Chapter 6

by

David G. Messerschmitt

Goal

• Appreciate the importance of complexity management in networked computing
• Understand better the role of architecture in complexity management
• Examine infrastructure layering in more depth
Complexity

- A system that cannot be understood in all its detail by a single person or small group of people is complex
- The intricacy of the logic embodied in software
  - suffers no physical limitations
  - complexity is a primary limitation
  - advances allow us to extend that complexity

Some sources of complexity

- Problem domain is complex
- Top-down design (as opposed to independent actors in the economy)
- Software is not adaptable like people
- Large team efforts required
- Integration of heterogeneous suppliers
Caution

- The applications considered in this course are relatively simple.
- We have addressed:
  - only the top of the hierarchy
  - ignored details
  - but this is the essence of hierarchical design: make that which is complex appear simple.

Some solutions to complexity

- Modularity properties
  - separation of concerns
  - reuse
- Interoperability through interfaces
  - abstraction
  - encapsulation
Modularity

- A system is modular when it is divided into subsystems (called modules) with good properties
  - Modules have distinct functional groupings
  - Hierarchy supports views at different granularity and scale
  - Separation of concerns among modules
  - Reusability of some modules

Hierarchy

- Software: Allows a system to be understood at different granularity
- Organization: Allows a manager to focus on high-level objectives, delegating low-level detail
Hierarchy in hardware architecture

Separation of concerns

• The assignment of functionality to different modules should allow them to be designed and implemented as independently as possible.

• The level of interaction
  – may be internally high
  – should be externally low

• They can then be assigned to different groups or companies for design
  – minimum coordination costs
### Physical-world example

<table>
<thead>
<tr>
<th>Poor modularity</th>
<th>Customer service</th>
<th>Credit checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer service</td>
<td>Janitorial</td>
<td>Floor polishing</td>
</tr>
<tr>
<td>Loan department</td>
<td>Physical plant</td>
<td></td>
</tr>
<tr>
<td>Better modularity</td>
<td>Customer service</td>
<td>Credit checking</td>
</tr>
<tr>
<td></td>
<td>Janitorial</td>
<td>Floor polishing</td>
</tr>
</tbody>
</table>

### Infrastructure example

<table>
<thead>
<tr>
<th>Poor modularity</th>
<th>Application</th>
<th>End-to-end network</th>
<th>Switch-to-switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Better modularity</td>
<td>Application</td>
<td>End-to-end network</td>
<td>Switch-to-switch</td>
</tr>
<tr>
<td>Host</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Level of interaction high
**Parts of a module**

- Module
- Interface
- Implementation

What other modules see: Interface
What only the implementer sees: Implementation

---

**Interfaces**

- Focus of module interaction and interoperability
- Two purposes:
  - Informs other modules how to interact
  - Informs implementer about what has been promised to other modules
Hardware interface

- Physical connection
- Electrical properties
- Formats of data passing through the interface (structure and interpretation)

Possible software interface

Menu of actions

What are some other examples of types of interaction at interfaces?
Module interaction through interfaces

Data customizing an action and disclosing its results

action, parameters

returns

Both subsystems are affected by the interaction

Data types

• Data passing an interface is often specified in terms of a limited number of standard data types
• Data type = range of values and allowable manipulation
• Data type does not presume a specific representation, to allow heterogeneous platforms
  – Representation must be known when data passes a specific module interface
Example data types

- Integer
  - “natural number between -32,767 and +32,768”
  - Could be represented (in many ways) by 16 bits
    - since $2^{16} = 65,536$
- Float
  - “number of the form $m \times 10^n/32768$, where $m$ is in the range -32,767 to +32,768 and $n$ is in the range -255 to +256”
  - Could be represented by $16 + 8 = 24$ bits

More data types

- Character
  - “values assuming a-z and A-Z plus space and punctuation marks”
    - could be represented by 7 or 8 bits
- Character string
  - “collection of $n$ characters, where $n$ is customizable”
    - could be represented by $7 \times n$ bits
Compound data types

- **Programmer-defined composition of basic data types**

- **Example:**
  ```
  Employee {
      String name;
      String address;
      Integer year_of_birth;
      etc.
  }
  ```

Protocols

- A defined **sequence** of actions between/among two or more subsystems required to achieve some higher-level functionality

- Interface specification focuses on **actions** (including formats of parameters and returns) *and protocols*
Example protocol: deposit

- `set_balance`
- `add deposit amount`
- `set_balance`

Anatomy of an action invocation

① Decides it needs to invoke an action of a server module
② Invokes the action by name
③ Passes parameter data to the server
④ Processes parameters in accordance with the specified action; generates return values
⑤ Passes the return values back to the client
⑥ Process the return values to complete the interaction
More on layering

by

David G. Messerschmitt

Goals

• Understand better
  – how layering is used in the infrastructure
  – how it contains complexity
  – how it coordinates suppliers
  – how it allows new capabilities to be added incrementally
Understanding Networked Applications

Layer above is a client of the layer below.

Layer below as a server to the layer above.

Each layer provides services to the layer above.

….by utilizing the services of the layer below and adding capability.

Interaction of layers

Host A

Application module

Middleware

Operating system

Network

Host B

Application module

Middleware

Operating system
Layering

Elaboration or specialization

Existing layers

Layering builds capability incrementally by adding to what exists

Three types of software

Application

• Components and frameworks:
  What is in common among applications

• Infrastructure:
  Basic services (communication, storage, concurrency, presentation, etc.)
Part of Microsoft vs. DOJ dispute

Understanding Networked Applications

Major layers

Understanding Networked Applications
Open layer interfaces

- Applications
- Application components
- Middleware
- Operating system
- Network

Data and information

- Application
  - Deals with information
  - Assumes structure and interpretation
- Infrastructure
  - Deals with data
  - Ignores structure and interpretation
Data and information in layers

- The infrastructure should deal with data, or at most minimal structure and interpretation of data suitable for a wide range of applications
- The application adds additional structure and interpretation
- This yields a separation of concerns

Package = file, message

- In the simplest case, the infrastructure deals with a package of data (non-standard terminology)
  - collection of bits
    - specified number and ordering
- The objective of the infrastructure is to store and communicate packages while maintaining data integrity
- File for storage, message for communication
Data integrity

- Retain the
  - values
  - order
  - number

of bits in a package

Example

Web server

Screen
Web browser
HTML

Application

Operating system

File system

File
Message

Network

Fragmentation
Collection of packets
Assembly

Understanding Networked Applications 37
A First Course

Understanding Networked Applications 38
A First Course
Information in the infrastructure

- Sometimes it is appropriate for the infrastructure to assume structure and interpretation for data
  - to add capabilities widely useful to applications
  - to help applications deal with heterogeneous platforms, where representations differ
- At most, data types

Data and information

Application
- Deals with information
  - Assumes structure and interpretation
  - Assumes standard data types

Infrastructure
- Deals with data types
Storage

Application
Deals with information

Assumes standard data types
and SQL = structured query language

Database management system (DBMS)
File system

The infrastructure
can provide data management functions

Communication

Application
Deals with information

Assumes standard data types
and performs conversions

Distributed object management
Network

The infrastructure can transparently convert representations across platforms
Idea behind remote action invocation

Using a common intermediate form

Perform all conversions

Convert to/from common representation
Representation is a coding of information by data in a form that can be manipulated by a lower layer; the results remain meaningful at the higher layer.

Information is data with known and consistent structure and interpretation in the context of the current layer.
Information appliances

Middleware
Operating system
Network
IA
Application

Question

• What advantages and disadvantages do you see for the information appliance?
Horizontal structure in layers

A spanning layer is ubiquitous and hides the layers below

Spanning layer

A spanning layer is ubiquitous and hides the layers below
### Abstraction

- A property of well-designed interfaces to modules
- Hide detail, displaying only what is necessary
- Simplify, displaying only what is meaningful to the outside
- Important for complexity management

### Encapsulation

- Module implementation details (anything not explicit at interface) should be inaccessible from the outside
  - So other modules cannot become inadvertently dependent on implementation
  - In the case of components, for proprietary or security reasons
Summary of modularity

- **Divide and conquer**: decomposition of the system into modules with well-defined functional groupings
- **Separation of concerns**: great dependency internally, little dependency externally
- **Abstraction**: hide detail and simplify
- **Encapsulation**: make internal implementation inaccessible
- **Reusability**: meet generalized needs, configurable