

Safety Design for Embedded Systems Development

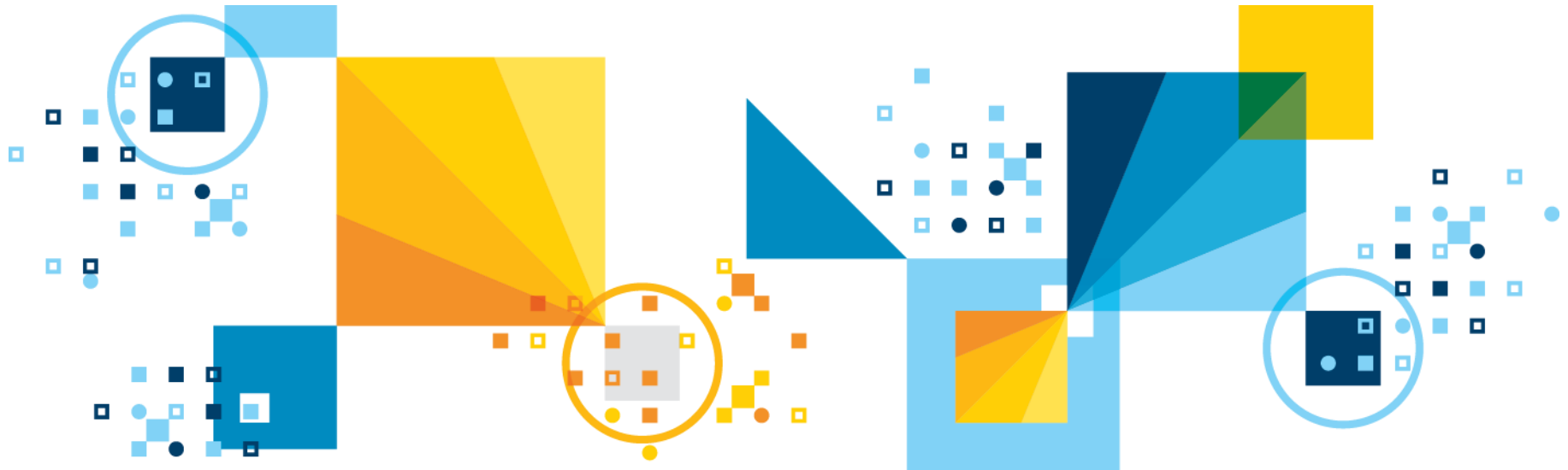
A Design-Pattern Approach

Bruce Powel Douglass, Ph.D.

Chief Evangelist, IBM IoT

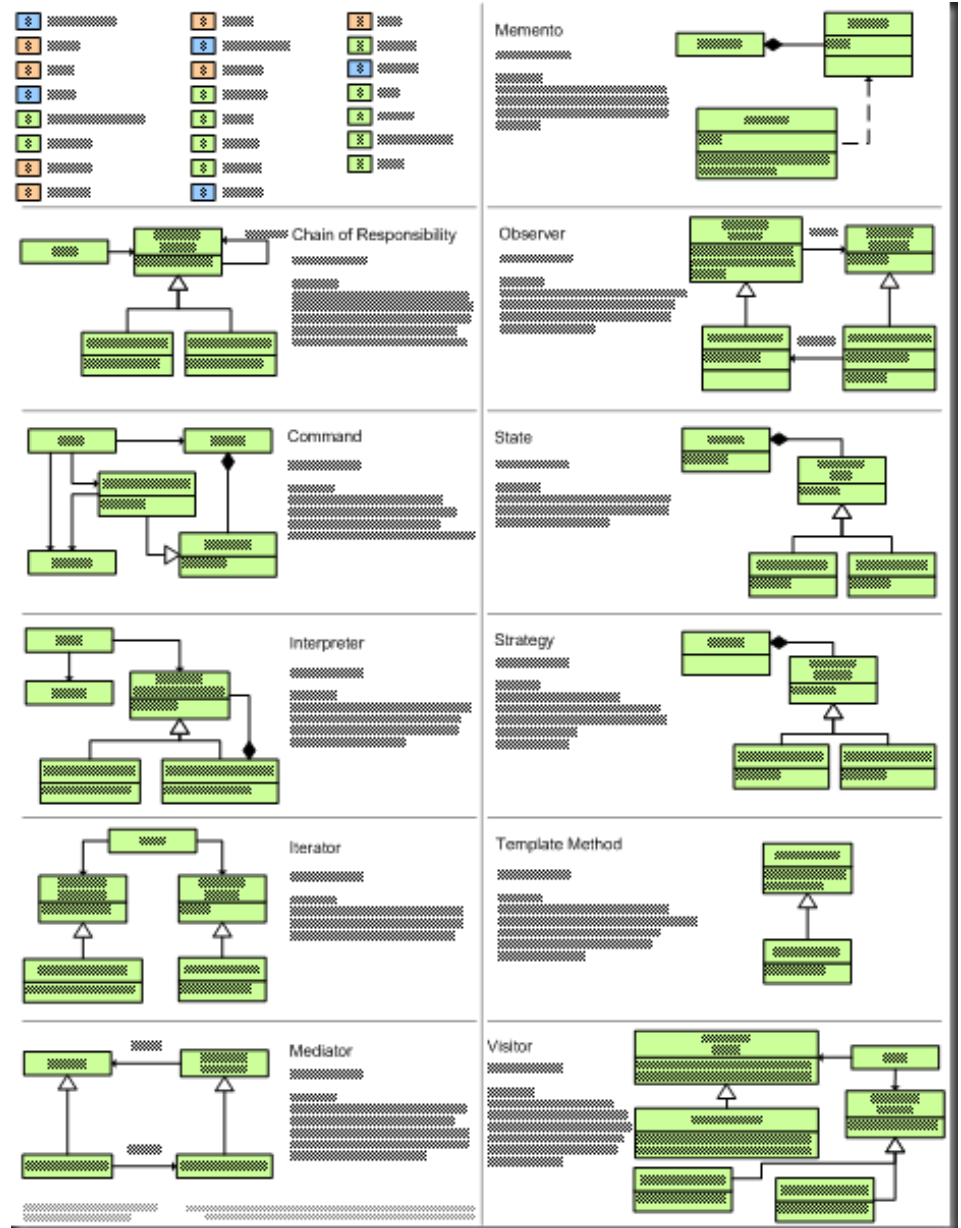
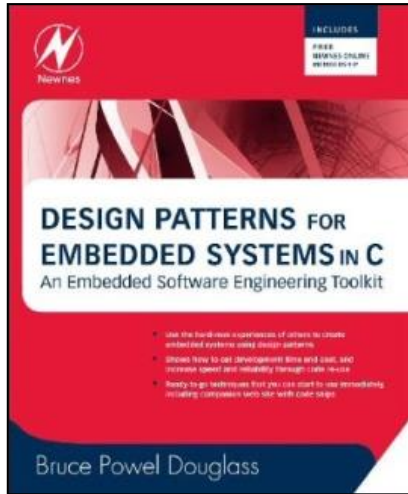
www.bruce-douglass.com

Twitter: @IronmanBruce



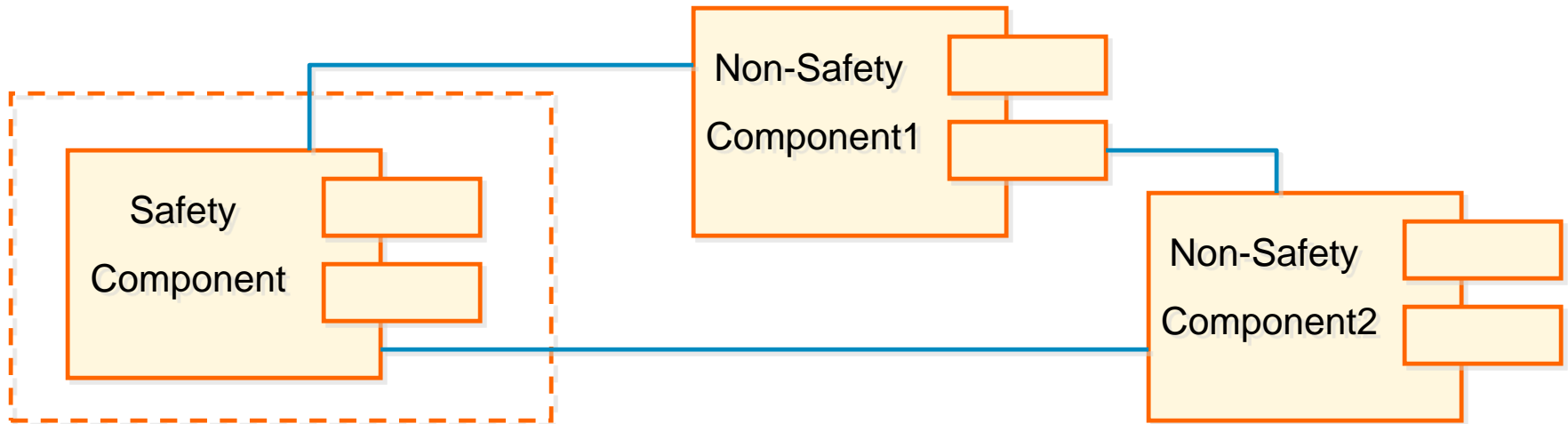
Safety and Reliability Patterns

- Isolation Pattern
- CRC Pattern
- Smart Data Pattern
- Protected Single Channel Pattern
- Homogeneous Redundancy Pattern
- Heterogeneous Redundancy Pattern
- Triple Modular Redundancy Pattern
- Monitor-Actuator Pattern



Isolation Pattern

- Isolate safety functions from non-safety critical functions
 - Separate CPU
 - Separate memory
 - Separate devices
 - Separate process (e.g. ARINC 653-style OS support)
- Safety-relevant systems are 300-1000% more effort to produce
 - Isolation of safety systems allows more expedient development
- Care must be taken that the safety system is truly isolated so that a defect in the non-safety system cannot affect the safety system

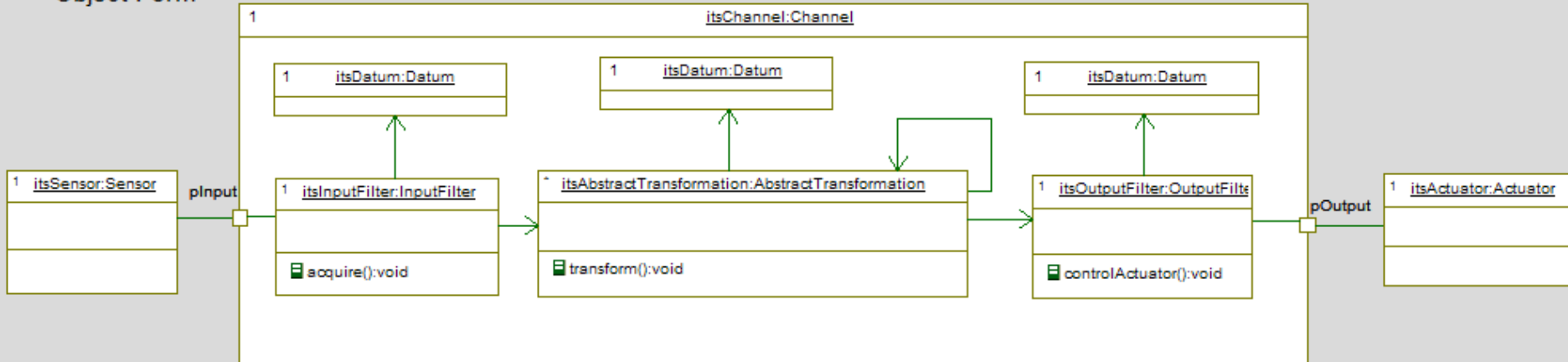


Channel Pattern

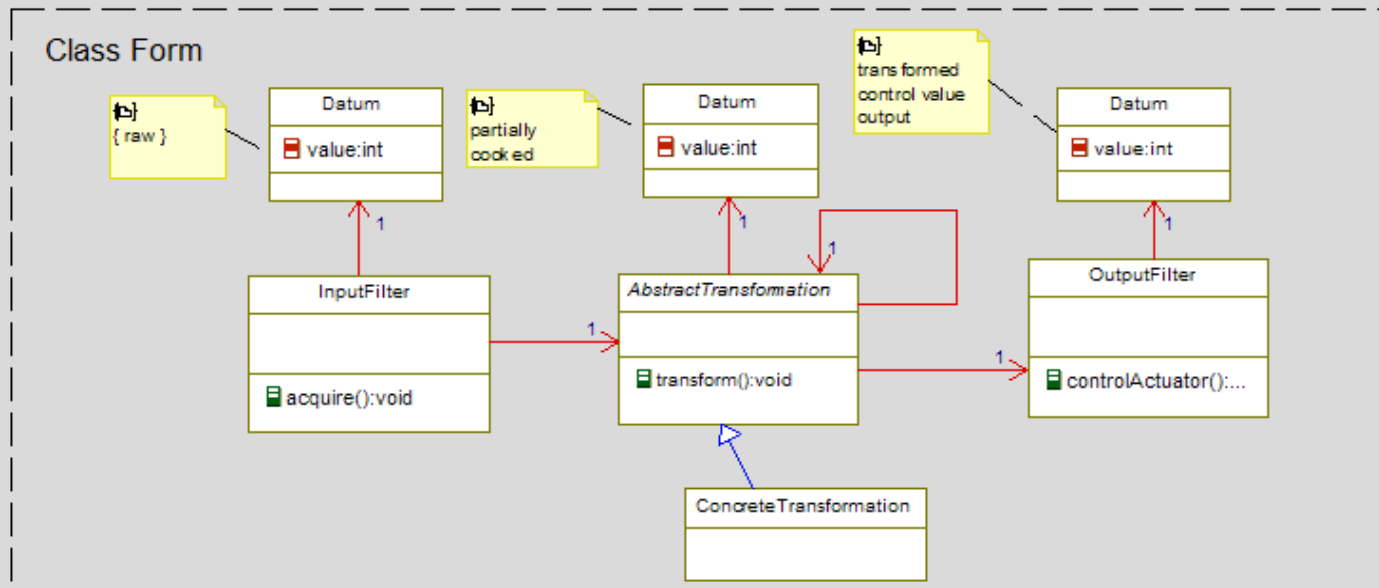
- Problem
 - Efficient execution of a system in which data is successively transformed in a series of steps
 - Want to organize and manage a hi-reliability, hi-availability, or safety-critical system that must provide redundancy of end-to-end behaviors
- Solution
 - Construct the system as a channel, a large scale subsystem which handles data from acquisition all the way through dependent actuation. Provide as many independent channels as necessary.
- Consequences
 - A simple organizational pattern that permits redundancy to be easily added.
 - May use additional memory since channels are designed to be independent, requiring replication (redundancy)

Channel Pattern

Object Form

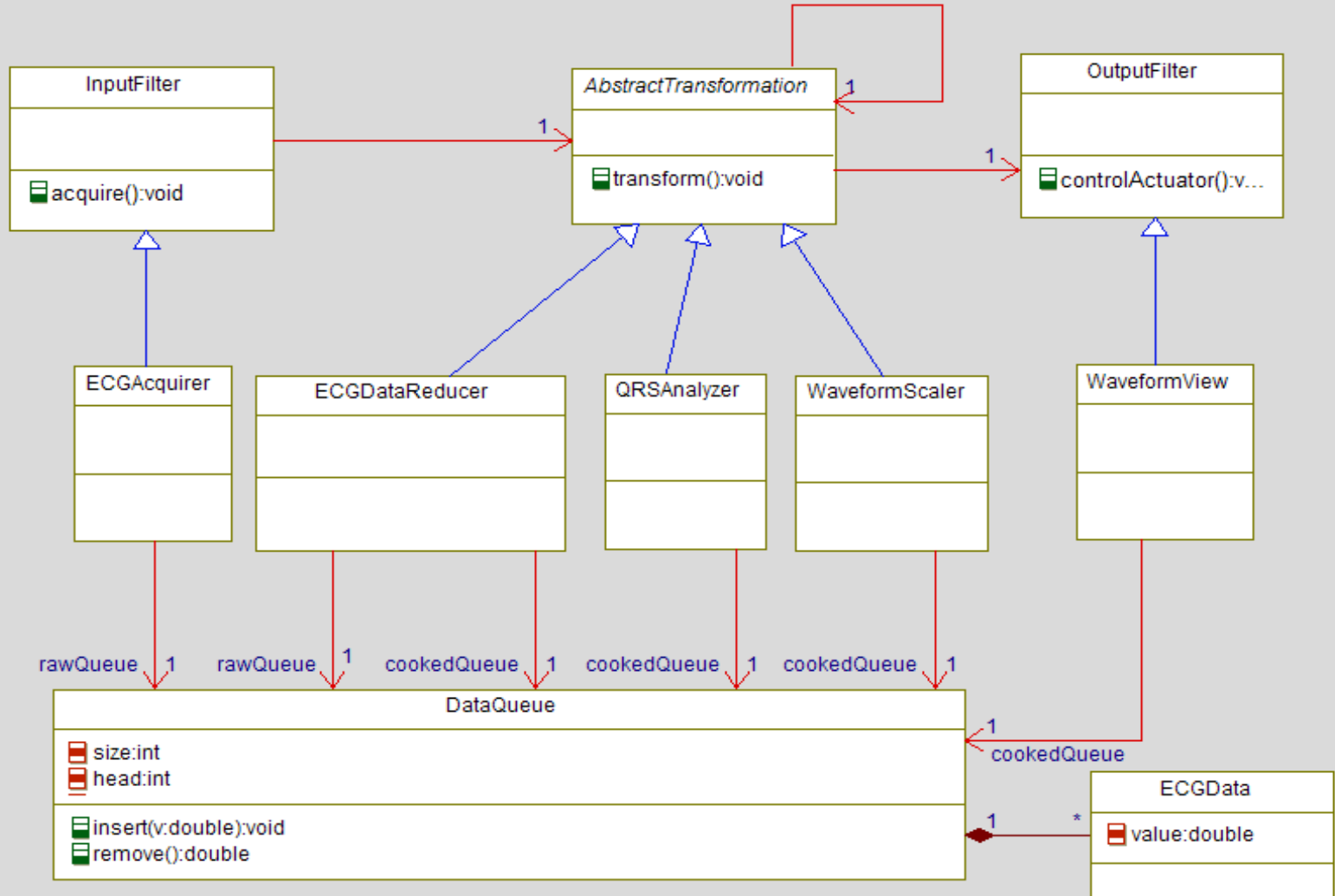


Class Form



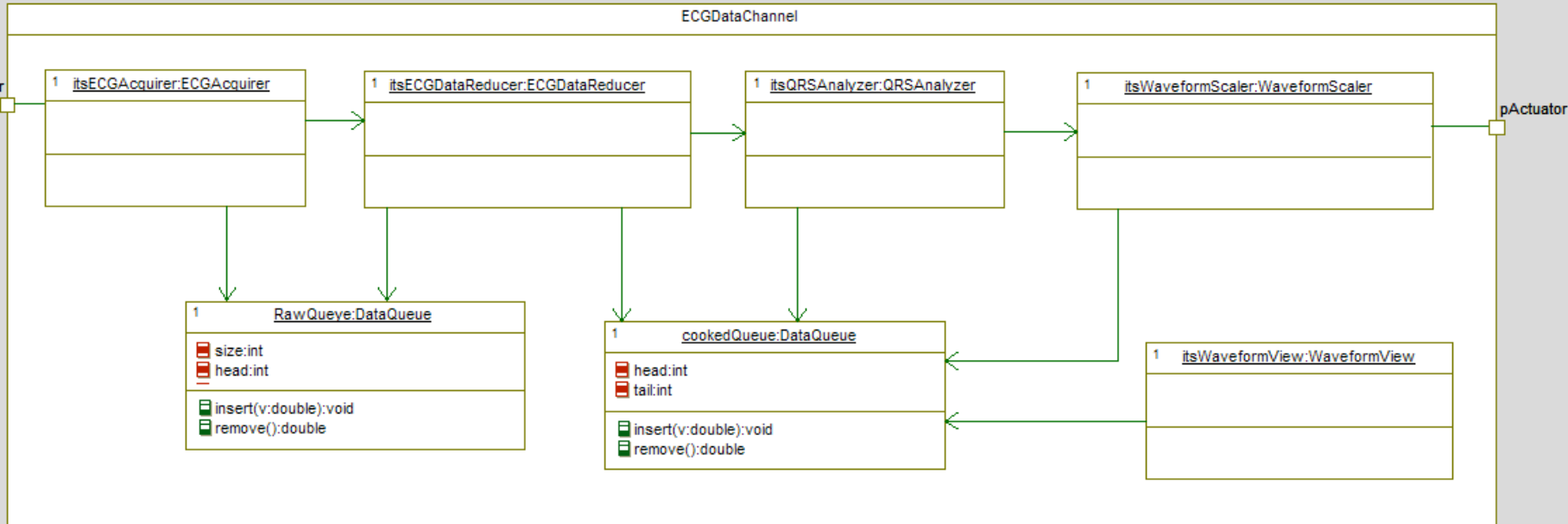
Channel Pattern Example

Class Form

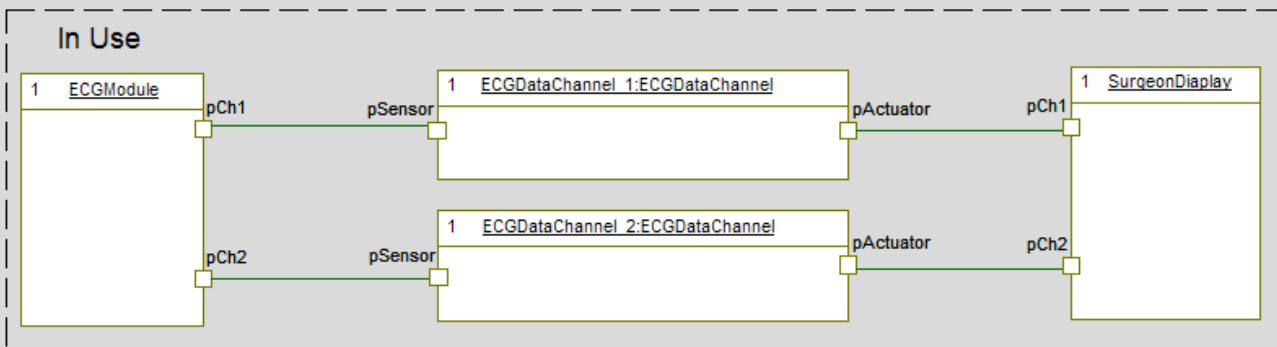


Channel Pattern Example

Object Form



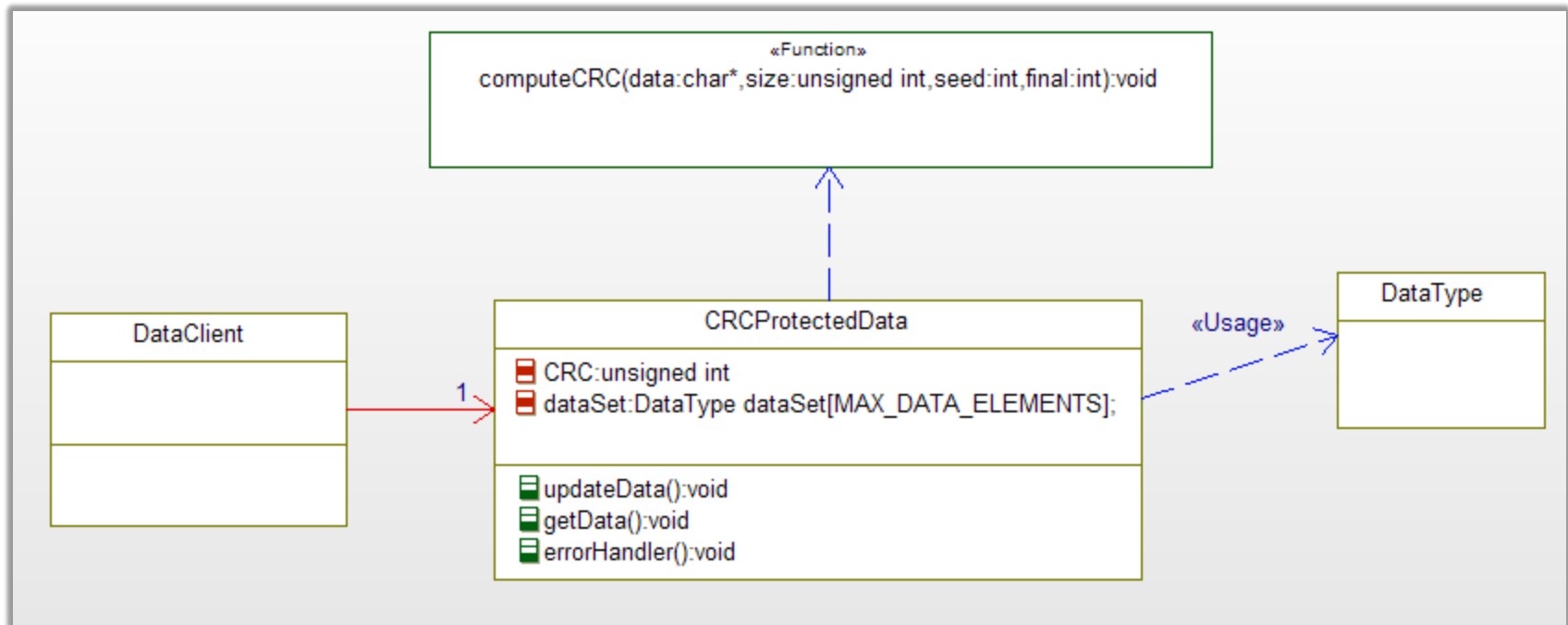
In Use



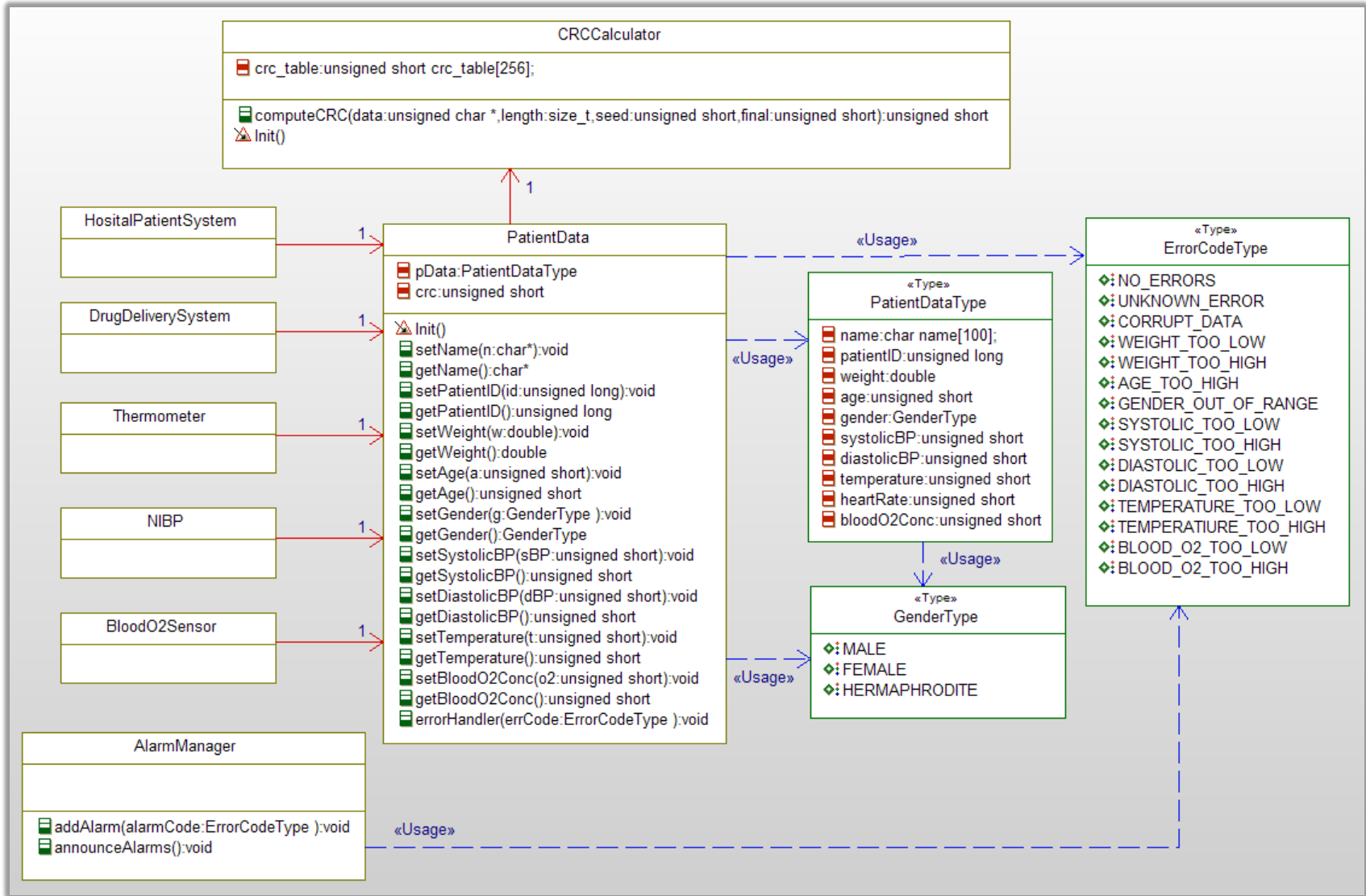
CRC Pattern

- Problem
 - This pattern addresses the problem that variables may be corrupted from a variety of causes such as environmental factors (such as EMI, heat, and radiation), hardware faults (such as power fluctuation, memory cell faults, and address line shorts), or software faults (other software erroneously modifying memory). This pattern addresses the problem of data corruption in large data sets.
- Solution
 - The pattern adds cyclic redundancy checks to identify data corruption and trigger appropriate action when it occurs
- Consequences
 - CRC uses a small amount of memory for strong bit-corruption identification
 - Table-driven implementations use additional block of memory to hold table but are computationally efficient

CRC Pattern



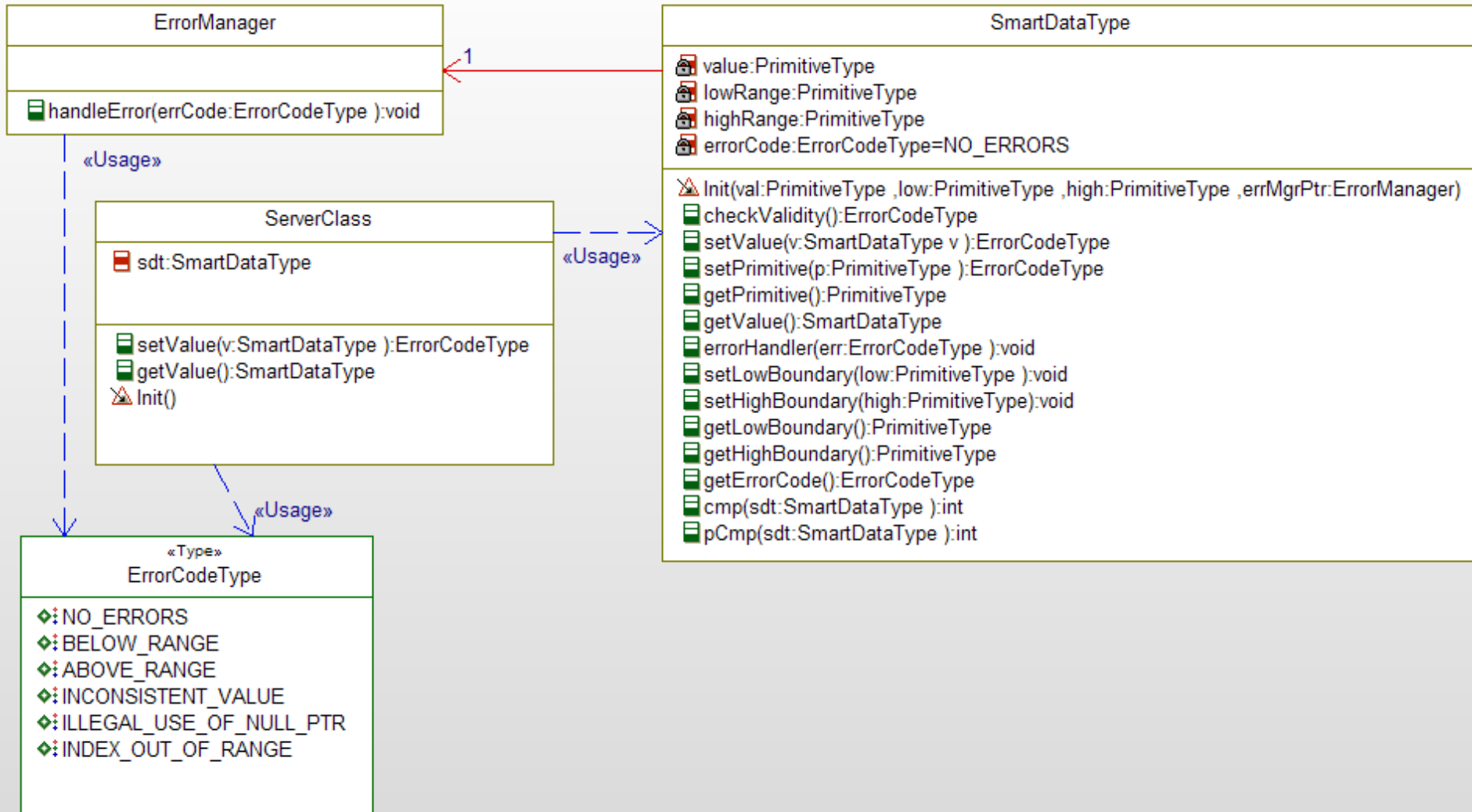
CRC Pattern Example



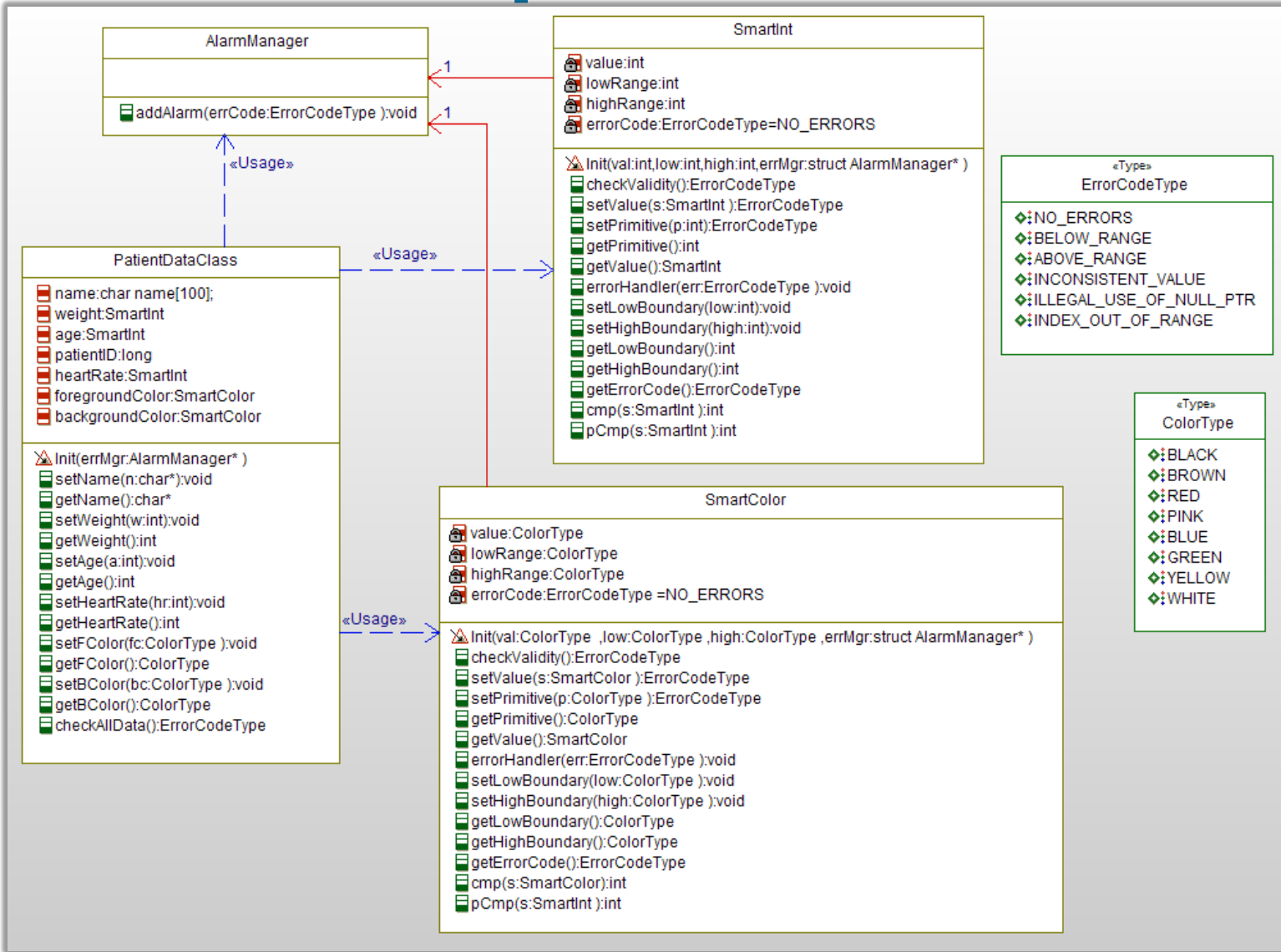
Smart Data Pattern

- Problem
 - The problem this pattern addresses is to build functions and data types that essentially check themselves and provide error detection means that cannot be easily ignored.
- Solution
 - The key concepts of the pattern are to
 - Build self-checking types whenever possible
 - Check incoming parameter values for appropriate range checking
 - Check consistency and reasonableness among one or a set of parameters.
- Consequences
 - The downside for using smart data types is the performance overhead for executing the operations.
 - The upside is that the data is self-protecting and provides automatic checking when the data is set.
 - It is also possible for the programmers to avoid using the functions and access the values directly if they are so inclined, defeating the purpose of the smart data type.

Smart Data Pattern



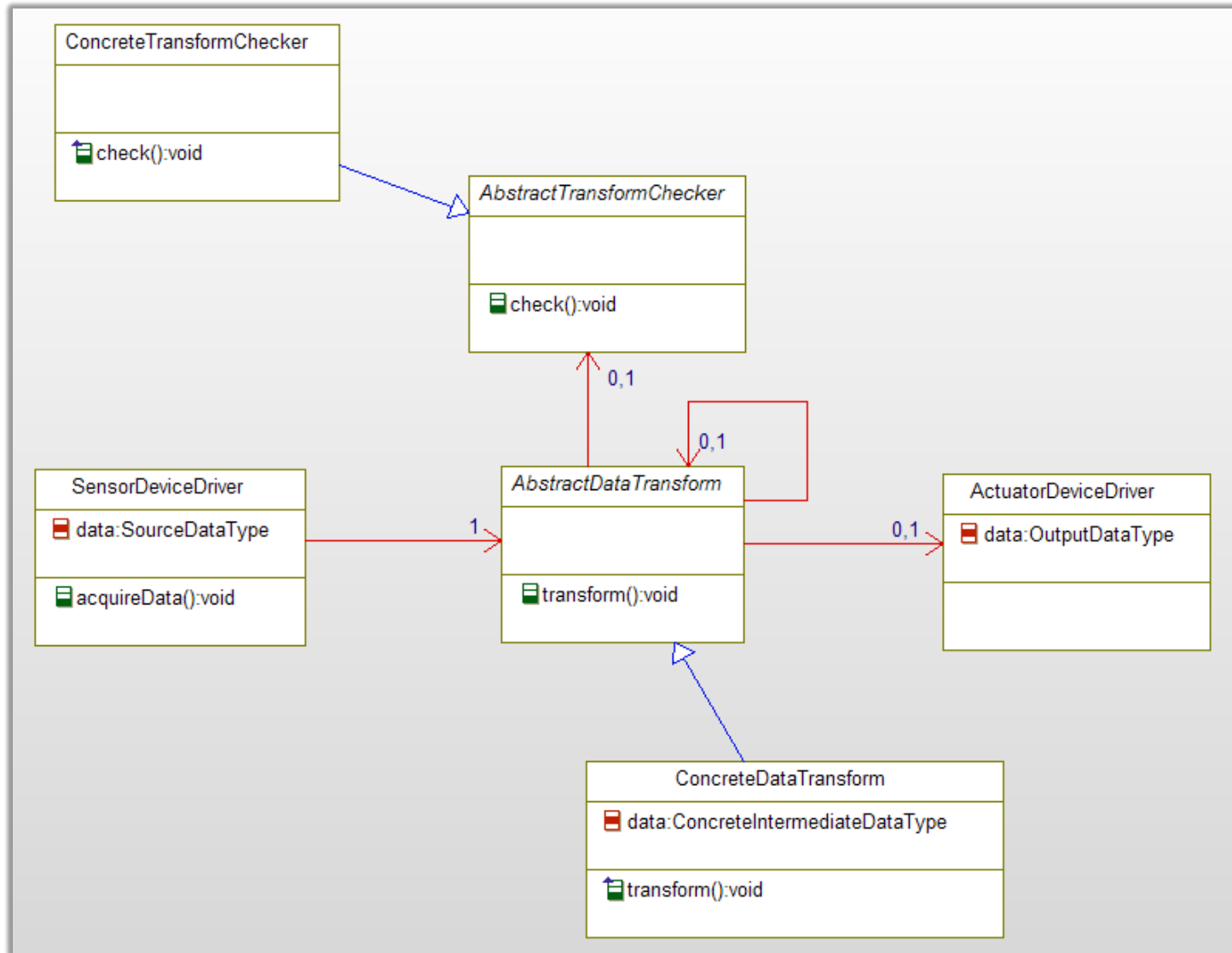
Smart Data Pattern Example



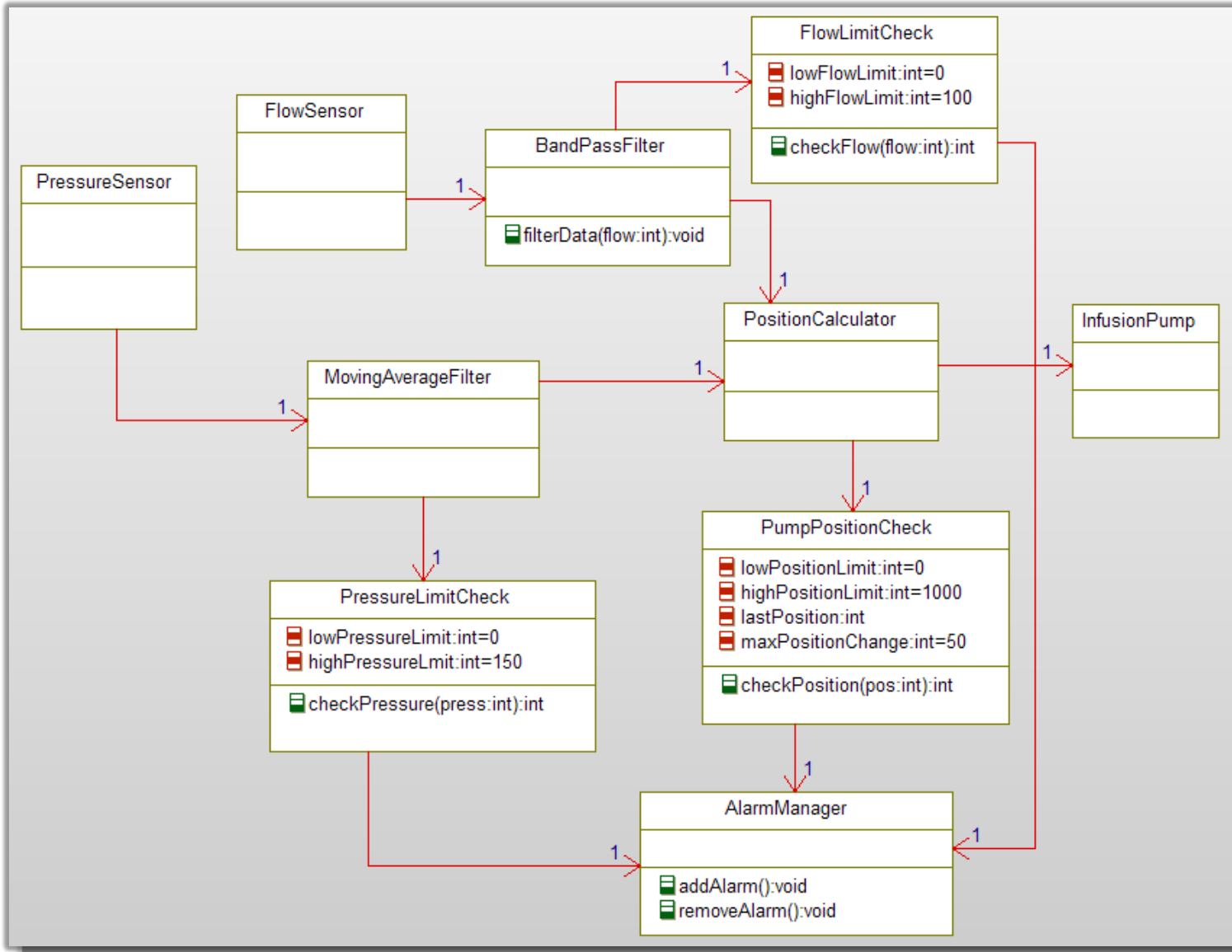
Protected Single Channel Pattern

- Problem
 - Provide protection against errors (design flaws) in a cost effective way
- Solution
 - A variant of the Channel pattern the uses light-weight redundancy to provide identification of errors
- Consequences
 - Low design cost
 - Low recurring cost
 - Not able to continue in the presence of faults

Protected Single Channel Pattern



Protected Single Channel Pattern Example

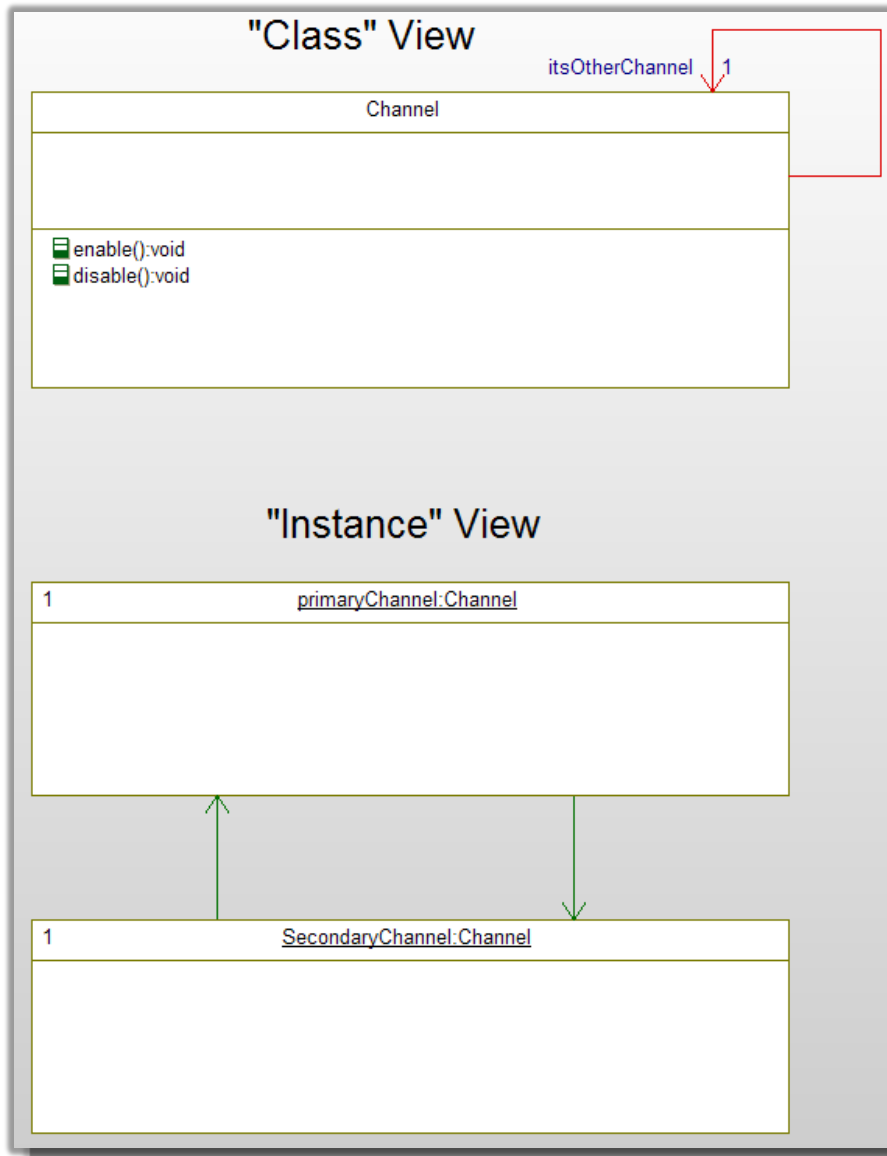


Multichannel Redundancy Pattern

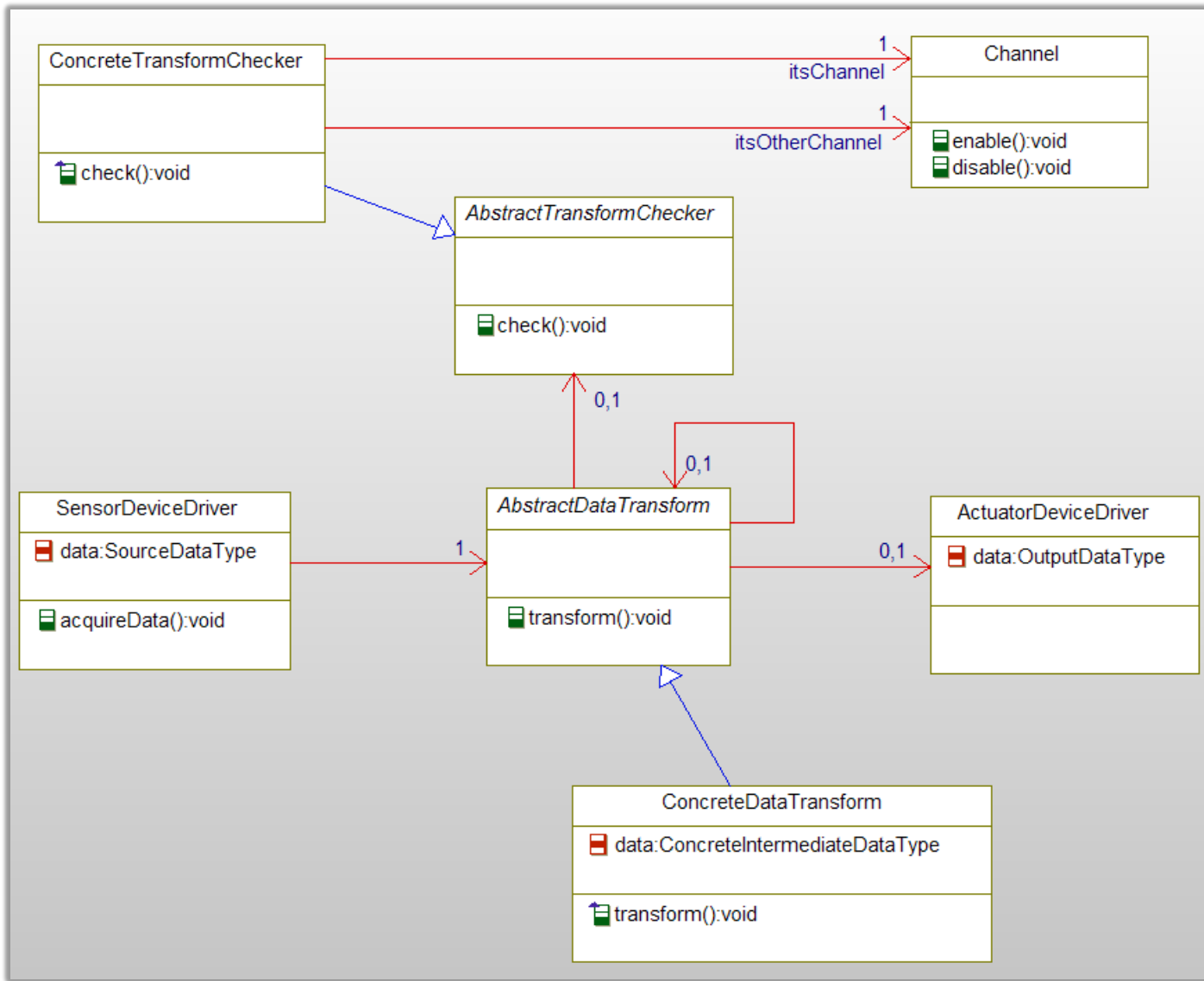
- Problem
 - Provide the ability to continue in the presence of a fault
- Solution
 - Provide “redundancy in the large” by replicating channels
 - Common variants
 - Dual Channel Homogeneous Redundancy (DCHo)
 - Dual Channel Heterogeneous Redundancy (DCHe)
 - Triple Modular Redundancy (TMR)
- Consequences
 - Low design-time cost (DCHo, TMR)
 - High design-time cost (DCHe)
 - High recurring cost (all)
 - Able to continue in the presence of a failure (all)
 - Able to continue in the presence of an error (DCHe, or TMR-He)

Note: Homogeneous channels are exact replicas; Heterogeneous channels use different designs, algorithms, hardware types, and/or code.

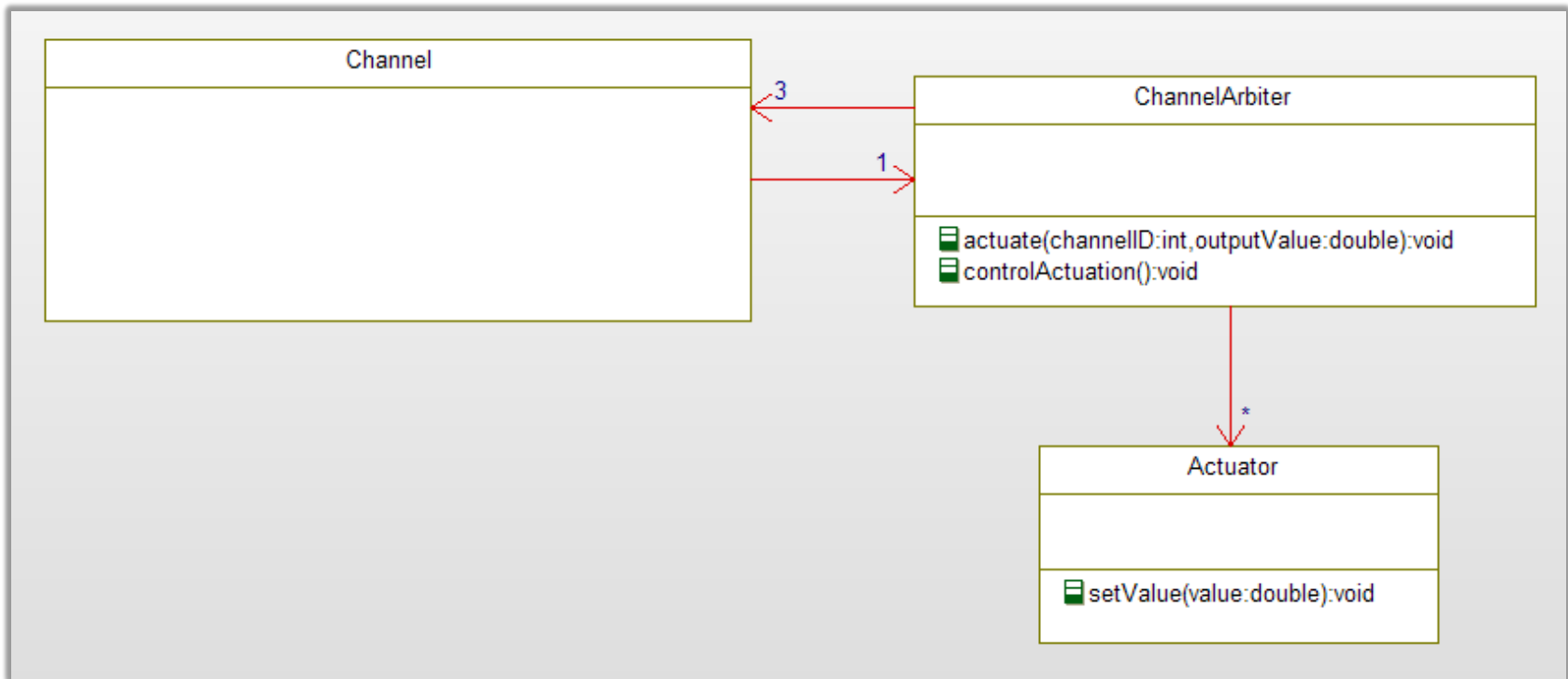
Homogeneous Redundancy Pattern (high level view)



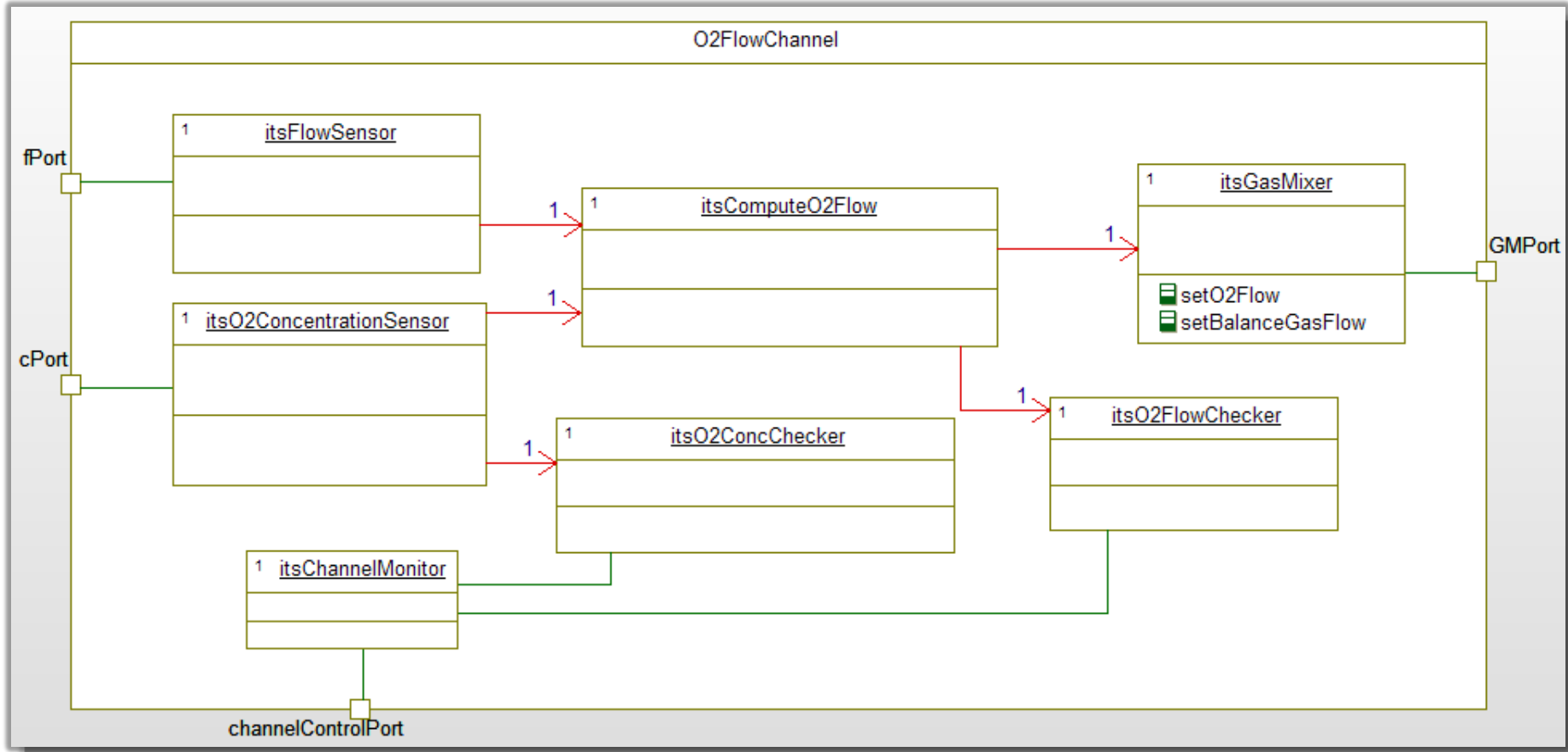
Homogeneous Redundancy Pattern (detailed view)



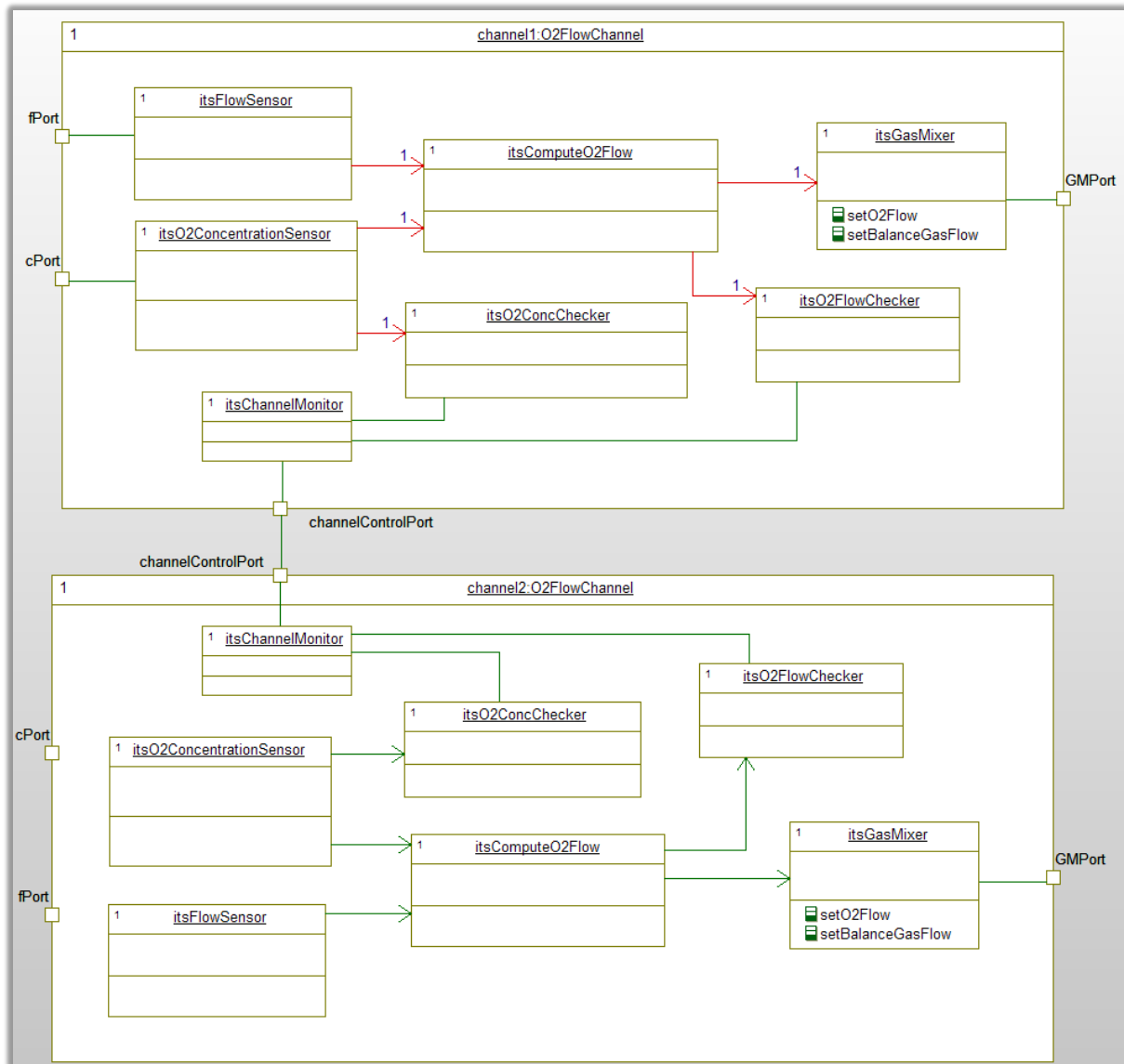
Triple Modular Redundancy (TMR) Pattern



Homogeneous Redundancy Pattern Example

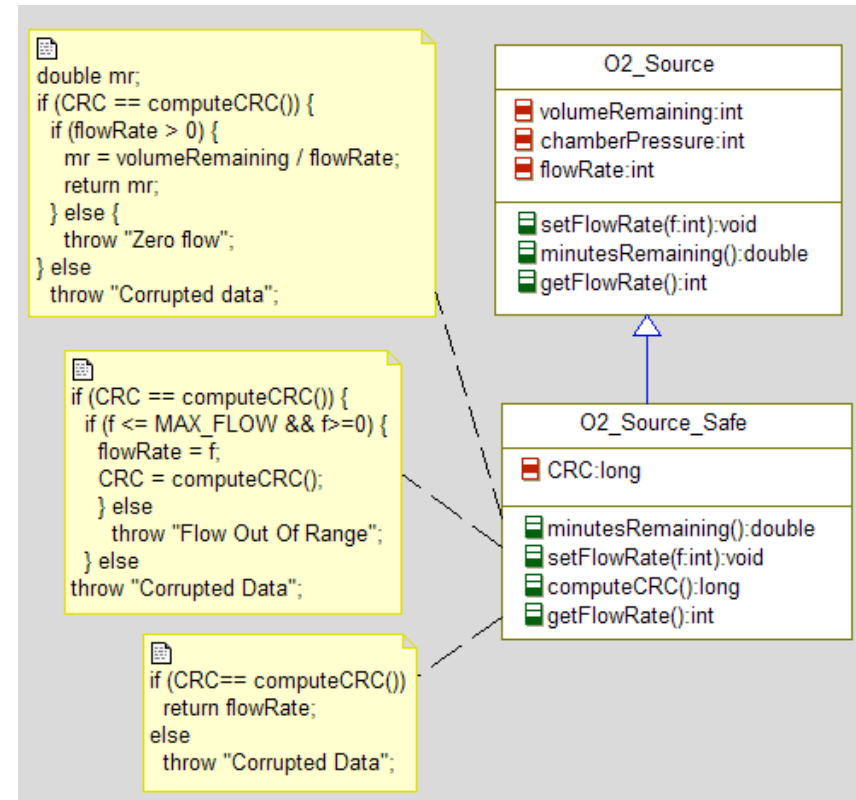
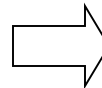
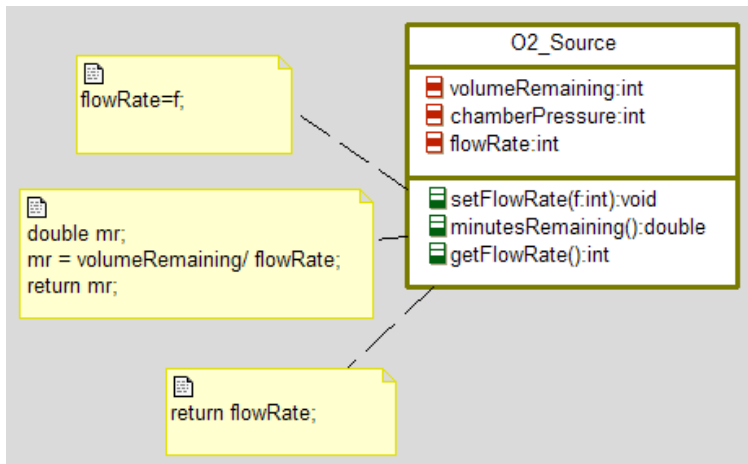


Homogeneous Redundancy Pattern Example



Redundancy “in the small” with Defensive Design

- Redundancy “in the small” adds low-level checking on
 - Data value range
 - Data value consistency
 - Computational accuracy
 - Explicit pre- and post-condition checks




Download Papers, Presentations, Models, & Profiles for Free

Harmony aMBSE Deskbook Version 1.00
Agile Model-Based Systems Engineering Best Practices with IBM Rhapsody

Bruce Powel Douglass, Ph.D.
 Chief Evangelist
 Global Technology Ambassador
 IBM Internet of Things

bruce.douglass@us.ibm.com

**Black Edition:
 Rhapsody Only**



© Copyright IBM Corporation 2017. All Rights Reserved Harmony aMBSE Deskbook 1



www.bruce-douglass.com

