v1.0

Sistemi Concorrenti e di Rete LS

II Facoltà di Ingegneria - Cesena

a.a 2008/2009

[module 3.1]

DISTRIBUTED PROGRAMMING OVERVIEW

DISTRIBUTED PROGRAMMING

- *Distributed* programming
  - the overall computation is realized by means of a set of sequential processes that are executed concurrently (in parallel) by processors distributed over the nodes of a network, communicating by exchanging messages through the network links
  - the model is referred to the (abstract) concurrent machine
    - independently from the architecture of the physical machine
- Message Passing model
  - no shared memory is available among the processors
  - also called *distributed memory model*
- Key aspects
  - concurrency of the components
  - decentralization & lack of global clock
    - no total order among the events of the system
    - specific algorithms to recover global properties
  - independent failures of the components

CHALLENGES (1/4)

- **Heterogeneity & openness**
  - heterogeneity of elements composing distributed concurrent systems
    - networks, computers, OS, programming languages, implementation...
  - openness
    - dynamic change of system structure & dynamics
  - importance role of:
    - middlewares
    - virtual Machines
    - standards & public interfaces
- **Security**
  - Confidentiality
    - protection against the disclosure of unauthorized individuals
  - Integrity
    - protection against alteration or corruption
  - Availability
    - protection against interference with the means to access the resource

CHALLENGES (2/4)

- **Scalability**
  - designing systems that remain effective when there is a significant increase in the number of resources & users
    - e.g. Internet
  - design challenges
    - controlling the cost of physical resources
      - adding dynamically resources by need
    - controlling the performance loss
    - preventing software resources running out
      - e.g. IP numbers...
    - avoiding performance bottleneck
- **Failure Handling**
  - aspects
    - *detecting* failures*, masking* failures*, tolerating* failures, *recovery* from failures
  - measure
    - **availability** = proportion of time that the system is available for use
  - techniques
    - redundancy (replication)

CHALLENGES (3/4)

- **Transparency**
  - system conceived as a (centralized) whole rather than a collection of independent components
    - users / programmers perspective
  - different kinds of transparency
    - access transparency
      - local interfaces = remote interfaces
    - location transparency
      - no need of knowing addresses
    - concurrency transparency
      - no need to know that there is concurrency
    - replication transparency
      - no need to know that there are replica
    - failure transparency
      - no need to deal with failures / recovery
    - mobility transparency
      - no need to change programs & co if resources / users move
    - scaling transparency
      - no need to change programs / applications when scaling up / down

CHALLENGES (4/4)

- **Decentralized concurrency control**
  - distributed mutual exclusion
  - elections & consensus
  - transactions

MAIN MODELS IN DISTRIBUTED PROGRAMMING / SYSTEMS

- **Interaction** model
- **Failure** model
- **Security** model

INTERACTION MODEL

- Main aspects
  - process communication and coordination
  - communication protocols for distributed algorithms
- *Inter-Process Communication*
  - from shared-memory architecture to multi-processing & distributed architectures
    - decentralization
    - synchronization more based on communication than sharing
  - communication as information exchange
    - sending and receiving messages
    - sender process and receiver process
    - integration with synchronization & coordination mechanisms
- Basic types of communication
  - *asynchronous*
  - *synchronous*
  - *rendez-vous* (rpc)

SYNCHRONOUS / ASYNCHRONOUS COMMUNICATION

- *Synchronous* communication
  - exchange of a message as an *atomic action* between a sender and a receiver
    - e.g. telephone calls
    - typically two-way data flow
  - blocking behaviour
    - sender ready and receiver not ready
    - sender not ready and receiver ready
  - > *communication acts as basic synchronization mechanism*
    - no support required other than send and receive primitives
- *Asynchronous* communication
  - no temporal dependence between the execution sequence of the processes
    - e.g. emails
    - one-way data flow
  - need of proper communication media
    - buffering capability

ADDRESSING

- Asymmetric
  - the receiver does not know the sender
  - e.g. telephone call
- Symmetric
  - the receiver knows the address of the sender
  - e.g. emails

BASIC INTERACTION MODELS

- *Channel*-based
  - processes directly 'connected' through **channels**
  - strongly coupled / control-driven communication model
- *Space*-based
  - processes communicate through **space** abstraction
  - loosely coupled / data-driven communication model

FAILURE MODEL

- Both processes and communication channels may *fail*
  - *Omission* failures
    - a process or a communication channel fails to perform actions that it is supposed to do
  - *Arbitrary* or *Byzantine* failures
    - a process arbitrarily omits intended processing steps or takes unintended processing steps the worst ones
  - *Timing* failures
    - violating timing constraints
      - e.g. in synchronous sytstems, in real-time systems

SECURITY MODEL

- Protecting objects
  - access control & authorization
- Securing processes and their interactions against threats
  - authentication
  - secure channels
  - use of cryptography and shared secrets

MIDDLEWARES FOR DISTRIBUTED COMPUTING

- Software Layers

Applications, services
Middleware / Software infrastructures
Operating Systems
Computer and network hardware

  - *Platform* = typically Application + Middleware

- *Middleware*
  - a layer of software whose purpose is to mask heterogeneity and provide a convenient programming model to application programmers
- *Infrastructure*
  - (software) layer providing some kinds of functionality that can be exploited as a service
  - typically distributed
  - often used as a synonim of middleware

MIDDLEWARE EXAMPLES

- Procedure oriented
  - RPC
- Object / component oriented:
  - CORBA
  - Java RMI
  - Microsoft DCOM, .NET
  - ...
- Service oriented:
  - Web Services
  - OSGi (?)
  - JXTA (Peer-to-peer)
  - Jini
- Agent oriented
  - JADE
  - Coordination middleware (JavaSpaces, TuCSoN,...)
  - Mobile agents: SOMA (DEIS Bologna)

ARCHITECTURES FOR DISTRIBUTED SYSTEMS

- Defining the division of responsibilities between components and their placement on computer in the network
  - major aspect of distributed system design
- Main architectures
  - **Client-Server**
    - two roles: clients as service users, and servers as service providers
    - conceptual centralization: servers, providing services
    - typically N clients and M servers, with N >> M, multiple clients for the same server
  - **Peer-to-Peer**
    - no roles, only peers
    - no centralization
    - high openness & dynamism
  - **Mobile code / agents**
    - programs or processes / agents travelling through hosts of the network

DESIGN REQUIREMENTS FOR DISTRIBUTED SYSTEMS

- Requirements
  - **Functional**
    - the functionality we desire for the systems, as a solution of some problem
  - **Non-functional** (NF requirements)
    - define system properties and constraints
- Quality of Service as NF requirements
  - *Reliability*
    - measuring the capability of a system to perform as specified without interruption
  - *Availability*
    - measuring the capability of a system to be up and running
  - *Performance*
    - responsiveness, throughput, balancing computational loads
  - *Dependability*
    - qualitative term for the ability of the system to perform properly
    - encapsulates reliability, availability, safety, maintainability, performability, testability

DISTRIBUTED ALGORITHMS

- Algorithms designed for loosely-connected distributed systems that communicate by sending and receiving messages over a communication network
- Some recurrent problems (classes)
  - *Distributed mutual exclusion*
    - Ricart-Agrawal algorithm
  - *Distributed termination*
    - Dijkstra-Scholten algorithm
  - *Distributed snapshots*
    - Chandy-Lamport algorithm
  - *Distributed consensus*
    - Byzantine general algorithm