Mobile and Wireless Middleware

Distributed System Concepts

- Device
  - Fixed
  - Mobile
- Connection
  - Permanent
  - Intermittent
- Execution context
  - Static
  - Dynamic

Type of Device

- Fixed
  - Relatively powerful machine
  - Large amounts of memory and fast processor
  - Non-battery powered, always-on electricity
  - Not necessarily portable
- Mobile
  - Slow CPU speed
  - Little memory
  - Limited battery power
  - Small screen size and other size limitations

Type of Network Connection

- Permanent
  - Assume stable connectivity
  - Disconnections are either:
    - Explicitly performed for administrative reasons
    - Caused by unpredictable failures
  - Failures treated as exceptions to normal behaviour
- Intermittent
  - Unpredictable disconnections are normal, not the exception
  - Wireless network performance varies depending on network technologies (GSM, 802.11, Bluetooth)
  - Bandwidth variation and connection loss is common
  - Typically the case for mobile distributed systems

Type of Execution Context

Static	Dynamic
High and stable bandwidth	Varying bandwidth and network performance
Mostly fixed location	Location non fixed
Low frequency of changes to host membership on the network	Rapid changes in hosts
Straightforward discovery of services	Complex service lookup

Types of Distributed Systems

- Traditional
  - Original form, existing for more than 20 years
  - Non-functional requirements: scalability, openness, heterogeneity, fault-tolerance, resource-sharing
- Nomadic
  - Compromise between fixed and mobile systems
  - Composed of a set of mobile devices and fixed core infrastructure
  - Mobile devices move between locations, usually connected to the fixed infrastructure via a wireless network
  - Computing power and services provided by the core network to mobile clients
  - Traditional non-functional requirements hold as core is fixed
  - Additional complexity if mobile devices allowed to provide services

Types of Distributed Systems II

- Ad Hoc
  - No fixed infrastructure
  - Groups/networks form and evolve independently
  - Applications: casual information sharing, military battlefield, emergency networks in disaster areas
  - Non-functional requirements
    - Scalability: large routing tables and messages in big networks with ad hoc routing
    - Heterogeneity: integration of different connectivity technologies (Eg: Bluetooth and 802.11)
    - Fault tolerance: disconnection the norm
    - Security: more difficult than in fixed networks, message encryption to avoid spoofing
- Future
  - Hybrid networks with fixed components and ad hoc areas
  - Heterogeneous with different connectivity technologies

Middleware Systems

- Building applications on top of the network layer is difficult
- Goal of middleware is to enable communication between distributed components
- It provides a higher level of abstraction built using OS primitives
- It also provides resource sharing and fault tolerance facilities
- Fixed middleware examples: CORBA, Java RMI
    - Not generally suitable in a mobile setting

Middleware System Concepts

- Computational load
  - Heavyweight
  - Lightweight
- Communication paradigm
  - Synchronous
  - Asynchronous
- Context representation
  - Transparency
  - Awareness

Type of Computational Load

- Heavyweight
  - High reliability of communications
    - Expensive to guarantee “exactly once”
  - High level fault tolerance
    - Expensive replication and synchronisation
- Lightweight
  - Lower quality of service
  - Minimum set of resources handled

Type of Communication Paradigm

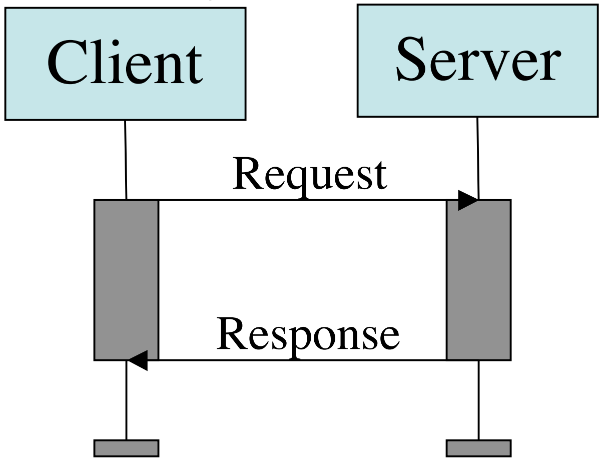
- Synchronous
  - Client blocked during execution
  - May not be appropriate depending on time taken by response (UI)
- One-way
  - Control returned to client as soon as request accepted
  - Appropriate where client semantics don’t depend on result

Type of Communication Paradigm II

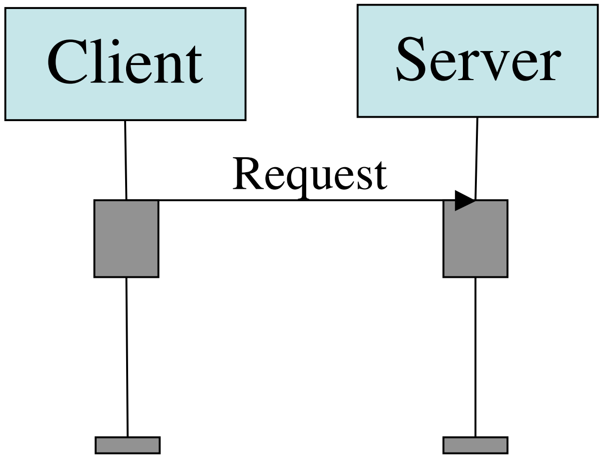
- Deferred synchronous
  - Control returned as soon as request accepted
  - Client can obtain result later by “polling”
- Asynchronous
  - Control returned to client with server explicitly calling the client with result
  - Hollywood principle: “don’t call us, we’ll call you”
  - No need for polling

Type of Communication Paradigm III

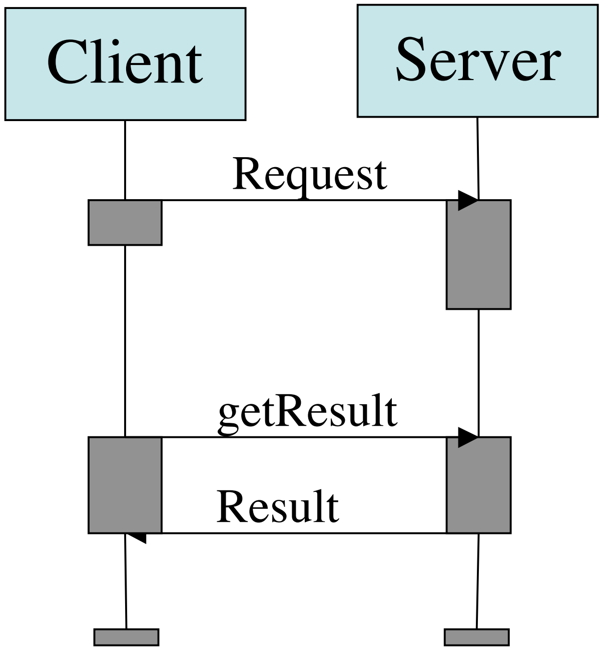
Synchronous



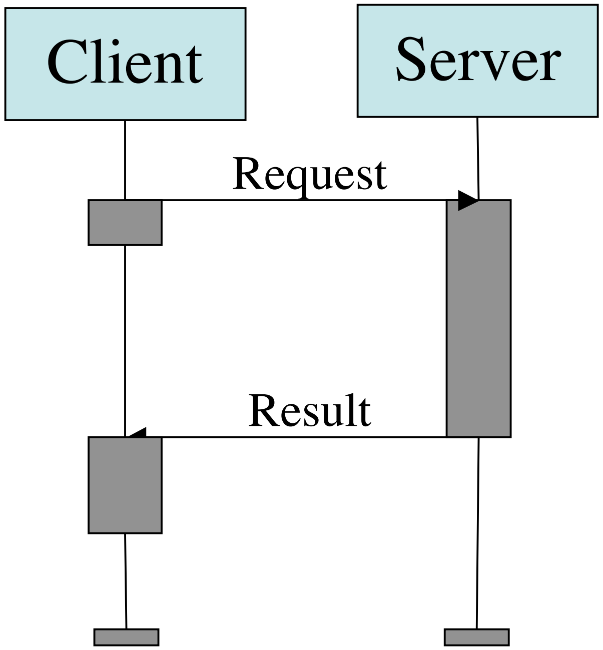
One-way



Deferred synchronous



Asynchronous



Type of Context Representation

- Transparency
  - Context info collected/maintained by middleware
    - Location, bandwidth, latency, available services
  - Context info hidden from applications
  - Example: Middleware detects congestion and redirects request to replica on non-congested portion of network
- Awareness
  - Application aware of contextual information
  - Applications responsible for strategic QoS decisions
  - Ex: application chooses which replica to contact
  - Complex, transparency more common

Middleware Classification

	Fixed Distributed Systems	**Ad hoc and Nomadic Systems**
Device	Fixed	Mobile
Computational load	Heavy	Light
Connection	Permanent	Intermittent
Communication	Mostly synchronous	Asynchronous
Context	Static	Dynamic
Transparency	Transparency	Awareness

Fixed Middleware Approaches

- Categories aren’t rigid, trend of merging approaches
- Object and component oriented
  - Evolved from Remote Procedure Call (RPC)
  - Supports communication between distributed objects
  - Basic interaction is synchronous
  - Examples: CORBA, COM, RMI
  - Limited application to mobile networks
    - Heavy computational load
    - Synchronous communications
    - Transparency

Fixed Middleware Approaches II

- Message oriented
  - Communication via message-passing, client sends message with request, server may respond with a reply message
  - Supports async communication, client can continue after message is sent and collect server response later
  - Resource-rich devices for persistent message queues
  - Examples: Java Message Queue and IBM MQSeries
  - Possible use in mobile settings, with some adaptation
- Transaction oriented
  - Used in database-based applications
  - Support transactions involving distributed components
  - Both sync and async with high reliability
  - Not very suitable for mobile settings due to computational load and transparency

Mobile Middleware Approaches

- Avoid middleware
  - Rely on app to deal with non-functional requirements
  - Often using a context-aware approach
  - Example: J2ME, .Net Compact Framework
  - Non-solution, app has to provide all non-functional requirements
- Traditional middleware applied in mobile environment
  - Object-oriented middleware adapted to mobile setting
    - Mainly for nomadic settings
    - CORBA adapted to mobile devices (ALICE, DOLMEN)
    - Permanent connectivity assumption, with hand-off support and minimal support for disconnection due to asynchronous communication

Mobile Middleware Approaches II

– Semi-asynchronous paradigm
  - RPC with enhancements to cope with intermittent connections
  - Examples: Rover and Mobile DCE

– Message-oriented systems
  - Java Messaging Server (JMS) adapted to mobile, supports publish/subscribe and point to point asynchronous communications

– Focus on backbone fixed network providing services to mobile devices
  - Less suitable for unstructured networks or mobile-provided services

Mobile Middleware Approaches III

- QoS-oriented middleware
  - Mobiware: mobile devices are terminal nodes with main services provided by fixed infrastructure
    - Mobile devices probe and adapt to changing resources over the wireless link
    - Focused on delivery of multimedia apps to mobile devices with adaptation to different QoS and seamless mobility
    - Assumes roaming but permanent connections, varying bandwidth
    - Built on CORBA and Java distributed objects
  - L2imbo: tuple-space based service aware system

Context-awareness and Middleware

- Context-awareness characteristics
  - Refers to aspects of physical or virtual world
  - At a single point in time or history
  - Can contain private data
  - May have heterogeneous sources
  - May have large numbers of clients
  - Maybe inaccurate or with difficult to verify accuracy
- Middleware needs to support development, maintenance, deployment and execution of context-aware applications

Context-awareness Challenges

- Privacy
  - Provide mechanisms to protect privacy of sources and clients, and a means to configure the level of control over access
- Scalability
  - Provide mechanisms to handle potentially large numbers of sources and clients
- Extensibility
  - Provide mechanisms to accommodate new and unanticipated sources and context information
- Synchrony
  - Provide mechanisms for both synchronous (real-time) and asynchronous communications
- Quality of information
  - Provide mechanisms to measure the quality of information and allow for inaccuracies and uncertainty

Context-awareness based MW

- Problem: limited resources and need of providing context information to app
- Solution: reflective middleware
  - Able to modify itself by
    - Inspection: internal behavior is exposed, possible to monitor middleware implementation (and context info)
    - Adaptation: Internal behaviour can be changed by modifying existing features or adding new ones
  - MW core has the minimal set of functionality, app in charge of adapting the middleware
  - Main idea is to change the behaviour of the mw and app based on the evolving context
  - Good fit for scarce resources (small mw core) and dynamicity of mobile environment (reflection)

Context-awareness based MW II

- Location-aware middleware
  - Most studied aspect of context
  - Apps: tourist guidance, location-based advertisement
  - Needed different versions due to heterogeneity of coordinate information (GPS, GSM)
  - Middleware systems that provide a common interface to different positioning systems
    - Oracle iASWE, Nexus, Alternis, SignalSoft, CellPoint

Context-awareness based MW III

- Data-sharing oriented middleware
  - Intends to allow sharing of data between mobile nodes with intermittent network connections
  - Replicas used to maximise availability
    - Need to ensure move towards consistency, conflict resolution
    - Lack of single synchronisation standard, each protocol implemented for a limited subset of devices and data
    - Examples: Coda, Odyssey, Bayou and Xmiddle

Context-awareness based MW IV

- Tuple-space based middleware
  - An asynchronous and decoupled approach to distributed communication
  - A tuple-space is a repository of tuples, typed vectors, that can be accessed concurrently by different processes using read or write operations
  - A tuple-space looks like a globally shared data space, independent of device or platform differences
  - Examples: Lime, TSpaces, L2imbo

Service discovery in mobile MW

- Service discovery is simple in fixed and nomadic systems where a fixed infrastructure contains all the information and services
- Discovery is more complex/expensive in ad-hoc systems
  - Standard service discovery frameworks
    - UPnP adopted by Microsoft, runs on SOAP/HTTP/TCP/IP
    - Salutation is a platform and OS independent framework
    - Jini is Java based
  - Purpose of frameworks is to allow independent devices and services to form a single dynamic distributed system

Service discovery in mobile MW II

- Jini and JMatos
  - Goal is to turn the (mobile) network into a framework where resources (HW & SW) and services can be found, added and deleted
  - Main concept is service, with members of a Jini system federating in order to share access to services
  - A lookup service is the primary "marketplace" for services
  - Assumes (fixed) infrastructure that allows services and users to join the Jini system and connects Jini-enabled devices
  - Large footprint limits its use on smaller devices, JMatos is a lightweight alternative that doesn't use RMI

Service discovery in mobile MW III

- Salutation
  - Focus on interoperability of services across platforms and operating systems
  - Service discovery and data transmission is managed by Salutation Managers that interact via RPC