Software Implementation to Reduce the Impact of Interface Revision

STC 2014
April 3, 2014

LOCKHEED MARTIN

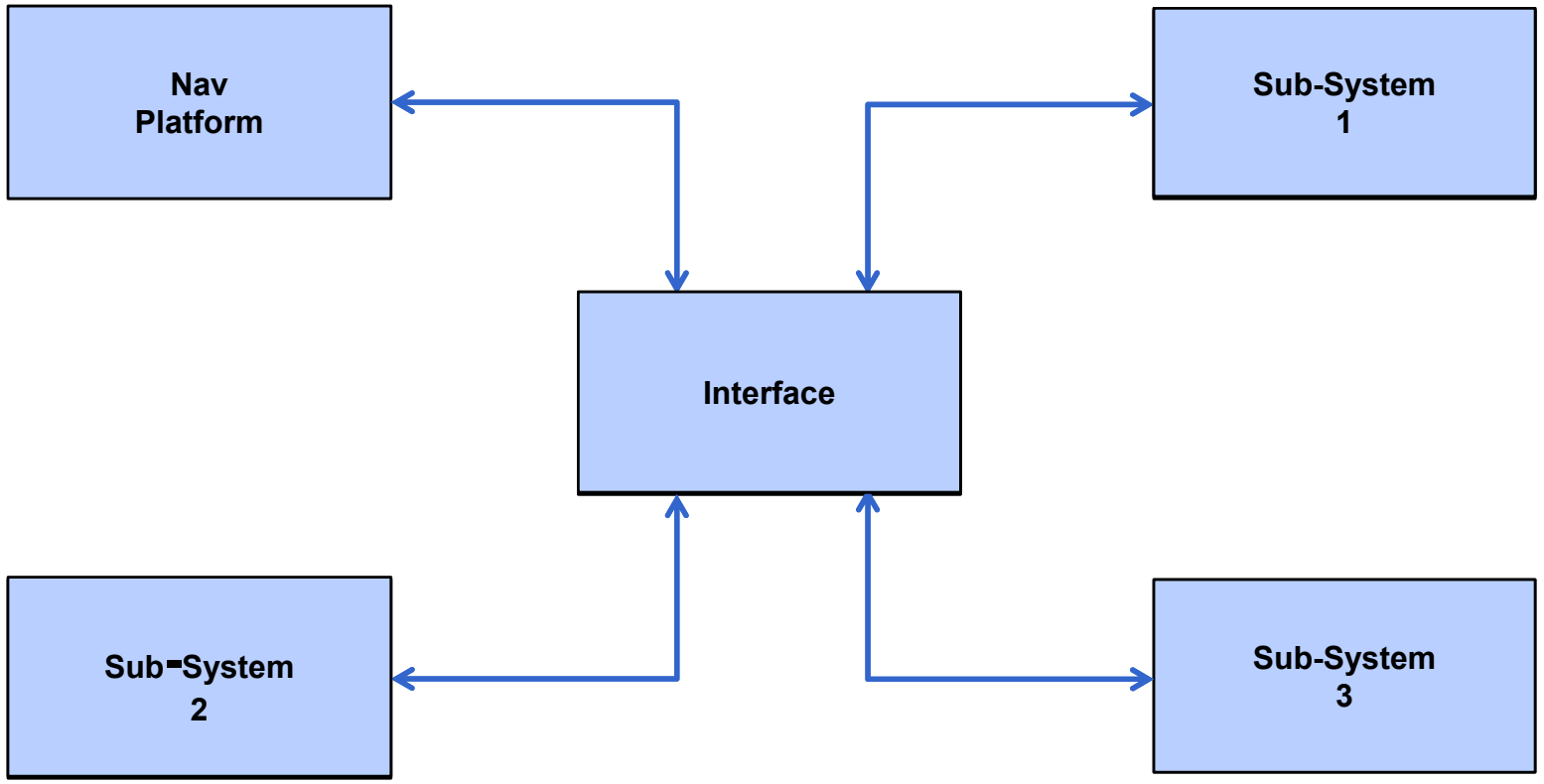
Gerry Tyra
Embedded Software Engineer, Sr. Staff

Problem: Change happens

Corollary to Murphy's Law:
Small changes ripple through a project

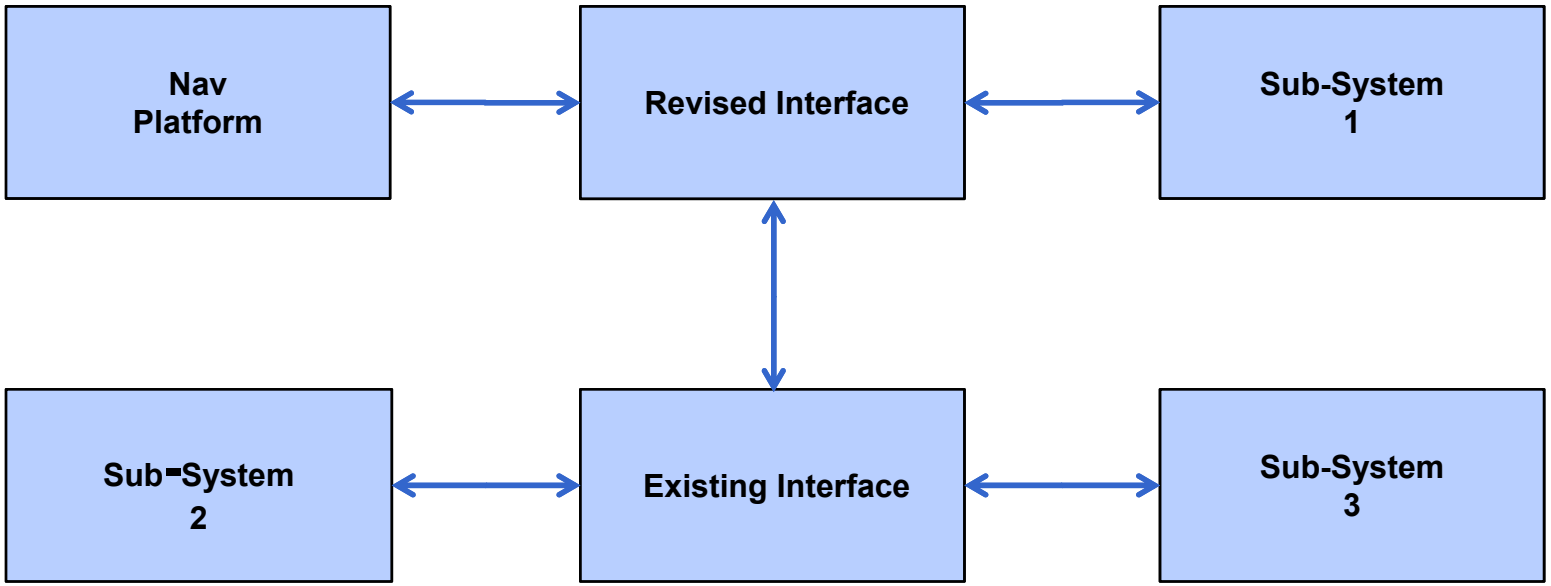
Solution:
1) Stop the ripple
2) Optional metadata to track the change

Basic Avionics Package



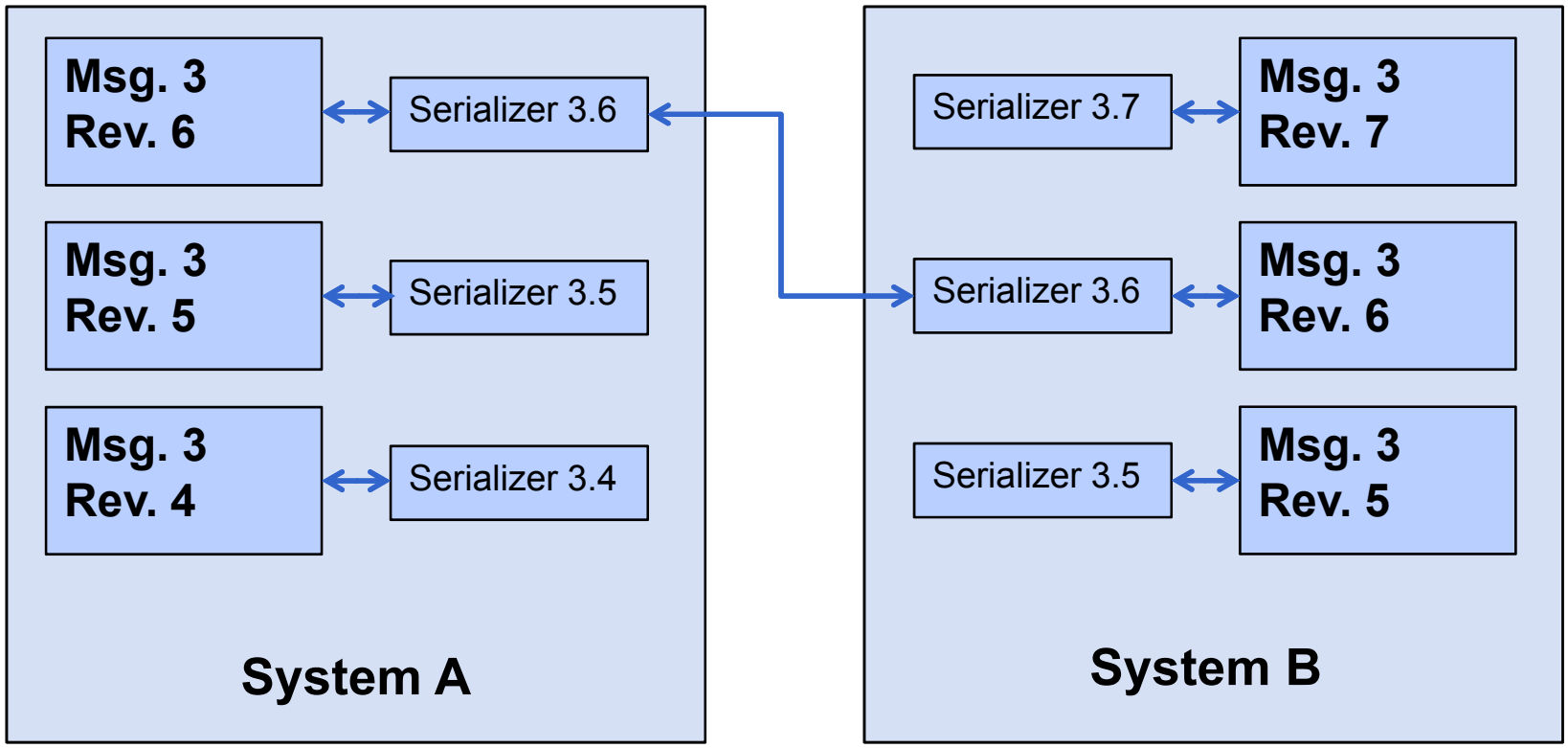
Navigation data goes everywhere

Stopping the Change Ripple



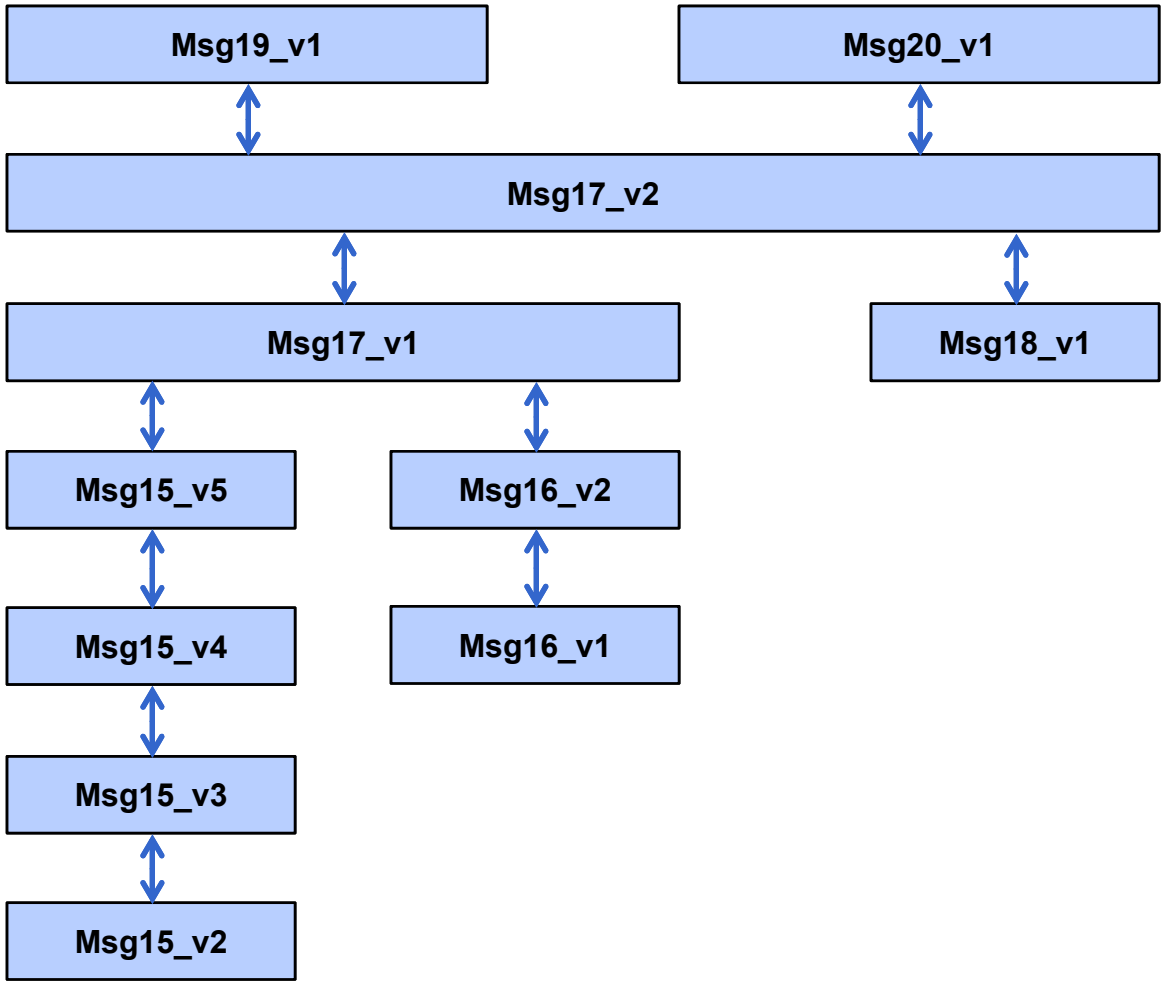
Map the Existing Interface to the Revised Interface
Don't force legacy code to *immediately* conform

Pub/Sub Messaging System:
Just add a version number

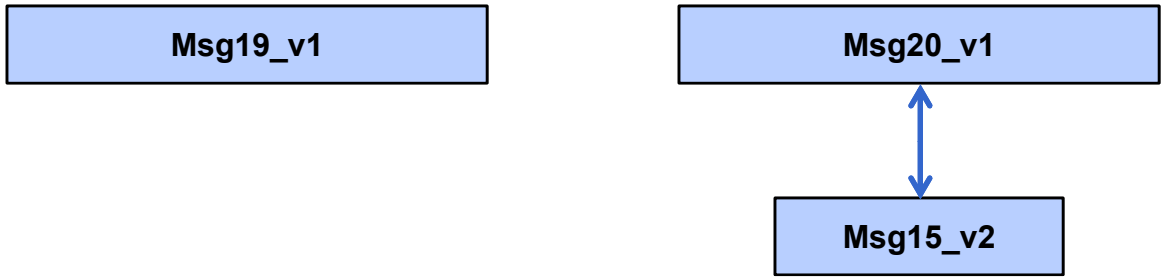


Negotiate to the Highest Common Revision

Revision Tree for Test Case



House Keeping



Unused messages can be deleted,
even if older legacy interfaces are still required

Use of templates simplifies message construction

```
template< class T, class P > class ElementCur : public ElBase<T,P>
{
private:
    T        data;

public:
    T&       get() {return data;};
    void     set(T& t) {data = t;};
    void     setValue(T t) {data = t;};
};
```

The ElementCur template allocated memory for the data.

The template provides the get and set functions.

```
class DllExport Msg15_v2 : public MsgBase
{
public:
    template< class T > class ElType
    {
    public:
        typedef ElementCur< T, Msg15_v2 > type;
    };

    Msg15_v2( MBmap& );

    ElType< long >::type  Alt;
    ElType< float >::type Lat;
    ElType< float >::type Lng;
};
```

typedef simplifies revision and masks complexity.

With C++11 compiler use *using*

The data elements are defined in terms of the template.

Create a new version of the Message

```
class DllExport Msg15_v3 : public MsgBase
{
public:
    template< class T > class ElType
    {
    public:
        typedef ElementCur< T, Msg15_v3 > type;
    };

    Msg15_v3( MBmap& );

    ElType< float >::type Lat;
    ElType< float >::type Lng;
    ElType< long >::type  Alt;

private:
    void init();
};

Msg15_v3::Msg15_v3( MBmap& ptr ) : MsgBase( 15, 3, ptr )
{
    peer[15] = this;
    older[15] = new Msg15_v2(peer);
    init();
};

void Msg15_v3::init()
{
    Msg15_v2* ptr2 = static_cast<Msg15_v2*>(older[15]);

    ptr2->Lat.init( this, &Msg15_v3::Lat );
    ptr2->Lng.init( this, &Msg15_v3::Lng );
    ptr2->Alt.init( this, &Msg15_v3::Alt );
};
```

Constructor instantiates older revision and initializes it

init() maps older revision data elements to current elements

Do housekeeping on older version

```
class DllExport Msg15_v2 : public MsgBase
{
public:
    template< class T > class ElType
    {
    public:
        typedef ElementPrev< T, Msg15_v3 > type;
    };

    Msg15_v2( MBmap& );

    ElType< long >::type  Alt;
    ElType< float >::type Lat;
    ElType< float >::type Lng;
};
```

Change the template type

Change the target type

ElementPrev template does the heavy work

```
template< class T, class P > class ElementPrev : public ElBase<T,P>
{
private:
    typedef typename P::ElType<T>::type P::* mPtr;
    typedef T& (P::*getFunc)();
    typedef void (P::*setFunc)( T t );

    T          data;
    P*         parent;
    mPtr       member_ptr;
    getFunc    member_getter;
    setFunc    member_setter;

public:
    void init( P* pIn, mPtr ptrIn, getFunc gfIn = NULL, setFunc sfIn = NULL )
    {
        parent = pIn;
        member_ptr = ptrIn;
        member_getter = gfIn;
        member_setter = sfIn;
    };

    T& get()
    {
        if (member_getter) return (parent->*member_getter)();
        else return (parent->*member_ptr).get();
    };

    void set(T& t)
    {
        if (member_setter) (parent->*member_setter)( t );
        else (parent->*member_ptr).set(t);
    };

    void setValue(T t)
    {
        if (member_setter) (parent->*member_setter)( t);
        else (parent->*member_ptr).setValue(t);
    };
};
```

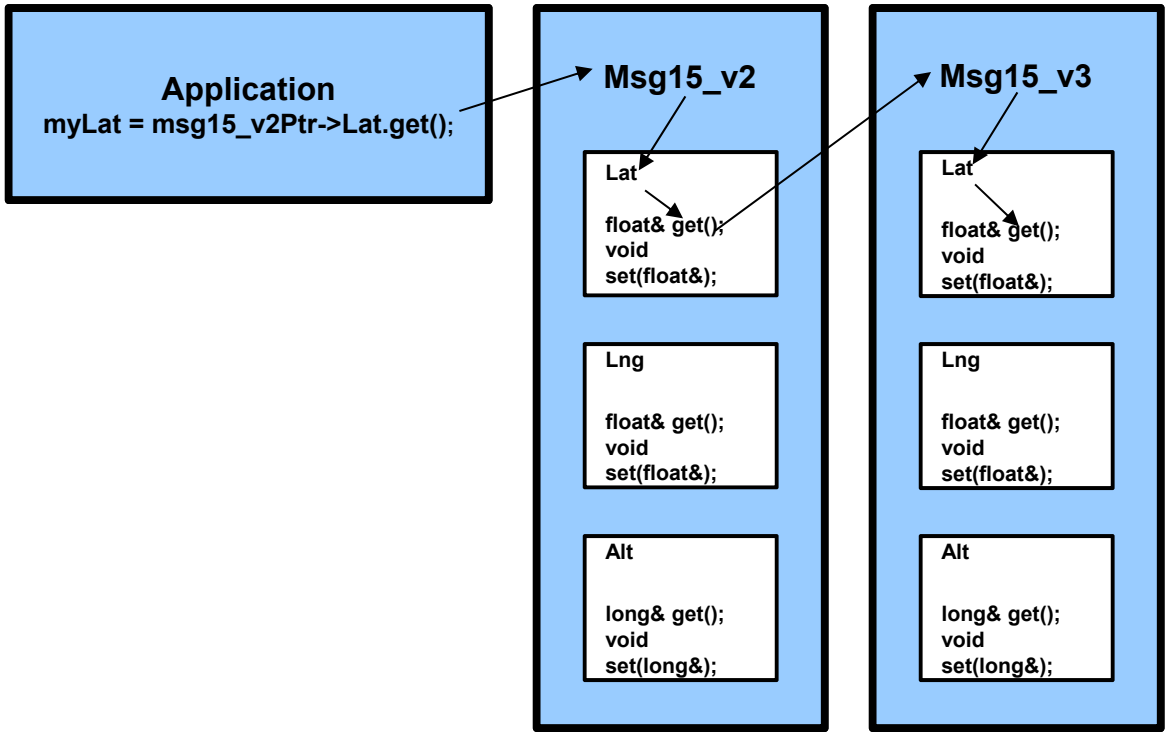
init() defines the instance and member being referenced. Optional function pointers for more complex relationships.

Get data from defined member, using a function if provided.

Set data in defined member by reference, using a function if provided

Set data in defined member by value, using a function if provided

Calling Sequence



Pointing in the right direction

```
// Static function, returns the correct type based on the requested type
MsgBase::getInstance(int msgNum)
{
    static MBmap tempMap;
    static bool firstTime = true;
    MsgBase* msg = NULL;

    if (firstTime)
    {
        tempMap.clear();
        firstTime = false;
    }
    switch (msgNum)
    {
    case 19:
        msg = new Msg19_v1( tempMap );
        break;
    case 15:
    case 16:
    case 17:
    case 18:
    case 20:
        msg = new Msg20_v1( tempMap );
        break;
    default:
        break;
    }
    return msg;
}

MsgBase* msgBase = MsgBase::getInstance( 15 );

Msg15_v3* msgV3 = static_cast<Msg15_v3*>(msgBase->getVersion(15, 3));
Msg15_v2* msgV2 = static_cast<Msg15_v2*>(msgBase->getVersion(15, 2));
```

The requested message type is mapped into the current hierarchy, returning the current parent type.

First, get an instance of the message tree. Then get pointers to the desired versions of the message type.

The cast may be static or dynamic. Failure may be NULL, null object or throw an exception

For More Details

gerry.tyra@lmco.com
gerry@planet-tyra.com

Visual Studio 2010 project available as a zip file at:
WWW.planet-tyra.com/oddsNends/softwarerevision.zip