Faculteit der Wiskunde en Informatica

Toolkit for Conceptual Modeling (TCM)
User's Guide and Reference

F. Dehne, R.J. Wieringa

Rapportnr. IR-437
november 1997

vrije Universiteit *amsterdam*

Toolkit for Conceptual Modeling (TCM)
User's Guide and Reference

for version 1.6

Frank Dehne
Roel J. Wieringa

Faculty of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081a, 1081 HV Amsterdam
The Netherlands
E-mail: tcm@cs.vu.nl

November 17, 1997

Abstract

The Toolkit for Conceptual Modeling (TCM) is a suite of graphical editors for a number of graphical notation systems that are used in software specification methods. The notations can be used to represent the conceptual structure of the software - hence the name of the suite.

This manual describes version 1.6 of TCM. TCM runs on Unix systems with X Windows. The TCM ftp site is ftp://ftp.cs.vu.nl/pub/tcm. The TCM home page is http://www.cs.vu.nl/~tcm. The use of TCM is free for education, academic research and other non-commercial purposes.

This version of TCM contains graphical editors for several kinds of documents, namely diagrams, tables and trees.

- Editors are available for the following kinds of diagrams: generic graph diagrams, entity-relationship diagrams, class-relationship diagrams, state transition diagrams, recursive process graphs, data (and event) flow diagrams, and JSD process structure and system network diagrams.
- Editors are available for the following kinds of tables: generic tables, transaction decomposition tables, transaction-use tables and function-entity type tables.
- Editors are available for the following kinds of trees: generic textual trees and function refinement trees.

The current version of TCM supports constraint checking for single documents (e.g. name duplication, cycles in is-a relationships). TCM distinguishes built-in constraints (of which a violation cannot even be attempted) from immediate constraints (of which an attempted violation is immediately prevented) and soft constraints (against which the editor warns, after it is asked to check the drawing). The current version of TCM does not yet support constraint checking across documents for integrated conceptual modeling. Future versions of TCM will be enhanced with cross-diagram checking functions.

All editors share as much as possible of their user interface, which is designed to be usable without user manual. There is a simple on-line help facility and this user's guide annex reference is supplied with TCM in PostScript format and in HTML format, which can be read by a web browser.

This document describes the user interaction with TCM. It is intended to be a guide for both beginners and advanced users. The last appendix contains a mini-tutorial of the notation techniques supported by this version of TCM. Full information can be found in:
R.J. Wieringa. *Requirements Engineering - Frameworks for Understanding.* Wiley, 1996. ISBN 0 471 95884 0. http://www.cs.vu.nl/~roelw/books.html.

Contents

List of Figures

Chapter 1

Introduction

1.1 An Overview of TCM

1.1.1 The Purpose of TCM

The Toolkit for Conceptual Modeling (TCM) is a collection of software tools to represent conceptual models of software systems in the form of diagrams, tables, trees, etc. A **conceptual model** of a system is an abstraction of the behavior or decomposition of the system. A conceptual model can be presented visually by means of diagrams, graphs, trees, tables or structured text. During software development, a number of stakeholders must reach a common understanding of the behavior and structure of the software. These are for example users and sponsors (or their representatives), analysts, designers and programmers. An important function of conceptual models is to facilitate this understanding.

The notations for software specifications that are supported by TCM are explained in appendix C. A more detailed analysis of the nature and use of some of these techniques and heuristics in various existing methods for software specification is presented in the book [18] [1].

1.1.2 What is Included in TCM

This document describes version 1.6 of the TCM software tools. This manual is the successor of [6]. This version of the TCM software (henceforth called "TCM") is a collection of document editors, i.e. diagram, table and tree editors, each for a different graphical notation system. Currently, TCM has the following editors:

- Diagram Editors.
  - Generic Diagram Editors.
    - **TGD: Tool for Generic Diagrams.** This tool is intended for arbitrary diagrams, not covered by one of the other diagram editors.
  - Data View editors.
    - **TERD: Tool for Entity Relationship Diagrams.** This tool is intended for entity relationship diagrams (ERDs) that are used in Entity Relationship (ER) modeling [4]. The ER method is explained in [18]. The ER notation that TCM supports is explained in appendix C.1.1.

[1] Bibliographic references can be found on page 132.

    * **TCRD: Tool for Class-Relationship Diagrams.** This tool is intended for class-relationship diagrams (CRDs) that are used in object oriented modeling. The CR notation variant that TCM supports is explained in appendix C.1.2.

  - Behavior view.
    * **TSTD: Tool for State Transition Diagrams.** This tool is intended for state transition diagrams (STDs) according to the Mealy notational convention. This technique is explained in appendix C.2.1.
    * **TRPG: Tool for Recursive Process Graphs.** This tool is intended for recursive process graphs (RPGs), also called life cycle diagrams. This technique is explained in appendix C.2.3.
    * **TPSD: Tool for Process Structure Diagrams.** This tool is intended for process structure diagrams (PSDs) that are used in the Jackson System Development method (JSD) [9]. The JSD method is discussed in [18]. This technique is explained in appendix C.2.2.
  - Function view.
    * **TDFD: Tool for Data Flow Diagrams.** This tool is intended for data flow diagrams (DFDs) that are used in the method of Structured Analysis (SA) [5, 22]. This method is discussed in [18]. The technique is explained in appendix C.3.1.
    * **TDEFD: Tool for Data and Event Flow Diagrams.** This tool is intended for data and event flow diagrams (DEFDs), that are used in the method of Structured Analysis for Real Time Systems (SA/RT) [16] and in the Yourdon Systems Method [23]. The technique is explained in appendix C.3.2.
    * **TSND: Tool for System Network Diagrams.** This tool is intended for system network diagrams (SNDs) that are used in the Jackson System Development method (JSD) [9]. The JSD method is discussed in [18]. The technique is explained in appendix C.3.3.

- Table Editors.
  - **TGT: Tool for Generic Tables.** This tool is intended for generic tables not covered by one of the other table editors.
  - **TTDT: Tool for Transaction Decomposition Tables.** This tool is intended for transaction decomposition tables. These tables are used in some of the methods presented in [18]. The technique is explained in appendix C.4.1.
  - **TTUT: Tool for Transaction-Use Tables.** This tool is intended for transaction-use tables. These tables are used in some of the methods presented in [18]. The technique is explained in appendix C.4.2.
  - **TFET: Tool for Function-Entity type Tables.** This tool is intended for function-entity type tables. These tables are used in some of the methods presented in [18]. The technique is explained in appendix C.4.3.
- Tree Editors.
  - **TGTT: Tool for Generic Textual Trees.** This tool is intended for generic trees in which all nodes are text labels.
  - **TFRT: Tool for Function Refinement Trees.** This tool is intended for function refinement trees, also called *function decomposition tree.* Function refinement trees of this type are used and discussed in [18]. See also appendix C.5.

All editors look very similar and have many operations in common. All diagram editors have almost the same edit commands and all table editors have almost the same edit commands.

The current version of TCM supports constraint checking for single documents (e.g. name duplication, cycles in is-a relationships) but does not yet support constraint checking across documents for integrated conceptual modeling.

1.1.3 Using TCM in Software Specification

Figure 1.1 gives an overview of how the different document editors in TCM can be used in various methods for software specification. Each of these methods uses a subset of the notations available in TCM and defines a number of consistency rules across notations. Thus, each method allows the specification of a model of the required software product and uses different notations to specify different views upon this model. The consistency rules guarantee that there *is* a model that is being specified. Views that are inconsistent according to these rules, do not represent a model. The consistency rules of ER, DF and JSD are defined in [18] and the consistency rules of YSM are defined in [23]. The current TCM version does not yet implement all the consistency rules of these methods. The implementation of consistency rules of these methods is planned for TCM version 2.0 as is described in [7].

In TCM there is no consistency checking for the ISAC and Information Engineering methods. But tools for TCM can be used to draw the diagrams. The same applies to object oriented (OO) methods: TCM currently contains tools for drawing diagrams and tables for OO models, albeit in a bit different notation. However, in the future we want to build support for integrated OO modeling into TCM as well.

1.2 How to Read this Manual

If you want to try TCM for the very first time, you first have the read section 1.4 to get started. You probably do not have to read much more of this manual for carrying on. The user interface is designed to be as intuitive as possible and to be proof against users who are trying things at random. The most important commands that you need are explained in the on-line help windows.

This manual is intended to be a reference manual for the user in the first place, hence the amount of details. You certainly need not read it all to be able to use TCM. Chapter 2 describes the common features between all TCM editors, such as saving documents, loading documents, printing documents etc. Chapter 3 describes how you edit a diagram abstracting from a particular diagram editor. In chapters 4, 5 and 6 you find all the material specific to the different diagram editors, grouped by data view editors, behavior view editors and function view editors, respectively. These chapters are organized in the same order as the list of diagram editors in section 1.1.2 and the list of tools in the TCM startup tool. Chapter 7 describes how you can edit tables in general and it explains the (small) differences between the four table editors. Chapter 8 describes how to edit a tree document in general and it explains the differences between the two tree editors. In appendix A, there is a list of frequently asked questions (FAQ). In appendix B there is a specification and explanation of the TCM file format. Finally, in appendix C a mini tutorial is given to the notations and methods supported by the TCM tools.

You can search for information in this manual either top down, via the table of contents, or bottom up, via the index pages at the end.

1.3 How to Obtain the Latest Version of TCM

There is an ftp site ftp://ftp.cs.vu.nl/pub/tcm from which you can download distributions with executables for different Unix platforms. Currently, TCM runs on Solaris 2.x (Sparc and x86), SunOS

Method	Tool	Notational technique
ISAC	TGD	To make activity diagrams and cause/effect graphs
	TGT	To make problem/owner matrices
Information Engineering (IE)	TFRT	To make function refinement trees
	TERD	To make ER diagrams
	TFET	To make function-entity type tables
Entity-Relationship (ER) modeling	TFRT	To make function refinement trees
	TERD	To make ER diagrams
	TTUT	To make transaction-use tables
Structured Analysis (SA)	TFRT	To make function refinement trees
	TERD	To make ER diagrams
	TTUT	To make transaction-use tables
	TTDT	To make transaction decomposition tables
	TDFD	To make data flow diagrams
Jackson Systems Development (JSD)	TPSD	To make process structure diagrams
	TSND	To make system network diagrams
	TGT	To make action allocation tables
Structured Analysis for Real-Time Systems (SA/RT) or Yourdon Systems Method (YSM).	TFRT	To make function refinement trees
	TERD	To make ER diagrams
	TTUT	To make transaction-use tables
	TTDT	To make transaction decomposition tables
	TDEFD	To make data and event flow diagrams
	TSTD	To make state transition diagrams
OO methods	TERD or TCRD	To specify the data view
	TRPG or TSTD or TPSD	To specify the behavior view
	TDFD or TDEFD	To specify the function view
	TFRT	To specify function refinement trees
	TTDT	To specify transaction decomposition tables
	TTUT	To specify transaction-use tables
	TFET	To specify function-entity type tables

Figure 1.1: Using TCM in several software specification methods.

4.x.x (Sparc), Linux (x86), IRIX 5.x (SGI), AIX 4.x (RS6000) and HP-UX 10.x. Each distribution contains executables of the TCM software and recent documentation. See the file `README.TCM` on the ftp site for more details.

TCM has its own home page: http://www.cs.vu.nl/~tcm. This page contains supplementary information about TCM. It is possible to download the TCM software via the page http://www.cs.vu.nl/~tcm/software.html. You can find a HTML version of the most recent version of the user manual in http://www.cs.vu.nl/~tcm/usersguide.html. The user manual is also contained as PostScript file and as a set of HTML files in the `doc` subdirectory of the software distribution.

If you plan to upgrade TCM, see the file `CHANGELOG` on the ftp site for the most important changes between the consecutive versions.

1.4 TCM and the Unix Environment

1.4.1 Getting Started

TCM runs on Unix systems with X Windows version 11 release 5 or higher and uses the Motif library release 1.2 or higher for the graphical user interface. For up-to-date system requirements see the file `README.TCM` in the software distribution.

When you have retrieved the TCM distribution in the form of a Unix `tar` file compressed with the `gzip` program [2], you have to install [3] it in a new directory, preferably `/usr/local/tcm` [4]. Each user should set the environment variable `TCM_HOME` to the directory where TCM resides, and add `$TCM_HOME/bin` to the `PATH` shell variable and add `$TCM_HOME/man` to the `MANPATH` shell variable. TCM will not work when `TCM_HOME` is not set. Do not forget to export these variables. You can start a particular tool in two ways:

1. Call it from the program `tcm`. `tcm` displays a window containing a push button for each available editor (see figure 1.2). To start an editor click the corresponding button. The diagram and tree editors are started up immediately. When a table editor is chosen, first a dialog is displayed in which you can change some options. Be careful, because each time you click a button a new editor is started.

   To kill the `tcm` startup program issue `Quit` from the File menu. The processes of the running editors are independent from the running startup program, so when you quit a startup program, all other programs continue to run.

2. Call the editor directly from the shell, e.g. `terd`, `tdfd`, `tcrd`. Perhaps you want to end the command with an ampersand to run it in the background. You can supply a single document name as command line argument. If this argument is the name of an existing file then the editor tries to load a document from it. If the document does not exist, a new document is created in the editor having the argument as document name. In section 1.4.2 we give an overview of the flags that can be given in the command line and the Unix environment variables that are used by TCM.

1.4.2 Unix options, files and variables

The Unix manual pages in the TCM distribution give an up-to-date overview of all arguments, environment variables and files that are used by TCM. Here we will give some explanation of the most important ones. TCM has the following command line arguments:

[2] gzip (GNU zip) is a compression utility designed to be a replacement for "compress". It can be downloaded from ftp://prep.ai.mit.edu/pub/gnu.

[3] An up-to-date installation guidance can be found in the file `README.TCM`.

[4] Note to users at the Vrije Universiteit, faculty of Maths and CS : all TCM software is installed in `/home/tcm`.

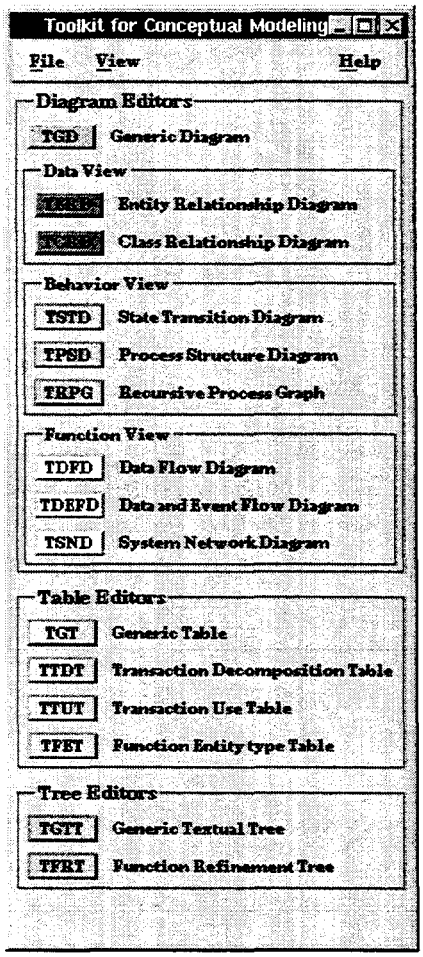


Figure 1.2: The TCM startup window.

- *Filename.* This will be the document name when the editor is started, instead of `untitled`. If the file exists and contains a valid document for the concerning editor, the document is loaded while starting up. If the file does not exist, only the document name in the editor will be modified.
- `-drawing` *WidthxHeight.* Specifies the size of the drawing area in pixels. The default size is 1330x1330 pixels [5]. Increasing the values makes it possible to draw bigger documents, but it can slow down the editor significantly. Smaller values make it possible to run TCM faster on a machine with little memory.
- `-help`. Write all allowed command line arguments to standard output and exit.
- `-maxdrawing` *WidthxHeight.* The drawing area can not be larger than *width* pixels wide and *height* pixels high. The default maximum values are 3000x3000 pixels. When the drawing is larger than the current drawing area, the drawing area is made larger up to this maximum. However, when the drawing area is extremely large, the editor becomes slower and may eventually crash because there were insufficient resources to allocate the background pixmap.
- `-priv_cmap`. Start the editor with a private color map. Normally editors use the default color map.
- `-projdir` *directory.* Set the project directory (working directory) to this directory.
- `-version`. Write the TCM version to standard output and exit.
- In addition to these options, the standard X11 toolkit options like `-background` and `-display` are accepted as command line arguments (see the X window man page, X(1) or X11(7)).

Table editors additionally have the following optional command line arguments:

- `-table` *RowsxColumns.* The table editor will be initialized with a table having *rows* rows and *columns* columns.
- `-cell` *WidthxHeight.* Sets the minimal cell size of the table editor to *width* pixels wide and *height* pixels high.

In addition to the TCM tool executables themselves and shared libraries, the tools depend on the following files:

- `$TCM_HOME/lib/tcm.conf`. This is the TCM system wide configuration file. This file contains initialization values for the tools, like the name of the printer, the Unix command to print a file, the default page size etc. When the user has no file `$HOME/.tcmrc`, then this file is read by each tool upon startup. The menu entries of the editor are initialized with the values from the configuration file. The settings in `tcm.conf` are already adapted to the Unix platform that you use, but, when you have installed TCM, it is maybe a good idea to check them. In the configuration file itself you find directions for how to modify it.
- `$HOME/.tcmrc`. Each user of TCM can override `tcm.conf` by his own configuration file, which has to be installed in `$HOME/.tcmrc`.
- `$TCM_HOME/lib/Tcm`. This file contains some X Resources. These X resources are already built-in in the tools, so this file is not read in by TCM. But you can customize with this file the fonts and colors of the Motif widgets that TCM uses, by setting these resources with different values in your X defaults database. Each string of the form "Tcm.resource:definition" sets a resource. See question A.19 in the FAQ for how exactly you can set X resources.

[5] one pixel is 1/83 inch which is about 0.306 millimeter.

- `$TCM_HOME/lib/help/`. This is a directory that contains a collection of text files for the on-line help windows that are issued via the entries of the Help menu. If you have installed TCM, make sure that these help files are readable for all.
- `$TCM_HOME/lib/banner.ps`. PostScript banner page that can be used when the printer does not print a banner page. To use this banner by default you have to edit `tcm.conf` or `.tcmrc`.
- `$TCM_HOME/bin/psf`. A Perl script that is used to filter PostScript output written by TCM. It assumes that the Perl interpreter is in `/usr/bin/perl`. Otherwise, you have to edit the first line of `$TCM_HOME/bin/psf` to make it call your local interpreter [6]. Psf is supplied with TCM and can be used stand alone as well, see `man psf`.
- `$TCM_HOME/bin/text2ps`. A small program that converts ASCII text files to PostScript. It is used by TCM for printing text, for instance the on-line help pages. Text2ps is supplied with TCM and can also be used stand alone, see `man text2ps`.

The TCM editors depend on the following Unix environment variables.

- `TCM_HOME`. The parent directory of the directories where the TCM executables, libraries and manual pages are expected to reside. The tools will not start when `TCM_HOME` is not set. When `TCM_HOME` is set to the wrong directory, the configuration and help files can not be read, which will be reported upon start up.
- `PATH`. You are advised to add `$TCM_HOME/bin` to your PATH.
- `MANPATH`. You are advised to add `$TCM_HOME/man` to your MANPATH. Each tool has a short Unix manual page. E.g. `man terd` will show the man page of the ER diagramming tool TERD.
- `LD_LIBRARY_PATH`. If TCM is installed in a directory other than `/usr/local/tcm` and the TCM distribution has shared object libraries [7], it is necessary to append `$TCM_HOME/lib` to this variable. Otherwise the shared libraries can't be found by TCM. In the FAQ in appendix A you will find more information about shared libraries and this variable.
- `PRINTER`. The name of the default printer for printing documents. This variable overrides the `PrinterName` option in the TCM configuration files.

1.4.3 Graphical User Interface

The user interface of TCM complies for a large part with the OSF/Motif Style Guide [11]. The user interface is built from the OSF/Motif widget set [10]. The result is that user interaction through menus, dialogs, buttons, scroll bars and text areas work in the same way as other Motif applications and environments such as for instance Netscape for X Windows and CDE (common desktop environment). Only the drawing of diagrams and tables itself is quite different and this part does not make use of Motif. In any case, the TCM editors should be well-behaved under all popular X Window managers, e.g. fvwm, mwm, olwm, vtwm, twm. In [12] you find the specification of the user interaction with Motif applications in general, so it is not necessary to repeat that in this manual.

TCM looks optimal on a full color display and a resolution of at least an XVGA screen (1024x756). It should still be usable though on a black and white display and/or an SVGA screen (800x600). For more detailed information about the other system requirements of TCM regarding Motif, X Windows and Unix see appendix A.

[6] When your system does not have Perl at all, this is not a disaster. Only the Print duplex and tumbled page options do not work.

[7] This is the case when the directory `$TCM_HOME/lib` contains files ending on `.so`.

1.5 Questions and Comments

Questions and comments about the TCM software and about this manual are welcome and can be sent by e-mail to tcm@cs.vu.nl. Appendix A contains a collection of frequently asked questions (FAQ).

Chapter 2

Document Editing

This chapter describes the features common to *all* the TCM editors. This includes most of the user interface components, loading documents, saving documents, printing documents and the on-line help.

2.1 The User Interface of TCM

When you start up an editor, you will see the so-called main window. For a screen dump of the main window see figure 2.1. To be able to work with TCM you need a two- or three-button mouse. In this manual they are called from left to right: button-1, button-2 and button-3 (or from right to left, if you have a special left-hand adjusted mouse). You can always work with two buttons because button-3 is only used for popping up the Edit pop-up menu in the drawing area, whereas the same menu is also accessible via the menu bar.

Except the basic drawing commands in the drawing area, all parts of the user interface can be accessed by keystrokes as well as by mouse operations. This manual assumes that you are using the mouse as much as possible.

Tiled buttons. On the left edge of the main window of the diagram and tree editors there are two sets of tiled buttons containing a bitmap symbol. These contain two kinds of toggle buttons: radio buttons and check buttons. **Radio buttons** are a set of mutually exclusive selection options. The visual cue is a little diamond that is filled or unfilled. A **check button** is a non-mutually exclusive selection option. The visual cue is a little box that is filled or unfilled. When you pass the mouse pointer over a tiled button for a second or two, a one line bubble help clue is popped up giving the full name of what the tile represents.

Menu bar. The menu bar located under the main window's title bar organizes most of the commands and features of the editors. The menu bar works in a straightforward way: press button-1 on an entry and keep it pressed down. A pull-down menu appears. Drag the mouse to the desired command and then release button-1. The menu is dismissed and the command is executed. Some menus contain nested submenus, called **cascading menus**, that work in a similar way. You cancel a menu by moving somewhere outside the menu and then releasing button-1.

Some frequently used commands can also be called directly, without going through a menu, by means of a keystroke shortcut, called an **accelerator**. For example <Ctrl+L> is an accelerator for loading a document from file. You can see in the text of the menu entries which commands have an accelerator. Some menu entries contain check buttons that indicate that a certain property of the editor is switched on or off. If you select this entry, the value of the property will be inverted. See

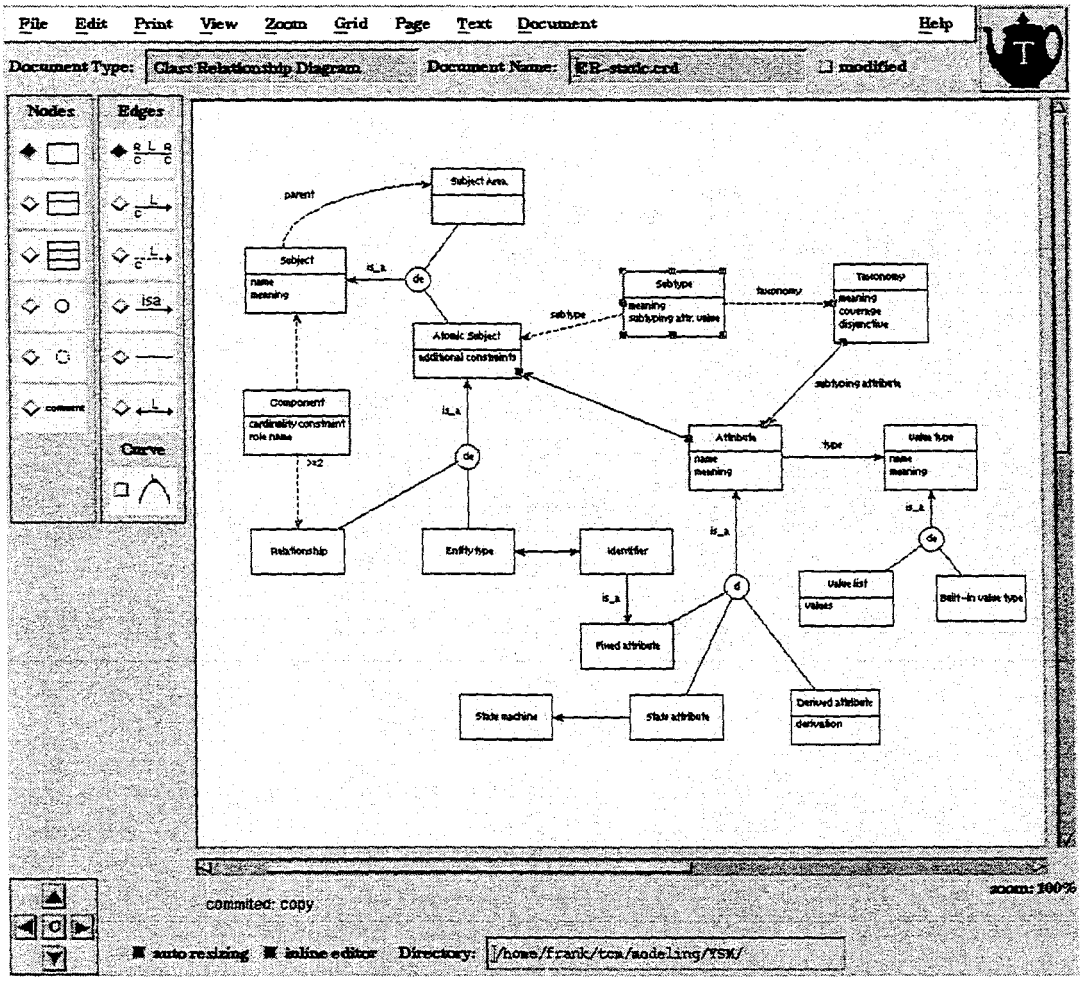


Figure 2.1: TCM main window.

for example the Show Page Boundary entry in the Page menu. Some other menu entries contain a submenu of radio buttons indicating that a certain editor property has a value that is one from a set of menu choices. Try for example the Page Size entry in the Page menu.

Drawing area. The drawing area, also called pane or canvas, is used to create, edit and delete the graphical items of your document by using the mouse. The mouse operations that are distinguished by TCM are summarized in figure 2.2.

Move	Move the mouse to position the pointer on a place in the drawing area
Click	Press and release the mouse button
Double click	Press and release the mouse button twice in rapid succession
Press	Press and hold down the mouse button
Drag	Move the mouse while pressing
Release	Let go of the mouse button

Figure 2.2: Mouse operations.

The whole drawing area is larger than the main window. You can use the **scroll bars** on the right and bottom side of the drawing area to view the drawing area that requires more space than is available at any one time. By resizing the main window you can resize the visible part of the drawing area, keeping the other parts of the main window the same size as much as possible.

TCM has its own coordinate system. By default, the TCM coordinates have the same distance as the coordinates of X Windows. The origin is in the top left corner of the drawing area. Like the X Window coordinates, the x-coordinates increase from left to right and the y-coordinates increase from the top down. By zooming in or out the ratio between the TCM and X Window coordinates can be updated.

Document Type. This is visible as an uneditable text field above the drawing area. See figure 2.3 for how the document types are called.

Document name. This is visible as an editable text field above the drawing area. See section 2.2 for how to change the document name.

Modified. This is visible as a toggle above the drawing area. When the document has been modified, but it is not saved yet, it is on. If the toggle is on and you have loaded or created a new document, TCM warns you that the old document will be lost, and you get the opportunity to save the old document first.

Status area. The result of the last issued command is displayed below the drawing area in an unshaded and uneditable text field.

Directory. The name of the project or working directory is visible in an editable text field at the bottom of the main window right below the status area. See section 2.3 for how to change the project directory.

Zoom value. The current zoom percentage is shown in the bottom-right corner. By performing the zoom commands of the Zoom menu this value is updated.

Autoresizing. This is visible as a toggle beneath the status area. When it is on, the shapes in the diagram or the cells in the table are automatically resized to make it fit the text that they contain (see section 2.5).

In-line editor. This is visible as a toggle beneath the status area next to the autoresize toggle. When it is on, text can be typed directly into the drawing area. When it is off, text editing takes place in a text edit dialog (see section 2.5).

Arrow Buttons. In the bottom-left corner of the main window there are four arrow shaped buttons by which you can move the entire document over the drawing area. Amidst these four buttons there is a button labeled C, by which you can center the drawing on the first page in the drawing area, at least when the drawing is not larger than a single page. When the drawing is larger than one page, the drawing will be centered on the group of pages that the drawing occupies.

2.2 Changing the Document Name

You can type in a new name in the document name text field above the drawing area. When you enter <Return>, the document name is changed. TCM checks whether the name has a valid suffix for the current type of document. See figure 2.3 for the required suffixes. Furthermore, the document name should contain only non-white space printable characters and it should not contain the characters {, } or /.

When a document is loaded, the document name field is set to the name of that document. If a new document is created, either by starting the editor without a file name or by issuing the New command, the newly created document will receive the default name `untitled`.

2.3 Changing the Project Directory

You can change the project (working) directory by editing the directory text field at the bottom of the main window. You can change it by editing it and entering <Return>. TCM checks if the directory exists and if it is accessible. This directory is intended for storing the files related to the current (sub)project that you are working on. It is used as the starting directory for the file selection dialogs for loading and saving documents.

2.4 Loading and Saving Documents

The File pull-down menu contains the following entries:

- **New (<Ctrl+N>).** Creates a new document called `untitled.`*xxx*. Where *xxx* is the suffix for the specific editor (figure 2.3). You can change the name of the new document in the document name text field. Before the new document is created, the current document that was being edited

Editor	Document Type	Name Suffix
TCRD	Class-Relationship Diagram	.crd
TDEFD	Data and Event Flow Diagram	.defd
TDFD	Data Flow Diagram	.dfd
TERD	Entity-Relationship Diagram	.erd
TFRT	Function Refinement Tree	.frt
TFET	Function-Entity type Table	.fet
TGD	Generic Diagram	.gd
TGT	Generic Table	.gt
TGTT	Generic Textual Tree	.gtt
TPSD	Process Structure Diagram	.psd
TRPG	Recursive Process Graph	.rpg
TSND	System Network Diagram	.snd
TSOD	Static Object Diagram	.sod
TSTD	State Transition Diagram	.std
TTDT	Transaction Decomposition Table	.tdt
TTUT	Transaction-Use Table	.tut

Figure 2.3: Document editors, document types and document name suffixes.

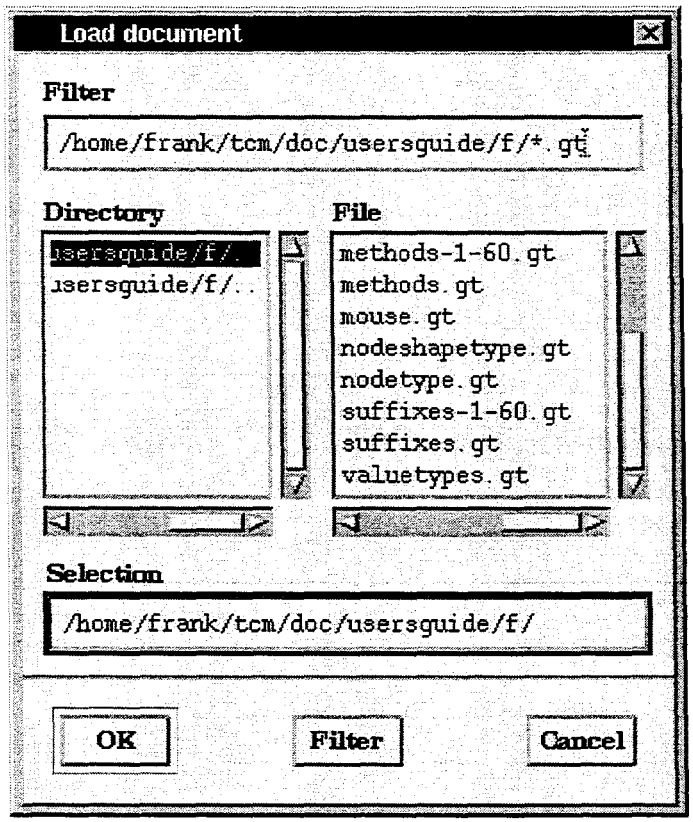


Figure 2.4: TCM File selection dialog.

is deleted from the editor. Unless you have saved this document, the old document cannot be restored.

- **Load (<Ctrl+L>).** Pops up a file selection dialog to select the external file name for loading a document from file (figure 2.4). Before the document is loaded, the current document that was being edited is deleted from the editor.

- **Append.** Pops up a file selection dialog to select the external file name for merging the current document with another one from a file.

  Diagrams that are appended, can be positioned at an arbitrary place in the drawing area by means of a paste box, that works like the Paste command from the Edit menu (see section 3.9). Because appending is implemented as a Paste command, appending a document can be undone with Undo or be repeated by issuing the Paste command again.

  Tables can be appended either below or to the right of the current table. This is determined by an option menu in the append table file selection dialog. In both cases, the original table stays the same but new rows are added to the bottom and/or new columns are added to the right of the table. Append table has an Undo too.

- **Save.** Saves the document to an external file. If the document name is still `untitled`, a file selection dialog pops up, like Save As. Otherwise the document is immediately saved in the file *document-name.suffix* in the project directory. When you attempt to overwrite an existing file, TCM gives a warning and you can cancel your action.

- **Save As (<Ctrl+S>).** Pops up a file selection dialog to select or type in the file name to save the document to. After clicking `OK`, it acts the same as Save.

- **Save Selection As.** Pops up a file selection dialog to select or type in the file name to save the selected part of the document to. In diagram editors the selected shapes and the corresponding part of the graph is saved to file. In table editors the rows and columns that contain one or more selected cells are saved.

- **Quit (<Ctrl+Q>).** Quits TCM. If your current document is modified, TCM asks you if you want to save the document. Before you quit, TCM always asks for an extra confirmation [1].

The file selection dialog (figure 2.4) allows you to select a file in the right side listing or to navigate through the file system by selecting a directory, including `..`, the parent directory, in the left side listing. You can select a file or directory by either: 1. quickly double clicking on an entry, 2. single clicking on an entry and clicking `OK` or 3. filling in the field labeled `Selection` and clicking `OK`. The `Filter` field on top determines which file names are displayed. You can edit the filter setting, which takes effect after clicking the `Filter` button.

When you change directory in the load or save to file dialog then the directory field in the main window is updated to that directory after that you dismissed the dialog. So with the load or save to file dialog you can browse through the file system and the latest visited directory is always remembered in the directory text field. The export dialog from the Print menu also remembers its last visited directory but independently from the directory from the load or save to file dialog.

2.5 Editing Documents

There are two types of document edit commands: commands that are issued by the mouse, when the mouse pointer is in the drawing area, and the commands listed in the Edit menu. All diagram and

[1] Bloodthirsty persons can also quit the editor by sending it a kill signal or by deleting the main window.

tree editors share the same set of edit commands (chapter 3). All table editors too share the same set of edit commands (chapter 7). However, the edit commands of diagram and tree editors on one hand and table editors differ to a large extent.

All document edit commands, except the simple selection commands and the key-stroke text edit commands are undo-able and redo-able (multiple levels). All editors have certain commands to select items, to move and resize items, to edit the text of items, to add items and to delete items. Only the text edit commands are very similar across all editors and therefore they are described in this chapter. For the other commands you are referred to chapter 3 (diagrams and trees) and chapter 7 (tables).

2.5.1 Editing Text in a Document

In order to be able to type in a label of a shape in a diagram or the text in a table cell, the shape or cell should be the *only* currently selected shape or cell. For going into edit mode you can do the following. Move the mouse pointer into the single selected shape or cell, and when the mouse pointer has turned into a I, you can start editing by typing characters or by clicking button-1 again. In both cases the edit mode starts.

There are two edit modes: in-line editing and out-line editing. **In-line editing** takes place directly in the drawing area and during in-line editing a black triangle shaped cursor is visible. **Out-line editing** takes place in a separate window with a text editor that is popped up when the edit mode is entered. That window contains an editable Motif text entry area, a menu bar, scroll bars and two buttons: `OK` and `Cancel`. You can indicate which of the two possible edit modes has to be used by a toggle button labeled `in-line editor`. That toggle is near the bottom of the main window and it is also accessible via the View menu. In general, in-line editing is more suitable for quickly editing short labels, whereas out-line editing has scroll bars and some extra edit operations and is more suitable for editing large chunks of texts. With out-line editing it is also possible to cut and paste text within and between text edit windows.

2.5.2 The In-line Text Editor

Here we summarize all operations that are available in the in-line editor.

- **Start editing.** Go into edit mode like it is explained above. When a triangle shaped black cursor is displayed in the drawing area, you are in the in-line editor.
- **Stop editing.** From the in-line editor you leave edit mode by clicking button-1 outside the shape or cell that is being edited or by clicking button-2 at any position of the drawing area. The text that is edited will be updated, and when the autoresize toggle is on, the shape or cell will be resized to make it fit the text that it contains. After the text is updated, you can still undo the update including the autoresize, by issuing the undo command from the Edit menu. Text editing operations are not undo-able commands but stop editing (from both the in-line as the out-line editor) is an undo-able command.

  When you issue another edit command (with the mouse or from the Edit menu) while you are in in-line edit mode, then edit mode is leaved and the text is updated and then the new edit command is executed.
- **Add character after the cursor.** Type in the character. Labels may contain all printable ASCII characters, except the <Tab> and may have any length. In many notational techniques labels may even contain spaces and newlines.
- **Delete character after the cursor.** Use the <Delete> key.

- **Delete character before the cursor.** Use the <BackSpace> key.
- **Move cursor one character left.** Use the <ArrowLeft> key.
- **Move cursor one character right.** Use the <ArrowRight> key.
- **Move cursor one line up.** Use the <ArrowUp> key.
- **Move cursor one line down.** Use the <ArrowDown> key.
- **Delete all characters.** Use the <Escape> key.
- **Move cursor before the first character.** Use the <Home> key.
- **Move cursor after the last character.** Use the <End> key.
- **Directly position the cursor.** Click with button-1 on the desired cursor position.

2.5.3 The Text Edit Dialog

Text edit dialogs are almost complete text editors, see figure 2.5. Text edit dialogs are not only used for out-line editing text labels but also for editing document and subject annotations. The operations in the dialog are mostly standard Motif operations. Here we will summarize the most important ones:

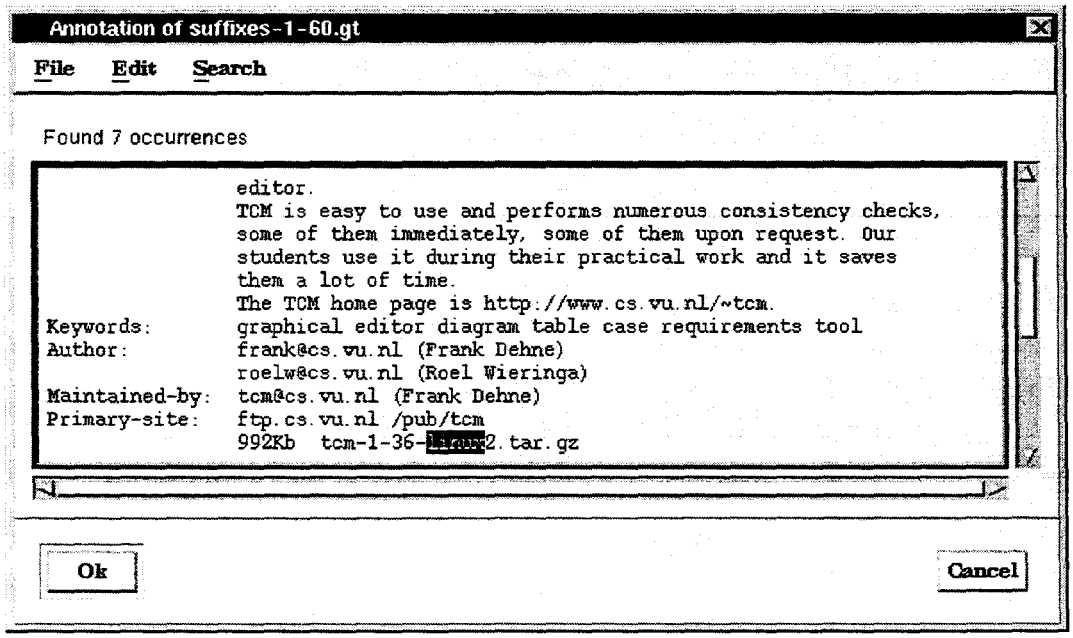


Figure 2.5: TCM text edit dialog.

- **Start editing.** When you have popped up the text edit dialog and the text area has the input focus then you can start editing. A blinking I-beam insertion cursor indicates where text will be inserted.

- **Stop editing.** You leave the edit session by clicking the `OK` button. The dialog is dismissed and the text (a shape label, an annotation, a cell text etc.) will be updated. When the autoresize toggle is on, the shape or cell will also be resized to make it fit the entire text. When the text editor is used for out-line editing, you can undo the update after you have clicked the `OK` button.
- **Cancel editing.** You cancel with the `Cancel` button. When you have canceled, the dialog is dismissed and the text that was being edited will not be updated. The modifications that you have made in the window are lost.
- **Add character after the cursor.** Type in the character. Labels may contain all printable ASCII characters, except the <Tab>.
- **Delete character after the cursor.** Use the <Delete> key.
- **Delete character before the cursor.** Use the <BackSpace> key.
- **Move cursor left.** Use the <ArrowLeft> key to move the cursor one character left.
- **Move cursor right.** Use the <ArrowRight> key.
- **Move cursor one line up.** Use the <ArrowUp> key.
- **Move cursor one line down.** Use the <ArrowDown> key.
- **Move cursor one page up.** Use the <PageUp> key.
- **Move cursor one page down.** Use the <PageDown> key.
- **Delete all (<Ctrl+D>).** Use the **Delete All** command in the edit menu or press <Ctrl+D> in the text edit dialog [2].
- **Move cursor to beginning of line.** <Home> moves the cursor in front of the first character of the current line.
- **Move cursor to end of line.** <End> moves the cursor after the last character of the current line.
- **Directly position the cursor.** You can click with button-1 on the desired cursor position. If you want to change the cursor position without changing the selection, press <Ctrl> while you click button-1.
- **Select text.** You can select a part of the text by dragging with button-1 over the region that you want to select; or when you click button-1 twice in a word then the word is selected; or when you click thrice, the line is selected; or when you click four times, the entire text is selected. Selected text is highlighted in reverse video.
- **Clear selection.** Click button1 anywhere outside the selected region.
- **Copy (<Ctrl+C>).** Copy the selected text into the Motif **clipboard**. Motif has built-in a clipboard which acts like a cut-paste buffer for copying and moving text between text areas of the same or different Motif applications.

[2] The <Escape> key cancels the dialog which is a built-in feature of Motif. This is an important difference between the in-line and the out-line editor.

- **Cut (<Ctrl+X>).** Cut the selected text into the Motif clipboard. This is like Copy but Cut deletes the selected portion after copying it to the clipboard. Note that before you can cut or copy, you have to select some text.

- **Paste (<Ctrl+Y>).** Pastes the contents of the Motif clipboard into the text edit area. The text is inserted at the current cursor position.

- **Find (<Ctrl+F>).** This command pops up a prompt dialog (figure 2.6) for finding a text string in the text edit area. The find text dialog has an input field to type the text that you are looking for and it has a check button and four push buttons that mean the following:

  - **case sensitive.** This check button says that the case of the string to find is significant. By default, Find (and Replace) are case insensitive.
  - **Find Next.** Finds the next string that matches the string to find. The insertion cursor of the edit area is put in front of the string that is found and the scroll bars are adjusted if needed.
  - **Find All.** Highlights all strings that matches the string to find and the dialog says how many occurrences are found.
  - **Dismiss.** Closes the dialog.
  - **Clear.** Clears the text entry field for the string to find.

- **Replace (<Ctrl+Z>).** This command pops up a prompt dialog (figure 2.7) for finding and replacing text strings in the text edit area. The replace text dialog has two input fields, one to type the text that you are looking for and one to type the text that you want as a replacement. The replace dialog has a check button and five push buttons that mean the following:

  - **case sensitive.** Same as in the Find dialog.
  - **Find Next.** Same as in the Find dialog.
  - **Replace Next.** The next string found after the insertion cursor is replaced by the string to replace with. Before you do a Replace Next, you can do a Find Next so that you see which string you replace. Also note that the string to find should not be empty, but the replacement string may be empty.
  - **Replace All.** All strings that match the string to find are replaced by the string to replace with (global substitution).
  - **Dismiss.** Closes the dialog.
  - **Clear.** Clears both text entry fields.

  In the diagram and table editors the find dialog and the replace dialog are also used for finding and replacing text in the entire diagram or table. These operations are issued from the Find and Replace entries in the Text menu of the main window (see section 2.11).

- **Load (<Ctrl+L>).** This command shows a file selection dialog by means of which you can select an arbitrary text file. When you press `OK`, the contents of the file is loaded into the text editor. You can load any text file of any size.

- **Save as (<Ctrl+S>).** This command shows a file selection dialog for saving the contents of the text editor to a file.

- **Print (<Ctrl+P>).** This command directly sends the contents of the text editor to the current printer. The text is first converted to PostScript and it receives a header. At the moment it is only possible to directly print text with the Print of this dialog in Helvetica font and with point size 9. If you want the text printed differently, you have to save it first to a file and then post-process it yourself for instance with the program `text2ps` that is supplied together with TCM (see `man text2ps`).

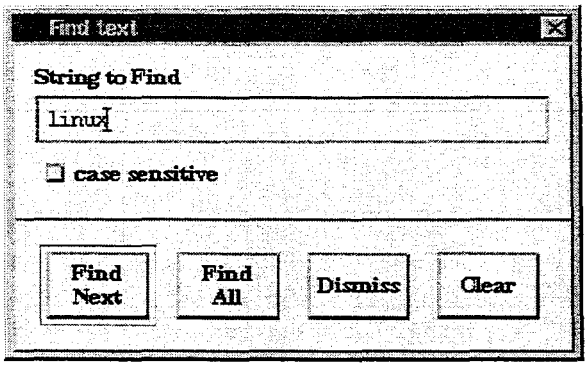


Figure 2.6: TCM find dialog.

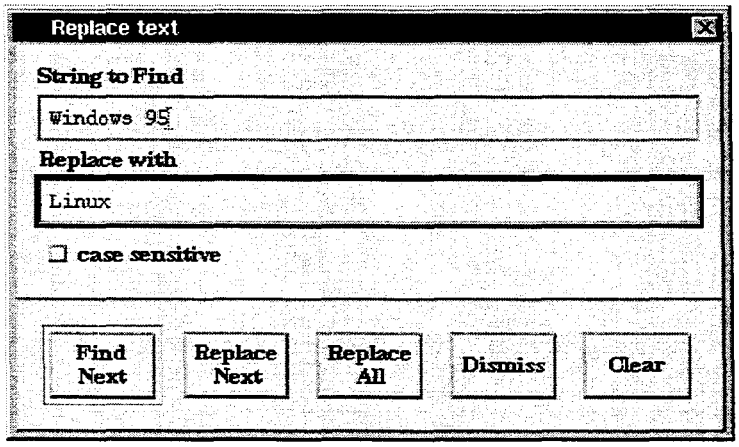


Figure 2.7: TCM replace dialog.

2.6 Viewing Documents

The View menu contains certain commands that have to do with **how** a document is viewed or how it can be edited in the document itself. View commands do not actually change something in the document [3]. The document editors have different View menu entries, but all editors share at least

[3]The Zoom and Grid operation are in separate menus but these are view commands too (as opposed to edit commands).

the following View menu commands:

- The **Refresh** command (<Ctrl+V>) is the first entry in the View menu. It redraws the contents of the drawing area. This command can be needed when pixel droppings or left-overs occur in the drawing area.
- The **In-line editor** toggle. Turns in-line editing on or off. In-line editing means that text editing takes place directly in drawing area whereas out-line editing means text editing takes place in a separate text edit window (see section 2.5.1). It (re)sets also its counterpart toggle in the main window.
- The **Autoresize** toggle. Turns Autoresizing on or off. Autoresizing means that shapes or cells of a table adapt themselves automatically to fit the texts that they contain. It (re)sets also its counterpart toggle in the main window.

2.7 Printing Documents

The page layout determines how a document is sent to the printer or saved as PostScript. See section 2.10 for the commands which determine the page layout. When a document is printed or saved as PostScript, each page that contains a part of the drawing is printed respectively saved. The Print menu contains the following entries:

- **Print (<Ctrl+P>).** You can send the drawing directly to a PostScript printer with this command [4]. The default printer is the value of the `PRINTER` environment variable. When this variable is not set, the printer name in the TCM configuration file `$HOME/.tcmrc` or `$TCM_HOME/lib/tcm.conf` is used instead. In the Printer Options submenu you can see and modify the printer settings that are used and in the Page menu you can change the page layout. In general, the drawing is printed exactly as it is displayed in the drawing area. For seeing what is exactly sent to the printer you can issue the Show Preview command from the same menu.
- **Export (<Ctrl+E>).** This pops up a file selection dialog for saving the picture to some graphics format. Currently, you can save as plain PostScript (the same as what is sent directly to the printer) or as Encapsulated PostScript (EPSF). EPSF is a format that is more suitable than plain PostScript for including drawings into LaTeX or Troff documents. In the dialog there is an option menu called "Format for exported document", in which you can set the intended output format. In the dialog box a default export file name is already filled in. This is the document name with suffix `.ps` or `.eps`. If you attempt to overwrite an existing file, it pops up a dialog from which you can cancel the operation. Furthermore, when you have dismissed the dialog, the last visited directory in this dialog is remembered as well as the last chosen output format in the option menu.
- **Show Preview.** Starts up an external PostScript viewer, for previewing the document as it would be printed. Hopefully this command saves some trees. By default, TCM uses the external PostScript viewer program `ghostview`. But you can define a different default viewer in the TCM configuration file or set temporarily a PostScript viewer in the `Preview Command` dialog from the Printer Options submenu.
- **Printer Queue.** Use this option to see the printer queue of the current printer in a pop-up window. By clicking on a line in the queue you select a print job. The `Remove` button tries to

[4] When you have a printer that can not print PostScript, this command will print a PostScript listing. See the frequently asked questions for how to get a more valuable result.

remove the selected job from the queue. `Update` redisplays the current queue. `Dismiss` removes the window. TCM uses external Unix programs for the printer queue. If it cannot find the right programs, it warns that the queue cannot be viewed or that jobs cannot be removed. The external Unix programs that are used can be changed temporarily in the Printer Options submenu or more permanently in the TCM configuration file.

- **Printer Options.** This is a menu that contains some options for the printer. The default values are read from the file `$HOME/.tcmrc` (if it exists) or from `$TCM_HOME/lib/tcm.conf` otherwise. In the Printer Options menu these values can be changed temporarily i.e. only during the lifetime of the editor. If you want to save the changed options, you have to edit the TCM configuration file. The TCM configuration file is only read when an editor is started.
  - **Printer Name.** Use this to change the printer name. The default printer is the value of the `PRINTER` environment variable. When `PRINTER` is not set, the value from the TCM configuration file is used instead.
  - **Number of Copies.** Use this to change the number of copies that will be print. The default is 1. Change the setting by moving the little slider from left to right. If the number of copies is greater than one then for each time you print one single print job is generated and often the printing will go faster then when you send separate copies.
  - **Print Command.** The default print command is read from the TCM configuration file. The TCM configuration file contains the default print command of the specific Unix variant.
  - **Printer Queue Command.** The default command for showing the printer queue is read from the TCM configuration file. The TCM configuration file contains the printer queue command of the specific Unix variant.
  - **Printer Remove Command.** The default command for removing a job from the printer queue is read from the TCM configuration file. The TCM configuration file contains a command of the specific Unix variant.
  - **Preview Command.** The default command for previewing the PostScript document is read from the TCM configuration file. The TCM configuration file contains the option `PreviewCommand` which has by default the value `"/usr/local/bin/ghostview"`, but other PostScript viewers like pageview or xpsview should work as well.
  - **Print Duplex Pages.** Causes the output to be printed in duplex mode, i.e. pages are printed two-sided if the printer supports that. The binding is as if the resultant pages are to be bound together with their leftmost edge.
  - **Print Tumbled Pages.** This option is only useful with the Duplex option on. It causes the "backside" pages to be flipped relative to the front side pages.
  - **Print Extra Banner Page.** An (extra) PostScript banner page will be printed in front of the printed document. The banner page contains the document name, the current date and the login name of the user. This is useful when the printer does not print banners or when you think that this banner is cool.

2.8 The Zoomer

The Zoom menu contains the following entries:

- **Zoom In (<Ctrl+I>).** Zooms in to the document. The drawing is made larger [5]. Even the fonts are scaled when your X server supports scalable X fonts.

[5] Up to an almost macro-cosmic scale.

- **Zoom Out** (<Ctrl+O>). Zooms out from the document. The drawing is made smaller [6].
- **Home View** (<Ctrl+H>). Returns to normal view, i.e. no zooming.
- **Whole Document.** Zooms out from the document with a percentage that makes that the whole document fits to one page. The page size and orientation can be set in the Page menu.
- **Zoom Factor.** Set the zoom factor with a pop-up slider dialog. The default zoom factor is 1.2.

A **slider dialog** is a dialog that is used to set an editor option from a subrange of values by adjusting a horizontal slider. For an example of a slider dialog see figure 2.8.

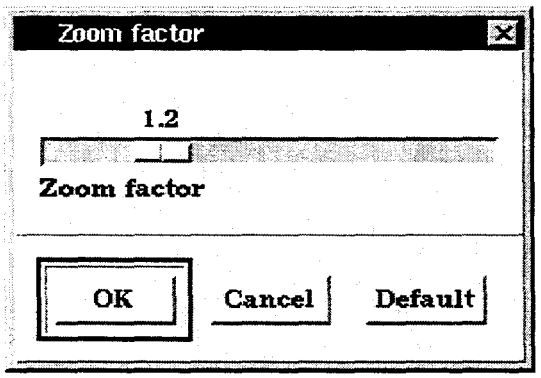


Figure 2.8: TCM slider dialog.

The current zoom percentage is always visible in the bottom-right corner of the main window and is updated by the zoom commands. The **zoom percentage** is the ratio between the TCM coordinates and the X Window coordinates, which is 100% when there is no zooming. By zooming in, you make this percentage bigger, by zooming out, smaller. The **zoom factor** is the factor by which the zoom percentage is increased or decreased during zooming in respectively zooming out. The zoom percentage and factor are settings of the editor not of the document. So when a document is saved to file, the zoom percentage and factor are not stored. But when the drawing is saved as PostScript or as EPSF, the output is scaled by the zoom percentage.

2.9 The Grid

The grid facilitates the alignment of the shapes in the drawing area. Only the diagram and tree editors have the Grid menu because tables are aligned in a different way. The grid cannot be printed. The options of the Grid menu are settings of the editor and therefore they are not saved to file when a document is saved. The Grid menu contains the following entries:

- **Show Grid.** Draw or hide the grid in the drawing area. The grid is visible as a raster of dashed vertical and horizontal lines at an equal distance. This distance is called the **grid size.**
- **Grid Size.** Set the size of the grid with a pop-up slider dialog. The default grid size is 30 pixels.
- **Point Snapping.** When point snapping is on, the positions of the shapes are constrained to discrete positions at a certain **point distance.** When point snapping is off, the shapes can be placed at any position.

[6] Down to an almost subatomic scale.

- **Point Distance.** Set the point distance with a pop-up slider dialog. The default is 10 pixels. The point distance is used when point snapping is on. The grid size and the point distance are independent from each other.

2.10 The Page Layout

The Page menu commands determine the page layout. Any change in page layout is directly made visible in the drawing area [7]. The page layout has effect when plain PostScript is generated either when the document is printed, previewed or exported as plain PostScript. The Page menu commands have no effect on the document when saved as Encapsulated PostScript, as this format is independent from the page layout by definition. The Page menu contains the following entries:

- **Page Orientation.** Sets the page orientation to Portrait (default) or Landscape. When the page boundaries are shown, you see that the boundaries are repositioned. This affects the orientation in which the document is printed or saved as plain PostScript.
- **Show Page Boundary.** Draw or hide the page boundary. The positions of the boundaries are determined by the page size and the page orientation.
- **Page Size.** Sets the page size. The current available sizes are A4 (default, 210x297 mm.), Letter (8.5x11 inches), Legal (8.5x14 inches) and Executive (7.5x10 inches). The default page size can be changed in the TCM configuration file.
- **Include Page Numbers.** This is an option in the Page menu to show page numbers. Page numbers will be shown and printed when that option is on. Page numbers are normally displayed at the bottom of the page, but when the document info is displayed as a footer, page numbers are displayed at the top of the page.
- **Include Document Info.** This is a toggle menu which makes it possible to show information about the document in the drawing area and on paper, because this information will be saved as plain PostScript too. This information can be added as a header or as a footer. The information consists of the document name and type, the creation date and time, the author, the current tool, the current user and the current date and time. By issuing the refresh command or by generating PostScript the information is updated (including the time strings). As an aside, the user cannot alter this information in the editor, except the document name.

The options that can be set in the Page menu are options of the editor not of the edited document. So when you save a document to file, the page settings are not saved.

2.11 The Text Menu

The Text menu contains commands for performing operations with and on text that cannot be done from within the in-line or out-line editor. The following commands are available:

- **Find.** Pops up a find dialog like the Find text operation presented in section 2.5.3. The find dialog looks like figure 2.6, except that the find dialog from the Text menu has an extra check button named `substring`. When this option is set, this indicates that the text to find should not match the entire shape label or cell text but only a substring. Find substring is the default. When the Find string is an empty string then it matches only with empty labels.

[7] TCM is WYSIWYP, what you see is what you print.

- **Replace.** Pops up a replace dialog like the Replace text operation presented in section 2.5. The replace dialog looks like figure 2.7, except that the replace dialog from the Text menu has an extra check button named **substring**. This indicates that the text to find should match only a substring of the entire shape label or cell text. Replace substring is the default. The string to find and the string to replace with can be empty but not both at the same time.

- **Default Font.** This entry shows a menu containing three submenus for setting the default font family, default font style and default point size. Each new shape or table row or column will have this font. Page headers and footers, page numbers and table row and column labels are also drawn in the default font.

  The font families include Helvetica (default), Times, Courier, New Century Schoolbook and Greek. The font styles are Roman (default), Italic, Bold and Bold-italic. The available point sizes are 6, 8, 10 (default), 12, 14, 18 and 24. When the X server has Adobe fonts installed then the X fonts and the corresponding PostScript fonts look the same. The fonts of the PostScript output contain the ISO Latin-1 character set.

- **Update Font.** This entry shows a submenu containing three submenus to update the font family, the font style and the point size. The entries of these menus are commands to alter the font of existing shapes or cells. When you issue one of these commands, each selected shape or cell is redrawn with this new font. Note that a shape in a diagram that has more than one text label cannot have different fonts for the different text labels.

- **Default Alignment.** This entry contains a menu that sets how multi-line texts are aligned by default in diagram and tree editors: Left, Centered or Right. In the case of table editors there is a distinct Default Column Alignment and a Default Row Alignment entry. The default column alignment can either be Left, Center or Right. The row alignment can either be Top, Center or Bottom.

- **Update Alignment.** This menu contains a submenu to update existing multi-line texts. Each selected shape receives this text alignment. In table editors there are distinct entries to update the column alignment and the row alignment.

2.12 Checking and Annotating Documents

The Document menu contains the following entries.

- **Document Info.** Pops up a text view dialog that contains some information about the tool and the document that is being edited.

  The following information about the current tool session is given:

  - Tool name and version.
  - File format version. This is the file format that the tool generates. The tools are supposed to read all file format versions less than or equal to that version.
  - The Unix login name of the user running this editor.
  - The current date and time.
  - The project (working) directory.
  - How many changes (in terms of how many edit operations) were made to the current document after it was created or read in.

  It shows the following information about the document being edited:

- Document type.
- Document name.
- Unix login name of the author. This is the login of the user that created the document.
- Date and time of creation.

If the document was read from a file, it also shows:

- Unix file name from which the document was loaded.
- The file format version found in that file.
- Tool that wrote the file.
- Date and time when the file was written by the tool.

Except the current document name, none of these values can be changed from within the tool. Note that by default the name of the document is equal to the file name when it is saved. When a document file is loaded and it contains a document name that is different from the file name (because the file had been renamed), a question dialog is raised which gives the user the choice between these two names for the new document name.

- **Check Document.** Shows the result of the checks for soft constraints (constraints that can be temporarily violated) in a text view dialog. You should correct the errors yourself. The checks are specific for the concerning document type, so check out the section in this manual where that type of document is explained. For an example of the result of checking an erroneous data flow diagram see figure 2.9.

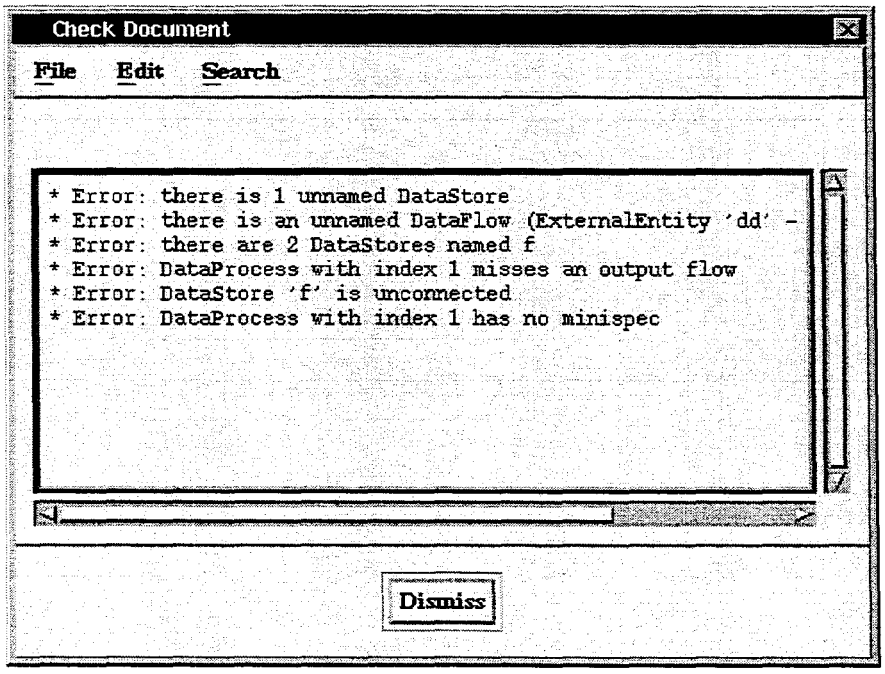


Figure 2.9: The result of check document on a DFD.

- **Annotate Document.** Pops up a text edit dialog for giving the document an annotation. This is free text and can for example contain the description or purpose of the document, the history of the document or references to other documents. The annotation text can be loaded from an ASCII file, saved to an ASCII file and printed (as Postscript). Furthermore, it is possible to Cut or Copy (a part of) the text to the Motif clipboard and it is possible to paste the text from the clipboard in another text edit window. So for example, when you want to add annotation text as a comment to a diagram, you can select the text by dragging with mouse button-1 (the selected text is shown in reverse video) and copy the text from the annotation text edit dialog to the clipboard. Then you open the out-line editor of the comment node (the in-line editor toggle should be off), and you choose Paste from the Edit menu of the out-line editor, the text is pasted from the clipboard into that window. When you press the `OK` button of the out-line editor, the text appears as a comment node.

Both Document Info as Check Document use a **text view dialog** which resembles a text edit dialog except that the text is read-only. So you cannot edit the text (also including Cut, Replace and Paste) and you cannot load text from file. It further works pretty much the same as a text edit dialog. Text view dialogs are also used for the on-line help.

2.13 On-line Help

The on-line help is kept as simple as possible, because it is possible to read this user manual as HTML document, including hyper-links and with an index and a table of contents. It is not our intention to duplicate everything in the form of on-line help built-in in the editors. There is a collection of Help menu entries and each Help menu entry pops up a text view dialog. The on-line help contains the basic bare minimal that you have to know to be able to work with the editor. From the on-line help dialog you can save the text to a file, print it as PostScript or you can copy some of it to the clipboard. Furthermore, there is a Find command. The Help menu of the editors contains the following topics:

- **Getting Started** advises what you can do best when you start up this tool for the first time.
- **Introduction to** *Tool* tells something about the software specification technique that this tool is intended to support.
- **Main window** explains the functions of all the components that you see in the main window.
- **Mouse commands** explains what mouse acrobatics you need to create and update the document of your dreams.
- **Edit menu commands** explains the function of each command in the edit menu.
- **File menu commands** explains loading from file and saving to file.
- **Print & Page commands** tells how you can print or export as PostScript, including how you change the page layout and set the printer options.
- **Miscellaneous commands** contains some other things that are worth to mention like fonts, find and replace, zooming, etc.
- **Version** reveals when and by whom TCM is made.
- **Copyright.** TCM does not have copyleft but copyright. See question A.12 in the FAQ.
- **Change Log.** Show the change log, i.e. the differences between the consecutive versions of TCM.

Chapter 3

Diagram Editing

3.1 Definitions

Diagrams that are made by a TCM diagram editor are a special kind of **graph** with a certain **representation** (layout, view). Each TCM editor is a specialization of a generic graph editor. A graph consists of a set of **subjects**. A subject is either a **node** or an **edge** that connects two nodes. Subjects are of different **node types** and **edge types**. For example, a TDFD data flow diagram graph could contain nodes of type Data Process and nodes of type Data Store, and it could contain edges of type Data flow.

The graphs that are edited in TCM have only edges that connect exactly two (not necessarily different) nodes. So some kind of relationship between more than two nodes, for instance a specialization relationship in an ER diagram is specified in TCM as a node (which is called in this case a taxonomy junction node) that is connected by three or more edges (which is in this case one is-a relationship edge and two or more so-called empty edges).

In a representation of a graph, nodes and edges are shown as **shapes**. In the case of nodes the representing shapes are boxes, diamonds, ellipses etc. In the case of edges, shapes are lines, arrows etc. In general, there is a many-many relationship between a certain subject type and a certain shape type and also between a particular subject instance and a particular shape instance. It is possible that a subject type is represented by more than one shape type. For instance a class relationship diagram made by TCRD can contain different kinds of class boxes. It is also possible that a particular subject instance is represented by more than one shape instance. This is necessary when you want to represent the same subject at different positions in the diagram. But, nevertheless, in most cases, the relationship between subjects and representations is one-one. For this reason, subjects and shapes can be identified most of the time.

A **TCM diagram** is composed of a graph, its representing shapes, and extra diagram information, such as the diagram name, the author and the creation date.

A TCM diagram is empty after creation. You can build it up by interactively drawing it with the TCM user interface, node by node, edge by edge.

TCM documents should satisfy certain **constraints**. Most constraints are specific for the particular diagram technique supported by the editor. Many constraints are checked by TCM. The constraints that are checked, are classified according to how TCM deals with possible violations.

1. **Built-in constraints.** These are constraints which are built-in and which can never be violated because there is no command in the user interface to achieve that. We give three examples of built-in constraints: 1. an edge connects exactly two (not necessarily different) existing nodes,

2. an entity type in TERD is always represented by a box and 3. the label of an is-a relationship edge is always "is_a".

2. **Immediately enforced constraints.** These are constraints that are immediately enforced by TCM if that is possible. When you perform a command that would violate a constraint that is immediately enforced, this command is rejected immediately by TCM and a pop-up window with an error message is displayed. We give two examples of immediately enforced constraints: 1. an entity type and a relationship node in TERD cannot be connected by an is-a relationship and 2. two entity types cannot have the same (non-empty) name.

3. **Soft constraints.** These are constraints that can be violated. Soft constraints are needed because the TCM user interface demands that you draw only one node or one edge at the time, and only edit the label of an existing subject. So there are certain commands that produce diagrams that might violate a constraint and that can not be immediately enforced.

   We give two examples of soft constraints: 1. all entity types have a non-empty name and 2. a relationship node in TERD should be connected to entity type nodes by two or more different edges.

   Soft constraints are checked by TCM when the **Check Document** command is issued by you. Check Document displays a list of error messages in a pop-up window. As opposed to the previous two classes of constraints, you are responsible for correcting the diagram.

   It is possible that some immediately enforced constraints can still be violated by some user command. Such a constraint is then classified as both an immediately enforced and a soft constraint. Example: if a part of a diagram is copied and pasted into the same diagram, the unique name constraint is a soft constraint as it can be violated by the paste command. But when you have edited the text of one of the nodes in the diagram, and you issue the stop edit command, the unique name constraint can and will be immediately enforced.

3.2 Creating Nodes

The set of tiled buttons labeled **Nodes** contains radio buttons indicating the current type of node that can be created. You can change the current node type selection by clicking another button with button-1.

For creating a node, first choose the desired node type from the node tiles. Click button-1 in the background of the drawing area, at the desired position. A new node shape appears, having its center at the position where is clicked and having the shape as indicated by the selected node type in the tiles. The node shape is surrounded by a set of **selection handles** which are black or grey rectangles, showing that the shape is selected. Selection handles of a node shape are also used for resizing the node.

3.3 Creating Edges

The set of tiled buttons named **Edges** contains radio buttons indicating the current type of edge that can be drawn. Furthermore there is a so-called check button labeled **Curve**. If the curve button is on, each new edge that is created, having more than one segment, is drawn as a Bezier curve. If the curve button is off, a multi-segment line is drawn as a set of connected straight line segments. To create a new edge, first choose the desired edge type in the edge tiles. There are three sorts of edges that you can create:

- **Straight edges.** Drag with button-2 from a node to a node. This means you push and hold down button-2 somewhere inside the source node and move the mouse with button-2 still pushed down. During dragging the mouse pointer turns into a + and a line from the source node follows the movement of the mouse. You can always abort creating an edge while dragging by clicking button-1. When you are arrived in the target node and release button-2, then a straight edge is created. The two points of the edge at the borders of the connected nodes are called the **end points** of the edge.

- **Segmented edges.** These edges consist of more than one segment. The points where the segments are connected are called **intermediate points.** To create such a segmented edge, release button-2 in the background while you are dragging. Then the first intermediate point is created. After that, the line follows the mouse all by itself. Then you can add another intermediate point by clicking button-2 somewhere else in the background. You complete the edge by moving into the target node and clicking button-2.

- **Curved edges.** You can also create an edge drawn as a Bezier curve [1]. For this, turn on the Curve toggle in the tiled menu. The Bezier curve requires *exactly two* intermediate points, which makes four edge points in total. These intermediate points are added in the same way as creating normal multiple line segments as described for segmented edges. When the four points are determined, a smooth curve is drawn between the end points and the two intermediate points act as controlling points.

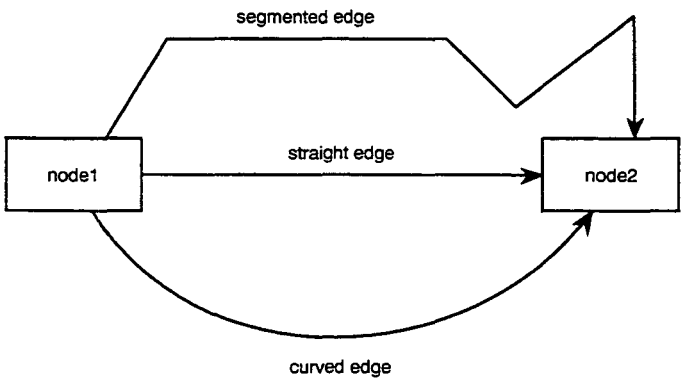


Figure 3.1: Three forms of edges.

After creating the edge, the edge is selected which is visible by a selection handle on each line point. You can adjust the line points by moving the handles; see section 3.6.

When an edge is created and it is the only *straight* edge connecting this particular couple of nodes, it will go (virtually) through the center points of the nodes. When there are multiple straight edges connecting the same pair of nodes, they will be equally distributed by default over the opposing sides. Segmented and curved edges are not equally distributed when they connect the same pair of nodes. TCM tries to intersect the first and last segment of a segmented line orthogonally with the sides of the connected nodes. If this is geometrically impossible, the segment will be directed to the center of

[1] See http://www.moshplant.com/direct-or/bezier for more information about Bezier curves.

the node like with straight edges [2].

Tip: When you want to enforce that a straight edge is always orthogonal to a node, then you can draw a segmented line of three points that looks like a straight line. You enforce then that the line is always connected orthogonally with both of the nodes (providing that this is geometrically possible). This gives often a more pleasant optical result when you want to connect two nodes that differ a lot in size.

It is possible to move the end points of an edge as well, in the same way as you move the intermediate points. When you drag the handle of an end point to another position then the new end point will be the position at the border of the node shape that is the closest to where you released the handle. However, when you drag the handle of an end point into *a different* node shape then the edge will be **redirected** to that node.

3.4 Selection Commands

The current selection is a subset of the displayed shapes. Shapes in the selection have visible selection handles. The shape in the selection that was first selected has black selection handles and the other shapes in the selection have grey selection handles. The following operations on the selection exist:

- **Select a single shape.** Click button-1 on an unselected shape (node, edge). The selection will contain only this shape which will show black handles.
- **Add a shape to the selection.** Click button-2 on an unselected shape. The first selected shape will show black selection handles, the other selected shapes will show grey selection handles.
- **Remove a shape from the selection.** Click button-2 on a selected shape. The selection handles disappear.
- **Empty the selection.** Click button-2 somewhere in the background. All selection handles disappear.
- **Select a part of the diagram.** Drag button-1 starting in the background. The area that you are selecting, is enclosed by a stippled box, and its bottom-right corner is attached to the mouse. If you release button-1 then every shape that was (partly) inside the stippled box is selected and every other shape outside the box is unselected. One of these selected shapes will show black selection handles, the others will become grey.
- **Make selected shape first selected.** You can enforce that a selected shape is the first selected by clicking button-1 on an already selected (grey) shape. The selection handles of the old first selected shape will become grey and the clicked shape will become black.
- The **Select All** command from the Edit menu selects all shapes of the diagram. By clicking button-2 in the background you deselect all shapes.

3.5 Editing Text

For getting into edit mode make sure that the shape with the label that you want to edit is the only selected shape. When you move the mouse pointer into the single selected shape, the mouse pointer turns into a I. By clicking button-1 on the shape or typing the first character you go into edit mode.

[2] Please check this out yourself, reading this manual is certainly not the proper way for learning these kinds of things. This is like swimming, which can't be learned by only reading a manual.

See section 2.5.1 for the different text edit commands that exist in TCM. These apply to all document editors including all diagram editors.

Each TCM diagram and tree editor has a special node type called **Comment**, which solely consists of a text label and is intended to add comment to the diagram. This node type cannot be connected by any other edge, except in TGD, the generic diagram editor. But comment nodes can be selected, moved and of course edited.

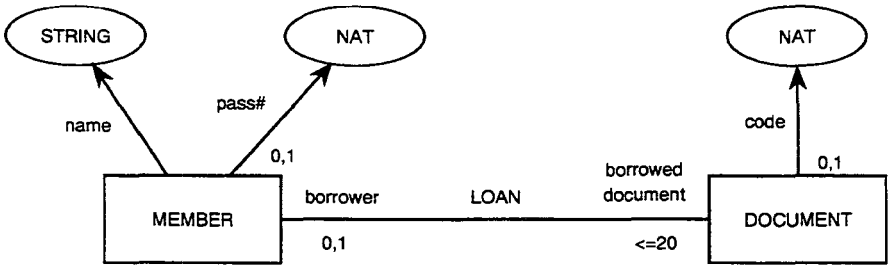


Figure 3.2: Example diagram with some text labels.

Tip: If you want to edit the label of an edge, and you try to select it first with button-1 you could miss the edge and create an unwanted node. To avoid this, make the selection empty and select the edge by button-2 instead, because, when you miss then no node is created.

3.6 Moving Shapes

You can reposition the following things in a diagram:

- **A node shape.** You can move a node shape by positioning the mouse into that node shape and then drag it with button-1. The shape does not need to be selected. A grey node outline moves along with the mouse. The mouse pointer turns into a ✣. If you release button-1 then this node shape is redrawn at the new position and all its adjacent lines are redrawn too.

- **The selection.** If you drag one of the selected node shapes, all other selected shapes are moved with over the same distance, including the line handles of the selected edges.

- **Intermediate line handle.** You cannot move the line as a whole, but you can drag the individual handles of a line. When a line is selected and the mouse is in one of the handles (which is indicated by a ✣), then you can drag this handle and a grey outline of the new line follows the mouse. You can move the intermediate handles of a line to an arbitrary place in the drawing area.

- **End line handle.** When you drag and release one of the end point handles into the connected node or somewhere in the background, then the end point will be placed at the border of the connected node at a position that is the closest to the place where the handle was released. However, when you drag the handle into a different node then the edge will be **redirected** to that node. The result is that the edge now connects to a different node [3].

[3] For the philosophers amongst us: is it still the same edge when it connects to a different node?

- **The label of an edge.** You can move the label of an edge by dragging it with button-1. The edge need not be selected. You cannot move the label of a node; when you move or resize a node, the label positions inside the node are recalculated, as well as the labels of the adjacent edges.

- The **Align** command in the Edit menu has six variants as you can see in the Align submenu of the Edit menu. The alignment takes place according to the position and size of the first selected shape, which should be a node shape. Depending on the chosen alignment type, the rest of the nodes in the selection will be aligned to the first shape, either horizontally (Same center y-coordinates), vertically (Same center x-coordinates), to the top (Same top y-coordinates), to the bottom (same bottom y-coordinates), to the left (Same left x-coordinates) or to the right (Same right x-coordinates).

When you want to abort a move command, while you are moving, you have to click button-2.

3.7 Resizing Shapes

Resizing a node shape is possible by dragging button-1 on the selection handles. The mouse pointer turns into one of the eight symbols that indicate to what direction the node shape can be resized:
⇖ ↑ ⇗
← →
⇙ ↓ ⇘. Resizing a node shape is possible without selecting it, although you do not see the selection handles then. When you want to abort the command while you are resizing, you have to click button-2.

Unlike moving, resizing only works on one shape at the time i.e. it does not influence the other shapes of the selection. If you want to resize multiple shapes in one command then you could use the Same Size command in the Edit menu. The **Same Size** command makes the size of all selected node shapes the same size as the first selected shape (the shape with the black handles). The first selected shape should be a node shape of course. The edges in the rest of the selection are ignored by this command.

Tip: when node shapes have been moved or resized then their adjacent edges are redrawn and all their handles and labels are placed back to their default positions. This is sometimes undesirable [4]. To avoid that your effort gets lost, postpone moving edge labels and handles to their final positions until when you think that the connected nodes are OK. Draw the graph first and then you can bother about the layout.

3.8 Deleting Subjects

The edit command **Delete** (called from the Edit menu, or by the accelerator <Ctrl+D>), removes all subjects from the selection. If a node is deleted then all its adjacent edges are deleted too. If a node has duplicate shapes (see section 3.10) of which some but not all are selected then a question dialog is raised asking whether you want to delete all duplicates (and henceforth the node in the graph) or that you only want to delete the selected duplicate shapes.

The Edit menu command **Delete All** removes all subjects, selected or not. Before the command is executed, you are first asked by means of a question dialog if you really want to delete your masterpiece.

Fortunately, like all commands, deletion commands can be undone with Undo.

[4] But imagine the situation that labels do not move when nodes and edges are moved, that would be far more frustrating.

3.9 Cutting and Pasting Subjects

The diagram editors have a **buffer** for cutting, copying and pasting parts of a diagram (selected shapes plus the subjects that they represent) inside the same editor.

You can either **Cut** (<**Ctrl+X**>) or **Copy** (<**Ctrl+C**>) the selection to the buffer. When you perform a cut, all selected subjects are copied into the buffer, and the selection is removed from the diagram (including the unselected edges that are adjacent to the selected nodes). When you perform a copy, no subjects are removed from a diagram, but a copy of the selection is made and put into the buffer. Both Cut and Copy copy the unselected nodes that are connected by selected edges into the paste buffer.

Pasting means that the contents of the paste buffer is being copied into the diagram. When you perform a **Paste** (<**Ctrl+Y**>) command, a stippled box is shown that has the size of the pasted area which is attached to the mouse pointer near its top-left corner. When you click button-1, the subjects in the paste buffer are copied into the diagram. The Append Diagram command is also implemented as a paste command. When you want to abort pasting while you are moving the paste box, you have to click button-2.

You can paste the same contents of the paste buffer more than once because always a copy of the contents is being made. Pasted nodes and edges are new nodes and edges, not duplicates of existing nodes or edges. The paste buffer remains intact when you load another diagram, so it is possible to cut and paste between different diagrams (although in the same editor).

3.10 Creating and Deleting Duplicates of a Node

It is possible to represent one node by several instances of the same shape. The **Duplicate** command copies the node shapes in the selection and puts them into a paste box [5]. The copied node shapes can be positioned at an arbitrary place, just like with the Paste command. Edges cannot be duplicated yet [6]. Unlike the Copy command, the new node shapes have the same node subject as the original. When a node has duplicate shapes then all these shapes have a small asterisk in the top-left corner.

When the label of a duplicate shape is updated then all its duplicate shapes are updated too and when a node is deleted (with Delete) then all the duplicate shapes of this node are deleted too by that command.

The duplicate command is intended to be used when the same node has to be represented at different places to avoid cluttering up the diagram. For instance, in a DFD, the same external entity or data store is often drawn at different places in the same diagram, and TCM keeps track that these are in reality the same subjects.

The **Delete** command can be used to delete all selected duplicate shapes, see section 3.8. When all the shapes of a node are deleted then the node itself is deleted too. But you can delete individual duplicate shapes as well.

3.11 Other Edit Commands

- The **Annotate subject** command pops up a text edit dialog in which you can type arbitrary text to annotate the subject. See section 2.5 for using the text edit dialogs.
- With the **Find** menu entry in the Text menu you call a dialog in which you can search for text in the diagram. The find dialog is described in section 2.5.3. From this dialog you can **find the next text** or **find all texts** that matches the string to find. In the first case, the first

[5] But the duplicate command does not use the paste buffer.

[6] Duplicate lines are just as meaningful, so they are added to the TCM wish list.

shape that is found is selected and the scrollbars of the main window are moved to center the shape in the main window. When you click `Find Next` again, the following shape is selected (round-robin). When you choose `Find All`, all shapes that contain a string that matches are selected.

- With the **Replace** menu entry in the Text menu you call a dialog in which you can replace texts in the diagram. The replace dialog is described in section 2.5.3. It has a find next command that works the same as in the find dialog. Furthermore, the replace dialog has a replace next and a replace all button. **Replace next** means that for the next shape (the shape that is found with find next) all their text strings that match are substituted by the string to replace (that is the second string filled in in the dialog). In the case of **Replace all** this happens to all shapes in one command (global substitution).

  Note that find and replace work on entire shapes at the same time. But keep in mind that a shape could have multiple text labels and each individual text label could match the string to find or the string to replace multiple times (at least when you look for a substring). When you want to find or replace within a single text label or table cell, you should load that text label first in the out-line text editor and then do a find or replace within that dialog.

  Note also that you can fill an empty string for the string to find in both dialogs. An empty string matches only the text shapes that are empty. You can also fill in an empty string in the replace with text field, providing that the string to find field is not empty.

3.12 Undo and Redo

The last issued edit commands can always be undone and, when undone, also redone again. **Undo** is the first entry of the Edit menu (in the menu bar and also under button-3). Undo has the accelerator <Ctrl+U>. Undo is multiple levels deep. **Redo** is the second entry of the Edit menu and has the accelerator <Ctrl+R>. You can redo the last undone commands. Redo is also multiple levels. However, when you have undone a command and then issued a new command then the undone command can not be redone anymore. Both undo as redo show in their menu entry the name of the command that can be undone respectively redone. In the current implementation the undo can be several hundreds of levels deep, but when a new diagram is created by Load or New, the undo history is deleted. When undo or redo is not possible, the corresponding menu entry is shaded and can not be issued.

While you are busy with performing a command, like creating an edge, moving the paste box or resizing or moving some shapes, you can abort the command by clicking button-1 during edge creation or button-2 for the other commands. When you have aborted, nothing is changed and the undo or redo of that command is not possible nor necessary and the undo menu entry lists the command issued before the aborted command.

Figure 3.3 summarizes all atomic commands by which you can edit the diagram. These commands are either issued by the mouse, via the Edit or Text menu, or via an accelerator. Each of these commands is undo- and redo-able.

3.13 The Generic Diagram Editor (TGD)

The generic diagram editor is a diagram editor that has exactly all the features that are described in this chapter.

Command	User Action	Command	User Action
Align nodes	Select nodes and issue an entry from Align submenu of Edit menu.	Find all texts	Isue Find in the Text menu. Fill in the text to find and click the Find All button.
Annotate subject	Select a subject, issue Subject Annotation from the Edit menu. Fill in the annotation and click OK.	Find next text	Isue Find in the Text menu. Fill in the text to find and click the Find Next button.
Append diagram	Issue Append from the File menu, choose a file, click OK and paste it.	Make nodes same size	Select nodes and issue Same Size from the Edit menu.
Copy to buffer	Select shapes and issue Copy from Edit menu.	Paste from buffer	Issue Paste from the Edit menu. Release the paste box at the desired position.
Create edge	Drag button-2 between nodes	Resize node	Drag with button-1 on the handles of a selected node.
Create node	Click button-1 in the background	Resize node	Drag with button-1 on the handles of a selected node.
Cut to buffer	Select shapes and issue Cut from the Edit menu.	Replace all texts	Issue Replace from the Text menu, fill in the texts to find and to replace with and click the Replace All button.
Delete all	Issue Delete All from the Edit menu.	Replace next text	Issue Replace from the Text menu, fill in the texts to find and to replace with and click the Replace Next button.
Delete subjects	Select one or more duplicate nodes and issue Delete from the Edit menu and click No in the dialog.	Select all	Issue Select All from the Edit menu.
Drag edge handle	Select shapes but no duplicate nodes and issue Delete from the Edit menu or select one or more duplicate nodes and issue Delete from the Edit menu and click Yes in the dialog.	Select area	Drag button-1 in the background.
Drag edge handle	Drag button-1 on an edge handle (edge will be redirected when an end handle is dragged into another node).	Update text alignment	Select shapes and issue an entry from the Update Alignment submenu of the Text menu.
Drag selection	Drag button-1 on a selected node.	Update font family	Select shapes and issue an entry from the Update Font Family submenu of the Text menu.
Drag shape	Drag button-1 on a node.	Update font style	Select shapes and issue an entry from the Update Font Style submenu of the Text menu.
Drag edge label	Drag button-1 on an edge label.	Update point size	Select shapes and issue an entry from the Update Point Size submenu of the Text menu.
Duplicate nodes	Select nodes and issue Duplicate from the Edit menu. Release the paste box at the desired position.	Update text	Go into edit mode, edit the text and click OK in the text edit dialog or stop the in-line editor

Figure 3.3: All atomic diagram edit commands.

3.13.1 Nodes and Edges

The generic diagram editor has just two node types **Generic Node**, and **Comment**, and one edge type, **Generic edge**. It has no immediately enforced and no soft constraints. A generic node can be represented by a number of node shapes and a generic edge can be represented by a number of line types, see figure 3.4.

Note that comment nodes may be connected by edges as well. And given the fact that you can make a comment node invisible by removing its text, you can make rather fancy drawings with TGD.

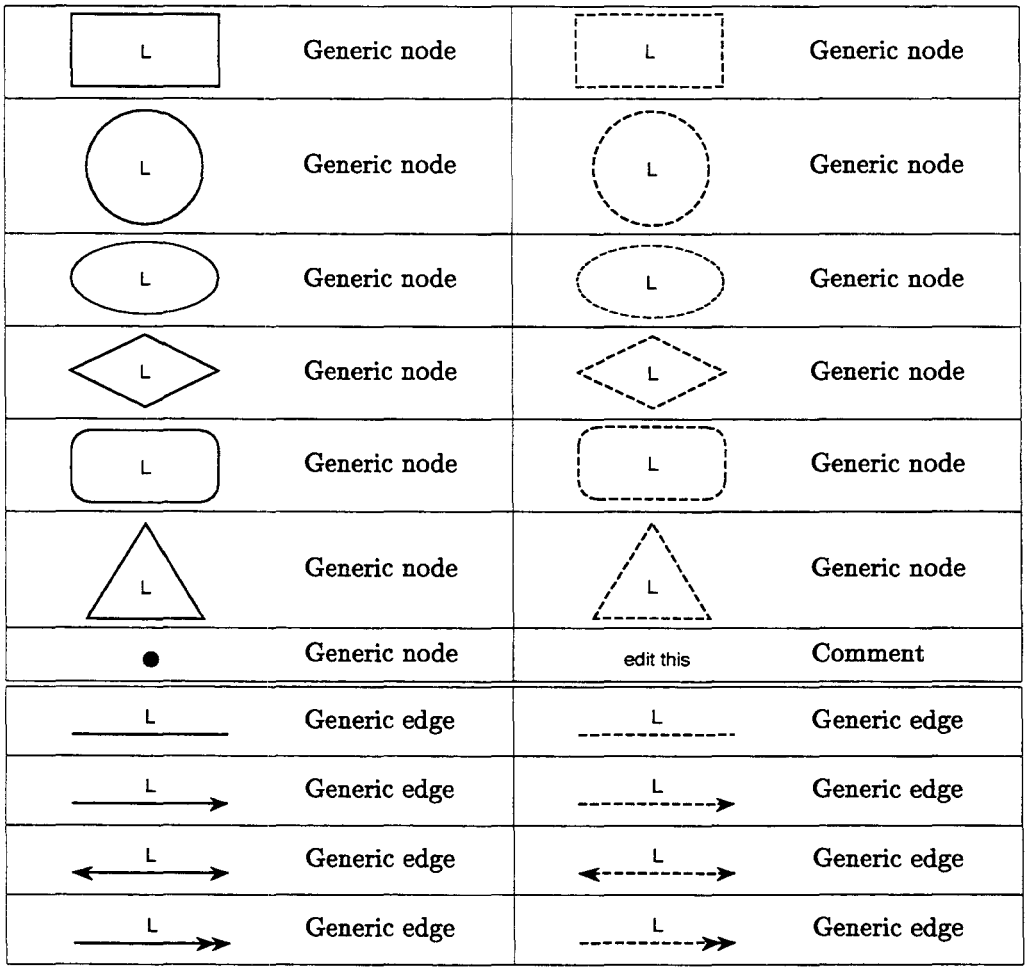


Figure 3.4: Generic diagram nodes and edges.

Chapter 4

Data View Editors

4.1 The Entity-Relationship Diagram Editor (TERD)

4.1.1 Nodes and Edges

See figure 4.1 for the subjects and the representing shapes that are used in TERD. In figure 4.2 you can see which node types can be connected by which edge types. The constraints of which node types can be connected by which edge types are immediately enforced. Note that function edges can be represented by two shape types, uni-directional (single) arrows and bidirectional (double) arrows. The latter denotes a one-one relationship. So, in figure 4.2 a single function arrow may also be a bidirectional arrow.

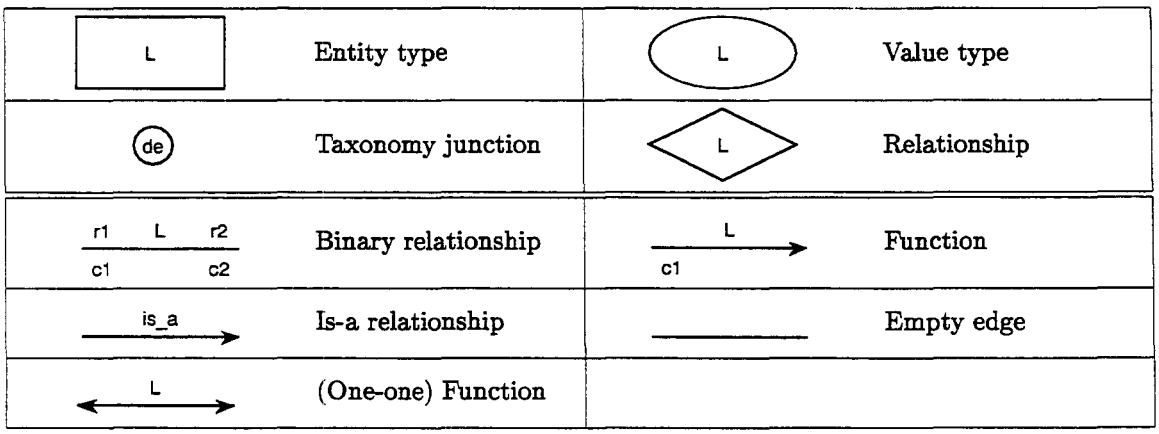


Figure 4.1: Entity-relationship diagram nodes and edges.

4.1.2 Cardinality Constraints and Role Names

Most edge types have a name label, which is shown as an L in figure 4.1. The default position of a name label is near the center of the middlemost segment of the edge. The middlemost segment is "number of segments div 2". When the label is first edited, the default position becomes the center of the segment where the line is selected for editing. When the line is not vertical, the name label is by

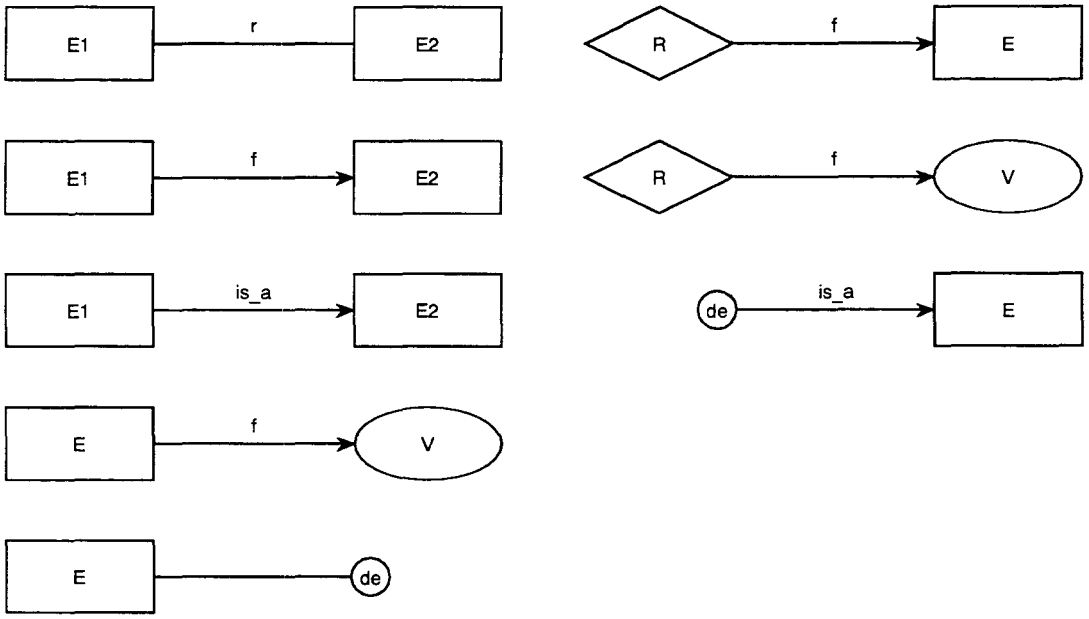


Figure 4.2: Permitted Entity-relationship connections.

default positioned at a fixed distance above the line, preventing that the label goes through the line and becomes unreadable. When the line is vertical, the label is positioned slightly to the left of the line, preventing that a short label goes through the line and becomes unreadable. Some edge types like Binary relationships and Functions also have **role names** and/or **cardinality constraints**. See figure 4.3 for the default positions of the roles names (`r1` and `r2`) and cardinality constraints (`c1` and `c2`). These labels are editable just like the name labels. The role name `r1` and the cardinality constraint `c1` have their default positions near the middle of the first quarter of the first segment of the line. `r2` and `c2` have their default positions near the middle of the last quarter of the last segment of the line [1]. When the line is more or less horizontal, role names are positioned above the line and when the line is more or less vertical, role names are positioned at the left side of the line (just like the name label). Cardinality constraint labels are positioned at the side opposing the role name labels. See figure 3.2 for an example ER diagram with role names and cardinality constraints.

Remark: A binary relationship whose c2 cardinality constraint is 1, is equivalent to a function edge. The difference is that, instead of a 1, an arrow head is drawn. Nevertheless, TERD considers them as separate edge types.

TERD enforces immediately that cardinality constraints conform to the BNF syntax of figure 4.4. But TERD does not check if subranges are contradictory such as the constraints `1..0` and `2,>=3`.

4.1.3 Taxonomic Structures

A **taxonomic relationship** (also called specialization, generalization, inheritance or is-a hierarchy) is constructed from a taxonomy junction node connected to an entity type by a single is-a relationship

[1] To be even more pedantic than usual: what is the first and what is the last segment of a line is actually determined by the direction in which the edge was originally drawn. For undirected edges like a binary relationship this direction is irrelevant, as both sides have possibly a cardinality constraint and a role name. For directed edges like functions, this direction is visible by the shape of the line.

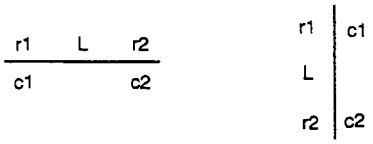


Figure 4.3: Default label positions of a binary relationship edge.

constraint	→	*constraint* , *expression*
	|	*expression*
expression	→	*number* .. *number*
	|	*number*
	|	< *number*
	|	<= *number*
	|	>= *number*
	|	> *number*
number	→	$digit^+$
digit	→	**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**

Figure 4.4: Cardinality constraint syntax.

edge and connected to two or more other entity types by two or more empty edges. The entity type where the is-a relationship edge ends is the **generalization** (or supertype), the other entity types are **specializations** (or subtypes). See, for example, figure 4.5. If two entity types are is-a related then a single is-a relationship arrow can be drawn between them, but when an entity type is specialized into more than one different entity type then it is customary to draw one compound is-a relationship with a taxonomy junction.

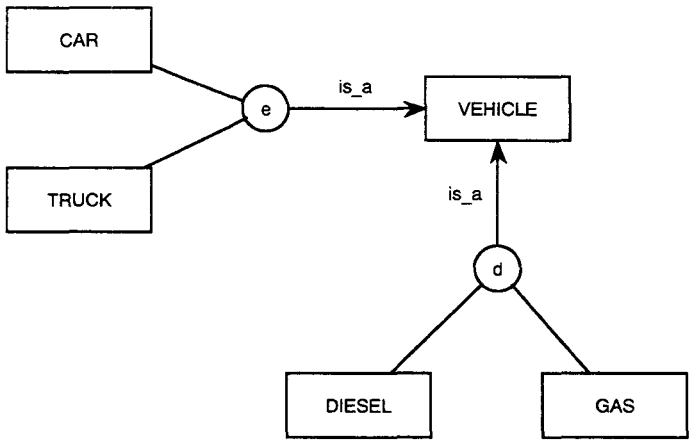


Figure 4.5: Example taxonomic structure.

The **Show Is-a Only** toggle in the View menu of TERD shows the shapes which constitute the is-a hierarchy. When you turn it on, the shapes that are not part of the hierarchy are made invisible. When you turn it off, all shapes are made visible again.

An is-a relationship has a built-in name label "is_a", an empty edge does not have a label. Taxonomy junctions should have one of the following labels : "", "d", "e" or "de". A "d" stands for disjunctive and an "e" stands for exhaustive. These constraints are immediately enforced.

4.1.4 Constraint Checking

In addition to the connection constraints, the syntax of cardinality constraints and the other graphical conventions that TERD enforces, TERD checks a lot of immediately enforced and soft constraints. These are summarized in figure 4.6.

Some of these constraints are enforced immediately during most but not all editor commands. If that is the case, they are additionally checked by Check Diagram as a soft constraint.

Note that the unique name constrains do not concern duplicate node shapes. This is one of the reasons for the existence of duplicate shapes.

4.2 The Class-Relationship Diagram Editor (TCRD)

4.2.1 Nodes and Edges

See figure 4.7 for the subjects and the representing shapes that are used in TCRD. In figure 4.8 you can see which node types can be connected by which edge types. These are immediately enforced constraints. Note that object classes can be represented by three different box types. In figure 4.8

Constraint	Check	Description
ER1	Immediately	Names of entity types, relationships and functions contain at least one letter.
ER2	Immediately	Taxonomy junctions have one of the labels: "e", "d", "de" or "".
ER3	Immediately	Non–specialization relationships should not be labeled "is_a".
ER4	Immediately/Soft	Taxonomy junctions are properly connected (one is_a edge, two or more empty edges, all to different subjects).
ER5	Immediately	Specialization relationships should not contain cycles.
ER6	Soft	Relationships nodes are properly connected: they have at least two connections to an entity type.
ER7	Immediately/Soft	Entity types should have mutually unique non–empty names.
ER8	Immediately/Soft	Name+component types of a relationship should be unique.
ER9	Soft	Binary relationships (edge) should either have a non–empty name or at least one non–empty role name.
ER10	Soft	Relationship nodes should be named.
ER11	Soft	Value type nodes should be named.
ER12	Soft	Value type nodes should be connected by at least one named function edge.
ER13	Immediately	Edges from the same subject to the same value type node should have different names.
ER14	Immediately	Edges from the same relationship node to the same subject should have different names.

Figure 4.6: Immediately checked and soft constraints on ERDs.

only the double box type variant is shown but each double box could be replaced by a single or triple box.

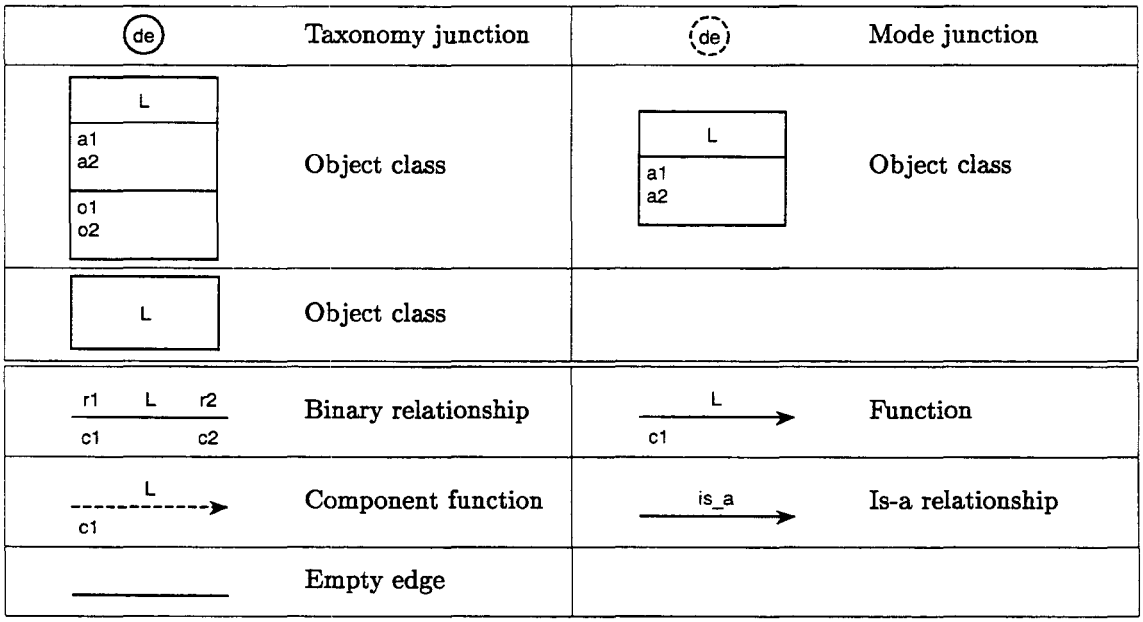


Figure 4.7: Class-relationship diagram nodes and edges.

4.2.2 Classes and Relationships

The object class node type of TCRD can be seen as an extension of the entity type node type of TERD. Like TERD, TCRD has binary relationships and functions. Value type nodes are not needed in TCRD because attribute names and value types are now included in the object class in the form of attribute definitions. Also unlike TERD, TCRD does not have a separate node type for relationships. Relationship nodes can be represented by the same type of class box as an object class box. The only difference between a real object class instance and a relationship instance, is that the latter has two or more **components**. A relationship instance is called a **link**. The components make up the identity of a link. The components of a link are objects or other links. The component relationship is a projection function represented by a dashed arrow. This function is called a **component function**. Like ordinary functions, component functions can have a cardinality constraint label (see figure 4.9). There is no other visual difference between an object class instance and a relationship instance in TCRD than the component functions. So, in TCRD a link can have attributes, actions and it can be part of a taxonomic structure.

TCRD inherits all the applicable constraints of TERD, on the understanding that object classes in TCRD replace the entity types from TERD. Furthermore, see section 4.1.2 for the use of cardinality constraints and role names, see section 4.1.3 for the use of taxonomies and see figure 4.6 for all the other constraints that are checked in TERD.

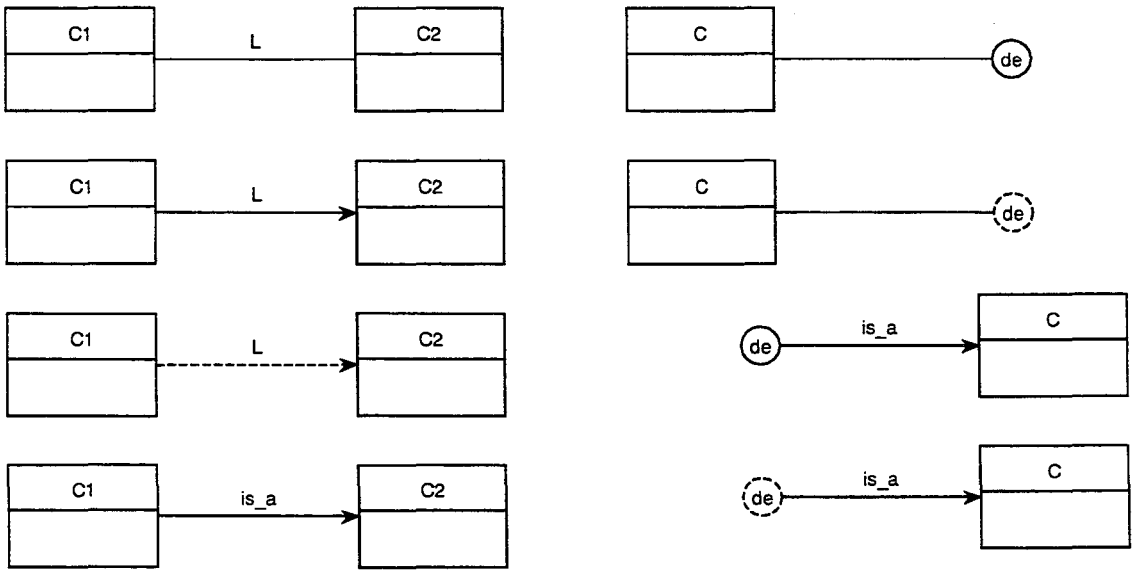


Figure 4.8: Permitted Class-relationship connections.

Figure 4.9: Example CR diagram, showing a relationship class.

4.2.3 Attributes and Actions

The class-relationship editor TCRD offers object classes that can have a number of **attributes** and a number of **actions** [2]. See figure 4.10 for an example.

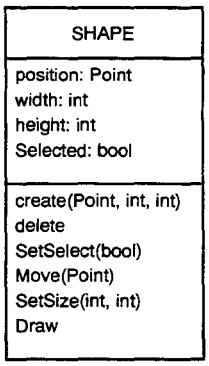


Figure 4.10: Example object class with attribute and action definitions.

An object class can be represented by a single box in which only the name label is visible, by a double box in which the name label and all the attributes are visible, or by a triple box in which the name label, all the attributes and all the actions are visible. For each representing shape type there is a separate tiled button in the list of nodes in the main window, but they are all representations of the same node type, Object class. You can change the representation of an existing object class by the **change box type** command in a submenu of the Edit menu. These commands modify only the shapes of the selected boxes, The other shapes, unselected boxes and shapes that are not boxes, are not updated. These commands only change the shape, the attributes and actions are preserved.

Each attribute definition and each action definition occupies a *single* text line. Any text on a single line in the appropriate part of a box is considered as an attribute or action definition. Attribute names are in the second compartment and action names are in the third compartment of the class box. You can perform the following operations on attributes and actions:

- **Append one or more attributes or actions.** To add attributes or actions to the end of the attributes or actions list of an object class, click the mouse pointer in the empty part at the bottom of the compartment in the class box and start editing. You can add more than one attribute or action at a time by separating them by newlines. When you stop editing and the autoresize is on, the size of the object class box is adjusted to the width and the height of the new list of attributes and actions and the new attributes and actions in the box are always left adjusted.

- **Insert one or more attributes or actions.** To insert new attributes or actions somewhere in the current attribute or action list, click the text line that should precede them and go into edit mode. First enter a newline and then add the new attributes or actions like described above.

- **Update an attribute or action.** To update an existing attribute or action, click its text line and edit it.

[2] In the past, an action in TCM was called **event** or **transition**. You never know, some day we might prefer yet another name ... how about *operation* or *member function*?

- **Remove an attribute or action.** To remove an existing attribute or action, click its text line for editing and delete the text with <Escape> (in-line editor) or do a delete all in the out-line editor. If you stop editing, the attribute or action is deleted and the size of the class box is adjusted.

4.2.4 Taxonomic Structures

Like TERD, TCRD has taxonomic structures with taxonomy junctions, is-a relationships and empty edges. TCRD offers a second kind of taxonomy junction which is called **mode junction** and which is represented by a small dashed circle instead of a small solid circle. Mode junctions are used for constructing **mode specializations**. For an example mode specialization, see figure 4.11. The concept of mode specialization as a form of type migration is explained in appendix C. Except the different shape of the circle, mode specializations are drawn in the same way as static specializations, having the same constraints as in section 4.1.3.

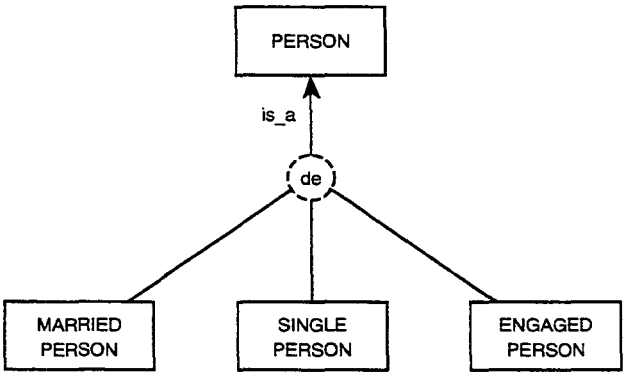


Figure 4.11: Example mode specialization.

An is-a relationship (without a junction) between two object classes, and an is-a relationship connected to a taxonomy junction are considered as **static specializations**, whereas is-a relationships connected to a mode junctions are considered as **dynamic specializations**. Both kinds of specializations can be combined, with the exception that a mode type should not be specialized statically. This is an immediately enforced constraint.

4.2.5 Constraint Checking

TCRD checks also the soft and immediately enforced constraints that are summarized in figure 4.12. Note that TCRD checks a lot of constraints but it does not check yet all plausible constraints on CRDs. For instance, name conflicts between actions and attributes of classes that are is-a related are not yet discovered and the syntax of attributes and actions is still very liberal.

Constraint	Check	Description
CR1	Immediately	Names of object classes, relationships and (component)functions contain at least one letter.
CR2	Immediately	Taxonomy junctions and mode junctions have one of the labels: "e", "d", "de" or "".
CR3	Immediately	Non-specialization relationships should not be labeled "is_a".
CR4	Immediately/Soft	Taxonomy junctions and mode junctions are properly connected (one is_a edge, two or more empty edges, all to different subjects).
CR5	Immediately	Specialization relationships should not contain cycles.
CR6	Immediately	A static partition (with a taxonomy junction) should not be part of a dynamic partition (with a mode junction).
CR7	Soft	Binary relationships (edge) should either have a non-empty name or at least one non-empty role name.
CR8	Immediately/Soft	An object class without component functions cannot be specialized into a relationship object class.
CR9	Immediately/Soft	Names of object classes should be non-empty and mutually unique.
CR10	Immediately/Soft	When two (component) functions connect the same pair of object classes then they have mutually unique names.
CR11	Immediately/Soft	When two relationship edges connect the same pair of object classes then they have mutually unique names.
CR12	Soft	From an object class either depart zero component functions or more than one component function.
CR13	Immediately	Each attribute and each action definition contains at least one letter and no newlines.
CR14	Immediately	The action and attribute definitions in the same object class are mutually unique.
CR15	Immediately/Soft	Names+component types of relationship edges should be mutually unique.

Figure 4.12: Immediately checked and soft constraints on CRDs.

Chapter 5

Behavior View Editors

5.1 The State Transition Diagram Editor (TSTD)

5.1.1 Nodes and Edges

TSTD follows the notational convention of Mealy machines. In short, a **Mealy machine** is a finite state machine in which each transition can have one or more actions. See figure 5.1 for the shapes and subjects.

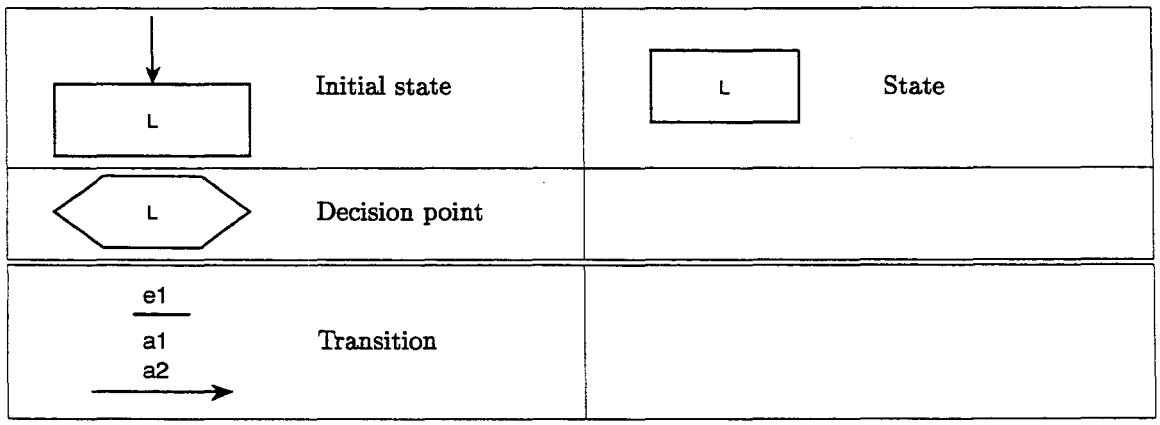


Figure 5.1: State transition diagram nodes and edges.

5.1.2 States

There are tree kinds of nodes: **initial states**, (normal) **states** and **decision points**. All nodes should have mutually unique names. Nameless states, also called **transitory states**, are not permitted. **Non-deterministic** state transition diagrams are permitted.

An initial state can have one or more initialization actions. You can add an action by selecting the initial state, and click on it for editing near the arrow on top of the box. Then an edit cursor appears at the right side of the arrow. You can type in the actions, each line of text will become a separate action. If you stop editing, the actions will become left aligned, the arrow will be resized if necessary

and on top of the actions a horizontal separator line will be drawn. Only when the initial state has actions then the separator line is drawn. Note that these actions are in this notational convention part of the initial state *node*, whereas the other actions (and events) of a diagram are as a notational convention part of an *edge*.

5.1.3 Transitions, Events and Actions

Transitions are drawn as arrows. They do not have an editable name label. Instead they have an event label and a number of action labels. The action labels each occupy exactly one line of text; the event label can contain multiple lines. The event and the actions are separated by a horizontal **separator line**. When a transition is created, a separator line will also be created and it will be positioned near where a name label would be positioned. When the transition line segment near the separator is drawn vertically, the separator is connected to the right side of the line segment. When the segment is drawn horizontally, the separator is positioned a little above the line, see figure 5.2 for the two default positions.

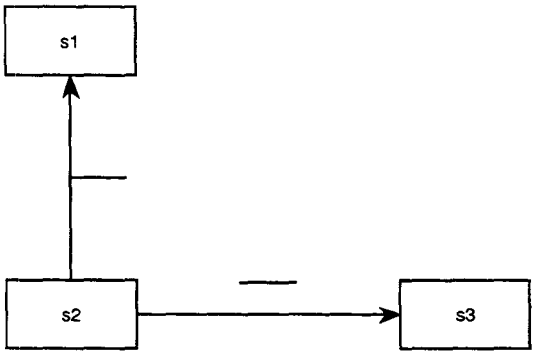


Figure 5.2: Default STD separator positions.

If you click for editing a bit above the separator line, an edit cursor appears and you can enter an event string. When you stop editing, the event string will be positioned above the separator line and the separator line will be resized, if necessary.

If you click for editing a bit below the separator, an edit cursor appears there and you can enter a list of actions. Each line of text will become a separate action. If you stop editing, the actions will become left aligned and, if necessary, the separator line will be resized. When the transition line segment is horizontal and the separator is positioned above the segment, all labels are moved up so that they are all positioned above the line. See figure 5.3 for an example diagram with actions and events.

It is possible to drag the separator line together with the event and action labels. There are two different sorts of movements possible [1]:

1. **Move it to another line segment.** You do this by selecting the transition and click for editing on the desired transition segment. The separator line and the existing event and action labels are moved to that line segment. When you stop editing, the separator will remain in that line segment. Summarizing: to move it to a different segment, empty the selection and then double click the desired segment.

[1] Experiment with this, please.

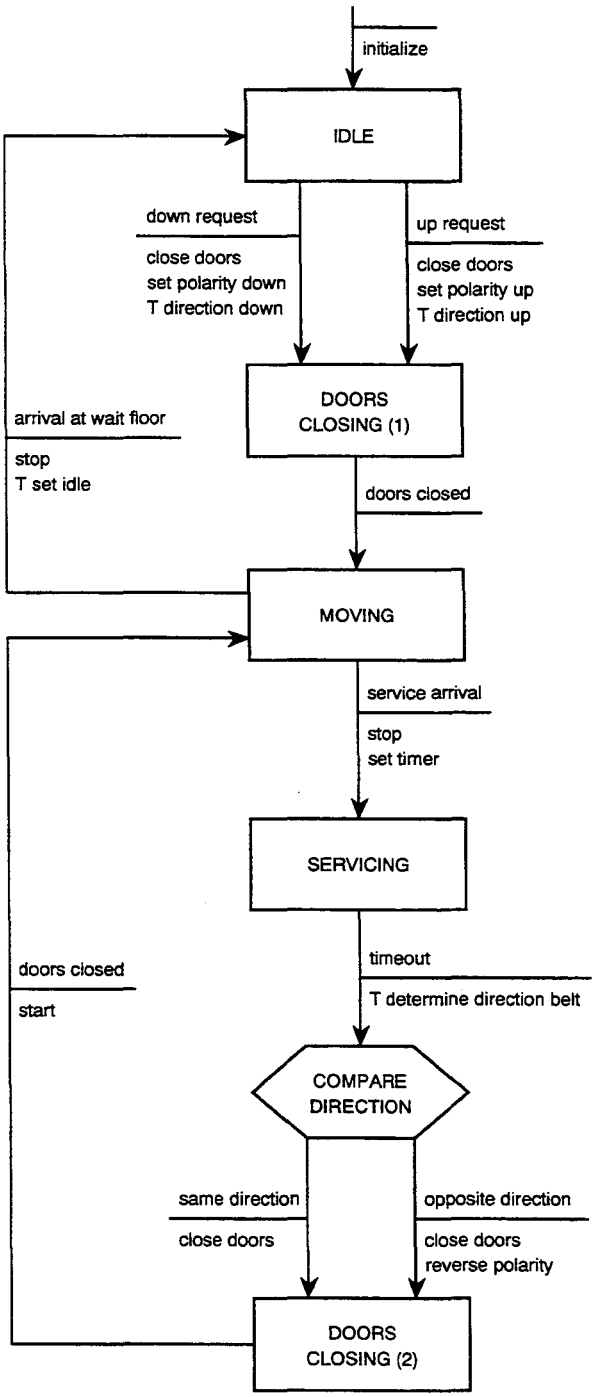


Figure 5.3: STD with events and actions.

2. **Move within a line segment.** Drag with button-1 the separator or one of the labels to the desired position. The final position is determined by the direction of the transition line segment.

   - If the line segment is more or less horizontal, you can freely move the separator and labels from left to right. But you can only move the separator up and down to two predefined positions: either all labels are positioned entirely above the line segment or all labels are positioned entirely below the line segment.
   - If the line segment is more or less vertical, you can freely move the separator and labels up and down the segment, but you can only move the separator left and right to two predefined positions: either all labels are entirely positioned at the left side of the line segment or all labels are entirely positioned at the right of the line segment.

The automatic positioning of labels looks best when the segments are horizontal or vertical. When the transition arrow segments are diagonal, then label positioning still works but sometimes the result does not look as great as figure 5.3.

5.1.4 Constraint Checking

TSTD checks the following constraints that are summarized in figure 5.4.

Constraint	Mode	Description
ST1	Immediately	You cannot connect different initial states.
ST2	Soft	Each STD contains exactly one initial state.
ST3	Soft	Each STD contains at least one action.
ST4	Immediately/Soft	Each (initial) state and each decision point has a mutually unique non-empty name.
ST5	Immediately/Soft	Each node is reachable from one single initial state by a finite sequence of transitions.
ST6	Soft	From a decision point should depart two or more transitions.
ST7	Immediately	A transition should not lead from and to the same decision point.
ST8	Soft	Each transition has a non-empty event label.
ST9	Soft	Transitions from and to the same state should have an action.
ST10	Immediately	All actions inside the same transition or initial state are mutually uniquely named.
ST11	Immediately	All names of (initial)states and decision points and all events and actions contain at least one letter.
ST12	Immediately/Soft	Transitions from and to the same node have mutually unique event strings.

Figure 5.4: Immediately checked and soft constraints on STDs.

5.2 The Process Structure Diagram Editor (TPSD)

5.2.1 Nodes and Edges

TPSD has just one special type of node, called Process and one type of edge called Empty edge (see figure 5.5). The process node is represented by a box. In the top right corner of the process node box, a process operator can be specified. You edit this operator by selecting the box and then click in the top right corner. You go into edit mode then and the operator is typed in as a single character. See figure 5.6 for the different operators.

Figure 5.5: Process structure diagram nodes and edges.

Box	Operator	Description
L		sequence, the default.
L *	*	iteration.
L o	o	choice.
L !	!	for premature termination.
L ?	?	for premature termination.

Figure 5.6: Process structure operators.

5.2.2 The Process Tree

TPSD is a bit different from the other existing TCM diagram editors because in TPSD the layout (representation) is significant for the meaning of the diagram. So TPSD has to make sure that there is always a one-one relationship between each node instance and its representing shape instance. Therefore, duplicate shapes cannot be made in TPSD. During editing, TPSD enforces that the graph is a set of undirected trees. It has the constraint that an edge can only be added to the graph when the resulting graph would not contain a cycle. This constraint is also implemented for the tree editors (chapter 8).

In TPSD, the highest box in the drawing area represents the root of the tree, which is called the **main root**. This is the process node whose representing box has the smallest y-coordinate. The **main tree** is the tree that is connected to the main root.

Each process node has a set of children that is ordered from left to right. The ordering of the children is determined by the position of the shapes in the drawing area [2].

The **Update Sequence Labels** commands in a submenu of the TPSD View menu make it possible to show the process sequence numbers in the lower right corners of the boxes. The **show no sequence labels** command makes all the sequence numbers invisible. The **update action sequences** command draws the sequence numbers of all actions. **Actions** are the leaf nodes of the tree, excluding the "quit" boxes. See figure 5.7 for a PSD with numbered actions. The **update process sequences** command draws sequence numbers in all process nodes. Each parent receives a sequence number that is one higher than the highest number amongst its children. Note that the sequence numbers are not updated automatically when you edit the diagram. The labels are only recalculated and updated when you perform an update sequence labels command again. However, when you save the PSD to file, updating sequence labels is called implicitly.

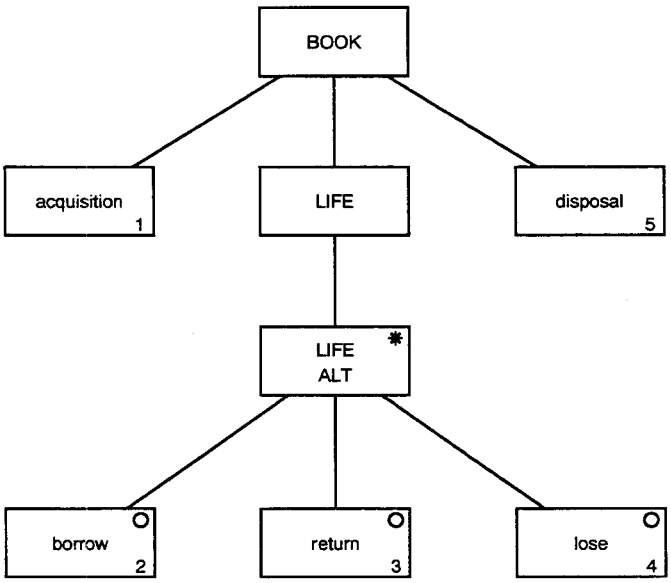


Figure 5.7: Example PSD with numbered actions.

5.2.3 Constraint Checking

The Check Diagram command in the Diagram menu checks that the tree is a syntactically correct JSD process structure diagram. Because of the immediately enforced constraints, Check Diagram assumes that there are no duplicate nodes and no cycles. In figure 5.8 all immediately and soft constraints are summarized.

Note that, if the box of a child has a smaller y-coordinate (is higher) than its parent, then a warning is given because that is an indication of a poor layout.

[2] More precisely: the order of the children is determined by the x-coordinates of the end points on the parent side of the edges between the parent and the children. This sounds a bit complex, but this includes the common left to right ordering with straight edges.

Constraint	Mode	Description
PS1	Immediately	PSDs have a tree structure (no cycles).
PS2	Soft	There is a unique main root. This means that there should not be two or more boxes having the same smallest y–coordinate.
PS3	Soft	There is a single tree. There should not be a process node in the diagram that is not part of what is considered as the main tree.
PS4	Soft	The main root has the sequence operator (i.e. empty operator).
PS5	Soft	Each process has a non–empty name.
PS6	Soft	Two processes can only have the same name when they are both actions.
PS7	Soft	A "quit" box does not have children.
PS8	Soft	A "posit"–"admit" combination is directly connected to the main root.
PS9	Soft	A "quit" box is an indirect child of a "posit" box.
The list of children of a process node should conform to one of the following rules:		
PS10	Soft	The list is empty. This means that the parent is an action.
PS11	Soft	The list is a sequence, i.e. all operators of the children are empty.
PS12	Soft	The list is a choice. There are two or more children and all their operators are "o"s.
PS13	Soft	The list is a single iteration. There is one child and its operator is a "*".
PS14	Soft	The list is a single ''quit'' box. The operator is a "!" and the name is "quit" (case ignored).
PS15	Soft	The list is a ''posit''–''admit'' combination. Both operators are "?", the first name is "posit" and the other name is "admit" (case ignored).

Figure 5.8: Immediately checked and soft constraints on PSDs.

5.3 The Recursive Process Graph Editor (TRPG)

5.3.1 Nodes and Edges

See figure 5.9 for the TRPG shapes and subjects. There are two node types and three node shape types. Process graph roots have their name label written on top of a downwards pointing arrow. In general the process graph roots are named after the process graph document, but in upper case letters. By default a process graph is named UNTITLED. Process graph nodes have two node shape representations. They can be small unnamed circles or bigger rounded boxes which can contain a name label. Process graph nodes are connected by event edges. See figure 5.10 for an example recursive process graph.

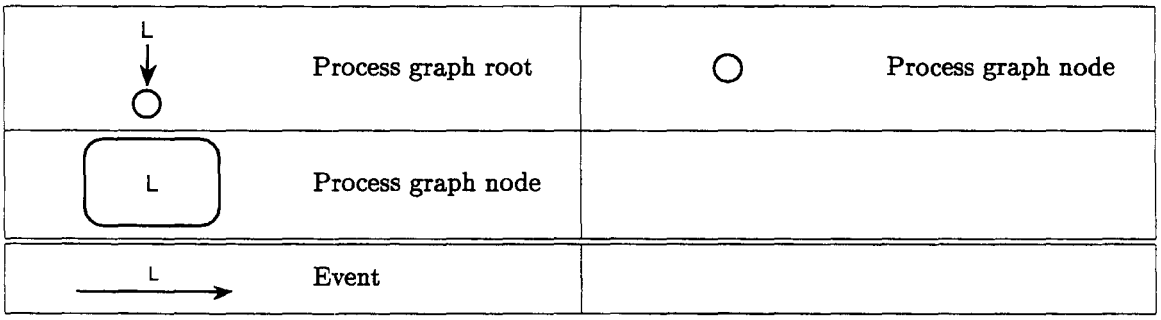


Figure 5.9: Recursive process graph nodes and edges.

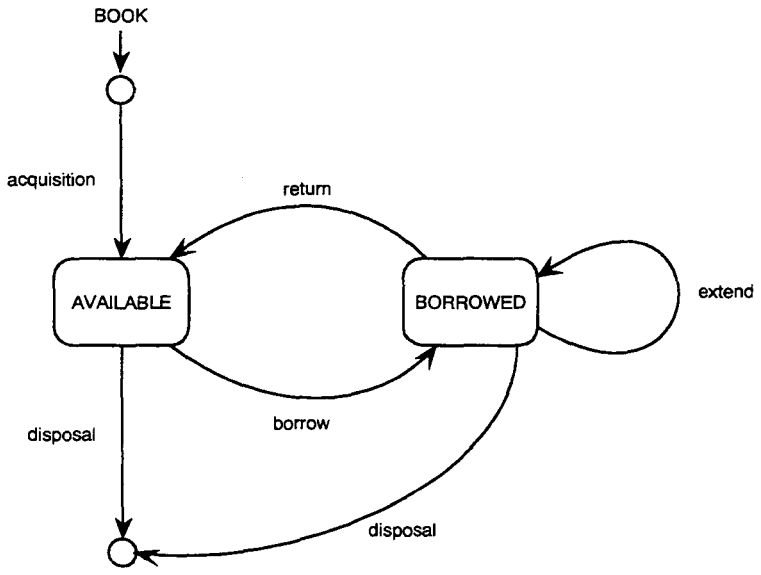


Figure 5.10: Example recursive process graph.

5.3.2 Constraint Checking

TRPG checks the immediately enforced and soft constraints that are summarized in figure 5.11.

Constraint	Check	Description
RP1	Immediately	If a pair of events connects the same pair of nodes and has the same direction and then these labels should have different names.
RP2	Immediately	You can not connect two different process graph roots.
RP3	Immediately	Each process graph node is accessible from at most one process graph root.
RP4	Soft	Each process graph node is accessible from exactly one process graph root.
RP5	Immediately	All process graph roots have different non–empty names.
RP6	Soft	Each event has a non–empty label.
RP7	Immediately	The name of an event or process graph node or root contains at least one letter.

Figure 5.11: Immediately checked and soft constraints on RPGs.

Chapter 6

Function View Editors

6.1 The Data Flow Diagram Editor (TDFD)

6.1.1 Main window

Both TDFD as TDEFD have an extra menu in the main window called **DFD** which contains some commands that are specific for data flow diagrams.

Furthermore there is an editable text field labeled `Diagram` which shows the index of the data flow diagram. See section 6.1.3 for the function of that field.

6.1.2 Nodes and Edges

See figure 6.1 for the subjects and the representing shapes that are used in TDFD. In figure 6.2 you find the possible connections of node types by edge types. These are immediately enforced constraints.

TDFD uses the standard data flow diagramming conventions from [5, 22] with the exception that diagrams are always drawn as a graph in TCM and therefore, they can never have dangling edges. TDFD uses a different notation from the one being used in [18]: data flow diagrams should be drawn as a graph, so a flow in a decomposition going to or coming from "nowhere" is not permitted (unlike for instance [18], figures 9.16 and 9.17). See figure 6.3 for an example of the TCM notation for DFDs. For more information see appendix section C.3.1.

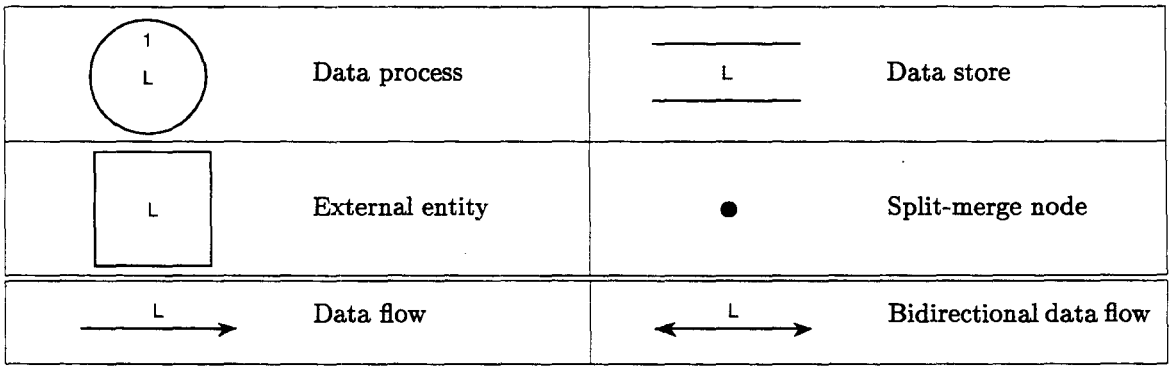


Figure 6.1: Data flow diagram nodes and edges.

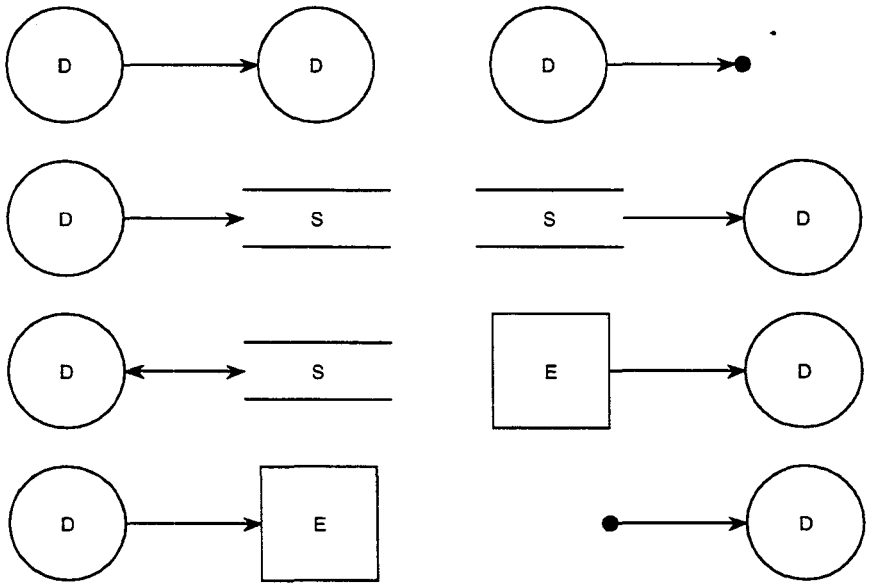


Figure 6.2: Permitted data flow diagram connections.

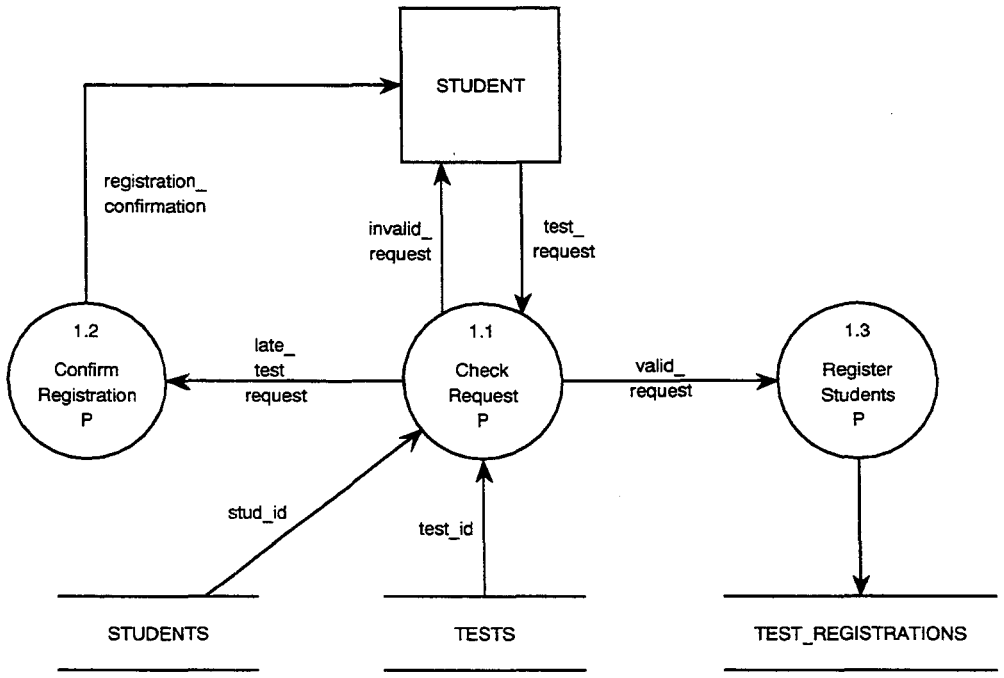


Figure 6.3: Data flow diagram in graph notation.

6.1.3 Data Flow Diagram Levels and Indexes

Data processes have unique indexes. Data process shapes show their index labels when the **Indexes** check button in the list of tiled node buttons is on. When you turn the toggle off, then the indexes become invisible and they cannot be edited. Index labels of data process shapes are positioned near the top of the circle. When they are visible, they can be edited separately from the name label by selecting it for editing by clicking in the upper part of the circle. TDFD immediately enforces that index labels are unique and that they conform to the BNF syntax in figure 6.4.

When you create a new data process, TDFD assigns it automatically a unique index number. By default TDFD assigns the processes the numbers one to the current number of processes. When you issue the command **Renumber indexes** from the DFD menu in the main window then all process indexes are renumbered to sequential indexes from one to the number of processes. When you create a new process, the lowest unused index number is chosen.

When you draw a **leveled or hierarchical data flow diagrams**, data processes can be **decomposed** into subdiagrams. If the diagram you are drawing is a decomposition of a data process with label n, then you can enter the label n in the text field labeled `Diagram`. When you fill in the text field, the index of the diagram is updated when you press <Return>. New processes that are created in the diagram receive then as index, the diagram index plus a unique number. For instance, when you have set the diagram index to 2 (i.e. the diagram is supposed to contain the decomposition of some data process with index 2), then all new processes will receive the indexes 2.1, 2.2, etc. So, when you edit a context diagram or a level 1 diagram then this field is empty. When you edit the diagram of a process decomposition then it contains the index of that process.

Index	→	C |**0**
C	→	$N\ Z^*$ | $N\ Z^*$. C
Z	→	**0** | N
N	→	**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**

Figure 6.4: Data process index syntax.

Warning: in the version of TCM that is described in this manual you have to create for each decomposition a separate document. In this version of TDFD it is not possible to perform a decomposition within the editor. Not even all invalid combinations of index and level numbers within the same decomposition are checked yet [1].

6.1.4 Minispecs

Data processes that are not decomposed are called **primitive data processes** and should be specified by a so-called **minispec**. In this version of TCM it is possible to give a data process a minispec in the form of an arbitrary piece of text. When you (first) select a data process and choose the Minispec command from the DFD menu, then a text editor dialog (see chapter 2.5) is popped up in which you can edit a minispec in whatever notation you prefer. Minispecs are stored together with the document and they can be individually loaded and saved to file and be printed [2].

[1] Both decomposition of data processes as more extensive checks on DFDs (like correct use of indexes and **balancing** of data flows) are on our wish list.

[2] Printing or saving multiple minispecs as one report is on our current wish list.

6.1.5 Splitting and Merging Flows

Data flows can be split or merged. To draw a split or merged data flow in a graph, a split-merge node is introduced. This is represented by a small black uneditable dot. You can draw data flow edges from a data process to this dot and data flow edges from the dot to a data process. See figures 6.5(a) and 6.5(b). If the outgoing flows of a splitting node or the incoming flows of a merging node are unnamed then this means that identical copies are split respectively merged.

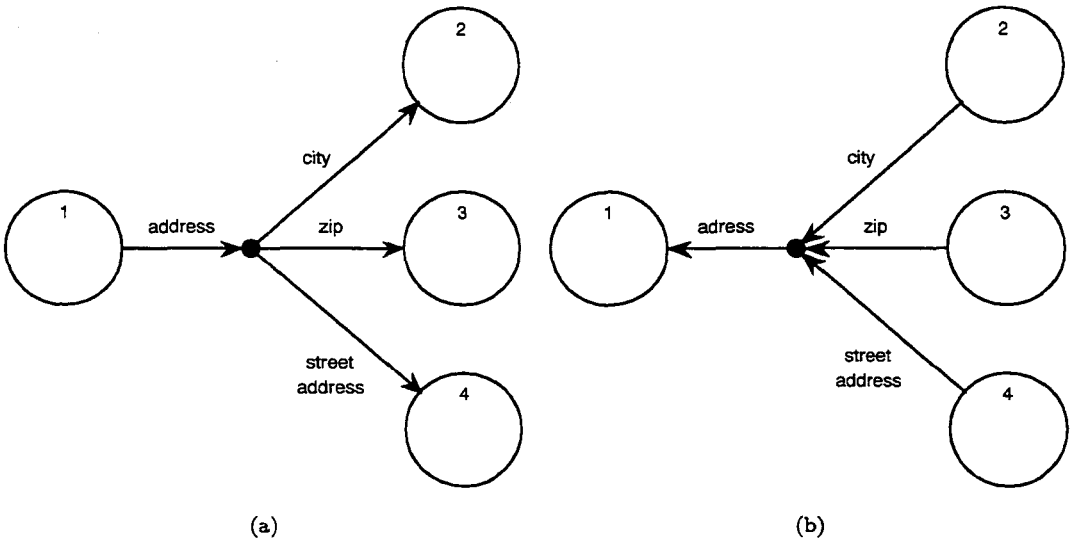


Figure 6.5: Example splitting and merging data flows.

6.1.6 Constraint Checking

In addition to the constraints that were mentioned in the previous sections of this chapter, TDFD also checks some other constraints that are summarized in figure 6.6. Some of these constraints can not be enforced immediately during *all* editor commands. If that is the case, they are additionally checked by Check Diagram as a soft constraint.

6.2 The Data and Event Flow Diagram Editor (TDEFD)

TDEFD is a proper superset of TDFD. The features that are specific for TDEFD are explained in this section. In short, TDEFD is TDFD extended with control processes and event flows. All nodes and edges of TDFD are available in TDEFD and all constraints in TDFD are applicable to TDEFD. It is also possible to read a `.dfd` diagram into TDEFD (although a warning is given). The other way around, reading a `.defd` diagram in TDFD, is only possible when it does not contain event flows or control processes.

Constraint	Mode	Description
DF1	Soft	Each data process has >=1 input flows and >=1 output flows.
DF2	Soft	Each store and external entity has >=1 connections.
DF3	Immediately	Each flow connects two different nodes.
DF4	Soft	All flows that do not connect a store, have a name.
DF5	Immediately	Two flows from and to the same node, have different names.
DF6	Immediately/Soft	Data processes, data stores and external entities should have mutually unique non–empty names.
DF7	Immediately/Soft	Process indexes should be non–empty and unique.
DF8	Immediately	Split–merge nodes should connected to all different data processes.
DF9	Soft	Each split–merge node has at least one incoming and at least one outgoing data flow.
DF10	Soft	Each split–merge node either splits or merges, i.e. either the number of incoming or the number of outgoing data flows is greater than one, but not both can be greater than one.
DF11	Soft	Data flows that are input to a splitting split–merge node should have a non–empty name
DF12	Soft	Data flows that are output from a merging split–merge node should have a non–empty name

Figure 6.6: Immediately checked and soft constraints on DFDs.

6.2.1 Nodes and Edges

TDEFD has data processes and **control processes**, represented by a solid and a dashed circle, respectively. Both types of nodes have possibly an index label, see section 6.1.3. TDEFD has two types of flows: **time discrete flows** and **time continuous flows**. Time discrete flow are the same flows as in TDFD, but time continuous flows are new and they are represented by a double headed arrow (two heads on the same side).

TDEFD has **event flows** that are represented by dashed arrows. When an event flow has as label 'T', it is a **trigger** and when it has as label 'E' or 'E/D', it is a **prompt**. Event flows have a time discrete variant and a time continuous variant too, represented by a dashed single headed arrow respectively by a dashed double headed arrow.

See figure 6.8 for the permitted connections in the data and event flow diagram editor.

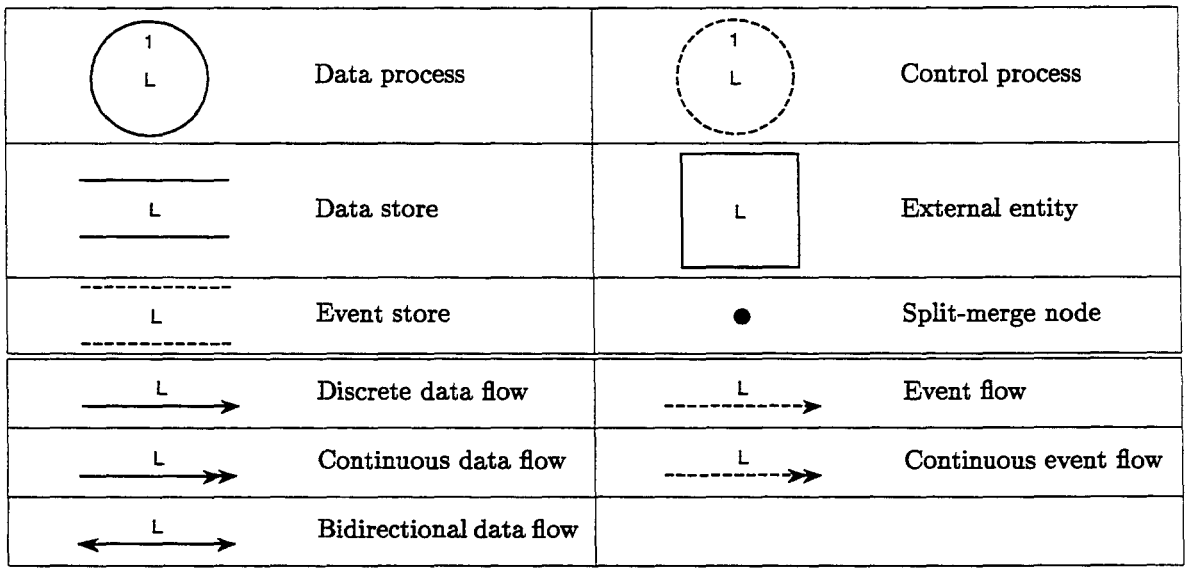


Figure 6.7: Data and event flow diagram nodes and edges.

6.2.2 Constraint Checking

TDEFD checks all the constraints of TDFD. For these constraints see figure 6.6. The other constraints that TDEFD checks are listed in figure 6.9.

6.3 The System Network Diagram Editor (TSND)

6.3.1 Nodes and Edges

In order to be able to edit a system network diagram as a graph, a system network connection in TSND is a compound connection consisting of three parts: one node and two edges. The node is one of State vector, Data stream or Controlled data stream. The two edges are a Connection start edge and a Connection end edge. These two types of edges do not have a name label but they can both have a cardinality constraint label. The cardinality constraints should conform to the same syntax as in section 4.1.2. Compound connections connect system network processes (abbreviated

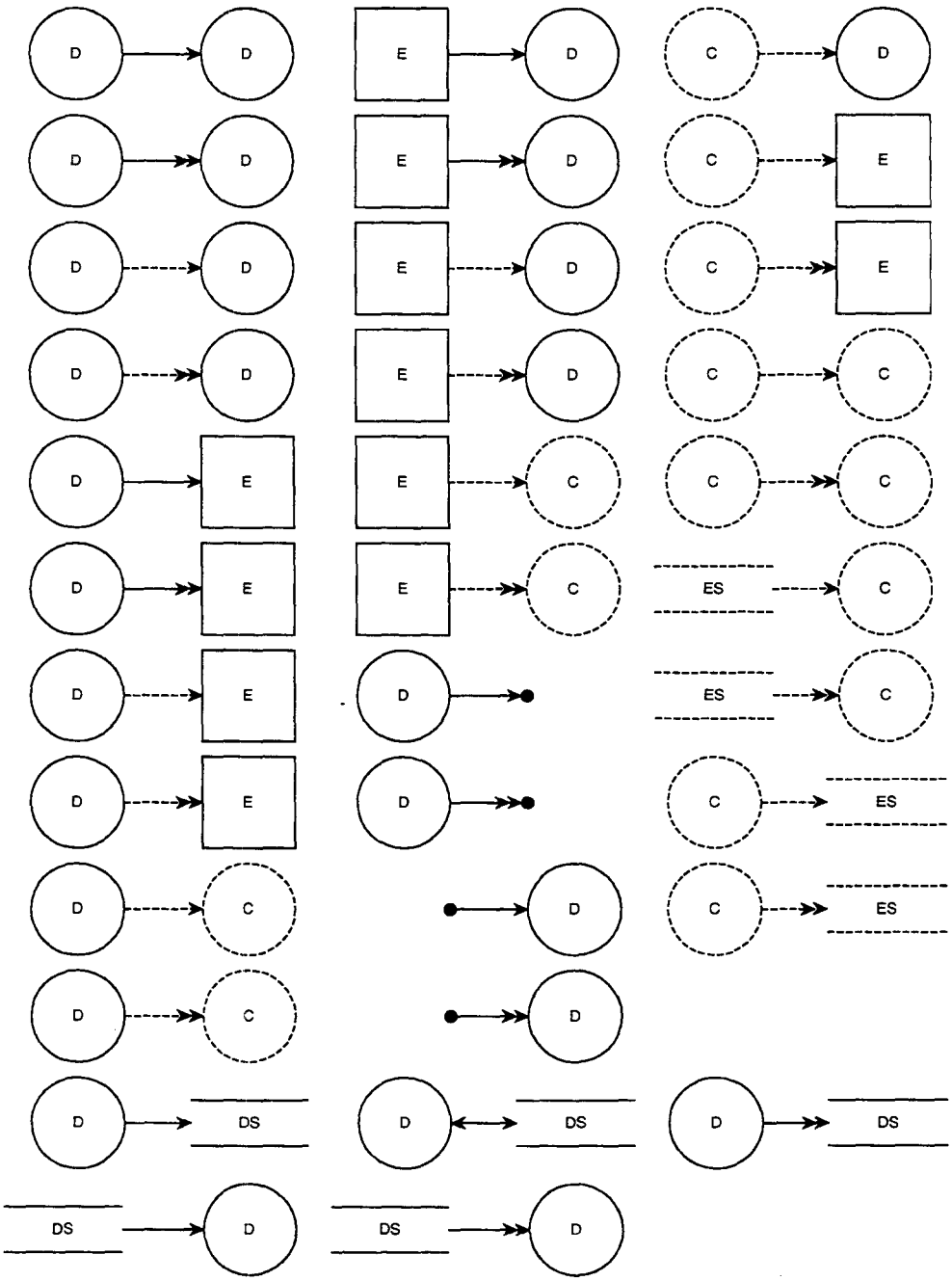


Figure 6.8: Permitted data and event flow diagram connections.

Constraint	Mode	Description
DEF1	Immediately/Soft	Data and control processes, data and event stores and external entities have mutually unique non–empty names.
DEF2	Soft	Each control process has at least one output flow.
DEF3	Immediately	Flows to and from an event store are unnamed.
DEF4	Soft	Each event flow is connected.
DEF5	Immediately	All flows that connect a split/merge node are either all discrete or all continuous.
DEF6	Immediately	(Continuous) data flows should not be labeled as a trigger or prompt.
DEF7	Immediately/Soft	Each event flow from a control process to a data process should be either labeled as a prompt or as a trigger.
DEF8	Immediately	Event flows between two controll processes should not be labeled as a prompt.
DEF8	Immediately	An event flow that goes to an external entity or that does not come from a control process should not be labeled as a prompt or a trigger.
DEF9	Immediately	Continuous event flows should not be labeled as a trigger or a prompt.

Figure 6.9: Immediately checked and soft constraints on DEFDs (not part of TDFD).

to SN processes). See figure 6.10 and 6.11 for the representations and the permitted connections (immediately enforced). For an example system network diagram, see figure 6.12.

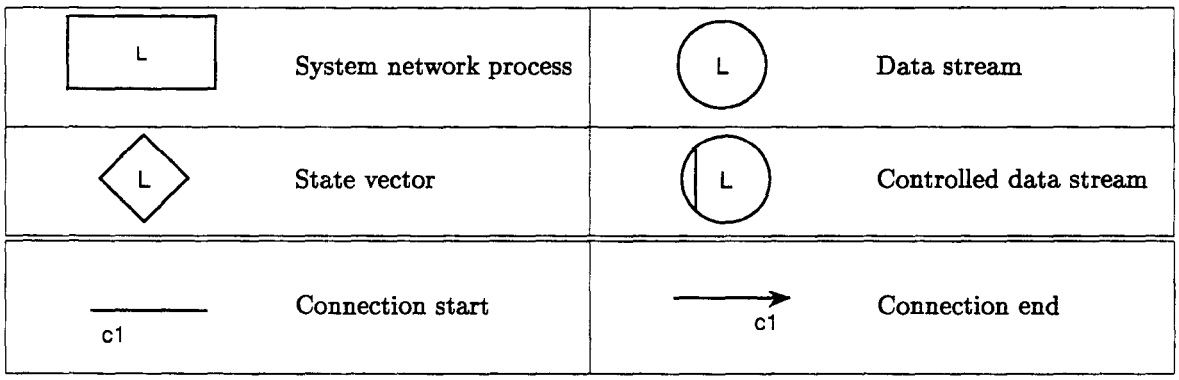


Figure 6.10: System network diagram nodes and edges.

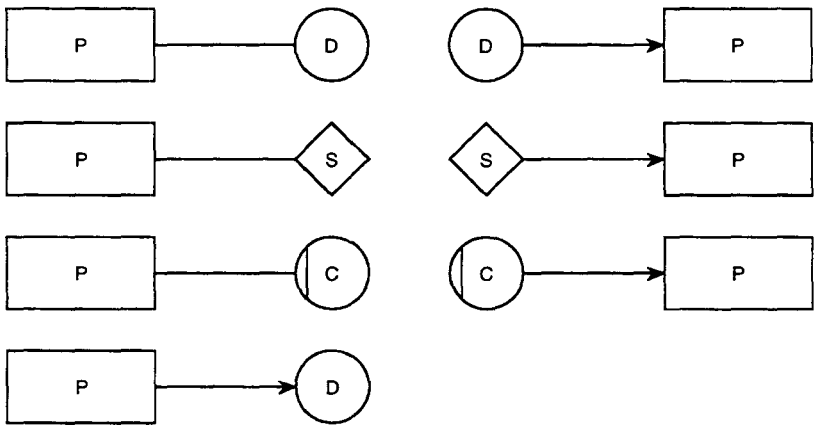


Figure 6.11: Permitted system network diagram connections.

6.3.2 Constraint Checking

In addition to the constraints mentioned in the previous section about TSND, TSND checks the immediately enforced and/or soft constraints that are summarized in figure 6.13.

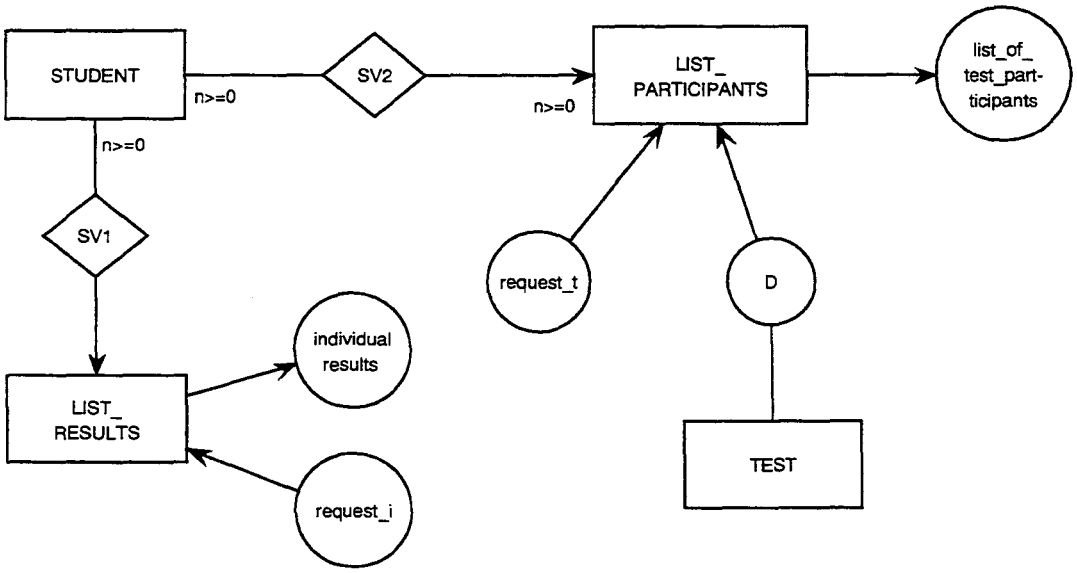


Figure 6.12: Example system network diagram.

Constraint	Mode	Description
SN1	Immediately/Soft	All nodes should have mutually unique non–empty names.
SN2	Immediately	All data stream, state vector and controlled data stream nodes are connected by at most one connection start edge
SN3	Soft	All system network process nodes should be connected.
SN4	Soft	All data stream nodes should be connected.
SN5	Soft	All state vector and controlled data stream nodes should be connected by exactly one connection start edge and at least one connection end edge.

Figure 6.13: Immediately checked and soft constraints on SNDs.

Chapter 7

Table Editing

All TCM table editors are very similar. With each table editor you can create and manipulate textual tables, i.e. tables in which the cells are filled with a multi-line text string. The table editors offer a lot of layout facilities and TCM has special purpose table editors that have constraints built-in for a specific modeling technique. Furthermore, the tables that are made by TCM are kept graphically consistent and TCM keeps track of the contents of the tables, which is important when you have to do a lot of updates.

7.1 Editing Tables

When you start up a table editor, or issue the New command in a table editor, a new table is created having N x M empty cells (by default, N = 7 and M = 7). New rows and columns of empty cells can be created with the Add Row and Add Column commands. Selected rows and columns can be deleted with the Delete Rows and Delete Columns commands.

If you start up a table editor from the `tcm` start-up tool, before the main window of the editor is displayed, a text field list dialog is presented. This dialog has four fields: number of rows, number of columns, minimal row height and minimal column width. In these fields the default values are already filled in. You can fill in different values if you want.

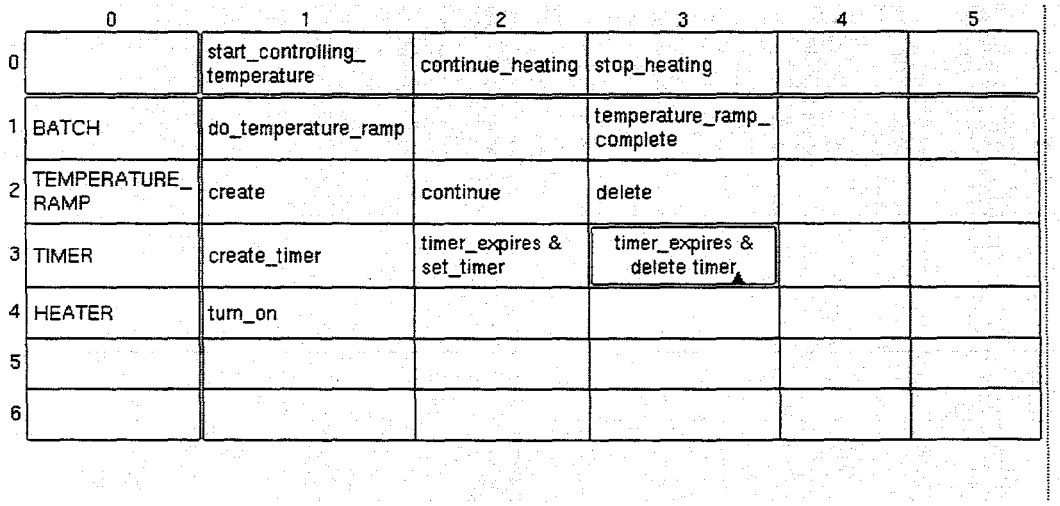
By default, the table is positioned on the drawing area having its top-left corner a little right below the top-left corner of the drawing area. You can reposition the entire table by means of the four arrow buttons in the bottom-left corner of the main window. Like the diagram editors, the page boundaries are displayed. If you print the table or save it as plain PostScript, the table is positioned on the printed page exactly as it is positioned on the drawing area (what you see is what you print).

Each row and each column has a sequence number label. These labels are used to select an entire row or column or to move an entire row or column to a new position. These labels cannot be edited or printed. In the View menu there is an option to hide the labels. See figure 7.1 for a snap-shot of the table editor.

7.1.1 Definitions

We first give some definitions of the terms that are used in the rest of this chapter.

The document that you edit is called a **table**. A table contains a number of **cells** which are *invisible* rectangles. The table has a number of **rows** and a number of **columns**. Each cell is part of one row and of one column. All rows in the table have the same number of cells and all columns of a table have the same number of cells. All cells in a row have the same height and all cells in a



	0	1	2	3	4	5
0		start_controlling_ temperature	continue_heating	stop_heating		
1	BATCH	do_temperature_ramp		temperature_ramp_ complete		
2	TEMPERATURE_ RAMP	create	continue	delete		
3	TIMER	create_timer	timer_expires & set_timer	timer_expires & delete timer		
4	HEATER	turn_on				
5						
6						

Figure 7.1: Snap-shot of a table being edited.

column have the same width. All cells in a row have the same center y-coordinate and all cells in a column have the same center x-coordinate. All rows and columns are packed, i.e. rows and columns cannot overlap and the distance between two rows or two columns is zero when there are no other rows between them.

Each row has a **row label** which is an uneditable label containing the row sequence number (rows are ordered according to their y-coordinates), positioned near the left edge of the first cell of the row. Each column has a **column label** which is an uneditable label containing the column sequence number (columns are ordered according to their x-coordinates), positioned above the first cell of the column.

Each border line between two cells is called a **line piece**. A line piece has a certain line style: solid, dashed, fat, invisible and so on. So, line pieces are the items of a table that can be made visible and make the table appear like a grid. Line pieces are either horizontal or vertical. Horizontal line pieces are part of the same column as the neighboring cells and vertical line pieces are part of the same row as the neighboring cells.

A cell itself is invisible, only its four border lines are possibly visible. A cell can contain some piece of **cell text**. Cell text is an editable multi-line text string. The cell text is positioned in the cell according to the **column alignment**, **row alignment**, **text margin width** and **text margin height** (which are all explained in the next sections). A cell can be selected. A selected cell has a rectangle drawn inside its four border lines.

7.1.2 Selection Commands

A selected cell is highlighted by an extra black rectangle inside the cell boundaries (see figure 7.1). The distance between the selection rectangle and the line pieces is a pixel or two. Here is a list of all the table editor selection commands:

- **Select a single cell.** Click button-1 on an unselected cell. This cell becomes the only selected cell of the table.
- **Select an area of cells.** Drag with button-2 pressed down. Note that by this command cells are *added* to the selection, never removed.

- **Select a row of cells.** Click button-1 on the row label. The cells in the row become the only selected cells in the table.
- **Select a column of cells.** Click button-1 on the column label. The cells in the column become the only selected cells in the table.
- **Select all cells.** Choose **Select All** from the Edit menu.
- **Add a cell to the selection.** Click button-2 on an unselected cell.
- **Add a row to the selection.** Click button-2 on a row label. The row should have one or more unselected cells.
- **Add a column to the selection.** Click button-2 on a column label. The column should have one or more unselected cells.
- **Remove a cell from the selection.** Click button-2 on a selected cell.
- **Remove a row from the selection.** Click button-2 on a row label. All cells of the row should be selected. All cells of the row become unselected.
- **Remove a column from the selection.** Click button-2 on a column label. All cells of the column should be selected. All cells of the column become unselected.
- **De-select all cells.** Click button-1 or button-2 somewhere outside the cells, in the drawing area.

7.1.3 Editing Text

Text editing works the same for all document editors. See section 2.5.1 for the different edit commands and the two different edit modes, **in-line editing** and **out-line editing**.

When a cell text has been edited and the autoresizing toggle is on, the cell sizes automatically adapt to the new text size. I.e. if the text is too high to fit into the cell, the cell, and consequently the entire row is made higher, and if the text is too wide to fit into the cell, the cell, and consequently the entire column, is made wider. If the cell text becomes less wide or high, and the autoresize toggle is on, then the row will be made less high, respectively, the column will be made less wide, down to the size that the other texts in that row or column still fit and that the row height or column width do not get below the minimal row height or column width of the table. The **minimal row height** and **minimal column width** is the cell height and cell width when the table is initialized. Cells can not be made smaller than the minimal row height and column width. The minimal row height and column width can be modified in Table menu.

7.1.4 Copying and Moving Text

To **move a single cell text** from one cell to another, drag and drop it with button-1 from an *unselected* source cell to a destination cell (just like dragging an edge label in a diagram editor). The old text of the destination cell will be overwritten. If the text is dropped somewhere outside a cell or you click button-2 while dragging, the command will be aborted.

Copying a single cell text from one cell to another works in the same manner as moving a text. The difference is that the source cell should be *selected*.

When the autoresizing toggle is on, cell sizes automatically adapt to the new situation when text is copied or moved.

7.1.5 Cutting and Pasting Text

The above method for moving or copying cell texts works only for one cell at the time. To perform this on a whole group of cell texts there are the Cut, Copy and Paste commands. To cut cell texts to the cell text **buffer**, use the **Cut Texts** <Ctrl+X> command in the Edit menu. It clears the texts of the selected cells and copies them into the buffer. Cut can also be a helpful command when you want to clear a part of the table. You can copy cell texts to the buffer via the **Copy Texts** <Ctrl+C> command in the Edit menu. It copies the text into the buffer, but, unlike Cut, it does not clear the cells.

The cell texts in the buffer can be pasted into the table. The **Paste Texts** <Ctrl+Y> command in the Edit menu makes a **paste box**, that is attached to the mouse pointer. The size of the paste box is about the size of the cell texts in the buffer. You can move the paste box with the mouse and click button-1 somewhere into the table to release it. The cell in which the mouse cursor (and the top-left of the paste box) was at the time you clicked button-1 becomes the top-left cell of the cell area in which the texts are pasted. Which cell is pasted by which cell text is determined by the relative row and column position (not of the actual size of the texts) when the original texts were cut or copied into the paste buffer. When you paste, the texts from the buffer are *copied* from the buffer into the table. The old cell texts are overwritten. If the box is released with the mouse pointer somewhere out of the table, the paste command is aborted. If the paste box is released (partly) outside the table, only the part that covers the table is modified.

7.1.6 Adding Rows and Columns

To add rows, use the **Add Rows** command in the Edit menu. A prompt dialog is popped up asking for the number of rows to be added (default is 1). In the dialog there is a toggle to choose between adding the new rows above the selection or appending them to the bottom of the table. "Above the selection" means one row above the highest selected cell in the current selection, or when the selection is empty, to the bottom of the table.

Adding columns, via the **Add Columns** command, is like adding rows. There is a choice between adding the new columns to the left of the current selection or appending to the right of the table.

After adding rows or columns the table is redrawn including the new rows and columns in such a way that the top-left corner of the table remains at the same position.

7.1.7 Deleting Rows and Columns

To delete rows, use the **Delete Rows** command in the Edit menu. This command deletes every row in which one or more cells are selected.

To delete columns, use the **Delete Columns** command in the Edit menu. This command deletes every column in which one or more cells are selected.

To delete all cells, use the **Delete All** command. This results into an empty table [1]. Before everything is deleted, a question dialog asks if you are sure about what you are doing.

To remove all unused rows and columns, use the **Purge** command. This command deletes all rows and columns in which all cells have empty cell texts.

When you delete rows and/or columns, the table is redrawn in such a way that the top-left corner stays at the same position.

7.1.8 Moving Rows and Columns

Moving a row is possible via dragging a row label from the source row to the desired destination

[1] Contrary the common belief an empty table is still a table.

row. If the dragged label is released in one of the cells of the destination row, the source row, where the label came from, is moved to the position of the destination row and the destination row and the rows between the source and destination row are all shifted one row up (when the source row was above the destination row) or one row down (when the source row was beneath the destination row) [2].

Moving a column works in a similar way. If you drop a column label into a cell of another column, the source column moves to that position and the destination column and the columns between those two are all shifted one column left or right.

Note that when you move a row or column, the row and column labels are not moved with. They stay in the same consecutive order, of course.

After either command, the resulting table has the same position and has the same size.

7.1.9 Sorting Rows and Columns

With the **Sort Rows** command in the Edit menu you can sort rows alphabetically. When the selection is empty, the table is sorted according to the contents of the first column. If there are selected cells, sorting is according to the column of the left-most selected cell.

With the **Sort Columns** command in the Edit menu you can sort columns alphabetically. When the selection is empty, the table is sorted according to the contents of the first row. If there are selected cells, sorting is according to the row of the top-most selected cell.

After either command, the resulting table has the same position and the same size.

7.1.10 Resizing Rows and Columns

Resizing rows and columns can be done by hand but only when the autoresizing toggle is off. To resize a row or column, drag a line piece between two rows or columns: if you enter a line piece between two rows or between two columns, the mouse pointer turns into a pair of vertical respectively horizontal arrows. To **resize a row** you can drag the line piece with button-1 up or down. If you drop the line at a new position, the row *above* the dragged line, is resized. You cannot resize a row to less than the minimal row height of the table (the minimal row height can be modified in the Table menu). The part of the table below the row that is resized, will be repositioned (but not resized).

To **resize a column** you can drag a line piece with button-1 left or right. If you drop the line at a new position, the column to the *left* of the line that is dragged, is resized. You cannot resize a column to less than the minimal column width of the table (which you can be modified in the Table menu). The part of the table to the right of the column that is resized, will be repositioned (but not resized).

7.1.11 Undo and Redo

The table editors have a multiple-level undo for their commands. A command which is undone, can be redone again. All table edit commands listed in the Edit and Text menu can be undone. Furthermore, the table edit commands issued by the mouse can also be undone. In table 7.2 all undo-able table editor commands are listed together with how they can be called.

7.1.12 Table Text Commands

The Text menu contains certain commands that work on cell texts.

- **Find.** With the Find menu entry in the Text menu you call a dialog by which you can search for some text in the table. The find dialog is described in section 2.5.3. From this dialog you

[2] This sounds more complicated than it actually is, experiment with this.

Command	User Action	Command	User Action
Add columns	Select desired column position, issue Add Columns from the Edit menu, give the number of columns and click OK.	Purge cells	Issue Purge from the Edit menu.
Add rows	Select desired row position, issue Add rows from the Edit menu, give the number of rows and click OK.	Replace all cell texts	Issue Replace from the Text menu, fill in the texts to find and to replace with and click the Replace All button.
Append table	Issue Append from the File menu, choose a file, set desired position and click OK.	Replace next cell text	Issue Replace from the Text menu, fill in the texts to find and to replace with and click the Replace Next button.
Copy cell texts	Select the cells to be copied and issue Copy texts from the Edit menu.	Resize column	Drag with button-1 a line piece at the right side of the column to be resized to its desired position. Works when auto resizing is off.
Cut cell texts	Select the cells to be cut and issue Cut texts from the Edit menu.	Resize row	Issue Select All from the Edit menu.
Delete all cells	Issue Delete All from the Edit menu.	Select cell area	Issue Select All from the Edit menu.
Delete columns	Select the columns to be deleted and issue Delete Columns from the Edit menu.	Select cell area	Select the row according to which has to be sorted and issue Sort columns from the Edit menu.
Delete rows	Select the rows to be deleted and issue Delete Rows from the Edit menu.	Sort columns	Select the row according to which has to be sorted and issue Sort columns from the Edit menu.
Find all cells	Issue Find from the Text menu, fill in the text to find and click the Find All button.	Sort rows	Select the column according to which has to be sorted and issue Sort Rows from the Edit menu.
Find next cell	Issue Find from the Text menu, fill in the text to find and click the Find Next button.	Update cell text	Edit the text in the in-line or out-line text editor and stop in-line editing resp. click the OK button in the out-line editor.
Move cell text	Drag with button-1 on a cell text. When the source cell is selected the cell text is copied, otherwise it is moved.	Update column alignment	Select the columns to be updated and issue one of the items of the Update Column Alignment submenu in the Text menu.
Move column	Drag the label of the column to be moved with button-1 into a cell having the desired column position.	Update font family	Select cells and issue an entry from Update font style submenu in the Text menu.
Move row	Drag the label of the row to be moved with button-1 into a cell having the desired row position.	Update font style	Select cells and issue an entry from Update font family submenu in the Text menu.
Move table	Click the arrow buttons and/or the center button.	Update line style	Select cells and issue entry from Update point size submenu in the Text menu.
Paste cell texts	Issue Paste from the Edit menu. Release the paste box at the desired position.	Update point size	Select cells and issue an entry from one of the submenus of the Update Line Style submenu in the Table menu.
		Update row alignment	Select cells and issue entry from Update point size submenu in the Text menu.

Figure 7.2: All atomic table edit commands.

can **find the next text** or **find all texts** that matches the string to find. In the first case the first cell that is found is selected and the scrollbars of the main window are moved to center the cell in the main window. When you click `Find Next` again, the next cell is selected (top down, from left to right). When you choose `Find All`, all cells that contain a string that matches is selected. The find dialog contains two toggles, one to determine that the matching has to be case sensitive (default off) and the other to determine that a substring of the cell has to match (default on).

- **Replace.** With the **Replace** menu entry in the Text menu you call a dialog by which you can replace texts in the table. The replace dialog is described in section 2.5.3. It has a find next command that works the same as in the find dialog. Furthermore, the replace dialog has a **replace next** and a **replace all** button. Replace next means that in the next cell (the cell that is found with find next) the text strings that match are substituted with the string to replace (that is the second string filled in in the dialog). In the case of replace all this happens to the entire table in one command (global substitution).

  Note that find and replace work on entire cells. But keep in mind that in a cell, the cell text could match the string to find or the string to replace multiple times (at least when you search a substring). When you want to find and replace within a single cell, you should load that text label first in the out-line text editor and then do find and replace within the out-line edit dialog.

- **Default Font.** This entry shows a submenu containing three submenus to set the default font family, default font style and default point size. Each new row or column will receive this default font. Furthermore, all existing rows or columns that contain only empty cells, will receive this default font too. The row and column sequence labels are also drawn in the default font.

- **Update Font.** This entry shows a submenu containing three submenus to update the font family, the font style and the point size of selected cells. Select the rows, columns or individual cells that you want to change and choose the desired update font command. For instance it you have a table in Helvetica format and you want to have it entirely drawn in Times font then you have to select all cells with Select All and then choose Times from the Update Font Family menu. The font families are changed whereas the point sizes and font style remain the same. The update font commands are undo-able commands.

- **Default Row Alignment.** When a new row is created, it will have a certain text alignment (top, center or bottom) which is visible when the row contains one or more cell texts. You can set the default row alignment via this entry which has a submenu of radio buttons. When you change the default row alignment, all new rows receive this alignment as well as all the rows that contain no cell texts.

- **Update Row Alignment.** The **row alignment** determines if a text is positioned near the top, center or bottom of the cell. All cells in a row have the same row alignment. To change the row alignment of one or more rows, choose one of the predefined set of align types from the submenu containing: Top, Center and Bottom. If you select one of these, the alignment of all rows in which one more cells are selected, is changed to this new alignment. Update row alignment is an undo-able command.

- **Default Column Alignment.** When a new column is created, it will have a certain text alignment (left, center or right) which is visible when the column contains one more texts. You can set the default column alignment via this entry which has a submenu of radio buttons. When you change the default column alignment, all new columns receive this alignment as well as all the columns that contain no cell texts.

- **Update Column Alignment.** The **column alignment** determines if a text is positioned near the left, center or right of the cell. All cells in a column have the same column alignment. To change the column alignment of one or more columns, choose one of the predefined set of align types from the submenu containing: Left, Center and Right. If you select one of these, the alignment of all columns in which one more cells are selected, is changed to this new alignment. Update column alignment is an undo-able command.
- **Text Margin Width.** All cells of a table have the same **margin width** which is the minimal distance between the cell texts and the vertical lines of the table. You can update this distance by means of a slider pop-up dialog. If, after the update, some of the texts do not fit into their cells and the autoresizing toggle is on, these cells are resized.
- **Text Margin Height.** All cells have the same **margin height** which is the minimal distance between the cell texts and the horizontal lines of the table. You can update this distance by means of a slider pop-up dialog. If, after the update, some of the texts do not fit into their cells and the autoresizing toggle is on, these cells are resized.

7.1.13 Table Layout Commands

The Table menu contains commands that determine the layout of the table.

- **Default Line Style.** Each newly created line piece will have a certain shape of line. The possible values are normal (by default), dual, dashed, dotted, fat and invisible. This entry shows a submenu of radio buttons containing the possible line style values. When you create new rows or columns, their line pieces have the default line style.
- **Update Line Style.** The style of each line piece in the table can be set independently. The possible values are : normal (default), dual, dashed, dotted, fat or invisible. Each line style menu entry is a submenu with all six possible line styles. The update line style commands are undo-able commands. The possible update line style commands are:
  - **Update Top Lines.** Each line piece that borders on the top of a selected cell is updated.
  - **Update Bottom Lines.** Each line piece that borders on the bottom of a selected cell is updated.
  - **Update Left Lines.** Each line piece that borders on the left of a selected cell is updated.
  - **Update Right Lines.** Each line piece that borders on the right of a selected cell is updated.
  - **Update Surrounding Lines.** Each line piece that borders on exactly one selected cell is updated.
  - **Update All 4 Lines.** Each line piece that borders on a selected cell is updated.
- **Minimal Row Height.** All rows have at least this height. Every new row that is created has this height and if you resize a row by hand (when autoresizing is off), you cannot make the row less high then this height. This entry pops-up a slider dialog for inspecting and updating the minimal row height. If, after the update, some of the rows are less high than the minimum, they will be resized to the minimum. Furthermore, when the autoresize toggle is on, autoresizing is applied to the rows.
- **Minimal Column Width.** All columns have at least this width. Every new column that is created has this width and if you resize a column by hand (when autoresizing is off), you cannot make the column less wide then this width. This entry pops-up a slider dialog for inspecting

and updating the minimal column width. If, after the update, some of the columns are less wide than the minimum, they will be resized to the minimum. Furthermore, when the autoresize toggle is on, autoresizing is applied to the columns.

- **Default Number of Rows.** When a new table is created on start-up or by the New command or when the table was empty and the first column is created, the table will have a default number of rows. You can inspect and update this number via a pop-up slider dialog called from this menu entry.
- **Default Number of Columns.** When a new table is created on start-up or by the New command or when the table was empty and the first row is created, the table will have a default number of columns. You can inspect and update this number via a pop-up slider dialog called from this menu entry.

7.2 The Generic Table Editor (TGT)

This editor has exactly the features described in the previous section. The contents of the cells are unrestricted. Check document will not find any violations. In the remaining sections the specific table editors are described. These work almost the same as the generic editor. The table editors are able to read in each others tables (although a warning message is given when you do this).

7.3 The Transaction Decomposition Table Editor (TTDT)

According to this modeling technique, the entries in row 0 contain transaction names. The entries in column 0 contain object class (or entity type) names. The other entries contain zero or more action names. To graphically separate row 0 and column 0 from the rest of the table, the initial table separates them by default by a dual line. The editor does not check for that layout, however. For an example see figure 7.4. Currently, the editor checks the constraints in figure 7.3.

Constraint	Check	Description
TD1	Immediately	All non–empty entries contain at least one letter.
TD2	Immediately	All entries in the topmost row (row 0) are unique.
TD3	Immediately	All entries in the leftmost column (column 0) are unique.
TD4	Soft	There are no empty entries in row 0.
TD5	Soft	There are no empty entries in column 0.
TD6	Soft warning	Each object class takes part in at least one transaction: a warning is given when all entries (except the first entry) in a row are empty.
TD7	Soft warning	Each transaction uses at least one object class: a warning is given when all entries (except the first entry) in a column are empty.

Figure 7.3: Immediately checked and soft constraints on TDTs.

	start_controlling_ temperature	continue_heating	stop_heating
BATCH	do_temperature_ramp		temperature_ramp_ complete
TEMPERATURE_ RAMP	create	continue	delete
TIMER	create_timer	timer_expires & set_timer	timer_expires & delete timer
HEATER	turn_on		

Figure 7.4: Example transaction decomposition table.

7.4 The Transaction-Use Table Editor (TTUT)

According to this modeling technique, this table has five columns. The entries in row 0 are initialized with the labels `Create`, `Read`, `Update` and `Delete`. The entries in column 0 contain transaction names. The other entries contain zero or more object class (or entity type) names. To graphically separate row 0 and column 0 from the rest of the table, the initial table separates them by a dual line. The editor does not check for that layout, however. For an example see figure 7.5. The editor checks the immediately and soft constraints of figure 7.6.

	Create	Read	Update	Delete
Borrow	LOAN	MEMBER	DOCUMENT	
Overdue		LOAN		
Remind		LOAN		
Renew		DOCUMENT, MEMBER	LOAN	
Return		MEMBER	DOCUMENT	LOAN

Figure 7.5: Example transaction-use table.

7.5 The Function-Entity type Table Editor (TFET)

This kind of the table is also called *Function-Entity Matrix* in [18]. According to this modeling technique, the entries in row 0 contain transaction names. The entries in column 0 contain object class (or entity type) names. The other entries contain **CRUD** strings: a string containing zero or one occurrences of the characters C, R, U and D, and that does not contain any other character. To separate row 0 and column 0 from the rest, the initial table separates them from the rest by a dual line. The editor does not check for that layout, however.

In [18] it is shown how you can define **business areas** in a function-entity type table. To draw business areas in TFET you could use the Update Surrounding Lines in the Table menu. See figure 7.8 for an example table. Currently, the editor checks the constraints in figure 7.7.

Constraint	Check	Description
TU1	Immediately	All non-empty entries contain at least one letter.
TU2	Immediately	All entries in the leftmost column (column 0) are unique.
TU3	Soft	There are exactly five columns.
TU4	Soft	There are no empty entries in column 0.
TU5	Soft	The first entry in row 0 is an empty string and the entries one to four are: Create, Read, Update, Delete.
TU6	Soft warning	Each transaction uses at least one object class: a warning is given when all entries (except the first entry) in a column are empty.

Figure 7.6: Immediately checked and soft constraints on TUTs.

Constraint	Check	Description
FE1	Immediately	All non–empty entries contain at least one letter.
FE2	Immediately	All entries in the leftmost column (column 0) are unique.
FE3	Immediately	All entries in the topmost row (row 0) are unique.
FE4	Soft	There are no empty entries in row 0.
FE5	Soft	There are no empty entries in column 0.
FE4	Immediately	All entries that are not in row 0 or in column 0 contain a "CRUD" string.
FE5	Soft warning	Each transaction uses at least one object class: a warning is given when all entries (except the first entry) in a column are empty.
FE6	Soft warning	Each object class is created by some transaction: a warning is given when all entries (except the first entry) in a row does not contain the letter 'C'.

Figure 7.7: Immediately checked and soft constraints on FETs.

	Document acquisition	Document disposal	Document circulation	Inter-library traffic	Document preservation	Finance	Budget planning	Member services
TITLE	C	D	U					
DOCUMENT	C	D	U	U	U			
ORDER	C					UD		
PUBLISHER	C							
BUDGET						R	CRU	
PAYMENT						CD		
INVOICE						CD		
PASS			R					CD
MEMBER			R					CUD

Figure 7.8: Example function-entity type table partitioned into business areas.

Chapter 8

Tree Editing

8.1 Editing Trees

Tree editors are document editors to edit text that is presented in the form of a tree. The nodes are pieces of text, the edges show the hierarchical relationship between the nodes. Each tree has a unique root node and the tree cannot contain cycles, by definition. These are immediately enforced constraints.

At the moment there are not many differences between the two tree editors. In fact, the tree editors can read in each others trees.

8.2 Edit and View Mode

The tree editors edit trees, like diagram editors edit graphs. See chapter 3 for all the diagram editing commands. During editing a tree, it should be visible which node is the root of the tree, so each root node is represented by a downwards pointing arrow, see figure 8.1.

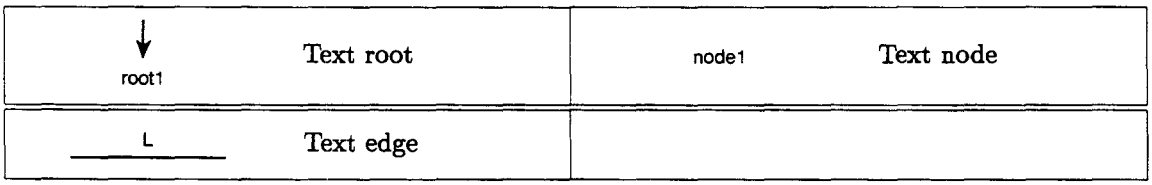


Figure 8.1: Textual tree nodes and edges.

The trees that are used in [18] look rather different from the graph-like documents that you edit in this editor. Therefore, the tree editors have a **forked tree** mode in which the graph is redrawn to look like a real tree [1]. In the forked tree mode it is not possible to edit the tree, but it can be loaded, saved, printed, previewed, zoomed and moved by the arrow buttons.

Below the Node and Edge buttons at the left edge of the main window, there is a pair of radio buttons to toggle between 'editable graph' mode and 'forked tree' mode.

In the 'forked tree' mode, the arrows are deleted from the root nodes and all children of a parent node will be connected to the parent as a 'fork': there is a simple built-in layout algorithm which

[1] Biologists would not agree, however.

takes care of the proper drawing of the line pieces. The algorithm does not reposition the text nodes, so the user determines where the nodes are drawn, but TCM decides, taking the node positions into account, how the edges are drawn. See figures 8.2 and 8.3 for the same tree in the two different modes. You see that the final tree layout is determined by the node positions only.

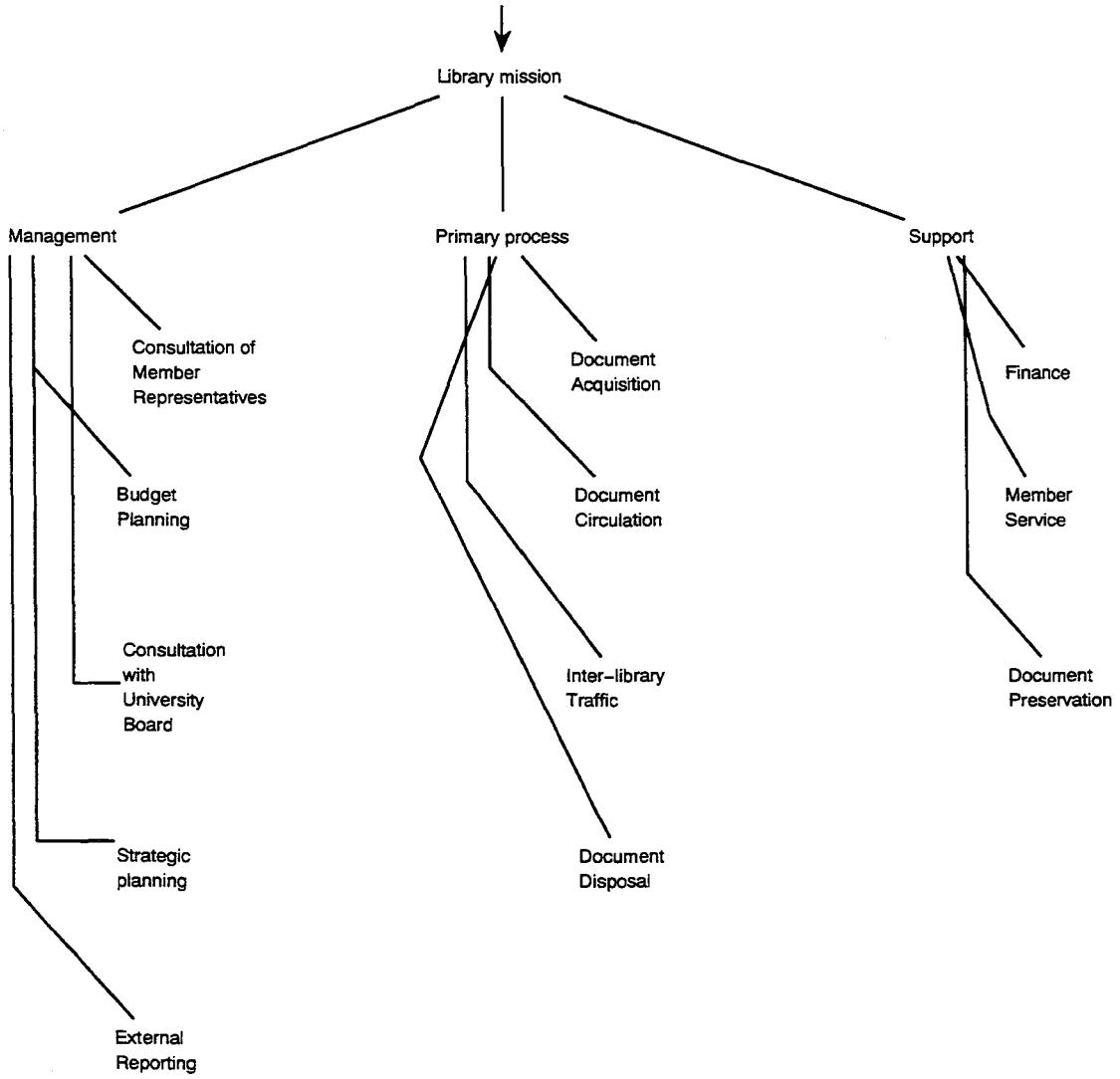


Figure 8.2: Example tree in edit mode.

8.3 The Generic Textual Tree Editor (TGTT)

This editor behaves as is described in the previous section. It does not impose any other constraints on the trees that are drawn.

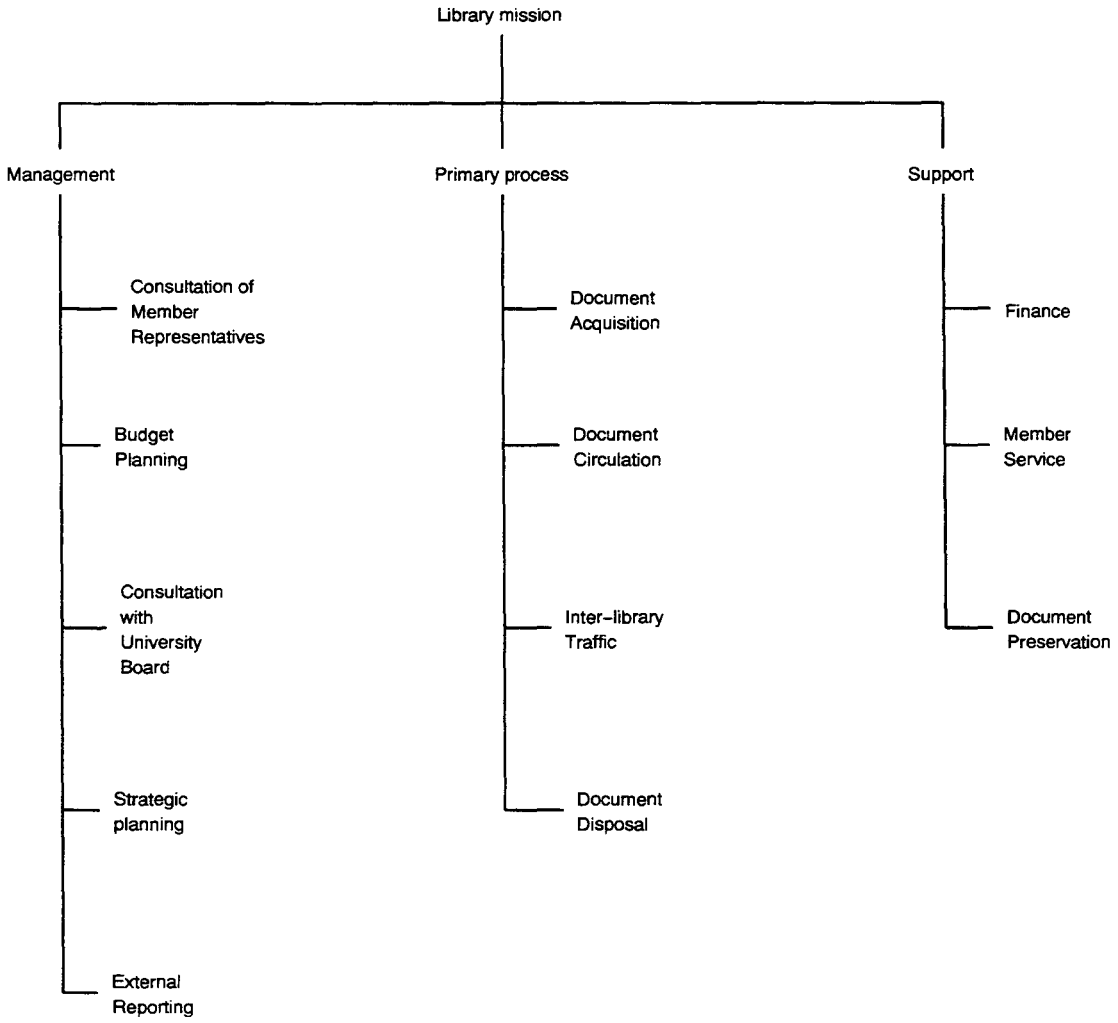


Figure 8.3: Example tree in forked tree (view) mode.

8.4 The Function Refinement Tree Editor (TFRT)

This editor is intended to be used for making function refinement trees, previously called **function decomposition trees** in [18]. It has a soft constraint that there is a single root node and that all nodes are reachable from that root node.

Appendix A

Frequently Asked Questions

A.1 What is the use of this FAQ?

Like most lists of frequently asked questions, this FAQ contains a compilation of questions that have actually been asked to us in the past or questions of which we think users might want to ask. All the questions and answers are formulated by us so they are probably biased with our opinions.

A.2 What is TCM?

TCM stands for the Toolkit for Conceptual Modeling. It is a suite of graphical specification editors that is intended to be used for software specification. But TCM is also very useful for drawing graph-like diagrams and for drawing tables.

The acronym TCM stands also for a number of other things like Trellis Coded Modulation (encryption theory), The Computer Museum (http://www.tcm.org), Traditional Chinese Medicine and the Texas Chainsaw Massacre (a horror movie). Obviously our TCM has nothing to do with all this.

A.3 Where can I get TCM?

TCM distributions with executables are uploaded to the ftp-site ftp://ftp.cs.vu.nl/pub/tcm. Up to now compilations have been made for Sun Solaris 2.x (Sparc and x86), SunOS 4.x.x (Sparc), Linux 2.0 (x86), IRIX 5.x (SGI), AIX 4.x (RS6000) and HP-UX 10.x. You can download the software via the web page http://www.cs.vu.nl/~tcm/software.html. On this web page and in the file README.TCM at the ftp site you can find which distributions currently exist and what is included in them.

A.4 How do I install TCM?

For a starter, read section 1.4. Alternatively, you can find installation instructions in the file README.TCM and on the web page http://www.cs.vu.nl/~tcm/software.html.

A.5 How do I start up TCM?

You start the TCM start-up tool with the command `tcm`, or you can start up individual editors like we explained in section 1.4. If starting up TCM does not succeed then we have some suggestions for

you:

1. Assuming that TCM is installed correctly (see question A.4), you should set `$TCM_HOME` to the globally accessible TCM directory. Check also that the tools are accessible and in your `PATH` variable (check this for instance with the Unix command `which`). If not, add `$TCM_HOME/bin` to your `PATH` variable.

2. Make sure that you run X Windows and that your `DISPLAY` variable is set correctly (you can test this with `echo $DISPLAY`). Note also that when you run an X application remotely, your X display should be open for it. See the manual pages xhost(1) and xauth(1) for how you can safely open your X display for remote applications.

3. Check the version of the operating system you are running, for instance with `uname -a`. Maybe you have installed a TCM distribution for a different OS. With the command `file 'which tcm'` you can see what type of executables the distribution contains.

4. If you cannot start TCM because of messages about libraries that cannot be opened like: "lib*BlaBla*.so: can't open file" then proceed with question A.22.

5. If TCM immediately crashes with a message like "X Error of failed request" then proceed with question A.21.

6. If TCM does start up but displays some dubious error messages about failed assertions or implementation errors then proceed with question A.43.

7. If all this does not help, you can always send an e-mail to tcm@cs.vu.nl.

A.6 Where can I find the user manual of TCM?

All TCM distributions contain a user manual. In the file `$TCM_HOME/doc/usersguide.ps` from the TCM distributions you can find a PostScript copy of the user's guide of the concerning TCM version. An HTML version is in `$TCM_HOME/doc/usersguide.html`. The latter can be read with a HTML web browser. There is also an on-line help in the Help menu of the editors but that is very limited.

You can also retrieve the TCM manuals from the TCM ftp site ftp://ftp.cs.vu.nl/pub/tcm. The web page http://www.cs.vu.nl/~tcm/usersguide.html contains the latest version of the user's guide in HTML format. This user's guide is also separately available in PostScript format, compressed by gzip, in ftp://ftp.cs.vu.nl/pub/tcm/tcm-usersguide.ps.gz.

A.7 What else can I read about the methods supported in TCM?

If you want to know a little bit more about the methods supported in TCM, you can read appendix C. There is a book written by Roel Wieringa [18] that treats a number of methods that are supported by TCM in more depth. See http://www.cs.vu.nl/~roelw/RE1.ps for the table of contents. http://www.cs.vu.nl/vakgroepen/infsys/roelw_publ.html contains the bibliography of Roel Wieringa. A number of his reports and articles can be downloaded in PostScript format via this page or directly from the ftp-site ftp://ftp.cs.vu.nl/pub/roelw. For finding information about requirements engineering and software specification methods on the web, the page
http://research.ivv.nasa.gov/~steve/resg/archive/RESources.html is an excellent starting point.

A.8 On what systems does TCM currently run?

In principle, TCM should be able to compile and run on all Unix systems that have X Windows and Motif. At the moment compilations have been made on Solaris 2.x (Sparc and x86), SunOS 4.x.x, Linux 1.2 (x86, a.out), Linux 2.0 (x86, ELF), IRIX 5.x (SGI), AIX 4.x (RS6000) and on HP-UX 10.x. See the file README.TCM at ftp://ftp.cs.vu.nl/pub/tcm for what distributions currently exist or see http://www.cs.vu.nl/~tcm/software.html. If you wish another compilation for TCM (providing it is Unix with X/Motif), you can give us a request by sending e-mail to tcm@cs.vu.nl, although we do not promise anything.

A.9 What should I do when I do not have Motif?

Most commercial Unix systems have the Motif library standardly included. However, Motif is often not included in older operating systems like SunOS or in free operating systems like Linux. Therefore there are two Linux and two SunOS distributions of TCM. One has the Motif library statically linked in and the other has not. The distributions that contain a statically linked Motif library have the string `-motif-` in their file names. Motif itself is copyrighted so we are not allowed to put the Motif library itself (libXm.so) in the distributions. Of course you can also buy Motif yourself. See the web page http://www.rahul.net/kenton/faqs/mfaq_index.html for the Motif FAQ including where you can get Motif.

A.10 Can I obtain the source code of TCM?

Due to copyright, the TCM source code of the version that is described in this manual, is not publically available. So normally you cannot obtain the sources, except when we are convinced that giving the sources is beneficial for both of us.

A.11 Can TCM be ported to my system?

It is not difficult to port TCM to some Unix, if it has a decent C++ compiler and it has X11 and the Motif library. Porting merely consists of editing a configuration file and probably a handful of conditionally compiled source code lines have to be added or changed, because some Unix calls might be different or because the compiler is somewhat idiosyncratic. If you wish another compilation for TCM (providing it is Unix with X/Motif), you can give us a request by sending e-mail to tcm@cs.vu.nl, although we do not promise anything.

A.12 Do I have to obtain a license for TCM?

The copyright notice can be read in the file COPYRIGHT.TCM that can be found at ftp://ftp.cs.vu.nl/pub/tcm/COPYRIGHT.TCM. This notice treats this issue.

A.13 In what programming language is TCM written?

This version of TCM is written in C++ and developed partly under Solaris and partly under Linux. In fact it is programmed in a subset of C++ and it tries to follow a consistent set of programming rules and recommendations. The same source tree has been compiled with the Sun CC compiler SC3.0.1 on Solaris, with GNU g++ (v. 2.6.3 and v. 2.7.2) on Linux, Solaris and HP-UX, on SunOS with the

AT&T C++ compiler (v. 3.0.1), on IRIX 5.x with the Delta C++ compiler and on AIX 4.x with the xlC C++ compiler.

With the Sun CC compiler and with the Delta C++ shared object libraries were created. The other compilations are statically linked. The reason that with g++ no shared libraries were created was because of the curious way that this compiler treats C++ templates.

TCM uses the Xlib and Xt libraries (X11R5 or X11R6) and it uses the Xm library of OSF/Motif (versions 1.2 and 2.0 both work). However, when TCM is compiled against Motif 1.2, it can not be run with Motif 2.0 and vice versa. Check the file README.TCM which Motif version is expected.

A.14 Can I receive e-mail notification when TCM is updated?

Yes. Simply follow the link to "The URL-minder: Your Own Personal Web Robot" in http://www.netmind.com/URL-minder/URL-minder.html and register the following URL: http://www.cs.vu.nl/~tcm/software.html. This page is always changed when a new TCM release is made. This free service is brought to you by Netmind at: http://www.netmind.com/. Note that there's nothing special about TCM as far as URL-minder is concerned. You can register *any* software web page and, more generally, any http or ftp URL with this service.

A.15 Why is Motif used for the GUI?

There are two widespread complaints about Motif. The first complaint is that it is expensive commercial software. The second complaint is that it is not suited for being used in object oriented programs written in C++ [1].

Our answer to the first complaint is that people do not have to buy Motif for working with TCM. On some systems, like Solaris, Motif is part of the standard software. On others, like Linux, Motif is not included. But we have made Motif libraries part of the TCM executables for these systems. This is allowed by the Motif copyright. Furthermore, the application environment specification of Motif is free, and work is in progress to create a public domain Motif clone, called LessTif [2] (see http://www.lesstif.org).

Our answer to the second complaint, about C++ and Motif, is that in theory, it is true. But with the right discipline you can write object oriented programs in almost any programming language and without discipline you cannot write object oriented programs in C++ (as C++ is a hybrid between an object oriented and a procedural programming language). Young [1] gives a good introduction to Motif programming in C++. What counts most is that the program is based on an object oriented specification and design and that a suitable mapping is found to the programming language. We claim that this is the case for TCM: It is based on an object oriented specification and design and this design maps almost directly to the program which is a combination of C++ and Motif, whose interface happens to be a C library. TCM *could* also be written in another programming language. In fact, this is already happening, see question A.17.

But why is Motif used and not some of the other GUI libraries? Motif gives us a lot of advantages. It is fast and very reliable. Motif is standardized and very well documented. There are a lot of very good learning and reference books for Motif such as [1, 2, 3, 8] [3]. The Motif interface is more or less

[1] Is it possible nowadays to write a good program that is not object-oriented? If not, is that because people use *object oriented* as a synonym for *good*?

[2] We have tried to build TCM on Linux with LessTif version 0.80. We got working executables but not the entire GUI looked good. We will look if that can be fixed and maybe we can soon upload a version of TCM for Linux that works with LessTif and that looks as good as the original.

[3] At least these are the books that we used. There are a number of more recent books. See the Motif FAQ on http://www.rahul.net/kenton/faqs/mfaq_index.html or posted in news:comp.x.windows.motif.

standard in Unix systems, often in form of the common desktop environment (CDE). There are also many standard and free Motif widgets. The user does not need much time to learn to work with a Motif GUI and the user does not need to install special software to be able to use it.

Of course we found also some disadvantages: it costs some time for a programmer to master Motif and there are certain things that have to be done in the underlying layers of Motif: Xt and Xlib, which are more complicated from a programmer's point of view. Furthermore, Motif applications are compiled, not interpreted, which makes Motif not well suited for very rapid prototyping or programming by trial and error, like you can do in Tcl/Tk. This means that you have to know what you want to program in advance.

A.16 Did you use other tools or widget sets to build TCM?

TCM is almost entirely made with our bare hands using the standard Unix tools for programming. But TCM itself proved to be a great tool for making the design. For instance, TCRD was used for the class structures, TFRT for the system decomposition and the source code organization and TGD was used to draw the structure of the widgets. Furthermore, all diagrams and tables in the user manual and the technical documentation are drawn with TCM.

For generating documentation from the source code we used the great and simple to use package DOC++. DOC++ can be got from: http://www.zib.de/Visual/software/doc++/index.html.

For testing and debugging TCM, we made use of the program called purify (http://www.rational.com/products/purify/index.html). Purify instruments a program to detect runtime memory corruption errors and memory leaks. It is a very powerful and helpful tool.

The user interface of TCM is build with the Motif 1.2 widget set. The bubble help texts are implemented with the Lite Clue widget (http://www.compgen.com/widgets/index.html#ALiteClue).

For the representation of the diagrams and tables we did not use any special widgets. The lowest level of drawing is done with Xlib calls. The graph and its constituent parts are implemented as C++ classes, all written by ourselves.

A.17 Will there be a version for Win95, Macintosh etc. in the future?

The TCM project has recently got a sister project called WHERE, which is performed at the NASA Software Research Laboratory (see http://research.ivv.nasa.gov/projects/WHERE). Part of this project is the translation of TCM into Java (see http://research.ivv.nasa.gov/projects/WHERE/TCMJava). The software is at this moment still in beta stage, but it looks very promising. Because it is written in Java, TCMJava runs on Solaris, Windows NT, Linux and probably also on Windows 95, Macintosh and OS/2.

Via http://research.ivv.nasa.gov/projects/WHERE/TCMJava you can download TcmJava. Note that for running TCMJava you should install the Java Development Kit or the Java Runtime Environment, but on the above mentioned web page it is explained where you can get them.

Our plan is to continue the development of our TCM for X Windows and Unix systems written in C++. There are no plans to port TCM to systems that do not have X Windows and Motif (because we see TCMJava as that port). By the way, a port of TCM to another GUI library is no quick or easy job. The X11/Motif part consists of about 10000 lines of source code which should be rewritten.

A.18 How can I configure TCM?

TCM can be configured by means of a configuration file. The file `$TCM_HOME/lib/tcm.conf` is the system-wide configuration file. It may be overridden by a user's private configuration file in `$HOME/.tcmrc`. The configuration file consists of a number of sections which start with a keyword identifying the section and then a {. The section ends by a }. Sections are separated by white space (one or more spaces, tabs or newlines). Within a section there are a number of attribute-value pairs separated by white space and also enclosed by curly braces (basically it is the same format as the TCM file format in appendix B). Comments in the file start with a hash (#). The text in the same line after the hash is ignored. The section names and the attributes are case sensitive. In the file itself you find comment text which gives directions for what the attributes mean and what attribute values are possible.

A.19 Can I set X Resources for TCM?

Yes that is possible but only for colors and fonts. The resource names start with `Tcm*`. In the file `$TCM_HOME/lib/Tcm` you find the X resources with the settings that are currently used in TCM (but the file is not used by TCM). You can copy this file and adapt it to your wishes. You can load it with `xrdb -merge` or copy it in your `.Xresources` or `.Xdefaults` in your home directory. When you use the xrdb command then it must be issued each time when you restart X.

A.20 Why does TCM sometimes show fewer or different colors?

This is probably caused by other programs that already run and which occupy too much entries in the X color map – leaving not enough for TCM. Especially Netscape hogs up all the colors when it starts up. You can best start up TCM first because it takes only what it needs, and then start up Netscape and the other greedy beasts. Netscape can also be started with the options `-install` for using a private color map or with `-ncols <N>` to set the maximum number of colors that it uses. When TCM cannot find any colors (on a color display) then it starts with a private color map. When you start TCM with the options `-priv_cmap`, it always uses a private color map.

A.21 Why does TCM crash with "X Error of failed request"?

When you get a message like `X Error of failed request: BadAlloc (insufficient resources for operation)` etc., then this is probably caused by a lack of memory on your X server for the pixmap of the drawing area. The drawing area is implemented as a large X color pixmap, which consumes a lot of memory. The default pixmap is about 1300 by 1300 pixels. That is enough to draw a document on two A4 pages. You can create a larger or smaller drawing area by starting the editor with the command line option `-drawing "width"x"height"` or by changing the initial drawing area size in View menu of the TCM start-up tool. For instance, you can start TERD with `terd -drawing 800x600` which gives a drawing area that is 800 pixels wide and 600 pixels high.

A.22 Why won't TCM start saying "lib*BlaBla*.so: can't open file"?

This is due to to the fact that the system cannot find a shared library needed by TCM. When you execute the command `ldd 'which tcm'`, you see against which libraries TCM is linked and where these are found (if they are found) on your system. The remedy to this problem depends on the kind of *BlaBla*.

- If *BlaBla* is either libglobal.so, libgui.so, libeditor.so, libdiagram.so or libtable.so then TCM cannot find its own libraries. Check if they are installed in `$TCM_HOME/lib`. If they are, set your `LD_LIBRARY_PATH` variable to `$LD_LIBRARY_PATH:$TCM_HOME/lib`.
- If *BlaBla* is libXm.so then TCM cannot find the Motif library. Look in the `README.TCM` of the TCM distribution where it should reside. Check also the version number of the Motif library. As it is impossible to run with version 1.2 when it needs 2.0 and vice versa. When you do not have Motif, you should obtain a distribution in which Motif is statically linked (see question A.9). If you have Motif, you could try to set the `LD_LIBRARY_PATH` to include the Motif library directory.

  In the past, the Linux executables of TCM were stripped too rigorously so that a Motif library with a different minor revision number than the one we used, could not be dynamically linked. Hopefully the current version of TCM has fixed this problem.
- If *BlaBla* is libX11.so, libXt.so or something else with an X in it, then TCM cannot find one of the X Windows libraries. Look in the `README.TCM` where TCM expects them to reside and look where they actually are on your system. Install them in the expected directory, or append to your `LD_LIBRARY_PATH` the directory where they actually are.
- If *BlaBla* is something else, like for instance libC.so.5 or libg++.so.27 then some other Unix or C library cannot be found. Either set your `LD_LIBRARY_PATH`, if they are at a different place than expected or install them if that is possible.

If you are still in big need, you can always try to ask us for help or for the library that you need (provided that this is allowed) or maybe it is possible to build a distribution in which that library is statically linked.

A.23 Why does TCM complain about old versions when I load a diagram?

There are two possibilities.

1. The first most probable one is that your documents were indeed made by a newer version of TCM. For instance, you or someone else might have made the document with a newer version of TCM, maybe at another site. TCM editors read older file formats but they cannot read newer file formats and they always generate the current (most recent) file format. In Document Info you can see what file format the editor generates and what file format you have read. The remedy is to upgrade TCM to the newer version.
2. A second possibility is that the file format was not written correctly. This is a bug in the Linux version that was once reported. The third line of the saved file should contain the file format version, for instance `{ Format 1.2 }`. But instead some random number was written. Probably the cause is a buggy version of libg++.so and libstdc++.so. At least, it has been reported to us that after an upgrade of /usr/lib/libg++.so.27.1.0 and /usr/lib/libstdc++.so.27.1.0 to /usr/lib/libg++.so.27.2.1 respectively /usr/lib/libstdc++.so.27.2.1 this problem ceased to exist.

A.24 How can I print my TCM documents?

Maybe you do not have to print at all and is it sufficient to use the Show Preview command. If you have a PostScript printer, you can print your document directly with the Print command. If that does not work, check if the printer name is correctly set (Printer Name), and if the printer is on-line (Printer Queue). Otherwise try if you can save the document to file as PostScript and try to send the PostScript file to the printer.

If you have a non-PostScript printer that is capable of printing graphics, write your document as plain PostScript to a file. You have to convert this PostScript file to a format that is suitable for your type of printer. The Ghostscript interpreter/previewer, called as `gs`, can help to do this. You can select different output devices, which include drivers for most current printers. Do `gs -h` or `man gs` for more information. You can find Ghostscript from any of the GNU distribution sites, such as ftp://ftp.uu.net/systems/gnu or ftp://gatekeeper.dec.com/pub/GNU.

A.25 How can I include my TCM document in LaTeX?

Make sure that you save your document as encapsulated PostScript. Encapsulated PostScript documents use a bounding box which depends on the relative sizes inside the picture, not on the underlying page. This makes it possible to include it in an arbitrary place, scale it, rotate it etc.

The encapsulated PostScript file can be included in a LaTeX document by an extension package like `Psfig` or `epsf` (for LaTeX) or `graphicx` (for LaTeX 2_ε). With these packages it is possible to scale and rotate the included figure. All example diagrams and tables in this user's guide were made with TCM and included as encapsulated PostScript with the graphicx package.

If you do not need to scale or rotate, make sure that the document in the TCM editor fits to one page. To fit you can move it to the first page with the Center button and scale it with the Whole Document command in the Zoom menu.

A.26 Is it possible to create process decompositions in TDFD?

Not really. For each level you have to draw a separate diagram. The Diagram text field helps you to compose the right data and control process indexes, but it does not offer much more.

We made a specification of an editor that manipulates leveled data flow diagrams [7]. Our plan is to make an editor in which you can edit an entire data flow model, in particular all diagram levels. So stay tuned.

A.27 Are there plans for consistency checks across diagrams?

Yes. The first step will be implementing hierarchical diagram editors. That means that for example the entire DFD hierarchy is edited as one document and the inter-DFD consistency checks are all performed locally just like it is done now. The next step will be support for more specific methods (see question A.29).

A.28 Is it possible for the user to define its own symbols?

No. Within an editor the nodes, edges, shapes and the constraints on them can not be reconfigured by the user.

A.29 Do you have any plans to support other methods?

Yes. The method that we want to give extra support is the Yourdon Systems Method (YSM) (mainly for educational purposes). This toolkit should support YSM as a representative of the family of structured methods. This toolkit includes building (and checking) models that are made from diagrams built by different editors.

After that we want to support UML [13, 14] (both for educational and research purposes). UML is object oriented.

But keep in mind that these are all *plans*, no promises or commitments.

A.30 Do you have plans for code generation?

Yes. But these are still plans. But it is certainly possible to write your own program that generates specifications in whatever language from TCM files. The TCM file format is specified in appendix B.

A.31 Is it possible to drag and drop with TCM?

No. There is no real drag and drop between editors even not between the same type of editor. Drag and drop between the same type of editor (see it as an extension of the cut-paste buffer) is put on our wish list. But for the time being, if you want to move things from one editor to the other, you have to save it to file (with Save or Save selection) and load it (with Load or Append) into the other editor.

A.32 What file formats does TCM generate?

At the moment PostScript, EPSF and the TCM format is generated. TCM only reads its own format.

The TCM file format is specified in appendix B. If you take a look into a saved file, you will see that it is almost human-readable. Furthermore, it reflects the structure of a repository for TCM models, consisting of multiple documents. Each diagram file has four distinct parts: storage information, document information, the graph and the representation. Each table file has five distinct parts: storage information, document information, overall table information, the columns and the rows. With a little bit of imagination one can see many possible programs interfacing with TCM via this file format.

A.33 I want to draw an XYZ diagram, which tool should I use?

In figure 1.1 there is a list in which you can see which tool is intended for which notational technique. If you want to do something different: TCM contains a generic diagram editor, a generic table editor and a generic tree editor. In these editors almost no constraints are built-in so it is easy to (ab)use them. The specific tools offer some other specific graphical features, though, but the constraints can be a hindrance if they are not applicable to your diagram technique. But restrictions invigorate creativity.

A.34 Is it possible to make an editor for XYZ diagrams?

First of all you could probably use of one of the existing editors. When the diagrams can be described more or less as a graph, it is not very difficult to implement yet another editor for the XYZ diagram

technique, the same counts for most tabular formats. But even though it would be quite easy for us to make editors for many popular notations, a new editor has to fit in our research and educational interests as there are so many other interesting things to do.

A.35 Why did you make TCM while there are already better drawing programs?

Get lost.

A.36 How can I have more influence on the layout?

When you want to place your shapes were *you* want, not where TCM thinks that you want, you could set Point Snapping off and then you can move all the nodes, line handles and text labels exactly to the pixel where you want. If you set the autoresize toggle off, you can resize the diagram shapes and table rows and columns by hand. If you have difficulty to access a particular shape because it is too close to another shape, you can try to temporarily zoom in. But, when you use TCM a lot, you will inevitably find a situation in which an editor messes up your drawing. If that is so, please let us know.

A.37 How can I connect an edge with an edge?

When people draw a diagram by hand, you often see that nodes are connected in a tree-like fashion. In TCM it is quite difficult to draw this because you have to edit the diagram as a graph. In the TCM diagram editors, each line should connect two node shapes; a line cannot connect to another line. The only way to draw something that looks like a "fork" is to draw partially overlapping segmented lines. But an even number of lines that overlap in the drawing area becomes invisible, due to the fact that figures are drawn in so-called "exclusive-or" mode (see also question A.38). However in PostScript, overlapping lines are visible. As another option, if that is allowed in the diagram technique, you can use a small black dot or another small shape as the junction of the tree.

The tree editors have a special 'forked tree' mode where the branches are drawn as a fork. You can not edit the tree in this mode, that is only possible in the 'editable graph' mode.

A.38 Why are there black pixels left in the drawing area?

If you find the pixels droppings disturbing, do a refresh (<Ctrl+V>). If you are not interested in technical details, skip the rest of this question.

TCM draws in *exclusive or* (XOR) mode. In XOR mode, the new destination pixel is produced by the exclusive or of the old destination pixel with the source pixel. In this mode, you can easily draw and erase a figure. You draw a figure to let it appear, and when you redraw it, you erase it. By sequencing drawing and redrawing you can simulating a shape being moved or dragged. Furthermore, in XOR mode, shapes can overlap each other without damaging each other. The overlapping part is then white, and when one of the shapes is moved or removed, it will be redrawn, so that the overlapping part will appear black again. For some unknown reason, there is sometimes a small rounding error in drawing certain shapes that makes that some black pixels remain.

A feature of TCM is that it maintains a second pixmap in the background which contains a copy of the contents of the drawing area. It is used to copy back the contents into the drawing area quickly when the drawing area has been overlapped by other windows, resized etc. Although this solution consumes a lot of memory, it makes drawing graphics relatively fast. A drawback is that

pixel droppings may occur with some types of shapes, for instance on the corners of rectangles because they are copied, not xor-ed.

A.39 Why doesn't the BackSpace key function correctly in Linux?

All applications that run under XFree (the X11 port for Linux), seem to have the problem that the BackSpace key acts like the Delete key: they both delete the character after the cursor (they have the same key code). You can assign them the right key codes in a file called `.Xmodmap` in your home directory. Add to that file the following two lines (and restart X):

```
keycode 22 = BackSpace
keycode 107 = Delete
```

A.40 Why don't the menu accelerators function in Linux?

This was the case in the a.out version. In the ELF version it works.

A.41 Why does TCM now have a tea pot as logo?

Because there is an awful lot of T in it.

A.42 What has become of the tiny dragons?

They became annoying so we asked St.George to take care of them
(see http://www.innotts.co.uk/~asperges/george.html).

A.43 What do these "Assertion failed:" messages produced by TCM mean?

TCM is not a perfect program [4] and it is under continuous development. So it can happen that the internal graph, a shape view or a table in the editor becomes inconsistent [5]. Therefore, assertions are built-in in the code. They are used for internal consistency checking or for checking if a command can be executed safely. They prevent harm as much as possible. When these assertions fail, warning messages are added to the message log dialog and the message log dialog is displayed. The messages contain amongst others the source code file and line number where the assertion was checked and failed. This does not say much to an ordinary user, but they can be very helpful for us to fix bugs. So when you send a bug report to us, you should include these assertion failed messages. Furthermore, as the diagram file is in ASCII, you can also include a faulty document as plain text in your e-mail.

[4] But we do our best, really.

[5] When you are smart enough, you can even try to enforce an inconsistent editor state by editing a TCM document file by hand and load that inconsistent document into the editor. Most of the times TCM notices that it has read a mess but sometimes it becomes so confused that it crashes gracefully.

A.44 TCM does not behave as I expected. What should I do now?

Please send a bug report to tcm@cs.vu.nl. Even if you are not sure that it is a bug or a feature, your comments are very much appreciated. It will help to improve the editors and the documentation.

A.45 TCM crashed. What should I do now?

Do what you were asked and send a bug report to tcm@cs.vu.nl. Try to provide enough information that it can be fixed, like the name and version of the editor that you are using, on which platform you were working, what commands you performed etc. Please check also whether the error is reproducible. Furthermore, if you have a faulty TCM document, you can send it to us so that we can have a look at it. TCM document files are ASCII so you can e-mail them as a plain text message (attachment).

Appendix B

TCM File format

B.1 Introduction

Each TCM document is stored in a separate Unix file as plain text in an (almost) readable format. This chapter contains a specification of the TCM file format of documents generated by the TCM version of this manual, which generates the TCM file format version 1.2. The TCM editors read also older file formats, down to file format version 1.0, but they only generate the latest file format of that TCM tool. Older file format versions are not described here. Each file can be converted to the recent format by reading it in from a recent tool and then saving it back to file.

B.2 Elements of a TCM document

A document is stored as a number of **sections** consisting of a number of **fields**. The order of fields within a section is significant. The order of the sections in a diagram file is not significant except that the file should start with first the Storage section and then the Document section. The order of the sections of a table is significant as is explained in section B.5.

A section has a keyword indicating the kind of section and, when there is possibly more than one section of a certain keyword in the same file, it has an identifier which makes the section unique within the file. The rest of the section is enclosed by curly braces. Sections are separated by white space. **white space** is a sequence of one or more spaces, tabs, carriage-return or newline characters.

A field is an attribute-value pair enclosed by curly braces. The order of the fields in a section is significant. Attribute values are one of the types listed in figure B.1. Fields within a section are separated by white space. The attribute name and the value within a field are also separated by white space. Section names and attribute names are case sensitive.

Although the files generated by TCM do not contain comment, comment text is accepted in a TCM file. Comments in a file start with a hash (#). The text in the same line after the hash is ignored.

When a file is read, TCM checks the syntax and checks the semantics, e.g. that all required fields are present, that the attribute values have the correct type and it checks references (such as that an edge section has fields referring to existing node sections). It reports errors that are found together with a line number in the file in the error message log dialog.

type	description
<activation>	Trigger, Stimulus,Time,Unspecified.
<alignment>	Left, Center,Right.
<attribute>	<id>.<word>
<bool>	False,True
<contenttype>	AtomicSubject, Attribute.DataType,Unspecified.
<datatype>	Int, Real, Char, String.
<direction>	Up, Down, Left, Right.
<id>	Identifier that is represented as a long unsigned integer. <id>s in a field refer to a clause.
<linestyle>	Normal, Dual, Dashed, Dotted,Fat,Invisible
<number>	Natural number in decimal notation.
<persistence>	Instantaneous, Continuing.
<rational>	Rational number in decimal notation.
<string>	Sequence of zero or more characters enclosed by double quotes " ". Strings cannot contain newline characters. They are encoded as \ r ('backslash r'). A double quote is encoded as \" and a backslash in a string is encoded as \\.
<word>	Sequence of printable characters ended by a white space, a '{' or a '}'. Henceforth a word may not contain white space or the characters '{' and '}'.
<xlfd>	Font string in the X Logical Font Description format. Currently the font families: courier, times, helvetica, new century schoolbook and symbol are accepted.

Figure B.1: The value types of attributes stored in a file.

B.3 Storage and Document Information

Each TCM document starts with the Storage section and then the Document section in which the following information is stored:

Storage section

```
Storage
{
   { Format <rational> }   # File format, e.g. 1.20.
   { GeneratedFrom <word> }# Tool name and version that wrote this e.g. TGD-version-1.66.
   { WrittenBy <word> }    # Unix login name of who generates this.
   { WrittenOn <string> }  # Writing Date/time (Unix ctime format).
}
```

Document section

```
Document
{
    { Type <string> }      # Document type: e.g. "Generic Diagram".
    { Name <word> }        # Document name, e.g. mydocument.gd.
    { Author <word> }      # Unix login name of document creator.
    { CreatedOn <string> } # Document creation time e.g. "Wed Sep 24 15:42:47 MET DST 1997".
    { Annotation <string> }# Free annotation text.
}
```

B.4 Diagram Editor File Format

A stored diagram has a section for each node, edge, view and shape (in no particular order) which follow after the required Storage and Document section. Each node type, edge type and shape type has its own keyword. A view has the keyword `View`. In figures B.3 to B.5 you can see which keywords (and accessory sections) are generated and read by which tool. After the node, edge, view or shape keyword there is an identifier which is unique within the file. These identifiers are used for referring from one section to another section. Figure B.6 gives an overview of the global structure of the diagram file format in which file format section types are represented as object classes. The CRD does not show the significant order of the fields in a section. On the other hand, the CRD shows specializations and other relationships between classes and also some cardinality constraints which can not be made explicit in the file format sections. Specific node sections would be subclasses of class Node, specific edge sections would be subclasses of class Edge and specific Shape sections would be subclasses of classes NodeShape or Line.

Node sections

You can see what node types exist in figure B.2. Each node has a name, annotation and parent field. Some node types have additional fields which are given below.

```
<NodeType> <id>                      # e.g. EntityType 123456.
{
    { Name <string> }                # name of the node.
    { Annotation <string> }          # annotation text of node.
    { Parent <id> }                  # parent node, always 0 in current TCM version.
    # possibly other node attributes
}
```

Node type	Tool	Node type	Tool
Comment	All diagram editors	PSProcess	TPSD
ControlProcess	TDEFD	ProcessGraphNode	TRPG
ControlledDataStream	TSND	ProcessGraphRoot	TRPG
DataProcess	TDFD, TDEFD	Relationship	TERD
DataStore	TDFD, TDEFD	SNProcess	TSND
DataStream	TSND	SplitMergeNode	TDFD, TDEFD
DecisionPoint	TSTD	State	TSTD
EntityType	TERD	StateVector	TSND
EventStore	TDEFD	SubjectArea	TCRD,TERD (not used)
ExternalEntity	TDFD, TDEFD	TaxonomyJunction	TCRD,TERD
GenericNode	TGD	TextNode	TGTT, TFRT
InitialState	TSTD	TextRoot	TGTT, TFRT
ModeJunction	TCRD	ValueType	TERD
ObjectClass	TCRD		

Figure B.2: Node types and the tools in which they occur.

Edge type	Tool	Edge type	Tool
BidirectionalDataFlow	TDFD, TDEFD	EmptyEdge	TCRD, TERD,TPSD
BinaryRelationship	TCRD, TERD	Event	TRPG
ComponentFunction	TCRD	EventFlow	TDEFD
ConnectionEnd	TSND	Function	TCRD, TERD
ConnectionStart	TSND	GenericEdge	TGD
ContinuousDataFlow	TDEFD	IsaRelationship	TCRD,TERD
ContinuousEventFlow	TDEFD	TextEdge	TGTT, TFRT
DataFlow	TDFD, TDEFD	Transition	TSTD

Figure B.3: Edge types and the tools in which they occur.

The parent field is not used in the current TCM version but it will be used for hierarchical diagrams. The parent identifier refers to an existing node section. The idea is the following: a node in a diagram can be further specified as a sub-diagram. The newly created nodes and edges in that sub-diagram have that node as parent. Nodes and edges in the top-level diagram have parent 0 (which means they have no parent). Furthermore, in sub-diagrams, shapes representing higher level nodes may also occur. In TGD these structures will be almost unconstrained. In TDFD (TDEFD) and TERD (TCRD) these structures are more constrained, for instance only data processes respectively subject areas can be parent nodes and flows respectively relationships have to be balanced. The parent relationship is also used in the tree editors (TFRT, TGTT) but here the entire hierarchy is presented in one view. In some other editors like TSTD the parent relationship will remain unused (at least in YSM) and in that case the entire diagram is treated as a top-level diagram and the parents of the subjects are always set to 0. In the current version of TCM there is also only a top-level diagram so the parent fields are always 0.

Other attributes of node types are:

- ControlProcess (TDEFD)

```
{ Index <word> }                # index label, e.g. 1.2.2
```

- DataProcess (TDFD, TDEFD)

```
{ Index <word> }                # index label, e.g. 1.2.2
{ ProcessGroup <bool> }         # is data process a process group?
# possibly other data process attributes #
```

In this version of TCM, the process group field is always False (because hierarchical DFDs are not implemented yet). When the data process would be a process group [1] then it is parent of a number of children (processes, stores, flows and/or split-merge nodes.). When the data process

[1] A process group is also called a *compound process* or a *decomposed process*.

Node shape type	Tool	Node shape type	Tool
ArrowBox	TSTD	DoubleBox	TCRD
ArrowTextBox	TGTT, TFRT	Ellipse	TGD,TERD
BlackDot	TDFD, TDEFD,TGD	Hexagon	TSTD
Box	TCRD,TERD, TSTD,TGD,TSND	HorizontalBar	TDFD, TDEFD
Circle	TGD,TSND	LLCircle	TSND
DashedBox	TGD	MiniArrowEllipse	TRPG
DashedCircle	TGD	MiniEllipse	TRPG
DashedDiamond	TGD	RoundedBox	TGD,TRPG
DashedDot	TCRD	Square	TDFD, TDEFD
DashedEllipse	TGD	TCircle	TDFD, TDEFD
DashedRoundedBox	TGD	TextBox	All diagram editors
DashedTCircle	TDEFD	Triangle	TGD
DashedTriangle	TGD	TripleBox	TCRD
Diamond	TERD,TGD,TSND	URBox	TPSD
Dot	TCRD,TERD		

Figure B.4: Node shape types and the tools in which they occur.

Line type	Tool	Line type	Tool
Arrow	TDFD, TDEFD,TGD,TRPG	DoubleArrow	TERD, TDFD, TDEFD,TCRD,TGD
C1Arrow	TERD,TCRD	DoubleHeadedArrow	TDEFD,TGD
C2Line	TERD,TCRD	EndC1Arrow	TSND
DashedArrow	TDEFD,TGD	HLineArrow	TSTD
DashedC1Arrow	TCRD	Line	TGTT,TGD
DashedDoubleArrow	TGD	PredefinedArrow	TERD,TCRD
DashedDoubleHeadedArrow	TDEFD,TGD	PredefinedLine	TERD,TFRT,TCRD,TPSD
DashedLine	TGD	StartC1Line	TSND

Figure B.5: Line types and the tools in which they occur.

is not a process group then it is a primitive process and then the process has the following attributes:

```
{ Persistence <persistence> }    # Instantaneous or Continuing
{ Minispec <string> }            # Mini specification text
# possibly other leaf node data process attributes #
```

In this version of TCM it is possible to set the persistence (by default it is instantaneous) and to edit a minispec text, but they are further not used. When the persistence is instantaneous then the data process section contains the field:

```
{ ActivationMechanism <activation> } # only when Instantaneous
# possibly other discrete leaf node data process attributes #
```

The activation mechanism can be set in the current DFD editors (by default it is unspecified). When the instantaneous data process is activated by a stimulus then a stimulus field follows:

```
{ Stimulus <string> }     # input edge name; only if activated by stimulus
```

Otherwise when it is activated by time, a TimeExpression field is given instead:

```
{ TimeExpression <string> }# only when activated by time
```

The activation mechanism can also be Trigger. The trigger edge is then not given in this section but there should be an input edge labeled 'T' to this process.

So it is possible to set the activation mechanism and specify a stimulus or time expression and this information is written to file but it is further not used in the current version of TCM.

- DataStore (TDFD, TDEFD)

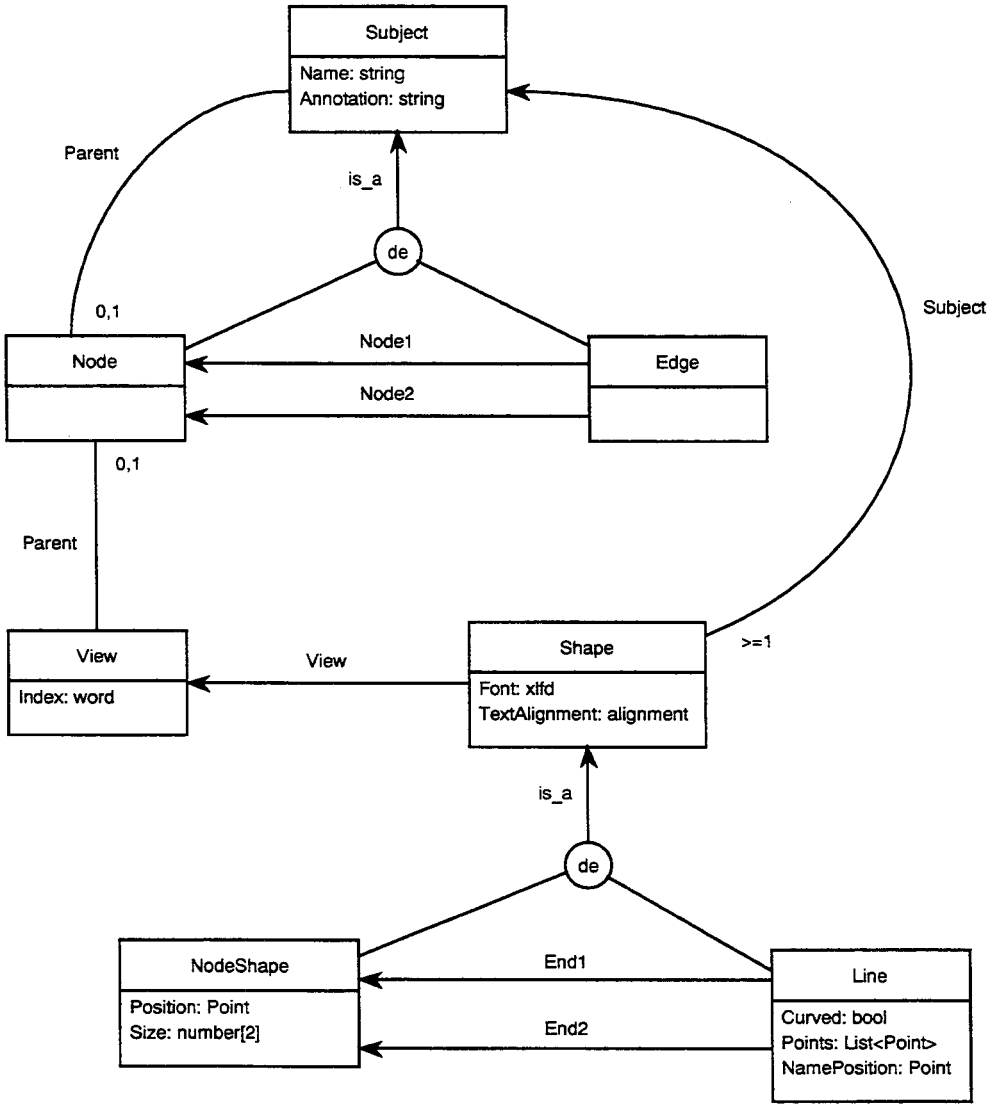


Figure B.6: Diagram file format sections seen as a CRD.

```
{ AtomicSubjects <number> } # number of entities/relationships
{ AtomicSubject <string> }  # entities/relationships (>= 1 fields)
```

In a future version of TDFD and TDEFD you will be able to specify the contents of data stores. For the moment the number of atomic subjects is always set to 0.

- InitialState (TSTD)

```
{ ControlProcess <string> }# the control process.
{ Actions <number> }       # number of actions of initial state.
{ Action <string> }        # initial action (>= 0 action fields).
```

In the current version of TSTD the name of the control process can not be specified. So the control process field contains an empty string. The number of actions specifies how many initial action fields follow. Actions are arbitrary single line strings (they cannot contain a newline).

- ObjectClass (TCRD)

```
{ Attributes <number> }     # number of attributes of object type
{ Attribute <string> }      # attribute string (>= 0 fields).
{ Actions <number> }        # number of actions of object type.
{ Action <string> }         # action string (>= 0 fields).
```

The number of attributes specifies how many attribute fields follow and the number of actions specifies how many action fields follow. Each attribute and each action is a single-line string.

- PSProcess (TPSD)

```
{ Operator <string> }       # process operator, e.g. "*"
{ IsRoot <bool> }           # is it a root process ?
{ IsAction <bool> }         # is it an action (leaf) process ?
{ Sequence <number> }       # sequence number in process tree.
```

The process operator is always a string of length 1 (when the process has no operator the string is a single space).

Edge sections

For which edge types are generated by which tools see figure B.3.

```
<EdgeType> <id>                 # e.g. BinaryRelationship 654321
{
    { Name <string> }           # name of edge.
    { Annotation <string> }     # annotation of edge.
    { Parent <id> }             # parent node.
    { Node1 <id> }              # 'departure' node
    { Node2 <id> }              # 'arrival' node
    # possibly other edge attributes #
}
```

Edge types also have a parent field which is intended to be used for hierarchical editors that are still to be build. For the moment the Parent identifier is always 0. The Node1 and Node2 identifiers should refer to existing node sections in this file. The other attributes of edge types:

- BinaryRelationship

```
{ Constraint1 <string> }   # first cardinality constraint.
{ Constraint2 <string> }   # second cardinality constraint.
{ RoleName1 <string> }     # first role name.
{ RoleName2 <string> }     # second role name.
```

- Function, ComponentFunction, StartC1Edge, EndC1Arrow (TERD, TCRD, TSND)

```
{ Constraint <string> }    # cardinality constraint.
```

- Transition (TSTD)

```
{ Event <string> }         # event string (including condition).
{ Actions <number> }       # number of actions in transition.
{ Action <string> }        # action string (>= 0 action fields).
```

  The actions field specifies how many action fields follow. Each action string is a single line text string.

- DataFlow, BidirectionalDataFlow, ContinuousDataFlow (TDFD, TDEFD)

```
{ Components <number> }    # number of sub-flows
{ Component <id> }         # sub-flow (>= 0 fields)
```

  When the components field is greater than zero then it has for each component a distinct field. This component field should refer to an existing data flow edge section. In the current version of TCM it is not yet possible to specify the components of a data flow. Therefore the components field is always 0. When the flow has no components then it has a certain data contents:

```
{ ContentType <contenttype> } # AtomicSubject, Attribute, DataType or Unspecified
```

  This field is only present when components is 0. This field is by default unspecified. It is not possible to set this field in the current version of TDFD or TDEFD. When the ContentType is not unspecified then according to the content type it has *one* of these three fields:

```
{ AtomicSubject <string> } # entity type/relationship name.

{ Attribute <attribute> }  # attribute of an atomic subject.

{ DataType <datatype> }    # values of a simple data type.
```

  But again, these fields cannot be filled in by the current version of TCM. Therefore data flow sections have no component fields and they have Unspecified as content type.

- EventFlow, ContinuousEventFlow (TDEFD)

```
{ Components <number> }    # number of sub-flows
{ Component <id> }         # sub-flow (>= 0 fields)
```

When components > 0 then it has for each component a distinct field. This component field should refer to an existing event flow edge. In the current version of TDEFD it is not possible to specify the components of an event flow. Therefore the components fields is always 0.

View sections

A view is a set of shapes that represents a sub-diagram. Sub-diagrams are hierarchical and are defined by a parent relationship between nodes. Which shapes exactly may occur in the view is determined by the diagram technique. For instance, in a DFD a view shows the refinement of a data process and in an ERD a view shows the refinement of a subject area. The current version of TCM has no hierarchical sub-diagrams implemented. So there is one single view that contains all the shapes of the diagram.

```
View <id>
{
    { Index <word> }          # index of hierarchical view.
    { Parent <id> }           # the parent node of the view.
}
```

The index of a view is the same kind of unique index that is used for data and control processes. The top-level view has index 0. The children of the top-level view are numbered 1 to n, the children of non-top-level view x have index $x.1$ to $x.n$. Each view except the top-level view has a parent node. The parent field therefore refers to an existing node section. The top-level view has as parent 0. The diagrams of the current version of TCM have only a top-level view.

The shapes that are contained in the view are not listed in the view section itself. But all shape sections have a reference to the view section in which they are contained.

Node shape sections

See figure B.4 for which node shape types are made by which tool.

```
<NodeShapeType> <id>              # e.g. Box 214365
{
    { View <id> }                 # view in which the shape occurs.
    { Subject <id> }              # the node that the shape represents.
    { Position <number> <number> } # center (x,y) position of shape.
    { Size <number> <number> }    # width and height of shape.
    { Font <xlfd> }               # text font of text strings.
    { TextAlignment <alignment> } # multi-line text alignment.
}
```

Each node shape is contained in an existing view and represents an existing node. The name label of the node shape is not given in this section but it is equal to the name of the node subject, but attributes of the labels, i.e. the font and text alignment are specified in this section. The current node shape sections do not have other attributes than the ones that are given above.

Line sections

See figure B.5 for which line type is made by which tool.

```
<LineType> <id>                   # e.g. Arrow 563412
{
    { View <id> }                 # diagram in which the shape occurs.
    { Subject <id> }              # edge that the line represents.
    { End1 <id> }                 # 'departure' node shape.
    { End2 <id> }                 # 'arrival' node shape.
    { Curved <bool> }             # straight (False) or curved (True)
    { Points <number> }           # number of line points
```

```
    { Point <number> <number> }       # line point (>=2 points).
    { NamePosition <number> <number> }# position of name label.
    { Font <xlfd> }                   # text font of text labels.
    { TextAlignment <alignment> }     # multi-line text alignment.
    # possibly other line attributes
}
```

Each line is contained in an existing view and represents an existing edge and connects two existing node shapes. The points field specifies how many point fields follow. A line has at least two points. The name position field gives the position of the name label. The text of the label can be found in the edge section. Because line labels can be positioned at free will, whereas node shape labels can not, only line sections have a name position field. Other attributes of line types are:

- C1Arrow, DashedC1Arrow, EndC1Arrow, StartC1Line (TERD, TCRD, TSND)

  ```
  { C1Position <number> <number> }  # position of constraint label.
  ```

- C2Line (TERD, TCRD)

  ```
  { C1Position <number> <number> }  # position of 1st constraint label.
  { C2Position <number> <number> }  # position of 2nd constraint label.
  { R1Position <number> <number> }  # position of 1st role name label.
  { R2Position <number> <number> }  # position of 2nd role name label.
  ```

  These constraint and role name labels can also be positioned at free will and therefore their positions are saved too.

- HLineArrow (TSTD)

  ```
  { AnchorPoint <number> <number> } # connection point with line.
  { Separator  <direction>  }       # relative position of separator
  { LineNumber <number> }           # (>=1) line segment to which separator belongs.
  ```

  The HLineArrow is the presentation of a transition edge in a STD. The separator field indicates whether the separator is above, below, to the left or to the right of the transition arrow. When it is to the left or right, the anchor point is the point where the separator line is connected to the arrow. When it is above or below, the anchor point is simply the middle of the separator line. Note that the length of the separator line is determined by the lengths of the event and action labels and it is not stored separately. The line number indicates the line segment to which the separator line belongs. The highest numbered line segment has the arrow head of the transition.

B.5 Table Editor File Format

Like diagrams, the table editor file format starts first with a Storage section and then a Document section. Directly after the Document section follows the Table section:

Table section

```
Table {
    { TopLeft <number> <number> }    # top-left (x,y) of entire table.
    { NumberOfRows <number> }        # number of rows in table (cells per column).
    { NumberOfColumns <number> }     # number of columns in table (cells per row).
    { MarginWidth <number> }         # min. distance between text and column line.
    { MarginHeight <number> }        # min. distance between text and row line.
}
```

In an earlier version of TCM other attributes where stored as well such as DefaultLineStyle, DefaultRowAlignment, DefaultColumnAlignment etc. but they are now treated as attributes of the table editor not of a table.

Row sections

After the Table section follow the row sections in consecutive order (they are numbered from 0 to NumberOfRows-1):

```
Row <number> {
    { Height <number> }          # all cells in row have the same height.
    { Alignment <alignment> }    # all texts in row have the same row alignment.
    { NumberOfCells <number> }   # a row having n cells has n+1 lines
    { LineStyle <linestyle> }    # each line piece can have a different style.
    { Text <string> }            # texts are stored per row.
    { Font <xlfd> }              # XLFD font description.
    { LineStyle <linestyle> }    # alternate: line-text-font-...-text-font-line.
    ...
}
```

The NumberOfCells of a row has to be equal to the NumberOfColumns field in the table section. This field indicates how many cells follow. The line pieces to the left and to the right of the cells are seen as part of the row. Each line piece has a separate line style. In the row sections the cell text strings and their fonts are stored. The size of the cell is determined by the row and column sizes and the text alignment of a cell is determined by the row and column alignment. In a row section, line style fields and cell text and font fields alternate and it always starts and ends with a line style field.

Column sections

After the Table section follow the column sections (numbered from 0 to NumberOfColumns-1):

```
Column <number> {
    { Width <number> }           # all cells in column have the same width.
    { Alignment <alignment> }    # all texts in column have the same column alignment.
    { NumberOfCells <number> }   # a column having n cells has n+1 lines
    { LineStyle <linestyle> }    # each line piece can have a different line style.
    ...
}
```

The NumberOfCells of a column has to be equal to the NumberOfRows field in the table section. Because the cell texts are already specified in the row sections a column section only needs the line style fields for horizontal line pieces. There are NumberOfCells+1 line style fields in a column.

Appendix C

Mini-tutorial on Notation Techniques

This appendix contains a short tutorial on the use of the notation techniques that are supported by TCM. Detailed information is given in R.J. Wieringa, *Requirements Engineering: A Framework for Understanding*, Wiley, 1996.

C.1 Data View Notations

C.1.1 Entity-Relationship Diagrams (ERDs)

The TCM convention for ERDs is described in detail in [18, chapters 7, 8].

Entity types

As usual, a named rectangle represents a named entity type.

Binary relationships

Binary relationships are presented by lines.

Cardinality constraints

Cardinality constraints are represented by annotations placed at the end points of these lines. For

Figure C.1: The placement of cardinality constraints.

example, in figure C.1, each business has an employment relationship to more than zero persons and each person has 0 or 1 employment relationships to a business. The end points of the line can also be annotated with the role that the entity at that end of the line plays in the relationship. Figure C.2 gives an example.

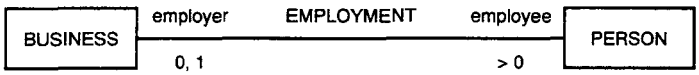


Figure C.2: The placement of role names.

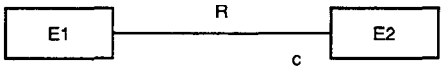


Figure C.3: The meaning of cardinality constraints.

In general, a cardinality constraint is represented by a set of natural numbers (see figure 4.4 for the syntax). For example, if c is a set of natural numbers, the constraint in figure C.3 is that each instance of E1 is related to n instances of E2, where $n \in$ c [1]. If no constraint label is shown, the convention is that the constraint is the entire set of natural numbers, i.e. it is no constraint. For example, in figure C.3, each instance of E2 is related to any number instances of E1. This includes the case that it is related to 0 instances of E1.

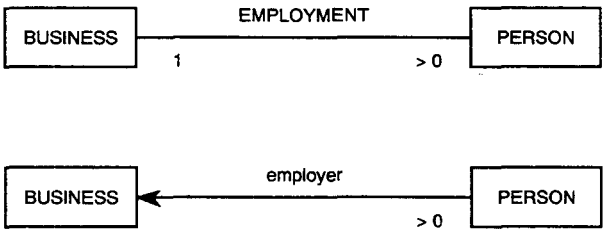


Figure C.4: The arrow representation of many-one constraints.

There are various conventions for the placement of the cardinality constraints, all of which are a source of confusion. The choice made in TCM is motivated as follows. We use the convention that a cardinality constraint of 1 can be abbreviated by an arrowhead. So the two diagrams in figure C.4 are equivalent as far as their cardinality constraints are concerned. They both mean that each person is related to exactly 1 business and that each business is related to at least one person. This means that the relationship is a mathematical function from persons to businesses, which explains the arrow convention. To facilitate a smooth transformation between these two representations, the cardinality constraint labels must be placed where they now are.

Note that the naming of the relationship usually must change when we switch to the arrow notation. In the line notation, there is no natural reading direction for the relationship name. For example, figure C.5 conveys the same information as figure C.1. In the arrow representation, by contrast, there is a natural reading direction and we adapt the relationship name accordingly. Often, the role name of the entity type at the arrowhead becomes the relationship name.

There are many other conventions to represent binary relationships. Figure C.6 shows different ways of representing the following constraints:

- Each existing E1 is related to at least one existing E2 and

[1] More precisely, each *existing* instance of E1 is related to n *existing* instances of E2.

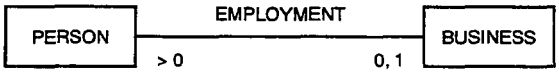


Figure C.5: The line representation of binary relationships is direction-independent.

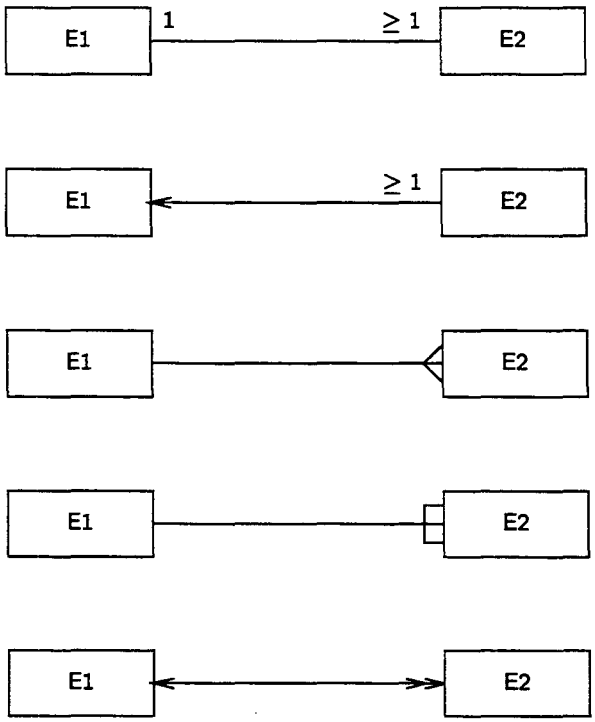


Figure C.6: Different conventions for representing the same constraints. TCM supports the conventions used in the upper two diagrams.

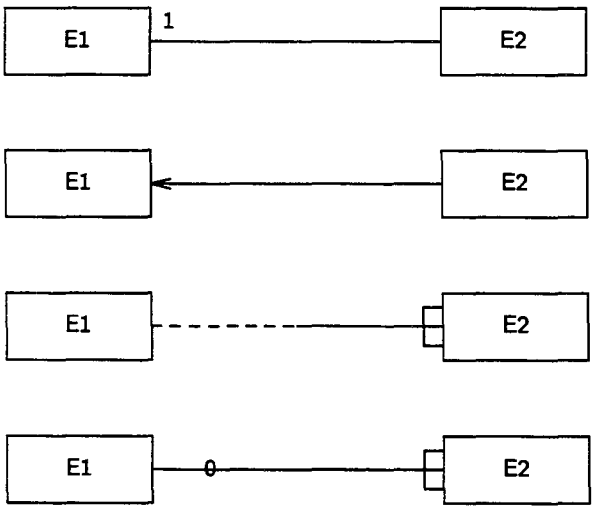


Figure C.7: Different conventions for representing the same constraints. TCM supports the conventions used in the upper two diagrams.

- Each existing E2 is related to exactly one existing E1.

Figure C.7 shows different ways of representing the following constraints:

- Each existing E1 is related to at any number (including 0) existing E2 and
- Each existing E2 is related to exactly one existing E1.

Relationships of higher arity

Figure C.8: The diamond representation for relationships.

A relationship is a Cartesian product of two or more entity types, called its *components.*[2] Relationships of arity higher than 2 are represented by a diamond, connected by arrows to the boxes that represent its components. These arrows represent the projection functions of a Cartesian product on its components. Figure C.8 contains exactly the same information as figure C.1. Note the placement of the cardinality constraints, which is at the root of the arrow. This agrees with the placement convention of constraints on relationship lines. In fact, one can view the arrows in figure C.8 as binary relationships between EMPLOYMENT and its two components. The meaning is that each business is related to at least one employment instance (and hence to exactly one person), and that each person is related to exactly one employment instance (and hence to exactly one business). This agrees with the meaning of figure C.1.

Value types

Value types (often called "data types") are represented by ovals.

Attributes

Entity attributes are represented by arrows from an entity type to an oval and relationship attributes are represented by arrows from a relationship diamond to an oval. This means that the TCM convention does not distinguish between "ordinary" relationships, which do not have attributes, and "associative entity types", which are relationships that can have attributes.

Is-a relationships

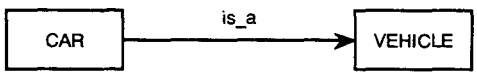


Figure C.9: The representation of is-a relationships.

An is-a relationship is a binary relationship that is an inclusion function. For example, figure C.9 shows that each CAR instance is also a VEHICLE instance. Extensionally, the set of all possible cars

[2]To be more precise, it is a *labeled* Cartesian product.

is a subset of the set of all possible vehicles. Intensionally, the set of properties shared by all cars includes the set of properties shared by all vehicles. CAR is called a *specialization* of VEHICLE and VEHICLE is called a *generalization* of CAR.

If there is more than one specialization of an entity type, then these must be grouped into *specialization groups*. This is represented by connecting the rectangles representing the specializations to a small circle called the *taxonomy junction* and connecting this with an is-a arrow to the rectangle representing the generalization. The taxonomy junction must be annotated as follows:

- A "d" means that the specializations are mutually disjoint.
- An "e" means that the specializations jointly exhaust the generalization.
- A "de" means the conjunction of "d" and "e", i.e. the specializations partition the generalization.

A generalization can be specialized by any number of specialization groups. For example, figure 4.5 means the following:

- Cars are vehicles and trucks are vehicles.
- The union of the set of all cars and all trucks equals the set of all vehicles. So vehicles are trucks or cars (or both).
- Diesel vehicles are vehicles and gas vehicles are vehicles.
- There is no vehicle both a diesel and a gas vehicle.
- There may be vehicles that are neither diesel nor gas vehicles.

C.1.2 Class-Relationship Diagrams (CRDs)

Classes

The CRD notation of TCM follows the convention that a class is represented by a rectangle subdivided into three areas, that contain, from top to bottom, the class name, the attributes, and the events that can occur in the life of the class instances. TCM can hide one or both of the event and attribute areas from view.

Relationships

Figure C.10: The CRD representation of relationships.

Relationships are represented by rectangles just as classes are. They are connected to their components by means of dashed arrows. The meaning is exactly the same as in the ERD case. Figure C.10 has exactly the same information content as figures C.1, C.5 and C.8. The line representation (figure C.1) is also allowed in the CRD convention. The advantage of the CRD convention over the diamond representation is that a rectangle allows easier placement of text inside the area. In addition, the CRD convention used in TCM allows representation of such complex structures as represented in figure C.11, which cannot be represented in the ERD convention. Figure C.11 represents the following structures. (To reduce clutter in the notation, we ignore the fact that relationships are actually *labeled* Cartesian products.)

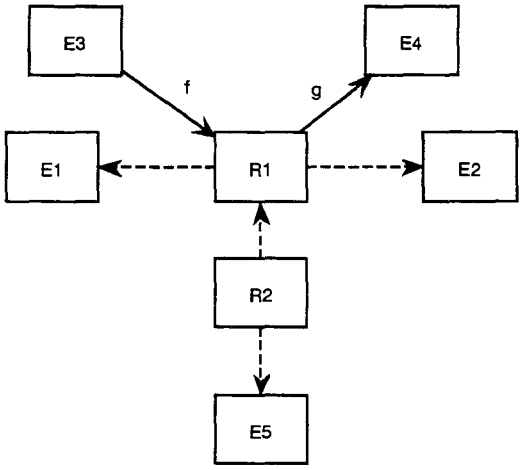


Figure C.11: The CRD convention can represent complex mathematical structures.

- R1 = E1 × E2
- f : E3 → R1
- g : R1 → E4
- R2 = R1 × E5

Is-a relationships

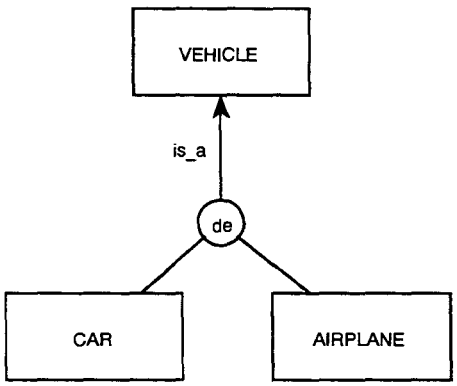


Figure C.12: Static specialization.

The CRD convention for representing is-a relationships extends the ERD convention with constructs to represent static and dynamic specialization. A static specialization group is represented by a small closed circle, called a taxonomy junction, and a dynamic specialization group is represented by a dashed circle, called a mode junction (see figure 4.7). In figure C.12, an instance of CAR will never become an instance of AIRPLANE and vice versa. An instance is a member of a specialization for life. By contrast, in figure 4.11, an instance of MARRIED PERSON may move to another of the

subclasses of PERSON. Here, an instance is an instance of a specialization only for part of its life. We call these specialization *mode classes*. For example, MARRIED PERSON is a mode class of PERSON, because a married person is a mode of a person. Details of static and dynamic specialization are given elsewhere [19, 20].

C.2 Behavior View Notations

C.2.1 State Transition Diagrams (STDs)

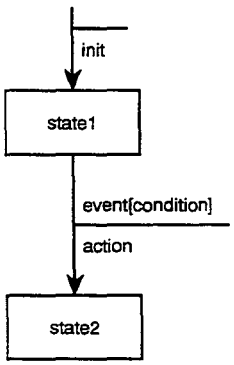


Figure C.13: The Mealy representation of state transition diagrams.

TCM supports the Mealy notation for finite state transition diagrams (figure C.13). States are named, and are represented by rectangles. State transitions are represented by arrows and are labeled by event[condition]/action pairs. The event is the *trigger* of the transition and can be viewed as the occurrence of an input. The condition is a *guard*. The precise meaning of the guard depends upon the method in which the notation is used. A minimalistic interpretation is that if the guard is false, an occurrence of the event will not trigger the transition. A more closed interpretation is that additionally, if the guard is true, an occurrence of the event will trigger the transition.

It also depends upon the method what kinds of conditions can occur in the guard. The current version of TCM does not support any method in particular and therefore imposes no constraints on what one can put in a guard.

The action part of the transition label is the output action generated by the transition.

Each Mealy STD must have an initial state, pointed at by an arrow that leaves from no node, and that can be labeled by an initialization action.

TCM also has *decision points* which are intermediary states that the machine may have between system transactions. Decision points are represented by a hexagon.

Mealy machines are used in Yourdon-style structured analysis, where they are used to specify control processes [23]. The interface of the control process must equal the interface of the Mealy machine. See section C.3.2 for control processes.

C.2.2 Process Structure Diagrams (PSDs)

Process structure diagrams are used in JSD to represent behavior. A PSD is a tree in which the nodes are labeled [9], [18, chapters 11, 12]. The leaves of the tree represent atomic actions and the root represents the entire process. Sequence is represented by a left-to-right ordering of the children

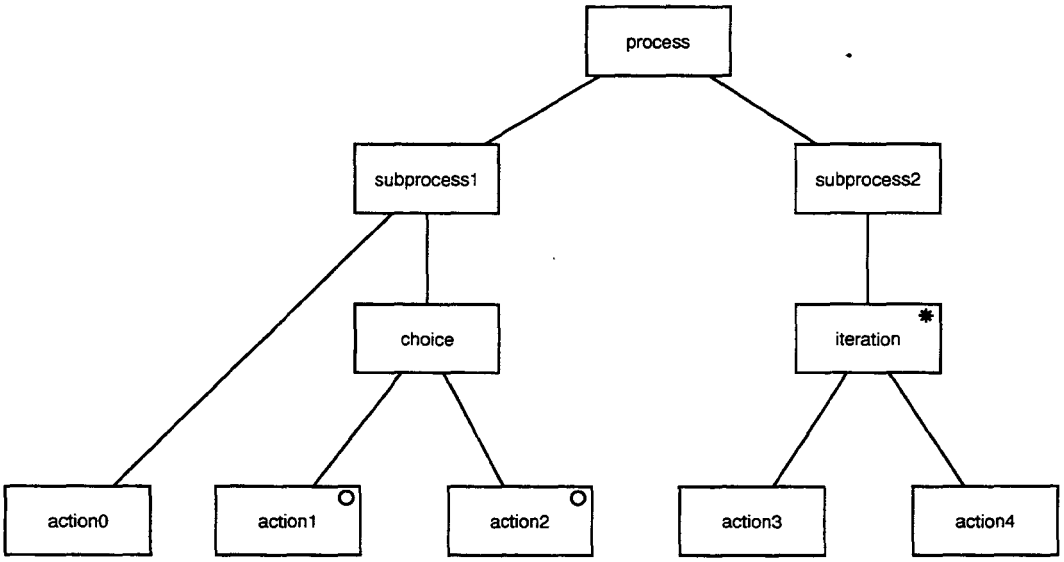


Figure C.14: A process structure diagram.

of a node. Iteration is represented by an asterisk label and choice by a small circle in the nodes that represent the options. Figure C.14 gives an example.

PSDs are equivalent to regular expressions.

A Mealy machine roughly equivalent to this is shown in figure C.15. The names of the nodes in a PSD can be reused as state names in a Mealy STD. However, the Mealy convention forces us to categorize an action as an input or output action, whereas in PSDs this is not the case. In figure C.15 we arbitrarily categorized all PSD actions as output actions.

In JSD, PSDs are used to represent processes in reality and to represent processes in the machine. If used to represent processes in reality, common actions between PSDs represent synchronous communication between these processes. If used to represent processes in the software, communication between processes is represented by means of system network diagrams, described in section C.3.3 below.

C.2.3 Recursive Process Graphs (RPGs)

A recursive process graph is a rooted directed graph in which the nodes represent states and the edges represent atomic actions or other processes. Figure C.16 shows an RPG equivalent to the PSD of figure C.14. Nodes in RPGs can be labeled, just as in Mealy STDs. Figure C.17 shows an RPG with labeled nodes.

An RPG has an initial node, which is pointed at by a small arrow and which can be labeled by the name of the process.

An edge in an RPG can be labeled with the name of an action or of a process. If it is labeled with a process name, the transition is equivalent to performing this process. Figure C.18 illustrates this. The RPG in figure C.18 is equivalent to that of figure C.17.

The call to another process can be recursive, as illustrated in figure C.19. This describes the process with possible traces $a^n c$ for $n \geq 1$.

Recursive process graphs are defined formally by Spruit and Wieringa [15], based upon the idea of recursive transition networks [21].

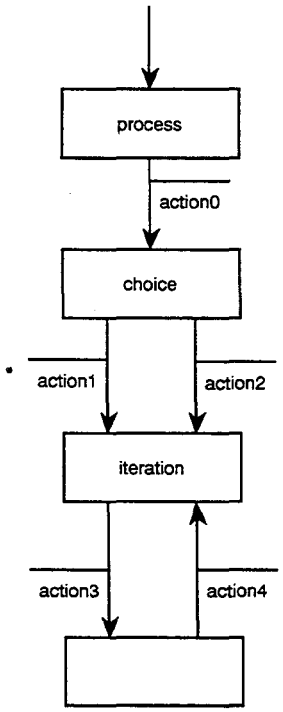


Figure C.15: A Mealy diagram roughly equivalent to figure C.14.

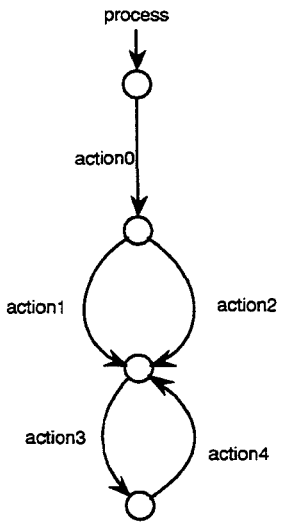


Figure C.16: A recursive process graph.

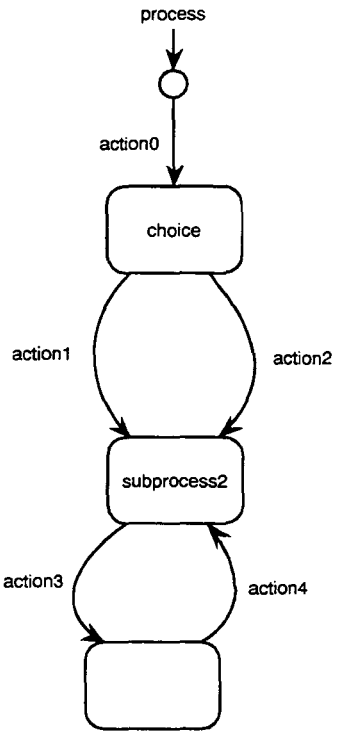


Figure C.17: A recursive process graph with labeled nodes.

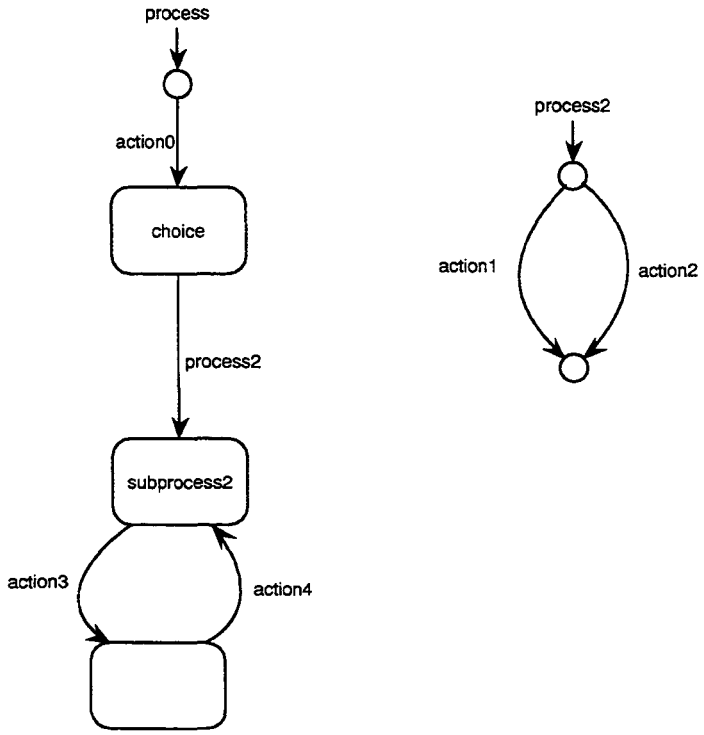


Figure C.18: A recursive process graph with a call to another process.

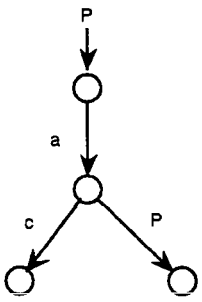


Figure C.19: A recursive process graph with a recursive call.

C.3 Function View Notations

C.3.1 Data Flow Diagrams (DFDs)

The components of a DFD

The TCM convention for DFDs is quite middle-of-the-road and follows mostly the DeMarco style [5], [18, chapters 9, 10]. A DFD is a directed graph with three kinds of nodes:

- Circles represent processes, also called data transformations or functions. A process is some computation by a software system.
- Squares represent external entities, these are entities with which the software system must interact.
- Two parallel lines represent a data store, which is a piece of software memory (e.g. a file or a variable).

The directed edges represent data flows between these nodes.

In figure 6.3, there are three processes, Confirm Registration, Check Request and Register students. When the external entity STUDENT sends a message test_request, which is a request to participate in a test, then the process Check Request retrieves the identifier of the test from the data store TESTS and the student identifier from the STUDENTS data store (the data stores are most likely implemented as files or in a database). If the test and student exist, and the student is allowed to participate in the test, then process Register students stores this fact in the TEST_REGISTRATIONS data store and Confirm Registration confirms this to the external entity. To make the DFD in figure 6.3 more precise, this model must be supplemented with precise process specifications, and a specification of the structure of the data stores and data flows.

Hierarchical DFDs

DFDs can be hierarchical. This means that a process can be specified by means of another DFD, which has the same external interface as the process being specified. Such a process is called a *compound* process. A process specified in another way (e.g. by means of a piece of text) is called *primitive*. This can be indicated by the letter P in the node that represents the process.

Compound processes give rise to a tree of DFDs. Processes in this tree are labeled by means of a Dewey numbering system that indicates the location of the process in the tree. For example, process 1.2 is the process with label 2 in the DFD that specifies the compound process with label 1. The current version of TCM does not support hierarchical DFD editing. The next version is planned to provide this support.

Combining DFDs with ERDs

DFDs can be combined with ERDs. The ERD should then represent the structure of the data in the data stores. The next version of TCM will provide this support.

C.3.2 Data and Event Flow Diagrams (DEFDs)

Control processes

DEFDs extend DFDs with a new kind of node, the control process, and new kinds of edges: event flows and continuous flows. A control process is represented by a dashed circle and represents an aspect of behavior. It must be specified by means of an STD that has the same interface as the control process.

This means that the event flows entering the control process must occur as events in the Mealy STD, and vice versa, and that the event flows leaving the control process must occur as actions in the STD, and vice versa. Figure C.20 contains a DEFD of which the control process is specified in figure C.21.

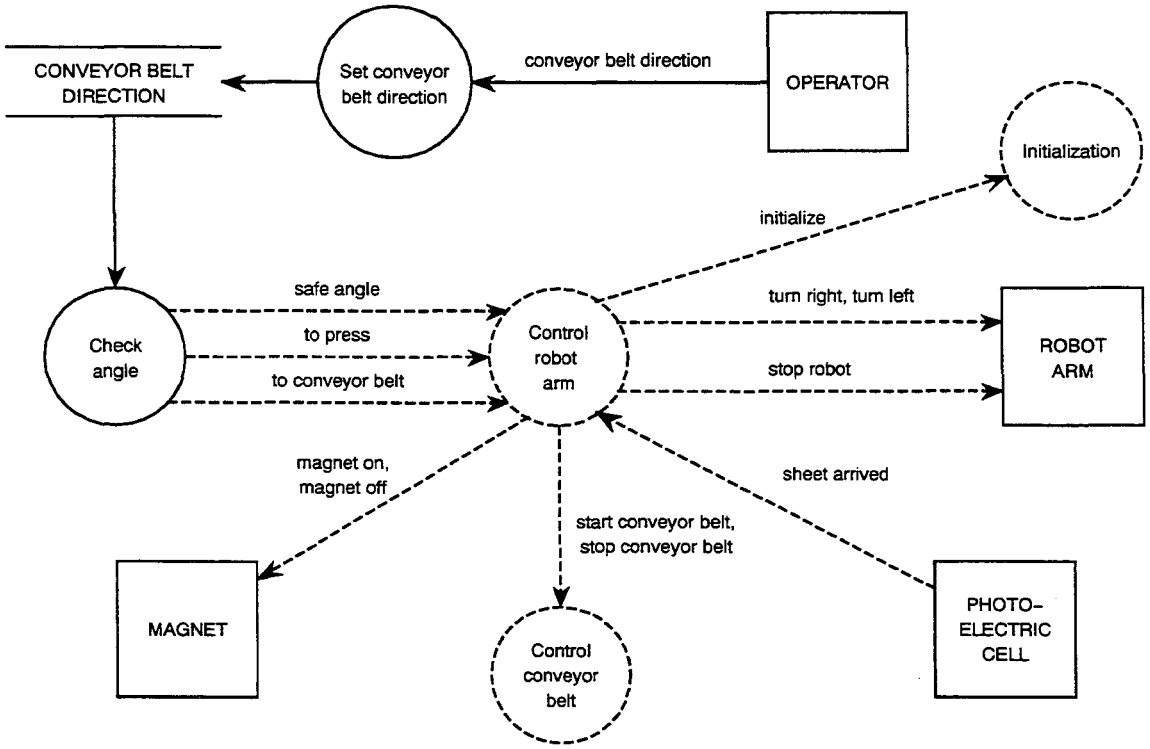


Figure C.20: A DEFD for a robot control process.

Event flows

Event flows are represented by dashed arrows. An event flow can carry a signal without any data contents. The precise meaning depends upon the method that uses this technique. See for example the YSM manual [23].

Discrete and continuous flows

A discrete flow carries a value that changes in discrete steps, a continuous flow carries a value that changes in a continuous way. Discrete flows are represented by arrows with a single arrowhead, continuous flows are represented by arrows with a double arrowhead. Again, the precise meaning depends upon the method used.

C.3.3 System Network Diagrams (SNDs)

SNDs are used by JSD [9] to represent communication between processes. SNDs are directed graphs with two kinds of nodes, that represent processes and communications. A process node must be specified by a PSD, just as a control process in an DEFD must be specified by an STD. There are three kinds of communication nodes:

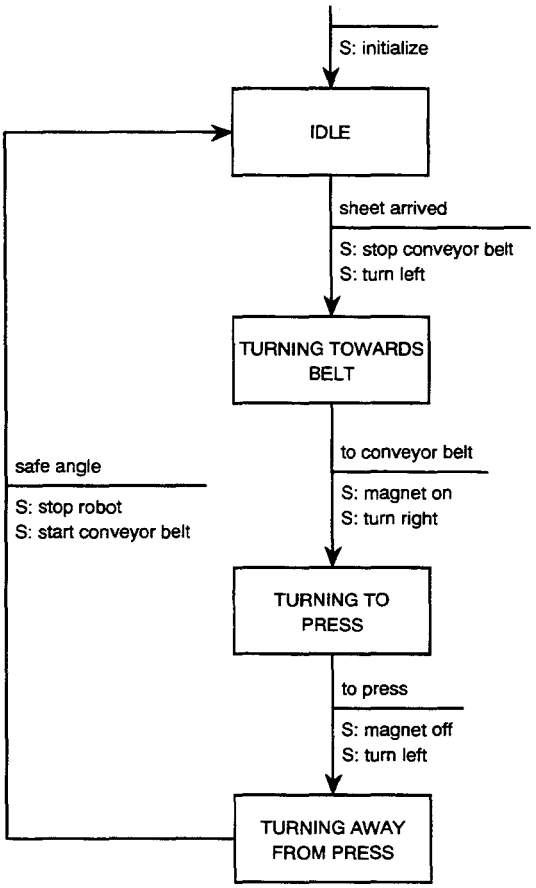


Figure C.21: STD for the robot control process of figure C.20.

- *Data streams*, represented by a circle. These are FIFO queues, somewhat like Unix pipes between two processes. Communication through a data stream connection is asynchronous.

- *State vector connections*, in which the reader process reads the state of the writer process. Initiative of the communication lies with the reader. The writer is not disturbed by the read action. The communication is synchronous. A state vector connection is represented by a diamond connected to the reader by an arrow and to the writer by an undirected line. The direction of the arrow represents the direction of data flow.

- *Controlled data stream connections*, represented by a circle with a small vertical line in it. The circle is connected to a reader and a writer, where an arrow is used to indicate the direction of data flow. Communication is synchronous and takes place on the initiative of the reader. The reader checks the current state of the writer and if this satisfies a certain condition, may update this state by sending it a message.

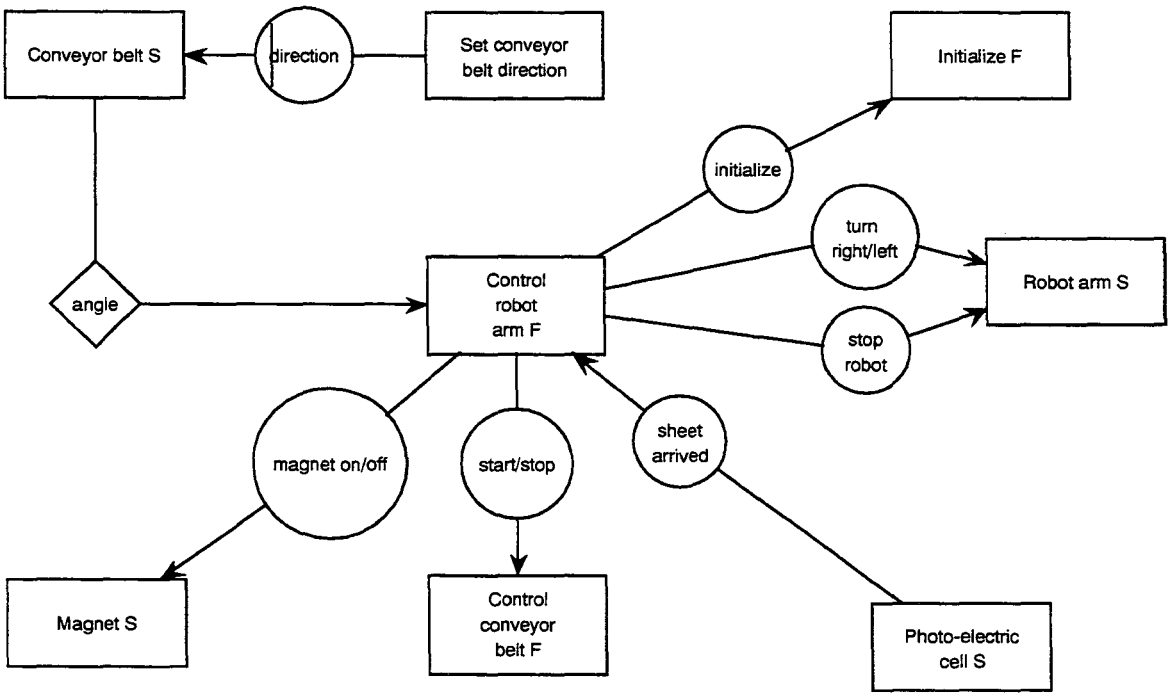


Figure C.22: An SND of the robot controller of figure C.20.

Figure C.22 shows an SND of the robot controller of figure C.20. All rectangles represent software entities. External entities are not shown. We used the convention to end the name of a software entity that represents an external entity with an S (for "surrogate"), and to end the name of a software entity that embodies a software function with an F. Each of the surrogate and function processes in the model must be specified by a PSD.

C.4 Tabular Notations

C.4.1 Transaction Decomposition Tables

A transaction decomposition table is used to set off software entities against external atomic system functions, called *transactions*. The entries of the table then represent the work performed by the software entities during the transaction. For example, figure 7.4 says that the transaction start_controlling_temperature requires some actions to be taken by software entities: A BATCH object must perform action do_temperature_ramp, etc.

Transaction decomposition tables can also be used in combination with ERDs and DFDs. The left-hand column then represents entity types or data stores, and the entries contain the letters C, R, U or D to indicate whether an instance of the entity type is created, read, updated or deleted during the transaction. The resulting table is also called a *CRUD table.*

Transaction decomposition tables can also be used in JSD to discover communications. They also help to maintain traceability. Methodological details are provided elsewhere [18, 17].

C.4.2 Transaction-Use Tables

A transaction-use table is a simple way to discover entity types from required system transactions. The leftmost column lists external system functions and the top row lists the basic Create, read, Update and Delete actions. The entries list the entity types or relationships that are created, read, updated or deleted during the function. See figure 7.5. Elaborate examples are given elsewhere [18, chapter 8].

C.4.3 Function-Entity Type Tables

A function-entity type table is almost the same as a transaction decomposition table. The top row now lists higher-level system functions and the leftmost row represents subject areas, which are at a higher level of abstraction than entity types (a subject area may encompass several entity types). The entries contain C, R, U or D, as in simple transaction decomposition tables.

Function-entity types are used in Information Engineering to find subsystems. These are identified by clustering subject areas and functions in such a way to minimize data flows between the clusters. See [18, chapter 6] for details and examples.

C.5 Function Refinement Trees (FRTs)

A function refinement tree is a tree in which the root represents the entire system mission and the leaves represent system functions. The hierarchy of nodes represents the refinement of functions into subfunctions. All nodes in the tree represent *external* functions.

An FRT can be used in combination with a hierarchical DFD to represent the hierarchy of DFDs. It is used in information engineering to represent external functions of an information system [18, chapter 6]. Of course, a tree can be used to represent any hierarchical decomposition and TCM imposes no constraints on the syntax of the tree.

Bibliography

[1] Douglas A.Young. *Object Oriented Programming with C++ and OSF/Motif.* Prentice Hall, 1991.

[2] Douglas A.Young. *The X window system, programming and applications with Xt.* Prentice Hall, 1994. OSF Motif edition.

[3] Marshall Brain. *Motif programming – The essentials and more.* Digital Press, 1992.

[4] P.P.-S. Chen. The entity-relationship model – Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[5] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press/Prentice-Hall, 1978.

[6] F.Dehne and R.J.Wieringa. Toolkit for conceptual modeling, user's guide for tcm 1.2.0. Technical Report IR-401, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, April 1996.

[7] F.Dehne and R.J. Wieringa. The yourdon systems method and the toolkit for conceptual modeling. Technical Report IR-414, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, December 1996.

[8] Dan Heller. *Motif Programming Manual*, volume 6 of *The Definitive Guide to the X Window System.* O'Reilly & Associates, 1991. For OSF/Motif Version 1.1.

[9] M. Jackson. *System Development.* Prentice-Hall, 1983.

[10] OSF (Open Software Foundation). *OSF/Motif Programmer's Guide*, 1992. For OSF/Motif Release 1.2.

[11] OSF (Open Software Foundation). *OSF/Motif Style Guide*, 1992. For OSF/Motif Release 1.2.

[12] OSF (Open Software Foundation). *OSF/Motif User's Guide*, 1992. For OSF/Motif Release 1.2.

[13] Rational. *Unified Modeling Language: Notation Guide, Version 1.1.* Rational Software Corporation, 2800 San Tomas Expressway, Santa Clara, CA 95051-0951, 1 September 1997. URL http://www.rational.com/uml/1.1/.

[14] Rational. *Unified Modeling Language: Semantics, Version 1.1.* Rational Software Corporation, 1 September 1997. URL http://www.rational.com/uml/1.1/.

[15] P.A. Spruit and R.J. Wieringa. Some finite-graph models for process algebra. In J.C.M. Baeten and J.F. Groote, editors, *2nd International Conference on Concurrency Theory (CONCUR'91)*, pages 495–509, 1991.

[16] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems.* Prentice-Hall/Yourdon Press, 1985. Three volumes.

[17] R.J. Wieringa. Combining static and dynamic modeling methods: a comparison of four methods. *The Computer Journal*, 38(1):17–30, 1995.

[18] R.J. Wieringa. *Requirements Engineering: Frameworks for Understanding.* Wiley, 1996. ISBN 0 471 95884 0.

[19] R.J. Wieringa, W. de Jonge, and P.A. Spruit. Roles and dynamic subclasses: a modal logic approach. In M. Tokoro and R. Pareschi, editors, *Object-Oriented Programming, 8th European Conference (ECOOP'94)*, pages 32–59. Springer, 1994. Lecture Notes in Computer Science 821. Revised and extended version appears in [20].

[20] R.J. Wieringa, W. de Jonge, and P.A. Spruit. Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems*, 1(1):61–83, 1995.

[21] W.A. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, October 1970.

[22] E. Yourdon. *Modern Structured Analysis.* Prentice-Hall, 1989.

[23] Yourdon Inc. *Yourdon™ Systems Method: Model-Driven Systems Development.* Prentice-Hall, 1993.

Index