Walter Blair
603 Assignment 8

Mediator Pattern

For my last assignment on the Observer pattern, I looked at some GUI components of Libreoffice (https://github.com/LibreOffice/core) which is written in C++. Though I've never spent any time looking at C++, the software seems to be clearly enough written and documented to show a nice example of the Observer pattern as well as the Mediator pattern. I wrote last time about the relationship between the Observer XChangeListener and its subject XConfigurationController, and I'm in the same neck of the woods in my discussion of the Mediator pattern.

XContainer is an interface for defining containers of addable and removable items that broadcast configuration changes – I think this could be a popout window that contains various interactive elements, and this is closely tied to the Observer pattern I discussed in the last assignment (Snippet 1).

```
30
31  /** supports quick access to the information if a container currently
32      contains elements.
33
34      <p>The XContainer interface is provided for containers
35      which need to broadcast changes within the container; that means
36      the actions of adding or removing elements are broadcast to the
37      listeners.  </p>
38
39      <p>This can be useful for UI to enable/disable some functions
40      without actually accessing the data.  </p>
41
42      @see XContent
43      @see XIndexAccess
44      @see XNameAccess
45      @see XEnumerationAccess
46   */
47  published interface XContainer: com::sun::star::uno::XInterface
48  {
49
50      /** adds the specified listener to receive events when
51          elements are inserted or removed.
52
53         <p>It is suggested to allow multiple registration of the same listener,
54          thus for each time a listener is added, it has to be removed.
55
56          @see XContainerListener
57       */
58      void addContainerListener( [in] com::sun::star::container::XContainerListener xListener );
59
```

Snippet 1. XContainer.idl - interface for creating container types that broadcast their changes (related to the Observer pattern I discussed last assignment)

Stepping away from the Listener component of Xcontainer, Libreoffice needs to simplify the communication between these broadcasting XContainers, so a Mediator pattern is a great way to handle

simplified communication between similar but distinct components. Enter ContainerMediator! To my untrained eye, it looks to me like ContainerMediator maintains references to each XContainer, but I can't find where each colleague XContainer maintains a reference to its mediator (Snippet 2). I believe given our lecture notes that the code shown in Snippet 2 is actually showing that ContainerMediator passes to each XContainer a reference to itself, so that each container maintains a reference to the same ContainerMediator. Afterall, each container needs to know who to go talk to when it is trying to broadcast configuration change messages.

```
33  namespace dbaccess
34  {
35
36      class OPropertyForward;
37
38      class OContainerMediator :   public ::cppu::BaseMutex
39                                  ,public ::cppu::WeakImplHelper< css::container::XContainerListener >
40      {
41      private:
42          typedef ::rtl::Reference< OPropertyForward >          TPropertyForward;
43          typedef std::map< OUString, TPropertyForward >      PropertyForwardList;
44          PropertyForwardList                                   m_aForwardList;
45          css::uno::Reference< css::container::XNameAccess >    m_xSettings;    // can not be weak
46          css::uno::Reference< css::container::XContainer >     m_xContainer;   // can not be weak
47
48      protected:
49          virtual ~OContainerMediator() override;
50
51      public:
52          OContainerMediator(
53              const css::uno::Reference< css::container::XContainer >& _xContainer,
54              const css::uno::Reference< css::container::XNameAccess >& _xSettings
55             );
56
57          virtual void SAL_CALL elementInserted( const css::container::ContainerEvent& _rEvent ) override;
58          virtual void SAL_CALL elementRemoved( const css::container::ContainerEvent& _rEvent ) override;
59          virtual void SAL_CALL elementReplaced( const css::container::ContainerEvent& _rEvent ) override;
60          virtual void SAL_CALL disposing( const css::lang::EventObject& Source ) override;
61
62          void notifyElementCreated(const OUString& _sElementName
63                                  ,const css::uno::Reference< css::beans::XPropertySet>& _xElement);
64
```

Snippet 2. ContainerMediator.hxx showing the interface of a ContainerMediator. The key code here is the reference for each XContainer that would allow each container to contact this mediator.

I believe that my latter interpretation is correct, that the container/mediator here follow the pattern indicated in our lecture notes where each container contains a reference to the mediator. Looking at the C++ code file for ContainerMediator doesn't reveal anything more than I found in the header file above, but looking at a ConcreteColleague that implements XContainer I think helps me. OTableContainer is I believe a concrete implementation of the XContainer interface, and Snippets 3 and 4 below show that this container first declares and initializes a null pointer to a mediator and then assigns a mediator to which it has a reference. The ConcreteMediator then uses its interface magic discussed above to maintain a reference to the container object. Voila! We have a ConcreteColleague that maintains a reference to its ConcreteMediator and vice versa.

```
97
98  OTableContainer::OTableContainer(::cppu::OWeakObject& _rParent,
99                                   ::osl::Mutex& _rMutex,
00                                   const Reference< XConnection >& _xCon,
01                                   bool _bCase,
02                                   const Reference< XNameContainer >& _xTableDefinitions,
03                                   IRefreshListener*  _pRefreshListener,
04                                   std::atomic<std::size_t>& _nInAppend)
05      :OFilteredContainer(_rParent,_rMutex,_xCon,_bCase,_pRefreshListener,_nInAppend)
06      ,m_xTableDefinitions(_xTableDefinitions)
07      ,m_pTableMediator( nullptr )
08  {
09  }
10
```

Snippet 3. TableContainer.cxx, where the TableContainer initializes a null reference to a TableMediator. This shows that the ConcreteColleague is ready to be assigned to a ConcreteMediator.

```
21          Reference<XPropertySet> xDest(xRet,UNO_QUERY);
22          if ( xTableDefinition.is() )
23              ::comphelper::copyProperties(xTableDefinition,xDest);
24
25          if ( !m_pTableMediator.is() )
26              m_pTableMediator = new OContainerMediator(
27                      this, m_xTableDefinitions.get() );
28          if ( m_pTableMediator.is() )
29              m_pTableMediator->notifyElementCreated(_rName,xDest);
30      }
31
```

Snippet 4. Later in the same TableContainer.cxx file, the concrete TableMediator is assigned