



Business Models in a World Characterised by Distributed Generation

NNE5/2001/256

User manual e³-value editor

Jaap Gordijn, Hans Akkermans, Arthur Koks, Joost Schildwacht

Vrije Universiteit Amsterdam

Identifier:

Date: April 22, 2004

Class: Deliverable

Responsible Partner: VUA

Distribution: Public

Overview: This document describes how to use the e3-value editor

The BUSMOD Consortium consists of:

IBERDROLA	Principal Contractor & Coordinator	Spain
LABEIN	Principal Contractor	Spain
VUA	Principal Contractor	The Netherlands
ECN	Principal Contractor	The Netherlands
SINTEF	Principal Contractor	Norway
UMIST	Principal Contractor	United Kingdom
EnerSearch	Principal Contractor	Sweden

Control Versions:

Version	Date	Author	Description of Changes
---------	------	--------	------------------------

Table of Contents

<u>Chapter 1</u>	<u>Installation of the e3-value editor</u>	8
<u>Chapter 2</u>	<u>Guide on how to create a value model using the e3-value tool</u>	10
2.1	Guide on how to create a value model using the e ³ -value tool	10
2.1.1	Step 0: Opening the editor	10
2.1.2	The graphical components	11
2.1.3	Step 1. Adding market segments and actors to the edit workspace	13
2.1.4	Step 2. Add value activities	15
2.1.5	Step 3. Adding value interfaces, ports and exchanges	17
2.1.6	Step 4. Add stimuli to the e ³ -value model	19
2.1.7	Step 5. Editing the e ³ -value Properties	21
2.1.8	Step 6. Adding value objects and value transactions.	22
2.1.9	Step 7: Modelling value transactions	23
2.1.10	Step 8. Editing scenario ports, adding weights	26
2.1.11	Step 9. Creating formulas and adding them to the model	26
2.1.12	Step 10. Saving your business model	29
2.2	How to create profitability sheets with the e3-value editor	30
2.2.1	Introduction	30
2.2.2	Profitability sheet generation	30
2.2.3	Profitability sheet document structure	33
<u>Chapter 3</u>	<u>E3-value editor technical reference</u>	37
3.1	RDF generation steps	37
3.1.1	Create instances	37
3.1.1.1	Simple mapping	37
3.1.1.2	Merging objects	38
3.1.1.3	Rules for merging model concepts	39
3.1.2	Assign attributes	41
3.1.3	Generate RDF files / file stream	41
3.2	Profitability sheet generation	42
3.2.1	Approach	42
3.2.2	Profitability sheet generation steps	42
3.2.3	Assign occurrences to the value ports and the value transactions.	43

3.2.4	Parse all formulas	56
3.2.5	Create a 'Formula Sheet' for all formulas (or placeholders).	56
3.2.6	Assign 'Default Valuation Formulas' to each value port	57
3.2.7	Update the Formula sheet with 'Default Valuation Formulas'	59
3.2.8	Create Excel sheets for each e ³ -value ontology class	59
3.2.9	Create Excel (profitability) sheets for any Value Source instance.	60
<u>Chapter 4</u> <i>E3-value editor: tips, tricks and error messages</i>		<u>63</u>
4.1	<i>Import shortcuts and tricks</i>	63
4.2	<i>Error messages</i>	64
4.3	<i>Formulas: Reserved names</i>	72
4.4	<i>Formula Syntax</i>	73
4.5	<i>Formulas: Formula limitations</i>	76
4.6	<i>Merging Objects</i>	76
4.7	<i>Valuation functions</i>	79
<u>Chapter 5</u> <i>E3-value editor: instructions for the programmer</i>		<u>82</u>
5.1	<i>Batik</i>	82
5.1.1	Details	82
5.1.2	Implementation	83
5.2	<i>Jakarta POI HSSF</i>	83
5.2.1	Details	83
5.2.2	Implementation	83
5.2.3	Limitations	84
5.3	<i>JGo™ Java diagram graphics libraries</i>	84
5.3.1	Details	84
5.3.2	Implementation	84
5.4	<i>GOLD Parser</i>	85
5.4.1	Details	85
5.4.2	Implementation	85
5.5	<i>RDF2Java</i>	86
5.5.1	Details	86
5.5.2	Implementation	86
5.6	<i>Customizing third party software</i>	87
5.6.1	JGo	87

5.6.1.1	JGoDocument.java	87
5.6.1.2	JGoTextEdit.java	88
5.6.1.3	JGoView	89
5.6.2	Gold	89
5.6.2.1	GOLDParser.java	89
5.6.2.2	LookAheadStream.java	89

List of Figures

<i>Figure 1. Test if a recent version of the Java Runtime Environment is installed.</i>	8
<i>Figure 2 starting the tool from the console</i>	9
<i>Figure 3. The “about”-button reveals the version and other important settings.</i>	9
<i>Figure 4. Overview e³-value editor.</i>	10
<i>Figure 5. Edit text in the edit box.</i>	12
<i>Figure 6. Step 1 : drop market segments and actors on the workspace.</i>	13
<i>Figure 7. Edit text in the edit box.</i>	14
<i>Figure 8. Enlarge an object by click and dragging the green rectangles.</i>	14
<i>Figure 9. Adding value activities with the e³-value tool.</i>	15
<i>Figure 10. Move text on an object to another location.</i>	16
<i>Figure 11. e³-value model after completing step 1 and step 2.</i>	16
<i>Figure 12. Adding value interface on an activity, an actor or a market segment.</i>	17
<i>Figure 13. Adding and removing value ports.</i>	17
<i>Figure 14. Move the text on the value exchange line to another location.</i>	18
<i>Figure 15. Value model after step 3.</i>	18
<i>Figure 16. Selecting and rotating the ‘and’ and ‘or’ scenario elements.</i>	19
<i>Figure 17. Connect the black dots on the scenario elements to create a scenario path.</i>	19
<i>Figure 18. Final e³-value model after fulfilling step 1 until 4.</i>	20
<i>Figure 19. Right clicking the market segment opens the context menu.</i>	21
<i>Figure 20 Choosing the Edit E3Properties pops the E3Properties Editor.</i>	21
<i>Figure 21 The context menu of a value port opens the collection editor of value objects</i>	22
<i>Figure 22 E3Properties Editor invoked from the collection editor on a value object</i>	23
<i>Figure 23 E3Properties Editor invoked on a value transaction</i>	23
<i>Figure 24 Example: Incorrect value transactions</i>	24
<i>Figure 25 Example: Incorrect value transactions</i>	25
<i>Figure 26 Example: Correct value transactions</i>	25
<i>Figure 27 The properties option in the context menu of a scenario port</i>	26
<i>Figure 28. Assigning formulas to objects</i>	27
<i>Figure 29 Save as dialog - export profitability sheets</i>	31
<i>Figure 30 Merged concepts dialog example</i>	32
<i>Figure 31 Error message dialog example</i>	32
<i>Figure 32 “Unseen objects” dialog example</i>	33
<i>Figure 33 Actor-sheet example</i>	34
<i>Figure 34 Object-merging: Two graphical objects may conceptually be identical.</i>	38
<i>Figure 35 - some merged instances in a Map</i>	39
<i>Figure 36 Example of multiple graphical objects representing a single conceptual object</i>	77
<i>Figure 37 Example of multiple graphical objects representing a single conceptual object</i>	77
<i>Figure 38 Example of default valuation formulas</i>	81

List of Tables

<i>Table 1 Error message overview</i>	71
<i>Table 2 Reserved formula names</i>	73
<i>Table 3 Writing formulas - top level object reference syntax</i>	73
<i>Table 4 Writing formulas - sub-level object reference syntax</i>	75
<i>Table 5 Writing formulas - Attribute reference syntax</i>	75
<i>Table 6 Criteria for merging graphical objects</i>	79

Chapter 1 Installation of the e3-value editor

Three steps are needed to run the tool:

- Ensure the Java Runtime Environment version 1.4.2 or later is installed.
- Get E3Current.jar on your system.
- Run the command `Java -jar E3Current.jar`

How to accomplish this in Windows 98, ME, NT, 2000 or XP will be explained next.

To check the JRE is properly installed open the Windows Start Menu and choose the run option. Give the command "cmd" and a command box will open. In this command box give the command "Java -version" and the version number of your JRE if any is displayed.

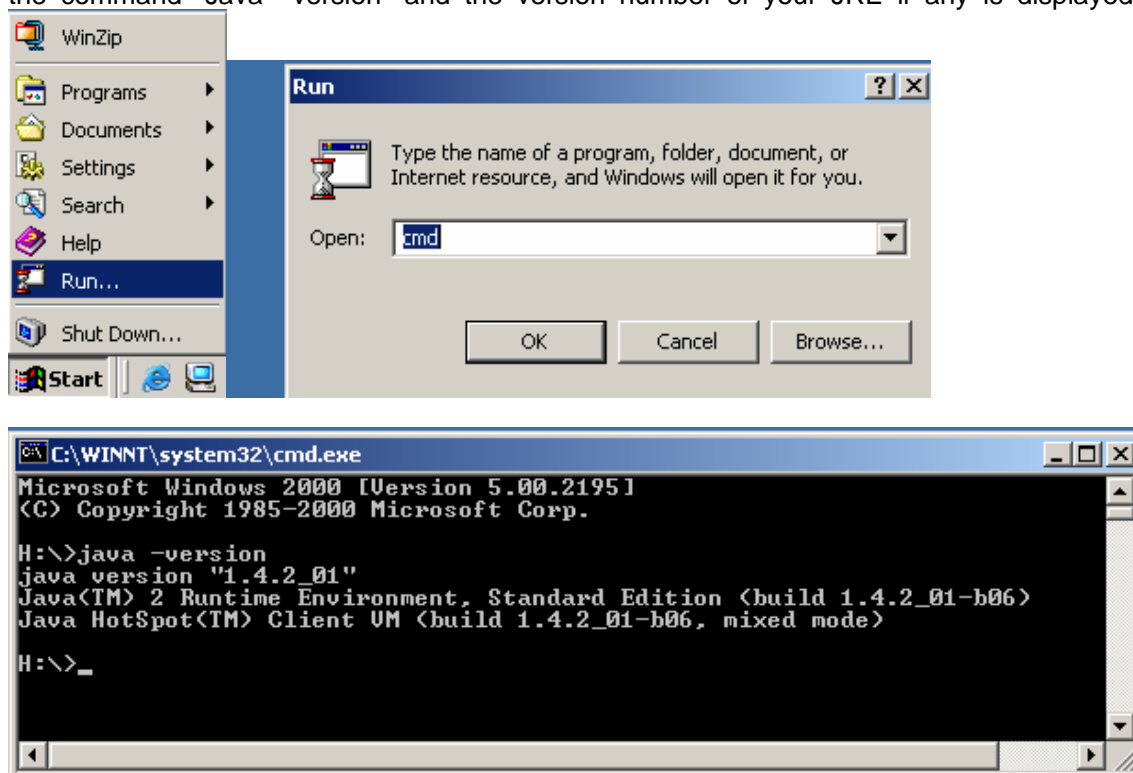


Figure 1. Test if a recent version of the Java Runtime Environment is installed.

If this test fails, you have to visit: <http://Java.sun.com/j2se/1.4.2/download.html> and follow the instructions.

To download your copy of E3Current.jar visit: <http://www.cs.vu.nl/~gordijn/research.htm> and press the here link at the top.

Finally, you click on the jar file that you just downloaded and the tool will start otherwise open a command box as before and issue the command

“Java -jar yourpath\E3Current.jar” as in the figure below.

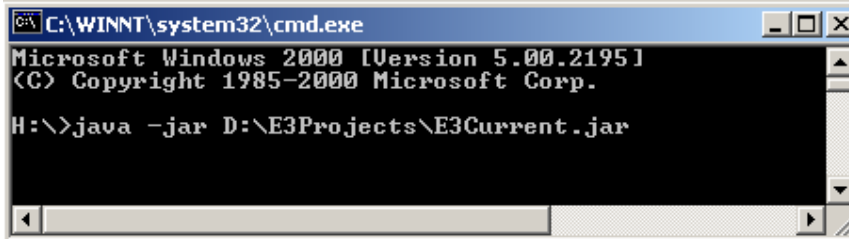


Figure 2 starting the tool from the console

Once you have started the tool the 'About' button display some information about the settings. Important is the version number. Every time you download a new version, you are advised to keep a copy of the old version, as there is no guarantee the business models you created before can be loaded in a newer version.



Figure 3. The “about”-button reveals the version and other important settings.

Chapter 2 Guide on how to create a value model using the e3-value tool

2.1 Guide on how to create a value model using the e³-value tool

In this chapter we create a value model by using the e3-value tool. You can use it as an example as how to create your own model. Also the tool provides help functionality for guidance. The whole process is described in steps, starting with the opening the editor.

2.1.1 Step 0: Opening the editor

Create a value model from scratch using the e³-value tool After loading the e³-value edit tool, the following main screen appears:

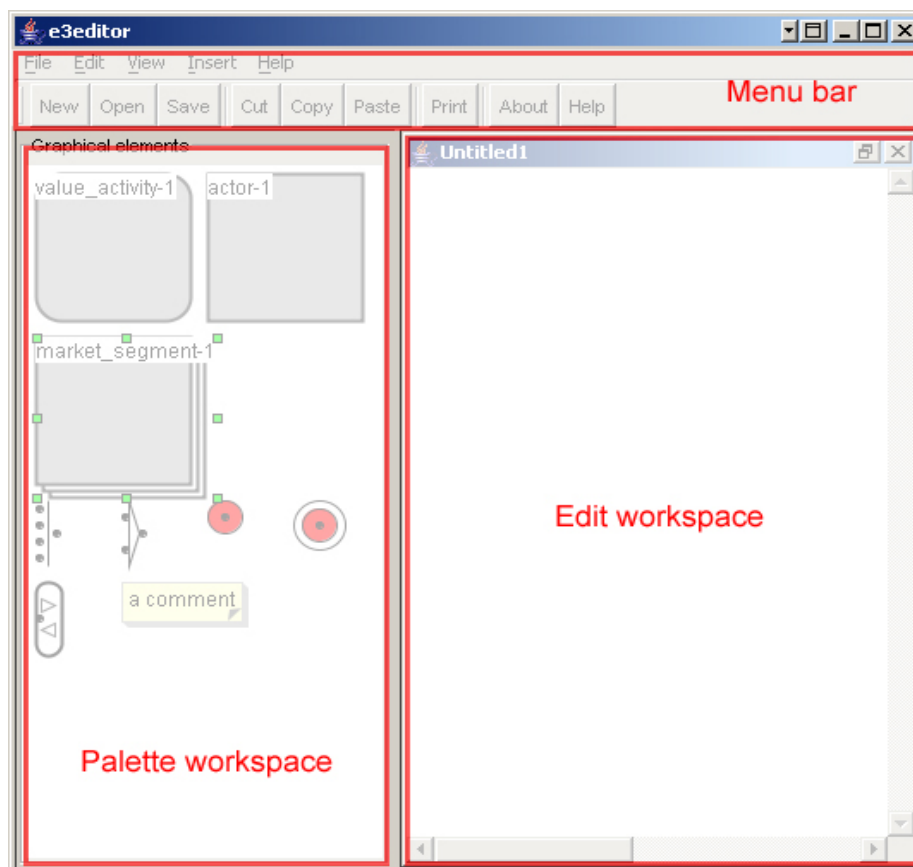


Figure 4. Overview e³-value editor.

Menu bar: The menu bar displays functions for storing, opening and printing a model. The following functions do also have shortcuts: New: CTRL-N, Open: CTRL-O, Save: CTRL-S, Cut: CTRL-X, Copy: CTRL-C, Paste: CTRL-V, Print: CTRL-P.

Palette workspace: In this part of the screen at the left you select palette icons to build up your model.

Edit workspace: This central part of the screen displays the model under construction. You drag icons from the palette workspace and drop them on this area to build your model.

2.1.2 The graphical components

In the next figure on the next page the collection of visual elements is shown. Most of the concepts in the e^3 -value ontology you will find in this figure. The grey areas represent actors (both composite and elementary), market segments and value activities. Because in the graphical editor they have significant common behaviour we refer to these concepts as “value sources”. On the edge of these value sources you find oval white areas. These are the value interfaces. The triangles on these value interfaces represent value ports. Some ports are directed outward from a value source others are directed inward. We distinguish in- and out ports. On the outward side ports are connected to other value sources. These connections represent value exchanges. The dotted lines on the value sources represent connection elements; they connect scenario elements. In the figure all types of scenario elements are displayed: The start stimulus, the end stimulus, the OR fork, the OR join, the AND fork, the AND join and the value interface. The value interface we mentioned before in its role as a bundler of value ports.

The editor also knows scenario ports (black dots on the scenario elements), the exchange label, the name label and the comment. These are only visual constructs and therefore not part of the ontology. When you move your mouse on one of the graphical objects a *tool tip* might show up. It tells you the unique (in the scope of this diagram), non-editable ID that the graphical editor assigned to this object and the name of the object. At creation every graphical object is given a default name by the editor. This default name is a combination of the object type and its ID. The user can change this name in whatever he chooses. In some cases the tool regards objects with the same name as identical even as the ID's might differ. This will be explained later on. See section 4.6 .

At the bottom of the figure you see the Value Transactions Collection Editor. It is one of the several types of windows, which can be opened by the user. It displays a list of several value transactions and allows the user to maintain this list and to edit the value transaction. We will discuss these windows later in greater detail. See section 2.1.9.

A value transaction is like the value object an invisible object but part of the e^3 -value ontology and hence part of the diagram. They are maintained in separate windows on the workspace, which are opened on with the right mouse button. In the Value Transactions Collection Editor you see value transaction vt132 highlighted, representing that the edit and the delete button applies here. In this particular state, the workspace highlights the value exchanges, which belongs to this particular value transaction vt132. Clicking on a value exchange will toggle its belonging to this value transaction. In this way you can partition the

value exchanges into value transactions.

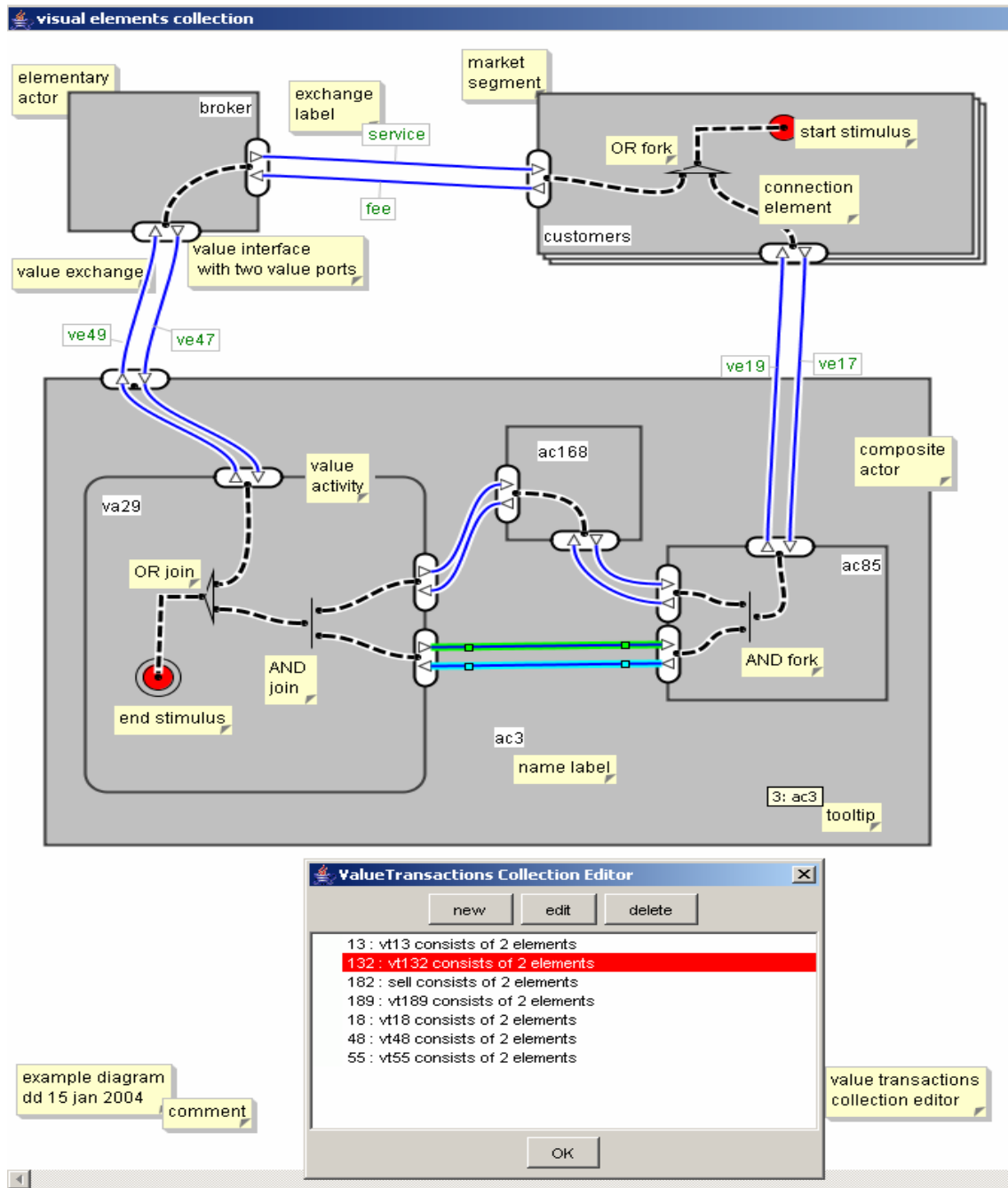


Figure 5. Edit text in the edit box.

2.1.3 Step 1. Adding market segments and actors to the edit workspace

From the palette workspace, drag the 'market segment' icon or the 'actor' icon and drop the item to the icon on the edit workspace. This action is represented in the figure below. The newly added element gets a unique name, which can be changed by the user.

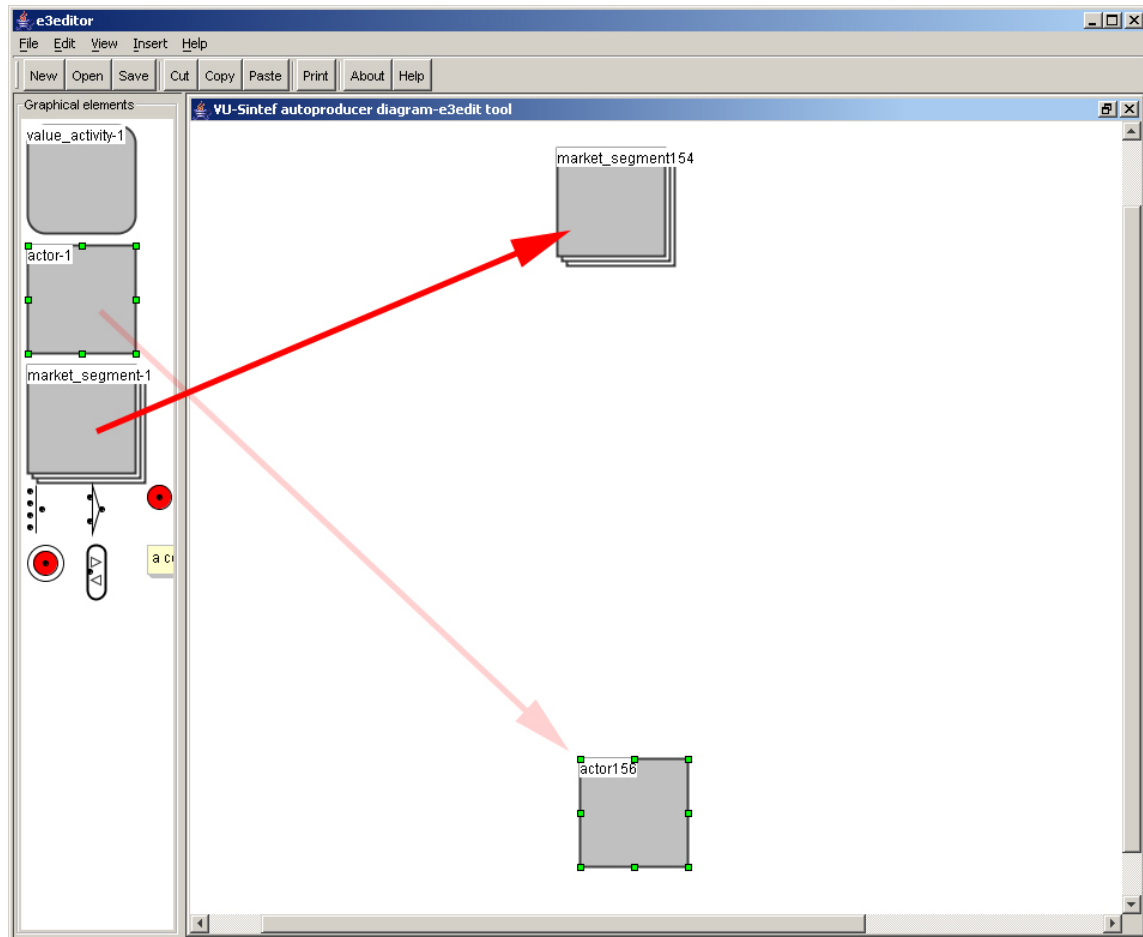


Figure 6. Step 1 : drop market segments and actors on the workspace.

Step 1b. Repeat this step until all market segments are in place.

Step 1c. Next, drag an actor icon from the palette workspace and drop the icon to some position on the edit workspace. If you drop an actor on a market segment or on another actor, the dropped actor becomes part of the second object. Later you can release this actor again by dragging it to another position.

Step 1d. Repeat this step until all actors are in place.

Edit text

To change the name of a market segment or actor, click once on the text 'market segment_{x}' which will bring you in the edit mode. Change the name to your liking and exit the edit mode by clicking anywhere in the edit workspace.



Figure 7. Edit text in the edit box.

You can change the size of each object (e.g. an actor) by clicking on one of the green rectangles on each corner of the object, click, hold and drag the green rectangle to a desired direction.

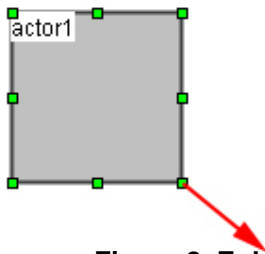


Figure 8. Enlarge an object by click and dragging the green rectangles.

2.1.4 Step 2. Add value activities

In this step we add value activities to actors and market segments. From the palette editor, drag the 'value activity' icon to the edit workspace and drop it on some actor or market segment.

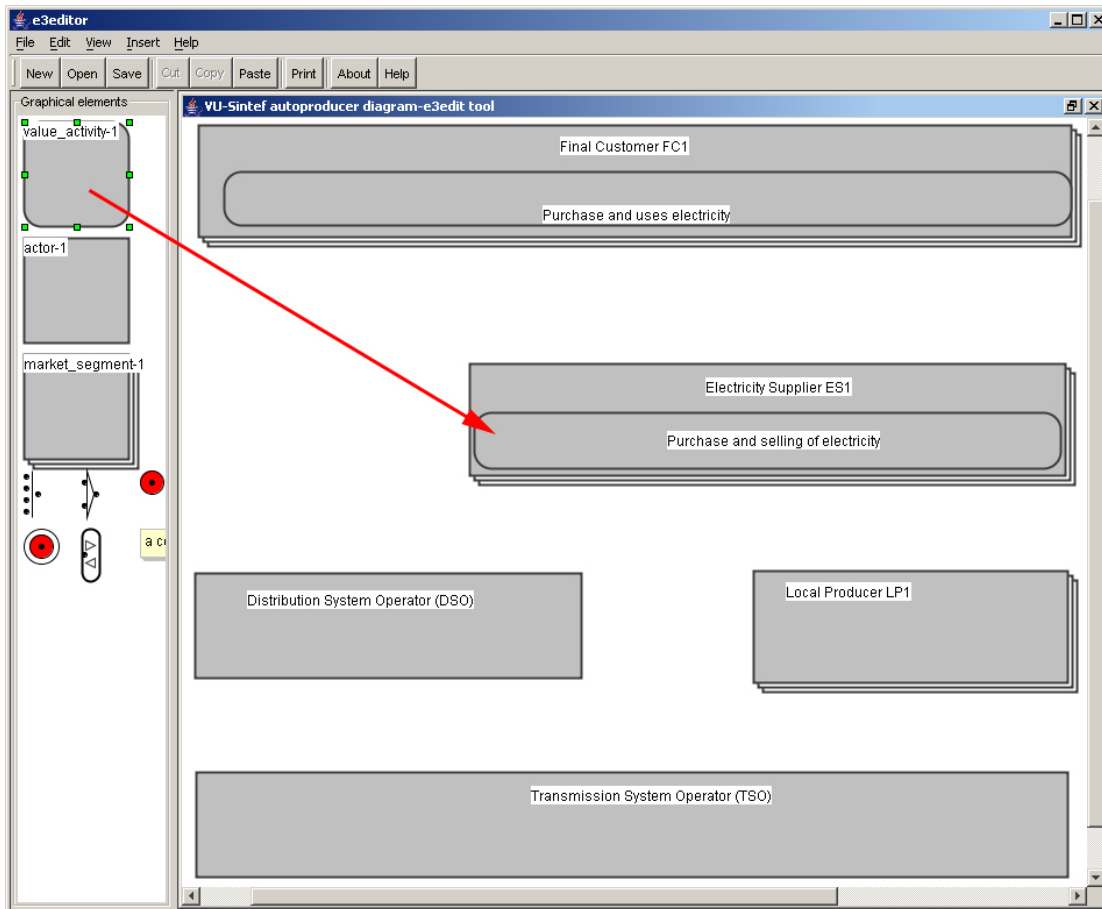


Figure 9. Adding value activities with the e³-value tool.

At this moment it is worthwhile to consider the attachment of value sources to each other because reattaching or releasing now is still easy. Changing locations only without changing the attachments will remain easy of course, but reattaching and releasing after the exchanges are already in place is less efficient.

Again you can edit the label as described before. Alternatively you can drag the text by first selecting the text (see Figure 10. Move text on an object to another location.) and then press the Escape button. Now you will get green rectangles around the text box. You can move the text by clicking, holding and moving the textbox with your cursor.

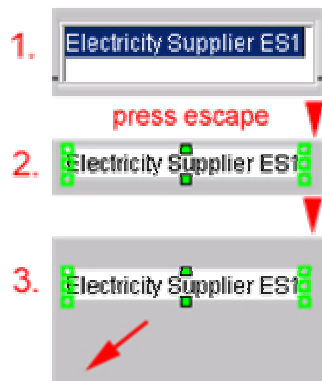


Figure 10. Move text on an object to another location.

The results from step 1 and 2 combined are presented in the following figure:

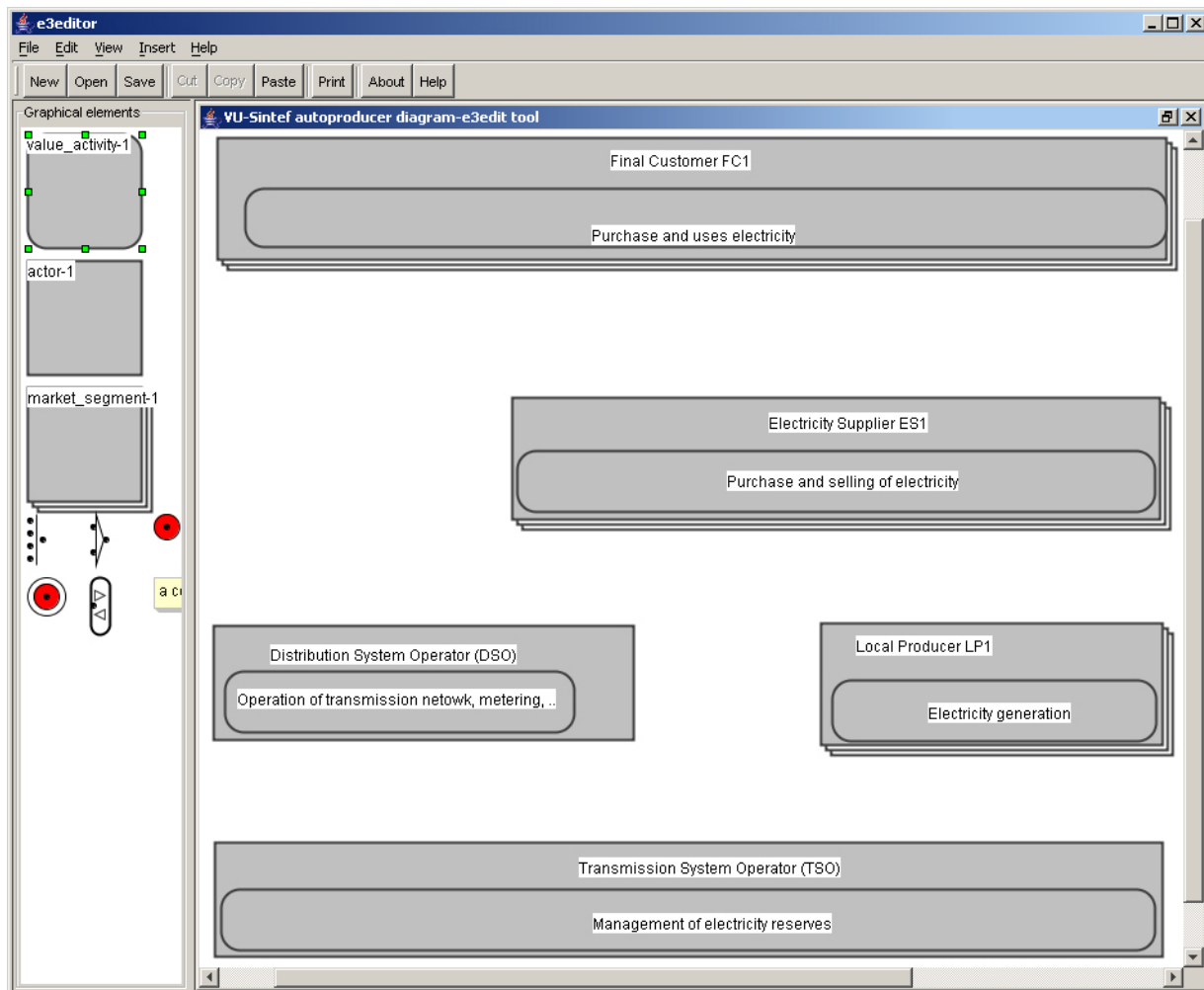


Figure 11. e³-value model after completing step 1 and step 2.

2.1.5 Step 3. Adding value interfaces, ports and exchanges

In this step we add value exchanges between value sources. From the palette editor select the value interface icon and drag it on a value source. The position at which you release the mouse button is critical (!) and therefore visualized in the next figure. When dragging the icon (1), you will see a white rectangle with a mouse arrow above it. Do not focus on this arrow, but do focus on the rectangle (2). Be sure the rectangle is really inside the value source. The tool will adjust the interface to the edge so give it some leeway.



Figure 12. Adding value interface on an activity, an actor or a market segment.

You can add value ports to an interface by double clicking it. The Interface Properties Dialog pops up. When you click on the 'In++' button an extra 'in' value port will be added. When you click on the 'Out++' button an extra 'out' value port will be added. To remove ports you select them and press the 'delete' key or the 'back space' key on your keyboard. Alternatively right clicking on a port opens the properties dialog with the 'Cut' option.

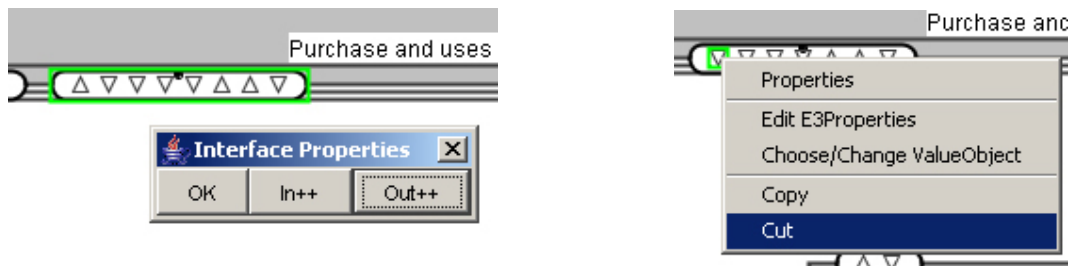


Figure 13. Adding and removing value ports.

Step 3a. Add selected value exchanges.

Step 3b. Repeat step 3a until all the value exchanges are in place.

Step 3c. Next, we will add a value exchange between two value ports. Select a port with your mouse, keep the button pressed and move to another port. You will see that the editor tries to make a connection from the selected port to some other port directed by your mouse. When the 'hand' cursor is on its destination, release the mouse button. The value exchange line will then be drawn. The editor allows only sensible exchanges between ports.

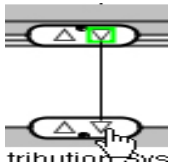


Figure 12. Creating a value exchange.

Step 3d. Create all the value exchanges by repeating step 3c. Sometimes all ports of an

interface are to be connected to another not yet existing interface. In this case a shortcut comes in handy. Select the existing interface and click with the 'Ctrl' and the 'Shift' keys pressed on the position where the new interface will appear with all its ports rightly connected.



Figure 14. Move the text on the value exchange line to another location.

Changing texts and positions of the labels attached to the value exchanges is achieved as before.

After step 3, the following overall model is created:

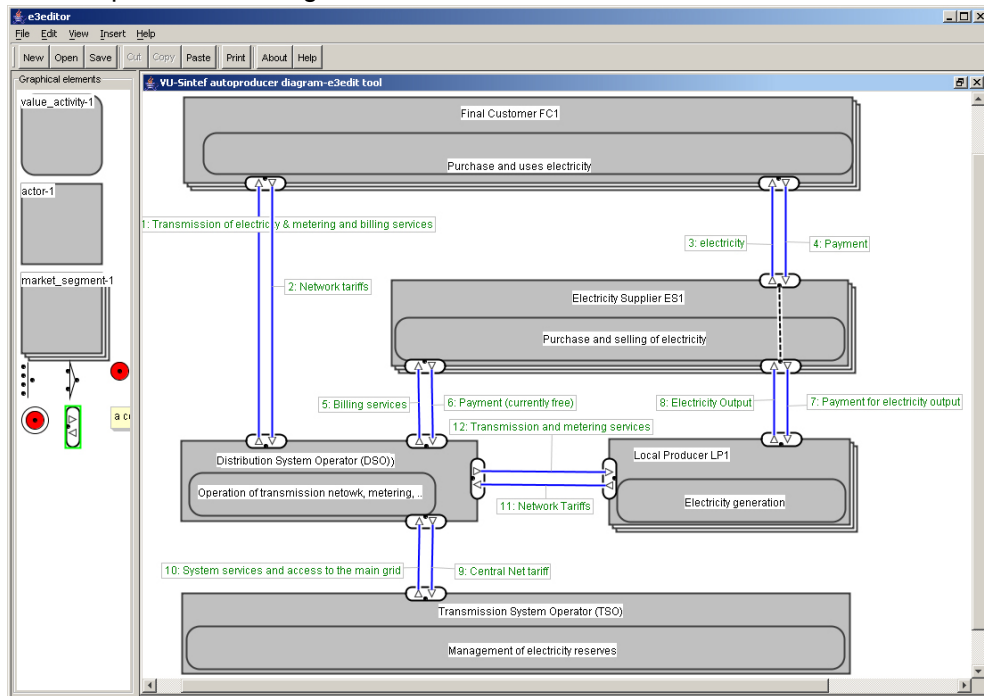


Figure 15. Value model after step 3.

2.1.6 Step 4. Add stimuli to the e³-value model

In this step we are adding the start and stop stimuli in the value diagram.

Step 4a. Drag the start stimulus from the palette and drop it on a value source in the workspace.

Step 4b. Repeat this with 'and' or 'or' scenario elements (1). You can rotate the 'and' or 'or' element by double clicking it. The 'Scenario Element Dialog' will pop up. If you click on 'rotate 90' (2) button, the icon will rotate in the palette editor. Press 'OK' to leave the Scenario Element Dialog.

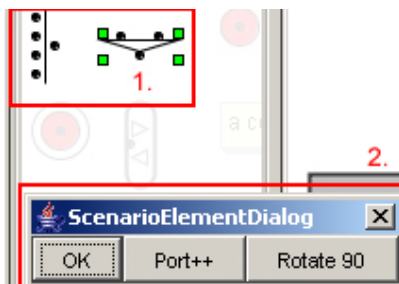


Figure 16. Selecting and rotating the 'and' and 'or' scenario elements.

Step 4c. Finally you add one or more stop stimuli to your value sources

Step 4d. Finish the scenario path by connecting the black dots from the scenario elements to each other. E.g. click on the black dot of a scenario element until it gets green. Next, hold the right mouse button and drag the mouse cursor to the middle of a value exchange. On the black dot of the value exchange, release the mouse button. A scenario path is automatically drawn (right in the figure).

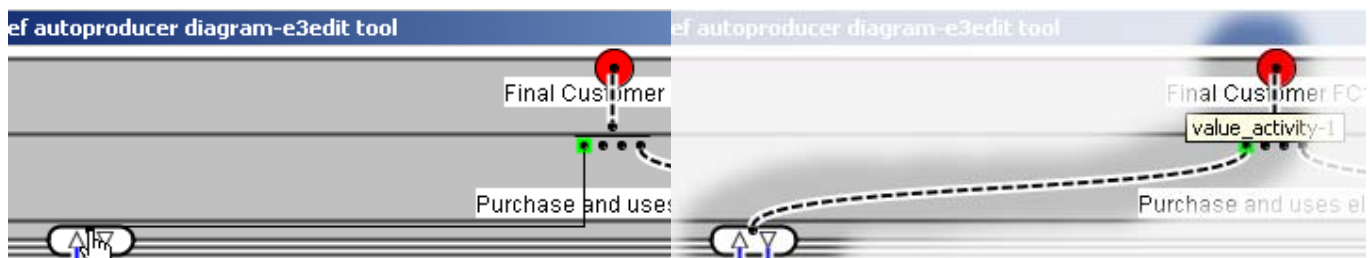


Figure 17. Connect the black dots on the scenario elements to create a scenario path.

Step 4e. Repeat these steps until all scenario paths are in place.

After step 1 until 4, the following final e^3 -value model has been created in the e^3 -value editing tool.

It is a simple but complete visualization of a business model.

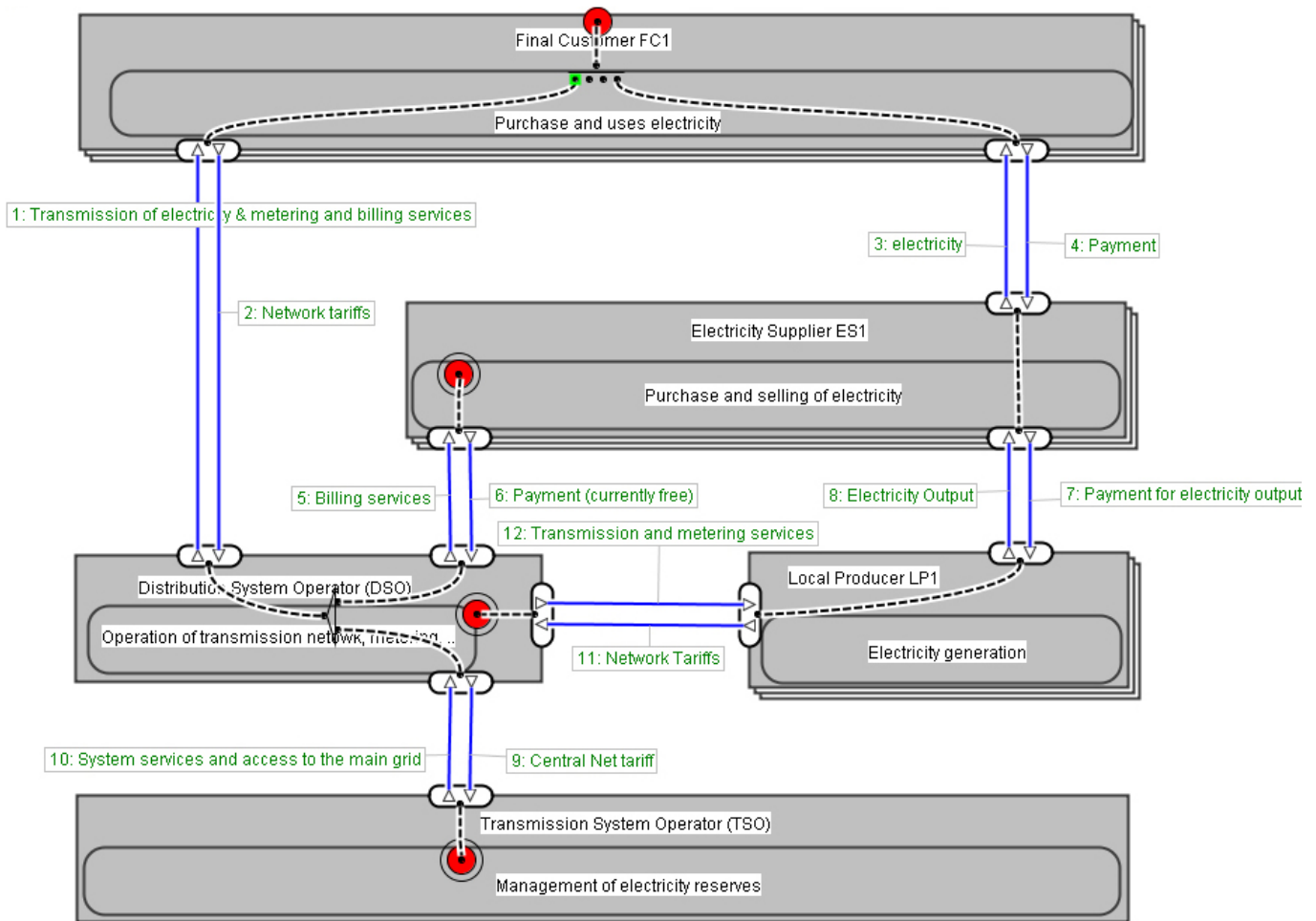


Figure 18. Final e^3 value model after fulfilling step 1 until 4.

2.1.7 Step 5. Editing the e^3 -value Properties

All elements from the e^3 -value ontology share the properties of the e^3 -value_object. Right clicking on such element allows the user of the editor to edit these properties.

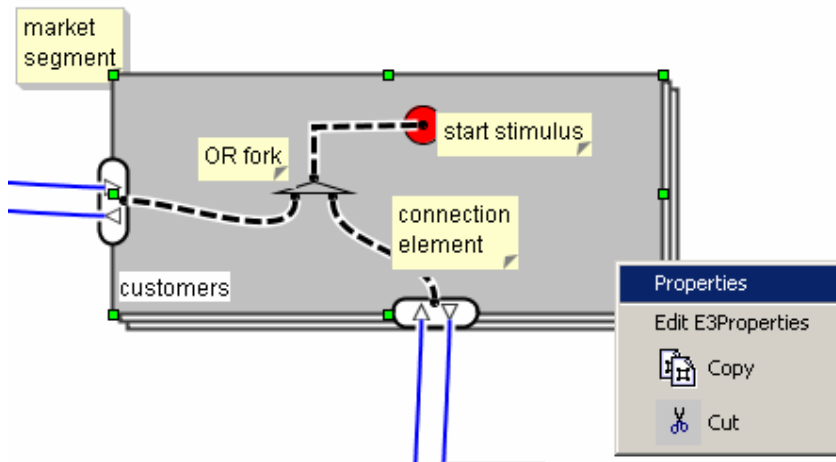


Figure 19. Right clicking the market segment opens the context menu.

The properties option opens a dialog for all the graphical parameters. Here you can change the colour, the thickness of the edge and so on. The copy and cut options are as in Windows. More interesting is the Edit E3Properties option. It opens the following dialog:

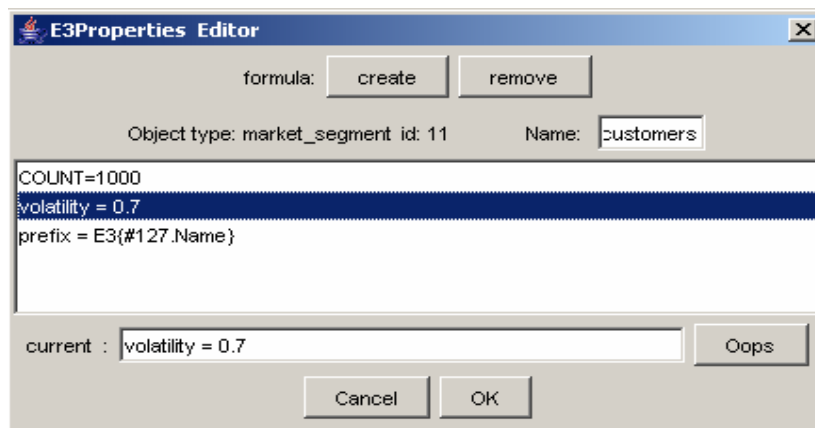


Figure 20 Choosing the Edit E3Properties pops the E3Properties Editor.

The "Name:" text field allows you to change the name of this e^3 -value object. Further, the editor allows you to maintain a list of formulas assigned to this market segment. The highlighted formula is being edited in the text field below. The "oops" button allows you to undo your edits so far. The last formula for prefix displays an E3 formula. This will be explained later in full detail. The editor may add construct dependent formulas, such as a count formula for start stimuli. The capitals in the formula indicate that COUNT is a reserved

word and the formula is not to be removed. The left hand side of such formulas is not to be changed.

2.1.8 Step 6. Adding value objects and value transactions.

In the e^3 -value ontology, value ports have a value object. Additionally, value exchanges are part of a value transaction. The editor provides facilities to create value objects and value transactions, and to relate these to value ports and value exchanges respectively. Right clicking in the background of the workspace allows you to open the collection editor for value objects or value transactions. These are semi modal windows in the sense that they do not completely disable the workspace. Firstly value ports/ value exchanges are highlighted indicating they belong to the value object/ value transaction selected in collection editor. Secondly clicking any value port/ value exchange will toggle its membership to the selected value object/ value transaction. This will be reflected by the highlighting in the workspace and by the displayed number of members in the collection window.

Assigning a value object to a value port will release it from other value ports. For convenience, this value object will also be assigned to all free value ports connected to this first value port by value exchanges. This propagates recursively through the model.

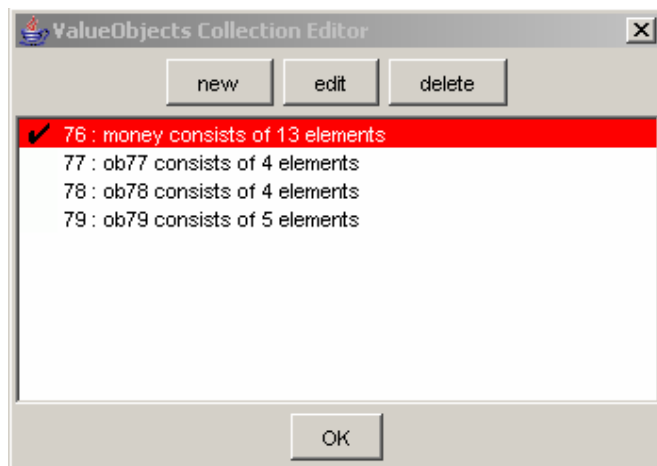


Figure 21 The context menu of a value port opens the collection editor of value objects

In Figure 21, the value object money is checked. This indicates that the value port from which the editor was opened transmits money. Initially this is also the value object selected in the collection editor as indicated by the highlight. In the workspace, all the value ports transmitting money are highlighted. From the highlighted line in the figure, you see that thirteen value ports transmit money. Because a value object is not a visible object it is not possible to right click it to get its context menu. Editing its E3properties is therefore accomplished with the edit button in the collection editor. It opens the usual E3Properties Editor as a dialog on the selected value object. In the next figure you see this editor invoked on the value object money.

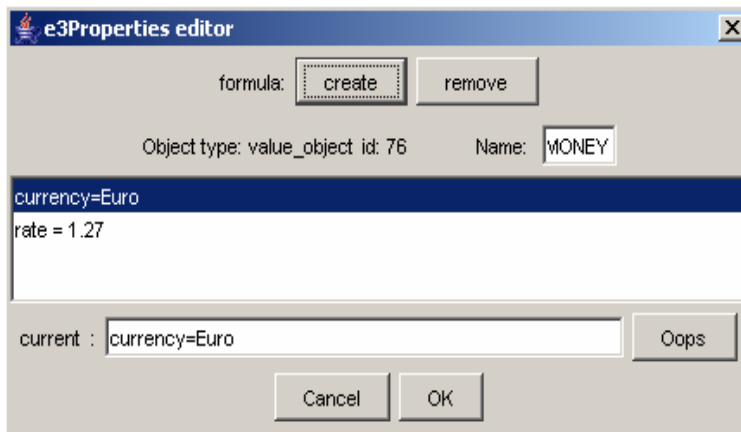


Figure 22 E3Properties Editor invoked from the collection editor on a value object

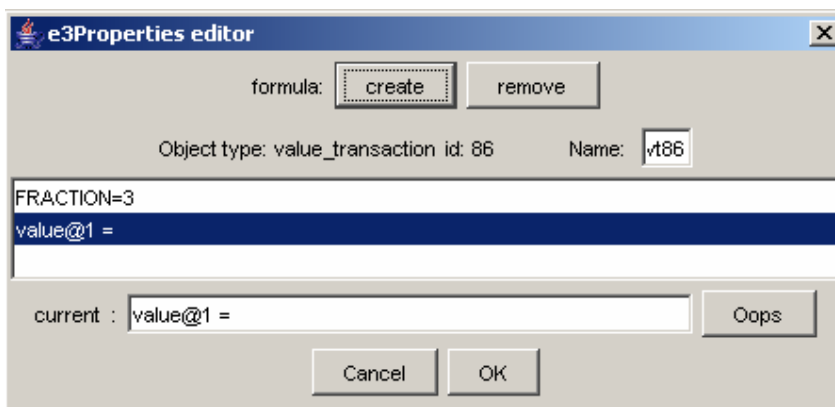


Figure 23 E3Properties Editor invoked on a value transaction

The FRACTION formula in the above editor represents an integer attribute in the e^3 -value ontology. It applies in cases when several value transactions originate from the same value interface. It measures as a weight the part of the occurrences that are transmitted through this value transaction. The FRACTION formula is automatically added with initial value 1. When creating new formulas the editor automatically generates a left hand side that can be changed in the lower text field.

2.1.9 Step 7: Modelling value transactions

When modelling value exchanges, it is important that you also create the appropriate value transactions, and assign them to the value exchanges correctly.

The e^3 -value tool uses value transactions in the process of generating parts of the profitability sheets, and therefore it is required that these are included in your models. If the value transactions are not present, or if they have not been modelled correctly, this will cause problems with profitability generation. In the best event, the e^3 -value tool will detect

the problem and display an error message. In the worst-case scenario, some of the value exchanges will simply be ignored.

Make sure that your model is complete (and correct) before generating profitability sheets!

Below you will find a number of simple examples of correct and incorrect usage of value transactions.

Example 1 – Incomplete model: No value transactions

In this simple example, we have not added any value transactions to the model:

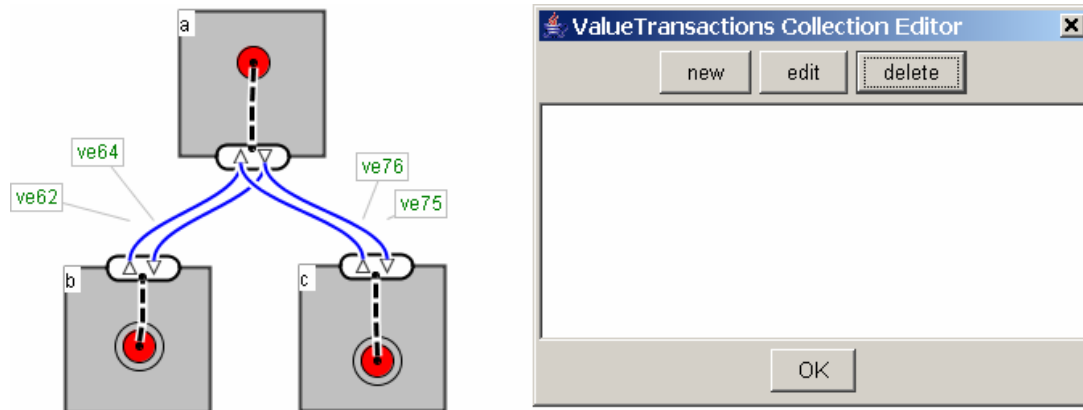


Figure 24 Example: Incorrect value transactions

Attempting to generate a profitability sheet from this model will result in a failure, and an error message will be displayed; without value transactions, the e^3 -value tool will not be able to find a (scenario) path.

Example 2 – Incomplete model: Incomplete value transactions

In this simple example, one value transaction has been added to the model, but the other one has been 'forgotten'.

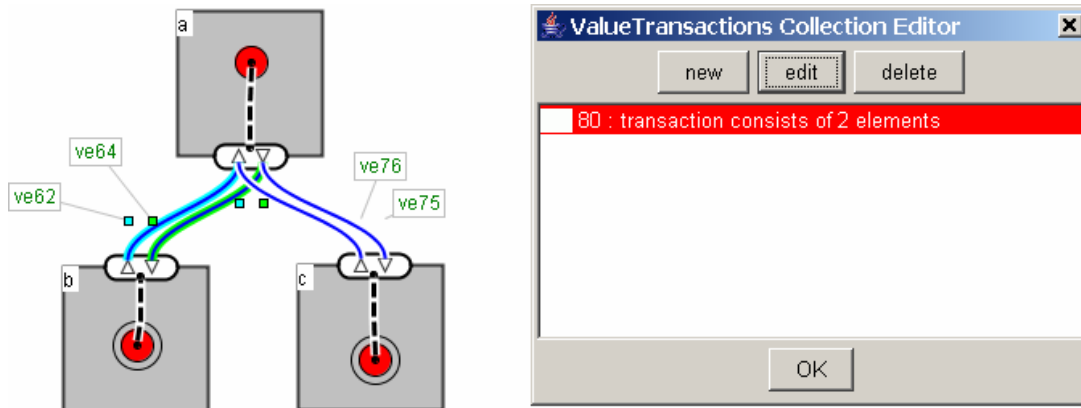


Figure 25 Example: Incorrect value transactions

Generating a profitability sheet from this example model would *not result in an error* in contrast to the previous example). Instead, it will result in a corrupt profitability sheet. The e^3 -value tool will simply ignore the two value exchanges (ve76 and ve75) because they are not involved in a value transaction.

Example 3 – Complete model

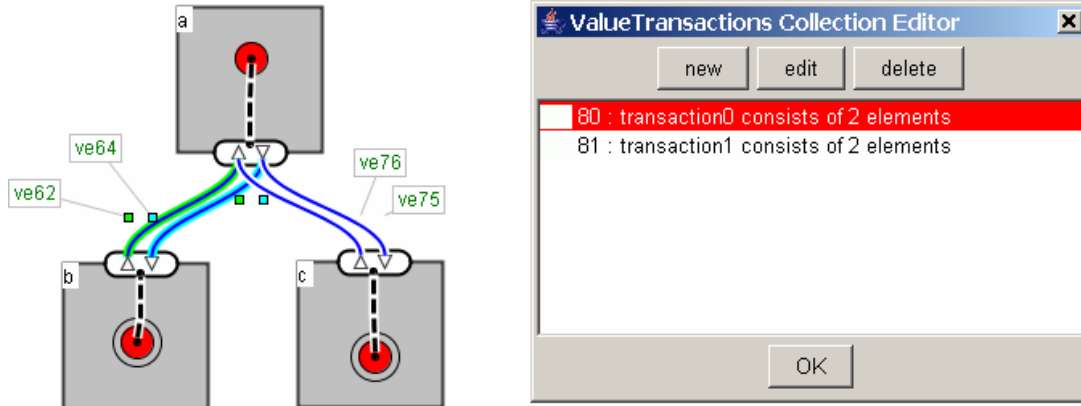


Figure 26 Example: Correct value transactions

In the example above, the value exchanges have been correctly bundled into two value transactions. The e^3 -value tool will recognise both scenario paths and generate a correct profitability sheet.

2.1.10 Step 8. Editing scenario ports, adding weights

Scenario ports are not part of the e^3 -value ontology and therefore do not have E3 properties. Clicking the right mouse button for the context menu gives only one option for the properties. Making this choice brings forward a card layout as shown in the figure below.

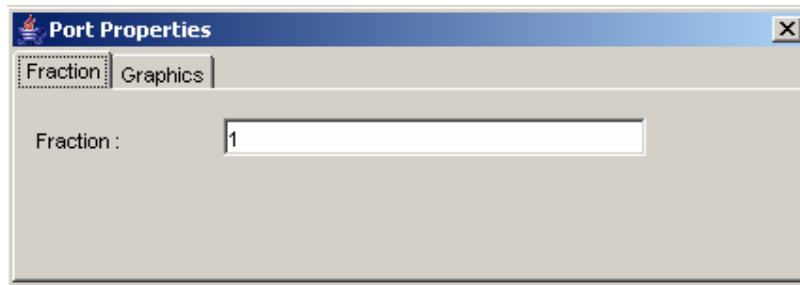


Figure 27 The properties option in the context menu of a scenario port

The graphics card gives access to the graphical parameters of the scenario port. The fraction card shown here allows us to change the value of the Fraction attribute of this scenario port. This fraction plays a similar role as the fraction of value transactions in the previous section. It is most important in OR and AND forks as it controls as before the fraction of the occurrences flowing through the assigned scenario path.

2.1.11 Step 9. Creating formulas and adding them to the model

In this paragraph we explain formulas as they are represented in the e^3 -value tool, how to write them, and how to attach them to objects in your model.

Formulas in the e^3 -value tool are based on Microsoft Excel (98). This makes it possible to write formulas in the e^3 -value tool and maintaining them when we want to generate profitability sheets (which are created in the Excel 98 format). This means that you can use Excel-functions and expressions in the e^3 -value tool!

Before we go into the specifics of formula syntax and structure, we will discuss how to create them and attach them to other objects in the e^3 -value tool.

Step 1. To create a formula, right click the object you wish to attach it to.

In the context menu, select 'Edit E3properties'. The 'E3properties Editor' screen appears:

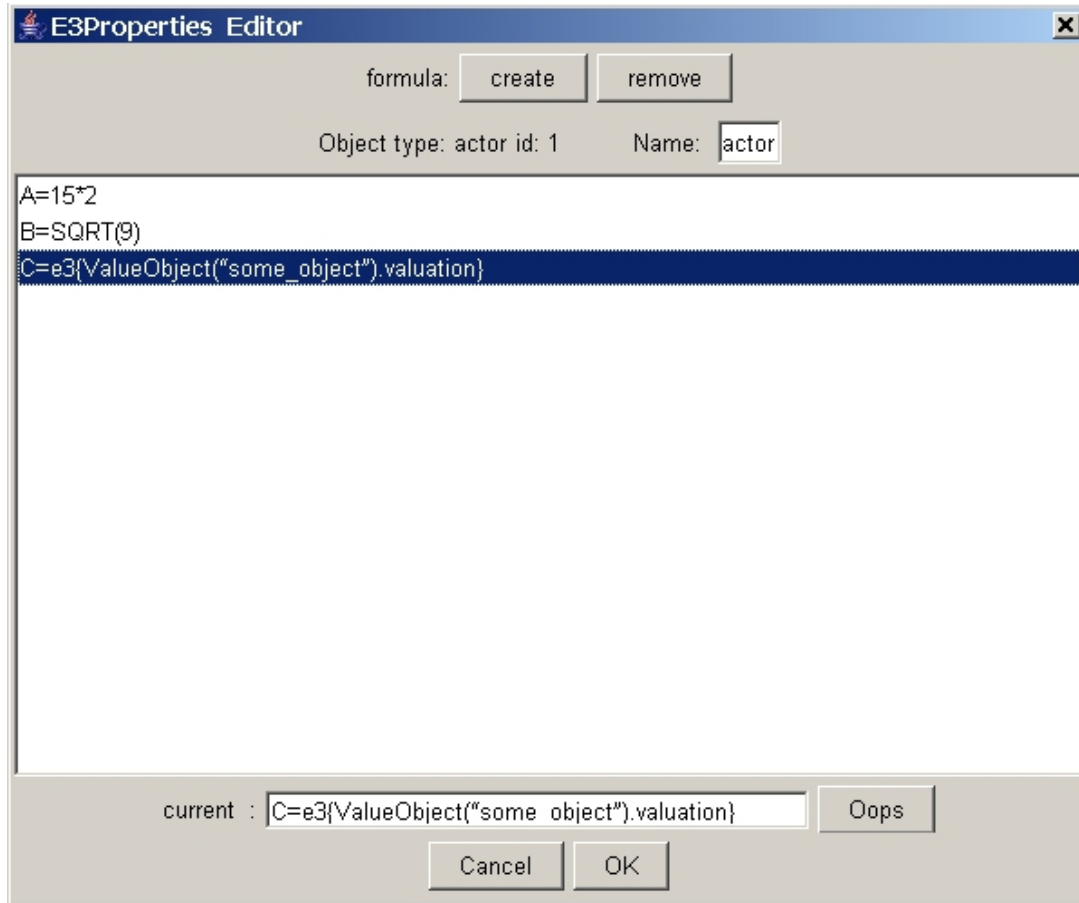


Figure 28. Assigning formulas to objects

Step 2. Now create a new formula by clicking 'create'. A new formula should appear in the list below. Highlight it and then enter the formula in the 'current' field at the bottom of the screen. A formula should always start with an identifier name, followed by a '=' symbol, and then the formula itself. Click 'OK' to save your formula.

Writing formulas

While the formulas are based on Excel, and you can use Excel expressions and syntax in the *e³-value* tool, there is one difference between *e³-value* formulas and Excel formulas. This difference lies in the way in which formulas can refer to other values (references).

Where, in Excel, you would typically refer to cells, in the *e³-value* tool you refer to attributes of *e³-value* objects.

To be able to refer to *e³-value* objects and their attributes we use a simple language that consists of a small number of simple syntax rules.

To insert these references into your Excel-style formulas just add 'e3 {REFERENCE}' into your formulas where you would normally write cell references in Excel. Of course, you

should make sure to replace 'REFERENCE' with a reference to an attribute.

Although the syntax counts quite a number of expressions, the structure itself is very simple and straightforward. Any reference can be written as:

$$'e3 \{TopLevelObject.SubLevelObject.AttributeName\}'$$

Reference expressions always start with 'e3 {', and always end with '}'

Object-references are separated by a '.'.

A top-level object is an object that has to be unique in the scope of a model. Actors, Market segments, Value Objects, all Scenario constructs and Value Interfaces are seen as top-level objects.

If the object is a sub-level object it can be accessed from another object that is on the level above it. This may be done multiple times if necessary, depending on the number of levels. However, you need to start always at a top-level object.

If the object is a top-level object the 'SubLevelObject' reference may be omitted.

The best way to give more insight into the way this works in practise is by giving a simple example. Say you want to refer to the valuation of a value-object with the name "x" (the value-object being the object and the valuation being the object's attribute.). You could write a simple formula that would look like this:

$$'Y=e3\{ValueObject("X").valuation\}'$$

Reserved formula names

A small number of formulas have been reserved for e^3 -value specific purposes:

- "COUNT"
- "FRACTION"
- "OCCURRENCES"
- "VALUATION"
- "NORM_VALUE"

If you use the names listed above as formula names (within the corresponding scope), your formulas will be used as if they were a corresponding e^3 -value attribute. For example, adding a formula named "OCCURRENCES" to a start stimulus will result in that formula being used to calculate the value of the occurrences attribute of that start stimulus.

For a full list of these names, including their respective descriptions, please refer to section 4.3

Formula limitations

Currently the *e³-value* tool does not support formulas containing *unary operators* (e.g. the '*minus*' symbol in the expression '-1'). This functionality will be added in future versions of the tool. For more detailed information please refer to section 4.5

2.1.12 Step 10. Saving your business model

Finally, do not forget to save the business model. Press the 'save' button in the menu bar or press CTRL-S. Thus a save dialog pops up if you are saving the diagram for the first time otherwise it will save to the same file you have chosen before. In all circumstances you should save your diagram to the xsvg format, which allows you to reload the diagram in exactly the same state. Besides you might consider saving to the jpg or svg formats which gives you a graphical complete but ontological incomplete representation of your model. Exporting to the RDF format gives an ontological complete but graphical incomplete representation and finally exporting to the xls format gives a financial representation of the business model. These last two formats will be discussed later on in detail.

2.2 How to create profitability sheets with the e3-value editor

2.2.1 Introduction

One of the main objectives of the tool is to produce profitability sheets to assess the viability of the business model. These sheets will be in the spreadsheet format .xls of Microsoft Excel. In this spreadsheet tool the sheets can be loaded, financial parameters can be changed and the financial outcomes of the business model can be inspected or processed further.

2.2.2 Profitability sheet generation

After you have created and saved your business model the e^3 -value tool can generate the profitability sheets.

In this paragraph, we show you a step-by-step demonstration on how to generate profitability sheets using the e^3 -value tool, and we will briefly discuss the structure of the profitability sheets produced in the form of Excel '98 documents.

Step 1. Assuming that you have started the e^3 -value tool and loaded a correctly modelled e^3 -value business model, the first step is to open a 'save as' dialog. Select 'save as' from the 'File menu'.

Step 2. Select '**Microsoft** Excel export only' from the 'Files of type' drop-down box:

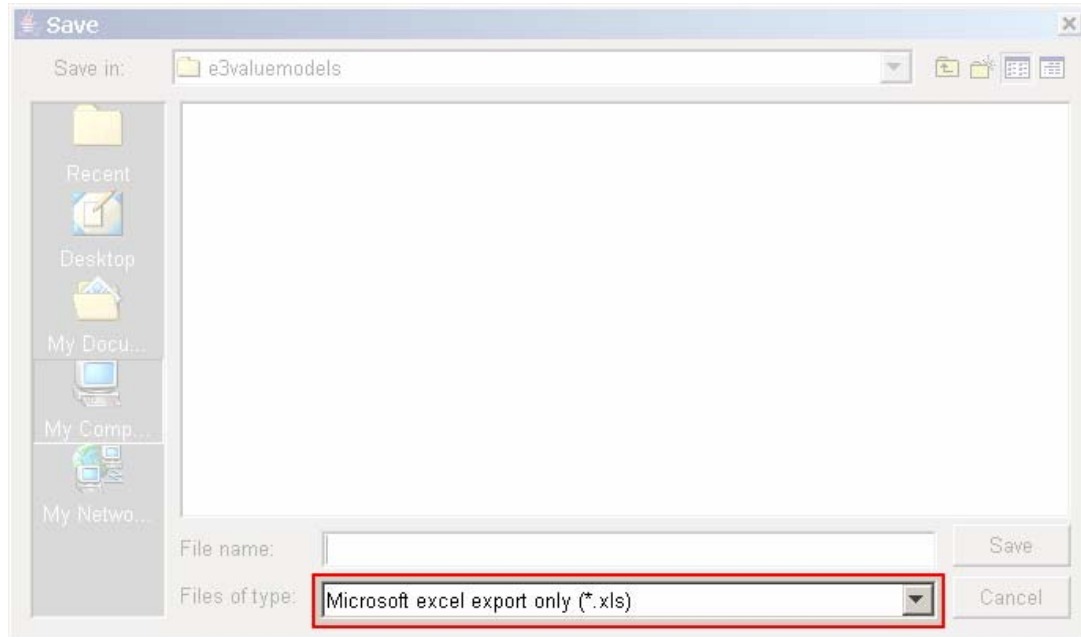


Figure 29 Save as dialog - export profitability sheets

Step 3. Using the 'save as' dialog, browse to the directory in which you want to store the profitability sheets, and enter a file name in the 'File name' text field. (Note: the filename should end with '.xls'). Then click 'Save'.

Step 4. User information dialogs.

During profitability sheet generation, the e^3 -value tool may display a number of information dialogs (depending on the model).

"Merged concepts" dialog. - If your model contains two or more graphical objects that conceptually represent the same object, a 'merged objects' dialog will be displayed, showing a list of graphical objects that have been merged for the profitability sheet generation. For more specific information please refer to the section 'Merging instances'.

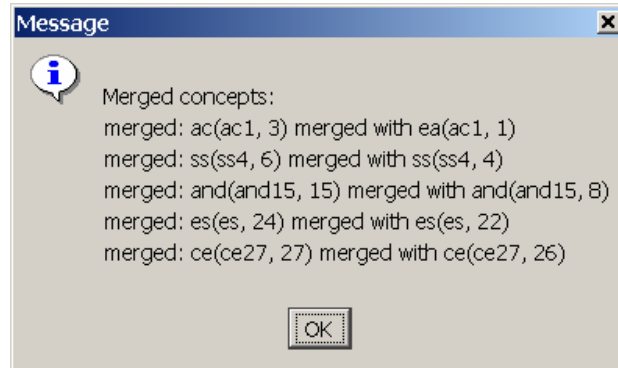


Figure 30 Merged concepts dialog example

This dialog will only be displayed if there is relevant information to show – e.g. if your model does not contain any objects that need to be merged this dialog will not be displayed.

“Error message” dialog. - At this point, it is possible that you are notified of an error in the generation process through an error message dialog:

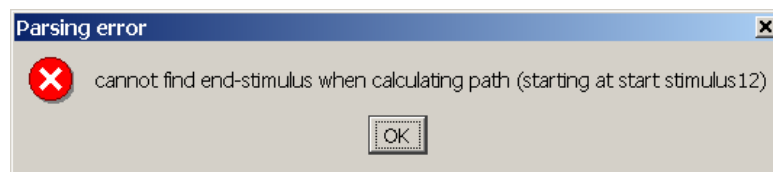


Figure 31 Error message dialog example

When generating the profitability sheets, the e^3 -value tool checks your model against a number of rules. If it detects any conflicts or errors in the model, it will notify the user by displaying a specific error message.

If this dialog appears, you should fix the model (see section 4.2 ‘Error Messages’ for more detailed information on specific error messages) and repeat Step 1.

“Unseen objects” dialog. – This dialog displays a list of object instances that have not been ‘seen’ during generation, meaning that the object is not involved in any scenario paths.

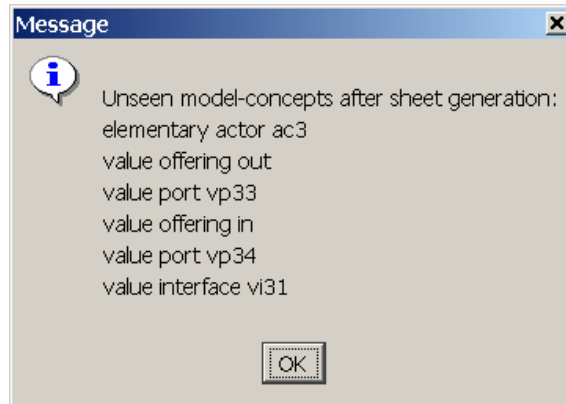


Figure 32 “Unseen objects” dialog example

If an object appears in this dialog it is possible (but not necessary) that the model is flawed, or that it contains object that have no real purpose. If this dialog appears, you are advised to check the model for errors.

Step 6. The spreadsheet has been created and saved. The .xls file can be opened and viewed in Microsoft Excel ('98 or later).

2.2.3 Profitability sheet document structure

In this paragraph we will discuss the general structure and layout of the profitability sheets generated by the e^3 -value tool. Please note that this structure may change slightly with the release of future versions of the tool.

The Excel documents generated by the e^3 -value tool can be divided into three sections, each section consisting of one or more.

- Formula sheet
- Model concept sheets
- Actor / Value Activity / Market Segment sheets

Each of these sections will be discussed individually.

Formula sheet

This is a single Excel sheet that contains a list of all object instances and their respective attributes and formulas that are in the model. The formula sheet is purely used as a storage place for attributes and formulas.

Model concept sheets

A separate sheet is created for each e^3 -value class (e.g. 'value interface') of which instances exist in the model. Each of these sheets contains an overview of all instances of its particular class, and of course the respective attributes and formulas for each of these instances.

Actor / Value Activity / Market Segment sheets

A separate sheet is created for each instance of the following classes:

- Elementary actor
- Composite actor
- Market Segment
- Value Activity

The sheet-layout is the same for all four types of sheets.

	A	B	C	D	E	F
1	value_interfaces	value_ports	value_exchanges	OCCURRENCES	valuation	economic value
2	vi15	total for vi15		66.66666667	1	66.66666667
3		vp17	ve36	66.66666667	1	66.66666667
4		vp18	(all connected)	66.66666667		0
5	vi23	total for vi23		33.33333333	1	33.33333333
6		vp26	(all connected)	33.33333333		0
7		vp25	ve33	33.33333333	1	33.33333333
8						
9	total for actor			100	2	100

Figure 33 Actor-sheet example

The sheets are structured as follows:

- Column A – List of value interfaces for this actor.
- Column B – Value ports for each interface (total for this value interface is shown at the top).
- Column C – Value exchanges for each value port
- Column D – Number of occurrences for each port / exchange-combination.
- Column E – The corresponding valuation function.
- Column F – Economic value (valuation multiplied by the number of occurrences).

Each sheet contains an overview of the occurrences, valuation, and economic value of its corresponding object instance. These values are grouped per value interface (column A in the example) and aggregated at the bottom of the sheet (Row 9 in the example)

Note: In case a value port offers or requests a value object with the name 'MONEY' the number of occurrences, valuation function and economic value fields will appear for each combination of value ports and value exchanges. (Because the valuation function might be different for each individual combination)

If a value port offers or requests a value object of a type *other* than 'MONEY', only one row will appear in the spreadsheet, displaying the number occurrences, valuation function and economic value for that port and 'all it's connected' value exchanges (Because the same valuation applies to all port/exchange-combinations if the value object is not 'MONEY

Chapter 3 E3-value editor technical reference

In this chapter we will describe the process of generating various outputs from the (graphical) business models created by the user.

3.1 RDF generation steps

In this section, we will describe the process of generating an RDF representation from the diagrams, which are drawn in the JGo environment by the user.

In order to be able to represent diagrams in the RDF format we use a set of Java classes, which mirror the *e³-value* ontology. These classes (found in the '*www.cs.vu.nl.gordijn.e3value.ontology*' package) were first generated from the original RDFS schema using the RDF2Java tool. The process of exporting diagrams comes down to mirroring the instances of objects created by the user (using the editor GUI) as instances of the 'RDF' classes. It can be broken down to the following steps.

1. **Create instances** of the 'RDF'-classes for each relevant model concept in the diagram.
2. **Assign attributes** (including references to other instances) to each of these newly created instances.

Each of these steps is discussed in its corresponding paragraphs below.

3.1.1 Create instances

In this paragraph, we will discuss the process of instantiating objects of the 'RDF2Java' classes that will be used to generate RDF files later on.

3.1.1.1 Simple mapping

The first step in the process of generating the RDF is to instantiate an 'RDF2Java' counterpart for every single instance that exists in the JGo (user interface) world. In addition, there are model concepts that are not explicitly represented in the user interface that we do want to include in the final RDF representation (namely 'value offerings'). These will also be instantiated. We start by iterating the 'JGo' instances in the diagram.

For each instance we determine the class type, which is mapped to its corresponding 'RDF' class. A new instance of the corresponding 'RDF' class is created¹.

¹ in some cases model concepts are *merged*, in which case there are a number of exceptions regarding instantiation of the respective classes. Please refer to the paragraph 'merging instances' for more information.

If the model concept's class type is that of 'value_interface', two additional instances of the 'RDF' class 'value_offering' will be created ², one for each of the value interface's implicit value offerings.

All newly created instances are stored in a HashMap where the *key* is a string containing the unique ID from the 'JGo' instance, and the corresponding *value* is a reference to the 'RDF' instance. This is done so we can look the instances up later on.

3.1.1.2 Merging objects

Besides the straight-forward mapping of 'JGo' instances to 'RDF2Java' instances where one 'JGo' instance corresponds with one 'RDF2Java' counterpart, there may also be cases where the user creates two (or more) objects which conceptually represent the same model concept, but are *practically* two different instances in the 'JGo' world. For example:

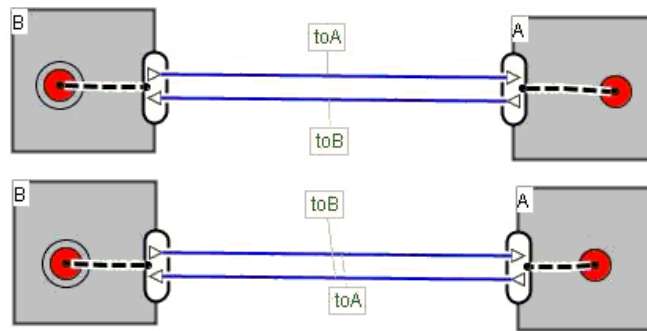


Figure 34 Object-merging: Two graphical objects may conceptually be identical.

In the example (Figure 34) you can see that it is possible to use the user interface to model the same model concept more than once. Obviously, when exporting to RDF, we would want to see each model concept represented only once. This means we need some mechanism to detect 'JGo' objects that represent the same model concept, and merge them (if possible) into one 'RDF' instance.

This mechanism is implemented during the process of instantiating the 'RDF' model concepts. See below for a step-by-step guide to the way this mechanism works:

1. For each 'JGo' instance *j*:
 - a. Check whether we have already created one or more 'RDF' instance *r* where
 $(r.name = j.name \text{ AND NOT } r.name = "")$
 AND $(r.class \text{ matches } j.class)$

² Note: By matching the class types, we mean that both class types should *correspond to each other*, NOT that they should be of the exact same type. For example, a 'JGo' instance of class "E3tor_value_exchange" would be considered to match any 'RDF' instance of class 'value exchange' because they represent the same type of model concept.

- If false, proceed to step 3, otherwise proceed to 1b.
- b. for each r , check whether it is allowed to merge with j based on a set of rules³. If false: abort export process and throw an error message.
 2. Instead of making a new 'RDF' instance to mirror j , make a new entry in the HashMap where the *key* is a string containing the unique ID from instance j , but the value is a reference to instance r .
 3. Create a new 'RDF' instance and proceed as usual.

If the 'RDF' instance for a certain model concept needs to be 'merged' with another concept, and no conflicts are detected, we will add a new entry to the HashMap, but instead of creating a new instance, we add a reference to the existing instance. Effectively the instances are now 'merged'. The reason the duplicate pointers are added in the table is because later on, when assigning attributes to the instances, we will need to be able to look up which instance represents which model concept.

	'RDF' instance
#1	Reference to actor_a
#2	Reference to value_exchange_toB
#3	Reference to actor_a

Figure 35 - some merged instances in a Map

3.1.1.3 Rules for merging model concepts

When two 'JGo' objects are considered for merger, there is a number of constraints to be considered.

1. Both objects must have the same name (excluding "" – empty string)
2. Both objects must be of a same class type.
3. Both objects should be in the same *scope*.

The scopes of both objects should be equal, but can depend on the class type. For example: For some classes it may be sufficient if both instances are in the global scope (which is always true per definition).

If one or more of these criteria are **not** met, the two model concepts will simply not be merged. A new instance will be created for each of them. If they **are** met, another number of criteria is evaluated. While these criteria are specific for each class type, they can be divided into two categories.

1. The collections of formulas of both objects must be merge-able – if two objects have one or more formulas with the same name (header), the formulas' bodies must be equal.
2. The attributes of both objects should be equal. This includes fractions, but also references to other objects (in which case both the name and class type should be equal).

³ See section: 'rules for merging model concepts' (section 3.1.1.3)

A specific list of criteria (per class):

Market Segment	Merge if the same name is used by any other MS and formulas are merge-able.
Actor	Merge if: same name is used by another Actor and both Actors' parents must ((Be null, or have the same name, or be of the same class)) and (parent must be of class el_ac, comp_ac, ms, activity) and formulas must be merge-able.
ValueObject	Merge if same name is used by any other V.O. Formulas must be merge-able.
SimpleDependencyElement	Merge if same name is used by any other simple D.E. of the same class, both share attributes of the same name and their formulas are merge-able.
ComplexDependencyElement (except ValueInterface)	Merge if same name used by any other D.E. of the same class
ConnectionElement	Merge if same name used by any other C.E. if both share the same characteristics - names of the objects in the following fields must be equal: up_de, down_de. In addition, the formulas must be merge-able.
Value Activity in scope A.	Merge if same name used by another Activity and both Activities' parents must ((be null or have the same name, and be of the same class)) (Parent must be of class el_ac, comp_ac, ms, activity) and formulas must be merge-able.
Value Activity in scope M.S.	Merge if same name used by another Activity and both Activities' parents must ((be null or have the same name, and be of the same class)) (parent must be of class el_ac, comp_ac, ms, activity) And formulas must be merge-able.
Value Offering in scope V.I.	Not applicable
Value Port in Scope V.O.	Merge if same name used by another V.P. which shares the same parent (of class V.I.) and is equally directed.
Value Exchange in Scope V.P.	Merge if same name used by another V.E. which is connected to the same ports (both in and out)

Value Interface	Merge if same name used by any other V.I. and formulas are merge-able
Value Transaction	Merge if same name used by any other V.T. and formulas are merge-able

If the first three criteria (same name, class type, and scope) are met, but one of more of the criteria in the above list is *not*, then the export process will be aborted and an error message will be displayed.

3.1.2 Assign attributes

After all 'RDF instances' have been created, we assign the correct attributes (and object references) to each instance. This is done by iterating over the collection of all (JGo) objects that exist in the document of the graphical business model.

1. For each 'JGo' object instance '*j*':
 - a. Look up the corresponding 'RDF' object instance '*r*' from the HashMap.
 - b. Invoke the 'initialize(E3Object)' method on object '*r*', where '*j*' is passed as the argument variable – e.g.: '*r.initialize(j)*'

Each of the 'RDF' classes has an 'initialize' method, which, if invoked, collects all relevant attributes and references of the 'JGo' object passed as an argument.

Attributes are directly copied and stored in the 'RDF' object.

References to other 'JGo' objects are replaced (with a reference to the corresponding 'RDF' object) before they are stored. 'JGo' objects that have been 'merged' into a single 'RDF' counterpart both have their own unique 'key' pointers as entries in the HashMap, where each of these are pointing to the same 'RDF' instance. This means that the attributes and references of both 'JGo' objects will be assigned to the same 'RDF' object, effectively merging them.

3.1.3 Generate RDF files / file stream

The actual RDF generation is done by the RDF2Java API, which has a built-in function to generate RDF files directly from the 'RDF' object instances we created earlier. It uses the Java reflection API directly serialize and de-serialize Java objects, and produces RDF output which can then be read using various RDF tools / analyzers (e.g. OntoServer).

3.2 Profitability sheet generation

3.2.1 Approach

To be able to generate profitability sheets we assume that we have a conceptual representation of the business model in the form of a set of 'RDF2Java' objects (instances of the 'RDF2Java' classes, which were created by exporting the graphical representation of the business model to RDF: **Error! Reference source not found.**

3.2.2 Profitability sheet generation steps

Profitability sheet generation takes place through the following steps:

1. Assign the number of occurrences to each value port and each value transaction in the model.
2. Parse all formulas
3. Create a 'Formula Sheet' to which we write all formulas (or placeholders) that are present in the model.
4. Assign 'Default Valuation Formulas' to each value port
5. Update the Formula sheet with 'Default Valuation Formulas'
6. Create Excel sheets for each e^3 -value ontology class
7. Create Excel (profitability) sheets for instance of the Actor, Market Segment, and Value Activity classes.

Each of these steps is discussed below.

3.2.3 Assign occurrences to the value ports and the value transactions.

The first step in calculating the economical value of a value port is to determine the number of times its transactions are executed. This is done by taking the number of occurrences assigned to each start stimulus and propagating it across the model through the scenario paths.

Below you will find a highly simplified representation of the algorithm used to propagate the occurrences throughout the business model. The purpose of this paragraph is not to give a fully detailed explanation of the process, but instead to give an impression of the mechanism that is used.

The syntax that was used is an informal mixture of Java syntax and pseudo-code. Some of the method names may not literally correspond to the actual Java code. This was done for readability purposes. The algorithm described below also assumes that the business model is correctly modelled. In reality, of course, it contains code which actually checks the integrity of the model. These parts of the code were however omitted here, for the sake of readability.

Please see the next page for a detailed overview.

AssignOccurrences{

```
    For each STARTSTIMULUS {

        Pathfound = TraverseDependencyElement (STARTSTIMULUS ,STARTSTIMULUS.occurrences)

        //the TraverseSimpleDependencyElement method returns a boolean value. If this value is
        //false this means that there was no complete scenariopath and no ENDSTIMULUS to
        //be found, in which case an error is thrown.

        IF NOT Pathfound throw ERROR ("no end stimulus reached").
    }
}
```

TraverseSimpleDependencyElement(Element, occurrences){

```
    //This method takes handles the propagation of occurrences for simple dependency
    //elements: STARTSTIMULUS, and ENDSTIMULUS

    Returnvalue = FALSE // false by default

    IF Element HAS BEEN VISITED throw new ERROR ("loop detected")
    ELSE ELEMENT.setVisited(TRUE);

    IF Element ISA STARTSTIMULUS{
        NEXTELEMENT = ELEMENT.getNextElement();
        IF NEXTELEMENT ISA ANDFORK {
            Returnvalue =
            TraverseAND (NEXTELEMENT, occurrences, ELEMENT)
        }
    }
```

```
IF NEXTELEMENT ISA ORFORK {  
    Returnvalue =  
    TraverseOR (NEXTELEMENT, occurrences)  
}  
  
IF NEXTELEMENT ISA SIMPLEDEPENDENCYELEMENT{  
    Returnvalue =  
    TraverseSimpleDependencyElement(NEXTELEMENT, occurrences)  
}  
  
IF NEXTELEMENT ISA VALUEINTERFACE{  
    Returnvalue =  
    TraverseValueInterface (NEXTELEMENT, myOccurences, ELEMENT, Null)  
}  
  
//Set visited value to be false so the element may be visited again (in the context //of a different  
//scenariopath.  
ELEMENT.setVisited(FALSE);  
  
//We are returning the returnvalue of the just executed 'traverse methods'. This //makes sure that in the  
//event an end stimulus has been found to complete the //scenariopath, it is recursively passed along.  
  
RETURN Returnvalue  
}  
  
IF Element ISA ENDSTIMULUS{  
  
    Element.SetOccurences(occurrences)  
  
    //We are returning the value 'true' to signal that the scenariopath  
  
    //has been completed  
  
    RETURN TRUE;
```

```
}  
}
```

TraverseAND (Element, occurrences, SourceElement){

//This method takes handles the propagation of occurrences for AND forks

Returnvalue = TRUE // true by default

IF Element HAS BEEN VISITED throw new ERROR ("loop detected")

ELSE ELEMENT.setVisited(TRUE);

Element.SetOccurences(occurrences)

Collection NextElements = ELEMENT.getDownConnectedElements()

IF ELEMENT IS_DIRECTED_AS_AND-SPLIT{

For each Element in NextElements{

myOccurences = Occurences

IF NEXTELEMENT ISA SIMPLEDEPENDENCYELEMENT{

*tempReturnvalue = TraverseSimpleDependencyElement(NEXTELEMENT,
myOccurences)*

IF tempReturnvalue IS FALSE THEN Returnvalue = FALSE

}

IF NEXTELEMENT ISA ANDFORK{

```
        tempReturnvalue =
        TraverseAND (NEXTELEMENT, occurrences, ELEMENT)
        IF tempReturnvalue IS FALSE THEN Returnvalue = FALSE
    }

    IF NEXTELEMENT ISA ORFORK{
        tempReturnvalue =
        TraverseOR (NEXTELEMENT,myOccurrences)
        IF tempReturnvalue IS FALSE THEN Returnvalue = FALSE
    }

    IF NEXTELEMENT ISA VALUEINTERFACE{
        tempReturnvalue =
        TraverseValueInterface (NEXTELEMENT, myOccurences, ELEMENT, Null)
    }

    //Set visited value to be false so the element may be visited again (in the context
    //of a different scenariopath.
    ELEMENT.setVisited(FALSE);

    //We are returning the combined returnvalue of the just executed 'traverse
    //methods'. This makes sure that in the event an end stimulus has been found to
    //complete the scenariopath, it is recursively passed along. If even one of them
    //returns false our returnvalue should also be false.

    RETURN Returnvalue
}
}

ELSE IF ELEMENT IS_DIRECTED_AS_AND-MERGE{

    //remember which element was the source from which this fork was traversed //forthe very first time. Only
```

allow further traverse for the same source

```
IF FIRST_TOUCHED_BY_ELEMENT == NULL{  
FIRST_TOUCHED_BY_ELEMENT = Source_Element  
}  
IF FIRST_TOUCHED_BY_ELEMENT == Source_Element{
```

For each Element in NextElements{

myOccurrences = Occurrences

```
IF NEXTELEMENT ISA SIMPLEDEPENDENCYELEMENT{
```

```
tempReturnvalue = TraverseSimpleDependencyElement(NEXTELEMENT,  
myOccurrences)
```

```
IF tempReturnvalue IS FALSE THEN Returnvalue = FALSE
```

```
}
```

```
IF NEXTELEMENT ISA ANDFORK{
```

```
tempReturnvalue =
```

```
TraverseAND (NEXTELEMENT, occurrences, ELEMENT)
```

```
IF tempReturnvalue IS FALSE THEN Returnvalue = FALSE
```

```
}
```

```
IF NEXTELEMENT ISA ORFORK{
```

```
tempReturnvalue =
```

```
TraverseOR (NEXTELEMENT,myOccurrences)
```

```
IF tempReturnvalue IS FALSE THEN Returnvalue = FALSE
```

```
}
```



```
IF NEXTELEMENT ISA VALUEINTERFACE{
    tempReturnvalue =
        TraverseValueInterface (NEXTELEMENT, myOccurrences, ELEMENT, Null)
}

//Set visited value to be false so the element may be visited again (in the context
//of a different scenariopath.
ELEMENT.setVisited(FALSE);
}

//We are returning the combined returnvalue of the just executed 'traverse
//methods'. This makes sure that in the event an end stimulus has been found to
//complete the scenariopath, it is recursively passed along. If even one of them
//returns false our returnvalue should also be false.

If (NextElements.COUNT == 0) Returnvalue = false

RETURN Returnvalue
}
}
```

```
TraverseOR (Element, occurrences){
```

```
//This method takes handles the propagation of occurrences for OR forks
```

```
Returnvalue = TRUE // true by default
```

```
IF Element HAS BEEN VISITED throw new ERROR ("loop detected")

ELSE ELEMENT.setVisited(TRUE);

Element.SetOccurences(occurrences)

Collection NextElements = ELEMENT.getDownConnectedElements()

For each Element in NextElements{

    //find the fraction of occurrences to send to this element by
    // dividing the total of fraction (all elements in NextElement) by the fraction of NextElement

    myOccurences = Occurences * (NextElements.getTotalFraction() / NextElement .getMyFraction())

    IF NEXTELEMENT ISA SIMPLEDEPENDENCYELEMENT{

        tempReturnValue = TraverseSimpleDependencyElement(NEXTELEMENT, myOccurrences)

        IF tempReturnValue IS FALSE THEN ReturnValue = FALSE
    }

    IF NEXTELEMENT ISA ANDFORK{

        tempReturnValue =

        TraverseAND (NEXTELEMENT, occurrences, ELEMENT)

        IF tempReturnValue IS FALSE THEN ReturnValue = FALSE
    }

    IF NEXTELEMENT ISA ORFORK{

        tempReturnValue =

        TraverseOR (NEXTELEMENT,myOccurrences)

        IF tempReturnValue IS FALSE THEN ReturnValue = FALSE
    }
}
```

```
IF NEXTELEMENT ISA VALUEINTERFACE{
    tempReturnvalue =
        TraverseValueInterface (NEXTELEMENT, myOccurences, ELEMENT, Null)
    }
}

//Set visited value to be false so the element may be visited again (in the context
//of a different scenariopath.

ELEMENT.setVisited(FALSE);

//We are returning the combined returnvalue of the just executed 'traverse
//methods'. This makes sure that in the event an end stimulus has been found to
//complete the scenariopath, it is recursively passed along. If even one of them
//returns false our returnvalue should also be false.

If (NextElements.COUNT == 0) Returnvalue = false

RETURN Returnvalue
}
```

```
TraverseValueInterface (Element, occurrences, SourceElement, SourceTransaction){

//This method takes handles the propagation of occurrences for ValueInterfaces

Returnvalue = TRUE // true by default

IF Element HAS BEEN VISITED throw new ERROR ("loop detected" )
```

```
ELSE ELEMENT.setVisited(TRUE);

ELEMENT.SetOccurrences(occurrences)

// Find all valueExchanges that are connected to ELEMENT in such a way that they are admissible for traversal –
// meaning that depending on the direction in which they are attached to value ports (inward, outward, first, or second)
// ports are chosen, or they are rejected. This is done because we only want to traverse forward, not backwards.

Collection nextValueExchanges = findAdmissableValueExchanges( )

// Find all valueTransactions that contain valueExchanges which are contained in nextValueExchanges

For each element in nextValueExchanges{
    Collection NextTransactions. Add( exchange.getValueTransactions ( ))
}

// calculate totalFraction

For each element in nextTransactions{
    totalFraction = totalFraction + exchange.getFraction ( )
}

// remember via which sub transaction this value interface was traversed/for the very first time.

// Only allow further traverse for the same source. This mechanism allows value interfaces to work as AND-fork
merges.

ALLOW_TRAVERSE = false // default value

IF FIRST_TOUCHED_BY_VALUEINTERFACE == NULL AND FIRST_TOUCHED_VIA_TRANSACTION == NULL{
    FIRST_TOUCHED_BY_VALUEINTERFACE = Source_Element
```

```
        FIRST_TOUCHED_VIA_TRANSACTION = SourceTransaction
    }

    IF FIRST_TOUCHED_BY_ELEMENT == Source_Element AND FIRST_TOUCHED_VIA_TRANSACTION ==
SourceTransaction {

        ALLOW_TRAVERSE = TRUE;
    }

    IF ALLOW_TRAVERSE {

        // traverse

        For each element in nextTransactions{

            Collection myExchanges = Transaction.getExchanges( );

            For each element in myExchanges {

                // Ask the value exchange to give us the value interface which is on the other side (as opposed
                to ELEMENT)

                ValueInterface NEXT_VI = Exchange.getValueOtherExchange(ELEMENT)

                myOccurences = (totalFraction / Transaction.getFraction() ) * Occurences

                TraverseValueInterface (NEXT_VI, myOccurences, ELEMENT, Transaction)

            }

        }

    }

    //All Value interfaces that were connected via value exchanges have now been traversed.

    //Now it is time to traverse all dependency elements connected via connection elements.

    Collection NextElements = getDownConnectedDependencyElements( );
```

For each Element in NextElements{

myOccurrences = Occurrences

IF NEXTELEMENT ISA SIMPLEDEPENDENCYELEMENT{

tempReturnValue = TraverseSimpleDependencyElement(NEXTELEMENT, myOccurrences)

IF tempReturnValue IS FALSE THEN ReturnValue = FALSE

}

IF NEXTELEMENT ISA ANDFORK{

tempReturnValue = TraverseAND (NEXTELEMENT, occurrences, ELEMENT)

IF tempReturnValue IS FALSE THEN ReturnValue = FALSE

}

IF NEXTELEMENT ISA ORFORK{

tempReturnValue = TraverseOR (NEXTELEMENT,myOccurrences)

IF tempReturnValue IS FALSE THEN ReturnValue = FALSE

}

IF NEXTELEMENT ISA VALUEINTERFACE{

tempReturnValue = TraverseValueInterface (NEXTELEMENT, myOccurrences, ELEMENT, Null)

IF tempReturnValue IS FALSE THEN ReturnValue = FALSE

}

}

*//Set visited value to be false so the element may be visited again (in the context
//of a different scenariopath.*

ELEMENT.setVisited(FALSE);

*//We are returning the combined returnvalue of the just executed 'traverse
//methods'. This makes sure that in the event an end stimulus has been found to
//complete the scenariopath, it is recursively passed along. If even one of them
//returns false our returnvalue should also be false.*

If (NextElements.COUNT == 0 AND NextVis.COUNT == 0) Returnvalue = false

RETURN Returnvalue

}

3.2.4 Parse all formulas

The formulas are represented as strings in the graphical editor and have the structure of the formulas of Excel with the one exception that values can be represented by an e^3 -value formula. These e^3 -value formulas have a strict syntax. They start with 'e3{' and they end with '}'. The text between these fixed symbols consists of two parts. The first part represents an e^3 -value object in the model and the second part is either an ontology property of that object or a formula of that object. The first part resembles the structure of a package name as it consists of parts separated by dots allowing for navigation in a hierarchical structure. Two examples of e^3 -value formulas will clarify the structure.

- E3{ElementaryActor("L1").ActorInMarketSegment("Listeners").Count}
- E3{ValueExchangeHasInPort("MusicPort").Valuation}

The first formula starts with an absolute reference to an elementary actor named "L1" that must be present in the scope of the model. It is followed by a relative reference ActorInMarketSegment("Listeners") to a MarketSegment named "Listeners" to which "L1" must belong. Count is an ontological property of MarketSegment.

The second formula starts with a relative reference that only makes sense if this formula belongs to a value exchange. In that case it refers to its port named "MusicPort". So this second function represents the valuation function of this port. In the profitability sheets this formula will be replaced by a reference to the field containing the value of this valuation function. Only when the sheets are numerically processed a numerical value will be substituted for this e^3 -value formula.

In the parsing operation all e^3 -value formulas in the model are replaced by the reduced form E3{#UID.name}. Here UID is an integer that uniquely identifies the e^3 -value object in the model and name is the name of a function or of an ontological property of that object. The Gold parser, a third party package as described before, does this job. It needs to be fed before with the grammar we use in our formulas.

We traverse all e3 objects in the model. For each object we traverse all the formulas. Each formula is broken up in E3 formula parts and plane Excel parts. The E3 formula parts are reduced as just described and the Excel parts are kept unchanged. The resulting formula is stored in a Map belonging to the object.

3.2.5 Create a 'Formula Sheet' for all formulas (or placeholders).

The first step of actually creating Excel sheets is to create the so-called 'Formula Sheet'. The 'Formula Sheet' is a single sheet containing all formulas and attributes of all objects from the business model. The purpose of the 'Formula Sheet' is that it serves as a reference for all other sheets.

Initially the formulas are first 'dumped' onto the formula sheet in the form of literal strings. These 'dummy' strings are not the actual formulas and serve as nothing but placeholders.

Each time a formula- or attribute-string is dumped to the formula sheet, its coordinates on the sheet are remembered in the form of a cell reference.

After all formulas have been assigned a location (coordinates) on the Formula Sheet, we can iterate over all the dummy formulas and replace object references with the actual cell references that we have just assigned. For example, the expression

'e3 {SomeObject.AttributeName}'

Would be replaced by the expression:

'Formulasheet!A2'

The first run in which we wrote the dummy formulas to the spreadsheet is necessary because we need to assign the cell locations first, before we are able to reference them.

After the formula sheet has been created it is referenced by the other sheets. This prevents redundancy of information.

3.2.6 Assign 'Default Valuation Formulas' to each value port

It is of course possible to define a valuation function for each value port instance in the model, but this would be a lot of work, and it would also not be necessary in most cases. To reduce the number of valuation functions in the models the *e³-value* tool uses so called 'default valuation functions'. These functions are formulas that are automatically generated for each port in the model, and they are used for profitability sheet generation when the *e³-value* tool is unable to find a custom valuation for a port under consideration.

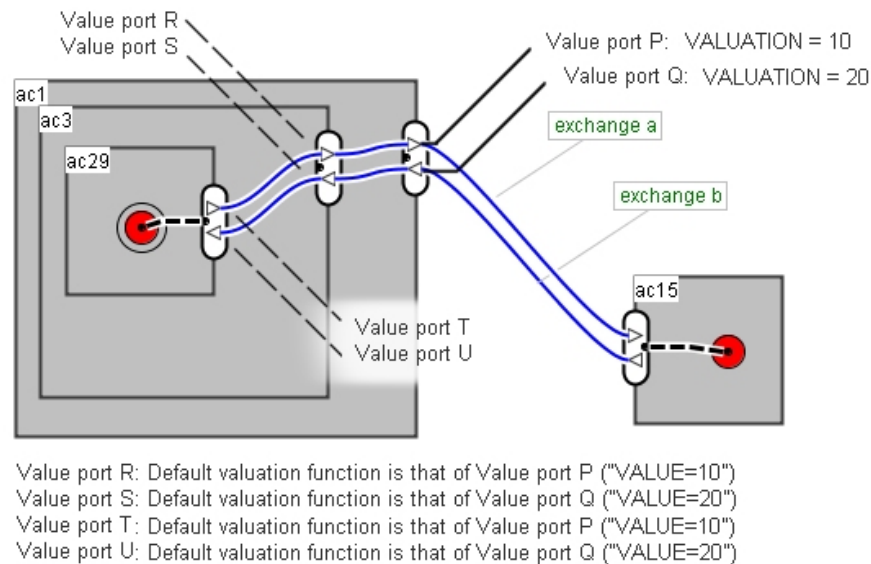


Figure 1 Example of default valuation formulas

“Default” valuation formulas are automatically generated by collecting the existing ‘VALUE’ formulas for all value ports that are part of a value interface which is attached to a ‘autonomous’ actor, activity or market segment. (IE: all actors, value activities, and market segments that connected via an *in-* or *out-connected* value exchange) These valuation formulas are then cascaded onto the other value ports in the model (the ones that only have *first-* or *second-connected* value exchanges). The default valuation functions are cascaded throughout the model as follows:

Value interfaces are divided into two categories: The ones that contain value ports, which connect value exchanges through the in-out mechanism⁴, and the ones that do not.

The value ports belonging to value interfaces of the first category are taken as a basis, and a starting point, from which the valuation functions are spread inwards across the rest of the model; onto the value ports, which belong to value interfaces that belong to the second category. For each value port ‘*p0*’ that exists in the business model and belongs to the first category:

1. For value port
 - 1.1. If the direction of ‘*p0*’ is inward, add all ports connected to ‘*p0*’ through ‘first connected’ value exchanges to the collection ‘*c_next*’ and add all ports connected to ‘*p0*’ through ‘second connected’ value exchanges to the collection ‘*c_previous*’

⁴ As opposed to the first/second mechanism

- 1.2. If the direction of '*p0*' is outward, add all ports connected to '*p0*' through 'second connected' value exchanges to the collection '*c_next*', and add all ports connected to '*p0*' through 'first connected' value exchanges to the collection '*c_previous*'.
- 1.3. If '*p0*' belongs to the first category
 - 1.3.1. '*p0.DEFAULT_VALUATIONFUNCTION = p0.find ValuationFunction()*'
- 1.4. else
 - 1.4.1. '*p0.DEFAULT_VALUATIONFUNCTION*' is the weighed average of the default valuation functions of the peer ports in '*c_previous*', where the weight is determined by number of occurrences of the peer port; for example: '*((peer_port1.defaultvaluationfunction * peer_port1.occurences) + (peer_port2.defaultvaluationfunction * peer_port2.occurences) + (...))/(peer_port2.occurences +peer_port2.occurences+...)*'
- 1.5. For each value port '*p1*' in '*c_next*'
 - 1.5.1. Execute step 1.

Note: All value ports of the first category are required to have a valuation function (either assigned to the port itself or, if the port exchanges a money object, via a bi-laterally agreed valuation function – where the valuation function is located on the value exchange, or via an in-out connected peer-port) If no valuation can be found for these value ports, an exception occurs and an error is thrown.

3.2.7 Update the Formula sheet with 'Default Valuation Formulas'

After default valuation formulas have been assigned to the value ports in the model, the formula sheet should be updated, replacing the default valuation function formula placeholders with the actual formulas. The reason that this step is separated from step 3.2.5 is that in order to be able to create the default valuation formulas it is required that for all value ports the coordinates of its default valuation function is known (even if it does not exist!). Therefore, adding 'dummy' formulas to the spreadsheet creates placeholders. These 'dummies' are later replaced by the actual formulas.

3.2.8 Create Excel sheets for each e³-value ontology class

For each e³-value class we create an individual sheet (provided the class is used in the business model). These sheets contain a list of all instances of that class, and their respective properties. This is done so that users will not have to bother searching through the entire formula sheet to look up a specific attribute value.

3.2.9 Create Excel (profitability) sheets for any Value Source instance.

For each instance of the aforementioned classes a separate excel sheet is created.

	A	B	C	D	E	F
1	value_interfaces	value_ports	value_exchanges	OCCURRENCES	valuation	economic value
2	vi15	total for vi15		66.66666667	1	66.66666667
3		vp17	ve36	66.66666667	1	66.66666667
4		vp18	(all connected)	66.66666667		0
5	vi23	total for vi23		33.33333333	1	33.33333333
6		vp26	(all connected)	33.33333333		0
7		vp25	ve33	33.33333333	1	33.33333333
8						
9	total for actor			100	2	100

Figure 2

These sheets are created by adding cells in the following way:

1. For each value interface '**a**' attached
 - 1.1. Create a new entry to column 'A'
 - 1.2. For each value port '**b**' in '**a**'
 - 1.2.1. Create a new entry to column 'B'
 - 1.2.2. In case a '**b**' exchanges a "MONEY" object
 - 1.2.2.1. For each value exchange '**c**' connected to '**b**'
 - 1.2.2.1.1. Create a new entry to column 'C'
 - 1.2.3. In case a value port exchanges other objects than a "MONEY" object:
 - 1.2.3.1. Create a single (generic / composite) entry to column 'C' representing all value exchanges connected to '**b**' since all of them share the same valuation function.

So, the follow columns are created (also see Figure 2)

- Column A – List of value interfaces for this actor.
- Column B – Value ports for each interface (total for this value interface is shown at the top).
- Column C – Value exchanges for each value port
- Column D – Number of occurrences for each port / exchange-combination.
- Column E – The corresponding valuation function.

- Column F – Economic value (valuation multiplied by the number of occurrences).

In the *e³-value* tool, valuation functions are represented as special formulas, and they can be attached to value ports or value exchanges. To create a valuation function for one of these objects you can simply create a new formula and name it “VALUATION”.

When generating profitability sheets from your models, the *e³-value* tool will attempt to decide which valuation should be used for which value ports, and the outcome of this decision depends on the type of value object that has been attached to each of the value ports. It makes a distinction between the following three possibilities:

- The port under consideration is attached to a value object of the type “MONEY”.
- The port under consideration is attached to a value object of another type.
- The port under consideration is not attached to a value object.

This factor plays a role in determining where the tool will search for a valuation function to assign to the value port.

This search is conducted using the following rules:

1. In case a value port exchanges a “MONEY” value object via a value exchange:

- 1.1. If a valuation formula is found for the value exchange under consideration connected to the port under consideration, it is used.
- 1.2. If a valuation formula is found for the port under consideration, it is used (the ordering of these two rules models that bi-literal price arrangements override uni-lateral, overall arrangements)
- 1.3. If a valuation formula is found for the “peer”-port connected to the port under consideration (via the value exchange under consideration), it is used.
- 1.4. If the value port under consideration belongs to a container (actor / value activity/market segment) which in turn belongs to another (composite) container, the *e³-value* tool will use a default valuation function (created by propagating the valuation functions of other ports nearby to the value port under consideration – See section:3.2.6 “Default valuation functions”)
- 1.5. If no valuation formula is found, this results in an error. The same holds if a different valuation formula is found on both ports.

2. In case a value port exchanges other objects than a “MONEY” object:

- 2.1. If a valuation formula for that port is specified it is used.

The value entered in the 'occurrences' field of each value exchange is derived from the fact that the number of occurrences of each value transaction in the model is known (see section 3.2.3). Therefore, we can find the number of occurrences for a value exchanges by taking the sum of the occurrences of all value transactions in which it is involved.

The economical value is calculated by multiplying the each of the occurrence values listed in column C with the corresponding valuation function from column D.

Finally, a new row is created at the bottom of the sheet containing the aggregated values of each of the columns to give an overview of the actual profitability of this actor.

Chapter 4 E3-value editor: tips, tricks and error messages

This chapter contains user documentation for the *e³-value* tool and is intended purely as a source for reference information.

4.1 *Import shortcuts and tricks*

- Sometimes you want to select a value port or a scenario port. Using the normal way by clicking the object the editor tries drawing a connection to another port. Now you have two options. The first is to let it happen and then to delete the unwanted connection. You are back where you started. The second option is to release the connection at some impossible place from any port. Your port is selected and your goal achieved.
- CTRL-Z will undo your previous action(s).
- Selecting a single value interface and clicking on another value source will connect all ports of the selected interface instantly to a new value interface with corresponding ports on the other value source. A value transaction is created with all new value exchanges.
- Selecting a value interface and pressing the home and end keys will move a value interface around the edge of its value source.
- Selecting a single value port or scenario port and pressing the up and down arrow will change that port's position in the value interface or scenario element. Pressing the del key or the backspace key will delete a port and select another.
- Selecting several value ports on one value interface and the pressing the right mouse button will allow you to split these ports of to a new value interface.
- Selecting several value interfaces and then pressing the right mouse button will allow you to join these value interfaces into one interface.

4.2 Error messages

In this paragraph, you will find a list of possible error messages that may be displayed by the e^3 -value tool during RDF-export and Profitability-sheet generation.

There are a few specific error messages that may be displayed by the e^3 -value tool because of incomplete or incorrect transaction modelling which you should look out for, namely:

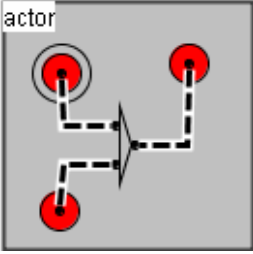
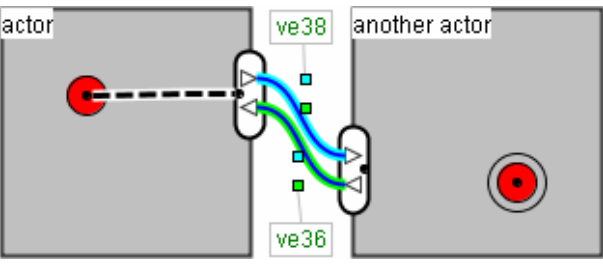
- **“Cannot find end-stimulus when calculating path”**
Indicates that some value exchanges may not have been bundled into value transactions.
- **“Value interface not modelled correctly: Each connected transaction should connect every of it’s ports via an exchange”**
- **“Value port is involved in a value transaction (more than once)”**
- **“Value port not involved in value transaction”**

Error messages related to modelling (visual)

Message:	“AND-port has not been connected correctly (possible loop)”
Explanation:	See: “Object is involved in a loop.”
Screenshot:	N / A

Message:	“At least one start stimulus required”
Explanation:	When generating profitability sheets it is required that one (or more) start-stimuli are included in the model.
Screenshot:	N / A

Message:	“Both ports connected by a value exchange should have the same value object.”
Explanation:	Both ports connected by the same value exchange should be assigned to the same value object.
Screenshot:	N / A

<p>Message:</p>	<p>“Can not connect start-/end-stimulus as 'down' / 'up'”</p>
<p>Explanation:</p>	<p>A collision of scenario paths has been detected. Make sure that each possible scenario-path is connected to a start stimulus on one end and an end stimulus on the other.</p>
<p>Screenshot:</p>	 <p>The screenshot shows a diagram with a grey background. On the left, there is a box labeled 'actor' containing three red circular stimuli. On the right, there is another red circular stimulus. Dashed black lines represent paths connecting the stimuli. A central point where two paths cross is highlighted with a black arrow, indicating a collision.</p>
<p>Message:</p>	<p>“Cannot find end-stimulus when calculating path” (starting at start stimulus [ID])”</p>
<p>Explanation:</p>	<p>There exists one or more scenario paths that cannot be traced to an end-stimulus. Check if there are end-stimuli and whether they are connected to said start stimulus through a scenario path. This error may also occur if the value transactions in the diagram are not chosen / assigned correctly.</p>
<p>Screenshot:</p>	 <p>The screenshot shows a diagram with two grey boxes. The left box is labeled 'actor' and contains a red circular stimulus. The right box is labeled 'another actor' and contains a red circular stimulus. Between the boxes, there are two vertical oval shapes representing value transactions, labeled 've38' and 've36'. A dashed black line connects the stimulus in the 'actor' box to the 've38' transaction. A solid blue line connects 've38' to 've36', and a solid green line connects 've36' to the stimulus in the 'another actor' box. The paths are not fully connected, illustrating the error.</p>

<p>Screenshot:</p>		<p>Note: though the scenario path may be connected correctly, the connection made by the two value exchanges in the example to the left will not be recognised as such unless both exchanges are bundled through a value transaction.</p>
--------------------	--	---

<p>Message:</p>	<p>“AND has an OCCURRENCES-Merge error”</p>
<p>Explanation:</p>	<p>Occurrences of incoming paths involved in an and-merge are not equal (note: an error margin of 1% is taken into account)</p>

<p>Screenshot:</p>	
--------------------	--

<p>Message:</p>	<p>“Value interface has an OCCURRENCES-Merge error”</p>
<p>Explanation:</p>	<p>Occurrences of incoming paths involved in an “and-merge” are not equal (note: an error margin of 1% is taken into account)</p>

<p>Screenshot:</p>	
--------------------	--

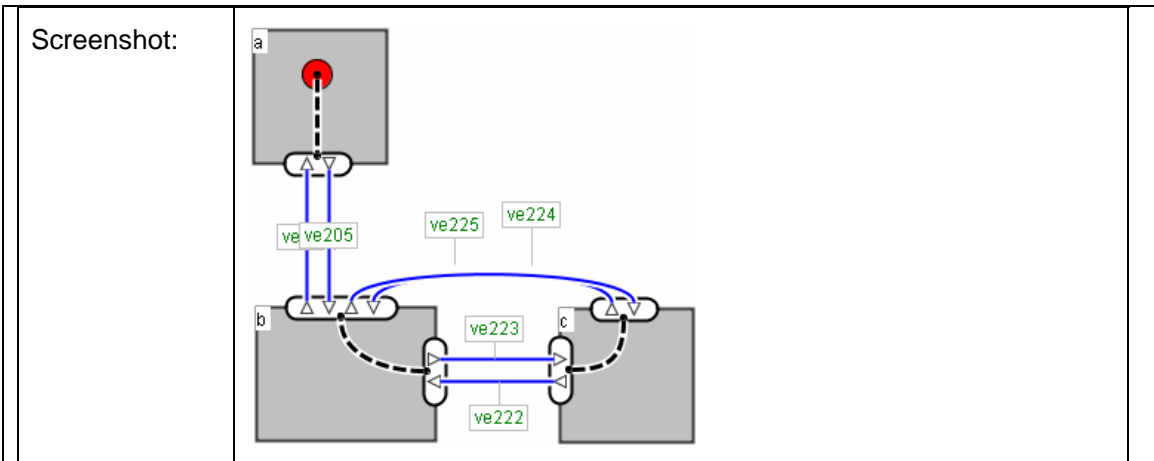
<p>Message:</p>	<p>“Value interface not modelled correctly: Each connected transaction should connect every of it’s ports via an exchange”</p>
-----------------	--

<p>Explanation:</p>	<p>Make sure that all value transactions have been correctly assigned to the value exchanges in the model.</p>
---------------------	--

<p>Screenshot:</p>	
--------------------	--

<p>Message:</p>	<p>“Object is involved in a loop.”</p>
-----------------	--

<p>Explanation:</p>	<p>A scenario-path loop has been detected.</p>
---------------------	--



Message:	“Value port is involved in a value transaction (more than once)”
Explanation:	Value ports are only allowed to be involved in the same value transaction once.
Screenshot:	N / A

Message:	“Value port not involved in value transaction”
Explanation:	If a value port is involved in a value transaction, so should the other ports in that value interface.

Screenshot:

Error messages related to formulas

Message:	“Grammar not loaded, can not continue.”
Explanation:	The formula.cgt file needs to be in the <i>e³-value</i> tool install directory.
Message:	“Invalid COUNT defined on market segment [ID] ”
Explanation:	An invalid value has been entered for the formula named “COUNT” for market segment [ID] .An Integer is expected.
Message:	“Invalid down-attribute for connection element [ID] ”
Explanation:	An invalid value has been entered for the down-attribute for connection element [ID] .An Integer is expected.
Message:	“Invalid up-attribute for connection element [ID] ”
Explanation:	An invalid value has been entered for the up-attribute for connection element [ID] .An Integer is expected.
Message:	“Invalid OCCURRENCES defined on start stimulus [ID] ”
Explanation:	An invalid value has been entered for the formula named “OCCURRENCES” for start stimulus [ID] .An Integer is expected.
Message:	“Missing ”}” after “e3{” in formula [f] of the class [c] named [n] ;
Explanation:	When using object references in formulas, always close the ‘e3{’ tag with a ‘}’.
Message:	“No COUNT defined for market segment [ID] ”
Explanation:	Every market segment is required to have a formula named “COUNT”.
Message:	“No OCCURRENCES defined for start stimulus [ID] ”
Explanation:	Every start stimulus is required to have a formula named

	"OCCURRENCES".
Message:	No model concept with [ID] found in model.
Message:	Not found globally: a [Object] named: [Name]
Explanation:	A formula is referring to an object which does not exist in the model.
Message:	"No 'FRACTION' formula found for value transaction [ID] "
Explanation:	No "FRACTION" formula has been found for this transaction. This formula is required.
Message:	"No valuation function found for value_port [ID] "
Explanation:	No "VALUATION" function found when dealing with a port that requests or offers a 'MONEY' object. Please see the section "Valuation Functions" for more details.
Message:	"Parse error"
Explanation:	There is a syntax error in one of the formulas entered,
Message:	"Syntax error in formula of [Object] "
Explanation:	Syntax error detected.
Message:	"Syntax error (multiple "=" symbols in formula)"
Explanation:	Syntax error detected; Only one "=" symbol per formula allowed.
Message:	"Unary operator used - please see user documentation for guidelines on writing formulas"
Explanation:	The current version relies on a third party software package (Apache POI / HSSF) for spreadsheet generation, which does not support unary operators such as the "-" operator in the expression "-x". A work-

	around for this problem is to rewrite the expression as "(0-x)". This functionality will be added later. See section 'Writing formulas' for more information.
--	---

Table 1 Error message overview

4.3 Formulas: Reserved names

A small number of names have been reserved for e^3 -value -specific attributes:

Formula name		
COUNT	Attribute:	“Count” attribute
	Scope:	Used as an attribute of market segments.
	Constraints:	Formula body must be an integer.
FRACTION	Attribute:	“Fraction” attribute
	Scope:	Used as an attribute of value transactions
	Constraints:	Formula body must be an integer.
OCCURRENCES	Attribute:	“Occurrences” attribute
	Scope:	Used as an attribute of start stimuli
	Constraints:	Formula body must be an integer.
VALUATION	Attribute:	Valuation function
	Scope:	Used as a valuation function if formula is attached to a value port or a value exchange.
	Constraints:	Formula body must be an integer.

NORM_VALUE	Attribute:	Valuation function
	Scope:	Used as a 'default' valuation function if no user-specified valuation function can be found.
	Constraints:	This formula is reserved for internal purposes only. Users are not allowed to specify custom values.

Table 2 Reserved formula names

Note that these names should be spelled exactly as above. They are case-sensitive!

4.4 Formula Syntax

Below is a list of expressions (and syntax) ordered by categories (top-level object references, sub-level references and a selection of attribute references)

Top-level object reference
MarketSegment("object_name")
ElementaryActor("object_name")
CompositeActor("object_name")
ValueObject("object_name")
ConnectionElement("object_name")
DependencyElement("object_name")
ValueInterface("object_name")

Table 3 Writing formulas - top level object reference syntax

Sub-level object reference
ActorHasValueInterface("object_name")
ActorInMarketSegment("object_name")
ElementaryActorPerformsValueActivity("object_name")
CompositeActorConsistsOfValueInterface("object_name")
ConnectionElementHasDownDependencyElement("object_name")
ConnectionElementHasUpDependencyElement("object_name")
DependencyElementWithDownConnectionElement("object_name")
DependencyElementWithUpConnectionElement("object_name")
MarketSegmentConsistsOfActor("object_name")
MarketSegmentHasValueInterface("object_name")
ValueActivityHasValueInterface("object_name")
ValueActivityPerformedByElementaryActor("object_name")
ValueExchangeHasFirstValuePort("object_name")
ValueExchangeHasSecondValuePort("object_name")
ValueExchangeHasInPort("object_name")
ValueExchangeHasOutPort("object_name")
ValueExchangeInValueTransaction("object_name")
ValueInterfaceAssignedToMarketSegment("object_name")
ValueInterfaceAssignedToValueActivity("object_name")
ValueInterfaceConsistsOfOffering("object_name")
ValueOfferingConsistsOfValuePort("object_name")
ValueOfferingInValueInterface("object_name")
ValuePortFirstConnectsToValueExchange("object_name")

ValuePortInConnectsToValueExchange("object_name")
ValuePortInValueOffering("object_name")
ValuePortOutConnectsToValueExchange("object_name")
ValuePortRequestsOrOffersValueObject("object_name")
ValuePortSecondConnectsToValueExchange("object_name")
ValueTransactionConsistsOfValueExchange("object_name")

Table 4 Writing formulas - sub-level object reference syntax

Attribute reference	Applicable when Parent Object is a:
Name	e3object
UID	e3object
UpFraction	ConnectionElement
DownFraction	ConnectionElement
FirstFraction	ConnectionElement
SecondFraction	ConnectionElement
InFraction	ConnectionElement
OutFraction	ConnectionElement
Direction	Port
Valuation	Port or ValueExchange
Fraction	Transaction

Table 5 Writing formulas - Attribute reference syntax

4.5 Formulas: Formula limitations

The current version of the tool relies on a third party API - Jakarta POI / HSSF - for spreadsheet generation. This API currently does not support unary operators, e.g.: the "-" operator in the expression "-x". Because of this, it is currently not possible to use such expressions when writing formulas in the tool. A work-around for this problem is to rewrite the expression as "(0-x)".

HSSF is expected to support unary operators in the near future, and this feature will be integrated in the tool as soon as possible.

4.6 Merging Objects

In the e^3 -value tool it is possible to model multiple (graphical) objects so that they are representations of the same e^3 -value object.

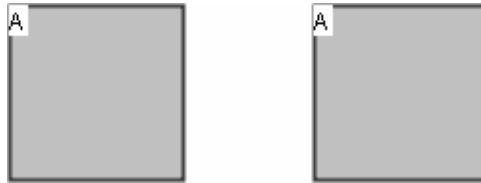


Figure 36 Example of multiple graphical objects representing a single conceptual object

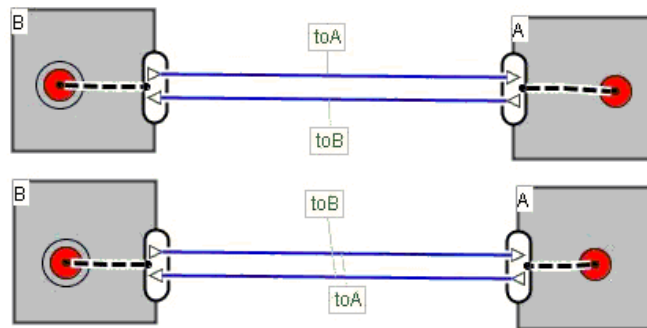


Figure 37 Example of multiple graphical objects representing a single conceptual object

These are two examples of models containing multiple graphical objects that represent a single conceptual object instance. When generating profitability sheets the e^3 -value tool takes the model created by the user and transforms it (if necessary) into a conceptual model, merging the object instances where possible. This new model is then used to generate the actual spreadsheet.

Besides the object name there is a number of other conditions that are tested:

4. Both objects must have the same name (excluding "" – empty string)
5. Both objects must be of a corresponding class type.
6. Both objects should be in the same *scope*.

The scopes of both objects should be equal, but can depend on the class type. For example: for some classes it may be sufficient if both instances exist in the global scope (which is always true per definition), but two value ports will not be merged unless they have the same names as well as exist in the scope of the same value interface.

If one (or more) of these criteria is **not** met, the two model concepts will simply not be merged. A new instance will be created for each of them. If they **are** met, another number of criteria is evaluated. While these criteria are specific for each class type, they can be divided into two categories.

3. The collections of formulas of both objects must be mergable – if two objects have one or more formulas with the same name (header), the formulas' bodies must be equal.
4. The attributes of both objects should be equal. This includes fractions, but also references to other objects (in which case both the name and class type should be equal).

Merge Criteria (per class):

Market Segment	Merge if: the same name is used by any other market segment and their formulas are mergable.
Actor	Merge if: the same name is used by another Actor, both actors exist in the same scope, and their formulas are mergable.
Value Object	Merge if: the same name is used by any other value object and their formulas are mergable.
Simple Dependency Element	Merge if the same name is used by another simple dependency element of the same class, both share attributes (including connections to connection elements) of the same name, and their formulas are mergable.
Complex Dependency Element (except ValueInterface)	Merge if the same name is used by any other dependency element of the same class.
Connection Element	Merge if the same name is used by any other connection element if both share the same characteristics: the names of the objects in the following fields must be equal for both instances: up_de, down_de; and their formulas are mergable.
Value Activity (in scope Actor / Market Segment)	Merge if the same name is used by another value activity, both exist in the same scope, and their formulas are mergable.
Value Offering	Not applicable.
Value Port (in Scope Value Offering)	Merge if the same name is used by another value port which shares the same parent (of class Value Interface) both instances have the same direction (inward / outward).

Value Exchange (in Scope Value Port)	Merge if the same name is used by another value exchange which is connected to the ports bearing the same name (both in and out).
Value Interface	Merge if the same name is used by any other value interface and their formulas are mergable.
Value Transaction	Merge if the same name is used by any other value transaction and formulas are mergable.

Table 6 Criteria for merging graphical objects

If more of the criteria in the above list are *not* met, then the export process will be aborted and an error message will be displayed.

4.7 Valuation functions

When you are creating models using the e^3 -value tool, it is important to understand the way in which value ports are assigned valuation functions. This will be discussed in this section.

In the e^3 -value tool, valuation functions are represented as special formulas, and they can be attached to value ports or value exchanges. To create a valuation function for one of these objects you can simply create a new formula and name it "VALUATION".

When generating profitability sheets from your models, the e^3 -value tool will attempt to decide which valuation should be used for which value ports, and the outcome of this decision depends on the type of value object that has been attached to each of the value ports. It makes a distinction between the following three possibilities:

- The port under consideration is attached to a value object of the type "MONEY".
- The port under consideration is attached to a value object of another type.
- The port under consideration is not attached to a value object.

This factor plays a role in determining where the tool will search for a valuation function to assign to the value port.

This search is conducted using the following rules:

3. In case a value port exchanges a "MONEY" value object via a value exchange:

- 3.1. If a valuation formula is found for the value exchange under consideration connected to the port under consideration, it is used.
- 3.2. If a valuation formula is found for the port under consideration, it is used (the ordering of these two rules models that bi-literal price arrangements override uni-literal, overall arrangements)

- 3.3. If a valuation formula is found for the “peer”-port connected to the port under consideration (via the value exchange under consideration), it is used.
- 3.4. If the value port under consideration belongs to a container (actor / value activity/market segment) which in turn belongs to another (composite) container, the *e³-value* tool will use a default valuation function (created by propagating the valuation functions of other ports nearby to the value port under consideration – See paragraph: 3.2.6 “Default valuation functions”)
- 3.5. If no valuation formula is found this results in an error. The same holds if a different valuation formula is found on both ports.

4. In case a value port exchanges other objects than a “MONEY” object:

- 4.1. If a valuation formula for that port is specified it is used.

Note: If any of the steps above result in an error, profitability-sheet generation is aborted.

Default valuation functions

It is of course possible to define a valuation function for each value port instance in the model, but this would be a lot of work, and it would also not be necessary in most cases. To reduce the number of valuation functions in the models the *e³-value* tool uses so called 'default valuation functions'. These functions are formulas that are automatically generated for each port in the model, and they are used for profitability sheet generation when the *e³-value* tool is unable to find a custom valuation for a port under consideration.

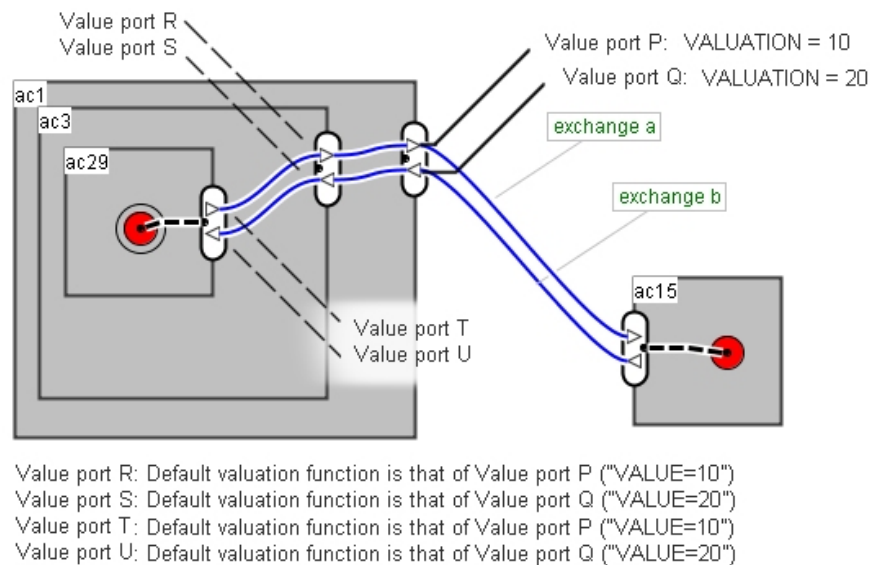


Figure 38 Example of default valuation formulas

"Default" valuation formulas are automatically generated by collecting the existing 'VALUE' formulas for all value ports that are part of a value interface which is attached to a 'autonomous' actor, activity or market segment. (IE: all actors, value activities, and market segments that connected via an *in-* or *out-connected* value exchange.) These valuation formulas are then cascaded onto the other value ports in the model (the ones that only have *first-* or *second-connected* value exchanges).

Chapter 5 E3-value editor: instructions for the programmer

In this chapter, you will find an overview of the third party software packages the e^3 -value tool uses, as well as a more detailed description of their functions and implementation.

The e^3 -value tool uses the following third party software packages.

- Batik
- GOLD Parser
- Jakarta POI HSSF
- JGo™ Java diagram graphics libraries
- RDF2Java

Each package will be discussed individually.

5.1 Batik

5.1.1 Details

Batik is a Java(tm) technology based toolkit for applications or applets that want to use images in the Scalable Vector Graphics (SVG) format for various purposes, such as viewing, generation or manipulation. SVG is a W3C recommendation; it is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text.

Name:	<u><i>Batik SVG toolkit</i></u>
Version:	1.5
URL:	http://xml.apache.org/batik/index.html
Developer:	The Apache Software Foundation
Developer URL:	http://www.apache.org/foundation/

5.1.2 Implementation

The business model created in the *e³-value* graphical editor can be represented in two graphical formats. One of them is the SVG format. JGo uses the Batik software in order to produce documents in this format. The API is only used indirectly via the JGo software. We only had to include the batik software in our libraries.

5.2 Jakarta POI HSSF

5.2.1 Details

The *e³-value* tool uses the POI HSSF API for spreadsheet generation. The Apache Software Foundation as part of their 'Jakarta' project have developed the POI HSSF.

The POI project consists of APIs for manipulating various file formats based upon Microsoft's OLE 2 Compound Document format using pure Java.

HSSF is the POI Project's pure Java implementation of the Excel '97(-2002) file format. It provides a way to read spreadsheets create, modify, read and write XLS spreadsheets.

Name:	Jakarta POI HSSF
Version:	2.0 - Final 20040126
URL:	http://jakarta.apache.org/poi/hssf/index.html
Developer:	The Apache Software Foundation
Developer URL:	http://www.apache.org/foundation/

5.2.2 Implementation

The HSSF API is used by the *e³-value* profitability sheet generator component, and it is used to create XLS workbooks as well as write them. The API is used 'as is', meaning that no alterations have been made to the source code.

5.2.3 Limitations

Currently the e^3 -value tool does not support formulas containing *unary operators* (e.g. the '*minus*' symbol in the expression '-1'). This is because of the fact that the HSSF API currently does not support them. Because of this, it is currently not possible to use such expressions when writing formulas in the tool. A work-around for this problem is to rewrite the expression as "(0-x)".

HSSF is expected to support unary operators in the near future, and this feature will be integrated in the tool as soon as possible.

5.3 JGo™ Java diagram graphics libraries

5.3.1 Details

JGo™ is a Java graphics library that makes it easy to build custom interactive diagrams. It has built-in support for many shapes, text, images, containers, links, arrowheads, scrolling, zooming, selection, drag-and-drop, resizing, in-place text editing, tool tips, and printing. JGo offers a package of classes that need to be extended to create a custom application. In these extended classes the behaviour and attributes specific to that application are defined. JGo is based on the Model View Control architecture.

Name:	JGo: Java diagram graphics libraries
Version:	5.0
URL:	http://www.nwoods.com/go/jgo.htm
Developer:	Northwoods Software
Developer URL:	http://www.nwoods.com

5.3.2 Implementation

All graphical elements in the editor are derived from JGoObject. These are the atoms of the framework. They are owned by a JGoDocument, essentially a container class (the Model), seen in a JGoView (the View) and managed by the editor (the Control). Some JGoObjects are JGoAreas, graphical containers that can hold other JGoObjects. This way hierarchical graphical objects can be built. Some JGoObjects are JGoPorts. JGoPorts are linkable to each other using JGoLinks. Deriving from these JGo classes enabled us to use most of their rich functionality. We managed to do this without altering the JGo source code. Only in three instances we had to introduce some slight modification. One fixed a bug and the other two

are enhancements.

5.4 GOLD Parser

5.4.1 Details

GOLD Parser is a software package that allows the e³-value tool to parse formulas written using a simple grammar specifically designed to be able to refer through object in the e³-value ontology.

Name:	GOLD Parser
Version:	2.06
URL:	http://www.devincook.com/goldparser/
Developer URL:	http://www.devincook.com/goldparser/contributors/index.htm

5.4.2 Implementation

Formulas in the e³-value tool are based on Microsoft Excel ('98). This makes it possible to write formulas in the e³-value tool and maintaining them when we want to generate profitability sheets (which are created in the Excel '98 format). However, there is one difference between e³-value formulas and Excel formulas. This difference lies in the way in which formulas can refer to other values (references).

Where, in Excel, you would typically refer to cells, in the e³-value tool you refer to attributes of e³-value objects. To this end, we have developed a simple grammar / formula language (using the GOLD Parser grammar-building tool) that allows end-users to refer to e³-value objects in the scope of the business models.

A second component of the GOLD Parser solution is the GOLD *engine*, a parsing engine that has been implemented in the e³-value Profitability sheet generator. This engine reads the output of the grammar-building tool (a so called 'grammar file') and uses it to parse the complex formulas created by the user into their most elementary form. These elementary references are then mapped to cell coordinates in the Excel sheets.

The GOLD Parser engine is used 'as is', meaning that no alterations have been made to the source code.

5.5 RDF2Java

5.5.1 Details

The RDF2Java tool allows RDF files to be imported into Java and Java objects to be exported to RDF files.

Simple Java classes (these classes are derived from the *e³-value* ontology RDFSchema using the same RDF2Java tool) serve this purpose (similar to the JavaBeans mechanism, put and get methods are used). It uses the Java reflection API to directly serialize and de-serialize Java objects.

Name:	RDF2Java
Version:	20010705
URL:	http://www.dfki.uni-kl.de/~schwarz/RDF2Java/doc/index.html
Developer:	DFKI (German Research Centre for Artificial Intelligence), Kaiserslautern.
Developer URL:	http://www.dfki.de/dfkiweb/start.htm

5.5.2 Implementation

We have used the RDF2Java tool to generate simple Java classes from the *e³-value* RDFSchema. These generated Java classes mirror each of the classes from the *e³-value* ontology. The classes are used by the following *e³-value* tool components:

- *e³-value* RDF exporter
- *e³-value* Profitability sheet generator

The RDF exporter creates instances of these 'ontology mirror classes' to represent the business models created by the user in a way that is conforming to the *e³-value* ontology. These instances are assigned attributes (literals as well as object references) and then exported to an RDF stream using the RDF2Java export function.

The Profitability sheet generator uses the output from the RDF exporter to generate profitability sheets. The RDF exporter offers the sheet generator a conceptual representation of the business models created by the user in the form of class instances of the 'ontology

mirror classes'.

5.6 Customizing third party software

We used third party software unchanged as much as possible. In a few cases a slight change was necessary. For future reference we will describe these changes, their purpose and their necessity

5.6.1 JGo

5.6.1.1 JGoDocument.java

The method `setMaintainsPartID` is changed to the following:

```
/**
 * Change whether this document assigns a unique PartID to each JGoldentifiablePart
 * as it is added to the document.
 * <p>
 * Setting this property to true will also invoke ensureUniquePartID().
 * Setting it to false will clear the HashMap that findPart uses, but will
 * not modify any JGoldentifiablePart PartID.
 */
public void setMaintainsPartID(boolean bFlag)
{
    boolean old = myMaintainsPartID;
    if (old != bFlag) {
        myMaintainsPartID = bFlag; // in original code myMaintainsPartID = old;
        fireUpdate(JGoDocumentEvent.MAINTAINS_PARTID_CHANGED, 0, null, (old ? 1 : 0), null);
        if (bFlag)
            ensureUniquePartID();
        else
            myParts = null;
    }
}
```

The original code is showed as a comment.

A short reflection will convince you that the original code was at fault. It resulted in the impossibility to maintain unique ID's automatically in the editor. Repairing the bug in the derived class E3tor_diagram is impossible because the field `myMaintainsPartID` is private.

5.6.1.2 *JGoTextEdit.java*

```
public JComponent createComponent(JGoView view) {
    JTextComponent textc;
    if (myMultiline)    textc = new JGoJTextArea(myOriginalText, this);
    else textc = new JGoJTextField(myOriginalText, this);

    textc.addCaretListener(    new Listener(textc) ); // this line is added
    return textc;
}
```

And the following class is added:

```
private class Listener implements CaretListener{
    private JTextComponent c;
    Listener(JTextComponent c){
        this.c = c;
    }
    public void caretUpdate(final CaretEvent e){
        Rectangle r = getBoundingRect();
        int longest_line =0 , lines = 0;
        for(StringTokenizer token = new StringTokenizer(c.getText(),"\n"); token.hasMoreTokens();++lines) {
            String s = token.nextToken();
            int l = s.length();
            if(longest_line < l) longest_line = l;
        }
        r.width = 20 + longest_line * 6;
        r.height = 17 + lines * 17;
```



```

    setBoundingRect(r);

    update();

}

}

```

These changes result in textfields that grow while adding text. The original JgoText field only resize after completing the editing. One might consider to derive JGoTextEdit a custom class E3TextEdit that implements the desired feature by overriding `createComponent`.

5.6.1.3 JGoView

The method `pickNearestPort` is rewritten in order no nearest port is picked that is too close to a port already selected. The return statement of the method has been replaced by the next piece of code.

```

if ( currentBestPort == null || myTempStartPort == null ) return currentBestPort;

Point dest = currentBestPort.getSpotLocation(JGoObject.Center);

Point src = myTempStartPort.getSpotLocation(JGoObject.Center);

int dist = (src.x - dc.x)*(src.x - dc.x) + (src.y - dc.y)*(src.y - dc.y) ;

if( dist < 100 && (dest.x - dc.x)*(dest.x - dc.x) + (dest.y - dc.y)*(dest.y - dc.y) >
    dist ) return null;

return currentBestPort;

```

It is not possible to achieve this effect in the derived class `View` because `myTempStartPort` is a private field in `JGoView`.

5.6.2 Gold

The JAVA-Gold parser environment normally reads the strings to parse from a file. We have changed the Java-Gold parser engine to read from a string as follows:

5.6.2.1 GOLDParse.java

The method `openFile(filename)` will be called with the string to parse, rather than the filename that contains the file to be parsed.

5.6.2.2 LookAheadStream.java

The method `openFile(file)` is changed into:

```
public boolean openFile(String file) throws ParseException
{
    // try
    // {
    // buffR = new PushbackReader(new FileReader(new File(file)));
    buffR = new PushbackReader(new StringReader((file)));
    unreadme = new char[kMaxBufferSize];
    bufferPos = 0;
    // }
    // catch(IOException ioex)
    // {
    // throw new ParseException("Source file could not be opened.");
    // }
    return true;
}
```

This code does not open a file, but places the string to parse directly in the PushbackReader.