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

updated to version 2.20

Frank Dehne

Faculty of Mathematics and Computer Science
Vrije Universiteit
De Boelelaan 1081a, 1081 HV Amsterdam
The Netherlands
(till December 2000)

Roel J. Wieringa
Henk R. van de Zandschulp

Department of Computer Science
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
tcm@cs.utwente.nl

January 20, 2003

Abstract

The Toolkit for Conceptual Modeling (TCM) is a collection of software tools to present conceptual models of software systems in the form of diagrams, trees and tables. A conceptual model of a system is a structure used to represent the behavior or decomposition of the system. TCM is meant to be used for requirements engineering, i.e. the activity of specifying and maintaining requirements for desired systems, in which a number of techniques and heuristics for problem analysis, function refinement, behavior specification, and decomposition specification are used.

TCM contains editors for two major sets of software specification techniques: Structured Analysis (SA) and the Unified Modeling Language (UML). The first set includes amongst others editors for ER-diagrams, data and event flow diagrams and state-transition diagrams. The set of UML editors includes amongst others a class-diagram editor, a use-case diagram editor and an activity diagram editor. Furthermore, TCM contains three generic editors for generic diagrams, generic tables and generic trees and also a number of special purpose editors such as two editors for JSD and a process graph editor.

The current version of TCM supports constraint checking for single documents (e.g. name duplication and 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 provides a warning when it checks the drawing). TCM is planned to support constraint checking across documents.

All editors have a similar user interface, and most of the time you don't need this manual. There is a simple on-line help facility. This user's guide annex reference manual is available in PostScript, PDF and in HTML format.

This document is intended to be a guide for both beginners and advanced users of TCM. Appendix A contains a mini-tutorial of the notation techniques supported by this version of TCM.

TCM runs on Unix and Linux systems with X Windows and even on Windows, running the CYGWIN/XFree86 environment. The ftp-site is ftp://ftp.cs.utwente.nl/pub/tcm. The TCM home page is http://www.cs.utwente.nl/~tcm. TCM is distributed under the GNU General Public License (GPL).

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 a structure used to represent the requirements or architecture of the system. TCM is meant to be used for specifying and maintaining requirements for desired systems, in which a number of techniques and heuristics for problem analysis, function refinement, behavior specification, and architecture specification are used. TCM takes the form of a suite of graphical editors that can be used in these design tasks.

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 A. 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 [22]. The UML diagram techniques are discussed in [23]. Bibliographic references can be found on page 182.

1.1.2 What is Included in TCM

This manual is the successor of [5] and [7]. 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:

- Generic Editors.
  - **TGD: Tool for Generic Diagrams**. This tool is for arbitrary graphs, it does not perform semantic checks.
  - **TGT: Tool for Generic Tables**. This tool is intended for arbitrary tables, it does not perform semantic checks.
  - **TGTT: Tool for Generic Textual Trees**. This tool is intended for generic trees in which all nodes are text labels. It does not perform semantic checks.

- Structured Analysis Editors.
  - **TESD: Tool for Entity Relationship Diagrams**.
    Entity-relationship diagrams are used in structured analysis and an extension of them, class diagrams (static structure diagrams), are used in the UML. This tool is intended for entity relationship diagrams (ERDs) that are used in Entity Relationship (ER) modeling based on the subset of the UML static structure diagrams. The ERD part of the UML class diagram technique is explained in [17, 18, 19]. The ER notation that TCM supports is explained in appendix A.1.1.
  - **TEFD: Tool for Data and Event Flow Diagrams**. This tool is intended for data and event flow diagrams (EFDs), that are used in the method of Structured Analysis for Real Time Systems (SA/RT) [20] and in the Yourdon Systems Method [31]. The technique is explained in appendix A.1.2.
  - **TSTD: Tool for State Transition Diagrams (Mealy)**. This tool is intended for state transition diagrams (STDs) according to the Mealy notational convention. This technique is explained in appendix A.1.3.
  - **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 [22]. The technique is explained in appendix A.1.4.
  - **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 [22]. The technique is explained in appendix A.1.5.
  - **TFRT: Tool for Function Refinement Trees**. This tool is intended for function refinement trees, also called *function decomposition trees*. Function refinement trees of this type are used and discussed in [22]. See also appendix A.1.6.
- UML Editors.

  The Unified Modeling Language (UML) is an object oriented modeling language. The notation and semantics of UML diagrams can be found in [17, 18, 19].
  - **TUCD: Tool for Use Case Diagrams**. This tool is intended for Use Case Diagrams that are used in UML for specifying interactions between a user and a computer system. The technique is explained in appendix A.2.1.
  - **TSSD: Tool for Static Structure Diagrams**. This tool is intended for Static Structure Diagrams that are used in UML for specifying class and object diagrams. The technique is explained in appendix A.2.2.
  - **TATD: Tool for Activity Diagrams**. This tool is intended for Activity Diagrams that are used in UML for specifying interactions between a user and a computer system. The technique is explained in appendix A.2.3.
  - **TSCD: Tool for Statechart Diagrams**. This tool is intended for Statechart Diagrams that are used in UML for specifying the sequences of states that an object of a given class goes through during its life in response to received stimuli, together with its responses and actions. The technique is explained in appendix A.2.4.
  - **TSQD: Tool for Sequence Diagrams**. This tool is intended for Sequence Diagrams that are used in UML for specifying an interaction arranged in time sequences.
  - **TCBD: Tool for Collaboration Diagrams**. This tool is intended for Collaboration Diagrams that are used in UML for specifying an interaction between an object and its related objects. The technique is explained in appendix A.2.5.

  - **TCPD: Tool for Component Diagrams**. This tool is intended for Component Diagrams that are used in UML as an implementation diagram for specifying the dependencies among software components. The technique is explained in appendix A.2.6.
  - **TDPD: Tool for Deployment Diagrams**. This tool is intended for Deployment Diagrams that are used in UML as an implementation diagram for specifying the structure of the run-time system. The technique is explained in appendix A.2.7.

- Miscellaneous Editors.
  - **TERD: Tool for Entity Relationship Diagrams (classic)**. This tool is intended for classic entity relationship diagrams (ERDs) that are used in Entity Relationship (ER) modeling [3]. The technique is explained in appendix A.3.1.
  - **TCRD: Tool for Class-Relationship Diagrams**. This tool is intended for class-relationship diagrams (CRDs) that are used in object oriented modeling. The CRD notation variant that TCM supports is explained in appendix A.3.2. TCRD is similar to TSSD and is actually the predecessor of TSSD.
  - **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) [8, 30]. This method is discussed in [22]. The technique is explained in appendix A.3.3. The notation of TDFD is a subset of the TEFD editor; i.e. TDFD only edits diagrams without control processes and event flows.
  - **TPSD: Tool for Process Structure Diagrams (JSD)**. This tool is intended for process structure diagrams (PSDs) that are used in the Jackson System Development method (JSD) [11]. The JSD method is discussed in [22]. This technique is explained in appendix A.3.4.
  - **TSND: Tool for System Network Diagrams (JSD)**. This tool is intended for system network diagrams (SNDs) that are used in the Jackson System Development (JSD) method [11]. The JSD method is discussed in [22]. The technique is explained in appendix A.3.5.
  - **TRPG: Tool for Recursive Process Graphs**. This tool is intended for recursive process graphs (RPGs), also known as life cycle diagrams. This technique is explained in appendix A.3.6.
  - **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 [22]. The technique is explained in appendix A.3.7.

All editors look very similar and have many operations in common. All diagram editors have almost the same set of edit commands and all table editors have almost the same set of edit commands. 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 provides a warning when it checks the drawing). As of version 2.10 TCM support hierarchic graphs, so that it can handle for example hierarchic statecharts. Features to be added later include constraint checking across documents and executable models.

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 [22] and the consistency rules of YSM are defined in [31]. 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 3.0 as is described in [6].

The same applies to object oriented (OO) methods, in particular those using UML. TCM currently contains a number of tools for drawing diagrams for OO models, albeit sometimes incomplete or in a bit different notation than the standard. However, in the future we want to build support for integrated OO modeling with UML into TCM as well.

In TCM there is and will be no consistency checking for other methods such as the ISAC [12] and Information Engineering [13] methods. But tools for TCM can be used to draw the diagrams.

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. 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 a mini tutorial is given to the notations and methods supported by the TCM tools. In appendix B, there is a list of frequently asked questions (FAQ). In appendix C there is a specification and explanation of the TCM file format.

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 a ftp site ftp://ftp.cs.utwente.nl/pub/tcm from which you can download distributions with binaries for different Unix platforms. A mirror of the TCM ftp site can be found at ftp://ftp.cs.vu.nl/pub/tcm. See the file `README` on the ftp site for more details.

Currently, TCM runs on Solaris 2.x (Sparc and x86), SunOS 4.x (Sparc), Linux (x86), OSF/1 (Digital Unix on DEC Alpha), IRIX 6.x (SGI), AIX 4.x (RS6000), HP-UX 10.x and even on Windows, running the CYGWIN/XFree86 environment. Also TCM has been reported to run under FreeBSD and Darwin (Mac OS X). There is also a distribution with the source code. Each distribution contains binaries of the TCM software and recent user documentation. The source code distribution contains no binaries but the source code, the user manual and technical documentation (developer's guide, source code documentation, specifications etc.). Solaris Sparc and Linux x.86 are the platforms on which we develop and for which the most recent binaries are available. The source code of TCM is

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/TESD	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/TESD	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/TESD	To make ER diagrams
	TTUT	To make transaction−use tables
	TTDT	To make transaction decomposition tables
	TEFD	To make data and event flow diagrams
	TSTD	To make state transition diagrams
Unified Modeling Language (UML)	TUCD	To specify use case diagrams
	TSSD	To specify static structure diagrams
	TATD	To specify activity diagrams
	TSCD	To specify statechart diagrams
	TSQD	To specify sequence diagrams
	TCBD	To specify collaboration diagrams
	TCPD	To specify component diagrams
	TDPD	To specify deployment diagrams

Figure 1.1: Using TCM in several software specification methods.

publically available, under the GNU public license. See the file COPYING in the TCM ftp distribution directory.

TCM has as home page http://www.cs.utwente.nl/~tcm. This page contains supplementary information about TCM. It is possible to download the TCM software via the page http://www.cs.utwente.nl/~tcm/software.html. You can find a HTML version of the recent version of the user manual in http://www.cs.utwente.nl/~tcm/usersguide/index.html. The user manual is also contained as PostScript and PDF file in the `doc` subdirectory of each software distribution. There you can also find the manual as a set of HTML files. 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 Installation and 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. Instead of the official Motif it is also possible to use the free Motif-clone Lesstif or Open Motif. We had some problems running TCM under Lesstif in the past; please try to use Open Motif if at all possible.

For Linux we have made some distributions in which the Motif library is already linked with the TCM executables. These statically linked distributions carry the name `statmotif` as part of their name. For using these distributions you don't need Motif, Open Motif or Lesstif at all.

For up-to-date system specific information see http://www.cs.utwente.nl/~tcm/software.html and read also the file `INSTALL` in one of the software distributions before you install and use TCM.

1.4.1 Unix tar-files with binaries

Most TCM distributions are in the form of a Unix `tar` file compressed with the `gzip` program [1]. To install a binary distribution (a tar.gz file) unzip and untar the TCM distribution by for instance: `tar xzvfp 'distribution'.tar.gz`. If your tar-program does not know the `z`(unzip) option then unzip the file explicitly by e.g. `gunzip`. A new directory named tcm-'version'/ will be created in the current directory containing the TCM binaries and documentation in subdirectories.

For an explanation of the TCM (environment) variables mentioned below see section 1.4.5. Every user of TCM should set the `TCM_HOME` environment variable to the directory where TCM is installed. It is recommended that they add `TCM_BIN` to their `PATH` and add `TCM_MAN` to their `MANPATH` environment variable. TCM will not work when `TCM_HOME` is not set. Do not forget to export these variables. It is a good idea to set the variables in your `.profile` or `.login` startup script of your Unix shell.

The distributions for Solaris contain shared object libraries. If you have one of these distributions and you have TCM installed in a directory other than `/opt/tcm`, each user should include this directory in the `LD_LIBRARY_PATH` variable by for instance: `LD_LIBRARY_PATH=TCM_LIB:$LD_LIBRARY_PATH`. Users of Linux do not need to set `LD_LIBRARY_PATH`.

To configure amongst others the path names of some external Unix programs that are used by TCM you can modify the `TCM_CONFIG/tcm.conf` file.

1.4.2 RPM distributions with binaries

For Linux we have also provided the software in the form of RPM packages. These distributions work with Redhat 6 and 7 and almost certainly with any other Linux system that has support for RPM. Install the distribution by:

[1] gzip (GNU zip) is a compression utility designed to be a replacement for "compress". It can be downloaded from ftp://ftp.gnu.org or from one of the many mirror sites that are mentioned inhttp://www.gnu.org/order/ftp.html.

```
rpm --install tcm-<version>.i386.rpm
```

Or upgrade an existing distribution by:

```
rpm --upgrade tcm-<version>.i386.rpm
```

By default TCM is installed in `/opt/tcm`. With the rpm option `--prefix <dir>` you can install it in a different directory. Some of the documentation will be installed by default in `/usr/doc/tcm-<version>`. There are two sorts of binary RPMs. One contains the name `statmotif` and the other `dynmotif`. The 'statmotif' RPMs have Motif statically linked into the executables. The 'dynmotif' RPMs are dynamically linked against Motif 1.2. It uses the libXm.so (version 1.2) of Motif, Open Motif or Lesstif on your system.

The RPM installation creates some initialization scripts for TCM in `/etc/profile.d`. It is therefore not necessary to set yourself environment variables like for the tar.gz distribution. You only have to login again before you start using TCM.

1.4.3 Source code distributions

This is not treated in this document. See the `INSTALL` file contained in the source distribution for how you can compile TCM quick and easy. The source code also contains an extensive developer's guide that explains all ins and outs of compiling and porting TCM on your system.

1.4.4 Starting up

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. `tesd`, `tefd`, `tssd`. Perhaps you want to end the command with an ampersand (like `terd&`) 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 1.4.5 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.

1.4.5 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:

- *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.

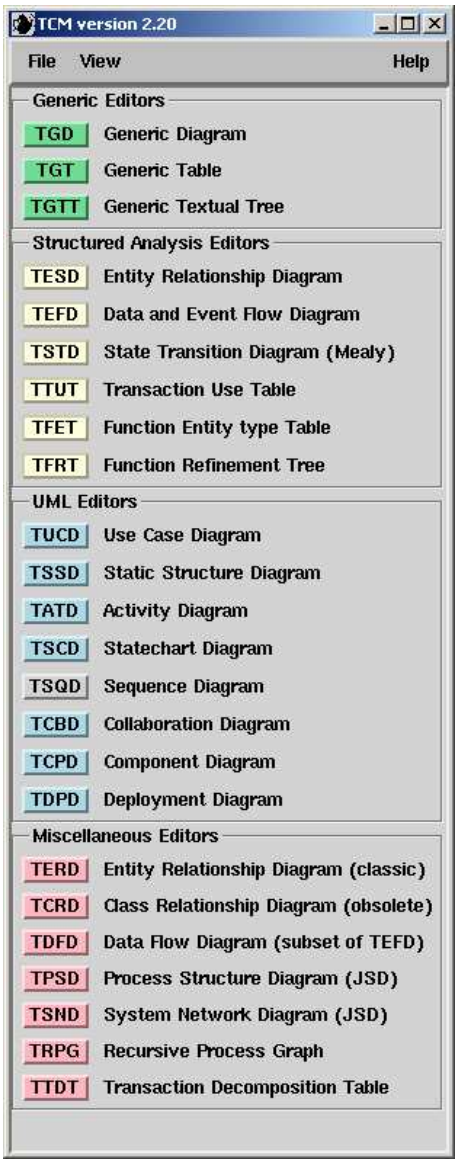


Figure 1.2: The TCM startup window.

- `-drawing` *Width*x*Height*. Specifies the size of the drawing area in pixels. The default size is 1330x1330 pixels [2]. Increasing the values makes it possible to draw larger documents, but it can slow down the editor significantly. Smaller values make it possible to run TCM faster on a machine with little memory. You can change the default size also in the configuration file `$TCM_HOME/lib/tcm.conf`.

- `-help`. Write all allowed command line arguments to standard output and exit.

- `-maxdrawing` *Width*x*Height*. 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. When there are not enough colors available in the default color map for the main window on startup, the editor is always started with a private color map, whether this option is given or not. It is possible to set in `$TCM_HOME/lib/tcm.conf` that you always want to start editors with a private colormap.

- `-projdir` *directory*. Set the project directory (working directory) to this directory. Files are loaded from and saved in this directory.

- `-version`. Write the TCM version to standard output and exit.

- In addition to these options, the standard X11 toolkit options such as `-bg` and `-display` are accepted as command line arguments (see the X window man page, X(1) or X11(7)).

All editors additionally have some optional command line arguments to generate printable output directly from an existing document file. When these options are given, no windows are created, but the X display is used however, so you should run X Windows with an opened display. These options are:

- `-toEPS [<outputfile>.eps]`. Generate Encapsulated PostScript from existing document and write it to the optional `<outputfile>.eps` or standard output and exit.

- `-toFig [<outputfile>.fig] [-latex]`. Generate Fig format from existing document and write it to the optional `<outputfile>.fig` or standard output, when no file name is given, and exit. When the `[-latex]` option is given, LaTeX fonts are generated, otherwise normal Postscript fonts are generated. For more information about Xfig and its file format see http://www.xfig.org/userman.

- `-toPNG [<outputfile>.png]`. Generate PNG graphics format from an existing document and write it to optional `<outputfile>.png` or standard output and exit. PNG is a format that is understood by most web browsers and by MS-Word. PNG is similar to the GIF-format but less controversial (see this article and http://burnallgifs.org). Generation of PNG format is done with the `fig2dev` command that processes a temporal fig file. fig2dev is part of the Transfig package that can be downloaded from ftp://www-epb.lbl.gov/xfig.

- `-toPS [<outputfile>.ps]`. Generate PostScript from existing document and write it to optional `<outputfile>.ps` or standard output and exit.

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

Table editors additionally have the following optional command line arguments:

- `-cell` *Width*`x`*Height*. Sets the default cell size of the table editor to *width* pixels wide and *height* pixels high.
- `-table` *Rows*`x`*Columns*. The table editor will be initialized with a table having *rows* rows and *columns* columns.

Depending on whether you find TCM precompiled into your (Linux) distribution (e.g. Debian and SuSE Linux) or whether you have to install it yourself TCM may use the environment variables listed below.

If no TCM specific environment variables have been set at all the values as found in the precompiled TCM distribution will be used.

- `$TCM_HOME`. The home directory where TCM and its components (subdirectories) are stored by default. As of version 2.10 any of these components can be placed in directories not directly under the `$TCM_HOME` directory.
- `TCM_BIN`. The location where the TCM binaries can be found. `TCM_BIN` will be respectively set to
  - `$TCM_BIN`. (if `$TCM_BIN` is defined)
  - `$TCM_HOME/bin/`. (if `$TCM_HOME` is defined)
  - `TCM_INSTALL_DIR`. (if neither `$TCM_BIN` and `$TCM_HOME` is defined)
- `TCM_LIB`. The location where the TCM library files can be found. Respectively set to `$TCM_LIB`, `$TCM_HOME/lib/` or `TCM_INSTALL_LIB`.
- `TCM_CONFIG`. The location where the TCM configuration files can be found. Respectively set to `$TCM_CONFIG`, `$TCM_HOME/lib/` or `CONFIG_INSTALL`.
- `$TCM_SHARE` (reserved for future use).
- `TCM_HELP`. The location where the TCM help files can be found. Respectively set to `$TCM_HELP`, `$TCM_HOME/lib/help/` or `HELP_DIR`.
- `TCM_DOC`. The location where the TCM documentation files can be found. Respectively set to `$TCM_DOC`, `$TCM_HOME/doc/` or `TCM_INSTALL_DOC`.
- `TCM_MAN`. The location where the TCM man files can be found. Respectively set to `$TCM_MAN`, `$TCM_HOME/man/` or `MAN_DIR`.

The TCM tools depend on the following files and programs:

- `TCM_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_BIN/psf` to make it call your local interpreter [3]. Psf is supplied with TCM and can be used stand alone as well, see `man psf`.
- `TCM_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`.

[3] When your system does not have Perl at all, this is not an major problem. Only the Print duplex and tumbled page options do not work.

- `TCM_CONFIG/tcm.conf`. This is the TCM configuration file. This file contains initialization values for the tools, like the default export format, the various print banner options, the default page size, the scale and grid settings etc. Each tool reads this file upon startup. The menu entries of the editor are initialized with the values from the configuration file. The settings in `tcm.conf` are already set to default values, but, when you have installed TCM, feel free to adjust them to your favorite settings. In the configuration file itself you find directions for how to modify it.

  Some settings in `tcm.conf` are commented out such as the name of the printer, the Unix commands to print a file, to view or remove a print request. The reason is that TCM normally determines these settings itself but in case the commands are not found or the wrong ones are chosen then they can be set explicitly in `tcm.conf`.

- `$HOME/.tcmrc`. Each user of TCM can override any setting in `tcm.conf` by his own configuration file, which has to be installed in `$HOME/.tcmrc`. The file syntax should be the same as of the TCM configuration file `tcm.conf`, i.e. of the form { *option value* }.

- `TCM_CONFIG/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 B.18 in the FAQ for how exactly you can set X resources.

- `TCM_CONFIG/banner.ps`. PostScript banner page that can be used when you explicitly want to print a banner page. To use this banner by default you have to edit `tcm.conf` or `.tcmrc`.

- `TCM_CONFIG/colorrgb.txt`. This is a text file with the names of the colors that can be used in TCM, together their associated RGB (red, green, blue) values.

- `TCM_HELP/`. This is a directory that contains a collection of plain 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 users.

The TCM editors depend on the following Unix environment variables.

- `LD_LIBRARY_PATH`. If TCM is installed in a directory other than `/opt/tcm` and the TCM distribution has shared object libraries [4], it is necessary to append `TCM_LIB` to this variable. Otherwise the shared libraries can't be found by TCM. In the FAQ in appendix B you will find more information about shared libraries and this variable. Also, if your X- or Motif libraries are in a non-default place then it can be necessary to set this variable.

- `MANPATH`. You are advised to add `TCM_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.

- `PATH`. You are advised to add `TCM_BIN` to your PATH.

- `PRINTER`. The name of the default printer for printing documents. This variable can be overridden by the `PrinterName` option in the TCM configuration files.

- `TCM_HOME`. The parent directory of the directories where the TCM binaries, libraries and manual pages are expected to reside if no additional TCM environment variables have been set.

  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.

[4] This is the case when the directory `TCM_LIB` contains files ending on `.so`.

As of version 2.20 TCM also runs on Windows systems under the CYGWIN/XFree86 environment. See the `README.cygwin` and the `INSTALL.cygwin` file in the software distribution before you install and use TCM under CYGWIN. In order to run TCM under CYGWIN all executables in the CYGWIN distribution will have the `.exe` extension. Also some scripts in `TCM_BIN` are added or adapted.

1.4.6 Graphical User Interface

The user interface of TCM complies for a large part with the OSF/Motif Style Guide [15]. The user interface is built from the OSF/Motif widget set [14]. 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. kde, gnome, fvwm, mwm, etc. In [16] 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 B.

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 the TCM mailing list tcm-users@cs.utwente.nl. You can subscribe by sending an empty message to: tcm-users-request@cs.utwente.nl. Mail that is only intended for us (the authors) can be sent to tcm@cs.utwente.nl.

Before you ask questions that were already asked and answered before: Appendix B 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.

TCM needs in principle a 3-button mouse. The left and middle buttons are the most essential for drawing nodes and edges and the right button is only used for a pop-up menu. However, pressing or dragging with the left mouse button while you are pressing the Shift key has the same effect as pressing or dragging the middle button. This means that you can use TCM as well with a 2-button mouse. Instead of button-2, you use Shift+button-1. Another solution is to change the function of the right and middle mouse button with: `xmodmap -e "pointer = 1 3 2"`. If you're using a so-called 5-button mouse (IntelliMouse with wheel-button), you can change the function of the right and middle mouse button with: `xmodmap -e "pointer = 1 3 2 4 5"`.

In this manual the mouse buttons 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). We almost never mention button-3 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.

2.1.1 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.

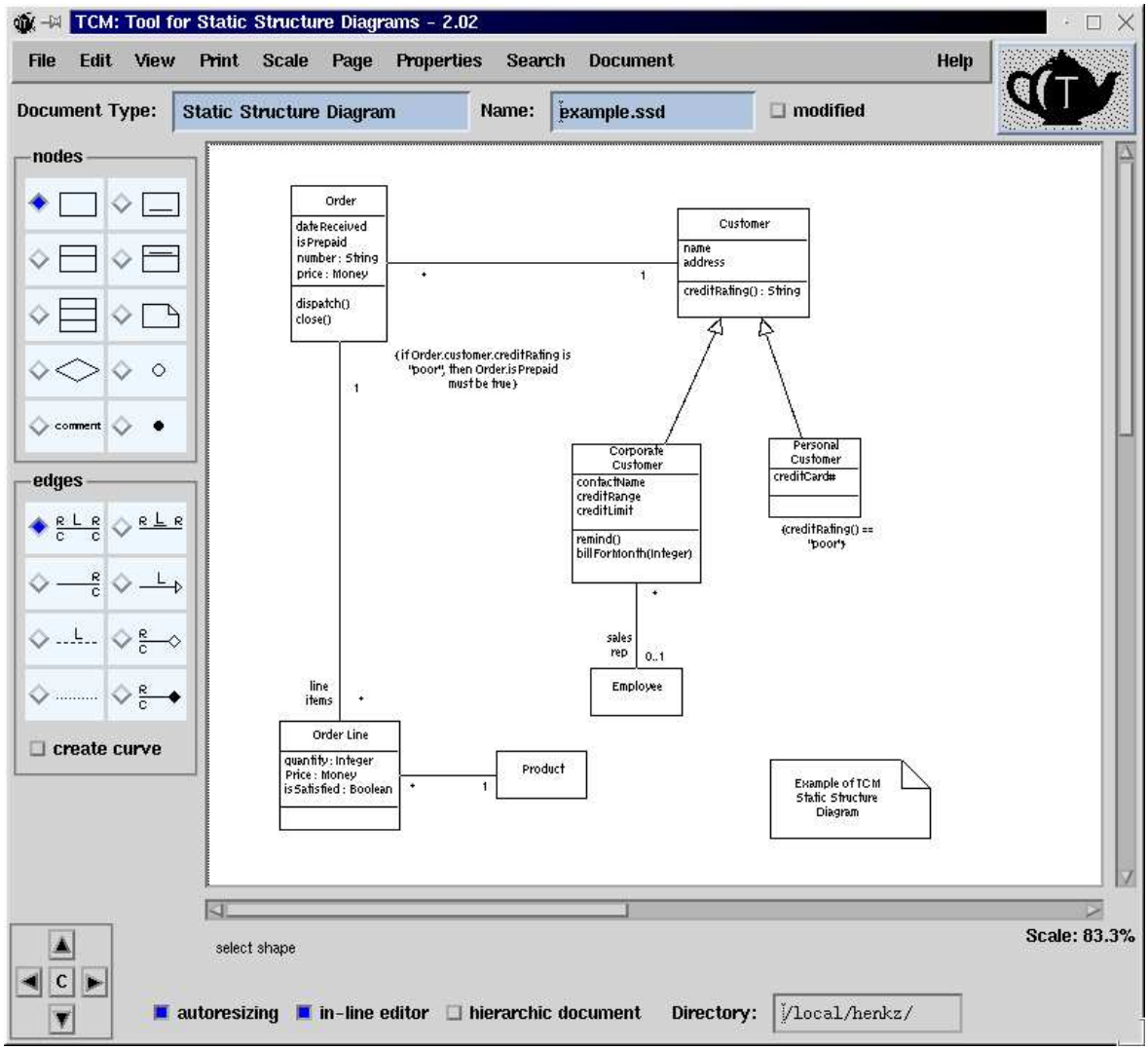


Figure 2.1: TCM main window.

2.1.2 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 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.

2.1.3 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
Click	Press and release the mouse button
Press	Press and hold 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 scaling the ratio between the TCM and X Window coordinates can be updated.

2.1.4 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.

2.1.5 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.

2.1.6 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.

2.1.7 Status area.

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

2.1.8 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.

2.1.9 Scale value.

The current scale percentage is shown in the bottom-right corner. By performing the scaling commands of the Scale menu, this value is updated.

2.1.10 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). When it is off, you should resize the shapes and cells manually to make them the right size.

2.1.11 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 window (see section 2.5).

2.1.12 Hierarchic document.

This is visible as a toggle beneath the status area, next to the In-line editor toggle. When it is on, the current document is hierarchic, i.e., nodes in this document can be hierarchically related. This toggle is only relevant for hierarchic editors (most diagram editors allow hierarchic documents).

2.1.13 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 set of pages that the drawing occupies.

Editor	Document Type	Name Suffix
TATD	Activity Diagram	.atd
TCBD	Collaboration Diagram	.cbd
TCPD	Component Diagram	.cpd
TCRD	Class–Relationship Diagram (obsolete)	.crd
TDFD	Data Flow Diagram (no eventflow)	.dfd
TDPD	Deployment Diagram	.dpd
TEFD	Data and Event Flow Diagram	.efd
TERD	Entity–Relationship Diagram (classic)	.erd
TESD	Entity–Relationship Diagram	.esd
TFET	Function–Entity type Table	.fet
TFRT	Function Refinement Tree	.frt
TGD	Generic Diagram	.gd
TGT	Generic Table	.gt
TGTT	Generic Textual Tree	.gtt
TPSD	Process Structure Diagram (JSD)	.psd
TRPG	Recursive Process Graph	.rpg
TSCD	Statechart Diagram	.scd
TSND	System Network Diagram (JSD)	.snd
TSQD	Sequence Diagram	.sqd
TSSD	Static Structure Diagram	.ssd
TSTD	State Transition Diagram (Mealy)	.std
TTDT	Transaction Decomposition Table	.tdt
TTUT	Transaction–Use Table	.tut
TUCD	Use Case Diagram	.ucd

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

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 with the **Project Directory** entry in the File menu of the TCM startup window (see figure 1.2). You can also 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

You can instantly start a TCM editor with a specific document via the **Open Document** entry in the File menu of the TCM startup window (see figure 1.2). The **Open Document** dialog is simular to the **Load Document** dialog described below. Selecting a TCM document will result in starting the TCM editor associated with the document extension. For example selecting a **.ucd** document will launch the **TUCD** (Use Case Diagram) Editor with the designated document loaded.

The File pull-down menu of the TCM editors 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 is deleted from the editor. If your current document is modified, TCM asks you if you want to save the document. 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 (<Ctrl+A>).** 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

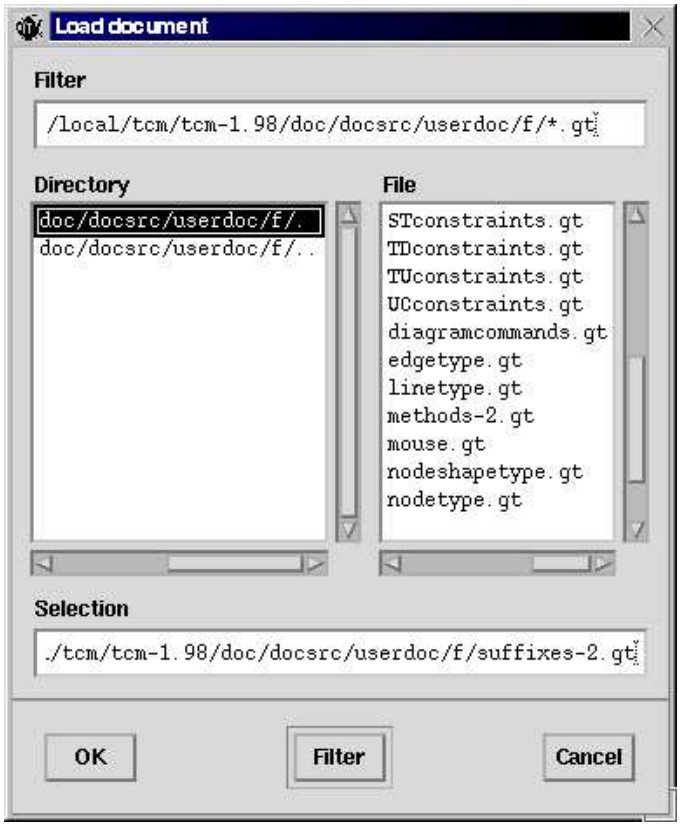


Figure 2.4: TCM File selection dialog.

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. Of course, you can also quit the editor by sending it a kill signal or by deleting the main window.

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 tree editors share the same set of edit commands (chapter 3). All table editors share the same set of edit commands too (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 , 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 as 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, edit mode is aborted, the text is updated and 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:

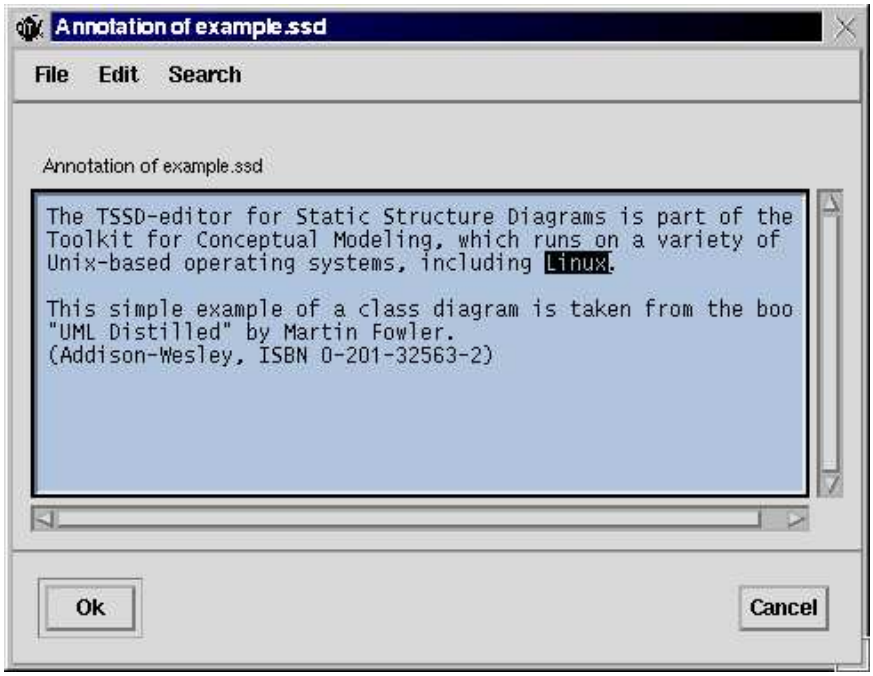


Figure 2.5: TCM text edit dialog.

- **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. 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.
- **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 a clipboard built-in which acts like a cut-paste buffer for copying and moving text between text areas of the same or different Motif applications.
- **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 Search 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`).

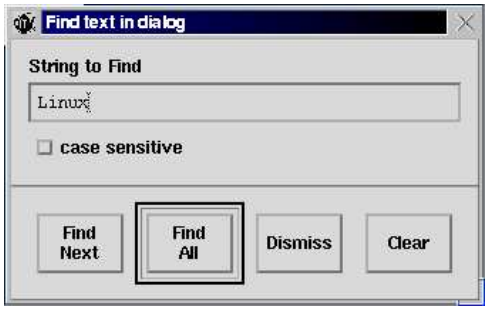


Figure 2.6: TCM find dialog.

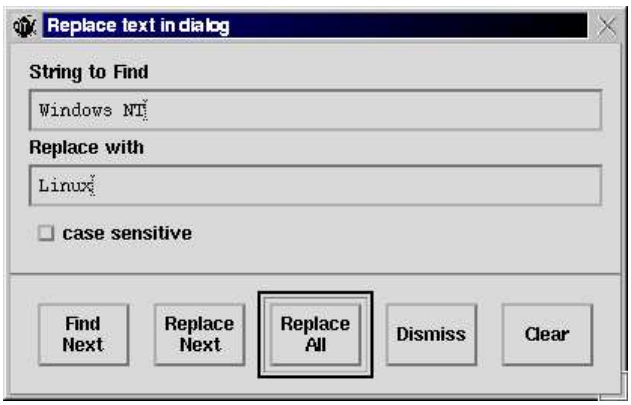


Figure 2.7: TCM replace dialog.

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. The Scale operations are in a separate menu but these are view commands too (as opposed to edit commands). The document editors have different View menu entries, but all editors share at least 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 **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.
  - **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.
- The **Autoresizing** 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 also (re)sets its counterpart toggle in the main window.

- 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 also (re)sets 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 2.9 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 or saved. The Print menu contains the following entries:

- **Print (<Ctrl+P>).** You can send the drawing directly to a PostScript printer with this command. 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.

  The default printer is the printer name in the TCM configuration file `$HOME/.tcmrc` or `TCM_CONFIG/tcm.conf`, whereby the options in the first configuration file is user-specific and overrides the options in the latter, system wide, configuration file. When this variable is not set, the printer will be set to the value of the `PRINTER` environment variable.

  In the Printer Properties 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 (PS; the same as what is sent directly to the printer), as Encapsulated PostScript (EPSF; a format that is more suitable than plain PostScript for including drawings into LaTeX or Troff documents), as Fig (either with PS fonts or with LaTeX fonts), or as Portable Network Graphics (PNG), which was created in response to the GIF licensing debacle and is optimized for graphics use on the Internet and other on-line services. PNG is even supported by MS-Word, so you can incorporate your TCM output in MS-Word too if you want to.

  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`, `.eps` or `.fig`. 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.

  The Fig format is the file format that is read and written by Xfig (http://www.xfig.org). Figures in Fig format can be converted into many other file formats using fig2dev from the TransFig package. Note, that we use fig2dev in TCM itself too for the generation of the PNG format.

- **Show Preview**. Starts up an external PostScript viewer, for previewing the document as it would be printed. Hopefully this command saves some trees. You can use your preferred external PostScript viewer program `ghostview`, `gv` or `pageview`. You can specify which viewer is used in the configuration file `$HOME/.tcmrc` (user specific) or in `TCM_CONFIG/tcm.conf` (system-wide). Also, you can set another PostScript viewer in the `Preview Command` dialog window, 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 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 files `TCM_CONFIG/tcm.conf` (system-wide) or `$HOME/.tcmrc` (user specific).

- **Printer Properties**. This is a menu that contains entries for some properties of the printer. The default values are read from the user-specific file `$HOME/.tcmrc` (if it exists) and from the system-wide files `TCM_CONFIG/tcm.conf` and `TCM_CONFIG/tcm.conf`. `.tcmrc` has a higher precedence than `tcm.conf`.

  In the Printer Properties 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 some 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 found in a TCM configuration file is used instead.
  - **Number of Copies**. Use this to change the number of copies that will be printed. 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 one of the TCM configuration files.
  - **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`. That is some Unix command like `"/usr/X11/bin/ghostview"`, but an other PostScript viewers such as pageview or xpsview should work as well.
  - **Print Colors**. Specifies whether colors are printed or saved in PostScript or not. When it is off, colors become gray-scales.
  - **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 banner page options** Here you can configure the various banner page options TCM supports.
    - **Print Default Banner Page**. When you mark this option, a default system banner page will be printed in front of the printed document. This can be useful when you share a network printer with many other users.

* **Print No Banner Page**. When you mark this option, no printer banner page will be printed in front of the printed document. This can be useful when you are the only user of this printer, or you just want to save some trees.
* **Print TCM 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 Scaler

The Scale menu contains the following entries:

- **Make Larger (<Alt+L>)**. Upscales the document. The drawing is made larger, up to about 1000The font sizes are scaled when your X server supports scalable X fonts.
- **Make Smaller (<Alt+S>)**. Downscales the document. The drawing is made smaller, down to about 10drawing. The font sizes are scaled when your X server supports scalable X fonts.
- **Normal Scale (<Alt+N>)**. Returns to normal view, i.e. no scaling.
- **Whole Drawing (<Alt+W>)**. Scales 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.
- **Scale Factor (<Alt+F>)**. Set the scale factor with a pop-up slider dialog. The default scale 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.

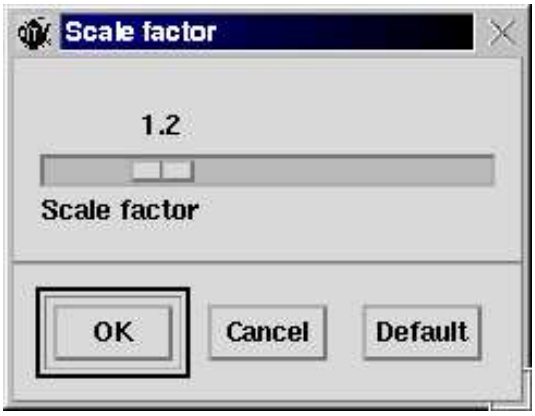


Figure 2.8: TCM slider dialog.

The current scale percentage is always visible in the bottom-right corner of the main window and is updated by the scaling commands. The **scale percentage** is the ratio between the TCM coordinates and the X Window coordinates, which is 100% when there is no scaling. By making the drawing larger you make this percentage larger, by making the drawing smaller, you make percentage smaller. The **scale factor** is the factor by which the scale percentage is increased or decreased during scaling. The scale percentage and factor are saved to file together with the document. When the drawing is saved as PostScript, EPSF, Fig or PNG-format the output is also scaled by the scale percentage.

2.9 The Page Layout

The Page menu commands determine the page layout. Any change in page layout is directly made visible in the drawing area. TCM is WYSIWYP, i.e. What You See Is What You Print. 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:

- **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 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.
- **Page Size**. Sets the page size. The current available sizes are A4 (default, 210x297 mm.), A3 (297x420 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, as a footer or both as a header and 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 page orientation, the page numbering and the inclusion of document info are saved in a document file. However, the page size and boundary are are options of the editor not of the edited document. So when you save a document the latter options are not saved.

2.10 The Properties Menu

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

- **Update Line Style**. This entry pops up a dialog window for choosing a line style. See figure 2.9. The line styles include solid, dashed, dotted, dual (i.e. double solid lines), invisible (nothing is drawn) and wide dotted. In the case of the table editors the dialog window shows an extra list of radio to indicate which sides of a cell should be updated. This command is only available in the generic diagram editor (TGD) and the table editors.
- **Update Line Width**. This entry pops up a dialog window to set the line width. See figure 2.10. The default line width is 1 pixel. When you issue a command to change the line width, each selected shape or cell is redrawn according to this new line width.

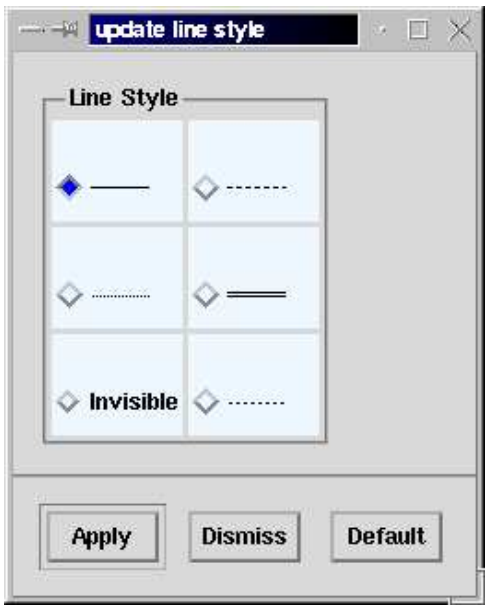


Figure 2.9: TCM line style chooser dialog.

- **Update Text Font**. This entry pops up a dialog window to update the font family, the font style and the point size (figure 2.11). For each of the three attributes there is a list of toggle buttons. Also, each list of toggle buttons in the dialog has an extra check button called `update` *attribute* that indicates whether that font particular attribute should be updated or not.

  Note that a shape in a diagram that has more than one text label cannot have different fonts for the different text labels.

  The font families include Helvetica (default), Times, Courier, New Century Schoolbook and Symbol (Greek). The font styles are Roman (default), Bold, Italic and Bold-Italic. The standard available point sizes are 8, 10 (default), 12, 14, 18 and 24. When you use Adobe fonts 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.

  In `TCM_CONFIG/tcm.conf` you can see which (types of) fonts are used for the TCM documents by default. If you wish some other font you can edit this file (or better, put the options in a personal file `$HOME/.tcmrc`). TCM uses scalable X fonts by default (you see that when you use the Scale commands). If these fonts are not installed or don't look good for your tastes then you can tell TCM to work with unscalable fonts by specifying:

  ```
  { ScalableFonts False }.
  ```

  If you wish fonts of a certain foundry, e.g. only Adobe fonts, then specify that with the following line in `TCM_CONFIG/tcm.conf` or `$HOME/.tcmrc` :

  ```
  { FontFoundry Adobe }
  ```

  If you wish some other font you can edit this file (or better, put the options in a personal file `$HOME/.tcmrc`). TCM uses scalable X fonts by default (you see that when you use the Scale

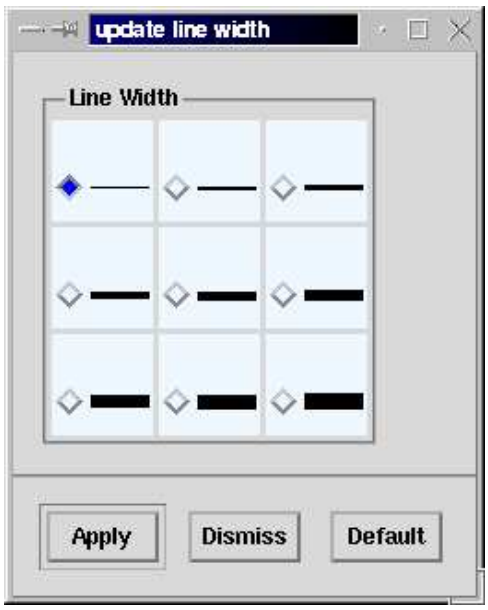


Figure 2.10: TCM line width dialog.

commands). If these fonts are not installed or don't look good for your tastes then you can tell TCM to work with unscalable fonts by specifying:

```
{ ScalableFonts False }.
```

If you want additional font sizes on top of the standard font sizes listed above you can put these in your personal config file `$HOME/.tcmrc` (preferred) or add these to the TCM config file `TCM_CONFIG/tcm.conf`. If you want e.g. to add Point Size 9 and 11 to the standard sizes, the following lines should be added:

```
{ AddPointSize 9 }.
{ AddPointSize 11 }.
```

- **Update Text Alignment**. This entry pops up a dialog window to update existing multi-line texts. See figure 2.12. Each selected shape receives this text alignment. In table editors there are distinct entries to update the column alignment and the row alignment. The text alignments include Left, Center (default) and Right.
- **Set/Unset Text Underlining**. This menu option sets/unsets (toggles) the text underlining of the selected shapes.
- **Update Line Color**. This entry shows a selection dialog box to update the line color (figure 2.13. The scrolled list contains the names of all available colors. The corresponding color can be previewed in a colored rectangle under that list. When you click `Apply` the line color of each selected shape or cell is updated.
- **Update Text Color**. This works similar to Update Line Color except that when you click `Apply` the colors of the texts are updated.

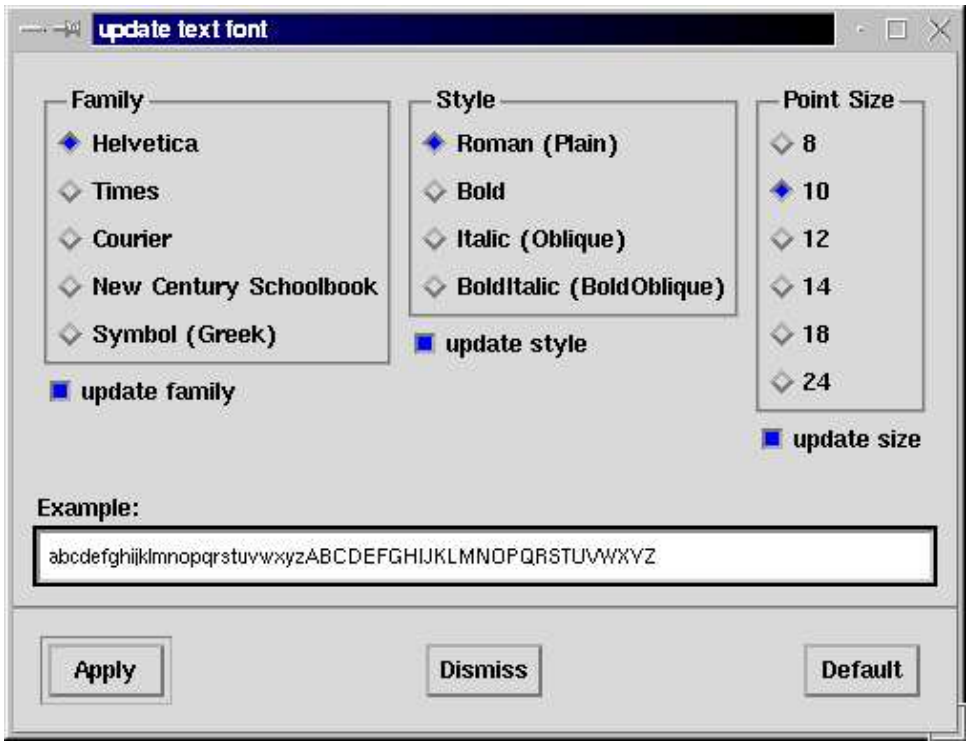


Figure 2.11: TCM font chooser dialog.

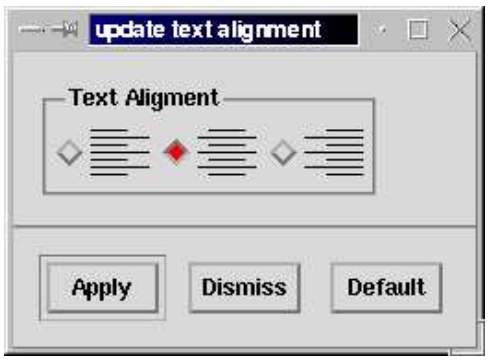


Figure 2.12: TCM text alignment dialog.

- **Update Fill Color**. The dialog window here shows an extra toggle button to indicate whether the shape or cell should be filled or not. The chosen color from the list will be the fill color when the shape or cell is filled.
- **Default Properties**.
  - **Default Line Width**. This works similar as Update Line Width. The default line width is the line width of every new shape or new lines in a table.
  - **Default Text Font**. This entry pops up a similar font chooser dialog window as the Update font entry. The texts of every newly created shape and the texts of new cells in a table, will get this default font. Page headers and footers, page numbers and table row and column labels are also drawn in the default font.
  - **Default Text Alignment**. This entry pops up a similar dialog window as Update Text Alignment 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.
  - **Default Line Color**. This works similar to Update Line Color. This is the color of every newly created shape or the new lines in table.
  - **Default Text Color**. This works similar to Update Text Color. This is the color of the text every newly created shape or the text in new cells in a table.
  - **Default Fill Color**. This works similar to Update Fill Color. This is the fill color of every newly created shape or the new lines in table. The dialog contain a toggle to indicate that shapes or cell are filled by default or not.
- **Node/Edge or Cell Annotation**.

  Pops up a text edit dialog for giving the selected node/edge or cell an annotation. This is free text and can for example contain the description or purpose of the node, edge or cell. See section 2.12 for advanced text edit operations on annotations.

2.11 The Search Menu

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

Figure 2.13: TCM color chooser dialog.

- **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 Search 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.

- **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 Search 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.

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. When TCM encounters a never file format version it will attempt to read it in, but success is not guaranteed. You better upgrade to a newer version then.

- 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.

- **Document Source**. Pops up a text view dialog that contains the source of the document file as being loaded last time from disk.

  All text editor options as described earlier are available here. Advanced users of TCM may hack the file directly if they want to, but do not expect intelligible feedback if you make a mistake. For making the changes take effect, save the contents of the dialog back to file and then (re)load the document via the File menu.

  Please take a look at appendix C for a detailed specification and explanation of the TCM file format.

- **Document Annotation**. 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.

- **Check Document Syntax**. 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 document type involved, 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 2.14.

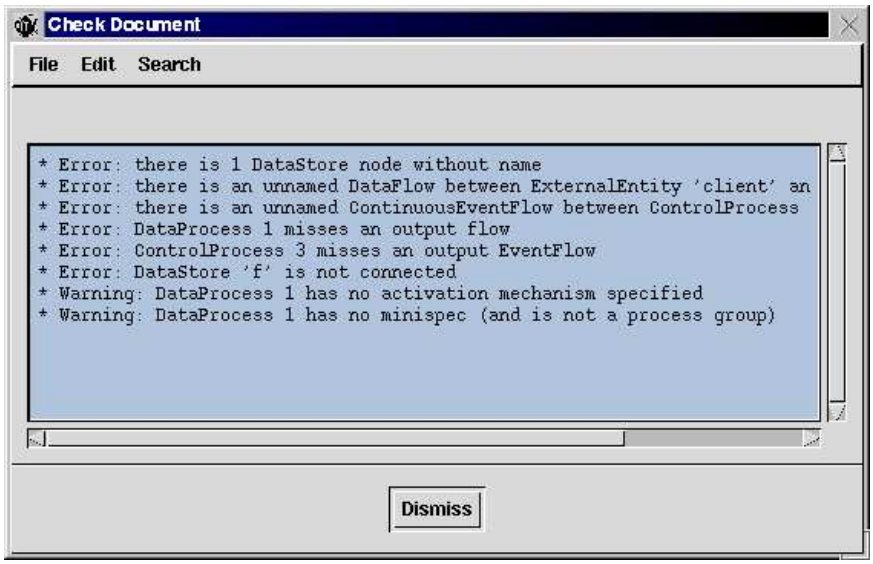


Figure 2.14: The result of check document on a DFD.

- The **Hierarchic Document** toggle. Turns Hierarchic Document on or off. Hierarchic Document means that nodes in this document can be hierarchically related. This toggle is only relevant for hierarchic editors (most diagram editors allow hierarchic documents). It also (re)sets its counterpart toggle in the main window.

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 clicks and movements you need to create and update the document that you want.
- **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, scaling, etc.
- **Version** gives the version and the authors of TCM.
- **Copying.** TCM 2.0 and higher is being released under the GNU GENERAL PUBLIC LICENSE. See http://www.gnu.org and question B.11 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 some TCM diagram editor are a special kind of **graph** with a certain **representation** (layout, view). Each TCM editor is a specialization of the generic graph editor. A graph consists of a set of **subjects**. A subject is either a **node** or an **edge** that connects two subjects. Subjects are of different **node types** and **edge types**. For example, a TDFD data flow diagram graph consists amongst others of nodes of type data process and nodes of type data store, and edges of type data flow.

The graphs that are edited with TCM contain edges that connect exactly two (not necessarily different) subjects. So some kind of special relationship between more than two nodes, for instance the specialization relationship in an ER diagram (made by the TESD editor), is specified in TCM as a node (a taxonomy junction node in this case) that is connected by three or more different edges (generalization or is-a relationship edges in this case).

In general, edges connect two (not necessarily) different subjects. So in principle it is possible that an edge connects to another edge. This feature is available in the generic diagram editor (TGD) and also in a limited form in a few of the specific diagram editors. In the other editors an edge connects only nodes, it never connects 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-diagram made by TSSD can contain different kinds of class boxes (e.g. boxes with three compartments that show the class name, attributes and operations, and boxes that only contain two compartment to show only the name and the attributes). Also, it is possible that a particular subject instance is represented by more than one shape instance. This feature can be used when you want to represent the same subject at different positions in the same diagram (duplication). But in most cases, the relationship between subjects and their representations is one–one and for this reason, subjects and their shape representations will often be mixed up.

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 has an empty graph after creation. You can build a graph by interactively drawing it with the TCM user interface, node by node, edge by edge.

Some diagrams are **hierarchic.** This means that the underlying mathematical structure is a

hierarchic graph rather than a graph. In a hierarchic graph, nodes are organized in trees: a node may be a child of another node. The parent–child relationship is represented by inclusion: a node shape contains the node shapes of its children. (In hierarchic documents, duplication is forbidden. This ensures that the parent–child relationship is acyclic.) In non-hierarchic diagrams, overlapping of nodes has no special meaning.

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 two examples of built-in constraints: 1. an edge connects exactly two (not necessarily different) existing subjects, 2. an entity type in TESD is always represented by a box or a double box.

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. Examples of immediately enforced constraints are: 1. an entity type and a relationship node in TESD cannot be connected by a Generalization 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. Two examples of soft constraints are: 1. all entity types have a non-empty name and 2. a relationship node in TESD 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 and the nodes and edges that cause these errors are selected in the diagram. 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`, at the left side of the main window, contains radio buttons indicating the current type of node that can be created. You can change the current node type selection by clicking another tiled 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 `create curve`. If the curve button is on, new edges that are created are 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 create curve toggle button 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. You can toggle between the curved and segmented edge representation by invoking the **Convert From/To Curves** command from the Edit menu.

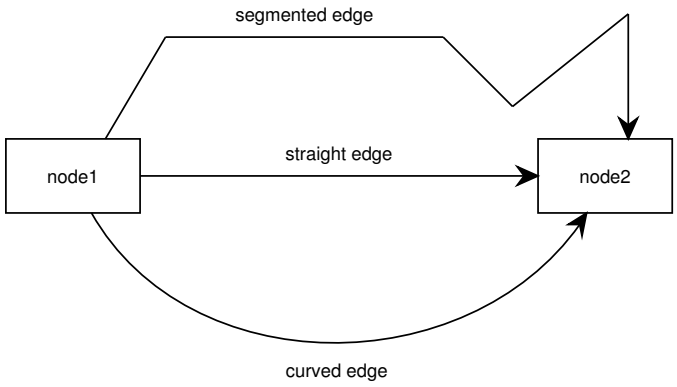


Figure 3.1: Three forms of edges.

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

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 the node like with straight edges. For understanding how it works it helps when you try it out yourself.

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 especially 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 with button-1 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 with button-1 the handle of an end point into *a different* node shape then the edge will be **redirected** to that node. This means that you can change the nodes that an edge connects.

After an edge is created you can add extra intermediate points. When you drag with button-1 on an existing line, a new intermediate point is created at the position where you stop dragging, and a new segment will be added to the line. Press Button-2 on a line handle to remove the line handle/segment from an existing line.

In TGD and some other editors it is possible to connect an edge with an edge like in figure 3.2. You create these kinds of edges in the same way as edges that connect nodes. Only you start to press button-2 while the mouse pointer is positioned at an edge and/or you release button-2, after dragging, when the mouse is on some edge.

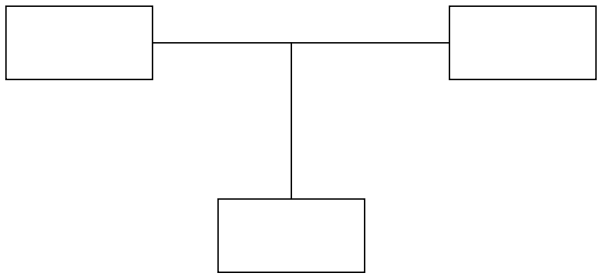


Figure 3.2: An edge connects another edge.

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.

Shapes have a **selection area.** For lines, it is a band around the segment(s) of the line. The selection area for node shapes changes when you toggle hierarchy on or off: In non-hierarchic documents,

it is just the inside of the node shape. In hierarchic documents, the selection area is a band around the visible parts of the node shape (boundary lines, interior lines, text labels).

The following operations on the selection exist:

- **Select a single shape**. Click button-1 in the selection area of an unselected shape (node shape, line). The selection will contain only this shape which will show black handles.
- **Add a shape to the selection**. Click button-2 in the selection area of 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 in the selection area of 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 in the selection area of an already selected (grey) shape. The selection handles of the old first selected shape will become grey and the clicked shape's handles 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  . By clicking button-1 on the shape or typing the first character you go into edit mode. 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.

Every 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 the generic diagram editor, TGD. But comment nodes can be selected, moved and of course edited, like any other node type.

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 no node is created.

3.6 Moving Shapes

You can reposition the following things in a diagram:

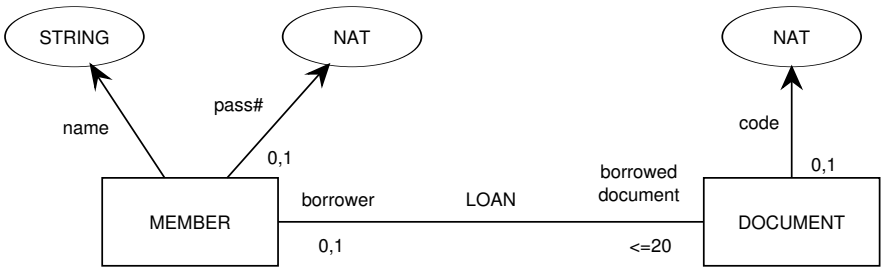


Figure 3.3: Example diagram with some text labels.

- **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 with button-1 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.

  When you drag a line outside a line handle with button-1, a new line handle will be created and the line is extended with an extra segment. When you hit a line handle with button-2 (instead of button-1), the line handle is deleted and consequently the line will be deprived of a segment.
- **End line handle**. When you drag and release one of the end point handles with button-1 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 [2].
- **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 nodes** command in the Edit menu has six variants as you can see in the Align Nodes submenu of the Edit menu. See figure 3.4 for the various alignments. 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.

[2] The question is if it is still the same edge when it connects to a different node...

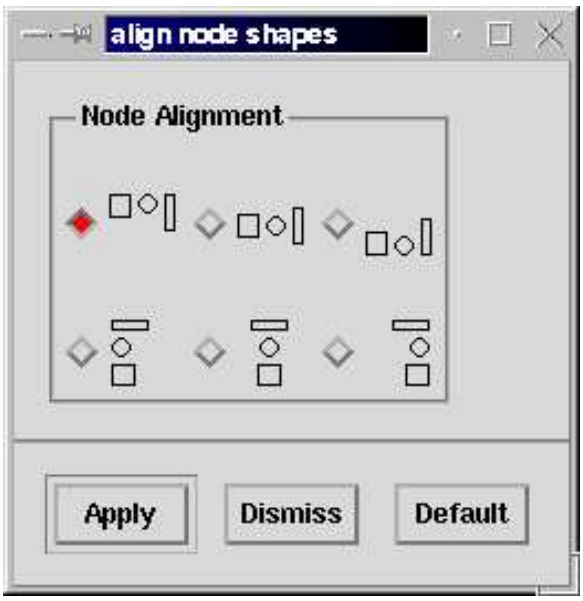
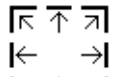


Figure 3.4: TCM Node Alignment Dialog.

3.7 Resizing Shapes

Resizing a node shape in all directions 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:  . You can use button-2 to resize a node shape in the desired direction only. 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 [3]. 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]But imagine the situation that labels do not move when nodes and edges are moved, that would be far more frustrating.

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 everything.

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

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 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 (but only 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 in non-hierarchic diagrams. The **Duplicate** command copies the node shapes in the selection and puts them into a box, lust like the Paste command. However, the duplicate command does not make use of the paste buffer. The copied node shapes can be positioned at an arbitrary place, just like the Paste command. Edges cannot be duplicated [4]. 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

[4] Duplicate lines are meaningful too in our opinion, so we want to add them to TCM in the future.

or data store is often drawn at different positions in the same diagram, and TCM keeps track that these are in reality representations of 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 Changing Shape Properties

All Update *shape property* commands are undo-able commands and they are called from the Properties menu of the main window.

- The **Update Line Style** command from the Properties menu pops up a dialog window with a number of toggle buttons for the different line styles: solid, dashed, dotted, dual, invisible and wide dotted (see figure 2.9). Select the line style that you want and press `Apply` and then the line styles of all the selected shapes are updated to this style.
- The **Update Line Width** command from the Properties menu shows a dialog window with a number of toggle buttons for the different line widths. The line width icons range from 1 pixel to 9 pixels wide, and are used to set the line widths of shapes. When you press `Apply` then of every selected shape the line width is set to the chosen line width.
- **Update Line Ends**. This entry pops up a dialog window to set the line ends of a line. See figure 3.5. You can change the 'begin of a line' as well as the 'end of a line'; for each line end a set of tiled toggle buttons is displayed, each containing a bitmap symbol for the corresponding line end. When you issue a command to change the line ends, each selected line is redrawn according to this new line ends. This command is currently only available in the generic diagram editor (TGD).
- The **Update Text Font** command from the Properties menu pops up a dialog window in which you can select a font (see figure 2.11). A font consists of the attributes font family, font style and point size. For each of the three attributes there is a list of toggle buttons. Also, each list of toggle buttons in the dialog has an extra check button called `update` *attribute* that indicates whether that font particular attribute should be updated or not. This makes it possible, for instance, to only change point sizes or font families of some texts but to keep the other font attributes the same.

  The dialog also shows a preview of some text in the selected font so you can see how it will look in your diagram. When you press the `Apply`-button then the dialog is dismissed and of each selected shape, the font is updated to the selected font.

  With the **Default Text Font** entry from the Default Properties submenu you get a similar dialog window. Here you can set the default text font. The text shapes of every newly created node or edge will get this font. The page headers and numbers are also drawn in this default font.
- The **Update Text Alignment** command from the Properties menu shows a dialog window with three toggle buttons, `left`, `center` and `right`. Choose the alignment and press `Apply`. Then the text alignments of all the text shapes in the selected shapes are updated to the selected alignment. Note that this alignment determines only the alignment of texts of multiple lines, not the positioning of the text shape itself in or near its parent shape.
- The **Set/Unset Text Underlining** command from the Properties menu sets/unsets (toggles) the text underlining of the selected shapes.

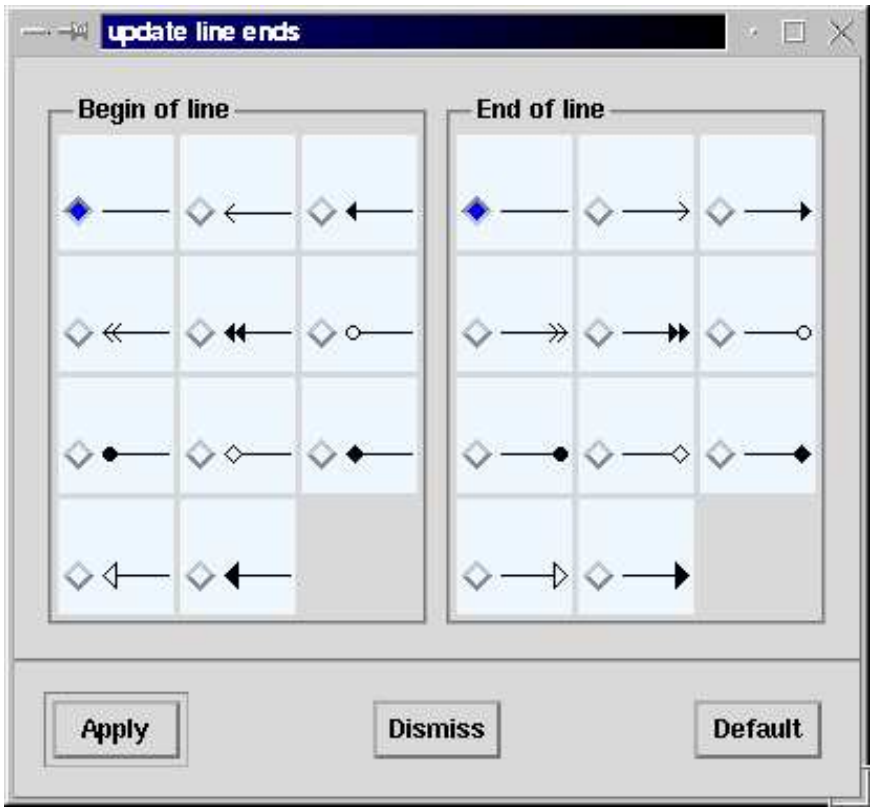


Figure 3.5: TCM line ends dialog.

- The **Update Line Color** command from the Properties menu pops up a color chooser dialog (see figure 2.13). The color chooser shows the names of all the available colors in a scrolled list. With clicking button-1 in the list you can select a color name. The associated color is made visible in a label below the scrolled list, named `preview`. When you press the `Apply` button then the line colors of all the selected shapes are updated to this color.

- The **Update Text Color** command from the Properties menu pops up a color chooser dialog. This works the same as the Update Line Color command except that after pressing `Apply` the colors of all text shapes of the selected shapes are updated to the chosen color.

- The **Update Fill Color** command from the Properties menu pops up a color chooser dialog. This works the same as the Update Line Color command except that after pressing `Apply` all selected shapes are filled with the chosen color.

3.12 Miscellaneous Edit Commands

- The **Node/Edge annotation** command from the properties menu 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 Search menu you call a dialog in which you can search for some text string 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 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 next 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 Search 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. The out-line text editor only works on a single text shape at the time.

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.

3.13 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 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 3.6 summarizes all atomic commands that are available in every diagram editor. These commands are either issued by the mouse, via the Edit or Properties menu, or via an accelerator. Each of these commands is undo- and redo-able.

3.14 The Generic Diagram Editor (TGD)

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

3.14.1 Nodes and Edges

The generic diagram editor has three node types **generic node**, **empty node** and **comment**, and one edge type, **generic edge**. Empty nodes are nodes that don't have a name label. TGD has no immediately enforced and no soft constraints. A generic or empty node can be represented by a number of node shape types and a generic edge can be represented by a number of line types, see figure 3.8.

Each node shape and each line can be drawn in the line styles solid, dashed or dotted. You can set the default line style of the node shapes via an option menu below the tiled node buttons in the main window. The default line style for lines can be set via the same kind of option menu below the tiled edge buttons. The line style of a node shape or line can be changed via the Update Line Style command in the properties menu. This command sets the line style of all selected shapes to the chosen line style.

All node shapes in TGD, except comments, bulls eyes, black dots and solid horizontal and vertical bars can have an index text label. If you want to see these text labels select the `create/edit index` toggle button below the node tiled buttons. By default the index labels are numbers between 1 and the total number of nodes having an index. Each new node shape will get the lowest currently unused index number higher than 0. When you deselect the `create/edit index` toggle, all node indexes remain visible, but new node shapes will not get an index number. When the `create/edit index` toggle is selected you can edit the node indexes just like the name of a node. You can create a new index label for a node shape by editing the empty index label of the node shape. The node index can be made any string in TGD. If you issue the Renumber Indexes command from the Edit menu of TGD then the current indexes of the diagram are renumbered. Each node shape that has an index will get as index a unique number between 1 and the total number of node shapes with an index.

In TGD it is possible to convert between different node shape types. For this, select the shapes that you want to convert to some other node shape type and choose the desired shape type from the submenu of `Convert Node Shape Type` in the Edit menu. This will pop up a dialog window

Command	User Action	Command	User Action
Add edge handle	Drag button-1 on an edge.	Make nodes same size	Select nodes and issue Same Size from the Edit menu.
Align nodes	Select nodes and issue an entry from Align submenu of Edit menu.	Paste from buffer	Issue Paste from the Edit menu. Release the paste box at the desired position.
Node/Edge annotation	Select a subject, issue Node/Edge annotation from the Properties menu. Fill in the annotation and click OK.	Resize node in all directions	Drag with button-1 on the handles of a selected node.
Append diagram	Issue Append from the File menu, choose a file, click OK and paste it.	Resize node in desired directions	Drag with button-2 on the handles of a selected node.
Copy to buffer	Select shapes and issue Copy from 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.
Create edge	Drag button-2 between nodes	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.
Create node	Click button-1 in the background	Select all	Issue Select All from the Edit menu.
Cut to buffer	Select shapes and issue Cut from the Edit menu.	Select area	Drag button-1 in the background.
Delete all	Issue Delete All from the Edit menu.	Set/Unset text underlining	Select shapes and issue Set/Unset Text Underlining from the Properties menu.
Delete edge handle	Click button-2 on an edge handle	Update fill color	Select shapes and issue Update Fill Color from the Properties menu. Choose a color from the list in the dialog and click Apply.
Delete subjects	Select one or more duplicate nodes and issue Delete from the Edit menu and click No in the dialog.	Update line color	Select shapes and issue Update Line Color from the Properties menu. Choose a color from the list in the dialog and click Apply.
Delete shapes	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.	Update line ends	Select shapes and issue Update LineEnds from the Properties menu. Choose a line end for the beginning and the end of the line in the dialog and click Apply.
Drag edge handle	Drag button-1 on an edge handle (edge will be redirected when an end handle is dragged into another node).	Update line style	Select shapes and issue Update Line Style from the Properties menu. Choose a line style in the dialog and click Apply.
Drag selection	Drag button-1 on a selected node.	Update line width	Select shapes and issue Update Line Width from the Properties menu. Choose a line width in the dialog and click Apply.
Drag shape	Drag button-1 on a single selected node.	Update text	Go into edit mode, edit the text and click OK in the text edit dialog or stop the in-line editor
Drag edge label	Drag button-1 on an edge label.	Update text alignment	Select shapes and issue Update Text Alignment from the Properties Menu. Choose the alignment from the dialog and click Apply.
Duplicate nodes	Select nodes and issue Duplicate from the Edit menu. Release the paste box at the desired position.	Update text font	Select shapes and issue Update Text Font from the Properties menu. Choose the desired font from the dialog and click Apply.
Find all texts	Isue Find in the Text menu. Fill in the text to find and click the Find All button.	Update text color	Select shapes and issue Update Text Color from the Properties menu. Choose a color from the list in the dialog and click Apply.
Find next texts	Isue Find in the Text menu. Fill in text to find and click the Find Next button.		

Figure 3.6: All atomic diagram edit commands.

containing toggle buttons for the various node shape types you can convert to. See figure 3.7 for the available node shape types. This makes it possible to change a box in an ellipse or vice versa. Note however that only the representation is changed. The node itself is not changed.

In TGD, every node, including comment nodes, can be connected by any type of edge and given the facts that you can make comment nodes invisible by emptying its text string or make the line style of a node shape invisible and that you can connect edges with edges too, you can make very complex drawings in TGD. The arrows and other kinds of line endings of the lines in TGD can be changed via the Update Line End command in the Properties menu.

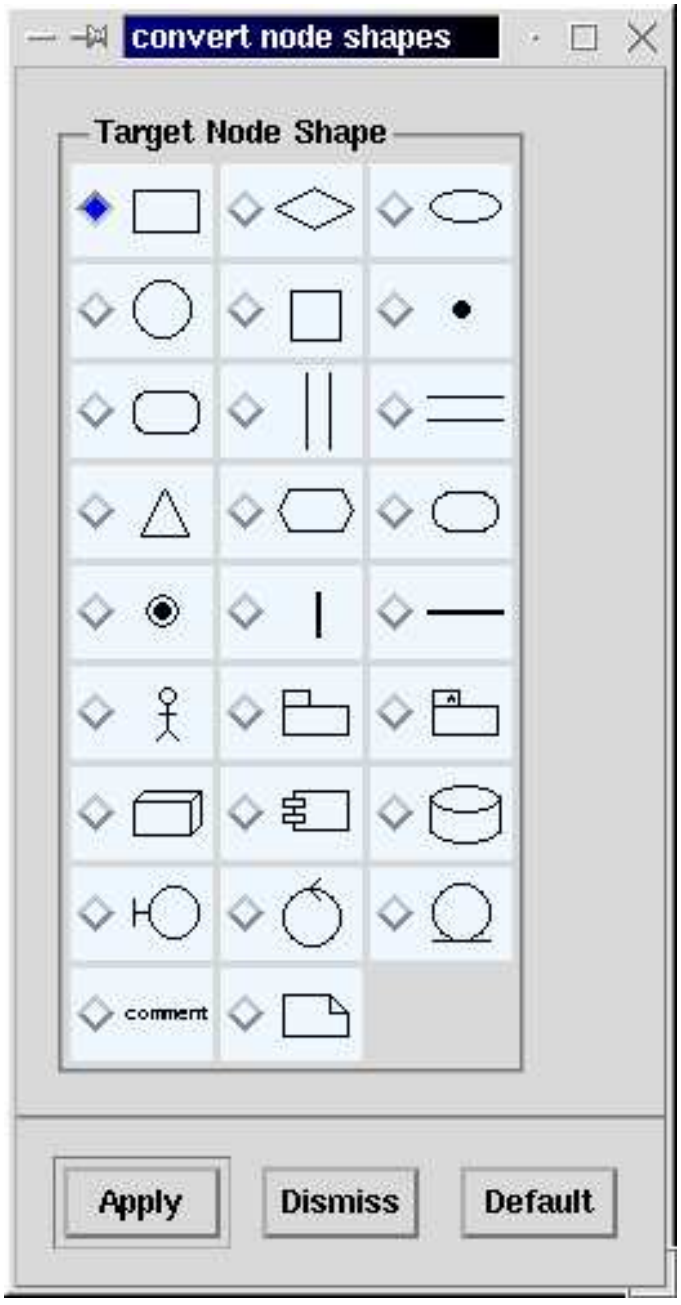


Figure 3.7: Possible node shape types to convert to.

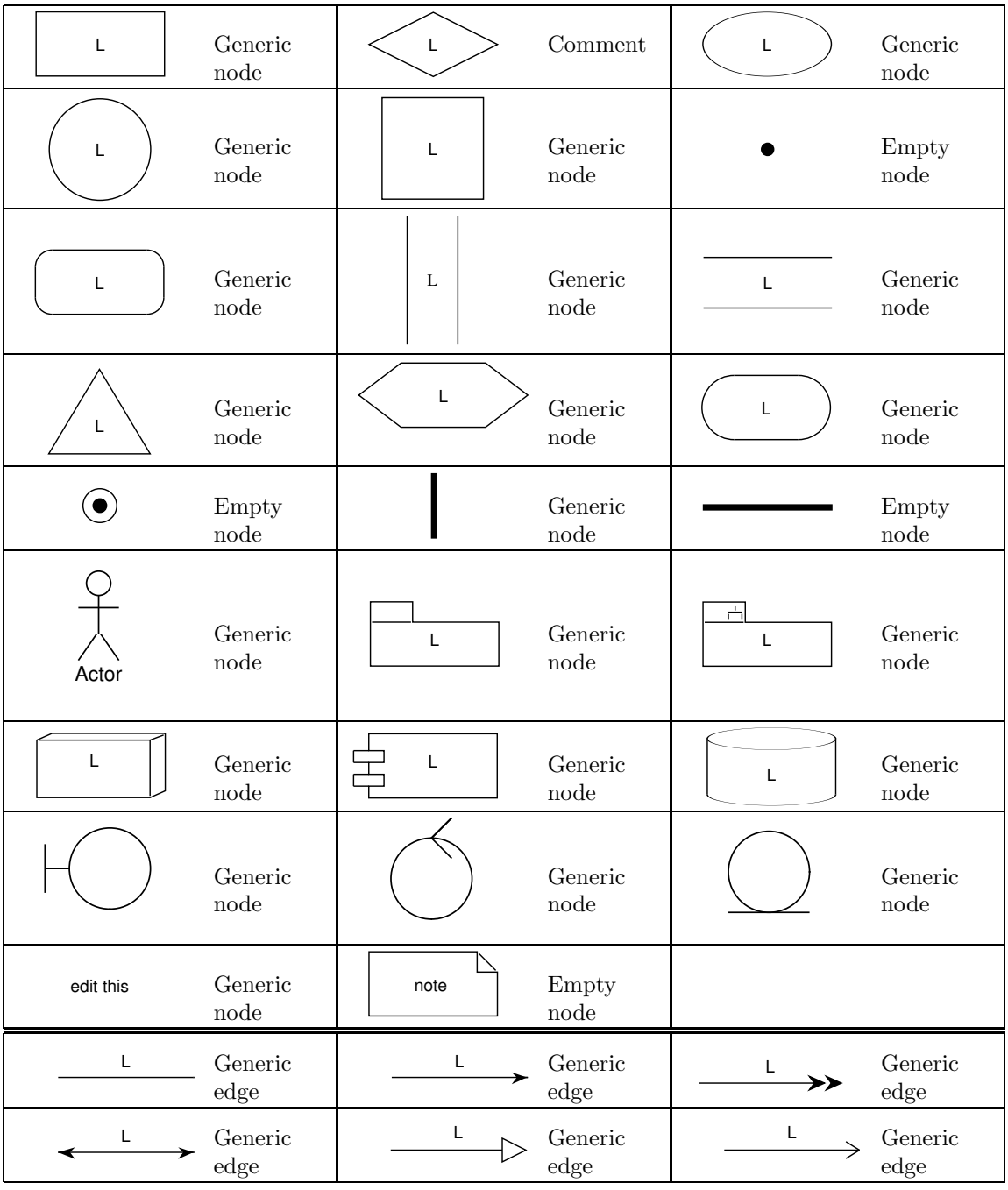


Figure 3.8: Generic diagram nodes and edges.

Chapter 4

Data View Editors

Data view editors manipulate diagrams commonly used for data modeling. This includes various entity-relationship as well as class diagram editors. They are described in a particular order in this chapter because this is the way TCM, and hence the user interface, is built up : TSSD specializes TCRD, which specializes TESD, which specializes TERD, the oldest of the lot. Each section may therefore refer to a previous section in this chapter.

4.1 The classic Entity-Relationship Diagram Editor (TERD)

This editor manipulates ER diagrams with the syntax defined in [22].

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.

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 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

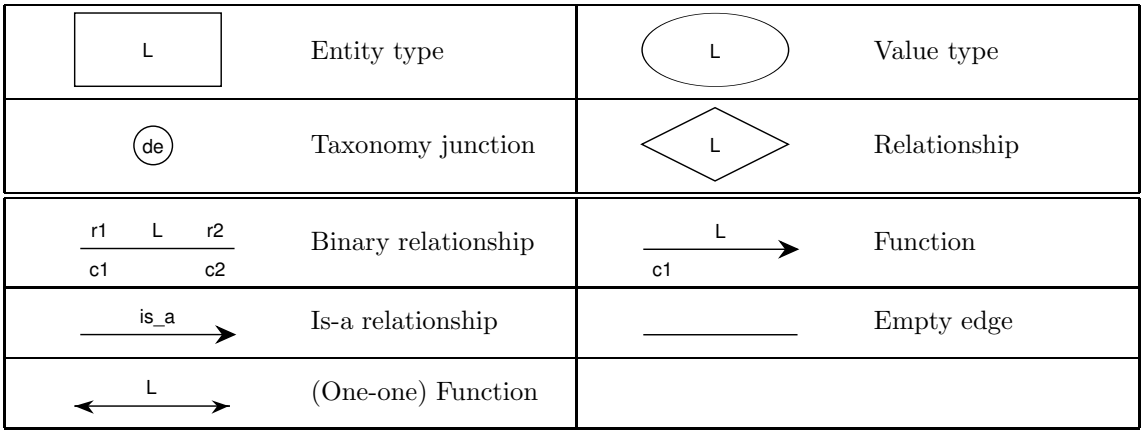


Figure 4.1: Entity-relationship diagram nodes and edges.

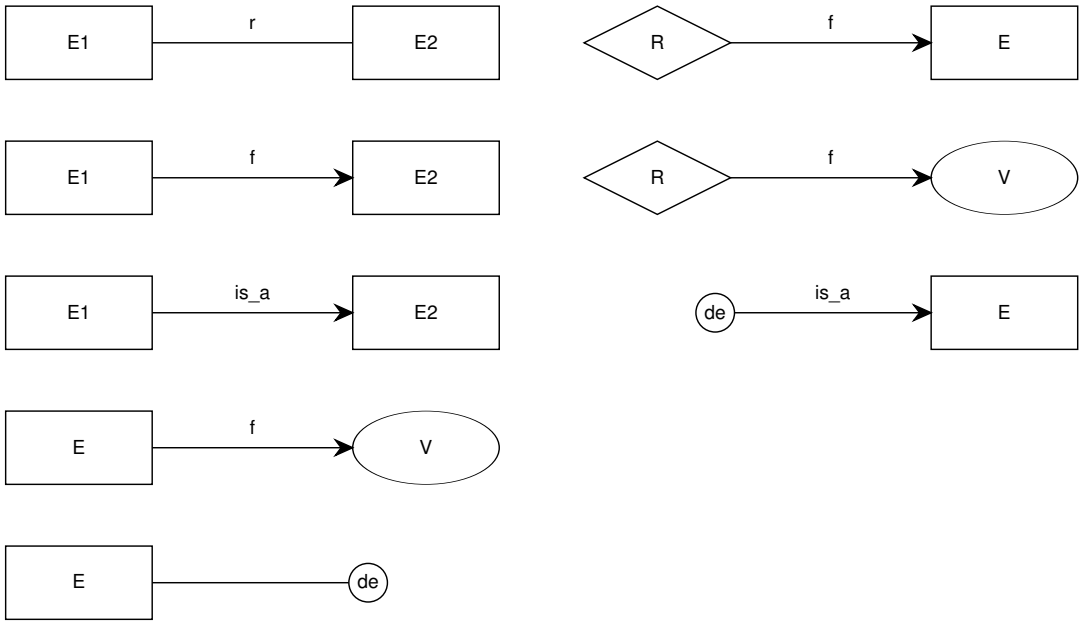


Figure 4.2: Permitted Entity-relationship connections.

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 3.3 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.

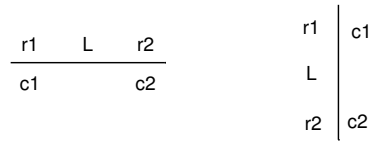


Figure 4.3: Default label positions of a binary relationship edge.

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`.

$$
\begin{array}{lcl}
\textit{constraint} & \rightarrow & \textit{constraint} \text{ , } \textit{expression} \\
 & | & \textit{expression} \\
 & & \\
\textit{expression} & \rightarrow & \textit{number} \text{ .. } \textit{number} \\
 & | & \textit{number} \\
 & | & < \textit{number} \\
 & | & <= \textit{number} \\
 & | & >= \textit{number} \\
 & | & > \textit{number} \\
 & & \\
\textit{number} & \rightarrow & \textit{digit}^{+} \\
 & & \\
\textit{digit} & \rightarrow & \mathbf{0}|\mathbf{1}|\mathbf{2}|\mathbf{3}|\mathbf{4}|\mathbf{5}|\mathbf{6}|\mathbf{7}|\mathbf{8}|\mathbf{9}
\end{array}
$$

Figure 4.4: Cardinality constraint syntax.

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 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

[1] Or more pedantic: 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.

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.

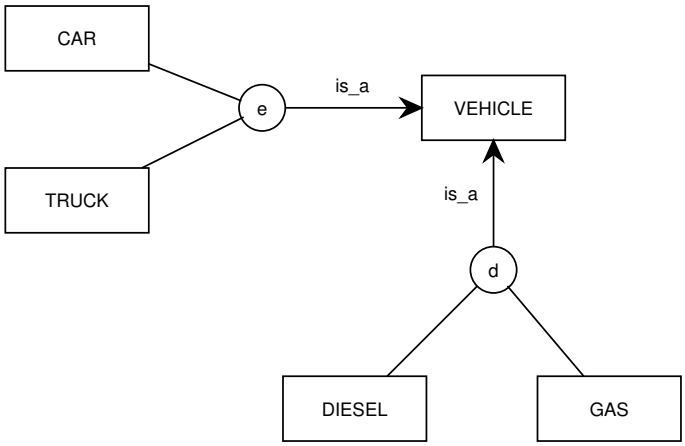


Figure 4.5: Example taxonomic structure.

The **Show ISA Hierarchy** 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 constraints do not concern duplicate node shapes. This is one of the reasons for the existence of duplicate shapes, i.e. showing the *same* node on *different* places in the diagram.

4.2 The Entity-Relationship Diagram Editor (TESD)

This editor manipulates ER diagrams with the syntax defined in [23]. These ER diagrams are a subset of UML Static Structure diagrams.

4.2.1 Nodes and Edges

See figure 4.7 for the subjects and the representing shapes that are used in TESD. In figure 4.8 you can see which node types can be connected by which edge types. The constraints of which node

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.

types can be connected by which edge types are immediately enforced. Note that entity types can be represented by two different box types. In figure 4.8 only the single box type variant is shown but each single box could be replaced by a double box.

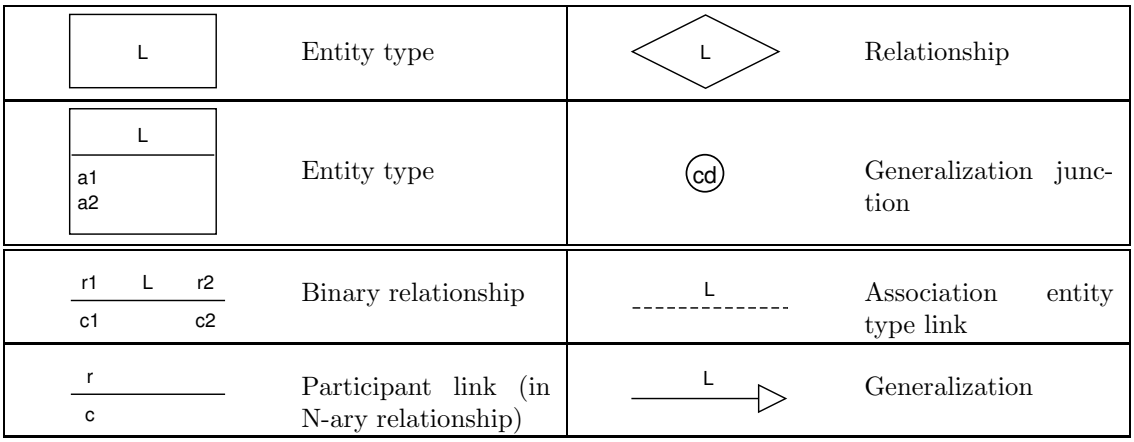


Figure 4.7: Entity-relationship diagram nodes and edges.

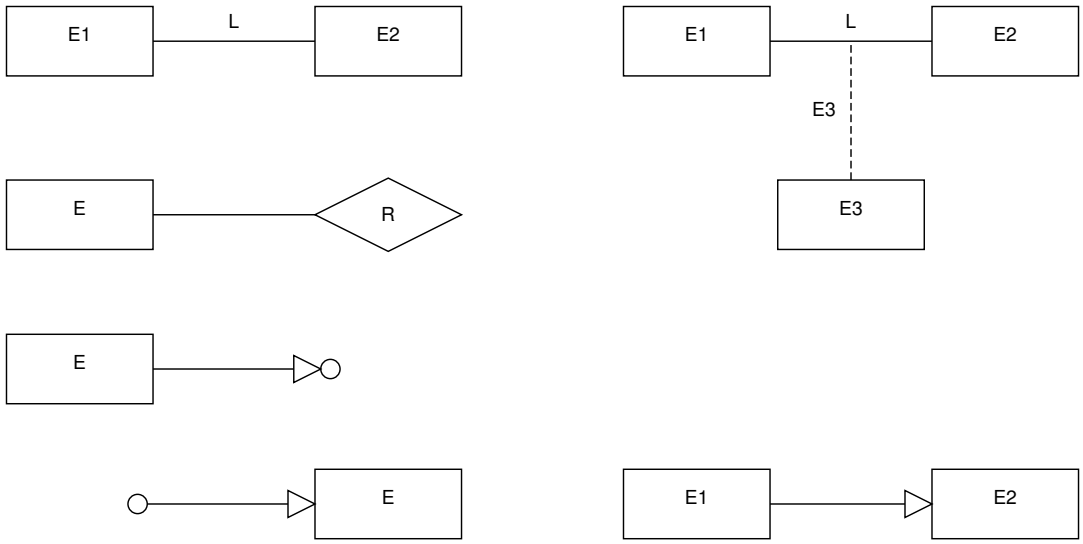


Figure 4.8: Permitted Entity-relationship connections.

4.2.2 Cardinality Constraints

TESD enforces immediately that cardinality constraints conform to the BNF syntax of figure 4.9. TESD does not check if subranges are contradictory such as the constraints `1..0` and `2,3..*`. Mind that the syntax differs from the syntax in TERD (figure 4.4).

$$
\begin{array}{lcl}
\textit{constraint} & \rightarrow & \textit{constraint} \text{ , } \textit{expression} \\
 & | & \textit{expression} \\
 & & \\
\textit{expression} & \rightarrow & \textit{number} \text{ .. } \textit{number} \\
 & | & \textit{number} \\
 & | & \textit{number} \text{ .. } \textit{star} \\
 & | & \textit{star} \\
 & & \\
\textit{number} & \rightarrow & \textit{digit}^{+} \\
 & & \\
\textit{digit} & \rightarrow & \mathbf{0}|\mathbf{1}|\mathbf{2}|\mathbf{3}|\mathbf{4}|\mathbf{5}|\mathbf{6}|\mathbf{7}|\mathbf{8}|\mathbf{9} \\
 & & \\
\textit{star} & \rightarrow & *
\end{array}
$$

Figure 4.9: ESD Cardinality constraint syntax.

4.2.3 Read direction arrows

Binary relationships in TESD (and TSSD) can optionally have a read direction arrow connected to the name label. This arrow is a clue for the user for how to read the label. It has no semantics. Via the submenu called “Change Read Direction” in the Edit menu a read direction arrow can be added or removed. The option “None” removes the arrow from all selected binary relationships. The option “From Shape” shows an arrow directed to the “from”-shape, i.e.‘ the shape from which the relationship line originates when it was drawn. The option “To Shape” shows an arrow directed to the “to”-shape. Or simply just click on the ’read direction area’ behind the name label to switch between the three options ’To Shape’, ’From Shape’ and ’None’.

4.2.4 Taxonomic Structures

The is-a relationship in TERD is called generalization in TESD, being represented by a open arrow.

A **taxonomic relationship** consists of a taxonomy or generalization node connected to three or more entity types by generalization edges. On connecting a generalization edge from an entity type to a generalization node, the open arrow end will disappear, resulting in an empty edge representation. See, for example, figure 4.10.

The **Show ISA Hierarchy** toggle in the View menu of TESD 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 can have a label name, with the exception of a generalization arrow leading towards a generalization node which can not have a label. Generalization nodes should have one of the following labels : ””, ”d”, ”c”, ”dc” or ”cd”. A ”d” stands for disjunctive and a ”c” stands for complementary. These constraints are immediately enforced.

4.2.5 Constraint Checking

TESD checks also the soft and immediately enforced constraints that are summarized in figure 4.11.

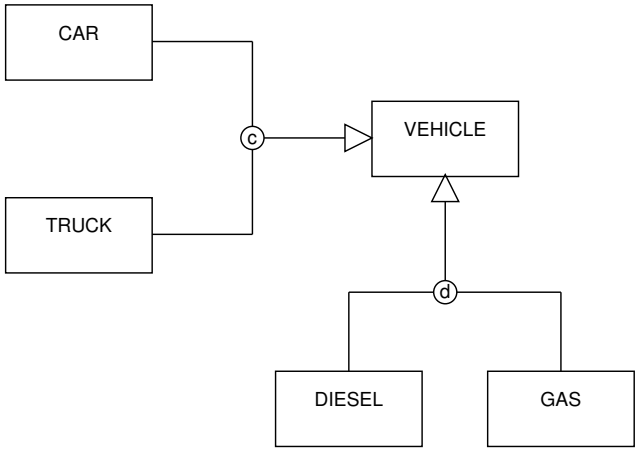


Figure 4.10: ESD Example taxonomic structure.

Constraint	**Check**	**Description**
ES1	Immediately	Names of entity types, relationships, associations and links contain at least one letter.
ES2	Immediately	Generalization relationships should not contain cycles.
ES3	Soft	Associations and links are properly connected: they have at least two connections to a class, relationship or n–ary association..
ES4	Soft	Binary associations (edge) should either have a non–empty name or at least one non–empty role name.
ES5	Soft	Association nodes and link nodes should be named.
ES6	Immediately	Edges from the same association node to the same subject should have different names.
ES7	Immediately	Each attribute definition contains at least one letter and no newlines.
ES8	Immediately	The attribute definitions within the same entitytype are mutually exclusive.
ES9	Immediately	Taxonomy junctions have one of the labels : "", "d", "c", "dc" or "cd".
ES10	Immediately/Soft	Taxonomy junctions are properly connected (one generalization edge departing from the junction and two or more generalization edges arriving at the junction); all edges connecting different subjects.

Figure 4.11: Immediately checked and soft constraints on ERDs.

4.3 The Class-Relationship Diagram Editor (TCRD)

4.3.1 Nodes and Edges

See figure 4.12 for the subjects and the representing shapes that are used in TCRD. In figure 4.13 you can see which node types can be connected by which edge types. These are immediately enforced constraints. Note that classes can be represented by three different box types. In figure 4.13 only the double box type variant is shown but each double box could be replaced by a single or triple box.

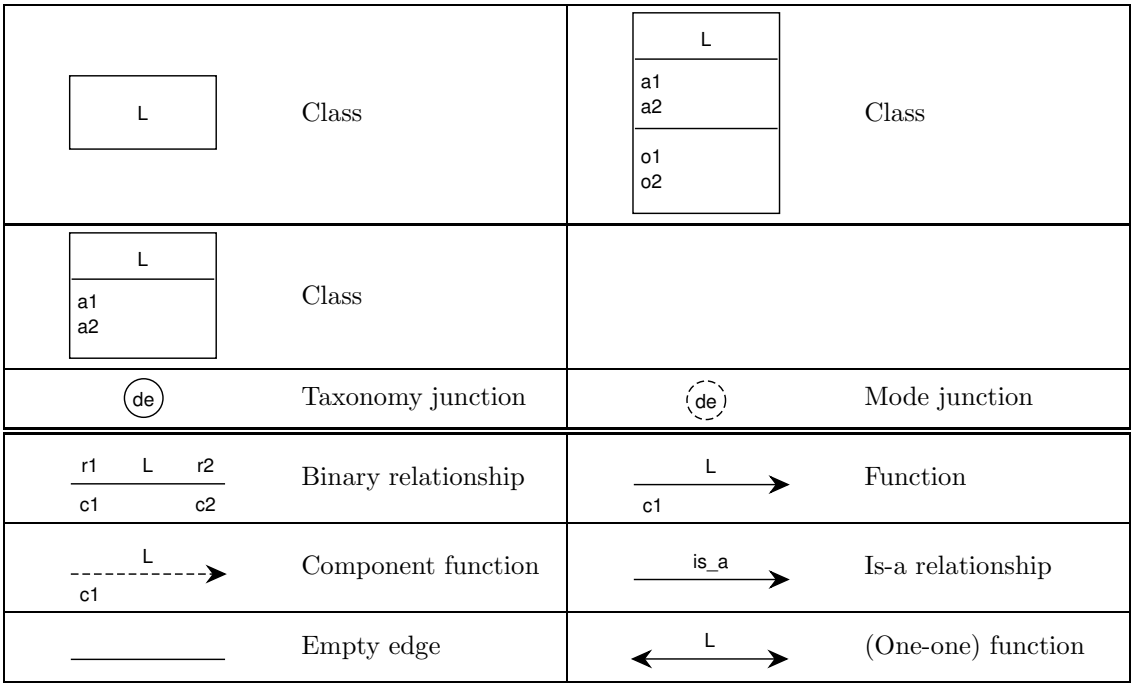


Figure 4.12: Class-relationship diagram nodes and edges.

4.3.2 Classes and Relationships

The 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 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. The only difference between a real 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 classes 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 4.14). There is no other visual difference between a class instance and a relationship instance in TCRD than the component functions. So, in TCRD a link can have attributes, operations and it can be part of a taxonomic structure.

TCRD inherits all the applicable constraints of TERD, on the understanding that classes in TCRD replace the entity types from TERD. Furthermore, see section 4.1.2 for the use of cardinality con-

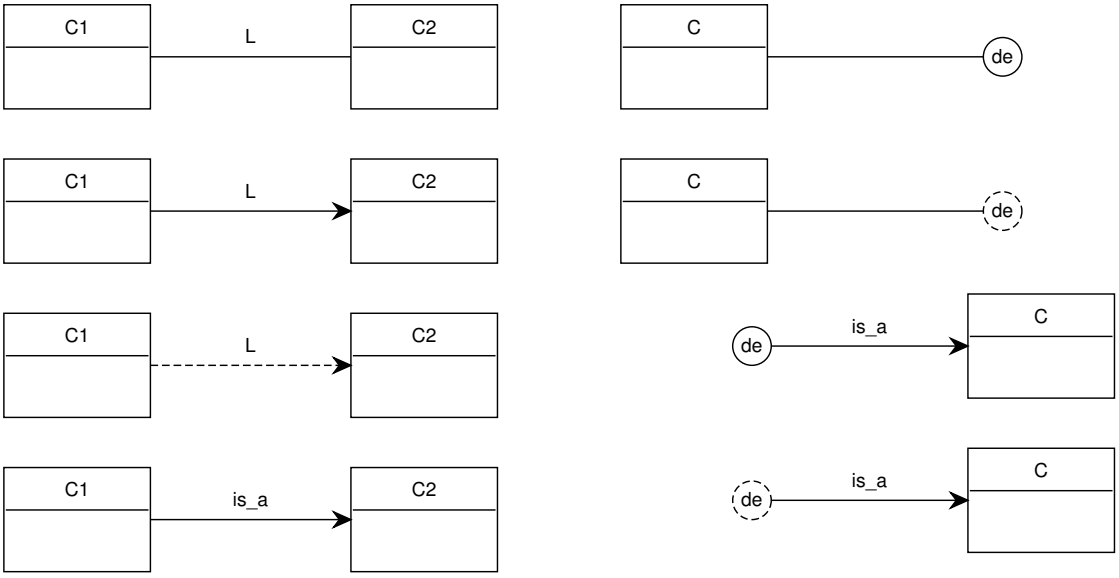


Figure 4.13: Permitted Class-relationship connections.

Figure 4.14: Example CR diagram, showing a relationship class.

straints 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.

4.3.3 Attributes and Operations

The class-relationship editor TCRD offers classes that can have a number of **attributes** and a number of **operations** [2]. See figure 4.15 for an example.

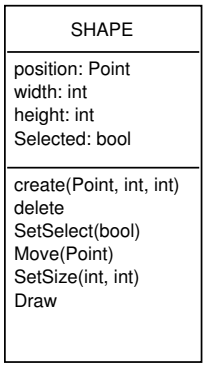


Figure 4.15: Example class with attribute and operation definitions.

A 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 operations 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, Class. You can change the representation of an existing 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 operations are preserved.

Each attribute definition and each operation 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 operation definition. Attribute names are in the second compartment and operation names are in the third compartment of the class box. You can perform the following operations on attributes and operations:

- **Append one or more attributes or operations**. To add attributes or operations to the end of the attributes or operations list of a 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 operation at a time by separating them by newlines. When you stop editing and the autoresize is on, the size of the class box is adjusted to the width and the height of the new list of attributes and operations and the new attributes and operations in the box are always left adjusted.

- **Insert one or more attributes or operations**. To insert new attributes or operations somewhere in the current attribute or operation 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 operations like described above.

[2] In the past, an operation in TCRD was also called **event**, **transition** or **action**.

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

- **Remove an attribute or operation**. To remove an existing attribute or operation, 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 operation is deleted and the size of the class box is adjusted.

4.3.4 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 4.16. The concept of mode specialization as a form of type migration is explained in appendix A. 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.

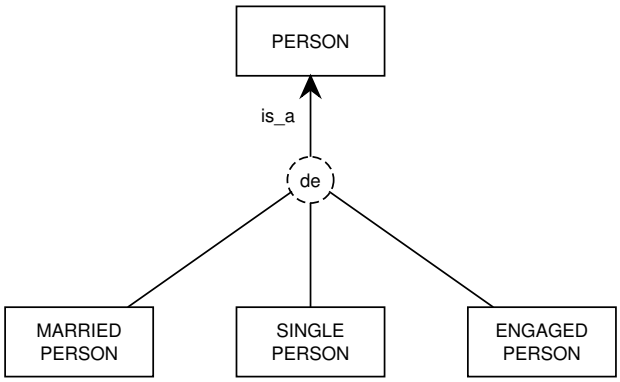


Figure 4.16: Example mode specialization.

An is-a relationship (without a junction) between two 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.

4.3.5 Constraint Checking

TCRD checks also the soft and immediately enforced constraints that are summarized in figure 4.17. Note that TCRD checks a lot of constraints but it does not check yet all plausible constraints on CRDs. For instance, name conflicts between operations and attributes of classes that are is-a related are not yet discovered and the syntax of attributes and operations 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 4.17: Immediately checked and soft constraints on CRDs.

4.4 The Static Structure Diagram Editor (TSSD)

4.4.1 Nodes and Edges

See figure 4.18 for the subjects and the representing shapes that are used in TSSD. In figure 4.19 you can see which node types can be connected by which edge types. These are immediately enforced constraints. Note that classes can be represented by three different box types, whereas objects can be represented by two different box types. In figure 4.19 for classes only the double box type variant is shown but each double box could be replaced by a single or triple box. For objects only the single box variant is shown but each single box could be replaced by a double box.

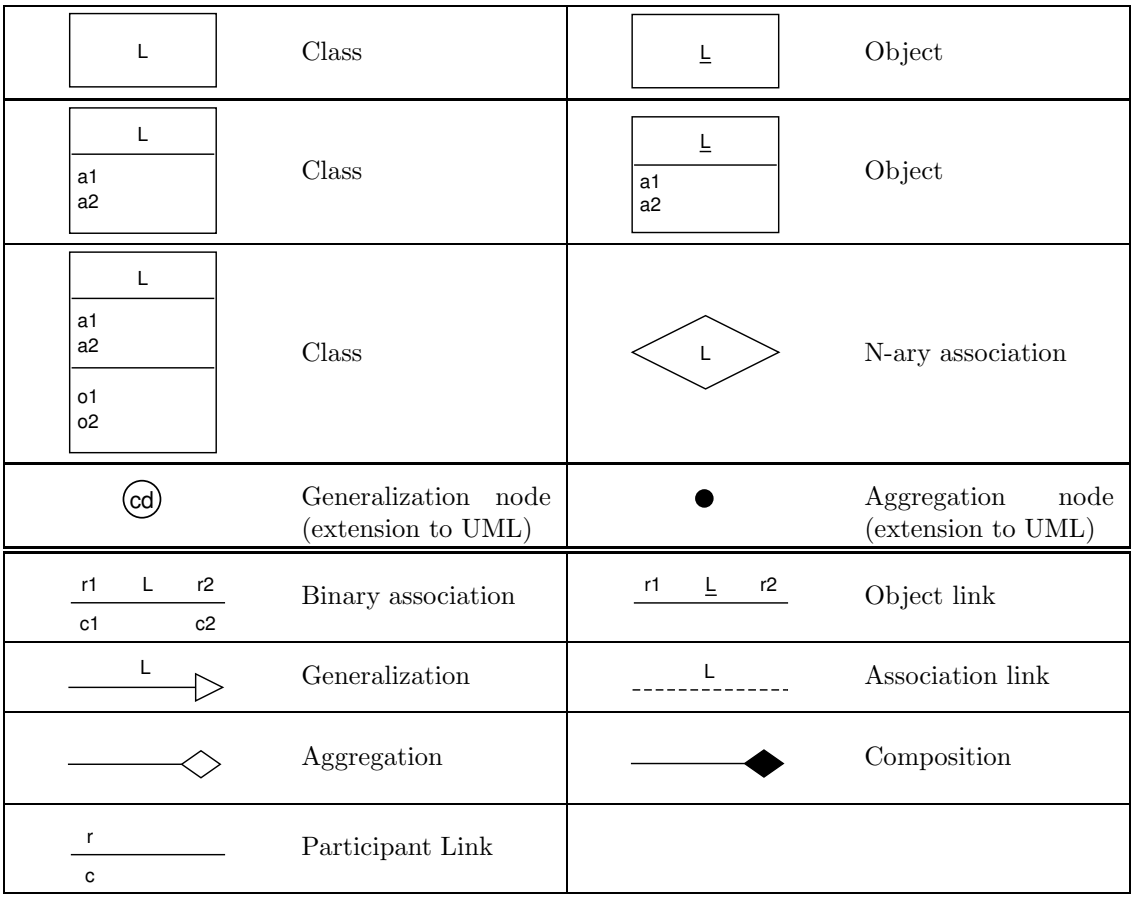


Figure 4.18: Static structure diagram nodes and edges.

4.4.2 N-ary Associations

TSSD inherits all the applicable constraints of TESD. TSSD also supports N-ary associations, being represented by a diamond node shape, connecting the associated classes or objects. See figure 4.20.

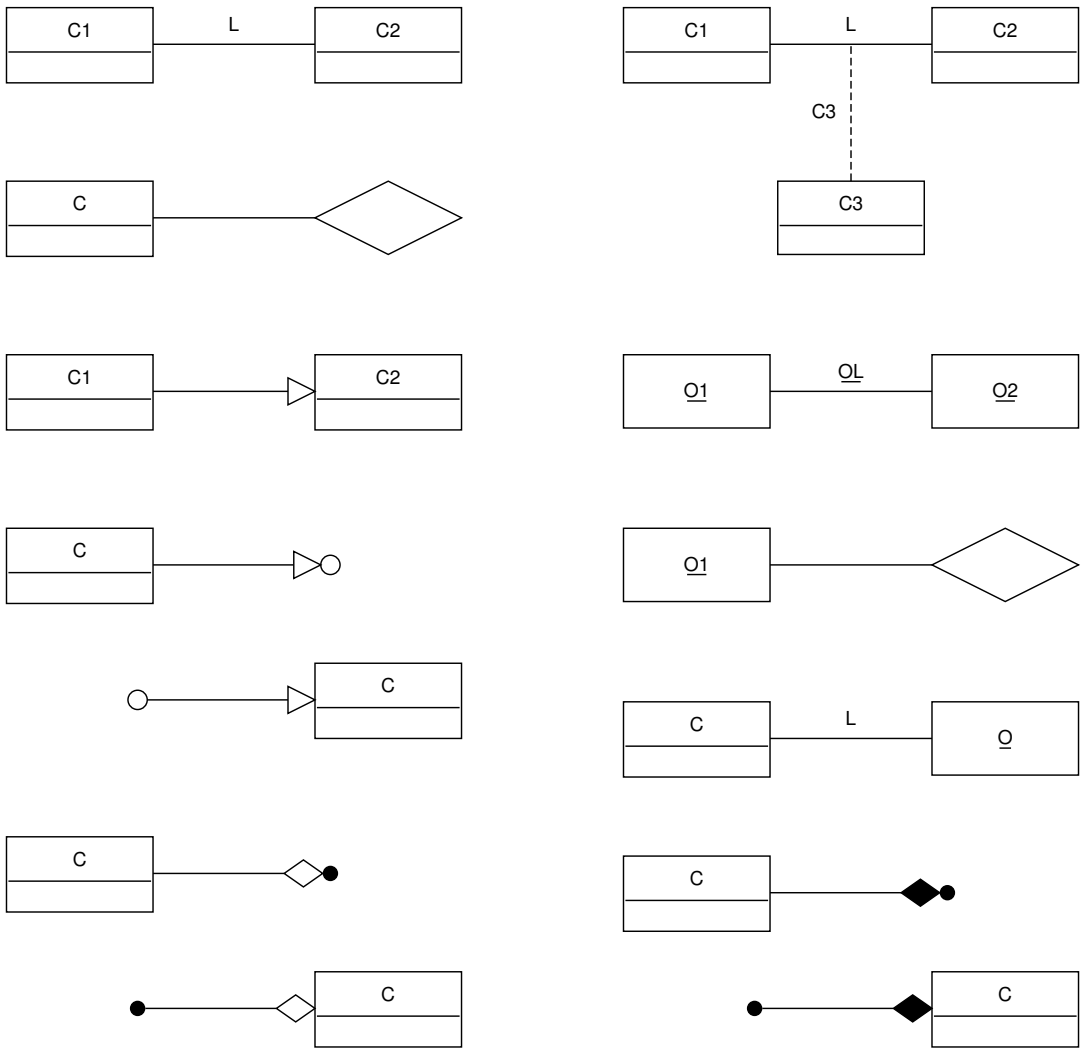


Figure 4.19: Permitted Static structure connections.

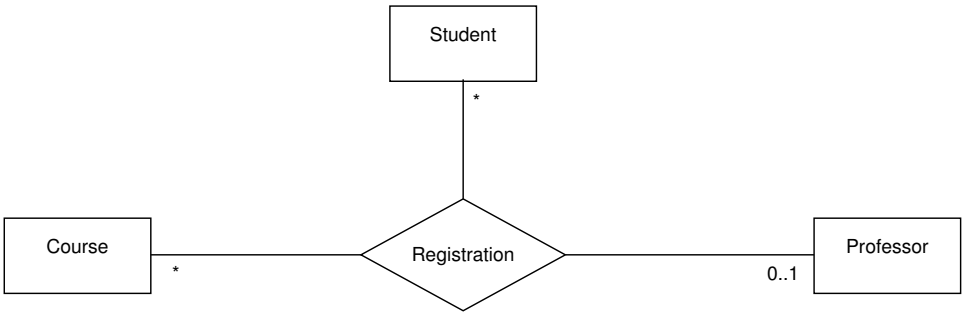


Figure 4.20: Example of a n-ary association.

4.4.3 Objects

In addition to classes, objects can be specified. An object is an instance of a class, which can be represented by the same type of class box with the name of the instance being underlined. Objects can be represented by two different box types : single boxes where only the name label is visible, or double boxes in which the name label and all the attributes are visible. The top compartment contains the object and class name in the form of **objectname:classname**. The bottom compartment may list the attribute names and values. Operations are not shown in objects, because they are all the same for all objects of a class. See figure 4.21 for permitted object notations. Object nodes can be connected to other objects by means of an object link edge. Use a participant link to connect an object to a n-ary relationship node.

Figure 4.21: Example Object notations.

4.4.4 Taxonomic Structures

In addition to UML, TSSD supports taxonomic structures using generalization and aggregation nodes. A generalization node is being represented by a small circle. Classes may be connected to a generalization node using the generalization edges. Like TESD, generalization nodes may have a label. The following labels are permitted : "", "d", "c", "dc" or "cd". A "d" stands for disjunctive and a "c" stands for complementary. These constraints are immediately enforced. The open arrow of the generalization arrow will disappear on connecting it from a class box to a generalization node; an empty edge will be shown instead.

An aggregation node is being represented by a small black circle. Classes may be connected to an aggregation node using the aggregation or composition edges. The open diamond of the aggregation arrow and the black diamond of the composition arrow will disappear on connecting it from a class box to an aggregation node; an empty edge will be shown instead. See figure 4.22 for some permitted notations.

4.4.5 Constraint Checking

TSSD checks also the soft and immediately enforced constraints that are summarized in figure 4.23.

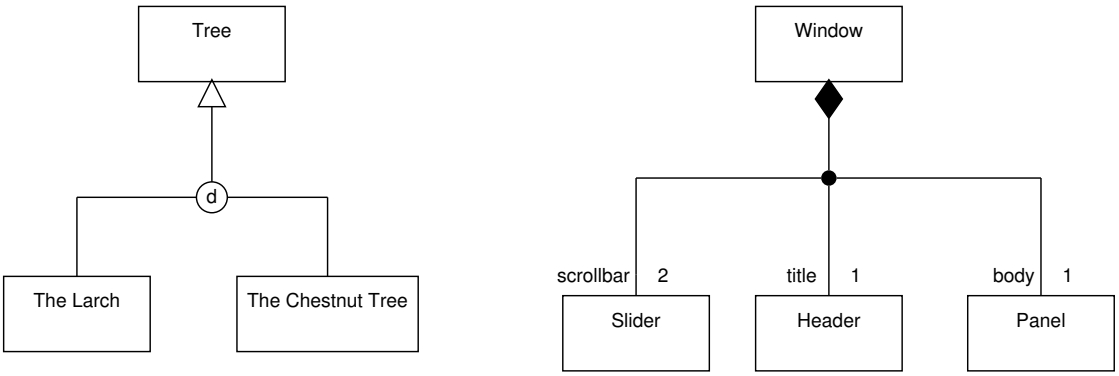


Figure 4.22: Example taxonomic notations.

Constraint	Check	Description
SSD1	Immediately	Names of classes, objects, associations and links contain at least one letter.
SSD2	Immediately	Generalization relationships should not contain cycles.
SSD3	Immediately	Aggregation relationships should not contain cycles.
SSD4	Soft	Associations and links are properly connected: they have at least two connections to a class, object or n−ary association..
SSD5	Soft	Binary associations (edge) should either have a non−empty name or at least one non−empty role name.
SSD6	Soft	Association nodes and link nodes should be named.
SSD7	Immediately	Edges from the same association node to the same subject should have different names.
SSD8	Immediately	Each attribute and each operation definition contains at least one letter and no newlines.
SSD9	Immediately	The attribute and operation definitions within the same class are mutually exclusive.
SSD10	Immediately	The attribute definitions within the same object are mutually exclusive.
SSD11	Immediately	Taxonomy junctions have one of the labels : "", "d", "c", "dc" or "cd".
SSD12	Immediately/Soft	Taxonomy junctions are properly connected (one generalization edge departing from the junction and two or more generalization edges arriving at the junction); all edges connecting different subjects.
SSD13	Immediately/Soft	Aggregation junctions are properly connected (one aggregation or composition edge departing from the junction and two or more aggregation or composition edges arriving at the junction); all edges connecting different subjects.

Figure 4.23: Immediately checked and soft constraints on SSDs.

Chapter 5

Behavior View Editors

Behavior view editors manipulate various kind of diagrams to represent system or object behavior.

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.

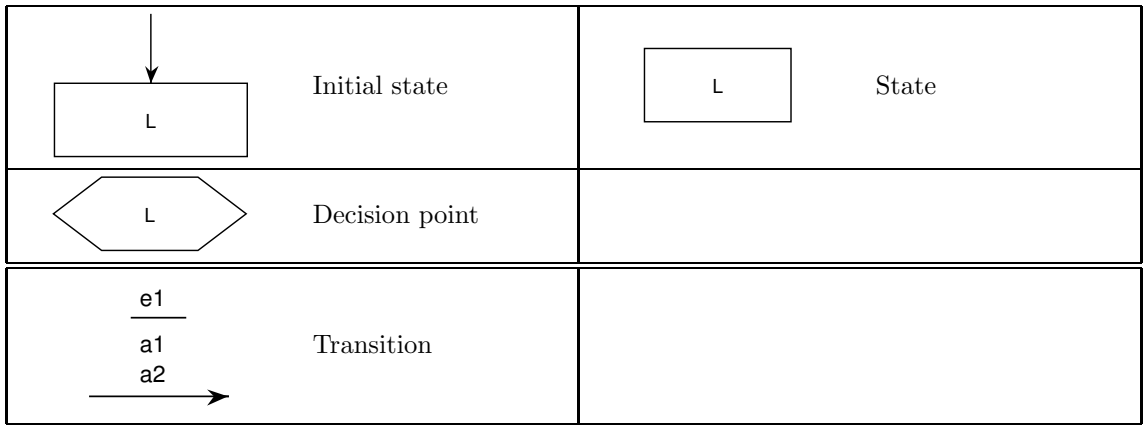


Figure 5.1: State transition diagram nodes and edges.

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 or near the arrow on top of the box. When an edit cursor appears at

the right side of the arrow, you can type in the actions. Each line of text will be considered as a separate action. If you stop editing, the actions will be placed at the right side of the arrow and the texts become left aligned. The arrow will be resized to accommodate the height of the texts 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 a single editable name label like most of the other line or arrow types in TCM. Instead they have a distinct event label and an arbitrary 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 normally be positioned. When the transition line segment near the separator is drawn vertically, the separator is by default connected to the right side of that line segment. When the segment is drawn horizontally, the separator is by default positioned somewhat above the line. See figure 5.2 for the two default positions.

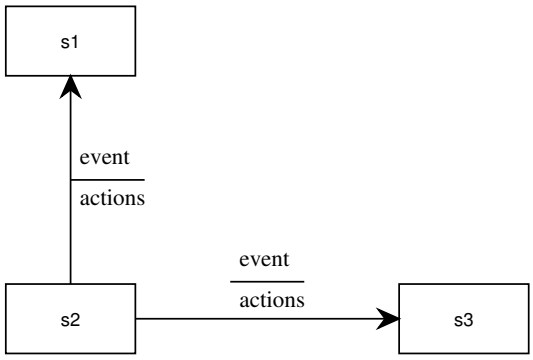


Figure 5.2: Default STD separator positions.

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. You are advised to experiment with this. These movements are:

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

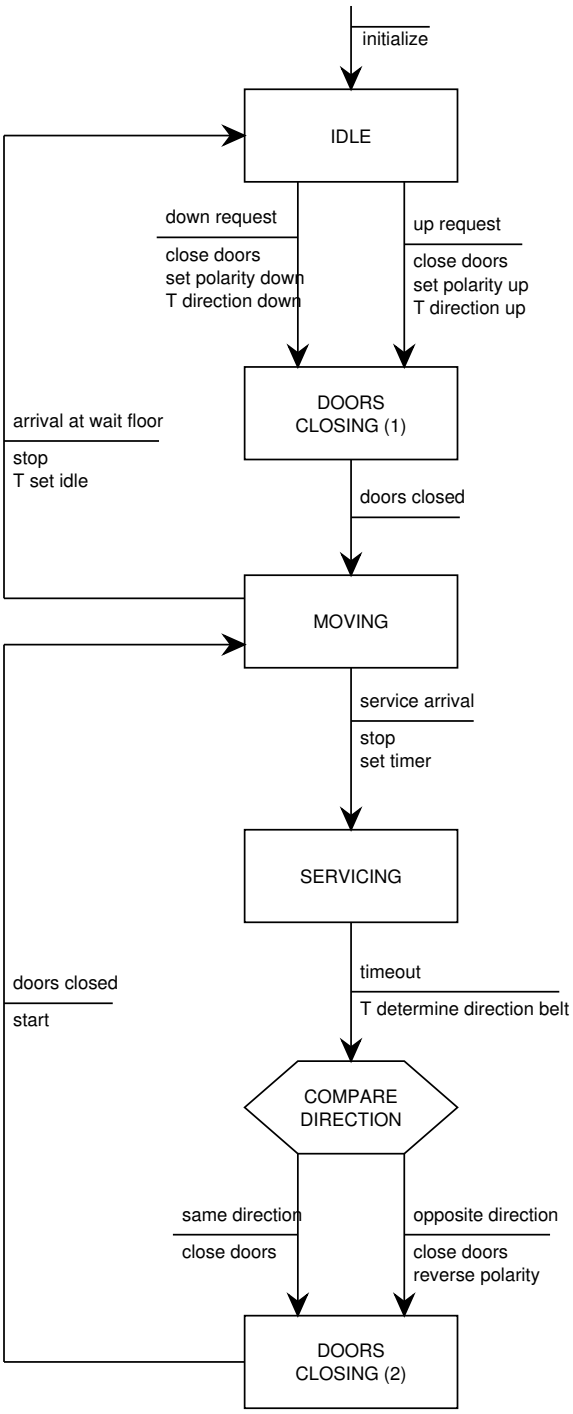


Figure 5.3: STD with events and actions.

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.

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 or curved, 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 Activity Diagram Editor (TATD)

5.2.1 Nodes and Edges

An activity diagram is a special case of a state diagram in which most states are action states or subactivity states, and most transitions are completion transitions. The purpose of an activity diagram is to focus on flows driven by internal processing (as opposed to external events). See figure 5.5 for the shapes and subjects.

	Action state		Inactive state
	Horizontal synchronization bar		Vertical synchronization bar
	Start state		End state
	Decision		
	Control flow		

Figure 5.5: Activity diagram nodes and edges.

5.2.2 States

There are four kinds of nodes: **start and end states**, (active and wait) **states**, **decisions** and **synchronizations**. Nameless states and non-deterministic activity diagrams are not permitted.

A decision, represented by a small diamond, can be used as a decision point (one incoming arrow and two or more outgoing arrows), as a merge point (two or more incoming arrows and one outgoing arrow), or as a combined merge/decision point. See figure 5.6 for an example of decision and merge.

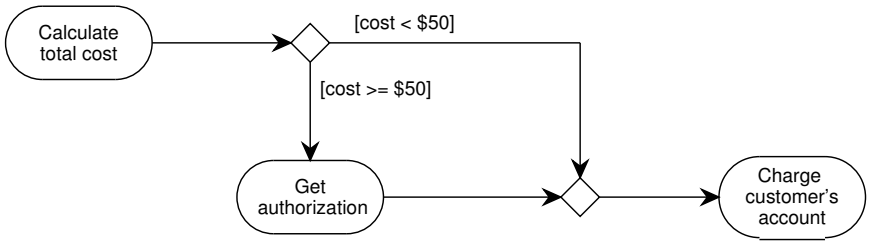


Figure 5.6: Example decision and merge.

Synchronizations, represented by horizontal or vertical synchronization bars, should have one incoming and two or more outgoing control flows or two or more incoming flows and one outgoing flow. Via the Update Node Shape Type option in the edit menu it is possible to convert between these two synchronization representations.

See figure 5.7 for an example diagram with horizontal synchronization bars.

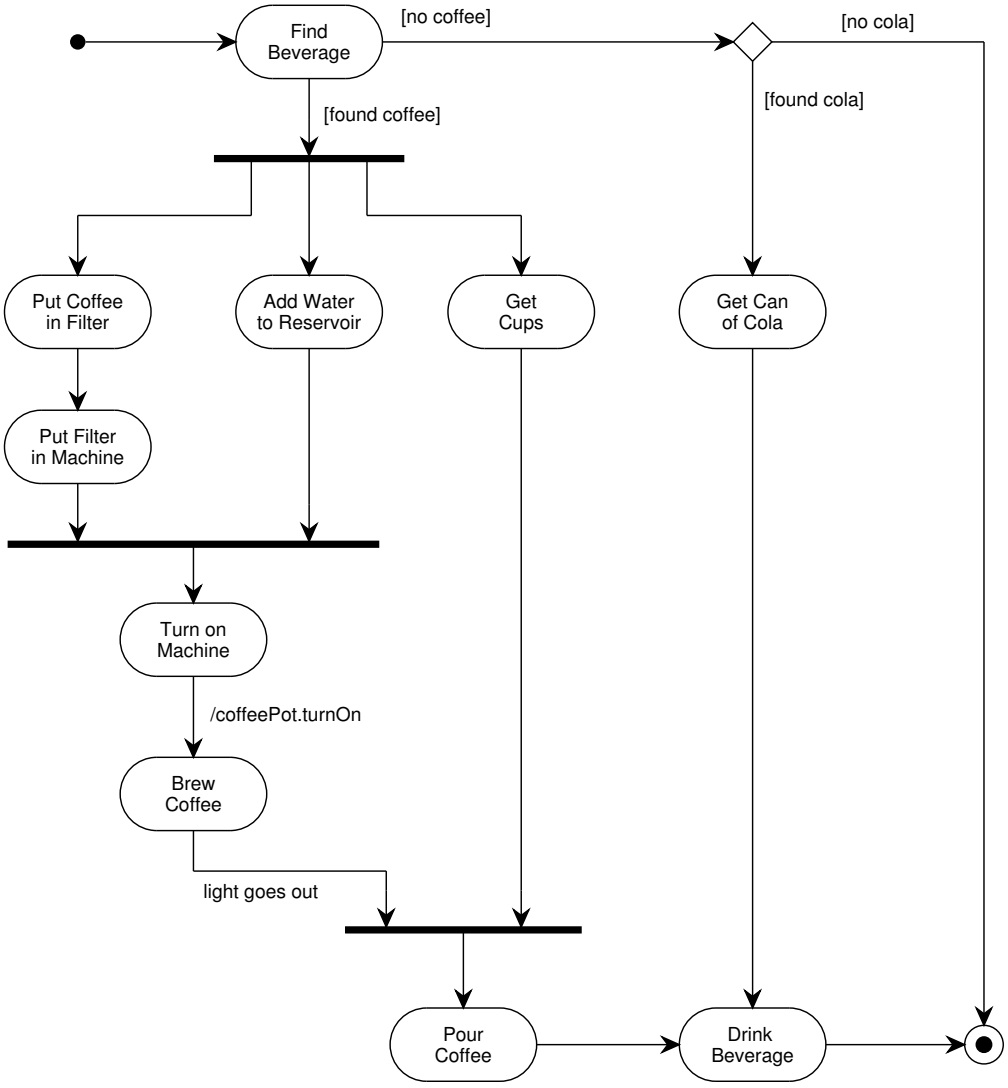


Figure 5.7: ATD with action states, synchronization bars and a decision.

5.2.3 Transitions

Transitions are drawn as arrows, which have a single editable name label like most of the other line or arrow types in TCM. Restrictions on transitions can be found in figure 5.8.

5.2.4 Constraint Checking

TATD checks the following constraints that are summarized in figure 5.8.

Constraint	Mode	Description
AT1	Soft	Each ATD contains exactly one initial state.
AT2	Soft	Each ATD contains at least one action state.
AT3	Soft	Each ATD contains at least one final state.
AT4	Immediately	You cannot have a transition from and to the initial state.
AT5	Immediately	You cannot have a transition from and to a final state.
AT6	Immediately	You cannot connect different final states.
AT7	Immediately/Soft	Each node is reachable from the initial state by a finite sequence of transitions.
AT8	Immediately/Soft	Each node can reach a final state by a finite sequence of transitions.
AT9	Immediately	A transition should not lead from and to the same decision or merge point.
AT10	Immediately	A transition should not lead from and to the same synchronization bar.
AT11	Soft	Synchronization bars should have one incoming and two or more outgoing control flows or two or more incoming flows and one outgoing flow.
AT12	Soft	Transitions from an action state should not contain event triggers.
AT13	Soft	Transitions from an inactive state should contain event triggers.
AT14	Soft	Transitions from and to the same state should have an action.
AT15	Soft	Transitions from a decision point should have no event.
AT16	Soft	Transitions from a decision point should have a guard.
AT17	Soft	Transitions from a merge point should have a no event.
AT18	Soft	Transitions from a merge point should have a no guard.
AT19	Immediately	All names of action states and inactive states contain at least one letter.
AT20	Immediately/Soft	Transitions from and to the same node have mutually unique labels.

Figure 5.8: Immediately checked and soft constraints on ATDs.

5.3 The Process Structure Diagram Editor (TPSD)

5.3.1 Nodes and Edges

TPSD has just one special type of node, called Process and one type of edge called Empty edge (see figure 5.9). 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 5.10 for the different operators.

Figure 5.9: Process structure diagram nodes and edges.

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

Figure 5.10: Process structure operators.

5.3.2 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 [1].

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 5.11 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.

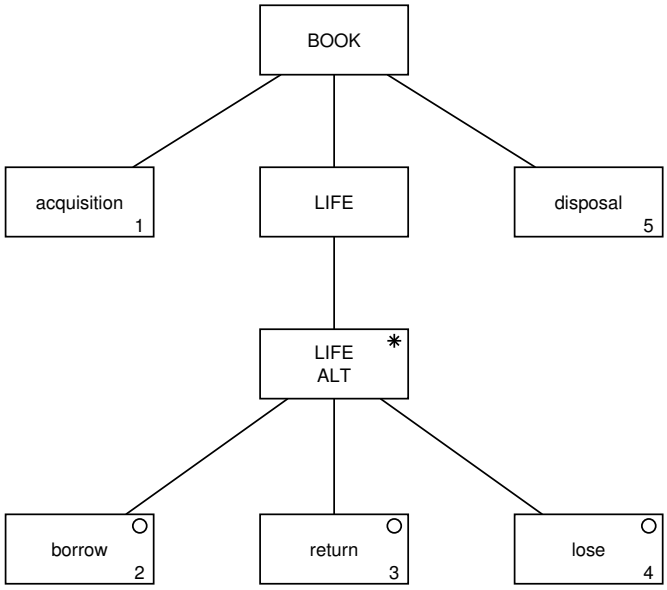


Figure 5.11: Example PSD with numbered actions.

5.3.3 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 5.12 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.

[1] 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 5.12: Immediately checked and soft constraints on PSDs.

5.4 The Recursive Process Graph Editor (TRPG)

5.4.1 Nodes and Edges

See figure 5.13 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 larger rounded boxes which can contain a name label. Process graph nodes are connected by event edges. See figure 5.14 for an example recursive process graph.

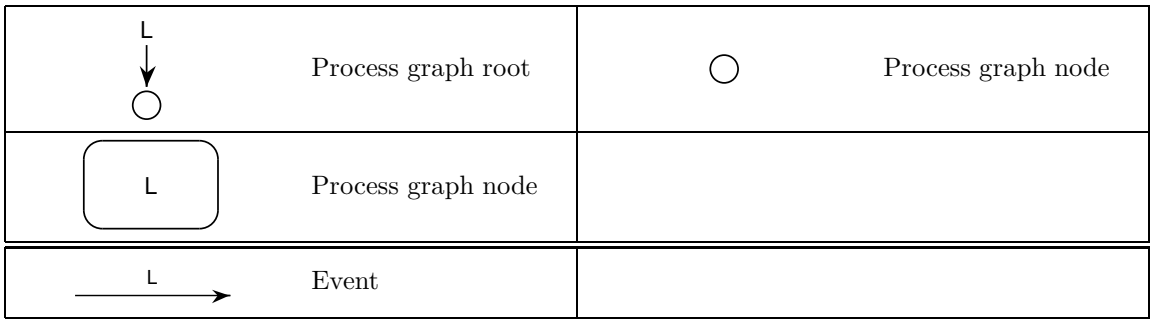


Figure 5.13: Recursive process graph nodes and edges.

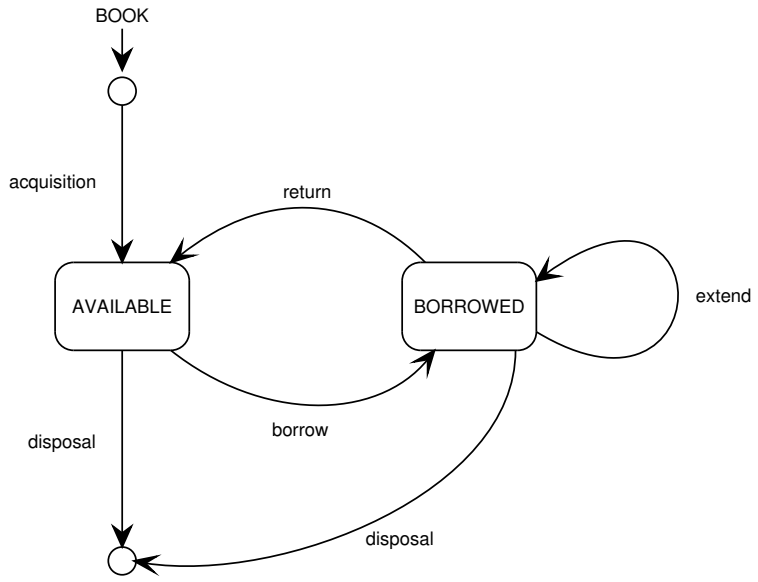


Figure 5.14: Example recursive process graph.

5.4.2 Constraint Checking

TRPG checks the immediately enforced and soft constraints that are summarized in figure 5.15.

Constraint	Check	Description
RP1	Immediately	If a pair of events connects the same pair of nodes and has the same direction 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 5.15: Immediately checked and soft constraints on RPGs.

5.5 The Collaboration Diagram Editor (TCBD)

5.5.1 Nodes and Edges

Collaboration diagrams describe the interactions among classes and associations. These interactions are modeled as exchanges of messages between classes through their associations. See figure 5.16 for the shapes and subjects.

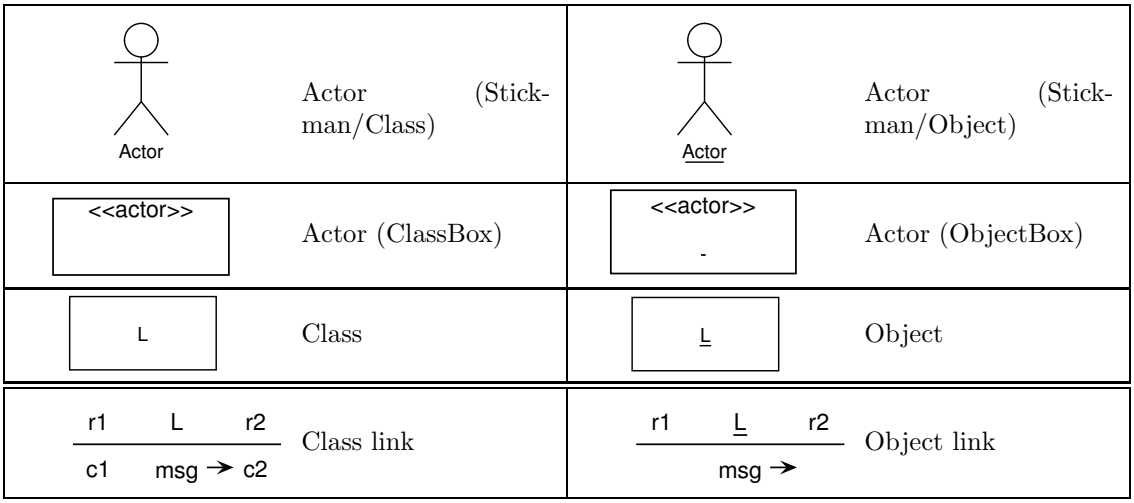


Figure 5.16: Collaboration diagram nodes and edges.

5.5.2 Interactions

Interactions are drawn as any other line type in TCM. In addition to the standard line types, interactions also have an arbitrary number of message labels located at the opposite side of the name area of the link.

On creation of an interaction an initial dummy message will be created, consisting of a default message text ("`edit this`") followed by a message arrow. See figure 5.17 for an example of a default interaction.

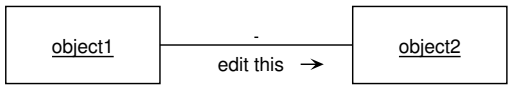


Figure 5.17: Default interaction.

The order of the interaction may be described with a sequence of numbers, usually starting with number 1.

The message text can be edited as usual. The message direction can be changed either by:

- clicking on it with the left-button: this will toggle the message direction.
- by adding a "direction prefix" add the end of the message text :
  - '`>`' : Add message direction from "left shape" towards "right shape".
  - '`<`' : Add message direction from "right shape" towards "left shape".
  - '`^`' : Add message direction from "upper shape" towards "down shape".
  - '`v`' : Add message direction from "down shape" towards "upper shape".

  In this way message directions can be added easy without having to switch between keyboard and mouse alternately. E.g. adding two messages to an interaction:
  `"1.2: create() > [CR] 1.3: respond() <"`
  will result in:

  `1.2: create()` →
  `1.3: respond()` ←

5.5.3 Constraint Checking

TCBD checks the immediately enforced and soft constraints that are summarized in figure 5.18.

Constraint	**Description**	**Check**
UC1	Names of objects contain at least one non−space character.	Soft
UC2	Stereotype label <<Actor>> can not be changed	Immediately

Figure 5.18: Immediately checked and soft constraints on CBDs.

5.6 The StateChart Diagram Editor (TSCD)

Statechart diagrams describe all the possible states a particular object can get into and how the object's state changes as a result of events that reach the object.

In contrast to most editors, TSCD starts in the hierarchical document mode. To select a node, click near any visible part in the node, for example on the inner side of the border.

5.6.1 Nodes and Edges

See figure 5.19 for the shapes and subjects.

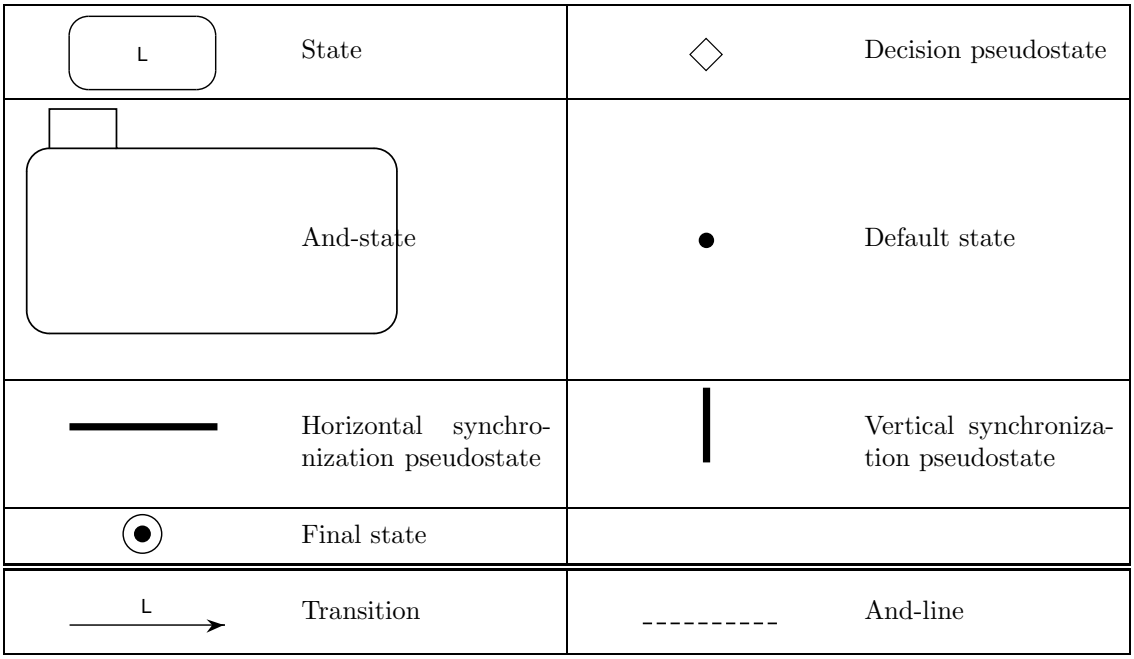


Figure 5.19: Statechart diagram nodes and edges.

5.6.2 And-states

Drawing an and-state requires special attention. An and-state is divided into substates by dashed lines. You can draw and lines using the middle mouse button, like you would draw edges between different nodes. TSCD includes a feature that automatically adds intermediary points to the and line[2] Unfortunately, TSCD is not able to move or resize the and line with its intermediary points automatically when you move or resize the and-state node. To move an and-state, always also select the and lines. After resizing an and-state, you have to readjust the intermediary points in the and lines.

See figure 5.20 for an example diagram with an and-state.

[2]This is needed because a line from a node to itself always has to have at least two intermediary points. The intermediary points help in determining where on the border the and line needs to be attached.

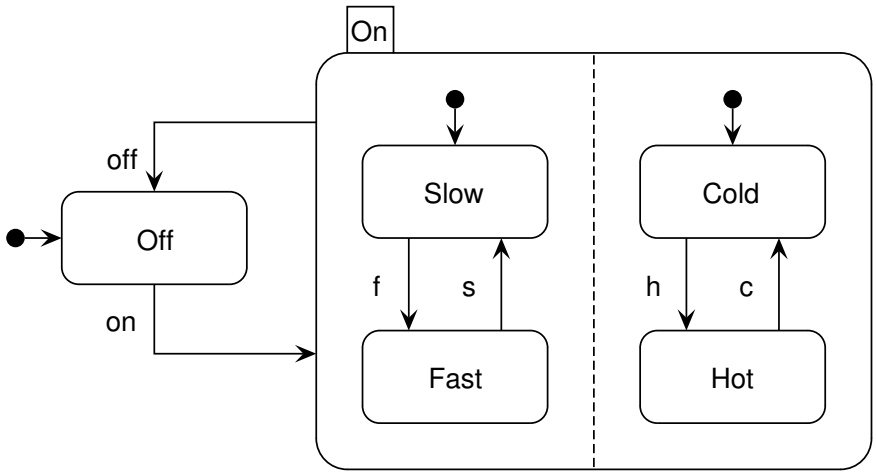


Figure 5.20: SCD with an and-state and and line.

Chapter 6

Architectural View Editors

The architectural view of a software system decomposes the system into parts that communicate with each other. This includes data flow diagrams and JSD system network diagrams.

6.1 The Data Flow Diagram Editor (TDFD)

6.1.1 Main window

Both TDFD as TEFD 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 [8, 30] 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 [22]: 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 [22], 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 A.3.3.

6.1.3 Data Flow Diagram Levels and Indexes

Data processes have unique indexes. Data process shapes show their index labels when the **create/edit index** check button in the list of tiled node buttons is on.

When you turn the toggle off, the index labels remain visible, but can not be edited anymore. New node shapes will be created without an index label.

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

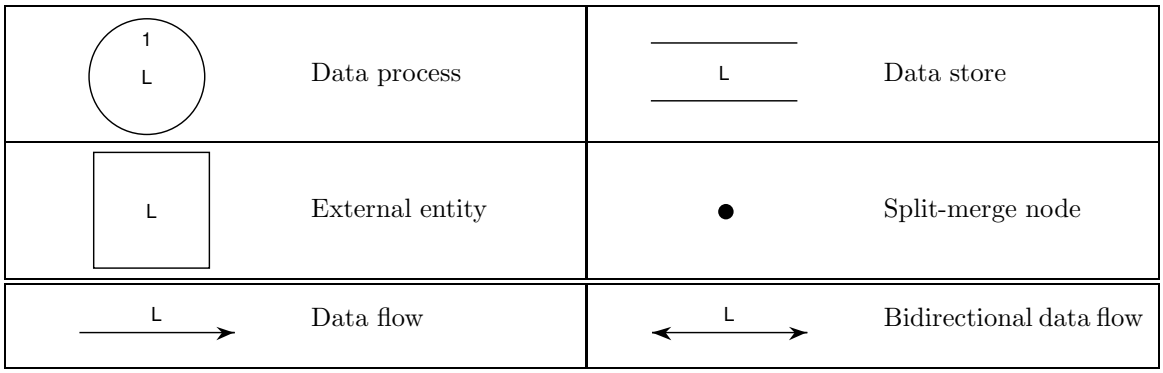


Figure 6.1: Data flow diagram nodes and edges.

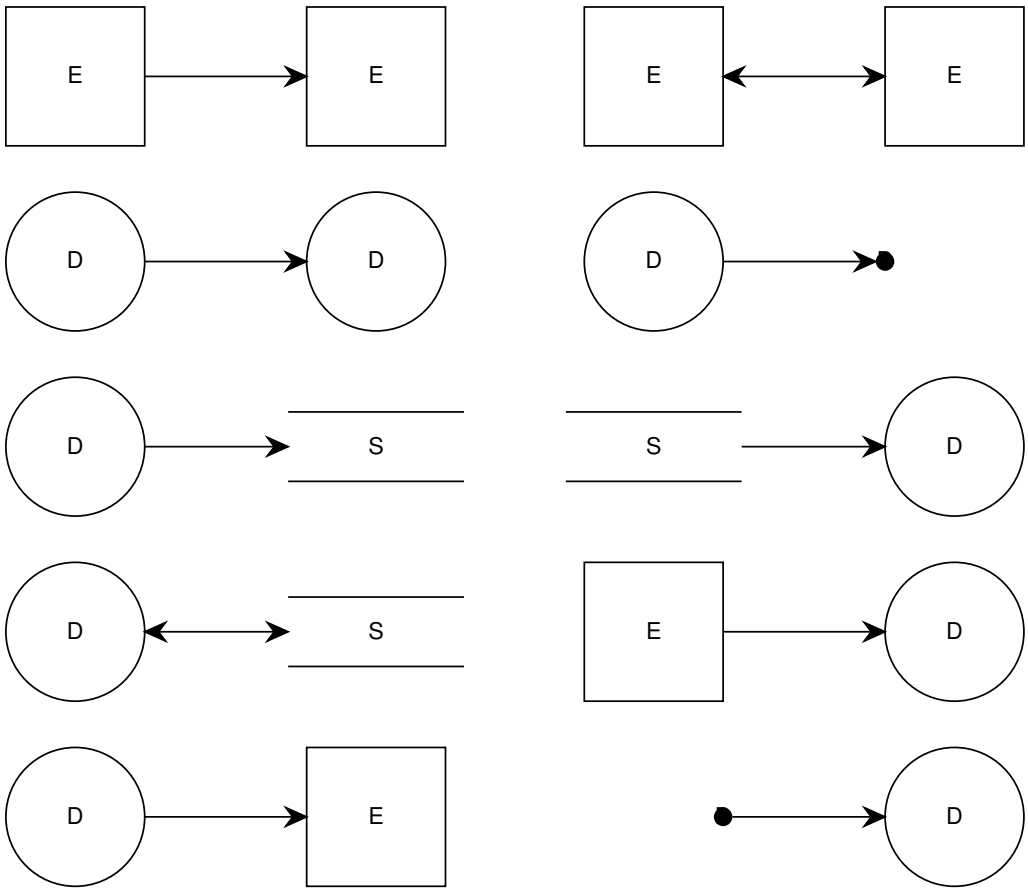


Figure 6.2: Permitted data flow diagram connections.

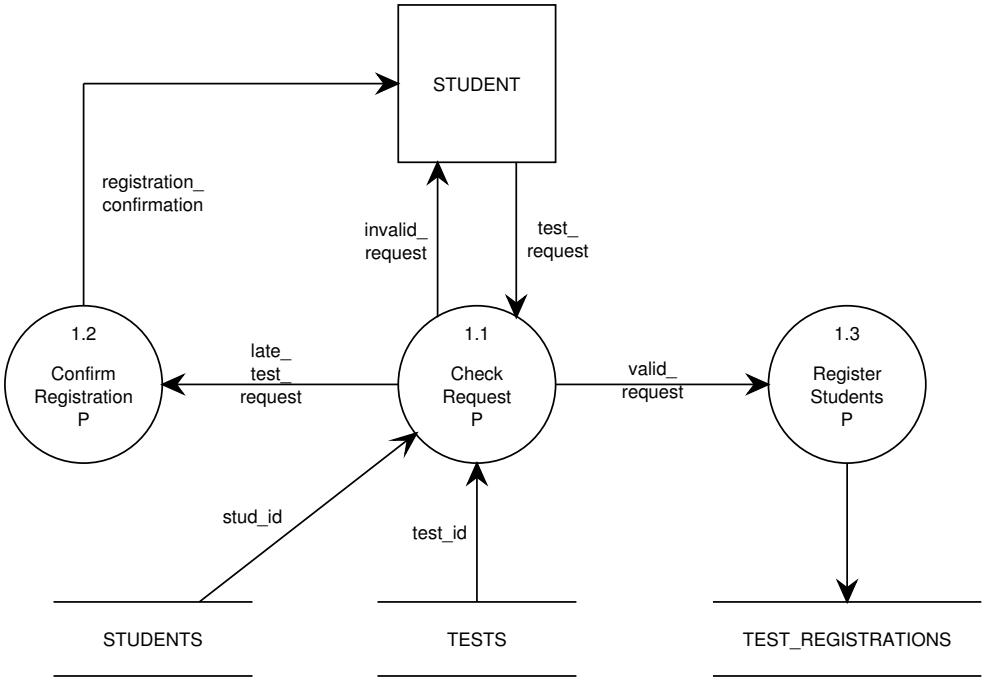


Figure 6.3: Data flow diagram in graph notation.

issue the command **Renumber indexes** from the Edit menu, then the 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.

$$
\begin{array}{lcl}
\mathit{Index} & \rightarrow & C \mid \mathbf{0} \\
C & \rightarrow & N\ Z^* \mid N\ Z^*\ .\ C \\
Z & \rightarrow & \mathbf{0} \mid N \\
N & \rightarrow & \mathbf{1}|\mathbf{2}|\mathbf{3}|\mathbf{4}|\mathbf{5}|\mathbf{6}|\mathbf{7}|\mathbf{8}|\mathbf{9}
\end{array}
$$

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 decom-

position 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].

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 5(a) and 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.

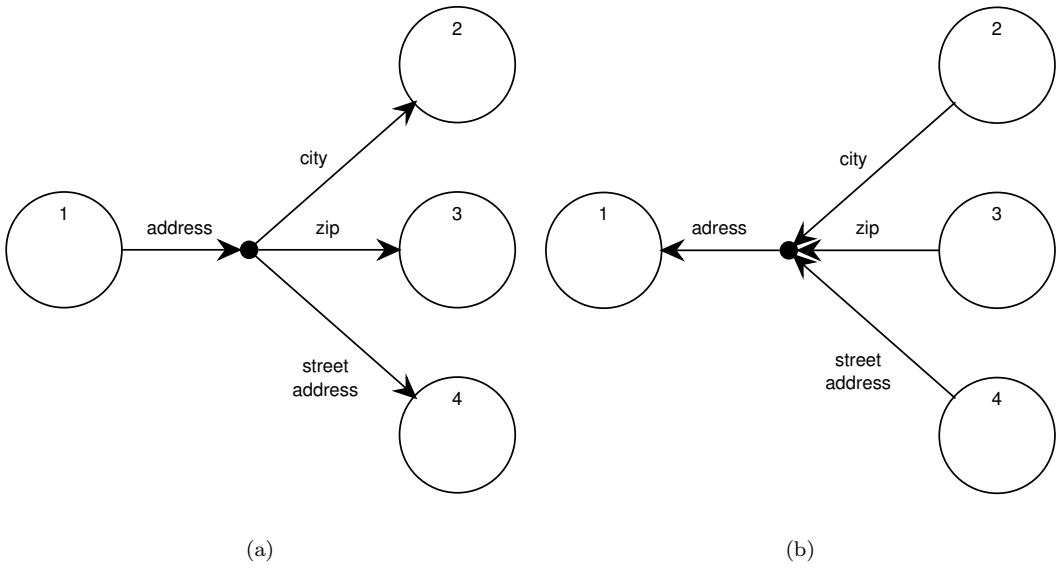


Figure 6.5: Example splitting and merging data flows.

[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.

6.1.6 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.

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 The Data and Event Flow Diagram Editor (TEFD)

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

6.2.1 Nodes and Edges

TEFD 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. TEFD 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).

TEFD 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.

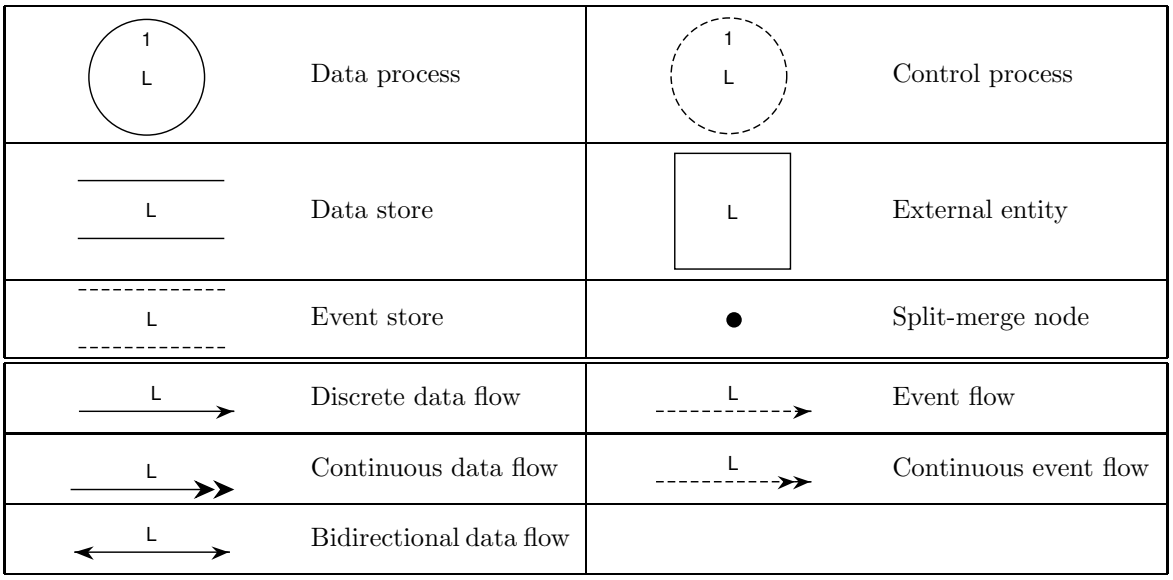


Figure 6.7: Data and event flow diagram nodes and edges.

6.2.2 Constraint Checking

TEFD checks all the constraints of TDFD. For these constraints see figure 6.6. The other constraints that TEFD 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 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.

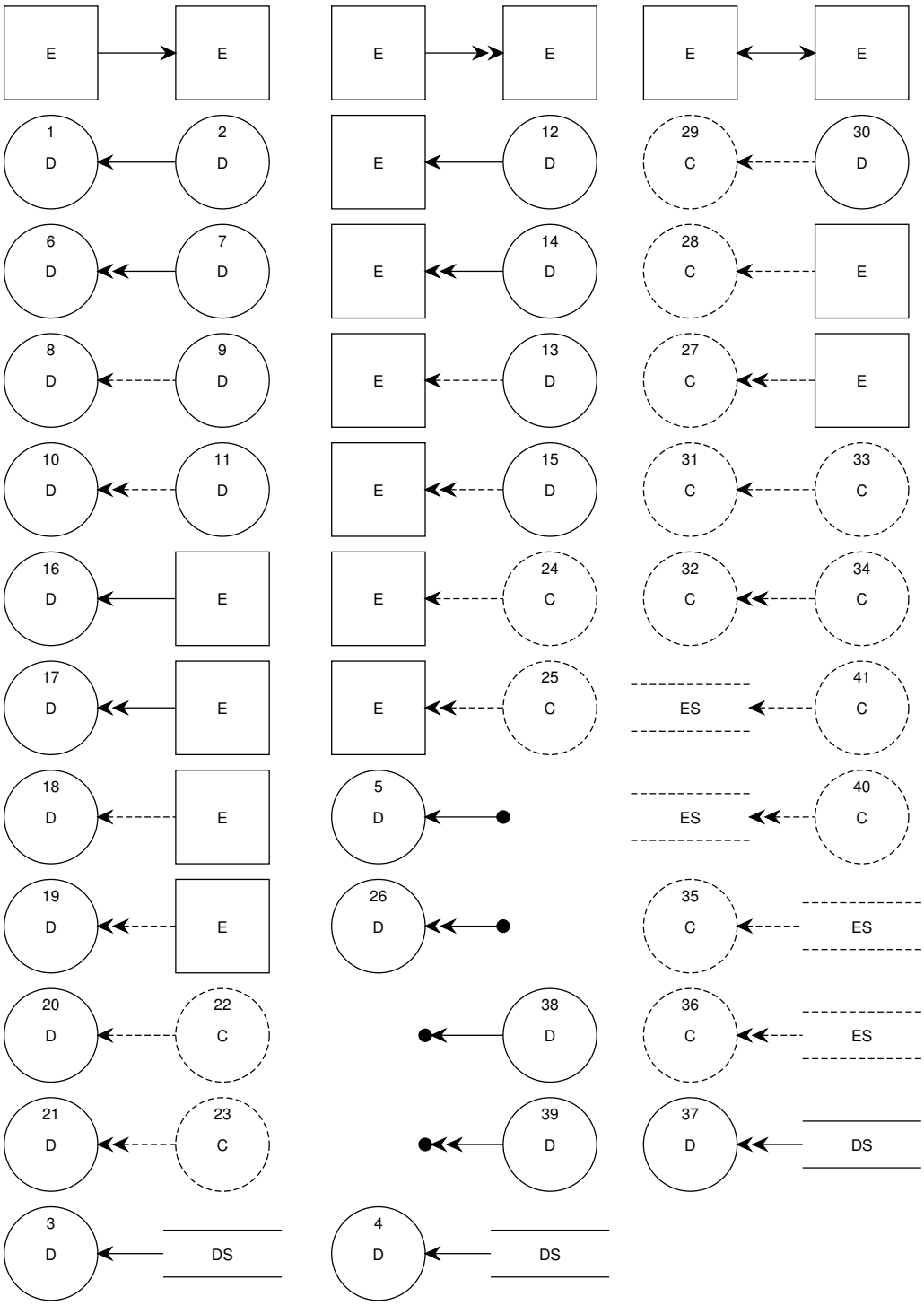


Figure 6.8: Permitted data and event flow diagram connections.

Constraint	Mode	Description
EF1	Immediately/Soft	Data and control processes, data and event stores and external entities have mutually unique non−empty names.
EF2	Soft	Each control process has at least one output flow.
EF3	Immediately	Flows to and from an event store are unnamed.
EF4	Soft	Each event flow is connected.
EF5	Immediately	All flows that connect a split/merge node are either all discrete or all continuous.
EF6	Immediately	(Continuous) data flows should not be labeled as a trigger or prompt.
EF7	Immediately/Soft	Each event flow from a control process to a data process should be either labeled as a prompt or as a trigger.
EF8	Immediately	Event flows between two controll processes should not be labeled as a prompt.
EF8	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.
EF9	Immediately	Continuous event flows should not be labeled as a trigger or a prompt.

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

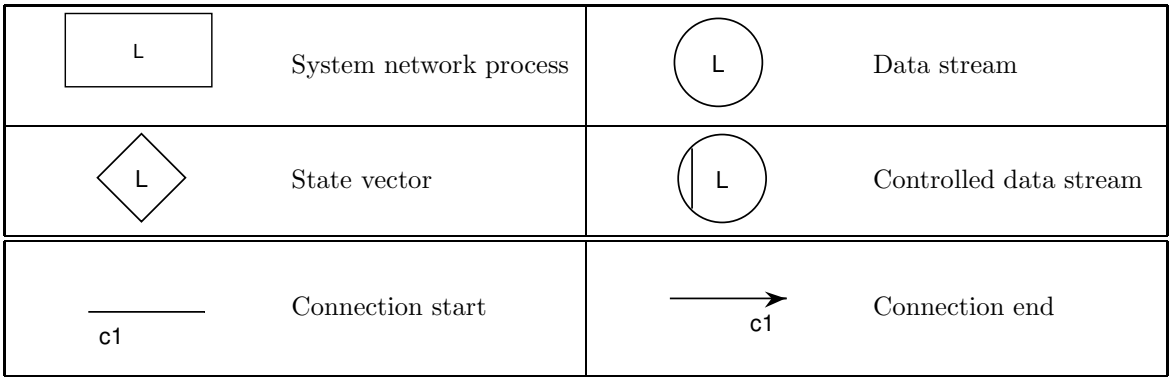


Figure 6.10: System network diagram nodes and edges.

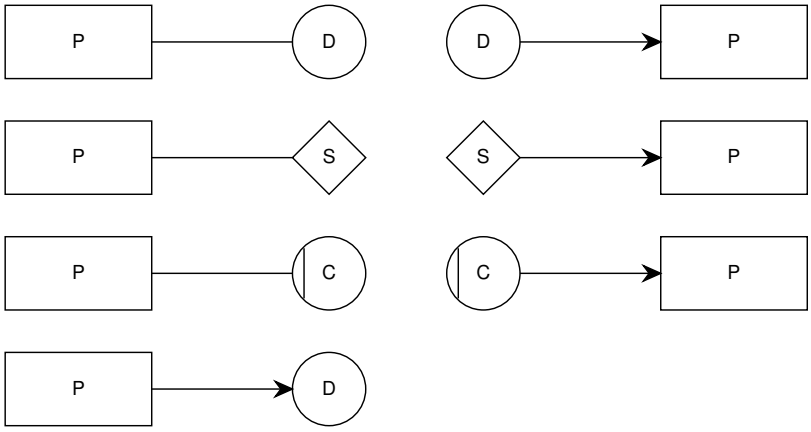


Figure 6.11: Permitted system network diagram connections.

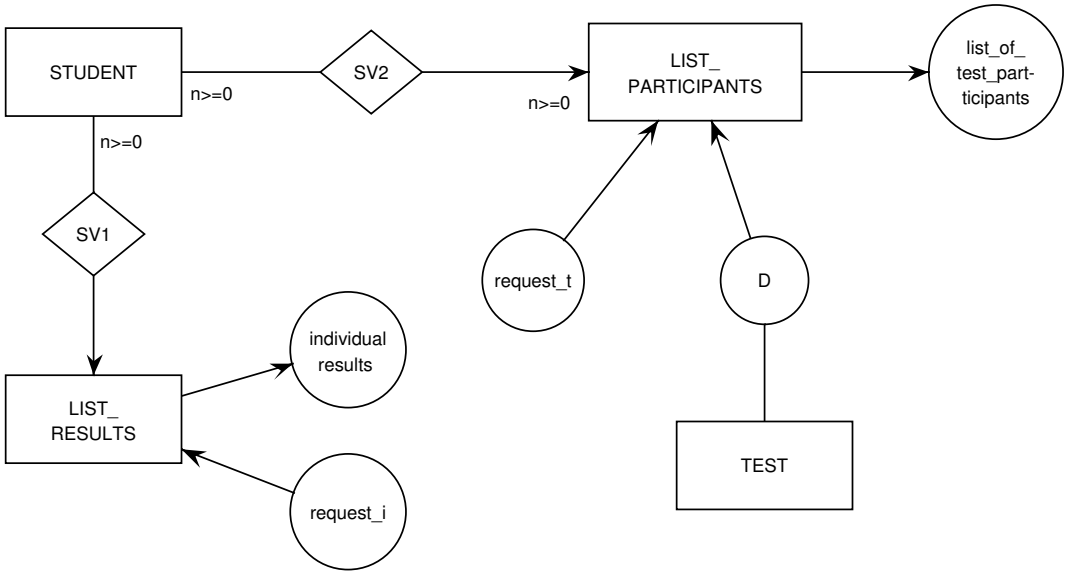


Figure 6.12: Example system network diagram.

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.

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.

6.4 The Use Case Diagram Editor (TUCD)

6.4.1 Nodes and Edges

See figure 6.14 for the subjects and the representing shapes that are used in TUCD. In figure 6.15 you can see which node types can be connected by which edge types. These are immediately enforced constraints. Note that actors can be represented by two different actor types : a StickMan and a ClassBox . In figure 6.15 for actors only the stick-man type variant is shown but each stick-man could be replaced by a class box. You can change the representation of an actor by the **change actor type** command in a sub-menu of the Edit menu.

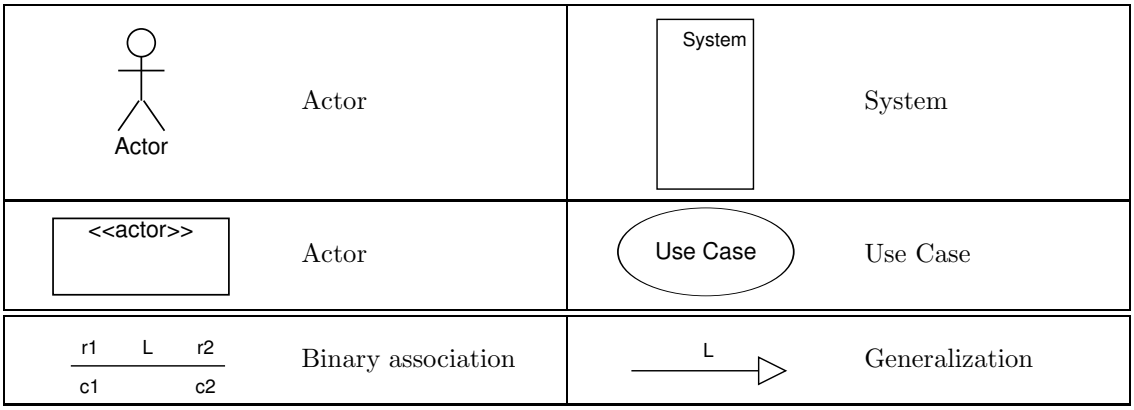


Figure 6.14: Use case diagram nodes and edges.

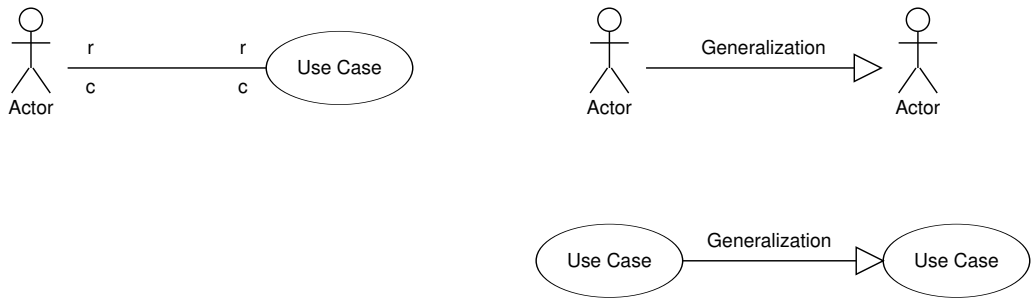


Figure 6.15: Permitted Use case diagram connections.

6.4.2 Constraint Checking

TUCD checks also the soft and immediately enforced constraints that are summarized in figure 6.16.

Constraint	Check	Description
UC1	Soft	Names of actors and use cases contain at least one non−space character.
UC2	Immediately	Specialization relationships should not contain cycles.
UC3	Immediately	Actors and use cases should have mutually unique names.
UC4	Immediately	Only one system per diagram is allowed.
UC5	Immediately/Soft	No use cases allowed outside a system.
UC6	Immediately/Soft	No actors allowed within a system.
UC7	Immediately	Specialization relationships should be between two actors or between two use cases.
UC8	Immediately	Binary association should be between actor and use case.
UC9	Immediately	Binary association has no label (name).
UC10	Immediately	Stereotype label <<Actor>> can not be changed

Figure 6.16: Immediately checked and soft constraints on UCDs.

6.5 The Component Diagram Editor (TCPD)

This is a very lightweight diagram editor, only added to support basic drawing of component diagrams. There is no constraint checking built in.

6.5.1 Nodes and Edges

See figure 6.17 for the subjects and the representing shapes that are used in TCPD.

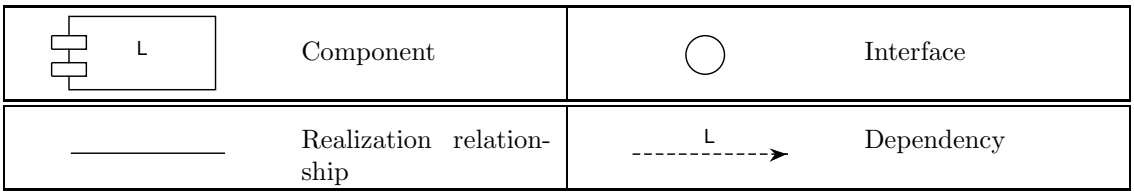


Figure 6.17: Component diagram nodes and edges.

6.6 The Deployment Diagram Editor (TDPD)

This is a very lightweight diagram editor, only added to support basic drawing of deployment diagrams. There is no constraint checking built in.

TDPD is a superset of TCPD. The features that are specific for T are explained in this section. In short, TDPD is TCPD extended with UML nodes. All nodes and edges of TCPD are available in TDPD. It is also possible to read a `.cpd` diagram into TDPD. The other way around, reading a `.dpd` diagram in TCPD is only possible when it does not contain UML nodes.

6.6.1 Nodes and Edges

See figure 6.18 for the subjects and the representing shapes that are used in TDPD.

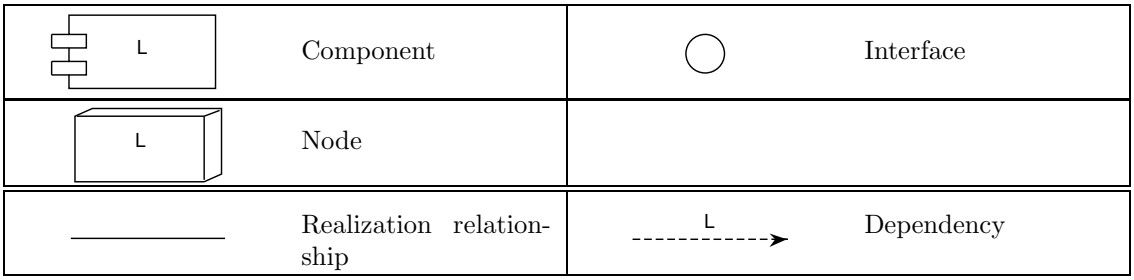


Figure 6.18: Deployment diagram nodes and edges.

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. These commands can be issued from the Edit menu.

If you start up a table editor from the `tcm` start-up tool, before the main window of the editor is displayed, a dialog window with a list of text fields is presented. This dialog has four fields: number of rows, number of columns, default row height and default column width. In these fields the default values are already filled in. You can change these values for the editor that will be launched.

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. In the View menu there is an option to hide the labels. In the Printer Options submenu of the Print menu you can choose to print these labels or not. 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

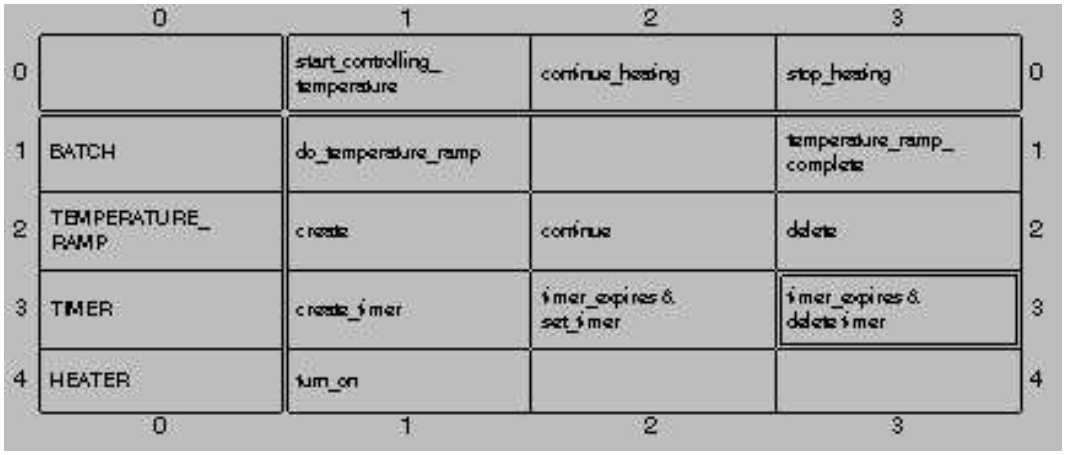


Figure 7.1: Snap-shot of a table being edited.

of a table have the same number of cells. All cells in a row have the same height and all cells in a 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 both near the left edge of the first cell of the row and near the right edge of the last 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), both positioned above the first cell of the column and below the last 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, 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.

- **Move the selection**. With the four arrow keys on your keyboard you can move a single-cell or multiple-cell selection with one cell.
- **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**.

For going into edit mode in a table editor, you have to be sure that only a single cell is selected. When a single cell is selected and you type in characters or you click with button-1 on the cell then you enter edit mode. You leave edit mode when you either click Button-2 with the mouse pointer in any position or you click button-1 *outside* the cell that is being edited.

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. The **default row height** and **default column width** is the cell height and cell width when the table is initialized. The default row height and column width of the table can be modified in the Default Properties submenu of the Properties 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 some 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

Figure 7.2: Add Row dialog window.

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 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.

[1] Contrary the common belief an empty table is still a table.
[2] This sounds more complicated than it actually is, experiment with this.

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. 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. 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 Properties menu can be undone. Furthermore, the table edit commands issued by the mouse can also be undone. In table 7.3 all undo-able table editor commands are listed together with how they can be called.

7.1.12 Changing Properties of a Table

The Properties menu contains certain commands that change properties of cells and/or their texts.

- **Update Line Style**.

  The style of each line piece in the table can be set individually. The possible line styles are: solid (default), dashed, dotted, dual or invisible. When you call Update Line Style from the Properties menu, a pop-up dialog window is displayed, see figure 7.4. The dialog window contains two list of toggles, the left one is for the different line styles and the right one for specifying which lines you want to update. The possible updates are:

  - **Update Top Sides**. Each line piece that borders on the top of a selected cell is updated.
  - **Update Bottom Sides**. Each line piece that borders on the bottom of a selected cell is updated.
  - **Update Left Sides**. Each line piece that borders on the left of a selected cell is updated.
  - **Update Right Sides**. Each line piece that borders on the right of a selected cell is updated.
  - **Update Surrounding Sides**. Each line piece that borders on exactly one selected cell is updated.
  - **Update All Four Sides**. Each line piece that borders on a selected cell is updated. Update line style is an undo-able command.

  The entry **Default Line Style** in the Default Properties submenu pops up a dialog window to set the default line style. Each newly created line piece will have this line style. It contains toggles with the possible values solid, dashed, dotted, dual and invisible.

- **Update Line Width**.

  The width of each line piece in the table can be set individually. The line width ranges from 1 till 6. When you call Update Line Width from the Properties menu, a pop-up dialog window is displayed, see figure 7.5. The dialog window contains two list of toggles, the left one is for

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.	Replace all cell texts	Issue Replace from the Search menu, fill in the texts to find and to replace with and click the Replace All button.
Add rows	Select desired row position, issue Add rows from the Edit menu, give the number of rows and click OK.	Replace next cell text	Issue Replace from the Search menu, fill in the texts to find and to replace with and click the Replace Next button.
Append table	Issue Append from the File menu, choose a file, set desired position and click OK.	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.
Copy cell texts	Select the cells to be copied and issue Copy texts from the Edit menu.	Resize row	Drag with button−1 a line piece at the bottom side of the row 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.	Select all cells	Issue Select All from the Edit menu.
Delete all cells	Issue Delete All from the Edit menu.	Select cell area	Drag with button−2 over the cells to be selected.
Delete columns	Select the columns to be deleted and issue Delete Columns from the Edit menu.	Set/Unset Text Underlining	Issue Set/Unset Text Underlining from the Properties 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 Search 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 Search 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 choose one of the options of the Update Column Alignment dialog from the Properties menu.
Move column	Drag the label of the column to be moved with button−1 into a cell having the desired column position.	Update text font	Select cells and issue Update Text Font from the Properties menu. Choose the desired font from the dialog and click Apply.
Move row	Drag the label of the row to be moved with button−1 into a cell having the desired row position.	Update line style	Select cells and issue Update Line Style from the Properties menu. Choose the style and specify the sides to update and click Apply.
Move table	Click the arrow buttons and/or the center button.	Update line width	Select cells and issue Update Line Width from the Properties menu. Choose the width and specify the sides to update and click Apply.
Paste cell texts	Issue Paste from the Edit menu. Release the paste box at the desired position.	Update row alignment	Select the rows to be updated and choose one of the options of the Update Row Al− ignment dialog from the Properties menu.
Purge cells	Issue Purge from the Edit menu.		

Figure 7.3: All atomic table edit commands.

Figure 7.4: Line style dialog.

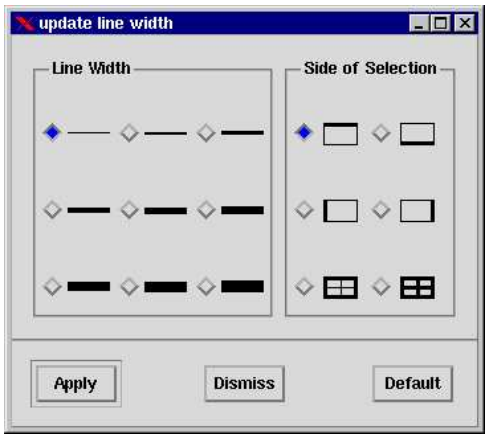


Figure 7.5: Line width dialog.

the line widths and the right one for specifying which lines you want to update. This works the same as specifying of which lines you want to change the line style as described in the previous item.

The entry **Default Line Width** in the Default Properties submenu pops up a dialog window to set the default line width. Each newly created line piece will have this line width.

- **Update Text Font**. This entry pops up a dialog window in which you can select a text font. A font consists of the attributes font family, font style and point size. For each of the three attributes there is a list of toggle buttons. Also, each list of toggle buttons in the dialog has an extra check button called `update` *attribute* that indicates whether that font particular attribute should be updated or not. This makes it possible, for instance, to only change point sizes or font families of some cell texts but to keep the other font attributes the same.

  The dialog also shows a preview of some text in the selected font so you can see how it will look in your diagram. When you press the `Apply`-button the dialog is dismissed and of each selected shape the font is updated to the selected font. Update font is an undo-able command.

  With the **Default Text Font** entry from the Default Properties submenu you get a similar dialog window. Here you can set the default text font. Each new text (i.e. text entered in an empty cell) will get this font. The row and column sequence labels and the page headers and numbers are also drawn in this default font.

- **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, this entry pops up a dialog window showing toggles with the three possible alignment types: Top, Center and Bottom. If you select one of these, and press `Apply`, 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.

  The default row alignment can be set via **Default Row Alignment** entry from the Default Properties submenu. 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 with a similar pop-up window as Update Row Alignment. 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 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, this entry pops up a dialog window showing toggles with the three possible alignment types: Left, Center and Right. If you select one of these, and press `Apply`, 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.

  The default column alignment can be changed via the entry **Default Column Alignment** from the Default Properties submenu, similar to how you change the default row alignment.

- **Set/Unset Text Underlining** This option sets/unsets (toggles) the text underlining of the selected cells.

The following default table properties can be set via the Default Properties submenu of the Properties menu:

- **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

via the entry Text Margin Width of the Default Properties submenu and a slider pop-up dialog is displayed. 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 via the entry Text Margin Height of the Default Properties submenu and a slider pop-up dialog is displayed. If, after the update, some of the texts do not fit into their cells and the autoresizing toggle is on, these cells are resized.

- **Default 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 default row height. Furthermore, when the autoresize toggle is on, autoresizing is applied to the rows.

- **Default 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 default column width. 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.

7.1.13 Miscellaneous Commands

- The **Cell annotation** command from the properties menu pops up a text edit dialog in which you can type arbitrary text to annotate a cell in the table. See section 2.5 for using the text edit dialogs.

- **Find**. With the Find menu entry in the Search 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 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 Search 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.

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. 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 7.7. Currently, the editor checks the constraints in figure 7.6.

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 7.6: Immediately checked and soft constraints on TDTs.

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

	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 7.7: Example transaction decomposition table.

editor does not check for that layout, however. For an example see figure 7.8. The editor checks the immediately and soft constraints of figure 7.9.

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

Figure 7.8: 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 [22]. 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 [22] 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 Line Width from the Properties menu. See figure 7.11 for an example table. Currently, the editor checks the constraints in figure 7.10.

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 7.9: 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.
FE4	Soft	There are no empty entries in column 0.
FE5	Immediately/Soft	All entries that are not in row 0 or in column 0 contain a "CRUD" string.
FE6	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.
FE7	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 7.10: 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 7.11: 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 [22] 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 (all edit commands in the menu are grayed out) but the tree can be loaded, saved, printed, previewed, scaled 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.

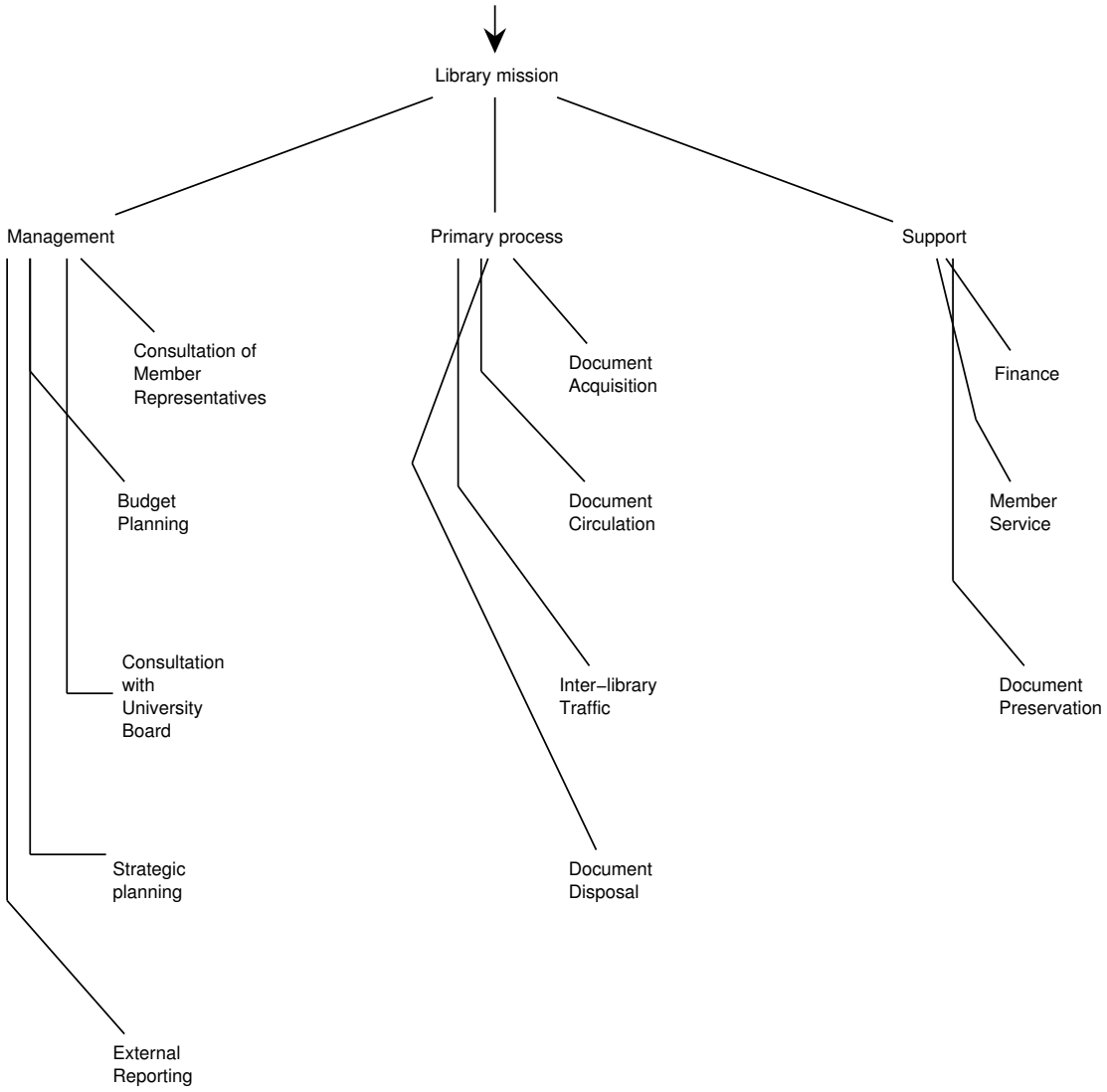


Figure 8.2: Example tree in edit mode.

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.

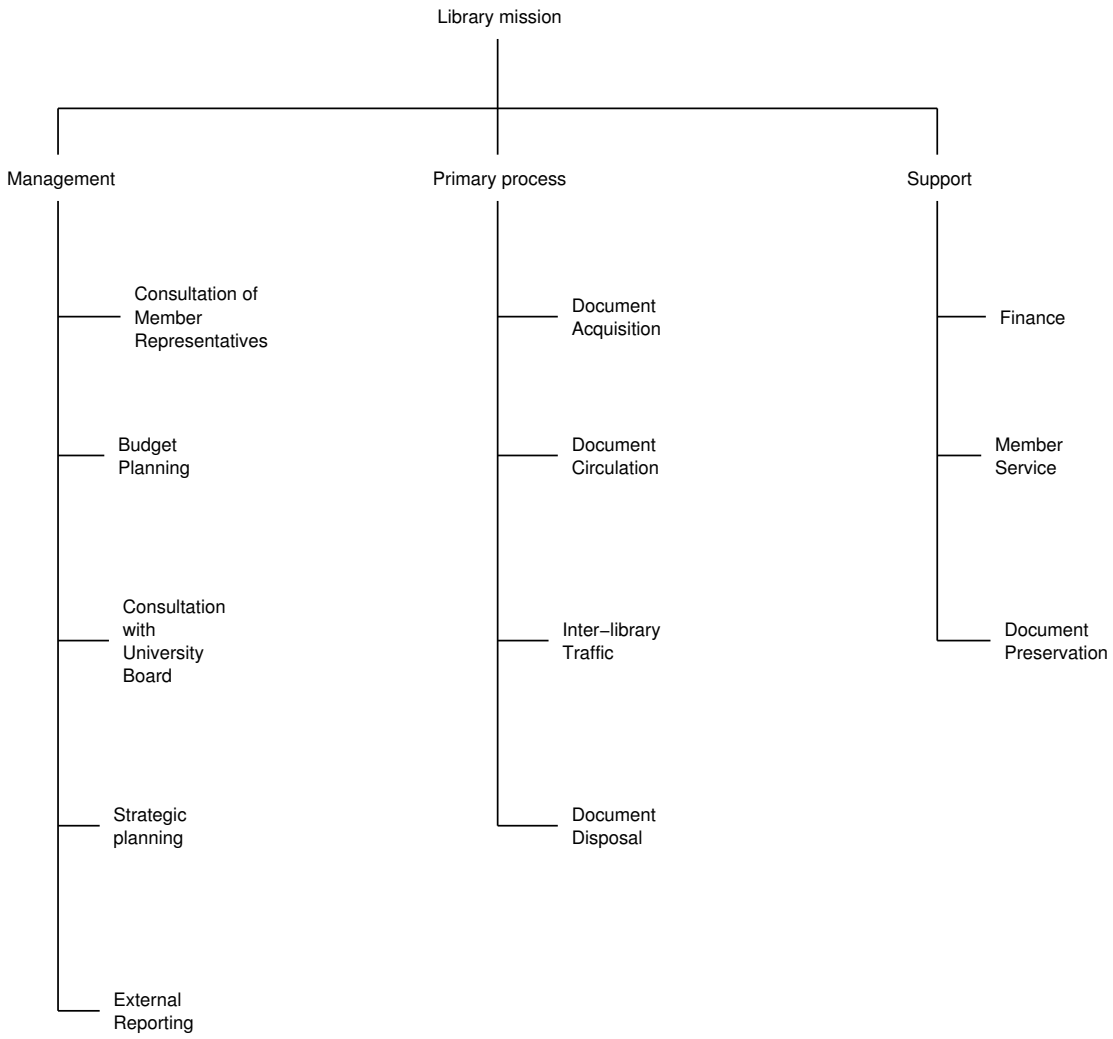


Figure 8.3: Example tree in forked tree (view) mode.

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 [22]. It has a soft constraint that there is a single root node and that all nodes are reachable from that root node.

Appendix A

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 on the notations of structured analysis and the UML is given in R.J. Wieringa, *Design Methods for reactive Systems: Yourdon, Statemate and the UML*, Department of Computer Science, University of Twente, 1999. The miscellaneous notations are documented in [22].

A.1 Structured Analysis Notations

A.1.1 Entity-Relationship Diagrams (TESD)

The TCM convention for TESD is described in detail in [23].

Entity types

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

Binary relationships

Binary relationships are presented by lines.

Cardinality properties

Cardinality properties are represented by annotations placed at the end points of these lines. (Cardinality properties are also called "cardinality constraints" by many authors.) For example, in figure A.1,

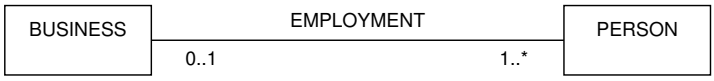


Figure A.1: The placement of Cardinality constraints.

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 A.2 gives an example.

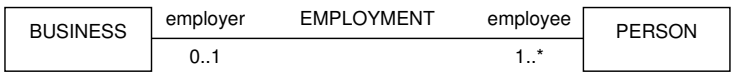


Figure A.2: The placement of role names.

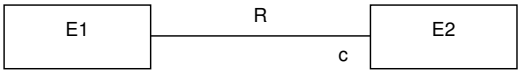


Figure A.3: The meaning of cardinality properties.

In general, a cardinality property is represented by a set of natural numbers (see figure 4.9 for the syntax). For example, if c is a set of natural numbers, the property in figure A.3 is that each instance of E1 is related to n instances of E2, where $n \in$ c. (More precisely, each *existing* instance of E1 is related to n *existing* instances of E2.) If no cardinality property is shown, the convention is that c is the entire set of natural numbers. For example, in figure A.25, 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.

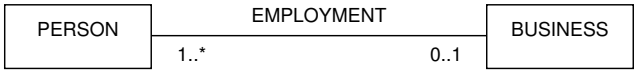


Figure A.4: The line representation of binary relationships is direction-independent.

Note that there is no natural reading direction for a relationship name. For example, figure A.4 conveys the same information as figure A.1. If there is a reading direction, one can adorn the relationship name with a small arrow that indicates this. See figure A.5. Often, a directed relationship name is really a role name of one of the participating entity types.

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

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

Figure A.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

A relationship is a Cartesian product of two or more entity types, called its *components*. (To be more precise, it is a *labeled* Cartesian product.) Relationships can always be represented by a diamond, connected by lines to the boxes that represent its components. These lines actually represent the projection functions of a Cartesian product on its components. For example, figure A.8 contains exactly the same information as figure A.1.

Relationships with arity higher than 2 cannot be represented by a line. They can only be represented by a diamond. Figure A.9 gives an example. The figure also illustrates the notation for a cardinality property of a relationship with arity higher than 2. A cardinality property is expressed by

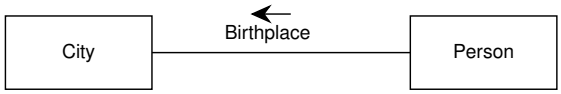


Figure A.5: Reading direction of a relationship name.

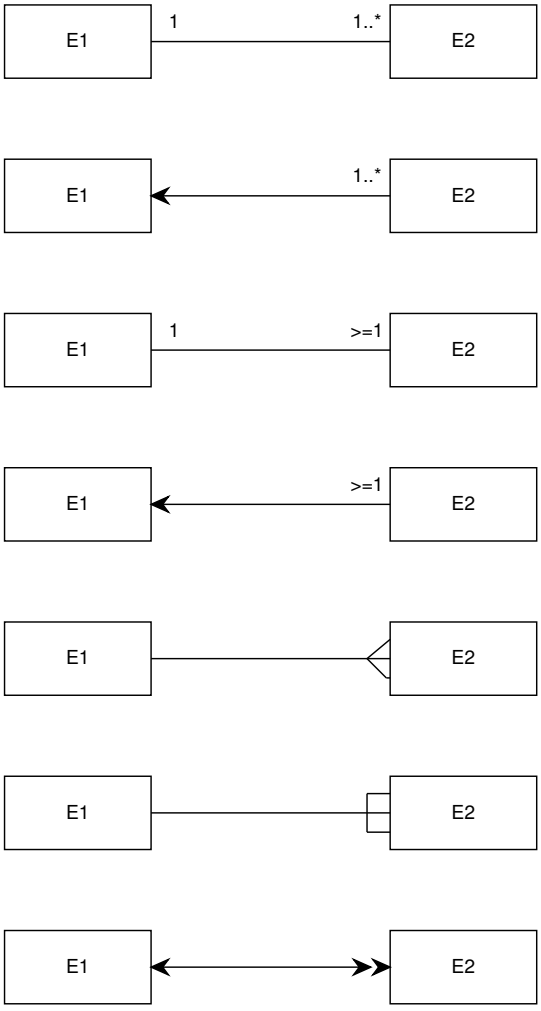


Figure A.6: Different conventions for representing the same constraints. TESD supports the convention used in the top diagram.

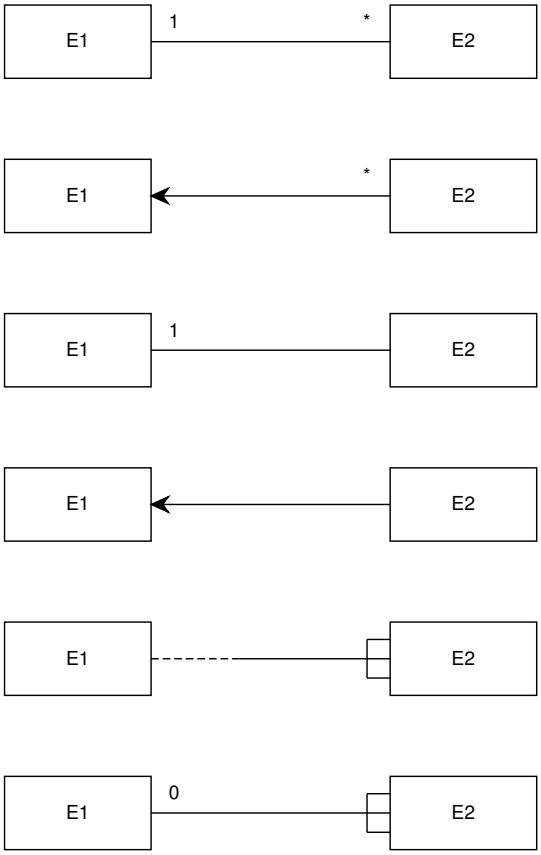


Figure A.7: Different conventions for representing the same constraints. TESD supports the convention used in the top diagram.

Figure A.8: The diamond representation for relationships.

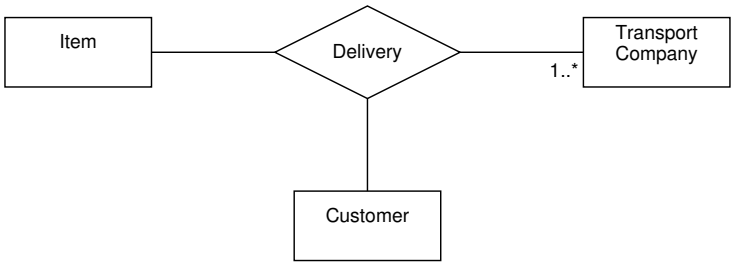


Figure A.9: A ternary relationship with a cardinality property.

an expression c written at the end of a line, close to an entity type box. It represents the number of instances of that entity that participate in the relationship simultaneously. The property in figure A.9 says that each transport company participates in at least one delivery. (This is not very realistic but is does illustrate the convention.)

Attributes

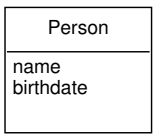


Figure A.10: Representation of attributes.

Entity attributes are represented by listing them in a separate compartment below the entity type name. Representation of entity attributes is optional.

Associative entities

If a relationship itself has attributes, it is represented by an entity box that contains the relationship name and the attribute declarations, connected to the relationship line or relationship diamond with a dashed line. See figures A.11 and A.12 for illustrations.

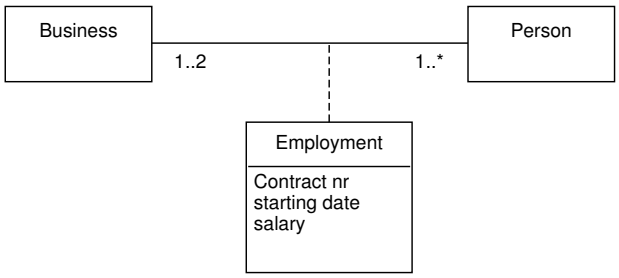


Figure A.11: Representation of associative entities (line representation).

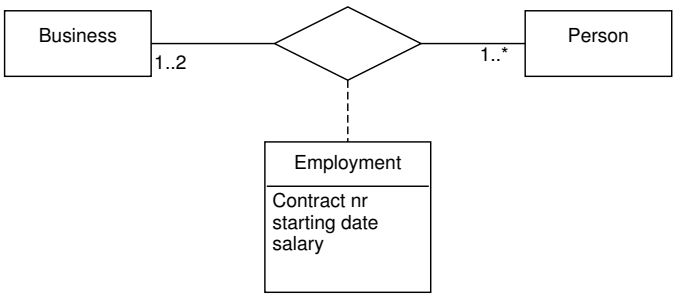


Figure A.12: Representation of associative entities (diamond representation).

Is-a relationships

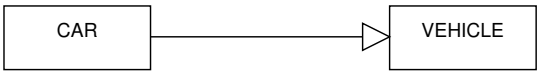


Figure A.13: The representation of is-a relationships.

An is-a relationship is a binary relationship that is an inclusion function. For example, figure A.13 shows that each CAR instance is also a VEHICLE instance. Extensionally, the set of all possible cars 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* or *generalization node* and connecting this with an is-a arrow to the rectangle representing the generalization. The generalization node must be annotated as follows:

- A "d" means that the specializations are mutually disjoint.
- An "c" means that the specializations jointly covers the generalization.
- A "dc" means the conjunction of "d" and "c", i.e. the specializations partitions the generalization.

A generalization can be specialized by any number of specialization groups. For example, figure 4.10 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.

A.1.2 Data and Event Flow Diagrams (TEFD)

Data flow diagrams (DFDs) are are available in two TCM editors, called TDFD (one of the miscellaneous editors) and TEFD (one of the structured analysis editors). TEFD allows you to do everything that TDFD can, and it additionally allows you to draw control processes, event flows and to distinguish time-discrete from time-continuous flows. This section explains both editors. DFDs are described in detail in [23].

The components of a DFD

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. There are two kinds of processes: Data processes and control processes (the latter are not supported in TDFD). TEFD supports both processes.

- 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.

Control processes

DEFDs extend DFDs with a new kind of node, the control process, and new kinds of edges: event flows and time-continuous flows. See subsection A.3.3 for DFDs. A control process is represented by a dashed circle and represents an aspect of behavior. It must be specified by means of a 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 A.14 contains a DEFD of which the control process is specified in figure A.15.

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 [31].

Time-Discrete and time-continuous flows

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

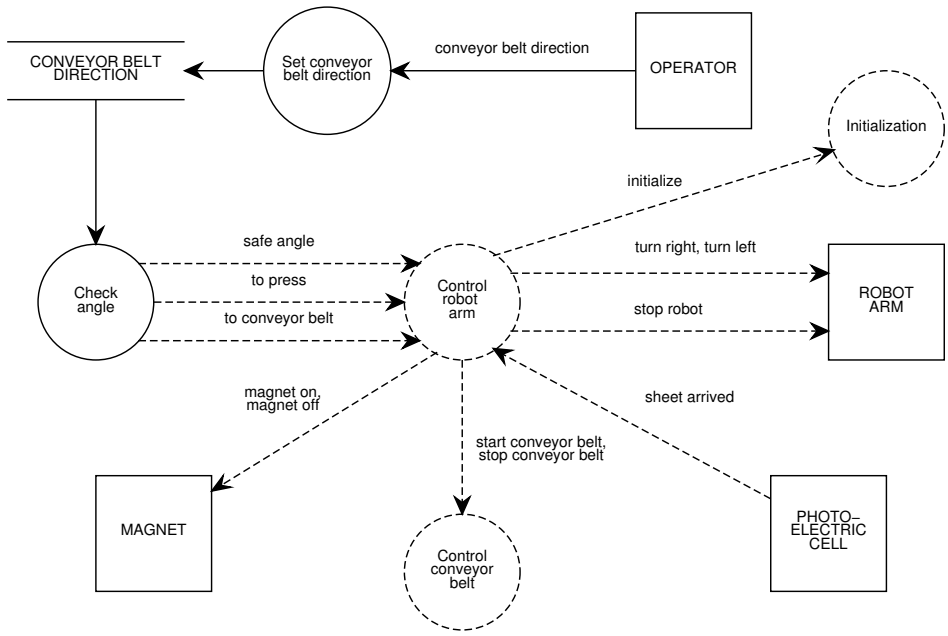


Figure A.14: A DEFD for a robot control process.

A.1.3 State Transition Diagrams (TSTD)

State transition diagrams (Mealy, Moore and statechart) are described in [23]

TCM supports the Mealy notation for finite state transition diagrams (figure A.16). States are named, and are represented by rectangles. State transitions are represented by arrows and are labeled by event [guard] / action pairs. The event is the *trigger* of the transition and can be viewed as the occurrence of an input. The guard is a condition. 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.

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. Figure A.17 shows the a Mealy diagram for a simple coffee machine in which at two points, an external process is triggered (the actions that start with T:) that must send the Mealy machine an answer. While waiting for an answer, the machine is in the decision point.

Mealy machines are used in Yourdon-style structured analysis, where they are used to specify control processes [31]. The interface of the control process must equal the interface of the Mealy machine. See section A.1.2 for control processes.

A.1.4 Transaction-Use Tables (TTUT)

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,

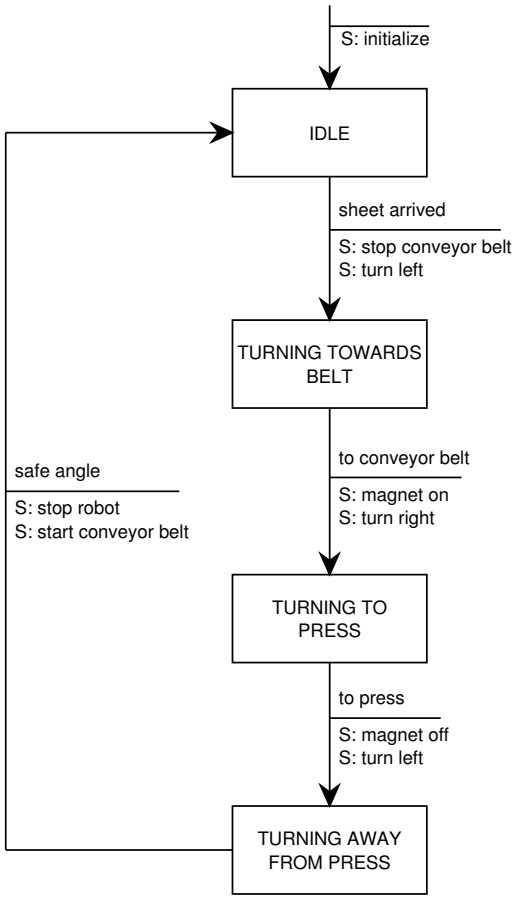


Figure A.15: STD for the robot control process of figure A.14.

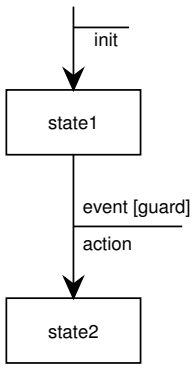


Figure A.16: The Mealy representation of state transition diagrams.

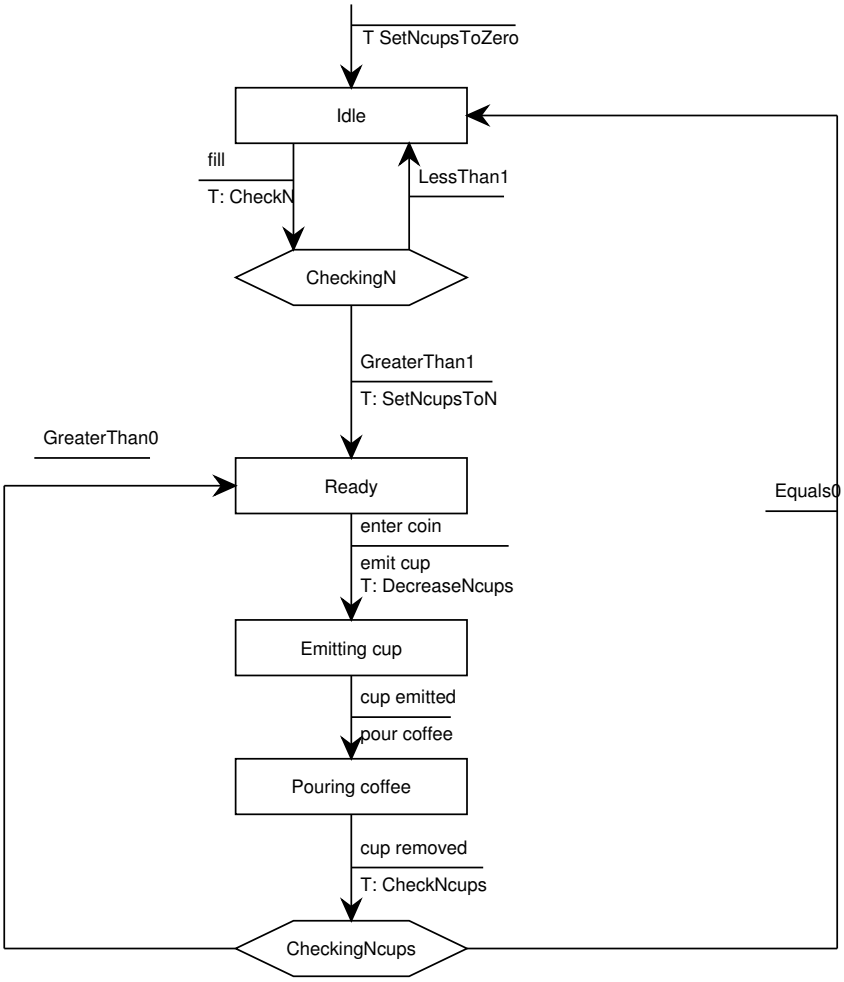


Figure A.17: State transition diagrams.

Update and Delete actions. The entries list the entity types or relationships that are created, read, updated or deleted during the function. See figure 7.8. Elaborate examples are given elsewhere [22].

A.1.5 Function-Entity Type Tables (TFET)

The top row of a function-entity table lists system functions and the leftmost column represents, for example, entity types. The entries contain C, R, U or D, to indicate that this function Creates, Reads, Updates or Deletes entities of this type. Instead of entity types, the leftmost column may list relationships, or subject areas, or data stores in a DFD, with corresponding changes in the meaning of the CRUD entries.

A function-entity type table is a kind of traceability table (see [23]). It is almost the same as a transaction decomposition table (see section A.3.7). 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 [22] for details and examples of their use in Information Engineering.

A.1.6 Function Refinement Trees (TFRT)

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.

A 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 [22]. Of course, a tree can be used to represent any hierarchical decomposition and TCM imposes no constraints on the syntax of the tree.

A.2 UML Notations

This section lists the UML notations available in TCM, as they are treated in [23]. This is a subset of the full UML notation.

A.2.1 Use case diagrams (TUCD)

A use case is a functionality of a system, and actor is a user (person or device) of the system. A use case diagram is a graph in which the nodes represent actors and use cases, and the lines represent connections between use cases and actors. The meaning of a line is that the actor is involved in a use case.

A use case diagram is actually a special case of a class diagram with two special kinds of nodes. Nodes of the same kind can be connected by a generalization arrow.

Actors

An actor is represented by a match stick figure or by a rectangle labeled ≪actor≫. Both shapes can be labeled by an actor name. Two actors can be connected by a generalization arrow. Actor names must be unique. Use “duplicate node” if you want to represent one actor several times in the diagram.

Use cases

A use case is represented by an ellipse. It can be labeled. Two use cases can be connected by a generalization arrow.

A.2.2 Static structure diagrams (TSSD)

A static structure diagram is an extension of an ER diagram. What is an entity in an ESD is an object in a SSD. The extensions of TSSD with respect to TESD are the possibility to declare the behavior of an object and to represent instances. There is a change in terminology when we change from ESDs to SSDs:

Entity-relationship diagram	UML static structure diagram
entity type	class
entity	object
relationship	association
tuple	link
associative entity	associative object
cardinality property	multiplicity property

Stereotypes and properties

We can give a diagram element a special meaning by labeling it with a special name between «guillemets». Such a diagram element is called a **stereotype**. For example, in a static structure diagram, we can specialize classes into stereotypes with a special meaning, by writing the stereotype name in guillemets above the class name. See figure A.18.

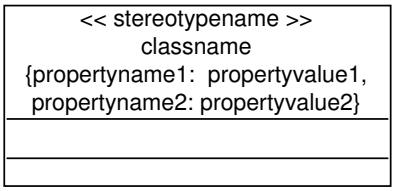


Figure A.18: Stereotypes and properties.

Figure A.18 also shows that we can annotate the name compartment of a class box with properties. A property is represented by a user-defined property name and a property value. This is included in the name compartment as a comment between curly braces, and it has no UML-defined semantics. You are free to include any text between curly braces.

Behavior

The behavior of an object is declared in a third compartment below the attribute compartment of a class box. The UML allows the declaration of the operations that the instances of the class can perform and of the signals that the instances can receive. See [23] for details.

Objects

An object is represented by a named rectangle with an attribute compartment. The name of an instance is underlined. The attribute compartment contains the attribute values of the instance. See figure A.19. Notice the association from John to the class City. This tells us that John has exactly one birthplace but it does not tell us which one this is, because City is a class.

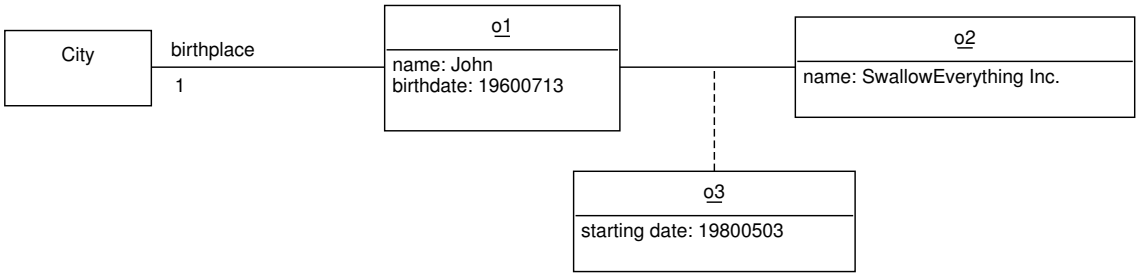


Figure A.19: Representation of objects.

A.2.3 Activity diagrams (TATD)

An activity diagram is a graph in which the nodes represent activities and the arrows represent transitions between activities. Figure A.20 gives an example.

Activity

Activities are represented by two parallel lines connected by semicircles. The name of the activity can be entered in the shape.

Transition

Transitions are represented by unlabeled arrows. A transition represents the completion of the activity from which it departs.

Choice nodes

A choice node is represented by a diamond. A transition that emanates from a diamond can be labeled by a [condition] that tells us when this branch is taken. A choice point is not a state of the system.

Fork and join nodes

Fork and join nodes are represented by fat horizontal or vertical lines. If more than one arrow leaves the node, it is a fork node and there must be exactly one arrow entering it. A join node represents the start of two or more parallel processes.

If more than one arrow terminates at the node, it is a join node and there must be exactly one arrow that departs from it. A join node represents the merging of two or more parallel process into one process.

Initial and final state

The start of an activity diagram is represented by a bullet. There must be exactly one bullet in a completed diagram.

A final state of an activity is represented by a bull's eye. There must be at least one final state in an activity completed diagram.

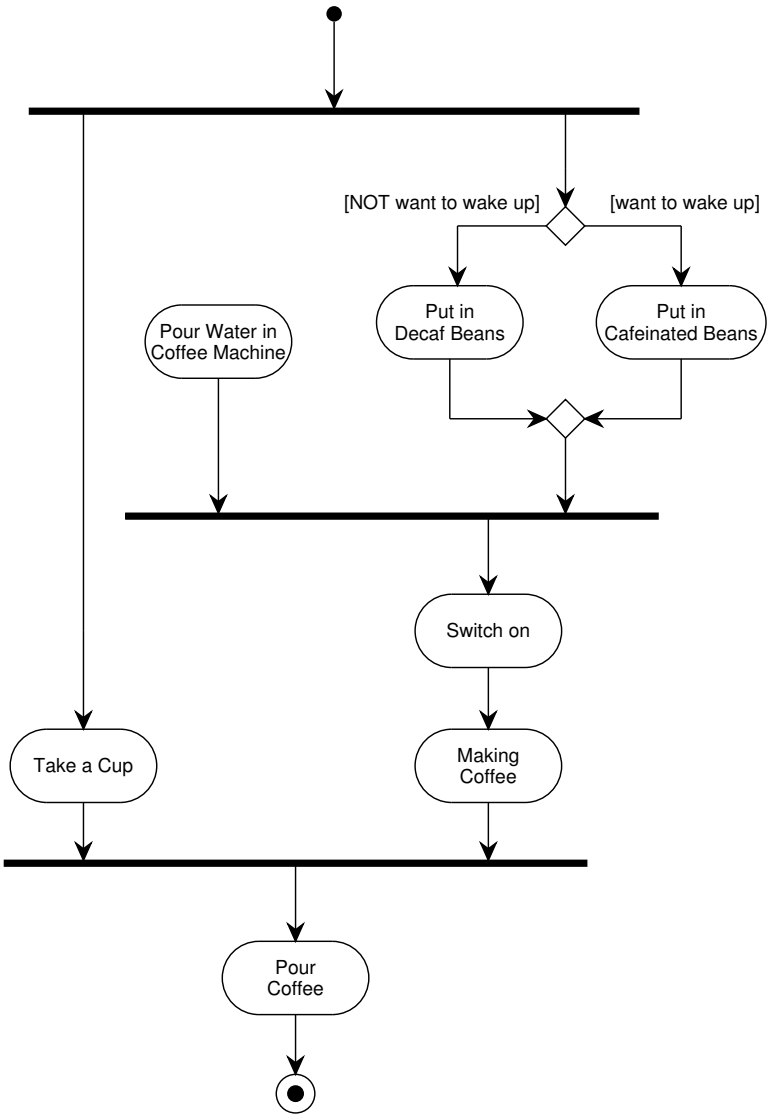


Figure A.20: An activity diagram.

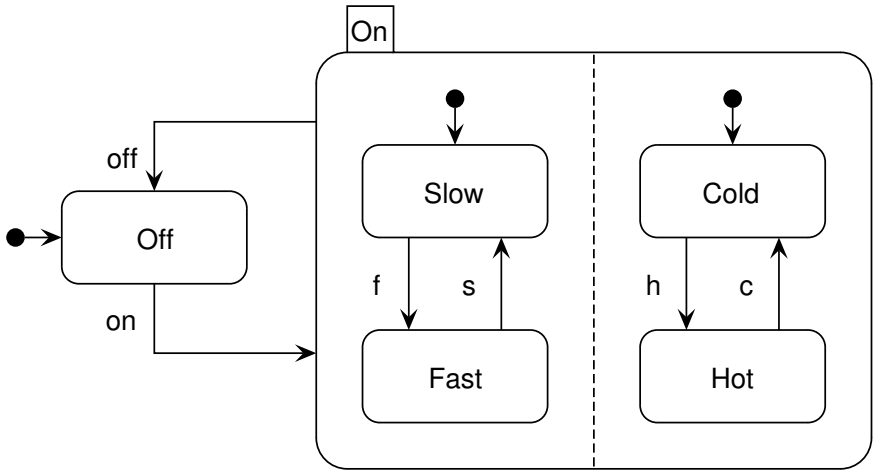


Figure A.21: Example of a statechart.

A.2.4 Statechart diagrams (TSCD)

TSCD is used to draw *statecharts.* Statecharts are based on state–transition diagrams known from TSTD.

A statechart describes the behaviour of a system, i. e., the possible orders of events and states. A state in a statechart may consist of one or several *state nodes.* A state node can be refined in two ways:

An or node serves to describe substates. If the state of the system contains the or node, it contains exactly one of the subnodes.

The subnodes of an or node are simply drawn inside the or node. The initial subnode has an arrow starting at a black dot.

An and node serves to describe parallel behaviour. If the state of the system contains the and node, it contains all of the subnodes. These subnodes are typically refined further, to describe in which state the single parallel components can stay.

The subnodes of an and node are drawn as compartments which partition the and node. As there is no space left for the and node's name, it is attached to a small box on the outside.

Possible state changes are indicated by *transitions.* They are drawn as arrows with a label e[g]/a, where e denotes the event which triggers the transition, g is a guard (the transition can only be taken if the guard holds), and a denotes the action executed when the transition is taken (for example, send an event to another statechart).

In addition to these basic elements, one can indicate the initial state with an arrow from a black dot and the final state with a bull's eye.

For an example of a statechart, see figure A.21. The statechart describes a fan's behaviour. This kind of fan can produce a cold or hot, and a slow or fast air stream. Its initial state is Off. When switched on, it enters the and node On and its subnodes. Here, it again selects the initial nodes Slow and Cold. If the user sends event f to the system, it switches to Fast. When the user switches the fan off (by sending an off event), the fan leaves the On node and all its subnodes (forgetting the slow/fast and cold/hot settings), and enters the Off node again.

A.2.5 Collaboration diagrams (TCBD)

A collaboration diagram is an object diagram that shows the objects and links involved in a scenario, and also shows the messages passed in the scenario.

Messages

In addition to other UML diagrams a collaboration diagram has message flows representing messages being sent between objects via links. See figure A.22 for an example of the initial dialog between a client and a ATM.

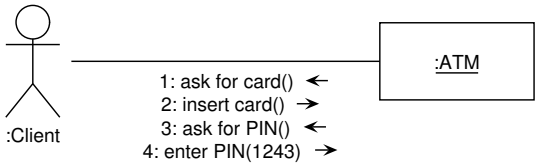


Figure A.22: Collaboration diagram messages.

A.2.6 Component diagrams (TCPD)

A UML component diagram is a directed graph in which the nodes represent components and the edges, which are directed, represent dependencies. A component is, roughly, any software-like resource delivered during software development or needed by the delivered software. This includes the executables and sources of the software system, utilities needed by the software system, shared libraries, etc. A dependency may be a compilation dependency, and import dependency, etc. The exact meaning of the nodes and edges must be described in the diagram documentation.

The interface of an executable component is represented by small rectangles protruding from the component box.

Each class in the class model must be allocated to an executable component. This can be represented in a component diagram by enclosing a class icon inside a component icon. Alternatively, it can be represented by drawing a dependency arrow from the class to the component(s) it is allocated to.

A.2.7 Deployment diagrams (TDPD)

A UML deployment diagram is a graph in which the nodes represent resources and the edges represent communication channels. A resource is a hardware/software combination that offers computing power. This includes mainframes, servers, workstations, PC's, laptops, handheld computers, organizers, mobile telephones, faxes, printers, etc. A channel is any hardware/software combination that offers communication possibility to resources. This includes local and wide area networks, wireless communications, cables, etc.

Each executable component can be allocated to one or more resources. This can be represented in a UML deployment diagram by drawing a component icon inside a resource icon. Alternatively, it can be represented by drawing a dependency arrow from the component to the resource(s) it is allocated to.

A.3 Miscellaneous Notations

A.3.1 Classic Entity-Relationship Diagrams (TERD)

The TCM convention for ERDs is described in detail in [22].

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

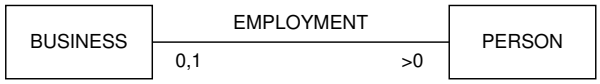


Figure A.23: The placement of cardinality constraints.

example, in figure A.23, 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 A.24 gives an example.

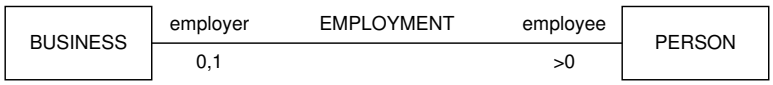


Figure A.24: The placement of role names.

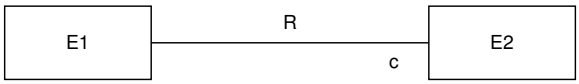


Figure A.25: The meaning of cardinality constraints.

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 A.25 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 A.25, 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.

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

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

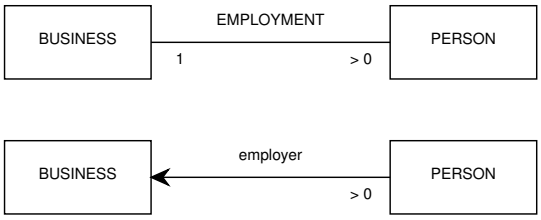


Figure A.26: The arrow representation of many-one constraints.

cardinality constraint of 1 can be abbreviated by an arrowhead. So the two diagrams in figure A.26 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.

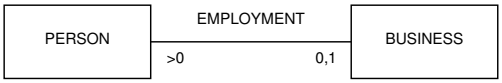


Figure A.27: The line representation of binary relationships is direction-independent.

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 A.27 conveys the same information as figure A.23. 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.

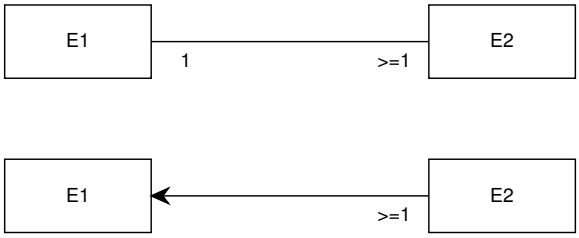


Figure A.28: Different conventions supported by the classic TERD editor for representing the same constraints.

There are many other conventions to represent binary relationships. Figure A.28 shows different ways of representing the following constraints:

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

Figure A.29 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.

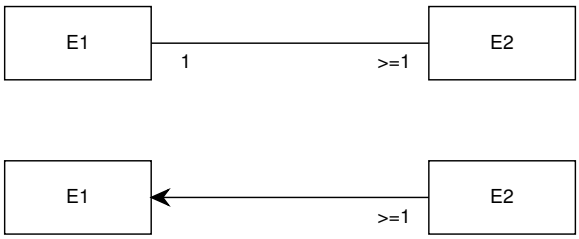


Figure A.29: Different conventions supported by the classic TERD editor for representing the same constraints.

Relationships of higher arity

Figure A.30: 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 A.30 contains exactly the same information as figure A.23. 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 A.30 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 A.23.

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

An is-a relationship is a binary relationship that is an inclusion function. For example, figure A.31 shows that each CAR instance is also a VEHICLE instance. Extensionally, the set of all possible cars 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.

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

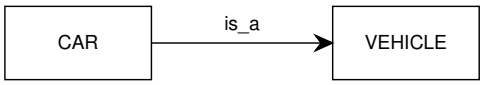


Figure A.31: The representation of is-a relationships.

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.

A.3.2 Class-Relationship Diagrams (TCRD)

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

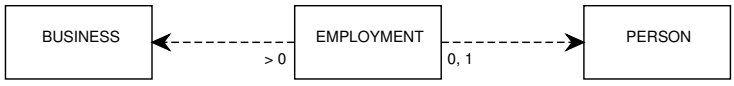


Figure A.32: The CRD representation of relationships.

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 A.32 has exactly the same information content as figures A.23, A.27 and A.30. The line representation (figure A.23) 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,

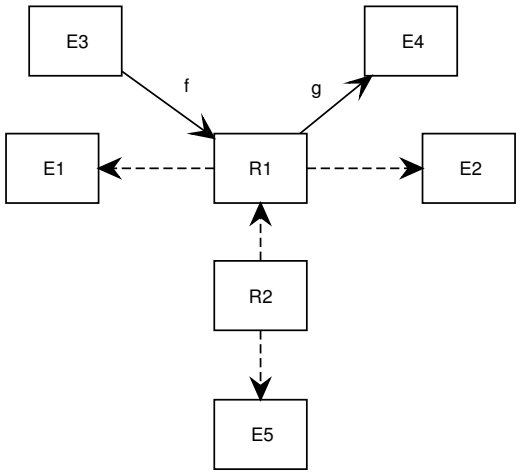


Figure A.33: The CRD convention can represent complex mathematical structures.

the CRD convention used in TCM allows representation of such complex structures as represented in figure A.33, which cannot be represented in the ERD convention. Figure A.33 represents the following structures. (To reduce clutter in the notation, we ignore the fact that relationships are actually *labeled* Cartesian products.)

- $\mathsf{R1} = \mathsf{E1} \times \mathsf{E2}$
- $\mathsf{f} : \mathsf{E3} \to \mathsf{R1}$
- $\mathsf{g} : \mathsf{R1} \to \mathsf{E4}$
- $\mathsf{R2} = \mathsf{R1} \times \mathsf{E5}$

Is-a relationships

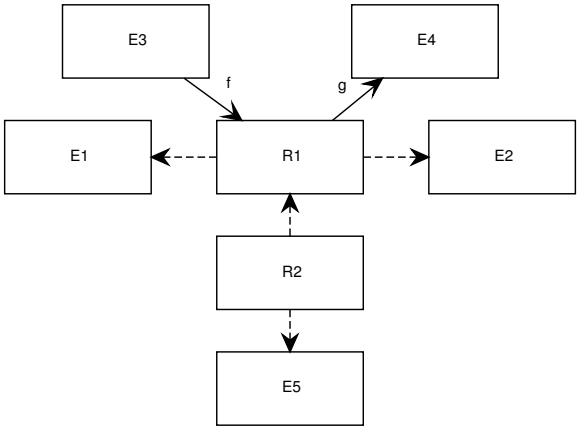


Figure A.34: Static specialization.

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 4.12). In figure A.34, 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 4.16, 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 [24, 25].

A.3.3 Data Flow Diagrams (TDFD)

TDFD contains a subset of TEFD. Please see section A.1.2.

A.3.4 Process Structure Diagrams (TPSD)

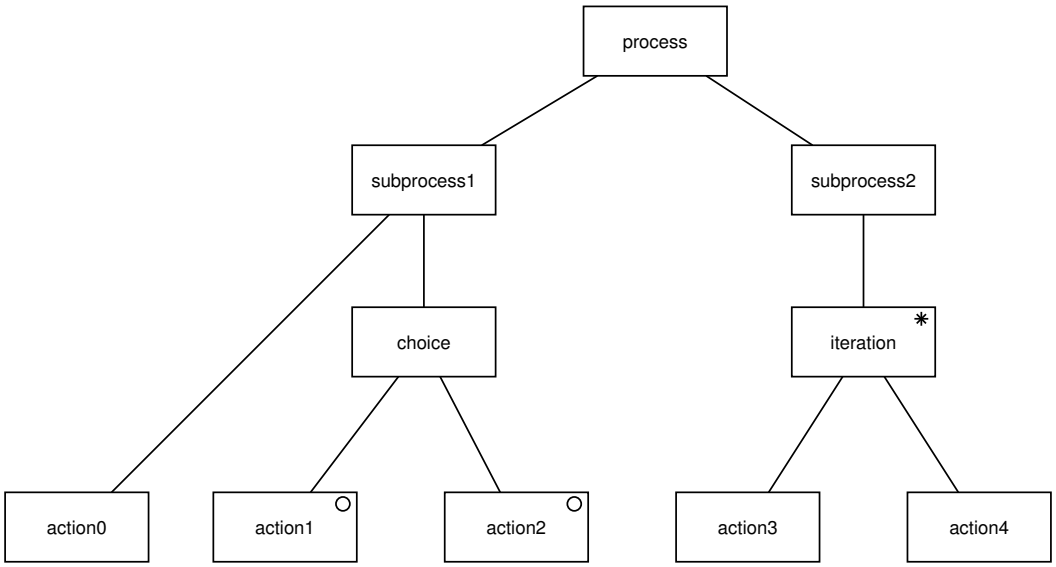


Figure A.35: A process structure diagram.

Process structure diagrams are used in JSD to represent behavior. A PSD is a tree in which the nodes are labeled [11, 22]. 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 of a node. Iteration is represented by an asterisk label and choice by a small circle in the nodes that represent the options. Figure A.35 gives an example.

PSDs are equivalent to regular expressions.

A Mealy machine roughly equivalent to this is shown in figure A.36. 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 A.36 we arbitrarily categorized all PSD actions as output actions.

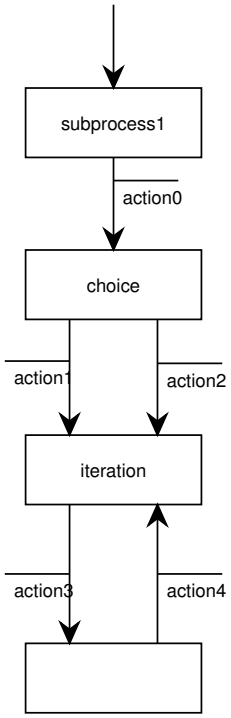


Figure A.36: A Mealy diagram roughly equivalent to figure A.35.

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 A.3.5 below.

A.3.5 System Network Diagrams (TSND)

SNDs are used by JSD [11] 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 a DEFD must be specified by a STD. There are three kinds of communication nodes:

- *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.

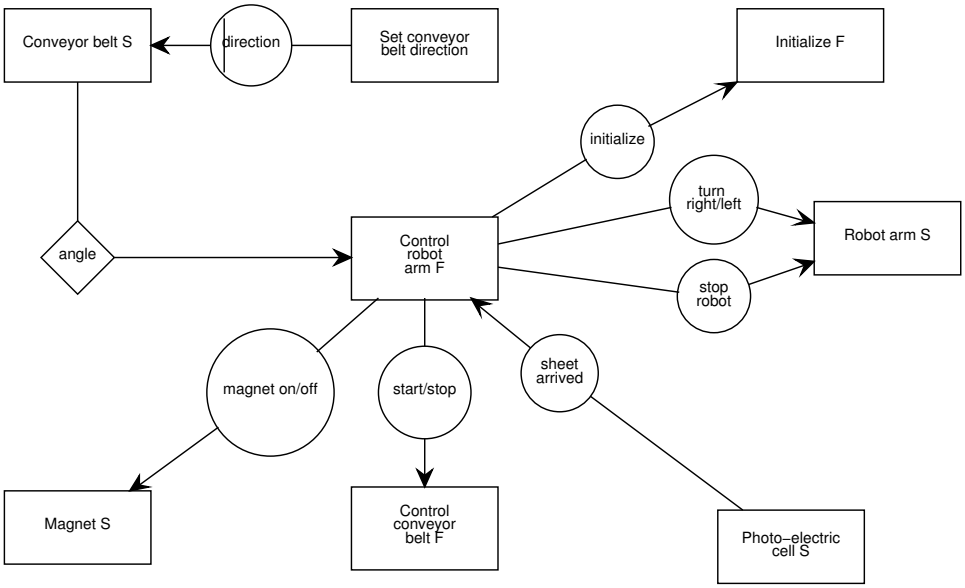


Figure A.37: An SND of the robot controller of figure A.14.

Figure A.37 shows a SND of the robot controller of figure A.14. 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.

A.3.6 Recursive Process Graphs (TRPG)

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 A.38 shows a RPG equivalent to the PSD of figure A.35. Nodes in RPGs can be labeled, just as in Mealy STDs. Figure A.39 shows a 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 a 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 A.40 illustrates this. The RPG in figure A.40 is equivalent to that of figure A.39.

The call to another process can be recursive, as illustrated in figure A.41. This describes the process with possible traces $\mathsf{a}^n\mathsf{c}$ for $n \geq 1$.

Recursive process graphs are defined formally by Spruit and Wieringa [26], based upon the idea of recursive transition networks [27].

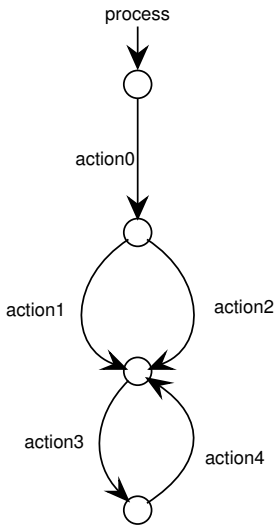


Figure A.38: A recursive process graph.

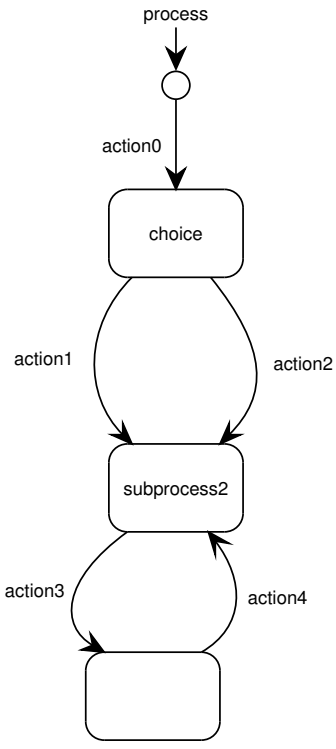


Figure A.39: A recursive process graph with labeled nodes.

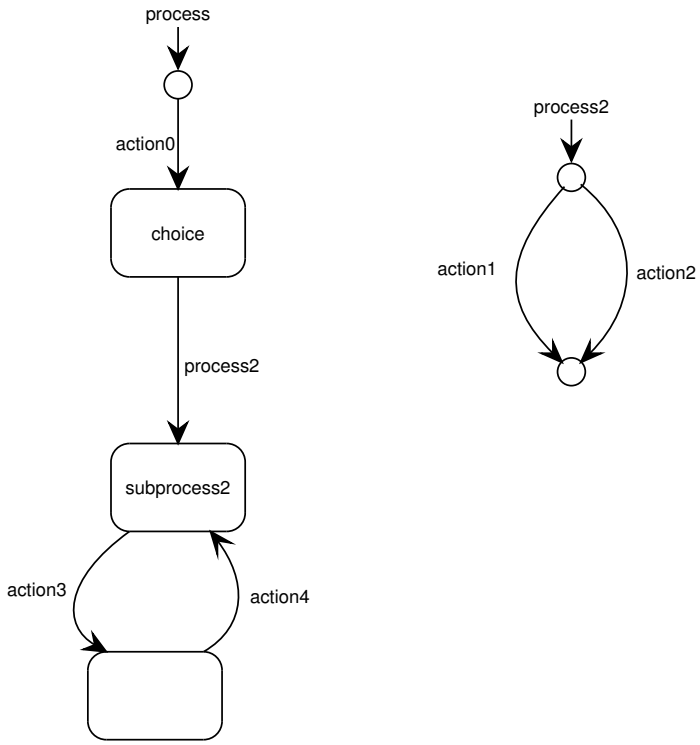


Figure A.40: A recursive process graph with a call to another process.

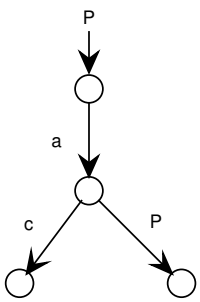


Figure A.41: A recursive process graph with a recursive call.

A.3.7 Transaction Decomposition Tables (TTDT)

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 7.7 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 [22, 21].

Appendix B

Frequently Asked Questions

Like most lists of frequently asked questions, this FAQ contains a compilation of questions and answers. The questions are either asked in the past or we expect that many users might want to ask these questions. For an overview and explanation of the TCM (environment) variables like `$TCM_HOME` please see section 1.4 of this manual.

B.1 What is TCM?

TCM stands for the Toolkit for Conceptual Modeling, made by the Vrije Universiteit Amsterdam and the University of Twente, The Netherlands. It is a suite of graphical editors that is intended to be used for software specifications. TCM is also found to be useful by its users for drawing generic diagrams or tables, especially on the Linux platform.

The acronym TCM stands also for a number of other things that have nothing to do with our TCM, such as Trellis Coded Modulation (encryption theory), The Computer Museum (www.tcm.org), Traditional Chinese Medicine and the Texas Chain-saw Massacre (a horror movie).

B.2 Where can I get TCM?

TCM binaries, sources and documentation are uploaded to the site ftp://ftp.cs.utwente.nl/pub/tcm. This site is mirrored at ftp://ftp.cs.vu.nl/pub/tcm. As of version 2.0, source code distributions are available too. The binaries that are available include binaries for Sun Solaris 2.x (Sparc and x86), Linux 2.x (x86, libc), IRIX 5.x (SGI), AIX 4.x (RS6000), HP-UX 10.x (HP9000) and OSF/1 (Dec Alpha). All distributions can be downloaded freely from the TCM ftp-site and they are in ordinary tar.gz files and for the Linux platform we made RPM distributions. You can download the software also via the web page www.cs.utwente.nl/~tcm/software.html.

B.3 How do I install TCM?

The installation instructions can be read in section 1.4 of this manual. The INSTALL file in the distributions and the web page www.cs.utwente.nl/~tcm/software.html contain installation instructions too.

B.4 How do I start up TCM?

You start TCM with the Unix command `tcm`, or you can start up individual editors like we have explained in section 1.4. If starting up TCM does not succeed or when TCM starts up and crashes immediately then it is probably one of the following problems:

1. 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_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 are not found or cannot be opened, e.g. “libXm.so: can’t open file” then proceed with the answer of question B.21.

5. If TCM immediately crashes with a message like “X Error of failed request” then proceed with the answer of question B.20.

6. If TCM does start up but it displays some dubious error messages about failed assertions or implementation errors then proceed with the answer of question B.39.

7. When you work under Linux, make sure that your system has the glibc (also known as libc6) libraries version 2.1 installed. TCM does not work with the older libc5 libraries nor with glibc 2.0. Also the X libraries should have been compiled with glibc 2.1.

8. If all this does not help, you can always send a description of your problems in an e-mail to the mailing list tcm-users@cs.utwente.nl. See question B.13 for how to subscribe to this list.

B.5 Where can I find the user manual of TCM?

All TCM distributions contain a copy of the user manual. The file
`TCM_DOC/usersguide-<version>.ps.gz` is the user manual in gzipped PostScript format. An HTML version can be read via `TCM_DOC/usersguide/index.html` You can also download the user manual separately from our FTP-site ftp://ftp.cs.utwente.nl/pub/tcm. There you can find copies both in gzipped PostScript as in PDF-format.

The web-page www.cs.utwente.nl/~tcm/usersguide/index.html is the user manual on the WWW.

B.6 What else can I read about the methods supported in TCM?

If you want to know only a bit more about the methods supported in TCM, you can read the mini-tutorial in appendix A.

Roel Wieringa has written a book [22] that treats a number of methods that are supported by TCM in more depth (Entity-Relationship modeling, Structured Analysis and JSD).

From www.cs.utwente.nl/~roelw/RE1.ps you can download the table of contents of that book.

The home page of Roel Wieringa, www.cs.utwente.nl/~roelw contains links to a bibliography of his reports and articles and some of these can be downloaded in PostScript format.

When you look for the official UML documentation, take a look at www.omg.org/uml on the website of the Object Management Group. When you are new to conceptual modeling and you want to read a book that gives a brief introduction to UML, you can try [9]. The Unified Modeling Reference Manual [1] is much more extensive however and is intended for people who are already familiar with some modeling method.

B.7 Is TCM available for my platform?

In principle, TCM should be able to compile and run on all Unix systems that have X Windows and Motif. Before compiling or porting TCM on your system, please read the developer's guide [4] that can also be found as a PostScript and PDF document in the `doc` subdirectory of the TCM source code distribution. For compiling the sources you need a C++ compiler (g++ is the best choice) and you need Motif, LessTif or Open Motif (preferred).

At the moment distributions have been made for Solaris 2.x (Sparc and x86), SunOS 4.x.x, Linux 2.0 (x86, ELF), IRIX 5.x (SGI), AIX 4.x (RS6000), HP-UX 10.x and OSF/1 (or Digital Unix) and even on Windows, running the CYGWIN/XFree86 environment. TCM should run under FreeBSD and Darwin (Mac OS X) too. See the file README or the web-page www.cs.utwente.nl/~tcm/software.html for the distributions currently available. We try to keep the Solaris and Linux distributions and the source code distributions up-to-date but for the other platforms it is possible that only a somewhat older version exists.

B.8 What to do if I don't 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 we have made different Linux and 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 `-statmotif` in their file names. The distributions that dynamically link to the Motif library have the string `-dynmotif` in their file names.

Motif itself is copyrighted so we are not allowed to put the Motif library itself (libXm.so) in the distributions. www.rahul.net/kenton/faqs/mfaq_index.html contains the Motif FAQ including where you can get Motif. Instead of purchasing Motif, you can use the free Motif clone LessTif (www.lesstif.org) or preferably Open Motif (www.opengroup.org/openmotif). We've experienced "flakiness" with TCM compiled with Lesstif, so please try to use Open Motif Open Motif if at all possible.

It is possible when you compile the TCM sources to link it with Open Motif/LessTif or to use Open Motif/Lesstif when you have installed one of our 'dynmotif' distributions.

B.9 Can I obtain the source code of TCM?

Yes of course, as of version 1.94, TCM is distributed under the GNU public license. That means that sources should be made available too. The most recent sources are available as tar.gz file in ftp://ftp.cs.utwente.nl/pub/tcm.

B.10 How can I compile or port TCM for 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 (see also question B.7. Porting TCM merely consists of editing a configuration file (and maybe one or two Makefiles) and possibly a handful of conditionally compiled source code lines has to be added or changed, because some Unix calls might be different or because the compiler is somewhat idiosyncratic. Before compiling, please read the developer's guide [4].

If you don't succeed in compiling TCM (providing it is Unix with X/Motif), you can ask for help via our mailing list tcm-users@cs.utwente.nl. If you have successfully built an executable that is not available via our ftp-site you can take contact with us about putting this distribution on our site and about the incorporation of your Makefiles and source code changes in future versions of TCM.

B.11 What are the terms of usage?

TCM version 1.94. and higher are distributed under the GNU license. This license can be read in the file ftp://ftp.cs.utwente.nl/pub/tcm/COPYING.

B.12 In what programming language is TCM written?

This version of TCM is written in C++ and developed under Solaris and Linux. In fact it is programmed in a subset of C++, as C++ contains so many features that we don't need. The same source tree has been compiled on Solaris with the Sun CC compiler (v. SC3.0.1 and SC4.2), on Solaris, Linux, OSF/1 and HP-UX with the GNU g++ compiler (many versions of 2.7.2 and higher), 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, 2.0 and 2.1 all work). However, when TCM is compiled against with one version of Motif it can't be run with another version. Check the file ftp://ftp.cs.utwente.nl/pub/tcm/README for which Motif version is expected for our 'dynmotif' distributions.

B.13 How do I stay up-to-date with the developments of TCM?

Subscribe to the TCM mailing list. Send an empty message to tcm-users-request@cs.utwente.nl for subscription information.

B.14 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

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

is allowed by the Motif copyright. Furthermore, the application environment specification of Motif is free and there exist a public domain Motif clone, called LessTif (www.lesstif.org). You're advised to use Open Motif Open Motif, see question B.8.

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 [28] 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 B.16.

But why is Motif used and not some of the other GUI libraries that are available? Motif gives us a lot of advantages. It is fast and very reliable. Motif is standardized and very well documented. There are a good learning and reference books for Motif such as [28, 29, 2, 10] [2]. The Motif interface is more or less 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 Perl, Python or Tcl/Tk. This means that you have to know what you want to program in advance and to be prepared to spend some time to implement a certain feature.

B.15 Did you use other tools or widget sets to build TCM?

Not extensively. 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, TSSD 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: www.zib.de/Visual/software/doc++/index.html. For testing and debugging TCM, we made use of the program called purify (www.rational.com/products/purify/index.html). Purify instruments a program to detect run-time 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 that pop up in the user interface are implemented with the Lite Clue widget (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.

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

B.16 Does TCM run under Windows, Macintosh, Java etc. ?

As of version 2.20 TCM also runs on Windows systems under the CYGWIN/XFree86 environment. See the `README.cygwin` and the `INSTALL.cygwin` file in the software distribution before you install and use TCM under CYGWIN.

As of version 2.10 TCM should run under FreeBSD and Darwin (Mac OS X).

The TCM project had in the past a sister project called WHERE, which is performed at the NASA Software Research Laboratory (see research.ivv.nasa.gov/projects/WHERE). Part of this project was the translation of TCM into Java (see research.ivv.nasa.gov/projects/WHERE/TCMJava). The software is in alpha stage, but many things work. Because it is written in Java, TCMJava runs on Solaris, Windows NT, Linux and probably also on others.

Via 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 what versions you need.

Our plan is to continue the development of our TCM for X Windows and Unix systems written in C++. We don't have plans to port TCM to systems that do not have X Windows and Motif, for the simple reason that it will cost us too much time. A port of TCM to another GUI library is no quick or easy job. The X11/Motif part consists of over 10000 lines of source code which should be adapted or rewritten.

B.17 How can I configure TCM?

Some behavior of TCM can be tailored with configuration files. `TCM_CONFIG/tcm.conf` is the system-wide configuration file. The options of this file may be overridden by a user's private configuration file in `$HOME/.tcmrc`.

The configuration files contain a number of attribute-value pairs separated by white space and enclosed by curly braces (basically it is the same as the TCM file format in appendix C). Comments in the file start with a hash (`#`). The text in the same line after the hash is ignored. The name of the attributes are case sensitive. In the file itself you find comment text which gives directions for what the attributes mean and what alternative attribute values are possible.

B.18 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_CONFIG/TCM` you find the X resources with the settings that are currently used in TCM (but the file is not used by TCM, the same settings are hard-coded in the sources). You can copy this file and adapt it to your wishes. You can load it with `xrdb -merge` or copy it in the file `.Xresources` or in `.Xdefaults` in your home directory. When you use the `xrdb` command then it must be issued each time when you restart X.

B.19 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. You are probably running in 8-bit (256 color) mode, and have other programs that use colors running on your display. Common culprits are background pictures or patterns, and Netscape, which is a notorious color hog. You can best start up TCM first

because it takes only what it needs, and then start up Netscape and the other color eaters. 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 option `-priv_cmap`, it always uses a private color map. If you want to use TCM by default with a private colormap you can set the option `PrivateColormap` to `True` in the file `TCM_CONFIG/tcm.conf` or in `$HOME/.tcmrc`.

B.20 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 one A3 or 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 TGD with `tgd -drawing 800x600` which gives a drawing area that is 800 pixels wide and 600 pixels high. If you want to have this initial size permanently you can set the options `DrawingWidth` and `DrawingHeight` in `TCM_CONFIG/tcm.conf` or `$HOME/.tcmrc`.

B.21 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_LIB`. If they are, expand your `LD_LIBRARY_PATH` variable with the value of `TCM_LIB`.

- If *BlaBla* is libXm.so then TCM cannot find (the right version of) the Motif library. Check 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 B.8). 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 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.

B.22 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, probably at home or work. TCM editors are supposed to be upward compatible and should read older file formats. As of TCM version 1.99 (file format version 1.30) the TCM editors also attempt to read in newer file formats than the one that they generate; incompatibilities will be shown in the message log dialog window. They always generate one file format. In the entry Document Info in the Document menu you can see what file format the editor generates and what file format you have read. The best solution is to upgrade TCM.

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.

B.23 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.

B.24 How can I include a TCM picture in my text document?

If you work in a LaTeX environment or any other type setter or word processor that accepts PostScript then you can save your document a PostScript. It is best to 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 `graphicx` (for LaTeX 2_ε). With such a package it is probably possible to scale and rotate the included figure (see the package's documentation). All example diagrams and

tables in this user manual were made with TCM and included as encapsulated PostScript with the graphicx package. If you do not want to scale or rotate explicitly, 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 Drawing command in the Scale menu.

For word processors that have no good PostScript support you can export your document to PNG format. PNG is a format that is understood by most webbrowsers and by MS-Word. PNG is similar to the GIF-format but less controversial (see this article and burnallgifs.org).

B.25 Is it possible to create process decompositions in TDFD?

Not at this moment. For each level you have to draw a separate diagram. The Diagram text field in TDFD and TEFD helps sets the prefix of new data and control process indexes. It does not offer much more.

We made a specification of an editor that manipulates leveled data flow diagrams [6]. Our plan is to make an editor in which you can edit an entire data flow model, in particular all diagram levels.

B.26 Are there plans for consistency checks across diagrams?

Yes. The first step was 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 B.28).

B.27 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. It is however possible to program extensions and modifications in the TCM source code.

B.28 Do you have any plans to support specific methods?

Yes. We want to support UML [17, 18] more extensively. UML is object oriented. The UML support is for both educational as research purposes. Also, we want to give specific support for a structured method, in particular for our own variant of the Yourdon Systems Method (YSM) (mainly for educational purposes).

The toolkit support for UML and YSM as representative of the family of object oriented respectively structured methods includes building (and checking) models that are made from diagrams built by different editors.

B.29 Do you have plans for code generation?

Yes and some others are already working on that. For instance, in a prototype someone generated SQL from .esd files (ER-diagrams) and in another one XML is generated from .std files (State-transition diagrams). If you are interested, contact us or see our 'links' page.

It is not very hard to write your own program that generates specifications in whatever language from TCM files. The TCM file format simple, human-readable and it is specified in appendix C.

B.30 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.

B.31 What file formats does TCM generate?

At the moment PostScript, EPSF, PNG, the Fig format and the TCM format are generated. TCM only reads its own format. The Fig format can be read by the Xfig program and from this program or by using the command `fig2dev` you can generate numerous other graphical file formats such as LaTeX, combined PostScript and LaTeX, PiCTeX, GIF, JPEG, XPM and many more.

The TCM file format is specified in appendix C. If you take a look into a saved file, you will see that it is 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.

B.32 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. Alternatively, you can try to extend or build your own diagram editor with the source code. See also question B.33.

B.33 Is it possible to make an editor for XYZ diagrams?

First of all you could probably use one of the already existing editors. Especially the generic diagram editor TGD is quite versatile.

If you have the sources and you wish to build a specific editor yourself or extend an existing editor, then it is not extremely difficult to do that, at least when the kind of diagram that you want to edit can be described as a graph. Or, if the document has some kind of tabular format, then you can make a variant of the generic table editor. If you want to develop your own editor, based on TCM, you are advised to read the developer's guide [4].

B.34 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 scaling it, making the drawing larger. But, when you use TCM a lot, you will inevitably find a situation in which an editor messes up your drawing. If that is the case, please send us a bug-report.

B.35 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 general, this is not allowed in a TCM editor. This is because most types of diagrams are graphs, so each line should connect two node shapes; a line cannot connect to another line. In TGD it is possible to draw something that look like a "fork", as you can connect in TGD a line (edge) with another line (edge). 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.

B.36 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, a 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 rendered white, and when one of the shapes is moved or removed, it will be redrawn, so the overlapping part will then appear black again. For some reason a small rounding error can sometimes occur in drawing some kind of shape and that results in some remaining black pixels. When you do a refresh, these pixels disappear.

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 (double buffering). It is used to copy back the contents into the drawing area quickly when the drawing area has been resized or overlapped by other windows. 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.

B.37 Why doesn't the BackSpace key function correctly in Linux?

Applications that run under XFree (the X11 port for Linux), seemed 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
```

B.38 Why does TCM have a tea pot with a 'T' as logo?

Because everything in TCM starts with a T.

B.39 What do these "assertion failed:" messages mean?

It can happen that internal data structures such as the internal graph, a shape view or a table in the editor become inconsistent [3]. Therefore, we have built some assertions in the code. They are used for internal consistency checking or for checking if a command can be executed safely. 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, who does not want to browse the source code, But they can be very helpful for searching bugs. So when you send a bug report to us, you should include these assertion failed messages. Furthermore, as the diagram file is in plain text, you can also include a faulty document as plain text in your e-mail.

B.40 I have a question. What should I do now?

Maybe you can find the question and answer here in the FAQ. Otherwise you can post a message to the TCM mailing list (see question B.13).

B.41 I found a bug. What should I do now?

Please send a bug report to tcm@cs.utwente.nl. Try to provide enough information that we can do something about it. You should give the name the version and the intended platform of the editor and also a resume of the commands you gave or other things you did. Please check also whether the error is reproducible or not. Also, if you have a faulty TCM document, you can e-mail it to us so that we can have a look at it. TCM document files are plain text so you can e-mail them as a plain text message (attachment).

B.42 How can I contribute to TCM?

Please check the documentation directory of the TCM distribution for the Wishlist - containing an overview of wishes not yet implemented - and some notes on future developments. In the specifications subdirectory (TCM_DOC/specifications) you can find some examples of the way we specify our additions to TCM.

Adhering to this standard specification format will enlighten our evaluation of your contribution a great deal. We might give you valuable feedback before implementing your contribution which otherwise could lack major functionality, conflict with other parts of TCM, or could turn out to be difficult to integrate within the existing TCM software.

After all : TCM is designed for specifying the software design before building it... :-)

Finally, check the TCM mailing list (see question B.13), for maybe someone else is already inventing the same wheel.

[3] When you want you can even try to enforce an inconsistent editor state by editing a TCM document file by hand and load that 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.

Appendix C

TCM File format

C.1 Introduction

Each TCM document is stored in a separate Unix file as plain text. This chapter contains a specification of the TCM file format of documents that are generated by the TCM version that is described in this manual (file format version 1.31). 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 latest format by reading it in with an editor of the latest version and then saving it back to file.

You can see and even modify a document file within TCM. With Document Source from the Document menu, the contents of the file, from which the TCM document was loaded, is read into a text edit dialog. You can then view and edit these file contents and save it back to the file (or to another file). You can see the updates when you append or load that file again.

C.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 and Page sections. The order of the row and column sections of a table is significant as is explained in section C.5.

A section starts with 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 of one of the types listed in figure C.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.

It is possible to include comment text in a TCM file. Comments in a file start with a hash (#). The text in the same line after the hash is comment text and is ignored.

When a file is read, TCM checks the syntax and checks the semantics, i.e. 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 subject fields referring to existing subject sections). TCM reports errors that are

type	**description**
<activation>	Trigger, Stimulus,Time,Unspecified.
<alignment>	Left, Center,Right.
<attribute>	<id>.<word>
<bool>	False,True
<color>	A color name enclosed by double quotes. The allowed color names are listed in the file $TCM_HOME/lib/colorrgb.txt.
<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.
<line−end>	Empty, OpenArrow, FilledArrow, DoubleOpenArrow, DoubleFilledArrow, BlackTriangle, WhiteTriangle, BlackCircle, WhiteCircle, BlackDiamond, WhiteDiamond
<linestyle>	Solid, Dashed, Dotted, Invisible,Dual
<namedirection>	None,FromShape,ToShape
<number>	Natural number in decimal notation.
<persistence>	Instantaneous, Continuing.
<rational>	Rational number in decimal notation.
<fillstyle>	Unfilled, Grayfilled, Filled.
<string>	Sequence of zero or more characters enclosed by double quotes " ". Strings cannot contain newline characters. Newlines 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 C.1: The value types of attributes stored in a file.

found together with a line number in the file where the error is encountered. The errors are displayed in the error log pop-up-window.

C.3 Storage, Document and Page Information

Each TCM document in a file starts with the Storage section with information about the file such as the file format number. This is followed by the Document section in which you can find general information about the document, like the name and the author. Then you have the Page section with some information about the page layout and then the Scale section that contains (only) the scale of the diagram.

Storage section

```
Storage
{
   { Format <rational> }   # File format, e.g. 1.26.
   { GeneratedFrom <word> }# Tool name and version e.g. TGD-version-1.84.
   { WrittenBy <word> }    # Unix login name that has generated this file.
   { WrittenOn <string> }  # Writing date/time (in 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> } # Creation time
                           # e.g. "Wed Sep 24 15:42:47 MET DST 1997".
    { Annotation <string> }# Free annotation text about this document.
    { Hierarchy <bool> }   # For diagram types that allow hierarchic
                           # documents only: Is this document hierarchic?
}
```

Page section

```
Page
{
    { PageOrientation <word> } # Portrait or Landscape
    { PageSize <word> }        # PageSize: A4, A3, Letter, Legal,...
    { ShowHeaders <bool> }     # Show a header on every page?
    { ShowFooters <bool> }     # Show a footer on every page?
    { ShowNumbers <bool> }     # Show a page number on every page?
}
```

Scale section

```
Scale
{
    { ScaleValue <rational> }  # The scale of the diagram. Normally 1.00.
}
```

C.4 Diagram Editor File Format

A stored diagram contains a section for each node, edge, view and shape (in no particular order) which follow after the three obligatory storage, document and page sections. Each node type, edge type and shape type section starts with a keyword. A view has the keyword `View`. In figures C.2 to C.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 C.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 C.2. Each node has a name, annotation and parent field. Some node types have additional fields which are mentioned below.

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

The parent field is not used in the current TCM version but it will be used for hierarchical diagrams. In hierarchical diagrams the parent identifier refers to an existing node section. That would mean that a node in a diagram is 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 the generic editor, TGD, these structures will be almost unconstrained. In data flow diagram editors (TDFD and TEFD) and data view editors (TERD and 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 can also used in the tree editors (TFRT, TGTT) but here the entire hierarchy is presented in one view.

When the parent relationship is not used in some editor then 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 are only top-level diagrams so the parent fields are always 0.

Other attributes of node types are:

- DataProcess (TDFD, TEFD)

  ```
  { 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

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

Tool	Node type	Tool	Node type
All UML diagram editors	Note	TGD	GenericNode
All diagram editors	Comment	TGTT, TFRT	TextRoot
TATD	ATDActionStateNode	TGTT, TFRT	TextNode
TATD	ATDDecisionStateNode	TPSD	PSProcess
TATD	ATDFinalStateNode	TRPG	ProcessGraphNode
TATD	ATDWaitStateNode	TRPG	ProcessGraphRoot
TATD	ATDSynchronizationNode	TSCD	SCDAndState
TATD	ATDInitialStateNode	TSCD	SCDDecisionState
TCBD	CBDActorNode	TSCD	SCDDefaultState
TCBD	CBDClassNode	TSCD	SCDFinalState
TCBD	CBDObjectNode	TSCD	SCDOrState
TCPD,TDPD	CPDComponentNode	TSCD	SCDSynchronizationState
TCPD,TDPD	CPDInterfaceNode	TSND	ControlledDataStream
TCRD	ClassNode	TSND	DataStream
TCRD	ModeJunction	TSND	SNProcess
TCRD,TERD	TaxonomyJunction	TSND	StateVector
TCRD,TERD (in future used).	SubjectArea	TSSD	SSDObjectNode
TDFD, TEFD	DataProcess	TSSD	SSDAggregationNode
TDFD, TEFD	DataStore	TSSD,TESD	SSDClassNode
TDFD, TEFD	ExternalEntity	TSSD,TESD	SSDGeneralizationNode
TDFD, TEFD	SplitMergeNode	TSSD,TESD	SSDAssociationNode
TDPD	DPDResourceNode	TSTD	InitialStateNode
TEFD	ControlProcess	TSTD	DecisionPoint
TEFD	EventStore	TSTD	State
TERD	EntityType	TUCD	UCDActorNode
TERD	ValueType	TUCD	UCDUseCaseNode
TERD	RelationshipNode	TUCD	UCDSystemNode
TGD	EmptyNode		

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

Tool	**Edge type**	**Tool**	**Edge type**
All UML diagram editors	CommentLink	TGD	GenericEdge
TATD	ATDTransitionEdge	TGTT, TFRT	TextEdge
TCBD	CBDAndEdge	TRPG	Event
TCBD	CBDTransitionEdge	TSCD	CBDClassLinkEdge
TCPD,TDPD	CPDDependencyEdge	TSCD	CBDObjectLinkEdge
TCPD,TDPD	CPDRealizationEdge	TSND	ConnectionEnd
TCRD	ComponentFunction	TSND	ConnectionStart
TCRD, TERD	BinaryRelationship	TSSD	SSDAggregationEdge
TCRD, TERD	Function	TSSD	SSDCompositionEdge
TCRD, TERD	IsaRelationship	TSSD	SSDObjectLinkEdge
TCRD, TERD, TPSD	EmptyEdge	TSSD,TESD	SSDAssociationLinkEdge
TDFD, TEFD	BidirectionalDataFlow	TSSD,TESD	SSDBinaryAssociationEdge
TDFD, TEFD	DataFlow	TSSD,TESD	SSDGeneralizationEdge
TDPD	DPDCommunicationEdge	TSSD,TESD	SSDParticipantLinkEdge
TEFD	ContinuousDataFlow	TSTD	Transition
TEFD	ContinuousEventFlow	TUCD	UCDBinaryAssociationEdge
TEFD	EventFlow	TUCD	UCDGeneralizationEdge

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

Tool	**Node shape type**	**Tool**	**Node shape type**
All UML diagram editors, TESD, TGD	NoteBox	TERD, TUCD, TGD	Ellipse
All diagram editors	TextBox	TFRT, TGTT	ArrowTextBox
TATD, TSCD	MiniDiamond	TGD	Folder
TATD, TDFD, TEFD, TSCD, TSSD, TGD	BlackDot	TGD	Disk
TATD, TSCD, TGD	BullsEye	TGD	Triangle
TATD, TGD	EllipsedBox	TGD	SubFolder
TATD, TSCD, TGD	SolidHorizontalBar	TGD	VerticalBar
TATD, TSCD, TGD	SolidVerticalBar	TPSD	URLabeledBox
TATD, TRPG, TSCD, TGD	RoundedBox	TRPG	MiniArrowEllipse
TCBD	ActorObjectBox	TSCD	SCDAndStateBox
TCBD	ObjectStickman	TSND	LeftLineCircle
TCPD, TDPD, TERD, TESD, TCRD, TRPG, TSSD	MiniEllipse	TSSD	SSDDoubleObjectBox
TCPD, TDPD,TGD	BuildingBlock	TSSD, TCBD	SSDSingleObjectBox
TCRD	DoubleBox	TSSD	SSDTripleClassBox
TCRD	TripleBox	TESD, TSSD	SSDDoubleClassBox
TCRD, TERD, TSND, TSTD, TGD	Box	TESD, TSSD, TCBD	SSDSingleClassBox
TDFD, TEFD, TGD	HorizontalBar	TSTD	ArrowBox
TDFD, TEFD, TGD	Square	TSTD, TGD	Hexagon
TDFD, TEFD, TSND, TGD	Circle	TUCD	UCDBoundaryBox
TDPD,TGD	Cube	TUCD, TCBD	UCDSingleClassBox
TERD, TSSD, TESD, TSND, TGD	Diamond	TUCD, TCBD, TGD	Stickman

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

Tool	Line type
All diagram editors except TSTD and TSND	Line
TCBD	T4TListLine
TCRD, TERD, TESD, TSSD,TUCD	C2R2Line
TCRD, TERD, TSND	T1Line
TSCD	SCDAndLine
TSND	T1Arrow
TSSD	SSDR2Line
TSSD,TESD	SSDRCLine
TSTD	TransitionArrow

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

number of children (processes, stores, flows and/or split-merge nodes.). When the data process 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, the following stimulus field is added:

```
{ 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.

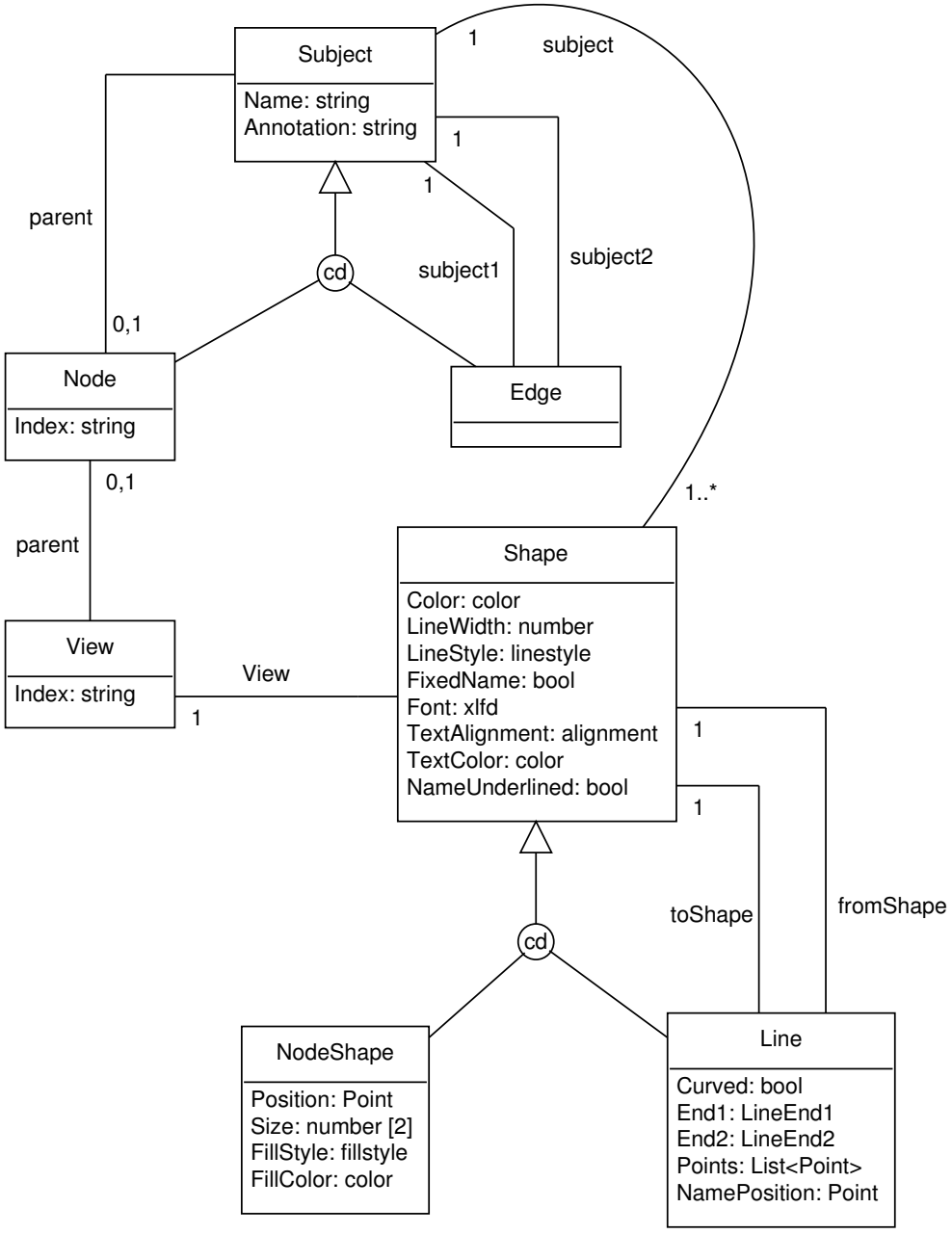


Figure C.6: Diagram file format in the form of a class-diagram.

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, TEFD)

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

In a future version of TDFD and TEFD 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 (>= 1 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).

- ClassNode (TCRD), SSDClassNode (TSSD, TESD)

```
{ Attributes <number> }     # number of attributes of class.
{ Attribute <string> }      # attribute string (>= 1 fields).
{ Operations <number> }     # number of operations of class.
{ Operation <string> }      # operation string (>= 1 fields).
```

The number of attributes specifies how many attribute fields follow and the number of operations specifies how many operation fields follow. Each attribute and each operation is a single-line string. The class nodes in TSSD have after the operations also the following two attributes:

```
{ Stereotype <string> }     # class stereotype string.
{ Properties <string> }     # class properties string.
```

- SSDObjectNode (TSSD)

```
{ Attributes <number> }     # number of attributes of object node.
{ Attribute <string> }      # attribute string (>= 1 fields).
```

Object nodes in TSSD do not have operations.

- 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 C.3.

```
<EdgeType> <id>                # e.g. BinaryRelationship 654321
{
    { Name <string> }          # name of edge.
    { Annotation <string> }    # annotation of edge.
    { Parent <id> }            # parent node.
    { Subject1 <id> }          # 'departure' subject
    { Subject2 <id> }          # 'arrival' subject
    # 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 Subject1 and Subject2 identifiers should refer to existing subject sections in this file. It is in principle possible that and edge (line) connects another edge (line). At this moment this feature of edge-edge connections is only available in TGD and in a limited form in TSSD (for association link edges and connection of notes).

The other attributes of edge types:

- BinaryRelationship (TERD,TCRD), SSDBinaryAssociationEdge (TSSD,TESD), SSDAggregationEdge (TSSD), SSDCompositionEdge (TSSD), UCDBinaryAssociationEdge (TUCD)

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

- ClassLinkEdge, ObjectLinkEdge (TCBD)

  ```
  { Constraint1 <string> }   # first cardinality constraint.
  { Constraint2 <string> }   # second cardinality constraint.
  { RoleName1 <string> }     # first role name.
  { RoleName2 <string> }     # second role name.
  { Messages <number> }      # number of messages of edge.
  { Message <string> }       # message string (>= 1 message fields)
  { Direction <direction> }  # message direction (ToShape, FromShape) (>= 1)
  { Flow <flow type> }       # flow type (FlatFlow, NestedFlow, Asynchronous) (>= 1)
  ```

- Function (TERD, TCRD), ComponentFunction (TCRD), ConnectionStart (TSND), ConnectionEnd (TSND)

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

- SSDObjectLinkEdge (TSSD)

  ```
  { RoleName1 <string> }    # first role name.
  { RoleName2 <string> }    # second role name.
  ```

- SSDParticipantLinkEdge (TSSD, TESD)

  ```
  { Constraint <string> }  # cardinality constraint.
  { RoleName <string> }    # role name.
  ```

- Transition (TSTD)

```
{ Event <string> }          # event string (including condition).
{ Actions <number> }        # number of actions in transition.
{ Action <string> }         # action string (>= 1 action fields).
```

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

- DataFlow (TDFD, TEFD), BidirectionalDataFlow (TDFD, TEFD), ContinuousDataFlow (TEFD)

```
{ Components <number> }     # number of sub-flows
{ Component <id> }          # sub-flow (>= 1 component 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 TEFD. 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 (TEFD), ContinuousEventFlow (TEFD)

```
{ Components <number> }     # number of sub-flows
{ Component <id> }          # sub-flow (>= 1 component 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 TEFD 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, a view in a DFD shows the refinement of a data process and a view in an ERD 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 should refer 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 indexes of the individual nodes are stored separately in the node section. The reason is that not all nodes need to have an index, and in some editors a different naming scheme could be used (for instance, in TGD, nodes can have an arbitrary index label and in TDFD, data stores do not have an index label).

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 C.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.
    { Color <color> }              # the line color of the shape.
    { LineWidth <number> }         # the line width of the shape.
    { LineStyle <linestyle> }      # the line style of the shape.
    { FillStyle <fillstyle> }      # the way the shape is filled.
    { FillColor <color> }          # the fill color when the shape not unfilled.
    { FixedName <bool> }           # string of name-textshape is fixed?
    { Font <xlfd> }                # text font of text strings.
    { TextAlignment <alignment> }  # multi-line text alignment.
    { TextColor <color> }          # the color of the text in the shape.
    { NameUnderlined <bool> }      # name-textshape is underlined?
}
```

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 the node shape section but it is equal to the name of the node subject, but some other attributes of the labels, i.e. the font and text alignment are specified in this section.

Node shapes have a (line) color, a text color, and a fill color (only visible with a fill style that is not unfilled). By default the line color and text color are black, the shape is unfilled and the line width is 1.

The node shapes SSDSingleClassBox, SSDDoubleClassBox and SSDTripleClassBox in TSSD and TESD have additionally the following two attributes to indicate whether the stereotype and properties labels of the subject should be shown:

```
    { ShowStereotype <bool> }
    { ShowProperties <bool> }
```

Line sections

See figure C.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.
    { FromShape <id> }                 # 'departure' shape.
    { ToShape <id> }                   # 'arrival' shape.
    { Curved <bool> }                  # straight (False) or curved (True)
    { End1 <line-end> }                # type of line end at departure side.
    { End2 <line-end> }                # type of line end at arrival side.
    { Points <number> }                # number of line points
    { Point <number> <number> }        # line point (>=2 points).
    { NamePosition <number> <number> }# position of name label.
    { Color <color> }                  # the color of the line (default=black).
    { LineWidth <number> }             # the width of the line (default=1).
    { LineStyle <linestyle> }          # the style (solid, dashed etc.).
    { FixedName <bool> }               # string of name-textshape is fixed?
    { Font <xlfd> }                    # text font of text labels.
    { TextAlignment <alignment> }      # multi-line text alignment.
    { TextColor <color> }              # the color of the text (default=black).
    { NameUnderlined <bool> }          # name-textshape is underlined?
    # possibly other line attributes
}
```

Each line is contained in an existing view and represents an existing edge and connects two existing (not necessarily different) shapes, called FromShape and ToShape. Both at the beginning point as at the end point of the line there is some **line end** which can be some kind of arrow head or a little circle, diamond or triangle or the line end is just Empty. The Points field in the line section specifies how many point fields will follow. A line has at least two points. Each point has a distinct Point field. The name position field gives the coordinates of the name label. The text of this label can be found in the edge section of the subject, in its Name field. Because the labels of a line can be positioned at free will, whereas node shape labels can not, only line sections have a distinct name position field. For the rest, line sections have a number of fields that also occur in node shape section, like for the colors, text alignment and line width. Extra attributes for some specific line types are:

- T1Arrow (TSND), T1Line (TERD, TCRD, TSND)

  ```
  { T1Position <number> <number> }  # position of an extra label.
  ```

- SSDRCLine (TSSD, TESD)

  ```
  { T1Position <number> <number> }  # position of 1st extra label.
  { T2Position <number> <number> }  # position of 2nd extra label.
  ```

- SSDR2Line (TSSD)

  ```
  { T1Position <number> <number> }  # position of 1st extra label.
  { T2Position <number> <number> }  # position of 2nd extra label.
  { NameDirection <namedirection>}  # direction to read the name.
  ```

- C2R2Line (TERD, TCRD, TSSD, TESD, TUCD)

```
{ T1Position <number> <number> }  # position of 1st extra label.
{ T2Position <number> <number> }  # position of 2nd extra label.
{ T3Position <number> <number> }  # position of 3rd extra label.
{ T4Position <number> <number> }  # position of 4th extra label.
{ NameDirection <namedirection>}  # direction to read the name.
```

These labels can also be positioned at free will and therefore their positions are saved separately.

- C2R2MListLine (TCBD)

```
{ T1Position <number> <number> }  # position of 1st extra label.
{ T2Position <number> <number> }  # position of 2nd extra label.
{ T3Position <number> <number> }  # position of 3rd extra label.
{ T4Position <number> <number> }  # position of 4th extra label.
{ NameDirection <namedirection>}  # direction to read the name.
{ Messages <number> }             # number of message labels.
{ TnPosition <number> <number> }  # position of xth message label.
                                    (>= 1 message label fields)
```

These labels can also be positioned at free will and therefore their positions are saved separately.

- TransitionArrow (TSTD)

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

The transition arrow 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.

C.5 Table Editor File Format

Like diagrams, the table editor file format starts first with a Storage section and then a Document section and a Page section. Directly after the Page section follows the Table section with some attributes of the entire table.

Table section

```
Table {
    { TopLeft <number> <number> }    # top-left (x,y) of entire table.
    { NumberOfRows <number> }        # nr. of rows in table (cells per column).
    { NumberOfColumns <number> }     # nr. of columns in table (cells per row).
    { MarginWidth <number> }         # min. distance text and column line.
    { MarginHeight <number> }        # min. distance text and 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 itself not of a table stored in a file.

Row sections

After the Table section follow the row sections in consecutive order (they are numbered from 0 to NumberOfRows-1). Each row section starts with the following three attributes:

```
Row <number> {
    { Height <number> }           # all cells in row have the same height.
    { Alignment <alignment> }     # all texts in row have the same alignment.
    { NumberOfCells <number> }    # a row having n cells has n+1 lines
    ... rest of the row attributes.
}
```

The NumberOfCells of a row has to be equal to the NumberOfColumns field in the table section. This field indicates how many cells the rows contain. The rest of the row attributes consist of the attributes of the cells and lines (line pieces) in a row. The line pieces to the left and to the right of the cells in a row are seen as part of the row too. For each line piece a separate line style and width field are stored. About the cell itself, the cell text string, the fonts and some annotation text is stored. So for every cell the following information is stored:

```
    ...
    { LineStyle <linestyle> }        # Line style of the line piece
    { LineWidth <number> }           # Line width of the line piece
    { Text <string> }                # texts in a cell.
    { Font <xlfd> }                  # XLFD font description.
    { Annotation <string> }          # annotation text of this cell.
    ...
}
```

The size of the cell is determined by the sizes of its row and column. Text alignment of a cell is determined by the combined row and column alignment. In a row section, the two line fields and the three cell fields alternate and, because the number of line pieces is the number of cells plus one, the row always end with two extra fields for the line style and line width of the last line piece.

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 alignment.
    { NumberOfCells <number> }  # a column having n cells has n+1 lines
    ...
}
```

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 and width fields for horizontal line pieces. There are NumberOfCells+1 line style and width fields in a column. Therefore the rest of the column section consists of NumberOfRows+1 times the following two attributes:

```
...
{ LineStyle <linestyle> }        # line style of a line piece
{ LineWidth <number> }           # line width of a line piece
...
```

Bibliography

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

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

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

[4] F. Dehne and H. van de Zandschulp. Toolkit for conceptual modeling, design and implementation. Technical report, Faculty of Computer Science, University of Twente, 2000. http://www.cs.utwente.nl/~tcm.

[5] 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.

[6] 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.

[7] F. Dehne and R.J. Wieringa. Toolkit for conceptual modeling, user's guide for tcm 1.6.x. Technical Report IR-437, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, November 1997.

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

[9] M. Fowler and K. Scott. *UML Distilled*. Addison-Wesley, 1997.

[10] 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.

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

[12] M. Lundeberg, G. Goldkuhl, and A. Nilsson. *Information Systems Development — A Systematic Approach*. Prentice-Hall, 1981.

[13] J. Martin. *Information Engineering*. Prentice-Hall, 1989. Three volumes.

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

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

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

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

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

[19] UML Revision Task Force. *OMG UML Specification*. Object Management Group, March 1999. http://uml.shl.com.

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

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

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

[23] R.J. Wieringa. Design methods for reactive systems: Yourdon, statemate and the uml. Technical report, Department of Computer Science, University of Twente, 1999. Course notes.

[24] 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 [25].

[25] 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.

[26] R.J. Wieringa and J.-J.Ch. Meyer. Actor-oriented specification of dynamic and deontic integrity constraints. In B. Talheim, J. Demetrovics, and H.-D. Gerhardt, editors, *3rd Symposium om Mathematical Fundamentals of Database and Knowledge Base Systems (MFDBS 91)*, pages 89–103. Springer, 1991. Lecture Notes in Computer Science 495.

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

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

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

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

[31] Yourdon Inc. *Yourdon*$^{\text{TM}}$ *Systems Method: Model-Driven Systems Development*. Prentice-Hall, 1993.

Index