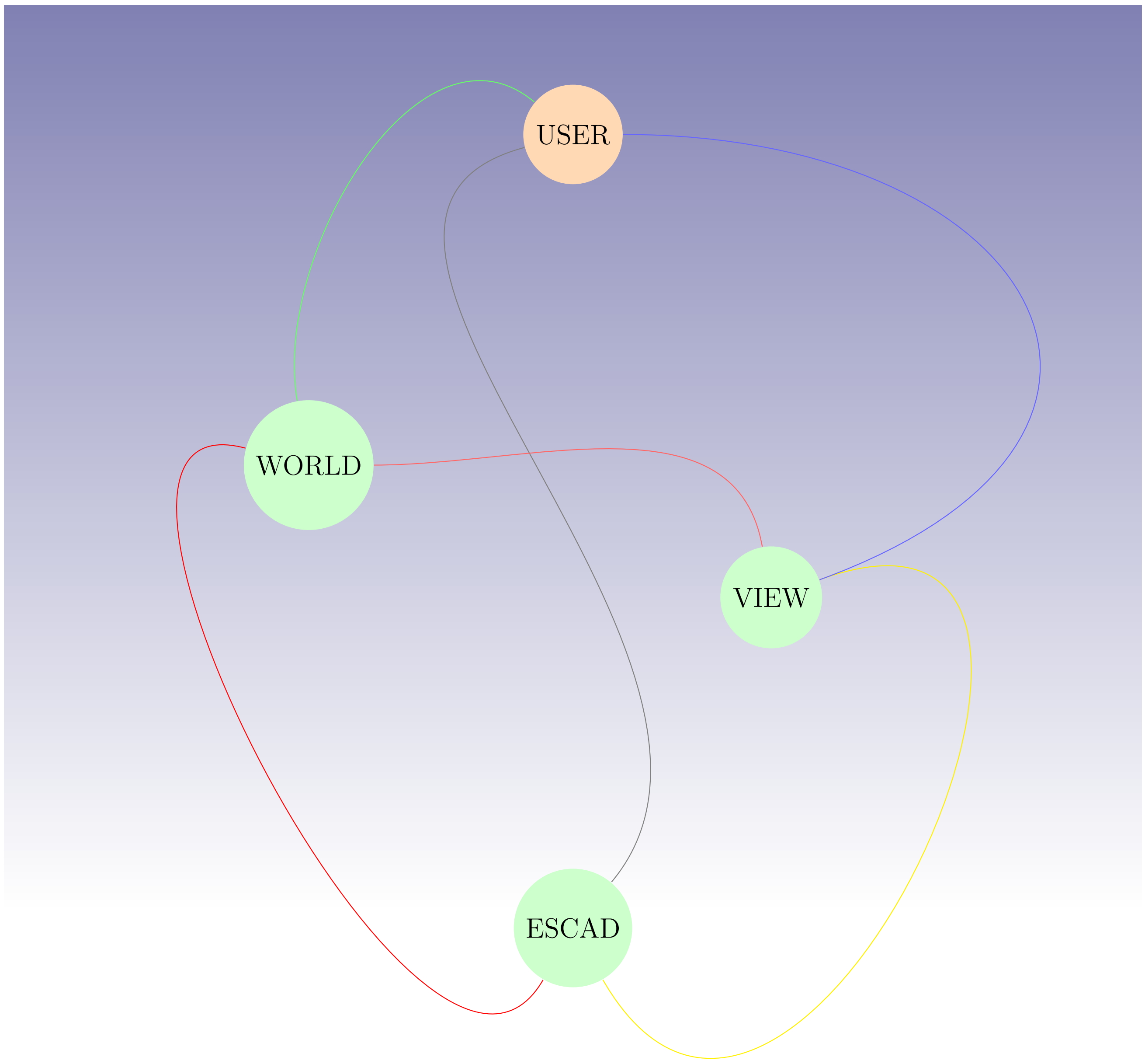
Manual for ESCAD version 0.1 - 17th October 2021

by: Markus Kollmar



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

Contents

1 Introduction **4**
- 1.1 Documentation-driven development (why this manual) . . . 4
- 1.2 Philosophy (why escad) . . . 4
- 1.3 State (what happens currently) . . . 5
- 1.4 License (what you can do or not) . . . 5
- 1.5 Installation (get it) . . . 5
  - 1.5.1 Linux or other unix-like systems . . . 5
  - 1.5.2 Windows . . . 6
- 1.6 Theory (how it works) . . . 7
  - 1.6.1 Symbol . . . 7
  - 1.6.2 Relation . . . 8
  - 1.6.3 Attribute . . . 8
  - 1.6.4 Taxonomy . . . 8
  - 1.6.5 Expansion . . . 8
- 1.7 Design patterns (interaction concepts) . . . 9
  - 1.7.1 Modeling . . . 9
  - 1.7.2 Hiding . . . 9
  - 1.7.3 Mounting . . . 9
  - 1.7.4 Exporting . . . 10
  - 1.7.5 Importing . . . 10
  - 1.7.6 Reporting . . . 10
  - 1.7.7 Generating . . . 10
  - 1.7.8 Automating . . . 10

2 Usage **11**
- 2.1 Tutorial . . . 11
- 2.2 User stories (included examples explained) . . . 12
  - 2.2.1 3D (escad version 0.2) . . . 12
  - 2.2.2 Contact tracing (escad version 0.2) . . . 12
  - 2.2.3 Documentation . . . 12
  - 2.2.4 Exporting . . . 12
  - 2.2.5 Flow . . . 13
  - 2.2.6 Importing graphviz dot . . . 13
  - 2.2.7 Learning page (version 0.2) . . . 13
  - 2.2.8 Lessons learned . . . 13
  - 2.2.9 Music/sound creation (version 0.2) . . . 13
  - 2.2.10 Mind-map . . . 13
  - 2.2.11 Modelling . . . 13
  - 2.2.12 Pedigree . . . 14
  - 2.2.13 Quiz . . . 14
- 2.3 Questions . . . 14
  - 2.3.1 Development . . . 14

2.3.2 Usage 14
2.3.3 Other 14

3 Reference **15**
3.1 Commandline LISP-REPL (currently recommended) 15
3.1.1 Abbreviations and data structures 15
3.1.2 Commands 15
3.2 Emacs with LISP-REPL and view buffer 17
3.3 Tk GUI 17
3.4 TCP-Socket 17
3.5 Connector-API 18
3.6 Library 18
3.7 HTML-client 18
3.8 Included expansions 18
3.8.1 export_expansion.lisp 18

4 Development **20**
4.1 Directory structure 20
4.1.1 doc/* 20
4.1.2 examples/* 20
4.1.3 lib/* 20
4.1.4 node_modules/* 20
4.1.5 public/* 20
4.1.6 test/* 21
4.1.7 ui/* 21
4.2 Programming your own expansion 21

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. So some things may not work as expected, look different than documented, or there are silly mistakes around. But i think it is better to regularly update the system and documentation, than work some month and there is no regularly progress visible. Documentation lives like the code! I recommend - and think it is very useful - to **read this manual**. Because you get a feeling about escad and its terminology. And unlike other manuals, escad tries a documentation-driven development process, which will be explained in the next section. This makes this documentation a view of the current development and shows you what can be awaited from escad and what not.

1.1 Documentation-driven development (why this manual)

Documentation-driven development means here (at least we try), at first we define in this manual what we want to develop, and how it looks for the user. Then the feature will be implemented after this specification. Thus we are forced to think more like what a user want, than what is more convenience for the developer. This is an advantage for the user, since he can see what a feature will look like, and documentation is mostly ahead and not behind the actual implementation. The specified feature in the manual, align to the mentioned version of escad. If some feature will be in a later version, this should be mentioned by a version notice. All follows this process in short:

1. Set version
2. Update/create documentation (manual)
3. Develop along documentation
4. Test
5. Deliver version (or tag it)

1.2 Philosophy (why escad)

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 environment 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 not good in some things. In fact it is really not yet a good graph analysis tool (however one could update that functionality). Escad wants to be a (relatively simple) worker which allows combining different domains in one model: get something out of your semantic-graph-model, avoid switching different tools too much, make LaTeX , 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.

1.3 State (what happens currently)

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. Many things are not completed. This is often to just give the idea what should come and is intended to be completed later if the system interacts satisfactorial as a whole. Future plans are (among many other things not mentioned):

- Increase domain functionality with practic use.
- Create good and clear documentation with examples.
- Make a good graphical user interface in browser via REST-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. Currently the best way to use escad is the terminal, or use the emacs editor, like in figure 1.1 where you can type in escad-lisp commands.

1.4 License (what you can do or not)

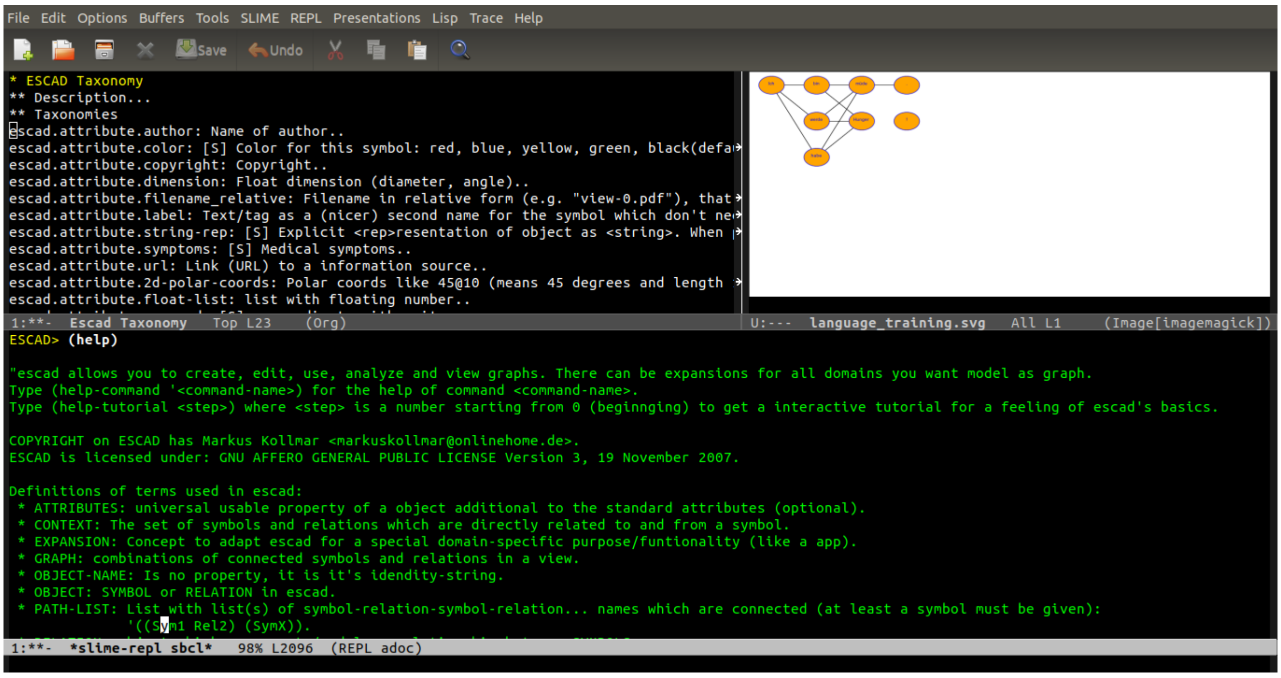
Escad strictly wants to be open source and helpful for many people. It is licensed under the *GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007*. For more information see the file `LICENSE` in your escad root-directory.

1.5 Installation (get it)

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.

1.5.1 Linux or other unix-like systems

1. Copy the escad root-directory from `https://github.com/mkollmar/escad` with all files (leave them in the given structure) to a place (e.g. your home-directory) or clone this github-repository with **git**.


File Edit Options Buffers Tools SLIME REPL Presentations Lisp Trace Help
Save
Undo

```
* ESCAD Taxonomy
** Description...
** Taxonomies
escad.attribute.author: Name of author..
escad.attribute.color: [S] Color for this symbol: red, blue, yellow, green, black(defa
escad.attribute.copyright: Copyright..
escad.attribute.dimension: Float dimension (diameter, angle)..
escad.attribute.filename_relative: Filename in relative form (e.g. "view-0.pdf"), that
escad.attribute.label: Text/tag as a (nicer) second name for the symbol which don't ne
escad.attribute.string-rep: [S] Explicit <rep>resentation of object as <string>. When
escad.attribute.symptoms: [S] Medical symptoms..
escad.attribute.url: Link (URL) to a information source..
escad.attribute.2d-polar-coords: Polar coords like 45@10 (means 45 degrees and length
escad.attribute.float-list: list with floating number..
1:**-  Escad Taxonomy   Top L23    (Org)
```

```
U:---  language_training.svg   All L1     (Image[imagemagick])
```

```
ESCAD> (help)

"escad allows you to create, edit, use, analyze and view graphs. There can be expansions for all domains you want model as graph.
Type (help-command '<command-name>) for the help of command <command-name>.
Type (help-tutorial <step>) where <step> is a number starting from 0 (beginnging) to get a interactive tutorial for a feeling of escad's basics.

COPYRIGHT on ESCAD has Markus Kollmar <markuskollmar@onlinehome.de>.
ESCAD is licensed under: GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007.

Definitions of terms used in escad:
 * ATTRIBUTES: universal usable property of a object additional to the standard attributes (optional).
 * CONTEXT: The set of symbols and relations which are directly related to and from a symbol.
 * EXPANSION: Concept to adapt escad for a special domain-specific purpose/funtionality (like a app).
 * GRAPH: combinations of connected symbols and relations in a view.
 * OBJECT-NAME: Is no property, it is it's idendity-string.
 * OBJECT: SYMBOL or RELATION in escad.
 * PATH-LIST: List with list(s) of symbol-relation-symbol-relation... names which are connected (at least a symbol must be given):
             '((Sym1 Rel2) (SymX)).
1:**-  *slime-repl sbcl*   98% L2096  (REPL adoc)
```

Figure 1.1: escad in emacs.

2. You should have/install **GNU CLISP** or **sbcl** common-lisp implementation (other implementations currently probably not work) preferably with your distribution-package-manager.

3. As shell **bash** is recommended.

4. For REST-server you should have or install **node** and **npm**. Then in the escad root-directory you should try to install the needed node-modules (`npm install -production`).

5. For some PNG/SVG export you need to have/install **graphviz** (not needed for basic escad usage).

6. For PDF exports you need to have/install a **latex** installation with **pdflatex** (not needed for basic escad usage).

7. For TK graphical interface you need to have/install a **TCL/TK** installation (not needed for basic escad usage).

8. Check the user-config section at the beginning of `escad.lisp` and edit if necessary.

9. Check the config section at the beginning of shell-script `escad` and edit if necessary.

10. You can now try to run escad (in terminal-mode) within the escad-root-directory by typing `./escad start`.

1.5.2 Windows

This system is currently not tested, but may be possible (e.g. with WSL).

1.6 Theory (how it works)

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 1.2 about the structure of escad. Whenever needed you can add classification info

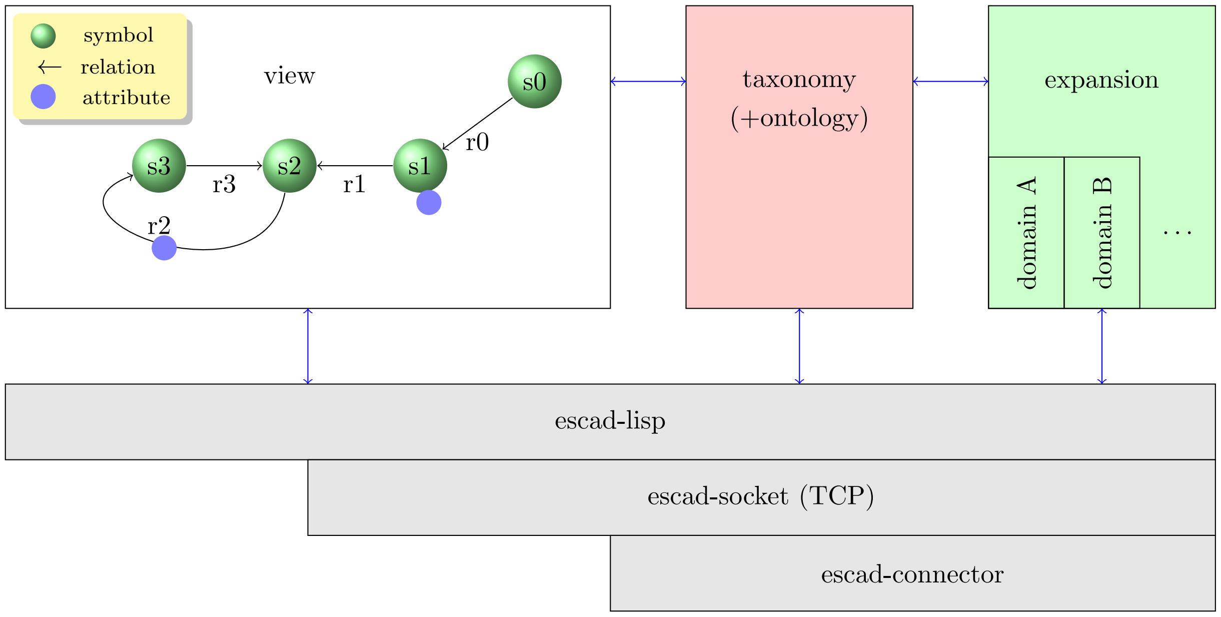


Figure 1.2: Overview of basic escad structure.

(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.

You can communicate with escad through common-lisp. This communication can also piped through a local TCP-connection. The REST-API uses this method too. Note that for security reasons you should not make such network-communications over the internet, since escad is not encrypted. However you could use technics like VPN if you want such behaviour.

1.6.1 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 1.2 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.

1.6.2 Relation

A relation (or in computer science also called edge) creates a relationship or dependency between two symbols. You can specify the meaning of this dependency further. Generally one can divide following classes of dependencies:

is_a can be used to describe/create symbol-hierarchies (also known as hyperonym–hyponym or supertype/superclass → subtype/subclass).

has_a can be used to define a possessive hierarchy/aggregation (no ownership) between symbols (also known as holonym/whole/entity/container → meronym/part/constituent/member).

part_of can be used to describe a ownership hierarchy/composition between symbols (also known as meronym/constituent → holonym/entity).

member_of can be used to describe a containment (also known as meronym/member → holonym/-container).

instance_of can be used to describe a concept-object (also known as token/object → type/class).

1.6.3 Attribute

An attribute is an *optional* specifier of a symbol/relation. It should be used carefull and not missused as an replacement for a additional symbol. Attributes merely are parameters which give the symbol or relation additional information, mostly needed in conjunction with expansions (e.g. giving a filename to a file-generation expansion).

1.6.4 Taxonomy

With taxonomy you can create in escad a classified specific (domain-based) meaning to your symbols and relations. Depending of the taxonomy there may result different (graphical) output or behaviour (functionality). This may sound clear at first, since without that a computer can not know what a symbol means just by interpreting its name.

However in semantic-knowledge field you may hear often words like ontology. Strictly spoken a taxonomy is merely a classification of things and terms. Ontology *uses* the taxonomy to capture and represent the meaning of a domain. Since escad is aimed to be a practical tool, it has some domain specific ontologie-knowledge included. This is done by the developers of a escad expansion and you may only effect it with some attributes. If you need a special ontologie you have to search for a expansion that fullfills it, or you have to build your own expansion.

1.6.5 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.

Table 1.1 shows the available expansions included in the current version of escad.

topic	**file**	**description**	**escad version**
3D	`3d_expansion.lisp`	Generate 3D files.	0.2
export	`export_expansion.lisp`	Export graph to PDF and SVG.	0.1
flow	`flow_expansion.lisp`	Automates processes in escad.	0.1
generation	`generator_expansion.lisp`	Generate objects in view.	0.1
import	`import_expansion.lisp`	Import graphviz dot-file in view.	0.1
report	`report_expansion.lisp`	Report view to latex and PDF.	0.1
standard	`standard_expansion.lisp`	Provides common useful concepts like sets, checks or canonizing.	0.1

Table 1.1: Contained expansions in escad and their functionality.

1.7 Design patterns (interaction concepts)

To get things done with escad, there are several tasks you have or want to do. In this section we provide you with common interaction concepts in escad. You also see commands, expansions or taxonomies which are needed for this. But detailed examples you will need in next chapter.

1.7.1 Modeling

Modeling is the way you describe escad your topic/task of interest. This is a very important task. If you model something, look at the given examples in the `examples` directory. Use the most approbiate taxonomy. Use expansions to ease your live where possible.

1.7.2 Hiding

To view graph networks it is sometimes possible you just want focus on special symbols. Or you want hide symbols which are for configuration tasks (e.g. *_view* or *_escad*). This is supported by hiding symbols. Note that the symbols still are active and the expansions still see them. It is only a presentation thing for the user. To hide a symbol create a relation to the _view symbol with taxonomy *has_hidden*. So you can read that as *this view has a hidden symbol X*.

1.7.3 Mounting

Sometimes you want hide complexity by defining another view which even may be on another computer/network/disk/file. To connect this seperate view with your actual view you can mount symbol(s) of it to your current view. This is like a door where you can hide complexity but if you need more you can open and connect your view with it. Mounting is done with a expansion. After usage you can unmount the seperate view. You would want to do this especially when a complex mounted view is connected through a slow network. Then escad will react slowly too, because it has to fetch/write data to it.

1.7.4 Exporting

Saves your view in a non-escad format. This is usefull for use your view in other programms.

1.7.5 Importing

Allows you to get a view stored in a non-escad format.

1.7.6 Reporting

Means to create/process a (new) representation/summary of your view. E.g. your view could be a description of a 3D-modell, and with reporting you could get a 3D-file in X3D-format. A report could also check your view for quality (e.g. if there would occur problems when activate a specific expansion).

1.7.7 Generating

Is used to create new symbols, relations or attributes automatically for you.

1.7.8 Automating

With *flows* automating in escad is possible without to know how to programm in lisp. This also enables live-expansions which periodically activated generate symbols to reflect the state of a continous process.

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 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 16. 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 (included examples explained)

To get a fast idea how you should work with escad, please read this section carefully. It gives you general hints how to work, and useful example-patterns to make the theory more clear.

2.2.1 3D (escad version 0.2)

Generate 3D-models.

2.2.2 Contact tracing (escad version 0.2)

Tool to provide contact tracing, useful by tracking viral deseases (e.g. corona).

2.2.3 Documentation

Create (technical) documentation in latex or PDF file-format (reporting).

2.2.4 Exporting

In case you can not do some things in escad yet or other tools may do better, you can export your graph. You can export the view to:

graphviz-dot is a file format for graphviz, which is powerful in graphically layout and drawing of graphs. Export with a expansion.

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

PDF Export with a expansion.

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

2.2.5 Flow

Note: this is not working yet, because it is in development! A flow is a graph wich models/automates a process. You can use escad to create a process a user should go through (e.g. a question+answer quiz) or a automation which escad should do for you.

2.2.6 Importing graphviz dot

Because interoperability is important in todays heterogenic software world, there should be a way to get graphs from other software. 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.

2.2.7 Learning page (version 0.2)

Standalone-HTML-javascript learning page generator Note: this is not working yet, because it is in development! 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.

2.2.8 Lessons learned

You model e.g. deseases and their dependencies. Then you can try to give symptom(s) and ask for possible cause-tracing.

2.2.9 Music/sound creation (version 0.2)

Todo...

2.2.10 Mind-map

Todo...

2.2.11 Modelling

Modelling semantic graphs is not a easy task. Of course you can make easily symbols and relations, but it heavily depends how you use them. In the past many semantic projects are died or got not very sucessfull used at a wide range. Barry Smith stated some reasons (`https://www.youtube.com/watch?v=pObuEjR3t8A`) like silo-syndrome, short-half-life-syndrome and reinvent-the-wheel-syndrome. Nowadays there is a sucessfull example in bioinformatic where to show how genes affect our biology (see BFO-ontology). However even with the best ontology you can have problems, if you not use it properly. To use it you should have a clear knowledge on what the terms mean and how to use.

Such problems may also occur with escad. To improve the situation, escad tries to have many examples for several domains and tasks. The aim is to have enough examples so that you have the possibility to adapt it easily to your needs. I believe it makes more sense to use semantics than to have endless discussions what is the ”correct” taxonomy/ontology.

2.2.12 Pedigree

Todo...

2.2.13 Quiz

Todo...

2.3 Questions

Note: this is not working yet, because it is in development! 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 start terminal
>
```

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

```
user@host:∼/escad$ ./escad 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 11 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 currently possible for experienced users via slime-mode in emacs.

3.3 Tk GUI

This is in development but currently in sleep, because other user interfaces are prefered.

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. However it is *not* intended to acess over external network, instead you should use it locally. If you need external acess use virtual private networks or the REST-API.

3.5 Connector-API

This is in development. The API is especially meant for a structured and safe acess to escad without the possibility to execute (potential more powerful/dangerous) lisp commands. It can connect graph-databases (planned is arangodb) with escad. A REST-API (even if it is not a standard in itself) uses standards such as HTTP, URI and JSON. With the paradigm CRUD (create, read, update, delete) there is for the user a clear set of actions available, which can be used on the objects in a view. HATEOAS (Hypermedia as the Engine of Application State) allows the consuming applications a guided way through the complexity of a graph.

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. Note that currently this part is less actual, since we first want concentrate on the big picture - before diving in the details. However details can be documented in parts which are relatively mature.

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.

4 Development

Everyone is needed and welcome for escad development. If you are a graphical designer, you are got in documenting or you like to program in lisp or you are interested in web-programming - all is required in escad. :-) The project is managed with the famous source-code management tool *git* (also known as used in the linux kernel development). The repository is hosted under `https://github.com/mkollmar/escad`.

4.1 Directory structure

The directory structure of the repository will be explained in this section. This structure also mirrors the different workflows which exist.

4.1.1 doc/*

Contains like the name says the escad documentation. Currently this is mainly this manual written in *latex*. But in future a unix man-page, more examples or online documentation would be great, too. The compilation of this latex-document can be done with the command

```
user@host:~/escad$ latexmk
>
```

. `doc/figures` contains figures needed for the manual. Those will be compiled if needed automatically via *latexmk*.

4.1.2 examples/*

`examples` contains examples which can be loaded in escad. But note that currently not all may work yet, they are more a print what a future interface should look like.

4.1.3 lib/*

`lib` contains the escad-lisp files.

4.1.4 node_modules/*

`node\_modules` contains the code-modules for node.js. Some functions are written for node (a javascript execution environment, similar the one in your browser). The REST-server is a example that needs node.

4.1.5 public/*

`public` contains the files which are accesible via web by the REST-server.

4.1.6 test/*

`test` is needed for development to test current implementation if it fullfills the requirements/functions needed.

4.1.7 ui/*

`ui` contains files needed for the various user interfaces that escad may support now or in future.

4.2 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 4.1 shows a minimal expansion and the listing 4.2 shows the entry in a taxonomy file.

Listing 4.1: 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 4.2: 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

app, 8
attribute, 8

command example
- lr, 11
- ls, 11
- nr, 11
- ns, 11

command reference
- as, 15
- asa, 16
- gra, 16
- gsa, 16
- help, 16
- lr, 16
- ls, 16
- nr, 17
- ns, 17
- r, 17
- s, 17
- tv, 17
- vs, 17

complexity, 5

edge, 8

node, 7

ontology, 8

Bibliography

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