

# Erstellung eines Mockbasierten Testframeworks für Saros

*Preview*

Voraussetzungen

Saros

JUnit

Dependency Injection

Testframework

Zum Erstellen von Unit-Tests

Etablierung einer DSL

Tests Schreiben

- \* Schnelles, Einfaches Erstellen von Tests
- \* Einfaches, Schnelles Ausführen
- \* Keine Setup notwendig

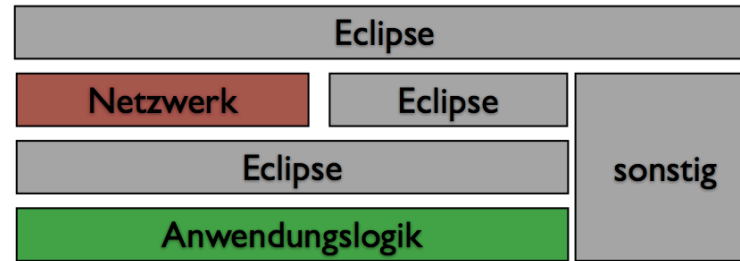
Prämissen

Code unverändert lassen

keine Magie

leicht verständlich

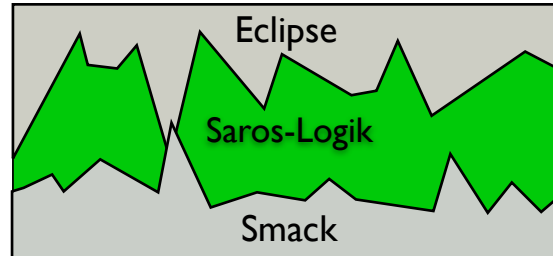
- \* Schnellläufig
- \* Einfach
- \* Schnelles Erstellen von Tests

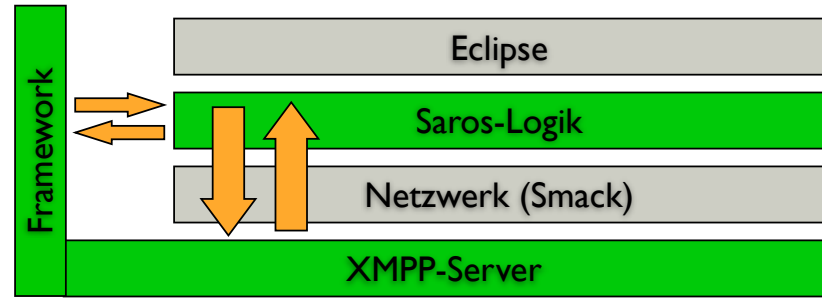


Eclipse

Saros-Logik

Netzwerk (Smack)







```
Saros saros = new Saros();  
saros.connect();  
assertTrue(saros.isConnected());
```

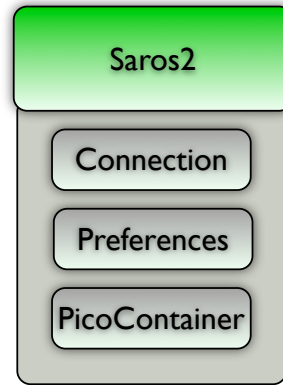
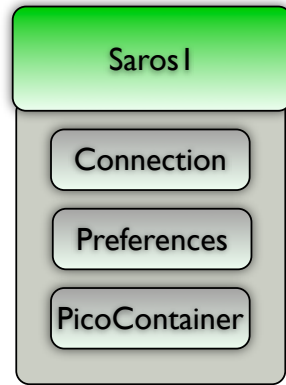
```
Saros saros1 = new Saros();  
Saros saros2 = new Saros();  
  
saros1.connect();  
saros2.connect();  
  
assertTrue(saros1.isConnected());  
assertTrue(saros2.isConnected());
```

\* wir wollen ja auch die Kommunikation zwischen zwei Exemplaren testen können

```
Saros saros1 = new Saros();  
Saros saros2 = new Saros();  
  
saros1.connect();  
saros2.connect();  
  
assertTrue(saros1.isConnected());  
assertTrue(saros2.isConnected());  
  
assertNotSame(saros1, saros2);
```

```
/**  
 * Create the shared instance.  
 */  
public Saros() {  
    // Only start a DotGraphMonitor if asserts are enabled (aka debug mode)  
    assert (dotMonitor = new DotGraphMonitor()) != null;  
    setInitialized(false);  
    setDefault(this);  
}  
  
public static void setDefault(Saros newPlugin) {  
    Saros.plugin = newPlugin;  
}
```

\* wir wollen ja auch die Kommunikation zwischen zwei Exemplaren testen können



```
PicoBuilder picoBuilder = new PicoBuilder(new CompositeInjection(
    new ConstructorInjection(), new AnnotatedFieldInjection()))
    .withCaching().withLifecycle();

/*
 * If given, the dotMonitor is used to capture an architecture diagram
 * of the application
 */
```

```
// Core Managers
this.container.addComponent(ChangeColorManager.class);
this.container.addComponent(ConsistencyWatchdogClient.class);
this.container.addComponent(ConsistencyWatchdogServer.class);
this.container.addComponent(DataTransferManager.class);
this.container.addComponent(DiscoveryManager.class);
this.container.addComponent(EditorAPI.class);
this.container.addComponent(EditorManager.class);
this.container.addComponent(ErrorLogManager.class);
this.container.addComponent(FeedbackManager.class);
this.container.addComponent(JDTFacade.class);
this.container.addComponent(LocalPresenceTracker.class);
this.container.addComponent(MUCManager.class);
this.container.addComponent(MUCManagerSingletonWrapperChatView.class);
this.container.addComponent(PingPongCentral.class);
this.container.addComponent(PreferenceManager.class);
this.container.addComponent(PreferenceUtils.class);
this.container.addComponent(PresenceManager.class);
this.container.addComponent(RosterTracker.class);
this.container.addComponent(SarosRosterTracker.class);
this.container.addComponent(SarosUI.class);
this.container.addComponent(SarosSessionManager.class);
this.container.addComponent(SessionViewOpener.class);
this.container.addComponent(SharedResourcesManager.class);
this.container.addComponent(SkypeManager.class);
this.container.addComponent(StatisticManager.class);
this.container.addComponent(StopManager.class);
this.container.addComponent(StreamServiceManager.class);
this.container.addComponent(AudioServiceManager.class);
this.container.addComponent(MixerManager.class);
this.container.addComponent(SubscriptionListener.class);
this.container.addComponent(UndoManager.class);
this.container.addComponent(UndoSheet.class);
```

```
@Inject
protected Saros saros;

@Inject
protected RosterTracker rosterTracker;
```

```
Saros saros1 = new Saros();  
Saros saros2 = new Saros();  
  
saros1.connect();  
saros2.connect();  
  
assertTrue(saros1.isConnected());  
assertTrue(saros2.isConnected());  
  
assertNotSame(saros1, saros2);
```

\* der weg war geebnet

## Statischen Methoden vs. Kontext

```
/**
 * Injects dependencies into the annotated fields of the given object. This
 * method should be used for objects that were created by Eclipse, which
 * have a different life cycle than the Saros plug-in.
 */
public static synchronized void injectDependenciesOnly(Object toInjectInto) {
    checkInitialized();

    ChildContainer dummyContainer = plugin
        .getComponent(ChildContainer.class)
        .dummyContainer.reinject(toInjectInto);
    plugin.container.removeChildContainer(dummyContainer);
}

public RosterView() {
    super();

    // Make sure that we get all dependencies
    Saros.injectDependenciesOnly(this);
}

@Inject
protected Saros saros;
@Inject
protected SarosUI sarosUI;
@Inject
protected SarosSessionManager sessionManager;
@Inject
protected StatisticManager statisticManager;
@Inject
protected ErrorLogManager errorLogManager;
@Inject
protected PreferenceUtils preferenceUtils;
@Inject
protected DataTransferManager dataTransferManager;
```

Unclassified usage (34 usages)

- Saros (34 usages)
  - de.fu\_berlin.inf.dpp.project.Internal (1 usage)
    - ResourceChangeValidator (1 usage)
  - de.fu\_berlin.inf.dpp.ui (6 usages)
    - AdvancedPreferencePage (1 usage)
    - CommunicationPreferencePage (1 usage)
    - FeedbackPreferencePage (1 usage)
    - GeneralPreferencePage (1 usage)
    - RosterView (1 usage)

\* die Komponenten müssten sich mit entsprechenden PicoContainer initialisieren

SarosContext

Kapselt PicoContainer

bietet reinject

bietet initComponents





## Benutze SarosContext statt PicoContainer

+ Util.getPlatformInfo());	583	433	// Remove the Bundle if an in
// Remove the Bundle if an instance of it was already registered	584	434	sarosContext.removeComponent(B
container.removeComponent(Bundle.class);	585	435	sarosContext.addComponent(Bund
container.addComponent(Bundle.class, getBundle());	586	436	
	587	437	// Make sure that all componen
	588	438	// instantiated
// Make sure that all components in the container are	589	439	sarosContext.getComponents(Ob
// instantiated	590	440	
container.getComponents(Object.class);	591	441	this.sessionManager = sarosCon
	592	442	

## Mache SarosContext an den kontextsensitiven Stellen zugänglich (z.B. SarosSessionManager)

```
@Inject
protected SarosContext sarosContext;
```

## Ersetze „Saros.injectDependenciesOnly“

public RosterView() {	614	613	public RosterView() {
super();	615	616	super();
// Make sure that we get all dependencies injected	617	618	// Make sure that we get all depend
Saros.injectDependenciesOnly(this);	619	620	SarosPluginContext.initComponent(this
dataTransferManager.getTransferModeDispatch().add(transferModeLister	620	621	dataTransferManager.getTransferModeDis
}	621	622	}
	622	623	

Schreiben von Tests

Erstellen von XMPP-Account

Hinzufügen von Buddies

Session starten

In Session einladen

```
if (!Saros.isTestSaros()) {  
    // SWT-, Eclipse-Code etc.  
}
```

```
TestSaros saros =  
  buildSaros()  
  .withUserName("alice")  
  .andReturn();
```

```
TestSaros saros =  
  buildSaros()  
  .withTestWorkspace()  
  .withTestProject()  
  .andReturn();
```

```
subscribeUsers(asList(saros1, saros2));
```

```
final ExpectedPacket expectedPacket =  
  getExpectedPacketOnSaros(sarosInstance)  
  .containsData("workspace_alice")  
  .withTransferdescription(description)  
  .andReturn();
```

```
waitFor(new TCondition("received packet") {  
  boolean isFullfilled() {  
    return expectedPacket.wasReceived();  
  }  
});
```

Refactoring der Stellen  
Saros.isTestSaros

Starten/Stoppen XMPP-Server

Funktionalitäten für Senden/  
Empfangen von Aktivitäten

Weitere Standardfunktionalitäten

Das was fehlt erledigen

Stückweise reviewn lassen und  
committen

Unterstützung leisten bei  
Verwendung

Implementierung von Tests aus  
TestLink

3 kleine Fallstudien

\* bei reviews werde ich mögliche Tests skizzieren

*Demo*