demetrix status report

Dimitri PISSARENKO
dimitri.pissarenko@gmx.net

9th May 2004, Vienna

Contents

Table of contents **2**

List of tables **2**

List of figures **2**

1 Purpose of the document **3**

2 Future of demetrix **3**
2.1 EIS integration 3
2.2 GUI variety 4
2.2.1 Graph-oriented GUIs 4
2.2.2 Tabular GUIs 6
2.2.3 Console interfaces 6
2.3 Implications 7

3 Material Flow Calculations **7**
3.1 The Tea Example 7
3.2 The Calculation Process 8
3.3 Tea Example in Demetrix 8
3.4 Tea Example Results in Demetrix and Umberto Demo 8

References **8**

A Tea example in detail **10**
A.1 Resources (Places) 10
A.2 Notation 11
A.3 Task formulae 11
A.3.1 $T1$ (water boiling) 11
A.3.2 $T2$ (pouring water on tea) 12
A.4 Link formulae 13
A.4.1 $L1$ ($P1 \rightarrow T1$) 13
A.4.2 $L2$ ($P2 \rightarrow T1$) 13
A.4.3 $L3$ ($T1 \rightarrow P6$) 13
A.4.4 $L4$ ($T1 \rightarrow P3$) 13
A.4.5 $L5$ ($P3 \rightarrow T2$) 13

A.4.6 $L6$ ($T2 \rightarrow P4$) 13
A.4.7 $L7$ ($P5 \rightarrow T2$) 13
A.4.8 $L8$ ($T2 \rightarrow P7$) 13
A.4.9 $L9$ ($P8 \rightarrow T2$) 13

B Listings **14**
B.1 TeaProcessChainGenerator.java 14
B.2 MaterialFlowNetCalcTest.java 26

List of Tables

1 Comparison of flow values calculated by demetrix and Umberto for the production of 1 kg Darjeeling infusion. 9
2 Comparison of flow values calculated by demetrix and Umberto for the production of 2 kg Darjeeling infusion. 9
3 Resources of the tea example 11

List of Figures

1 GUI of the *Umberto 4 Demo* software. This software is available free of charge at ifu Hamburg GmbH (2004). 4
2 GUI of the *EcoScan 3.0 Demo* software. This software is available free of charge at Environmental Expert S.L. (2004). 5
3 BeanShell session. 6
4 Transition specification of $T1$. 11
5 Transition specification of $T2$. 12

1 Purpose of the document

- To explain my current view about the future of demetrix
- To explain how demetrix can be used for performing simple material flow calculations

2 Future of demetrix

After looking at the products of some of demetrix's competitors, I came to the conclusion that demetrix shouldn't be a pure, standalone desktop application. Instead, demetrix will serve the needs of its users better, if it consists of clearly separated server-sided and client-sided components.
This modularity has two advantages:

EIS integration Ability to use demetrix both on desktop as well as a part of an enterprise information system (EIS)

GUI variety Ability to create different GUIs for demetrix

Let's look in detail at these arguments.

2.1 EIS integration

When looking at one of the commercial process modelling systems, (AUDIT Softwareentwicklungs- und Handelsges.m.b.H., 2004), I came to the conclusion that the ability to use demetrix as part of an EIS can be beneficial.
Originally, I planned to develop demetrix as a pure desktop application, which should become an open-source counterpart to commercial process modelling systems. The majority of them is designed as stand-alone applications. Schmidt and Häuslein (1997, p. 4) argue that process modelling software should be designed so that one can use it in standalone mode, in order to reduce the costs of development and thus increase the number of customers. In other words, a simpler (standalone desktop system), thus cheaper system is affordable to a broader range of customers.
I think that nowadays it is possible to develop sophisticated process modelling systems at a relatively little cost. There are two reasons for that:

1. Availability of open source middleware technology (JBoss, Inc. (2004)) and DBMS (hsqldb Development Team (2004), ozone-db.org (2004), MySQL AB (2004))
2. Flexibility of a piece of software is to the largest degree a matter of design quality, rather than that of effort, i. e. cost

Thus, the design of demetrix should take into account the scalability. Demetrix-based systems should grow with their customers, who may start with a simple desktop solution and extend it later to a sophisticated information system.

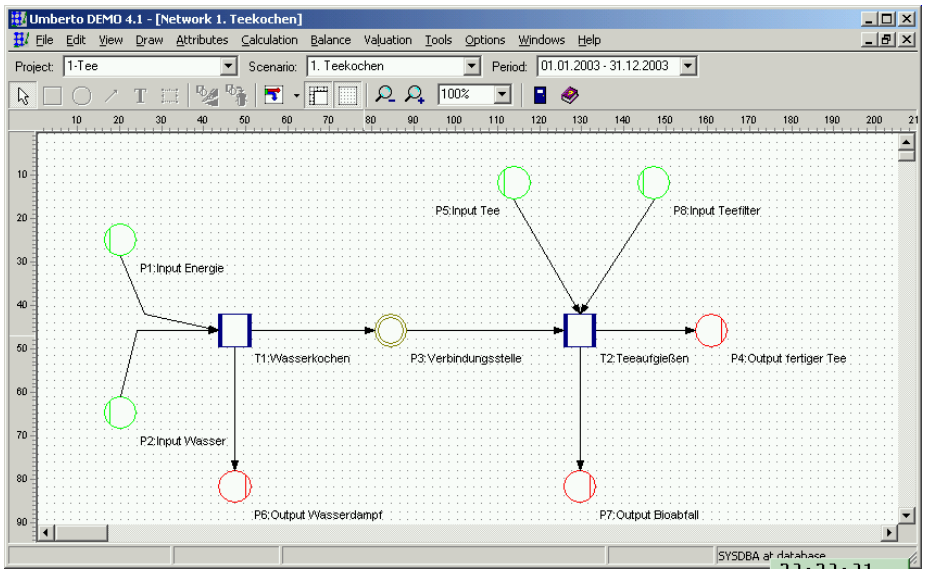

Figure 1: GUI of the *Umberto 4 Demo* software. This software is available free of charge at ifu Hamburg GmbH (2004).

2.2 GUI variety

Another reason for separating between application logic and presentation is the ability to use the same business logic with different presentation layers.
My brief survey of existing commercial process modelling systems led me to the conclusion that the GUIs of most of them can be divided into two groups (they will be discussed below in more detail):

1. Graph-oriented GUIs
2. Tabular GUIs

Let's assume that these two GUI types reflect the consumer's tastes concerning the GUI. Then, demetrix can reach a higher number of customers, if it can be used with different GUIs, each for a particular user group.
Another consideration is creating demetrix GUIs, which closely resemble the GUIs of existing commercial products. In this way, the transition from commercial products to demetrix may be eased[1].

2.2.1 Graph-oriented GUIs

Figure 1 shows an example of a graph-oriented GUI. Here, the user enters the process model by drawing a graph, which consists of processes (tasks), places (resources) and the links between them.

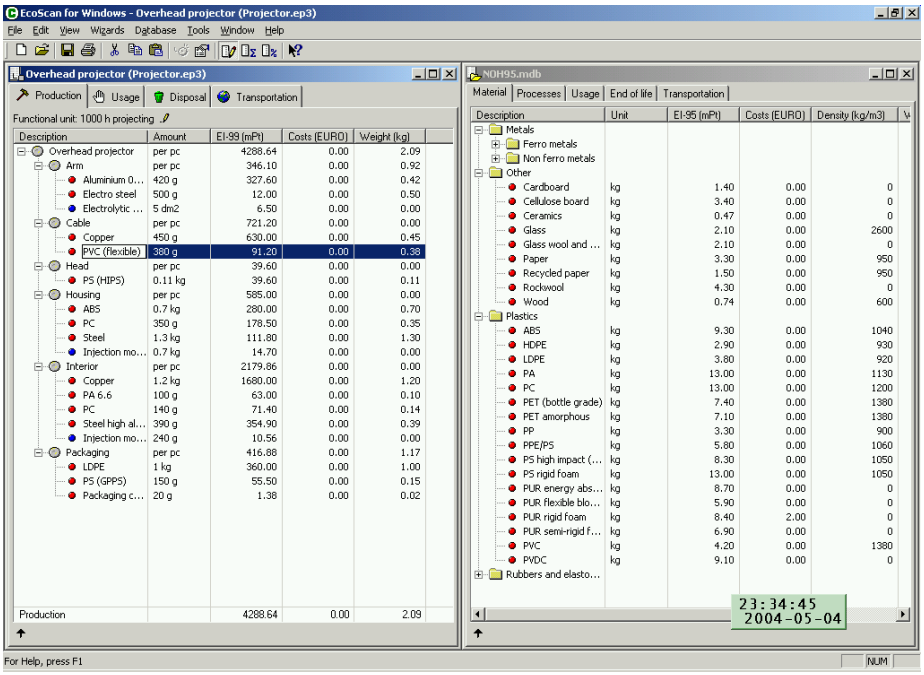

Figure 2: GUI of the *EcoScan 3.0 Demo* software. This software is available free of charge at Environmental Expert S.L. (2004).

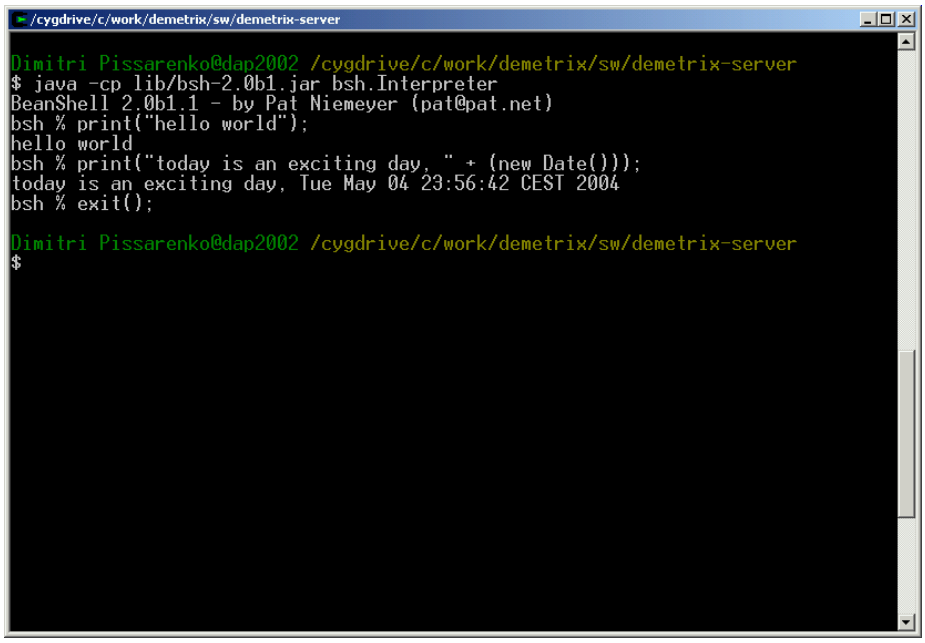

Figure 3: BeanShell session.

2.2.2 Tabular GUIs

Figure 2 shows an example of a tabular GUI. Here, the user assigns a process (from the list on the right hand side) to each resource (on the left hand side).

2.2.3 Console interfaces

Until now, I didn't see a process modelling system with a console interface. By the console interface I mean an interactive user interface without GUI elements. In this case, the interaction with the user happens by means of typing in commands and reading their output. Typical examples of convenient console interfaces include various Linux shells, for instance bash (Bourne Again Shell) as well as user interfaces of several computer algebra systems (Maxima, MATLAB).
Advantages of console interfaces are:

- A console interface may attract users, which know how to program. They can save a lot of time by typing in process chain definitions (in form of a script) instead of drawing them or filling in tables.
- Moreover, some people may prefer working with a script definition of a process chain to a GUI definition of it for reasons of clarity. A script may be more expressive than a drawing or a table.

[1]This strategy (imitating the GUI of a competitor) seems to be used by vendors of some Linux distributions (Xandros (2004)).

- Console interfaces may be more convenient for purposes of searching of errors in large process chain models. Debugging a process chain definition script may be more efficient than struggling through the diagrams.

An easy way to create console interfaces in Java applications is the BeanShell interpreter (Niemeyer, 2004). Figure 3 shows a simple BeanShell session.

2.3 Implications

The separation of demetrix into application logic, data holding and presentation components must proceed while taking into account following considerations:

- In order to be able to incorporate demetrix's application logic components into EIS, they must be published under a suitable licence.
- A useable GUI is an essential factor in demetrix's success. Therefore there should be a powerful open-source client.

This implies that parts (application logic) of demetrix must be released under an EIS-friendly licence (for instance, LGPL). At the same time, it is not sensible to release *all* parts of demetrix under the LGPL.
Presentation (GUI) parts of demetrix must be released under a more strict GPL licence. Behlendorf (1999) states that open source projects rarely succeed in creating powerful GUI. Imagine now that we release a sophisticated, but buggy demetrix GUI under the LGPL. Someone may take the code, debug it and release it under a commercial licence.
If that same buggy demetrix GUI is released under GPL, he can't reuse the existing code in his commercial application. He either has to write that client from scratch, or contribute bugfixes to the publicly available client. If the critical mass of the existing GPL client is attained, rewriting the client from scratch won't be efficient from the economic point of view.
Therefore, I intend to release the application logic (algorithmic) part of demetrix under LGPL and the clients under the GPL.

3 Material Flow Calculations

At the moment, the application logic of demetrix is able to perform one type of calculation - the material flow calculation. The nature of material flow calculation will be explained using the tea example.

3.1 The Tea Example

The tea example is taken from Viere (2003). Its graphical representation is given in figure 1. This model is used to calculate the amounts of energy and materials, which are required to produce a certain amount of Darjeeling infusion.
In this example, the user enters the amount of the end-product, the Darjeeling infusion. All the other values are calculated according to the formulae, which are assigned to tasks (processes) and links. The complete model is presented in section A.
The formulae are assigned to transitions (tasks) and links. Tasks (transitions) transform their inputs into outputs. For instance, task $T1$ transforms electrical

energy and cold water (inputs) into boiling water and water steam (outputs). How much of each input is required to produce a certain amount of the outputs? This question is answered by so-called *transition specifications*. The transition specifications for the tea example are given in section A.3.
The link formulae specify the flow amount, represented by that link. The link formulae for the tea example are given in section A.4.

3.2 The Calculation Process

At the beginning of the calculation process, there is only one flow amount, which is known. This is the flow amount of link $L6$ (Darjeeling infusion). The system investigates each link's flow value. If its flow formula is not calculated, all required formulae are calculated.
Look, for instance, at link $L1$. This link represents the amount of electrical energy, required to produce a certain amount of Darjeeling infusion. As stated in equation 8 (p. 13), the flow amount of $L1$ depends on the flow amount of $L4$. Flow amount of $L4$, in turn, is equal to flow amount of $L5$ (equation 11). Flow amount of $L5$ depends on the flow amount of $L6$ (equation 12). Link $L6$ is the last part of the chain because this value is fixed and needs not to be calculated (equation 13).
This procedure (recursive calculation of all formulae, on which a particular formula depends) is performed on all link flow formulae until either all formulae are properly calculated, or an error occurs.

3.3 Tea Example in Demetrix

The tea model is represented in demetrix as shown in section B.1. This Java class generates a process chain of the tea example with all formulae. The correctness of the calculation is verified using the test cases given in section B.2. Both of them test the calculation procedure using the tea example process chain, but with different amounts of Darjeeling infusion (different flow values of $L6$).

3.4 Tea Example Results in Demetrix and Umberto Demo

As can be seen from tables 1 and 2, the flow amounts calculated by demetrix are quite similar to those calculated by Umberto Demo.

References

AUDIT Softwareentwicklungs- und Handelsges.m.b.H. AUDIT-APCC, 2004. URL `http://www.audit.at/AUDIT-APCC.html`. (URL accessed on May 3rd, 2004).

B. Behlendorf. Open source's position in the spectrum of software, 1999. URL `http://www.huihoo.org/open_source/bb/bb004.html`. (URL accessed on May 5, 2004).

Environmental Expert S.L. Download form, 2004. URL `http://www.environmental-expert.com/software/ecoscan/form.htm`. (URL accessed on May 4, 2004).

Link	Source	Target	Flow (demetrix)	Flow (Umberto)
$L1$	$P1$	$T1$	427.0049	426.1272
$L2$	$P2$	$T1$	1.0761	1.0761
$L3$	$T1$	$P6$	0.0291	0.0290
$L4$	$T1$	$P3$	1.0470	1.0470
$L5$	$P3$	$T2$	1.0470	1.0470
$L6$	$T2$	$P4$	1.0000	1.0000
$L7$	$P5$	$T2$	0.0136	0.0136
$L8$	$T2$	$P7$	0.0506	0.0506
$L9$	$P8$	$T2$	0.0006	0.0006

Table 1: Comparison of flow values calculated by demetrix and Umberto for the production of 1 kg Darjeeling infusion.

Link	Source	Target	Flow (demetrix)	Flow (Umberto)
$L1$	$P1$	$T1$	854.0098	852.2544
$L2$	$P2$	$T1$	2.1522	2.1522
$L3$	$T1$	$P6$	0.0581	0.0581
$L4$	$T1$	$P3$	2.0940	2.0940
$L5$	$P3$	$T2$	2.0940	2.0940
$L6$	$T2$	$P4$	2.0000	2.0000
$L7$	$P5$	$T2$	0.0272	0.0272
$L8$	$T2$	$P7$	0.1012	0.1012
$L9$	$P8$	$T2$	0.0011	0.0011

Table 2: Comparison of flow values calculated by demetrix and Umberto for the production of 2 kg Darjeeling infusion.

hsqldb Development Team. hsqldb: Lightweight 100% Java SQL Database Engine, 2004. URL `http://hsqldb.sourceforge.net/`. (URL accessed on May 4, 2004).

ifu Hamburg GmbH. Interaktive Demo von Umberto 4, 2004. URL `http://www.umberto.de/download/demo.htm`. (URL accessed on May 4, 2004).

JBoss, Inc. JBoss: Overview, 2004. URL `http://www.jboss.org/overview`. (URL accessed on May 4, 2004).

MySQL AB. MySQL: The world's most popular open source database, 2004. URL `http://www.mysql.com/`. (URL accessed on May 4, 2004).

P. Niemeyer. BeanShell: Lightweight Scripting for Java, 2004. URL `http://www.beanshell.org/`. (URL accessed on May 4, 2004).

ozone-db.org. ozone, 2004. URL `http://www.ozone-db.org`. (URL accessed on May 4, 2004).

M. Schmidt and A. Häuslein. *Ökobilanzierung mit Computerunterstützung: Produktbilanzen und betriebliche Bilanzen mit dem Programm Umberto(R).* Springer Verlag, 1997. ISBN 3-540-61177-0.

T. Viere. umberto (R) demo Anleitung, 2003. (This document is distributed together with Umberto demo version, available at ifu Hamburg GmbH (2004)).

Xandros, 2004. URL `http://www.xandros.com/`. (URL accessed on May 4, 2004).

A Tea example in detail

This section contains all the data of the tea example. It is highly recommended to verify these data by downloading and running *Umbertro 4 Demo* (ifu Hamburg GmbH, 2004). The calculation using the data given in this section is tested in scope of the `testCalculateOnTeaExampleFlowAmounts` test case of the class `MaterialFlowNetCalcTest` (see section B.2).

A.1 Resources (Places)

The resources of the tea model are shown in table 3.

Number	Name
P1	Electricity
P2	Water
P3	Boiling water
P4	Darjeeling infusion
P5	Darjeeling tea
P6	Water steam
P7	Biological waste
P8	Paper

Table 3: Resources of the tea example

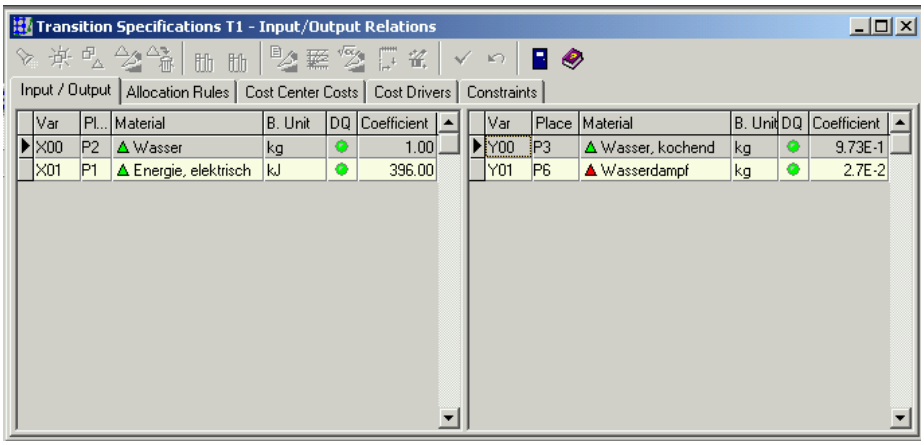

Figure 4: Transition specification of $T1$.

A.2 Notation

$flow(x)$	Flow amount of link x
Lx	Link number x
Px	Place (resource) number x
Tx	Task number x
E_{l,H_2O}	Energy required to boil 1 kg of water
$m_{\text{hot/cold},H_2O}$	Cold water per 1 kg of boiling water
m_{steam}	Steam per 1 kg of boiling water
m_{hot,H_2O}	Boiling water per 1 kg of Darjeeling infusion
m_{tea}	Tea per 1 kg of Darjeeling infusion
m_{waste}	Biological waste per 1 kg of Darjeeling infusion
m_{paper}	Paper per 1 kg of Darjeeling infusion

A.3 Task formulae

A.3.1 $T1$ (water boiling)

The transition specification for $T1$ is shown in figure 4. It should be interpreted in the following way.
$T1$ uses 396 kJ of electrical energy to produce 0.971 kg of boiling water. This

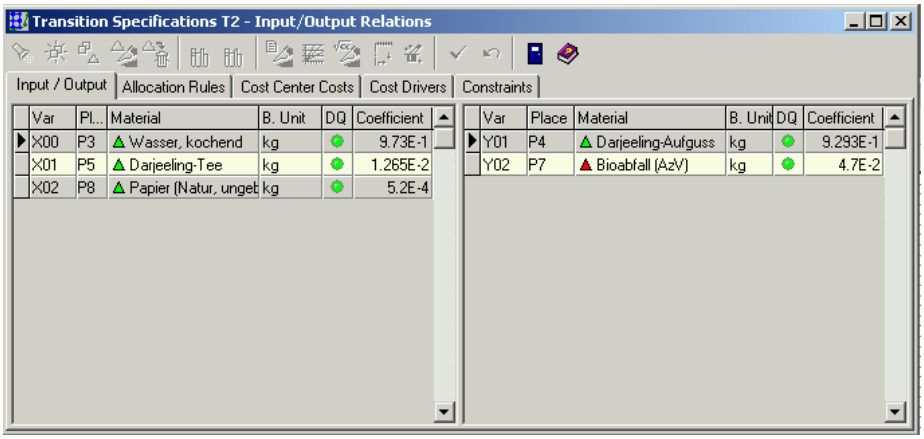

Figure 5: Transition specification of $T2$.

yields the formula

$$E_{l,H_2O} = \frac{396}{0.971}. \tag{1}$$

$T1$ transforms 1 kg of cold water into 0.971 kg of boiling water. Hence,

$$m_{hot/cold,H_2O} = \frac{1}{0.971} \tag{2}$$

In the production of 0.971 kg of boiling water, 0.027 kg of steam are generated. Hence, when producing 1 kg of boiling water, the amount of steam is equal to m_{steam}.

$$m_{\text{steam}} = \frac{0.027}{0.971} \tag{3}$$

A.3.2 $T2$ (pouring water on tea)

The transition specification for $T2$ is shown in figure 5. It should be interpreted in the following way.
In order to produce 0.9293 kg of Darjeeling infusion one needs 0.973 kg of boiling water. In order to produce 1 kg of Darjeeling infusion, one needs

$$m_{\text{hot},H_2O} = \frac{0.973}{0.9293} \tag{4}$$

kg of boiling water.
In order to produce 0.9293 kg of Darjeeling infusion one needs 0.01265 kg of tea. In order to produce 1 kg of Darjeeling infusion, one needs

$$m_{\text{tea}} = \frac{0.01265}{0.9293} \tag{5}$$

kg of tea.
When producing 0.9293 kg of Darjeeling infusion, 0.047 kg of biologial waste

are created as a by-product. When 1 kg of Darjeeling infusion is produced, the amount of biological waste is equal to

$$m_{\text{waste}} = \frac{0.047}{0.9293}. \tag{6}$$

In order to produce 0.9293 kg of Darjeeling infusion, 0.00052 kg of paper (teabag) is required. In order to produce 1 kg of Darjeeling infusion, one needs

$$m_{\text{paper}} = \frac{0.00052}{0.9293} \tag{7}$$

kg of paper.

A.4 Link formulae

Each link has only one property, which is either calculated or fixed. This is the flow amount, represented by that link.

A.4.1 $L1$ $(P1 \rightarrow T1)$

$$flow(L1) = E_{l,H_2O} \cdot flow(L4) \tag{8}$$

A.4.2 $L2$ $(P2 \rightarrow T1)$

$$flow(L2) = m_{hot/cold,H_2O} \cdot flow(L4) \tag{9}$$

A.4.3 $L3$ $(T1 \rightarrow P6)$

$$flow(L3) = m_{\text{steam}} \cdot flow(L4) \tag{10}$$

A.4.4 $L4$ $(T1 \rightarrow P3)$

$$flow(L4) = flow(L5) \tag{11}$$

A.4.5 $L5$ $(P3 \rightarrow T2)$

$$flow(L5) = m_{\text{hot},H_2O} \cdot flow(L6) \tag{12}$$

A.4.6 $L6$ $(T2 \rightarrow P4)$

$$flow(L6) = 1 \tag{13}$$

A.4.7 $L7$ $(P5 \rightarrow T2)$

$$flow(L7) = m_{\text{tea}} \cdot flow(L6) \tag{14}$$

A.4.8 $L8$ $(T2 \rightarrow P7)$

$$flow(L8) = m_{\text{waste}} \cdot flow(L6) \tag{15}$$

A.4.9 $L9$ $(P8 \rightarrow T2)$

$$flow(L9) = m_{\text{paper}} \cdot flow(L6) \tag{16}$$

B Listings

In this section the listings of two Java classes are given. In section B.1 you will find the definition of the tea model in demetrix. In section B.2 you will find the test case, which verifies the calculation of flow amounts using the tea example. Please be aware that the code presented here may be outdated by the time you read this document. Most recent code can be found at `http://sourceforge.net/project/showfiles.php?group_id=97569`.

B.1 TeaProcessChainGenerator.java

```
1  /*****************************************************************************
2   * Demetrix process modelling system
3   *
4   * Copyright (c) 2003, 2004 Dimitri A. Pissarenko
5   *
6   * This file is part of Demetrix.
7   *
8   * Demetrix is free software; you can redistribute it and/or
        modify
9   * it under the terms of the GNU Lesser General Public License as
         published by
10  * the Free Software Foundation; either version 2.1 of the
        License, or
11  * (at your option) any later version.
12  *
13  * Demetrix is distributed in the hope that it will be useful,
14  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16  * GNU Lesser General Public License for more details.
17  *
18  * You should have received a copy of the GNU Lesser General
        Public License
19  * along with Demetrix; if not, write to the Free Software
20  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
        02111-1307 USA
21  *
22  * For further information you may
23  *
24  * - send an e-mail in Russian, German or English to dimitri.
        pissarenko@gmx.net
25  * - look at http://sourceforge.net/projects/demetrix/
26  * - look at http://demetrix.sourceforge.net/
27  * - look at http://members.inode.at/d.pissarenko/
28  *
29  *****************************************************************************/

30
31  /*
```

```
32  * Created on 15.04.2004
33  */
34 package net.sourceforge.demetrix.server.test.materialFlowNetCalc;
35
36 import net.sourceforge.demetrix.server.materialFlowNetCalc.
       MaterialFlowNetCalc;
37 import net.sourceforge.demetrix.server.model.Link;
38 import net.sourceforge.demetrix.server.model.ProcessChain;
39 import net.sourceforge.demetrix.server.model.Resource;
40 import net.sourceforge.demetrix.server.model.Task;
41 import net.sourceforge.demetrix.server.properties.DoubleProperty;
42 import net.sourceforge.demetrix.server.properties.FormulaProperty
       ;
43 import net.sourceforge.demetrix.server.properties.
       FormulaPropertyParameter;
44
45 /**
46  * This is a convenience class, which generates the process chain
        from the
47  * tea example of the Umberto Demo manual.
48  *
49  * @see File Umberto_Demo_Anleitung.PDF in directory doc/demetrix
       -server
50  * @see File teaExampleMaterialFlowNetwork.png in directory doc/
       demetrix-server
51  * @author Dimitri Pissarenko
52  */
53 public class TeaProcessChainGenerator {
54         private static final String
              BIOLOGICAL_WASTE_PER_KILOGRAM_OF_DARJEELING_INFUSION =
55                 "biologicalWastePerKilogramOfDarjeelingInfusion";
56         private static final String BOILING_WATER_PARAMETER_NAME
               = "boilingWater";
57         private static final String
              BOILING_WATER_PER_KILOGRAM_OF_DARJEELING_INFUSION =
58                 "boilingWaterPerLiterOfDarjeelingInfusion";
59         private static final String
              COLD_WATER_PER_LITER_OF_BOILING_WATER =
60                 "coldWaterPerLiterOfBoilingWater";
61         private static final String DARJEELING_INFUSION = "
              darjeelingInfusion";
62         private static final String ENERGY_PER_LITER_PROPERTY_NAME
               =
63                 "energyPerLiter";
64         private static final String LINK5_FLOW = "link5Flow";
65         private static final String LINK6_FLOW = "link6Flow";
66         public static final String P1_NAME = "P1 (electricity)";
67         public static final String P2_NAME = "P2 (water)";
68         public static final String P3_NAME =
69                 "P3 (boiling water, connection place)";
```

```
70          public static final String P4_NAME = "P4 (Darjeeling
                infusion)";
71          public static final String P5_NAME = "P5 (Darjeeling tea)
                ";
72          public static final String P6_NAME = "P6 (water steam)";
73          public static final String P7_NAME = "P7 (biological waste
                )";
74          public static final String P8_NAME = "P8 (paper)";
75          private static final String
                PAPER_PER_KILOGRAM_OF_DARJEELING_INFUSION = "
                paperPerKilogramOfDarjeelingInfusion";
76          private static final String
                STEAM_PER_LITER_OF_BOILING_WATER =
77                  "steamPerLiterOfBoilingWater";
78          public static final String T1_NAME = "T1 (water boiling)";
79          public static final String T2_NAME = "T2 (pouring water on
                 tea)";
80          private static final String
                TEA_PER_KILOGRAM_OF_DARJEELING_INFUSION =
81                  "teaPerLiterOfDarjeelingInfusion";
82          /**
83           * @return Process chain of the tea example
84           */
85          public static ProcessChain createProcessChain() {
86
87                  /**
88                   * declare variables
89                   */
90                  Resource place1 = null; /* electricity */
91                  Resource place2 = null; /* water */
92                  Resource place3 = null; /* boiling water (
                        connection place) */
93                  Resource place4 = null; /* Darjeeling infusion */
94                  Resource place5 = null; /* Darjeeling tea */
95                  Resource place6 = null; /* water steam */
96                  Resource place7 = null; /* biological waste */
97                  Resource place8 = null; /* paper */
98
99                  Task task1 = null; /* water boiling */
100                 Task task2 = null; /* pouring water on tea */
101
102                 Link link1 = null; /* place1 -> task1 */
103                 Link link2 = null; /* place2 -> task1 */
104                 Link link3 = null; /* task1 -> place6 */
105                 Link link4 = null; /* task1 -> place3 */
106                 Link link5 = null; /* place3 -> task2 */
107                 Link link6 = null; /* task2 -> place4 */
108                 Link link7 = null; /* place5 -> task2 */
109                 Link link8 = null; /* task2 -> place7 */
110                 Link link9 = null; /* place8 -> task2 */
```

```
111
112             ProcessChain processChain = null;
113
114             /**
115              * init resources
116              **/
117             place1 = new Resource();
118             place1.setName(P1_NAME);
119
120             place2 = new Resource();
121             place2.setName(P2_NAME);
122
123             place3 = new Resource();
124             place3.setName(P3_NAME);
125
126             place4 = new Resource();
127             place4.setName(P4_NAME);
128
129             place5 = new Resource();
130             place5.setName(P5_NAME);
131
132             place6 = new Resource();
133             place6.setName(P6_NAME);
134
135             place7 = new Resource();
136             place7.setName(P7_NAME);
137
138             place8 = new Resource();
139             place8.setName(P8_NAME);
140
141             /**
142              * init tasks
143              */
144             task1 = new Task();
145             task1.setName(T1_NAME);
146
147             task2 = new Task();
148             task2.setName(T2_NAME);
149
150
151             /**
152              * init process chain
153              * add tasks and resources to the process chain
154              */
155             processChain = new ProcessChain();
156             processChain.addResource(place1);
157             processChain.addResource(place2);
158             processChain.addResource(place3);
159             processChain.addResource(place4);
160             processChain.addResource(place5);
```

```
161             processChain.addResource(place6);
162             processChain.addResource(place7);
163             processChain.addResource(place8);
164 
165             processChain.addTask(task1);
166             processChain.addTask(task2);
167 
168             /**
169              * init links
170              */
171             link1 = new Link(place1, task1);
172             link2 = new Link(place2, task1);
173             link3 = new Link(task1, place6);
174             link4 = new Link(task1, place3);
175             link5 = new Link(place3, task2);
176             link6 = new Link(task2, place4);
177             link7 = new Link(place5, task2);
178             link8 = new Link(task2, place7);
179             link9 = new Link(place8, task2);
180 
181             /**
182              * add links to process chain
183              */
184             processChain.addLink(link1);
185             processChain.addLink(link2);
186             processChain.addLink(link3);
187             processChain.addLink(link4);
188             processChain.addLink(link5);
189             processChain.addLink(link6);
190             processChain.addLink(link7);
191             processChain.addLink(link8);
192             processChain.addLink(link9);
193 
194             /**
195              * specify the transitions
196              */
197             specifyTransitions(processChain);
198 
199             return processChain;
200         }
201         /**
202          * This method initializes the tasks of the tea process
                chain according
203          * to the transition specifications of the Umberto example
                .
204          */
205         private static void specifyTransitions(ProcessChain
            processChain) {
206             Task task1 = null;
207             Task task2 = null;
```

```
208             Resource place1 = null;
209             Resource place2 = null;
210             Resource place3 = null;
211             Resource place4 = null;
212             Resource place5 = null;
213             Resource place6 = null;
214             Resource place7 = null;
215             Resource place8 = null;
216             Link link1 = null;
217             Link link2 = null;
218             Link link3 = null;
219             Link link4 = null;
220             Link link5 = null;
221             Link link6 = null;
222             Link link7 = null;
223             Link link8 = null;
224             Link link9 = null;
225
226             DoubleProperty link1DoubleProperty = null;
227             DoubleProperty energyPerLiterProperty = null;
228             DoubleProperty coldWaterPerLiterOfBoilingWater =
                    null;
229             DoubleProperty steamPerLiterOfBoilingWater = null;
230             DoubleProperty
                    boilingWaterPerLiterOfDarjeelingInfusion = null
                    ;
231             DoubleProperty darjeelingInfusion = null;
232             DoubleProperty teaPerLiterOfDarjeelingInfusion =
                    null;
233             DoubleProperty
                    biologicalWastePerKilogramOfDarjeelingInfusion
                     = null;
234             DoubleProperty paperPerKilogramOfDarjeelingInfusion
                    =null;
235             FormulaProperty formulaProperty = null;
236
237             /**
238              * Fetch all necessary tasks and resources
239              */
240             task1 = (Task) processChain.getNodeByName(T1_NAME);
241             task2 = (Task) processChain.getNodeByName(T2_NAME);
242             place1 = (Resource) processChain.getNodeByName(
                    P1_NAME);
243             place2 = (Resource) processChain.getNodeByName(
                    P2_NAME);
244             place3 = (Resource) processChain.getNodeByName(
                    P3_NAME);
245             place4 = (Resource) processChain.getNodeByName(
                    P4_NAME);
```

```
246             place5 = (Resource) processChain.getNodeByName(
                    P5_NAME);
247             place6 = (Resource) processChain.getNodeByName(
                    P6_NAME);
248             place7 = (Resource) processChain.getNodeByName(
                    P7_NAME);
249             place8 = (Resource) processChain.getNodeByName(
                    P8_NAME);
250
251             /**
252              * Fetch links
253              */
254             link1 = (Link) processChain.getEdge(place1, task1);
255             link2 = (Link) processChain.getEdge(place2, task1);
256             link3 = (Link) processChain.getEdge(task1, place6);
257             link4 = (Link) processChain.getEdge(task1, place3);
258             link5 = (Link) processChain.getEdge(place3, task2);
259             link6 = (Link) processChain.getEdge(task2, place4);
260             link7 = (Link) processChain.getEdge(place5, task2);
261             link8 = (Link) processChain.getEdge(task2, place7);
262             link9 = (Link) processChain.getEdge(place8, task2);
263
264             /**
265              * remove all tasks, resources and links from the
                    process chain
266              */
267             processChain.removeTaskOrResource(task1);
268             processChain.removeTaskOrResource(task2);
269             processChain.removeTaskOrResource(place1);
270             processChain.removeTaskOrResource(place2);
271             processChain.removeTaskOrResource(place3);
272             processChain.removeTaskOrResource(place4);
273             processChain.removeTaskOrResource(place5);
274             processChain.removeTaskOrResource(place6);
275             processChain.removeTaskOrResource(place7);
276             processChain.removeTaskOrResource(place8);
277             processChain.removeLink(link1);
278             processChain.removeLink(link2);
279             processChain.removeLink(link3);
280             processChain.removeLink(link4);
281             processChain.removeLink(link5);
282             processChain.removeLink(link6);
283             processChain.removeLink(link7);
284             processChain.removeLink(link8);
285             processChain.removeLink(link9);
286
287
288             /**
289              * specify flow amount of link6 (T2 -> P4)
290              */
```

```
291             darjeelingInfusion = new DoubleProperty();
292             darjeelingInfusion.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
293             darjeelingInfusion.setValue(new Double(1.));
294             link6.addProperty(darjeelingInfusion);
295
296
297
298             /**
299              * specify the flow amount of link5 (P3 -> T2)
300              */
301             boilingWaterPerLiterOfDarjeelingInfusion = new
                    DoubleProperty();
302             boilingWaterPerLiterOfDarjeelingInfusion.setName(
303                    BOILING_WATER_PER_KILOGRAM_OF_DARJEELING_INFUSION
                           );
304             boilingWaterPerLiterOfDarjeelingInfusion.setValue(
305                    new Double(0.973 / 0.9293));
306
307             formulaProperty = new FormulaProperty();
308             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
309             formulaProperty.setFormula(
310                    BOILING_WATER_PER_KILOGRAM_OF_DARJEELING_INFUSION

311                            + "*"
312                            + LINK6_FLOW);
313             formulaProperty.addParameter(
314                    BOILING_WATER_PER_KILOGRAM_OF_DARJEELING_INFUSION
                           ,
315                    boilingWaterPerLiterOfDarjeelingInfusion);
316             formulaProperty.addParameter(LINK6_FLOW, (
                    FormulaPropertyParameter) link6.getProperty(
317             MaterialFlowNetCalc.FLOW_PROPERTY_NAME));
318
319             link5.addProperty(formulaProperty);
320             task2.addProperty(
                    boilingWaterPerLiterOfDarjeelingInfusion);
321
322
323             /**
324              * specify the flow amount of link4 (T1 -> P3)
325              *
326              * The flow amount of this link is the same as that
                        of link5.
327              */
328             formulaProperty = new FormulaProperty();
329             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
330             formulaProperty.setFormula(LINK5_FLOW);
```

```
331             formulaProperty.addParameter(
332                     LINK5_FLOW,
333                     (FormulaPropertyParameter) link5.getProperty
                            (
334                             MaterialFlowNetCalc.
                                    FLOW_PROPERTY_NAME));
335             link4.addProperty(formulaProperty);
336
337             /**
338              * specify the flow amount of link1 (P1 -> T1)
339              */
340             energyPerLiterProperty = new DoubleProperty();
341             energyPerLiterProperty.setName(
                    ENERGY_PER_LITER_PROPERTY_NAME);
342             energyPerLiterProperty.setValue(new Double
                    (396 / 0.971));
343
344             formulaProperty = new FormulaProperty();
345             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
346             formulaProperty.setFormula(
347                     ENERGY_PER_LITER_PROPERTY_NAME
348                             + " * "
349                             + BOILING_WATER_PARAMETER_NAME);
350
351             formulaProperty.addParameter(
352                     ENERGY_PER_LITER_PROPERTY_NAME,
353                     energyPerLiterProperty);
354             formulaProperty.addParameter(
355                     BOILING_WATER_PARAMETER_NAME,
356                     (FormulaPropertyParameter) link4.getProperty
                            (
357                             MaterialFlowNetCalc.
                                    FLOW_PROPERTY_NAME));
358             link1.addProperty(formulaProperty);
359             task1.addProperty(energyPerLiterProperty);
360
361             /**
362              * specify the flow amount of link2 (P2 -> T1)
363              */
364             coldWaterPerLiterOfBoilingWater = new
                    DoubleProperty();
365             coldWaterPerLiterOfBoilingWater.setName(
366                     COLD_WATER_PER_LITER_OF_BOILING_WATER);
367             coldWaterPerLiterOfBoilingWater.setValue(new Double
                    (1. / 0.973));
368             formulaProperty = new FormulaProperty();
369             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
370             formulaProperty.setFormula(
```

```
371                 COLD_WATER_PER_LITER_OF_BOILING_WATER
372                         + "*"
373                         + BOILING_WATER_PARAMETER_NAME);
374         formulaProperty.addParameter(
375                 COLD_WATER_PER_LITER_OF_BOILING_WATER,
376                 coldWaterPerLiterOfBoilingWater);
377         formulaProperty.addParameter(
378                 BOILING_WATER_PARAMETER_NAME,
379                 (FormulaPropertyParameter) link4.getProperty
                        (
380                         MaterialFlowNetCalc.
                            FLOW_PROPERTY_NAME));
381         link2.addProperty(formulaProperty);
382         task1.addProperty(coldWaterPerLiterOfBoilingWater);
383
384         /**
385          * specify the flow amount of link3 (T1 -> P6)
386          */
387         steamPerLiterOfBoilingWater = new DoubleProperty();
388         steamPerLiterOfBoilingWater.setName(
                STEAM_PER_LITER_OF_BOILING_WATER);
389         steamPerLiterOfBoilingWater.setValue(new Double
                (0.027 / 0.973));
390         formulaProperty = new FormulaProperty();
391         formulaProperty.setName(MaterialFlowNetCalc.
                FLOW_PROPERTY_NAME);
392         formulaProperty.setFormula(
393                 STEAM_PER_LITER_OF_BOILING_WATER
394                         + "*"
395                         + BOILING_WATER_PARAMETER_NAME);
396         formulaProperty.addParameter(
397                 STEAM_PER_LITER_OF_BOILING_WATER,
398                 steamPerLiterOfBoilingWater);
399         formulaProperty.addParameter(
400                 BOILING_WATER_PARAMETER_NAME,
401                 (FormulaPropertyParameter) link4.getProperty
                        (
402                         MaterialFlowNetCalc.
                            FLOW_PROPERTY_NAME));
403         link3.addProperty(formulaProperty);
404         task1.addProperty(steamPerLiterOfBoilingWater);
405
406
407
408         /**
409          * specify flow amount of link7 (P5 -> T2)
410          */
411         teaPerLiterOfDarjeelingInfusion = new
                DoubleProperty();
412         teaPerLiterOfDarjeelingInfusion.setName(
```

```
413                 TEA_PER_KILOGRAM_OF_DARJEELING_INFUSION);
414             teaPerLiterOfDarjeelingInfusion.setValue(new Double
                    (0.01265 / 0.9293));
415
416             formulaProperty = new FormulaProperty();
417             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
418             formulaProperty.setFormula(
419                 TEA_PER_KILOGRAM_OF_DARJEELING_INFUSION
420                         + "*"
421                         + DARJEELING_INFUSION);
422             formulaProperty.addParameter(
423                 TEA_PER_KILOGRAM_OF_DARJEELING_INFUSION,
424                 teaPerLiterOfDarjeelingInfusion);
425             formulaProperty.addParameter(
426                 DARJEELING_INFUSION,
427                 (FormulaPropertyParameter) link6.getProperty
                        (
428                         MaterialFlowNetCalc.
                                FLOW_PROPERTY_NAME));
429
430             link7.addProperty(formulaProperty);
431             task2.addProperty(teaPerLiterOfDarjeelingInfusion);
432
433             /**
434              * specify flow amount of link8 (T2 -> P7)
435              */
436             biologicalWastePerKilogramOfDarjeelingInfusion =
                    new DoubleProperty();
437             biologicalWastePerKilogramOfDarjeelingInfusion.
                    setName(
438                 BIOLOGICAL_WASTE_PER_KILOGRAM_OF_DARJEELING_INFUSION
                        );
439             biologicalWastePerKilogramOfDarjeelingInfusion.
                    setValue(
440                 new Double(0.047 / 0.9293));
441
442
443
444             formulaProperty = new FormulaProperty();
445             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
446             formulaProperty.setFormula(
447                 BIOLOGICAL_WASTE_PER_KILOGRAM_OF_DARJEELING_INFUSION

448                         + "*"
449                         + DARJEELING_INFUSION);
450             formulaProperty.addParameter(BIOLOGICAL_WASTE_PER_KILOGRAM_OF_DARJEELING_INF
                    , biologicalWastePerKilogramOfDarjeelingInfusion
                    );
```

```
451             formulaProperty.addParameter(DARJEELING_INFUSION, (
                    FormulaPropertyParameter)link6.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME));
452             link8.addProperty(formulaProperty);
453             task2.addProperty(
                    biologicalWastePerKilogramOfDarjeelingInfusion)
                    ;
454
455             /**
456              * specify flow amount of link9 (P8 -> T2)
457              */
458             paperPerKilogramOfDarjeelingInfusion=new
                    DoubleProperty();
459             paperPerKilogramOfDarjeelingInfusion.setName(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
460             paperPerKilogramOfDarjeelingInfusion.setValue(new
                    Double(0.00052/0.9293));
461
462             formulaProperty=new FormulaProperty();
463             formulaProperty.setName(MaterialFlowNetCalc.
                    FLOW_PROPERTY_NAME);
464             formulaProperty.setFormula(
                    PAPER_PER_KILOGRAM_OF_DARJEELING_INFUSION
                     + "*" + DARJEELING_INFUSION);
465             formulaProperty.addParameter(
                    PAPER_PER_KILOGRAM_OF_DARJEELING_INFUSION,
                    paperPerKilogramOfDarjeelingInfusion);
466             formulaProperty.addParameter(DARJEELING_INFUSION, (
                    FormulaPropertyParameter)link6.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME));
467
468             link9.addProperty(formulaProperty);
469             task2.addProperty(
                    paperPerKilogramOfDarjeelingInfusion);
470
471             /**
472              * add all changed objects to the process chain
473              */
474             processChain.addTask(task1);
475             processChain.addTask(task2);
476             processChain.addResource(place1);
477             processChain.addResource(place2);
478             processChain.addResource(place3);
479             processChain.addResource(place4);
480             processChain.addResource(place5);
481             processChain.addResource(place6);
482             processChain.addResource(place7);
483             processChain.addResource(place8);
484             processChain.addLink(link1);
485             processChain.addLink(link2);
```

```
486                 processChain.addLink(link3);
487                 processChain.addLink(link4);
488                 processChain.addLink(link5);
489                 processChain.addLink(link6);
490                 processChain.addLink(link7);
491                 processChain.addLink(link8);
492                 processChain.addLink(link9);
493
494         }
495 }
```

B.2 MaterialFlowNetCalcTest.java

```
1  /*****************************************************************************
2   * Demetrix process modelling system
3   *
4   * Copyright (c) 2003, 2004 Dimitri A. Pissarenko
5   *
6   * This file is part of Demetrix.
7   *
8   * Demetrix is free software; you can redistribute it and/or
        modify
9   * it under the terms of the GNU Lesser General Public License as
         published by
10  * the Free Software Foundation; either version 2.1 of the
        License, or
11  * (at your option) any later version.
12  *
13  * Demetrix is distributed in the hope that it will be useful,
14  * but WITHOUT ANY WARRANTY; without even the implied warranty of
15  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
16  * GNU Lesser General Public License for more details.
17  *
18  * You should have received a copy of the GNU Lesser General
        Public License
19  * along with Demetrix; if not, write to the Free Software
20  * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
        02111-1307 USA
21  *
22  * For further information you may
23  *
24  * - send an e-mail in Russian, German or English to dimitri.
        pissarenko@gmx.net
25  * - look at http://sourceforge.net/projects/demetrix/
26  * - look at http://demetrix.sourceforge.net/
27  * - look at http://members.inode.at/d.pissarenko/
28  *
29  *****************************************************************************/
```

```
30
31 /*
32  * Created on 15.04.2004
33  */
34 package net.sourceforge.demetrix.server.test.materialFlowNetCalc;
35
36 import java.util.Iterator;
37
38 import org.apache.log4j.Logger;
39
40 import net.sourceforge.demetrix.server.materialFlowNetCalc.
       MaterialFlowNetCalc;
41 import net.sourceforge.demetrix.server.model.Link;
42 import net.sourceforge.demetrix.server.model.ProcessChain;
43 import net.sourceforge.demetrix.server.model.Resource;
44 import net.sourceforge.demetrix.server.model.Task;
45 import net.sourceforge.demetrix.server.properties.DoubleProperty;
46 import net.sourceforge.demetrix.server.properties.FormulaProperty
       ;
47 import net.sourceforge.demetrix.server.util.LoggerSetupHelper;
48 import junit.framework.TestCase;
49
50 /**
51  * Test class for the class MaterialFlowNetCalc
52  *
53  * @see net.sourceforge.demetrix.server.materialFlowNetCalc.
        MaterialFlowNetCalc
54  * @author Dimitri Pissarenko
55  */
56 public class MaterialFlowNetCalcTest extends TestCase {
57         public final static double JUNIT_DOUBLE_PRECISION=2.0;
58
59         private Logger logger=Logger.getLogger(getClass());
60         /**
61          * Constructor for MaterialFlowNetCalcTest.
62          * @param arg0
63          */
64         public MaterialFlowNetCalcTest(String arg0) {
65                 super(arg0);
66                 LoggerSetupHelper.setupLogger();
67         }
68
69         public static void main(String[] args) {
70                 junit.swingui.TestRunner.run(
                        MaterialFlowNetCalcTest.class);
71         }
72         /**
73          * This method tests the MaterialFlowNetCalc.
              calculateFlowAmounts
```

```
74          * method using the tea example from Umberto Demo program
                . The
75          * values of the flow amounts were taken directly from
                Umberto
76          * program.
77          *
78          * @see File Umberto_Demo_Anleitung.PDF in directory doc/
                demetrix-server
79          * @see File teaExampleMaterialFlowNetwork.png in
                directory doc/demetrix-server
80          */
81         public void testCalculateOnTeaExampleFlowAmounts()
82         {
83                 MaterialFlowNetCalc calculator=null;
84                 Task task1=null;
85                 Task task2=null;
86                 Resource place1=null;
87                 Resource place2=null;
88                 Resource place3=null;
89                 Resource place4=null;
90                 Resource place5=null;
91                 Resource place6=null;
92                 Resource place7=null;
93                 Resource place8=null;
94                 Link link1 = null;
95                 Link link2 = null;
96                 Link link3 = null;
97                 Link link4 = null;
98                 Link link5 = null;
99                 Link link6 = null;
100                Link link7 = null;
101                Link link8 = null;
102                Link link9 = null;
103                FormulaProperty formulaProperty=null;
104                DoubleProperty doubleProperty=null;
105                ProcessChain processChain=null;
106
107                /**
108                 * create tea process chain
109                 */
110                processChain=TeaProcessChainGenerator.
                       createProcessChain();
111
112                /**
113                 * perform calculation
114                 */
115                calculator=new MaterialFlowNetCalc();
116                calculator.calculateFlowAmounts(processChain);
117
118                this.logger.debug("**************************");
```

```
119             this.logger.debug("** FAILED FORMULAE START **");
120             this.logger.debug("**************************");
121
122             Iterator iterator=null;
123             iterator=calculator.getFailedFormulae().iterator();
124
125             while (iterator.hasNext())
126             {
127                    FormulaProperty formula=null;
128                    formula=(FormulaProperty)iterator.next();
129                    this.logger.debug("*****");
130                    this.logger.debug("failed formula: " +
                           formula.getFormula());
131                    this.logger.debug("why failed: " + formula.
                           getFailureReason());
132             }
133             this.logger.debug("************************");
134             this.logger.debug("** FAILED FORMULAE END **");
135             this.logger.debug("************************");
136
137             /**
138              * Fetch all necessary tasks and resources
139              */
140             task1 = (Task) processChain.getNodeByName(
                    TeaProcessChainGenerator.T1_NAME);
141             task2 = (Task) processChain.getNodeByName(
                    TeaProcessChainGenerator.T2_NAME);
142             place1 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P1_NAME);
143             place2 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P2_NAME);
144             place3 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P3_NAME);
145             place4 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P4_NAME);
146             place5 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P5_NAME);
147             place6 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P6_NAME);
148             place7 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P7_NAME);
149             place8 = (Resource)processChain.getNodeByName(
                    TeaProcessChainGenerator.P8_NAME);
150
151
152             link1 = (Link) processChain.getEdge(place1, task1);
153             link2 = (Link) processChain.getEdge(place2, task1);
154             link3 = (Link) processChain.getEdge(task1, place6);
155             link4 = (Link) processChain.getEdge(task1, place3);
156             link5 = (Link) processChain.getEdge(place3, task2);
```

```
157             link6 = (Link) processChain.getEdge(task2, place4);
158             link7 = (Link) processChain.getEdge(place5, task2);
159             link8 = (Link) processChain.getEdge(task2, place7);
160             link9 = (Link) processChain.getEdge(place8, task2);
161
162             assertTrue(link1!=null);
163             assertTrue(link2!=null);
164             assertTrue(link3!=null);
165             assertTrue(link4!=null);
166             assertTrue(link5!=null);
167             assertTrue(link6!=null);
168             assertTrue(link7!=null);
169             assertTrue(link8!=null);
170             assertTrue(link9!=null);
171             /**
172              * Verify that flow amounts of links are correct
173              */
174
175             /**
176              * link1
177              *
178              * 4.26127192510492E2 kJ electricity
179              */
180             this.logger.debug("link1=" + link1);
181             formulaProperty=(FormulaProperty)link1.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
182
183             assertTrue(formulaProperty!=null);
184             assertTrue(formulaProperty.isValueCalculated());
185             assertTrue(!formulaProperty.isCalculationFailed());
186             assertEquals(426.127192510492, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
187
188             /**
189              * link2
190              *
191              * 1.07607876896589E0 kg water
192              */
193             formulaProperty=(FormulaProperty)link2.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
194             assertTrue(formulaProperty!=null);
195             assertTrue(formulaProperty.isValueCalculated());
196             assertTrue(!formulaProperty.isCalculationFailed());
197             assertEquals(1.07607876896589, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
198
```

```
199             /**
200              * link3
201              *
202              * 2.9054126762079E-2 kg steam
203              */
204             formulaProperty=(FormulaProperty)link3.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
205             assertTrue(formulaProperty!=null);
206             assertTrue(formulaProperty.isValueCalculated());
207             assertTrue(!formulaProperty.isCalculationFailed());
208             assertEquals(0.029054126762079, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
209
210             /**
211              * link4
212              *
213              * 1.04702464220381E0 kg boiling water
214              */
215             formulaProperty=(FormulaProperty)link4.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
216             assertTrue(formulaProperty!=null);
217             assertTrue(formulaProperty.isValueCalculated());
218             assertTrue(!formulaProperty.isCalculationFailed());
219             assertEquals(1.04702464220381, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
220
221             /**
222              * link5
223              *
224              * 1.04702464220381E0 kg boiling water
225              */
226             formulaProperty=(FormulaProperty)link5.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
227             assertTrue(formulaProperty!=null);
228             assertTrue(formulaProperty.isValueCalculated());
229             assertTrue(!formulaProperty.isCalculationFailed());
230             assertEquals(1.04702464220381, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
231
232             /**
233              * link6
234              *
235              * 1.00 kg darjeeling infusion
236              */
```

```
237             doubleProperty=(DoubleProperty)link6.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
238             assertTrue(doubleProperty!=null);
239             assertEquals(1.00, ((Double)doubleProperty.getValue
                    ()).doubleValue(), MaterialFlowNetCalcTest.
                    JUNIT_DOUBLE_PRECISION);
240
241             /**
242              * link7
243              *
244              * 1.36123964274184E-2 kg darjeeling tea
245              */
246             formulaProperty=(FormulaProperty)link7.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
247             assertTrue(formulaProperty!=null);
248             assertTrue(formulaProperty.isValueCalculated());
249             assertTrue(!formulaProperty.isCalculationFailed());
250             assertEquals(0.0136123964274184, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
251
252             /**
253              * link8
254              *
255              * 5.05757021413967E-2 kg biological waste
256              */
257             formulaProperty=(FormulaProperty)link8.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
258             assertTrue(formulaProperty!=null);
259             assertTrue(formulaProperty.isValueCalculated());
260             assertTrue(!formulaProperty.isCalculationFailed());
261             assertEquals(0.0505757021413967, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
262
263             /**
264              * link9
265              *
266              * 5.59560959862262E-4 kg paper
267              */
268             formulaProperty=(FormulaProperty)link9.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
269             assertTrue(formulaProperty!=null);
270             assertTrue(formulaProperty.isValueCalculated());
271             assertTrue(!formulaProperty.isCalculationFailed());
272             assertEquals(0.000559560959862262, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
```

```
                ;
273         }
274         /**
275          * This test case is similar to the test case
276          * testCalculateOnTeaExampleFlowAmounts. The difference is
                 that
277          * in this test case the amount of Darjeeling infusion to
                be
278          * generated is 2, not 1.
279          *
280          * @see #testCalculateOnTeaExampleFlowAmounts()
281          */
282         public void testCalculateOnTeaExampleFlowAmounts2()
283         {
284                MaterialFlowNetCalc calculator=null;
285                Task task1=null;
286                Task task2=null;
287                Resource place1=null;
288                Resource place2=null;
289                Resource place3=null;
290                Resource place4=null;
291                Resource place5=null;
292                Resource place6=null;
293                Resource place7=null;
294                Resource place8=null;
295                Link link1 = null;
296                Link link2 = null;
297                Link link3 = null;
298                Link link4 = null;
299                Link link5 = null;
300                Link link6 = null;
301                Link link7 = null;
302                Link link8 = null;
303                Link link9 = null;
304                FormulaProperty formulaProperty=null;
305                DoubleProperty doubleProperty=null;
306                ProcessChain processChain=null;
307
308                /**
309                 * create tea process chain
310                 */
311                processChain=TeaProcessChainGenerator.
                       createProcessChain();
312
313                /**
314                 * change the flow amount of link6
315                 */
316                task2 = (Task) processChain.getNodeByName(
                       TeaProcessChainGenerator.T2_NAME);
```

```
317             place4 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P4_NAME);
318             link6 = (Link) processChain.getEdge(task2, place4);
319             processChain.removeLink(link6);
320             doubleProperty=(DoubleProperty)link6.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
321             doubleProperty.setValue(new Double(2.));
322             processChain.addLink(link6);
323
324             /**
325              * perform calculation
326              */
327             calculator=new MaterialFlowNetCalc();
328             calculator.calculateFlowAmounts(processChain);
329
330
331             /**
332              * Fetch all necessary tasks and resources
333              */
334             task1 = (Task) processChain.getNodeByName(
                    TeaProcessChainGenerator.T1_NAME);
335             task2 = (Task) processChain.getNodeByName(
                    TeaProcessChainGenerator.T2_NAME);
336             place1 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P1_NAME);
337             place2 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P2_NAME);
338             place3 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P3_NAME);
339             place4 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P4_NAME);
340             place5 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P5_NAME);
341             place6 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P6_NAME);
342             place7 = (Resource) processChain.getNodeByName(
                    TeaProcessChainGenerator.P7_NAME);
343             place8 = (Resource)processChain.getNodeByName(
                    TeaProcessChainGenerator.P8_NAME);
344
345
346             link1 = (Link) processChain.getEdge(place1, task1);
347             link2 = (Link) processChain.getEdge(place2, task1);
348             link3 = (Link) processChain.getEdge(task1, place6);
349             link4 = (Link) processChain.getEdge(task1, place3);
350             link5 = (Link) processChain.getEdge(place3, task2);
351             link6 = (Link) processChain.getEdge(task2, place4);
352             link7 = (Link) processChain.getEdge(place5, task2);
353             link8 = (Link) processChain.getEdge(task2, place7);
354             link9 = (Link) processChain.getEdge(place8, task2);
```

```
355
356             assertTrue(link1!=null);
357             assertTrue(link2!=null);
358             assertTrue(link3!=null);
359             assertTrue(link4!=null);
360             assertTrue(link5!=null);
361             assertTrue(link6!=null);
362             assertTrue(link7!=null);
363             assertTrue(link8!=null);
364             assertTrue(link9!=null);
365             /**
366              * Verify that flow amounts of links are correct
367              *
368              * They are equal to flow amounts in
369              * testCalculateOnTeaExampleFlowAmounts times 2.
370              */
371
372             /**
373              * link1
374              */
375             this.logger.debug("link1=" + link1);
376             formulaProperty=(FormulaProperty)link1.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
377
378             assertTrue(formulaProperty!=null);
379             assertTrue(formulaProperty.isValueCalculated());
380             assertTrue(!formulaProperty.isCalculationFailed());
381             assertEquals(2.*426.127192510492, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
382
383             /**
384              * link2
385              */
386             formulaProperty=(FormulaProperty)link2.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
387             assertTrue(formulaProperty!=null);
388             assertTrue(formulaProperty.isValueCalculated());
389             assertTrue(!formulaProperty.isCalculationFailed());
390             assertEquals(2*1.07607876896589, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
391
392             /**
393              * link3
394              */
395             formulaProperty=(FormulaProperty)link3.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
```

```
396             assertTrue(formulaProperty!=null);
397             assertTrue(formulaProperty.isValueCalculated());
398             assertTrue(!formulaProperty.isCalculationFailed());
399             assertEquals(2*0.029054126762079, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
400
401             /**
402              * link4
403              */
404             formulaProperty=(FormulaProperty)link4.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
405             assertTrue(formulaProperty!=null);
406             assertTrue(formulaProperty.isValueCalculated());
407             assertTrue(!formulaProperty.isCalculationFailed());
408             assertEquals(2*1.04702464220381, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
409
410             /**
411              * link5
412              */
413             formulaProperty=(FormulaProperty)link5.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
414             assertTrue(formulaProperty!=null);
415             assertTrue(formulaProperty.isValueCalculated());
416             assertTrue(!formulaProperty.isCalculationFailed());
417             assertEquals(2*1.04702464220381, ((Double)
                    formulaProperty.getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
418
419             /**
420              * link6
421              */
422             doubleProperty=(DoubleProperty)link6.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
423             assertTrue(doubleProperty!=null);
424             assertEquals(2*1.00, ((Double)doubleProperty.
                    getValue()).doubleValue(),
                    MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                    ;
425
426             /**
427              * link7
428              */
429             formulaProperty=(FormulaProperty)link7.getProperty(
                    MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
```

```
430                 assertTrue(formulaProperty!=null);
431                 assertTrue(formulaProperty.isValueCalculated());
432                 assertTrue(!formulaProperty.isCalculationFailed());
433                 assertEquals(2*0.0136123964274184, ((Double)
                        formulaProperty.getValue()).doubleValue(),
                        MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                        ;
434
435                 /**
436                  * link8
437                  */
438                 formulaProperty=(FormulaProperty)link8.getProperty(
                        MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
439                 assertTrue(formulaProperty!=null);
440                 assertTrue(formulaProperty.isValueCalculated());
441                 assertTrue(!formulaProperty.isCalculationFailed());
442                 assertEquals(2*0.0505757021413967, ((Double)
                        formulaProperty.getValue()).doubleValue(),
                        MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                        ;
443
444                 /**
445                  * link9
446                  */
447                 formulaProperty=(FormulaProperty)link9.getProperty(
                        MaterialFlowNetCalc.FLOW_PROPERTY_NAME);
448                 assertTrue(formulaProperty!=null);
449                 assertTrue(formulaProperty.isValueCalculated());
450                 assertTrue(!formulaProperty.isCalculationFailed());
451                 assertEquals(2*0.000559560959862262, ((Double)
                        formulaProperty.getValue()).doubleValue(),
                        MaterialFlowNetCalcTest.JUNIT_DOUBLE_PRECISION)
                        ;
452         }
453 }
```