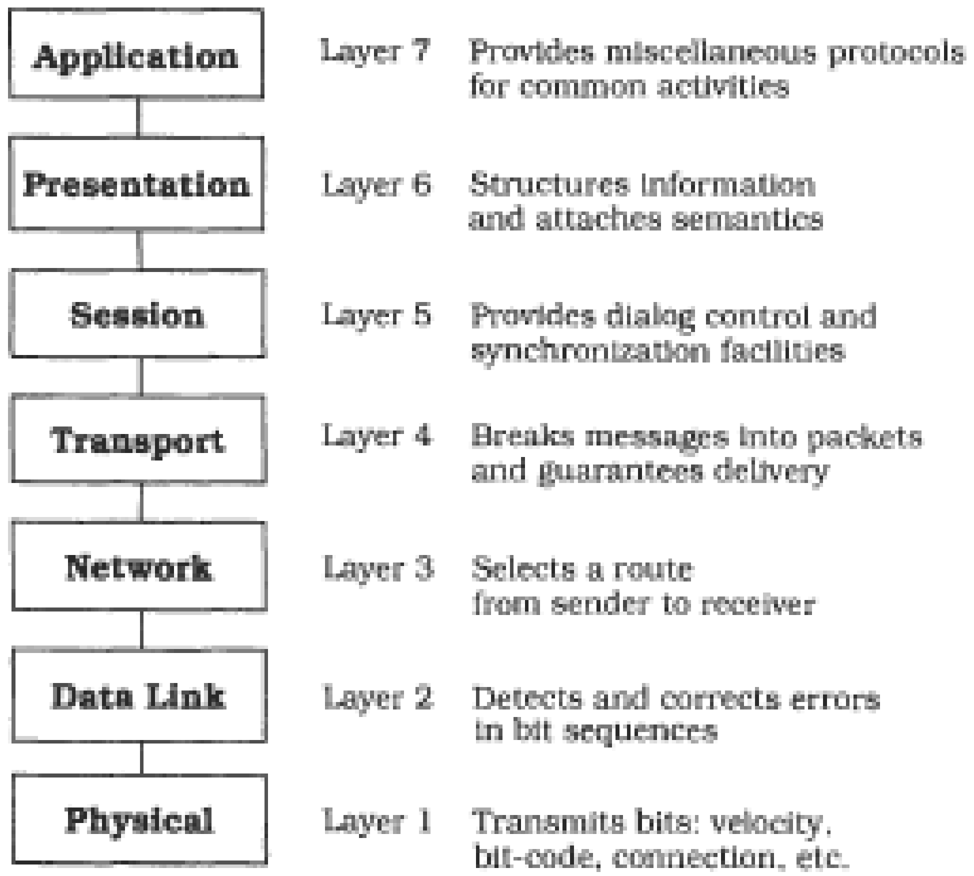
Layers*

The layers architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction.

**Pattern-Oriented Software Architecture Volume 1: A System of Patterns Volume 1 Edition by Frank Buschmann , Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal*

Example


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

Context

A large system that requires decomposition.

“From mud to structure”

Problem (1/2)

Imagine that you are designing a system whose dominant characteristic is a mix of low- and high-level issues, where ***high-level operations rely on the lower-level ones***.

Some parts of the system handle low-level issues such as hardware traps, sensor input, reading bits from a file or electrical signals from a wire.

At the other end of the spectrum there may be user-visible functionality such as the interface of a multi-user 'dungeon' game or high-level policies such as telephone billing tariffs.

Problem (2/2)

A typical pattern of communication flow consists of ***requests moving from high to low leve***l, and ***answers to requests, incoming data or notification about events traveling in the opposite direction***.

…

The system specification provided to you describes the high-level

tasks to some extent, and specifies the target platform. ***Portability to***

other platforms is desired.

…

The ***mapping of high-level tasks onto the platform is not straightforward***, mostly because they are too complex to

be implemented directly using services provided by the platform.

Forces (1/2)

- Late source code changes should not ripple through the system.
- Interfaces should be stable
- Parts of the system should be exchangeable.
- It may be necessary to build other systems at a later date with the same low-level issues as the system you are currently designing.

Forces (2/2)

- Similar responsibilities should be grouped to help understand.
- There is no 'standard' component granularity.
- Complex components need further decomposition.
- Crossing component boundaries may impede performance
- The system will be built by a team of programmers, and work has
- to be subdivided along clear boundaries

Solution (1/2)

… Structure your system into an appropriate number of layers and place them on top of each other.

Start at the lowest level of abstraction-call it Layer 1. This is the base of your system.

Work your way up the abstraction ladder by putting Layer J on top of Layer J - 1 until you reach the top level of functionality-call it Layer N.

Solution (2/2)

… Most of the services that Layer J provides are composed of services provided by Layer J -1.

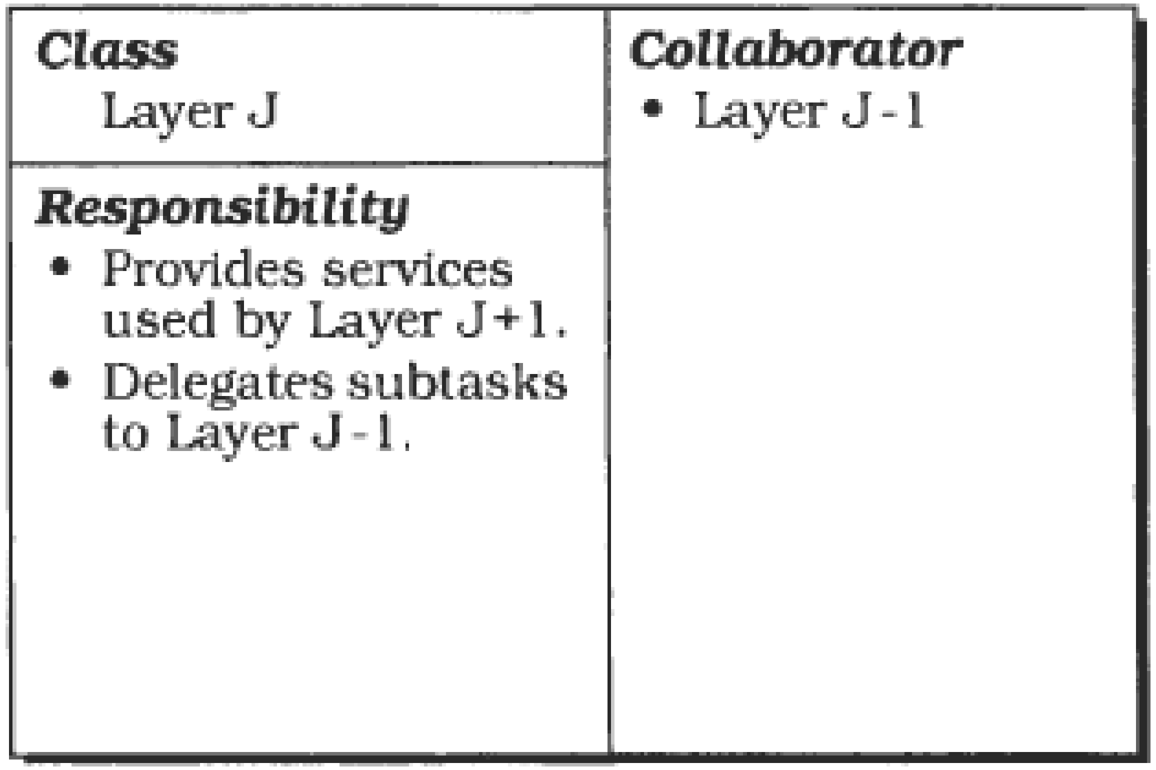
In other words, the services of each layer

implement a strategy for combining the services of the layer below in a meaningful way.

In addition, Layer J's services may depend on other

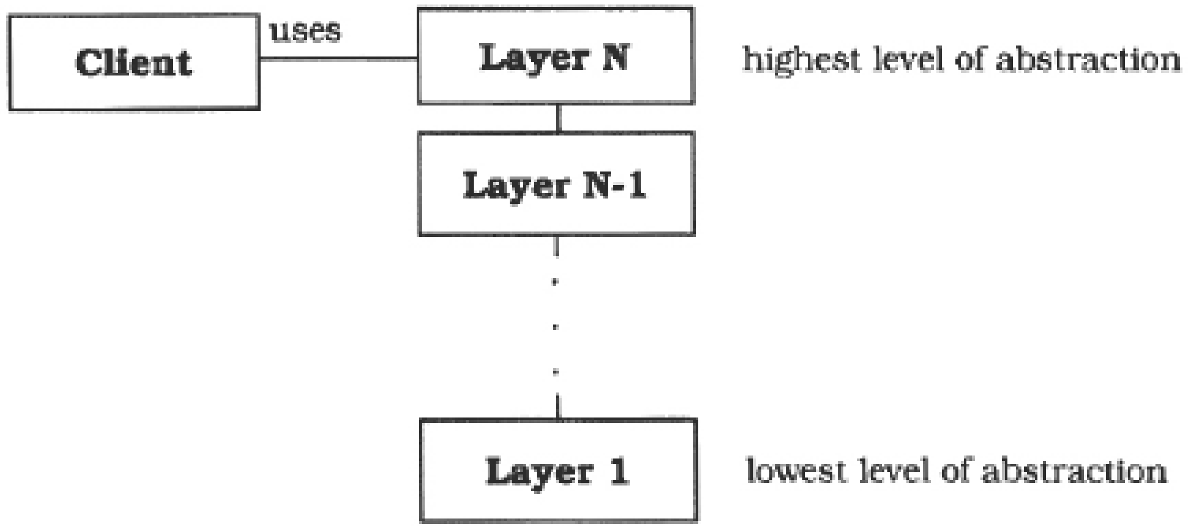
services in Layer J.

Structure (1/3)

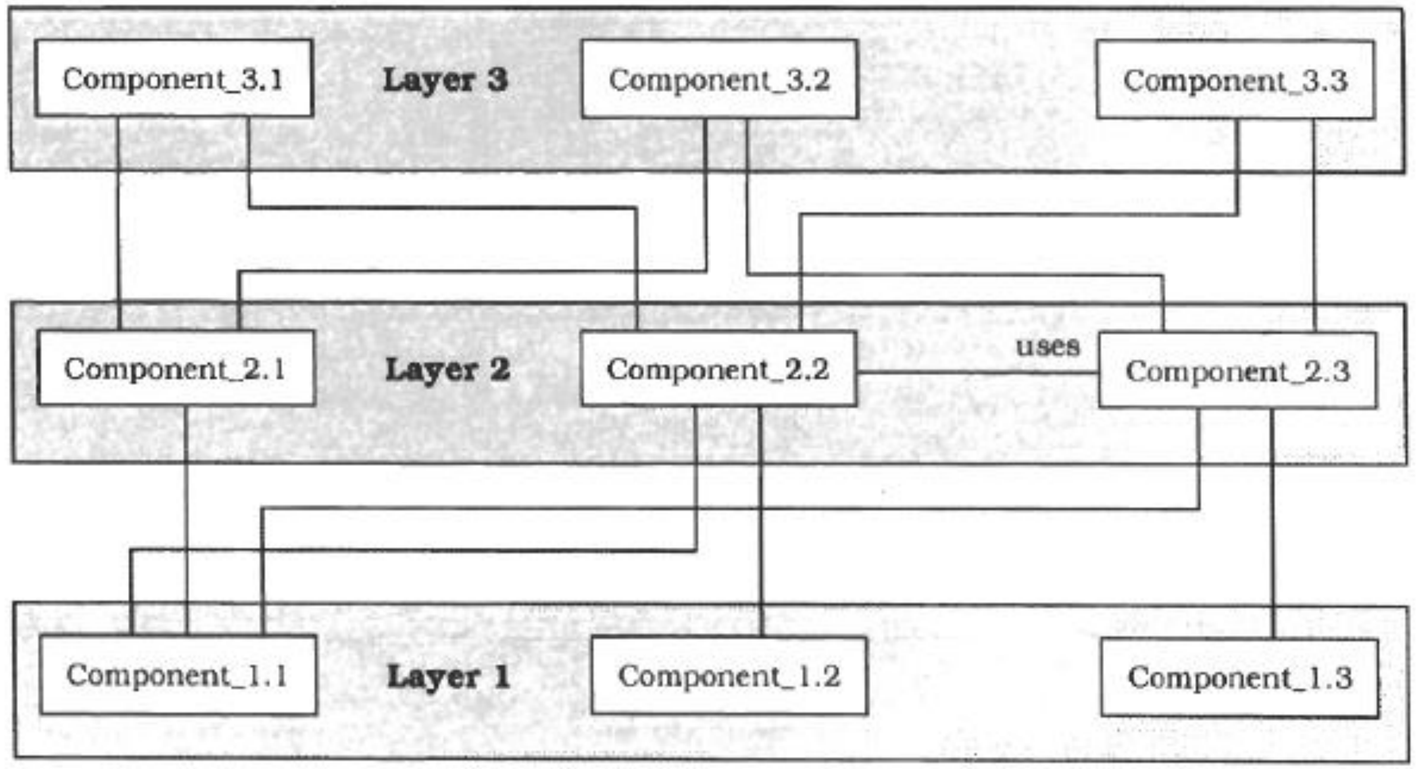


Class Layer J	***Collaborator***
Responsibility • Provides services used by Layer J+1. • Delegates subtasks to Layer J-1.	• Layer J-1

Structure (2/3)



Structure (3/3)



Dynamics (1/3)

Scenario I is probably the best-known one. (see program)

A client Issues a request to Layer N. Since Layer N cannot carry out the request on its own. It calls the next Layer N - 1 for supporting subtasks. Layer N - I provides these.

In the process sending further requests to Layer N-2. and so on until Layer I 1s reached. Here, the lowest-level services are finally performed.

If necessary, replies to the different requests are passed
back up from Layer 1 to Layer 2, from Layer 2 to Layer 3, and so on
until the final reply arrives at Layer N.

Dynamics (2/3)

Scenario II illustrates bottom-up communication

-a chain of actions starts at Layer 1, for example when a device driver detects input.

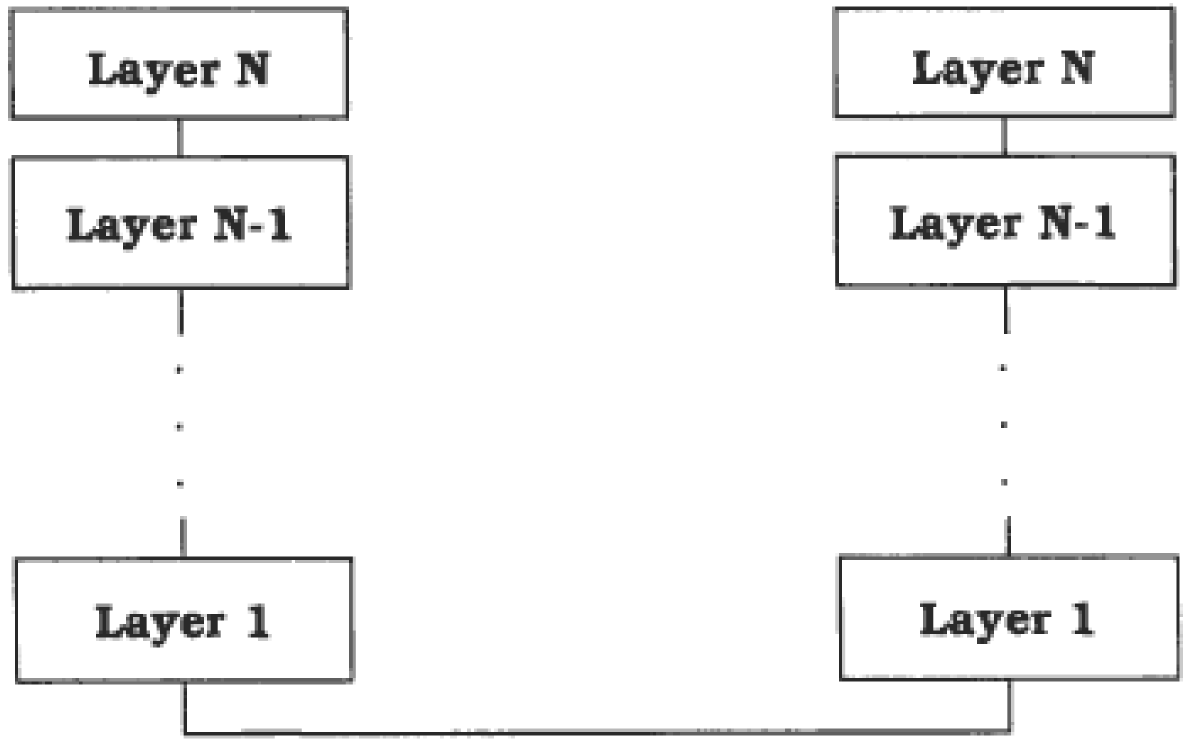
The driver translates the input into an Internal format and reports it to Layer 2, which starts interpreting it, and so on.

In this way data moves up through the layers until It arrives at the highest layer.

While top-down information and control flow are often described as '***requests***'. bottom-up calls can be termed '***notifications***'.

Dynamics (3/3)

Scenario V



Implementation

1. ***Define the abstraction criterion*** for grouping tasks into layers.
2. ***Determine the number*** of ***abstraction levels*** according to your abstraction criterion.
3. ***Name the layers and assign tasks to each*** of ***them***
4. ***Specify the services.***
5. ***Refine the layering.*** Iterate over steps 1 to 4.

implementation

- Decouple adjacent layers
- Design an error-handling strategy

Consequences

- *Reuse of layers.*
- *Support for standardization*
- *Dependencies are kept local*
- *Exchangeability.*
- *Cascades of changing behavior*
- *Lower efficiency*
- *Unnecessary work*
- *Difficulty of establishing the correct granularity of layers*