

Software Engineering
CSE 335, Spring 2006

Object-oriented programming: Role-based design

Stephen Wagner

Michigan State University

Terminology

- *Collaboration*: pattern of message exchange among multiple objects to achieve some goal or purpose

Terminology

- *Collaboration*: pattern of message exchange among multiple objects to achieve some goal or purpose
- *Role*: that subset of an object's characteristics needed to fulfill its responsibilities in a collaboration
- We would like to design *reusable* collaborations
- Key is to model collaborations with *abstract roles*

Concrete to Abstract

Concrete

the print button

the document receiving a message
from the button

the print button and
document collaboration

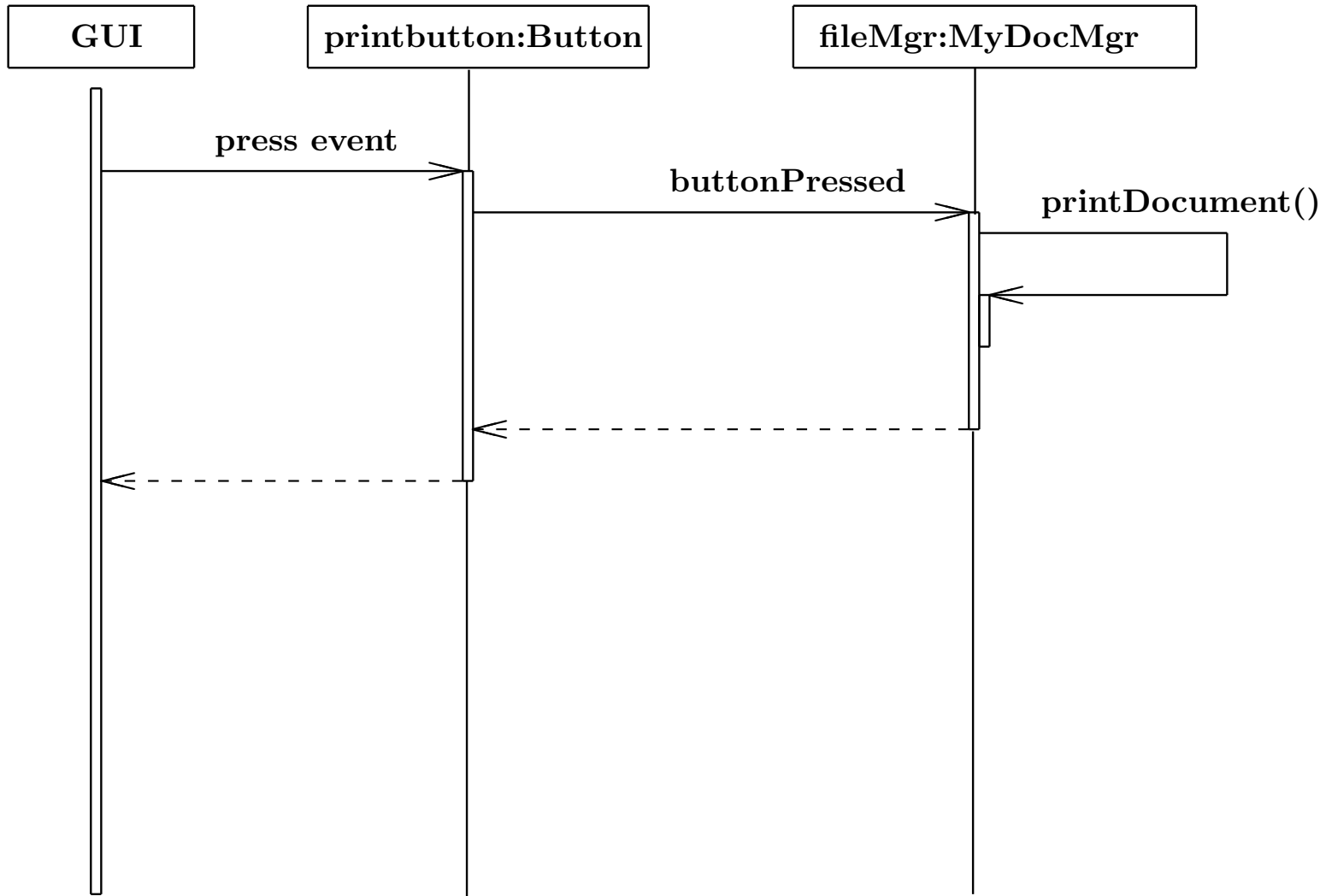
Abstraction

the Button class

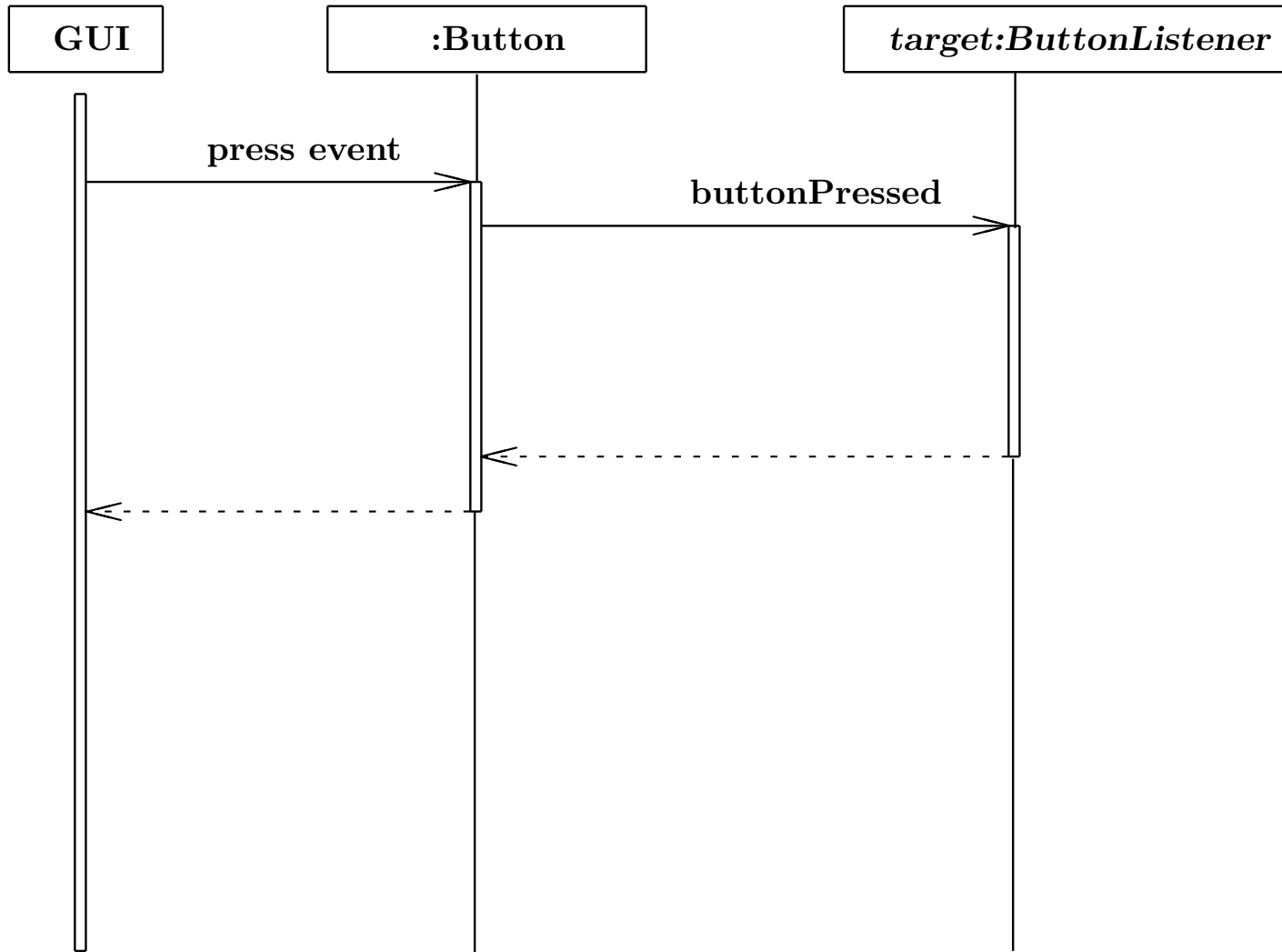
the abstract ButtonListener role

the Button-ButtonListener collaboration

A Collaboration



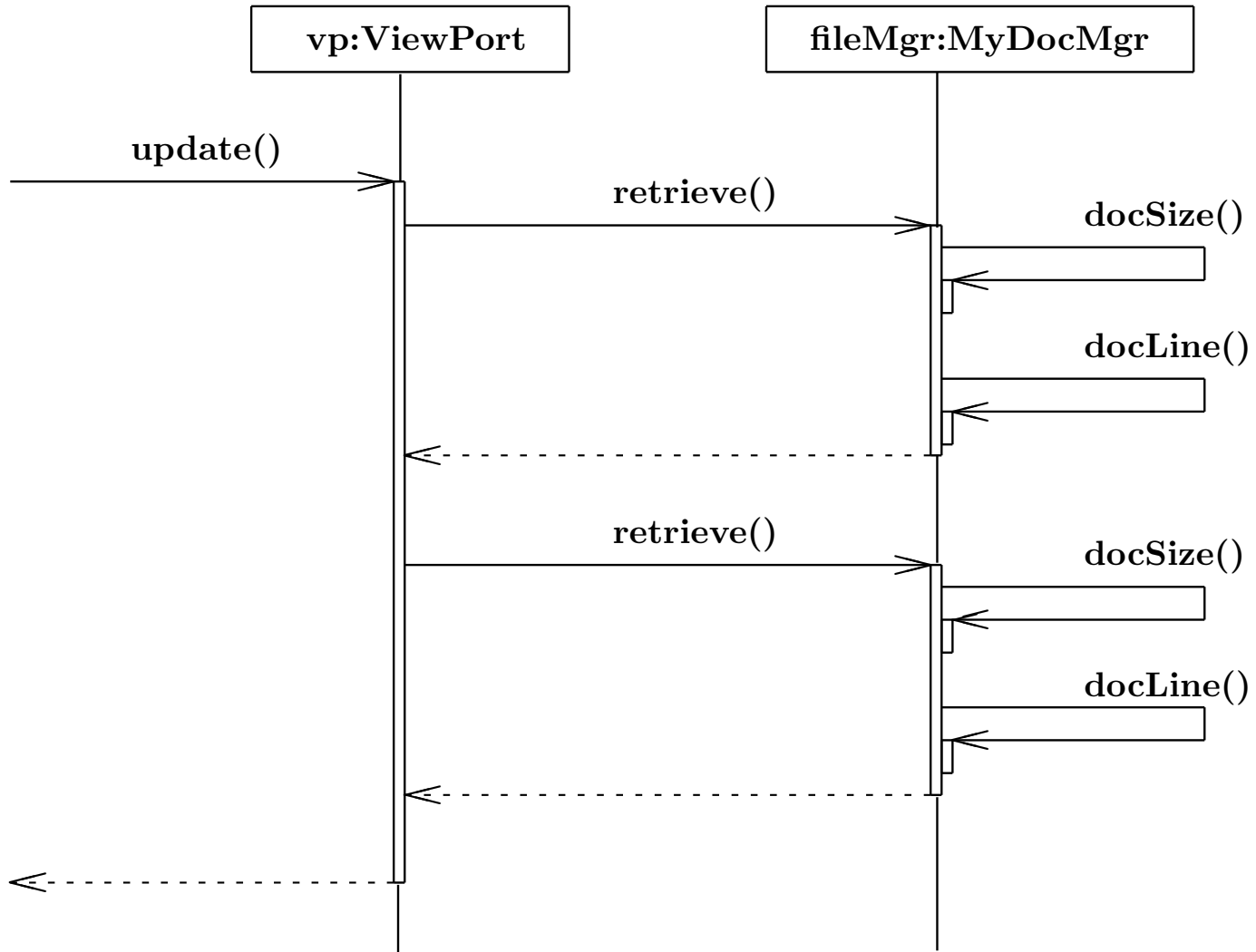
Collaboration with Abstract Roles



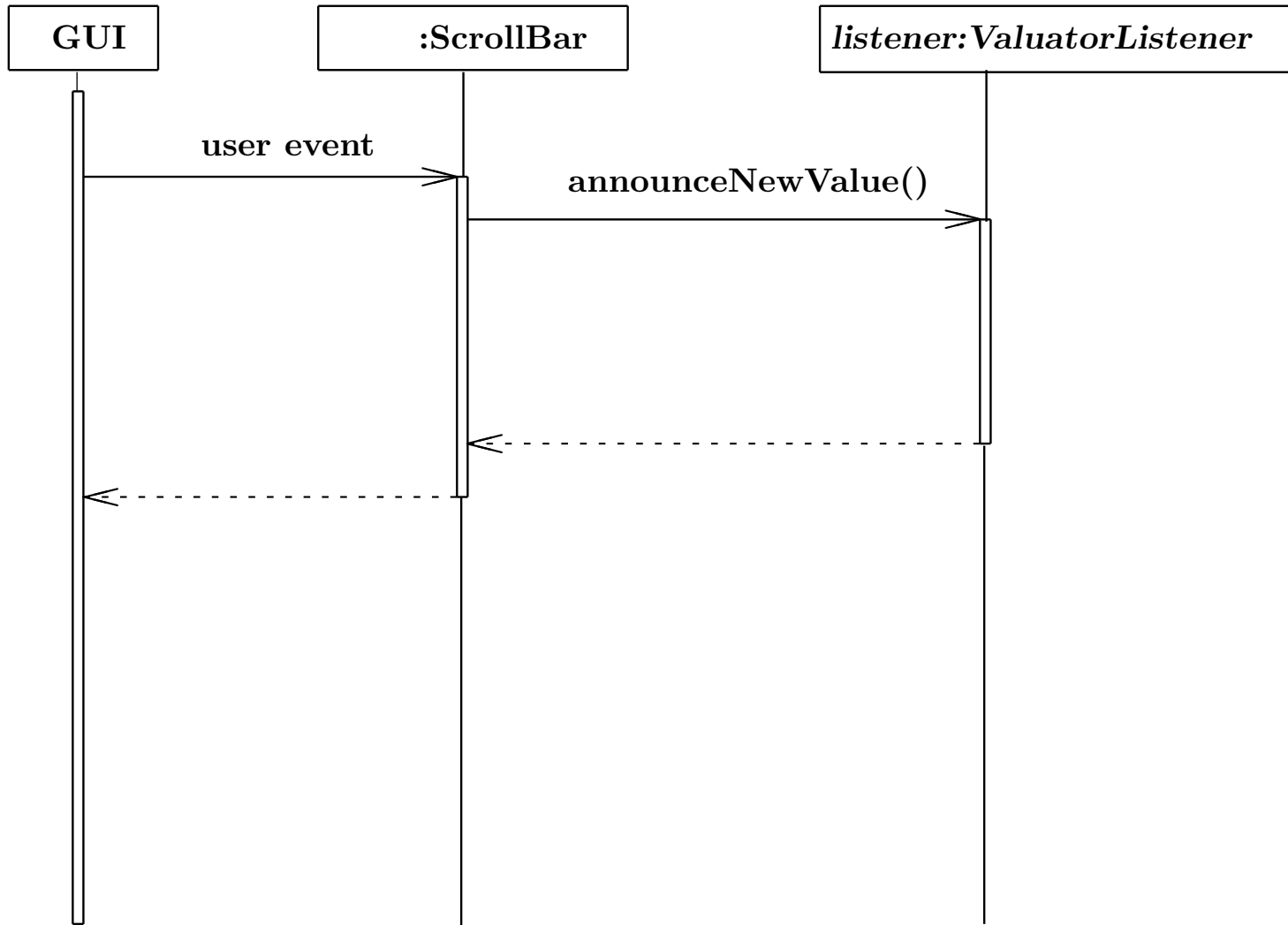
Another Application

- A scrollable viewport that displays the contents of a DocManager
- There are three objects
 - a Scrollbar
 - a Viewport
 - * keeps track of the first line to be displayed.
 - * has an update function that retrieves lines from its model
 - a DocManager
- What happens when we move the scroll bar?
- What behavior can be abstracted?

Another Collaboration



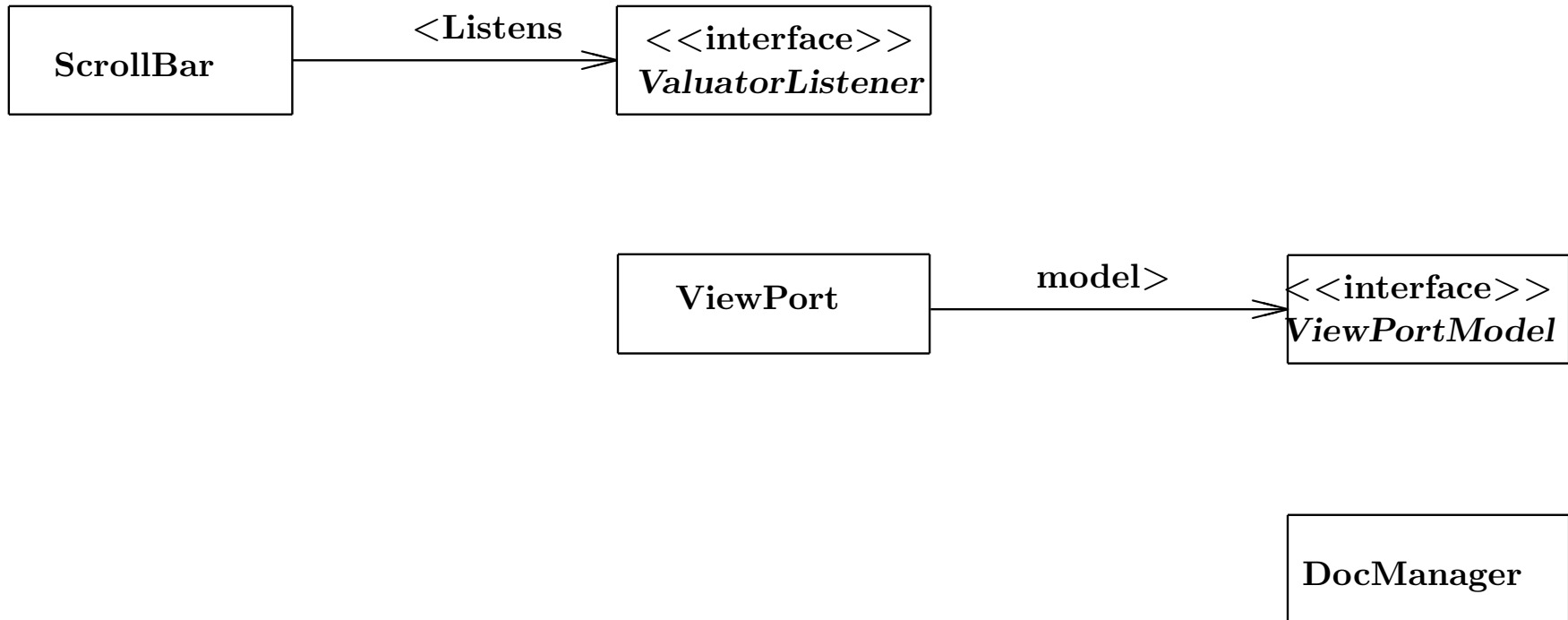
ScrollBar Collaboration



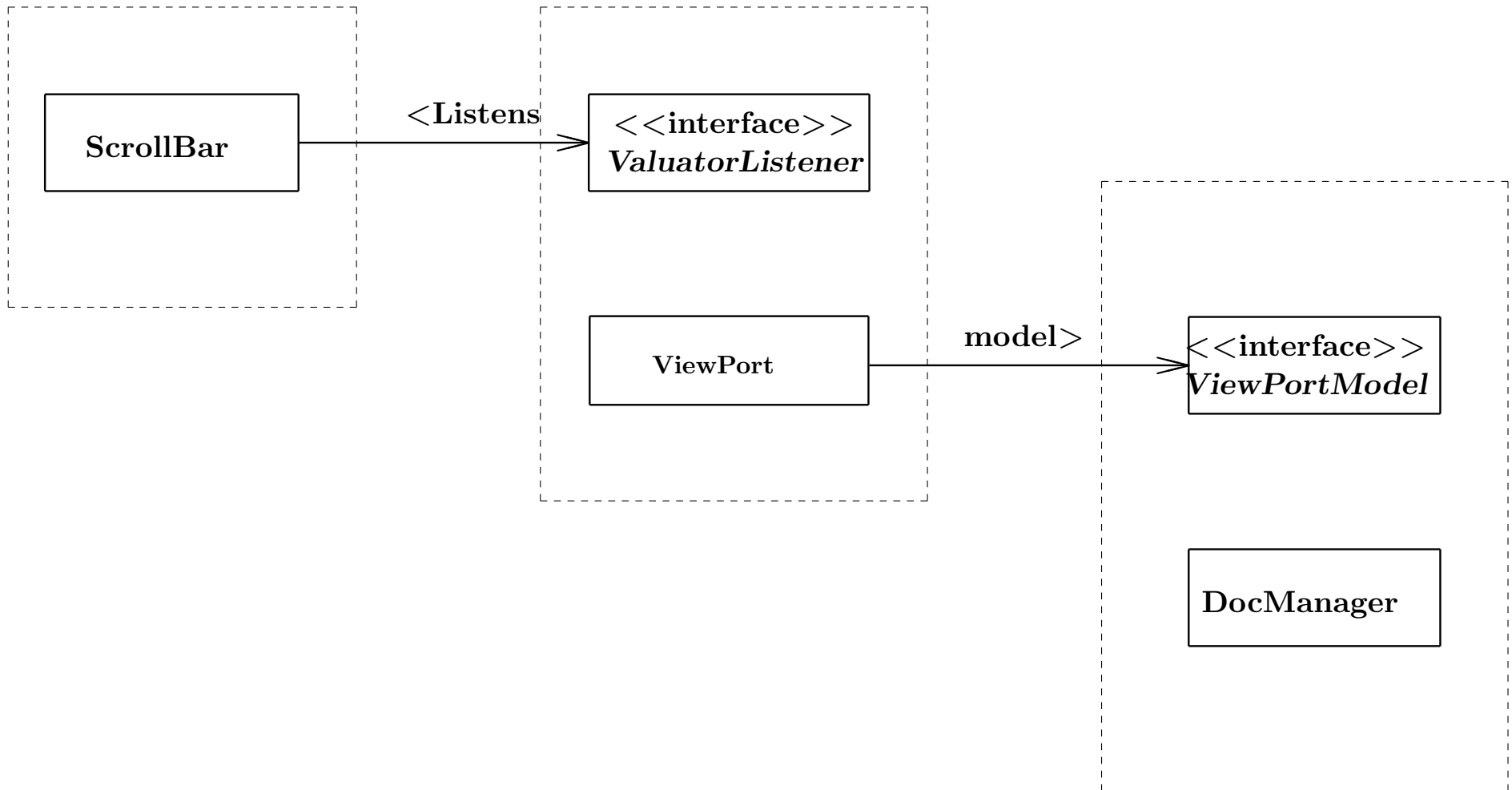
Synthesis

- The process of constructing an application by composing multiple collaborations
- Process
 - Gather together all relevant collaborations
 - Identify roles to be played by the same object
 - For each object, create an *adaptor class* to synthesize the role classes in the group for that object
 - Write *configuration code* that allocates and links the objects

Synthesis: (1) align collaborations



Synthesis: (2) compose roles



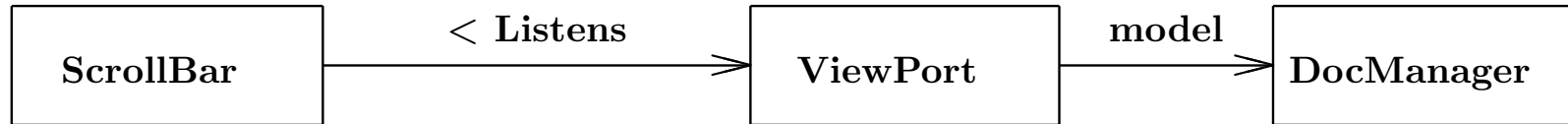
Synthesis: (3) design adaptor classes

- No need to adapt Scrollbar. Only class in its columns
- Other columns require us to adapt classes

```
class MyViewPort : public ValuatorListener,  
                  public ViewPort  
{ ..... };
```

```
class MyDocManager : public ViewportModel,  
                    public DocManager  
{ ..... };
```

Synthesis: (4) model configuration code



Synthesis: (5) write configuration code

```
MyViewPort          vp;  
ScrollBar           sb;  
MyDocumentManager  myMgr;
```

Synthesis: (5) write configuration code

```
MyViewPort          vp;  
ScrollBar           sb;  
MyDocumentManager  myMgr;
```

```
sb.registerListener(&vp);  
vp.registerModel(&myMgr);
```

Virtues of role-based designs

- Reuse: with a well-designed library, much of the work of building an application is concerned with the synthesis of collaborations
 - Good, because synthesis is relatively mechanical
 - Requires collaborations to be small and abstract
- Program understanding
 - Easier to understand roles in one collaboration than objects that are in many collaborations
 - Documentation should include sequence diagrams

Adding a home button

- Suppose we want to add a home button to our application that returns you to the beginning of the document
- What modifications do we have to make to our classes?

Reusability

- We want to design objects that can be reused
- In order to do this we need to avoid *tight coupling*
 - A reusable object should not explicitly refer to an object it interacts with
 - If Button contained a pointer to DocManager it would be an example of tight coupling
- To avoid this we use roles and interfaces

What about this?

```
template <class E>
class Button
{
    protected:
    E *target;
    ...
}

int main()
{
    Button<DocManager> printButton;
```

A General Purpose Browser

- In the homework you are building a browser
 - Combines a scrollbar and viewport
 - Allows you to display lines of text
- Suppose we wanted a more general browser that could display more than just text?
- What additional roles would be useful?