Death of a (Sw)Client

Bjoern Michaelsen, Twitter: @Sweet5hark

2018-09-27, LibreOffice Conference, Tirana

Death of a (Sw)Client

Part One: What is SwClient?

Observer pattern

The Observer pattern addresses the following problems:

- A one-to-many dependency between objects should be defined **without making the objects tightly coupled.**
- It should be ensured that when one object changes state an open-ended number of dependent objects are updated automatically.
- It should be possible that one object can notify an open-ended number of other objects.

[...]

The responsibility of observers is to register (and unregister) themselves on a subject (to get notified of state changes) and to update their state (synchronize their state with subject's state) when they are notified.

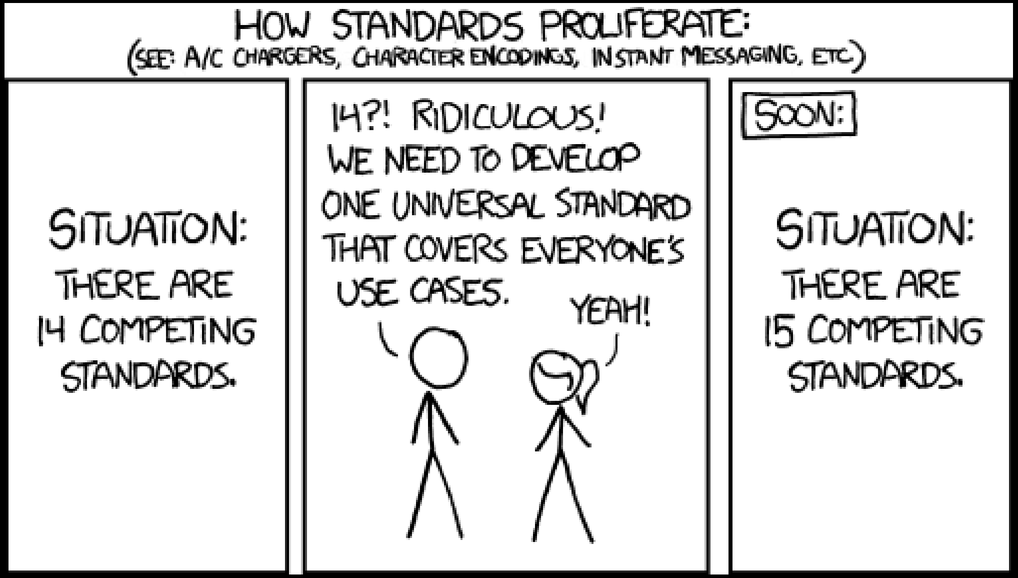
This makes subject and observers loosely coupled. Subject and observers have no explicit knowledge of each other.

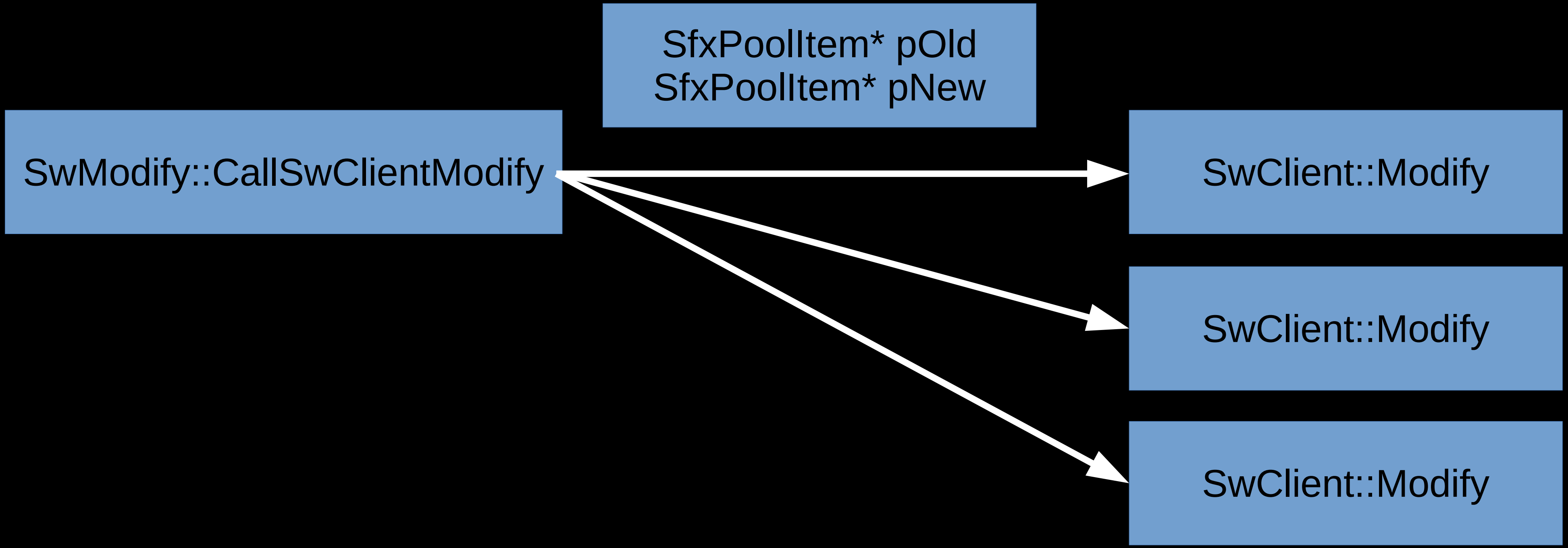
(source: Wikipedia CC-BY-SA/GFDL)

LibreOffice observer patterns

- XEventListener/OweakEventListener
- SfxListener
- SvtListener
- VclListener

- and: SwClient/SwModify


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.


SfxPoolItem* pOld
SfxPoolItem* pNew
SwModify::CallSwClientModify
SwClient::Modify
SwClient::Modify
SwClient::Modify

SwModify and SwClient

- A SwModify (Observable) notifies its SwClients (Listeners) about events
- The SwClients are collected in an **intrinsic doubly linked list**
- So: When an event happens at the SwModify it iterates over its SwClients and dumps this event in the handler function

Modification while iterating

- "a FOO client added as listener to a BAR during client iteration."
- SwModifies (Observables) keep a list of live iterators

In Writer ~everything is a SwClient

- The usual way to use SwClient/SwModify is to derive from one of them
- Even SwModify is derived from SwClient
- When you want to send or receive events you need to derive from SwClient

```
std::unique_ptr<sw::ModifyChangedHint> SwClient::CheckRegistration( const SfxPoolItem* pOld )
  {
    DBG_TESTSOLARMUTEX();
    // this method only handles notification about dying SwModify objects
    if( !pOld || pOld->Which() != RES_OBJECTDYING )
      return nullptr;

    const SwPtrMsgPoolItem* pDead = static_cast<const SwPtrMsgPoolItem*>(pOld);
    if(!pDead || pDead->pObject != m_pRegisteredIn)
    {
      // we should only care received death notes from objects we are following
      return nullptr;
    }
    // I've got a notification from the object I know
    SwModify* pAbove = m_pRegisteredIn->GetRegisteredIn();
    if(pAbove)
    {
      // if the dying object itself was listening at an SwModify, I take over
      // adding myself to pAbove will automatically remove me from my current pRegisteredIn
      pAbove->Add(this);
    }
    else
    {
      // destroy connection
      EndListeningAll();
    }
    return std::unique_ptr<sw::ModifyChangedHint>(new sw::ModifyChangedHint(pAbove));
  }
```

Reregistering horrors

- When a SwModify (Observable) dies …
- … by default all SwClients (Listeners) reregister at the pAbove of the SwModify

Multithreading, Locking & Mutexes

- Writer is a big mudball of SwClients throwing events at each other all over the place
- Sometimes even in circles: Looping Louie
- No hierarchy, no locality

Whats this?

GetRegisteredIn() is a void*

- There is no static guarantee about the type GetRegisteredIn() returns

- git grep GetRegisteredIn|grep static_cast|wc -l

  96

Manual iteration of SwClients

- SwModifies (and even third party classes) often iterate over SwClients directly
- Even iterating over a subset of SwClients (based on type) is done regularly
  - Renders the Observer pattern pointless as the Observable (SwModify) has to have deep knowledge about clients
  - Also: a good cache smashing excerise and general performance horror

Random calls into event handlers

- NotifyClients: git grep NotifyClients|grep -v calbck|wc -l

  43

- ModifyNotification: git grep ModifyNotification\(|grep -v calbck|wc -l

  94

Lapsed Listener Problem:
A “feature”?

- Unfortunately, throwing naked pointers all over Writer is fragile
- However, the default implementation cover it reasonably well …
- … so the “wise” Writer devs of old used it for memory management

SwDepend: A broken workaround for brokeness

- How to listen to more than one Observable (SwModify)
- “Solution”:
  - Have a minimal SwClient implementation as a proxy helper class
  - Forward its events to the “real” SwClient (also expect to be friended by that class)

Death of a (Sw)Client

Part Two: The long death of SwClient

Add unittests!

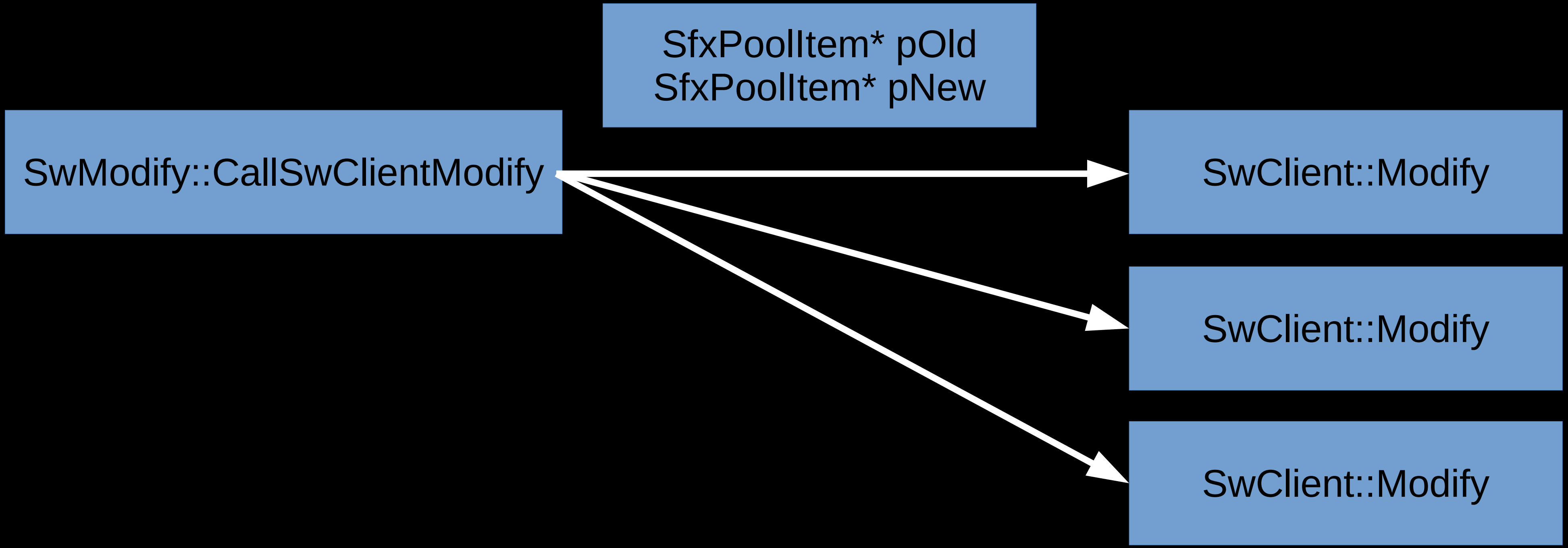
- Yes, it was the ~most fundamental implementation in Writer
- Nope, there hadnt been any unittests

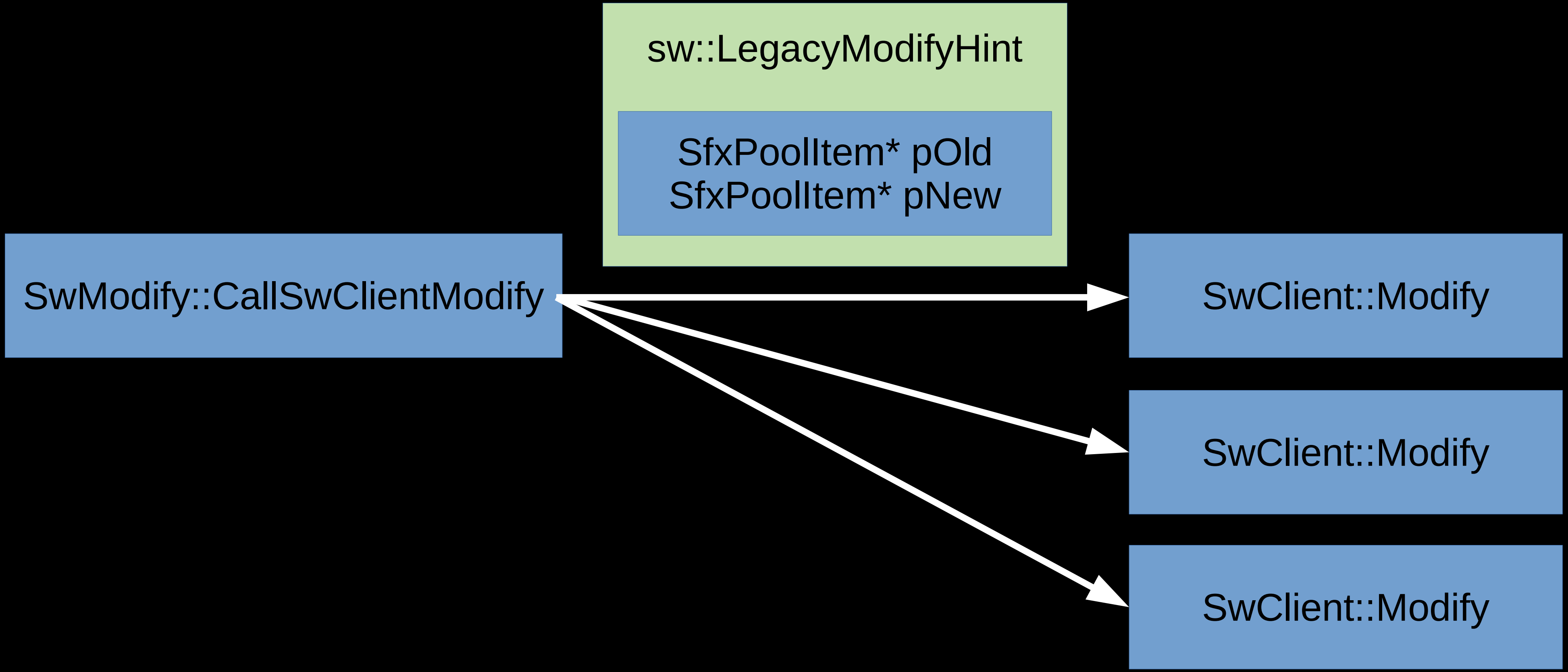
Remove Cargo Cult/Dead Code

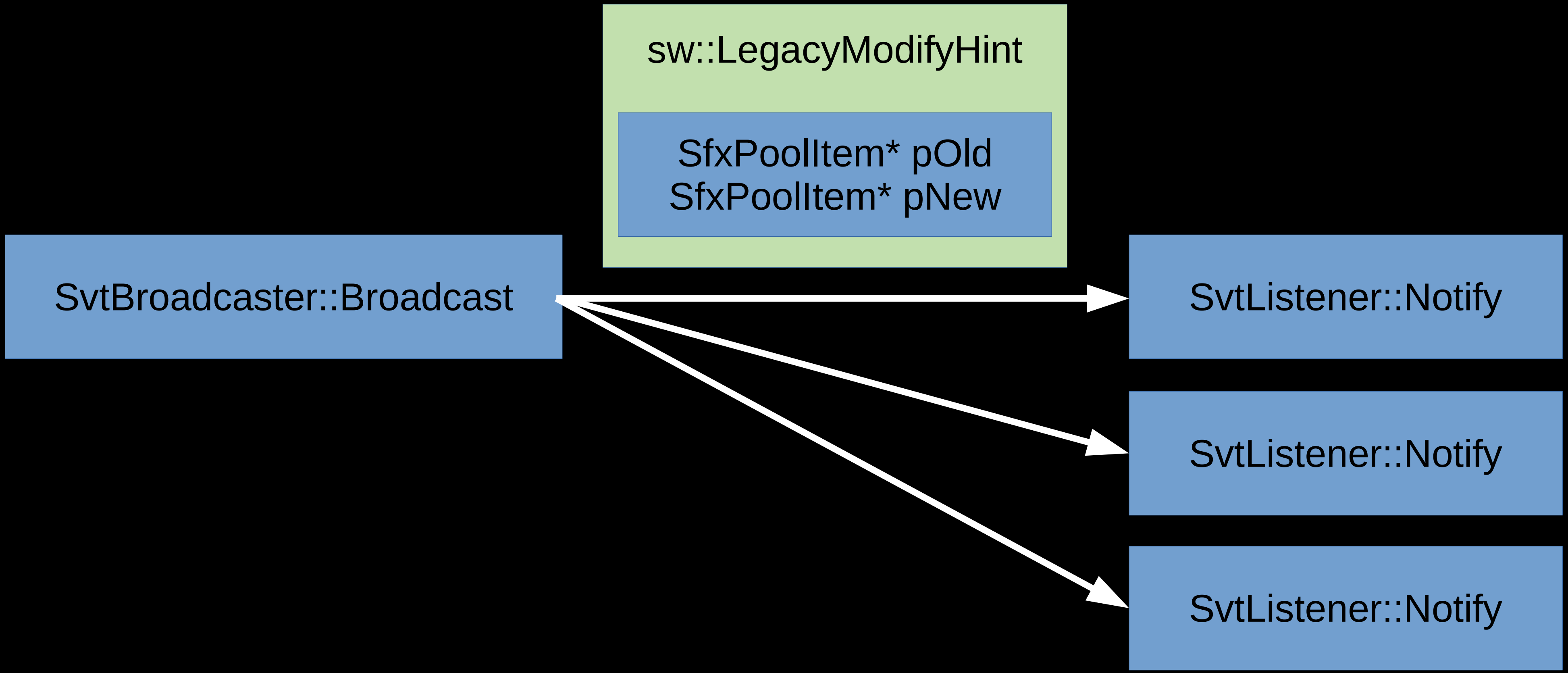
- Remove dead/unused/cargo cult code
- Cut back to a core that does the observer pattern and only that

SfxHints and LegacyHints

- Add sw::LegacyHints which is a SfxHint ...
- … and also wraps the old event messages.
- So the events are already in the new “format” and are tunneled through the old implementation.


SfxPoolItem* pOld
SfxPoolItem* pNew
SwModify::CallSwClientModify
SwClient::Modify
SwClient::Modify
SwClient::Modify


sw::LegacyModifyHint
SfxPoolItem* pOld
SfxPoolItem* pNew
SwModify::CallSwClientModify
SwClient::Modify
SwClient::Modify
SwClient::Modify


sw::LegacyModifyHint
SfxPoolItem* pOld
SfxPoolItem* pNew
SvtBroadcaster::Broadcast
SvtListener::Notify
SvtListener::Notify
SvtListener::Notify

Some internal renovation

- Use saner STL/boost container
  - boost/ring.hxx

- Make SwClientIterator somewhat more typesafe
  - template< typename TSource >
    class SwIterator<SwClient, TSource> final
  - add some static_asserts on TSource

Add sw::BroadcasterMixin

- Mixin-class that adds a good old SfxBroadcaster as a member to objects
- Still easy use by inheritance, but otherwise its **composition over inheritance**
- This allow incremental migration, and breaks the

    “everything **must** derive from SwClient”

  down to a

    “everything **can** derive from a sw::BroadcasterMixin”

Remove SwDepend

- SfxListener can listen to multiple observables (SfxBroadcasters), so no more need for SwDepend
- But:

  Incremental migration, so we might need both

  - SfxBroadcaster/SfxListener

  and

  - SwModify/SwClient

  for some time

Progress (What happened so far)

- git grep public.*SwClient sw/source/core/unocore/|wc -l

  10
- git grep SwClient sw/source/core/access/|wc -l

  4 (plus 2 in comments)
- git grep SwClient sw/source/core/layout/|wc -l

  28

Regressions :/

- Reported, triaged, bibisected and fixed:
  - tdf#117749
  - tdf#117774
- Reported, triaged, bibisected and not fixed on master:
  - tdf#118049
  - tdf#118725
  - tdf#118833
- Reported, triaged, bibisected and not fixed on 6.1:
  - tdf#120115

Death of a (Sw)Client

Bjoern Michaelsen, Twitter: @Sweet5hark

2018-09-27, LibreOffice Conference, Tirana