Embedded Systems

Design Patterns for Embedded Systems

Design Patterns for Embedded Systems

- Categories of Design Patterns include:
  - Accessing hardware
  - Concurrency and Resource Management
  - State Machine implementation and usage
  - Safety and Reliability

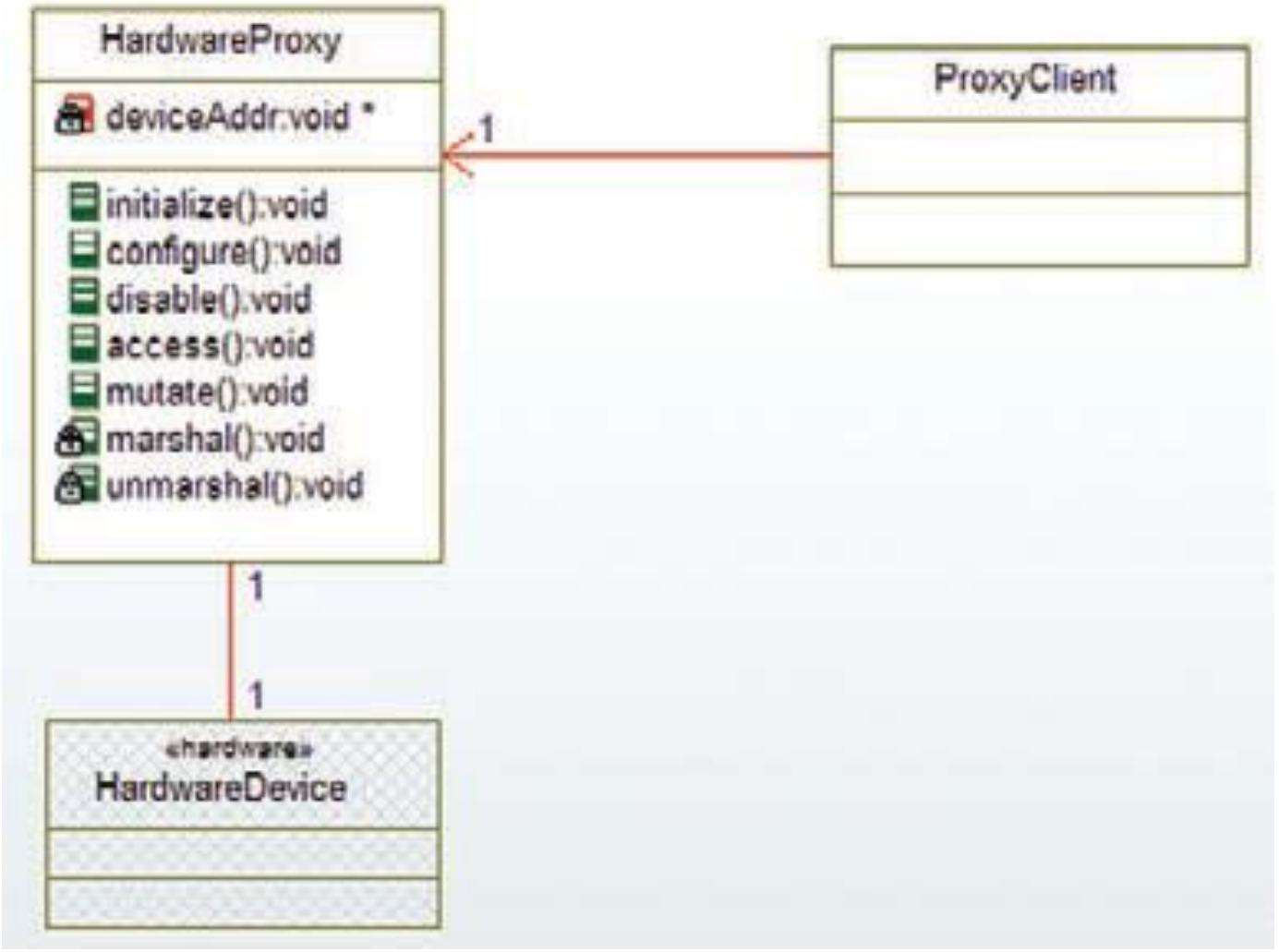
Hardware Proxy Pattern - Abstract

- Creates a software element responsible for access to a piece of hardware and encapsulation of hardware compression and coding implementation.
- Uses a class (or struct) to encapsulate all access to a hardware device, regardless of its physical interface.
- Provides an encoding and connection-independent interface for clients and so promotes easy modification should the nature of the device interface or connection change.

Hardware Proxy Pattern - Problem

- By providing a proxy to sit between the clients and the actual hardware, the impact of hardware changes is greatly limited, easing any modifications in the hardware.

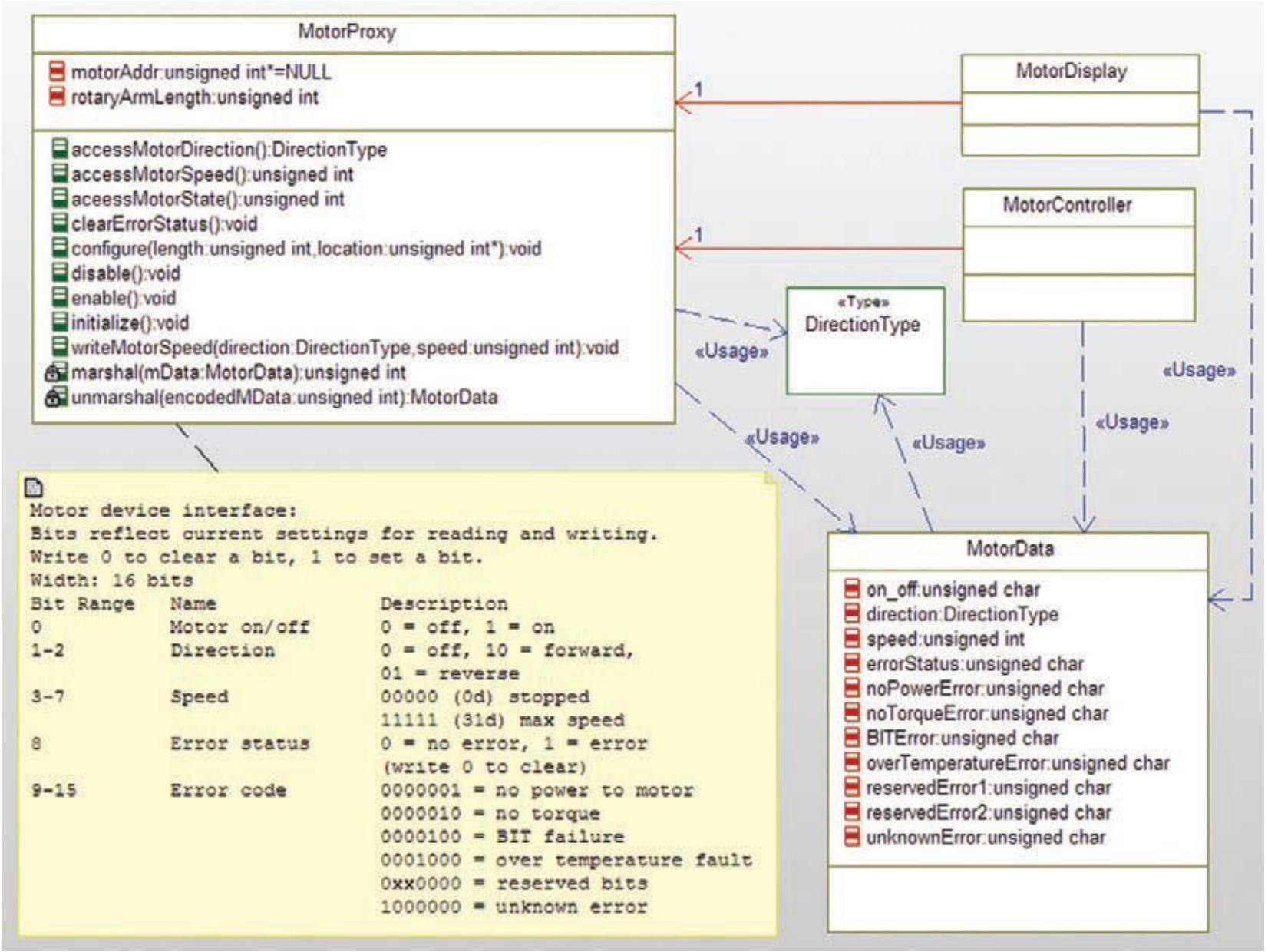
Hardware Proxy Pattern - Structure



Hardware Proxy Pattern - Consequences

- It provides flexibility for the actual hardware interface to change with absolutely no changes in the clients.
- This means that the proxy clients are usually unaware of the native format of the data and manipulate them only in presentation format.
- This can, however, have a negative impact on run-time performance.
- It may be more time efficient for the clients to know the encoding details and manipulate the data in their native format.

Hardware Proxy Pattern - Example



Hardware Proxy Pattern - Example

- Motor Management Functions
  - **configure()** – this function sets up the memory-mapped address of the motor and the length of the rotary arm; this function must be called first
  - **disable()** – this function turns off the motor but keeps the set values intact
  - **enable()** – this function turns the motor on with the current set values
  - **initialize()** – this function turns on the motor to default set values (off)

Hardware Proxy Pattern - Example

- Motor Control Functions
  - **writeMotorSpeed() –** sets the speed of the motor and adjusts for the length of the rotary arm (if set)
- Motor Error Management Functions
  - **clearErrorStatus()** – clears all error bits
  - **accessMotorState()** – returns the error status

Hardware Proxy Pattern - Example

- Internal Data Formatting Functions (private)
  - These functions aren't provided to the clients but are used internally to exchange the data between native and presentation formats.
  - • **marshal()** – converts presentation (client) data format to native (motor) format
  - • **unmarshal()** – converts native (motor) data format to presentation (client) format

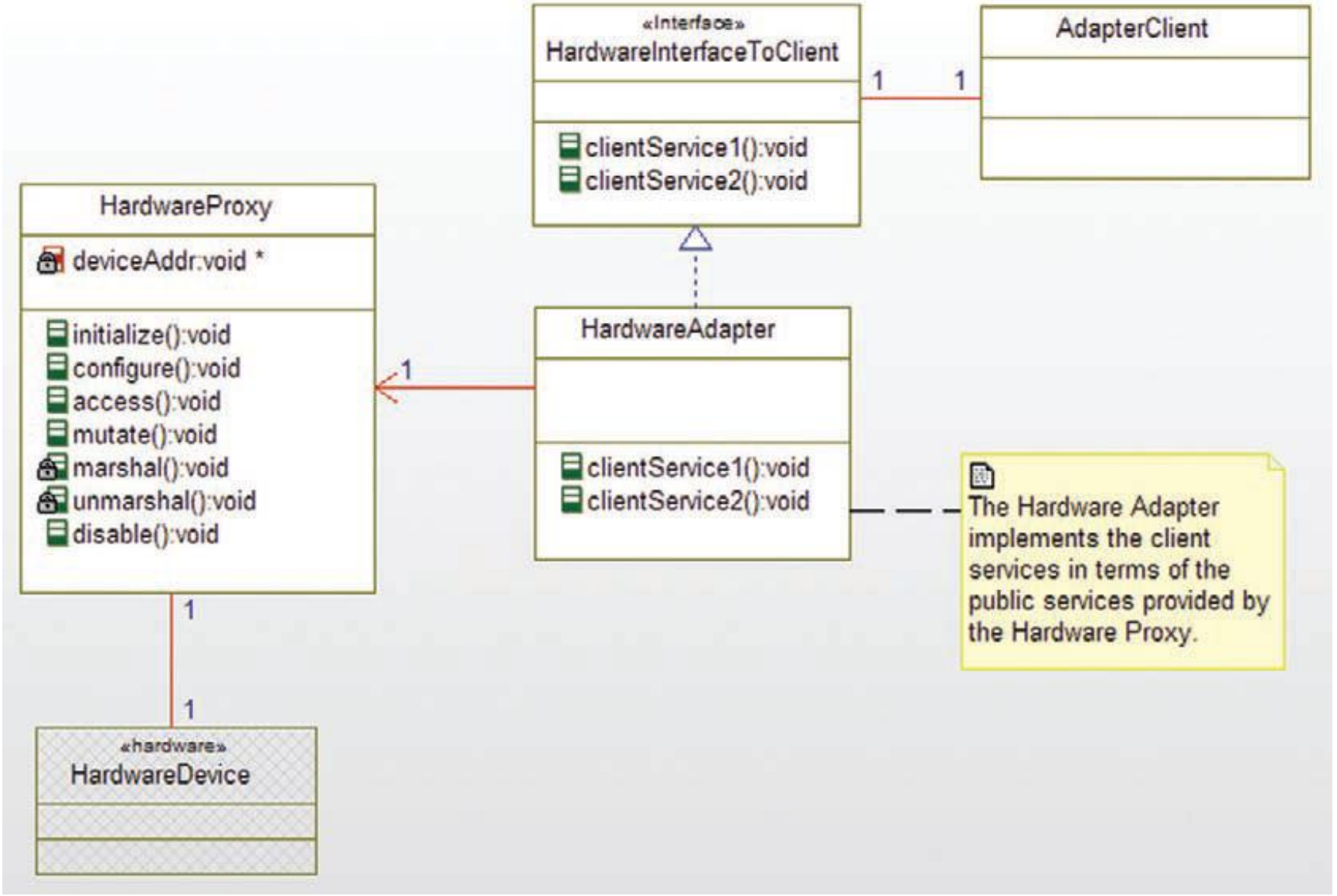
Hardware Adapter Pattern - Abstract

- Provides a way of adapting an existing hardware interface into the expectations of the application.
- It is useful when the application requires or uses one interface, but the actual hardware provides another.
- Creates an element that converts between the two interfaces.

Hardware Adapter Pattern – Problem

- It is useful when you have hardware that meets the semantic need of the system but that has an incompatible interface.
- The goal of the pattern is to minimize the reworking of code when one hardware design or implementation is replaced with another.

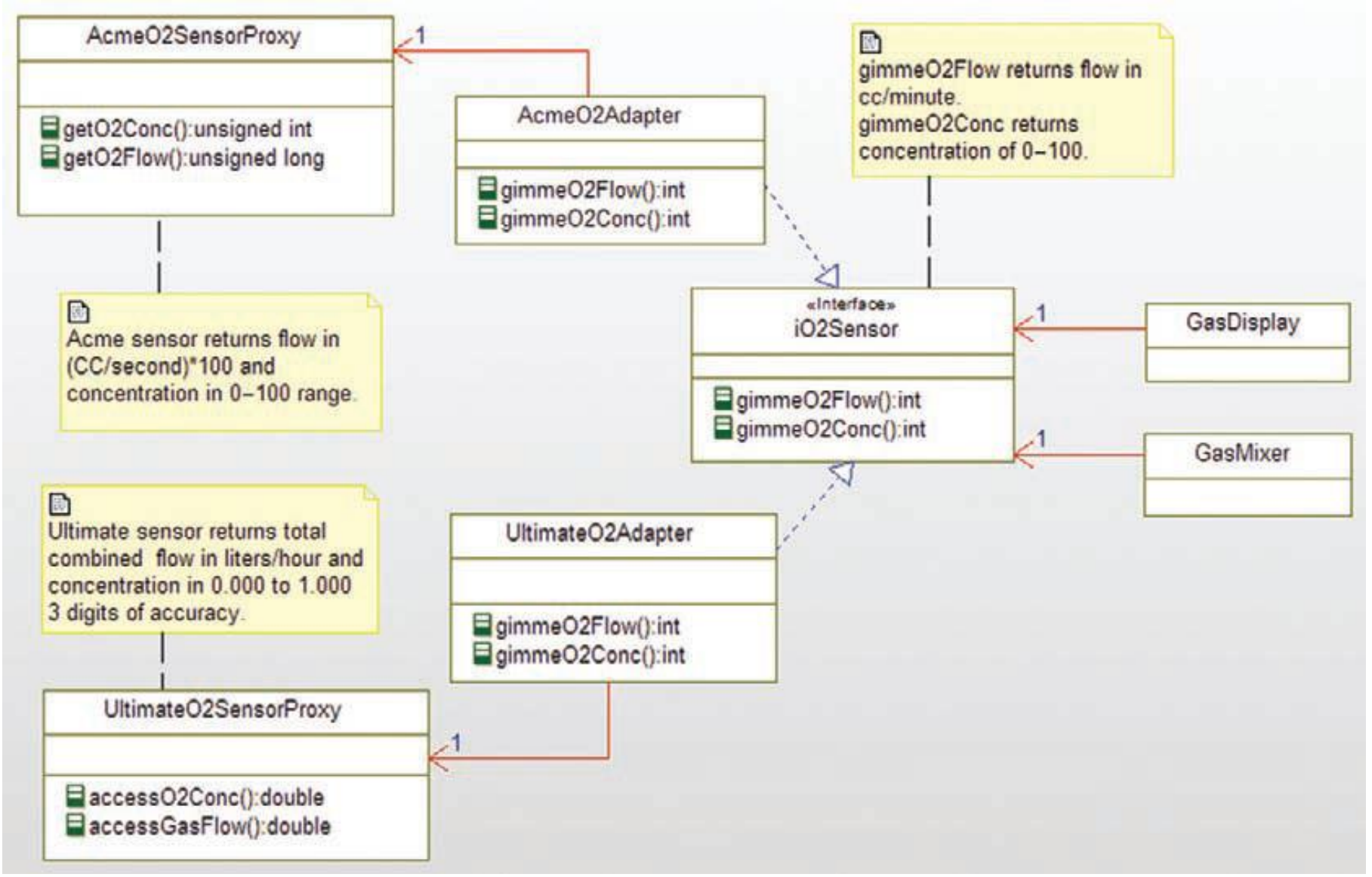
Hardware Adapter Pattern - Structure



Hardware Adapter Pattern -Consequences

- It allows various Hardware Proxies and their related hardware devices to be used as-is in different applications, while at the same time allowing existing applications to use different hardware devices without change.
- However, this pattern adds a level of indirection and therefore decreases runtime performance.

Hardware Adapter Pattern - Example



References

- **Chapter 3**: Douglass, Bruce Powel. **Design patterns for embedded systems in C: an embedded software engineering toolkit**. Elsevier, 2010.