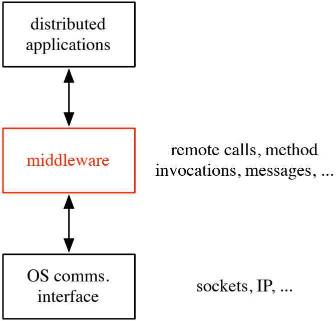
DS 2009: middleware

David Evans

de239@cl.cam.ac.uk

What is middleware?



- layer between OS and distributed applications
- hides complexity and heterogeneity of distributed system
- bridges gap between low-level OS comms and programming language abstractions
- provides common programming abstraction and infrastructure for distributed applications

Middleware properties

- middleware provides support for (some of)
    - naming, location, service discovery, replication
    - protocol handling, communication faults, QoS
    - synchronisation, concurrency, transactions, storage
    - access control, authentication
- middleware dimensions

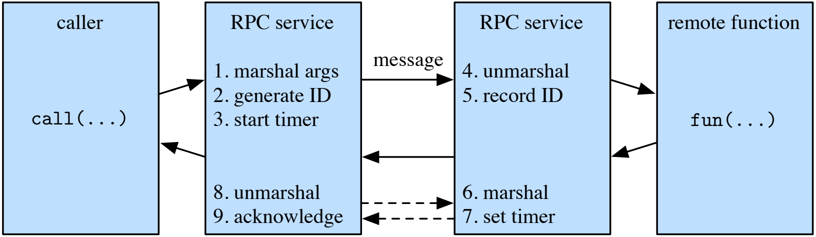
request/reply	vs.	asynchronous messaging
language-specific	vs.	language-independent
proprietary	vs.	standards-based
small-scale	vs.	large-scale
tightly-coupled	vs.	loosely-coupled components

Approaches to middleware

- Remote Procedure Call (RPC)
  - historic interest, but can still be very useful
- Object-Oriented Middleware (OOM)
  - Java RMI
  - CORBA
  - reflective middleware
- Message-Oriented Middleware (MOM)
  - Java Message Service
  - IBM MQSeries
  - Web Services
- Event-Based Middleware
  - Cambridge Event Architecture
  - Hermes

RPC: overview

- makes remote function calls look local
- client/server model
- request/reply paradigm usually implemented with message passing in RPC service
- marshalling of function parameters and return value



Properties of RPC

- language-level pattern of function call
  - easy to understand for programmer
- synchronous request/reply interaction
  - natural from a programming language point of view
  - matches replies to requests
  - built in synchronisation of requests and replies
- distribution transparency (in the no-failure case)
  - hides the complexity of a distributed system
- various reliability guarantees
  - deals with some distributed systems aspects of failure

Failure modes of RPC

- invocation semantics supported by RPC in the light of
  - networkand/or server congestion
  - client, network, and/or server failure
- at most once (RPC system tries once)
  - error return—programmer may retry
- exactly once (RPC system retries a few times)
  - hard error return—some failure most likely

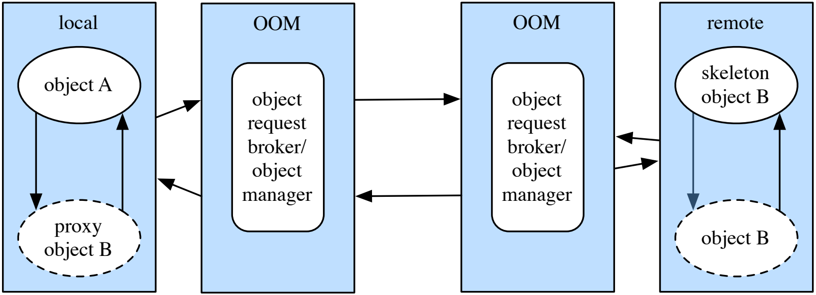
(note that “exactly once” cannot be guaranteed)

Disadvantages of RPC

- synchronous request/reply interaction
  - tight coupling between client and server
  - may block for a long time
  - leads to multi-threaded programming at client and, especially, server
- distribution transparency
  - not possible to mask all problems
- lacks notion of service
  - programmer may not be interested in specific servers
- RPC paradigm is not object-oriented
  - invoke functions on servers as opposed to methods on objects

Object-Oriented Middleware (OOM)

- objects can be local or remote
- object references can be local or remote
- remote objects have visible remote interfaces
- makes remote objects look local using proxy objects


local
object A
proxy object B
OOM
object request broker/ object manager
OOM
object request broker/ object manager
remote
skeleton object B
object B

Properties of OOM

- support for object-oriented programming model
  - objects, methods, interfaces, encapsulation, …
  - exceptions (also in some RPC systems, *e.g.*, Mayflower)
- location transparency
  - system maps object references to locations
- synchronous request/reply interaction
  - same as RPC
- services
  - easier to build using object concepts

Java Remote Method Invocation (RMI)

- remote methods in Java

  ```
  public interface PrintService extends
     Remote {
    int print(Vector printJob) throws
       RemoteException;
  }
  ```

- RMI compiler creates proxies and skeletons
- RMI istry used for interface lookup
- everything has to be in Java, unless you like pain (single-language system)

CORBA

- Common Object Request Broker Architecture
  - open standard by the OMG
  - language and platform independent
- Object Request Broker (ORB)
  - General Inter-ORB Protocol (GIOP) for communication
  - Interoperable Object References (IOR) contain object location
  - CORBA Interface Definition Language (IDL)
  - stubs (proxies) and skeletons created by IDL compiler
  - dynamic remote method invocation
- Interface Repository
  - querying existing remote interfaces
- Implementation Repository
  - activating remote objects on demand

CORBA IDL

- definition of language-independent remote interfaces
  - language mappings to C++, Java, Smalltalk, . . .
  - translation by IDL compiler
- type system
  - **basic**: long (32 bit), long long (64 bit), short, float, char, boolean, octet, any, . . .
  - **constructed**: struct, union, sequence, array, enum
  - **objects**: common super type `Object`
- parameter passing
  - in, out, inout
  - basic & constructed types passed by value
  - objects passed by reference

```
typedef sequence<string> Files;
interface PrintService : Server {
  void print(in Files printJob);
};
```

CORBA services

- naming service
  - names → remote object references
- trading service
  - attributes (properties) → remote object references
- persistent object service
  - implementation of persistent CORBA objects
- transaction service
  - making object invocation a part of transactions
- event service and notification service
  - asynchronous communication based on messaging (cf. MOM); not an integrated programming model with general IDL messages

Disadvantages of OOM

- synchronous request/reply interaction only
  - so CORBA onewaysemantics added
  - Asynchronous Method Invocation (AMI); can be yucky
  - but implementations may not be loosely coupled
- distributed garbage collection
  - releasing memory for unused remote objects
- OOM rather static and heavy-weight
  - bad for ubiquitous systems and embedded devices

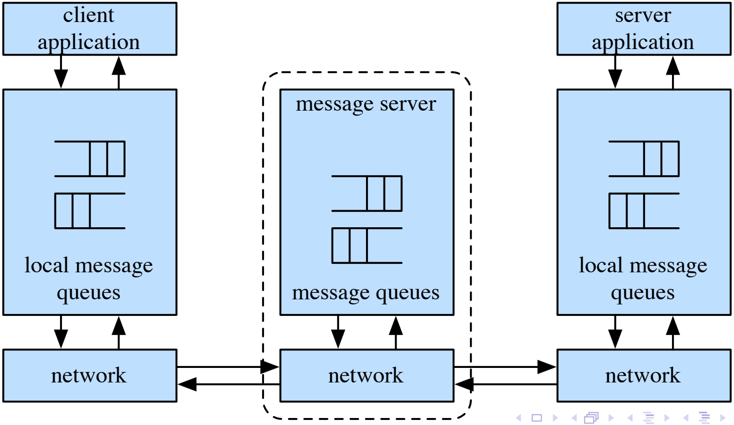
Reflective middleware

OOM with

- interfaces for reflection
  - objects can inspect middleware behaviour
- interfaces for customisability
  - dynamic reconfiguration depending on environment
  - different protocols, QoS, . . . ; *e.g.*, use different marshalling strategy over unreliable wireless link

Message-Oriented Middleware (MOM)

- communication using messages
- messages stored in message queues
- optional message servers decouple client and server
- various assumptions about message content



Properties of MOM

- asynchronous interaction
  - client and server are only loosely coupled
  - messages are queued
  - good for application integration
- support for reliable delivery service
  - keep queues in persistent storage
- processing of messages by intermediate message server
  - may do filtering, transforming, logging, …
  - networks of message servers
- natural for database integration

IBM MQSeries

(probably since called WebSphere MQ Awesomeness. . .)

- one-to-one reliable message passing using queues
  - persistent and non-persistent messages
  - message priorities, message notification
- Queue Managers
  - responsible for queues
  - transfer messages from input to output queues
  - keep routing tables
- Message Channels
  - reliable connections between queue managers
- messaging API

`MQopen`	open a queue
`MQclose`	close a queue
`MQput`	put message into opened queue
`MQget`	get message from local queue

Java Message Service (JMS)

- API specification to access MOM implementations
- Two modes of operation specified
  - **point-to-point**, one-to-one communication using queues
  - **publish/subscribe**, see Event-Based Middleware
- JMS Server implements JMS API
- JMS Clients connect to JMS servers
- Java objects can be serialised to JMS messages
- a JMS interface has been provided for MQ

Disadvantages of MOM

- poor programming abstraction (but has evolved)
  - rather low-level
  - request/reply awkward
  - can lead to multi-threaded code
- message formats unknown to middleware
  - no type checking (JMS addresses this—implementation?)
- queue abstraction only gives one-to-one communication
  - limits scalability (JMS pub/sub. . . ?)

Web services

use well-known web standards for distributed computing

- communication
  - message content expressed in XML
  - Simple Object Access Protocol (SOAP): a lightweight protocol for sync/async communication
- service description
  - Web Services Description Language (WSDL): interface description for web services
- service discovery
  - Universal Description Discovery and Integration (UDDI): directory with web service descriptions in WSDL

Properties of web services

- language-independent and open standard
- SOAP offers OOM and MOM-style communication
  - synchronous request/reply like OOM
  - asynchronous messaging like MOM
  - supports Internet transports (http, smtp, . . .)
  - uses XML Schema for marshalling types to/from programming language types
- WSDL says how to use a web service
  - `http://api.google.com/GoogleSearch.wsdl`
- UDDI helps to find the right web service
  - exports SOAP API for access

Disadvantages of web services

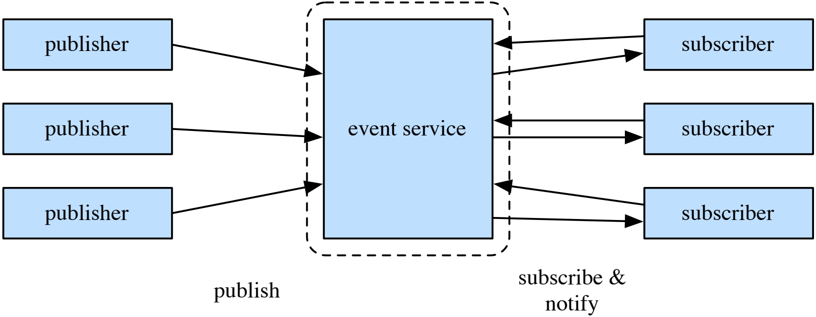
- low-level abstraction
  - leaves a lot to be implemented
- interaction patterns have to be built
  - one-to-one and request-reply provided
  - one-to-many?
  - still service invocation, rather than notification
  - nested/grouped invocations, transactions, . . .
- location transparency—depend on DNS?

What we lack so far

- general interaction patterns
  - we have one-to-one and request-reply
  - one-to-many? many to many?
  - notification?
  - dynamic joining and leaving?
- location transparency
  - anonymity of communicating entities
- support for pervasive computing
  - data values from sensors

Event-based middleware, aka publish/subscribe

- publishers (advertise and) publish events (messages)
- subscribers express interest in events using subscriptions
- event service notifies interested subscribers of published events
- events can have arbitrary content (typed) or name/value pairs



Topic-based and content-based pub/sub

- event service matches events against subscriptions
- topic-based
  - publishers publish events belonging to topic or subject
  - subscribers subscribe to topic
    `subscribe(PrintJobFinishedTopic, ...)`
- (topic and) content-based
  - publishers publish events belonging to topics
  - subscribers provide a filterbased on contentof events
    `subscribe(type=printjobfinshed, printer=“aspen”, ...)`

Properties of publish/subscribe

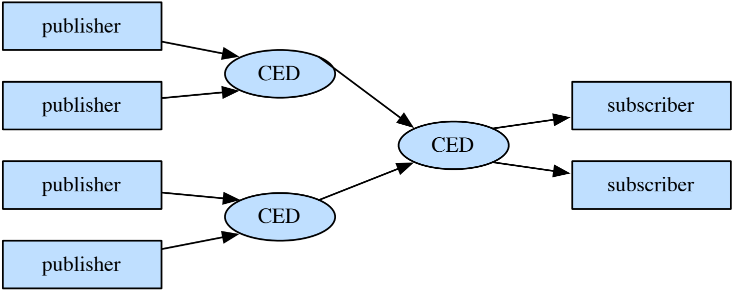
- asynchronous communication
  - publishers and subscribers are loosely coupled
- many-to-many interaction between pubs and subs
  - scalable scheme for large-scale systems
  - publishers do not need to know subscribers, and vice-versa
  - dynamic join and leave of pubs, subs, (brokers—see later)
- (topic and) content-based pub/sub very expressive
  - filtered information delivered only to interested parties
  - efficient content-based routing through a broker network

P/S leads to Composite Event Detection (CED)

- content-based pub/sub may not be expressive enough
  - potentially thousands of event types (primitive events)
  - subscribers interest: event patterns (define high-level events)

  `PrinterOutOfPaperEvent` or `PrinterOutOfTonerEvent`

- Composite Event Detectors (CED)
  - subscribe to primitive events and publish composite events



Middleware: summary

- middleware is an important abstraction for building distributed systems
  1. Remote Procedure Call
  2. Object-Oriented Middleware
  3. Message-Oriented Middleware
  4. Event-Based Middleware
- synchronous vs. asynchronous communication
- scalability, many-to-many communication
- language integration
- ubiquitous systems, mobile systems