



CAPSTONE PROJECT: VIKER TRANSFORMATION SYSTEM

ViKER Group 2: Kouthar Dollie (DLLKOU001), Matthew Coombe (CMBMAT001) and
Carryn Joseph (JSPCAR006)

Abstract

KnowID: Knowledge driven Information and Data access is a knowledge-to-data architecture that extends the partial knowledge-to-data pipeline of ARM+SQLP into the knowledge layer by incorporating a conceptual model into the architecture. The intention of this is to keep use of conceptual models in runtime with large data stores, but within a closed world assumption that end users are familiar with. The ViKER Transformation System creates the bridge between the knowledge layer and the ARM+SQLP by using a set of rules to bidirectionally transform between conceptual models, i.e. enhanced entity relationship diagrams, and abstract relational models.

1. Introduction

The purpose of the project is to develop a knowledge-based data access tool that transforms an enhanced entity-relationship (EER) diagram to an abstract relational model (ARM) and vice versa. The objective of this system is to assist in efficient data management, to ensure that decision-making is done effectively and to accommodate for the changing requirements of organizational needs.

The ViKER Transformation System is a desktop application that allows for the bidirectional transformation of the 2 aforementioned model types. The system graphically displays the transformed model and provides the option to view a log of transformed model entities and their details as well as save an image of the transformed model. The rules for the transformation of these models has been taken from 'KnowID: An architecture for efficient Knowledge-driven Information and Data access', a report written by Pablo Rubén Fillottrani and C. Maria Keet.

This project undertook a combination of the traditional analysis, design and implementation approach and the agile software development approach. The initial stages of the project followed a more traditional approach with analysis followed by design, with the addition of constant client involvement and feedback. While working on the evolutionary prototype, however, the approach became more agile to ensuring the project met all the requirements and increasing client satisfaction

2. Requirements captured

2.1. Functional Requirements

The functional requirements of the system are displayed in the use case diagram below. Additionally, the use case narratives provided highlight possible scenarios that could occur when using the system.

2.1.1 Use Case Diagram



2.1.2 Use Case Narratives

| | | | |
|--------------------|--|---------|------------------|
| Use Case: | Input and Transform a Diagram Successfully | ID: 001 | Level: Important |
| Actors: | Primary: Knowledge graph developer | | |
| Brief Description: | The knowledge graph developer inputs a file to the system and selects "Transform" button. The system reads the input file and uses the information within to transform the ARM/EER diagram to an EER/ARM diagram. The system displays the fully successful transformation and compiles an entity transformation log. The user can then save the transformed model and check the log. | | |
| Preconditions: | Input file exists on computer | | |

| | |
|---|---|
| Postconditions: | Model saved Entity transformations recorded Transformation status stored |
| Related Use Cases: | Extends: Transforms ARM/EER to EER/ARM |
| Typical course of events | |
| Actor Action | System Response |
| 1. Knowledge graph developer (User) starts up system | |
| 2. User inputs file and selects “transform” button | 3. System processes file information and transforms the ARM/EER diagram to an EER/ARM diagram |
| 4. User saves model, checks transformation status or checks entity transformation log | 5. System displays transformed model, compiles entity transformation log and sets the transformation status |
| 6. User exits system | |

| | | | |
|--|--|---|------------------|
| Use Case: | Unsuccessful Input of Diagram into System | ID: 002 | Level: Important |
| Actors: | Primary: Knowledge graph developer | | |
| Brief Description: | The knowledge graph developer inputs a file and selects “Transform” button. The system attempts to process the input file. In the case that the developer didn’t input a file correctly, the system will detect lack of file and prompt the developer to reenter the file. When the file is successfully input, the system attempts to extract the relevant information. The system detects that the file is corrupt (or poorly formatted) and prompts the user to enter a new (different) file. | | |
| Typical course of events | | | |
| Actor Action | | System Response | |
| 1. Knowledge graph developer (User) starts up system | | | |
| 2. User inputs file and selects “Transform” button | | 3. System reads the input file and fails to find information due to corrupt/poorly formatted file | |
| | | 4. System informs the user that the file is faulty and prompts for a new file | |
| 5. User inputs a new file | | | |

| | | | |
|--------------------|--|---------|------------------|
| Use Case: | Failure to Fully Transform the Model Diagram | ID: 003 | Level: Important |
| Actors: | Primary: Knowledge graph developer | | |
| Brief Description: | The knowledge graph developer inputs a file and initiates a transformation with “Transform” button. The system processes the file, obtains relevant information on entities and partially transforms | | |

| | |
|---|---|
| | the EER/ARM diagram to an ARM/EER diagram. On completion, the system displays the partially transformed model and compiles logs of failed entities and successful entities. The user can then save the model or view the entity logs. |
| Postconditions: | Model saved Log of failed entities stored Log of successful entities stored Transformation status stored |
| Related Use Cases: | Extends: Transforms ARM/EER to EER/ARM |
| Typical course of events | |
| Actor Action | System Response |
| 1. Knowledge graph developer (User) starts up System | |
| 2. User inputs a file and selects "Transform" button | 3. System reads the input file and uses the information to partially transform the ARM/EER diagram to an EER/ARM diagram |
| | 4. System compiles an entity transformation log, compiles a list of entities that failed to transform, and sets the transformation status to "Failure". |
| | 5. System displays the partially transformed model |
| 6. User saves the partially transformed model and chooses to view the transformation logs | 7. System displays entity transformation log and log of entities that failed to transform |
| 8. User exits the system | |

2.2. Non-Functional Requirements

2.2.1. Operational Requirements

| Description | |
|---|--|
| Technical Environment Requirements | <ul style="list-style-type: none"> Working computer with at least 0.5MB of memory. Java Development Kit (JDK) preinstalled on computer System accepts only XML files created by Draw.io |
| Portability Requirements | <ul style="list-style-type: none"> Use of system requires the source code available in a JAR file |

2.2.2. Performance Requirements

| Description | |
|---------------------------|---|
| Speed Requirements | <ul style="list-style-type: none"> File input speed dependent on file size |

| | |
|----------------------------------|---|
| | <ul style="list-style-type: none"> Transformation and log display speed dependent on the number of entities stored in the inputted file |
| Capacity Requirements | <ul style="list-style-type: none"> Maximum of one user can use the same system simultaneously System allows models with a max of 1000 entities |
| Availability Requirements | <ul style="list-style-type: none"> System is available to all users who have the ViKER JAR file |
| Reliability Requirements | <ul style="list-style-type: none"> System functionalities work 100% given the inputted files are in the correct format and the models follow the correct model rules |

2.2.3. Cultural and Political Requirements

| Description | |
|-----------------------------------|---|
| Multilingual Requirements | <ul style="list-style-type: none"> System is only available in English |
| Customization Requirements | <ul style="list-style-type: none"> System does not allow for customization |

2.3. Usability Requirements

1. Ease of learning

The system opens to a welcome page with one instruction – to input a file. Buttons that cannot be used at a certain time are disabled to limit the choice of actions for users who are unfamiliar with the system. For easy navigation, all functional buttons are placed at the bottom of the screen

2. Task Efficiency

When inputting a file, the system allows the user to manually navigate and search for the required file, however, files may also be inputted by inserting the file path to increase the task efficiency.

3. Ease of Remembering

When a transformation is performed it opens a new interface and the transformation log and entity details open into new interfaces as well allowing easy access to all data produced by the transformation

4. Understandability

A substantiable amount of understanding is dependent on the user's background knowledge on EER and ARM diagrams, however, labels throughout the system help bring light on what the system is showing and/or what the system has done.

5. Feedback

When inputting a file incorrectly, error handling provides feedback to the user on what went wrong and what to do. The system also provides feedback whenever a function is performed that makes no physical change to the interface.

3. Design overview

The ViKER system follows 2 architectural designs:

- MVC – Model-View-Controller
- Multi-layered architecture

These are essential for providing the user with a seamless experience, leaving all inner workings and computational logic hidden.

3.1. Layered Architecture

Figure 1 represents the layered architecture that is present in ViKER.

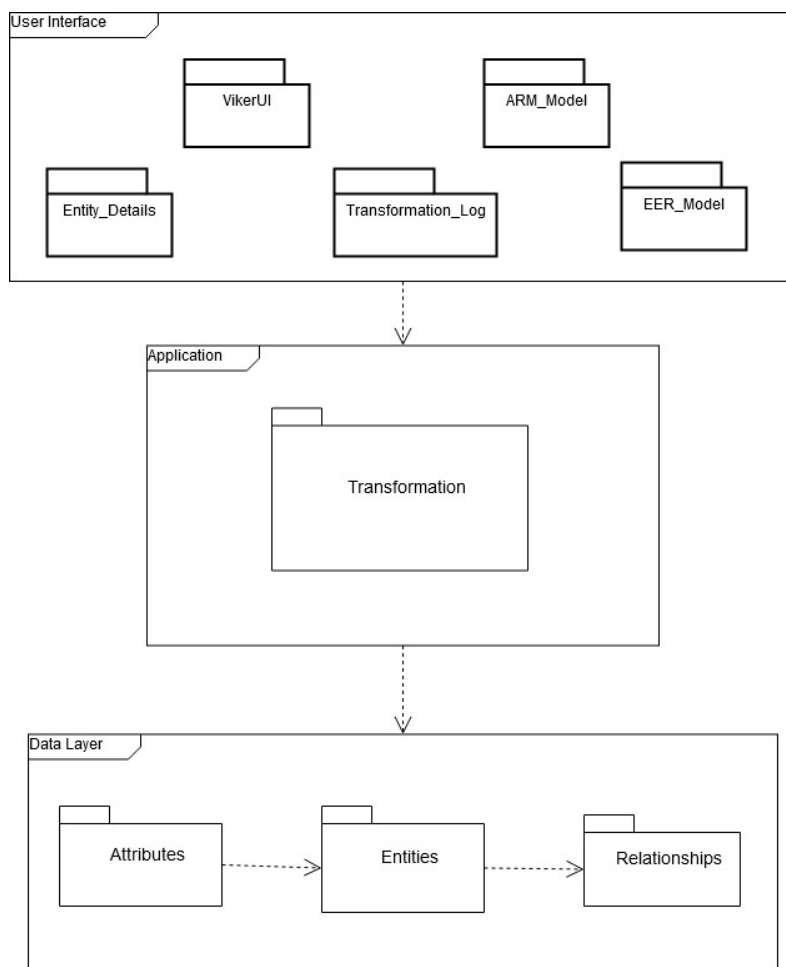


Figure 1: Layered A

User Interface Layer

- Made up of different panels determined by current program state
- Responsible for displaying important information to the user
- Developed using Java Swing tools

Application Layer

- Contains a controller class called Transformation which controls all important functionality
- Handles EER to ARM transformation and vice versa

Data Layer

- Stores the information extracted from input files (via Application Layer) into data structures
- Contains the frameworks for the data structures

3.2. Model-View-Controller

Figure 2 represents the MVC for the ViKER system. A user starts up the program and sees the VikerUI startup page and makes a file selection. This signals the Transformation to instantiate the Model. The Model will then transform the details, store the relevant data, and update the view to see the diagram that has been transformed. Since all details are stored in the model and the view is up to date, the user can view the details of entities, relationships, and attributes in the Transformation Log.

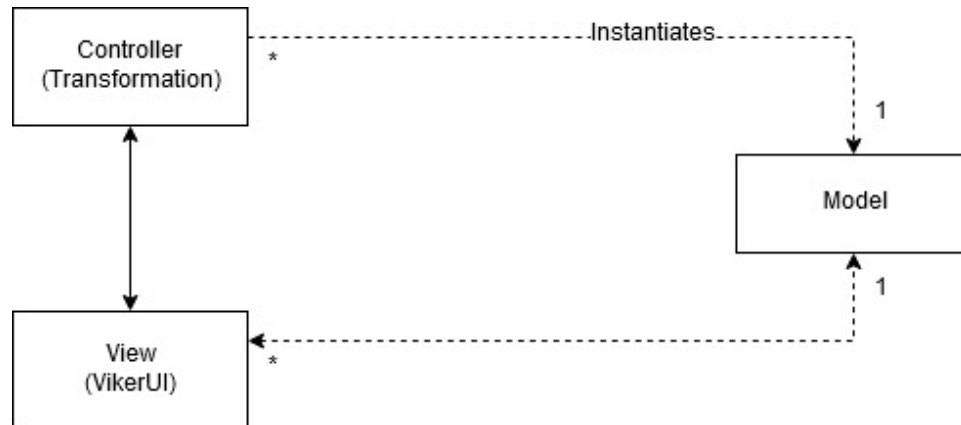


Figure 2: Model-View-Controller

3.3. Descriptions of Algorithms

There are 3 fundamental processes at large in the logic layer of the program. A file needs to be read, and information extracted from it, the information needs to be stored in the correct format in an appropriate data structure, and the data needs to be relayed to the user in a readable, easy to understand format. A large portion of this is string manipulation.

First step after reading an XML file is to process each line and check whether it represents an entity, relationship, or attribute. Once this is determined, more information from the line can be found such as whether the element is an identifier or not (based on underlined text or a star at the end), what cardinalities an entity in a relationship has, and the IDs of each element. The IDs are essential in determining uniqueness of a component and for drawing a finished diagram. Each element is stored as a component and each connection stored as a sourceID, targetID pair. When all the information for a line (entity/relationship/attribute) has been stored in local variables, it is then appropriately stored in Entity, Attributes, and Relationship objects. If information is not found for a detail, it is not stored in the data structure. This is for the user to see what information has been lost in a transformation.

This brings the process to the third step – relaying the information to the user. This is arguably the most involved part of the program. Since the input file format is XML, each element line has a setup giving information about what each element is connected to. These are represented by IDs for each entity, relationship, and attribute, as well as the IDs of connecting lines.

As all information for all entities, relationships, and attributes has been stored in the program, the only thing determining a transformation is the means of displaying the information. For displaying an ARM diagram, tables are created for each entity and populated with attributes. Primary keys are indicated with a star at the end. Relationships are simply connecting lines drawn on the panel between tables. For

displaying an EER diagram, displaying graphically is quite complicated and therefore the output EER diagram is displayed textually in an easy to understand manner.

3.4. Data Organization

For the storage of data in ViKER, we stuck to an object-oriented approach. The system makes use of the storing of entities, relationships, and attributes as objects nested within one another. The system makes strong use of encapsulation and abstraction and contains many methods to improve the flow and fluency. Many data structures are used, but the most prominent is the usage of ArrayLists.

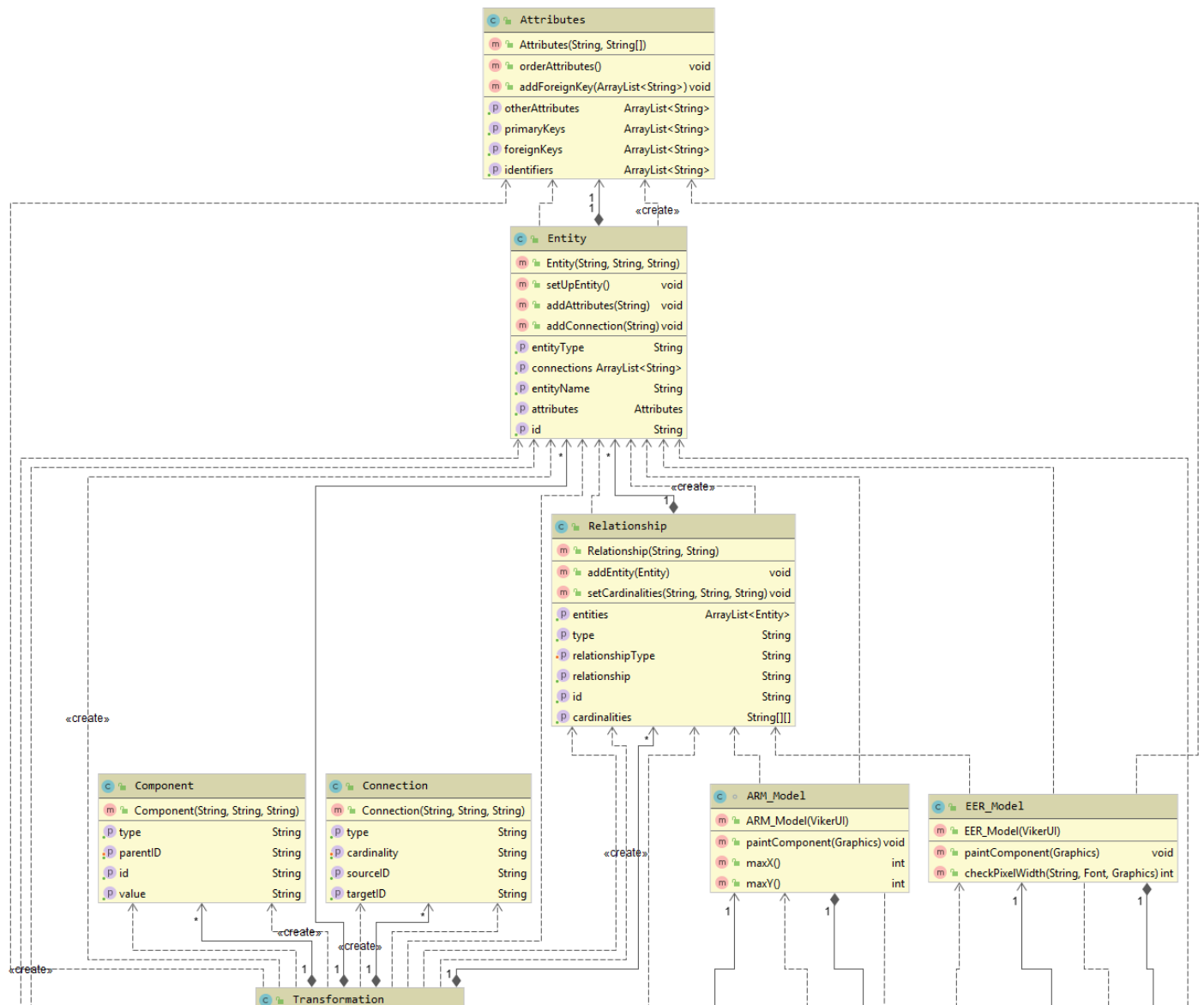
ArrayLists

ArrayLists are by far the most prominent data structure used in ViKER. It is essential to the core of representing the data to allow for storage of objects in lists. Relationships contain lists of entities to allow for n-ary relationships. Entities contain an Attributes object which contains 3 lists of attributes, each one pertaining to foreign keys, primary keys, and other attributes. ArrayLists provided an efficient means of storing and accessing data and their dynamic nature was important for the initial storage of data from an input file.

Usage of IDs

To allow for the possibility of components with the same name, allowing for a more robust program, ViKER makes use of unique identifiers for each component. Each relationship, attribute, and entity contain a unique ID to allow for the possibility of displaying the transformed diagram accurately and graphically. These IDs allow connecting lines to be drawn between the correct components. They were paramount in overcoming a struggle with identifying the correct objects.

3.5. Class Diagram





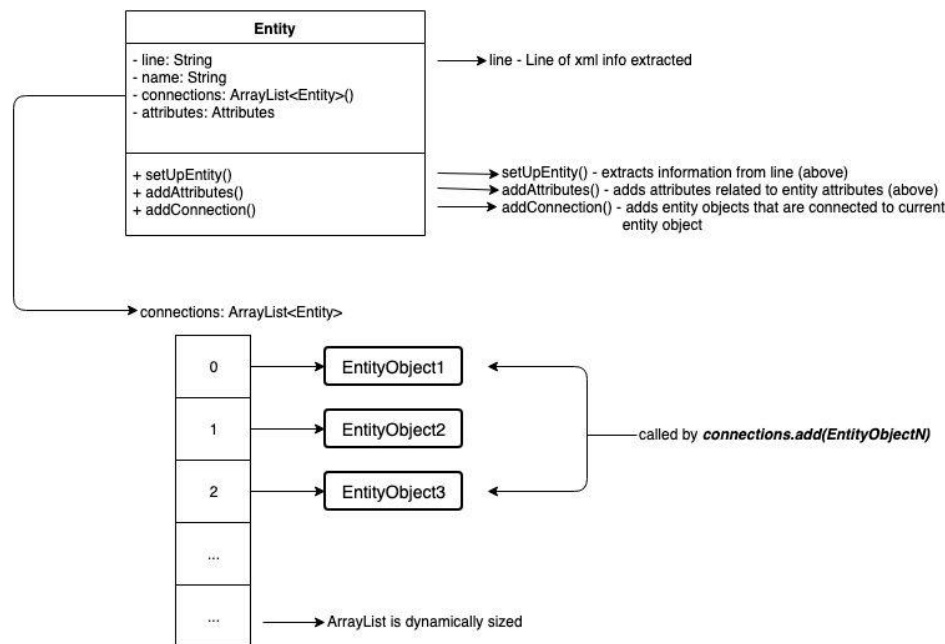
4. Implementation

4.1. Data Structures Used

The Viker Transformation Tool makes use of object-orientated perspective, which is indicative of information being stored in primitive and non-primitive data structures, as well as objects that represent real world ideas. Objects within the program are used to store relevant information in a collection of various data structure types, which are used to communicate information to users.

Entity Object

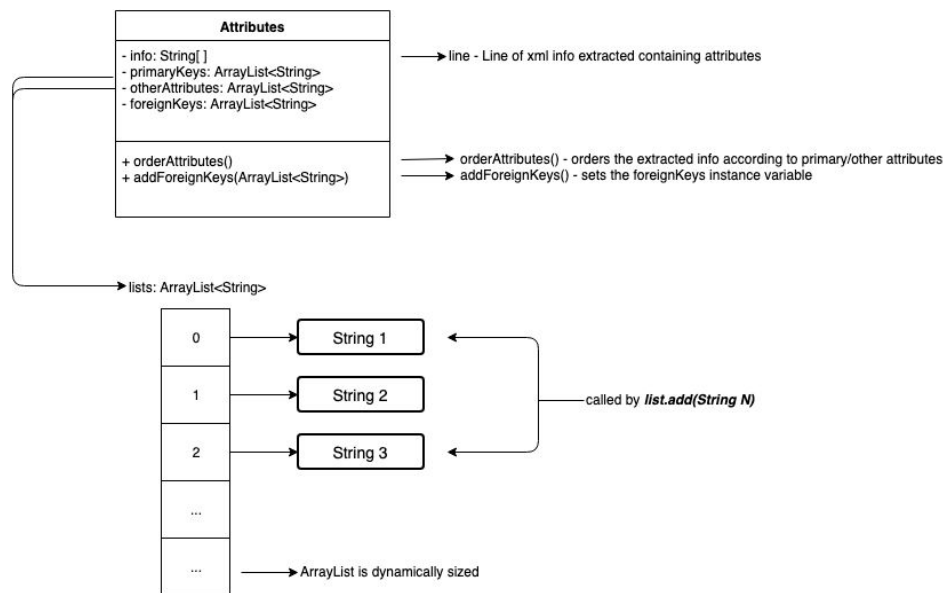
An Entity object is a class that was created to store the information about an entity that was extracted from the xml file. The collection of Entity objects is then traversed through to display the information of each entity to the user in the relevant user interface. It was easier to store all information pertaining to a specific entity within an abstract object, i.e. Entity object, than to store information separately in String objects and figure out which information relates to which entities. Subsequently, the Entity Object class is a real world representation of the entities that are found in EER models, and entity tables in ARM. Below is an illustration of how the Entity Object is used to store data (**NB. The following diagram does not follow the complete UML notation/standard; it is merely an illustration of how information is stored within the Entity Object; only shows relevant instance variables and methods pertaining to the storage of information**).



Attributes Object

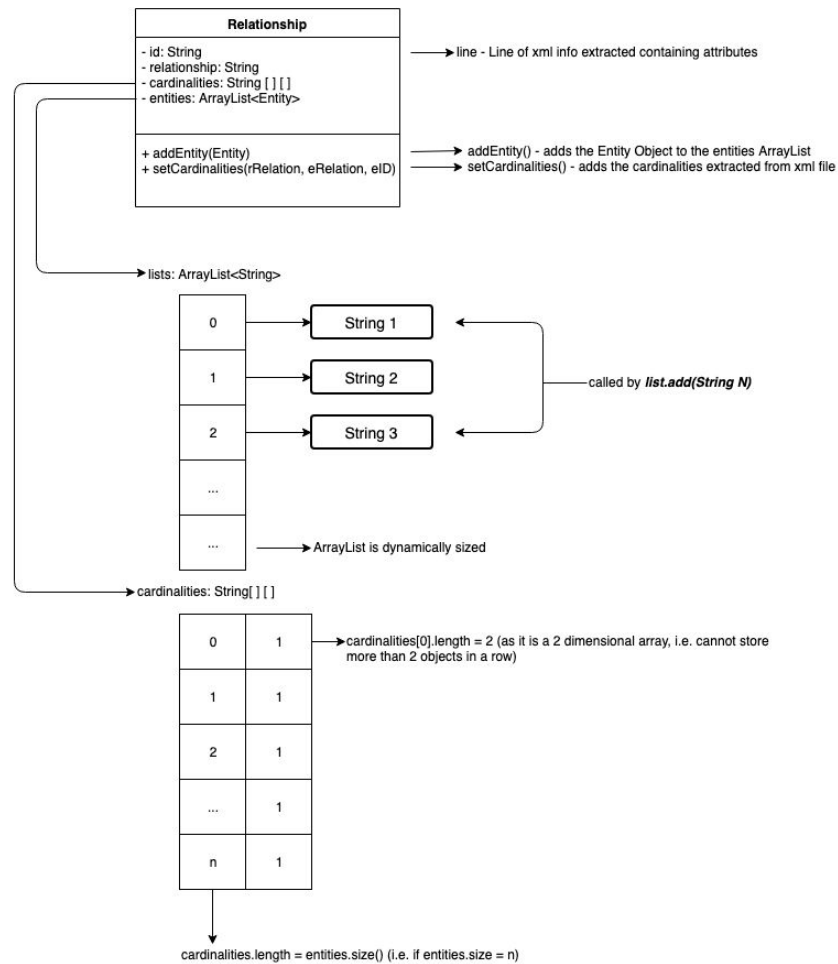
An Attributes Object is a class that is used to store attributes relating to a specific Entity Object, as well as sort the attributes according to their characteristic of being an Identifier or Uncategorized attributes in the EER model, or a Primary/Foreign Key and/or Uncategorized attributes. Each Entity Object uses an

Attributes Object to store the attributes related to the specific Entity Object. Since there are various types of attributes that would be extracted from the EER and ARM diagrams, it made sense to create an object that stores the attributes as well as sorts them accordingly. Below is an illustration of how the Attributes Object is used to store data (**NB. The following diagram does not follow the complete UML notation/standard; it is merely an illustration of how information is stored within the Attributes Object; only shows relevant instance variables and methods pertaining to the storage of information**).



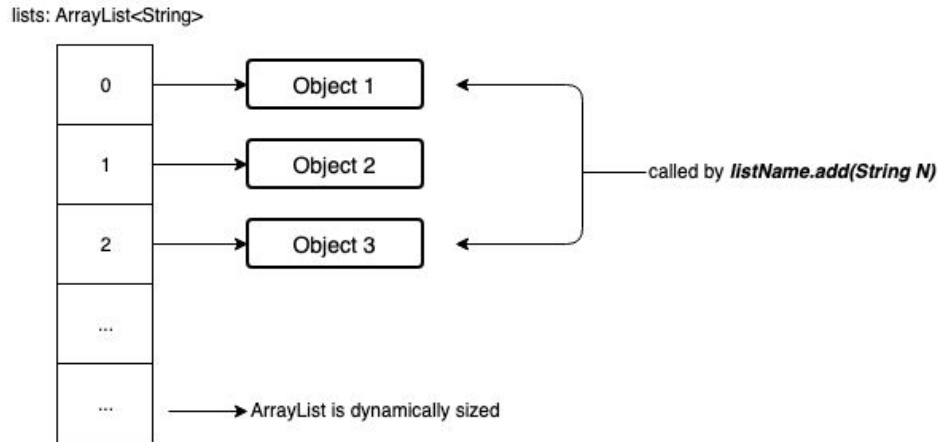
Relationship Object

A Relationship Object is used to store the information of each relationship component that is extracted from an EER and ARM diagram. The relationship components extracted would be connected to multiple entity components, within the diagram; this allows the program to store this specific information, instead of having to traverse through the input file, another time, to obtain information about the connections/relationships of entities. It, also, stores information about the multiplicities/cardinalities that exist between entities when dealing with EER diagrams. Below is an illustration of how the Relationship Object is used to store data (**NB. The following diagram does not follow the complete UML notation/standard; it is merely an illustration of how information is stored within the Relationship Object; only shows relevant instance variables and methods pertaining to the storage of information**).



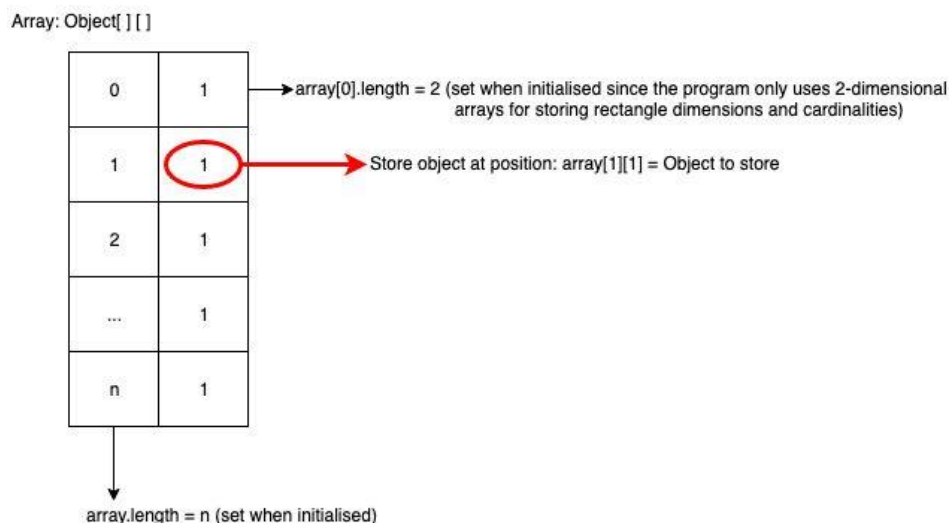
ArrayList

An ArrayList is a non-primitive data structure that was used to store collections of objects created, i.e. Entity, Attributes and Relationship Objects. Since the program did not require any major performance computations, due to the amount of information extracted from files, there was no need to incorporate an advance non-linear, non-primitive data structure, i.e. trees, graphs, etc. The positioning of objects within an ArrayList was not relevant as the traversal through the collections of objects did not require a specific order of items. In addition, the ArrayList has a dynamic size, which would be advantageous in most events of storing information that is extracted from diagrams, since the amount of information that is to be extracted from the xml file is unknown. Below is an illustration of how the ArrayList is used to store data (**NB. The following diagram does not follow the complete UML notation/standard; it is merely an illustration of how information is stored within an ArrayList; only shows relevant methods pertaining to the storage of information**).



Arrays

Arrays were primarily used for the drawing of the EER or ARM model within the JPanel of the transformation model. More specifically, it was used to store the dimensions of figures that were to be drawn within the transformation model, i.e. Rectangle dimension. When Arrays are created they need a fixed length, which would work in our favour when defining the dimensions of figures that are to be drawn onto the JPanel. This ensures that the dimensions only have two Integer values, i.e. width and height. It serves as a precaution in the event that an extra dimension is calculated as an error. Subsequently, it is used to calculate the coordinates, i.e. (x, y), of where entities and connections need to be made within the diagram on the JPanel. Arrays are, also, used in the Relationship class to store the cardinalities associated between components within an EER diagram. In this case, the size of the array would correspond to the number of Entity Objects that connect to the Relationship component, which reduces the number of errors that would occur by adding connections dynamically. It is a safer and more robust way of ensuring that the number of connections being added to the Relationship component is not greater than the number of entities that are connected to the Relationship component. Below is an illustration of how the Array is used to store data (NB. The following diagram does not follow the complete UML notation/standard; it is merely an illustration of how information is stored within an Array; only shows relevant methods pertaining to the storage of information).



4.2. Interface Development

The transformation tool developed requires a front-end user interface, i.e. the actual transformation that occurs from EER/ARM to ARM/EER should not be visible to the user. The user interfaces that were developed while keeping in mind the requirements that were specified by the client, which resulted in the following interfaces to be developed: 1) Welcome screen with functionality of inputting XML file, 2) Transformation Model display, 3) Transformation Log display, 4) Entity Details display. Since using Java as the programming language of choice, the interfaces and graphical components were developed using the Java Swing libraries, specifically javax.awt and javax.swing libraries.

The interfaces implement a Flat Design which is used to emphasize the usability of a program. The reason for this is that the transformation tool that was developed would, potentially, form part of a bigger system; it would therefore make sense to display the usability of the system as clearly and simply as possible. Not only does it emphasize the usability, it has a faster loading time, since the graphics are simple, has an efficient responsive design and the transformation content is represented clearly.

The program strives to adhere to a few Design Heuristics:

- Disposition of screen area: the components that need to be highlighted for users tend to be at the top/center of the screen and buttons at the side and/or bottom of the screen, which allows for additional functionality
- Orientation and navigation: the user is able to see the transformation model and, if needed, the details of each entity that has been transformed within the transformation log available
- Control and feedback: the user is able to input the file required, and the user indicates when the transformation is to take place, while the system indicates that the transformation has been completed by illustrating a transformation model diagram
- Input: only input required by the user is the xml file and clicking action which needs to be responded to, as the input required by users should be minimal to reduce attention needed by users
- Errors and help: in the event that user makes mistakes, the program informs the user that an error has been made and the user should either undo/redo the action or consult the user manual
- Readability and layout: program implements a flat design, which allows the user to easily decipher the functionalities available, as well as the layout being clear and simplified.

4.3. Important Functions & Methods

The most important methods that is utilized would be determined based on their involvement in the transformation process. The transformation process involves 1) extraction of information from XML file, 2) determine type of input model, 3) sorting of information, 4) displaying the transformation model.

VikerUI (JFrame)

It is the 'Welcome Window' displaying the welcome message and functionality allowing the user to input an XML file.

Methods:

- find_buttonActionPerformed(): find path of input file on device
- input_buttonActionPerformed(): 1) checks if file is of XML format, 2) reads file and stores file information in Transformation object

- `transform_buttonActionPerformed()`: enforces transformation in Transformation object and displays the transformation model window, i.e. `ARM_Transformation` or `EER_Transformation JFrame`.

Transformation

Acts as the controller class, i.e. handles all computations involving the storing and manipulation of data when transforming the input model.

Methods:

- `storeContents(String filepath)`: stores each line of file in array and determines if input model is ARM or EER
- `transformModel()`: depending on type of input model: 1) if EER → transforms into ARM, 2) if ARM → transforms into EER
- `EER()`: called if input model is an EER; sorts through extracted information and filters information into entity, relationship and attribute components and arrow/line connecting these components
- `ARM()`: called if input model is an ARM; sorts through extracted information and filters information into entities and all other components included in each entity table, i.e. attributes, relationships and dependencies
- `connectAttributes()`: sets the attributes for each entity object created from the input model
- `setRelationships()`: using the connections(arrows/lines) stored with the source/target ID's of each relationship object's entities that it is connected to is set
- `sortWeakEntities()`: traverse through weak entities and obtain the primary key/identifiers from associated strong entity and sets the foreign key/identifiers for the weak entity
- `setCardinalities()`: set the cardinality information for each relationship object and it's associated entities, when transforming EER to ARM
- `addEntityConnections()`: sets the dependencies of each entity object when transforming EER to ARM.

Entity

Stores all information of an entity object extracted from diagram.

Methods:

- `setUpEntity()`: sets ID and name of entity based on the type of input model
- `addAttributes(String info)`: extracts information from 'info' parameter and sets the attributes of the entity object

Relationship

Stores information of relationship component extracted from diagram.

Methods:

- `setCardinalities(String rRelation, String eRelation, String eID)`: sets the cardinalities for each entity that the relationship object is connected to, using the parameters: 1) `rRelation` – cardinality at relationship component on input diagram, 2) `eRelation` – cardinality at entity component on input diagram

Attributes

Stores all the attribute information related to a specific Entity Object

Methods:

- `orderAttributes()`: sorts the attribute information according to primary keys/ identifiers and uncategorized attributes
- `addForeignKey(ArrayList<String> arr)`: stores the attribute information of the strong entity associated with the specific weak entity

Component

Stores information of attributes that have not been assigned/connected to an Entity Object.

There are no significant methods used in the transformation process. Main method is constructor and setter methods used to set information and getter methods to obtain information, such as `parentID` (entityID that it belongs to)

Connection

Stores information of each arrow/line drawn in the input model.

There are no significant methods used in the transformation process. Main method is constructor and setter methods used to set information and getter methods to obtain information, such as `sourceID` and `targetID`, i.e. the source of arrow/line and target it is connecting to.

EER_Transformation (JFrame)

Displays transformation model (EER to ARM) in the `ARM_Model JPanel` component, as well as buttons that provide additional functionality to obtain transformation information and saving the model.

Methods:

- `transformationLogActionPerformed ActionListener`: assigned to `transformationLogButton` as an action listener and triggers the display of the transformation log form
- `inputNewModelActionPerformed ActionListener`: assigned to `newModelButton` as an action listener and triggers the display of the `VikerUI` to input a new XML file
- `saveModelActionPerformed ActionListener`: saves all the graphical components painted within the `ARM_Model JPanel` to a jpeg file.

ARM_Model (JPanel)

Draws the graphical components/lines to form the transformation model (EER to ARM) and displays the ARM model. This is all done by calling the `paintComponent()` method.

ARM_Transformation (JFrame)

Displays transformation model (ARM to EER) in the `EER_Model JPanel` component, as well as buttons that provide additional functionality to obtain transformation information and saving the model.

Methods:

- `transformationLogActionPerformed ActionListener`: assigned to `transformationLogButton` as an action listener and triggers the display of the transformation log form
- `inputNewModelActionPerformed ActionListener`: assigned to `newModelButton` as an action listener and triggers the display of the `VikerUI` to input a new XML file

- `saveModelActionPerformed ActionListener`: saves all the graphical components painted within the `EER_Model JPanel` to a jpeg file.

EER_Model (JPanel)

Draws the graphical components/lines to form the transformation model (ARM to EER) and displays the EER model. This is all done by calling the `paintComponent()` method. The `JPanel` makes use of the `DrawnEntity`, `DrawnRelationship` and `DrawnAttribute` classes to keep track of the graphical EER model components that have been drawn.

Transformation_Log (JFrame)

Displays the entities/entity tables that have been extracted from the input model. Allows the user an action listener to select a specific entity and display the details of the selected entity in the `Entity_Details JFrame`.

Methods:

- `setUpEntitiesList()`: traverse through the entities list in the Transformation Object and displays the entity names
- `- entityTableMouseClicked()`: action listener to select and display the details of the entity name selected in the `Entity_Details JFrame`.

Entity_Details (JFrame)

Displays the details of the entity that has been selected by the user in the `Transformation_Log JFrame`, by traversing the Transformation Object.

Methods:

- `displayEERDetails()`: displays the details of the EER transformation
- `displayARMDetails()`: displays the details of the ARM transformation

4.4. Sequence Diagrams

Observe sequence diagrams below (figure 1 & figure 2) of:

- EER to ARM transformation
- ARM to EER transformation

Sequence Diagrams below only show the order of relevant methods/functions that are called and/or used in each transformation. The display of the transformation log and entity details are not included in the sequence diagram of the transformation.

Figure 1: Sequence Diagram of EER to ARM transformation

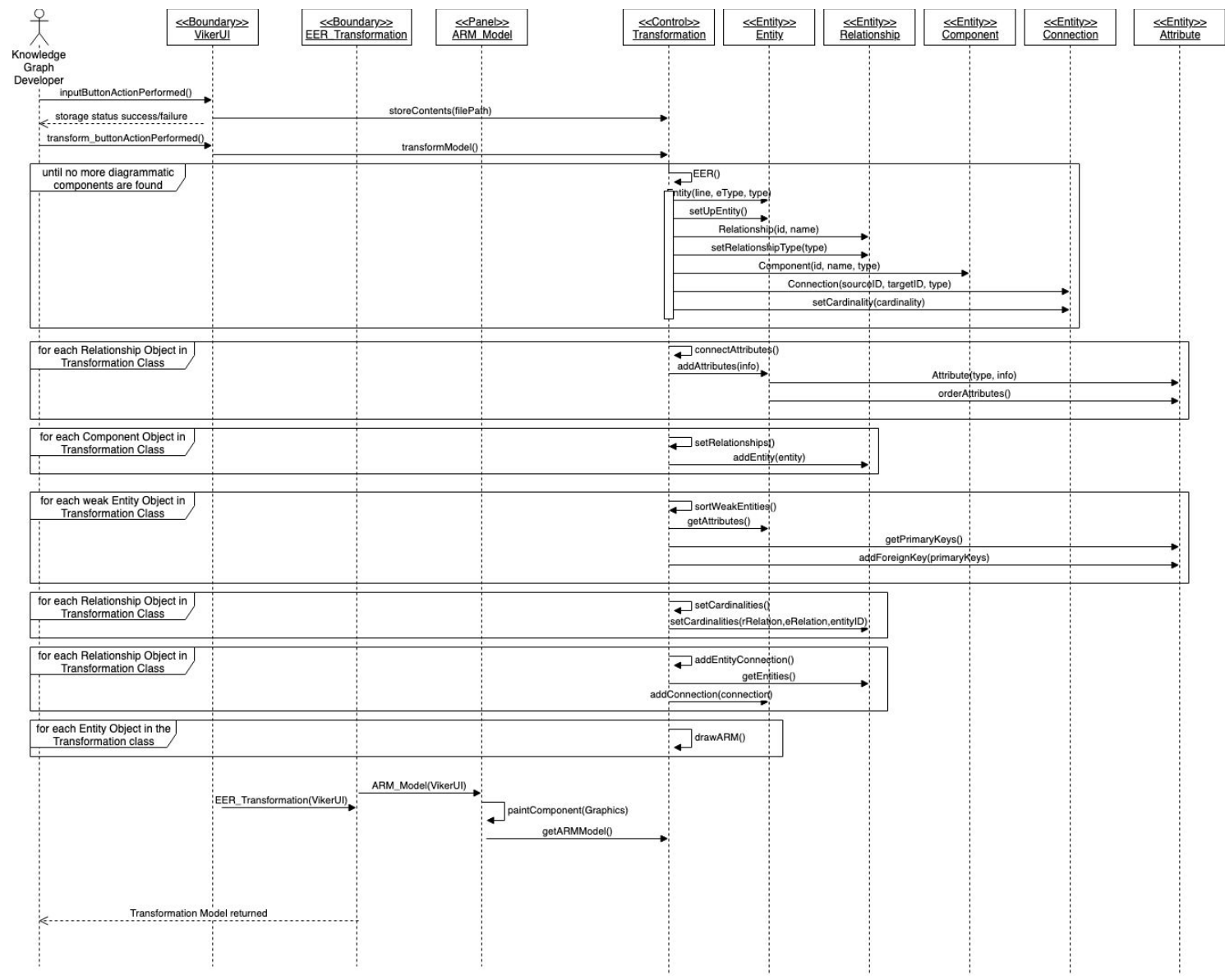
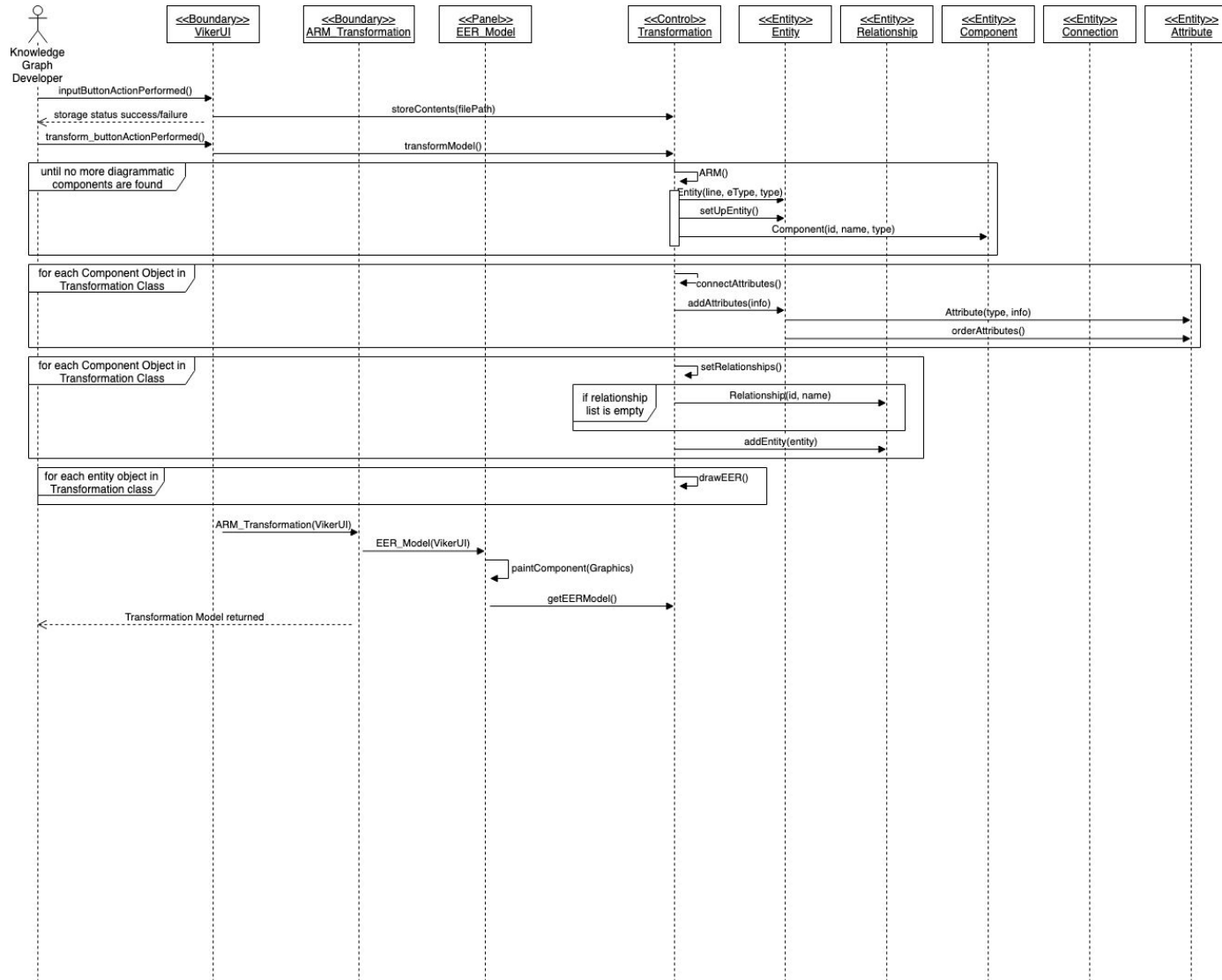


Figure 2: Sequence Diagram of ARM to EER transformation



5. Important Relationships between Classes and Interfaces

The transformation process relies heavily on the *Transformation.java*, since all the file contents are stored within this class. The interfaces all rely on the *Transformation.java* to display the EER/ARM transformation model and the transformation log, along with the entity details within the log.

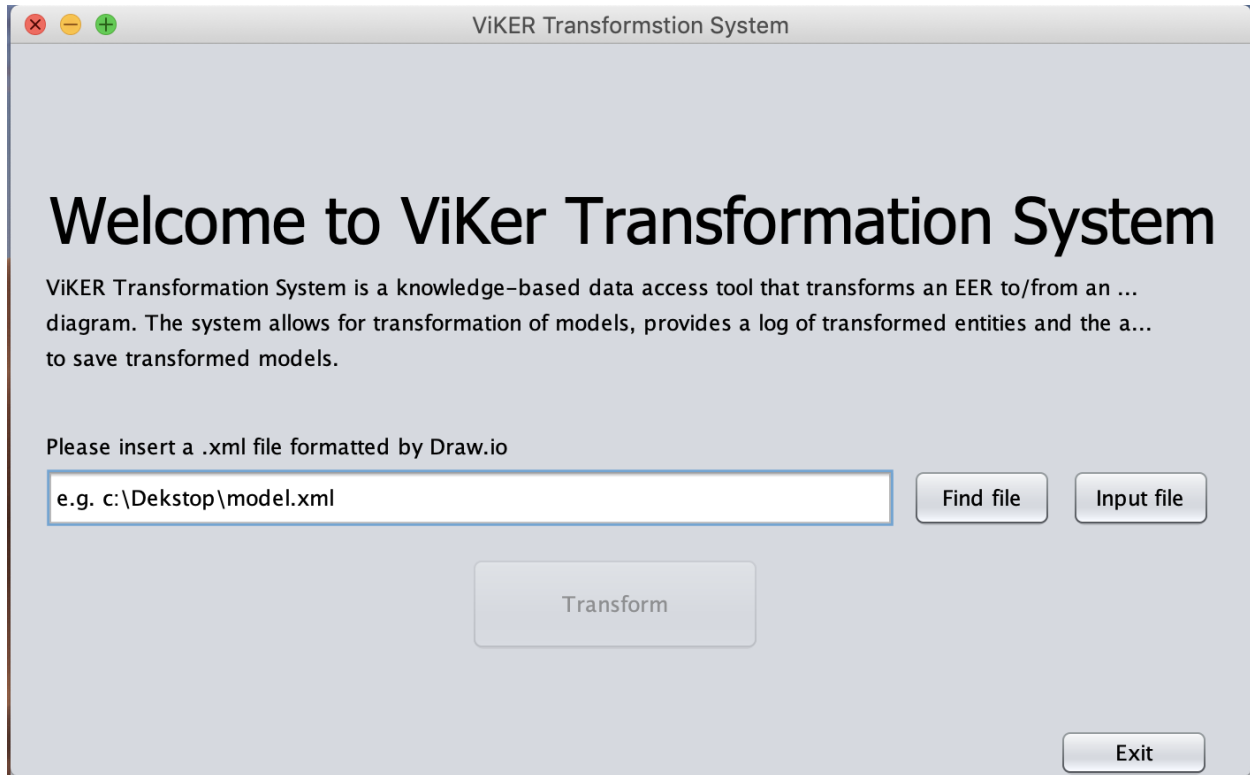


Figure 3: Screenshot of Welcome Window

The *VikerUI.java* (View) jframe is the welcome window which gives a brief description of the functionalities available to the user and the required input (see figure 3):

- When the 'Find file' button is pressed the jframe allows the user to browse the device for the desired XML file that the user would like to input into the transformation tool
- The "Input file" button is then pressed to store the contents of the XML file within a *Transformation.java* (Controller) Object
- The "Transform" button is then pressed to display the relevant *EER_Transformation.java* (View) jframe (figure 4) or *ARM_Transformation.java* (View) jframe (figure 5)

The *Transformation.java* (Controller) object utilises the *Entity.java*, *Relationship.java*, *Attributes.java*, *Component.java* and *Connections.java* classes in order to successfully process the transformation to be displayed.

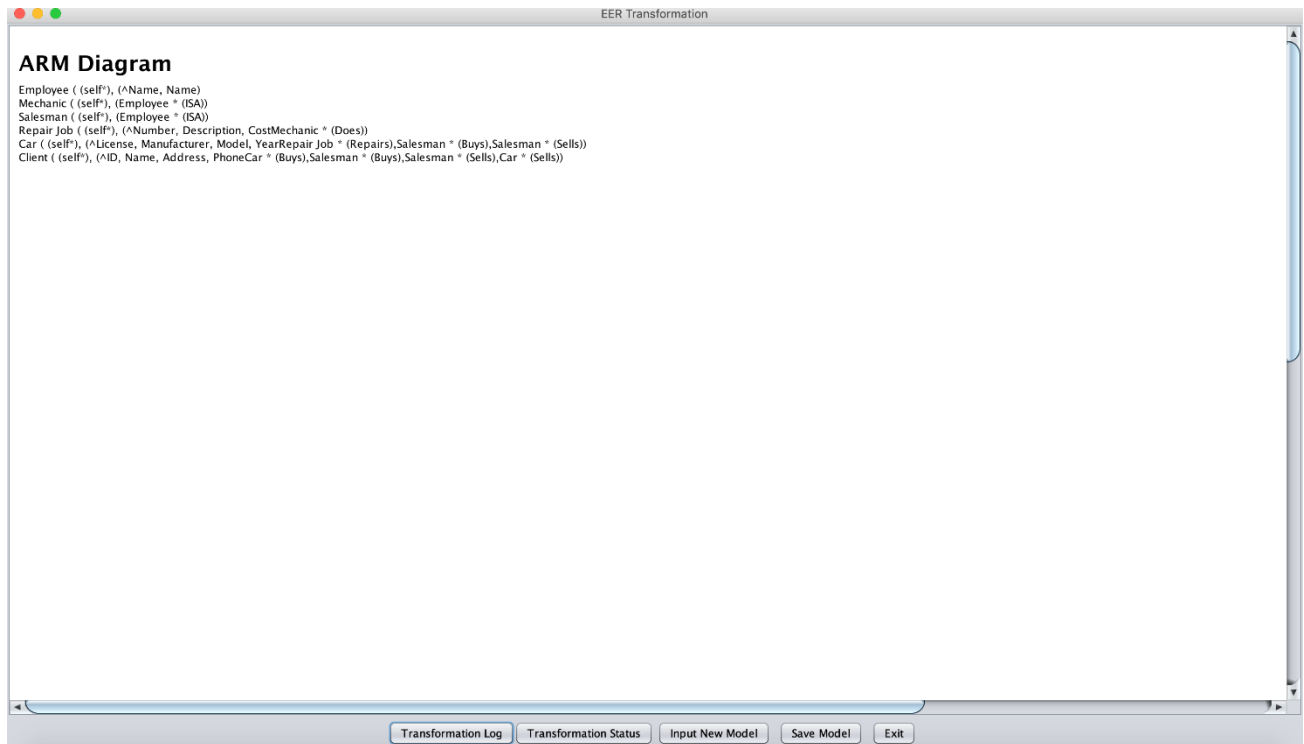


Figure 4: Screenshot of the EER_Transformation Frame

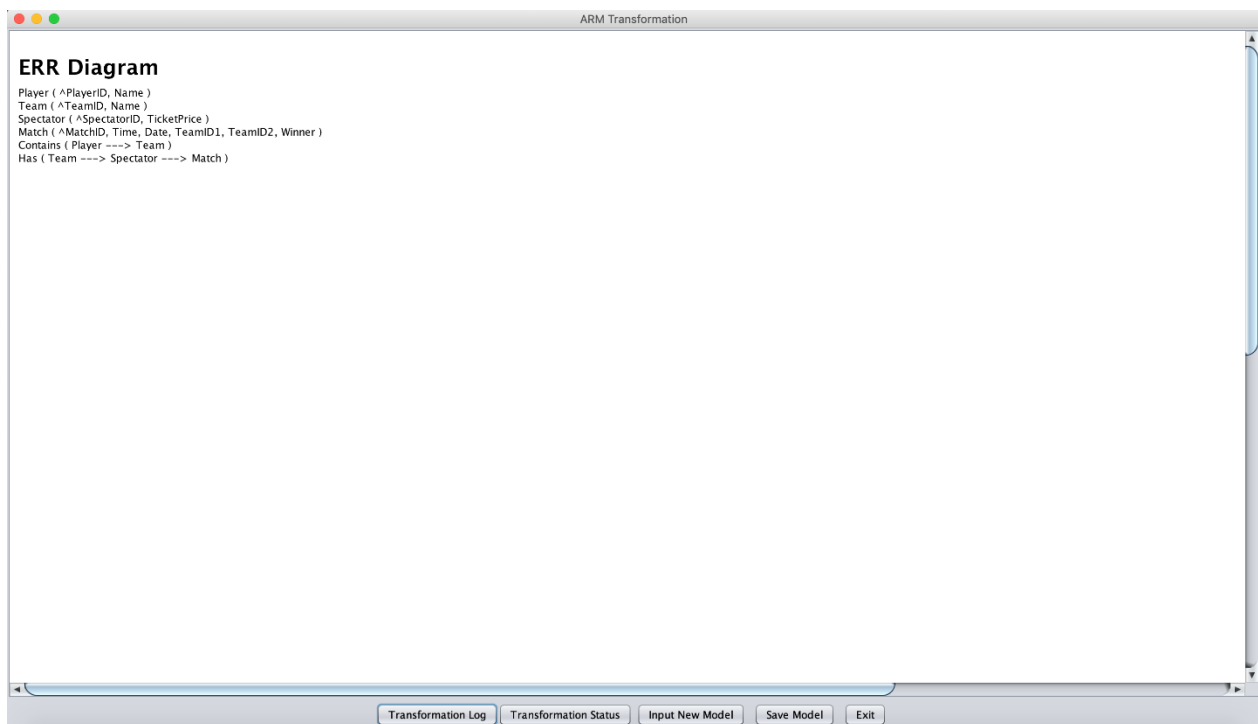


Figure 5: Screenshot of ARM_Transformation Frame

Figures 4 and 5, which are Views accesses and uses the *Transformation.java (Controller)* object to obtain the extracted and transformed information to display the transformation model in the *ARM_Model.java (View)* jpanel or *EER_Model.java (View)* jpanel, within the transformation jframes (figures 4 or 5). Both the above frames, i.e. figure 4 and 5, contain the following functionalities:

- “Transformation Log” button triggers the *Transformation_Log.java (View)* jframe (figure 6) to be initialized and displayed to display the entity objects/tables that have been extracted from the input file and transformed
- “Input New Model” button triggers the *VikerUI.java (View)* and displays the jframe to allow the user to input a new XML file
- “Save Model” button saves the graphical components that are contained within *ARM_Model.java (View)* jpanel or *EER_Model.java (View)* jpanel, within the transformation jframes (figures 4 or 5) into a *.jpeg* file
- “Transformation Status” button triggers the display of a dialogue box (Figure 8) indicating whether or not the transformation was a failure or success.

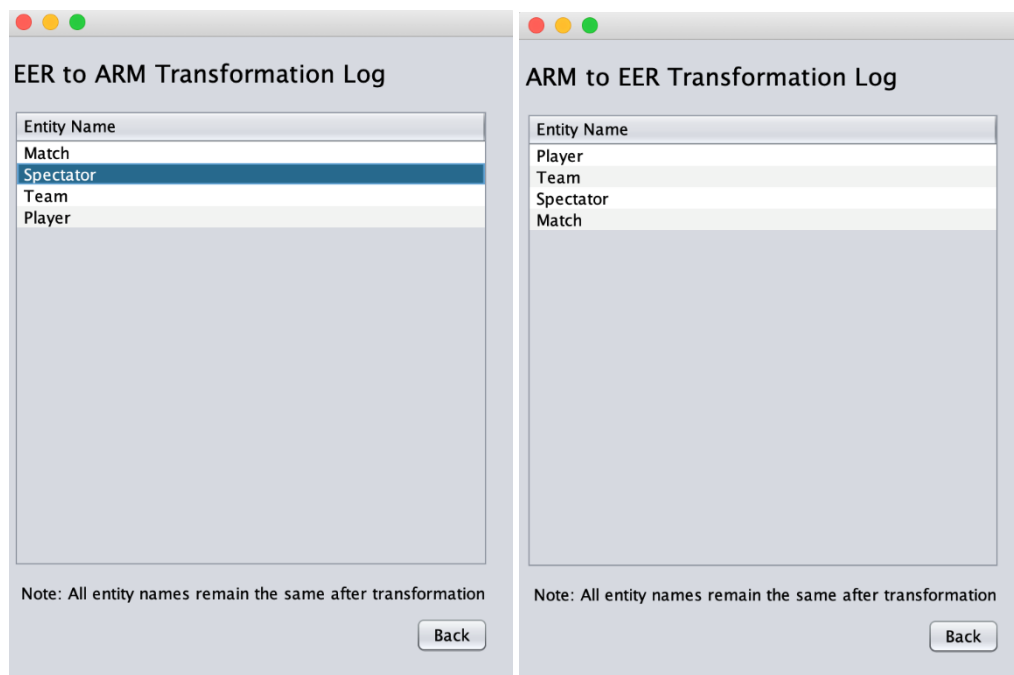


Figure 6: Screenshots of *Transformation_Log* Frame for both types of transformations

The *Transformation_Log.java (View)* jframe accesses the *Transformation.java (Controller)* class to obtain the names of the entity components extracted from the input XML file:

- When a specific entity name is selected by the user (Figure 6 – EER to ARM Transformation Log: “Spectator” highlighted) the *Entity_Details.java (View)* jframe (figure 7) is initialized and displayed.

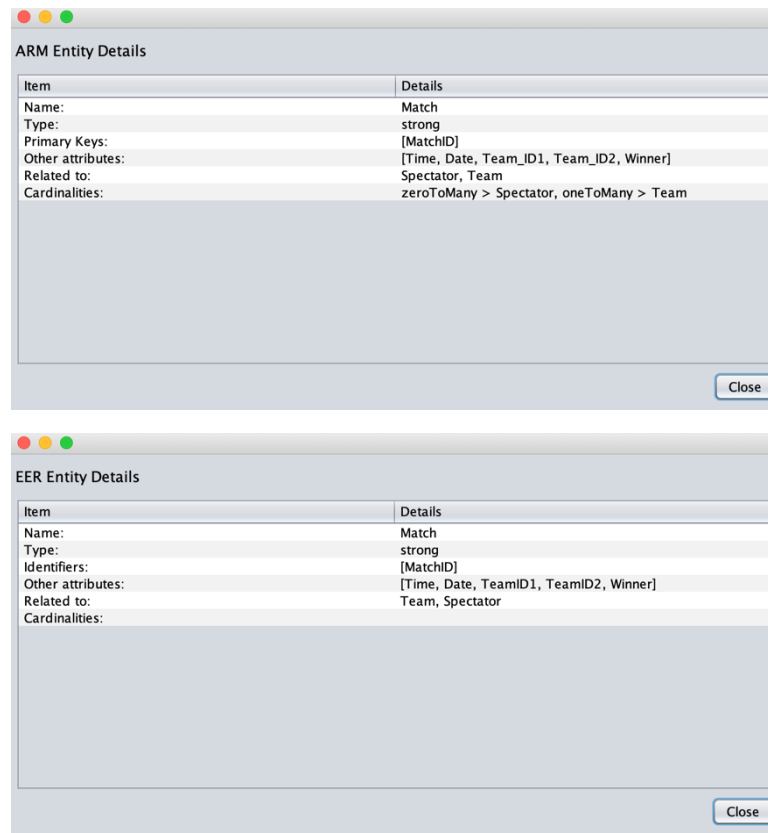


Figure 7: Screenshots of Entity_Details Frame for both transformations

The *Entity_Details.java* (View) JFrame accesses the *Transformation.java* (Controller) class to obtain the information for the selected entity within the *Transformation_Log.java* (View) JFrame.

Special Programming & Techniques

The transformation of the components extracted from the XML input file does not require much computations; however, it does rely on the storage of information that is extracted. Subsequently, the use of an object-orientated programming language, i.e. Java, was advantageous in this regard. Object-orientated programming allows for the use of objects, which are data structures, to store information relating to a specific component extracted from the input XML file. This allows the information that is stored to be maintained and manipulated in a safe manner. The aforementioned point is possible due to the use of private instance variables and methods to manipulate and/or access information stored within a specific object. The reason for choosing Java as the object-orientated programming language is due to the fact that Java is platform-independent, which means that the program is able to run on any computer system (cross-platform).

The architecture of the transformation tool implements an MVC (Model-View-Controller) pattern, which allows the JFrames (Views) to display information but never to do any computations with regard to manipulation of the data stored from the extraction of the input XML file. This allows for the interfaces to easily be adapted or to display the extracted information in a different format/s.

6. Program validation and Verification

The successful functioning of the transformation tool fundamentally relies on the input provided by users. This led to most testing being done by testing the types of input that was provided to the system in the form of an xml file, automatically generated by an online drawing tool (<https://www.draw.io/>). The xml files that are automatically generated from this online drawing tool depicted one of two types of diagrams, which includes 1) Enhanced Entity-Relationship (EER) Model, or 2) Abstract Relational Model (ARM). In addition, the tool provides an interactive user interface which led the testing of the tool to include the interactions of the user's input with the software. The following table (Table 1) summarizes the testing plan that was followed during the testing of the transformation tool.

Table 1: Summary of Transformation Tool Testing Plan

| Process | Technique |
|---|--|
| 1. Class Testing: Test methods and state behaviour of classes | Random, Partition and White-Box Tests |
| 2. Integration Testing: Test the interaction of user interfaces with sets of classes used for transformations between types of input diagrams and the databases used to store and assist in the illustration of transformations | Random and Behavioural Testing |
| 3. Validation Testing: Test whether tool meets requirements of client, and that of the defined scope | Use-case based black box and Acceptance tests |
| 4. System Testing: Testing if the tool works for multiple types of EER and ARM xml input files for transformations | Recovery, security, stress and performance tests |

Subsequently, for the complete validation and verification of the correct implementation of the tool's functions, additional test cases have been done. These test cases are described below (Table 2), with each test case description including the dataset used, reason for each test case and the types of test cases that the dataset satisfies, i.e. Normal functioning, Extreme boundary testing, Invalid data. The test cases were split into testing:

1. Basic functionality
2. EER to ARM transformations
3. ARM to EER transformations

Table 2: Summary of Tests Completed for Transformation Tool

| Test Case | Dataset Description | Reason for Choice of Testing | Test Cases |
|-----------|---------------------------------|---|---|
| 1 | Start/Run program | Ensure that the program start-up function works and is presented to the user | <u>Normal Functioning:</u> Passed – program starts |
| | | | <u>Extreme Boundary Cases:</u> N/A |
| | | | <u>Invalid Data:</u> N/A |
| 2 | Browse for input file on device | Ensures that the program allows the user the ability to browse and select the various file/s that is to be input into the transformation tool | <u>Normal Functioning:</u> Passed – program allows user to browse device and select input file |
| | | | <u>Extreme Boundary Cases:</u> Passed – allows the user to browse the device for a file only Passed – allows the user to browse the device for a file and select it for input |
| | | | <u>Invalid Data:</u> N/A |
| 3 | Xml file as input | <ul style="list-style-type: none"> Ensure that the program is able to take the correct file type as input for transformation Ensure that the program is able to detect that an incorrect file format has been input | <u>Normal Functioning:</u> Passed – program reads and stores contents of xml file |
| | | | <u>Extreme Boundary Cases:</u> Passed – input xml file and stored contents Passed – input no file, was not able to transform since the function is unavailable when no input provided |
| | | | <u>Invalid Data:</u> Passed – input text file format and error message displayed that incorrect file format was input |
| 4 | Exit function on all frames | Ensure that the user is able to exit the system at any point in the program | <u>Normal Functioning:</u> Passed – program exits at all points in the program |
| | | | <u>Extreme Boundary Cases:</u> Passed – able to exit the program immediately after start-up Passed – able to exit after doing transformation (no other functionalities done) |

| | | | |
|---|--|---|--|
| | | | <p>Passed – able to exit after a complete transformation model has been saved and transformation log looked at</p> <p><u>Invalid Data:</u> N/A</p> |
| 5 | Type of diagram that is input, i.e. EER or ARM | Ensure that the program is able to detect the type of diagram that is provided as input | <p><u>Normal Functioning:</u> Passed – program detects the type of diagram that is provided as input</p> <p><u>Extreme Boundary Cases:</u> Passed – program detects EER diagram that is provided as input Passed – program detects ARM diagram that is provided as input</p> <p><u>Invalid Data:</u> Passed – program detects that an incorrect xml format of a diagram is provided as input, i.e. the program informs users that the incorrect format of xml file was input</p> |
| 6 | EER to ARM transformation | Ensure that the transformation tool is able to transform an EER model to an ARM | <p><u>Normal Functioning:</u> Passed – program transforms EER and displays an ARM</p> <p><u>Extreme Boundary Cases:</u> Passed – program transforms EER with no components, i.e. empty EER, to an empty ARM Passed – program transforms EER with components to an ARM</p> <p><u>Invalid Data:</u> Passed – program detects if it is not an EER and does not transform into an ARM</p> |
| 7 | EER to ARM transformation - containing strong entity types | Ensure that the transformation tool detects and displays strong entity types | <p><u>Normal Functioning:</u> Passed – program detects and displays strong entity types from EER to ARM</p> <p><u>Extreme Boundary Cases:</u></p> |

| | | | |
|---|---|--|--|
| | | | Passed – programs detects and displays EER to ARM transformation that contains only strong entities Passed – program detects and displays EER to ARM transformation that contains strong and weak entities |
| | | | <u>Invalid Data:</u> N/A |
| 8 | EER to ARM transformation - containing weak entity types | Ensure that the transformation tool detects and displays weak entity types | <u>Normal Functioning:</u> Passed – program detects and displays weak entity types from EER to ARM |
| | | | <u>Extreme Boundary Cases:</u> Passed – program detects and displays EER to ARM transformation that contains no weak entity types Passed – program detects and displays EER to ARM transformation that contains only one strong entity type, i.e. all weak entity types except for one entity that is strong |
| | | | <u>Invalid Data:</u> N/A |
| 9 | EER to ARM transformation - containing attributes for each entity | Ensure that attributes are detected and displayed correctly in the ARM | <u>Normal Functioning:</u> Passed – program detects and displays attributes for each entity in the ARM |
| | | | <u>Extreme Boundary Cases:</u> Passed – program detects and displays EER to ARM transformation that contains no attributes for entities Passed – program detects and displays EER to ARM |

| | | | |
|----|--|--|--|
| | | | transformation that contains attributes for each entity |
| | | | <u>Invalid Data:</u> N/A |
| 10 | EER to ARM transformation – containing cardinalities between entity and relationship components | Ensure that cardinalities are detected, stored and displayed as dependencies/relations between entities in the ARM | <u>Normal Functioning:</u> Passed – program detects, stores and displays dependencies/relations between entities in the ARM <u>Extreme Boundary Cases:</u> Passed – program detects that there are no cardinalities connecting entity and relationship components in the diagram Passed – program detects all cardinalities connecting entity and relationship components <u>Invalid Data:</u> Passed – program detects that there are no cardinalities connected to an entity or relationship component |
| 11 | EER diagram provided as input, conforming to all formatting rules of the program | Ensure that all details obtained and stored from the EER diagram displays in the transformation log | <u>Normal Functioning:</u> Passed – program obtains and stores information from the diagram in the transformation log <u>Extreme Boundary Cases:</u> N/A <u>Invalid Data:</u> N/A |
| 12 | EER diagram provided as input, conforming to all formatting rules of the program, transformed into ARM | Ensure that the transformed EER to ARM diagram is displayed correctly and saved | <u>Normal Functioning:</u> Passed – program successfully transform the EER to an ARM and saves the model <u>Extreme Boundary Cases:</u> N/A <u>Invalid Data:</u> N/A |
| 13 | ARM to EER transformation | | <u>Normal Functioning:</u> |

| | | | |
|----|---|--|--|
| | | Ensure that the transformation tool is able to transform an ARM to an EER model | <p>Passed – program transforms ARM and displays an EER</p> <p><u>Extreme Boundary Cases:</u> Passed – program transforms ARM with no components, i.e. empty ARM, to an empty EER Passed – program transforms ARM with components to an EER</p> <p><u>Invalid Data:</u> Passed – program detects if it is not an ARM diagram and does not transform into an EER model</p> |
| 14 | ARM to EER transformation - containing entity components/tables | Ensure that the transformation tool detects and displays entities in the EER model | <p><u>Normal Functioning:</u> Passed – program detects and displays entity tables from an ARM to an EER</p> <p><u>Extreme Boundary Cases:</u> Passed – programs detects and displays EER to ARM transformation that contains no entity tables</p> <p><u>Invalid Data:</u> N/A</p> |
| 15 | ARM to ERM transformation - containing attributes | Ensure that attributes for each entity table is stored and displayed | <p><u>Normal Functioning:</u> Passed – program detects and displays attributes for each entity table from the ARM in the EER model</p> <p><u>Extreme Boundary Cases:</u> Passed – program detects that there are no attributes in the entity tables from the ARM and displays no attributes in the EER model</p> |

| | | | |
|----|--|---|--|
| | | | <u>Invalid Data:</u> N/A |
| 16 | ARM to EER transformation – containing dependencies between entity tables | Ensure that dependencies between entity tables are stored and displayed | <u>Normal Functioning:</u> Passed – program detects, stores and displays dependencies between entity tables from the ARM |
| | | | <u>Extreme Boundary Cases:</u> Passed – program detects that there are no dependencies between entity tables from the ARM |
| | | | <u>Invalid Data:</u> Passed – program detects when there are no dependencies connected to an entity table |
| 17 | ARM diagram provided as input, conforming to all formatting rules of the program | Ensure that all details obtained and stored from the ARM diagram displays in the transformation log | <u>Normal Functioning:</u> Passed – program obtains and stores information from the ARM diagram in the transformation log |
| | | | <u>Extreme Boundary Cases:</u> N/A |
| | | | <u>Invalid Data:</u> N/A |
| 18 | ARM diagram provided as input, conforming to all formatting rules of the program, transformed into EER model | Ensure that the transformed ARM to EER diagram is displayed correctly and saved | <u>Normal Functioning:</u> Passed – program successfully transform the ARM to an EER model and saves the model |
| | | | <u>Extreme Boundary Cases:</u> N/A |
| | | | <u>Invalid Data:</u> N/A |

***** Refer to Appendix B to obtain the description of each test case, with their input, behavior of the program and the expected output *****

Consequently, conducting the aforementioned test cases has significantly (not completely) eradicated any errors or faults that may occur while using the program for its purpose, i.e. the transformation of EER/ARM to ARM/EER. Precautionary alert & error dialogue boxes, as well as confirmation dialogue boxes have been put into place to ensure that users do not use the transformation tool incorrectly, with specific caution to the format of the input files that must be adhered to, which can be found in the User Manual (Appendix A) attached to this report. Multiple discussions have occurred to ensure that client requirements are met in the manner that satisfies the client, along with adhering to the formats and

transformation algorithms that were defined in the KnowID document. The main error/s that could possibly occur, which deviates from the formats and transformation steps defined in the KnowID document, would be the transformation from an EER/ARM to an ARM/EER model. These errors could include:

- Incorrect transformation format used in the diagram displayed for the transformation, in both the ARM and EER
- Incorrect information stored from the input diagram
- User inputs an xml file that conforms to all rules defined in the User Manual (Appendix A) and the program does not detect or store the contents of the file incorrectly and does not transform
- Algorithm extracted from the KnowID document and incorrectly implemented in the software

Subsequently, if users adhere to the rules defined in the User Manual (Appendix A) and use the program for its intended use the risk of an error occurring should be low

7. Conclusion

The aim when beginning the project was to create a smooth system that could fit into a larger architecture. It needed to be well structured, robust, and accurate in its strategy for transforming given input models. The functional requirements were to allow for transformation between EER and ARM bidirectionally according to a fixed set of rules, report on success or failure of a transformation, and be able to save and open diagrams.

Functionality was far more important as a requirement than user friendliness and ease of use. Thus, the bulk of the focus was placed on implementing and understanding the rules behind the transformations. These rules were implemented correctly, allowing for all constraints as stipulated. The transformation and implementation of rules was indeed a success.

Despite not being a functional requirement, it was decided that a graphical user interface be favoured over a textual user interface. This was to allow for the displaying of models and general ease of use of the tool. The UI remains consistent throughout the different stages of the program and is robust, allowing for the program to continue running while re-entering new diagrams. The save feature was implemented to save diagrams as .jpeg files, thus meeting another functional requirement.

In addition, the user interface has the required functionality of displaying information about failure and/or success of a transformation. It contains a log to keep track of all information the system has stored and allows for easy tracking of what information has been lost in a transformation. This was a fundamental addition as it helped with testing the program and verify its correctness. Since the tool was only required to take in an already established notation for an EER diagram, testing consisted of creating test examples with all the constraints listed in the requirements paper. Since the program handles all constraints appropriately, it is verified to be robust, accurate, and operational.

Considering that all functional requirements were met, and additional features even added, we believe the system to be a success.

CSC3003S Capstone Project:
Viker Transformation Tool
Appendix A: User Manual

This is the user manual for the Viker Transformation Tool. It strives to assist in the formation of the models that are to be transformed, which are created by users, and the general understanding and navigability of the tool as a whole.

Contents

| | |
|---|--|
| 1. System Overview | |
| 2. Format if input XML files..... | |
| 2.1. Draw.io Online Drawing Tool..... | |
| 2.2. EER Model Format..... | |
| 2.3. ARM Model Format..... | |
| 2.4. Exporting draw.io Model..... | |
| 3. Transformation Tool Usage and Navigation..... | |
| 4. Tool Restrictions..... | |

1. System Overview

The Viker Transformation Tool is a knowledge-based data access tool that is used to transform an Extended Entity-Relationship (EER) into an Abstract Relational Model (ARM), and vice versa; the tool will include the implementation of the translation bi-directionally. The user has to generate an XML file, which conforms to the format specified within this user manual, for each of the EER and ARM diagrams in order to input the file for transformation and display. Once the transformation has occurred, the user will be able to obtain a transformation log of all entity components/tables extracted from the XML file. The user will be able to view the transformation model diagram, save the model as a *.jpeg* file and input new models (as many times as desired) for transformation.

The purpose of the tool is to assist in efficient data management to ensure that decision-making is done effectively and to accommodate for the changing requirements of organizational needs; this tool is therefore a program that is, possibly, to be built onto in the future.

The development of the tool includes the knowledge-to-data architecture rules, stated within the KnowID document, to process transformations.

2. Format of input XML files

Before being able to use the tool for transformations the user needs to ensure that the input file is an XML file that is automatically generated by the online, opensource drawing tool, i.e. draw.io (www.draw.io), and that it conforms to the format specified for each type of diagram, i.e. EER and/or ARM diagrams.

2.1. Draw.io Online Drawing Tool

Draw.io is an online, opensource drawing tool that is available to users to generate the XML file that is needed for the corresponding EER/ARM diagram. It is simple to understand and navigate the platform, which makes it an ideal tool that users can utilize when generating their XML files to use the Viker Transformation Tool.

Firstly, users can following the link to the online tool (www.draw.io). This should open the browser to display Figure 1 (below).

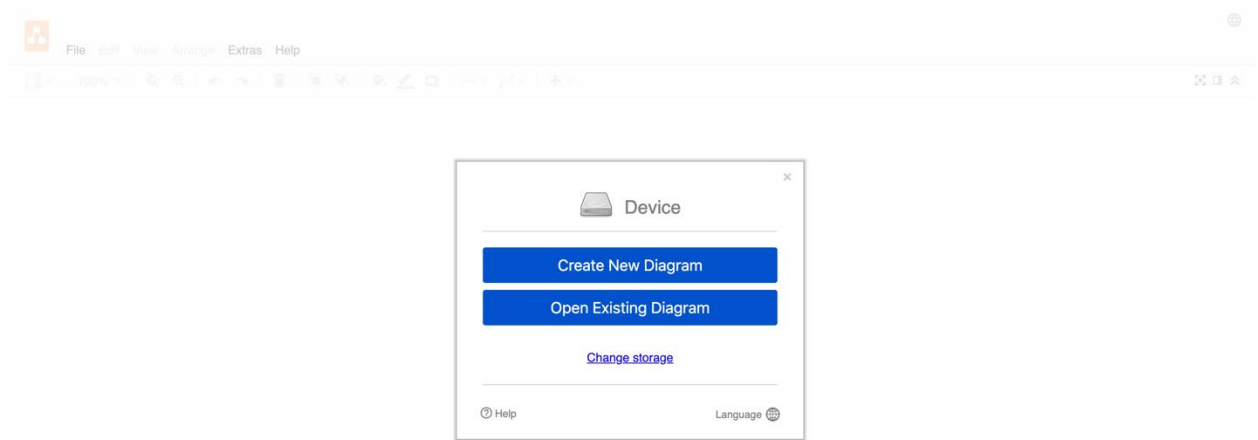


Figure 1: draw.io start-up page

Click the “Create New Diagram” button on the web-page which will lead to a window that prompts the user to rename the ‘Filename’, which is set to a default of ‘Untitled Diagram.drawio’ (Figure 2). The user should rename the filename to that which is desired. Ensure that the ‘Blank Diagram’ is selected from the ‘Basic (1)’ option in the list provided on the left of the window. Once the file is renamed (or set to default name), click the “Create” button at the bottom right of the window (Figure 3), which will lead you to the blank canvas on which the user will draw the EER/ARM diagram.

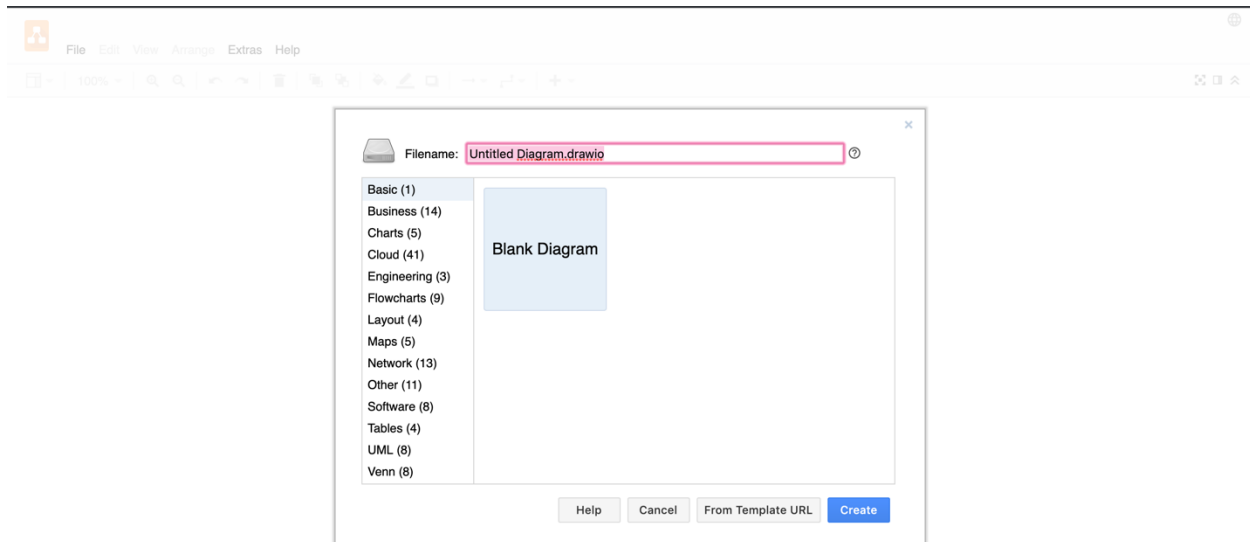


Figure 2: draw.io diagram creation window

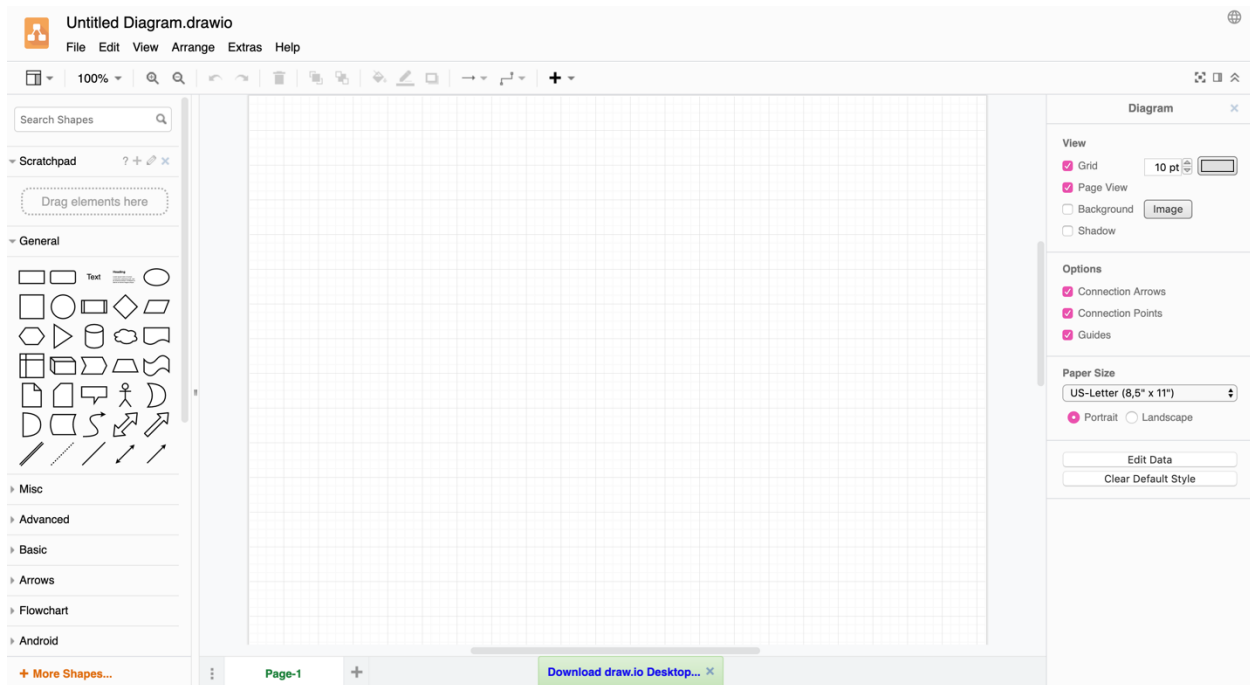


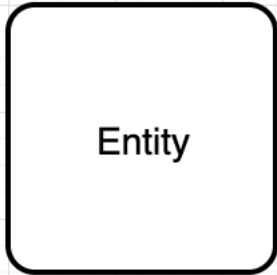
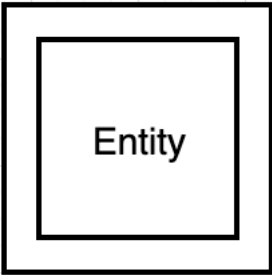
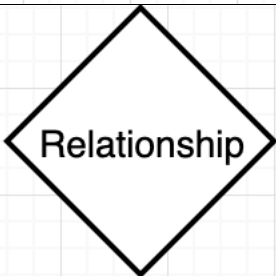
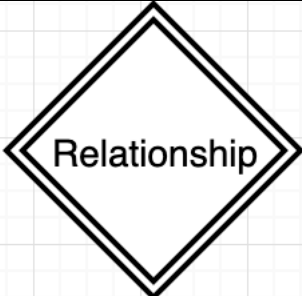
Figure 3: draw.io blank canvas





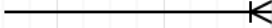





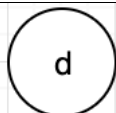
In Figure 3, the space at the center is referred to as the 'Canvas' (in this user manual only) the panel with list of shapes on the left-side of the screen is referred to as the 'Shapes Menu' (in this manual only), and the panel on the right-side of the screen is referred to as the 'Properties Panel' (in this manual only).

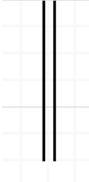









Each diagram that the user desire to draw has a specific format and drawing components, that must be adhered to, in order to represent specific notations within the EER/ARM diagrams. See the following sections to observe the rules of the formats for the EER and ARM diagrams.

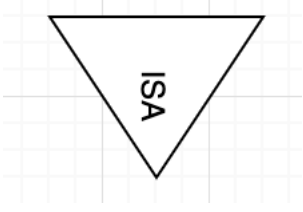

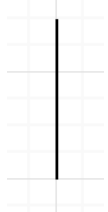



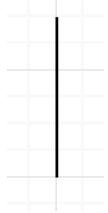



2.2. EER Model Format

An EER model that the Viker Transformation Tool is able to read and extract information from should conform to the specific rules defined within this section of the user manual. Below is a list, with corresponding illustrations, of components and their notations in an EER diagram generated by the draw.io drawing tool:

| Symbol | Object Representation | How to find in Shapes Menu |
|---|---|---|
|  | (Strong) Entity – represents an object in which information is stored | - 'Entity Relation' section: - Entity 3 (hover over components to get names) |
|  | Weak Entity – entity type that must be identified by a foreign key of an associated strong entity | - 'Entity Relation' section: - Entity 5 (hover over components to get names) |
|  | (Strong) Relationship – indicates how entities interaction with one another and their information | - 'Entity Relation' section: - 'Has' component with a single lined border (hover over components to get names) |
|  | Weak Relationship – indicates that the interaction between entities connected to this relationship component involves a weak entity | - 'Entity Relation' section: - 'Has' component with a double lined border (hover over components to get names) |

| | | |
|---|---|---|
|  | Attribute – a unique characteristic of an entity | -‘General’ section: - ‘Ellipse’ (hover over components to get names) |
|  | Attribute Identifier – a unique identifying characteristic (primary key) of an entity | -‘General’ section: - ‘Ellipse’ (hover over components to get names) - Highlight text inside ‘Ellipse’ component and underline the text |
|  | Connecting lines – solid connecting lines used to connect attributes to entities | -‘General’ section: - ‘Line’ (hover over components to get names) |
|  | Cardinality – one to one | -‘Entity Relation’ section: - ‘1 to 1’ (hover over components to get names) |
|  | Cardinality – one to many (mandatory) | -‘Entity Relation’ section: - ‘1 to Many’ (hover over components to get names) |
|  | Cardinality – many | -‘Entity Relation’ section: - ‘Many’ (hover over components to get names) |
|  | Cardinality – one or many (mandatory) | -‘Entity Relation’ section: - ‘1 Mandatory to Many Mandatory’ (hover over components to get names) |
|  | Cardinality – one and only one (mandatory) | -‘Entity Relation’ section: - ‘1 Mandatory’ (hover over components to get names) |
|  | Cardinality – zero or one (optional) | -‘Entity Relation’ section: - ‘0 to 1’ (hover over components to get names) |
|  | Cardinality – zero or many (optional) | -‘Entity Relation’ section: - ‘0 to Many (optional)’ (hover over components to get names) |
|  | Disjoint constraint component | -‘General’ section: - ‘Circle’ component (hover over components to get names) - Insert text ‘d’ inside the Circle component |

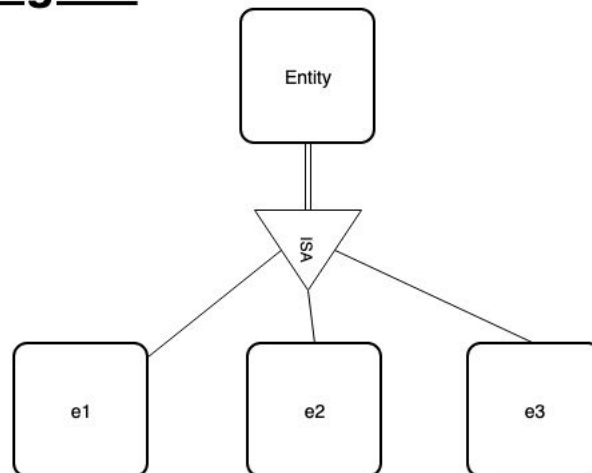
| | | |
|---|---|--|
|  | <p>Disjoint connection from superclass (disjointed entity) to Disjoint constraint component</p> | <p>Line drawn from superclass component to Disjoint constraint component:</p> <ul style="list-style-type: none"> -hover over superclass component until blue lines occur -drag mouse from blue line to Disjoint constraint component -select the connecting line and navigate to 'Properties Panel' - Underneath the 'Line' section set this icon  (by pressing the down arrow) to this icon  - Underneath the 'Line' section set this icon  (by pressing the down arrow) to 'None' - Underneath the 'Line' section set this icon  (by pressing the down arrow) to this icon  |
|  | <p>Disjoint connection from Disjoint constraint component to subclasses</p> | <p>Line drawn from Disjoint constraint component to subclasses:</p> <ul style="list-style-type: none"> -hover over disjoint component until blue lines occur -drag mouse from blue line to subclass/es -select the connecting line and navigate to 'Properties Panel' - Underneath the 'Line' section set this icon  (by pressing the down arrow) to this icon  - Underneath the 'Line' section set this icon  (by pressing the down arrow) to 'None' |

| | | |
|---|---|---|
|  | <p>Inheritance component</p> | <p>- 'General' section:</p> <ul style="list-style-type: none"> - 'Triangle' component (hover over components to get names) - Click on Triangle component (once to select) to get this icon  to appear above it - Click the icon once in order to get one triangle point to face down and two points to the side - Insert 'ISA' text into the Triangle component by double clicking |
|  | <p>Inheritance connection from superclass entity to Inheritance component</p> | <p>Line drawn from superclass component to Inheritance:</p> <ul style="list-style-type: none"> -hover over superclass component until blue lines occur -drag mouse from blue line to Inheritance component -select the connecting line and navigate to 'Properties Panel' - Underneath the 'Line' section set this icon  (by pressing the down arrow) to this icon  - Underneath the 'Line' section set this icon  (by pressing the down arrow) to 'None' |
|  | <p>Inheritance connection from Inheritance component to subclass entities</p> | <p>Line drawn from Inheritance component to subclasses:</p> <ul style="list-style-type: none"> -hover over inheritance component until blue lines occur -drag mouse from blue line to subclass/es -select the connecting line and navigate to 'Properties Panel' - Underneath the 'Line' section set this icon  (by pressing the down arrow) to this icon  - Underneath the 'Line' section set this icon  (by |

| | | |
|---------------------------|-----------------|--|
| | | pressing the down arrow) to 'None' |
| <u>EER Diagram</u> | Diagram heading | - 'General' section: - 'Text' component (hover over components to get names) - Text should be set to "EER Diagram", underlined text, bolded and set to 34 pt |

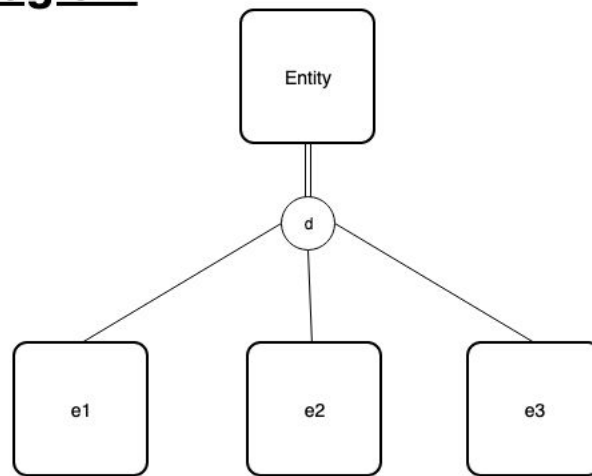
*****Important: Only the components shown in this section can be used to draw EER diagrams, i.e. no additional components can be used from draw.io to generate additional and/or uncommon symbols in a conventional EER diagram *****

EER Diagram



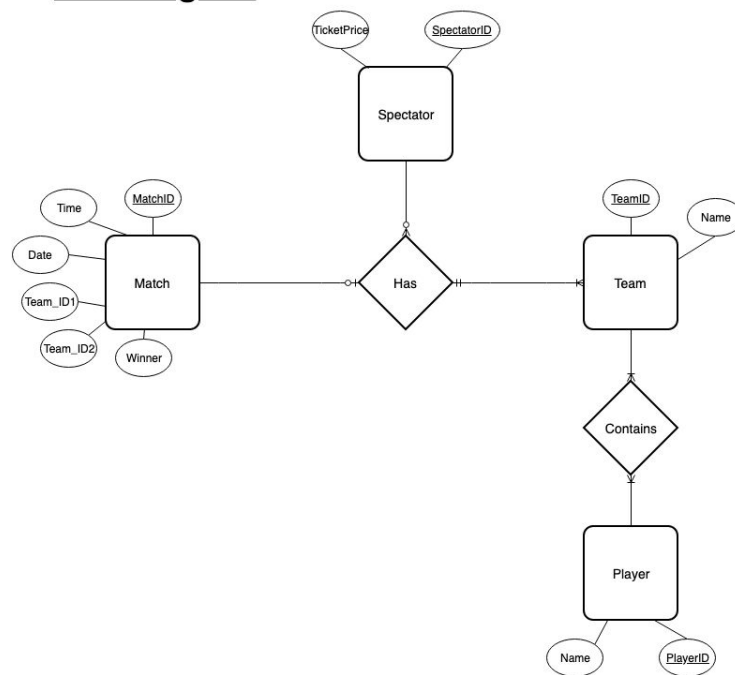
EER Model 1: Inheritance example

EER Diagram



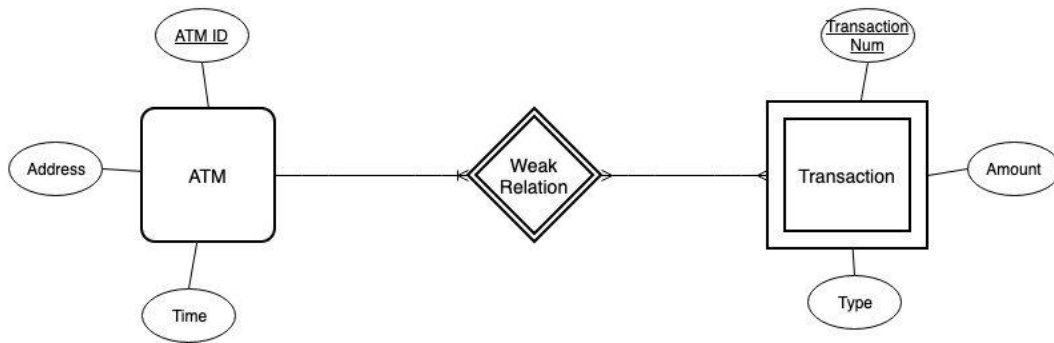
EER Model 2: Disjoint example

EER Diagram



EER Model 3: Basic EER example

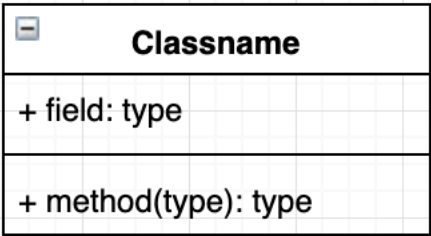
EER Diagram




EER Model 4: Weak Entity example

2.3. ARM Model Format

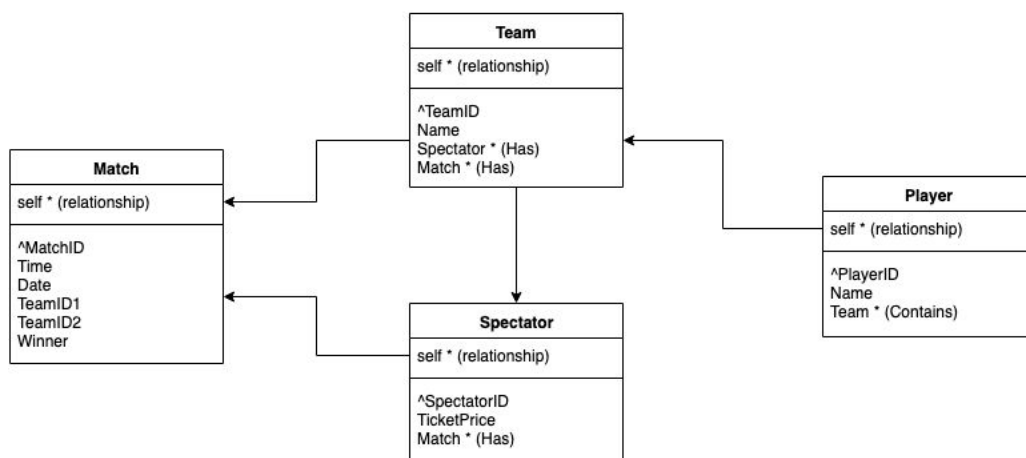
An ARM model that the Viker Transformation Tool is able to read and extract information from should conform to the specific rules defined within this section of the user manual. Below is a list, with corresponding illustrations, of components and their notations in an ARM diagram generated by the draw.io drawing tool:

| Symbol | Object Representation | How to find in Shapes Menu |
|--|------------------------|---|
|  <pre> classDiagram class Classname { + field: type + method(type): type } </pre> | Entity Table component | -'UML' section: - 'Class' component (hover over components to get names) - 'Classname' change to table name - '+field: type' section should change to 'self * (relationship)' - '+ method(type) : type' section should be where all attributes are added and dependences with their |

| | | |
|---|-----------------------------|--|
| | | dependency table names and relationship name, e.g. Attributes: 1) '^'PrimaryKeyName 2) AttributeName Dependencies: 'TableName' * ('relationshipName') |
|  | Dependency arrow/connection | Line drawn from Entity Table component to Dependency Entity Table component: -hover over Entity Table component until blue lines occur -drag mouse from blue line to Dependency Entity Table component |
| <u>ARM Diagram</u> | | -'General' section: -'Text' component (hover over components to get names) - Text should be set to "ARM Diagram", underlined text, bolded and set to 34 pt |

*****Important: Only the components shown in this section can be used to draw EER diagrams, i.e. no additional components can be used from draw.io to generate additional and/or uncommon symbols in a conventional EER diagram *****

ARM Diagram

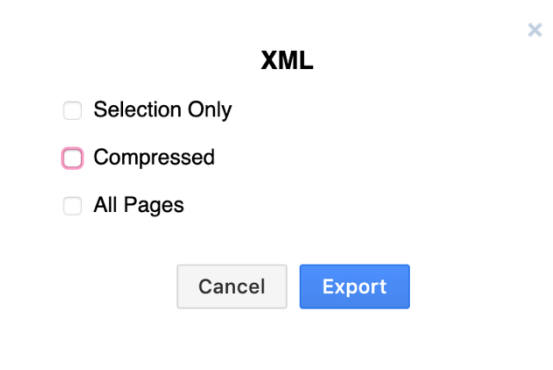


ARM Model 1: Basic ARM example

2.4. Exporting draw.io model

Once the model has been drawn implementing the formats defined in the sections above it can be exported as an XML file and saved to the device to be transformed by the Viker Transformation Tool.

To do so, click 'File', top left corner of the web-page, select 'Export as' and then select 'XML...'. A pop-up will display to confirm the exportation of the XML file.



Ensure that the pop-up displays the following, with all options unchecked. Then select 'Export' and 'Download'.

3. Transformation Tool Usage and Navigation

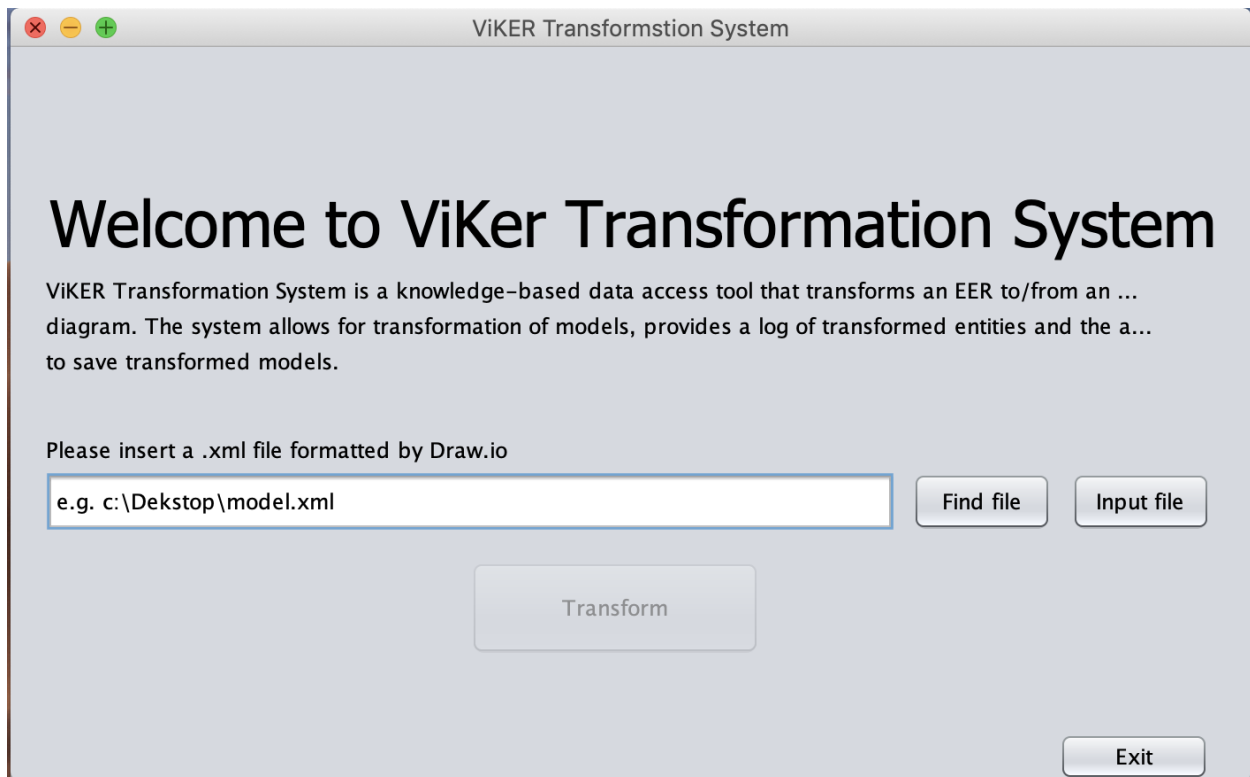


Figure 4: Welcome window

The Welcome window above allows users to browse their device and select an XML file for input. In order to do so, the user has to click the 'Find file' button and a browser window, listing files and folders of the current device, will be shown (Figure 5). The user should then click the 'Input file' button to allow the tool to read and store the contents of the xml file, which enables the 'Transform' button. The 'Transform' button should then be pressed to trigger the display of the EER/ARM transformation model (Figure 6).

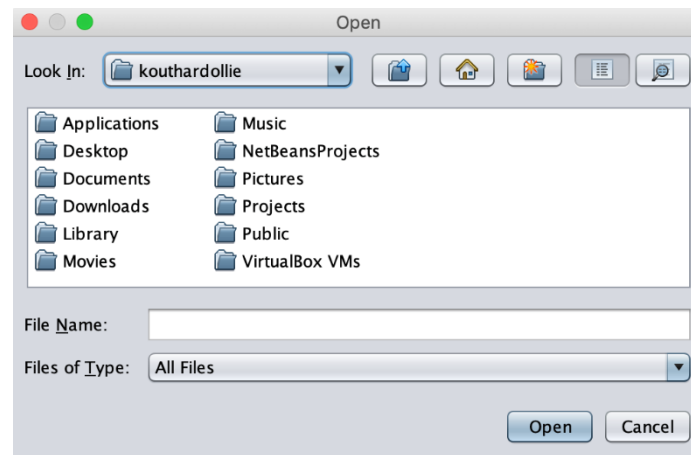


Figure 5: Browser for device

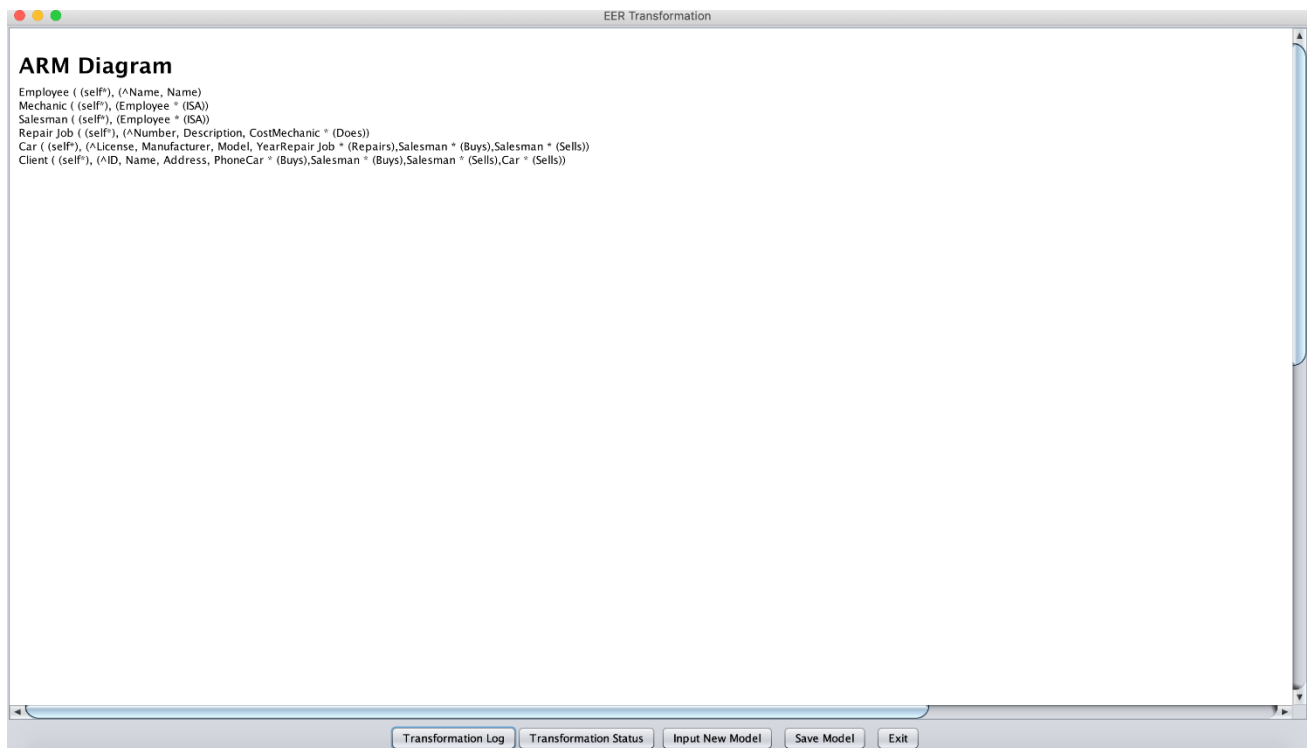


Figure 6: Transformation Model displayed

Once the transformation model (Figure 6) is displayed, the user is presented with a number of additional functionalities. The 'Transformation Log' button triggers the display of a list of entities that has been extracted from the XML file and transformed (Figure 7).

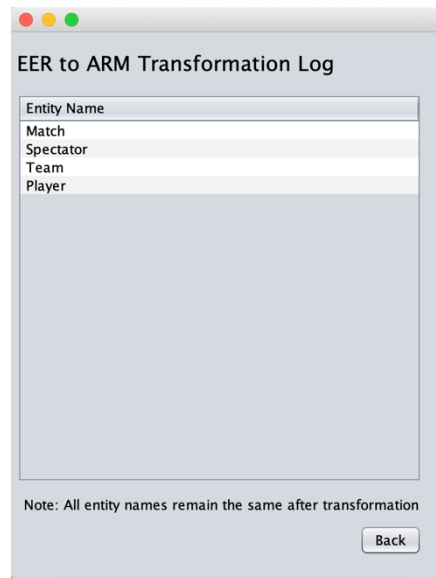


Figure 7: Transformation Log list display

The user is able to select (mouse click) on the name of the entity within the list to trigger the display of the entity details. This is shown in Figure 8, below.

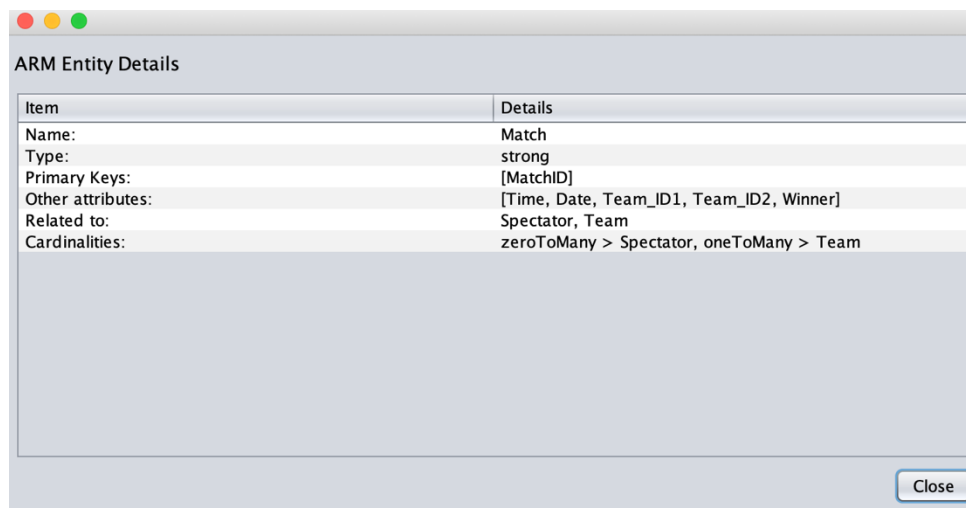


Figure 8: Entity Details displayed

Returning to the functionalities within Figure 6, the user is able to save the model displayed by pressing on the 'Save Model' button. A dialogue box displaying 'File saved' will pop-up. Another functionality includes obtaining the status of the transformation, triggered by the 'Transformation Status' to determine whether the transformation was a success or failure, which is displayed by a dialogue box.

4. Tool Restrictions

Some components may not be accounted for and errors may occur within the transformation if the formats and restrictions are not adhered to. The following restrictions apply:

- EER model:
 - Model must have diagram heading
 - Entities must have identifier attributes, i.e. attributes that are underlined
 - Cannot have multivalued attributes
 - Cannot have entities with the same names
 - Implements the Information Engineering Style notation for cardinalities
 - Model must be drawn with draw.io online drawing tool
- ARM model:
 - Model must have diagram heading
 - Entity tables must have primary keys specified
 - Model must be drawn with draw.io online drawing tool
- Cannot specify destination or filename of saved model

Appendix B: Test Plan

| Test case | Description | Input | Behaviour | Expected Output |
|-----------|--|--|--|--|
| 1 | Start/Run program to test if the transformation tool starts | Run java project | Project should start-up and run without any error messages | <i>VikerUI.java</i> jframe should display with welcome message |
| 2 | Test browsing and selection of file from device that tool is running on | User clicks 'Find file' button and browses and selects file for input | Program gets file selected by user | Program displays the filepath in the textbox |
| 3 | Test that program takes XML file as input | XML file selected for input and user clicks 'Input File' | Program should read content of file and determine the type of model that has been input | No error messages |
| 4 | Ensure that user is able to exit program at any point while using the tool | Usage of Transformation Tool at any point | Program should exit and terminate all functions and/or interfaces that are running | Program termination, i.e. program closed |
| 5 | Program should be able to determine the type of model that is input, i.e. EER or ARM | User selects EER/ARM XML file for input | Program should traverse through the contents of the XML file and determine the type of model that was input by the user | Display the correctly labelled transformation model, i.e. <i>EER_Transformation.java</i> or <i>ARM_Transformation.java</i> |
| 6 | Test the EER to ARM transformation | User selects an EER model as input (xml file) | Program should traverse through the contents stored and perform transformations on this data to display the ARM transformation model | Display the <i>EER_Transformation.java</i> |
| 7 | Ensure that EER input models containing strong entities are able to transform | User selects an EER model, containing strong entity types, as input (xml file) | Program should traverse through the contents stored and perform transformations on this data to display the ARM transformation model | Display the <i>EER_Transformation.java</i> with the strong entities |

| | | | | |
|----|---|---|--|--|
| 8 | Ensure that EER input models containing weak entities are able to transform | User selects an EER model, containing weak entity types, as input (xml file) | Program should traverse through the contents stored and perform transformations on this data to display the ARM transformation model | Display the <i>EER_Transformation.java</i> with the weak entities and their dependencies |
| 9 | Ensure that EER input models containing attributes for each entity are able to transform | User selects an EER model, containing attributes for each entity, as input (xml file) | Program should traverse through the contents stored and perform transformations on this data to display the ARM transformation model | Display the <i>EER_Transformation.java</i> with the attributes of each entity being displayed correctly |
| 10 | Ensure that EER input models containing cardinalities are able to transform | User selects an EER model, containing cardinalities between entities and relationship components, as input (xml file) | Program should traverse through the contents stored and perform transformations on this data to display the ARM transformation model | Display the <i>EER_Transformation.java</i> with the relationships connected to each entity is correctly displayed for dependencies |
| 11 | Ensuring that an EER diagram provided as input, conforming to all formatting rules of the program | User selects an EER model, conforming to all rules defined in user manual | Program should traverse through the contents stored and perform transformations | Display the <i>Transformation_Log.java</i> with all the diagrammatic components that were correctly identified and transformed |
| 12 | Ensuring that an EER diagram provided as input, conforming to all formatting rules of the program, transformed into ARM | User selects an EER model, conforming to all rules defined in user manual | Program should traverse through the contents stored and perform transformations on this data to display the ARM transformation model | Display the <i>EER_Transformation.java</i> With all diagrammatic components correctly displayed |
| 13 | Ensure that the transformation tool is able to transform an ARM to an EER model | User selects ARM model as input file | Program extracts xml information and stores and transforms data | Displays <i>ARM_Transformation.java</i> |
| 14 | Test that the ARM model containing | User selects ARM model as input file | Program extracts information, transforms it and | Display <i>ARM_Transformation.java</i> |

| | | | | |
|----|--|---|--|---|
| | entity tables will transform to EER | | uses it to display the transformation model | with extracted entity tables being correctly displayed |
| 15 | Test that the ARM model containing attributes within entity tables will transform to EER | User selects ARM model, containing attributes as input file | Program extracts information, transforms it and uses it to display the transformation model | Display <i>ARM_Transformation.java</i> with extracted entity tables being correctly displayed with their corresponding attributes |
| 16 | Test that the ARM model containing dependencies between entity tables will transform to EER | User selects ARM model, containing dependencies as input file | Program extracts information, transforms it and uses it to display the transformation model | Display <i>ARM_Transformation.java</i> with extracted entity tables being correctly displayed with their corresponding attributes, and their dependencies |
| 17 | Ensure that and ARM diagram provided as input, conforming to all formatting rules of the program | User selects an ARM model, conforming to all rules defined in user manual | Program should traverse through the contents stored and perform transformations on this data to display the EER transformation model | Display the <i>Transformation_Log.java</i> with all the diagrammatic components that were correctly identified and transformed |
| 18 | Ensure that ARM diagram provided as input, conforming to all formatting rules of the program, transformed into EER model | User selects an ARM model, conforming to all rules defined in user manual | Program should traverse through the contents stored and perform transformations on this data to display the EER transformation model | Display the <i>ARM_Transformation.java</i> with all diagrammatic components correctly displayed |

References

1. Journal Article: Pablo Ruben Fillottrani, P.R.F. and C. Maria Keet, C.M.K. (Not published yet) KnowID: An architecture for efficient Knowledge-driven Information and Data access.