Embedded Systems

Design Patterns for Embedded Systems

Design Patterns for Embedded Systems

- What is a Design Pattern?
- Basic Structure of a Design Pattern
- How to Read Design Patterns
- Using Design Patterns in Development
  - Pattern Hatching – Locating the Right Patterns
  - Pattern Mining – Rolling Your Own Patterns
  - Pattern Instantiation – Applying Patterns in Your Designs
- Example - Observer Design Pattern

What is a Design Pattern?

- “A generalized solution to a commonly occurring problem.”
- A problem must occur enough to be usefully generalizable
- The solution must be general enough to be applied in a wide set of application domains
- A way of organizing a design that improves the optimality with respect to one or a small set of design criteria, such as QoS

What is a Good Design?

- A good design is
  - composed of a set of design patterns applied to a piece of functional software
  - achieves a balanced optimization of the design criteria
  - incurs an acceptable cost

Basic Structure of a Design Pattern

- Name
  - provides a “handle” or means to reference the pattern.
- Purpose
  - provides the problem context and the QoS aspects the pattern seeks to optimize.
  - specifies under which situations the pattern is appropriate and under which situations it should be avoided
- Solution
  - structure and behavior of the pattern.
  - elements of the pattern and their roles in the pattern context.
- Consequences
  - set of pros and cons of the use of the pattern.

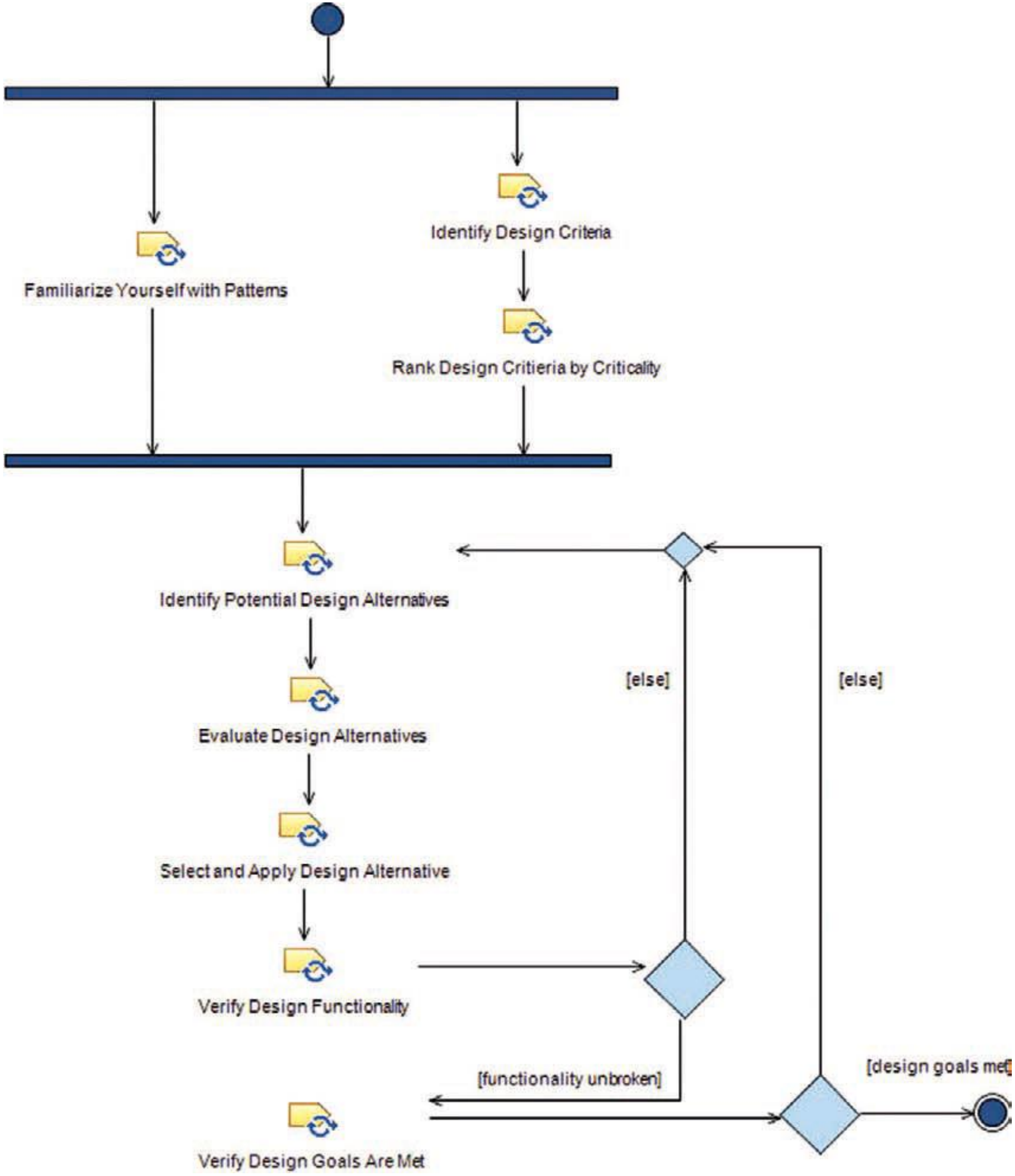
How to Read Design Patterns

- Abstract
  - overview of the problem, solution, and consequences.
- Problem
  - statement of the problem context and the qualities of service addressed by the pattern
- Pattern Structure
  - a structural UML diagram of the pattern showing the important elements of the pattern
  - Relations among elements of the pattern are shown as well.
- Consequences
  - describes the tradeoffs made when the pattern is used

How to Read Design Patterns -2

- Implementation strategies and source code
  - discusses issues around the implementation of the pattern on different computing platforms or in different source level languages.
- Example
  - illustrates how the pattern is applied in some particular case
- Each pattern is shown using both generic, standard UML, and C source code

Using Design Patterns in Development – Pattern Hatching



Identify Design Criteria
Familiarize Yourself with Patterns
Rank Design Critieria by Criticality
Identify Potential Design Alternatives
[else]
[else]
Evaluate Design Alternatives
Select and Apply Design Alternative
Verify Design Functionality
[design goals met]
[functionality unbroken]
Verify Design Goals Are Met

Common Design Optimization Criteria

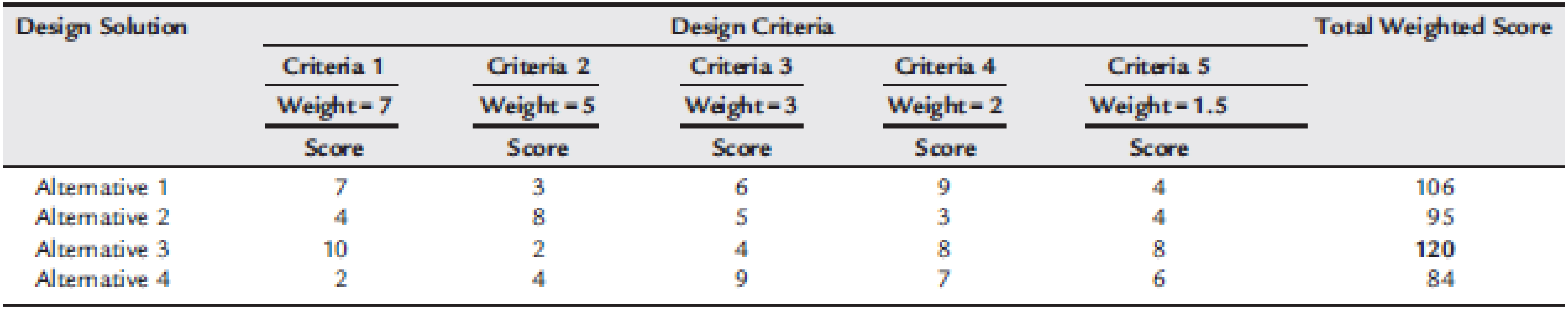
- Performance
  - Worst case
  - Average case
- Predictability
- Schedulability
- Throughput
  - Average
  - Sustained
  - Burst
- Reliability
  - With respect to errors or failures

Common Design Optimization Criteria

- Safety
- Reusability
- Distributability
- Portability
- Maintainability
- Scalability
- Complexity
- Resource usage, e.g., memory
- Energy consumption
- Recurring cost, i.e., hardware
- Development effort and cost

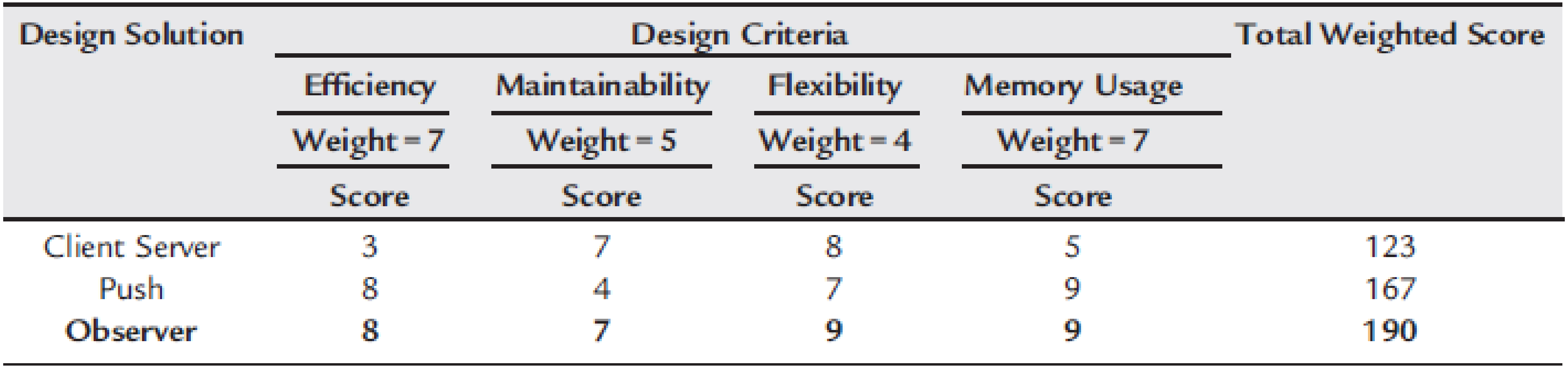
Design Tradeoff Spreadsheet

Table 2-3: Design tradeoff spreadsheet.



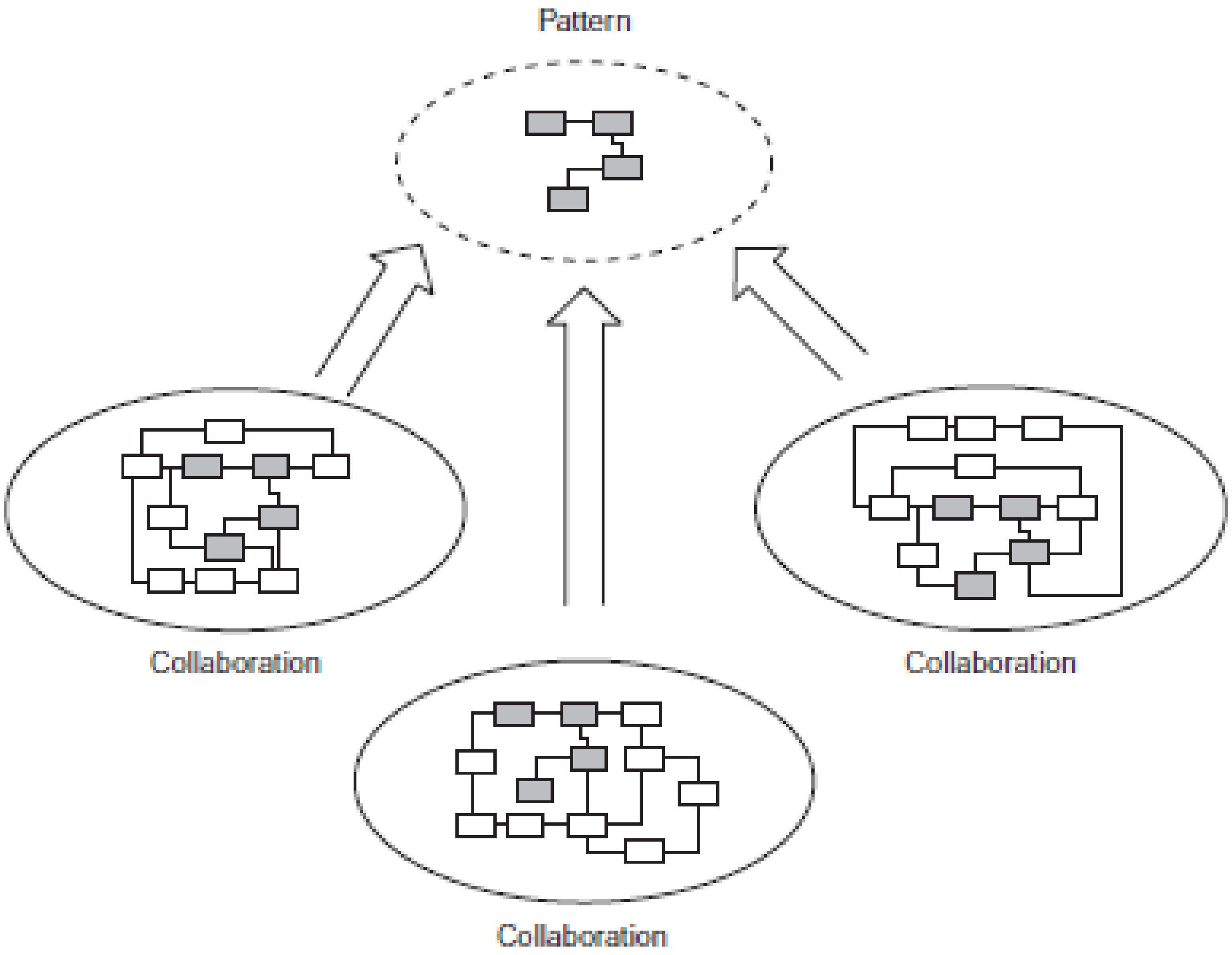
Design Solution	Design Criteria					Total Weighted Score
	Criteria 1	Criteria 2	Criteria 3	Criteria 4	Criteria 5	
	Weight = 7	Weight = 5	Weight = 3	Weight = 2	Weight = 1.5	
	Score	Score	Score	Score	Score	
Alternative 1	7	3	6	9	4	106
Alternative 2	4	8	5	3	4	95
Alternative 3	10	2	4	8	8	**120**
Alternative 4	2	4	9	7	6	84

Table 2-4: Design tradeoffs for ECG monitor system.

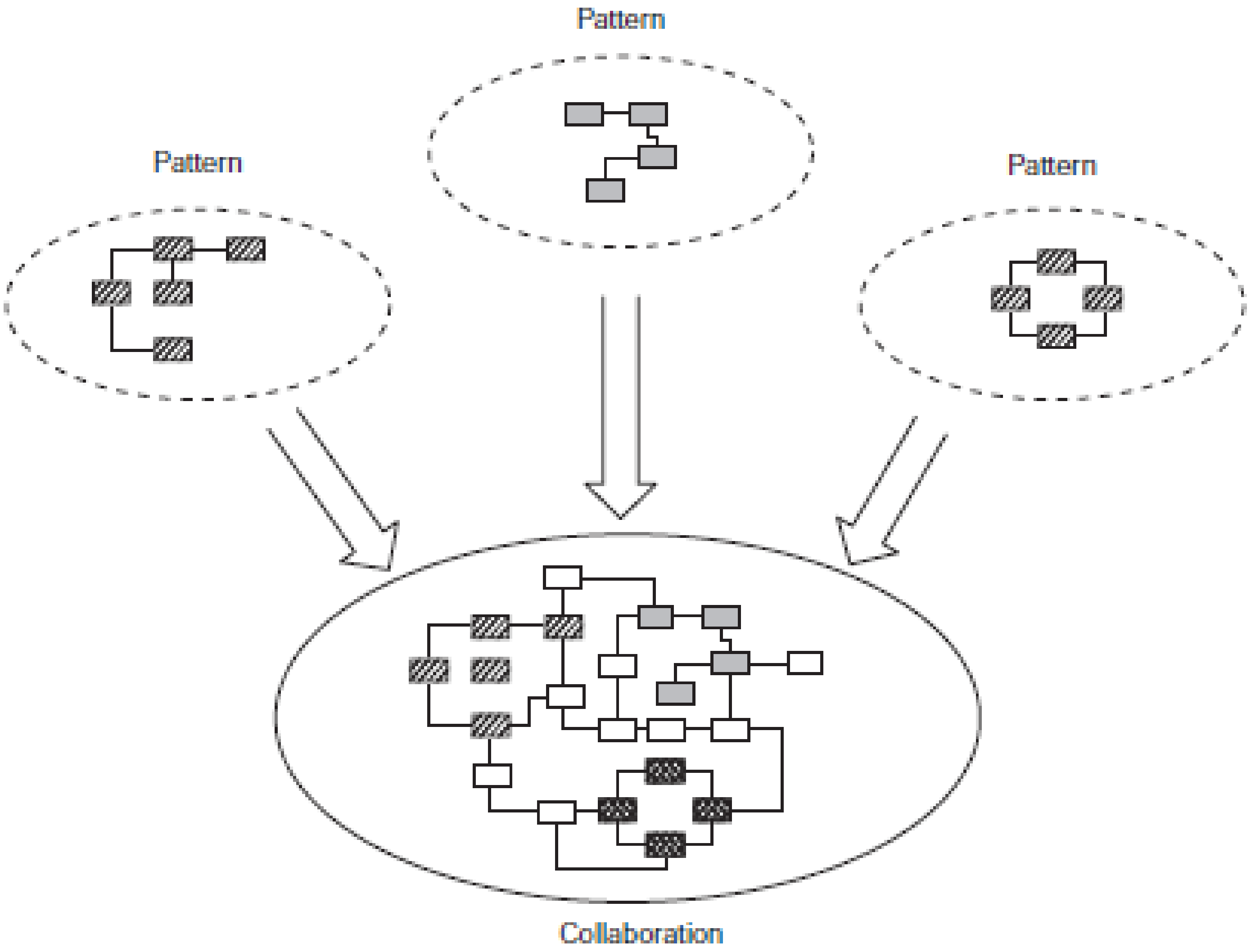


Design Solution	Design Criteria				Total Weighted Score
	Efficiency	Maintainability	Flexibility	Memory Usage	
	Weight = 7	Weight = 5	Weight = 4	Weight = 7	
	Score	Score	Score	Score	
Client Server	3	7	8	5	123
Push	8	4	7	9	167
Observer	**8**	**7**	**9**	**9**	**190**

Pattern Mining



Pattern Instantiation



Observer Design Pattern

- The Observer Pattern is one of the most common patterns around.
- provides a means for objects to “**listen in**” on others while requiring no modifications to the data servers.
- From Embedded Perspective, sensor data can be easily shared to other elements.

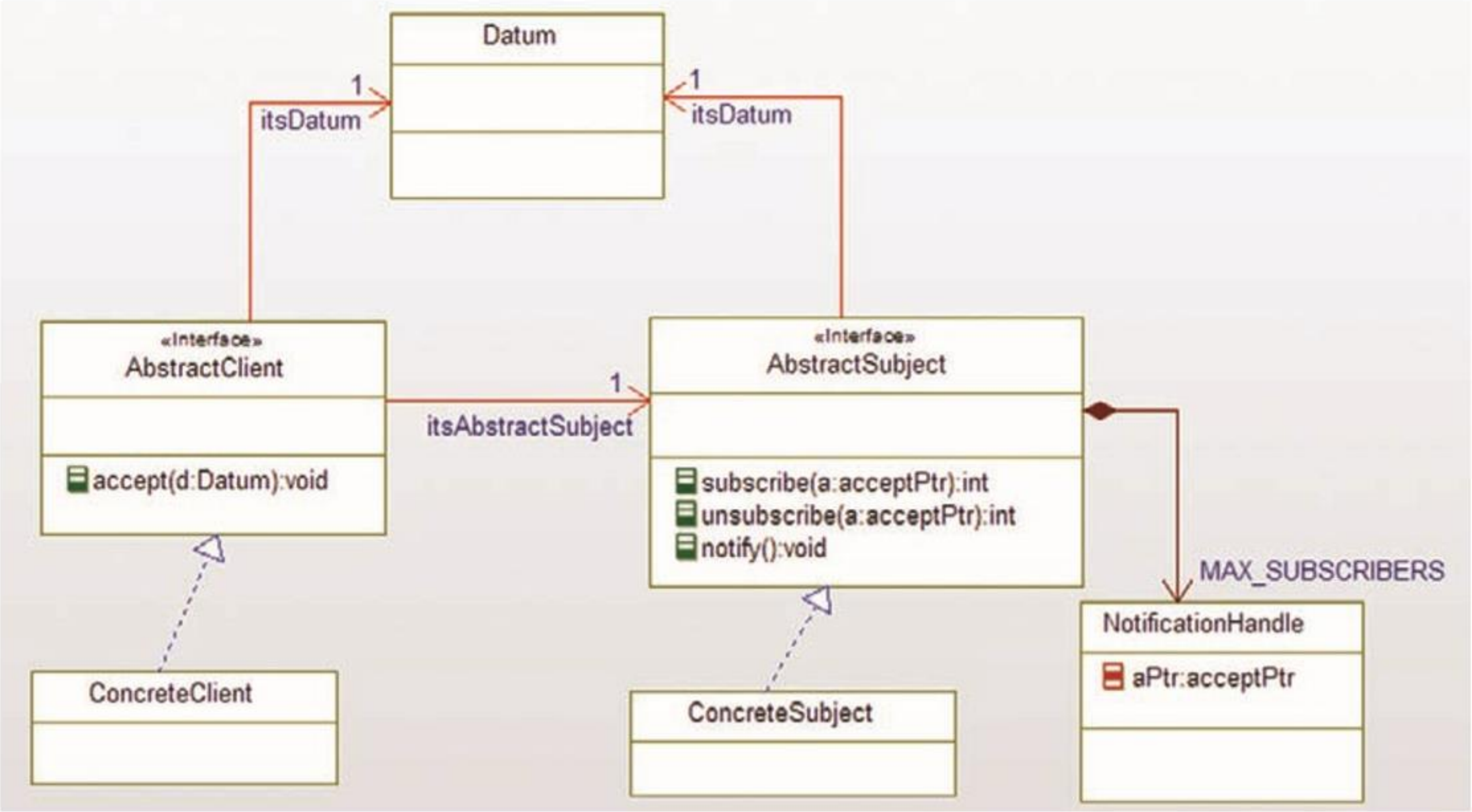
Observer Design Pattern - Abstract

- known as the “Publish-Subscribe Pattern”
- Provides notification to a set of interested clients that relevant data have changed.
- It does this without requiring the data server to have any a priori knowledge about its clients.
- Clients(Sensors) can use Subscribe function to add themselves to the notification list.
- The data server can then enforce whatever notification policy it desires.

Observer Design Pattern - Problem

- Each client can request data periodically from a data server in case the data have changed.
- If the data server pushes the data out, then it must know who all of its clients are.
- subscription and un-subscription services to data servers are allowed to clients.
- The pattern allows dynamic modification of subscriber lists.
- The server can enforce the appropriate update policy to the notification of its interested clients.

Observer Design Pattern - Structure



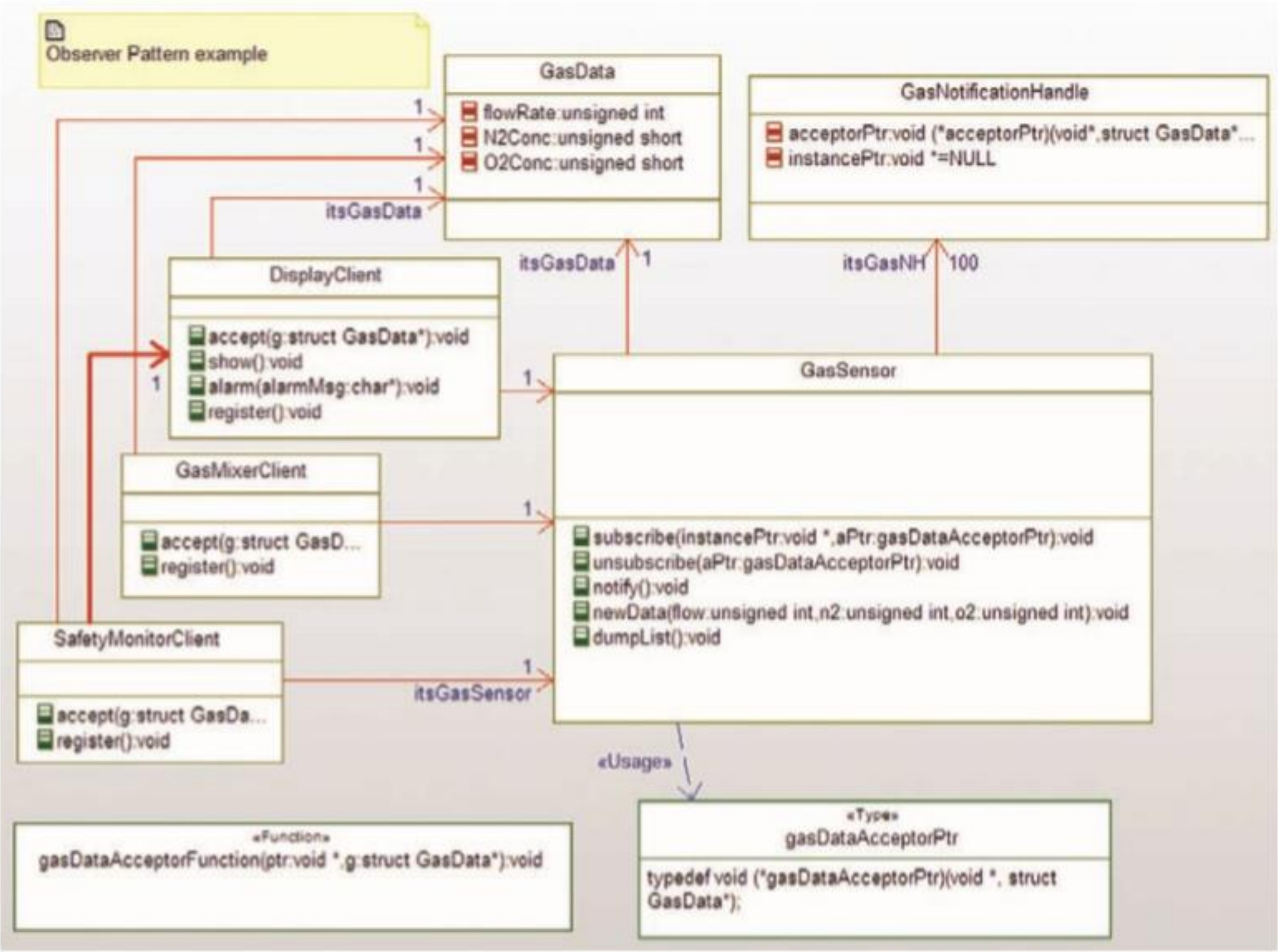
Observer Design Pattern - Consequences

- Simplifies the process of distributing data to a set of clients.
- Maintains the fundamental client-server relation while providing run-time flexibility of adding clients.
- Compute efficiency is maintained since clients can only know updates when data is changed.

Observer Design Pattern – Implementation Complexities

- The most complex aspects of this pattern are the implementation of the notification handle and the management of the notification handle list.
- The easiest approach for the notification list is to declare **an array** big enough to hold all potential clients. This wastes memory in highly dynamic systems with many potential clients.
- Another approach is to construct **a linked list** of all clients.

Observer Design Pattern - Example



References

- **Chapters 2 and 3**: Douglass, Bruce Powel. **Design patterns for embedded systems in C: an embedded software engineering toolkit**. Elsevier, 2010.