Chapter 2: Middleware

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
http://www.iks.inf.ethz.ch/

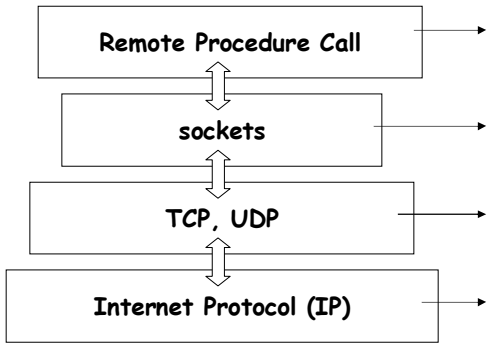
Contents - Chapter 2

- Understanding middleware
  - Middleware as a programming abstraction
  - Middleware as infrastructure
- Overview of conventional middleware platforms
  - RPC
  - TP Monitors
  - Object brokers
- Middleware convergence

Programming abstractions

- Programming languages and almost any form of software system evolve always towards higher levels of abstraction
  - hiding hardware and platform details
  - more powerful primitives and interfaces
  - leaving difficult task to intermediaries (compilers, optimizers, automatic load balancing, automatic data partitioning and allocation, etc.)
  - reducing the number of programming errors
  - reducing the development and maintenance cost of the applications developed by facilitating their portability
- Middleware is primarily a set of programming abstractions developed to facilitate the development of complex distributed systems
  - to understand a middleware platform one needs to understand its programming model
  - from the programming model the limitations, general performance, and applicability of a given type of middleware can be determined in a first approximation
  - the underlying programming model also determines how the platform will evolve and fare when new technologies evolve

RPC as a programming abstraction



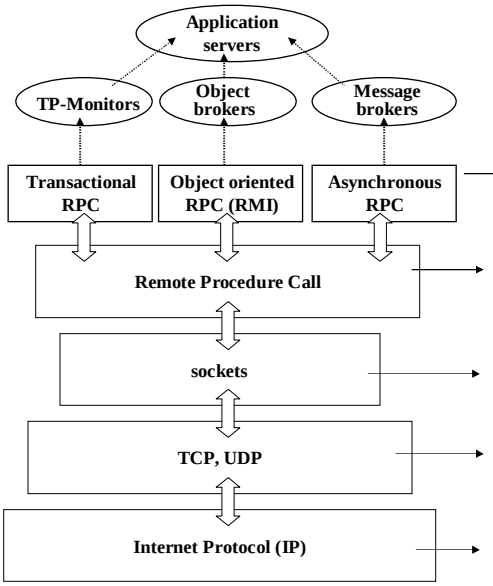
Remote Procedure Call:
hides communication details behind a procedure call and helps bridge heterogeneous platforms

sockets:
operating system level interface to the underlying communication protocols

TCP, UDP:
User Datagram Protocol (UDP) transports data packets without guarantees
Transmission Control Protocol (TCP) verifies correct delivery of data streams

Internet Protocol (IP):
moves a packet of data from one node to another

The genealogy of middleware



Specialized forms of RPC, typically with additional functionality or properties but almost always running on RPC platforms

Remote Procedure Call: hides communication details behind a procedure call and helps bridge heterogeneous platforms

sockets:
operating system level interface to the underlying communication protocols

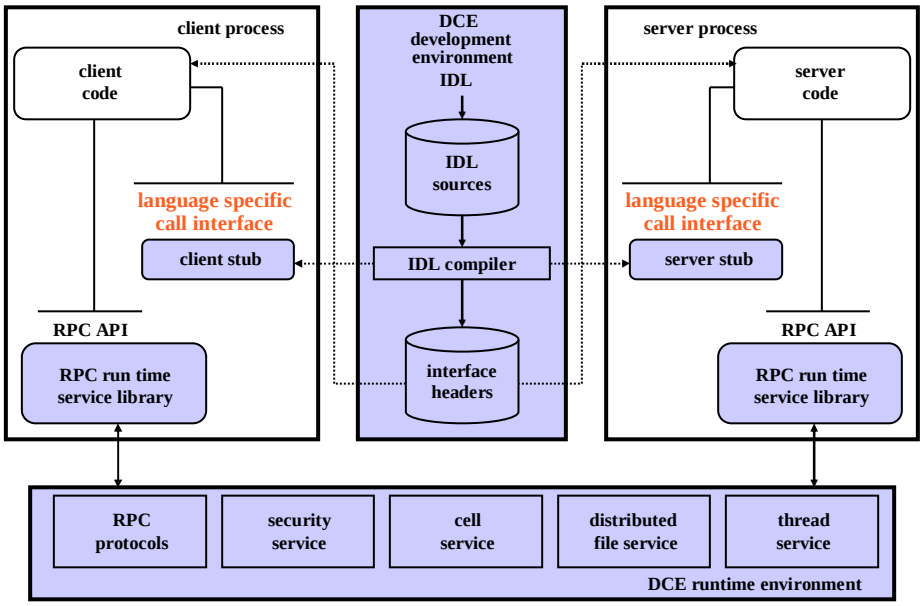
TCP, UDP:
User Datagram Protocol (UDP) transports data packets without guarantees
Transmission Control Protocol (TCP) verifies correct delivery of data streams

Internet Protocol (IP):
moves a packet of data from one node to another

And the Internet? And Java?

- Programming abstractions are a key part of middleware but not the only one:
  - a programming abstraction without good supporting infrastructure (i.e., a good implementation and support system underneath) does not help
- Programming abstractions, in fact, appear I many cases in reaction to changes in the underlying hardware or the nature of the systems being developed
- Java is a programming language that abstracts the underlying hardware: programmers see only the Java Virtual Machine regardless of what computer they use
  - code portability (not the same as code mobility)
  - the first step towards standardizing middleware abstractions (since now the can be based on a virtual platform everybody agrees upon)
- The Internet is a different type of network that requires one more specialization of existing abstractions:
  - The Simple Object Access Protocol (SOAP) of Web services is RPC wrapped in XML and mapped to HTML for easy transport through the Internet

Middleware as infrastructure



Infrastructure

- As the programming abstractions reach higher and higher levels, the underlying infrastructure implementing the abstractions must grow accordingly
  - Additional functionality is almost always implemented through additional software layers
  - The additional software layers increase the size and complexity of the infrastructure necessary to use the new abstractions
- The infrastructure is also intended to support additional functionality that makes development, maintenance, and monitoring easier and less costly
  - RPC => transactional RPC => logging, recovery, advanced transaction models, language primitives for transactional demarcation, transactional file system, etc.
  - The infrastructure is also there to take care of all the non-functional properties typically ignored by data models, programming models, and programming languages: performance, availability, recovery, instrumentation, maintenance, resource management, etc.

Understanding middleware

To understand middleware, one needs to understand its dual role as programming abstraction and as infrastructure

PROGRAMMING ABSTRACTION

- Intended to hide low level details of hardware, networks, and distribution
- Trend is towards increasingly more powerful primitives that, without changing the basic concept of RPC, have additional properties or allow more flexibility in the use of the concept
- Evolution and appearance to the programmer is dictated by the trends in programming languages (RPC and C, CORBA and C++, RMI and Java, Web services and SOAP-XML)

INFRASTRUCTURE

- Intended to provide a comprehensive platform for developing and running complex distributed systems
- Trend is towards service oriented architectures at a global scale and standardization of interfaces
- Another important trend is towards single vendor software stacks to minimize complexity and streamline interaction
- Evolution is towards integration of platforms and flexibility in the configuration (plus autonomic behavior)

Next topics

- RPC
- TP Monitors
- Object brokers