23 Decision Module

Decision Module

What	*nameofinstance*.xml
Description	Configuration for the Decision module
schema location	https://fewsdocs.deltares.nl/schemas/version1.0/decisionModule.xsd

Contents

Overview

The Decision Module in Delft-FEWS is used to implement decision logic and evaluation for barriers. With this module we can iteratively evaluate configurated decision rules. The configuration file of the Decision Module contains the definition of one or more Decision Trees. These Decision Trees defined in the Decision Module are associated with a Barrier definition which are defined by the Barriers configuration file.

Some important prerequisites

Barrier states should be defined

The decision logic (criteria) is linked to the barrier state. Barrier states could be for example "the barrier is open", "the barrier is closed", "the barrier is halfway closed", "at a stage where some additional criteria need to be evaluated", etc. For each of these states, separate decision logic may be relevant and should be evaluated.

Input from present and future model state

While evaluating the decision logic, relevant input may consist of information from both the present and future model state.

Some examples to illustrate the above:

- When the barrier is open, and when the forecast results show that water levels at location A will exceed 3 meters, the barrier should start closing when the water level at the barrier passes the 2 meter mark.
- When the barrier is closed, the barrier should be opened when the local water level gradient is negative.

Scalable

Furthermore, the decision logic should be "scalable" (i.e. it should be easy to add additional rules). If we continue with the above example, a decision rule could also be

- When the barrier is open, and when the forecast results show that water levels at location A will exceed 3 meters, the barrier should close. If the river discharge at location C is below 6000 m3/s, the barrier should start closing when the local water level passes the 2 meter mark. If the river discharge at location C exceeds the 6000 m3/s, the barrier should start closing at slack tide. The barrier is only allowed to close when the shipping through the location B has been blocked in advance.

Two types of criteria

With regard to decision logic, we can differentiate between two different types of criteria in the above example:

1. Criteria which indicate *that* a barrier state change should occur (for example, a state change from "the barrier is open" to "the barrier is closed"). For example, the forecast water level at location A exceeds 3 meters.
2. Criteria which indicate *when* a barrier state change should occur.
   For example, the local water level passes the 2 meter mark. (Note that this criteria is conditional to the example 1).

As the decision logic takes both "future" information (if there is a high water event in our forecast horizon, do ...) and information not included in the model state into account (in the above example, both the discharge at location C and the "shipping state" are not included state of the running model), we cannot evaluate the decision logic based on triggers in this model. As such, we want to do this in an external module (i.e. FEWS in this case).

When evaluating the decision logic, it is relevant to take into account the fact that the model state will change after the barrier state has been updated (changed). To evaluate the decision logic for subsequent steps in the process, this implies that we will need to re-run the model to take this state change into account. Also, if there are multiple barriers in our area of interest, this implies that if the state of one of these barriers changes, we need to update our model simulation before we can assess the decision logic for the other barrier(s).

The decision evaluation process

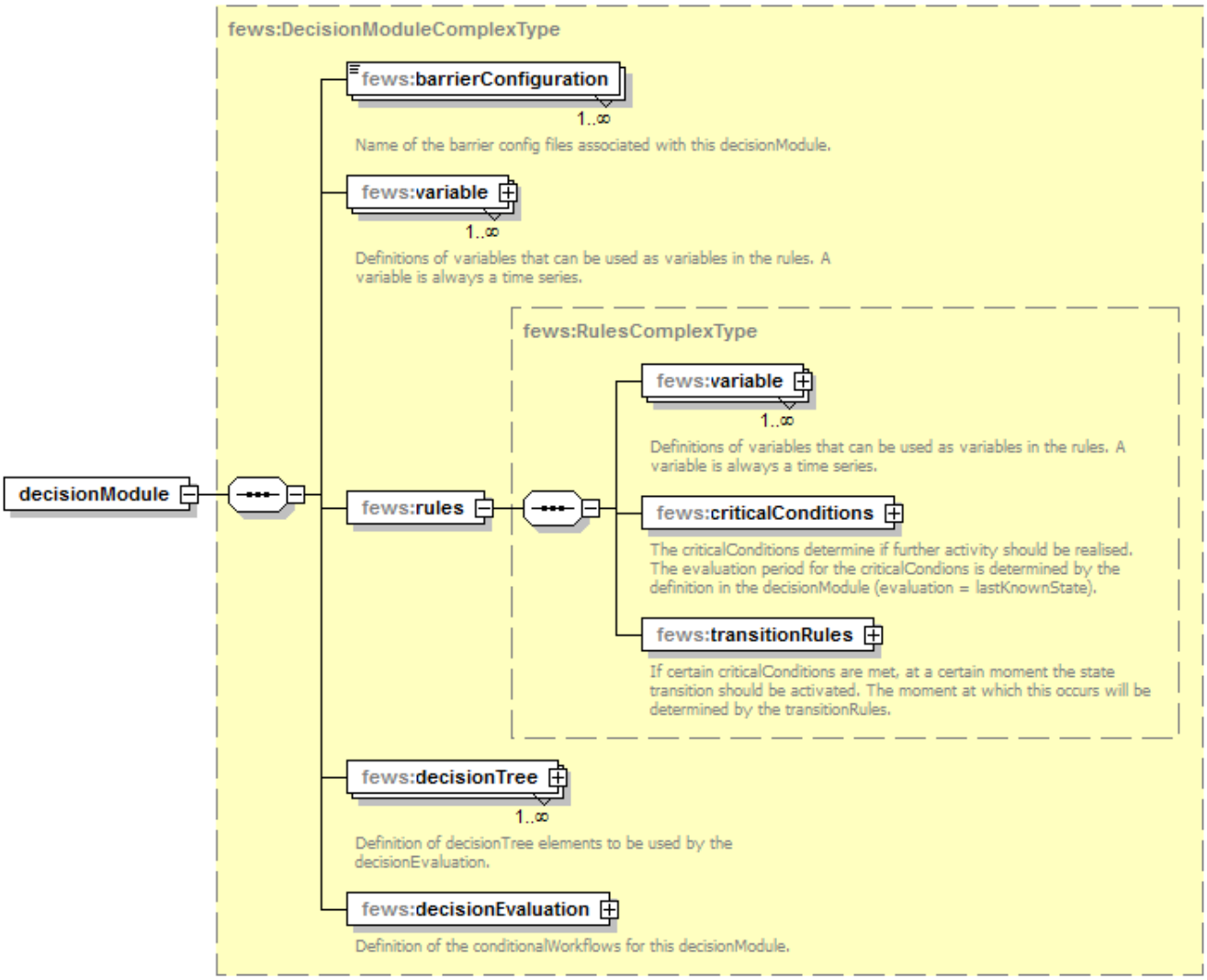
The entire process can be summarized as follows:

1. Run a baseline simulation (forecast) taking the actual state of the barriers as a starting point.
2. Evaluate the decision logic based on the baseline simulation.
3. If relevant criteria are met, the barrier state changes. Here, we distinguish between criteria which indicate *if* a state change is required. And criteria which indicate *when* a state change is required.
4. Run a new simulation taking the barrier state change into account.
5. Evaluate the decision logic of this simulation. (Note that the decision logic will only be evaluated over the period following the latest barrier state change.)
6. Loop this process starting from step 3 until no relevant criteria are met and therefore no state change is required.

While the decision logic is model independent, the model should be fed with the appropriate timeseries representing the appropriate barrier states (for example timeseries of crest height, crest level and gate height for various barrier elements).

The configuration files are based around timeseries representing the barrier state, which are used and updated in the evaluation process. Each value in this timeseries represents a model state. For example, if the value of this series is 0, this indicates the barrier is open. If the value of this series if 1, this indicates the barrier is closed. Etc. This section only describes the configuration of the Decision module file. For an explanation of Barriers configuration file, go to the Barriers section.

Configuration



Variables definition

All variables within the Decision Module are time series in the form of Time Series Sets. Within the Decision Module various items make use of a variableId. This variableId is used in the actual section as an alias to that time series.
The identifier assigned to a time series should contain alphanumeric characters only. It should not start with a numerical character.

variableId

required element used to identify the time series in the decisionModule block or in the rules block.

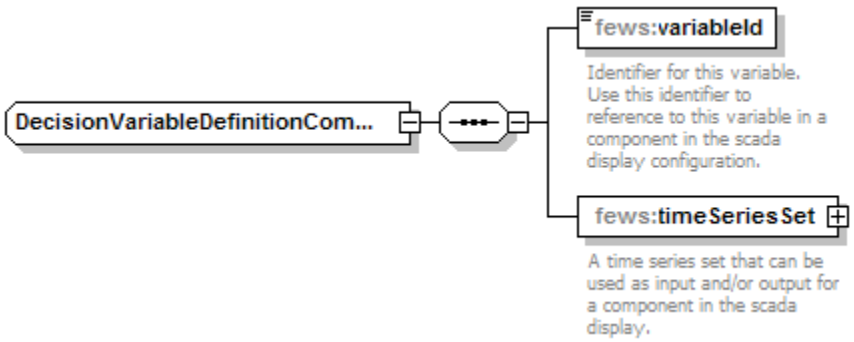
timeSeriesSet

required element used to identify the timeSerieSet.

Example variable definition

```
<variable>
        <variableId>SVKW.position.in</variableId>
        <timeSeriesSet>
                <moduleInstanceId>RMM_Structures_Forecast_Processing_Extrapolate</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>State.position.meting</parameterId>
                <locationId>SVKW</locationId>
                <timeSeriesType>simulated forecasting</timeSeriesType>
                <timeStep unit="minute" multiplier="10"/>
                <relativeViewPeriod unit="day" start="0" end="1" endOverrulable="true"/>
                <readWriteMode>add originals</readWriteMode>
        </timeSeriesSet>
</variable>
<variable>
        <variableId>SVKW.position.dss</variableId>
        <timeSeriesSet>
                <moduleInstanceId>RMM_DecisionTree</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>State.position.meting</parameterId>
                <locationId>SVKW</locationId>
                <timeSeriesType>simulated forecasting</timeSeriesType>
                <timeStep unit="minute" multiplier="10"/>
                <relativeViewPeriod unit="day" start="0" end="1" endOverrulable="true"/>
                <readWriteMode>add originals</readWriteMode>
        </timeSeriesSet>
</variable>
```

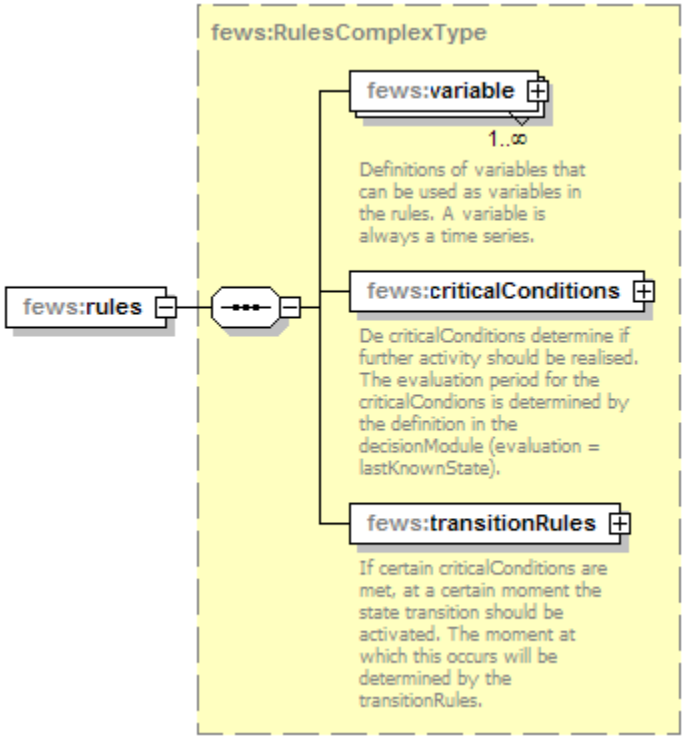
Schema definition



Rules definition

The Rules section defines the rules which will be used by the decision entries of each decisionTree.

Schema definition



variables

This section contain a set of variables defined only for the rule definitions. For a description of the variables see the section above.

criticalConditions

The criticalConditions determine if further activity should be realized. The evaluation period for the criticalCondions is determined by the definition in the decisionModule (evaluation = lastKnownState).

The definition of the criticalConditions reuses the criticalConditionLoopUp of the Lookup Table Module.

transitionRules

If certain criticalConditions are met, at a certain moment the state transition should be activated. The moment at which this occurs will be determined by the transitionRules.

DecisionTrees definition

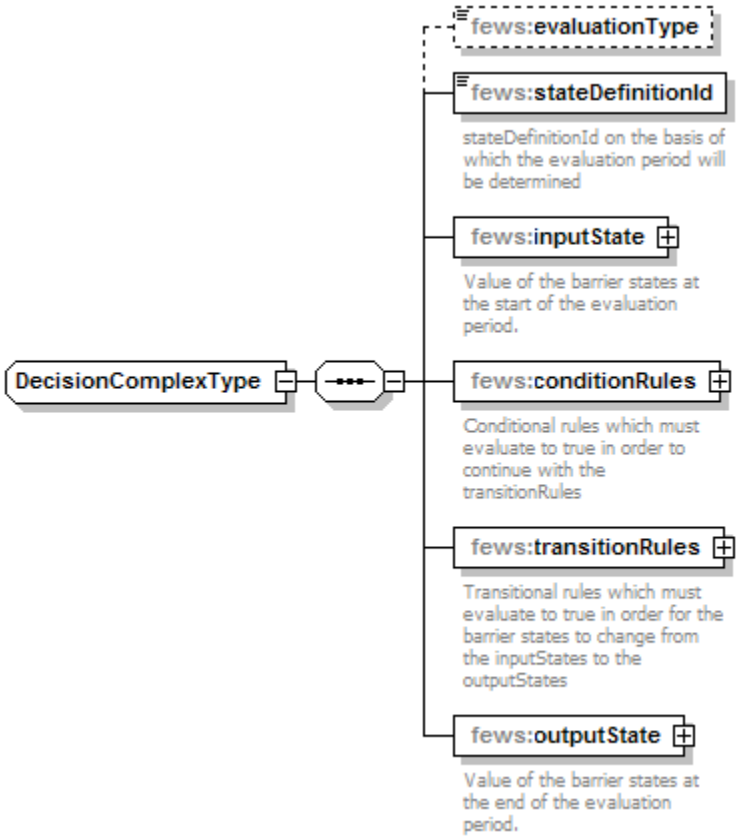
For each barrier a decisionTree is configured. Each decision element in this decisionTree should be unique (i.e. at each moment in time only one element can be valid).

Example decistionTree

```
<decisionTree>
        <barrierId>scheepvaart</barrierId>
        <decision id="Stremming (peilsluiting)">
                <evaluationType>lastKnownState</evaluationType>
                <stateDefinitionId>scheepvaart</stateDefinitionId>
                <inputState>
                        <stateValueId>geen stremming</stateValueId>
                </inputState>
                <conditionRules>
                        <allValid>
                                <anyValid>
                                        <isTrue>H.rotterdam.d</isTrue>
                                        <isTrue>H.dordrecht.d</isTrue>
                                </anyValid>
                                <isFalse>Q.lobith</isFalse>
                        </allValid>
                </conditionRules>
                <transitionRules>
                        <anyValid>
                                <isTrue>scheepvaart.stremming.peilsluiting.rotterdam</isTrue>
                                <isTrue>scheepvaart.stremming.peilsluiting.dordrecht</isTrue>
                        </anyValid>
                </transitionRules>
                <outputState>
                        <stateValueId>gestremd</stateValueId>
                </outputState>
        </decision>
        <decision id="Stremming (kenteringsluiting)">
           ...
        </decision>
        <decision id="Vrijgeven scheepvaart">
           ...
        </decision>
</decisionTree>
```

Decision definition

Schema definition



evaluationType

stateDefinitionId

inputState

conditionRules

transitionRules

outputState

DecisionEvaluation

The last section of this file deals with the evaluation of the decision logic. If we are in the first step of the iterative loop, the last know system (barrier) state is used as initial value. The section within the 'initialConditionalWorkflow' tag is run, where the timeSeriesSets associated with the variableId's are used for initial values. From this workflow the (external) model is called (for example, a workflow through which Sobek is run from the FEWS General Adapter).

Example decisionEvalution

```
<decisionEvaluation>
        <initialConditionalWorkflow>
                <variableId>SVKH.position.in</variableId>
                <variableId>SVKW.position.in</variableId>
                <variableId>scheepvaart.in</variableId>
                <workflowId>Sobek_DSS_Forecast</workflowId>
        </initialConditionalWorkflow>
        <conditionalWorkflow>
                <variableId>SVKH.position.dss</variableId>
                <variableId>SVKW.position.dss</variableId>
                <variableId>scheepvaart.dss</variableId>
                <stateChanges>
                        <evaluationType>FirstInTime</evaluationType>
                        <stateChange>
                                <decisionTreeId>SVKH</decisionTreeId>
                        </stateChange>
                        <stateChange>
                                <decisionTreeId>SVKW</decisionTreeId>
                        </stateChange>
                        <stateChange>
                                <decisionTreeId>scheepvaart</decisionTreeId>
                        </stateChange>
                </stateChanges>
                <workflowId>Sobek_DSS_Forecast</workflowId>
        </conditionalWorkflow>
</decisionEvaluation>
```

After running the model for the first time, we need to evaluate the decision logic prior to a (possible) second run. If a state change occurs in the decisionTree, we need to run the model again taking this state change into account. From the second iteration and onwards, the section within the 'conditionalWorkflow' tag will be run. Note that we need to evaluate three state changes in this case (SVKH = the position of the Hartelkering, SVKW = the position of the Maeslantkering and scheepvaart = the "shipping state"), each of which has a separate decisionTree definition.

evaluationType

If one state variable changes value, this will change the overall system state, and hence may effect the evaluation of the other state variables. There are to options which can be defined:

- All: all state variable changes will be taken into account in the next iteration.
- FirstInTime: if there is a state change in more than one state variable, only the state change which occurs first in time should be taken into account in the next iteration. After a new iteration with the model, the other state values will be re-evaluated in this case.