Lecture 5

Discovery services I

The problem

- Users/applications need to discover and use services or resources not available on their computers/mobiles.
- A resource may be a hardware device, storage, a content server, software, etc. The corresponding service allows the resource to be discovered and used remotely.
- Users are interested in finding *appropriate services* – according to the user definition: (low) cost, (easy) location accessibility, trustworthy, secure access etc.

- *Issues of concern*:
  - system architecture (different), including protocols;
  - service description (e.g., XML);
  - service provisioning during their use (transactions).

Examples

- *Bluetooth* Service Discovery Protocol: pairing followed by sharing resources, file transfer, etc.

- *Wi-Fi Direct* peer discovery: if two peers host the "chat" service, they can discover each other and start chatting.

- Service discovery with *Docker 2* (platform for deploying applications/services in a distributed system: a key-value store (etcd) is used to register services represented by keys and TTL, in addition to the Docker host IP and application port.

- Discovery can be executed across platforms at the TCP/IP level.

Using a directory

- Directory services provide more information than the name of the service.

- The lookup process may start with a name and produce the service reference (net address) and attributes, or, alternatively, start with a set of attributes and find out services/resources.

- There are:
  - *logically centralized solutions*, where global directories store addressing information for existing resources (X.500, LDAP);
  - *de-centralized ones*, where interrogating local managers identifies resources;
  - *mixed* - UDDI (Universal Description, Discovery and Integration) registry is a logically centralized, physically distributed service with multiple root nodes that replicate data with each other on a regular basis.

Sun's Jini, now Apache River

- "A Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users." Jini Arch Specs

- ***The main goal of Jini is to provide for service sharing.***

- The Jini architecture uses a *lookup service* and two protocols, *discovery* and *join*, for allowing services to be discovered and join or leave, dynamically.
- The events of join and leave are triggering signals to interested parties. This way, they are informed that new services are available or old services cease to be active.

Jini concepts

- Jini creates a *federation of services* provided in the network.
- *No central authority controls the federation.*
- Each individual computing entity can act as a client or server depending on whether it is requesting a service or providing one.
- A Java object represents each service. The object's interfaces expose the service offered to the network. Additionally, other service attributes can be included.
- Accessing a particular service involves using the published interfaces to invoke a remote procedure on the appropriate device object.

Jini lookup

- The lookup service is the central bootstrapping mechanism for the system; it provides the major point of contact between the system and users of the system.
- The lookup service maps interfaces indicating the functionality provided by services to sets of objects that implement those services.
- In addition, *descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people.*
- Entries in the lookup service are leased, allowing the lookup service to reflect accurately the set of currently available services.

Jini discovery and join

- The discovery/join protocols allow a new device to add a new service to the Jini system - a reference to the service is stored by the lookup service:
  1. first, the service provider locates a lookup service by issuing a request (unicast, or multicast) on the local network for any lookup services to identify themselves;
  2. when the lookup service gets a request, it sends an object back to the server. This object, known as a *registrar*, acts as a *proxy to the lookup service*, and runs in the service's JVM (Java Virtual Machine). Any requests that the service provider needs to make of the lookup service are made through this proxy registrar.
  3. then, a service object for the service is loaded into the lookup service - this service object contains the Java interface for the service, including the methods that users and applications will invoke to execute the service along with any other descriptive attributes.

- Therefore, the lookup service = the common repository of services offered by a network; it can run on any capable device.
- Clients use the repository to gain access to a particular service – gets the service proxy, then interacts directly with the network service via the proxy.

Jini reliability

- To handle network failures, Jini leases a resource to a client for a fixed period of time.
  - When the period expires, the client must renew the lease to continue accessing the service.
  - The lease automatically expires for all authorized users when a service goes down.
- Leases are either exclusive or non-exclusive. Exclusive leases ensure that no one else may take a lease on the resource during the period of the lease; non-exclusive leases allow multiple users to share a resource.
- Jini also supports redundancy in the infrastructure – several lookup services are distributed throughout the network. Servers can register their proxy server objects with all the lookup services they discover in the network.

Jini scalability

- Groups of Jini services form a *community.*
- Jini communities can link together, or federate into larger groups.
- The Jini lookup service for a particular community can register itself in other communities, thereby acting as the interface for sharing its resources with other communities' clients.

Use case

- Traffic monitoring uses a lot of cameras, static or mobile. They provide the same service, video-streaming, maybe with different QoS.
- Mobile clients can lookup the Jini directory where all city camera services are offered and select the ones of interest.
- Clients contact the servers and avail of the service.

Conclusions

- This solution is object based.
- The application is made of services (objects). Services can run on different hosts.
- Jini relies heavily on the ability to move objects across the network, from one JVM to another. Essentially, a "snapshot" of the object's state is taken using serialization (this is data). Class definitions for service proxy objects must also be downloaded, usually from where the service came from, using http or ftp. The service provider specifies the protocol used and also the location of the class files using the java.rmi.server.codebase property. The object's serialized data contains this codebase, which is used by the client to access the class files.
- Simple and powerful service and lease mechanism is applied to ensure that information is up to date.
- No administration required.
- Scalability is provided by federation.
- It can be used with other Java technologies for creating an integrated middleware system

Links

- https://river.apache.org/
- http://www.javacoffeebreak.com/books/extracts/jini/Overview.html
- http://www.oracle.com/technetwork/articles/javase/jinievents-140780.html