NFPs

Reid Holmes
NFPs

- NFPs are constraints on the manner in which the system implements and delivers its functionality.
- E.g.,
  - Efficiency
  - Complexity
  - Scalability
  - Heterogeneity
  - Adaptability
  - Security
  - Dependability

[TAILOR ET AL.]
FP vs NFP

- Products are sold based on their FPs.
  - e.g., Cell phone, Car, Tent.

- However, NFPs play a critical role in perception.
  - “This program keeps crashing”
  - “It doesn’t work with my [...]”
  - “It’s too slow”
Design guidelines for NFPs

- Provide guidelines that support various NFPs.
- Focus on architectural level:
  - Components
  - Connectors
  - Topologies
NFP: Efficiency

› Efficiency is a quality that reflects a system’s ability to meet its performance requirements.

› Components:
  › Keep them “small”.
  › Simple and compact interfaces.
  › Allow multiple interfaces to the same functionality.
  › Separate data from processing components.
  › Separate data from meta data.

› Connectors:
  › Carefully select connectors.
  › Be careful of broadcast connectors.
  › Encourage asynchronous interaction.
  › Be wary of location/distribution transparency.

› Topology:
  › Keep frequent collaborators “close”.
  › Consider the efficiency impact of selected styles.
Multiple Interfaces
Distribution transparency
Topological distance
NFP: Complexity

- Complexity is a property that is proportional to the size of a system, its volume of constituent elements, their internal structure, and their interdependencies.

- Components:
  - Separate concerns.
  - Isolate functionality from interaction.
  - Ensure cohesiveness.
  - Insulate processing from data format changes.

- Connectors:
  - Isolate interaction from functionality.
  - Restrict interactions provided by each connector.

- Topology:
  - Eliminate unnecessary dependencies.
  - Use hierarchical (de)composition.
Connector complexity

[Diagram with nodes and arrows indicating connectivity between components such as Memory Manager, Network Interface, Process Scheduler, Inter-Process Communications, Initialization, and Library.]

[TAILOR ET AL.]
NFP: Scalability / Heterogeneity

- **Scalability**: The capability of a system to be adapted to meet new size / scope requirements.

- **Heterogeneity**: A system’s ability to be composed of, or execute within, disparate parts.
  - **Internal**: Ability to accommodate multiple kinds of components and connectors.
  - **External**: Ability to adjust to different platforms and environments (e.g., portability).

- **Portability**: The ability of a system to execute on multiple platforms while retaining their functional and non-functional properties.
NFP: Scalability / Heterogeneity

- Components:
  - Keep components focused (avoid bottlenecks).
  - Simplify interfaces (ease adding new components).
  - Avoid unnecessary heterogeneity (arch mismatch).
  - Distribute data sources (avoid bottlenecks).
  - Replicate data (caution: mutable vs immutable data).

- Connectors:
  - Use explicit connectors (natural scaling points).
  - Clearly define connector responsibilities (avoid bottlenecks).
  - Choose the simplest connectors (complexity dec. perf.).
  - Direct vs. indirect connectors (loose coupling, easy ext.).

- Topology:
  - Avoid bottlenecks.
  - Place data close to consumer (reduce network traffic).
  - Location transparency (move / expand services, data).
NFP: Evolvability

- Evolvability: The ability to change to satisfy new requirements and environments.

- Components:
  - Same as for complexity.
    - Goal is to reduce risks by isolating modifications.

- Connectors:
  - Clearly define responsibilities (make it easy track risk).
  - Make connectors flexible.
  - Enable connector composition (support new comps.).

- Topology:
  - Avoid implicit connectors (hard to understand).
  - Encourage location transparency (supports obliviousness).
NFP: Evolvability
NFP: Dependability

- Reliability: The probability a system will perform within its design limits without failure over time.
- Availability: The probability the system is available at a particular instant in time.
- Robustness: The ability of a system to respond adequately to unanticipated runtime conditions.
- Fault-tolerance: The ability of a system to respond gracefully to failures at runtime.
  - Faults arise from: environment, components, connectors, component-connector mismatches.
- Survivability: The ability to resist, recover, and adapt to threats.
  - Sources: attacks, failures, and accidents.
  - Steps: resist, recognize, recover, adapt.
- Safety: The ability to avoid failures that will cause loss of life, injury, or loss to property.
NFP: Dependability

- **Components:**
  - Control external component dependencies (insulation).
  - Support reflection (e.g., querying about health).
  - Support exception handling (adjust to failures).
  - Specify key state invariants (best, normal, worst-case).

- **Connectors:**
  - Use explicit connectors (insulate components).
  - Provide interaction guarantees (know how to react).
  - Use advanced connectors (replicas, mocks, etc.).
    - [Support seamless dependability]

- **Topology:**
  - Avoid single points of failure.
  - Enable back-ups (e.g., via advanced connectors).
  - Support system health monitoring (e.g., perf. analysis).
  - Support dynamic adaptation (e.g., dynamic discovery).
NFP: Security

- Security: “The protection afforded a system to preserve its **integrity**, **availability**, and **confidentiality** if its resources.”

- Confidentiality
  - Preserving the **confidentiality** of information means preventing unauthorized parties from accessing the information or perhaps even being aware of the existence of the information. I.e., secrecy.

- Integrity
  - Maintaining the **integrity** of information means that only authorized parties can manipulate the information and do so only in authorized ways.

- Availability
  - Resources are **available** if they are accessible by authorized parties on all appropriate occasions.
Security arch. principles

- Least privilege:
  - Give each component only the privileges it requires.

- Fail-safe defaults
  - Deny access if explicit permission is absent.

- Economy of mechanism
  - Adopt simple security mechanisms.

- Open design
  - Secrecy != security.
Security arch. principles

- Separation of privilege
  - Introduce multiple parties to avoid exploitation of privileges.

- Least common mechanism
  - Limit critical resource sharing to only a few mechanisms.

- Psychological acceptability
  - Make security mechanisms usable.

- Defence in depth
  - Have multiple layers of countermeasures.
Access control

- Decide whether access should be granted.
  - Discretionary:
    - Based on the accessor’s identity, the resources, and whether the accessor has permissions.
  - Mandatory:
    - Policy based. (e.g., dominating labels)
  - Cross-cutting concern that should be investigated at an architectural level.
## Discretionary access control

<table>
<thead>
<tr>
<th></th>
<th>DB</th>
<th>Component</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Read-write; always</td>
<td>Bend</td>
<td>Y</td>
</tr>
<tr>
<td>Bob</td>
<td>Read-write; Between 9-5</td>
<td>Fold</td>
<td>N</td>
</tr>
<tr>
<td>Charles</td>
<td>No access</td>
<td>Spindle</td>
<td>N</td>
</tr>
<tr>
<td>Dave</td>
<td>No access</td>
<td>Mutilate</td>
<td>Y</td>
</tr>
<tr>
<td>Eve</td>
<td>Read-only; Always</td>
<td>Non</td>
<td>N</td>
</tr>
</tbody>
</table>
Mandatory access control
Trust management

- Trust is a subjective probability with which one agent assesses another agent will perform some specific action within a specific context.

- Reputation is the expectation of an agent’s behaviour based on their past behaviours.

- Trust cannot be isolated to individual components.
  - Dominant concern in decentralized applications.
  - Architecture provides a foundation for reasoning about trust-related issues.