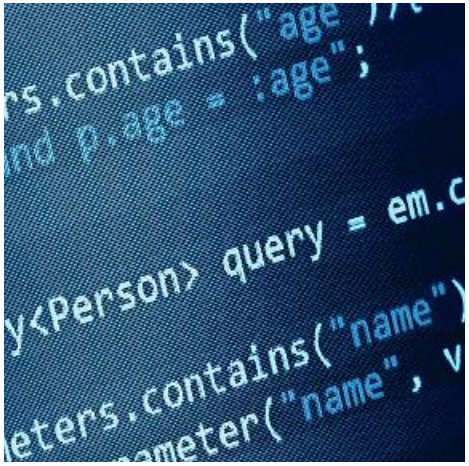
CS3500 Software Engineering

2017/2018



Dept. Computer Science
Dr. Klaas-Jan Stol

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Welcome to CS3500

Software Architecture

Part III

After studying this material and associated papers, you should be able to:

- Characterize architecturally significant requirements and distinguish them from non-architecturally significant requirements.
- Understand and describe common quality attributes.
- Be able to recognize common architectural styles and their consequences.

Contents

7. Architectural Significant Requirements

8. 6 Important Quality Attributes

9. 4 Common architecture styles

SECTION VII

Architectural Significant Requirements

1. Architectural significance (or not)

2. ASR characteristics

3. Cues for detecting ASRs

Distinguishing architectural from non-architectural design

- The purpose of architecture is to assure the satisfaction of a system's quality attributes.
- To that end, an architect makes architectural design decisions.
- To distinguish architectural from non-architectural design is asking:

Does this permit or prohibit the system to achieve its quality attributes?

How Not to distinguish Architectural from Non-Architectural (NA) design

- All architecture is design,
  but not all design is architectural.

- Architectural does not mean:
  - Max. 50 pages
  - Lack of detail
  - Big designs (can still be NA-design)
  - *... or any other arbitrary decision rule*

ASR characteristics

An ASR must have the following characteristics:

- A profound impact on the architecture
  Including this requirement likely to result in a different architecture than if not included.

- A high business or mission value
  If the architecture is going to satisfy this requirement it must be of high value to important stakeholders.

Architectural Significant Requirements

- Similar to architectural vs. non-architectural design, we distinguish: ASR vs. non-ASR
- How? No black-white solution!
- Luckily, we have heuristics! Hooray!

Seven cues that hint at architectural significance

1. Allocation of responsibility

2. Coordination Model

3. Data Model

4. Management of Resources

5. Mapping among architectural elements

6. Binding time decisions

7. Choice of technology

Next: Examples!

Seven categories

Category	Look for requirements addressing …
Allocation of responsibility	User roles, system modes, major processing steps, commercial packages
Coordination model	Properties of coordination incl. timeliness, concurrency, completeness, correctness, consistency. Names of external elements, protocols, sensors/ devices, middleware, network configurations
Data model	Processing steps, information flows, major domain entities, access rights, persistence
Management of resources	Time, concurrency, memory footprint, scheduling, multiple (parallel) users and activities, devices, energy usage, buffers and queues
Mapping among architectural elements	Allocation of processors, network coordination, communication between elements
Binding time decisions	Extension of functionality, run-time adaptation of functionality, regional/ language distinctions, portability, calibrations, configurations
Choice of technology	Named (specific) technology, changes/evolution of technology

Adapted from Bass et al. "Software Architecture in Practice", 3rd Ed.

EXAMPLE

Is it an ASR or a Non-ASR? (1/4)

Allocation of responsibility	**User roles, system modes, major processing steps, commercial packages**

A standard 500 dpi+ scanner will be used as this is the dpi used by law enforcements to scan.

Coordination model	**Properties of coordination incl. timeliness, concurrency, completeness, correctness, consistency. Names of external elements, protocols, sensors/devices, middleware, network configurations**

An SSD will be used and read/write times must be a minimum of 0.5ms per 4kb read and 0.2ms per 4kb write.

EXAMPLE

Is it an ASR or a Non-ASR? (2/4)

Data model	**Processing steps, information flows, major domain entities, access rights, persistence**

A live change can be made only during the process of filling the voting form. Once the vote gets sent, it will not possible to modify it anymore.

Management of resources	**Time, concurrency, memory footprint, scheduling, multiple (parallel) users and activities, devices, energy usage, buffers and queues**

The systems server will have a RAID 5 configuration.

EXAMPLE

Is it an ASR or a Non-ASR? (3/4)

Mapping among architectural elements	**Allocation of processors, network coordination, communication between elements**

A state-wide satellite internet connection will be available in all regions of Ireland for a stable and secure voting for all residents.

Binding time decisions	**Extension of functionality, run-time adaptation of functionality, regional/language distinctions, portability, calibrations, configurations**

The services will be offered in 5 languages as follows, English, Irish, French, German and Polish. [..] The user will be required to select the language they would like to use once they arrive at the voting machine.

EXAMPLE

Is it an ASR or a Non-ASR? (4/4)

Choice of technology	Named (specific) technology, changes/ evolution of technology

HTTPS or a similar protocol must be used to guarantee a secure connection that can protect the voter's data.

Reading Assignment

“Characterizing Architecturally Significant Requirements”

By: Lianping Chen, M. Ali Babar, and Bashar Nuseibeh



Originally published in:
IEEE Software, March/April 2013, pages 38-45.

Estimated reading time:
5 pages (text), 45 min.
You can skip the section “Methodology and Research Design” (½p.)

SECTION VIII

Six Important Quality Attributes

1. Availability
2. Modifiability
3. Performance
4. Security
5. Testability
6. Usability

Availability

Availability is concerned with:

- system failure and its consequences
- failure detection and prevention
- frequency of failure and repair time

System failure: a system stops delivering a specified service. Failures are observable by the user.

Fault: unobservable by the user. If not handled correctly, they may become failures.

Modifiability

Concerned with cost of change

What can change?

- Functions
- Runtime platform (OS, hardware → portability)
- Communication protocols
- Environment

When is the change made & who makes it?

- Implementation (source code) → developer
- Compile-time (compilation switches) → developer
- Configuration → administrator
- Execution (parameters) → user

Performance

Concerned with timing: how long does a system take to respond to an event (latency)

Events are generated by users, external systems, or within the system itself.

Events happen:

- Periodically (fixed pattern)
- Stochastically (probabilistic pattern)
- Sporadically (no pattern)

Security

Concerned with resisting unauthorized usage while servicing legitimate users.

Attack: attempt to breach security

- Unauthorized attempt to access system
- Attempt to deny services to legitimate users

Security provides:

- Nonrepudiation (undeniable transactions)
- Confidentiality
- Integrity (nobody can change your info.)
- Assurance (parties are who they say they are)
- Availability (system available for legitimate users)
- Auditing (tracking of activities – “audit trail”)

Testability

Ease with which software can be made to demonstrate its faults through testing.

Why important? 40% of effort is testing!

Must be possible to control components' internal state, inputs, and then observe outputs.

Usability

Concerned with how easy a user can accomplish a desired task and the support provided by the system.

Concerns:

- Learning system features
- Using a system efficiently
- Minimizing impact of errors
- Adapting to users' needs
- Increasing confidence and satisfaction

Usability is not just a nice interface!

Think undo / cancel options.

SECTION IX

Four Common Architectural Styles

1. Layers

2. Client-Server

3. Peer-to-Peer

4. Pipes & Filters

Styles and Patterns

- Architectural Styles and Architectural Patterns are similar things.
- Patterns are:
  "common solutions to recurring problems in a given context"
- A set of patterns forms a pattern language.
  [more on that some other time]

Why theory about styles/patterns?

(Almost) ALL systems use one or more styles.

Pattern	Frequency
Layers	18
Shared repository	13
Pipes & Filters	10
Client-Server	10
Broker	9
Model View Controller	7
Peer-to-Peer	3

Neil Harrison and Paris Avgeriou studied a sample of 47 systems. Layers was the most-used style.

Architectural styles

- The choice of architectural style has consequences for a system's quality attributes.
- But, degree of effect may depend on implementation details.
- The styles presented here have "some" (not exhaustive) associated benefits and drawbacks.
- Each system has an architecture, whether it's documented or not!

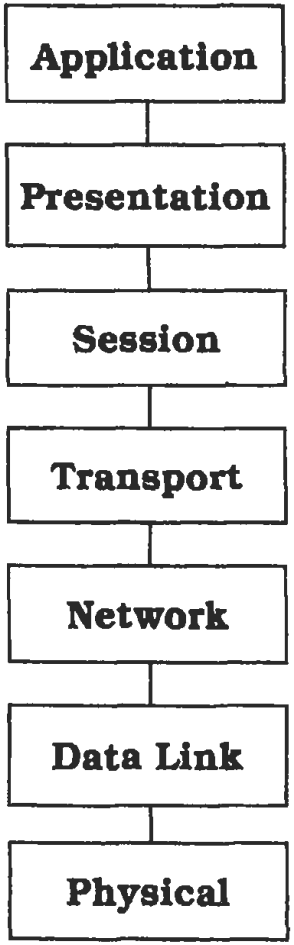
The Layers Style (a.k.a. N-tier)

The layered, or N-tier style decomposes a system into a set of layers, or tiers.

- Each layer represents a different level of abstraction to deal with issues or data.
- Each layer has a clearly defined interface that can be tested and which hides implementation details.
- Suitable to decompose large, complex systems when dealing with both high-level and low-level issues.

EXAMPLE

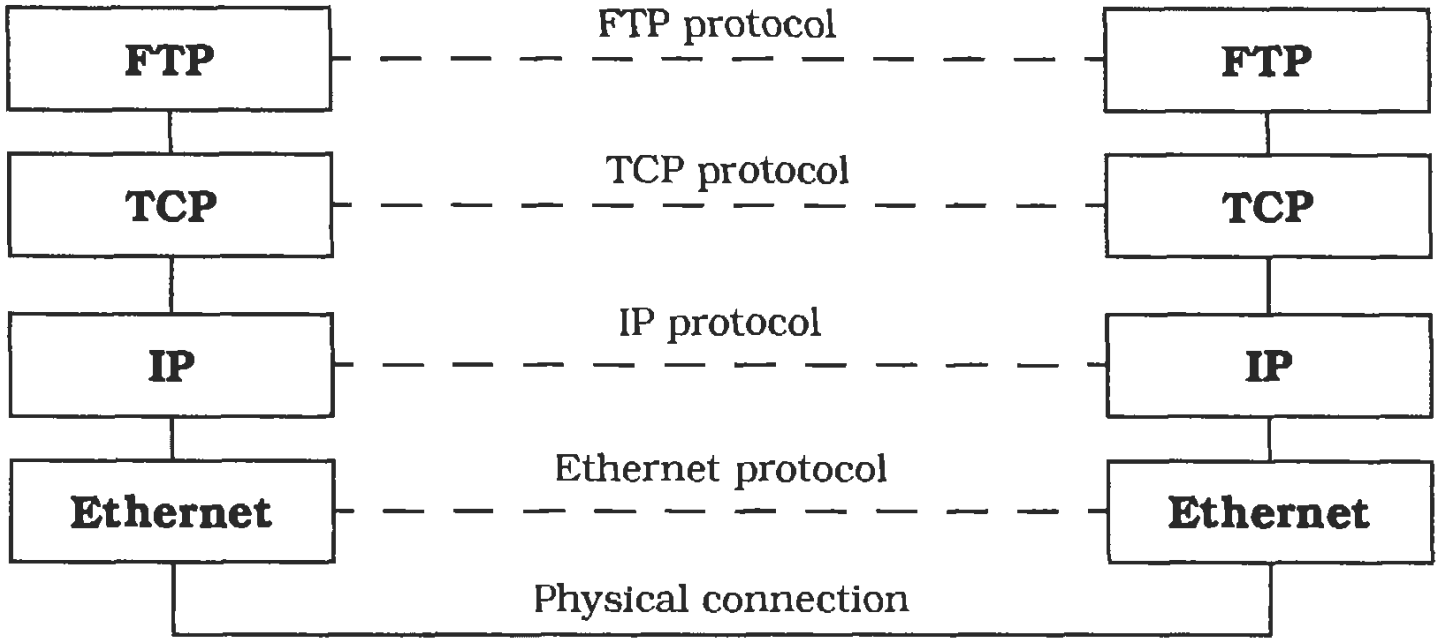
OSI 7-Layer Reference Model



Layer 7	Provides miscellaneous protocols for common activities
Layer 6	Structures information and attaches semantics
Layer 5	Provides dialog control and synchronization facilities
Layer 4	Breaks messages into packets and guarantees delivery
Layer 3	Selects a route from sender to receiver
Layer 2	Detects and corrects errors in bit sequences
Layer 1	Transmits bits: velocity, bit-code, connection, etc.

EXAMPLE

TCP/IP Stack

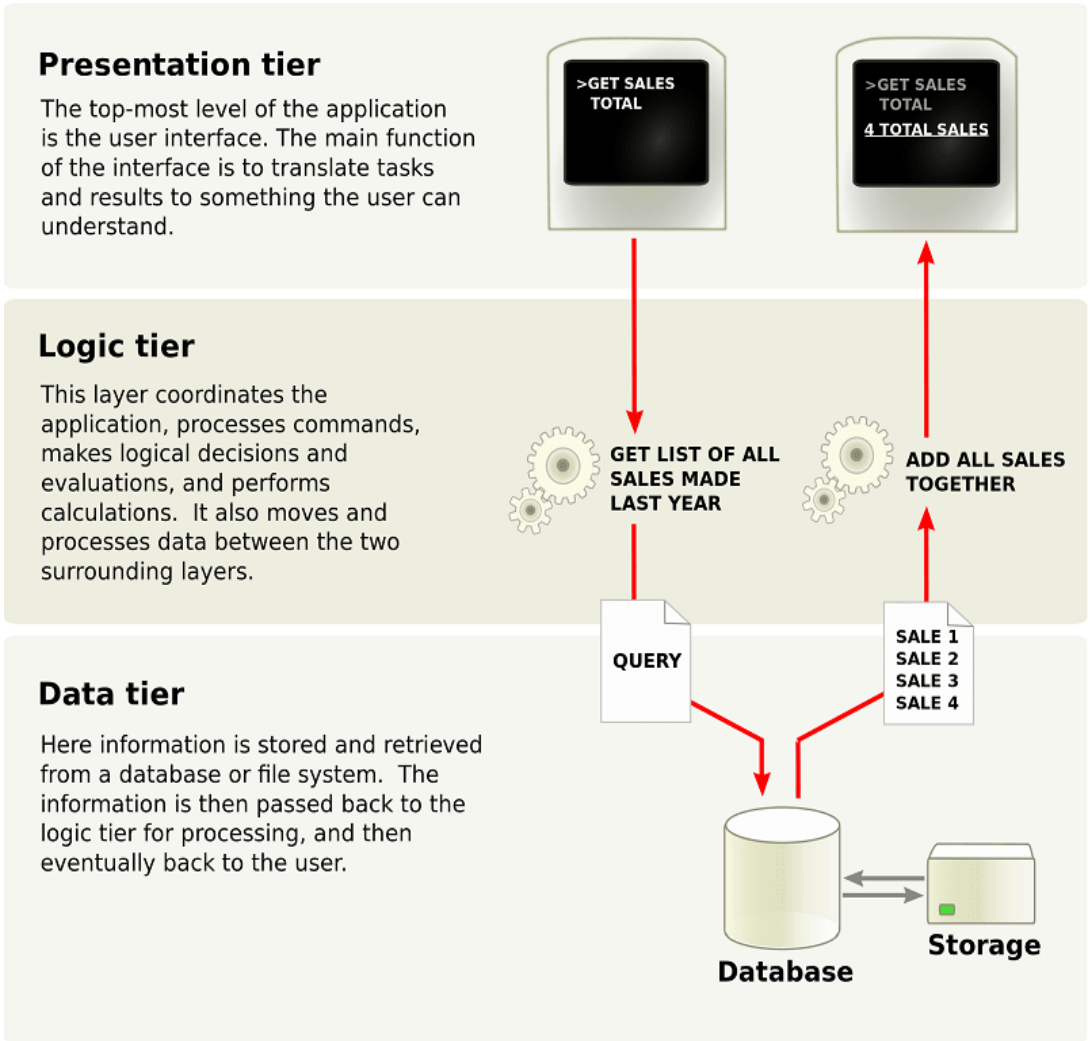


Two components communicate at a given level (layer):
For example: an FTP client makes a read request to an FTP server. At the FTP level you are not concerned about establishing a "session" – that is the responsibility of the TCP layer. The FTP layer simply implements its functionality using the TCP services.

EXAMPLE

N-Tier example

N-tier is effectively the same; often drawn horizontally instead of vertically. (example below is vertical)



Layers: Some benefits & drawbacks

\+ Layers improve modifiability & portability.

\+ Layers (should) have well-designed interfaces, which facilitates maintainability & testability.

\- Performance may suffer due to levels of indirection (imagine handling millions of events / sec.)

\- Not easy to establish right level of granularity of each layer.

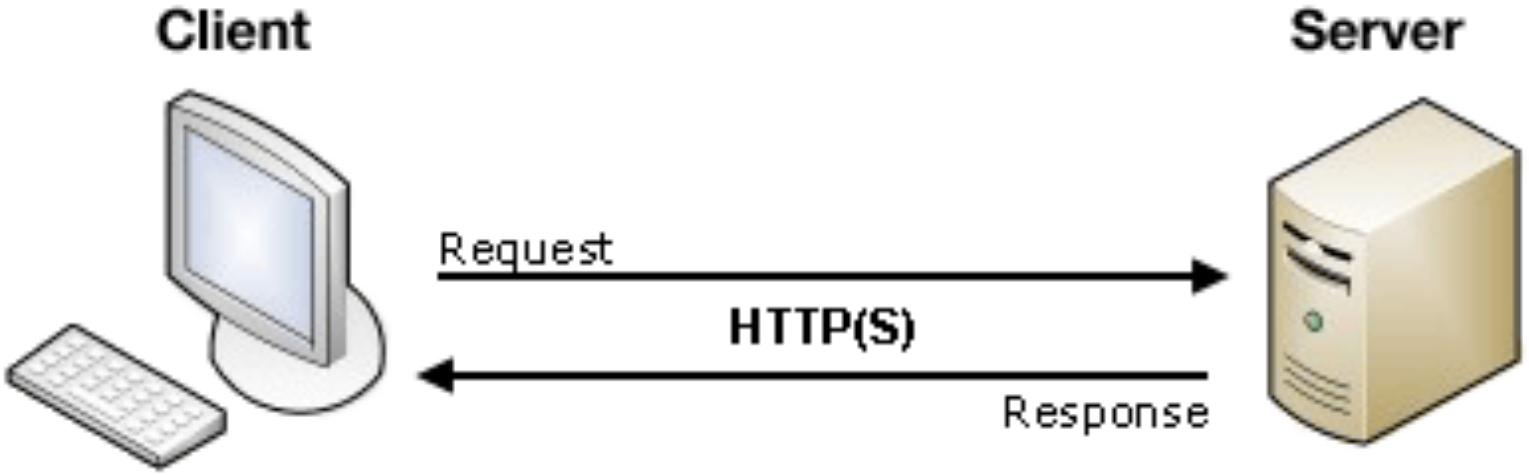
The Client-Server Style

The client-server style decomposes a system into a server that provides a service, and 1 or more clients that use that service.

- Both client and server are independent processes.
- Client initiates with a request, the server responds.
- Server must be able to handle multiple requests (clients) in parallel.

EXAMPLE

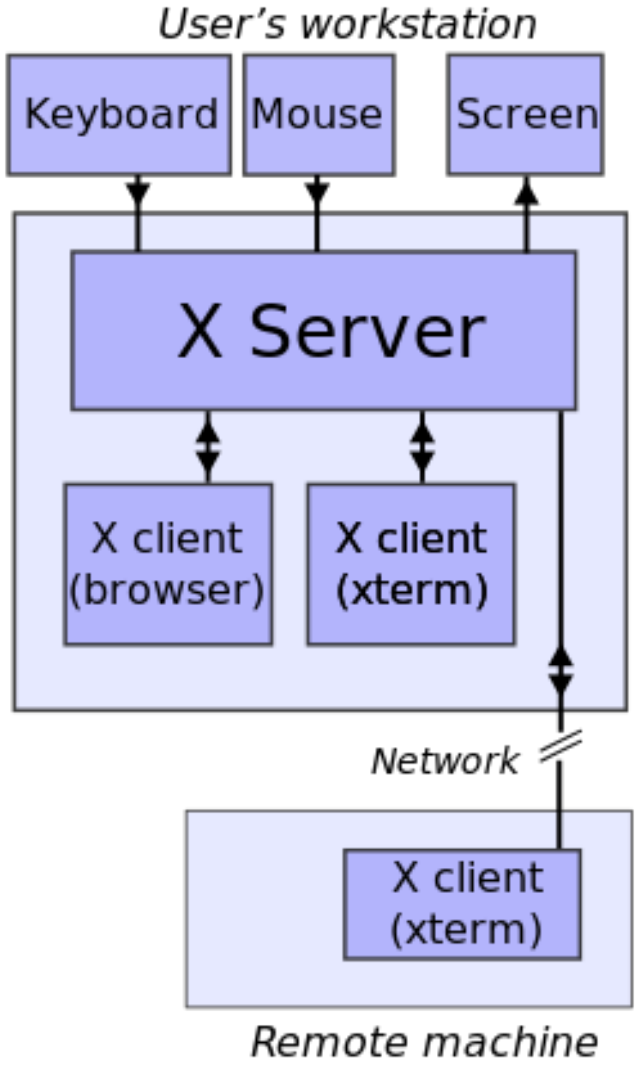
Client-Server architecture



Classic example of a client-server architecture is email: a client program (e.g. Outlook) connects to a mail server to retrieve new email. The client can, for example, make a request to delete the email or mark it as read.

EXAMPLE

X Window System (X11 or X)



- Clients are applications, e.g. a word processor. They make requests like "create window."
- The X Server "knows" how to create windows, and perform other GUI-specific operations.
- Communication over TCP/IP port 6000
- Used on Linux e.g. KDE

C/S: Some benefits and drawbacks

+ Modularity / Separation of concerns (e.g. only the X server needs to know how to create windows)

+ Scalability potential by adding more servers. (not always possible: e.g. cannot have 2 X servers)

+ Can reuse existing connectors e.g. TCP/IP or Remote Procedure Call (RPC) mechanisms.

- Robustness: server is single point of failure (replicating servers may not always be possible – e.g. X server).

- Traffic congestion if many requests arrive at server at the same time.

The Peer-to-Peer (P2P) Style

P2P is a generalized C/S style, where each peer component acts as both client and server.

- Each node (computer) runs a copy of the peer program.
- Each peer has interfaces describing the services they request from, and provide to, other peers.
- Requests are propagated through the network for a limited number of hops (TTL, Time-To-Live), to prevent requests hopping around forever, causing network congestion. Each hop, TTL decremented until 0.
- The P2P network may have supernodes that have indexing or routing capabilities.

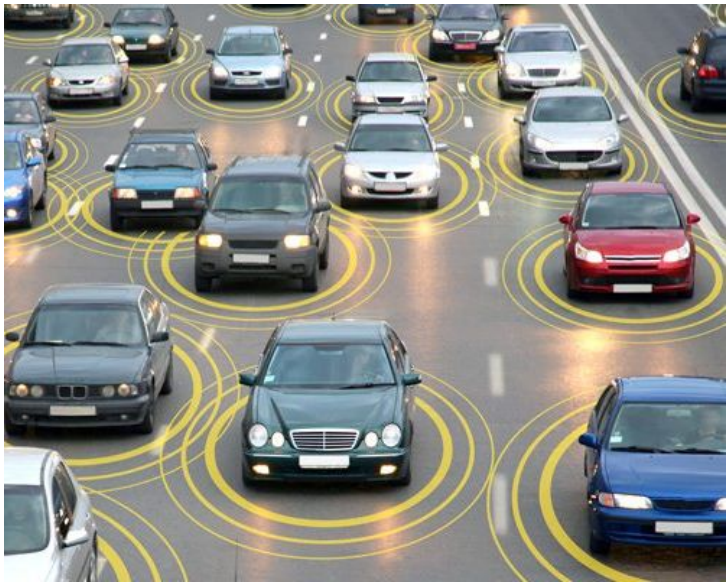
EXAMPLE

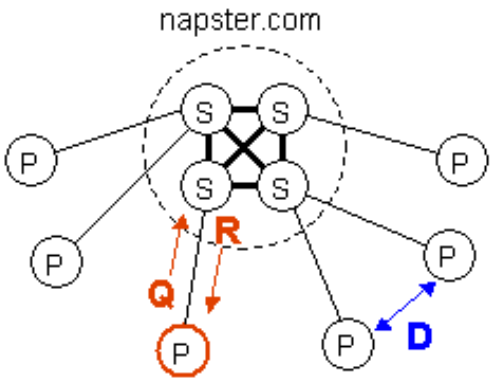
Gnutella, Napster, Bitcoin, future cars?

Bitcoin: A Peer-to-Peer Electronic Cash System

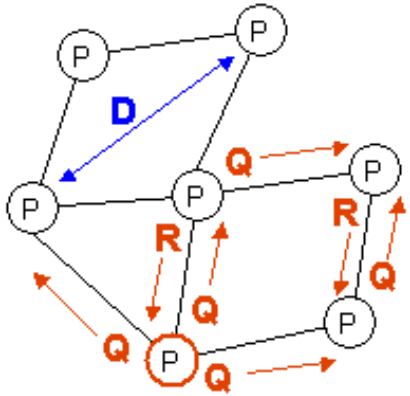
Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.





Napster



Gnutella

P peer
S server
Q query
R response
D file download

P2P: Some benefits and drawbacks

\+ Scalability: data (e.g. distributed files); no single node as bottleneck.

\+ Resilience (decentralized nature of network makes it hard to take it down)

\- Prediction: propagation of updates is not instant; self-organizing network might suffer from network congestion

\- Integrity: no “trusted” server (can you trust the files you download from BitTorrent?)

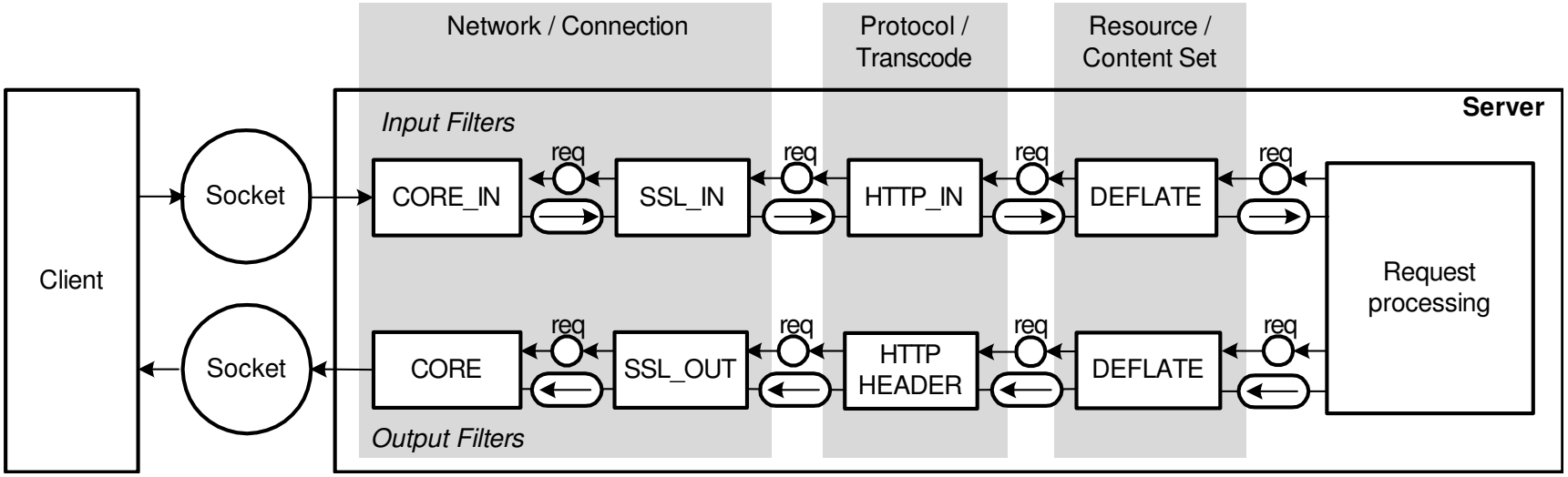
The Pipes-Filters Style

The Pipes-Filters style decomposes a large processing task into a sequence of smaller, independent processing steps.

- Each processing step is a filter
  - Defined input and output interfaces
- Filters are connected with pipes
  - Can be a buffer, file, etc.

EXAMPLE

Pipes-Filters: Apache 2 Filters



In figure 3.6 you see two example filter chains: The input filter chain to process the data of the request and the output filter chain to process the data of the response (provided by the content handler). The agent ``Request processing'' triggers the input filter chain while reading the request. An important use of the input filter chain is the SSL module providing secure HTTP (HTTPS) communication.

From:
http://www.fmc-modeling.org/category/projects/apache/amp/3_3Extending_Apache.html#SECTION005340000000000000000

P/F: Some benefits and drawbacks

\+ Performance (potential) when filters can be run in parallel.

\+ Reusability: Filters are loosely coupled modules with a well-defined interface.

\+ Can reuse existing connectors e.g. Linux pipes

\- Security: Must be implemented for each filter

\- Usability: usually not interactive.

\- Error handling is difficult.

Summary

- All architecture is design, but not all design is architectural.
- Architectural Significant Requirements: important to identify (using heuristics) because they impact the architecture.
- Quality Attributes are “ilities” – performance, security, availability, which are directly impacted by the choice of architectural style.
- Most systems use one or more architectural styles: understanding their consequences makes you a better architect.

Thank you for your attention

Questions & suggestions can be sent to:
k.stol@ucc.ie