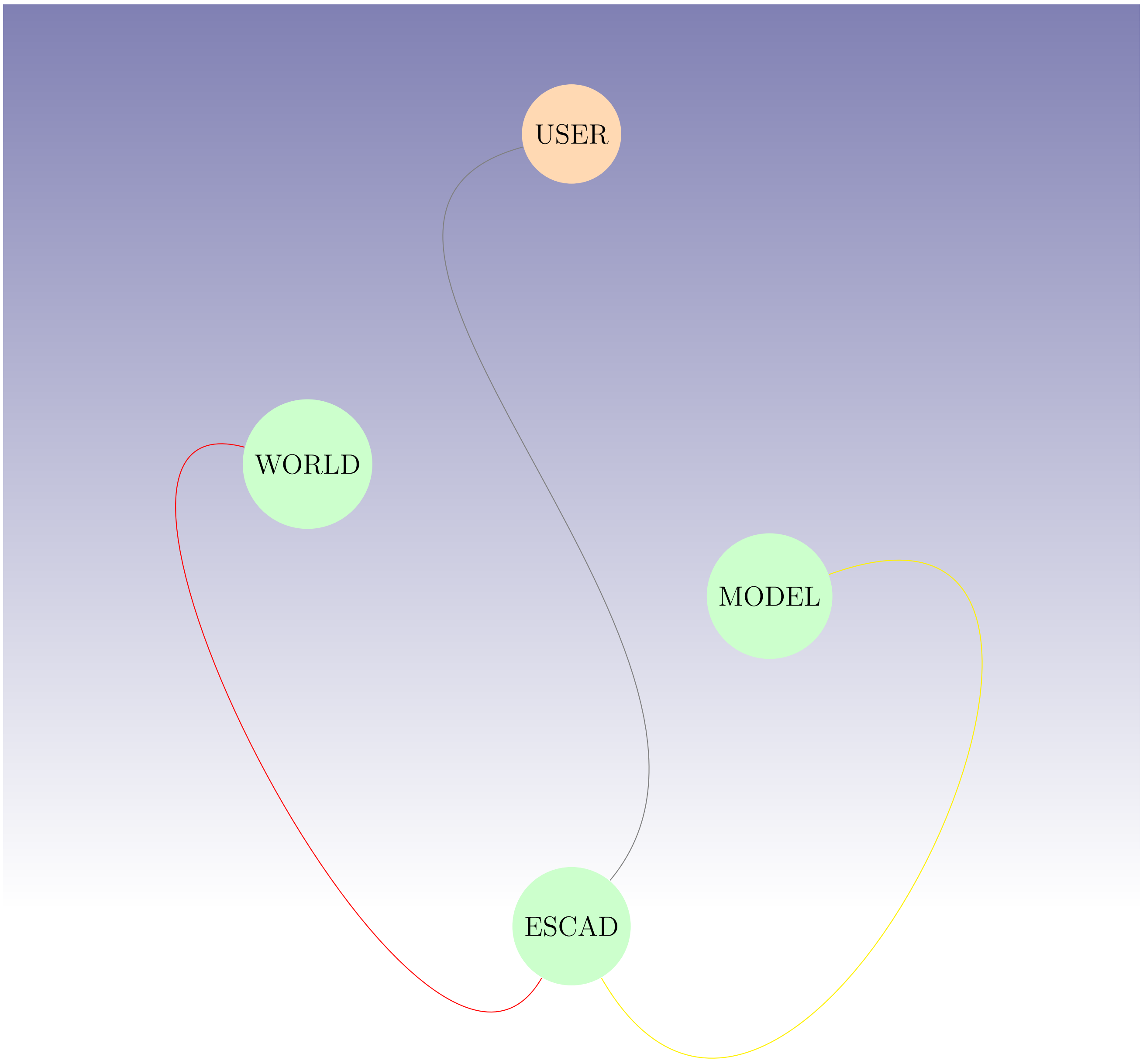
ESCAD MANUAL - 4th December 2020

by: Markus Kollmar



Alles hängt mit allem zusammen. Wir haben nur nicht das ganze Bild...

Contents

1 Introduction

First of all i have have to apologize that escad and this manual is not in a mature state. There is much to develop and to do. But i think it is better to regularly update the system than work some month and there is no regularly progress visible.

1.1 Philosophy

History tells us about separation. Separation between different professions was and is common. People have different interests and different knowledge. So this seems natural. Is this a problem? No. And yes. Nowadays science goes into a direction where **interdisciplinarity** seems more useful. The bounds of our disciplines are drawn by human but it not need to reflect the reality in nature. Real problems are mostly system-problems. Systems are combined of different disciplines. But do the (software) tools support this fact? Some may but many do not. This seems not a big problem in many cases. But when it comes to documentation or knowledge transfer tasks, this can be a big problem. The tools often do not interact with each other, and if so they often feel not integrated well. This problem increases when different manufacturers have tools which interact not or very bad.

Living in todays world is getting more and more complex. There are laws made by human which you have to obey. Additionally mother nature has their ever-lasting rules, overwriting all human rules, and which we should obey to keep the envrionment healthy for future mankind. Thus people may have the need to get information about this **complex** system and tools to make this understandable. Escad is really not good in many things. It is no unversial software. In fact it is really not yet a graph analysis tool (however one could update that functionality). Escad is a simple worker: get something out of your graph model, make PDF, 3D models, music or other documents, which would be boring to create by hand. You can additionally also use the **scripting** facility of escad.

1.2 State

Currently escad is in (wild) development. This means there are many ideas which should be developed. Some are just a quick hack to show what is possible and some are more detailed. However but only usable for experts in some areas. Future plans are (among many other things not mentioned):

1. Increase domain functionality with practic use.
2. Create good and clear documentation with examples.
3. Make a good graphical user interface.

Escad use open interfaces and provide a wide variety of different domain tools. And if there is (yet) not the thing you need, you can customize Escad with common-lisp code. Escad wants to be kind and helpful to the user. **Your help is really welcome**: if you want maintain this manual, increase escad features, write expansions, create logo, manage homepage or just send some feedback - all things are kind to start a escad-community.

1.3 License

Escad strictly wants to be open source, for more see file `LICENSE` in your escad root-directory.

1.4 Installation

Currently there are no preconfigured packages for convenient installation of escad. However installation should not be to difficult, since escad-development tries to minimize not shipped dependencies.

1.4.1 Linux or other unix-like systems

1. Install **GNU CLISP** or **sbcl** common-lisp implementation (other implementations may work, but are not explicitely tested) preferably with your distribution-package-manager.
2. For some PNG/SVG export install **graphviz**.
3. For PDF exports install a basic **latex** installation.
4. Copy escad files and directories (leave them in the given structure) to a place (e.g. your homedirectory).
5. Check the user-config section at the beginning of `escad.lisp` and edit if necessary.
6. Check the config section at the beginning of shell-script `escad_server` and edit if necessary.
7. Check the user-config section at the beginning of `escad.lisp` and edit if necessary.
8. You can now try to run escad with `escad_server`.

1.4.2 Windows

This system is currently not tested, but may be possible.

1.5 Theory

To understand a software-system it is often easier to understand the theory behind. In escad this is quite simple, since it is practical use of graphs (in informatically or mathematically means). **Graphs** consist only of **symbols** V (vertices, nodes, Knoten) and **relations** E (edges, Kanten). See figure 1.1 about the structure of escad. Whenever needed you can add classification info (taxonomy) or functionality (expansion). Mathematically (see also [BSMM08]) you can see this in equations 1.1:

$$\begin{aligned} V &= \{s0, s1, s2, s3\} \\ E &= \{r0, r1, r2, r3\} = \{(s0, s1), (s1, s2), (s2, s3), (s3, s2)\} \end{aligned} \tag{1.1}$$

You see in math you would need no name for relations, as it are ordered pairs of symbols. Escad wants names for symbols and relations. This is because relations can have some optional properties (symbols too), which you can acess via the relation-name. As convenience you can let escad give you automatic generated names, in case you do not care of the exact name.

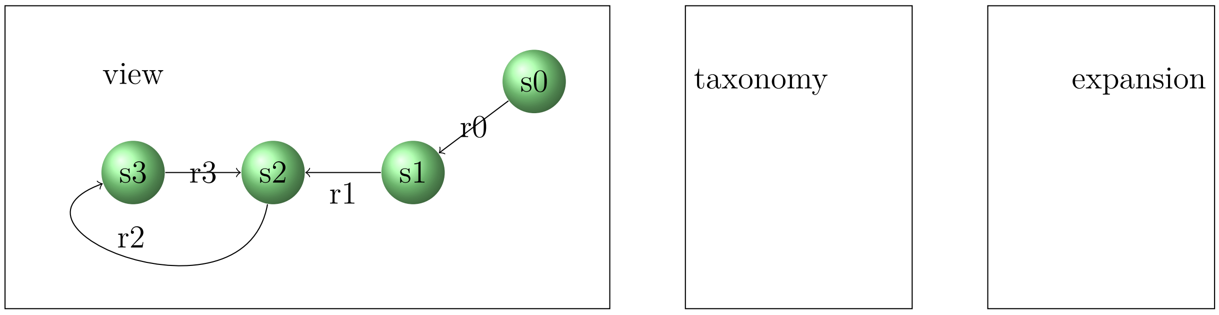


Figure 1.1: Overview of basic escad structure.

1.5.1 Symbol

In many papers symbols are also called node. However escad has the aim to model things into software. Thus a node is a *variable* for something in our thoughts. A symbol can represent a house, a number, a theory, a joke or whatever you want. This shows the great flexibility of escad. In figure 1.1 symbols are circles with names in it (e.g. s0) which you can name however you want as long it is a unique name in the current graph. In case you do not care, escad can generate a unique name for you.

1.5.2 Relation

Relation or in computer science called edge creates relations between symbols.

1.5.3 Taxonomy

With taxonomy you can create a ontology for the model. This is a domain classification which allows you to describe your symbols and relations in more detail. Depending of the taxonomy there may result different (graphical) output or behaviour (functionality).

1.5.4 Expansion

Expansions are ordinary *symbols* with a additional function, given by assigning a taxonomy and executed via activation of a symbol. Nowadays such easily achivable functionality may be called *apps*, who can do various things with your graph. This ranges from graphic output to new generated graph-elements. There are many possibilities and you can even write your own expansion(s). Those programms live in the escad environment and can use the provided feature, even of other expansions. However currently there is only a limited set of expansion, but this could increase in time. Feel free to write a expansion for your domain specific needs.

2 Usage

Here i assume it is easier to get what escad can do for you, by showing escad in work. Note that the output of the following examples may vary a little at your side (becausse of various reasons), but the basic things should be similar.

2.1 Tutorial

We want now do a short session in escad. After installation go to the escad root directory and start escad by typing in a shell in your terminal:

```
user@host:∼/escad$ ./escad_server start
>
```

This starts escad REPL (no special arguments given). After loading of common-lisp and escad, you should switch from the common-lisp in the escad namespace:

```
> (in-package :escad)
ESCAD>
```

Now you can execute all escad commands directly. To see some possible existing symbols type following escad-command:

```
ESCAD> (ls)
("_escad" "_view")
```

You can read more about this command at page 11. We see two symbols which escad has already created for us. The first symbol can contain settings for our current escad session. The second symbol stands for the current working space we are working. In escad this is called *view* and is just a place where you can work. It can hold graph(s), but does not have to. Escad has two views, you can use (toggle with command `tv`). But mostly you need just one view. However we heard that escad has symbols and relations. Lets look at the relations:

```
ESCAD> (lr)
NIL
```

Upps, we get *NIL* which is the lisp way of saying that there is nothing. But that is ok, since we just have not created anything yet. So lets create three symbols:

```
ESCAD> (dolist (name '("one" "two" nil)) (ns name))
NIL
```

This produces three new symbols, which names we not see because of the `dolist` nature:

```
ESCAD> (ls)
("_escad" "_view" "one" "two" "s0")
```

The third symbol name `s0` created escad for us, because we provided `nil`. Now lets create a relation:

```
ESCAD> (nr nil "one" "two")
"r0"
```

Now we got

```
ESCAD> (lr)
("r0")
```

Note that you can not insert a new relation or symbol which name already exists. You now have nearly seen all basic operations in escad. Most functionality can be achieved by this. Instead of learning many new commands, you can use in escad symbols which can also represent actions (e.g. exporting a PDF of your view). But how can you execute such a symbol. This makes the command *as*, which *activates* the given symbol:

```
ESCAD> (as "s0")
("Documentation text...")
```

You can activate every symbol, but most of them will just print some documentation about themselves, like you have seen in the last command. To get another functionality you have to assign a taxonomy which refers to a expansion. Those expansion is then loaded and executes a defined command (which is defined by the taxonomy). You can easily add taxonomy to a symbol with add a *property*:

```
ESCAD> (s "s0" :taxonomy "escad.symbol._escad.export.pdf")
("s0")
```

If you would activate this smybol now, it would produce a pdf with graphic output of your current view. Because you gave no file-name it would generate some. To give a filename you can add a special attribute-property:

```
ESCAD> (asa "s0" '("filename_relative" "my_file.pdf"))
("my_file.pdf")
```

Now you should get those pdf-file. This are the most basic commands you need to know in escad. To get detailed command info use command help:

```
ESCAD> (help-command 'cmd_name)
("Documentation text of cmd_name...")
```

To get basic help type:

```
ESCAD> (help)
("Documentation text...")
```

To list available taxonomies use command `(lta)`.

2.2 User stories

Because interoperability is important in todays heterogenic software world, there should be a way to get graphs from other software. Or in case you can not do some things in escad yet or other tools may do better, you can export your graph.

2.2.1 Import of graphviz-dot

Graphviz is powerful in graphically drawing of graphs. You can import those graphs with .dot extension. However only basic functionality of dot is currently supported. Import with a expansion.

2.2.2 Export of graphviz-dot

Graphviz is powerful in graphically layout and drawing of graphs. Export with a expansion.

2.2.3 Export of SVG

SVG is a vector graphic-format viewable through most modern webbrowsers.

2.2.4 Export of PDF

Export with a expansion.

2.2.5 Reporting

Reporting here means to get some basic statistically information about the graph. Additionally some expansions may allow you to dive deeper in the graph and provide some semantic informations too.

2.2.6 Standalone-HTML-javascript learning page generator

You can generate html-pages with integrated javascript functionality to train some excercises, e.g. language learning. It provides motivational intuitive training pages which you can easily generate without html or javascript knowledge. The user has to do excercises and gets result if anwer is correct or not.

2.2.7 Mindmap

This creates mindmaps in SVG format. Very usefull to view in a browser and to make your graph visible.

2.2.8 Flow

A flow is a graph wich models a process. You can use escad to analyse a flow or to create a process a user should go through (e.g. a question+answer quiz).

2.2.9 Creating (technical) documentation

This feature is planed in the future. The aim is to provide good quality output with use of latex, whereby you not need to know how to write latex (however it should be possible to insert latex directly whenever it is your wish).

2.3 Questions

Sometimes questions lead to a fast recognition of a problem or help to understand things better. So lets start asking...

2.3.1 Development

Can i help develop escad? Of course. Just contact the developer in github.

2.3.2 Usage

Is there a difference in the power of the different escad interfaces? Yes. The most powerful is currently the command line interface. New commands appear first there. However this is no rule. The aim is to keep the functionality equal across the interfaces in a later step.

Why you have choosen common-lisp as the language for escad? That is a good question. Why not javascript or another more popular language? Well may i ask why not lisp? In fact lisp has a very easy synthax, looks very nice in source code ;-) and for many tasks you need not more than three nested parentheses.

2.3.3 Other

Is there support for other languages as english? No not currently. If you want contribute feel welcome.

3 Reference

Here you should find the full information to work with escad, such as available commands or interfaces. Also you should get some basic things in order to get the idea in how you can extend escad or help in development.

3.1 Commandline LISP-REPL (currently recommended)

The commandline currently provides the full functionality. This means you got a common-lisp REPL with escad-package loaded. This gives you the full power of common-lisp with the ability of the escad-commands to work with a simple graph environment. To get this commandline go to the escad root directory and start escad-commandline by typing in a shell in your terminal:

```
user@host:~/escad$ ./escad_server start terminal
>
```

or because terminal-mode is the default you can also type:

```
user@host:~/escad$ ./escad_server start
>
```

Without correct command arguments you get a usage message. Now you can type in escad commands with a namespace-qualifier, in this case the help command (note the output is here ommitted):

```
> (escad:help)
...
```

In order to omitting this namespace just type once:

```
> (in-package :escad)
ESCAD>
```

Now you can type just (note the output is here ommitted):

```
ESCAD> (help)
...
```

3.1.1 Abbreviations and data structures

The following table 3.1 explains used symbols, abbreviations and the data-structures in the reference.

3.1.2 Commands

Note that this list may not be complete yet. To get the most actual overview check the inline documentation.

as

```
[ SN ]
```

```
STRING
```

<a>ctivate <s>ymbol in current view. What happens depends on the taxonomy of the symbol. Many symbols print out a string as their contents. Symbols which represent expansions will execute the configured function of the expansion.

pattern	**description**	**example**
ATTR	attribute taxonomy string	`"escad.attribute.author"`
(ATTR STRING ...)	attribute taxonomy value list	`("escad.attribute.author" "Author Name")`
NIL	common lisp nil means not true/-done	`NIL`
RN	relation name string	`"r0"`
SN	symbol name string	`"s0"`
STRING	string with unspecified or multiple semantic	`"a message string..."`
VN	view number	`0`
+	previous content can occur at least once or multiple times	
*	previous content can occur not or multiple times	
...	previous pattern can be continued	`(0 1 2 ...)`
()	basic common-lisp list	`(1 "Hello")`
[ ]	optional argument(s)	`(cmd [ ])`
function argument	function argument	
function result	function result, multiple values seperated by comma are possible	

Table 3.1: Explained abbreviations and symbols.

asa `SN (ATTR STRING ...)+`
`SN | NIL`
<A>dd/edit <s>ymbol <a>ttributes depending of key. NIL if nothing is added.

gra `RN attribute-string`
`STRING`
<G>et <r>elation <a>ttributes depending of given attribute-string.

gsa `SN attribute-string`
`STRING`
<G>et <s>ymbol <a>ttributes depending of given attribute-string.

help
`STRING`
Print <help>ful overview of escad, meaning of terms and all available commands.

lr `[ :filter :exclude-taxonomy ]`
`(RN*)`
<L>ist all <r>elations in current schematic which name match the filter. Additionally exclude relations which match the exclude-taxonomy.

ls `[ :filter :exclude-taxonomy ]`
`(SN*)`
<L>ist all <s>ymbols in current schematic which name match the filter. Additionally exclude symbols which match the exclude-taxonomy. See example at page 6 for usage.

nr
```
RN | nil SN SN [ :attributes :comment :taxonomy :weight ]
```
```
RN | nil
```
Create <n>ew <r>elation with given name and possible additional values in view. If the relation-name already exists do nothing and return nil. Default type is undirected relation. To make a directed or bidirected relation, set the appropriate taxonomy (note that ref_from and ref_to are only technical terms meaning you first tie the relation from that symbol to another. it can mean that is directe, but it is not guaranted that the author means that unless he makes that explicit with a relation that declares that).

ns
```
SN | nil [ :attributes :comment :taxonomy :weight ]
```
```
SN | nil
```
Create <n>ew <s>ymbol with given name and possible additional values in view. If the symbol-name already exists do nothing and return nil.

r
```
RN [ :comment :ref_from :ref_to :taxonomy :weight ]
```
```
RO | nil
```
Get/set <r>elation object.

s
```
SN [ :comment :taxonomy :weight ]
```
```
SO | nil
```
Get/set <s>ymbol object.

tv
```
VN
```
<T>oggle <v>iew since escad has two views change from the one to the other and gives back the number of the new view (0 or 1).

vs
```
(VN SC1 RC1 SC2 RC2)
```
Gives <v>iew <s>tatus.

3.2 Emacs with LISP-REPL and view buffer

This is in development.

3.3 Tk GUI

This is in development.

3.4 TCP-Socket

This section can refer fully to section 3.1. The same commands are available, except that they are transmitted via TCP-Socket as a text-stream.

3.5 REST-API

This is in development and not fully working yet.

3.6 Library

To use escad as a library in your common lisp programm, just load `package.lisp`.

3.7 HTML-client

This is in development and not fully working yet.

3.8 Included expansions

Those are the expansions which are included/shipped in this escad-package.

3.8.1 export_expansion.lisp

The following table 3.2 explains the export expansion.

taxonomy	escad.symbol._escad.export.dot
in	-
out	-
attribute	1. symbol name 2. [file name]
description	exports view to dot (graphviz)
taxonomy	escad.symbol._escad.export.pdf
in	-
out	-
attribute	1. symbol name 2. [file name]
description	exports view to pdf
taxonomy	escad.symbol._escad.export.svg
in	-
out	-
attribute	1. symbol name 2. [file name]
description	exports view to svg.

Table 3.2: Symbols and relations of export expansion.

3.9 Programming your own expansion

Writing escad expansions is not difficult. Go through following steps carefully and you will have a good basis for writing your own. The code in listing 3.1 shows a minimal expansion and the listing 3.2 shows the entry in a taxonomy file.

Listing 3.1: Minimal expansion code.

```
(in-package "COMMON-LISP-USER")
(defpackage :de.markus-herbert-kollmar.escad.expansion.hello_world
  (:use :common-lisp :escad)
  (:export :my-function)
  (:documentation "This␣is␣my␣expansion␣which␣does␣nothing␣usefull."))
(in-package :de.markus-herbert-kollmar.escad.expansion.hello_world)
(defun my-function (symbol-name-string)
"This␣function␣will␣be␣executed␣by␣calling␣this␣expansion.")
```

Listing 3.2: Entry in a taxonomy-file.

```
(:taxonomy "escad.symbol.hello_world" :doc "[E]␣Hello␣world."
:expansion "hello_world_expansion.lisp"
:package :de.markus-herbert-kollmar.escad.hello_world
:function "my_function"
:license "GNU␣GPL␣3")
```

Following things you should keep in mind:

- Choose namespace.
- Provide documentation within your expansion.
- Provide at least one function which can be called with the symbol-name-string as first argument und some possible further arguments.
- Provide a taxonomy file with your expansion declaration or request that your expansion will be taken into the default escad-taxonomy.

You are wellcome if you want include your expansion in this escad distribution.

Index

Bibliography

[BSMM08] Bronstein, Semendjajew, Musiol, and Mühlig. *Taschenbuch der Mathematik*. Harri Deutsch, 2008.