Observer / Template Methods

Comp-304 : Observer / Template Methods
Lecture 26

Alexandre Denault
Original notes by Hans Vangheluwe
Computer Science
McGill University
Fall 2007

Olympics

- The I.T. Systems for the Olympic represent quite an architecture challenge.
- Information about the events, such as the detailed scheduling, competitors and results are all stored on a centralized system.
- This information must then be distributed to various subsystem, each used by different category of people.

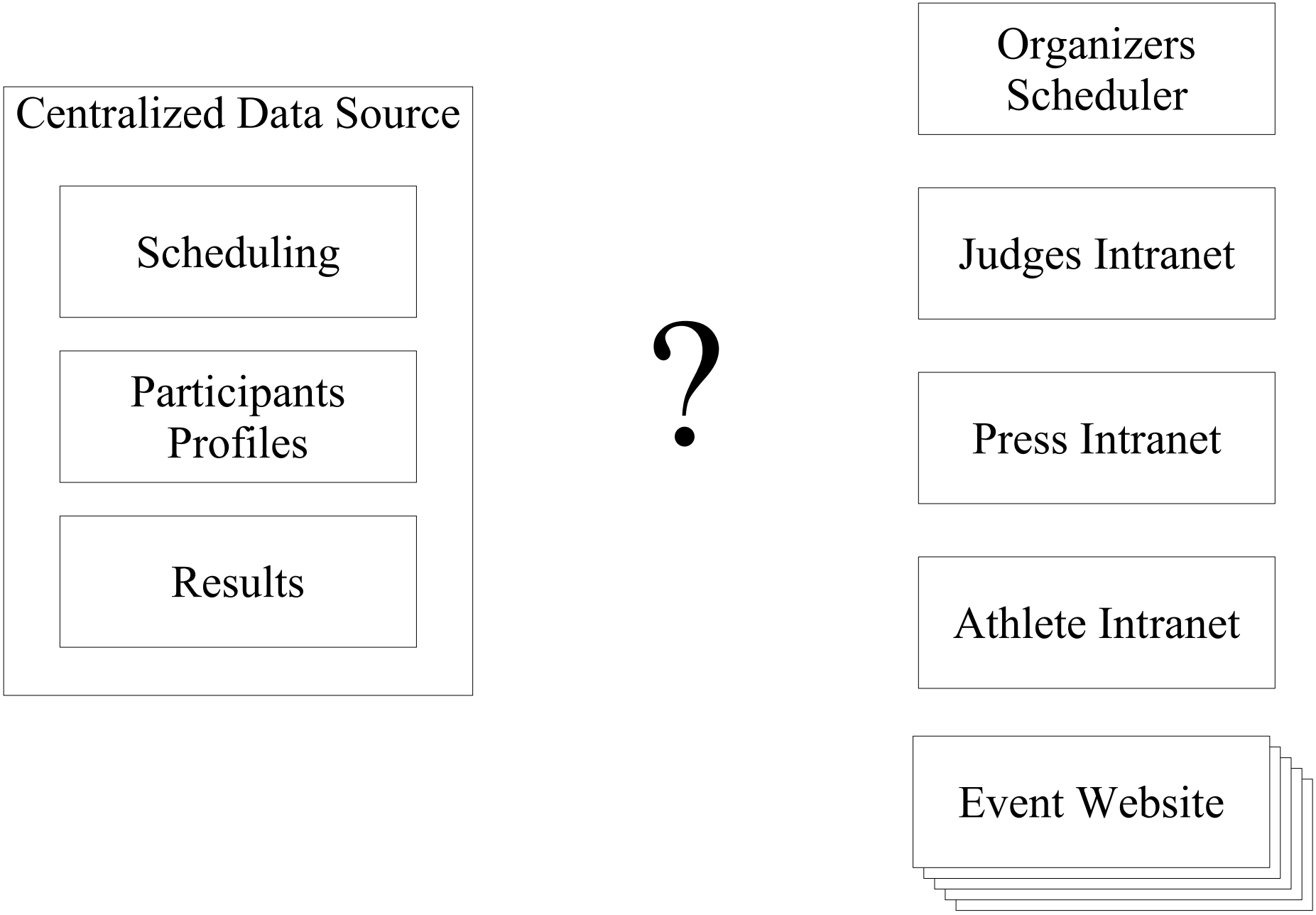
Information Dsitribution

	Scheduling	Participants Profiles	Results	
Organizers				
Judges				
Athletes				
Spectators				
Press				

Information Dsitribution

	Scheduling	Participants Profiles	Results	
Organizers	W			
Judges	R		W	
Athletes	R	W	R	
Spectators		R	R	
Press	R	R	R	

Data Source and Subsystems



Concerns

- Such a system must be very efficient:
  - It will have to deal with a tremendous amount of load.
- As such, the subsystems cannot continuously poll the data source for content.
- However, the data source cannot push all the content upon the subsystems.

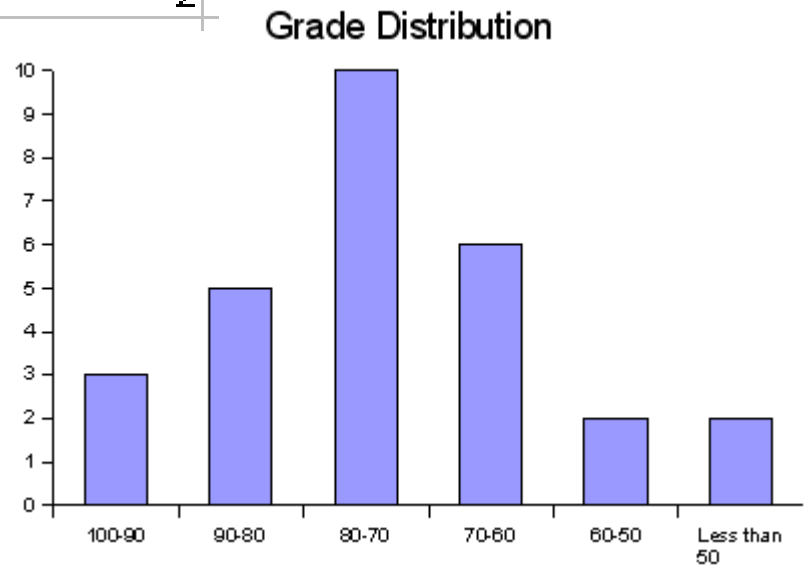
Observer Pattern

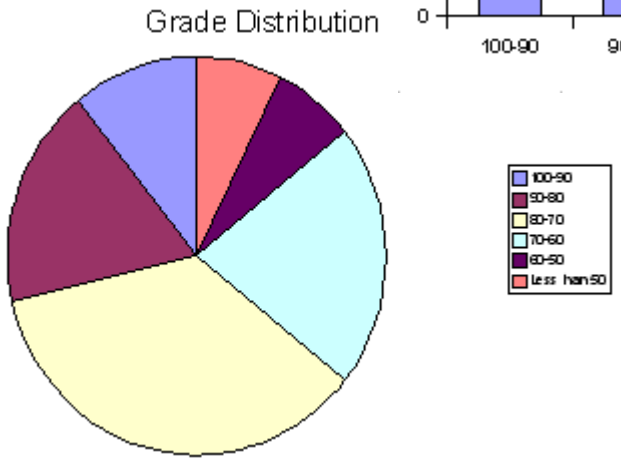
- The Observer pattern defines an one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- A.K.A. Dependents, Publisher Subscriber

Classical Example

Student	Grade
John	85.00%
Bob	95.00%
Jane	72.00%
Mary	63.00%
Cory	56.00%
Adam	90.00%
Nancy	21.00%
Stacy	55.00%
James	73.00%
William	75.00%
David	66.00%
Richard	68.00%
Patricia	78.00%
Linda	84.00%
Barbara	69.00%
Paul	83.00%
Elizabeth	75.00%

Grade Distribution	
Grade	# of students
100-90	3
90-80	5
80-70	10
70-60	6
60-50	2
Less than 50	2

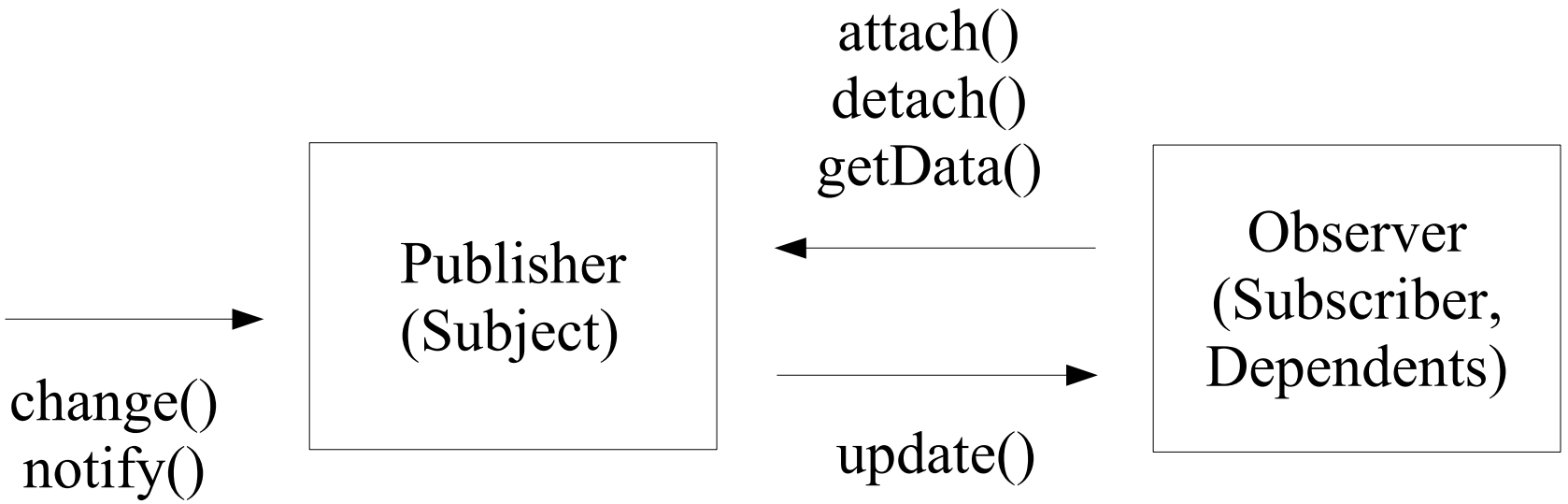




Motivation

- The main motivation behind observer is the desire to maintain consistency between related objects without making them tightly coupled.
- In our spreadsheet example, we don't want the different representations to be coupled with each other.
- However, if the information changes in the spreadsheet, all the different representations should be updated.

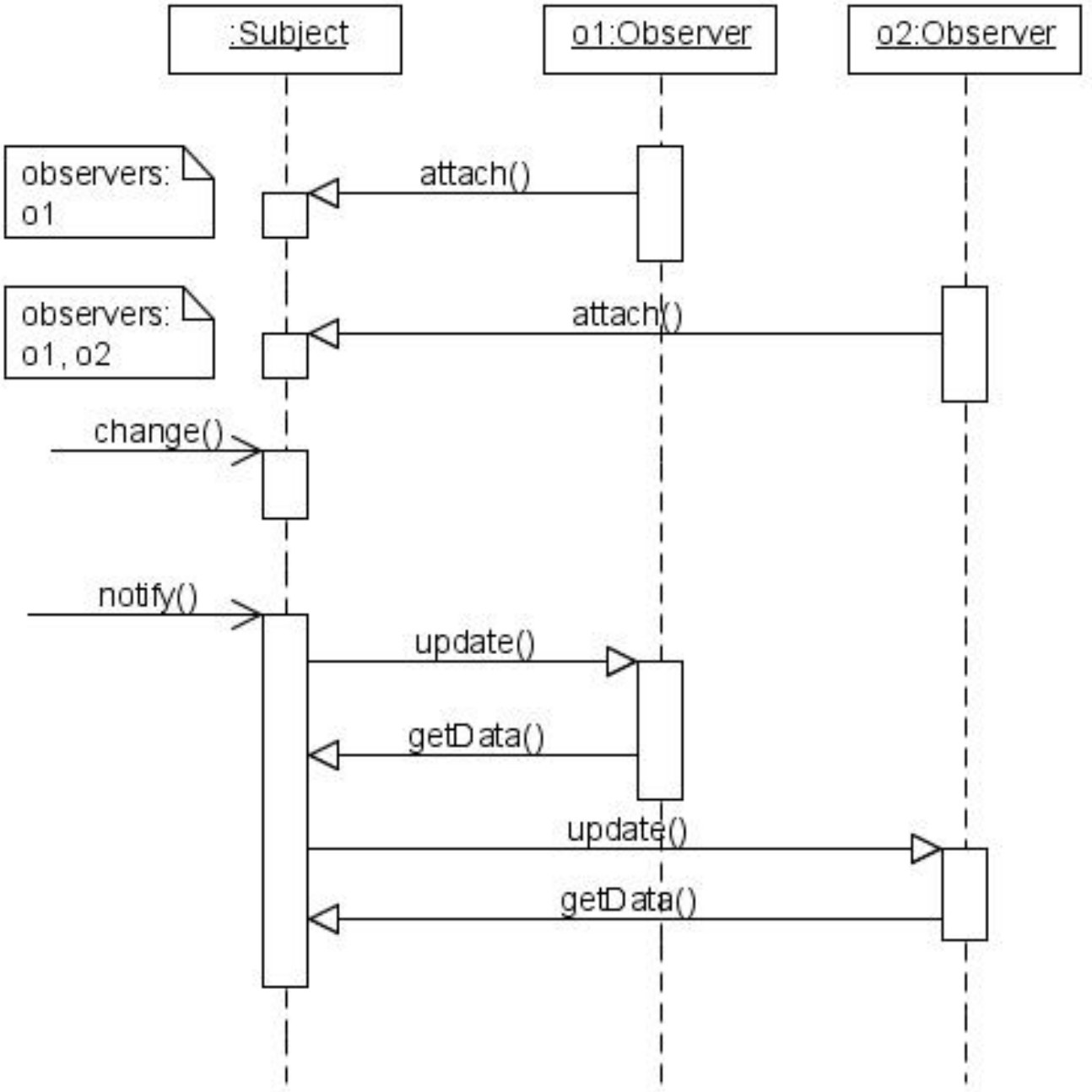
Participants



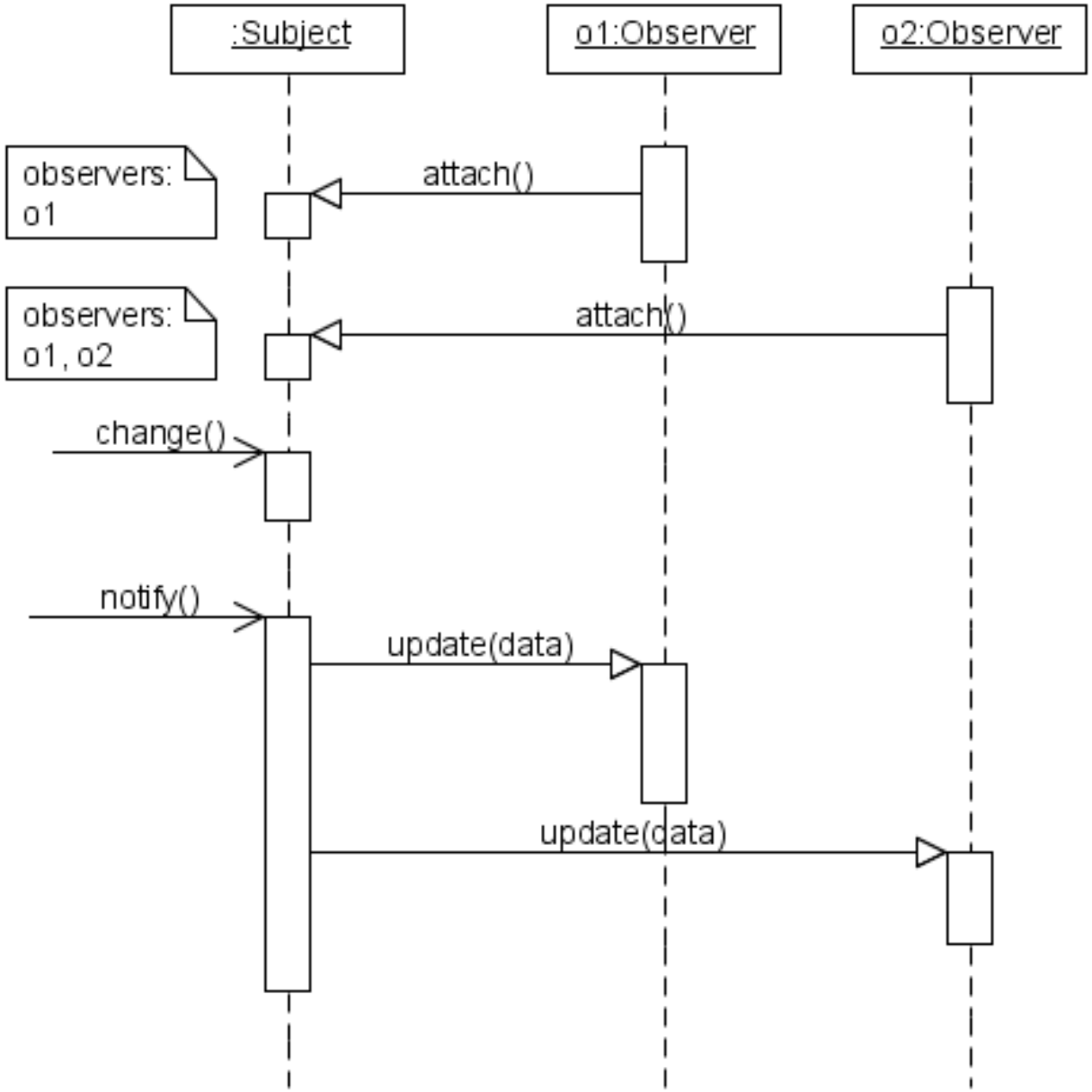
Actions

- Attach : let observer observe subject
- Detach : let observer not observe subject
- Notify : let observers know something has changed.
- Update : inform an observer that new data is available
- GetData : get data after notify (pull method)
- Update with data : send data to observers (push method)

Sequence (Pull)



Sequence (Push)



Applicability

- Abstraction has multiple aspects, each independent.
  - Encapsulation of both independently allows for reuse
- Unknown # of observers
- No assumptions made about observers
  - (except for update())

Consequences

- Abstract coupling between Subject and Observer.
  - The subject does not require knowledge of the observer.
  - The observer only needs to know how to get new data.
- Support for broadcast communication.
  - An update() triggers a broadcast communication across all observers.
- Unexpected updates.
  - The subject is blind to its observer. Thus, the cost of an update() is unknown.
  - Observers have no control to when they will receive updates.

Implementation Concerns

- The Observer pattern has numerous implementation concerns:
  - Push vs Pull
  - Who stores the subscription?
  - Observing more than one subject.
  - Who triggers update?
  - Deleting subjects and observers?
  - Subject's self-consistency
  - Complex subscriptions
  - Observers/Subject

Push vs Pull

- What are the advantages, disadvantages?

Pull

- In the pull model, observers are responsible to acquiring the new state after an update() is called.
- + Better transparency, subject doesn't need to know about observer.
- + Observer is free to determine if it wants to acquire the new state.
- - Observer must determine what is new without help from the subject.

Push

- In the push model, state changes are sent along the update message.
- + Efficient, observer does need to determine what was updated.
- - Requires the subject to know more about the observer (breaks abstraction).
- - Observer automatically receives the update, either it wants it or now.

Observing more that one subject

- In some situation, it might make senses that an observer be attached to more than one subject.
- Our current infrastructure is very poor for this.
  - We don't know who called the update method.
- How can we fix this?