Ad Hoc Networking Pattern Language

Michael Kircher,

Michael.Kircher@mchp.siemens.de,

Siemens AG,

Munich Germany

Prashant Jain

Prashant.Jain@mchp.siemens.de

Siemens AG,

Delhi, India

Abstract

Ad hoc networking is based on the principle of spontaneous addition, discovery, and usage of services in a network. Various technologies such as Jini™ [Sun01], UPnP [Micr01], and .NET are built around the concept of ad hoc networking. While each technology is either platform or language dependent, they share a common underlying architecture which can be expressed using a pattern language. The pattern language abstracts away from any specific language, platform, or technology. We present this pattern language here with primary focus on resource management patterns.

Introduction

Ad hoc networking is based on the principle of spontaneous addition, discovery, and usage of services in a network. The services can be of many types such as simple time services, Powerpoint presentations, or MP3 player services. Resource Management forms the heart of ad hoc networking. As resources are continuously added and removed from an ad hoc network, managing these resources in an efficient manner becomes very important.

Example

Assume you want to build an ad hoc networking solution for your company. The solution should enable you to transparently distribute and acquire applications in the form of distributed services across your network. Most ad hoc networks typically comprise of mobile devices running some form of software that accesses the distributed services.

Since mobile devices have limited resources, such as memory and processing power, the software of the mobile clients needs to have a small memory footprint [NoWe00]. They should only have to know how to get hold of a service they need.

As an example, assume your network is a mobile communication network and your clients are mobile phones. The services can be of many different types such as:

- a quoter service that delivers real time stock quotes.
- a weather service that delivers weather forecasts.
- a news service that delivers latest news customized according to user preferences.
- a sports service that delivers latest sports scores.
- an advertisement service that delivers advertisements according to user preferences and location.

In all these cases, the services themselves would reside on the mobile phone and would be responsible for communicating with back-end providers and obtaining the necessary information. However, since mobile phones have limited processing power and memory, it will not be possible to host several such services simultaneously. Therefore, in order to save resources in the mobile phone, services need to be loaded on demand and removed when not needed any more.

If a service needs to be loaded on demand, there needs to be a mechanism of finding that service dynamically. The service should be available in the network such that the mobile phone can easily find the service. Once the mobile phone finds a service, it should be able to download it and install it without requiring any user interaction.

Over time, if the user no longer needs or uses one or more services, there should be a mechanism to remove those services to conserve precious resources of the mobile phone. This again should not require any user involvement.

Problem

The limited resources and highly dynamic behavior of the clients require flexible resource management.

Devices such as mobile phones that form part of an ad hoc network are typically constrained by memory, processing power, and storage. Such devices can use different services when they are part of the network. A device that uses a service is called the client of the service. Each service when installed would consume some resources. If a device were to install all services up front, a lot of overhead would be incurred and a lot of resources would be consumed unnecessarily. On the other hand, some devices may require highly deterministic behavior and therefore mandate installation of services up front.

Over time a device may no longer require some of the acquired services. Unless the device explicitly terminates its relationship with the service

provider and releases the services along with the corresponding resources, the unused resources would continue to be needlessly consumed. This in turn can have a degrading effect on performance of both the device and the provider, as swapping or other memory management activities might occur. In addition, it can also affect service availability for other devices.

To address the above requirements requires the resolution of the following forces:

- *Optimality*: Unnecessary service acquisitions should be avoided, as the acquisitions themselves are potentially expensive. In addition, the system load caused by unused services must be minimized.
- *Simplicity*: The management of services used by a client should be simple by making it optional for the client to explicitly release the services that it no longer needs.
- *Availability*: Services not used by a client, or no longer available should be freed as soon as possible to make them available to new client.
- *Lifecycle*: The frequency of use of a service should influence the lifecycle of a service.
- *Control:* Service release is determined by parameters such as available memory and CPU load.
- *Actuality*: A device should not use an obsolete version of a service when a new version becomes available.
- *Transparency*: The solution should be transparent to a client. The solution should incur minimum execution overhead and softwaredevelopment complexity.

Solution

To address the above forces, the solution that is typically implemented by ad hoc networks can be described using a set of design patterns, that is a pattern language. The pattern language abstracts away from any specific programming language, platform, or domain. The following are the primary patterns of the ad hoc networking pattern language:

- ***Lookup*** [KiJa00]- describes the manner in which services are found in a distributed environment**.**
- **Lazy Acquisition** [Kirc01] - describes how services can be acquired on demand at the latest possible point in time in order to avoid unnecessary resource consumption caused by the service.
- ***Evictor*** [HeVi99][Jain01] - describes how resource consumption can be optimized by strategizing eviction of services.
- ***Leasing*** [JaKi00] - describes the availability of a resource in terms of time-based resource reservation with the resource provider. This allows

handling of partial failures and avoids accumulation of outdated and unwanted information.

In addition to these patterns, there are some additional patterns that provide value addition in ad hoc networking. These include,

- ***Pick & Verify*** [Wisc00] - describes how resources are acquired randomly from a pool and verified against availability afterwards.
- ***Locate & Track*** [Cohe00] - describes how to track located services even when their location changes.

Structure

The Ad Hoc Networking Pattern Language comprises of four main participants:

- *Service*: A service publishes its interface so that it can be discovered and used by clients in the ad hoc network.
- *Lookup Service*: A lookup service is used by services to register themselves and by clients to discover the services.
- *Client:* A client, such as a mobile device, is an entity which makes use of a service. The client finds a service using the Lookup Service.
- *Service Provider*: The service provider provides a service for clients by registering the service, or only part of it, with the lookup service.

Dynamics

To optimize resource usage, a client will typically not acquire or use a service until it really needs to do so. This just-in-time approach described by the Lazy Acquisition pattern helps conserve valuable resources for the client.

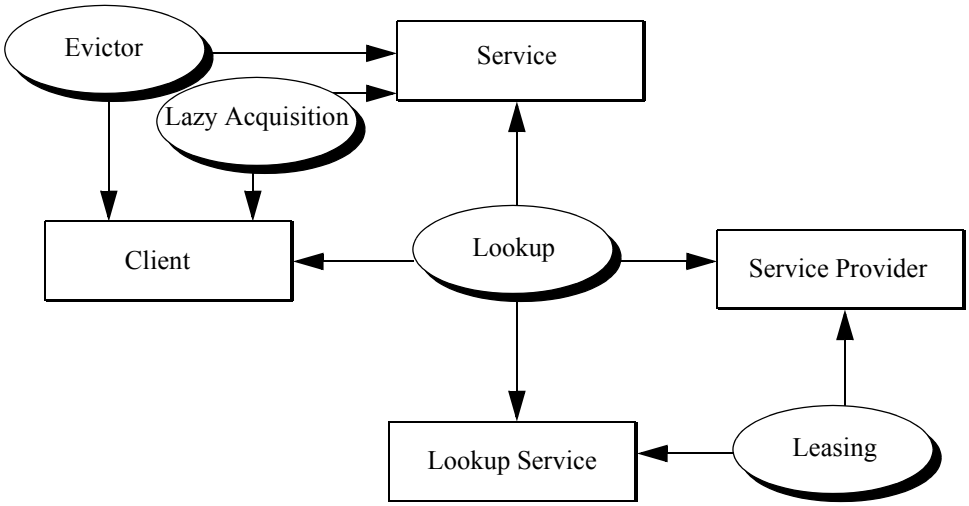
Once the client requires a service, it finds it through the Lookup Service. Note that prior to finding a service through the Lookup Service, the client must find the Lookup Service itself. All of this is well-documented by the Lookup pattern.

Once a client finds a service, it acquires a lease to gain access to the service. A lease is a grant of guaranteed access over a time period. Each lease is negotiated between the client and the service provider. The Leasing pattern describes this along with different variations that can be supported.

Once a lease has been acquired, the client directly uses the service. This typically requires loading of a service or part of a service into the client's address space. To optimize resource usage, this is typically done as and when required. The Lazy Acquisition pattern describes how this can be done.

Once the lease for the service expires or the service is no longer needed, the client typically stops using the service and the service along with its corresponding resources is evicted. This resource management is described by the Evictor pattern.

The following figure shows the Ad Hoc Networking participants (boxes) and how they relate with the patterns (ovals) of the pattern language.



The picture shows that the Evictor pattern is used by clients to evict services. The Lazy Acquisition pattern is used during the client's acquisition of a service. The Lookup pattern is used first by a service provider to register a service with the lookup service, and then by a client to look up a service from the lookup service. Finally, the Leasing pattern is used by the service provider for registrations at the lookup service.

In situations where the client cannot determine up front if a service is available for usage, the Pick&Verify pattern can be used to randomly pick a service from a pool of services. After acquiring the service the client then verifies if the service is actually available.

If the services are moving and hence the reference to them are changing over time, the Locate&Track pattern can help by keeping track of the changes in location.

Example resolved

To make the abstract idea more visible, we apply the above pattern language to a stock quoter application for mobile phones.

The quoter application consists of:

- a quoter service running at a back-end server machine, and
- a front-end client software to be downloaded on the mobile phone to access the back-end quoter service and query for stock quotes.

When the user demands for the quoter service the mobile phone will use a directory, implemented as a Lookup Service, to download the software and a reference to a valid quoter server. The client then has to ask the quoter server for a lease to be allowed to use its service.

Assume the quoter server can service only a limited number of clients on its ports and due to an overload is unable to tell the client up front which port is available. In such a case the Pick&Verify pattern helps the client to pick a port for a connection, verify if it is actually working, and then either pick another one if the port is not working, or use the one it just picked.

Since the quoter service would only be active while the stock exchange is open, the Locate&Track pattern can help locate a new quoter service corresponding to another stock exchange and continue to track it.

As the user continues to use his mobile phone, he is interested in the weather forecasts. Unfortunately, his phone has very limited memory capacity so that only one service can be loaded at a time.

The software will detect the limited resources and use an Evictor to remove the previously used quoter service so that the weather forecast service can be loaded into the device.

Known Uses

Sun's *Jini™* - A few years back Sun Microsystems introduced a new technology called Jini™. Jini™ offers a platform-independent plug & play technology. It supports ad hoc networking by allowing a service to be added to a network without requiring any pre-planning, installation, or human intervention. Once a service has been registered in a network, Jini™ allows other services and users to discover this new service. The ability to do all this in a transparent manner without manual intervention supports the basic principle of plug and play technology.

JinACE - is a framework implemented in C++ supporting ad hoc networking. It actually thought as a pendant to Jini, just implemented in C++. Instead of Java byte code, JinACE ships platform-dependent shared libraries across the network. [KiJa00a]

References

[Cohe00] B. Cohen, *Locate & Track Pattern*, The Jini Pattern Language - Worksop, OOPSLA conference, Minneapolis, USA, October 15-19, 2000, http://www.cs.wustl.edu/~mk1/AdHocNetworking/submissions

[HeVi99] M. Hennig and S. Vinoski: Advanced CORBA Programming with C++ - *Evictor Pattern*, Addison-Wesley, 1999

[Jain01] P. Jain, *Evictor Pattern*, Pattern Language of Programs conference, Allerton Park, Illinois, USA, 2001, http://www.cs.wustl.edu/~pjain/papers/Evictor.pdf

[JaKi00] P. Jain, and M. Kircher, *Leasing Pattern*, Pattern Language of Programs conference, Allerton Park, Illinois, USA, August 13-16, 2000, http://www.cs.wustl.edu/~mk1/Leasing.pdf

[KiJa00] M. Kircher, and P. Jain, *Lookup Pattern*, European Pattern Language of Programs conference, Kloster Irsee, Germany, July 5-9, 2000, http://www.cs.wustl.edu/~mk1/Lookup.pdf

[KiJa00a] M. Kircher, and P. Jain, *JinACE*, http://www.cs.wustl.edu/~mk1/AdHocNetworking, 2000

[Kirc01] M. Kircher, *Lazy Acquisition Pattern*, European Pattern Language of Programs conference, Kloster Irsee, Germany, July 4-8, 2001, http://www.cs.wustl.edu/~mk1/LazyAcquisition.pdf

[NoWe00] J. Noble, C. Weir, *Small Memory Software - Variable Allocation Pattern*, Addision-Wesley, 2000

[Sun01] Jini™, Sun Microsystems, http://www.sun.com/jini/

[Wisc00] M. Wischy, *Pick & Verify Pattern*, The Jini Pattern Language - Worksop, OOPSLA conference, Minneapolis, USA, October 15-19, 2000, http://www.cs.wustl.edu/~mk1/AdHocNetworking/submissions

[Micr01] *UPnP* Device Architecture 1.0, Microsoft Corp., http://www.upnp.org, 2001