Introduction to GRMG-Heartbeats

This overview addresses developers wishing to use GRMG and in particular the modular features which facilitate the implementation of GRMG and Heartbeat scenarios in a typical Web server situation.

I. Principles of GRMG:

The goal of GRMG (Generic Request and Message Generator) and Heartbeats is to test a chain of technical components for availability, of business processes or a mixture of both, in a J2EE server or in the Web AS This chain is called ***Scenario***. Thus GRMG can be used for technical as well as application monitoring.

II. GRMG consists of two parts:

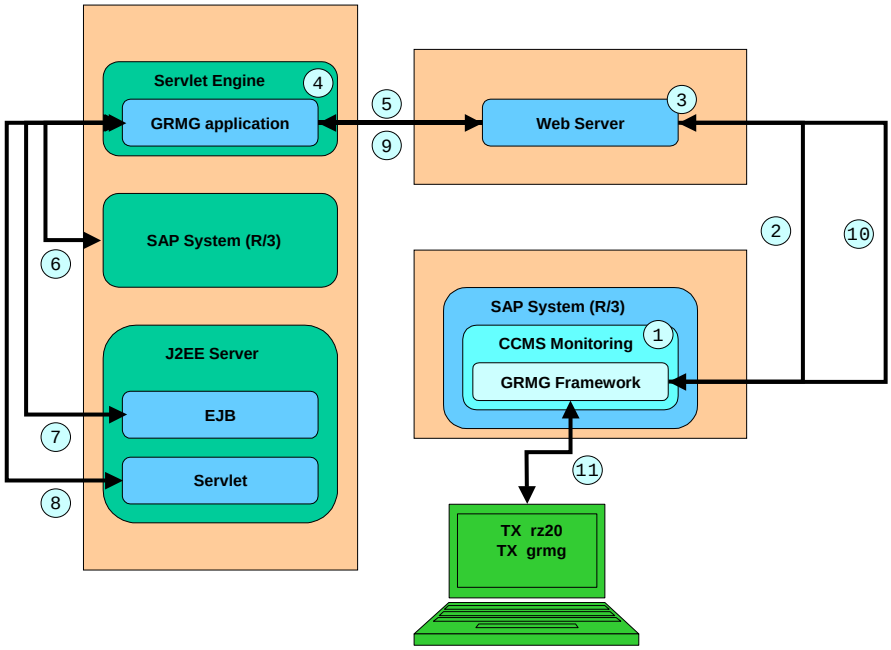
1. GRMG Framework:
    - Is embedded in CCMS.
    - is maintained by SAP CCMS administrator/developer
    - sends test request against the GRMG Application
    - receives a response from GRMG Application.

2. GRMG Application:
    - Is a callable application (built-in in JSP, Servlet, iView, etc.)
    - has well-defined interfaces to receive GRMG requests and to send GRMG responses.

GRMG request and GRMG response are supplied in well-formed XML documents (see below).

III. Typical Web server situation:

1. Web AS transfers GRMG request as HTTP POST request to a servlet or iView on a servlet engine or PRT resp.
2. This request will be received and processed by a ***GRMG application*** (JSP, Servlet, iView) on a J2EE Server, Web Server or PRT.
3. The GRMG application tests the different technical components or business processes (→ ***GRMG Scenario***): e.g. availability of EJBs, Web pages, DB, SAP Systems (R/3), etc.
4. The results of the different checks are collected by the GRMG application and assembled in a GRMG response object.
5. The GRMG response is transferred back to the GRMG framework – typically the output stream of the GRMG response may be uploaded into the HttpServletResponse of the servlet or iView.
6. The ***GRMG framework*** in the CCMS interprets the response and displays the results in the monitoring tree of CCMS.



The figure above provides an example of a GRMG test scenario. The GRMG Framework periodically calls a GRMG application that tests the availability of a chain of different software components (such as J2EE Servers or Web/Application Servers, as of SAP Web Application Server 6.20) or technical functions. The GRMG application is contained (or called) for instance in a Servlet or iView (EnterprisePortal).

In a typical Web server scenario as depicted in the figure above, the GRMG Framework within the CCMS (1) periodically sends a GRMG request (2) to the central Web Server (3) which in turn transfers to the Servlet Engine (4) the GRMG request by an HTTP POST request (5). The tests for the different technical components or business process steps are implemented in the GRMG Application. The GRMG application checks the availability of a SAP System (6), of an EJB (7) and of a Servlet (8). The results of the different checks are collected by the GRMG Application and combined in a GRMG response. The GRMG reponse is transferred back (9) to the GRMG Framework (10) where it is accordingly interpreted and displayed in a CCMS monitoring tree (11).

IV. GRMG Application:

In order to build a GRMG application and to customize the GRMG framework the following steps need to be done. We confine ourselves to the above-mentioned Web server situation.

i. Implement the GRMG scenario.
ii. Include the scenario in a servlet or iView, and deploy the application to the J2EE or servlet engine.
iii. Create a GRMG customizing XML file (well-formed) and upload the file to the SAP system which monitors the scenarios. Start the periodical scenario testing on the SAP system by transaction “grmg”.
iv. Monitor the results of the scenario/heartbeat tests in the SAP system using transaction “rz20”.

Implementation of the GRMG scenario:

The scenario may be implemented either by directly using the GRMG base classes, or more conveniently – in case of a Web application or PRT component – using the extended GRMG classes `ScenarioDevice`, `ScenarioPanel`, and `GRMFactory`:

- extend abstract class `ScenarioDevice` by implementing method `runScenario`
- Collect all `ScenarioDevices` in an array or in a `ScenarioPanel` and hand over to a `GRMFactory`
- implement a servlet or iView (doPost or doContent resp.) and start scenario test by calling the method `testScenario` of the `GRMFactory` within the servlet or iView. The GRMFactory fabricates the `GrmgResponse` object and eventually sends it back to CCMS.

Below is an explicit example in which a URL connection and an R/3 connect will be deployed and tested. Within the framework provided by GRMG in particular the implementation of the iView and the servlet in which the scenarios will be tested is very simple.

1. Implementation of `ScenarioDevice` defining the scenario process.

```
package test.scenario.samples;

import java.io.*;
import java.net.*;
import com.sap.mw.jco.JCO;
import com.sap.util.monitor.grmg.*;

public class ScenarioDeviceSample extends ScenarioDevice {

  public void runScenario(){

   GrmgComponent gcompWeb = getComponent("TestWeb");
   GrmgComponent gcompR3 = getComponent("TestR3");

   // test the various components of the scenario

   try{
    URL conn = new URL(gcompWeb.getPropertyByName("url").getValue());
    HttpURLConnection hconn =
      (HttpURLConnection) conn.openConnection();

     if (hconn.getResponseCode() == HttpURLConnection.HTTP_NOT_FOUND)
      setComponentData(gcompWeb, "ERROR","255","RT","001","",
                       "","","","Web page not found");
     else
      setComponentData(gcompWeb, "OKAY","000","RT","003","",
                       "","","","seems to be ok");
   }
   catch(Exception e){
    setComponentData(gcompWeb, "ERROR","255","RT","002",
                     e.getMessage(),"","","",e.getMessage());
   }

   try{
    JCO.createClient(
      gcompR3.getPropertyByName("SAPClient").getValue(),
      gcompR3.getPropertyByName("userid").getValue(),
      gcompR3.getPropertyByName("password").getValue(),
```

```
        gcompR3.getPropertyByName("language").getValue(),
        gcompR3.getPropertyByName("host name").getValue(),
        gcompR3.getPropertyByName("system_number").getValue()
        ).connect();
      setComponentData(gcompR3,"OKAY","001","RT","004",
                       "","","","","Connected!");
    }
    catch(Exception e){setComponentData(gcompR3,"ERROR","001","RT",
                        "004","","","","","Not connected. " +
                        e.getMessage());
    }
  }
}
```

2. Including the scenario test in an iView:

```
package test.scenario.samples;

import com.sap.util.monitor.grmg.GRMFactory;
import com.sapportals.portal.prt.component.AbstractPortalComponent;
import com.sapportals.portal.prt.component.IPortalComponentRequest;
import com.sapportals.portal.prt.component.IPortalComponentResponse;

public class ScenarioTestView extends AbstractPortalComponent{

 protected void doContent(IPortalComponentRequest request,
                          IPortalComponentResponse response){

   GRMFactory.testScenario(new ScenarioDeviceSample(),
                           request,response);
 }
}
```

3. Alternatively, including the scenario test in a servlet:

```
package test.scenario.samples;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.sap.util.monitor.grmg.GRMFactory;

public class ScenarioTestServlet extends HttpServlet {

public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
              throws ServletException, IOException {

  GRMFactory.testScenario(new ScenarioDeviceSample(),
                          request, response);
 }
}
```

4. ScenarioDevices are modular

This means that several `ScenarioDevice` objects can be composed into a single Scenario.

The previous example has two test components which may be implemented in two different `ScenarioDevice` classes:

i. ScenarioDevice for web page test:

```
package test.scenario.samples;

import java.io.*;
import java.net.*;
import com.sap.util.monitor.grmg.*;

public class ScenarioDeviceSample1 extends ScenarioDevice{

 public void runScenario(){

  GrmgComponent gcompWeb = getComponent("TestWeb");

  try{
   URL conn = new URL(gcompWeb.getPropertyByName("url").getValue());
   HttpURLConnection hconn=
      (HttpURLConnection)conn.openConnection();

   if(hconn.getResponseCode() == HttpURLConnection.HTTP_NOT_FOUND){
      setComponentData(gcompWeb, "ERROR", "255", "RT",
                       "001", "", "", "", "", "Web page not found");
   }
   else
    setComponentData(gcompWeb, "OKAY", "000", "RT", "003", "", "",
                     "", "", "seems to be ok");
   }

  catch(Exception e){
   setComponentData(gcompWeb, "ERROR", "255", "RT", "002",
                    e.getMessage(), "", "", "", e.getMessage());
  }
 }
}
```

ii. ScenarioDevice for R/3 connect test:

```
package test.scenario.samples;

import java.net.*;
import com.sap.mw.jco.JCO;
import com.sap.util.monitor.grmg.*;

public class ScenarioDeviceSample2 extends ScenarioDevice {

 public void runScenario(){

  GrmgComponent gcompR3 = getComponent("TestSAP");

  try{
   JCO.createClient(
    gcompR3.getPropertyByName("SAPClient").getValue(),
    gcompR3.getPropertyByName("userid").getValue(),
    gcompR3.getPropertyByName("password").getValue(),
    gcompR3.getPropertyByName("language").getValue(),
    gcompR3.getPropertyByName("host name").getValue(),
    gcompR3.getPropertyByName("system_number").getValue()).connect();

   setComponentData(gcompR3,"OKAY","001","RT",
                    "004","","","","","Connected!");
  }
  catch (Exception e){
   setComponentData(gcompR3, "ERROR","001","RT","004","",
                    "","","","Not connected. " + e.getMessage());
  }
 }
}
```

iii. Test both `ScenarioDevices` in servlet:

```
package test.scenario.samples;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.sap.util.monitor.grmg.*;

public class MultipleScenarioDevicesServlet extends HttpServlet{

 public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException{

  GRMFactory.testScenario(new ScenarioDevice[]
                          {new ScenarioDeviceSample1(),
                           new ScenarioDeviceSample2()},
                          request,response);
 }
}
```

or similarly in an iView:

```
package test.view;

import com.sap.util.monitor.grmg.*;
import com.sap.portals.heartbeats.samples.*;
import com.sapportals.portal.prt.component.AbstractPortalComponent;
import com.sapportals.portal.prt.component.IPortalComponentRequest;
import com.sapportals.portal.prt.component.IPortalComponentResponse;

public class TestView extends AbstractPortalComponent{

 protected void doContent(IPortalComponentRequest request,
                          IPortalComponentResponse response) {

  GRMFactory.testScenario(
   new ScenarioDevice[] {new ScenarioDeviceSample1(),
                         new ScenarioDeviceSample2()},
   request,response);
 }
}
```

or (alternatively e.g. in an iView) use a `ScenarioPanel`:

```
protected void doContent(IPortalComponentRequest request,
                          IPortalComponentResponse response) {

 ScenarioPanel scpan = new ScenarioPanel();
 scpan.addScenarioDevice(new ScenarioDeviceSample1());
 scpan.addScenarioDevice(new ScenarioDeviceSample2());

 GRMFactory.testScenario(scpan,request,response);
}
```

or (alternatively e.g. in an iView) use a several `ScenarioPanels`:

```
protected void doContent(IPortalComponentRequest request,
                          IPortalComponentResponse response) {

 ScenarioPanel scpan1 = new ScenarioPanel();
 scpan1.addScenarioDevice(new ScenarioDeviceSample1());

 GRMFactory.testScenario(scpan1,request);
```

```
 ScenarioPanel scpan2 = new ScenarioPanel();
 scpan2.addScenarioDevice(new ScenarioDeviceSample2());

 GRMFactory.testScenario(scpan2,request,response);
}
```

or (alternatively e.g. in an iView) use a `ScenarioPanel` to load `ScenarioDevices` dynamically:

```
protected void doContent(IPortalComponentRequest request,
                         IPortalComponentResponse response){

   ScenarioPanel scpan = new ScenarioPanel();

   try{
    URL url = new File(<location of file>).toURL();

    scpan.addScenarioDevice(url,
         "com.sap.portals.heartbeats.samples.ScenarioDeviceSample1");
    scpan.addScenarioDevice(url,
         "com.sap.portals.heartbeats.samples.ScenarioDeviceSample2");

    GRMFactory.testScenario(scpan, request, response);
   }
   catch(Exception e){e.printStackTrace();
   }
 }
```

5. GRMG customizing file:

```
<?xml version="1.0" encoding="UTF-8"?>

<customizing>
 <control>
  <grmgruns>X</grmgruns>
  <runlog> </runlog>
  <errlog> </errlog>
 </control>
 <scenarios>
  <scenario>
   <scenname>pmwdf101</scenname>
   <scenversion>010</scenversion>
   <sceninst>100</sceninst>
   <scentype>URL</scentype>
   <scenstarturl>
    http://pmwdf101.wdf.sap-ag.de:8090/heartbeat/servlet/ScenarioServlet
   </scenstarturl>
   <scenstartmod>Unknown</scenstartmod>
   <scentexts>
    <scentext>
     <scenlangu>EN</scenlangu>
     <scendesc>Test for scenario</scendesc>
    </scentext>
   </scentexts>
   <components>
    <component>
     <compname>TestWeb</compname>
     <compversion>001</compversion>
     <comptype>Unknown</comptype>
     <comptexts>
```

```
  <comptext>
   <complangu>EN</complangu>
   <compdesc>Component_Web</compdesc>
  </comptext>
 </comptexts>
 <properties>
  <property>
   <propname>url</propname>
   <propvalue>
    http://localhost:8090/examples/servlets/helloworld.html
   </propvalue>
  </property>
 </properties>
</component>
<component>
 <compname>TestSAP</compname>
 <compversion>001</compversion>
 <comptype>Unknown</comptype>
 <comptexts>
  <comptext>
   <complangu>EN</complangu>
   <compdesc>Component_SAP</compdesc>
  </comptext>
 </comptexts>
 <properties>
  <property>
   <propname>SAPClient</propname>
   <propvalue>001</propvalue>
  </property>
  <property>
   <propname>userid</propname>
   <propvalue>myUserName</propvalue>
  </property>
  <property>
   <propname>password</propname>
   <propvalue>myPassword</propvalue>
  </property>
  <property>
   <propname>language</propname>
   <propvalue>DE</propvalue>
  </property>
  <property>
   <propname>host_name</propname>
   <propvalue>csnmain.wdf.sap.corp</propvalue>
  </property>
  <property>
   <propname>system_number</propname>
   <propvalue>24</propvalue>
  </property>
 </properties>
</component>
<component>
 <compname>Cluster</compname>
 <compversion>002</compversion>
 <comptype>Unknown</comptype>
 <comptexts>
  <comptext>
   <complangu>EN</complangu>
   <compdesc>Component_Cluster</compdesc>
  </comptext>
 </comptexts>
```

```
     <properties>
      <property>
       <propname>urls</propname>
       <propvalue>http</propvalue>
      </property>
     </properties>
    </component>
   </components>
  </scenario>
 </scenarios>
</customizing>
```

Remark: 1. The tag "`scenstarturl`" tells GRMG the URL of the servlet or iView containing the GRMG application which runs the scenario tests for the components indicated in the XML file**. In the next subsection various sample URL types are listed for use in a servlet engine or in the Enterprise Portal.**

2. At the moment a template customizing xml file for the basic scenario tests shipped with the Enterprise Portal EP60 can be obtained from within the installed portal as follows:

Either

`<installation_portal>/usr/sap/<system_name>/java/sdm/root/`
`The-Host-Itself/sap.com/com.sap.portal.heartbeats`

`com.sap.portal.heartbeats.sda`
(→ unzip →)
`com.sap.portal.heartbeats.par`
(→ unzip →)
`grmgRequestCust.xml`

or

`<installation_portal>/usr/sap/<system_name>/j2ee/j2ee_<inst_number>/`
`cluster/server/services/servlet_jsp/work/jspTemp/irj/root/WEB-INF/deployment/pcd`

`com.sap.portal.heartbeats.par.bak`
(→ unzip →)
`grmgRequestCust.xml`

In EP50 a sample grmg customizing file can be found in:

`<installation_portal>/. . ./services/servlet_jsp/work/jspTemp/irj/root/WEB-INF/plugins/portal/resources/Heartbeats\xml`

Sample urls for tag `scenstarturl`:

In the sequel examples for the values of `scenstarturl` and for the configuration of SAP J2EE_Engine and R/3, CCMS will be listed.

Remark: *Before using https/ssl make sure that the J2EE_Engine (and IIS in case of EP5) are SSL enabled. See documents "Enabling SSL for the SAP J2EE Engine" (and "enabling SSL redirection with the ISAPI module") and the*

Installation/configuration documents for Portal/J2EE_Engine. Furthermore make sure that the server certificates are uploaded in the monitoring R/3 SAP System via transaction STRUST.

Ideally these preparatory steps should have been accomplished during installation/configuration of the system landscape.

In order to customize the GRMG XML file the following steps need to be done.

1. **(Servlet**

`http://<server>:<http_port>/heartbeat/servlet/ScenarioServlet`

)

2. **iView in EnterprisePortal 50 (on J2EE_Engine) with built-in heartbeat test**

i. Using http and standard URL: Should be avoided for security reasons!

```
http://<server>:<http_port>/irj/servlet/prt/portal/prtroot/Heartbeats.default
```

ii. Using https / ssl / base authentication:

In any case it is supposed that at least a server certificate on the J2EE_Engine is installed and the keystore and ssl services start up automatically. (See document "Enabling SSL for the SAP J2EE Engine" provided by SAP J2EE Engine). Furthermore the certificate needs to be uploaded in the monitoring R/3 client (for example via TX STRUST – see corresponding documentation of R/3 backend customizing).

In case of EP5 SSL enabling of the IIS needs to be guaranteed.

a) basic authentication via URL:

→ set value of `<scentype>` to URL
→ set value of `<scenstarturl>` to

```
https://iisuser:iispasswd@<IIS_server>:<IIS_https_port>/irj/servlet/prt/portal/prtroot/Heartbeats.default
```

b) via RFC destination:

→ define an RFC destination in TX sm59 in R/3:

- "Http connection to external server"
- set destination server to IIS server
- set path prefix to:

```
/irj/servlet/prt/portal/prtroot/com.sap.portal.heartbeats.PortalHeartbeat
```

- set service number to IIS https port

- set security to “basic authentication” [1]
- set connection type to “G”
- set SSL to “active”
- set user and password of IIS user

→ set value of `<scentype>` to `HRFC`

→ set value of `<scenstarturl>` to name of RFC destination

3. iView in Enterprise Portal 60 (on J2EE_Engine) with built-in heartbeat test

A Portal super admin user/password, e.g. `heartbeat/heartb` has to be created in the portal before monitoring can be triggered! Please observe that due to CCMS restrictions the password’s size should not exceed 6 characters!

i. Using http and standard URL containing user/password (!): Should be avoided for security reasons!

SP1:

```
http://<server>:<http_port>/irj/servlet/prt/portal/prtroot/com.sap.portal.heartbeats.PortalHeartbeat?j_user=heartbeat&j_password=heartb&login_submit=off
```

beyond SP1:

```
http://<user>:<password>@<server>:<http_port>/irj/servlet/prt/portal/prtroot/com.sap.portal.heartbeats.PortalHeartbeat
```

ii. Using https / ssl :

In any case it is supposed that at least a server certificate on the J2EE_Engine is installed and the keystore and ssl services start up automatically. (See document “Enabling SSL for the SAP J2EE Engine” provided by SAP J2EE Engine). Furthermore the certificate needs to be uploaded in the monitoring R/3 client (for example via TX STRUST – see corresponding documentation of R/3 backend customizing).

c) basic authentication via URL:

→ set value of `<scentype>` to `URL`

[1] You may wish to use SSL client certificates instead, in which case you need to check the button “SSL Client Certificates”, provide a client certificate and configure the J2EE_Engine such that communication with certified clients is enabled. Furter documentation is provided in *“Enabling SSL for the SAP J2EE Engine”* and in the corresponding R/3 backend documentation.

→ set value of `<scenstarturl>` to

```
https://<server>:<https_port>/irj/servlet/prt/portal/prtroot/com.sap.portal.heartbeats.PortalHeartbeat?j_user=heartbeat&j_password=heartb&login_submit=off
```

d) via RFC destination:

→ define an RFC destination in TX sm59 in R/3:

- "Http connection to external server"
- set destination server to server of J2EE_Engine
- set path prefix to:

```
/irj/servlet/prt/portal/prtroot/com.sap.portal.heartbeats.PortalHeartbeat
```

- set service number to ssl/https port of J2EE_Engine
- set security to "basic authentication" [2]
- set connection type to "G"
- set SSL to "active"
- set user and password of Portal user (heartbeat/heartb)

→ set value of `<scentype>` to `HRFC`
→ set value of `<scenstarturl>` to name of RFC destination

Observe that RFC is fully applicable only for R/3 version 6.40. Use URL for 6.30 since prefix in is confined to 64 characters in TX sm59.

6. Further features

a. Dynamic upload and composition of scenarios:

With a `ScenarioPanel` one can register `ScenarioDevice` objects using methods `addScenarioDevice(…)`.

Devices will be registered either from classpath, Jar file or some URL (dynamic loading of scenarios).

b. `GrmgXMLFile` represents a GRMG customizing XML file.

The instance will be created from a base customizing XML file, a `Document` object or an input stream – possibly with empty component lists.

[2] You may wish to use SSL client certificates instead, in which case you need to check the button "SSL Client Certificates", provide a client certificate and configure the J2EE_Engine such that communication with certified clients is enabled. Further documentation is provided in *"Enabling SSL for the SAP J2EE Engine"* and in the corresponding R/3 backend documentation.

The object allows dedicated manipulations on the file content.

For a `ScenarioDevice` object one may provide a component XML file representing the part of the scenario described by the `ScenarioPanel`. These XML files obey the DTD:

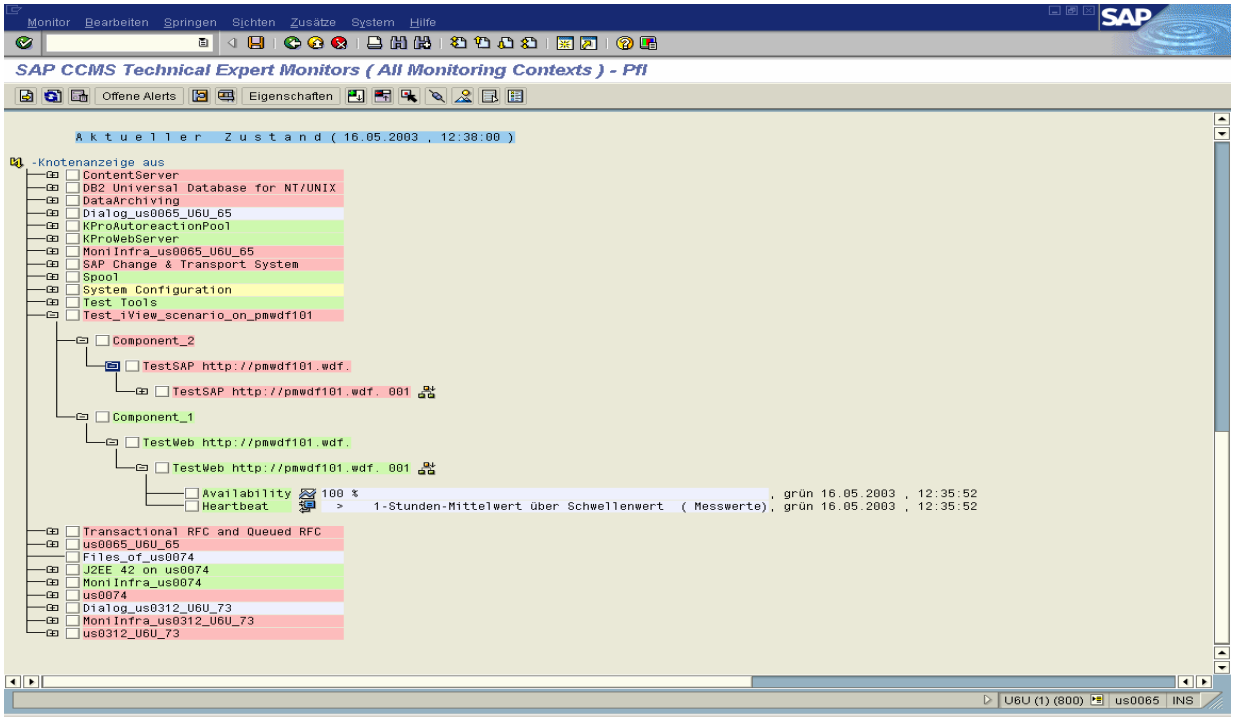
```
<!ELEMENT scenariodevice (component*)>
<!ELEMENT component (compname,compversion,
                     comptype,comptexts,properties)>
<!ELEMENT compname (#PCDATA)>
<!ELEMENT compversion (#PCDATA)>
<!ELEMENT comptype (#PCDATA)>
<!ELEMENT comptexts (comptext*)>
<!ELEMENT comptext (complangu, compdesc)>
<!ELEMENT complangu (#PCDATA)>
<!ELEMENT compdesc (#PCDATA)>
<!ELEMENT properties (property*)>
<!ELEMENT property (propname, propvalue)>
<!ELEMENT propname (#PCDATA)>
<!ELEMENT propvalue (#PCDATA)>
```

The scenario device XML files can be added to the overall GRMG customizing file (represented by the `GrmgXMLFile` object) with the help of method(s) `addComponent(…)`.

Using `ScenarioDevice`, `ScenarioPanel` and `GrmgXMLFile` and `GrmgCustomizingXMLGenerator` (by Pavel Petrov) one may now easily create/generate GRMG scenarios dynamically – e.g. for CCMS or for Web based scenario test facilities.

7. ***Deploy the iView or servlet to the PRT or servlet engine resp.***

8. ***Upload the GRMG customizing file to SAP system using TX "`grmg`" (see separate documentation) and monitor in TX rz20:***



→ For more information on GRMG (XML format, DTD etc.) see:

a. Enabling GRMG Heartbeat Monitoring

b. Setting up and operating GRMG monitoring scenarios

c. Instrumenting Components for Monitoring with GRMG