CSE503: Software Engineering

David Notkin
University of Washington
Department of Computer Science & Engineering
Winter 2002

Implicit invocation

- Components announce events that other components can choose to respond to
- Components register interest in those events that they want to respond to
- In implicit invocation, the `invokes` relation is the inverse of the `names` relation
- Invocation does not require ability to name
- The central goal is to ease change: the components invoked can be changed without modifying the announcing component

Old II mechanisms

- Field [Reiss], DEC FUSE, HP Softbench, etc.
  - Components announce events as ASCII messages
  - Components register interest using regular expressions
  - Centralized multicast message server
- Smalltalk's Model-View-Controller
  - Registering with objects
  - Separating UI views from internal models
  - May request permission to change

New II mechanisms: or extensive uses of them

- JDK's
  - Different versions have somewhat different event models
- Java beans, Swing, …
- CORBA and COM

Objective

- Most of you are at least comfortable with using events
  - Probably primarily in the context of existing components and frameworks
- Several issues to cover
  - Thinking of implicit invocation as more than "just" events
  - Identifying some concrete software engineering reasons to use it
  - Identifying some limitations

Not just indirection

- There is often confusion between implicit invocation and indirect invocation
  - Calling a virtual function is a good example of indirect invocation
    - The calling function doesn't know the precise callee, but it knows it is there and that there is only one
    - Not true in general in implicit invocation
- An announcing component should not use (in the Parnas sense) any responding components
  - This is extremely difficult to define precisely
  - Roughly, the post-condition of the announcing component should not depend on any computation of the implicitly invoked components

Mediators

- One style of using implicit invocation is the use of mediators [Sullivan & Notkin]
- This approach combines events with entity-relationship designs
- The intent is to ease the development and evolution of integrated systems
  - Manage the coupling and isolate behavioral relationships between components

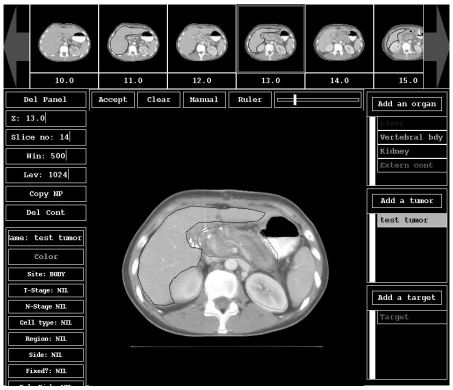
7

Experience

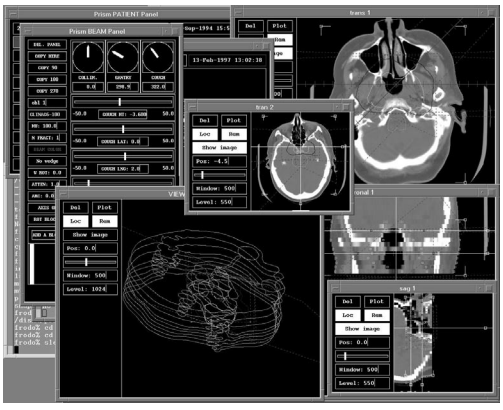
- A radiation treatment planning (RTP) system (Prism) was designed and built using this technique
  - By a radiation oncologist [Kalet]
  - A third generation RTP system
  - In clinical use at UW and several other major research hospitals
  - http://www.radonc.washington.edu/physics/prism/
  - See the screenshots on next slides

8

9



10



Example problem

- Given two sets S1 and S2, how do you ensure that they are always consistent?
  - You can independently add or delete elements from either set
- Ideas?

12

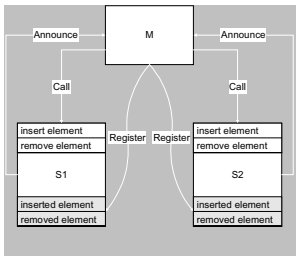
A standard solution

- Encapsulate both S1 and S2 in a component that
  - Exports insert-S1, insert-S2, delete-S1, delete-S2 methods
  - Implements these new methods to ensure consistency
- How effective is this solution?

13

Example solution: mediators

- Each set allows insertion and deletion of elements
- Each set also announces an "inserted" event and a "removed" event when the associated operation is performed
- A separate component, a mediator M, registers interest in the events from both sets
- Ex: If an element is inserted into S1, then M receives an "inserted" event; it then can invoke the insert method on S2



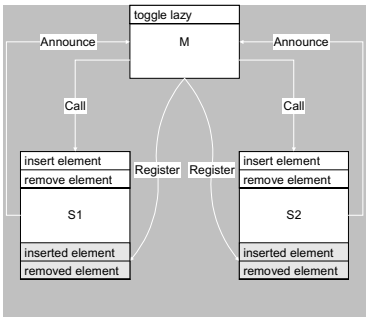
14

Mediator issues

- Must avoid circularity
- Events are first-class elements in interfaces
  - "interface" and "out-erface"
- Makes many changes easier
  - lazy equivalence
  - allow size of the sets to be changed directly
  - …

15

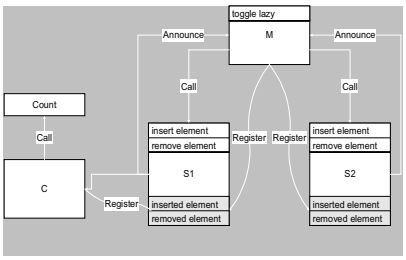
Mediator: with lazy update



- Only maintain equivalence of sets when a bit is set
- When the bit is moved from off to on, re-establish consistency

16

Mediators: lazy and count



- Track an integer with the count of the set elements
- With or without the lazy modification

17

Assessment

- For some classes of systems and changes, mediator-based designs seem attractive
- Lots of outstanding issues
  - Circularities in relations
  - Ordering of mediators
  - Distributed and concurrent variants
  - Reasoning (even informally) about systems built with implicit invocation
    - Even "just" debugging

18