Walter Blair
603 Assignment 7

Observer Pattern

Given that a GUI is a great place to look for the Observer pattern, I figured I'd use this assignment as an excuse to take my first look into LibreOffice. My first thought was to peek into GIMP, but I was noticed that GIMP is written in C, and I don't have the energy to untangle the presumed existence of OOD in C. I guess that shows that I am a real Millenial when it comes to coding – I flee from C. Anyway, LibreOffice (https://github.com/LibreOffice/core) is written in C++ so its use of OOD seems more intuitive.

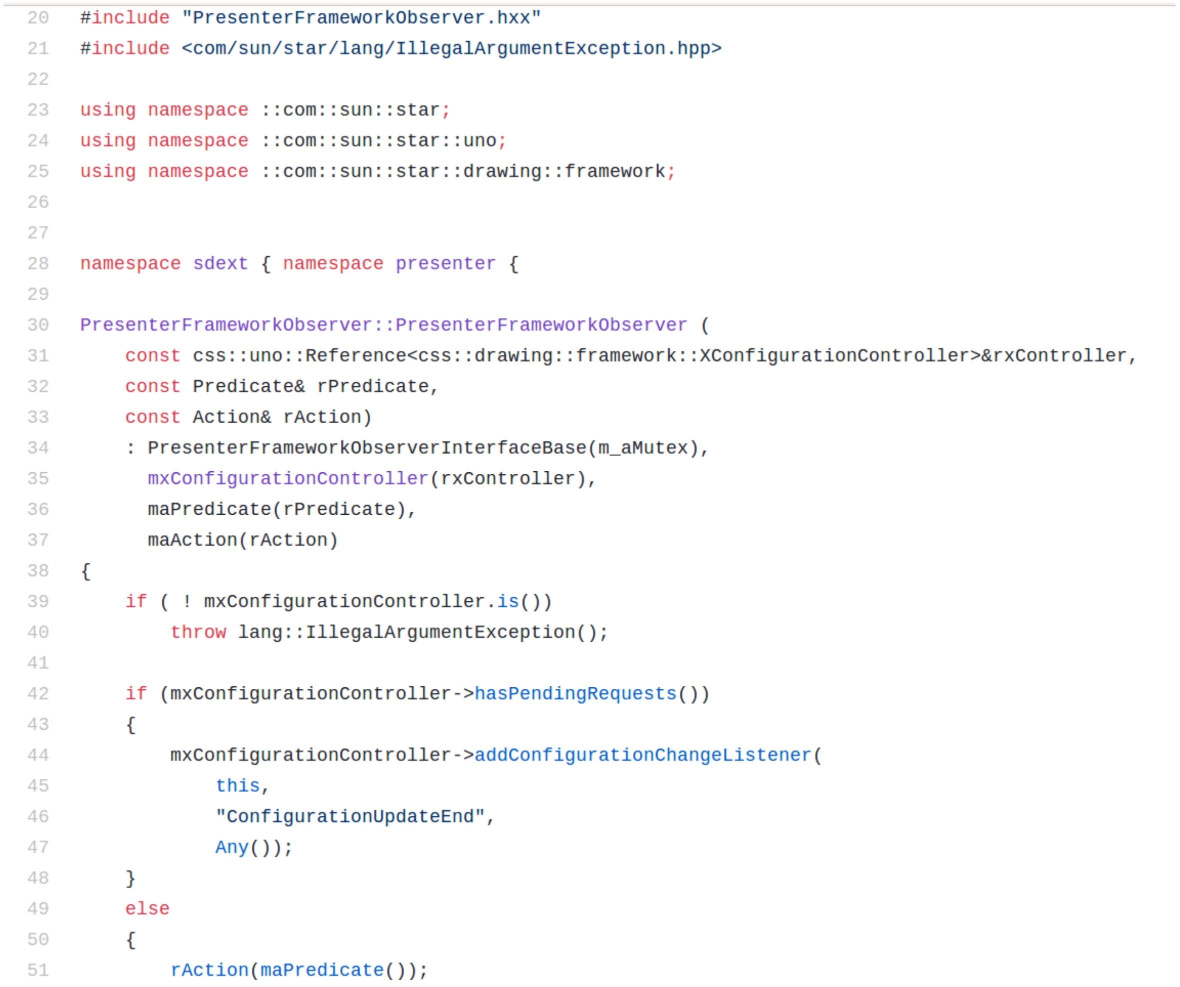
I searched the repository for Observer and followed the PresenterFrameworkObserver for a few minutes. I ditched this class, because I couldn't identify the subject of this observer, but it led me to the XConfigurationController and XConfigurationChangeListener. I'm fairly certain that X refers to the windows manager, so I believe that this Publish-Subscribe pattern I've found helps manage the current and requested states of the GUI as the user clicks buttons and such during operation.

I've never used C++, so I was interested to find these .idl files that at least in this case tell me everything I need to know about each class. The XConfigurationController.idl file explains that GUI changes are processed in two steps: First, the requested changes are broadcast from the request queue to the appropriate listener(s), any of which may add its own subsequent requests to the queue. Once the queue is empty, the second step is to actually activate the resources that will implement the changes. To my ear, this perfectly describes the publish-subscribe system that fits well with the Observer pattern.

Alright, so XConfigurationController is the abstract Subject, and XConfigurationChangeListener is its abstract Observer, so where are the concrete examples? Back to PresenterFrameworkObserver and the beginning of the C++ code for this class (Snippet 1).

My reading of the above snippet is that PresenterFrameworkObserver registers a particular number of listeners of type ConfigurationChangeListeners that will receive the requested changes for the ConcreteSubject mxController of type XConfigurationController.

I'm going to try revising the above paragraph as I try to grapple with my C++ newbishness. Line 44 is the code block where a new ConcreteObserver is registered on behalf of the ConcreteSubject mxConfigurationController. I was originally thinking that PresenterFrameworkObserver is sort of overseeing the addition of various listeners, but now I think that Line 45 is returning a PresenterFrameworkObserver, which would mean that the ConcreteObserver that is being registered to mxConfigurationController is in fact PresenterFrameworkObserver. If this is the correct reading, then my revised model would be the following (Figure 1).



```
#include "PresenterFrameworkObserver.hxx"
#include <com/sun/star/lang/IllegalArgumentException.hpp>

using namespace ::com::sun::star;
using namespace ::com::sun::star::uno;
using namespace ::com::sun::star::drawing::framework;


namespace sdext { namespace presenter {

PresenterFrameworkObserver::PresenterFrameworkObserver (
    const css::uno::Reference<css::drawing::framework::XConfigurationController>&rxController,
    const Predicate& rPredicate,
    const Action& rAction)
    : PresenterFrameworkObserverInterfaceBase(m_aMutex),
      mxConfigurationController(rxController),
      maPredicate(rPredicate),
      maAction(rAction)
{
    if ( ! mxConfigurationController.is())
        throw lang::IllegalArgumentException();

    if (mxConfigurationController->hasPendingRequests())
    {
        mxConfigurationController->addConfigurationChangeListener(
            this,
            "ConfigurationUpdateEnd",
            Any());
    }
    else
    {
        rAction(maPredicate());
```

Snippet 1. PresenterFrameworkObserver.cxx file shows Observer/Subject pattern between PresenterFrameworkObserver/mxConfigurationController.

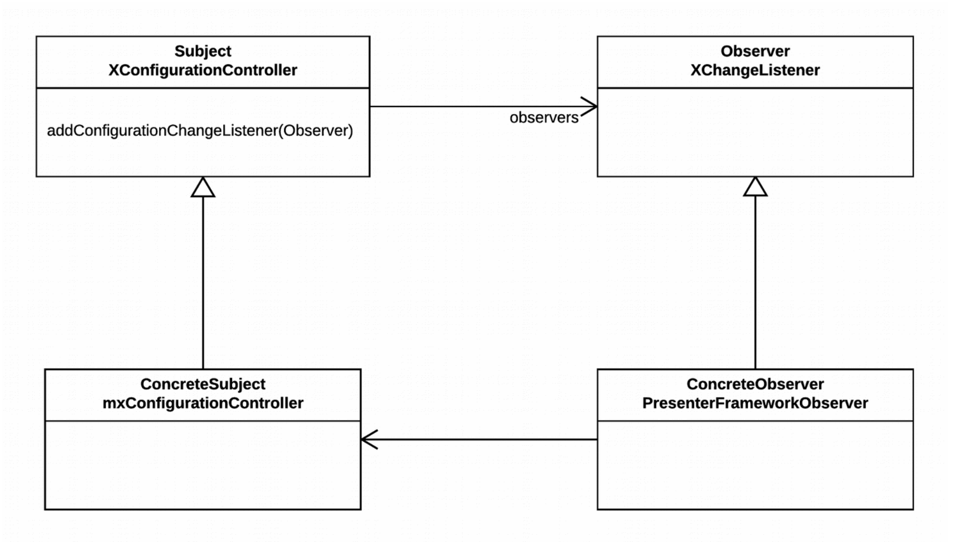


Figure 1. My understanding of the Observer pattern in this example.