RTC-Tools

Software Tools for Modeling Real-Time Control

Reference Manual

Dirk Schwanenberg, Bernhard Becker

Version: 0.1.22313

9 December 2012

RTC-Tools, Reference Manual

Published and printed by:

Deltares	telephone:	+31 88 335 82 73
Rotterdamseweg 185	fax:	+31 88 335 85 82
p.o. box 177	e-mail:	info@deltares.nl
2600 MH Delft	www:	http://www.deltares.nl
The Netherlands		

Contact:

RTC-Tools is still in beta version. If you have suggestions or ideas for enhancing the software, please contact:

Bernhard Becker		Dirk Schwanenberg	
telephone:	+31 88 335 8507	telephone:	+31 88 335 8447
fax:	+31 88 335 8582	fax:	+31 88 335 8582

e-mail:	rtc-tools@deltares.nl
www:	http://oss.deltares.nl/web/rtc-tools

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

The RTC-Tools (Real-Time Control Tools) software package originates from the integration of several project-specific reservoir simulation modules in flood forecasting systems for river basins in Austria, Germany and Pakistan. Its original design in Java in 2007, also referred to as the Delft-FEWS Reservoir Module, aims at the simulation of pool routing in reservoirs including related feedback controllers and operating rules.

Features for supporting a sequential, nonlinear version of Model Predictive Control (MPC) were introduced in 2008 and extended in 2009 and beyond. This includes the implementation of simplified hydraulic models such as the kinematic wave or diffusive wave models as additional internal model for the predictive controller as well as the introduction of adjoint systems for selected modeling components. The latter resulted in significant speed-ups of these controllers.

In 2010, the concept of triggers for switching on and off controllers and operating rules was introduced for enabling the simulation of more sophisticated heuristic control schemes. The software was redesign in C++ and enhanced by a C# OpenMI / DeltaShell wrapper for integration into modeling packages such as SOBEK or Delft3D. Furthermore, the new formats for saving states and externalize parameters were introduced for compliant with OpenDA for implementing automatic calibration and data assimilation applications.

In 2011, the software has been integrated further into Delta Shell for replacing existing RTC functionality in SOBEK in the context of Deltares' Next Generation Hydrosoftware development.

The decision to release RTC-Tools within an open source project has been made in 2012.

1.2 Fields of application

The RTC-Tools package aims at the simulation of various real-time control techniques in application to water resources systems. It includes feedback control strategies with triggers, operating rules and controllers as well as advanced Model Predictive Control (MPC) setups based on a combination of forecasting and optimization.

The MPC-based predictive control strategies require internal modeling of the controlled water system in the optimization procedure. Therefore, the package includes a number of simple hydrological and hydraulic models as well as various other simulation components for setting up water resources models. The model implementation reflects the requirements of the optimization procedure by supplying both a simulation mode and an adjoint mode for computing first-order model derivatives. Having in mind its application in operational forecasting systems, the software pays special attention to state handling. This includes by definition the system states of all existing components such as triggers, controllers, operating rules and modeling components.

RTC-Tools is designed for stand alone use or serves as a building block in a larger system architecture. We pay special attention to various interfaces for integration it into frameworks such as Delft-FEWS, Matlab or OpenDA. Furthermore, the OpenMI interface enables its on-

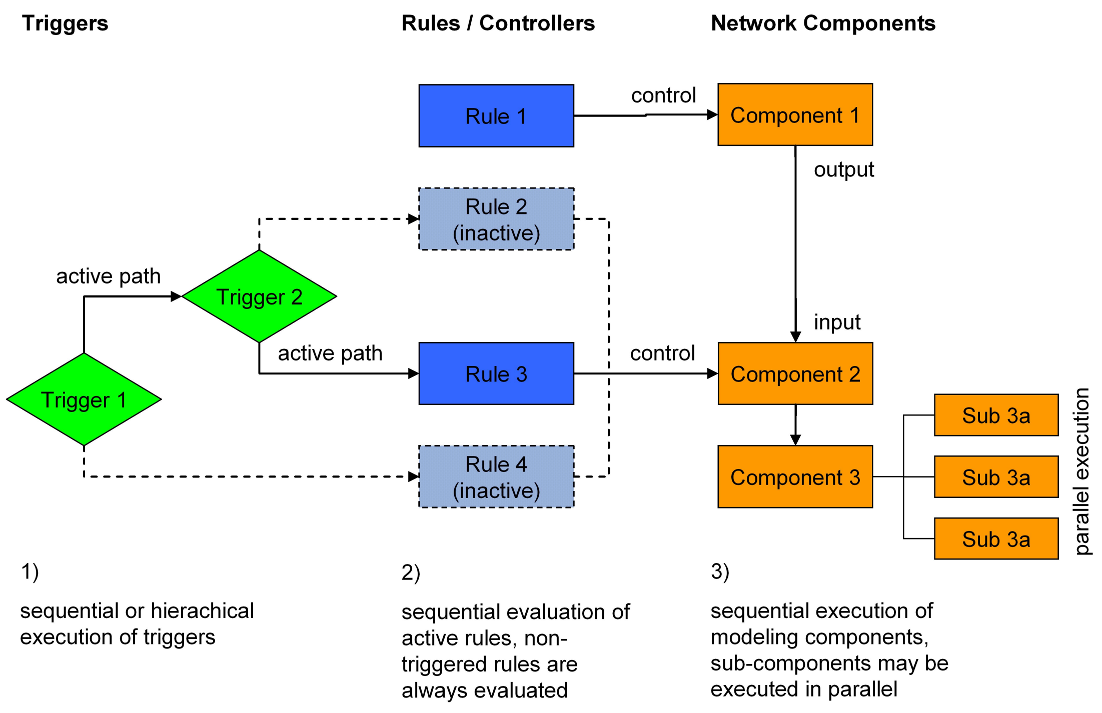


Figure 1.1: *RTC-Tools configuration in typical simulation set-up: triggers, reactive operating rules and controllers, network components*

line coupling to a wide range of hydraulic modeling packages.

The following examples present three typical applications of RTC-Tools:

- Stand-alone use as a forecasting model in Delft-FEWS:
  Assume we require a forecasting model for a medium-sized river basin including an HBV-style conceptual rainfall runoff model, a simple hydraulic routing components and the integration of several controlled flood detention polders. Although many other models supply the required features, the application of RTC-Tools can be advantageous, because it enables the complete representation of the system in a single model (Figure 1.1) and integrates seamless into Delft-FEWS with a minimum configuration effort and a maximum of interaction.
- Integrated application with sophisticated hydraulic model via OpenMI:
  Typical hydraulic models such as SOBEK, Mike11 or HEC-RAS have on-board features for modeling the real-time control of hydraulic structures. If more advanced features beyond the available ones are required, APIs may enable the user to link external code (Figure 1.2). A main advantage of RTC-Tools against a dedicated user-programming is the availability of a wide range of already existing and tested features, the option for extending or modifying them easily, and the overall framework with file io and interfaces.
- Predictive control of hydraulic structures:
  In particular in forecasting system, MPC provides an advanced option for supervisory control and decision-making for example for scheduling pump actions in polder systems or the release of reservoirs. The set-up consisting of an optimization of the hydraulic structure by MPC included the embedded representation of the water resources system (Figure 1.3).

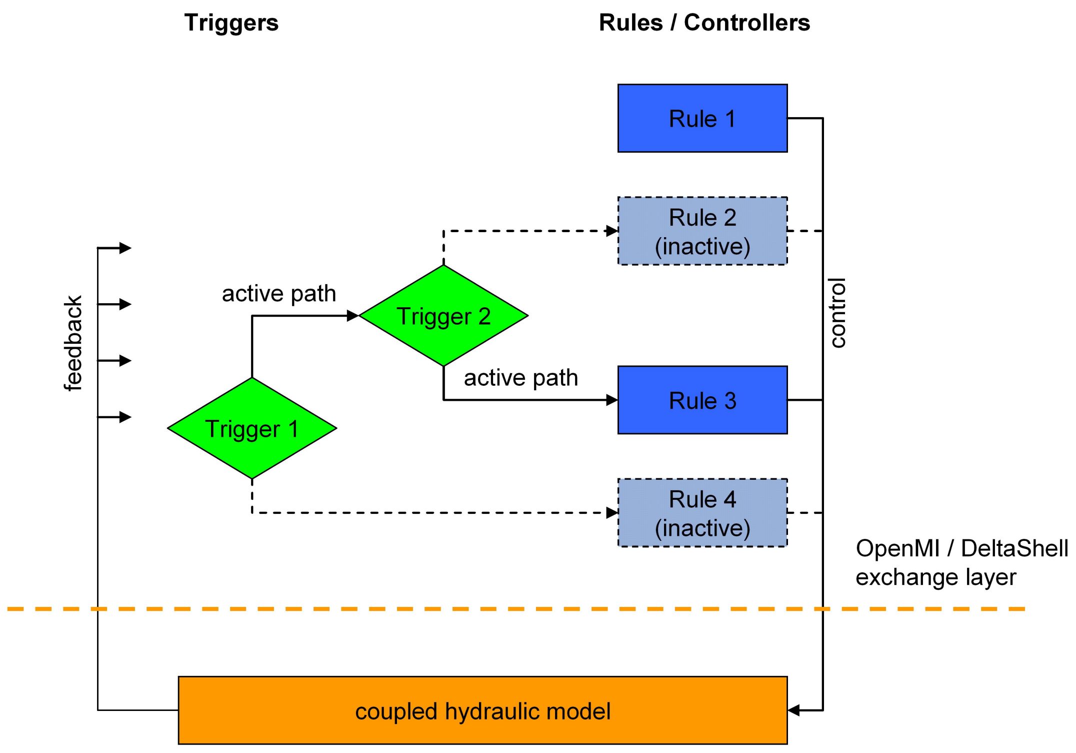


Figure 1.2: *RTC-Tools configuration in typical simulation set-up: triggers, reactive operating rules and controllers linked with a hydraulic model via OpenMI*

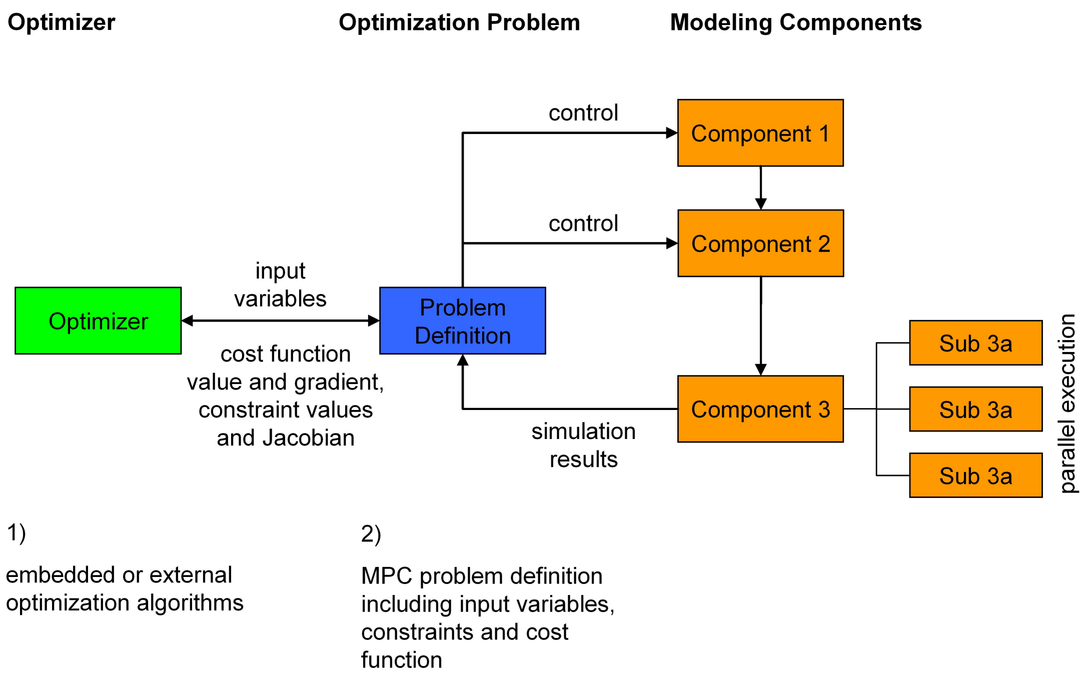


Figure 1.3: *RTC-Tools configuration in typical Model Predictive Control set-up: optimizer, optimization problem definition and network components*

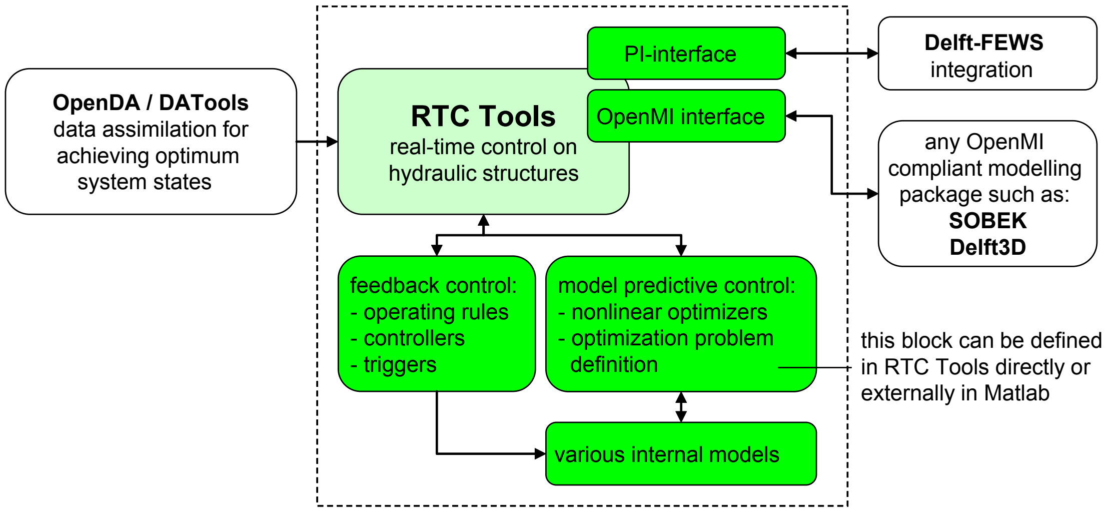

OpenDA / DATools
data assimilation for achieving optimum system states
RTC Tools
real-time control on hydraulic structures
PI-interface
OpenMI interface
Delft-FEWS
integration
any OpenMI compliant modelling package such as:
SOBEK
Delft3D
feedback control:
- operating rules
- controllers
- triggers
model predictive control:
- nonlinear optimizers
- optimization problem definition
this block can be defined in RTC Tools directly or externally in Matlab
various internal models

Figure 1.4: *Architecture and interfaces of RTC-Tools*

1.3 Architecture and supported interfaces

The software does not aim at a detailed simulation of large water resources systems. Therefore, it does not include any GUI or case management support. However, it is intended to solve specific RTC-related problems which we may want to integrate into larger models or forecasting systems. To support this feature, the software provides interfaces to the forecasting system Delft-FEWS (PI-XML interface) and modeling packages such as SOBEK and Delft3D via OpenMI. The latter enables a user to combine the operating rules and controllers of RTC-Tools with more detailed hydraulic modeling components founds in the hydraulic modeling packages mentioned above (see Figure 1.4).

The modeling components of RTC-Tools includes another important interface to OpenDA / DA Tools for applying data assimilation techniques in order to improve the system state of the modeling components. This feature may be used in the context of forecasting and predictive control application for improving the system state at forecast time and therefore also the lead time accuracy of the forecast itself.

The set-up of MPC can be done in RTC-Tools directly by using one of the integrated optimizers. Alternatively, a user may choose to externalize this part and use his preferred optimizer and optimization problem definition in Matlab.

1.4 Content of this document

This chapter gives an overview about RTC-Tools. We describe its main components and show how it used in typical settings as: a) a stand alone simulation model in a Delft-FEWS forecasting system, b) a MPC application for controlling a water system, c) a real-time control component linked via OpenMI to a simulation package such as SOBEK.

Chapter 2 provides background on the modeling components, governing equations and its numerical schematization. Chapter 3 covers the hierarchy of triggers and controllers / operating rules for setting up feedback control of a water system. In Chapter 4, we present the concept of Model Predictive Control (MPC).

Chapter 5 presents the integration of RTC-Tools into Delft-FEWS via the PI-XML interface and modeling packages such as SOBEK via OpenMI. The configuration of RTC-Tools in xml is discussed in Chapter A.

2 Modeling components

2.1 Overview

2.2 Reservoir

2.2.1 Mathematical model

The basic equation for pool routing in a reservoir is:

$$\frac{\mathrm{d}s(h)}{\mathrm{d}t} = I - Q \tag{2.1}$$

where

I	inflow into the reservoir
Q	release of the reservoir
s	storage in the reservoir (state variable)
h	water level in the reservoir
t	time.

We assume the relation between storage $s(h)$ and water level h to be an arbitrary function or a piecewise linear lookup table.

The release Q from the reservoir can be further spitted into an into a controlled release Q_c and an uncontrolled release Q_u according to

$$Q(h, dg) = Q_c(h, dg) + Q_u(h) \tag{2.2}$$

where dg is the setting of a hydraulic structure. Whereas the controlled release is a function of the water level h (under assumption that its maximum capacity depends also on the water level in the reservoir) and the setting of the structure dg. The uncontrolled release is only a function of the reservoir's water level h representing for example an uncontrolled spillway with a fixed crest level.

2.2.2 Numerical schematization

The explicit schematization, also referred to as Forward Euler, for the pool routing equation reads

$$s(h^k) = s(h^{k-1}) + \Delta t(I^k - Q_c^k(h^{k-1}, dg^k) - Q_u^k(h^k)) \tag{2.3}$$

and is conditionally stable for sufficiently small time steps. The unconditionally stable implicit version reads

$$\begin{aligned} s(h^k) = s(h^{k-1}) + \Delta t((1-\theta)I^{k-1} + \theta I^k \\ -(1-\theta)Q_c^{k-1}(h^{k-1}, dg^{k-1}) - \theta Q_c^k(h^k, dg^k) \\ -(1-\theta)Q_u^{k-1}(h^{k-1}) - \theta Q_u^k(h^k)) \end{aligned} \tag{2.4}$$

where $0.5 \leq \theta \leq 1.0$ is a time weighting coefficient shifting from a more accurate second-order Crank-Nicholson scheme for $\theta = 0.5$ to a more robust, first-order Backward Euler scheme for $\theta = 1.0$.

Table 2.1: *Available triggers, reactive operating rules and controllers and network components*

Triggers	Rules / Controllers	Modeling Components
deadBand	constant	gradient
deadBandTime	dateLookupTable	hydraulicModel
polygonLookup	deadBandValue	node
spreadsheet	dedicated-Aebi	branch
standard	dedicated-EifelRur	hydraulicStructure
	dedicated-ThunerSee	reservoir
	guideBand	reservoirConnection
	hydraulicController	routing
	limiter	unitDelay
	merger	
	minSimple	
	multiplier	
	pid	

2.3 Hydrological routing

2.3.1 Introduction

2.3.2 Routing

A subsequent connection of multiple reservoirs allows to model a simple routing process. This approach is a so-called water balance model or 0-dimensional model. Check section Section **??** for its description and numerical implementation.

2.3.3 Unit delay

The unit delay operator is an auxiliary tool for making data from time steps prior to the previous time step available in the simulation. By using this operator, we can refer to a historical release, for example in an operating rule, without abandoning the restarting features of the model based on the system state of a single time step. It reads

$$y^{k+1} = x^k \tag{2.5}$$

2.3.4 Unit hydrograph

Unit hydrograph provides a rainfall-runoff modeling based on the concept of the unit hydrograph.

2.3.5 HBV model

2.3.6 Lorent Grevers

2.4 Hydraulic routing

2.4.1 Introduction

Hydraulic routing, compared to the hydrological routing techniques presented above, is a more accurate approach and may include the simulation of hysteresis and backwater effects. On the other hand, hydraulic routing has more demands with respect to the numerical solution, computational effort, and may become unstable if not properly implemented and set-up.

RTC-Tools includes a hydraulic routing method based on a mixed kinematic and diffusive wave approach. The most relevant decisions to make are the choice of an appropriate routing (kinematic / diffusive), spatial schematization (central / upwind) and time stepping scheme (explicit / implicit). The following section provides some hints on choosing the proper model and how to set it up.

2.4.2 Kinematic wave model

2.4.2.1 Numerical background

The flow in one dimension is described by the De Saint-Venant equations consisting of mass (continuity) and momentum conservation. The continuity equation reads:

$$\frac{\partial A(h)}{\partial t} + \frac{\partial Q}{\partial x} = q_{lat} \tag{2.6}$$

while the non-conservative form of the momentum equation is defined by:

$$\frac{\partial v}{\partial t} + v\frac{\partial v}{\partial x} + g\frac{\partial h}{\partial x} = -c_f\frac{v\,|v|}{m}, \qquad c_f = \frac{g}{C^2} \tag{2.7}$$

with

A	wetted area [m^2]
Q	discharge [m^3/s]
q_{lat}	lateral discharge per unit length [(m^3/s)/m]
h	water level [m above reference level]
v	flow velocity [m/s]
g	acceleration due to gravity [m/s^2]
m	hydraulic radius [m] (may be approximated to water depth for large rivers)
C	Chézy coefficient [m$^{1/2}$/s]
c_f	dimensionless bottom friction coefficient [-].

The kinematic wave equations can be derived from the complete system (2.6), (2.7) by neglecting the terms for inertia (term 1) and convection (term 2). By additional substitution of *v* = *Q*/*A* equation (2.7) becomes

$$g\frac{\partial h}{\partial x} = -\frac{gQ\,|Q|}{C^2A^2m} \tag{2.8}$$

and can be converted to

$$Q = -\mathrm{sign}(\frac{\partial h}{\partial x})CA\sqrt{\left|\frac{\partial h}{\partial x}\right|m} \tag{2.9}$$

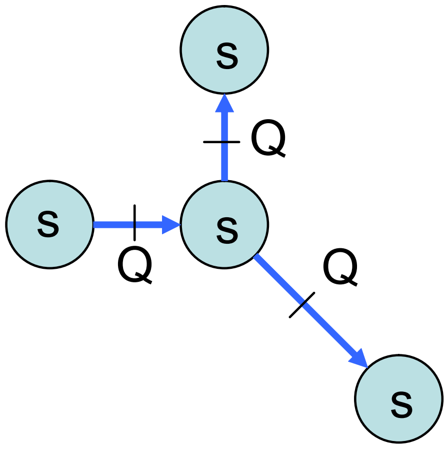


Figure 2.1: *Spatial schematization of the kinematic wave model on a staggered grid*

The continuity equation (2.6) stays unchanged. Under assumption of a representative cross section for a river reach, both variables *A* and *m* become a geometrical function of water level *h*. By equalizing water level and energy head, hydraulic structures are represented by a simplified structure formula with the general form

$$Q = f(h_{up}, h_{down}, dg) \tag{2.10}$$

in which *dg* = gate or weir setting (in case of a controlled structure).

The hydraulic structures are modeled by the following formulas for a weir and an orifice with fully opened gates

$$Q = \begin{cases} \frac{2}{3}\mathrm{w_s}\sqrt{\frac{2}{3}g}(\mathrm{h_{up}}-\mathrm{z_s})^{3/2}, & \text{if } h_{up}-\mathrm{z_s}>\frac{3}{2}(\mathrm{h_{down}}-\mathrm{z_s}) \\ \mathrm{w_s}(h_{down}-z_s)\sqrt{2\mathrm{g}(\mathrm{h_{up}}-\mathrm{h_{down}})}, & \text{otherwise} \end{cases} \tag{2.11}$$

in which w_s = width of the structure, z_s = crest level. In the case of a partially or fully closed gate ($h_{up} - z_s \geq \frac{3}{2}dg$), we apply

$$Q = \begin{cases} \mathrm{w_s}\mu\, dg\sqrt{2g(h_{up}-z_s-\mu\, dg)}, & \text{if } h_{down} < \mathrm{z_s}+\mathrm{dg} \\ \mathrm{w_s}\mu\, dg\sqrt{2\mathrm{g}(\mathrm{h_{up}}-\mathrm{h_{down}})}, & \text{otherwise} \end{cases} \tag{2.12}$$

in which *dg* = gate setting, μ = contraction coefficient.

2.4.2.2 Schematization

The spatial schematization of the kinematic wave model is done on a staggered grid. Computation nodes include the state variable storage *s*. Branches always connect two nodes and include the auxiliary variable discharge *Q* (a major difference to the full hydraulic model where *Q* is another state variable).

The numerical solution of the continuity equation (2.6) is resulting in:

$$\frac{A(h^{k+1}) - A(h^k)}{\Delta t} + \frac{Q_{down}^{k+1} - Q_{up}^{k+1}}{\Delta x} = q_{lat}^{k+1} \tag{2.13}$$

By substituting $S(h) = A(h)\Delta x$, we may transform equation (2.26) into a water balance in the domain of a node and get

$$s(h^{k+1}) = s(h^k) + \Delta t(Q_{up}^{k+1} - Q_{down}^{k+1} + Q_{lat}^{k+1}) \tag{2.14}$$

in which s = storage at a node (state variable), Q_{lat} is the total lateral inflow into the domain of the node.

The discharge in a flow branch is schematized based on equation (2.9) by

$$Q^{k+1} = f(h^k_{down}, h^k_{up}) = -\mathrm{sign}(\frac{h^k_{down} - h^k_{up}}{\Delta x})C(\bar{h}^k)A(\bar{h}^k)\sqrt{\left|\frac{h^k_{down} - h^k_{up}}{\Delta x}\right|}m(\bar{h}^k), \quad (2.15)$$

in which $\bar{h}^k = \frac{h^k_{down} + h^k_{up}}{2}$

In a branch with a hydraulic structure, the flow branch is replaced by the formula of the hydraulic structures modeled by an arbitrary equation in the form

$$Q^{k+1} = f(h^k_{down}, h^k_{up}, dg^{k+1}) \quad (2.16)$$

Stability of this method turns out to be reasonable as long as the Courant-Friedrichs-Lewy (CFL) condition is fulfilled.

2.4.3 Diffusive wave model

2.4.4 Kinematic versus diffusive wave routing

The fact that the diffusive wave model takes into account more terms of the full dynamic Saint Venant model does not mean that it is always preferred over the simpler kinematic wave approach. Simplicity and computational performance of the latter may have advantages; in particular if results of both approaches are the same, e.g. in river reaches with steep gradients.

The following aspects may guide you to the proper approach:

- Is the slope of your river reach smaller than xxx?
- Is backwater a relevant effect you need to consider?
- Do you want to consider hysteresis?

If you answer one of these questions with “Yes”, consider the diffusive wave approach. Otherwise, you may try the kinematic wave method first. Note that you can mix both methods by defining the related tag in each flow branch.

2.4.5 Model set-up

2.4.5.1 Schematization of the water network

The spatial schematization of your routing network is a trade-off between accuracy (a higher resolution means more accuracy) and CPU time. For flow routing purposes only, already a very course spatial schematization may achieve the required accuracy from the control point of view. We recommend the following procedure for defining your routing network:

1. Indicate all nodes you need to include: boundary conditions, upstream and downstream node of hydraulic structures, stream flow gauges, bifurcations and confluences, nodes with lateral inflows or extractions (can be also lumped into neighboring nodes).
2. Place additional nodes between the existing ones, if required for accuracy.

2.4.5.2 Level-storage relation

All nodes require a level-storage relation. One way to get those is a detailed analysis of the surrounding channel network (typically halfway to the next node) in terms of the area and storage at different elevations. An easier and often sufficient option is the selection of a typical cross section at the node and its multiplication by the length of the reaches around. If you selected the upwind schematization (see section 3.4.3), take the cross section you are using in the flow branch.

If you routing network includes flooding and drying, take care that the lowest level of your level-storage table or equation in your node is lower than the lowest elevation in the cross sections around. Since the model implementation does not include any dedicated procedure for flooding and drying, violating this condition may result in negative water depth at nodes and problems with robustness. If you use the upwind schematization together with the level-storage relation derived from it, this condition is already fulfilled.

2.4.5.3 Cross-section

Flow branches require a cross section. You may again aim at deriving an aggregated cross section from the available data of the branch. The simpler approach is again the selection of a typical cross section. Depending on the spatial schematization (see 3.4.3), select a typical cross section close to the upstream node for the upwind option and one halfway along the branch for the central option.

2.5 Other models

2.5.1 Accumulation

2.5.2 Arma

The arma model (just an ar-model at the moment) reads

$$
\begin{aligned}
e^{k+1} &= \begin{cases} x_{sim}^{k+1} - x_{obs}^{k+1}, & \text{if } x_{obs}^{k+1} \text{ is available} \\ c_{ar} e^{k}, & \text{otherwise} \end{cases} \\
y^{k+1} &= x_{sim}^{k+1} + e^{k+1}
\end{aligned}
\tag{2.17}
$$

where x_{obs} is an (external) observation, x_{sim} a simulation, e is the difference between observation and simulation, c_{ar} is the auto regression coefficient.

2.5.3 Expression

The expression consists of a mathematical expression of the following form:

$$y^{k+1} = x_1^k + x_2^k \tag{2.18}$$

The following operators are supported:

$+$ (summation)
$-$ (subtraction)
$*$ (multiplication)
$/$ (division)

$\min$ (minimum)
$\max$ (maximum).

The recursive use of expressions (another expression as one of the two terms or both) enables the implementation of more complex mathematical expressions (check the example in the configuration section).

2.5.4 Gradient

The governing equation of the gradient reads:

$$y^{k+1} = \frac{x^{k+1} - x^k}{\Delta t} \tag{2.19}$$

2.5.5 Neural Network

Define a neural network using the following equations for the neuron sum and the (nonlinear) transfer function:

$$\begin{aligned} y_\mu^k &= \textstyle\sum_{v=1}^{\mu-1} w_{\mu,v}^{\text{neuron}} x_v^k + \sum_{v=\mu}^{K} w_{\mu,v}^{\text{neuron}} x_v^{k-1} + \sum_{v=1}^{L} w_{\mu,v}^{\text{input}} u_v^k \\ x_\mu^k &= h_\mu \left(y_\mu^k \right) \end{aligned} \tag{2.20}$$

where:

x_ν^k	is the value of neuron v at time step k.
y_μ^k	is the weighted sum of inputs to neuron μ at time step k.
$w_{\mu,v}^{\text{neuron}}$	is the weighting given to neuron v when calculating the sum for neuron μ.
$w_{\mu,v}^{\text{input}}$	is the weighting given to input v, when calculating the sum for neuron μ.
u_ν^k	is the value of input v at time step k.
K	is the number of neurons.
L	is the number of inputs.
h_μ	is the activation function for neuron μ.

Note that the for $\mu = 1$, the first summation term in (2.20) is empty and hence should be taken as zero.

2.6 Numerical schemes

2.6.1 Ordinary differential equations

Most network components in RTC-Tools can be described by a set of non-linear ordinary differential equations (ODE) according to:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x, u, d) \tag{2.21}$$

where $x \in \mathbb{R}^l$ is the system state vector, $u \in \mathbb{R}^m$ the vector of controlled variables, $d \in \mathbb{R}^n$ the vector of disturbances, *l* the number of states, *m* the number of controlled variables, and *n* the number of disturbances. The equation above can be transformed into a discrete state space formulation by applying a finite difference quotient for the time derivative resulting in

$$\frac{x^{k+1} - x^k}{\Delta t} = f(x^{k,k+1}, u^{k,k+1}, d^{k,k+1}) \tag{2.22}$$

where the indices *k* and *k*+1 denote the discrete values at two subsequent time steps and Δt is the time interval $t^{k+1} - t^k$. Note that the function *f*( ) may include values both at the old time step *k* as well as the new time step *k*+1. In case of the controlled variable **u** and the disturbance **d**, this does not have any impact on the further numerical schematization, because these variables are considered as external inputs. However, the presence of the state vector **x** at time step *k*+1 at the right hand side will require an iterative solution of the equation system in most cases.

2.6.2 Forward Euler scheme

A straightforward option for further schematization of the equation above is the forward Euler approach, a simple explicit scheme according to

$$\frac{x^{k+1} - x^k}{\Delta t} = f(x^k, u^{k+1}, d^{k+1}) \Leftrightarrow x^{k+1} = x^k + \Delta t\, f(x^k, ...) \tag{2.23}$$

Dropping the term x^{k+1} enables us to solve the state space system without iterations. The scheme has first-order accuracy and performs well, if the CFL stability condition is satisfied. The latter is mainly achieved by limiting the maximum time step, which can become a limiting factor depending on the process model.

Note that we use **u** and **d** at the new time step *k*+1 which is uncommon in an explicit scheme. This is done by intention to obtain consistency in supplying data to different time stepping schemes and optionally incorporate the results of reactive controllers u^{k+1}, computed before at the same time step.

2.6.3 Implicit scheme

The theta scheme is an implicit scheme according to:

$$\frac{x^{k+1} - x^k}{\Delta t} = f\begin{pmatrix} \theta x^k + (1-\theta)x^{k+1}, \\ \theta u^k + (1-\theta)u^{k+1}, \\ \theta d^k + (1-\theta)d^{k+1} \end{pmatrix} \tag{2.24}$$

where θ is the time weighting factor. The scheme is unconditionally stable for a weighting factor of $0.5 \leq \theta \leq 1$. The scheme becomes second order accurate for $\theta = 0.5$ and fully implicit, although first order, for $\theta = 1$.

By rearranging equation (2.24) to

$$h(x^{k+1}) = \frac{x^{k+1} - x^k}{\Delta t} - f\,(...) = 0 \tag{2.25}$$

the equation system can be solved numerically by a Newton-Raphson algorithm with back-tracking according to

$$x^{k+1}_{l+1,m} = x^{k+1}_l - r^m \frac{h(x^{k+1}_l)}{h'(x^{k+1}_l)} \tag{2.26}$$

in which *l* is the outer Newton-Raphson iteration counter, and *m* is the inner backtracking counter. The backtracking factor *r* is reduced in the inner loop by a configurable backtracking reduction coefficient c_r by

$r^{m+1} = c_r r^m$ with $0 < c_r < 1$

The term $h(x_l^{k+1})/h'(x_l^{k+1})$ becomes a simple division in case of single equation and requires the solution of a system of linear equations in case of a set of equations. The following pseudo code points out the detailed workflow of the algorithm for a single equation.

```
k = 0
xk = xOld
//Newton-Raphson loop
do{
        k++
        r = 1.0
//backtracking loop
        do{
                xkl = xk-r*f(xk)/fDer(xk)
                r = backtrackingReduction*r;
        }
        while{
                abs(f(xkl))>abs(f(xk)) & r>backtrackingLimit
        }
        xk = xkl
}
while{
        abs(f(xk))> fTol & k < maxIter
}
xNew = xk
```

2.6.4 Discussion

2.6.4.1 Central versus upwind spatial schematization

The central approach, i.e. the water level in a flow branch is the mean of the upstream and downstream node, is more accurate. The upwind schematization is more robust. From a technical point of view, you can easily switch between the two, but keep in mind that its choice has some impact also on the definition of the level-storage relations at the node level and the cross sections in the flow branches.

In combination with a kinematic wave model, we recommend to use the upwind schematization. In case of a diffusive wave flow branch, use also the upwind schematization if there is a clear flow direction, otherwise, if water flows in both directions, the central schematization is better. If you still don't know what you should do, use the upwind schematization.

2.6.4.2 Explicit versus implicit time stepping

The explicit time stepping is a simple and fast numerical scheme. The CFL condition (todo: equation) needs to be satisfied for the flow branches and hydraulic structures for keeping it stable. This can be a severe restriction, if some of your network nodes are close together. In this case you may aggregate these nodes or proceed with the implicit time stepping scheme.

The implicit time stepping is unconditional stable, thus, allowing any time step. However, one issue you may face is a failure of the solver to fully converge. Therefore, have an eye on the optional residuum output.

If you set-up a model, our practical suggestion is as follows:

1 First choose an explicit time stepping with a very small time step. It is helpful to indicate bugs in your schematization, e.g. no data values will propagate from its origin into the network. Implicit models may not converge in this case and finding out the problem can become a difficult task.
2 Increase the time step and watch out for instabilities of the solution (wiggles going up and down from one time step to the other). Choose the final time step to be about 50
3 Switch to the implicit time stepping and a larger time step you may need from you application perspective and check if it performs faster. If so, proceed with the implicit time stepping.

3 Feedback control

3.1 Overview

3.2 Operating rules

3.2.1 constant

This simple rule defines a user-defined constant output *y* according to

$$y^{k+1} = \text{constant} \tag{3.1}$$

It is typically applied in combination with triggers (see next section) for switching between several defined states of a structure, e.g. fully opened or fully closed.

3.2.2 dateLookupTable

The date lookup table is a 2D lookup table with the time axis on one of the dimension. It reads

$$y^{k+1} = f(t, x^{k,k+1}) \tag{3.2}$$

The resolution of the time axis *t* is in days of the year. The value axis *x* may have any range. A typical application of the rule would be the definition of a minimum release of a reservoir as a function of the season and the water level of the reservoir.

3.2.3 deadBandValue

The dead band value rule is a discrete rule for suppressing the output of another rule until the change of an output becomes higher than a certain threshold. It reads

$$x^{k+1} = \begin{cases} x^k & \text{if } \left|x^{k+1} - x^k\right| < \text{threshold} \\ x^{k+1} & \text{otherwise} \end{cases} \tag{3.3}$$

It is often applied to limit the number of adjustments to movable elements of hydraulic structures in order to increase their life time.

3.2.4 dedicated-Aebi

This rule is dedicated to the representation of the so-called Aebi rule for controlling the lake Bieler in Canton Bern, Switzerland (check appendix A).

3.2.5 dedicated-Thunersee

This rule is dedicated to the water level control of lake Thun in Canton Bern, Switzerland (check appendix A).

Table 3.1: *Available triggers, reactive operating rules and controllers and network components*

Triggers	Rules / Controllers	Modeling Components
deadBand	constant	gradient
deadBandTime	dateLookupTable	hydraulicModel
polygonLookup	deadBandValue	node
spreadsheet	dedicated-Aebi	branch
standard	dedicated-EifelRur	hydraulicStructure
	dedicated-ThunerSee	reservoir
	guideBand	reservoirConnection
	hydraulicController	routing
	limiter	unitDelay
	merger	
	minSimple	
	multiplier	
	pid	

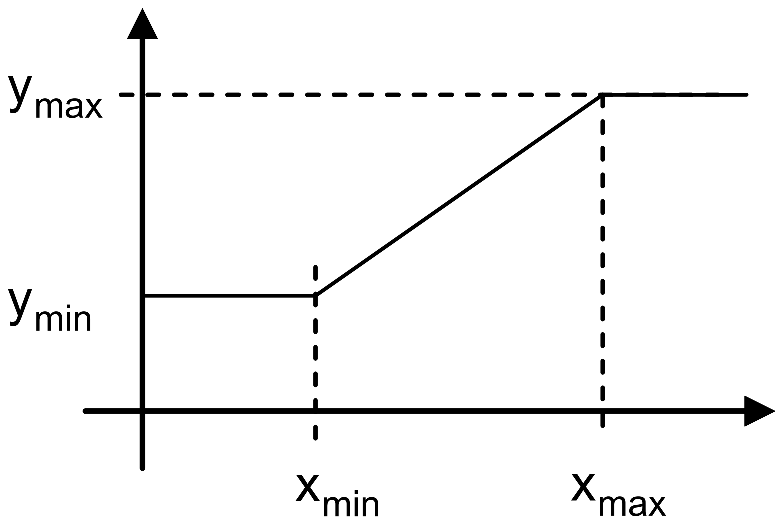


Figure 3.1: *Graphical representation of guideBand rule*

3.2.6 expression

see section 3.2.1 TODO(??): Referentie aanpassen

3.2.7 guideBand

The guide band rule provides a linear interpolation from input *x* to output *y*, if the input value is in the bandwidth between two input threshold. Otherwise, the output is limited to defined minimum and maximum output threshold. The rule reads

$$y^{k+1} = \begin{cases} y_{\min} & , \quad \text{if} \quad x^{k,k+1} \leq x_{\min} \\ y_{\min} + \frac{x^{k,k+1}(y_{\max}-y_{\min})}{x_{\max}-x_{\min}} & , \quad \text{if} \quad x_{\min} < x^{k,k+1} < x_{\max} \\ y_{\max} & , \quad \text{if} \quad x_{\max} \leq x^{k,k+1} \end{cases} \tag{3.4}$$

A graphical representation of the rule is presented in Figure 3.1.

Both input and output thresholds can be a function of time, e.g. depending on the day of the year, as well as a result from the execution of a prior rule.

The rule may be applied to keep the storage *S* of a reservoir within a certain bandwidth and uses the available storage in-between for equalizing the release. If the storage is approaching or down-crossing the lower storage limit S_{min}, the release is set to the minimum flow (zero is no minimum flow is defined). If the storage is approaching or up-crossing the upper limit S_{max}, the release is set to the maximum capacity.

3.2.8 interval

The interval controller is a simple feedback controller according to the control law:

$$y^{k+1} = \begin{cases} y_{above}, & if\, x^k > sp^k + \frac{1}{2}D \\ y_{below}, & if\, x^k < sp^k - \frac{1}{2}D \\ y^k, & \text{otherwise} \end{cases} \tag{3.5}$$

where x^k is a process variable, sp^k is a setpoint, *D* is a dead band around the setpoint, and y^{k+1} is the controller output.

3.2.9 limiter

In contrary to the deadBand rule defined above, the discrete limiter rule restricts the change of a variable to a certain absolute or relative threshold. It reads

$$y^{k+1} = \begin{cases} (1-p)x^k & \text{if } x^{k+1} < (1-p)x^k \\ (1+p)x^k & \text{if } x^{k+1} > (1+p)x^k \\ x^{k+1} & \text{otherwise} \end{cases} \tag{3.6}$$

where *p* is the maximum rate of change, defined as a relative or absolute value.

A typical application of the rule is the limitation of release changes from a reservoir in order to avoid problems downstream.

3.2.10 lookupTable

The rule supplies a 1D lookup table according to

$$y^{k+1} = f(x^{k,k+1}) \tag{3.7}$$

The controller is a simpler version of the dateLookupTable rule defined above. It is also a one on one implementation of the hydraulic controller in SOBEK.

3.2.11 lookup2DTable

The rule supplies a 2D lookup table according to

$$y^{k+1} = f(x_1^{k,k+1}, x_2^{k,k+1}) \tag{3.8}$$

3.2.12 merger

The merger rule provide a simple data hierarchy.

3.2.13 minSimple

The rule reads

$$y^{k+1} = f(t, x^{k,k+1}) \tag{3.9}$$

The formulation is identical to the dateLookupTable. Check the configuration for details and differences.

3.2.14 pid

The Proportional-Integral-Derivative controller (PID controller) is a generic feedback controller including an optional disturbance term commonly used in industrial control systems. It reads

$$\begin{aligned} e(t) &= x(t) - sp(t) \\ y(t) &= K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) + K_f d(t) \end{aligned} \tag{3.10}$$

where $e(t)$ is the difference between a process variable $x(t)$ and a setpoint $sp(t)$, K_p, K_i, K_d are the proportional, integral and derivate gains, respectively, the optional feed forward term consists of a multiplicator K_f and an external disturbance $d(t)$, $y(t)$ is the controller output.

The discrete form of (3.10) in RTC-Tools reads

$$\begin{aligned} e^k &= x^{k-1} - sp^{k-1} \\ E^k &= E^{k-1} + \Delta t\, e^k \\ y^k &= K_p e^k + K_i E^k + K_d \frac{e^k - e^{k-1}}{\Delta t} + K_f d^k \end{aligned} \tag{3.11}$$

where E is the integral of e. The implentation includes a limitation of the maximum velocity according to

$$y^k = \begin{cases} (1 - \Delta t \Delta y_{\max}) y^{k-1} & \text{if } y^k < (1 - \Delta t \Delta y_{\max}) x^{k-1} \\ (1 + \Delta t \Delta y_{\max}) y^{k-1} & \text{if } y^k > (1 + \Delta t \Delta y_{\max}) x^{k-1} \\ y^k & \text{otherwise} \end{cases} \tag{3.12}$$

Furthermore, the integral part is corrected by a wind-up correction according to

$$\begin{aligned} eI^k &\leq \frac{y_{\max} - K_p e^k + K_i E^k - K_d \frac{e^k - e^{k-1}}{\Delta t} - K_f d^k}{K_i} \\ eI^k &\geq \frac{y_{\min} - K_p e^k + K_i E^k - K_d \frac{e^k - e^{k-1}}{\Delta t} - K_f d^k}{K_i}, \end{aligned} \tag{3.13}$$

if the minimum and maximum settings of the actuator are met.

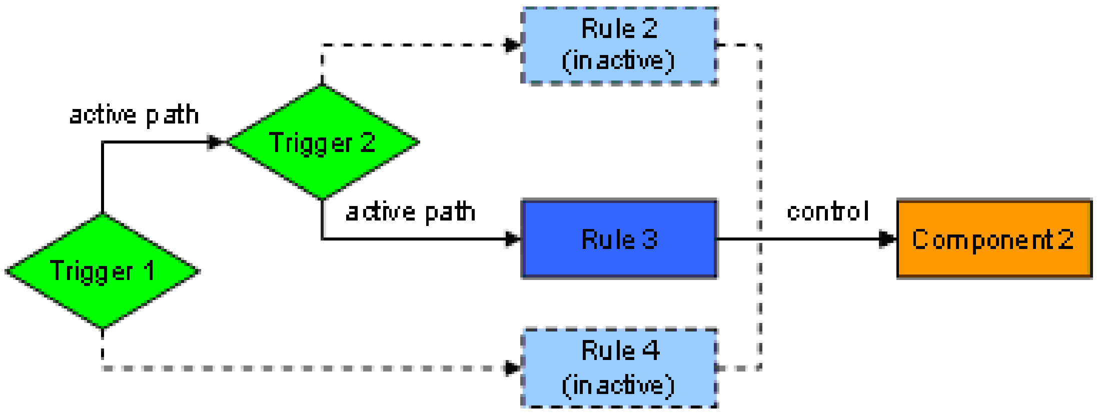


Figure 3.2: *Hierarchical definition of deadBand and standard triggers*

3.2.15 timeAbsolute

This simple rule reads:

$$y^{k+1} = d^{k+1} \tag{3.14}$$

in which *d* is an externally provided time series.

3.2.16 timeRelative

This rule reads:

$$y^{k+1} = x^{\tau} \tag{3.15}$$

where τ is a relative time reference. When the rule is switched on, the relative time is i) put to zero, ii) put to a value based on an existing *y* for which equation (3.15) is fulfilled.

3.3 Triggers

3.3.1 deadBandTrigger

The dead band trigger checks the input data for an upper or lower threshold crossing. The trigger is active, in case of an up-crossing of the upper threshold. It is inactive, in case of a down crossing of the lower threshold. In the range in-between, the trigger keeps its former status. The rule reads

$$y^{k+1} = \begin{cases} 1 & \text{if } x_1^{k,k+1} > x_2^{k,k+1} \\ 0 & \text{if } x_3^{k,k+1} < x_4^{k,k+1} \\ y^k & \text{otherwise} \end{cases} \tag{3.16}$$

The following operators are supported: $>, \geq, =, \neq, \leq, <$.

The dead band trigger as well as the standard trigger (see section 0) may include other triggers which are evaluated in case of an active or inactive trigger state. This feature enables a user to build complex decision trees for selecting unique rules for controlling a structure (Figure 3.2).

Rules which are referenced in a trigger and are associated with an inactive path will be deactivated. This procedure supports that a hydraulic structure is controlled by a unique controller or rule.

3.3.2 deadBandTime

The dead band time trigger checks a time series for a number of subsequent up-crossings *nUp* or down-crossing *nDown* from its current value. If the crossing is observed for at least a user-defined number of time steps, the new value is used. The trigger reads

$$x^{k+1} = \begin{cases} x^{k+1} & \text{if } \{x^{k-nUp+1}, ..., x^{k+1}\} > x^{k-nUp+1} \\ x^{k+1} & \text{if } \{x^{k-nDown+1}, ..., x^{k+1}\} < x^{k-nDown+1} \\ x^{k} & \text{otherwise} \end{cases} \tag{3.17}$$

An application for the trigger is the activation or deactivation of alarm levels. The increase of alarm level may happen immediately ($nUp = 0$), whereas the decrease of an alarm level should be done only after a number of time steps ($nDown = n$) without further threshold crossings.

3.3.3 expression

see section 3.2.2

3.3.4 merger

The merger provides a simple data hierarchy.

3.3.5 polygonLookup

The polygon lookup trigger checks, if a point is inside of a set of polygons. If this is the case, it returns the user-defined value of the specific polygon. The point is defined by the values of two time series, referred to as the x_1 and x_2 coordinate of the point. The rule reads

$$y^{k+1} = \begin{cases} y_1 & \text{if } (x_1^{k,k+1}, x_2^{k,k+1}) \in P_1 \\ \vdots & \vdots \\ y_n & \text{if } (x_1^{k,k+1}, x_2^{k,k+1}) \in P_n \\ y_{default} & \text{otherwise} \end{cases} \tag{3.18}$$

where *y* is the result of the rule and $\{P_1, ..., P_n\}$ is a set of polygons.

Figure 3.3 presents an example for the application of the trigger for the definition of warning level for controlling a lake release in Canton Bern, Switzerland.

3.3.6 set

The trigger enables a logical combination of other triggers, combined by a logical operator. It reads:

$$y^{k+1} = x_1^{k,k+1} \wedge x_2^{k,k+1} \tag{3.19}$$

The following operators are supported: $\wedge$ (AND), $\vee$ (OR), XOR.

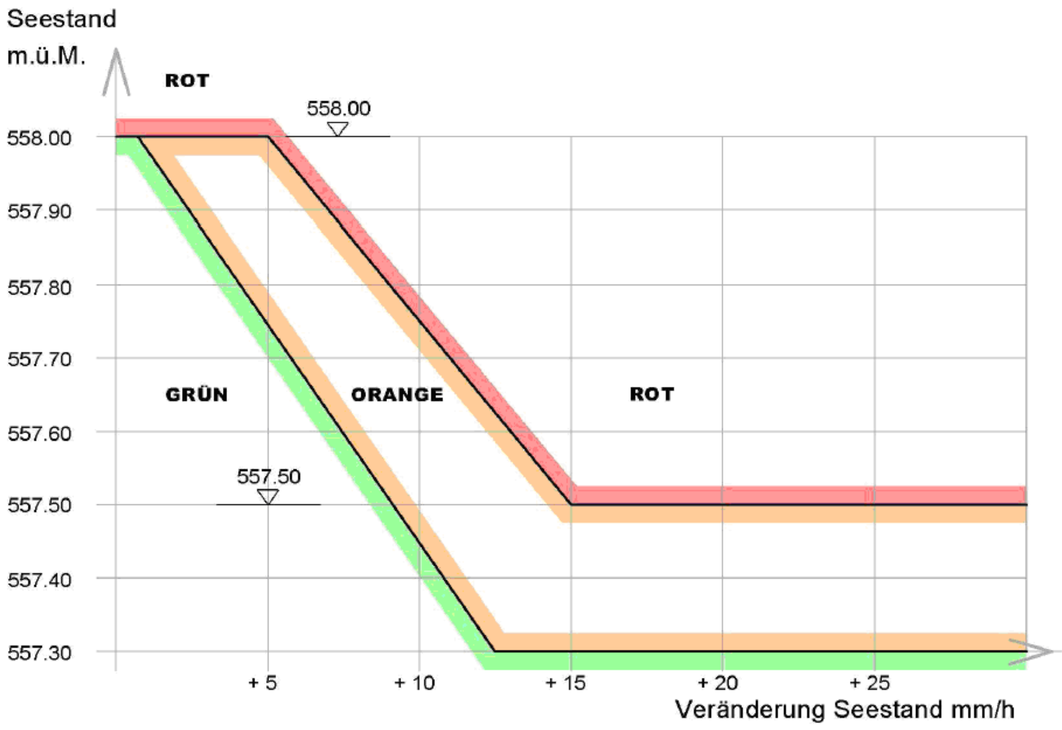


Figure 3.3: *Example for the application of the polygon trigger for the definition of warning levels for controlling a lake release, Lake Thun, CityCanton Bern, country-regionSwitzerland*

If more than two terms have to be combined, the set can be used recursively by defining another set as one of the two terms. Therefore, the expression $y^{k+1} = x_1^k \wedge (x_2^k \vee x_3^k)$ is represented by a hierarchy of two sets (check the example in the configuration chapter).

3.3.7 spreadsheet

The spreadsheet trigger allows the definition of a new trigger state based on its old state and a maximum number of three additional inputs. Besides off (0), on (1) states, all other positive integer values larger than 1 can be processed.

Figure 3.4 shows an application of the trigger for the combination of alarm level for Lake Thun, Canton Bern, Switzerland.

3.3.8 standard

The trigger reads

$$y^{k+1} = \begin{cases} 1 & \text{if } x_1^{k,k+1} > x_2^{k,k+1} \\ 0 & \text{otherwise} \end{cases} \tag{3.20}$$

The following operators are supported: $>, \geq, =, \neq, \leq, <$.

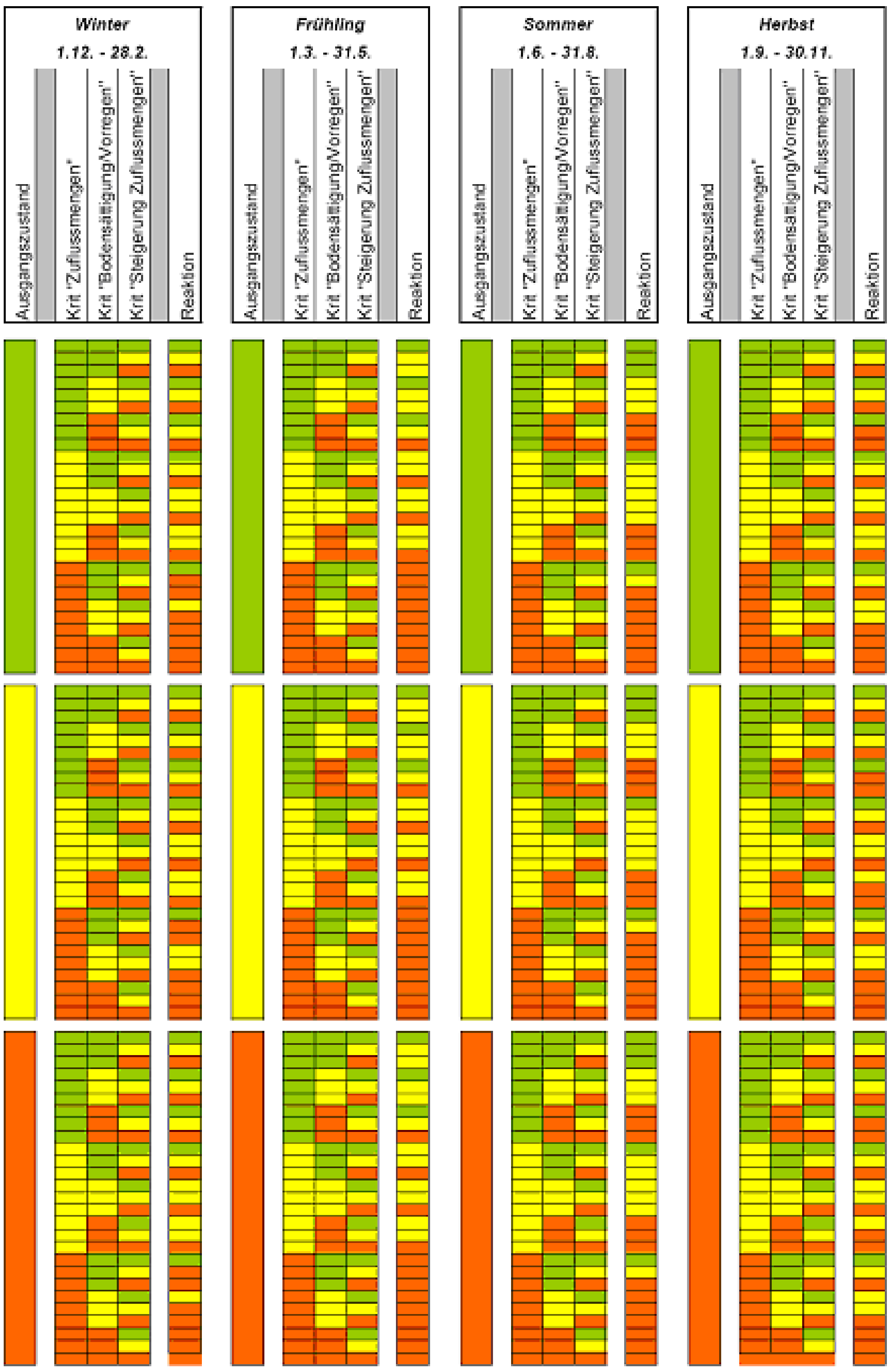


Figure 3.4: *Spreadsheet for combining alarm levels, Lake Thun, Canton Bern, country-regionSwitzerland*

3.4 Implementation of triggers and rules

Most triggers, operating rules and controllers have discrete algebraic formulations in the form of

$$x^{k+1} = g(x^k, u^{k,k+1}, d^{k,k+1}) \tag{3.21}$$

4 Model Predictive Control

4.1 Introduction

We consider a discrete time dynamic system according to

$$\begin{aligned} x^k &= f(x^{k-1}, x^k, u^k, d^k) \\ y^k &= g(x^k, u^k, d^k) \end{aligned} \tag{4.1}$$

where x, y, u, d are respectively the state, dependent variable, control and disturbance vectors, and $f(), g()$ are functions representing an arbitrary linear or nonlinear water resources model. If being applied in Model Predictive Control (MPC), (4.1) is used for predicting future trajectories of the state x and dependent variable y over a finite time horizon represented by $k = 1, ..., N$ time instants, in order to determine the optimal set of controlled variables u by an optimization algorithm. Under the hypothesis of knowing the realization of the disturbance d over the time-horizon, i.e. the trajectory $\{d^k\}_1^N$, a Sequential MPC approach, also referred to as Single Shooting, can be formulated as follows

$$\begin{aligned} &\min_{u \in \{0..T\}} \sum_{k=1}^{N} J(\widetilde{x}^k(u,d), \widetilde{y}^k, u^k) + E(\widetilde{x}^N(u,d), \widetilde{y}^N, u^N) \\ &\text{subject to} \quad h(\widetilde{x}^k(u,d), \widetilde{y}^k, u^k, d^k) \leq 0, \quad k = 1, ..., N \end{aligned} \tag{4.2}$$

where $\widetilde{x}^k(u,d), \widetilde{y}^k$ are the simulation results, $J()$ is a cost function associated with each state transition, $E()$ is an additional cost function associated to the final state condition, and $h()$ are hard constraints. In the corresponding simultaneous or collocated optimization approach, the states x become additional variables of the optimization problem, and the process model (4.1), gets an equality constraint of the optimization problem according to

$$\begin{aligned} &\min_{u,x,y \in \{0..T\}} \sum_{k=1}^{N} J(x^k, y^k, u^k) + E(x^N, y^N, u^N) \\ &\text{subject to} \quad h(x^k, y^k, u^k, d^k) \leq 0, \quad k = 1, ..., N \\ &\qquad\qquad\quad x^k - f(x^{k-1}, x^k, u^k, d^k) = 0 \\ &\qquad\qquad\quad y^k - g(x^k, u^k, d^k) = 0 \end{aligned} \tag{4.3}$$

Both methods lead to identical solutions, but have pros and cons for specific optimization problems in terms of runtime performance. The sequential approach (4.2) is more efficient, if hard constraints are defined on the control variables u only and gets inefficient for hard constraints on states x. Because of the state dependency on all prior control variables u, the constraint Jacobian becomes dense in the lower trianglar matrix. In this case, the simultaneous approach (4.3) shows better efficiency, since states become an input of the optimizer and the constraint Jacobian gets sparse.

RTC-Tools does not strictly distinguish between one and the other approach. In fact, optimization problems can be set up either in a sequential or simultaneous way. Furthermore, both methods can be mixed leading to the so-called multiple-shooting approach. The choice depends on the setup of the specific modeling components. For example, the pool routing in a reservoir can be configured in such a way that the control variable is the release only, and the reservoir level is a result of a simulation. Alternatively, the optimizer may provide both release and reservoir level for computing the mass balance residuum which becomes an

Table 4.1: *Model predictive control features implemented in RTC-Tools*

Optimizer	MPC problem definition
IPOPT (embedded)	constraints
MINOS (Matlab/TOMLAB)	min/max bounds and rate of change limits
SNOPT (Matlab/TOMLAB)	for optimization variables, system states, outputs
	cost function terms
	rate of change with variable exponent
	absolute with variable exponent
	several performance indicators

equality constraint of the optimization problem. It is obvious that the first setup corresponds to the sequential (4.2), the second one to the simultaneous approach (4.3).

4.2 Set-up of the optimization problem

4.2.1 Control variables

The setup of the optimization problem (see Section A.4) starts with the definition of the input variables of the optimizer, i.e. the control u in the sequential setup as well as the optional state x and dependent variable y in the simultaneous setup. Each variable can be of the types

- CONTINUOUS, representing a continuous variable
- INTEGER, representing an integer variable (Note that there is still no RTC-Tools-internal solver supporting Mixed Integer Nonlinear Programming (MINLP), therefore, this options depends on interfacing a suitable solver via the Matlab interface)
- TIMEINSTANT for the setup of a Time-Instant Optimization MPC (TIO-MPC) (experimental version!)

The input variable (Figure A.8) may includes optional bounds for the according to $u_{\min} \leq u \leq u_{\max}$. In case of the TIMEINSTANT of the TIO-MPC, the minimum and maximum bounds are compulsary and represent the two states of the control variable. At a time instant, the state is switches from one to the other.

Each variable can include a scaling factor (Note that this option is now only available for the internal IPOPT optimizer!). It is good practice to scale all variables in such a way that they are in the same order of magnitude. For example, if control variables cover both reservoir levels and releases, the scaling factor of the level should be defined in such as way that itsl range, i.e. the difference between maximum and minimum operating levels of the reservoirs, multiplied by the scaling factor lies in the range of the releases. User-defined scaling usually outperformed automatic, built-in scaling options of the optimizers.

The time step of the control variable u and the simulation can be different. This enables a courser discretization in the optimization compared to the simulation, for example in case of using an explicit model with severe time step restrictions. At this moment, the following aggregation options are supported:

- BLOCK, keeping the the recent value persistent until a new value is specified
- LINEAR, conducting a linear interpolation of the control variable between two instants at

which it is defined by the optimization algorithm

4.2.2 Constraints

RTC-Tools takes into account constraints in two ways: i) as a soft constraint in the objective function or ii) as a hard constraint. This section covers the definition of hard constraint (Figure A.9) either as an inequality constraint or equality constraint of the optimization problem. The next section provides more details on the definition of soft constraints and the objective function in general.

Hard constraints are available for control variables, states or dependent variables. In all cases, the entity may have bounds and a maximum rate of change according to

$$\begin{aligned} &u_{\min} \leq u^k \leq u_{\max} \\ &\Delta u_{\min} \leq u^k - u^{k-n} \leq \Delta u_{\max} \quad n = k-N, ..., k-1 \end{aligned} \tag{4.4}$$

where N is the number of time steps of the rate of change constraint looking back in time.

The definition of constraints on control variables is straightforward and requires no additional information. Constraints on states and dependent variables require information on how these are traced back to control variables. This includes the definition of the control variables, the modeling components involved and the number of time steps looking back in time.

Example 1:

Consider a reservoir represented by the mass balance equation below in simultaneous optimization mode, represented by

$$r^k = s^k - s^{k-1} - \Delta t(Q_{in}^k - Q_{out}^k) \tag{4.5}$$

where r is the residuum of the mass balance, s and Q_{out} are the two optimizer inputs for reservoir storage and release, respectively. For defining an equality constraint on the residuum r, we consider a bound constraints with $r_{\min} = r_{\max} = 0$ and define a state constraint referring to the two variables s and Q_{out}, the modeling component above and $N = 2$ time steps (note that the storage at s^{k-1} and s^k contributes to the mass balance equation).

Example 2:

Let's add a tailwater curve to the reservoir according to

$$tw^k = f(Q_{out}^k) \tag{4.6}$$

and define two constraints for i) keeping a minimum tailwater level and ii) limiting the maximum tailwater rate of change

$$\begin{aligned} &\text{i)} \quad tw^k \leq tw_{\min} \\ &\text{ii)} \quad \Delta tw_{\min} \leq tw^k - tw^{k-1} \leq \Delta tw_{\max} \end{aligned} \tag{4.7}$$

The implementation of the first constraint considers the variable Q_{out}, refers to the modeling components (4.6) and requires $N = 1$. The second constraint works the same except for the need for looking an additional time step back in time leading to $N = 2$.

Keep in mind that Matlab interface provides a platform for defining more general constraints via a user programming.

4.2.3 Cost function terms

RTC-Tools supports the following cost function terms (Figure A.10):

absolute (difference related to a set point)

$$J = w \sum_{k=1}^{T} \left| u^k - u_{sp} \right|^a \tag{4.8}$$

where w is a weighting coefficient, u_{sp} is a constant or time-dependent set point. The absolute term can be applied either on a control, state or dependent variable. The configuration supports the neglection of the upper branch ($u > u_{sp}$) or lower branch ($u < u_{sp}$) of the value range primarily for implementing a soft constraint on a variable up-crossing or down-crossing a threshold.

rate of change (difference of two subsequent values)

$$J = w \sum_{k=1}^{T} \left| u^k - u^{k-1} - u_{sp} \right|^a \tag{4.9}$$

where u is either a control, state or dependent variable, w is a weighting coefficient, and u_{sp} is an optional set point for the rate of change. The latter is again primarily used within soft constraints in combination with the neglection of the upper branch ($u^k - u^{k-1} > u_{sp}$) or lower branch ($u^k - u^{k-1} < u_{sp}$) of the value range.

Furthermore, additional objective function terms of arbitrary type can be defined in Matlab.

4.3 Adjoint models

Gradient-based solvers of the Sequential Quadratic Programming (SQP) or Interior Point (IP) types require a gradient vector of the cost function $\mathrm{d}J(x,u)/\mathrm{d}u$ and the constraint Jacobian $\mathrm{d}h(x,u)/\mathrm{d}u$ for performing efficiently. The computation of these derivatives by numerical differentiation is a straighforward approach, but requires at least $n + nnz + 1$ model execution for an optimization problem of n control variables and nnz non-zero entries in the constraint Jacobian. It becomes computationally inefficient for several hundreds or thousands of dimensions, and disqualifies the approach from being applied in an operational setting.

A significantly more efficient method for computing the derivatives at computational costs in the order of a single model execution is the set-up of an adjoint model for each modeling component. The RTC-Tools framework takes care for integrating these models both in the simulation mode as well as the reverse adjoint mode.

The set-up of an adjoint model is outlined for the explicit version of the diffusive wave model presented in section 3.2.1. Consider the mass balance equation given by

$$s^k = s^{k-1} + \Delta t \sum_i f(s^{k-1}, s_i^{k-1}, dg_i^k) \tag{4.10}$$

where s represent the storage at the node of interest and s_i is the storage at a connected node i and the function $f()$ denotes the flow contribution of a flow branch or hydraulic structure.

A straightforward way for the derivation of an adjoint model of an explicit calculation workflow is the application of algorithmic differentiation in reverse mode. It is basically a consequent application of the chain rule leading to

$$
\begin{aligned}
\widehat{s}^{k-1} &= \widehat{s}^{k}\left[1+\Delta t\sum_{i}\frac{\partial f(s^{k-1},s_i^{k-1},dg_i^{k})}{\partial s^{k-1}}\right]\\
\widehat{s}_i^{k-1} &= \widehat{s}^{k}\left[1+\Delta t\frac{\partial f(s^{k-1},s_i^{k-1},dg_i^{k})}{\partial s_i^{k-1}}\right]\\
\widehat{dg}^{k} &= \widehat{s}^{k}\left[\Delta t\frac{\partial f(s^{k-1},s_i^{k-1},dg_i^{k})}{\partial dg_i^{k}}\right]
\end{aligned}
\tag{4.11}
$$

where $\widehat{u}$ is the adjoint variable of u.

We compute the cost function gradient according to the following procedure:

1 Model simulation, (4.10), for computing all states and dependent variables
2 Initialization of the adjoint variables by the partial derivatives of the objective function, $\widehat{u}^k = \partial J(u^k, x^k, y^k)/\partial u^k$, with respect to control variables, states and dependent variables.
3 Model execution in adjoint mode, (4.11), in reverse order (with respect to the time loop and the execution of subsequent modeling components)

After conducting the steps above, the resulting adjoint variables represent the total derivatives of the cost function $\mathrm{d}J(u^k, x^k, y^k)/\mathrm{d}u^k$, i.e. the cost function gradient.

The computation of the constraint Jacobian is similar. The following procedure holds for a single constraint at a specific time step k:

1 ditto (once for all constraints)
2 Initialization of the adjoint variables by the partial derivatives of the constraint, $\widehat{u}^k = \partial h(u^k, x^k, y^k, d^k)/\partial u^k$, with respect to control variables, states and dependent variables.
3 Model execution in adjoint mode, (4.11), over N time steps which contribute to the non-zero Jacobian entries of the specific constraint.

Adjoint systems are still not implemented for all available components in RTC Tool. An overview about the status of implementation is given in Table 4.2

Table 4.2: *Implementation status of adjoint models*

	Adjoint available	**Comments**
Modeling components		
accumulation	yes	-
arma	yes	-
expression	yes	-
gradient	yes	-
hydraulicModel	yes	implementation is only available for the explicit scheme, implicit scheme will become available soon
hydrologicalModel	no	-
merger	yes	-
neuralNetwork	yes	-
reservoir	yes	-
reservoirBPA	yes	-
unitDelay	yes	-
unitHydrograph	yes	-
Rules		
all	no	the adjoint of smooth rules such as the PID controller may become implemeneted in the future for modeling mixed systems with MPC and feedback control
Triggers		
all	no	triggers switching on external input will be supported in the future

5 Interfaces

5.1 Matlab

The RTC-Tools integration in Matlab requires the same file structure than already presented in the section about the Delft-FEWS interface. Furthermore, we assume that the boundary conditions and the states are provided in XML format according to the definition above.

Registration of *.jar files of RTC-Tools, startup of TOMLAB optimizers in Matlab

```
addpath c:\ tomlab;
startup;
% register jars of rtcModule
javaaddpath(\{'..\..\lib\castor\castor-0.9.5.jar',...
                '..\..\lib\Delft\_RTC\_castor\Delft\_RTC\_castor.jar',...
                '..\..\lib\jdom\jdom.jar',...
                '..\..\lib\junit\junit-4.1.jar',...
                '..\..\lib\rtcModule.jar'\});
Model Predictive Control set-up in Matlab
% prepare arguments for initialisation of module
args = javaArray('java.lang.String',4);
args(1) = javaObject('java.lang.String',pwd);
args(2) = javaObject('java.lang.String','rtcModuleConfig.xml');
args(3) = javaObject('java.lang.String','rtcDataConfig.xml');
args(4) = javaObject('java.lang.String','rtcObjectiveConfig.xml');

% pre-processing and initialisation of the rtcModule
global rtcModule;
rtcModule = nl.deltares.fews.rtc.RtcModule();
rtcModule.init(args);

% initial guess for controlled variable
input = zeros(100,1);
% problem definition
Prob = conAssign(
  'jRes', 'jResDer', [], [], 0*ones(100,1), 20*ones(100,1),
  'Name', input, [], [], [], [], []);
Prob.SOL.optPar(10) = 0.00001;

% derivative check
[exitFlag,output] = checkDerivs(Prob, input, 1, 1, [], []);

% optimization
tic
Result = tomRun('minos', Prob, 1);
time=toc
tic
Result = tomRun('snopt', Prob, 1);
time=toc

% save
rtcModule.save(args);
function [j] = jRes1(input)

% function [j] = jRes1(input)
```

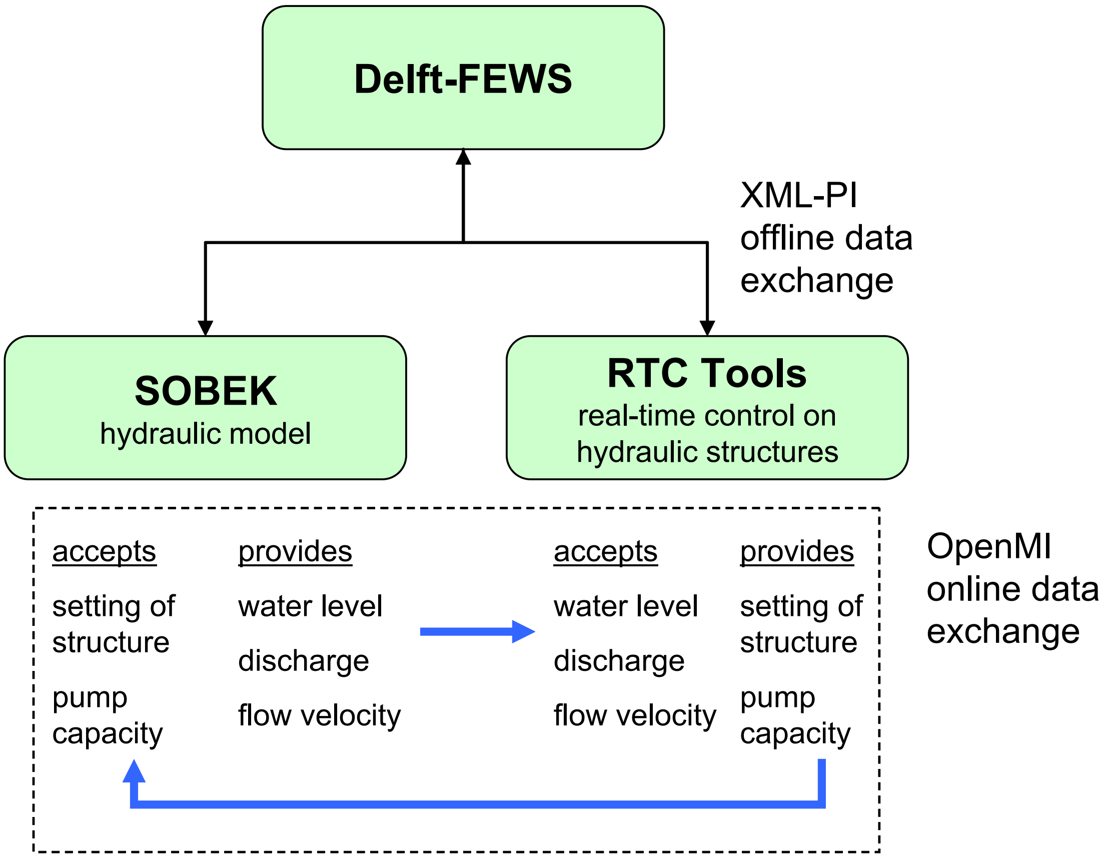


Figure 5.1: *Conjunctive use of a SOBEK and an RTC-Tools model under Delft-FEWS*

```
global rtcModule;
j = rtcModule.simulate(input);
function [jDer] = jDerRes1(input)

% function [jDer] = jDerRes1(input)
global rtcModule;
jDer = rtcModule.simulateGradient(input);
```

5.2 FEWS-PI

With the PI-XML interface RTC-Tools can be used in an Delft-FEWS environment. Under Delft-FEWS models are coupled offline, this means they do not exchange data during runtime. The models are run sequentially.

Figure 5.1 shows an example for a usage of RTC-Tools in a Delft-FEWS environment and compares the way of data exchange with the online coupling method. For online coupling RTC-Tools provides an interface according to the OpenMI standard.

5.3 OpenMI

OpenMI is a standard for coupling of water-related computer models (OpenMI Association, 2007; Moore and Tindall, 2005; Gregersen *et al.*, 2007). Models coupled via OpenMI can exchange data during runtime. RTC-Tools OpenMI compliant according to OpenMI version 1.4. This feature enables its online linkage to hydraulic modeling packages such as SOBEK

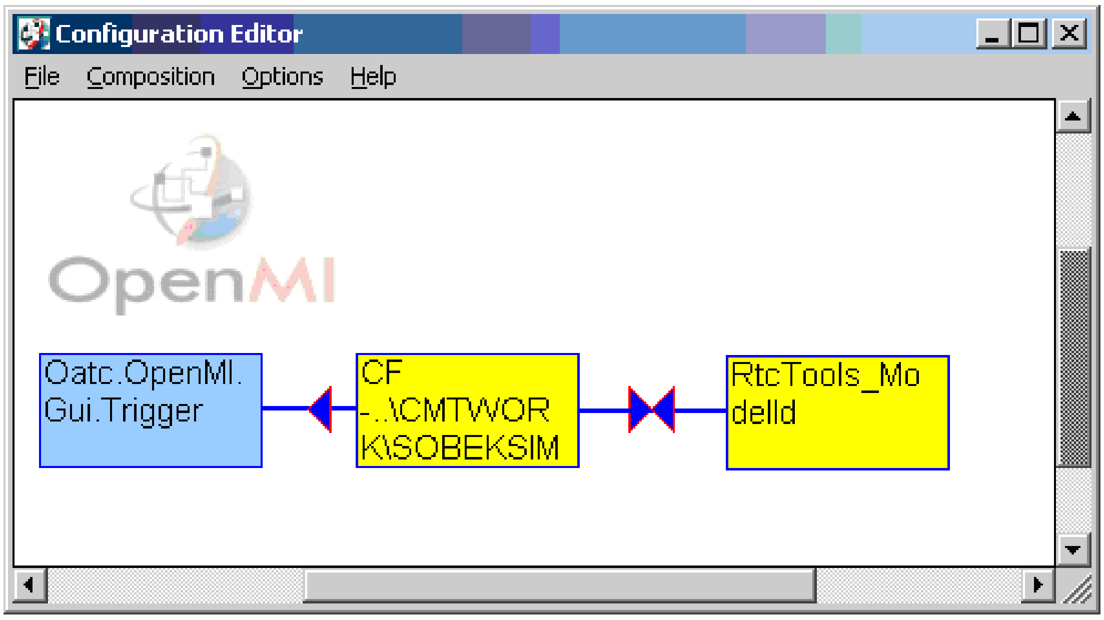
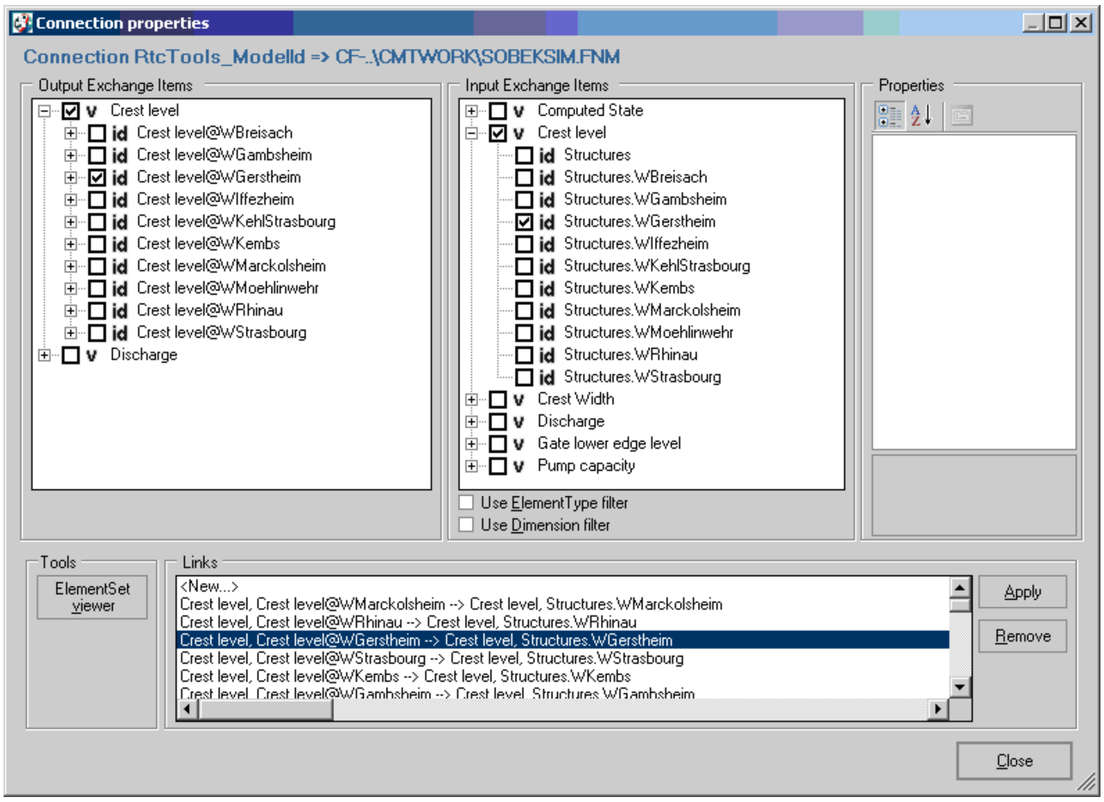


Figure 5.2: *OpenMI-composition of Trigger, SOBEK and RTC-Tools in OpenMI-Configuration-Editor (Schwanenberg et al., 2011)*

on a time step level, where the coupled models influence each other.

Figure 5.2 and Figure 5.3 show an example for an OpenMI composition with SOBEK and RTC-Tools in the OpenMI configuration editor as well as the configuration of a connection between the two models.



Connection properties

Connection RtcTools_ModelId => CF-..\CMTWORK\SOBEKSIM.FNM

Output Exchange Items

- ☑ v Crest level
  - ☐ id Crest level@WBreisach
  - ☐ id Crest level@WGambsheim
  - ☑ id Crest level@WGerstheim
  - ☐ id Crest level@WIffezheim
  - ☐ id Crest level@WKehlStrasbourg
  - ☐ id Crest level@WKembs
  - ☐ id Crest level@WMarckolsheim
  - ☐ id Crest level@WMoehlinwehr
  - ☐ id Crest level@WRhinau
  - ☐ id Crest level@WStrasbourg
- ☐ v Discharge

Input Exchange Items

- ☐ v Computed State
- ☑ v Crest level
  - ☐ id Structures
  - ☐ id Structures.WBreisach
  - ☐ id Structures.WGambsheim
  - ☑ id Structures.WGerstheim
  - ☐ id Structures.WIffezheim
  - ☐ id Structures.WKehlStrasbourg
  - ☐ id Structures.WKembs
  - ☐ id Structures.WMarckolsheim
  - ☐ id Structures.WMoehlinwehr
  - ☐ id Structures.WRhinau
  - ☐ id Structures.WStrasbourg
- ☐ v Crest Width
- ☐ v Discharge
- ☐ v Gate lower edge level
- ☐ v Pump capacity

☐ Use ElementType filter

☐ Use Dimension filter

Properties

Tools

ElementSet viewer

Links

```
<New...>
Crest level, Crest level@WMarckolsheim --> Crest level, Structures.WMarckolsheim
Crest level, Crest level@WRhinau --> Crest level, Structures.WRhinau
Crest level, Crest level@WGerstheim --> Crest level, Structures.WGerstheim
Crest level, Crest level@WStrasbourg --> Crest level, Structures.WStrasbourg
Crest level, Crest level@WKembs --> Crest level, Structures.WKembs
Crest level, Crest level@WGambsheim --> Crest level, Structures.WGambsheim
```

Figure 5.3: *Configuration window of the OpenMI connection from RTC-Tools to SOBEK in the OpenMI configuration editor connection properties window (Schwanenberg et al., 2011)*

6 References

Gregersen, J., P. Gijsbers and S. Westen, 2007. “OpenMI: Open modelling Interface.” *Journal of Hydroinformatics* 9 (3): 175–191. 34

Moore, R. V. and C. I. Tindall, 2005. “An overview of the open modelling interface and environment (the OpenMI).” *Environmental Science & Policy* 8 (3): 279–286. 34

OpenMI Association, 2007. “OpenMI A new era in integrated water management.” URL `http://www.openmi.org`. 34

Schwanenberg, D., B. Becker and T. Schruff, 2011. *SOBEK-Grobmodell des staugeregelten Oberrheins / Entwicklung*. report no. 1201242-000, Deltares. 35, 36

A Configuration / Schemas

A.1 General runtime definitions

The RTC-Tools schematization is specified in a set of XML files (Table A.1).

Table A.1: *RTC-Tools configuration files*

File	Content	Comments
rtcDataConfig.xml	time series definitions, interface definitions for file io, in-memory data exchange etc.	required
rtcObjectiveConfig.xml	definition of the optimization problem including the control variables, constraints and cost function terms	optional
rtcParameterConfig.xml	set of externalized parameters for modification in external applications such as Delft-FEWS or Matlab	optional
rtcRuntimeConfig.xml	definition of runtime relevant info: file names if diviating from standard naming convention, run mode (simulation, optimization etc.), logging information etc.	required
rtcScenarioTreeConfig.xml	definition of a scenario for the Tree-Based MPC option	optional
rtcToolsConfig.xml	RTC-Tools schematization including the modeling components, rules and controllers as well as triggers of the model	required

All configuration files are expected in the same working directory. We highly recommend to validate all XML files against the corresponding XSD schema definitions during the setup. We suggest using validating XML editors such as XMLSpy (`http://www.altova.com/xml-editor/`) or oXygen (`http://www.oxygenxml.com/`).

Figure A.1 provides an overview about the runtime configuration schema. The following figures provide further details on the configuration of input files (Figure A.2), the simulation period of the model (Figure A.3), the execution model of the model (Figure A.4), and setting of the IPOPT optimizer (Figure A.5 and Figure A.6).

A.2 Modeling components

TODO

A.3 Feedback control (rules and triggers)

TODO

A.4 Setup of optimization problems

The optimization problem is configured in the file `rtcObjectiveConfig.xml` and Figure A.7 provides an overview. Find further details in the following figures on the definition of optimization variables (Figure A.8), the definition of hard constraints (Figure A.9) and cost function terms (Figure A.10 and Figure A.11).

A.5 Setup of scenrio trees

TODO

A.6 Time series input and output

The exchange of time series from Delft-FEWS to RTC-Tools and back is managed in an xml file according to the scheme `rtcDataConfig.xsd` (see Figure A.12).

The file starts with a section with general settings: the directory with input data for RTC-Tools, the directory of output data and the location of the log file with runtime information from RTC-Tools. Keep in mind that the root directory is provided in the function call to RTC-Tools. Thus, directories and file should be provided relative to the root directory.

The general section is followed by the definition of input time series which will be read into the model during initialization. Furthermore, export series are time series which are generated by RTC-Tools during execution. Import and export time series comply with the PI-XML format of Delft-FEWS. Both version of the format are supported: i) pure XML, ii) a combination of XML and binary files (more efficient, but also less readable).

A.7 State handling

An xml state file including the system states (fixed name `rtcStateConfig.xml`) and an additional file with meta information about the states (fixed name `statesPI.xml`) have to be provided in the import directory.

The xml state file is used for updating the model states at the first time step of the simulation. It includes pairs of ID (according to the definition of the RTC-Tools time series IDs in the section above) and value. Note that this includes all imported and exported times series defined in the section above. At the end end of the simulation, the states at the last time step are written into the file “rtcStateConfig.xml” in the export directory.

The file including the meta data about the states is in Delft-FEWS XML-PI state format. The content of this file is not further processed. However, the file is copied into the export directory with an updated time stamp of the warm state.

A.8 Command line options of RTC-Tools

Specify the location of xsd schemas with the argument `-schemalocation`. An example for an RTC-Tools batch file:

```
RTCTools.exe -schemaLocation=d:\NHI_RTC-Tools\RTCTools\xsd\
```

If the argument `-schemaLocation` is not specified RTC-Tools searches for the xml schema definition files (xsd files) in the directory where the executable is located.

A.9 RTC-Tools in Delft-FEWS

We recommend the following set-up of the file system for implementing RTC-Tools in Delft-FEWS:

```
<Delft-FEWS root folder>
    <Modules>
        <RTCTools>
            <bin> (including the executables)
            <export> (output of RTC-Tools)
            <import> (input for RTC-Tools)
            diag.xml (PI-XML file with diagnostics)
            rtcDataConfig.xml (configuration of time series)
            rtcModuleConfig.xml (configuration of modelling components)
            rtcObjectiveConfig.xml (configuration of optimization problem)
```

A configuration example is given in Section C.4.

A.10 RTC-Tools in OpenMI

Below an example of an `*.omi`-file is given. `Assembly` refers to the location of the dynamic link library with the RTC-Tools computational core. This DLL must provide the OpenMI interface definition. `LinkableComponent` refers to “Deltares.RtcToolsWrapper.RtcToolsLinkableComponent”, this is hard-coded in the OpenMI interface definition. Table **??** explains the meaning of the argument keys.

```
<?xml version="1.0"?>
<LinkableComponent Type="Deltares.RtcToolsWrapper.RtcToolsLinkableComponent"
  Assembly="..\..\RTCToolsOpenMI\bin\Deltares.RtcToolsWrapper.dll"
  xmlns="http://www.openmi.org/LinkableComponent.xsd">
        <Arguments>
      <Argument Key="modelDirectory" ReadOnly="true" Value=".\RtcTools" />
      <Argument Key="logLevel" ReadOnly="true" Value="1" />
      <Argument Key="flush" ReadOnly="true" Value="true" />
      <Argument Key="MissingValue" ReadOnly="true" Value="0" />
      <Argument Key="OpenMiTimeStepSkip" ReadOnly="true" Value="144" />
      <Argument Key="SchemaLocation" ReadOnly="true"
       Value="..\..\..\RTCTools\xsd\" />
        </Arguments>
</LinkableComponent>
```

A.11 RTC-Tools in Matlab

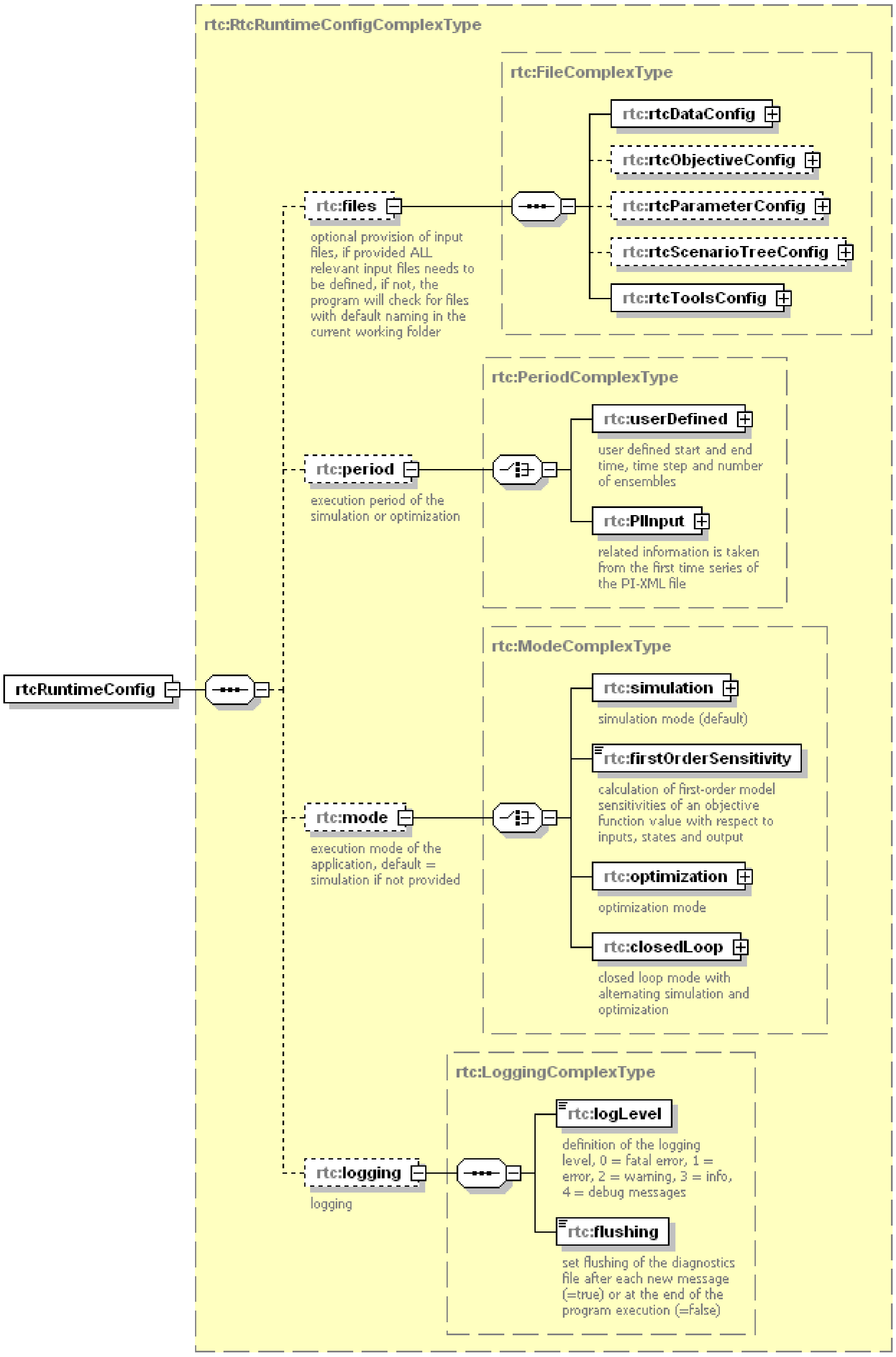

rtc:RtcRuntimeConfigComplexType
rtc:FileComplexType
rtc:rtcDataConfig
rtc:rtcObjectiveConfig
rtc:rtcParameterConfig
rtc:rtcScenarioTreeConfig
rtc:rtcToolsConfig
rtc:files
optional provision of input files, if provided ALL relevant input files needs to be defined, if not, the program will check for files with default naming in the current working folder
rtc:PeriodComplexType
rtc:userDefined
user defined start and end time, time step and number of ensembles
rtc:PIInput
related information is taken from the first time series of the PI-XML file
rtc:period
execution period of the simulation or optimization
rtc:ModeComplexType
rtcRuntimeConfig
rtc:simulation
simulation mode (default)
rtc:firstOrderSensitivity
calculation of first-order model sensitivities of an objective function value with respect to inputs, states and output
rtc:mode
execution mode of the application, default = simulation if not provided
rtc:optimization
optimization mode
rtc:closedLoop
closed loop mode with alternating simulation and optimization
rtc:LoggingComplexType
rtc:logLevel
definition of the logging level, 0 = fatal error, 1 = error, 2 = warning, 3 = info, 4 = debug messages
rtc:logging
logging
rtc:flushing
set flushing of the diagnostics file after each new message (=true) or at the end of the program execution (=false)

Figure A.1: *Overview about the RTC-Tools runtime configuration*

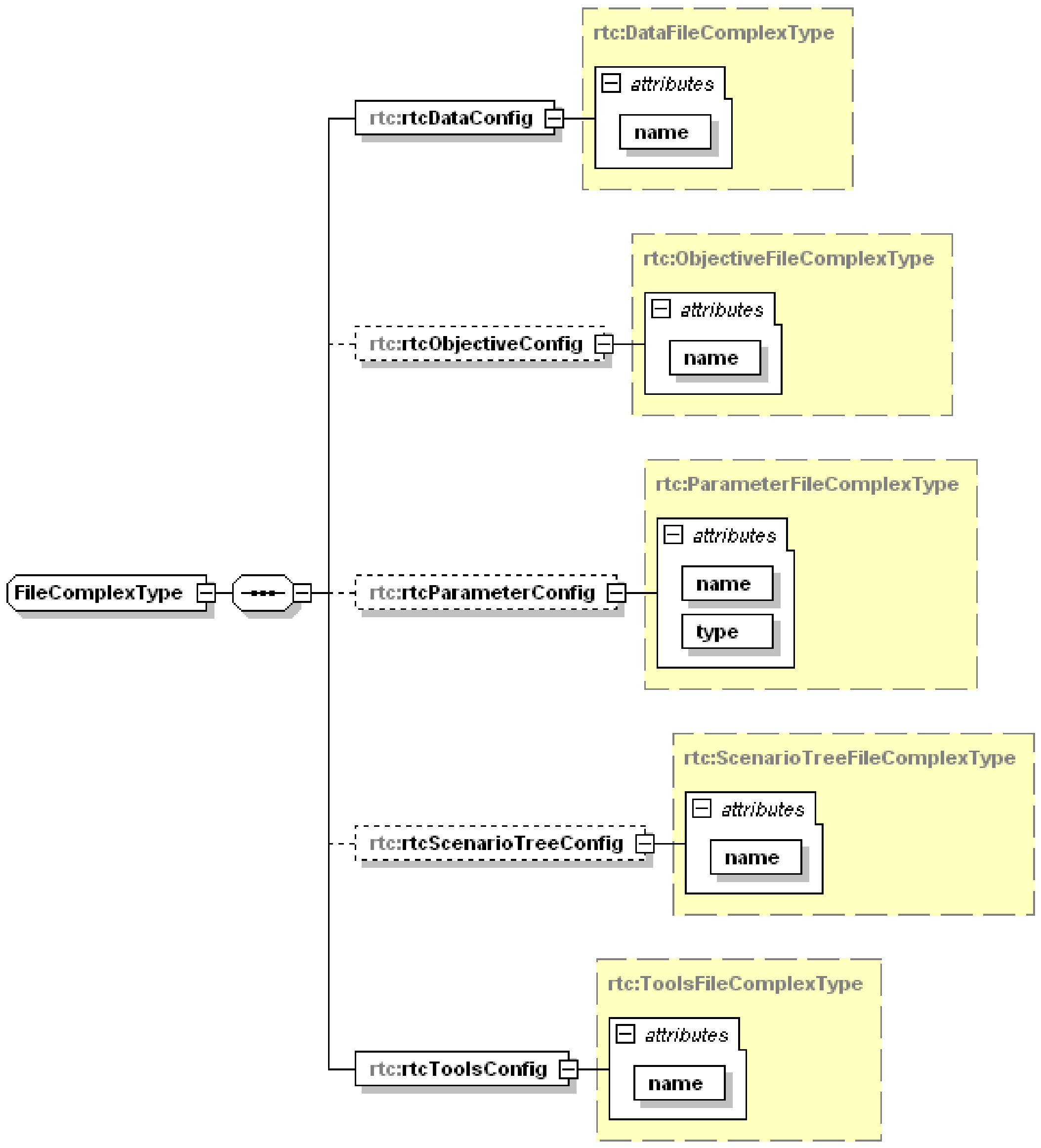


Figure A.2: *Configuration of input files*

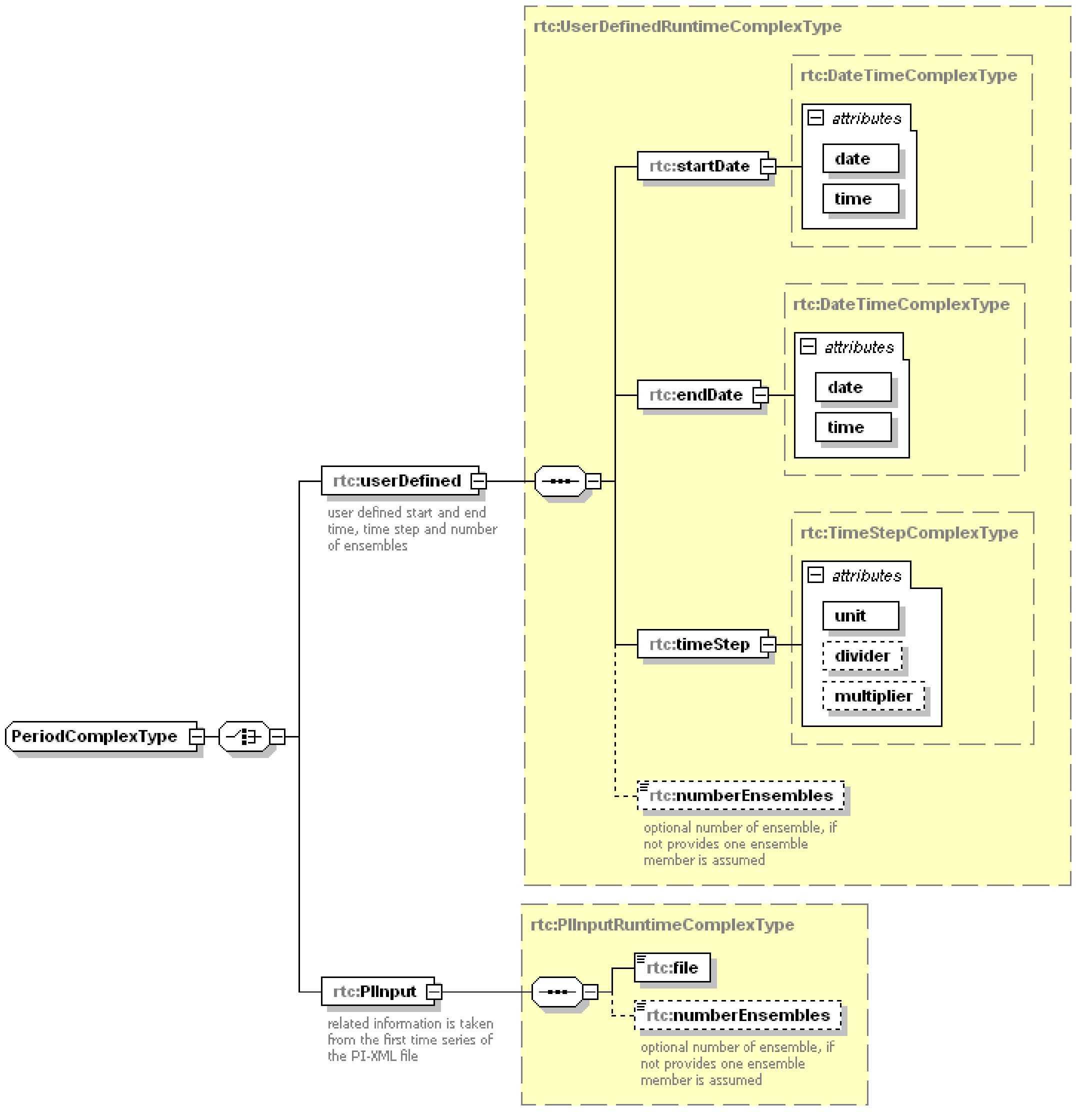

rtc:UserDefinedRuntimeComplexType
rtc:DateTimeComplexType
attributes
date
time
rtc:startDate
rtc:DateTimeComplexType
attributes
date
time
rtc:endDate
rtc:userDefined
user defined start and end time, time step and number of ensembles
rtc:TimeStepComplexType
attributes
unit
divider
multiplier
rtc:timeStep
PeriodComplexType
rtc:numberEnsembles
optional number of ensemble, if not provides one ensemble member is assumed
rtc:PIInputRuntimeComplexType
rtc:file
rtc:PIInput
rtc:numberEnsembles
related information is taken from the first time series of the PI-XML file
optional number of ensemble, if not provides one ensemble member is assumed

Figure A.3: *Definition of the simulation / optimization period*

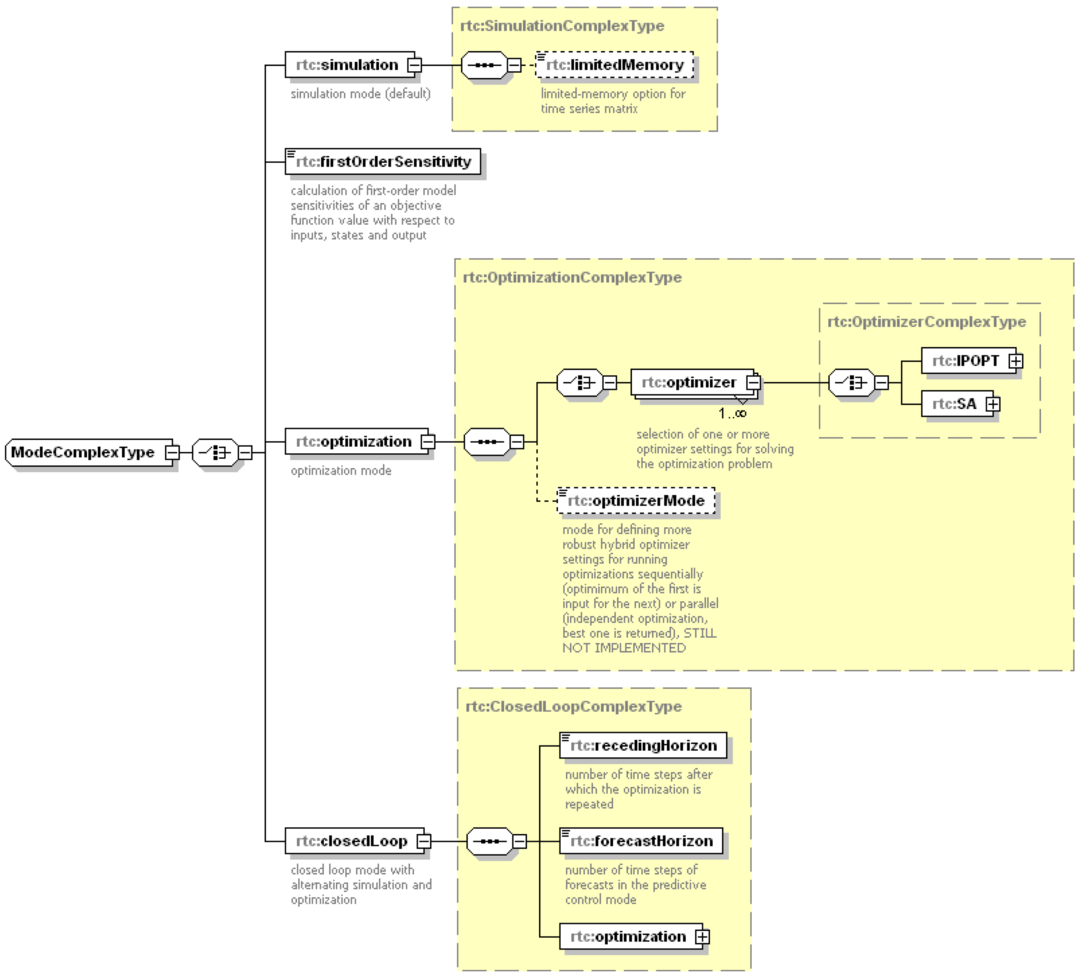
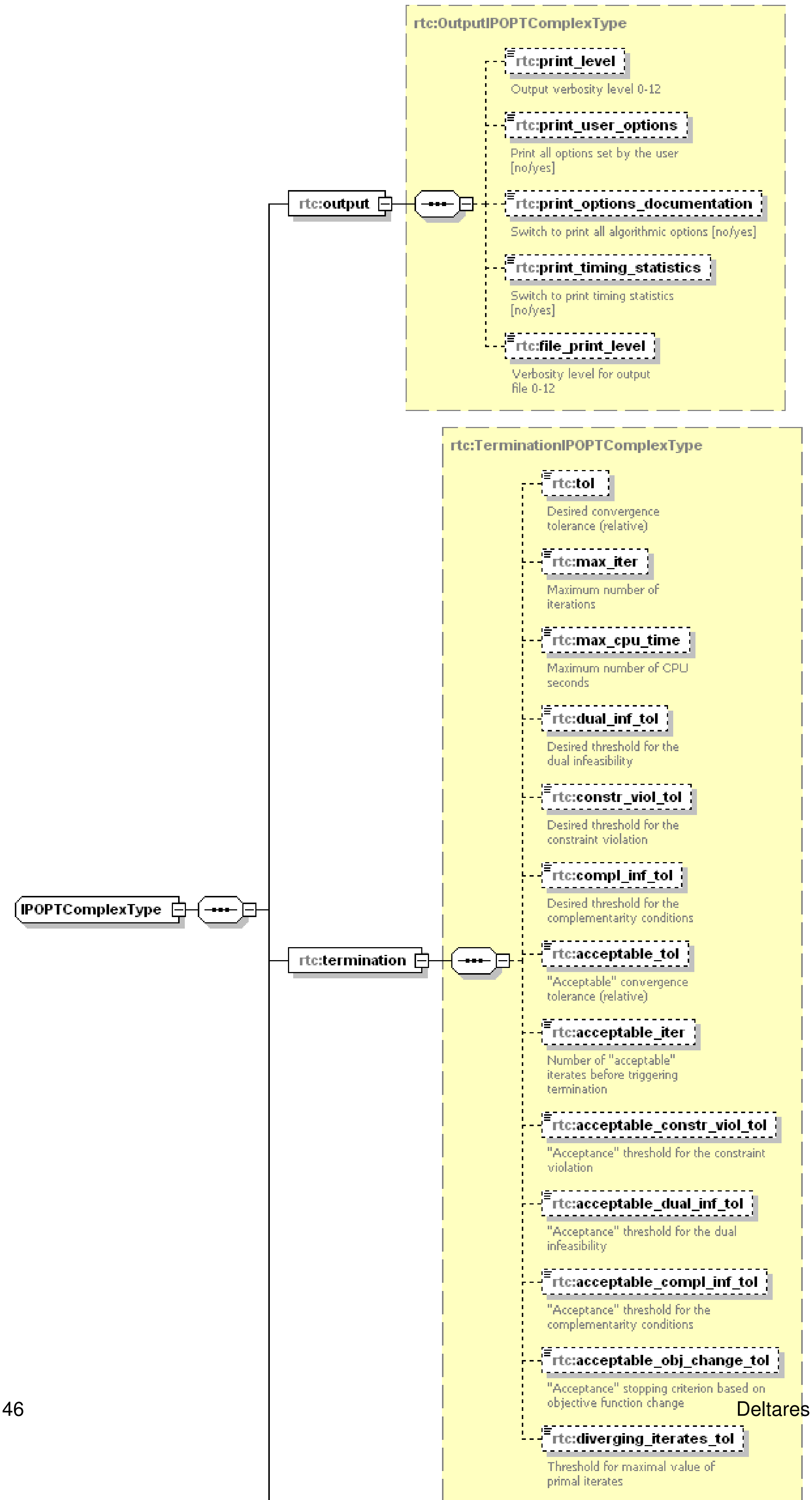

rtc:SimulationComplexType
rtc:simulation
simulation mode (default)
rtc:limitedMemory
limited-memory option for time series matrix
rtc:firstOrderSensitivity
calculation of first-order model sensitivities of an objective function value with respect to inputs, states and output
rtc:OptimizationComplexType
rtc:OptimizerComplexType
rtc:IPOPT
rtc:SA
rtc:optimizer
1..∞
selection of one or more optimizer settings for solving the optimization problem
ModeComplexType
rtc:optimization
optimization mode
rtc:optimizerMode
mode for defining more robust hybrid optimizer settings for running optimizations sequentially (optimimum of the first is input for the next) or parallel (independent optimization, best one is returned), STILL NOT IMPLEMENTED
rtc:ClosedLoopComplexType
rtc:recedingHorizon
number of time steps after which the optimization is repeated
rtc:closedLoop
closed loop mode with alternating simulation and optimization
rtc:forecastHorizon
number of time steps of forecasts in the predictive control mode
rtc:optimization

Figure A.4: *Configuration of execution modes in RTC-Tools*


rtc:OutputIPOPTComplexType
rtc:print_level
Output verbosity level 0-12
rtc:print_user_options
Print all options set by the user [no/yes]
rtc:output
rtc:print_options_documentation
Switch to print all algorithmic options [no/yes]
rtc:print_timing_statistics
Switch to print timing statistics [no/yes]
rtc:file_print_level
Verbosity level for output file 0-12
rtc:TerminationIPOPTComplexType
rtc:tol
Desired convergence tolerance (relative)
rtc:max_iter
Maximum number of iterations
rtc:max_cpu_time
Maximum number of CPU seconds
rtc:dual_inf_tol
Desired threshold for the dual infeasibility
rtc:constr_viol_tol
Desired threshold for the constraint violation
rtc:compl_inf_tol
Desired threshold for the complementarity conditions
IPOPTComplexType
rtc:termination
rtc:acceptable_tol
"Acceptable" convergence tolerance (relative)
rtc:acceptable_iter
Number of "acceptable" iterates before triggering termination
rtc:acceptable_constr_viol_tol
"Acceptance" threshold for the constraint violation
rtc:acceptable_dual_inf_tol
"Acceptance" threshold for the dual infeasibility
rtc:acceptable_compl_inf_tol
"Acceptance" threshold for the complementarity conditions
rtc:acceptable_obj_change_tol
"Acceptance" stopping criterion based on objective function change
rtc:diverging_iterates_tol
Threshold for maximal value of primal iterates

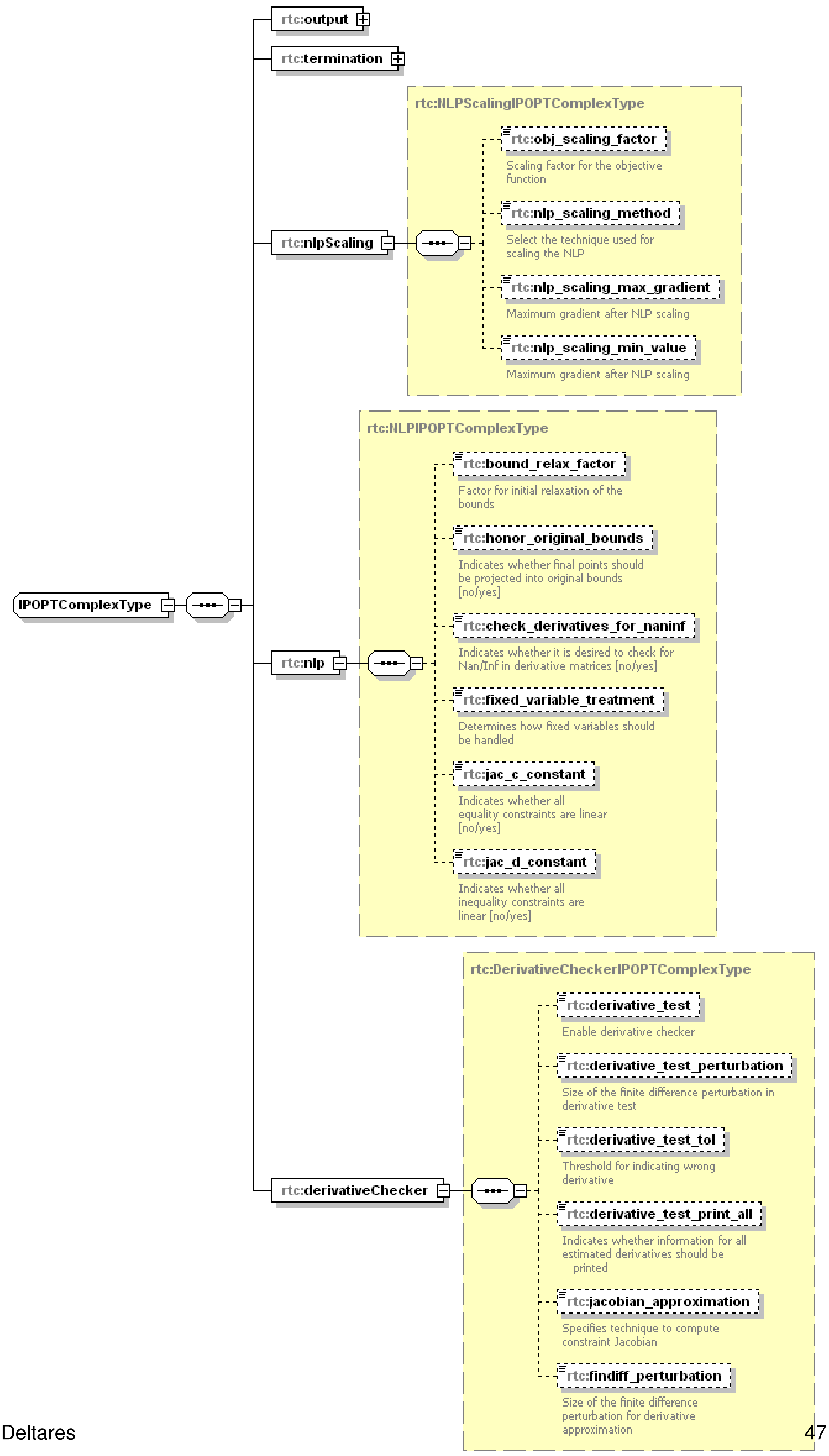

rtc:output
rtc:termination
rtc:NLPScalingIPOPTComplexType
rtc:obj_scaling_factor
Scaling factor for the objective function
rtc:nlp_scaling_method
Select the technique used for scaling the NLP
rtc:nlp_scaling_max_gradient
Maximum gradient after NLP scaling
rtc:nlp_scaling_min_value
Maximum gradient after NLP scaling
rtc:nlpScaling
rtc:NLPIPOPTComplexType
rtc:bound_relax_factor
Factor for initial relaxation of the bounds
rtc:honor_original_bounds
Indicates whether final points should be projected into original bounds [no/yes]
IPOPTComplexType
rtc:check_derivatives_for_naninf
Indicates whether it is desired to check for Nan/Inf in derivative matrices [no/yes]
rtc:nlp
rtc:fixed_variable_treatment
Determines how fixed variables should be handled
rtc:jac_c_constant
Indicates whether all equality constraints are linear [no/yes]
rtc:jac_d_constant
Indicates whether all inequality constraints are linear [no/yes]
rtc:DerivativeCheckerIPOPTComplexType
rtc:derivative_test
Enable derivative checker
rtc:derivative_test_perturbation
Size of the finite difference perturbation in derivative test
rtc:derivative_test_tol
Threshold for indicating wrong derivative
rtc:derivativeChecker
rtc:derivative_test_print_all
Indicates whether information for all estimated derivatives should be printed
rtc:jacobian_approximation
Specifies technique to compute constraint Jacobian
rtc:findiff_perturbation
Size of the finite difference perturbation for derivative approximation

Figure A.6: *IPOPT optimizer options, part 1*

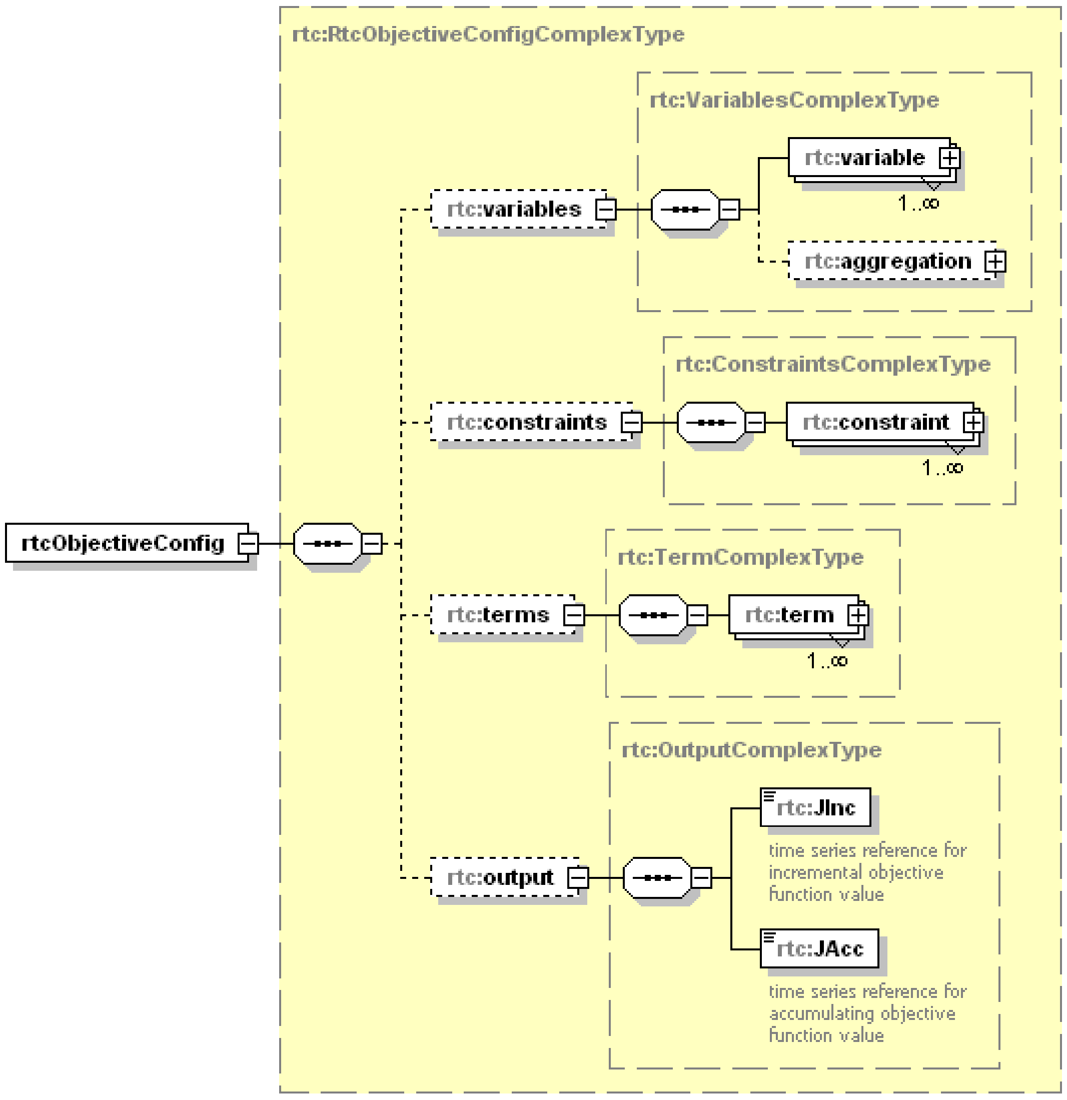


Figure A.7: *Overview about the configuration of optimization problems*

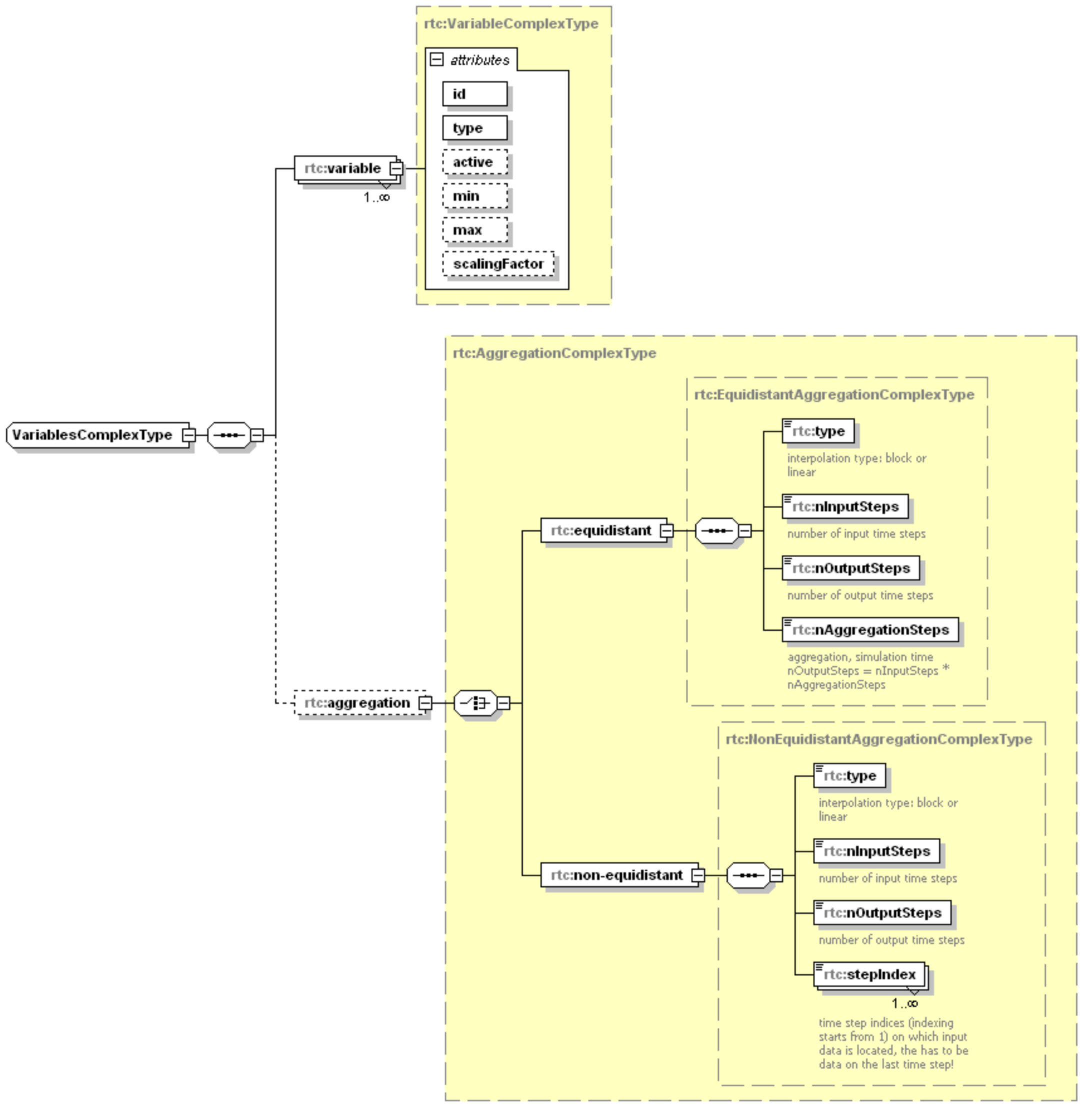

rtc:VariableComplexType
attributes
id
type
active
min
max
scalingFactor
rtc:variable
1..∞
VariablesComplexType
rtc:aggregation
rtc:AggregationComplexType
rtc:equidistant
rtc:non-equidistant
rtc:EquidistantAggregationComplexType
rtc:type
interpolation type: block or linear
rtc:nInputSteps
number of input time steps
rtc:nOutputSteps
number of output time steps
rtc:nAggregationSteps
aggregation, simulation time nOutputSteps = nInputSteps * nAggregationSteps
rtc:NonEquidistantAggregationComplexType
rtc:type
interpolation type: block or linear
rtc:nInputSteps
number of input time steps
rtc:nOutputSteps
number of output time steps
rtc:stepIndex
1..∞
time step indices (indexing starts from 1) on which input data is located, the has to be data on the last time step!

Figure A.8: *Overview about the configuration of optimization problems*

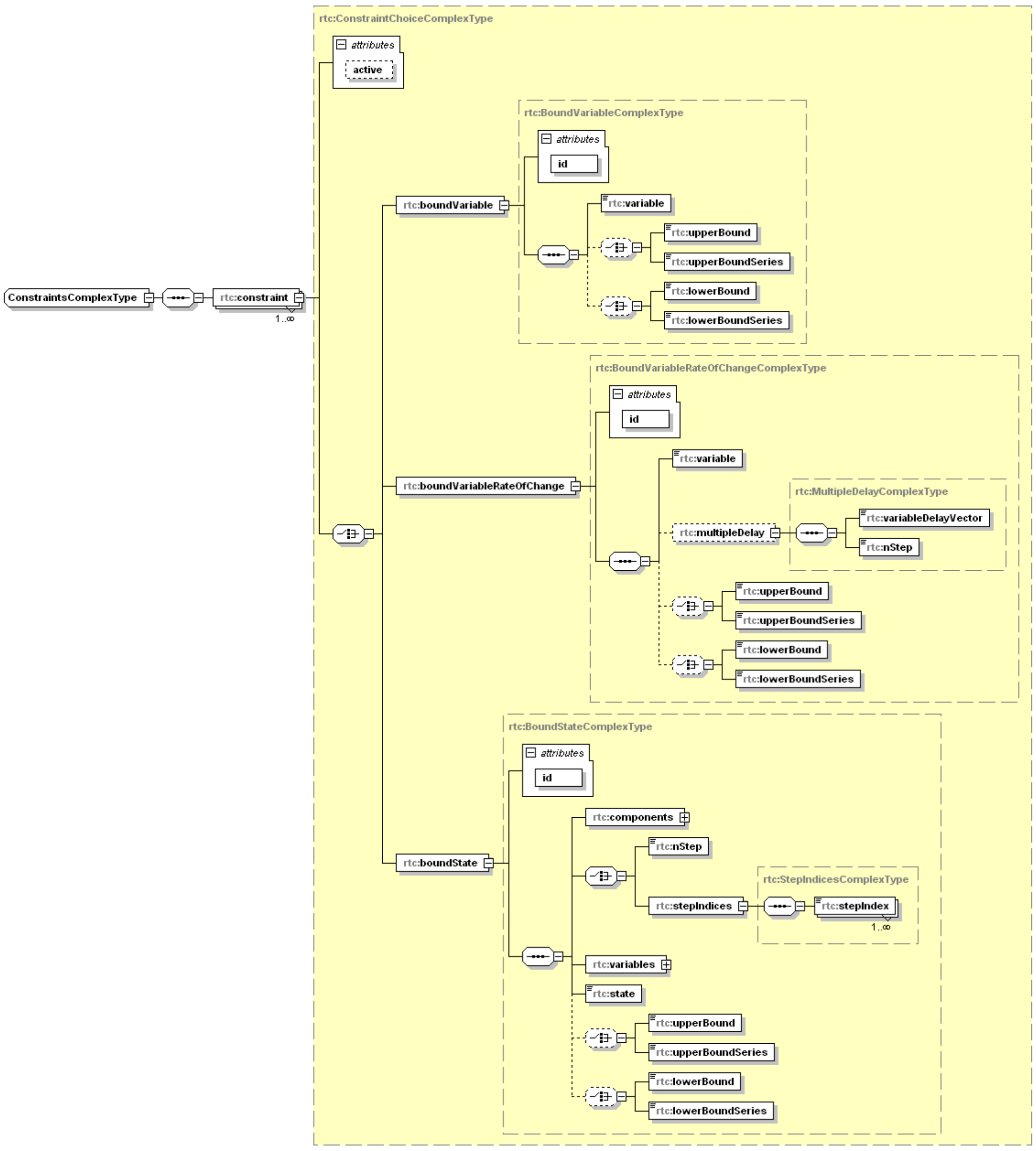


Figure A.9: *Overview about the configuration of optimization problems*

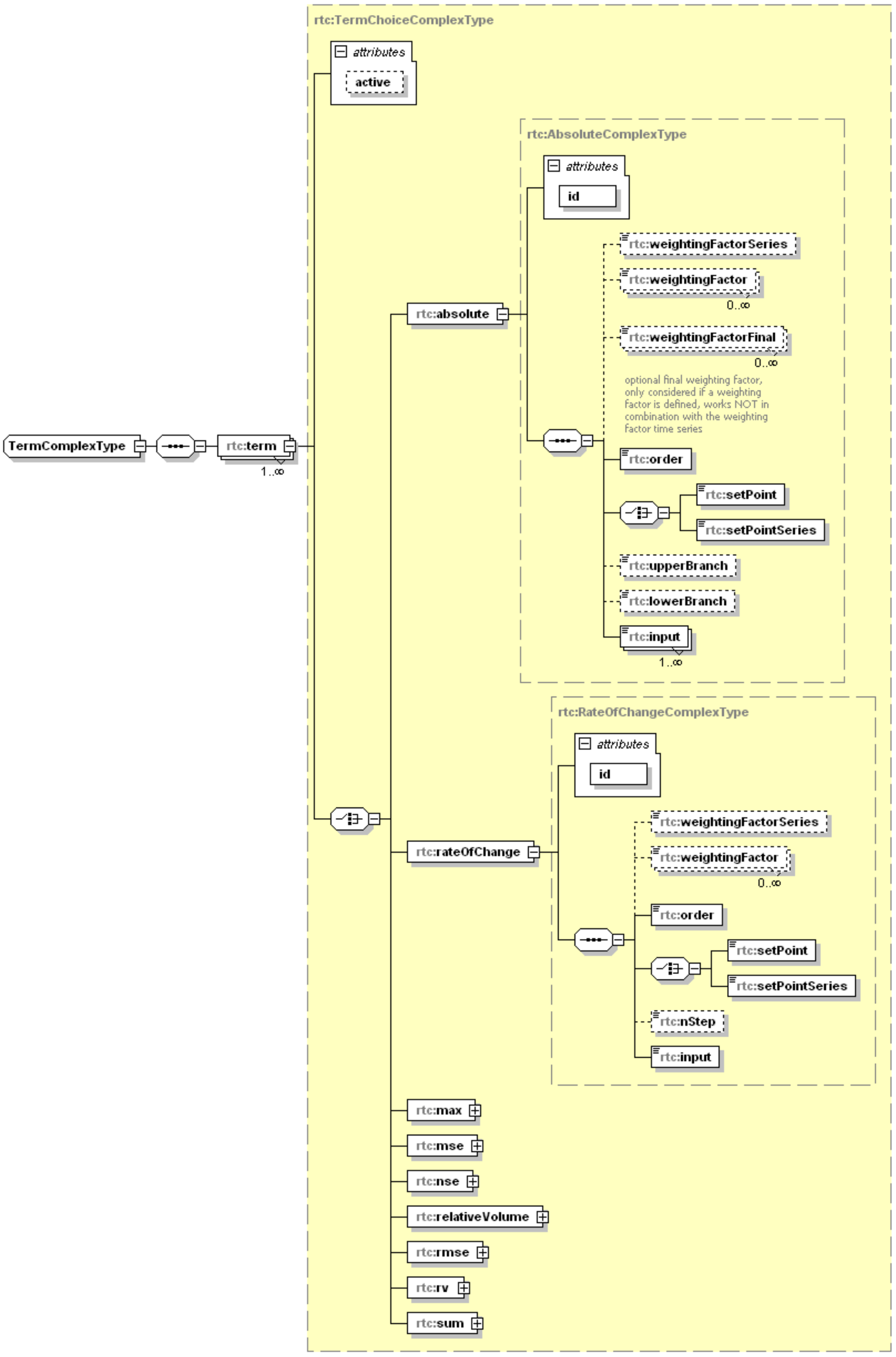
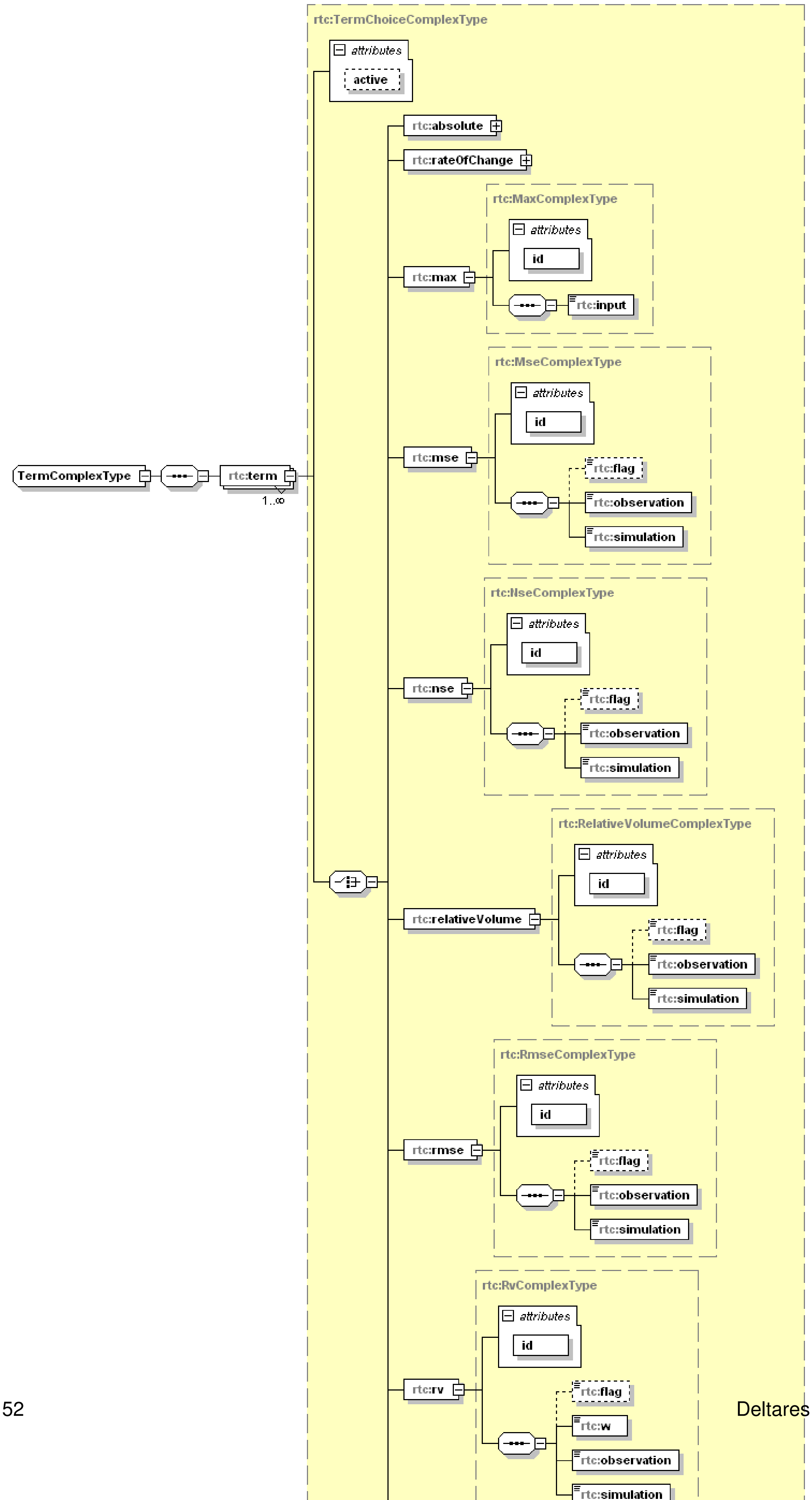


Figure A.10: *Overview about the configuration of optimization problems*


rtc:TermChoiceComplexType
attributes
active
rtc:absolute
rtc:rateOfChange
rtc:MaxComplexType
attributes
id
rtc:max
rtc:input
rtc:MseComplexType
attributes
id
rtc:mse
rtc:flag
rtc:observation
rtc:simulation
TermComplexType
rtc:term
1..∞
rtc:NseComplexType
attributes
id
rtc:nse
rtc:flag
rtc:observation
rtc:simulation
rtc:RelativeVolumeComplexType
attributes
id
rtc:relativeVolume
rtc:flag
rtc:observation
rtc:simulation
rtc:RmseComplexType
attributes
id
rtc:rmse
rtc:flag
rtc:observation
rtc:simulation
rtc:RvComplexType
attributes
id
rtc:rv
rtc:flag
rtc:w
rtc:observation
rtc:simulation

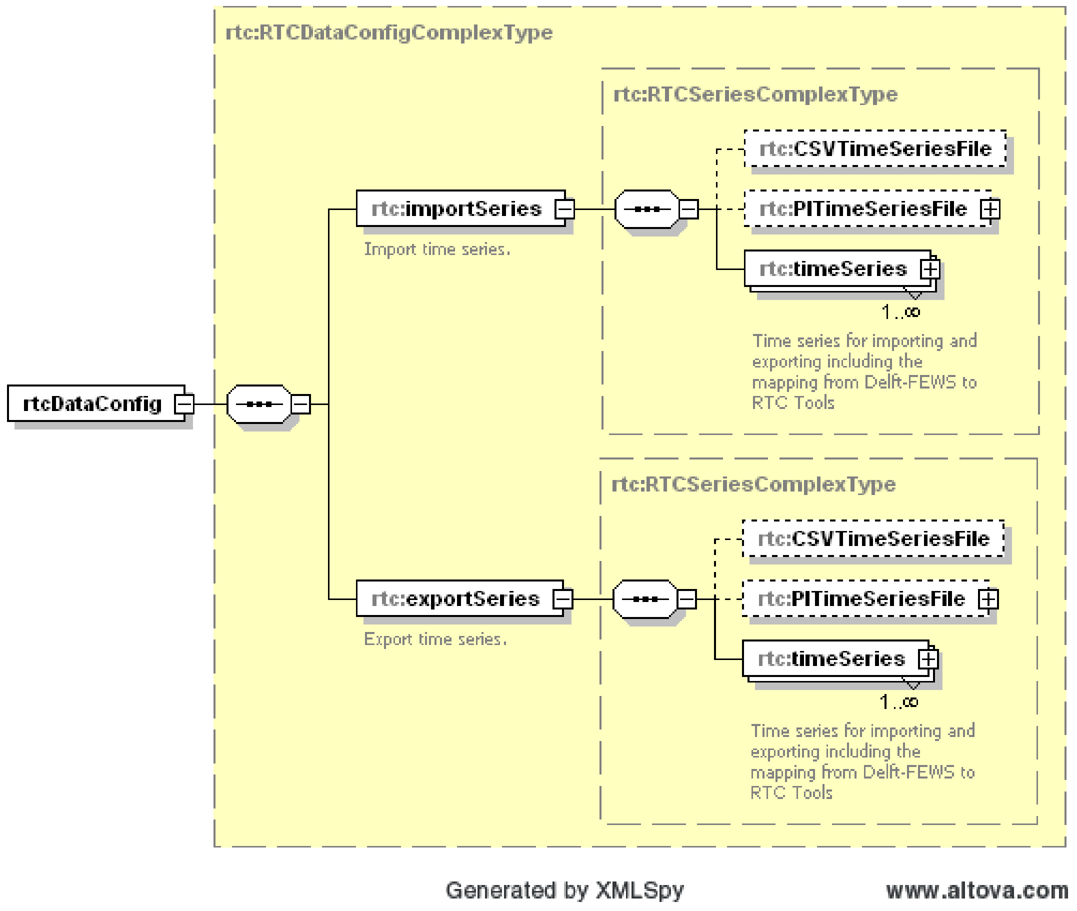


Figure A.12: *xml schema definition for* `rtctoolsDataConfig.xml`

B Details on dedicated rules

B.1 Aebi

Java code for Aebi rule according to the documents:

1 02-1 Hochwasseralarm Murgenthal V2.09.xlsm
2 03-Auszug_Regulierdienst_-_Manual.pdf
3 04-DOCP-#116766-v1-Erluterungen_zu_Murgenthalerberechnung_Sigmaplan.pdf
4 05-DOCP-#149934-v1-Regulierung_Murgenthalberechnung_Anleitung.pdf

```
// compute AareMurg_oKW and limit it to maximum change of 100 m3/s
double AareMurg_oKW = AareMurg;

if (AareMurg_oKW-AareMurg_oKWMin1 > 100.0) {
    AareMurg_oKW = AareMurg_oKWMin1 + 100.0;
} else if (AareMurg_oKW-AareMurg_oKWMin1 < -100.0) {
    AareMurg_oKW = AareMurg_oKWMin1 - 100.0;
}

// compute EGZ west and east
double LangetenFMin4;

if (pLangetenFall==1) {
    LangetenFMin4 = 1.2 * LangetenMin4;
} else {
    LangetenFMin4 = 1.5 * LangetenMin4;
}

double EZGwest = AareMurg_oKW -
  (AareBrueggMin2 + EmmenmattMin4 + 6.0*LangetenFMin4);
EZGwest = Math.max(EZGwest, 0.0);
EZGwest = Math.min(EZGwest, 300.0);
double LangetenF;

if (pLangetenFall==1) {
    LangetenF = 1.2 * Langeten;
} else {
    LangetenF = 1.5 * Langeten;
}
double EZGost = Math.min(6.0 * LangetenF, 100.0 + EZGwest);

// RTG
double RTG;

if (PegelBielersee<=429.6) {
    RTG = pFactorRTG * 700;
} else if (PegelBielersee<=429.7) {
    RTG = pFactorRTG * 720;
} else if (PegelBielersee<=429.8) {
    RTG = pFactorRTG * 730;
} else if (PegelBielersee<=429.9) {
    RTG = pFactorRTG * 750;
} else if (PegelBielersee<=430.0) {
    RTG = pFactorRTG * 760;
} else if (PegelBielersee<=430.1) {
    RTG = pFactorRTG * 780;
} else if (PegelBielersee<=430.15) {
    RTG = pFactorRTG * 800;
} else if (PegelBielersee<=430.2) {
    RTG = pFactorRTG * 810;
} else {
    RTG = pFactorRTG * 820;
}

// provisional maximum discharge at Port
double ProgAbflussMurg = AareBruegg +
  Emmenmatt + Math.max(EZGwest, 0.0) + EZGost;
double ProvMaxAbflussPort = RTG + AareBruegg - ProgAbflussMurg;

// compute discharge change
double MaxAbflussPort;
double ProgAbflusstendenzWehrPort = AareBruegg - ProvMaxAbflussPort;

if (AareBruegg >= AbflussReglPort) {
    if (ProgAbflusstendenzWehrPort < 0.0) {
      MaxAbflussPort = AareBruegg - 50.0;
    } else if (ProgAbflusstendenzWehrPort < 50.0) {
      MaxAbflussPort = AareBruegg - ProgAbflusstendenzWehrPort;
    } else {
      MaxAbflussPort = AareBruegg - 50.0;
    }

} else {
    if (ProgAbflusstendenzWehrPort < -50.0) {
      MaxAbflussPort = AareBruegg + 50.0;
    } else if (ProgAbflusstendenzWehrPort < 100.0) {
      MaxAbflussPort = AareBruegg - ProgAbflusstendenzWehrPort;
    } else {
      MaxAbflussPort = AareBruegg - 100.0;
    }
}

// apply minimum discharge

if (PegelBielersee < 430.00) {
    MaxAbflussPort = Math.max}(MaxAbflussPort, 200.0);
} else if (PegelBielersee <= 430.35) {
    MaxAbflussPort = Math.max}(MaxAbflussPort, 250.0);
} else {
    MaxAbflussPort = Math.max}(MaxAbflussPort, 300.0);
}

double AbflussAenderungPort = AareBruegg - MaxAbflussPort;
```

B.2 Thunersee

The dedicated rule for Thunersee keeps the lake water level on set point in case of active alarm levels orange and red. Furthermore, it takes care of a discharge reduction in case of a threshold crossing at gauge CityplaceBern.

Java code for Thunersee rule according to the documents:

1 1_1_betriebsreglement.pdf

```
// required released volume to reach set point
double s = reservoirStorageCharacteristics.convert(level)
    - reservoirStorageCharacteristics.convert(
      Math.min(levelSetpoint, stateOld[iInLevel]))
    + dt*inflow;

// release at new time step depending on time stepping scheme
double releaseTotal = 0.0;

if (poolRoutingScheme==ModuleModel.EXPLICIT}) {
    releaseTotal = s/dt;
} else if (poolRoutingScheme==ModuleModel.NEWTONRAPHSONBACKTRACKING}) {
    releaseTotal =
      (s-(1.0-theta)*dt*stateOld[iOutReleaseTotal])/(theta*dt);
}

// limiter
releaseTotal = Math.max(releaseTotal, (1.0-
    releaseLimiterPercentage/100.0)*stateOld[iOutReleaseTotal]);
releaseTotal = Math.min(releaseTotal,
    (1.0+releaseLimiterPercentage/100.0)*stateOld[iOutReleaseTotal]);

// reduction due to discharge CityplaceBern
double dischargeBern = releaseTotal + releaseCatchment;
double dischargeBernMax = limiterBern.convert(level);

if (dischargeBern>dischargeBernMax) {
    releaseTotal = Math.max(dischargeBernMax,
      0.7*stateOld[iOutReleaseTotal]);
}

// distribute to outlets
double releaseWeir = Math.max(Math.min(releaseTotal, wCapacity), 0.0);
double releaseTunnel = 0.0;

if (level>tunnelLevelThreshold) {
    releaseTunnel = Math.max(Math.min(releaseTotal-releaseWeir,
      tCapacity), 0.0);
}
releaseTotal = releaseWeir + releaseTunnel;
```

C Configuration examples

C.1 A kinematic wave model

From the case “kinematicWave2” :

```
<?xml version="1.0" encoding="UTF-8"?>
<rtcToolsConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
..\..\..\xsd\rtcToolsConfig.xsd"
xmlns="http://www.wldelft.nl/fews">
  <general>
    <description>test kinematic wave model</description>
    <poolRoutingScheme>ForwardEuler</poolRoutingScheme>
    <theta>0.0</theta>
  </general>
  <components>
    <component>
      <hydraulicModel id="test">
        <node id="N1">
          <storageCharacteristics>
            <storageTable>
              <elevationRecord elevation="0.0" value="0.0"/>
              <elevationRecord elevation="1.0" value="1000000.0"/>
            </storageTable>
          </storageCharacteristics>
          <levelBoundaryCondition>
            <input>
              <level>1HBC</level>
            </input>
          </levelBoundaryCondition>
          <output>
            <level>1H</level>
            <storage>1S</storage>
          </output>
        </node>
        <node id="N2">
          <storageCharacteristics>
            <storageTable>
              <elevationRecord elevation="0.0" value="0.0"/>
              <elevationRecord elevation="1.0" value="1000000.0"/>
            </storageTable>
          </storageCharacteristics>
          <output>
            <level>2H</level>
            <storage>2S</storage>
          </output>
        </node>
        <node id="N3">
          <storageCharacteristics>
            <storageTable>
              <elevationRecord elevation="0.0" value="0.0"/>
              <elevationRecord elevation="1.0" value="1000000.0"/>
            </storageTable>
          </storageCharacteristics>
          <inflowBoundaryCondition>
            <input>
```

```
              <inflow>3QBC</inflow>
            </input>
          </inflowBoundaryCondition>
          <output>
            <level>3H</level>
            <storage>3S</storage>
          </output>
        </node>
        <branch>
          <crossSection>
            <crossSectionTable>
              <elevationRecord elevation="0.0" value="50.0"/>
              <elevationRecord elevation="1.0" value="50.0"/>
              <elevationRecord elevation="2.0" value="50.0"/>
            </crossSectionTable>
          </crossSection>
          <roughness>
            <roughnessTable>
              <elevationRecord elevation="0.0" value="10.0"/>
              <elevationRecord elevation="1.0" value="10.0"/>
            </roughnessTable>
          </roughness>
          <nodeUp>N3</nodeUp>
          <nodeDown>N2</nodeDown>
          <length>10000.0</length>
          <output>
            <discharge>2Q</discharge>
          </output>
        </branch>
        <hydraulicStructure>
          <orifice>
            <width>50.0</width>
            <crestLevel>0.0</crestLevel>
            <contractionCoefficient>0.63</contractionCoefficient>
            <flowDirection>BOTH</flowDirection>
            <nodeUp>N2</nodeUp>
            <nodeDown>N1</nodeDown>
            <input>
              <openingHeight>1DGBC</openingHeight>
            </input>
            <output>
              <discharge>1Q</discharge>
              <openingHeight>1DG</openingHeight>
            </output>
          </orifice>
        </hydraulicStructure>
      </hydraulicModel>
    </component>
  </components>
</rtcToolsConfig>
```

C.2 A neural network model

From the case “neuralNetwork2”:

```
<?xml version="1.0" encoding="UTF-8"?>
<rtcToolsConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
..\..\..\xsd\rtcToolsConfig.xsd"
xmlns="http://www.wldelft.nl/fews">
  <general>
    <description>test kinematic wave model</description>
    <poolRoutingScheme>ForwardEuler</poolRoutingScheme>
    <theta>0.0</theta>
  </general>
  <components>
    <component>
      <neuralNetwork id="test">
        <layer id="L0" name="input layer">
          <neuron id="L0N0">
            <bias>0.0</bias>
            <transferFunction>SigmoidLogistic</transferFunction>
            <input>
              <external weight="1.0">u00</external>
              <internal weight="1.0">L2N0</internal>
            </input>
            <output>
              <x>x00</x>
              <y>y00</y>
            </output>
          </neuron>
          <neuron id="L0N1">
            <bias>0.0</bias>
            <transferFunction>SigmoidLogistic</transferFunction>
            <input>
              <external weight="1.0">u01</external>
              <internal weight="1.0">L2N1</internal>
            </input>
            <output>
              <x>x01</x>
              <y>y01</y>
            </output>
          </neuron>
        </layer>
        <layer id="L1" name="hidden layer">
          <neuron id="L1N0">
            <bias>0.0</bias>
            <transferFunction>SigmoidLogistic</transferFunction>
            <input>
              <internal weight="1.0">L0N0</internal>
              <internal weight="1.0">L0N1</internal>
            </input>
            <output>
              <x>x10</x>
              <y>y10</y>
            </output>
          </neuron>
          <neuron id="L1N1">
            <bias>0.0</bias>
            <transferFunction>SigmoidLogistic</transferFunction>
            <input>
              <internal weight="1.0">L0N0</internal>
              <internal weight="1.0">L0N1</internal>
            </input>
            <output>
              <x>x11</x>
              <y>y11</y>
            </output>
          </neuron>
        </layer>
        <layer id="L2" name="output layer">
          <neuron id="L2N0">
            <bias>0.0</bias>
            <transferFunction>Linear</transferFunction>
            <input>
              <internal weight="2.0">L1N0</internal>
              <internal weight="2.0">L1N1</internal>
            </input>
            <output>
              <x>x20</x>
              <y>y20</y>
            </output>
          </neuron>
          <neuron id="L2N1">
            <bias>0.0</bias>
            <transferFunction>Linear</transferFunction>
            <input>
              <internal weight="2.0">L1N0</internal>
              <internal weight="2.0">L1N1</internal>
            </input>
            <output>
              <x>x21</x>
              <y>y21</y>
            </output>
          </neuron>
        </layer>
      </neuralNetwork>
    </component>
  </components>
</rtcToolsConfig>
```

C.3 A pool routing model

From the case “neuralNetwork2”:

```
<?xml version="1.0" encoding="UTF-8"?>
<rtcToolsConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.wldelft.nl/fews"
xsi:schemaLocation="http://www.wldelft.nl/fews
..\..\..\xsd\rtcToolsConfig.xsd">
  <general>
    <description>singapore2</description>
    <poolRoutingScheme>Theta</poolRoutingScheme>
    <theta>1.0</theta>
  </general>
  <components>
    <component>
      <reservoir id="Reservoir1">
        <storageCharacteristics>
          <storageTable>
            <elevationRecord elevation="-1.0" value="0"/>
            <elevationRecord elevation="0.0" value="60000.0"/>
          </storageTable>
        </storageCharacteristics>
        <controlledOutlet id="Reservoir1.Hydropower" name="Hydropower">
          <capacityCharacteristics>
            <capacityTable>
              <elevationRecord elevation="-1.0" value="20.0"/>
              <elevationRecord elevation="1.0" value="20.0"/>
            </capacityTable>
          </capacityCharacteristics>
          <input>
            <release>QH.opt</release>
          </input>
          <output>
            <release>QH.sim</release>
          </output>
        </controlledOutlet>
        <uncontrolledOutlet id="Reservoir1.Spillway" name="Spillway">
          <capacityCharacteristics>
            <capacityEquation>
              <equation>
                <a>340.9789827345178</a>
                <b>-0.1</b>
                <c>1.5</c>
              </equation>
            </capacityEquation>
          </capacityCharacteristics>
          <output>
            <release>QS.sim</release>
          </output>
        </uncontrolledOutlet>
        <input>
          <inflow>I.obs</inflow>
        </input>
        <output>
          <inflow>I.sim</inflow>
          <release>Q.sim</release>
          <storage>S.sim</storage>
          <level>H.sim</level>
        </output>
      </reservoir>
    </component>
  </components>
</rtcToolsConfig>
```

C.4 A Delft-FEWS configuration

From the case “switzerland2”:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<generalAdapterRun xmlns="http://www.wldelft.nl/fews"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wldelft.nl/fews
  http://fews.wldelft.nl/schemas/version1.0/generalAdapterRun.xsd">
<general>
  <rootDir>$RTCMODULE_DIR$/AareDreiSeen</rootDir>
  <workDir>%ROOT_DIR%</workDir>
  <exportDir>%ROOT_DIR%/import</exportDir>
  <importDir>%ROOT_DIR%/export</importDir>
  <dumpFileDir>$DUMP_DIR$</dumpFileDir>
  <dumpDir>%ROOT_DIR%</dumpDir>
  <diagnosticFile>%ROOT_DIR%/diag.xml</diagnosticFile>
</general>
```

D RTC-Tools Application Programming Interface (RTC-Tools API)

D.1 Extension of the model library

D.1.1 Introduction

C++ classes of the model library are subdivided into components, rules and triggers available in the folders:

```
\RTCTools\src\schematization\components
\RTCTools\src\schematization\rules
\RTCTools\src\schematization\triggers
\RTCTools\src\schematization\
```

where the last folder includes more general classes, e. g. the unitDelay operator being used both as a trigger, rule or component.

You need to derive your own class from the available parent classes **component**, **rule** or **trigger**. If you want to use it both as a trigger and component, derive it also from both parent classes. The same procedure is valid for any other combination.

Besides writing the code of the new class, the class needs a factory method in the class **schematization** for constructing class instances. Furthermore, we suggest using the available xml/ data binding procedure for configuring new model library elements in the available `rtcToolsConfig`.

D.1.2 Class design

D.1.2.1 Basics on variables, member functions, inheritance

It is good practice to split the code into a header file (`*.h`) with the class declaration and another file with the implementation (`*.cpp`). Check the `arma.h` / `arma.cpp` class in

```
\src\schematization\components\
```

as an example. The source code is documented according to the requirements of Doxygen.

We advise to declare all variables as private and encapsulate them in the class. Their initialization is done based on the constructor. References to input and output time series in the RTC-Tools time series model are supplied as indices via the constructor. We use the index -1 by convention for indicating that an optional time series is not supplied.

D.1.2.2 Implementing the simulation and adjoint mode

The simulation mode (`void solve(...)`) needs to be implemented for all elements, i.e. for components, rules and triggers. The adjoint mode (`void solveDer(...)`) is executed ONLY for the components. Thus, if you implement a rule or trigger only, you can keep the body of the member function empty.

Consider if your time series input, states and outputs refers to the old or new time step. `stateOld[index]` picks up a value from the old time step, `stateNew[index]` from the new one. Here are some rules for deciding about the implementation:

- Inputs should pick up data from the new time step `stateNew[index]`. In some cases, you may want to add some flexibility to get data either from the old or the new time step according to your configuration settings. In that case use `stateOld[index]` and the factory method will add the number nSeries to the index, if the new time step is configured (dirty C hack on pointers!).
- States are always picked up from the old time step `stateOld[index]` and written to the new time step `stateNew[index]`.
- Outputs are always written to the new time step `stateNew[index]`.

D.1.3 xsd design / xml data binding / factory method

We suggest making your new element configurable by including it in the XSD file

```
\RTCTools\xsd\rtcToolsConfig.xsd
```

It includes a description of the content of the corresponding xml file with the model configuration. If the xsd is extended, execute the embedded data binder (http://www.codesynthesis.com/products/xsd/) by executing `/RTCTools/xsd/update_XSD.bat` for generating updated classes for reading the xml file (`/RTCTools/src/dataBinding/...`).

The factory method for constructing instances of your new class can be found in the class "schematization". You need to include the header file of your new class and implement the related factory function in the code base. Make use of the data binding class for extracting the required data for the constructor of the

D.2 Extension of the objective function library

D.2.1 Introduction

D.2.2 xsd design / data binding

D.2.3 Class design

D.3 Adding optimizers

D.3.1 Introduction

D.3.2 xsd design / data binding

D.3.3 Class design