Distributed systems
Lecture 11: Object-Oriented Middleware (OOM), clocks and distributed time

Michaelmas 2019
Dr Martin Kleppmann
(With thanks to Dr Robert N. M. Watson and Dr Steven Hand)
The story so far...

• Distributed systems are hard
• Looking at simple **client/server** interaction, and use of **Remote Procedure Call (RPC)**
  – invoking methods on server over the network
  – middleware generates **stub code** which can **marshal / unmarshal** arguments and replies
  – saw case study of NFS (RPC-based file system)
• Other RPC systems (e.g., DCE RPC)
Object-Oriented Middleware

- SunRPC / DCE RPC forward **functions**, and do not support complex types, exceptions, or polymorphism
- **Object-Oriented Middleware (OOM)** arose in the early 90s to address this
  - Assume programmer is writing in OO-style (and language)
  - **Remote objects** will behave like local objects, but their methods will be forwarded over the network a la RPC
  - **References to objects** can be passed as arguments or return values – e.g., passing a directory object reference
  - Promote NFS’s concept of a **handle** into the framework
- Makes it much easier to program – especially if your program is already object oriented!
CORBA (1989)

• First OOM system was CORBA
  – Common Object Request Broker Architecture
  – Specified by the OMG: Object Management Group
• OMA (Object Management Architecture) is the general model of how objects interoperate
  – Objects provide services
  – Clients makes a request to an object for a service
  – Client doesn’t need to know where the object is, or anything about how the object is implemented!
  – Object interface must be known (public)
Object Request Broker (ORB)

• The **ORB** is the core of the architecture
  – Connects clients to object implementations
  – Conceptually spans multiple machines (in practice, ORB software runs on each machine)
Invoking Objects

• Clients obtain an **object reference**
  – Typically via the **naming service** or **trading service**
  – (Object references can also be saved for use later)

• Interfaces defined by **CORBA IDL**

• Clients can call remote methods in 2 ways:
  1. **Static Invocation**: using stubs built at compile time (just like with RPC)
  2. **Dynamic Invocation**: actual method call is created on the fly. It is possible for a client to discover new objects at run time and access the object’s methods
CORBA IDL

• Definition of language-independent remote interfaces
  – **Language mappings** to C++, Java, Smalltalk, ...
  – Translation by **IDL compiler**

• Type system
  – **basic types**: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, ...
  – **constructed types**: struct, union, sequence, array, enum
  – **objects** (common super type **Object**)

• Parameter passing
  – **in, out, inout** (= send remote, modify, update)
  – basic & constructed types passed by value
  – objects passed by reference
CORBA Pros and Cons

• CORBA has some unique advantages
  – Industry standard (OMG)
  – Language & OS agnostic: mix and match
  – Richer than simple RPC (e.g. interface repository, implementation repository, DLL support, …)
  – Many additional services (trading & naming, events & notifications, security, transactions, …)

• However:
  – Really, really complicated / ugly / buzzwordy
  – Poor interoperability, at least at first
  – Generally to be avoided unless you need it!
Microsoft DCOM (1996)

• An alternative to CORBA:
  – MS had invested in COM (object-oriented local IPC scheme) so didn’t fancy moving to OMA

• **Service Control Manager (SCM)** on each machine responsible for object creation, invocation, ...
  – Essentially a lightweight ‘ORB’

• Added **remote operation** using MSRPC:
  – Based on DCE RPC, but extended to support objects
  – Augmented IDL called **MIDL**: DCE IDL + objects
  – Requests include **interface pointer IDs (IPIIDs)** to identify object & interface to be invoked
DCOM vs. CORBA

• Both are language neutral, and object-oriented
• DCOM supports objects with multiple interfaces
  – but not, like CORBA, multiple inheritance of interfaces
• DCOM handles distributed garbage collection:
  – remote objects are reference counted (via explicit calls)
  – ping protocol handles abnormal client termination
• DCOM is widely used (e.g. SMB/CIFS, RDP, ...)
• But DCOM is MS proprietary (not standard)...
  – and no support for exceptions (return-code based)..
  – and lacks many of CORBAs services (e.g. trading)
• Deprecated today in favor of .NET
Java RMI

• 1995: Sun extended Java to allow RMI
  – RMI = Remote Method Invocation
• Essentially an OOM scheme for Java with clients, servers, and an object registry
  – Object registry maps from names to objects
  – Supports bind()/rebind(), lookup(), unbind(), list()
• RMI was designed for Java only
  – No goal of OS or language interoperability
  – Hence cleaner design, tighter language integration
  – E.g., distributed garbage collection
RMI: new classes

• **Remote class:**
  – one whose instances can be used remotely
  – within home address space, a regular object
  – within foreign address spaces, referenced indirectly via an **object handle**

• **Serializable class:** [nothing to do with transactions!]
  – object that can be marshalled/unmarshalled
  – if a serializable object is passed as a parameter or return value of a remote method invocation, the value will be copied from one address space to another
  – (for remote objects, only the object handle is copied)
RMI: new classes

• **Remote class:**
  – one whose instances can be used remotely
  – within home address space, a regular object
  – within foreign address spaces, referenced indirectly via an **object handle**

• **Serializable class:** [nothing to do with transactions!]
  – object that can be **marshalled/unmarshalled**
  – if a serializable object is passed as a parameter or return value of a remote method invocation, the value will be copied from one address space to another
  – (for remote objects, only the object handle is copied)
RMI: the big picture

- Registry can be on server... or one per distributed system
  - client and server can find it via the `LocateRegistry` class
- Objects being serialized are annotated with a URL for the class
  - unless they implement `Remote` => replaced with a remote reference
OOM: summary

• OOM enhances RPC with objects
  – types, interfaces, exceptions, ...

• Seen CORBA, DCOM and Java RMI
  – All plausible, and all still used today
  – CORBA most general (language and OS agnostic), but also the most complex: design by committee
  – DCOM is MS-only; being phased out for .NET
  – Java RMI decent starting point for simple distributed systems... but lacks many features
  – (EJB is a modern CORBA/RMI/<stuff> megalith)
XML-RPC

• Systems seen so far all developed by large industry, and work fine in the local area...
  – But don’t (or didn’t) do well through firewalls ;-
• In 1998, Dave Winer developed XML-RPC
  – Use XML to encode method invocations (method names, parameters, etc)
  – Use HTTP POST to invoke; response contains the result, also encoded in XML
  – Looks like a regular web session, and so works fine with firewalls, NAT boxes, transparent proxies, ...
XML-RPC example

**XML-RPC Request**

```xml
<?xml version="1.0"?>
<methodCall>
  <methodName>util.InttoString</methodName>
  <params>
    <param>
      <value><i4>55</i4></value>
    </param>
  </params>
</methodCall>
```

**XML-RPC Response**

```xml
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>Fifty Five</string></value>
    </param>
  </params>
</methodResponse>
```

- Client side names method (as a string), and lists parameters, tagged with simple types
- Server receives message (via HTTP), decodes, performs operation, and replies with similar XML
- Inefficient & weakly typed... but simple, language agnostic, extensible, and eminently practical!
SOAP & web services

• XML-RPC was a victim of its own success
• WWW consortium decided to embrace it, extend it, and generally comply it up
  – SOAP (Simple Object Access Protocol) is basically XML-RPC, but with more XML bits
  – Support for namespaces, user-defined types, multi-hop messaging, recipient specification, ...
  – Also allows transport over SMTP (!), TCP & UDP
• SOAP is part of the Web Services world
  – As complex as CORBA, but with more XML ;-(
Moving away from RPC

• SOAP 1.2 defined in 2003
  – Less focus on RPC, and more on moving XML messages from A to B (perhaps via C & D)
• One major problem with all RPC schemes is that they were **synchronous**:
  – Client is blocked until server replies
  – Poor responsiveness, particularly in wide area
• 2006 saw introduction of AJAX
  – **Asynchronous Javascript with XML**
  – Chief benefit: can update web page without reloading
• Examples: Google Maps, Gmail, Google Docs, ...
• AJAX still does RPC (just asynchronously)
• Is a procedure call / method invocation really the best way to build distributed systems?
• **Representational State Transfer** (REST) is an alternative ‘paradigm’ (or a throwback?)
  – Resources have a name: URL or URI
  – Manipulate them via POST (create), GET (select), PUT (create/overwrite), and DELETE (delete)
  – More recently added: PATCH (partial update in place)
  – Send state along with operations
• Very widely used today (Amazon, Flickr, Twitter)
Client-server interaction: summary

• Server handles requests from client
  – Simple request/response protocols (like HTTP) useful, but lack language integration
  – RPC schemes (SunRPC, DCE RPC) address this
  – OOM schemes (CORBA, DCOM, RMI) extend RPC to understand objects, types, interfaces, exns, ...

• Recent WWW developments move away from traditional RPC/RMI:
  – Avoid explicit IDLs since can slow evolution
  – Enable asynchrony, or return to request/response
Clocks and distributed time

• Distributed systems need to be able to:
  – **order events** produced by concurrent processes;
  – **synchronize** senders and receivers of messages;
  – **serialize** concurrent accesses to shared objects; and
  – generally **coordinate** joint activity

• This can be provided by some sort of **clock**:
  – **physical clocks** keep time of day
    • (must be kept consistent across multiple nodes – why?)
  – **logical clocks** keep track of event ordering

• NB. Clock in digital electronics (**oscillator**) ≠ clock in distributed systems (**source of timestamps**)
Physical clock technology

• Quartz Crystal Clocks (1929)
  – resonator shaped like a tuning fork
  – laser-trimmed to vibrate at 32,768 Hz
  – standard resonators accurate to 6ppm at 31°C... so will gain/lose around 0.5 seconds per day
  – stability better than accuracy (about 2s/month)
  – best resonators get accuracy of ~1s in 10 years

• Atomic clocks (1948)
  – count transitions of the cesium 133 atom
  – 9,192,631,770 periods defined to be 1 second
  – accuracy is better than 1 second in 6 million years...
  – relativity can’t be ignored: think satellites
Coordinated Universal Time (UTC)

- Physical clocks provide **ticks** but we want to know the actual time of day
  - determined by astronomical phenomena
- Several variants of universal time
  - **UT0**: mean solar time on Greenwich meridian
  - **UT1**: UT0 corrected for polar motion; measured via observations of quasars, laser ranging, & satellites
  - **UT2**: UT1 corrected for seasonal variations
  - **UTC**: civil time, tracked using atomic clocks, but kept within 0.9s of UT1 by occasional leap seconds
Computer clocks

• Typically have a **Real-Time Clock (RTC)**
  – CMOS clock driven by a quartz oscillator
  – battery-backed so continues when power is off
• Also have range of other clocks (PIT, ACPI, HPET, TSC, ...), mostly **higher frequency**
  – free running clocks driven by quartz oscillator
  – mapped to real time by OS at boot time
  – programmable to generate interrupts after some number of ticks (~= some amount of real time)
Operating-system use of clocks

• OSes use time for many things
  – Periodic events – e.g., time sharing, statistics, at, cron
  – Local I/O functions – e.g., peripheral timeouts; entropy
  – Network protocols – e.g., TCP DELACK, retries, keep-alive
  – Cryptographic certificate/ticket generation, expiration
  – Performance profiling and sampling features

• Ticks trigger interrupts
  – Historically, timers at fixed intervals (e.g., 100Hz)
  – Now, tickless: timer reprogrammed for next event
  – Saves energy, CPU resources – especially as cores scale up

Which of these require physical time vs logical time? What will happen to each if the real-time clock drifts or steps due to synchronization?
• Object-Oriented Middleware (OOM)
  – CORBA, DCOM, RMI, XML-RPC, SOAP, REST
• Clocks and distributed time
  – Physical clock technology, UTC
  – What clocks in computers are for...

• More on physical time
• Time synchronization
• Ordering
  – The “happens-before” relation
  – Logical and vector clocks