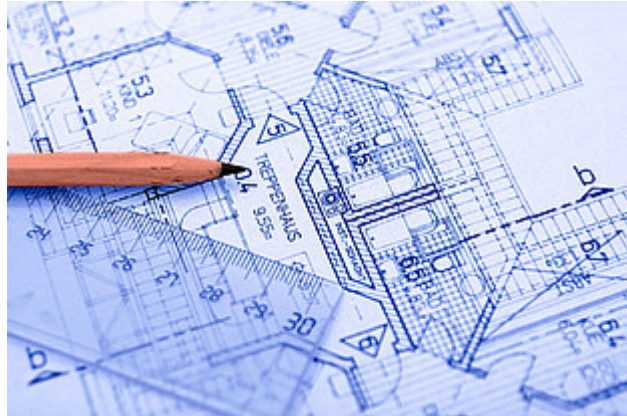


Engineering a Web Application with Palladio (Version 2.0)

The goal of this workshop is to engineer a web application with Palladio. For this workshop, a web application consists of a web interface allowing users to submit their fore- and surname into a database.

The workshop presents the tasks of modeling the web application and performing performance analyses step-by-step. It is based on the Palladio screencast series available at <http://www.palladio-simulator.com/tools/screencasts/>.



1. Installation [Estimated Time: 20 – 40 min.]

1. Download and install the current »Eclipse Modeling Tools« edition available at <http://www.eclipse.org/downloads/> (last tested with Eclipse 4.5.1/Mars.1 Release).
2. Start your newly installed Eclipse and go to your update sites (Help -> Install New Software...).
3. Click »Add...« and Enter »Palladio« as »Name« and »https://sdqweb.ipd.kit.edu/eclipse/palladiosimulator/releases/latest/« as »Location«.
4. Click »Select All«, hit »Next >« (2 times), accept the licence agreement, and click »Finish«.
5. After restart, add another update site (Help -> Install New Software...) with »Architectural Templates« as »Name« and »http://cloudscale.xlab.si/cse/updatesites/architecturaltemplates/nightly/« as »Location«.
6. Click »Select All«, hit »Next >« (2 times), accept the licence agreement, and click »Finish«.
7. After a successful installation, switch to the »Palladio« perspective for executing the next steps (Window -> Open Perspective -> Other... -> Palladio).

General Information can also be found here: http://sdqweb.ipd.kit.edu/wiki/PCM_Installation

2. Buttons [Estimated Time: 1 min.]

This workshop will refer to these buttons available in a Palladio installation:




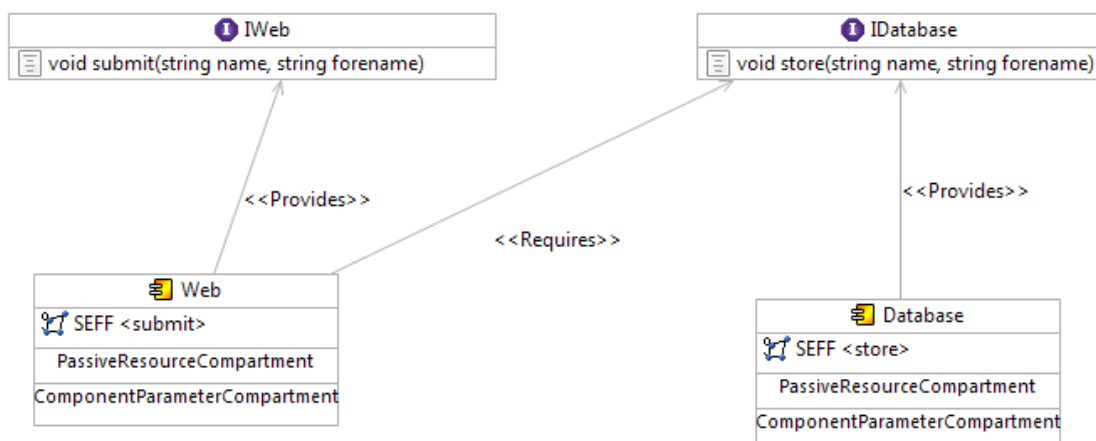
From left to right, these buttons are named as follows:

1. New Repository Model Diagram (🏠)
2. New System Model Diagram (🔧)
3. New Resource Model Diagram (💻)
4. New Allocation Model Diagram (📁)
5. New Usage Model Diagram (👤)

3. Repository Model [Estimated Time: 15 min.]

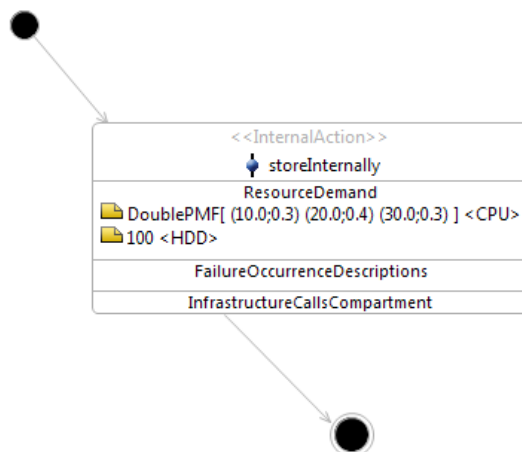
A component developer uses the repository model to specify a set of components that can later be deployed within a system. Let us model a database component for storing fore- and surnames as well as a web component allowing to access the database via a web interface.

1. Create a new general project »PalladioProject« (Right-click at the Project Explorer view -> New -> Project... -> General -> Project -> Project Name: »PalladioProject« -> Finish).
2. Select the newly created project and click the »New Repository Model Diagram« () button.
3. Select the newly created project folder and click »Next« (this determines the place to store the repository diagram). You may leave the name (»default.repository_diagram«) as it is.
4. Select the newly created project folder and click »Finish« (this determines the place to store the repository model). You may leave the name (»default.repository«) as it is. The diagram should open; on its right is a palette allowing to add elements to the diagram.
5. Add an »Interface« to the diagram and name it »IWeb«. Add a »Signature« from the palette to this interface. Go to the Properties view and set the signature to »void submit(string name, string forename)«. This models the interface to the web component.
6. Repeat step 5 for the »IDatabase« interface providing a service with the signature »void store(string name, string forename)«. This models the database interface providing a service to store a fore- and a surname within a database.
7. Next, we will add the components providing and requiring these interfaces. Add a »BasicComponent« to the diagram. Name it »Web«.
8. Add another »BasicComponent« to the diagram. Name it »Database«.
9. Connect the Database component to the IDatabase interface via a »ProvidedRole« from the palette. Connect Web and IWeb similarly.
10. As the Web component needs to write into a database, connect the Web component with IDatabase via a »RequiredRole« from the palette. Your diagram should look like shown below.



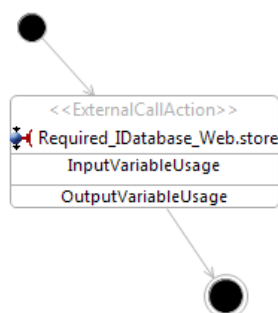
11. Next, we need to specify the behavior of the services our components provide. Therefore, we specify »Service Effect Specifications« (SEFFs) that model the performance-relevant behavior of our services. They are similar to activity diagrams.

- Double click the »SEFF <store>« entry of the Database component. A SEFF diagram editor opens. Remove the link between these two actions. Next, add an »InternalAction« from the palette to the diagram and name it »storeInternally«.
- Add a »ResourceDemand« from the palette to this action. Select »CPU« and press »OK«. As a stochastic expression, enter »**DoublePMF** [(10; 0.3) (20; 0.4) (30; 0.3)] «. This models that our database needs 10 CPU workunits in 30% of the cases to store a name. In 40% of the cases, it needs 20 CPU workunits and in 30% of the cases, it needs 30 CPU workunits. You may use the »Help« button as an online help within the stochastic expression editor to look up the meaning of these stochastic expressions.
- Add another »Resource Demand« to the action. This time, select »HDD« and enter »100« as a stochastic expression. This models that the database also needs 100 HDD workunits (constantly, in all of the cases).
- Connect the start action with the »storeInternally« action via a »Control Flow« link from the palette. Analogously, connect »storeInternally« with the end action. Your diagram should look like shown below. You can save and close the diagram now to get back to the repository diagram.



12. Let us also specify the SEFF of the Web component's »submit« service.


- Double click the »SEFF <submit>« entry of the Web component.
- Add an »ExternalCallAction« from the palette to the diagram and select the »store« OperationSignature from the dialog that opens. Confirm with »OK«. By this, we model the call to the database from our Web component.
- Connect the Actions appropriately via »Control Flow« such that you get a diagram like shown below. Save and close the diagram.

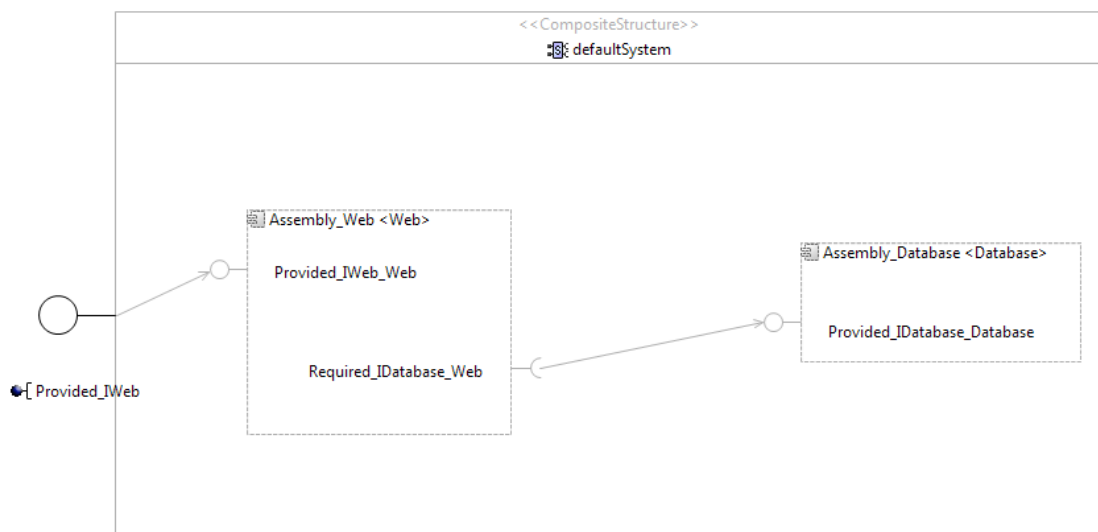


13. Save and close the repository diagram.

4. System Model [Estimated Time: 5 min.]


A system architect uses the available components in component repositories (cf. Step 3 – Repository Model) to compose a concrete component-based software system. Let us build such a system for the components we just created.

1. Click the »New System Model Diagram« () button.
 2. Create a new system diagram and model within the »PalladioProject« project folder. The diagram opens with a »defaultSystem« allowing to specify our web application system.
 3. Add an »AssemblyContext« from the palette to the »defaultSystem«. Load the repository we created before (Load Repository -> Browse Workspace... -> PalladioProject / default.repository -> OK -> OK). Select the »Database« component and confirm with »OK«.
 4. Analogously repeat 3. to add the »Web« component.
 5. Connect the two assemblies by using an »AssemblyConnector« from the palette: connect the required interface of »Assembly_Web« with the provided interface of »Assembly_Database«. We now assembled our components within the system.
 6. Finally, we need to provide an interface to our system such that it gets accessible by users. Add a »SystemOperationProvidedRole« from the palette to the »defaultSystem« and select the »IWeb« interface. Use an »OperationProvidedDelegationConnector« from the palette to connect the system's provided role to the providing role of the Web assembly.
- You should get a system like shown below. You can save and close the system diagram now.



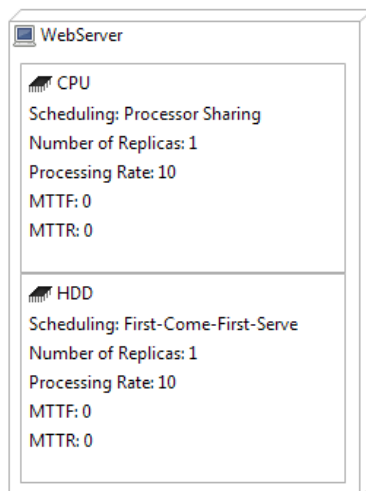
5. Resource Environment [Estimated Time: 5 min.]

A system deployer uses the resource environment to model CPUs, hard disk drives, networks, etc. Let us first consider a single server for our simple web application where we will deploy both, the web and the database components.

1. Click the »New Resource Model Diagram« () button.
2. Create a new resource environment diagram and model within the »PalladioProject« project folder. The diagram opens automatically.

3. Add a »ResourceContainer« from the palette to the diagram and name it »WebServer«.
4. Next we add a CPU specification to the latter container.
 - Add a »ProcessingResourceSpecification« to the newly created resource container. Select »CPU« as processing resource type and confirm with »OK«.
 - A stochastic expression editor pops up in which you specify a processing rate of »10«. This models that our CPU can handle 10 CPU workload units per second. Confirm with »OK«.
 - A new dialog pops up enabling us to select a scheduling policy for our CPU. Select »Processor Sharing« which is a round-robin strategy. This models the scheduler behavior of operating systems in a simplified form. Confirm with »OK«.
5. Analogously repeat 4. for adding a hard disk drive to the WebServer: Select »HDD«, specify a processing rate of »10« modeling that our hard disk drive can handle 10 HDD workload units per second, and select »First-Come-First-Serve« as scheduling policy which models a typical behavior of hard disk drives.

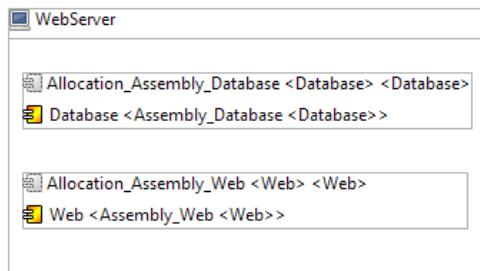
You should get a resource environment like shown below. You can save and close the resource environment diagram now.



6. Allocation Model [Estimated Time: 5 min.]


Another task for the system deployer is to allocate the components assembled within the system to the resource environment. Let us allocate the system's components to our WebServer.

1. Click the »New Allocation Model Diagram« (📄) button.
2. Create a new allocation diagram and model within the »PalladioProject« project folder. You also have to select the resource environment (»default.resourceenvironment«) and system (»default.system«) from our workspace. The diagram opens automatically after pressing »Finish«. It already includes the WebServer we specified before.
3. Add an »AllocationContext« from the palette to the WebServer. Select the »Assembly_Database« component and confirm with »OK« to allocate it to the WebServer.
4. Proceed analogously with adding the »Assembly_Web« component.
You should get an allocation like shown below. You can save and close the allocation diagram now.

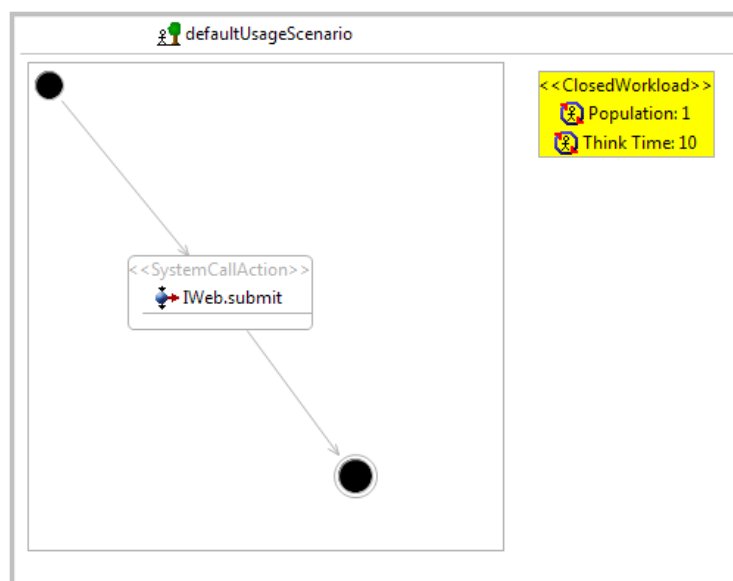


7. Usage Model [Estimated Time: 5 min.]

A domain expert specifies the behavior of users that will use our system. Let us specify that one user repeats submitting fore- and surnames to our web application. After each submit, the user pauses for 10 seconds.

1. Click the »New Usage Model Diagram« () button.
2. Create a new usage diagram and model within the »PalladioProject« project folder. The diagram opens automatically. The diagram already includes a default usage scenario that we will extend.
3. Resize the diagram to your needs. Then add an »EntryLevelSystemCall« from the palette to the diagram. Select the system we specified from your workspace (»default.system«). Then, select the »Provided_IWeb« provided interface of our system and click »OK«. In the next dialog, choose the »submit« operation and confirm with »OK«. This models the call to our system via its provided submit service.
4. Connect the actions appropriately by using the »Usage Flow« links from the palette.
5. Add a »ClosedWorkload« from the palette to our »defaultUsageScenario«. Specify a population of »1« user and a think time of »10« seconds. This models that one user is within our system, calls the submit service of our system, waits for 10 seconds, and finally repeats this process.

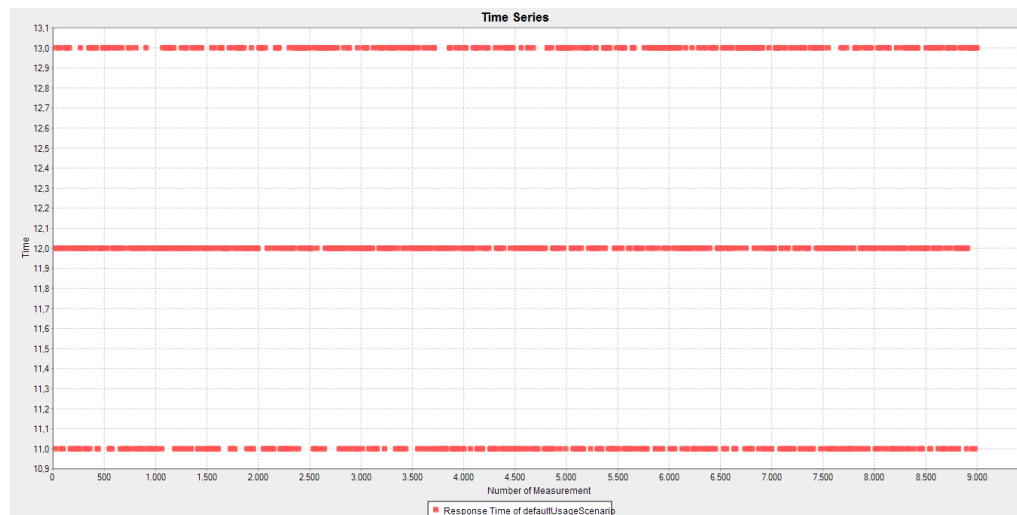
You should get a usage model like shown below. You can save and close the usage diagram now.



8. Performance Predictions [Estimated Time: 30 min.]

We are now prepared to execute performance predictions of the web application we modeled. This task is usually performed by system architects such that they can evaluate different design alternatives.

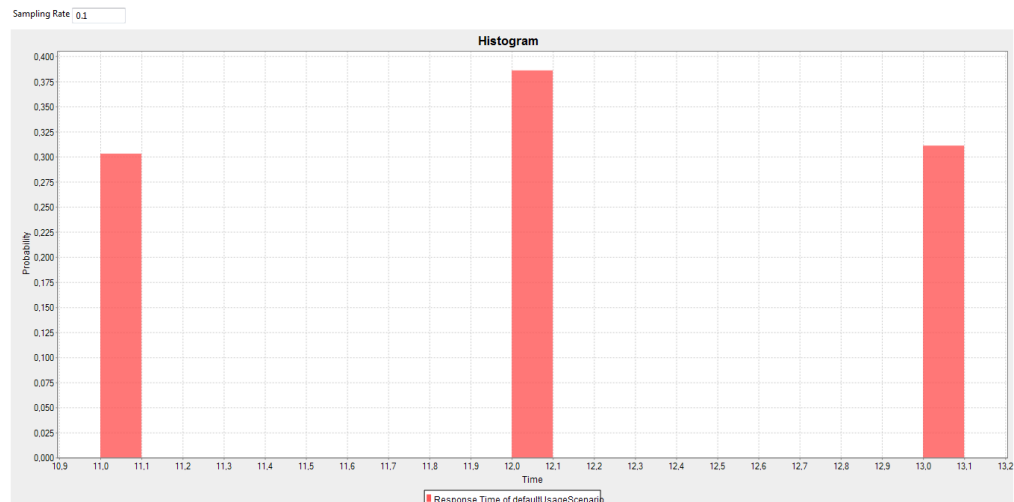
1. Open the »Run Configurations« dialog (Run -> Run Configurations...).
2. Doubleclick the »SimuBench« configuration category to create a new run configuration for the SimuCom solver, an often applied solver in Palladio.
3. Name the configuration »SimuCom-WebApplication«.
4. In the »Architecture Model(s)« tab, select the allocation (»default.allocation«) and usage (»default.usagemodel«) models we created from the workspace.
5. In the »Simulation« tab, set the »Experiment Name« to »SimuCom-WebApplication« and set the »Maximum measurement count« to »1000« for shorter simulation times. Finally, browse available »Data sources« for the »Experiment Data Persistency & Presentation (EDP2)« by pressing the »Browse...« button. Press »Add...«, select »In-Memory data source«, and press »Finish«. Select the new »LocalMemoryRepository« for storing our measurement results in main memory.
6. Apply your changes and press the »Run« button. The simulation may take a short while. You can follow the simulation status in the »Console« and the »Simulation Dock Status« views.
7. After the simulation, go to the »Experiments« view. Expand your datasource completely if not already expanded. You should be able to see different entries with measurements taken by different monitors.
8. Let's investigate some of these measurements.
 - a. Doubleclick the »Usage Scenario: defaultUsageScenario« entry. Doubleclick the »XY Plot« entry in the dialog that pops up. You will get a diagram similar to the one shown below.



On the x-axis you see the time when the measurement was taken and on the y-axis the response time as measured for the usage scenario. In our scenario, the response time of one measurement is the time that is needed for getting the result of »IWeb.submit« when calling it. The three horizontal lines result from our specification of the demanded CPU workunits as a PMF (10 in 30%, 20 in 40%, and 30 in 30% of the cases). With a CPU processing rate of 10 workunits per second, the CPU can cause 1, 2, or 3 seconds response time. The remaining 10 seconds are caused by

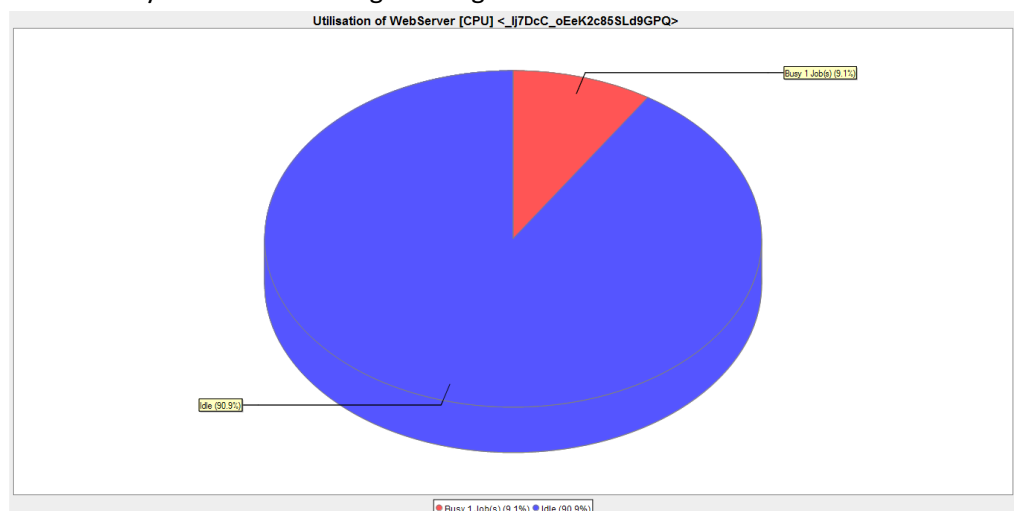
the hard disk drive demand (we demanded 100 workunits and have a hard disc processing rate of 10 workunits per second).

- b. Let's open a different diagram to see an alternative to an XY Plot. Doubleclick the »Usage Scenario: defaultUsageScenario« entry, again. This time, doubleclick the »Histogram« entry next. You will get a diagram similar to the one shown below.



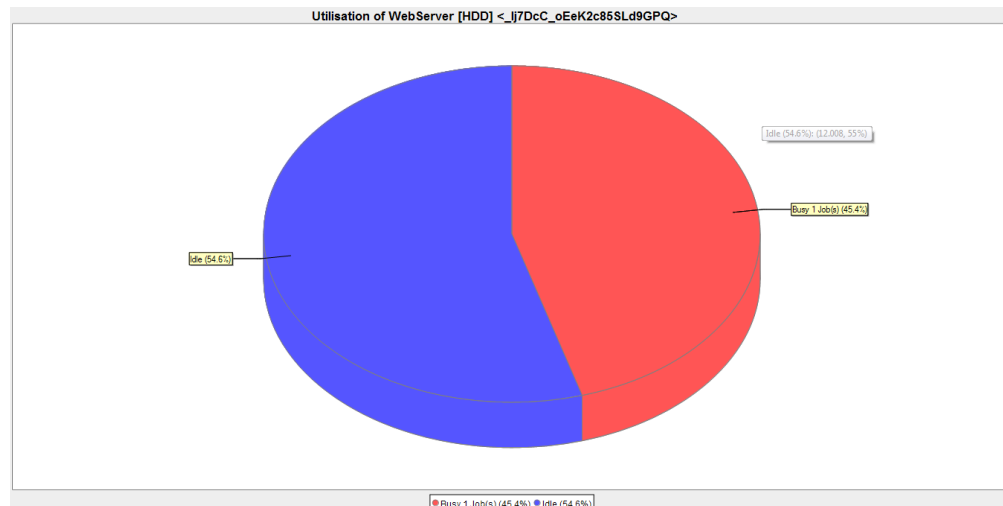
On the x-axis you see response times and on the y-axis the measured probability of a given response time. Here, we clearly see the correspondence of the probabilities we set in the CPU's PMF and the actual measurements (they closely lie around the 30%, 40%, and 30% marks).

- c. Let us now have a closer look on the utilization of the CPU. Doubleclick the »CPU [0] in WebServer (State of Active Resource Tuple)« entry for this. Doubleclick the »Pie Chart« entry next. You should get a diagram similar to the one shown below.



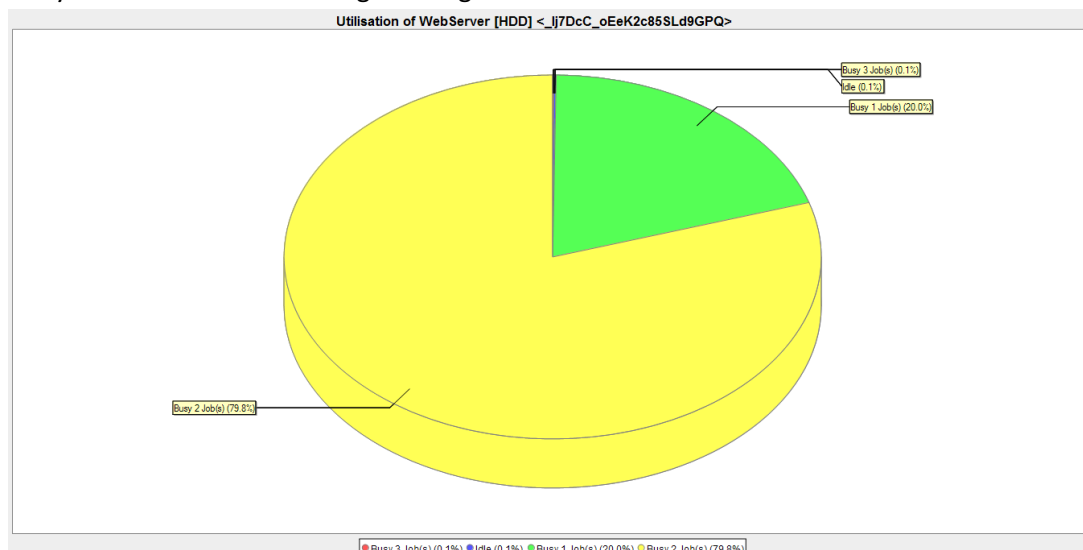
In approximately 9% of the time, the CPU handles 1 job and in 90% of the time, the CPU is idle. Therefore, we do not observe an overload situation here.

- d. Now we will similarly investigate the hard disc's utilization. Doubleclick the »HDD [0] in WebServer (State of Active Resource Tuple)« entry for this. Again, doubleclick the »Pie Chart« entry next. You should get a diagram similar to the one shown below.



In approximately 45% of the time, the hard disk drive handles 1 job and in 55% of the time, the hard disk drive is idle. Therefore, we also do not observe an overload situation here.

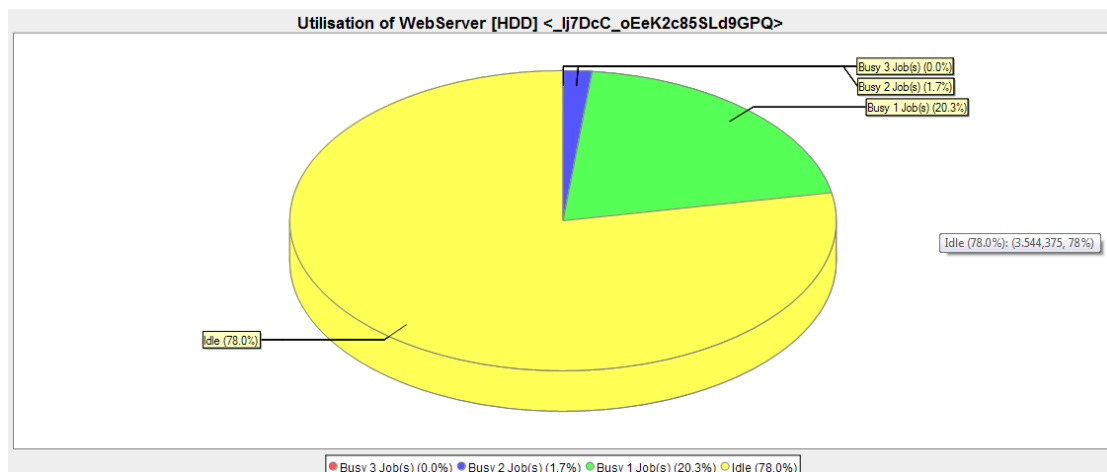
9. At least for one user within our system, the system is predicted to be stable. We may think about using a cheaper CPU as it is in 90% of the time idle. However, let's consider another scenario instead: What happens, if the system is used by three users instead of only one? To investigate this issue, change the »Population« of the ClosedWorkload in the usage model from »1« to »3« and rerun the simulation. In the »Experiments« view, navigate to the newly created experiment run. Doubleclick the »HDD [0] in WebServer (State of Active Resource Tuple)« entry and doubleclick the »Pie Chart« entry afterwards. You should get a diagram similar to the one shown below.



This time, the hard disk drive is hardly idle but needs to handle 1 job in approximately 20% of the time and 2 jobs in 80% of the time. This means that we have an overload situation here because the hard disk drive is unable to handle the amount of jobs fast enough. An investigation of the response times of the defaultUsageScenario confirms this as the response times increased to approximately 20 seconds per measurement.

10. A straight-forward idea to cope with the overload situation is to use a faster hard disk drive within our WebServer, i.e., to scale the hard disk drive up. Therefore, change the processing rate of the hard disk drive from »10« workunits to »100« workunits within the resource

environment and rerun the simulation. The resulting utilization pie chart for the hard disk drive should look like the one below.



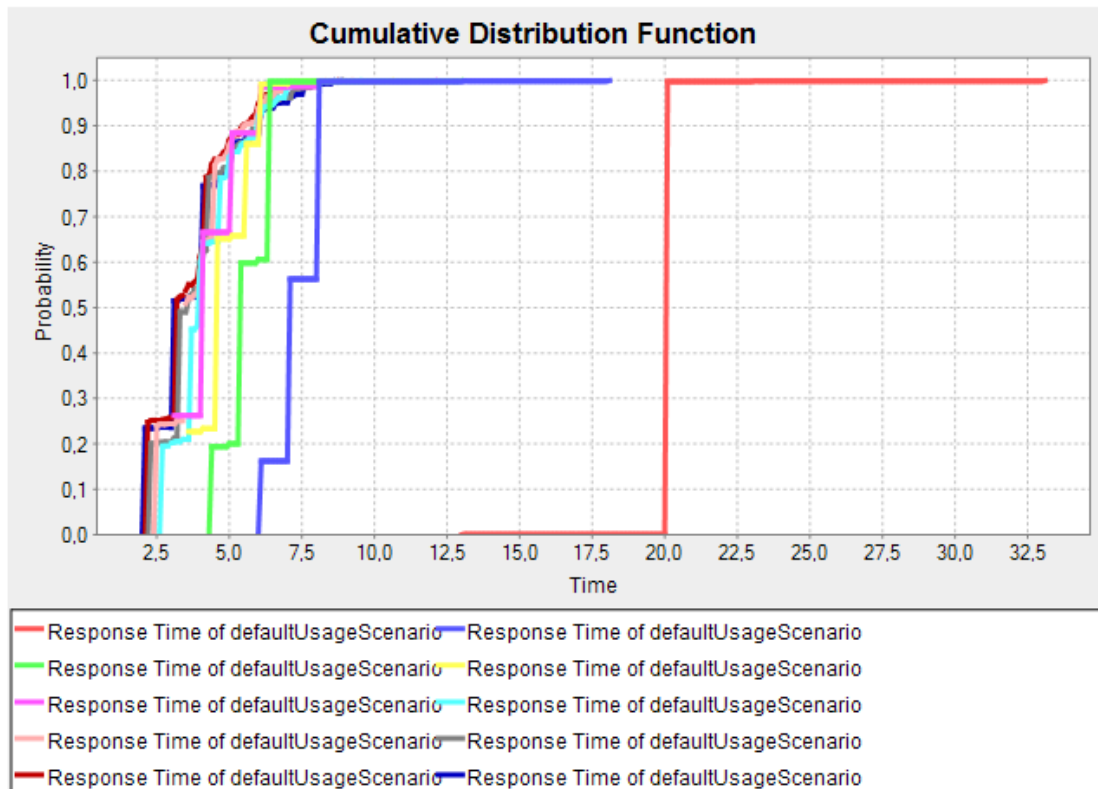
We see that our faster hard disk drive can handle a situation with three users as it is idle for 78% of the time.

11. As a side-remark: we could optimize the processing rate of the hard disk drive to perfectly fit a situation with three users. We could, for instance, aim at having the hard disk drive idle for only 10% of the time such that no over-provisioning occurs.

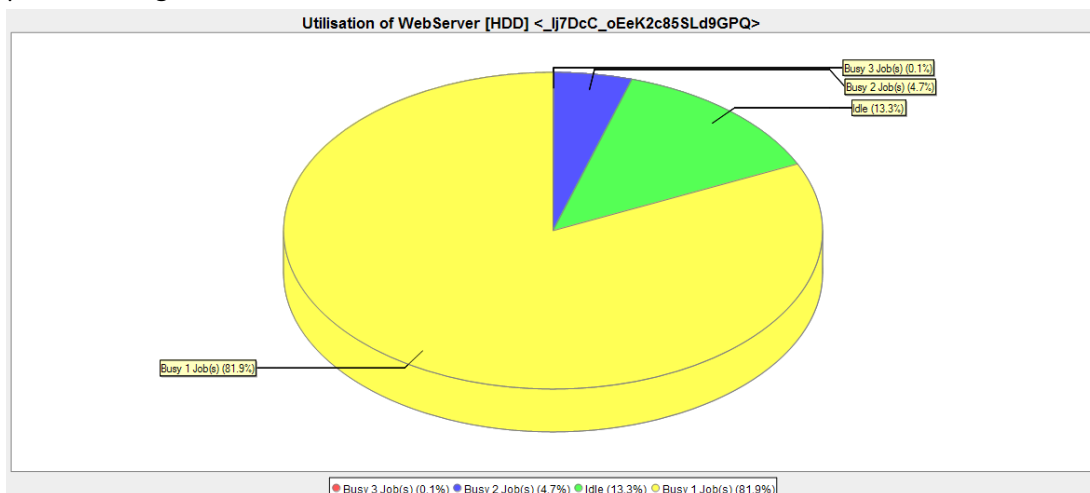
To automate this, Palladio provides a sensitivity analysis within a SimuCom run configuration. Within a SimuCom run configuration, go to the »Analysis Configuration« tab and enter appropriate values to the »Sensitivity Analysis Parameters« edit fields. For instance, select the »ProcessingRate« PCM Random Variable of the hard disc as »Variable« and enter »10« into the »Minimum« field, »100« into the »Maximum« field, and »10« into the »Step Width« field. By this, the simulation run performs ten measurements with hard disc processing rates of 10, 20, ..., 90, and 100 workunits.

The results can nicely be compared via a histogram-based cumulative distribution function (CDF) as follows. Doubleclick the »Response Time of defaultUsageScenario« entry of the first experiment run (with a hard disc processing rate of »10« workunits). Afterwards, doubleclick the »JFreeChart Response Time Histogram-based CDF«. Next, drag and drop each the »Response Time of defaultUsageScenario« of the other nine experiments into the opened CDF diagram. You will get a diagram similar to the one below.

Sampling Rate 0.1



From left to right, the illustrated measurement belong to hard disk drive processing rates of 100, 90, ..., and 10 workunits. Here, we see that a hard disc processing rate of 20 workunits (dark blue values) already improves the response times to a great extend. The corresponding pie chart diagram of the hard disc utilization is shown below.

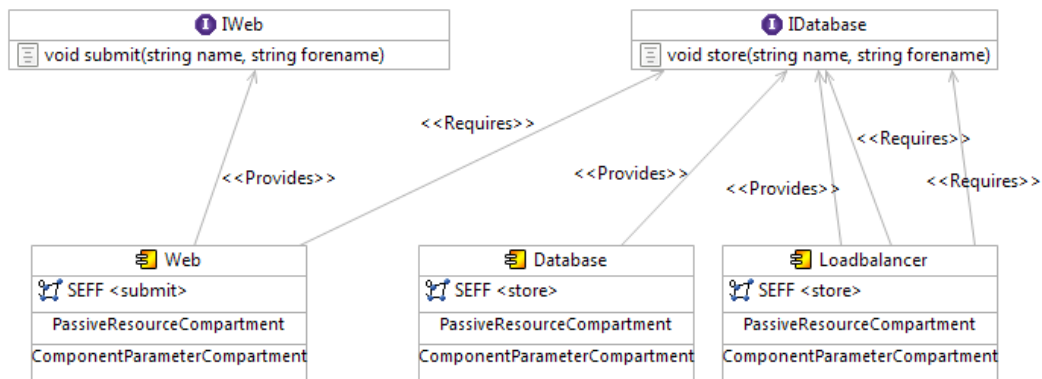


We see that we are close to our goal of having the hard disk drive idle for approximately 10% of the time. Next steps could be to further investigate the relation between the number of users and the hard disc. Alternatively, we may take also the CPU into these considerations as there will also be a point where the CPU becomes the bottleneck of the system. Feel free to experiment on your own regarding these issues.

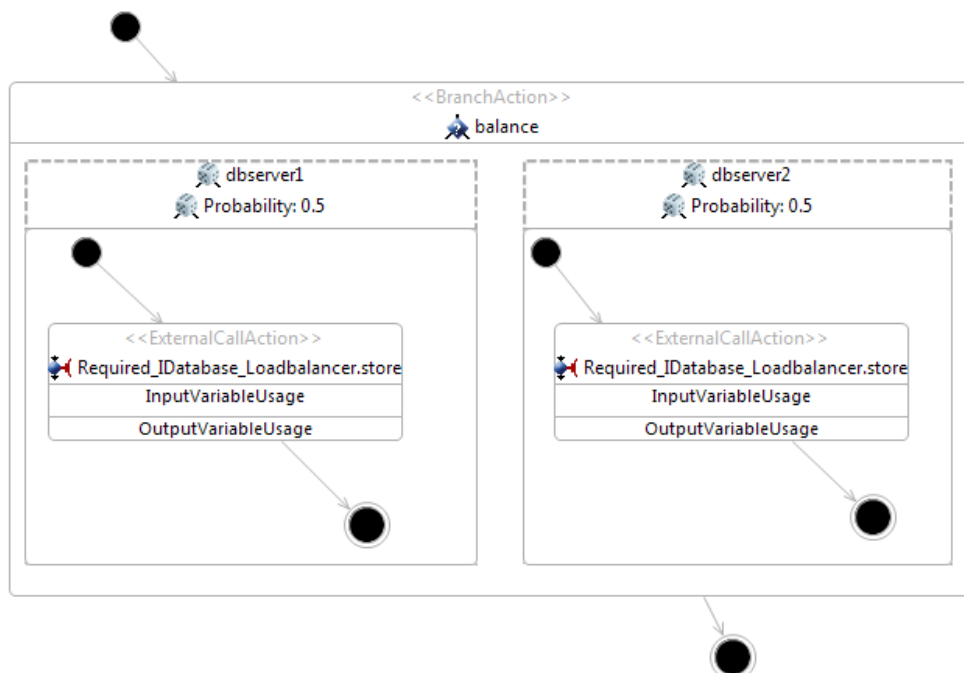
9. Adding a Loadbalancer [Estimated Time: 20 min.]

Besides scaling available processing resources like CPU and hard disk drives up (vertical scaling), we could also try to scale out via additional servers within our system (horizontal scaling). For achieving this, we will outsource our database component to two dedicated database servers. A load balancer on our webserver then assigns 50% of the requests to the first database server and 50% of the requests the second database server.

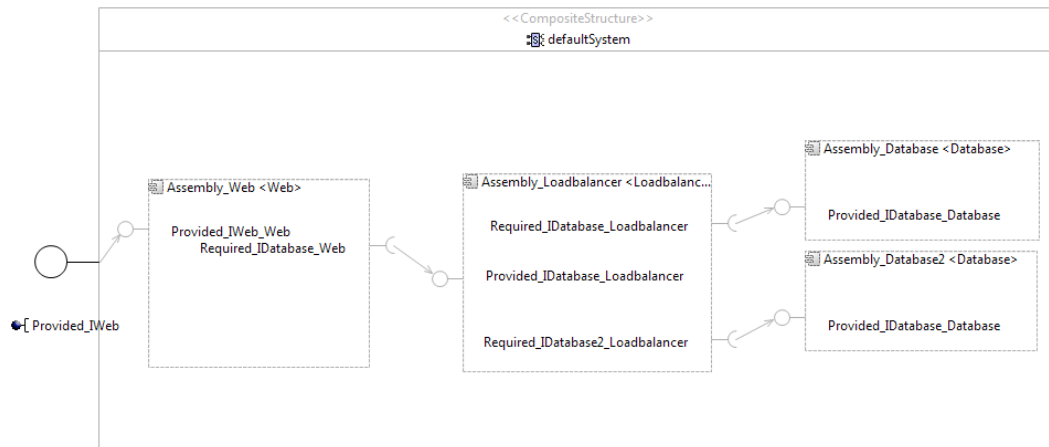
1. Modify the repository by adding a »Loadbalancer« BasicComponent to it that provides the IDatabase interface once and requires the interface twice.



Specify the SEFF for the »store« service of the »Loadbalancer« component as follows. First, add a »BranchAction« from the palette to the SEFF diagram and name it »balance«. Secondly, add two »ProbabilisticBranchTransitions« to the newly created branch action. Name them »dbserver1« and »dbserver2«, respectively. Assign each branch transition a probability of »0.5«. Thirdly, add to each branch transition an »ExternalCallAction«. For the »dbserver1« transition, select the »store« service of the first required role and for the »dbserver2« transition, select the »store« service of the second required role. Finally, connect the added elements appropriately by »Control Flow« links. Your SEFF should look like the one below.

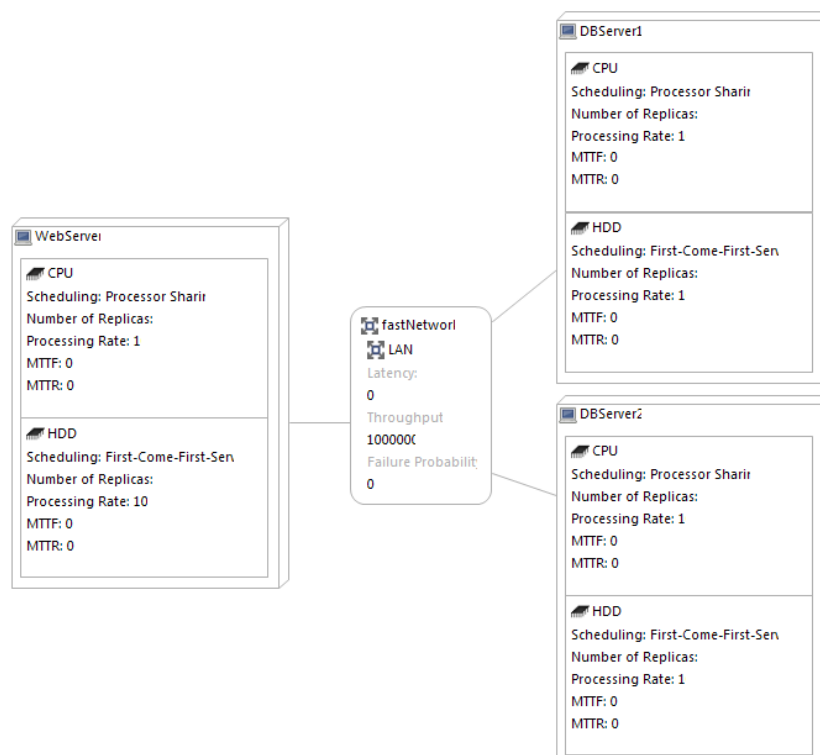


2. Next, we compose a new system by including the load balancer. For this, add the loadbalancer to the system diagram via a new »AssemblyContext«. As we have two databases now, we also add a database component via another new »AssemblyContext« and expand the assemblies name as well as the name of the second provided role of the load balancer by a »2« to ensure a unique naming. We connect the assemblies via »Assembly Connectors« such that we get a system diagram like shown below.

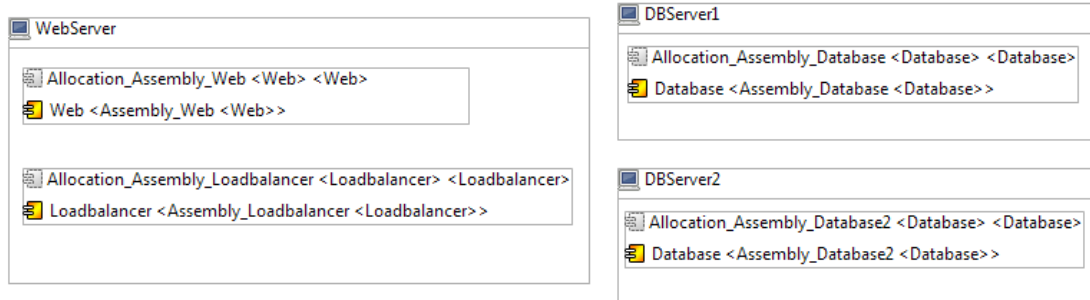


3. In the resource model diagram, we create two new resource containers »DBServer1« and »DBServer2«. Each server gets (1) a CPU with a processing rate of 10 workunits and with a processor sharing strategy as well as (2) a hard disk drive with a processing rate of 10 workunits and with a first-come-first-server strategy.

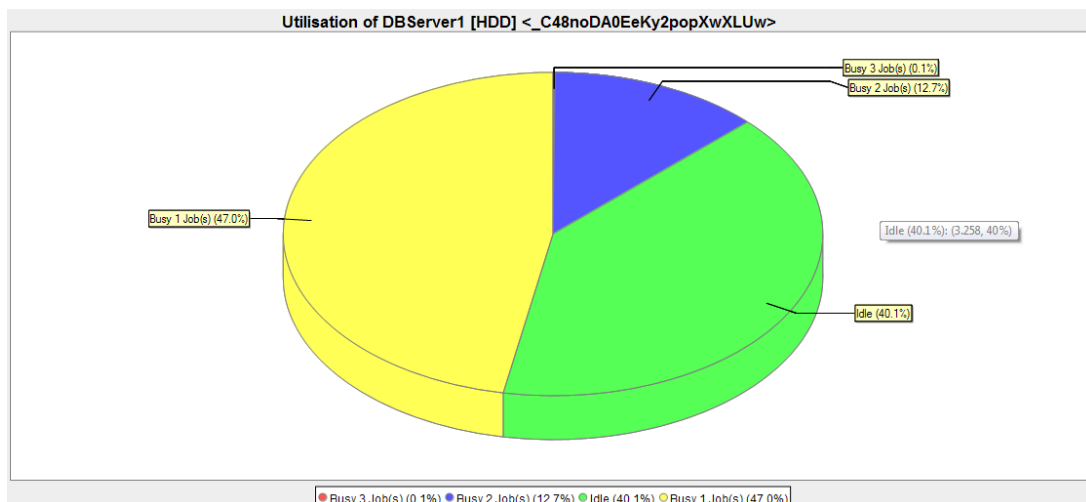
To connect the three servers, we also add a »LinkingResource« from the palette to the diagram. We set the latency to »0« and the throughput to »1000000«, thus, allowing to neglect network performance influence for the time being. We therefore name it »fastNetwork«. We connect the servers to our network via »Connection« links from the palette. Your diagram should look like the one shown below.



4. In the allocation model diagram, allocate the respective components as planned. You will get an allocation diagram like shown below. Note: you have to delete the »Database« allocation from the »WebServer« and add the »Loadbalancer« component instead.



5. We can leave the usage model diagram as it is – our changes were transparent to the users. Therefore, we can run a simulation and investigate the results next. After doing so, compare the two pie charts illustrating the hard disk drive utilization of DBServer1 and DBServer2, respectively. Both pie charts should be similarly to the one shown below.



We see that scaling out to two servers, each having a hard disk drive with a processing rate of 10 workunits, has a similar effect than scaling up a single server's hard disk drive to a processing rate of 20 workunits. Therefore, we found an alternative way of coping with the discovered overload situation.

Note that there are several other ways to improve our modeling or to test design alternatives. For instance, we could model a "real round-robin" instead of using 50% probabilities within our loadbalancer, which could cause different results. Another possibility would be to extend our model by caches for database accesses. Feel free to experiment on your own with the system.

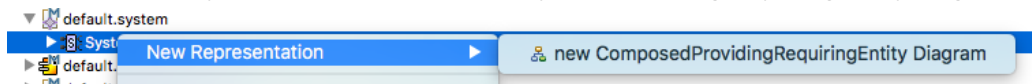
10. Using New Sirius Editors [Estimated Time: 15 min.]

The so-far discussed graphical Palladio editors were implemented using legacy technologies (the GMF framework). Recently, we reimplemented these editors using a modern technology (the Sirius framework). Owing to this new technology, the usage of editors has changed slightly. In this section, we exemplify this usage based on the new system and resource environment editors. (Note: you

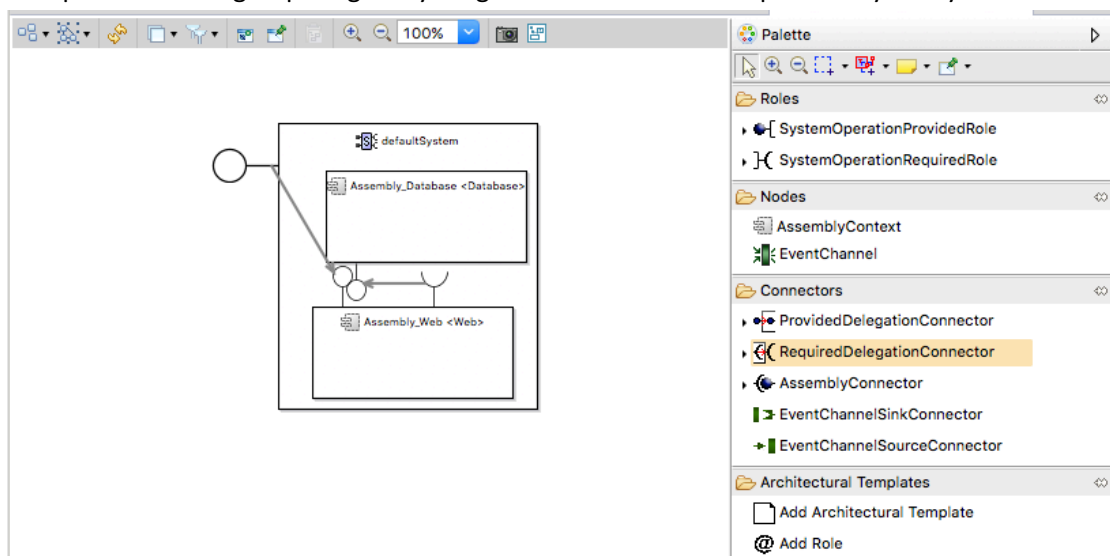
already installed these editors from the Architectural Template repository. Also note that we exemplify these steps in the model without loadbalancer.)

To start System model editing, execute the following steps:

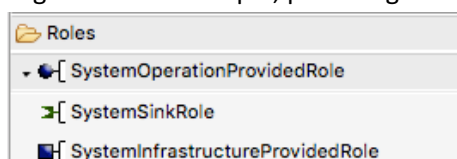
1. Switch to the »Modeling« perspective (or open the »Model Explorer« view manually).
2. In the »Model Explorer« view, right-click on your Palladio project and choose »Configure« -> »Convert to Modeling Project« to make your project compatible with the new framework.
3. Next, right-click on your project and choose »Viewpoints Selection«. Check »System Design« in the popup.
4. Expand your System model as shown below, right-click the »System« model element and choose »New Representation« -> »new ComposedProvidingRequiringEntity Diagram«.



5. Enter any name as diagram (i.e., representation) title, e.g., »new ComposedProvidingRequiringEntity Diagram«. The editor will open with your System loaded.



6. Try using the editor as you used the editors before. Generally, the editor should behave the same. Note that some elements of the palette have been reordered and/or grouped together. For example, providing entities roles have now a dedicated group:

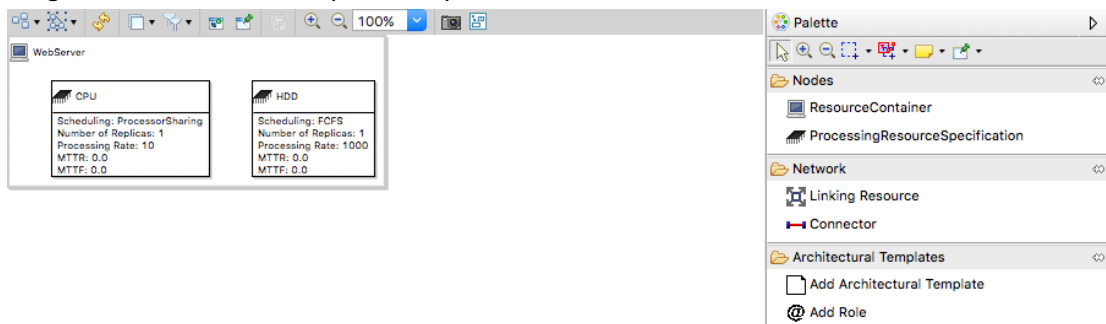


Also note that some elements are new, e.g., the »Architectural Templates« group which is explained in the next section.

Editing resource environments works analogously:

1. Right-click on your project and choose »Viewpoints Selection«. Check »Resource Environment Design« in the popup.
2. Expand your Resource Environment model, right-click the »Resource Environment« model element and choose »New Representation« -> »new Resourceenvironment Diagram«.

3. Enter any name as diagram (i.e., representation) title, e.g., »new Resourceenvironment Diagram«. The editor will open with your Resource Environment loaded.



4. Try using the editor as you used the editors before.

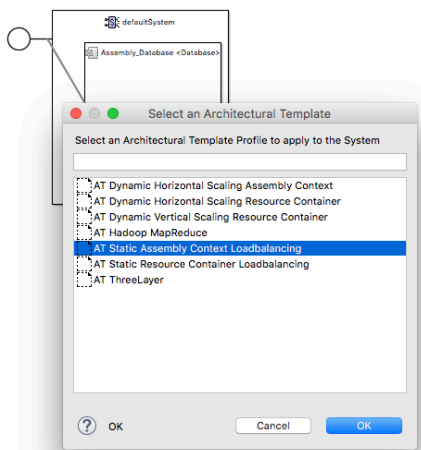
Sirius-based editors for the other Palladio models will appear soon; their usage will be similar.

11. Applying Architectural Templates [Estimated Time: 15 min.]

Architectural Templates allow software architects to apply reusable patterns to their Palladio models. For example, instead of manually modeling the load balancer like we did in Sec. 9, we can also apply the Architectural Template for load balancers. As this application consists only of a few small steps, architects can save a lot of modeling effort.

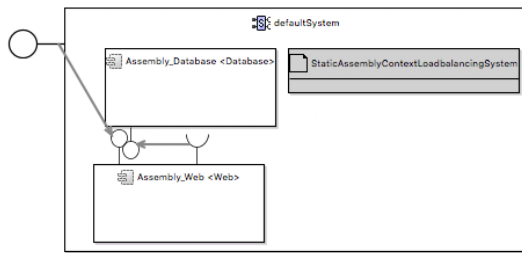
Execute the following steps on the System model opened with the Sirius-based System model editor:

1. Choose »Add Architectural Template« from the palette and click on the »defaultSystem« to apply a template on your system.
2. In the popup, choose the »AT Static Assembly Context Loadbalancing« Architectural Template to apply the template for a load balancer. Confirm with »OK«.

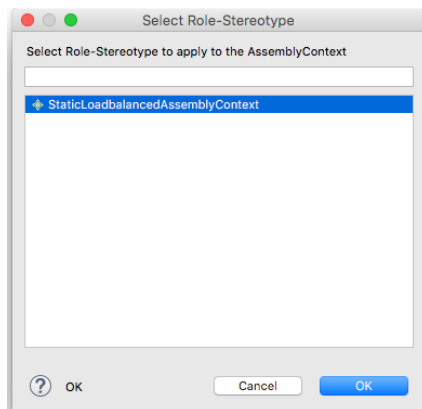


3. The system is now enabled for the load balancer Architectural Template. Whenever we want to work with an Architectural Template, enabling it for the system is the first step. The editor

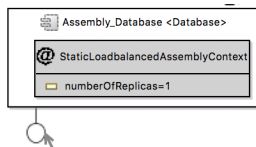
for the system illustrates the Architectural Template application with a grey box.



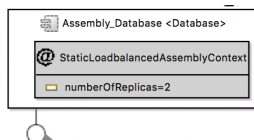
- Finally, we need to mark the Assembly Context that has to be load-balanced. To do so, choose »Add Role« from the palette and click on the »Assembly_Database <Database>« Assembly Context. In the popup, choose the »StaticLoadbalancedAssemblyContext« role and confirm with »OK«.



- The Assembly Context will now be load-balanced during simulation (note: you need to start the simulation using Experiment Automation; see Sec. 13). The editor for the system illustrates this added Role with a grey box within the Assembly Context.



- The grey box of the added Role allows to set the »Number of Replicas« of the load balancer, i.e., the number of Assembly Context for our database that will be load-balanced. Set it to »2« (by doubleclicking the parameter) to model semantically the same system as we did in Sec. 9 with our manually modeled load balancer.



- Use Experiment Automation (Section 14) to run analyzes based on Architectural Templates.

The complete catalogue of existing Architectural Templates is explained at:

<http://wiki.cloudscale-project.eu/index.php/HowTos>

As a final task, use this Wiki:

- Go to <http://wiki.cloudscale-project.eu/index.php/HowTos>
- Click on »Loadbalancing« and read the page.
- Answer this question: What is the difference between »Loadbalanced Resource Container« and »Loadbalanced Assembly Context«?

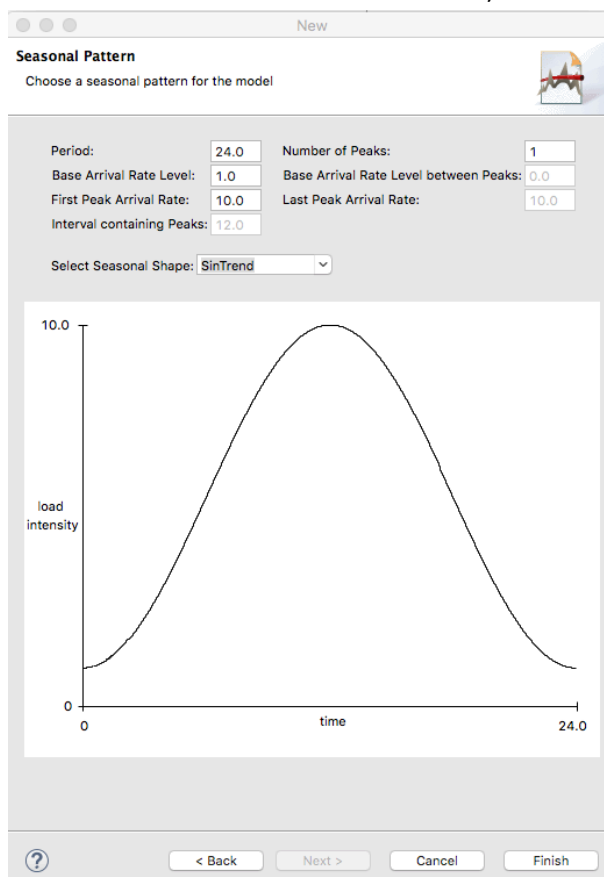
12. Specifying Usage Evolutions [Estimated Time: 15 min.]

Screencast: <https://www.youtube.com/watch?v=k6EKqPyl2Jg>

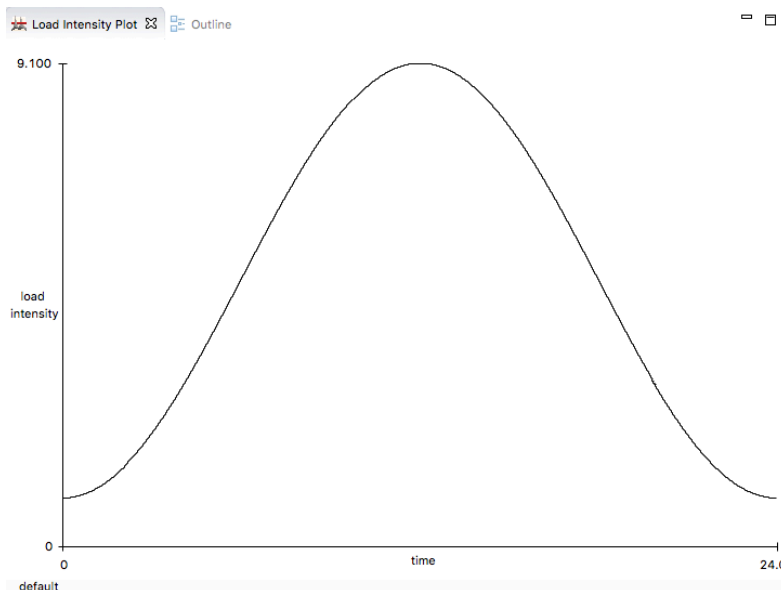
So far, we only modeled *static* usage scenarios. That is, our usage scenarios may used probabilistic characterisations, however, these did not vary over time (thus, the »static«). In contrast, a *dynamic* usage scenario would be characterized as a function of (simulated) time.

We recently filled this gap by introducing so-called Usage Evolution models, i.e., models that allow to vary usage scenario parameters like arrival rates, number of concurrent users, operation parameters, etc. over time. To create such models, a normal usage scenario model, a Descartes Load Intensity Model (DLIM) model to characterize time-dependent variations, and a Usage Evolution model that links the former two are needed:

1. Create a new DLIM model (Right-click at your project -> New -> Other... -> Descartes Load Intensity Model -> Descartes Load Intensity Model -> Next -> File Name: »default.dlim« -> Next -> Only mark »Modify Seasonal Part« -> Next -> Only modify »Number of Peaks« to »1«, »Base Arrival Rate Level« to »1.0«, and »Select Seasonal Shape« to »SinTrend« as shown in the screenshot below -> Finish).



2. You now created a DLIM model where you vary »load intensity« over »time«. You can visualize the your function by opening the »Load Intensity Plot« (Window -> Show View -> Other... -> Descartes Load Intensity Model -> Load Intensity Plot) and by opening your model in the default editor (double-click on your model if the editor is not already opened).



Note that the »9.100« appears to be a bug in the visualization; it should rather be a »10«.

- Next, let us link this DLIM model to our existing usage model such that we vary the number of users from 1 to 10 until half of the simulation time is over and from 10 to 1 until simulation finishes.

Create a new Usage Evolution model (Right-click at your project -> New -> Other... ->

CloudScale Diagrams -> ScaleDL Usage Evolution -> Next -> File Name:

»default.usageevolution« -> Next -> Browse... and select your pre-specified usage model

(»/PalladioProject/default.usagemodel«) -> Next -> Browse... and select your pre-specified DLIM model »/PalladioProject/default.dlim« -> Finish).

- An editor opens that has everything needed pre-configured (see screenshot below). You may investigate the properties of the »Usage Initial« node to see how we linked DLIM and usage models. Per default, we created a »Load Evolution« for the »defaultUsageScenario« based on the »Sequence default« DLIM model, i.e., we vary the number of users within our open workload (our wizard creates such a configuration by default).

The screenshot shows the Palladio IDE interface. At the top, there are tabs for 'default.usagemodel_diagram', '*default.dlim', and 'default.usageevolution'. Below the tabs is a 'Resource Set' section with a tree view showing the following resources:

- platform:/resource/PalladioProject/default.usageevolution
 - Usage Evolution Evolution
 - Usage Initial
 - platform:/resource/PalladioProject/default.dlim
 - platform:/resource/PalladioProject/default.usagemodel
 - platform:/resource/PalladioProject/default.system
 - platform:/resource/PalladioProject/default.repository
 - pathmap://PCM_MODELS/PrimitiveTypes.repository
 - pathmap://PCM_MODELS/Palladio.resourcetype

Below the tree view is a 'Selection' section with tabs for 'Parent', 'List', 'Tree', 'Table', and 'Tree with Columns'. The 'Tree' tab is selected. At the bottom, there is a 'Prop' tab with a table of properties for the 'Usage Initial' node:

Property	Value
Entity Name	Initial
Evolution Step Width	1.0
Id	_OZtsalkOEeWHIKKPU4cWmg
Load Evolution	Sequence default
Repeating Pattern	false
Scenario	Usage Scenario defaultUsageScenario

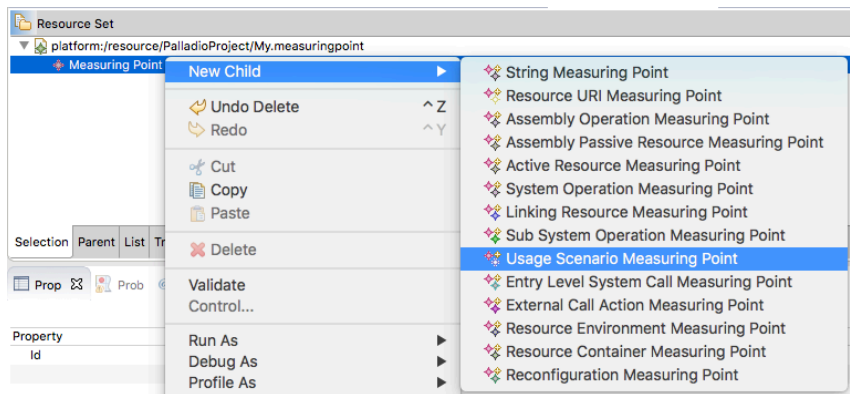
- Use Experiment Automation (Section 14) to run analyzes based on Usage Evolutions.

13. Specifying Monitor Repositories [Estimated Time: 20 min.]

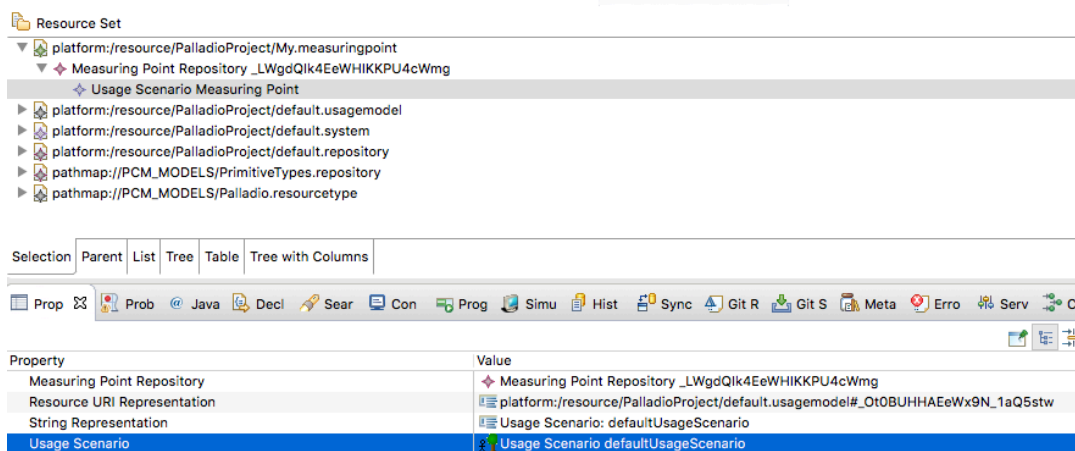
Ever wondered how we determine what comes out of your simulation? Typically, we use some defaults like usage scenario response times. However, if you need special measurements or do not want to see some default measurements, you can also take full control about that via Monitor Repository models! That is, such models allow you to specify what you are interested in and what should then be measured.

For specifying monitor repositories, you first have to specify a measuring points model, which specifies *where* you want to take measurements (e.g., at the usage scenario). Afterwards, we can specify a monitor repository model that states *what* we want to measure (e.g., response times) at such measuring points:

1. Create a new Measuring Point model (Right-click at your project -> New -> Other... -> Example EMF Model Creation Wizards -> Measuringpoint Model -> Next -> File Name: »My.measuringpoint« -> Next -> Model Object: »Measuring Point Repository« -> Finish).
2. In the now opened tree editor, add a new measuring point for our usage scenario (right click the »Measuring Point Repository« node -> New Child -> Usage Scenario Measuring Point).



3. Next, configure *our* usage scenario as »Usage Scenario« for this Usage Scenario Measuring Point (drag & drop our usage model into the editor to load the model («platform://resource/PalladioProject/default.usagemodel« will appear at the bottom of the editor) -> select »Usage Scenario defaultUsageScenario« as »Usage Scenario« within the properties view)

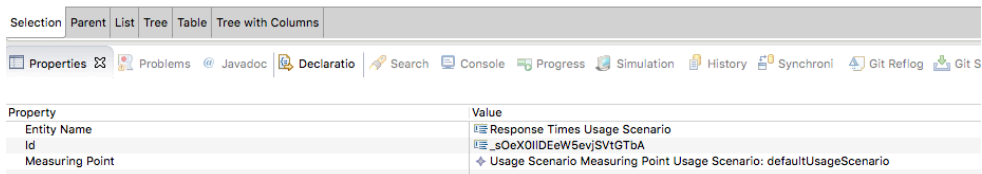
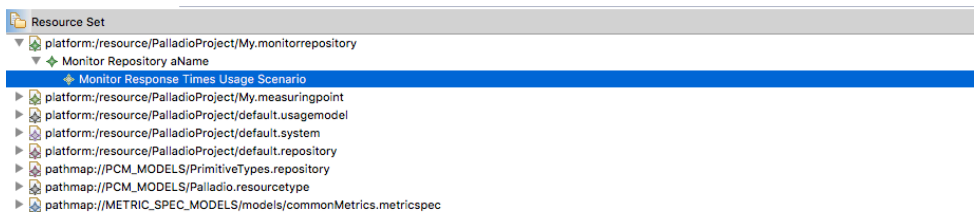


4. Save & close.

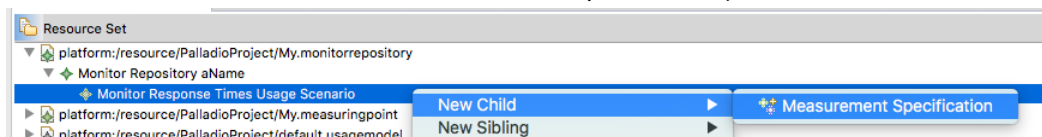
5. Create a new Monitor Repository model (Right-click at your project -> New -> Other... -> Example EMF Model Creation Wizards -> MonitorRepository Model -> Next -> File Name: »My.monitorrepository« -> Next -> Model Object: »Monitor Repository« -> Finish).
6. In the now opened tree editor, add a new monitor for our usage scenario (right click the »Monitor Repository« node -> New Child -> Monitor).



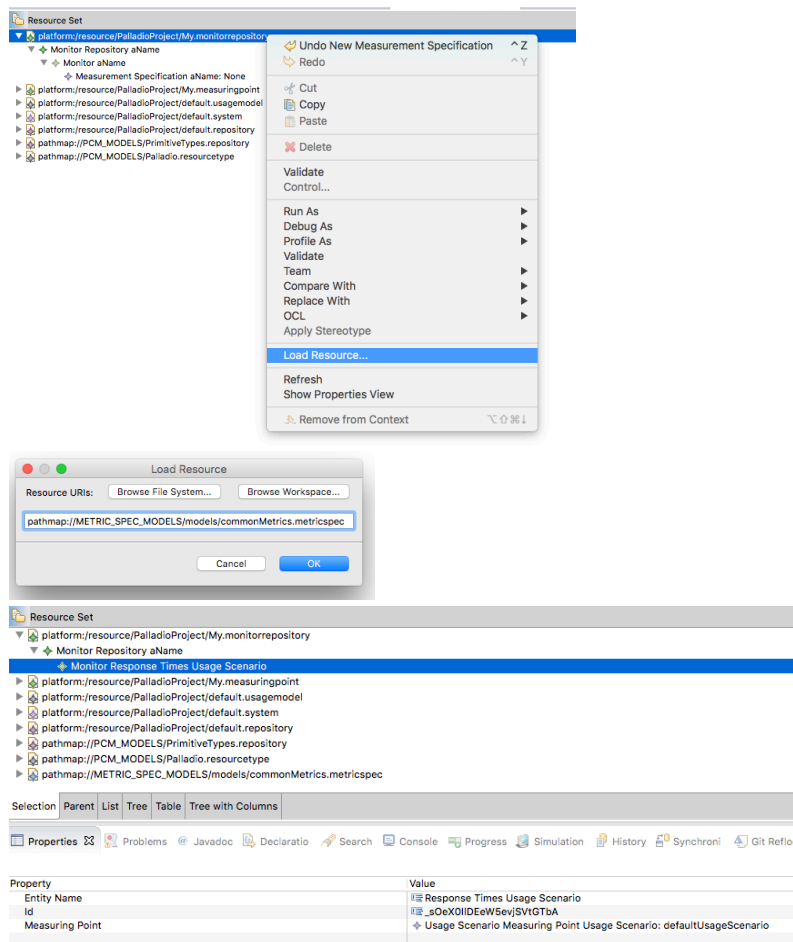
7. Next, configure our usage scenario measuring point as »Measuring Point« for this Monitor (drag & drop our measuring point model into the editor to load the model (»platform://resource/PalladioProject/My.measuringpoint« will appear at the bottom of the editor) -> select »Usage Scenario Measuring Point Usage Scenario: defaultUsageScenario« as »Measuring Point« and »Usage Scenario Response Times« as »Entity Name« within the properties view)



8. Add a new measurement specification for response times to our monitor (right click the »Monitor« node -> New Child -> Measurement Specification).



9. Next, configure Response Times as metrics we are interested in (right-click on the editor -> Load Resource... -> enter »pathmap://METRIC_SPEC_MODELS/models/commonMetrics.metricspec« as »Resource URI« -> OK (»pathmap://METRIC_SPEC_MODELS/models/commonMetrics.metricspec« will appear at the bottom of the editor) -> select »Numerical Base Metric Description Response Time« as »Metric Description« within the properties view)



10. Use Experiment Automation (Section 14) to run analyzes based on Monitor Repositories.

14. Using Experiment Automation [Estimated Time: 20 min.]

Screencast: <https://www.youtube.com/watch?v=sJSqil9Pwz4>

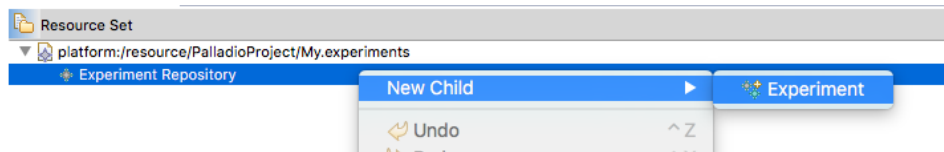
Prerequisite: You specified a monitor repository model (Section 13).

Optional: Experiment Automation optionally supports models with applied Architectural Templates (Section 11) and Usage Evolutions (Section 12).

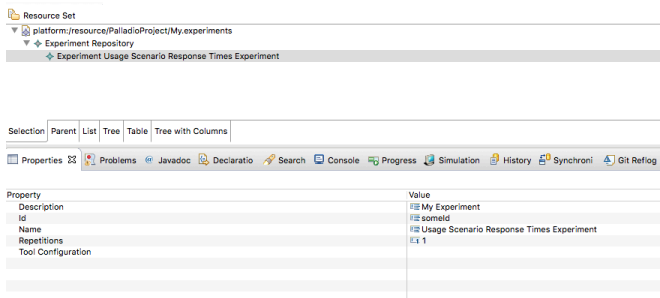
So far, we always used a »SimuBench« run configuration to start a Palladio-based analysis. Such analyses use Palladio's SimuCom simulator for measuring performance metrics. However, SimuCom run configurations lack support for Architectural Templates, Usage Evolutions, and Monitor Repositories. A dedicated model for runconfigurations – Experiment Automation models – solve this problem.

1. Create a new Experiment Automation model (Right-click at your project -> New -> Other... -> Example EMF Model Creation Wizards -> Experiments Model -> Next -> File Name: »My.experiments« -> Next -> Model Object: »Experiment Repository« -> Finish).

2. In the now opened tree editor, add a Experiment for our Experiment Repository (right click the »Experiment Repository« node -> New Child -> Experiment).



3. Next, configure our Experiment (select »My Experiment« as »Description«, »someId« as »Id«, »Usage Scenario Response Times Experiment« as »Name« and »1« as »Repetitions« within the properties view).



4. In this way, we also have to configure some more child nodes and the »Tool Configuration« to configure an experiment run. As especially the »Tool Configuration« is quite tricky, we stop with our description for manual configuration here.
5. Instead, we now copy an existing Experiment model into our project and only adapt it where we need it. Please do so with a Text Editor (!!) using this experiment model:
<https://github.com/CloudScale-Project/ArchitecturalTemplates/blob/master/plugins/org.scaledl.architecturaltemplates.examples.dynscalingcontainer/Experiments/Elasticity.experiments>
6. Run the experiment (open your run configurations -> double-click »Experiment Automation« -> choose your Experiments model as input -> Run)

15. Adding Variables [Estimated Time: 30 min.]

With Palladio, it is also possible to specify variables to characterize method input parameters as well as their return values. For such specifications, Palladio allows to also model dataflow (besides control flow). See the corresponding screencast at <http://www.palladio-simulator.com/tools/screencasts/> for a tutorial on this. This workshop may be extended by a detailed explanation in future versions.

Versions

- V2.1 (2015/11/12; Palladio 4.0; Eclipse 4.5.1/Mars.1): Added new Sirius editors, Architectural Templates, Usage Evolution, Monitor Repository, and Experiment Automation; Sebastian Lebrig (sebastian.lebrig@informatik.tu-chemnitz.de)
- V2.0 (2015/10/08; Palladio 4.0; Eclipse 4.5.1/Mars.1): Revised complete guide to provide up-to-date version; added time estimates; moved from Sensor Framework to EDP2 descriptions; Sebastian Lebrig (sebastian.lebrig@informatik.tu-chemnitz.de)
- V1.3 (2015/09/17; Palladio 4.0; Eclipse 4.5/Mars): Updated to recent Palladio version; Sebastian Lebrig (sebastian.lebrig@informatik.tu-chemnitz.de)

- V1.2 (2013/08/14; Palladio 3.4; Eclipse 4.2/Juno): Corrected typos; Sebastian Lehrig (sebastian.lehrig@uni-paderborn.de)
- V1.1 (2013/01/07; Palladio 3.4; Eclipse 4.2/Juno): Exchanged introduction image to a default Palladio image from website; Sebastian Lehrig (sebastian.lehrig@uni-paderborn.de)
- V1.0 (2012/11/16; Palladio 3.4; Eclipse 4.2/Juno): Initial version; Sebastian Lehrig (sebastian.lehrig@uni-paderborn.de)