ERMaker

Final Report
ICOM 4036
Prof. Wilson Rivera
Lumaris C. Ríos Meléndez | Luis I. Padró Rodríguez

Introduction

Entity Relationship diagrams(ER-diagrams) are typically used in computing to organize data within databases and information systems. These graphical representations of entities and their relationships to each other are particularly useful during the conceptual-design and review phases of databases.

Often drag and drop software as to other programming languages, are tedious to use, resulting in complex learning processes and time consumption. ERMaker is a high-level programing language developed as an alternative for drawing ER-diagrams.

Language Tutorial

ERMaker is capable of creating entities, assigning attributes to them, as to relate entities with each other through relationships. Watch the following video to get started.

INSERT VIDEO LINK

Reference Manual

Syntax

Due to its strong abstraction and resemblance to the English language, drawing ER-diagrams with ERMaker is very straightforward. ERMaker offers two functionalities.

1. <Cardinality><Entity><Modality(optional)><Relation><Cardinality><Entity>
   This expression creates the entities with the names provided in the <Entity> delimiters.These two entities are related by the relationship given in the <Relation> delimiter. If any of the entities or the relationship is not previously created they will be.

The cardinality of the relationship has to be assigned within the <Cardinality> delimiter. Cardinality can be expressed with the reserved words:”many”, “one” or “a”, where “many” represents a many side, and “one” and “a” represent one side. The leftmost <Cardinality> will be assigned to the end of the leftmost <Entity> and the rightmost <Cardinality> will be assigned to the rightmost <Entity>.

Modality is an optional field. It can be specified with the reserved words: “must”, “can” or “maynot”. If the modality is not specified “can” will be assigned to the relationship.

2. <Entity><composed|composed of><Attribute Set (comma delimited)>

   This expression allows the user to add an attribute or a list of attributes to an entity. If the entity has not been created yet the program will create a new one with the name provided in the <Entity> delimiter. All entities and attributes must be named differently. The reserved words “composed” and “composed of” will serve as identifiers for this function.

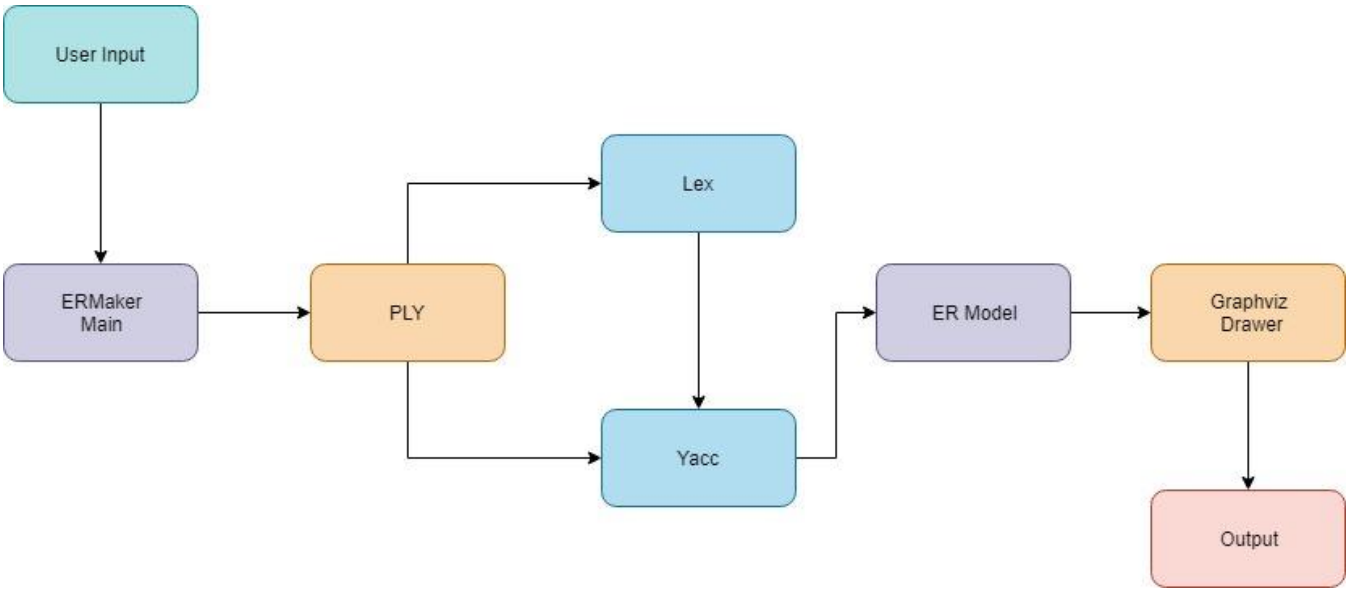
Reserved Words

All entities and relationships must be named differently. The following is a list of reserved words and phrases:

Cardinality	A One Many
Modality	Can Must May not
Entities	Any Noun
Relation	Any Verb
Other	Composed Composed of

Language Development

Architecture



Modules and Interfaces

The libraries that comprise the application are PLY and Graphviz . PLY contains the Yacc and the Lex modules that are used as the base for the implementation

of the parser and lexer. Then, the Graphviz library was used to render the graphs by using raw dot language to describe them.

Our custom modules include ERLex, ERModel and ERGraph. The ERLex module contains the main code of the application, which implements the different elements of the language in the lexer using Lex and the grammatical elements using Yacc. The ERModel module contains the definition and implementation of the graph object used to store the model for the ER. It consists of methods to instantiate a graph and add, edit, and traverse through the nodes and edges, where nodes represent entities and edges represent relations.The ERGraph module contains the elements needed for the construction of the raw dot structure, required to draw the graph in graphviz.

Development Environment

Python 3 - the language used for development.
PyCharm - the main IDE used for development.
Github - used as our Version Control to track changes and as a collaborative tool.

Testing Methodology

A bottom-up testing approach was used. For inner modules like the lexer and parser testing was done individually. To catch errors when identifying tokens in the lexer, we tested all kinds of tokens to see how they were being categorized and compared the output with the expected results. For the parser, we tested how different groups of tokens fell into the different grammatical statements that were defined following the grammar rules. They were classified as expected.

For the middleware, we tested how different grammatical statements were being represented in the graph model. That is, if the grammatical statements were being added correctly to the graph instance, The entities where added correctly with their respective relations and attributes. The ERGraph module was tested by providing the ERModel model and verifying if the sketched diagram described precisely the modeled ER-diagram.

After testing each module and sections of the code individually, integration testing was performed to see how each module communicated with each other.

Test Programs

```
>> elephants composed of apples, pears, butterfly, dung, magic
>> children composed of diapers, tears, hellfire, hopes, dreams
>> one elephants can have one children
>> many manatees can birth many children
>> a sheep must regret one children
>> sheep composed of darkmatter, solace, evil, satan
>> print
```

Conclusion

ERMaker has taught us what the development process of a programing language unfolds. Python was easy to use, powerful and versatile, making it a great choice for a development programing language. Developing the syntax and implementing it using PLY was one of the tasks with the least of issues. Our biggest drawback was learning how to use GraphViz.

Further implementation of other ER-diagram components can be considered as a future project. Nevertheless, ERMaker is a complete high level programing language intended to support databases designers in the process of sketching ER-diagrams.