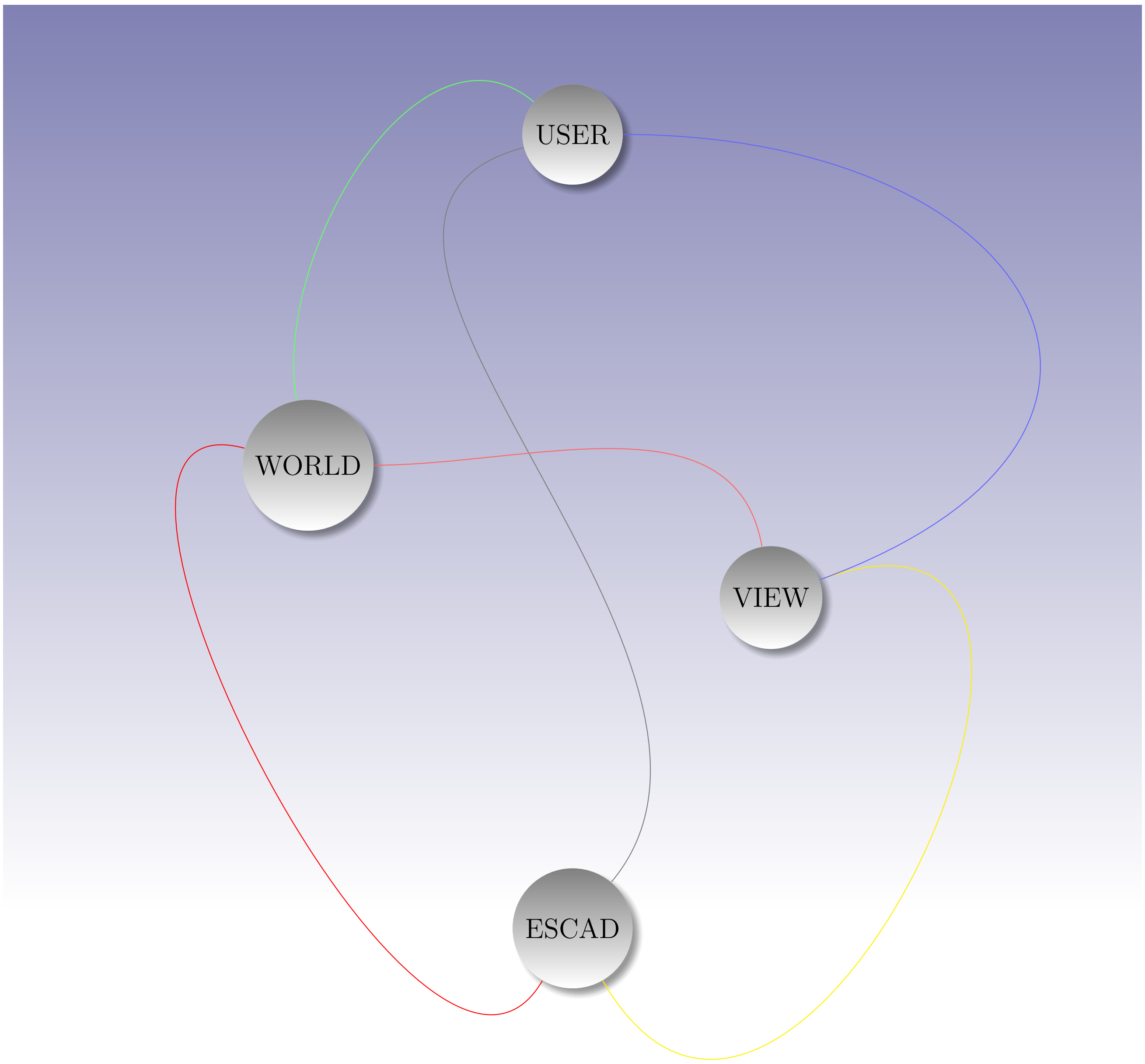
Manual for ESCAD version 0.1.0 - 16th August 2023

by: Markus Kollmar



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

Contents

1 Introduction

You have choosen interest in a friendly libre software. This software is aimed to be a workhorse for many tasks which are usefull to do with the help of graph-structure. However to be honest this escad version 0.1.0 shows that there is still much to develop. So some things may not work as expected or there are silly mistakes around. But i think it is better to regularly update.

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 some manuals, escad tries a documentation-driven development process, which will be explained in the next section. This makes this documentation a up to date view of the current escad and shows you what can be awaited from escad and what not.

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 important. The bounds of our disciplines are drawn by human but it not need to reflect the reality in nature. 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. Furthermore real problems are mostly system-problems or are increasingly seen as such. Systems are combined of different disciplines. But do the (software) tools support this fact? Some may but many do not. Escad does! Escad allows theoretically to model various disciplines in one software tool!

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.

1.2 Documentation-driven development

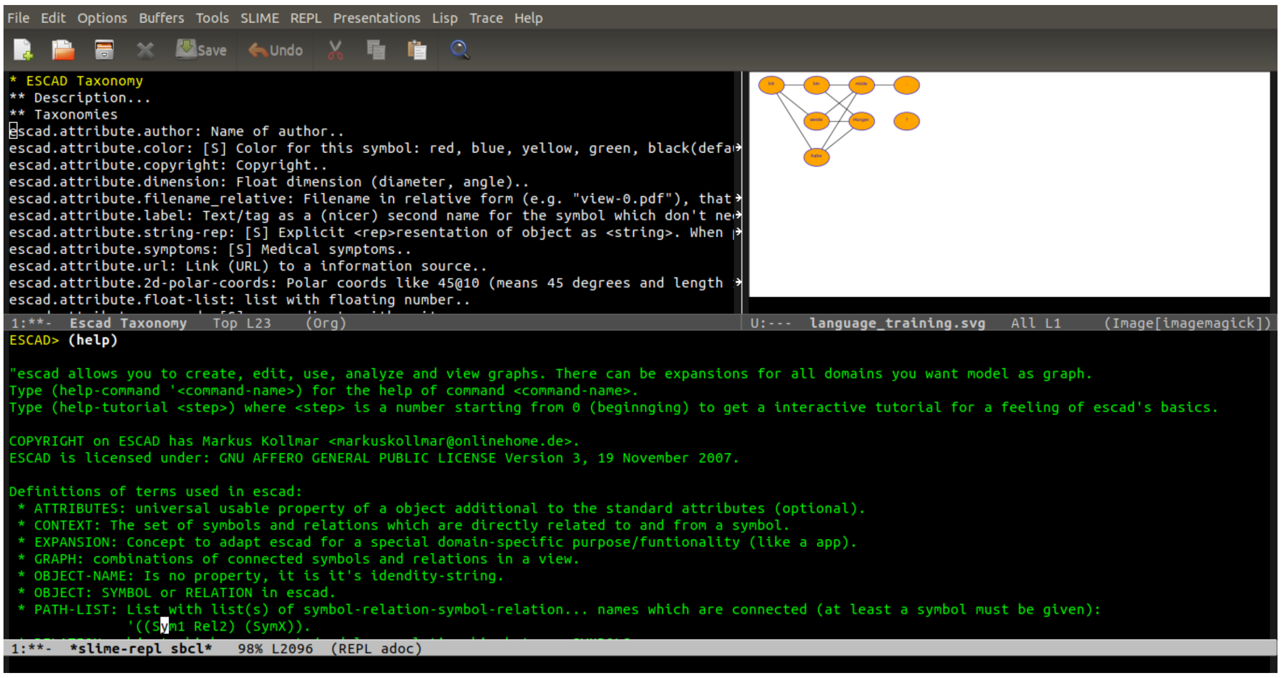
Documentation-driven development means here (at least we try), at first we define in this manual what we want to develop mostly from the user perspective. Then the feature will be implemented according this documentation. This is an advantage for the user, since he can see what a feature will look like, and documentation is ahead and not behind the actual implementation. The specified features 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. Features which are in the software but not documented here, should not be used or rely on, because they may change or are not ready for production yet.

1.3 State and screenshoots

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. Basically you can use escad in two ways:

- Command line entering common-lisp code, here in the editor emacs, see figure 1.1.
- Graphical user interface via web browser, see figure 1.2.

Escad use open technologies 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.

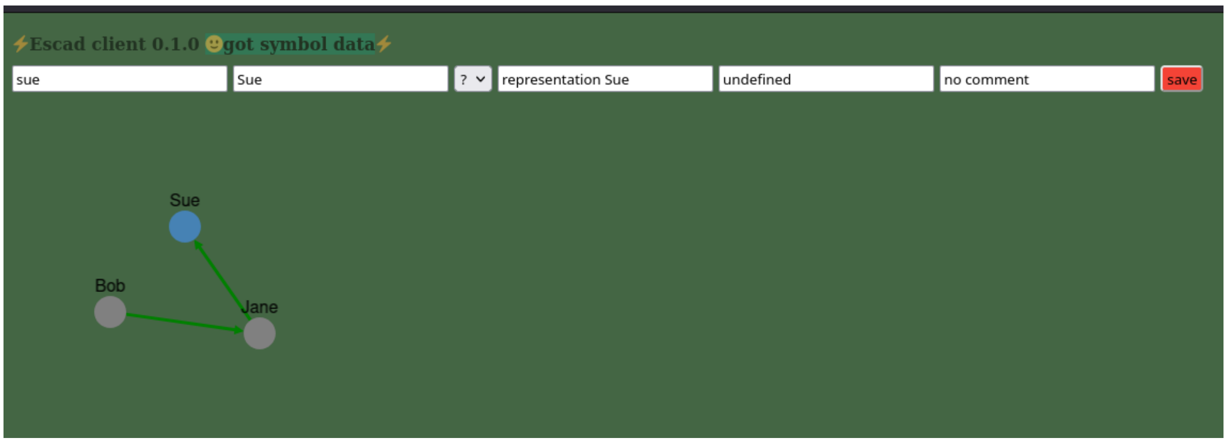

File Edit Options Buffers Tools SLIME REPL Presentations Lisp Trace Help
Save
Undo

```
* 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 (note the appearance may differ from yours).

1.4 License

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.


⚡Escad client 0.1.0 🙂got symbol data⚡

sue | Sue | ? | representation Sue | undefined | no comment | save

Figure 1.2: escad via graphical web-user-interface.

1.5 Installation and starting

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 unix-like systems

1. Get the repository from `https://github.com/mkollmar/escad` by clone it, or download repository as zip-file and extracting it. The escad executable is at the root folder and is called `escad`. If you want make the excecutable yourself (in the case the preinstalled does not work), then you currently need **sbcl** common-lisp implementation installed with your distribution-package-method. In the root directory of the repository execute `make executable` and after some while you will get it.

2. Optional: For some PNG/SVG export functionality you need to have/installed **graphviz** and for PDF exports you need to have/install a **lualatex** installed with your distribution-package-method.

3. You can now try to run escad (in terminal-mode) within the escad-root-directory. If you want the command line interface mode then type just `./escad`. If you want use the browser gui then type `./escad webgui` and enter in your browser `127.0.0.1:8529/`.

4. Optional: To make initial settings for escad check the config file `escad_conf.lisp` and edit if necessary. This file will be read and executed at start of escad and contains escad lisp commands, like the one you type in at escad command line.

1.5.2 Other systems

Sorry, for now other systems are currently not tested, but may be possible to work, since common-lisp is available for wide range of os. However some additional tools may be not available or have different behaviour so that not all things work like under linux. Another way is trying to start linux in a virtual environment, but knowledge is needed. If there is more man power it may be possible to improve that situation.

1.6 How it works and terms

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 graph data structure (in informatically or mathematically means). But simple means just in the basic building blocks, not in the power, since graphs can be quite big and nested. Such **graphs** consist only of **symbols** V (vertices, nodes, Knoten) and **relations** E (edges, Kanten). Mathematically (see also Bronstein et al., 2008) 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 uses 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. See figure 1.3 about the structure of escad. The three grey blocks are needed for the interaction with the user (commandline or web interface). Thus they are explained not here but will be looked later in detail. In the following sub-chapters the other three parts and their contents are explained in detail. However

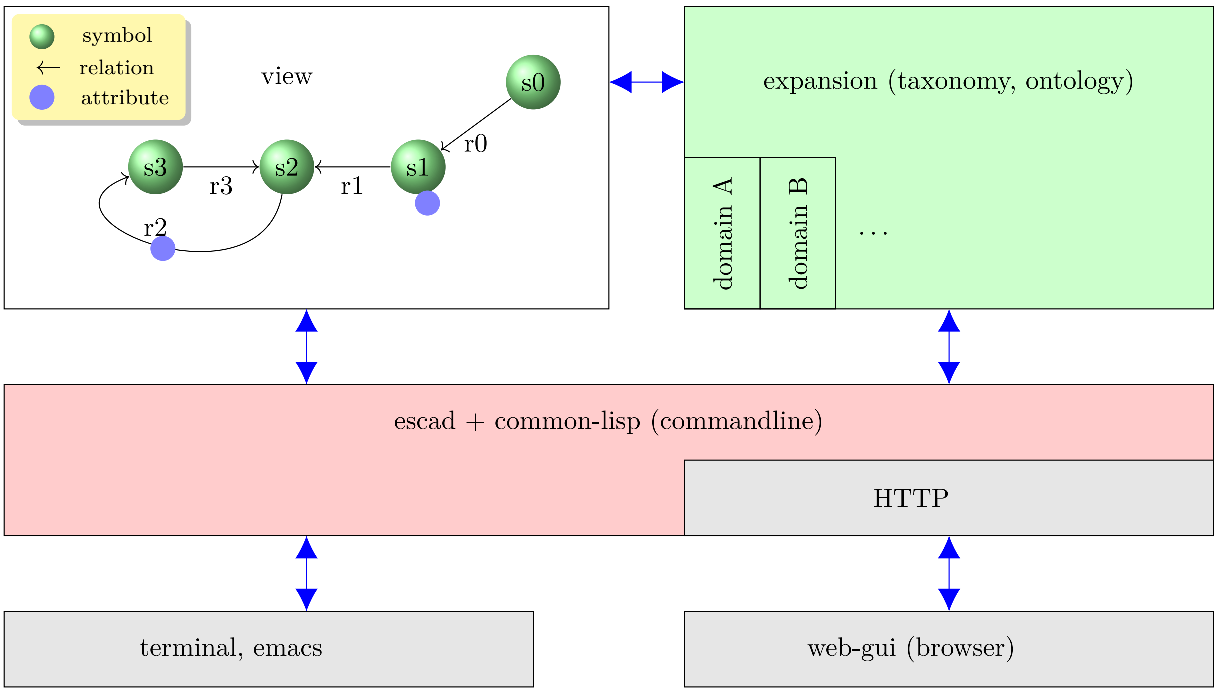


Figure 1.3: Overview of basic escad structure.

a short additional security note on the communication blocks: You can communicate with escad through common-lisp. This communication can also piped through a local TCP-connection by the user. Note that for security reasons you should not make such network-communications over the internet, since escad is not encrypted. This is also true for the HTTP connection. It is currently only meant for local network use. However you could use technics like VPN if you want secure transfer through the internet.

1.6.1 Symbol

In many papers a symbol is also called node. However escad has the aim to model things into software - a symbol for a concept. Thus a node is a *variable* for something in our thoughts. This symbol can represent a house, a number, a theory, a joke or whatever you want. This shows the great flexibility of escad. In figure 1.3 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 even 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. Try to use the most specific domain expansion for your model in order to get the most advanced tools. However sometimes the detailed model is not clear in detail so that you can use a most generic relation taxonomy.

1.6.3 Attribute

An attribute in escad is an *optional* specifier of a symbol/relation. There can be a huge discusion whether attributes or whether a symbol and relation should be used instead a symbol-attribute. Attributes can lower the complexity of an graph. On the other hand attributes are implicite relations, perhaps sometimes it is more useful to state explicitely relations (e.g. for graphical purposes). Take your choice and try to keep this choice consistent over your graph. Some 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 View

You can think of the view as a workplace where you can put in symbols and relations, which model an part of the real world. It is called view because it is the current view of a world modelled in your escad session. However like in the real world there are often multiple and different views of the world. Thus you can acess other views through a view-symbol in the standard expansion mentioned later.

1.6.5 Taxonomy

With taxonomy you can create in escad a classification (domain-based) to your symbols and relations by just tagging them. Depending of the taxonomy there may result different (graphical) output or behaviour (functionality) by escad. This may sound clear, since without that a computer can not know what a symbol means just by interpreting its name. But it is merely like a seperator in your folders - you can name and seperate things but you may not do much more. More is possible with the ontology.

1.6.6 Ontology

In semantic-knowledge field you may hear often words like ontology. Ontology *uses* the taxonomy to capture and represent the meaning of a domain and has rules what is possible with it and what not. Since escad is aimed to be a practical tool, it has some domain specific ontologie-knowledge included. This knowledge is provided by an expansion.

1.6.7 Expansion

Expansions are a collection of definitions of *symbols*, *relations* and *functions*. They provide mostly domain specific *ontologies* which assure that one can model with a specific taxonomy and rules of who symbols and relation are combinable and which functionality they provide. 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.

When we speak of *activation* of a expansion, what does that mean? You as user give a activation command or right-klick on the symbol, then a special function is executed. What special function is that and what does it make? Well in fact it is the *activate* function. This function can take additional parameters. You can specify them within the Attribute *activation_command*. The default value is *neighbourhood* which does what it says: It shows all in- and outgoing relations and the symbols which they are related to.

1.7 Usage patterns

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. It is very useful that you get familiar with them even if you do not know the exact details. So you know what tasks or problems may arise and which facilities escad provides and suggests for this. Detailed examples you will see in the next chapter.

1.7.1 Modeling

Modeling is probably one of the tasks you will do the most time with escad: You describe in 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 Structuring

The more you model, the more your view will get filled and it is hard to keep an overview (especially in the graphical user interface). You need to handle complexity (and in the end also performance). One method is *grouping* symbols which are in close relation to each other. For example if you model a 3D object like a cube, you may group all symbols defining the cube together. This can be done by the group-symbol.

1.7.3 Exporting

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

1.7.4 Importing

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

1.7.5 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).

1.7.6 Generating

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

2 Getting started

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 with escad-commandline

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 17. 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 load stored views or escad-lisp-scripts (e.g. in the *examples* directory) type:

```
ESCAD> (lov ``mindmap.lisp'')
(...)
```

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

2.2 Tutorial with graphical browser-interface

We want now do a web-session in escad. Go to the escad root directory and start escad by typing in a shell in your terminal:

```
user@host:~/escad$ ./escad start net-http
>
```

This starts escad and a server. With your web-browser you can browse the page which at first time loads a web-client in your browser. After that you can view and edit the graph like in figure 1.2.

2.3 User stories

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.

2.3.1 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.

2.3.2 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.

2.3.3 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=p0buEjR3t8A`) 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.

2.4 FAQ

Sometimes questions lead to a fast recognition of a problem or help to understand things better. Not all questions may be frequently asked, but who cares :-)

2.4.1 Scope

I really do not understand whether escad is for me? Do you search a nice software to make crazy pictures of your graph in a interactive way? Then you currently may not be right here, just look at something like gephi. Escad is for people who want to work exactly and reproducible with graphs in a more describing manner and in a second step there will be an output (similar like batch processing).

Can i make complex queries like in a graph database? Perhaps you need a graph database, escad is not a pure graph database. It is a tool, not designed as a speed optimized storage container. However it is planed to have a graph-database connection (e.g. to great arangodb) in order to support persistent huge graphs.

2.4.2 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. The aim is to keep the functionality equal across the interfaces in a later step. But each interface has it strength, thus it makes sense to use all. Where the commandline has powerful scripting capabilities, the web-interface allows a more intuitive and haptical way to interact with the graph and shows a fast overview.

Why you have choosen common-lisp as the language for escad? While having some experience with different languages, lisp has one of the clearest syntax for me. The REPL allows interactive rapid prototyping. Lisp is an old but mature language. One can implement easily a domain specific language (DSL) if needed.

2.4.3 Development

Can i help develop escad? Of course, very welcome! This currently is a project done in spare free time beneath work. So just contact the developer in github to improve escad together.

2.4.4 Other

Is there support for other languages as english? No not currently. The main author is german, but has spare time to keep manual in german as well. Currently english is choosen to reach a wide international range of people. If you want contribute with translation for your language 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

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)
...
```

The table 3.1 explains used symbols, abbreviations and the data-structures in the reference. In general escad provides two flavours of commands: object- or text-based. There is no superior flavour, just use the one which fits your concrete needs the best. This command reference is sorted alphabetically, so the order of commands does not say something about the quality or importance of them.

ad

```

```

```
STRING
```

<a>nalyze <d>iameter (length of longest path(s)) of current view and returns multiple values: the diameter of longest path(s) and the path(s) itself.

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''
cmd	command name	**asa**
(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''
RO	relation object	``r0''
SO	symbol object	``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 Library

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

3.3 HTML-webclient

Take your browser to communicate with escad in a graphical intuitive way.

3.4 Included expansions

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

Table 3.2 shows the available expansions included in the current version of escad. Of course there can exist many more not shipped one, but you should check this in escad yourself by exploring the _*escad* symbol (how this is done you can see in the 2). This is always the most up to date state.

topic	**file**	**description**	**since version**
export	`export_expansion.lisp`	Export graph to PDF and SVG.	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

Table 3.2: Contained expansions in escad and their functionality.

taxonomy is the unique name of the expnsion.

in shows which relations are allowed/make sense to point to the expansion. If there is none, the symbol is meant to stay unique.

out shows which relations are allowed to point out of the expansion to other symbols/expansions. If there is none, the symbol is meant to stay unique.

attribute which are allowed/expected/needed.

description tells what the expansion does.

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 3.3: Symbols and relations of export expansion.

3.4.1 export_expansion.lisp

The table 3.3 explains the export expansion.

3.4.2 generator_expansion.lisp

Todo...

3.4.3 import_expansion.lisp

Todo...

3.4.4 report_expansion.lisp

Todo...

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`.

A *Makefile* contains the basic task to generate the code or documentation. So it may be a good choice to look into it in order to get a basic idea how all works. Nevertheless the next sections contain how the development process is (currently) structured. It follows a brief overview over the directory structure and a explanation what it contains (or should contain in the future). The last section provides information in how to program a expansion and the interface

4.1 Development process and quality

As mentioned at page 4 escads development follows the documentation-driven approach. All follows this process in short:

1. Set version according to semantic versioning (Preston-Werner, 2022)
2. Update/create documentation (this manual)
3. Develop along documentation
4. Test
5. Tag version and deliver it.

A note to tests as it seems to become one big factor of good code quality. Surely test are a good method to check well defined interfaces. So here we implement tests too. However because at the early development stage of escad and limited man power, tests will not take as much room as there could be. Since interfaces may change often we will concentrate at often used functions and in more stable versions of escad the tests should be increased.

4.2 Directory and files

The directory structure and some basic files of the repository will be explained in this section. In the root of the repository there are following files with their purpose:

Makefile contains the instructions for the *make* programm and controls the most part of escad development process (generating documentation, binary or other things). Just enter `make` in the command line to see available options.

LICENSE holds the license to which terms you have to use escad.

README.md is mostly for github where escad is hosted and contains shor overview about the project.

4.2.1 doc/

Contains the escad documentation for user and developer. 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.

4.2.2 lisp/

`lisp/` contains the escad-lisp files with the implementation.

4.2.3 lisp/expansion/

`lisp/` contains all escad-expansions which are available.

4.2.4 lisp/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.

4.2.5 misc/

Contains miscellaneous things which have no other place where it fits.

4.2.6 test/

Here there are unit tests, which execution may be controlled with make.

4.2.7 web-gui/

The web based graphical user interface (gui) things, which are executed in the web-browser, live here.

4.3 Programming your own expansion

Writing escad expansions means inherit some escad classes and refine them. 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.

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.")
```

Following things you should keep in mind:

- Choose namespace.

- Provide documentation within your expansion.

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

Index

Bibliography

Bronstein, I. N. et al. (2008). *Taschenbuch der Mathematik*. 7th ed. Harri Deutsch GmbH. ISBN: 978-3-8171-2017-8.

Preston-Werner, Tom (2022). *Semantic Versioning 2.0.0*. URL: `https://semver.org/` (visited on 23/01/2022).

List of Figures

List of Tables