GwtCommon Module

14 Feb 2012

Vojtech Szöcs
Red Hat

How it all started...

- Let there be WebAdmin
- We embraced GWT MVP-style development using GWTP framework
- We learned how to integrate and use UiCommon models in our infrastructure

GwtCommon introduction

- We established new **concepts** that proved to be useful in WebAdmin
  - Model providers for managing UiCommon models within Guice/GIN context
  - Customized Editor Driver support for UiCommon models, including Editor widgets for those models
  - Standard infrastructure related events UserLoginChange event, UiCommonInit event, etc.
  - Auto-login using dynamic host page

GwtCommon introduction

- Besides those concepts, we have also written
  - Integration with UiCommon, so that we can use its models correctly via GIN-managed model providers
  - Custom widgets like model-bound action table, which has its buttons bound to model commands

But...

- WebAdmin is not the only Frontend application

Goals behind GwtCommon

- To create **reusable GWT module** that contains
  - Common infrastructure classes, reflecting our main concepts
  - UiCommon integration classes
  - Common features and behavior, encapsulated within reusable system components (e.g. model-bound dialog presenters and views)
  - Abstract classes for common widgets (e.g. action table), given that each project will customize their concrete UI

Impacts of using GwtCommon in WebAdmin

- `WebAdmin.gwt.xml` is shorter since we inherit `GwtCommon.gwt.xml`
  - GIN, GWTP MVP, UiCommonWeb, custom generators

- WebAdmin infrastructure classes usually extend base ones defined in GwtCommon
  - Reduced boilerplate code

Impacts of using GwtCommon in WebAdmin

```java
public class SystemModule extends BaseSystemModule {

    @Override
    protected void configure() {
        bindInfrastructure();
        bindConfiguration();
    }

    void bindInfrastructure() {
        bindCommonInfrastructure();
        bind(ApplicationInit.class).asEagerSingleton();
        bind(InternalConfiguration.class).asEagerSingleton();
    }

    void bindConfiguration() {
        bindPlaceConfiguration(ApplicationPlaces.loginPlace,
                ApplicationPlaces.virtualMachineMainTabPlace);
        bindResourceConfiguration(ApplicationConstants.class,
                ApplicationMessages.class,
                ApplicationResources.class,
                ApplicationTemplates.class);
    }

}
```

Moving more stuff to GwtCommon

- All the common infrastructure and UiCommon integration is already there

- For common features/behavior/widgets
  - Only the reasonable intersection between WebAdmin and UserPortal
  - We can move more of these from WebAdmin, **in case they are needed**
  - However, as with every “common” library, we should extract only things which we will actually use >1 times

Moving more stuff to GwtCommon

- Having a common module means more responsibility
  - Multiple applications use GwtCommon
  - GwtCommon modifications should not introduce regressions in existing Frontend projects (successful build is not enough)
  - GwtCommon should not enforce changes in other projects, just because some feature is required by one particular project

Some ideas about code reuse

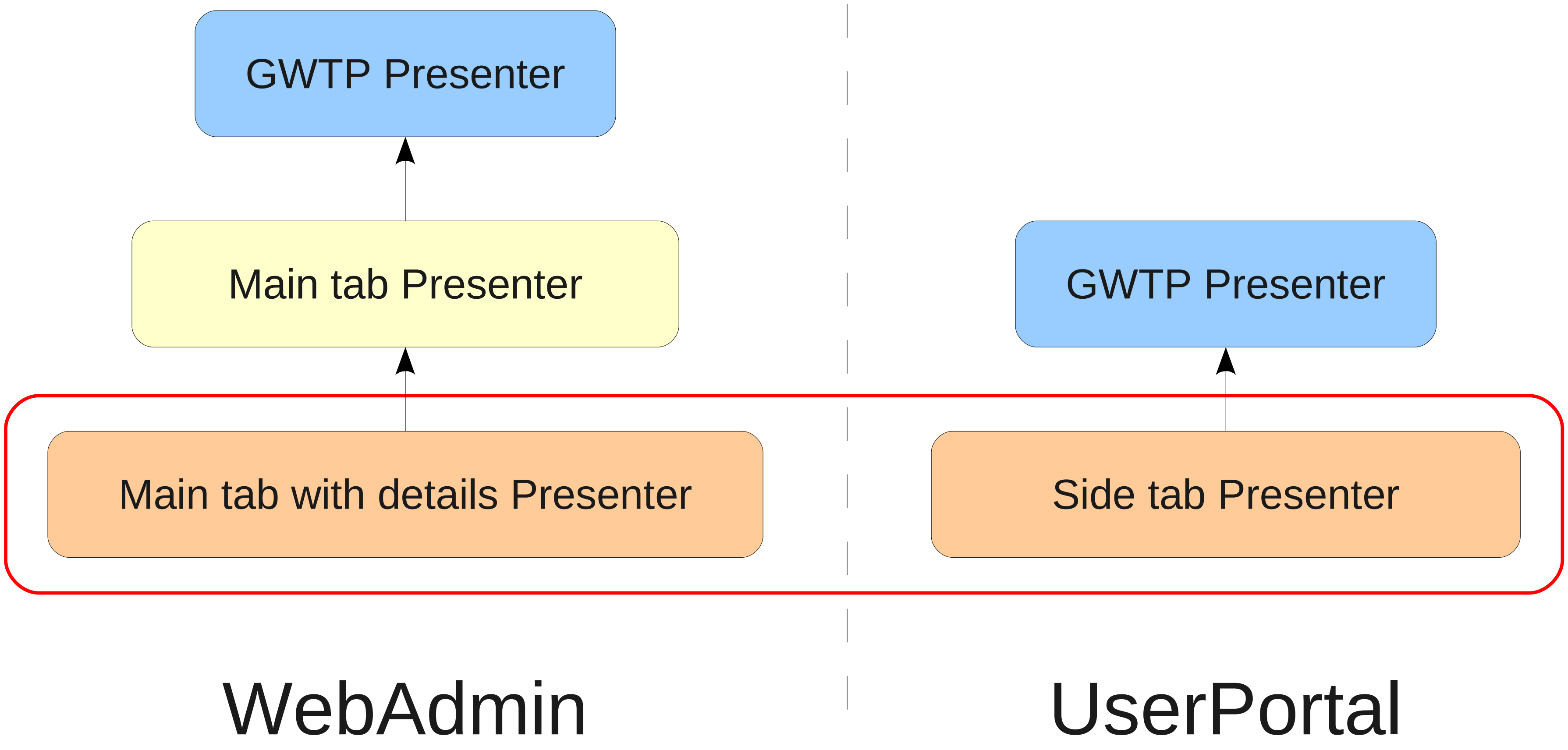
- “Never write the same code twice”
- What if two blocks of code are similar, but still slightly different?
  - Create some parameters!
- What if you need something more in some case?
  - Add conditional logic to decide what to do!
- What if you can't fix one caller without breaking another caller?
  - Add another layer of abstraction!

Some ideas about code reuse

- The resulting code is often hard to understand, maintain and nearly impossible to extend

- *It does not make sense to try to reuse everything just because the code looks similar*
- *Good reusable code is simple and easy to understand*

Code reuse pitfall example



GwtCommon UI reuse

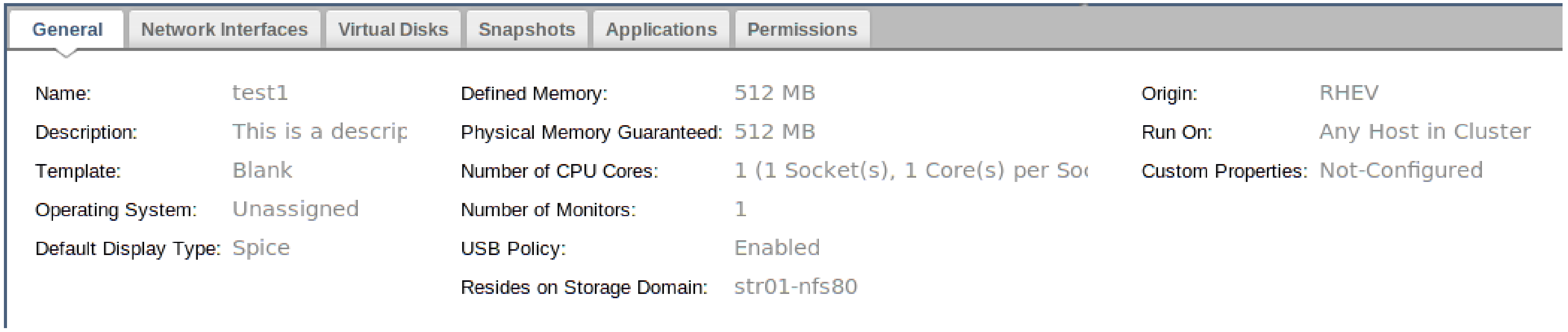
- Originally, there were no resources in GwtCommon, e.g. images, *.css files, *.ui.xml files
- GwtCommon provides UI (widget) abstractions that should be implemented in concrete environments, e.g. `AbstractActionTable` vs. `SimpleActionTable`
- Turns out that some parts of UI are (nearly) identical for multiple applications
  - Main and sub tab UI (forms and tables)
  - Dialog UI

GwtCommon UI reuse

- Common UI should be moved into GwtCommon
    - GWTP Views are architectural components, specific to each GWTP application
    - UI reuse should focus on UI only (widget level)

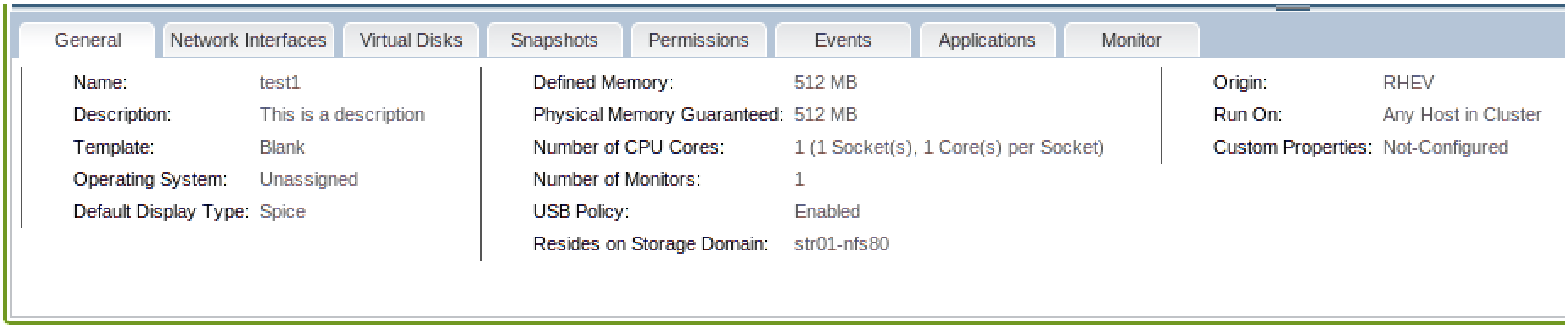
GwtCommon UI reuse

WebAdmin VM General sub tab UI



Name:	test1	Defined Memory:	512 MB	Origin:	RHEV
Description:	This is a descrip	Physical Memory Guaranteed:	512 MB	Run On:	Any Host in Cluster
Template:	Blank	Number of CPU Cores:	1 (1 Socket(s), 1 Core(s) per So	Custom Properties:	Not-Configured
Operating System:	Unassigned	Number of Monitors:	1		
Default Display Type:	Spice	USB Policy:	Enabled		
		Resides on Storage Domain:	str01-nfs80		

UserPortal VM General sub tab UI



Name:	test1	Defined Memory:	512 MB	Origin:	RHEV
Description:	This is a description	Physical Memory Guaranteed:	512 MB	Run On:	Any Host in Cluster
Template:	Blank	Number of CPU Cores:	1 (1 Socket(s), 1 Core(s) per Socket)	Custom Properties:	Not-Configured
Operating System:	Unassigned	Number of Monitors:	1		
Default Display Type:	Spice	USB Policy:	Enabled		
		Resides on Storage Domain:	str01-nfs80		

GwtCommon UI reuse

- Driven by UserPortal common UI, we will
  - Extract UI code into widgets that are Editors of corresponding UiCommon models
  - Reuse those model-bound widgets

- However, we should extract only stuff that will be used more than once
  - Trying to extract every piece of UI into GwtCommon has no real value except wasted time and energy
  - Let's try to be lean and follow YAGNI principle

That's all folks