Written Explanation of Design Decision

Prepared by: Andreas Limberopoulos and Ananya Behera

Our design makes good use of abstractions to ensure that the Open Closed Principle (OCP) is satisfied. We created several abstract interfaces for objects that we feel would facilitate extensions of the system due to a change in requirements. Any addition to the system, be it a new Monitor, a new Server or even an entirely different type of Match can be easily integrated into our design by extending already existing classes without the need to modify them. Furthermore, most clients depend on abstract interfaces rather than concrete implementations; this decouples the client from the concrete implementation. Any changes that conform to the contract of the abstract interface can be made to concrete implementations without causing dependency issues in the client. This is the essence of the Dependency Inversion Principle (DIP); high-level policy should not depend on low-level modules they should both depend on abstractions. We used a combination of design principles outlined in Fowler [Mar2000] and design patterns outlined in Gamma et al. [GHJ1995] to make our implementation of the CWC Monitoring System as extensible and reusable as possible. In doing so we used the following design patterns:

Abstract Factory
Principles used: Dependency Inversion Principle (DIP), Common Closure Principle (CCP), Common Release Principle (CRP) and Stable Abstractions Principle (SAP)

The introduction of a layer of abstraction between CricketFactory and Controller, by way of the DIP, creates a hinge point in the design. Changes to CricketFactory will not cause changes to the Controller because the dependency has been inverted and the Controller now depends on an abstract class, MatchFactory. Similarly we create abstract classes called Match and MatchEvent which invert the dependencies between Controller and CricketMatch and LastBall.

Packages within the Abstract Factory that are likely to change together are MatchFactory, Match and Event. Similarly CricketFactory, CricketMatch and LastBall should be grouped together (CCP). These packages also form a sensible unit of reuse (CRP).

The SAP says that a stable package should be an abstract package. This is another design principle used in Abstract Factory. All clients of CricketFactory, CricketMatch and LastBall depend on their abstract superclasses. These abstract classes are grouped together in an entirely abstract package.

Abstract Server
Principles used: DIP.

Instead of having the Controller directly communicate with the SOAP server we created an abstract Server class that, by way of the DIP, inverts the dependency. The controller now depends on an abstract interface. We can now substitute any server we want for our SOAP server, so long as it respects the contract outlined in the abstract Server class.

Observer
Principles used: Liskov Substitution Principle (LSP), Open Closed Principle (OCP), CCP, CRP and SAP

The Observer Pattern allows a subject to maintain a list of abstract observers without needing to know their concrete implementations because of the LSP the children of the abstract Observer class are substitutable anywhere that the parent is expected. This decouples the subject from its concrete observers. The subject doesn't depend on any concrete classes. A class simply needs to honour the contract of the Observer class and it can then observe a subject and extend the functionality of the system in any way (OCP).

Similar to the Abstract Factory, the Obserer and Subject classes form their own entirely abstract package. The CRP, CPP, SAP also apply to the Observer package.

Adapter
Principles used: DIP

We had to use an adapter to introduce the new server in stage two. The format of the data and the methods used to retrieve the data for server one and server two were not the same. Server one provided data in a way that conformed to the abstract Server classes contract. Clients depend on the abstract Server class (DIP). We used the adapter pattern to transform the data provided by server two into a form that honoured the contract of the Server class.

References:
[Mar2000] Robert C. Martin "Design Principles and Design Patterns", 2000. Available online from Object Mentor:
http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[GHJ1995] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software" Addison-Wesley, 1995 (Chs. 1, 3, 4, 5).