Semester Project
Design Patterns in Peer-to-Peer Systems

Dominik Grolimund
Supervisor: Prof. Peter Müller

Department of Computer Science, ETH Zurich

1 Introduction

In recent times, peer-to-peer applications gained a lot of popularity due to file sharing applications such as Napster, Gnutella, Kazaa, eDonkey, and others. Basically, peer-to-peer refers to a decentralized architecture in which all nodes (i.e. computers) have identical capabilities and responsibilities and all communication is symmetric. Peer-to-peer systems have the unique advantage of being able to harness idle resources (computation cycles, bandwidth, and storage) of participating computers. They typically work in a large-scale, highly unreliable and insecure environment such as the Internet.

A software system exposed to such an environment needs to be designed very well. Yet, a brief investigation of peer-to-peer software projects yields that most of them are implemented in an ad-hoc manner. Improvements of peer-to-peer systems can be expected from two different aspects: better system design, and better software design. While the former is being extensively researched at the moment by building new structured overlay networks such as Pastry, Tapestry, Chord, and others, progress of the latter seems to be much slower. Part of this is probably because good design solutions for classical distributed systems (client/server) are around for a long time, and lots of them can be adapted to peer-to-peer systems. On the other hand, peer-to-peer systems exhibit a number of characteristics that are very different from centralized, asymmetric distributed systems, which has a clear manifestation in the code as well. This observation is also supported by peer-to-peer framework initiatives, such as Sun's JXTA, or Microsoft's peer-to-peer SDK, that are trying to build a higher abstraction around the core problems of peer-to-peer systems[1].

In this semester project, I would like to investigate common core software design problems of peer-to-peer systems and see how they are solved in a number of well-designed (academic) open source projects. The goal of this project is to

[1]These frameworks still have a long way to go. I argue that because peer-to-peer applications are very specific, design patterns have a much higher probability of reuse than frameworks at the moment.

end up with a number of design solutions to common problems of peer-to-peer systems, be it adapted and collected from pure client/server designs, or be it new (proto) patterns for peer-to-peer systems. If proto-patterns can be discovered, it would be very interesting to investigate them further (out of the scope of this semester project[2]), eventually arriving at one or two design patterns.

2 Tasks

The following is a tentative description of the tasks involved:

1. Read Literature
   - P2P: Read about P2P in general to define core elements.
   - Design Patterns: Read about design patterns (hillside.net, c2c.com) to find out about the methods for pattern mining, identifying patterns, etc.
   - Specialized Design Patterns: Find specialized design patterns (distributed systems, client-server, POSA II, concurrency, state machines, etc.).
   - P2P Patterns: Do an extensive research on which P2P patterns exist and are already documented / 'patternized'.
2. Find Open Source P2P Software Projects
   - Search for good / academic P2P software projects, organize code, design diagrams, etc.
   - Get familiar with these projects.
3. Define Core Problems
   - Define about five core problems that are intrinsic to P2P systems.
4. Think of Solution to these Problems
   - Define the core problems in detail.
   - Can they be solved with known distributed systems design patterns?
   - If not, how could the solutions look like?
   - If all specified core problems can be solved with existing patterns, see which other core problems might involve a new solution, and readjust core problems.
5. Mine for Solutions of Core Problems

[2]Pattern conferences such as PLoP and EuroPLoP with their shepherding and writer's workshop suggest that writing an actual design pattern or a pattern language involves a long maturing process, see for instance: http://www.hillside.net, http://hillside.net/plop/2004, or http://hillside.net/europlop

- Mine for solutions in each project.
- Document solutions.

6. Investigate New Problems
   - While mining for patterns, maintain a list of new problems / design solutions found in projects.
   - Investigate these "new problems" further, similar to 5.
7. Evaluation / Generalization / Design Patterns
   - Evaluate each solution.
   - Can solution be generalized?
   - For interesting problems, investigate it further, trying to distil a proto-pattern.
8. Write Paper
   - Write a paper of about 20-30 pages, stating problems, solution and possible generalizations / proto-patterns.