Reducing Duplication

Tomáš Jelínek

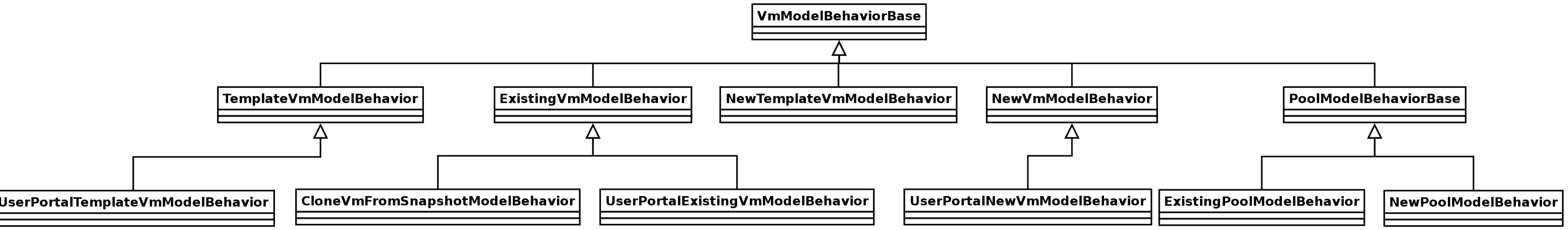
Software Engineer

March 5, 2013

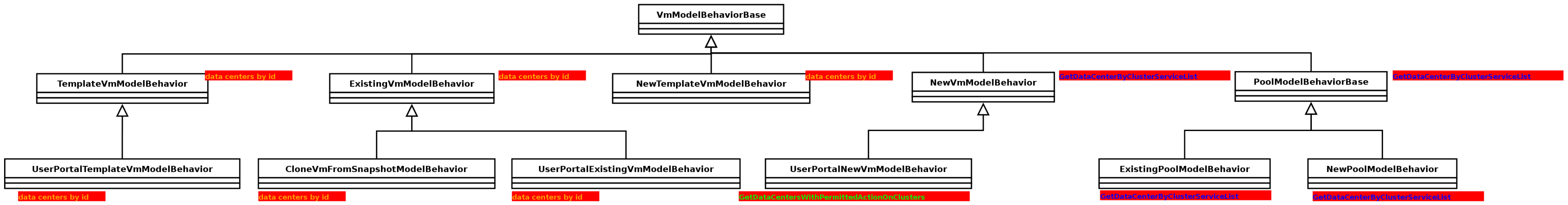
Topics covered

- Problem
- Possible solutions
- Builders
- Discussion

Problem 1


VmModelBehaviorBase
TemplateVmModelBehavior
ExistingVmModelBehavior
NewTemplateVmModelBehavior
NewVmModelBehavior
PoolModelBehaviorBase
UserPortalTemplateVmModelBehavior
CloneVmFromSnapshotModelBehavior
UserPortalExistingVmModelBehavior
UserPortalNewVmModelBehavior
ExistingPoolModelBehavior
NewPoolModelBehavior

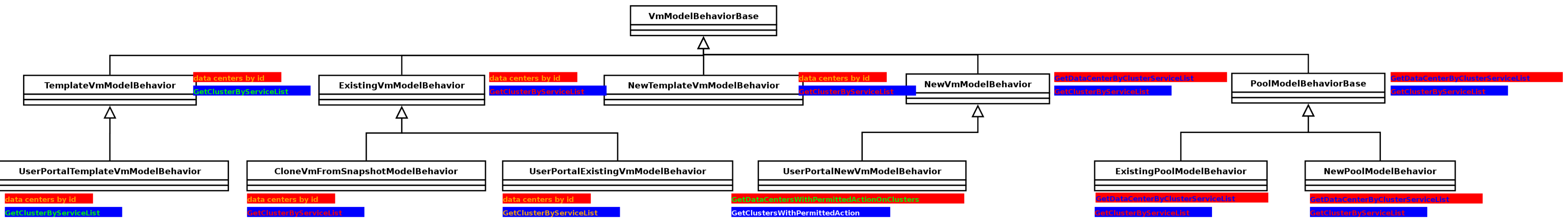
Problem 2


VmModelBehaviorBase
TemplateVmModelBehavior
data centers by id
ExistingVmModelBehavior
data centers by id
NewTemplateVmModelBehavior
data centers by id
NewVmModelBehavior
GetDataCenterByClusterServiceList
PoolModelBehaviorBase
GetDataCenterByClusterServiceList
UserPortalTemplateVmModelBehavior
data centers by id
CloneVmFromSnapshotModelBehavior
data centers by id
UserPortalExistingVmModelBehavior
data centers by id
UserPortalNewVmModelBehavior
GetDataCentersWithPermittedActionOnClusters
ExistingPoolModelBehavior
GetDataCenterByClusterServiceList
NewPoolModelBehavior
GetDataCenterByClusterServiceList

Legenda:

🟥 set data center to unitVmModel

Problem 3

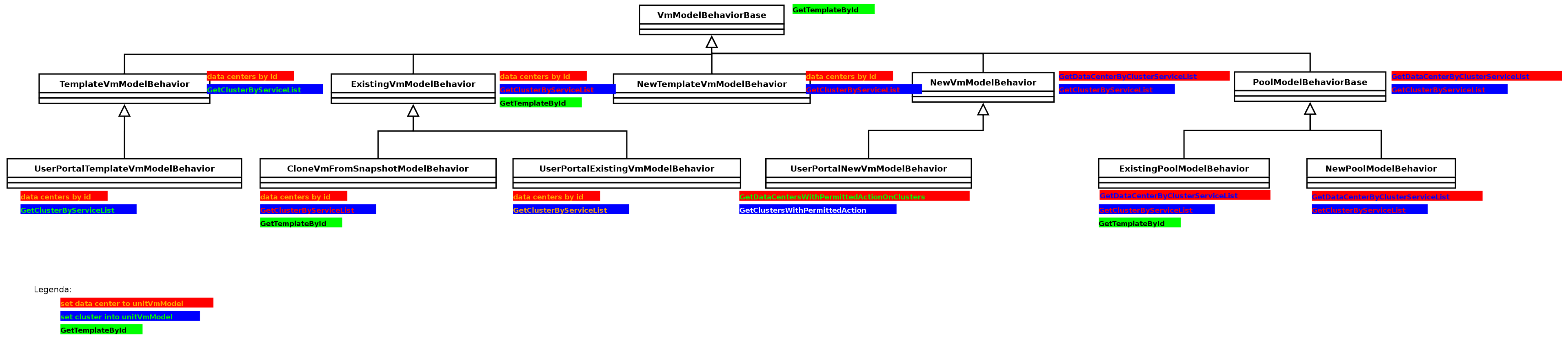


Legenda:

🟥 set data center to unitVmModel

🟦 set cluster into unitVmModel

Problem 4


VmModelBehaviorBase
GetTemplateById
TemplateVmModelBehavior
data centers by id
GetClusterByServiceList
ExistingVmModelBehavior
data centers by id
GetClusterByServiceList
GetTemplateById
NewTemplateVmModelBehavior
data centers by id
GetClusterByServiceList
NewVmModelBehavior
GetDataCenterByClusterServiceList
GetClusterByServiceList
PoolModelBehaviorBase
GetDataCenterByClusterServiceList
GetClusterByServiceList
UserPortalTemplateVmModelBehavior
data centers by id
GetClusterByServiceList
CloneVmFromSnapshotModelBehavior
data centers by id
GetClusterByServiceList
GetTemplateById
UserPortalExistingVmModelBehavior
data centers by id
GetClusterByServiceList
UserPortalNewVmModelBehavior
GetDataCentersWithPermittedActionOnClusters
GetClustersWithPermittedAction
ExistingPoolModelBehavior
GetDataCenterByClusterServiceList
GetClusterByServiceList
GetTemplateById
NewPoolModelBehavior
GetDataCenterByClusterServiceList
GetClusterByServiceList
Legenda:
set data center to unitVmModel
set cluster into unitVmModel
GetTemplateById

Problem – this is still simplified

- Far not all settings listed
- Also list models do some settings (e.g. ~3000 lines long VmListModel)
- In near future beside VM/Template/Pool also Instnace Type + Image will complicate this

Problem – Why?

- Cross cutting concerns of:
    - Different types like pool/template/vm
    - New vs Edit
    - User Portal vs Web Admin

Possible Solutions

- Inheritance
- AOP
- Use composition

Builders – My Requirements

- Support chaining
- Transparent support for async builders
- Support composition of builders (composite pattern)
- Simple usage
- Simple API

Builders 1

- Parent of the builders is the Builder

```
public interface Builder<S, D> {

    void build(S source, D destination, BuilderList<S, D> rest);
}
```

- Parent of all sync builders is BaseSyncBuilder

```
public abstract class BaseSyncBuilder<S, D> implements Builder<S, D> {

    @Override
    public void build(S source, D destination, BuilderList<S, D> rest) {
        build(source, destination);

        rest.head().build(source, destination, rest.tail());
    }

    protected abstract void build(S source, D destination);

}
```

Simple Sinc Builder

```java
class SimpleSyncBuilder extends BaseSyncBuilder<String, StringBuffer> {

    @Override
    protected void build(String source, StringBuffer destination) {
        destination.append(source.charAt(0));
    }

}
```

Simple Async Builder

```
class SimpleAsync implements Builder<String, StringBuffer> {

    @Override
    public void build(String source, StringBuffer destination,
                     BuilderList<String, StringBuffer> rest) {
        AsyncDataProvider.GetSecondLetter(new AsyncQuery(getModel(),
                new INewAsyncCallback() {
                    @Override
                    public void OnSuccess(Object target, Object returnValue) {
                        destination.append(returnValue);
                        rest.head().build(source, destination, rest.tail());
                    }
                }),source);
    }


}
```

Simple Usage

```
StringBuffer someResult = new StringBuffer();

BuilderExecutor<String, StringBuffer> executor =
        new BuilderExecutor<String, StringBuffer>(
                new SimpleSyncBuilder(),
                new SimpleAsyncBuilder()
        );



executor.build("ab", someResult);
```

Waiting for Result

```
BuilderExecutor<String, StringBuffer> executor =
        new BuilderExecutor<String, StringBuffer>(
               new BuilderExecutionFinished<String, StringBuffer>(){

                   @Override
                   public void finished(String source, StringBuffer destination) {
                       // done
                   }

               },
               new SimpleSyncBuilder(),
               new SimpleAsyncBuilder()
        );

        StringBuffer res = new StringBuffer();

        executor.build("ab", res);
```

Composite Builder

- Just a normal builder
- Can be chained to the rest
- For reusing related builders

```
new CompositeBuilder<String, StringBuffer>(
    SimpleSyncBuilder(),
    SimpleAsyncBuilder()
);
```

Advantages

- simple (the whole infrastructure just 4 classes and 1 interface)
- hides the difference between sync and async builders (reduces nested anonymous classes)
- modularize cross cutting logic
- naming of small peaces of logic
- makes the uicommon more testable
- makes uicommon more readable

Disadvantages

- lots of small classes
- new approach introduced

Thank you!