- Web service
  - Provides a service based on the operation defined in its interface.

- Applications will typically invoke multiple remote services
  - Service Oriented Architecture (SOA)
  - SOA = Programming model

Web Services vs. Distributed Objects

Web Services	Distributed Objects
• Document Oriented • Exchange documents	• Object Oriented • Instantiate remote objects • Request operations on a remote object • Receive result • … • Eventually release the object
• Document design is the key • Interfaces are just a way to pass documents	• Interface design is the key • Data structures just package data
• Stateless computing • State is contained within the documents that are exchanged (e.g., customer ID)	• Stateful computing • Remote object maintains state

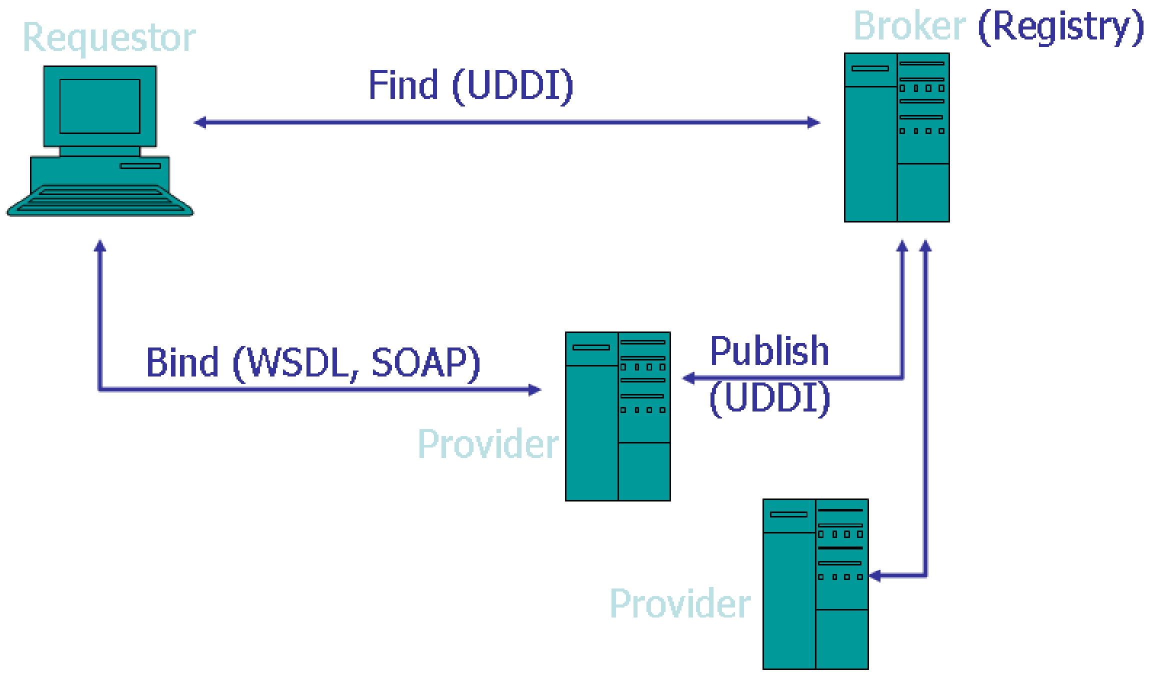
Web Services types

- Main types of web services
  - SOAP web services
    - SOAP is a protocol
  - RESTful web services
    - REST an architectural style not a protocol.

SOAP Web Services Architecture

Architecture needs to cater for:

- Providing a list of available services
  - E.g.what credit checking web services are available for a mortgage service to use?)
- Providing a description or specification for the service
  - (e.g.… what parameters does the selected credit check service expect.. client name/PPS number?, what does it return? …approval rating?)
- Sending messages between the client (I.e. *requestor* of the service) and service…
  - (… message to credit rating service from client application, containing client name etc.?)



Technologies

Many implementations/frameworks for web services but common technologies:

- **SOAP** (Simple Object Access Protocol)
  - provides a way to communicate between applications running on different operating systems, with different technologies and programming languages
  - XML based protocol (on HTTP)
- **WSDL** (Web Service Description Language)
  - machine-readable descriptions of Web services interfaces.
  - XML based
- **UDDI (**Universal Description, Discovery and Integration)
  - UDDI provides an interface for publishing and updating information about web services.

Service Descriptions

- Interface definitions needed to allow clients to communicate with services
- Java Remote Interfaces ~= IDL ~= WSDL
- WSDL (Web Services Description Language)
  - Describe operations, a set of services
  - Name, operations, parameters, where to send requests
  - Provide URI, Identify Transport Protocol
  - Organizations exchange WSDL documents
- All the information required by the client
- Describes either
  - Types of messages it can receive
  - Types of operations it can perform

Representational State Transfer (REST)

- The key characteristic of most web services is that they can process XML formatted SOAP messages
  - An alternative is the REST approach
- REST is a web standards based architecture
  - Uses HTTP Protocol for data communication
  - Resource-oriented
    - every component is a resource
    - a resource is accessed by a common interface using HTTP standard methods
- Clients use URLs and the HTTP operations GET, PUT, DELETE and POST to manipulate resources
  - The emphasis is on the manipulation of *data resources* rather than on interfaces.

REST

- When a new resource is created, it has a new URL by which it can be accessed or updated.
  - Clients are supplied with the entire state of a resource instead of calling an operation to get some part of it.

- The Amazon web services may be accessed either by SOAP or by REST

REST

- REST Server
  - provides access to resources

- REST client
  - accesses and presents the resources

- REST resources
  - each resource is identified by URIs/ Global IDs
  - representations of a resource
    - Text, JSON and XML
    - JSON is now the most popular format

RESTful Web Services

- A web service is:
  - A collection of open protocols
  - Standards used for exchanging data between applications or systems
  - Interoperability between different languages (Java and Python) or platforms (Windows and Linux)

- Web services based on REST Architecture are known as RESTful Web Services
  - Use HTTP methods to implement the concept of REST architecture
  - URI (Uniform Resource Identifier) to define a RESTful service
  - Resources representation: JSON

REST

- Describes how resources on web servers should be accessed via the HTTP protocol

- Resource identification through a uniform resource identifier (URI)

HTTP Methods in a REST based architecture

Basic *four* operations (CRUD: *Create, Read, Update, Delete*)

- PUT
  - Create, Used to create a new resource.
- GET
  - Read, Provides a read only access to a resource.
- POST
  - Update, Used to update an existing resource or create a new resource.
- DELETE
  - Delete, Used to remove a resource.

Fifth operation – determine options associated with a resource

- OPTIONS − Query, Used to get the supported operations on a resource.

Examples of REST services

- Various Amazon & Microsoft APIs
- Facebook Graph API
- Yahoo! Search APIs
- Flickr
- Twitter

SOAP vs RESTful Web Services

SOAP

- A protocol
  - comes with strict rules and advanced security features such as built-in ACID compliance and authorization.
- Permits XML data format only
- SOAP can't use REST
- Has higher complexity, and requires more bandwidth and resources, which can lead to slower page load times.

REST

- An architectural style
  - not a protocol
- Can use different messaging formats, such as HTML, CSV, JSON, XML
- REST can use SOAP web services because it is a concept
- Can use any protocol like HTTP, SOAP.
- Consumes fewer resources than SOAP because its messages are typically smaller.- better performance

SOAP versus REST

SOAP	**REST**
• API calls cannot be cached	• API calls can be cached
• Only XML	• Plain text, HTML, XML, JSON, CSV…
• Requires more bandwidth and computing power	• Requires fewer computing resources
• Better security, built-in extensibility, standardized	• Faster, scalability
• Usage: – Banks or payment gateways, PayPal	• Usage: – Facebook, Google, Twitter

Coordination of web services

- The SOAP infrastructure supports single request-response interactions between clients and web services.
- Many applications involve several requests that need to be done in a particular order. The need for web services as clients to be provided with a description of a particular protocol to follow when interacting with other web services.
- For composite web services, a transactions management protocol such as 2PC is required
  - WS-Coordination
- Simpler approach is Web Service choreography
  - Global description of a set of interactions
  - Defines coordination
  - Enhances interaction

Comparison of web services with distributed object model

- A web service has a service interface which can provide operations for accessing and updating the data resource it manages.

- At a superficial level, the interaction between client and server is similar to RMI, where a client uses a remote object reference to invoke an operation in a remote object.
  - For a web service, the client uses a URI to invoke an operation in the resource named by that URI.

Comparison of web services with distributed object model

- Remote object references are not very similar to URIs
  - The URI of a web service can be compared with the remote object reference of a single object.
  - However, in the distributed object model, objects can create remote objects dynamically and return remote reference to them.
  - The recipient of these remote references can use them to invoke operations in the object to which they refer.

Comparison of web services with distributed object model

- Servants
  - In the distributed object model, the server program is generally modelled as a collection of servants (potentially) remote objects.
  - In contrast, web services do not support servants. Therefore, web services applications cannot create servants as and when they are needed to handle different resources.
    - To enforce this situation, the implementation of web service interfaces must not have either constructors or main methods.