How to hook in Temporality

The best approach to hook in the temporality feature is to hook ourselves into the EStore. Doing so has many benefits like begging able to turn the feature on or off. This will make development easier since we will each be able to concentrate on the temporality feature without being affecting other ones like the persistence feature. We can then compose them to get the complete set of features. We could do this through an inheritance structure but doing it using delegation is more flexible. Here's the idea.

Using a simple chain of responsibility similar to the io package in Java would do the trick. Here's an example:

```
public MyDynamicEStoreEObjectImpl() {
      eSetStore(
            new MySecurityEStoreImpl(
                  new MyTemporalityEStoreImpl(
                        new MyMemoryEStoreImpl())));

}
```

However using this simple approach might lead to lots of objects created needlessly. If this is found to be true we could also use this approach and have a single EStoreImpl keeping state information and delegate the processing to stateless EStoreImpl.

```
public MyDynamicEStoreEObjectImpl() {
      EStore store = new MyEStoreStateHolderImpl();
      store.add(MySecurityEStoreImpl.INSTANCE);
      store.add(MyTemporalityEStoreImpl.INSTANCE);
      store.add(MyMemoryEStoreImpl.INSTANCE);
      eSetStore(store);
}
```

When the data holder delegates to the processor it adds an extra parameter to the call eSet(eObject, feature, value, **MyContextualData**).

It is possible to use this same base class in a dynamically created EClass (an EClass created using the programmatic API). All you have to do is write your own factory that implement the EFactory interface to instantiate your EObjects using your own base class. To hook this factory into EMF you get the package you defined for your dynamic EClasses and set your custom factory on it.

Here's an example of overriding the basicCreate method. All we have to do is replace the instantiation of the base class the EFactoryImpl uses by the instantiation of our own base class.

```
// Register my own factory to create EStore backed DynamicEObjects
companyPackage.setEFactoryInstance(new EFactoryImpl() {
      @Override
      protected EObject basicCreate(EClass eClass) {
          return
             eClass.getInstanceClassName() == "java.util.Map$Entry" ?
               new MyEStoreEObjectImpl.BasicEMapEntry(eClass) :
               new MyEStoreEObjectImpl(eClass);
      }
});
```

With the constructor of this base class written like this

```
        public MyDynamicEStoreEObjectImpl() {
              eSetStore(new MyMemoryEStoreImpl());
        }
```

So the base class can be specified both in generated code as well as dynamically created EClasses.