Optimization and calibration

All OpenMI compliant components can be accessed through a standardized interface (the OpenMI.Standard. ILinkableComponent interface). One obvious way to take advantage of this is to create generic optimization or calibration components. Such components can be designed in such a way that any OpenMI compliant numerical model can be optimized or calibrated by use of such component. However, such optimization or calibration component will use the OpenMI.Standard.IManageState interface, which is not required in the standard. So, if you are interested in using your optimization of calibration component for a specific OpenMI compliant component please check if this component implements the IManageState interface.

Since the design pattern will be the same for optimization and calibration only calibration is explained below.

It is assumed that your calibration component can be used within any OpenMI configuration using any OpenMI configuration editor (e.g. the configuration editor provided the OpenMI Association Technical Committee). So, your calibration component must be an OpenMI compliant component it selves (implement the OpenMI.Standard.IlinkableComponent interface).

In the example below the calibration component is used to calibrate a conceptual rainfall-runoff model. However, please notice that the calibration component can be used for any type of model.

When calibrating a rainfall-runoff model you typically run the model several times. After each run you compare the simulated runoff time series with the corresponding measured runoff time series. Based on this comparison you will estimate new model parameters and run again. This procedure is repeated until you have a satisfactory agreement between measured and simulated runoff.

When you create (implement) your calibration component, you should make sure that this component has an input exchange item that can connect to the simulated runoff and an input exchange item that can connect to the measured runoff. Your component should also have output exchange items from where the RR-model can retrieve the improved parameters. In order to make your component work for any model you can (e.g.) name the input exchange items "simulated" and "measured", and your output exchange items" parameter 1", "parameter 2", "parameter 3", etc.

In figure 1 below the example configuration for the calibration of the RR model is shown. The calibration component is an OpenMI compliant component and appears, as the remaining components in the configuration, as a yellow box. The calibration component will get the measured runoff from an OpenMI compliant time series component (1). Alternatively, the calibration component could get these data from a file or other sources. The calibration component provides model parameter to the RR-model through an OpenMI link (2) and obtains simulated runoff through another OpenMI link (3). Finally, the calibration component is linked to a trigger, which will drive the whole thing forward.

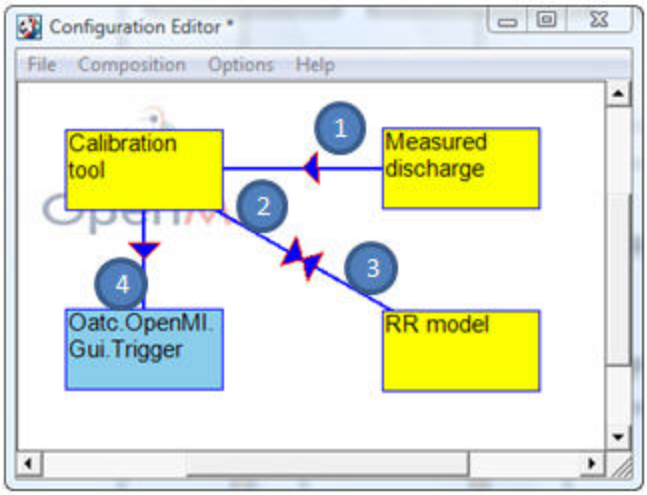

Figure 1: Example configuration for calibration of a rainfall-runoff model.

The sequence diagram in figure 2 shows the details of the communication between the components during the calibration.

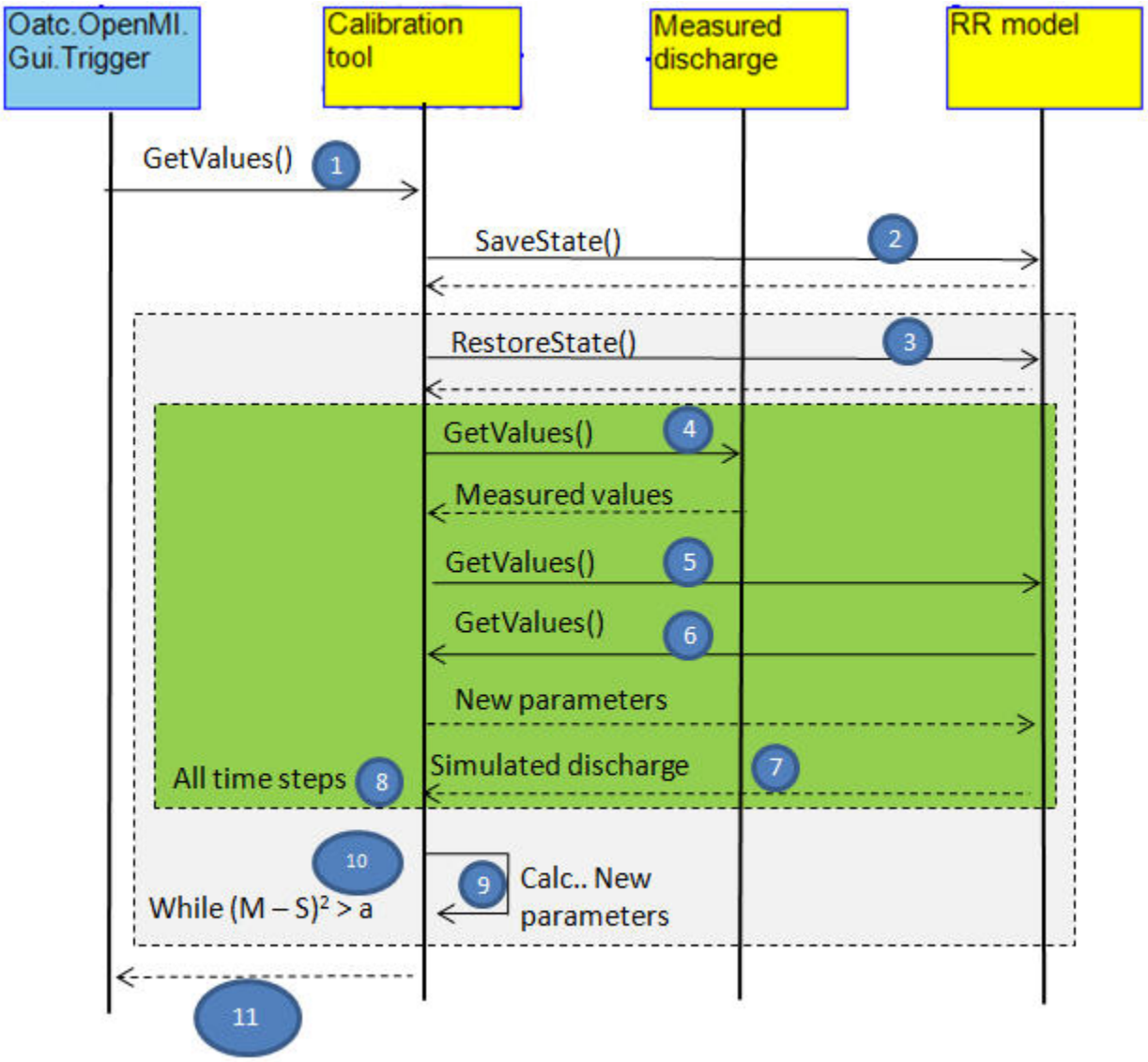

Figure 2 : Sequence diagram for calibration of a rainfall-runoff model.

1. The trigger invokes the GetValues method in the calibration component. The trigger could be linked to a dedicated trigger output item in the calibration component, which simply, when asked for data, will trigger the component to perform the calibration.

2. The calibration component invokes the SaveState method (OpenMI.Standard.IManageState interface) in the RR-model. This will enable the calibration component to restore this state after each model run.

3. The calibration loop has now been entered. The calibration component will invoke the RestoreState method in the RR-model.

4. The calibration component will invoke GetValues in the time series component in order to obtain the measured runoff for the first calibration time step. If the time series component implements the optional OpenMI.Standard.IDiscreteTimes interface, the calibration component can use those times for the time stepping. Alternatively, the calibration component will have to define its own time steps for the calibration.

5. The calibration component will invoke GetValues in the RR-model in order to retrieve the simulated runoff for the same time as the measured runoff.

6. Since the RR-model is configured to get some of its parameters through the OpenMI link from the calibration component, it will invoke GetValues in the calibration component in order to retrieve those. The sequence diagram only shows one link (corresponding to calibration only one parameter). However, if several parameters are to be calibrated, multiple links can be established. Typically, such parameters are not time dependent. So, asking for these before each calculation timestep in the RR-model is not needed. However, the simplest approach is to transfer these at each time step, and the performance hit for this will be relatively low.

7. The RR-model can now perform as many time steps as needed in order to progress until the time for which it was asked for a simulated runoff value and return this value.

8. The steps 3 through 7 are repeated until the simulated and measured values for the whole simulation period has been obtained.

9. The calibration component will based on the differences between measured and simulated runoff values and information from previous iterations use its internal calibration algorithms to estimate new model parameters.

10. Steps 3 through 9 are repeated until a satisfactory agreement between simulated and measured values is achieved.

11. The calibration algorithm returns to the trigger and the calibration is completed.