D5.7	**Prototype of data reconciliation in OpenModelica**
Access[1]:	**PU**
Type[2]:	**SW**
Version:	**1.0**
Due Dates[3]:	**M24, M36, M40**

Open Cyber-Physical System Model-Driven Certified Development

[1] Access classification as per definitions in PCA; PU = Public, CO = Confidential. Access classification per deliverable stated in FPP.

[2] Deliverable type according to FPP, note that all non-report deliverables must be accompanied by a deliverable report.

[3] Due month(s) according to FPP.

Executive summary[4]:

Data reconciliation aims at improving the accuracy of measurements by reducing the effect of random errors in the data. The principle difference between data reconciliation and other data improvements techniques is that data reconciliation uses a model to express the physical constraints on the variables of interest and adjusts their measured values such that the estimates satisfy the constraints: the variables are thus reconciled.

The objective of this deliverable is to allow the use of Modelica models to express the constraints on the variables of interest. The benefit is to be able to reuse validated Modelica models developed for other purposes such as power plant sizing, control, design verification or monitoring, thus avoiding the heavy costs of model development, verification and validation in other dedicated tools.

This deliverable implements in OpenModelica:

1. A new algorithm that extracts automatically the relevant constraint equations from the Modelica models;
2. The computation of the Jacobian matrix of the constraint equations with respect to the variables of interest;
3. The algorithm that computes the best estimates (the reconciled values) of the variables of interest from the set of constraint equations, the Jacobian matrix and the covariance matrix according to the mathematical procedure given in the VDI 2048 standard: Control and quality improvement of process data and their uncertainties by means of correction calculation for operation and acceptance tests.

The following inputs are given by the user on the Modelica model:

1. The list of variables of interest which specifies which variables are to be reconciled;
2. The covariance matrix which specifies the errors (uncertainties) on the variables of interest;
3. The equations that cannot be considered as valid constraints because they are not considered as exact. These equations are tagged by the user as approximated.

The algorithm computes the reconciled values and reduced uncertainties for the variables of interest. By performing statistical checks on the reconciled values, serious measurement errors can be identified.

[4] It is mandatory to provide an executive summary for each deliverable.

Deliverable Contributors:

	Name	Organisation	Primary role in project	Main Author(s)[5]
Deliverable Leader[6]	Adrian Pop	LIU		
Contributing Author(s)[7]	Daniel Bouskela	EDF		X
	Audrey Jardin	EDF		X
	Arunkumar Palanisamy	LIU		X
Internal Reviewer(s)[8]	Markus Högberg	EQUA		
	Sune Horkeby	Siemens TU		

Document History:

Version	Date	Reason for Change	Status[9]
0.1	16/11/2018	First Draft Version	Draft
0.2	28/11/2018	New Version with EQUA's reviex	InReview
1.0	30/11/2018	New version with minor updates coming from the review	Released

[5] Indicate Main Author(s) with an "X" in this column.
[6] Deliverable leader according to FPP, role definition in PCA.
[7] Person(s) from contributing partners for the deliverable, expected contributing partners stated in FPP.
[8] Typically person(s) with appropriate expertise to assess deliverable structure and quality.
[9] Status = "Draft", "In Review", "Released".

CONTENTS

ABBREVIATIONS

List of abbreviations/acronyms used in document:

Abbreviation	**Definition**
BLT	Block lower triangular

1 EXTRACTION ALGORITHM

The extraction algorithm extracts automatically the equations that are used by the data reconciliation procedure from the Modelica model.

1.1 Motivation

The objective of data reconciliation is to use physical models to decrease measurement uncertainties on physical quantities. Data reconciliation is possible only when redundant measurements are available for a given physical quantity.

Let m_1 and m_2 be two different measurements on physical quantities M_1 and M_2.

If there is a known physical law assumed to be exact $\varphi(M_1, M_2) = 0$ between M_1 and M_2, then m_1 and m_2 are redundant, but in general $\varphi(m_1, m_2) \neq 0$ because of measurement random errors.

The principle of data reconciliation is to find the best estimates $\hat{m}_1$ and $\hat{m}_2$ of M_1 and M_2 by minimizing

$$J = \sum_i \left(\frac{\hat{m}_i - m_i}{\sigma_i} \right)^2 \tag{1.1}$$

subject to

$$\varphi(\hat{m}_1, \hat{m}_2) = 0 \tag{1.2}$$

The σ_i are the standard deviations of the measurement errors. It is assumed here that the measurements are statistically independent, so that $cov(\varepsilon_i, \varepsilon_j) = 0$, where ε_i is the random error on variable m_i. A more general formulation is given in Chapter 3.

The objective is to reuse existing Modelica models to provide φ.

The problem to be solved is that φ cannot be identical to the Modelica model because, as valid Modelica models are square, $\varphi(\hat{m}_1, \hat{m}_2) = 0$ would provide a unique solution, so the optimization algorithm for J could not adjust $\hat{m}_1$ and $\hat{m}_2$.

The proposed solution is to extract automatically φ from the Modelica model such that φ has no square subsystems and $\varphi(\hat{m}_1, \hat{m}_2) = 0$ gives an infinite number of solutions for $\hat{m}_1$ and $\hat{m}_2$. Then the unique solution for $\hat{m}_1$ and $\hat{m}_2$ is given by the minimum value for J that verifies the constraint $\varphi(\hat{m}_1, \hat{m}_2) = 0$.

1.2 Example

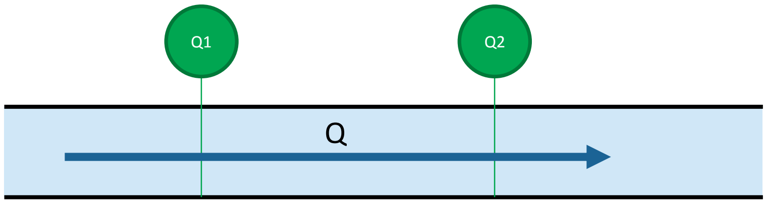

Fig. 1. Pipe with two sensors

Q1 and Q2 measure the mass flow rate Q through the pipe.

Q1 and Q2 are the variable of interest, subject to the physical constraint Q1 = Q2.

The Modelica model is the following.

```
model Pipe
  parameter Real p=2; // Unit: kg/s
  Real Q1, Q2;
  Real y1, y2;
equation
  Q1 = y1;
  Q2 = y2;
  y1 = y2;
  y1 = p;
end Pipe;
```

Fig. 2. Modelica model of the pipe

The Modelica Pipe model states that Q1 = Q2 = p = 2 kg/s;

The problem is: how to extract Q1 = Q2 from the Pipe model, therefore eliminating the intermediate variables y1 and y2, and getting rid of equation y1 = p?

1.3 Notation

$\mathcal{M}$ denotes the original valid Modelica model.

C denotes the set of constraints equations.

S denotes the set of intermediate equations.

x denotes the set of variables of interest. $x = [\, x_i \,]^T$, where x_i denotes the i^{th} variable of interest.

y denotes the set of intermediate variables. $y = [\, y_j \,]^T$, where y_j denotes the j^{th} intermediate variable.

F denotes the Jacobian matrix of *C*. *F* is not square.

F_{ij} denotes the components of F.

BLT denotes the block lower triangular decomposition of $\mathcal{M}$.

The initial BLT is the BLT constructed from $\mathcal{M}$ which is assumed to be a valid Modelica model.

All blocks B in the initial BLT are square: they have as many equations as variables.

B.rank denotes the rank of block B in the BLT.

B.size denotes the size of B in the initial BLT, i.e. the number of equations or the number of variables in B.

The objective of the extraction algorithm is to remove equations from the initial BLT so that all blocks B in the BLT are underdetermined.

B.square denotes the square (determined) or non-square (underdetermined) nature of block B. B.square = true means that B is square, B.square = false means that B is non-square.

B.nvar denotes the number of variables of interest in block B. Therefore, the number of intermediate variables in B is B.size – B.nvar.

1.4 Principle

The set of all equations is denoted $\mathcal{M}$. It is assumed that $\mathcal{M}$ is a valid Modelica model: it is a non-singular square system that has a unique solution.

Basic principles:

1. The set of equations *C* constraining the variables of interest x_i must be underdetermined for *all* x_i in order to be able be reconciled, therefore ensure that

$$\forall \text{ B} \in \text{BLT, B.nvar} \geq 1 \Rightarrow \text{B.square} = \text{false}$$

   Consequently:

   a. Remove constraint equations (e.g. boundary conditions) from the BLT in order not to have square systems of constraint equations;

   b. Insert remaining constraint equations into set *C*.

2. The intermediate variables y_j involved in set *C* must be eliminated from the constraint equations. Consequently:

   a. Insert the equations that compute the y_j from the x_i into set S.

In the example, this amounts to removing equations y1 = p and p = 2, inserting Q1 = y1 in set *C* and inserting Q2 = y2 and y1 = y2 in set *S*. Other combinations for sets *C* and *S* are possible such as y1 = y2 in set *C* and Q1 = y1 and Q2 = y2 in set *S*.

1.5 Preliminary definitions

1.5.1 Targets

The short target of a block B in the BLT the set of blocks B' such that B'.rank ≥ B.rank and B' and B share at least one variable.

The target of a block B in the BLT is constructed by the recursive union of the short target of B, then the short targets of all blocks in the short target of B, then the short targets of all blocks in the short targets of all blocks in the short targets of B, etc.

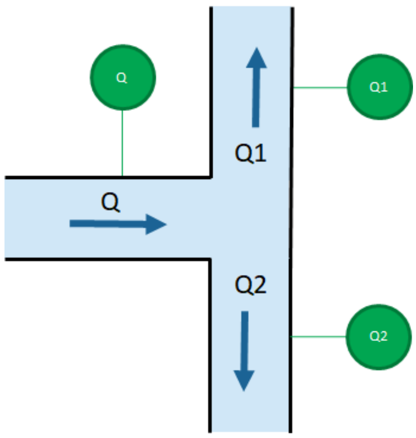

Fig. 3. Splitter with three sensors

```
model Splitter
  Real Q, Q1, Q2;
  Real y, y1, y2, a;
  Real A, Y;
equation
  Y = 2;        // Eq1
  y = Y;        // Eq2
  A = 0.5;      // Eq3
  a = A;        // Eq4
  y1 = a*y;     // Eq5
  y = y1 + y2;  // Eq6
  Q = y;        // Eq7
  Q1 = y1;      // Eq8
  Q2 = y2;      // Eq9
end Splitter;
```

Fig. 4. Modelica model of the splitter

	Y	y	A	a	Q	y1	y2	Q1	Q2
Eq1	x ← B1								
Eq2	x	x ← B2							
Eq3			x ← B3						
Eq4			x	x ← B4					
Eq7		x			x ← B5				
Eq5		x		x		x ← B6			
Eq6		x				x	x ← B7		
Eq8						x		x ← B8	
Eq9							x		x ← B9

Fig. 5. BLT of the splitter

For the BLT in Fig. 5, the targets are given in Fig. 6.

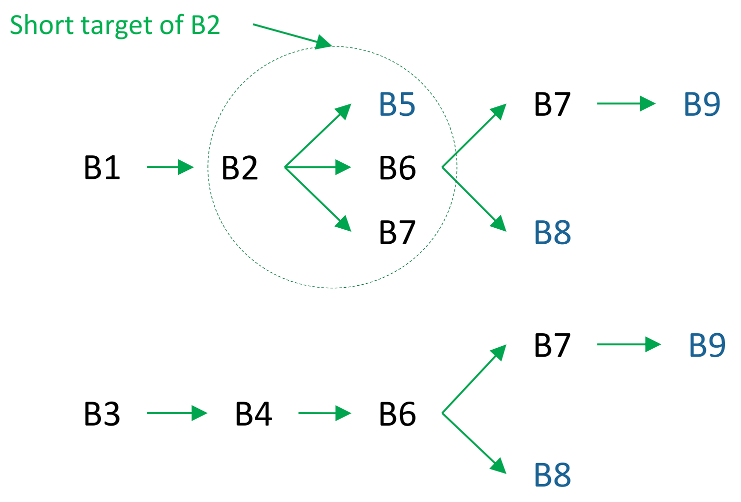

Fig. 6. Targets

B1.target = { B1, B2, B5, B6, B7, B8, B9 }
B2.target = { B2, B5, B6, B7, B8, B9 }
B3.target = { B3, B4, B6, B7, B8, B9 }
B4.target = { B4, B6, B7, B8, B9 }
B5.target = { B5 }
B6.target = { B6, B7, B8, B9 }
B7.target = { B7, B9 }
B8.target = { B8 }
B9.target = { B9 }

1.5.2 Square and non-square blocks

A block B in the BLT is square iff it has as many equations as variables. A variable can be a variable of interest *x* or an intermediate variable *y*.

The BLT decomposition of a valid Modelica model features only squares blocks.

A block B in the BLT is non-square iff it has less equations than variables, or equivalently iff it is underdetermined.

If block B' belongs to the target of B, and if B is non-square, then B' is non-square:

$$\text{B is non-square} \Rightarrow \forall \text{ B'} \in \text{B.target, B is non-square.} \tag{1.3}$$

1.5.3 Block B°

For a block B in the BLT, B° denotes the block of lowest rank in B.target such that B° contains at least one variable of interest.

B° has the following property:

$$\text{B}^\circ \in \text{B.target} \tag{1.4}$$

If B° is non-square, then all blocks in B°.target are non-square:

$$\text{B}^\circ \text{ is non-square} \Rightarrow \forall \text{ B} \in \text{B}^\circ\text{.target, B is non-square} \tag{1.5}$$

For the BLT above, B1° = B2° = B3° = B4° = B5° = B5, B6° = B8° = B8, B7°= B9°= B9.

Each block B in the BLT has at most one B°.

1.5.4 Block $B^\bullet$

For a block B in the BLT, $B^\bullet$ denotes the block of lowest rank in the BLT without any predecessor in the BLT such that $B \in B^\bullet$.target.

$B^\bullet$ has the following properties:

$$B \in B^\bullet.\text{target} \tag{1.6}$$

$$\forall B' \in \text{BLT},\ B^\bullet \in B'.\text{target} \Rightarrow B^\bullet = B' \tag{1.7}$$

For the BLT above $B1^\bullet = B2^\bullet = B5^\bullet = B6^\bullet = B7^\bullet = B8^\bullet = B9^\bullet = B1$, $B3^\bullet = B4^\bullet = B3$.

Each block B in the BLT has exactly one $B^\bullet$.

1.6 Algorithm

1. Find square and non-square blocks:
    a. Set C = { } and set S = { }
    b. For all B in BLT such that B° exists set B.square = true and set B°.used = false
    c. For all B in BLT such that B° exists
        i. For all B' in B.target such that B'.rank > B°.rank set B'.square = false

Comments: The idea is to remove one equation from all B° so that all blocks B in the BLT that have at least one variable of interest are non-square. This is done in section 2 of the algorithm. Section 1 tags as non-square all blocks following all blocks B°.

2. Extract equations from $\mathcal{M}$ into C and S:
    a. For all B in BLT such that B° exists and B.square = true
        If B ≠ B° then
            If B°.square = true and B°.used = false then // $B = B^\bullet$
                i. Insert B.size – 1 equations of B into S
                ii. Insert 1 equation of B° into S
                iii. Set B°.used = true
            else if B°.square = false or B°.used = true then // $B^\bullet < B < B^\circ$
                i. Insert B.size equations of B into S
        else // B = B°
            i. Insert B.nvar – 1 equations of B into C
            ii. Insert B.size – B.nvar equations of B into S
    b. For all B in BLT such that B° exists and B.square = false // B ≥ B°
        i. Insert B.nvar equations of B into C
        ii. Insert B.size – B.nvar equations of B into S

Comments: The BLT must be scanned by increasing block ranks. The idea is, for each block B, to insert in *C* as many equations in B as there are variables of interest in B, and insert in *S* as many equations in B as there are intermediate variables in B, with the following exceptions:

A. If B = B°, then insert as many equations in *C* as there are variables of interest in B minus one. This ensures that B° will be underdetermined with respect to the variables of interest, therefore all blocks B in the BLT with variables of interest. This is handled in section

B. If B = B•, then insert as many equation in *S* as there are intermediate variables in B minus one, and insert in *S* the equation in B° that has not been inserted in *C*. This is done in order to avoid breaking the relation between the equation in B° that is not in *C* and the intermediate variable in B• which is involved in this equation, so that the intermediate variable can be eliminated.

Section 2.a handles all blocks preceding B° starting with B• and finishing with B°.

Section 2.b handles all blocks following B°.

Only equations that are not tagged as approximated must be inserted into *C* and *S*. It is therefore possible that a square system *S* cannot extracted if the problem is ill-posed by the user, i.e. if too many equations are tagged as approximated.

1.7 Example

The Splitter model (cf. §1.5.1) is used.

The objective is to extract Q = Q1 + Q2 and Q1 = 0.5*Q.

<u>Algorithm: initial step</u>

C = { }, S = { }.

B1.size = 1, B1.nvar = 0, B1° = B5, B1.square = true, B1.target = { B1, B2, B5, B6, B7, B8, B9 }

B2.size = 1, B2.nvar = 0, B2° = B5, B2.square = true, B2.target = { B2, B5, B6, B7, B8, B9 }

B3.size = 1, B3.nvar = 0, B3° = B8, B3.square = true, B3.target = { B3, B4, B6, B7, B8, B9 }

B4.size = 1, B4.nvar = 0, B4° = B8, B4.square = true, B4.target = { B4, B6, B7, B8, B9 }

B5.size = 1, B5.nvar = 1, B5° = B5, B5.square = true , B5.used = false, B5.target = { B5 }

B6.size = 1, B6.nvar = 0, B6° = B8, B6.square = false, B6.target = { B6, B7, B8, B9 }

B7.size = 1, B7.nvar = 0, B7° = B9, B7.square = false, B7.used = false, B7.target = { B7, B9 }

B8.size = 1, B8.nvar = 1, B8° = B8, B8.square = false, B8.used = false, B8.target = { B8 }

B9.size = 1, B9.nvar = 1, B9° = B9, B9.square = false, B9.used = false, B9.target = { B9 }

<u>Algorithm: all blocks in the BLT</u>

B1

B1.square = true and B1 ≠ B1° and B1°.square = true and B1°.used = false ➔ Insert B1.size – 1 = 0 equation of B1 into S and insert 1 equation of B1° into S and set B1°.used = true ➔ Eq7 inserted into S

C = { }, S = { Eq7 }, B5.used = true

B2

B2.square = true and B2 ≠ B2° and B2°.square = false or B2°.used = true ➔ Insert B2.size = 1 equation of B2 into S ➔ Eq2 inserted into S

C = { }, S = { Eq7, Eq2 }

B3

B3.square = true and B3 ≠ B3° and B3°.square = false or B3°.used = true ➔ Insert B3.size = 1 equation of B3 into S ➔ Eq3 inserted into S

C = { }, S = { Eq7, Eq2, Eq3 }

B4

B4.square = true and B4 ≠ B4° and B4°.square = false or B4°.used = true ➔ Insert B4.size = 1 equation of B4 into S ➔ Eq4 inserted into S

C = { }, S = { Eq7, Eq2, Eq3, Eq4 }

B5

B5.square = true and B5 = B5° ➔ Insert B.nvar – 1 = 0 equation of B into C and insert B.size – B.nvar = 0 equation of B into S ➔ no change

C = { }, S = { Eq7, Eq2, Eq3, Eq4 }

B6

B6.square = false ➔ Insert B6.nvar = 0 equation of B6 into C and insert B6.size – B6.nvar = 1 equation of B6 into S ➔ Eq5 is inserted into S

C = { }, S = { Eq7, Eq2, Eq3, Eq4, Eq5 }

B7

B7.square = false ➔ Insert B7.nvar = 0 equation of B7 into C and insert B7.size – B7.nvar = 1 equation of B7 into S ➔ Eq6 is inserted into S

C = { }, S = { Eq7, Eq2, Eq3, Eq4, Eq5, Eq6 }

B8

B8.square = false ➔ Insert B8.nvar = 1 equation of B8 into C and insert B8.size – B8.nvar = 0 equation of B8 into S ➔ Eq8 is inserted into C

C = { Eq8 }, S = { Eq7, Eq2, Eq3, Eq4, Eq5, Eq6 }

B9

B9.square = false ➔ Insert B9.nvar = 1 equation of B9 into C and insert B9.size – B9.nvar = 0 equation of B9 into S ➔ Eq9 is inserted into C

C = { Eq8, Eq9 }, S = { Eq7, Eq2, Eq3, Eq4, Eq5, Eq6 }

<u>Proof that C = { Eq8, Eq9 }, S = { Eq7, Eq2, Eq3, Eq4, Eq5, Eq6 } is correct</u>

Eq8: Q1 = y1 = a*y (from Eq5) = a*Q (from Eq7) = A*Q (from Eq4) = 0.5*Q (from Eq3)

Eq9: Q2 = y2 = y – y1 (from Eq6) = Q – y1 (from Eq7) = Q – a*y (from Eq5) = Q – a*Q (Eq7) = Q – A*Q (Eq4) = Q – 0.5*Q (Eq3)

Therefore the two equations Q1 = 0.5*Q and Q2 = Q – 0.5*Q are extracted, which is equivalent to Q = Q1 + Q2 and Q1 = 0.5*Q.

Notice that Eq2 that computes Y from y is not useful. Therefore, set *S* contains more equations than necessary.

2 COMPUTATION OF THE JACOBIAN MATRIX *F*

C is the set of constraints equations. *C* depends on the variables of interest *x* and the intermediate variables *y*: *C* = *C* (*x*, *y*).

S is the square system of equations that compute the intermediate variables *y* as a function of the variables of interest *x*:

$$S(x, y) = 0 \tag{2.1}$$

The Jacobian matrix of *C* is defined as follows:

$$F = \frac{dC}{dx} \tag{2.2}$$

where *C* is the set of constraints equations.

As *C* depends on the variables of interest *x* and the intermediate variables *y*

$$F = \frac{dC}{dx} = \frac{\partial C}{\partial x} + \frac{\partial C}{\partial y} \cdot \frac{dy}{dx} \tag{2.3}$$

The Jacobian matrix of *S* is

$$\frac{dS}{dx} = \frac{\partial S}{\partial x} + \frac{\partial S}{\partial y} \cdot \frac{dy}{dx} = 0 \tag{2.4}$$

where *S* is the set of intermediate equations. The Jacobian matrix of *S* is zero because $S(x, y) = 0$.

Therefore

$$\frac{dy}{dx} = -\left[\frac{\partial S}{\partial y}\right]^{-1} \cdot \frac{\partial S}{\partial x} \tag{2.5}$$

and from (2.2)

$$F = \frac{\partial C}{\partial x} - \frac{\partial C}{\partial y} \cdot \left[\frac{\partial S}{\partial y}\right]^{-1} \cdot \frac{\partial S}{\partial x} \tag{2.6}$$

In practice, in order to avoid evaluating the inverse of matrices, *F* is computed as follows:

$$F_{ij} = \frac{dC_i}{dx_j} = \frac{\partial C_i}{\partial x_j} + \sum_{k=1}^{card(S)} \alpha_{jk} \cdot \frac{\partial C_i}{\partial y_k} \qquad 1 \le i \le card(C) \tag{2.7}$$

with the coefficients α_{jk} being the solution of the square system

$$\frac{\partial S_i}{\partial x_j} + \sum_{k=1}^{card(S)} \frac{\partial S_i}{\partial y_k} \cdot \alpha_{jk} = 0 \qquad 1 \le i \le card(S) \tag{2.8}$$

3 DATA RECONCILIATION PROCEDURE

3.1 Notation

f	Auxiliary conditions (vector of contradictions)
F	Jacobian matrix of the constraint equations
J	Objective function to be minimized
J^*	Lagrange function of J
r	Number of auxiliary conditions
S_x	Covariance matrix of the measured values x
$S_{\hat{x}}$	Covariance matrix of the reconciled values $\hat{x}$
x	Measured values of the variables of interest
$\hat{x}$	Reconciled values of the variables of interest

3.2 Data reconciliation procedure

The auxiliary conditions are the equations that bind the reconciled values:

$$f(\hat{x}) = 0 \tag{3.1}$$

In practice, (3.1) cannot be exactly verified if f is nonlinear.

f can be computed from the constraints equations set C as follows:

$$\begin{cases} f(x) = C(x, y) \\ S(x, y) = 0 \end{cases} \tag{3.2}$$

In (3.2), the intermediate equations set S eliminates the intermediate variables y.

The objective of the data reconciliation procedure is to minimize the distance between $\hat{x}$ and x corresponding to the S_x^{-1} quadratic norm [1]:

$$J = \|\hat{x} - x\|^2_{S_x^{-1}} = (\hat{x} - x)^T \cdot S_x^{-1} \cdot (\hat{x} - x) \tag{3.3}$$

The reconciled values are then given by:

$$\hat{x} = x - S_x \cdot F^T \cdot f^* \tag{3.4}$$

where S_x is the covariance matrix provided by the user and f^* is the solution of

$$(F \cdot S_x \cdot F^T) \cdot f^* = f \tag{3.5}$$

Using sets C and S, (3.5) amounts to solving the following system:

$$\begin{cases} (F \cdot S_x \cdot F^T) \cdot f^* = C(x, y) \\ S(x, y) = 0 \end{cases} \tag{3.6}$$

The covariance matrix of the reconciled values is given by

$$S_{\hat{x}} = S_x - S_x \cdot F^T \cdot F^* \tag{3.7}$$

where F^* is the solution of

$$(F \cdot S_x \cdot F^T) \cdot F^* = F \cdot S_x \tag{3.8}$$

The complete procedure is given in Fig. 7.

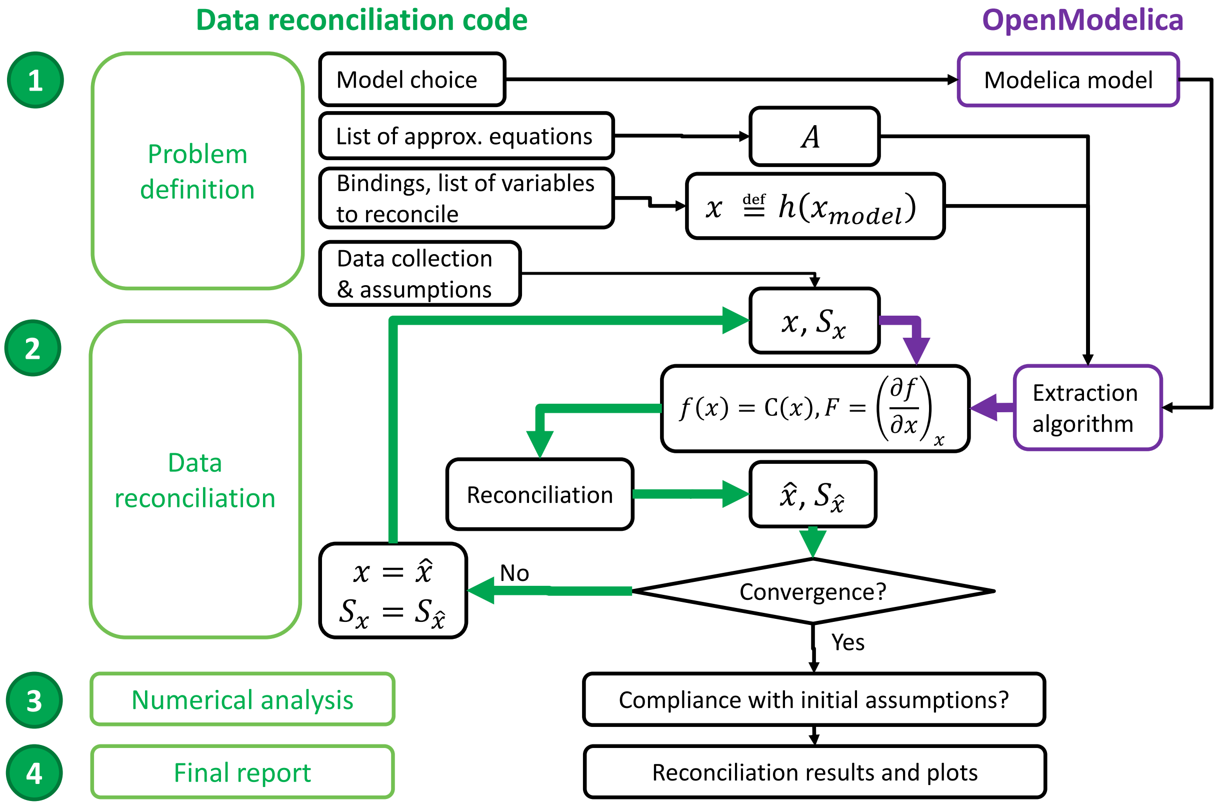

Fig. 7. Data reconciliation procedure

The value of the Lagrange function of J at the minimum point is

$$J^* = J + 2 \cdot [f + F \cdot (\hat{x} - x)]^T \cdot f^* \tag{3.9}$$

where J is given by (3.3), f is given by (3.2), f^* is given by (3.6) and F is given by (2.7)-(2.8).

The data reconciliation procedure is iteratively applied until the following convergence test is successful (see Fig. 7):

$$\frac{J^*}{r} \leq \varepsilon \tag{3.10}$$

where r is the number of constraints equations i.e. the cardinality of set C and ε is a convergence parameter set by the user.

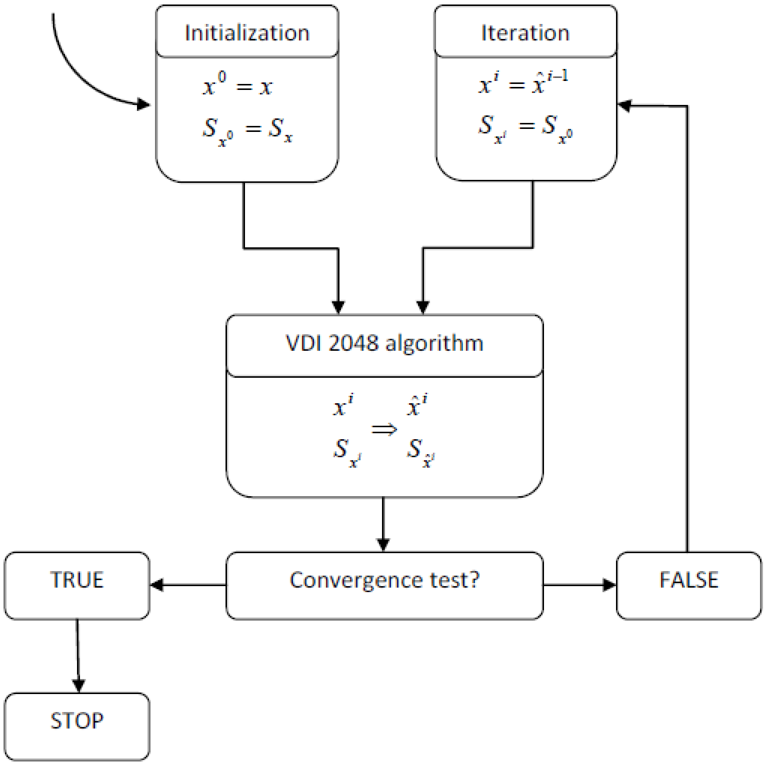

Fig. 8. Convergence test for the data reconciliation procedure

When applying data reconciliation in conformance with the VDI 2048 standard [2], it is assumed that the measured values of the variables of interest correspond to random variables whose distribution around the true values follows Gaussian distribution laws. With this assumption, the vector of contradictions f should also be distributed according to a Gaussian law. In order to check the validity of this hypothesis, it is important to analyze the reconciled values obtained from a statistical point of view.

In practice, this step is split into two tests:

- a first one that checks the correctness of the reconciled values in their entirety;
- a second one that verifies individually each reconciled value so as to detect some suspect errors.

The global test consists in executing a χ^2-test by checking if the following inequality is well satisfied:

$$J^* \leq \chi^2_{r,95\%} \tag{3.11}$$

where $\chi^2_{r,95\%}$ is given by a χ^2-table and r is the number of constraints equations i.e. the cardinality of set *C*.

If this inequality is not satisfied, the initial measured values must be rejected because the contradictions f are too large. In this case, the Gaussian hypothesis is not verified and the framework of the data reconciliation method is exceeded.

In addition, individual tests consist in checking for each measured value whether:

$$\left| \frac{\hat{x}_i - x_i}{s_{v_{i,i}}} \right| \leq \lambda_{95\%} \tag{3.12}$$

where $s_{v_{i,i}}$ is the coefficient of matrix $S_v = S_x - S_{\hat{x}}$ located at the intersection of the i^{th} row and the i^{th} column, and where $\lambda_{95\%} \approx 1.96$.

This test allows to detect which measured values x_i induce too large corrections. In particular, if inequality (3.12) is not satisfied, it means that the associated measured value x_i has to be queried. This test can thus be a good way to detect some fault in the measurement of a specific variable (*e.g.* identification of the bias of a specific sensor) or to point out a serious error in the estimation of the corresponding measurement accuracy or in the model (e.g. in case of leaks that are not taken into account in the model).

3.3 User interface

The user provides as inputs:

1. The Modelica model annotated with (cf. Fig. 9)
   a. The list of variables of interest x;
   b. The list of approximated equations;
2. The measured values of the variables of interest x and the associated covariance matrix S_x.

```
model Pipe1
    Real p;
    Real Q1(uncertain=Uncertainty.refine);
    Real Q2(uncertain=Uncertainty.refine);
equation
    p=2 annotation (__OpenModelica_ApproximatedEquation=true);
    Q1 = Q2;
    Q1 = p;
end Pipe1;
```

Fig. 9. Example of a Modelica model annotated to tag variables of interest and approximated equations

The tool provides as outputs:

1. The sets *C* of constraints equations and *S* of intermediate equations;
2. Notifications of the possible errors in building sets *C* and *S*;
3. If no errors are detected, the reconciled values $\hat{x}$ and the covariance matrix of the reconciled values $S_{\hat{x}}$.
4. The results of the statistical global and local tests

Errors correspond to violations of the following requirements:

1. All variables of interest must be involved in *C* or in *S*: if a variable of interest is not involved in *C*, then it is involved in *S* and reciprocally. A variable of interest can be involved in both sets or in only one of them.

2. *S* must have a square subset that contains all intermediate variables involved in *C*. If a square subset cannot be determined by the tool, then *S* must be square with respect to all intermediate variables involved in *C*.

4 IMPLEMENTATION IN OPENMODELICA

The Data Reconciliation procedure is implemented in OpenModelica in 4 different steps:

1) Extraction Algorithm: automatic extraction of the constraints equations and intermediate equations;
2) Automatic Verification: automatic verification of the result produced by the extraction algorithm;
3) Jacobian Calculation: generation of the runtime code to calculate the Jacobian matrix of the constraint equations;
4) Runtime Code Generation: generation of the runtime code (implementation in C) that computes the reconciled values.

4.1 Extraction Algorithm

The Extraction Algorithm was directly implemented in the OpenModelica Compiler as MetaModelica code. The Extraction Algorithm extracts automatically from the Modelica model the equations that are used by the Data Reconciliation procedure. The Extraction Algorithm basically outputs two sets of equation namely,

- Set-C: denotes the set of constraints equations;
- Set-S: denotes the set of intermediate equation.

The Modelica model with the data reconciliation problem is loaded into the compiler via a scripting interface (.mos file). A typical OpenModelica scripting file is given in Fig. 10).

```
loadModel(Modelica,{"3.1"});
getErrorString();
setCommandLineOptions("--preOptModules+=dataReconciliation");
getErrorString();
loadFile("DataReconciliationSimpleTests/Splitter1.mo");
getErrorString();
buildModel(DataReconciliationSimpleTests.Splitter1);
getErrorString();
```

Fig. 10. Example of a typical OpenModelica scripting file

A special flag "`--preOptModules+=dataReconciliation`" is added to the OpenModelica Compiler (omc) to extract the set of equations for the data reconciliation problem. The flag is set in the scripting interface via the `setCommandLineOptions()` API which was shown above.

The DataReconciliation module implemented in the OpenModelica Compiler will first categorize the variables into two groups:

- The *variables of interest*,
- The *intermediate variables*.

The variables of interest are identified with help of special type attributes defined in the Modelica model (`uncertain=Uncertainty.refine)` which are given by the user. An example of such a declaration is presented in Fig. 11.

```
model Pipe1
    Real p;
    Real Q1(uncertain=Uncertainty.refine);
    Real Q2(uncertain=Uncertainty.refine);
equation
    p=2 annotation (__OpenModelica_ApproximatedEquation=true);
    Q1 = Q2;
    Q1 = p;
end Pipe1;
```

Fig. 11. Example of an annotated Modelica model for data reconciliation

The OpenModelica Compiler parses the variables which are defined as "uncertain" in the category of variables of interest and the remaining variables are grouped into intermediate variables.

There is also a special annotation to tag the approximated equations:

```
annotation(__OpenModelica_ApproximatedEquation = true)
```

This annotation is recognized by the OpenModelica Compiler to make sure that the equations tagged as approximated are not extracted into either Set-C or Set-S.

After categorizing the variables, a BLT matching is performed on the Modelica model and then the Extraction Algorithm explained in sections 1.1 to 1.7 is performed, which results in Set-C and Set-S. An example of Set-C and Set-S is given in Fig. 13 for the Modelica model in Fig. **12**.

```
model Pipe1
  Real p=2;
  Real Q1(uncertain=Uncertainty.refine);
  Real Q2(uncertain=Uncertainty.refine);
equation
  Q1 = Q2;
  Q1 = p;
end Pipe1;
```

Fig. 12. Modelica model of the pipe in Fig. 1

```
SET_C (1)
========================================
1/1 (1): Q1 = Q2   [dynamic |0|0|0|0|]

SET_S (1)
========================================
1/1 (1): Q1 = p   [dynamic |0|0|0|0|]
```

Fig. 13. Set-C and Set-S for the pipe model in Fig. 12

4.2 Automatic Verification

In order to make sure that the Extraction Algorithm is correct and also to make good Error Reporting, Automatic Verification steps are implemented in the OpenModelica Compiler for the DataReconciliation procedure with the following conditions:

1. Set-C and Set-S must not have no equations in common: no equations in Set-C must belong to Set- and reciprocally.
2. All variables of interest are involved either in Set-C or Set-S (or both): if a variable of interest is not involved in Set-C, then it is involved in Set-S and reciprocally.
3. The number of equations in Set-C should be strictly less than the number of variables of interest.
4. Set-S should contain all intermediate variables involved in Set-C.
5. Set-S should be square with respect to all intermediate variables involved in Set-C: Set-S should have a square subset that contains all intermediate variables involved in Set-C.

If the any of above condition fails, then the data reconciliation problem is ill-posed and the error message is given to the users on which Condition above the problem failed. This Verification steps makes sure that the Extraction Algorithm is correct and also makes it easier for the verification of larger models.

An example of how the Extraction Algorithm works and produces outputs in OpenModelica is presented below with a small example.

<u>Example</u>

Modelica model

```
model FlatSimpleExple
    Real q1(uncertain=Uncertainty.refine)=1;
    Real q2(uncertain=Uncertainty.refine)=2;
    Real q3(uncertain=Uncertainty.refine);
    Real q4(uncertain=Uncertainty.refine;
equation
    q1=q2 + q3;
    q4=q2 + q3;
end FlatSimpleExple;
```

Script file: script.mos

```
loadModel(Modelica,{"3.1"});
getErrorString();
setCommandLineOptions("--preOptModules+=dataReconciliation");
getErrorString();
loadFile("DataReconciliationSimpleTests/ FlatSimpleExple.mo");
getErrorString();
buildModel(DataReconciliationSimpleTests. FlatSimpleExple);
getErrorString();
```

Running the mos file with the omc will result in the following output for Extraction Algorithm.

```
>> omc script.mos
```

Output

```
ModelInfo: DataReconciliationSimpleTests.FlatSimpleExple
==========================================================

orderedEquation (4, 4)
========================================
1/1 (1): q1 = 1.0   [binding |0|0|0|0|]
2/2 (1): q2 = 2.0   [binding |0|0|0|0|]
3/3 (1): q1 = q2 + q3   [dynamic |0|0|0|0|]
4/4 (1): q4 = q2 + q3   [dynamic |0|0|0|0|]


orderedVariables (4)
========================================
1: q4:VARIABLE(uncertain=Uncertainty.refine)  type: Real
2: q3:VARIABLE(uncertain=Uncertainty.refine)  type: Real
3: q2:VARIABLE(uncertain=Uncertainty.refine)  type: Real
4: q1:VARIABLE(uncertain=Uncertainty.refine)  type: Real


FINAL SET OF EQUATIONS After Reconciliation
==========================================================================
SET_C: {3, 4}
SET_S: {}


SET_C (2)
========================================
1/1 (1): q1 = q2 + q3   [dynamic |0|0|0|0|]
2/2 (1): q4 = q2 + q3   [dynamic |0|0|0|0|]


SET_S (0)
========================================


Automatic Verification Steps of DataReconciliation Algorithm
==========================================================================

knownVariables:{1, 2, 3, 4} (4)
========================================
1: q4:VARIABLE(uncertain=Uncertainty.refine)  type: Real
2: q3:VARIABLE(uncertain=Uncertainty.refine)  type: Real
3: q2:VARIABLE(uncertain=Uncertainty.refine)  type: Real
4: q1:VARIABLE(uncertain=Uncertainty.refine)  type: Real


ConstantVariables:{3, 4} (2)
========================================
1: q2:VARIABLE(uncertain=Uncertainty.refine)  type: Real
2: q1:VARIABLE(uncertain=Uncertainty.refine)  type: Real
```

```
-SET_C:{3, 4}
-SET_S:{}

Condition-1 "SET_C and SET_S must not have no equations in common"
==========================================================================
-Passed

Condition-2 "All variables of interest must be involved in SET_C or SET_S"
==========================================================================
-Passed

-SET_C has all known variables:{1, 2, 3, 4} (4)
========================================
1: q4:VARIABLE(uncertain=Uncertainty.refine)  type: Real
2: q3:VARIABLE(uncertain=Uncertainty.refine)  type: Real
3: q2:VARIABLE(uncertain=Uncertainty.refine)  type: Real
4: q1:VARIABLE(uncertain=Uncertainty.refine)  type: Real

Condition-3 "SET_C equations must be strictly less than Variable of Interest"
==========================================================================
-Passed
-SET_C contains:2 equations < 4 known variables

Condition-4  "SET_S  should  contain  all  intermediate  variables  involved  in
SET_C"
==========================================================================
-Passed
-SET_C contains No Intermediate Variables
```

4.3 Jacobian Matrix Calculations

After the Extraction Algorithm is performed, The Jacobian matrix is computed for the data reconciliation problem as explained in section 2. The Jacobian matrix F can be computed at run time. To do this OpenModelica generates two new run time functions for Jacobian matrix F namely:

```
int (*initialAnalyticJacobianF) (void* inData, threadData_t *threadData)
int (*functionJacF_column) (void* data, threadData_t *threadData);
```

The newly generated run time functions for matrix F contains the necessary data which computes the values of Jacobian matrix F for the data reconciliation problem.

4.4 Computing the reconciled values $\hat{x}$ and the covariance matrix $S_{\hat{x}}$ of the reconciled values

A new runtime code is being implemented which computes the reconciled values of the reconciled values $\hat{x}$ and their covariance matrix $S_{\hat{x}}$ which are explained in section 3.

The expression to compute the reconciled value is given below:

$$\hat{x} = x - S_x \cdot F^T \cdot f^*$$

The ${}_x$ and S_x are given by the users and the Jacobian matrix F is available. The expression to compute f^* is given below (see section 3 for more details):

$$(F \cdot S_x \cdot F^T) \cdot f^* = f$$

The run time code will accept the ${}_x$ and S_x from the users in the form of csv files which will be used to compute the reconciled values.

REFERENCES

[1] S. Narasimshan and C. Jordache, "Data Reconciliation & Gross Error detection", Gulf Publishing Company, 2000.

[2] VDI 2048 „Kontrolle und Verbesserung der Qualität von Prozessdaten und deren Unsicherheiten mittels Ausgleichsrechnung bei Betriebs- und Abnahmemessungen", ICS 17.020, 27.010.