Mobile Middleware Course

Principles and Patterns

Sasu Tarkoma

Contents

Principles

- A **principle** signifies strong belief in a certain state or property of a subject.
- Principles support the formation of a rule or a norm by observing the subject.
- Principles have a form of minimality character, because they cannot be further divided.
- A rule or a norm can be reduced to a principle, but principles are not reducible.

Patterns

- Design patterns are software engineering designs that have been observed to work well
- Patterns are found in different contexts, they provide a solution for a well-defined
- problem area, and digress the various dimensions of the problem
- Patterns are classified into different groups based on their level of abstraction.
  - Architectural patterns summarize good architectural designs
  - Design patterns capture the essence of medium level language independent, design strategies in object-oriented design
  - Idioms represent programming-language-level aspects of good solutions

Architecture and platforms

- An architecture is a guided by principles and grounded on architectural patterns. An architecture consists of components, and rules and constraints that govern the relationships of the components
- A platform is a concrete realization of a middleware architecture
- A protocol stack is a concrete realization of a set of protocols and an architectural framework how to use them in combination, typically using a stack pattern, although other kinds of organizations are also possible

Patterns

- Patterns are typically defined in terms of their motivation, underlying problem, structure, consequences, implementations, and known users
- Pattern that are applicable in a particular domain can be collected together. This kind of pattern collection is often called **pattern language**

Patterns continued

- The following table presents important information used to define patterns:
  - Pattern name: An informative name that uniquely identifies the pattern
  - Intent: Goals of the pattern and the reason for utilizing it
  - Motivation (Forces): A short problem statement that is presented using a scenario
  - Applicability: Describes the environments and contexts in which the pattern can be applied
  - Structure: Describes the structure of the pattern using different graphical representations
  - Collaboration: Describes how the various elements, namely classes and objects, interact in the pattern
  - Consequences: Describes the results that can be expected from using the pattern
  - Implementation: Describes an implementation of the pattern
  - Known Uses and Related Patterns: Examples of how the pattern has been applied in real systems

Principles

- Internet
- Web
- Service-oriented Architecture
- Security
- Mobile computing

Internet Principles

- **End-to-End Principle**
  - In its original expression placed the maintenance of state and overall intelligence at the edges, and assumed the Internet that connected the edges retained no state and concentrated on efficiency and simplicity.
  - Today's real-world needs for firewalls, NATs,Web content caches have essentially modified this principle.
- **Robustness Principle**
  - Be conservative in what you do, be liberal in what you accept from others.
  - This principle has been attributed to Jon Postel, editor of the RFC 793 (Transmission Control Protocol).
  - The principle suggests that Internet software developers carefully write software that adheres closely to extant RFCs but accept and parse input from clients that might not be consistent with those RFCs.

Web Principles

- The principles of the Web follow those of the underlying TCP/IP stack.
- Principles such as **simplicity** and **modularity** are at the very base of software engineering; **decentralization** and **robustness** are the foundational characteristics of the Internet and Web.
- Web principles are about supporting flexible publishing of resources on the Internet and then linking these resources together. In the context of data publishing, data **representation** and **transformations** are crucial.
- The principle of applying the **least powerful language** to do a particular job
- HTTP, URL, HTML, XML

REST Principles

- **Representational State Transfer (REST)**
- The principles behind REST are the following:
  - Application state and functionality are divided into resources
  - Every resource is uniquely addressable using a universal syntax for use in hypermedia links
  - All resources share a uniform interface for the transfer of state between client and resource, consisting of a constrained set of well-defined operations, a constrained set of content types, optionally supporting code on demand
  - The defining features of REST are: **client-server, stateless, cacheable, and layered**

SOA Principles

- **Service Oriented Architecture (SOA)** is a software architecture where functionality is structured around business processes and realized as interoperable services
- Reuse, granularity, modularity, composability, componentization, and interoperability
- Compliance to standards (both common and industry-specific)
- Services identification and categorization, provisioning and delivery, and monitoring and tracking

Security Principles

- The commonly agreed security aspects are the following:
  - Privacy
  - Integrity
  - Authentication
  - Authorization
  - Accountability
  - Availability

W3C Guiding Privacy Principles

- The W3C Platform for Privacy Protections (P3P) working group has established the following privacy guiding principles:
  - **Notice and Communication**. Service providers should provide timely and effective notices of their information policies and practices. User agents should provide effective tools for users to access these notices and make decisions based on them
  - **Choice & Control**. Users should be given the ability to make meaningful choices about the gathering, utilization, and disclosure of personal information
  - **Fairness&Integrity**. Users should retain control over their personal information
  - **Confidentiality**. Users' personal information should always be protected with reasonable security measures taking into account the sensitivity of the information and required privacy level

Mobile Principles: Device View (NoTA)

- **System level loose coupling**. This means that loose coupling of components is built into the system
- **Interconnect centric**. The interconnect is responsible for connecting different system components and services together via message passing
- **Service based,** which means that functionality is provided through services that have interface definitions
- **Message and data driven.** Message passing is the preferred mechanism for realizing mobile applications. In addition, the communications is typically data driven meaning that a request can be forwarded based on the current system parameters and information contained in the request. decoupling between components
- **Implementation-wise heterogeneous**

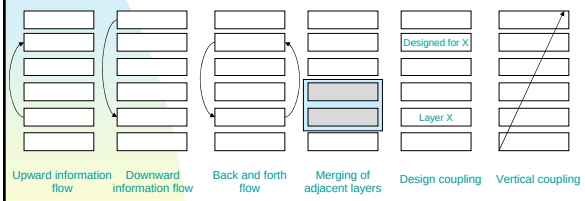
Mobile Principles: SIP

- Proxies are for routing
- Relegation of call state to endpoints
- Endpoint fate sharing,
  - Application fails when the endpoints fail
- The usage of dialog models and not call models
- Component based design
- Logical roles
- Internet-based design
- Generality over efficiency
- Separation of signaling and media.

Information flows in a protocol architecture

- **Upward** information flow, in which information is propagated from lower layers towards upper layers
- **Downward** information flow, in which information is propagated from higher layers towards lower layers. An interface is used to set a lower layer parameter
- **Back-and-forth** information flow, in which information is propagated in both directions
- **Merging of adjacent layers**, allows the combination of several adjacent layers into a super layer
- **Design coupling** without adding new interfaces. In this strategy two or more layers are coupled during design time without specifying a new interface between them for information sharing at runtime
- **Vertical calibration** across layers. This involves adjusting parameters across layers. The motivation is that joint tuning of parameters in the protocol stack can help to achieve better performance.

Interactions



Architectural patterns I

- **Layers**. A multilayered software architecture is using different layers for allocating the responsibilities of an application
- **Client-Server**. The client-server pattern is the most frequent pattern in distributed computing, in which clients utilize resources and services provided by servers
- **Peer-to-peer**. The peer-to-peer pattern is emerging communications model, in which each peer in the network has both client and server roles
- **Pipeline** (or pipes and filters). A pipeline consists of a chain of processing elements arranged so that the output of each element is the input of the next

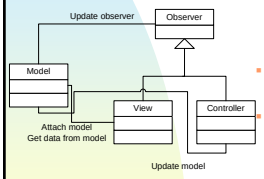
Architectural Patterns II

- **Multitier**. A multitier architecture is a client-server architecture in which an application is executed by more than one distinct software agent
- **Blackboard system**. In this pattern, a common knowledge base, the "blackboard", is iteratively updated by a diverse group of specialist knowledge sources, starting with a problem specification and ending with a solution
- **Publish/Subscribe**: Event-channel and Notifier

Architectural patterns for Mobile Computing

- **Model-View-Control (MVC)** is both an architectural pattern and a design pattern, depending where it is used
- **Broker**, which introduces a broker component to achieve decoupling of clients and servers
- **Microkernel**. This pattern provides the minimal functional core of a system, the microkernel, which is separated from extended functionality. The external functionality can be plugged in the microkernel through specific interfaces
- **Active Object**. The Active Object pattern provides a support for asynchronous processing by encapsulating the service request and service completion response

Model-View-Control



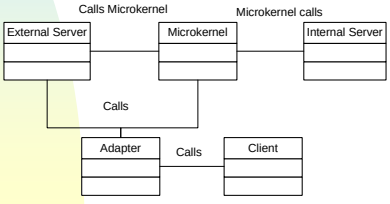
- The pattern divides the application into three parts
  - the controllers handling user input
  - the model providing the core functionality
  - the views displaying the information to the user
- The pattern ensures that the user interface is formed by the view and the controller is consistent with the model
- The pattern also specifies the *change-propagation* mechanism
  - Views and Controllers register with the Model to receive notifications about changes in the structure
  - When the state of the Model changes, the registered Views and Controllers are notified
- Used in Symbian OS and many other systems

Broker

- This pattern introduces a broker component to achieve decoupling of clients and servers
- Servers register with the broker, and make their services available to clients through method interfaces provided by the broker
- Clients access the functionality of servers by sending requests via the broker
- The tasks of the broker include
  - locating the appropriate server
  - forwarding the request to the server
  - transmitting results and exceptions back to the client

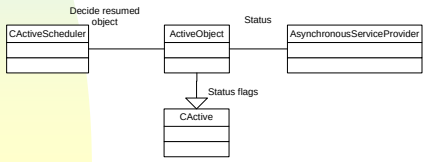
MicroKernel

- This pattern may be applied in the context of complex software systems serving as a platform for other software applications
- The desired characteristics for such systems include extensibility, adaptability, and interoperability
- A small core that is extensible with pluggable components



Active Object

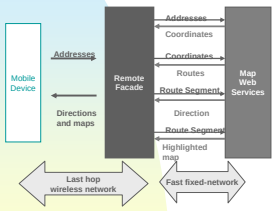
- The Active Object pattern provides support for asynchronous processing
- The pattern works by encapsulating and handling asynchronous service requests and service completion responses
- The pattern allows the client to be notified about the task's completion and perform other tasks asynchronously with the server
- Active Object runs in a same thread as the application. Helps to eliminate overhead in context-switching between threads
- The liability of Active Object is the fact that it is non-preemptive



Patterns for Mobile Computing

- Three categories
  - distribution
  - resource management and synchronization
  - communications
- Distribution patterns pertain to how resources are distributed and accessed in the environment.
  - remote facade, data transfer object, remote proxy, and observer
- Resource management and synchronization
  - session token, caching, eager acquisition, lazy acquisition, synchronization, rendezvous, and state transfer
- Communications
  - connection factory and client-initiated connections

Distribution: Remote Facade

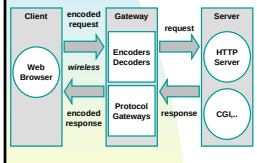


- The pattern provides a coarse-grained interface to one or several fine-grained objects
- The interface is provided through a remote gateway
  - accepts incoming requests conforming to the facade interface
  - subsequent fine-grained interactions between the remote facade (gateway) and third party interfaces
- An application using the pattern does not have to know which particular servers or remote functions are used to implement a requested operation

Distribution: Data Transfer Object (DTO)

- The *Data Transfer Object (DTO)* provides a serializable container for transferring multiple data elements between distributed processes
- The aim of the pattern is to reduce the number of remote method calls
- A DTO can be used to hold all the data that need to be transferred
- A DTO is usually a simple serializable object containing a set of fields along with corresponding getter and setter methods

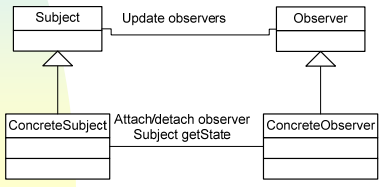
Distribution: Remote Proxy



- In this pattern, a proxy (or a gateway) is between a terminal and the network
- All or selected messages or packets from the client go through the proxy, which can inspect them and perform actions
- The proxy performs computationally demanding tasks on behalf of the client terminal
- The proxy serves as an adapter allowing other computers to communicate with the terminal without the need to implement terminal-specific protocols

Distribution: Observer

- The observer pattern explains how to define a one-to-many dependency between objects
- All the dependent objects are notified when the state of the object being observed change

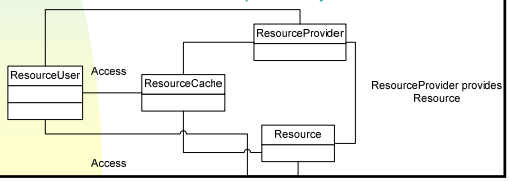


Resource Management: Session Token

- This pattern alleviates state management requirements of servers.
- A token is issued by a server to a client that contains data pertaining to the active session the client has with the server.
- The token contains a session identifier and possibly some security related data as well.
- When the client presents the token again to the server, the server can then associate the client with the proper session

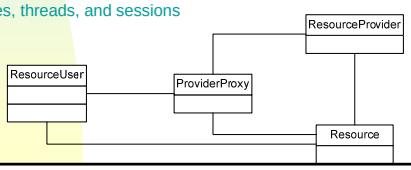
Resource Management: Caching

- The caching pattern suggests temporarily storing these resources in a local storage after their use, rather than immediately discarding them
- This cache of elements is first checked when a resource is requested
- If the element is found it is immediately delivered to the requesting application
- If an element is not found in the cache, the request is performed and an entry is created in the cache for the requested object



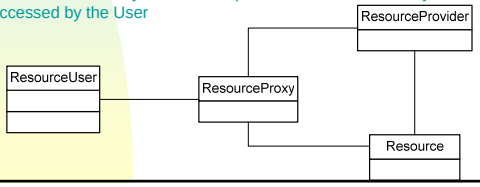
Resource Management: Eager Acquisition

- If the resources that are needed by an application are known beforehand, a system can utilize this information and prefetch these resources
- As a result, the resources are already locally available when they are needed and a remote request is not needed
- The eager acquisition pattern follows this design and tries to acquire resources that may be needed later
- Examples of resources include memory, network connections, file handles, threads, and sessions



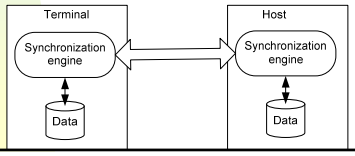
Resource Management: Lazy Acquisition

- In order to optimize the use of system resources the pattern suggests to defer the resource acquisition until the latest possible time
- The solution consists in acquiring the resources only when it becomes unavoidable
- The Resource Proxy is responsible for intercepting all the resource requests issued by the User
- The Resource Proxy does not acquire resources unless they are explicitly accessed by the User

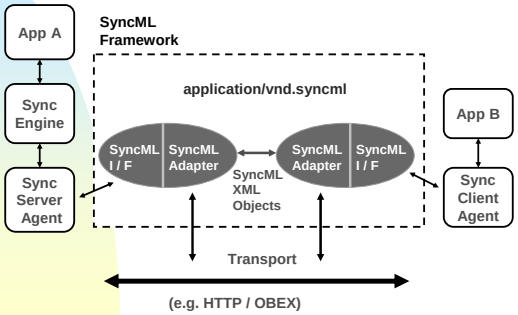


Synchronization

- In order to be able to manage multiple data items across multiple devices, this pattern advises to implement a device specific synchronization (sync) engine.
- The engine is for tracking modifications to data items, exchanging this information, and then updating the data accordingly when the connection is available.
- The engine is also responsible for detecting and resolving possible conflicts that may occur during the synchronization process.

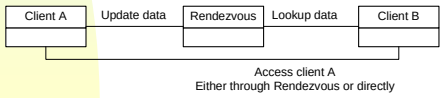


SyncML



Synchronization: Rendezvous

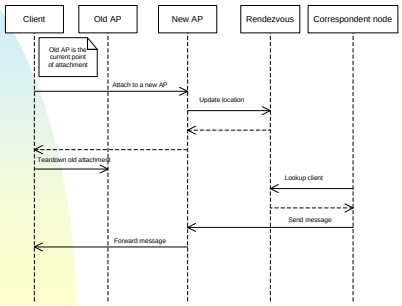
- Rendezvous can be seen as a central pattern in assisting a network to cope with mobile devices
- Rendezvous is a process that allows two or more entities to coordinate their activities
- In a distributed system, rendezvous is typically implemented using a *rendezvous point*
  - a logically centralized entity, an indirection point, on the network
  - accepts messages and packets and maintains state so that it can answer where a particular mobile device is located



Resource Management and Synchronization: State Transfer (handoff)

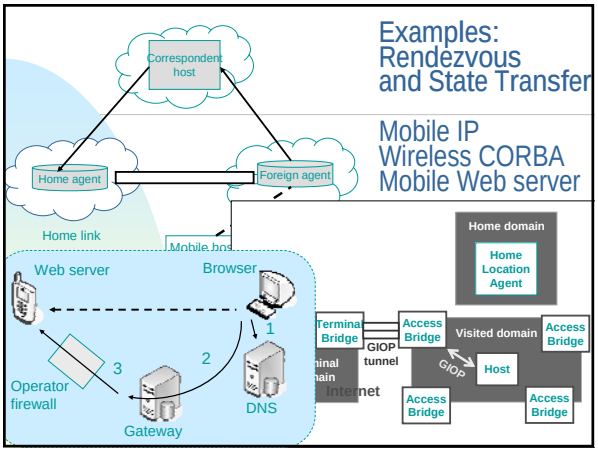
- Different kinds of handoffs or handovers have been specified and implemented in the mobile computing context
- Handoffs involve state transfer between access points.
- Handoffs are central in enabling seamless connectivity in any wireless communications system.

State Transfer



Examples: Rendezvous and State Transfer

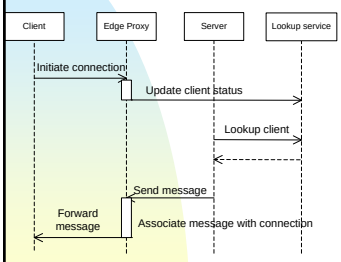
Mobile IP
Wireless CORBA
Mobile Web server



Communications: Connection Factory

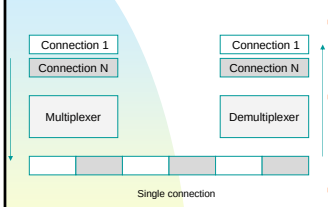
- This pattern suggests the decoupling of the application and the underlying data communications system by introducing a component that is used to create, access, and terminate connections
- The factory design pattern is utilized by the connection factory pattern in order to allow the management and reuse of connections in an efficient manner
- The connection factory pattern is used heavily in the Java architecture. The communications API of the Java ME features this pattern

Communications: Client-initiated connection



- In many cases it is impossible to reach a mobile client due to firewalls and NAT devices present on the communication path
- These problems in connectivity motivate the use of a client initiated connection to a publicly addressable server that can then push messages to the client using the connection

Communications: Multiplexed Connection



- It is not efficient to create many connections that may compete for system and network resources
- The Multiplexed Connection pattern utilizes a single logical connection and multiplexes several higher-level connections onto it
- This allows the choice of using arbitrary prioritization for messages multiplexed over the connection