



**Universiteit
Leiden**
The Netherlands

Opleiding Informatica en Economie

Enabling data-driven wireframe prototyping using Model Driven Development

Bachelor's thesis

Bram van Aggelen

1840045

Supervisors:

Guus Ramackers (first supervisor)

Joost Visser (second supervisor)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

www.liacs.leidenuniv.nl

16/07/2022

Abstract

The process of creating applications is a long and costly one often involving different parties and stakeholders, which increase the chances of misinterpretation. The more parties involved the more interpretation takes place. For the eventual users, both the look and feel and the functionality of the resulting software frequently remain unclear during the development process. This research addresses this issue by proposing a dynamic data-driven wireframe approach to prototyping based on evolving design time metadata.

The roadmap defines an approach based on Model Driven Engineering using the Unified Modeling Language and combines this with graphical UI driven prototyping using 'live' user data. This approach is implemented in the Django based LIACS Prose to Prototype / ngUML Software Development environment.

To ensure the wanted functionality could be implemented, changes to the ngUML datamodel were made. These consist of adding 3 new classes to ensure all functionality can be added. These new classes are the Section, Page and Category, all of which are subclasses of Application. These are UI components which enable full prototyping wireframes with live data. Besides that, a graph traversal algorithm is used to construct and manipulate complex graph based data sets. Where the Classifiers are the nodes and the Relationships are the edges.

Besides that a front-end User Interface is created, which enables the users to interface with the backend solution in an easy way. This includes a WYSIWYG editor, and a custom Page builder which can rearrange the different Sections.

The resulting metadata driven dynamic wireframe approach has been validated using a survey of a small set of users who were asked to perform a guided scenario using the tool developed. These users were made up of 2 groups, these being a group that has used (parts of) the tool before and people who have not yet used it.

Contents

Abstract	2
Contents	3
1. Introduction	5
1.1 Problem description	5
1.2 Research questions	5
1.3 Methodology	5
1.4 Academic contribution	6
1.5 Structure	7
2. Background	8
2.1 OMG UML	8
2.2 The ngUML base	8
2.3 Evaluation of WYSIWYG editors	9
2.4 MDD platform evaluation	9
2.6 Wireframes and their place in the development of applications	10
2.7 Combination of above mentioned topics	10
3. System design and implementation	11
3.1 Design of metadata model	11
3.2 Front-end	14
3.3 Backend implementation	16
3.3.1 Making use of relationships	16
3.2.2 Using the algorithm	17
4. Worked Example	19
4.1 Homepage	19
4.2 Changing the data model in the ngUML editor	19
4.3 Adding and setting up Applications	20
4.4 Creating pages	21
5. Validation	25
5.1 Validation plan	25
5.2 Validation by people involved with the project	26
5.3 Validation by outsiders	26
6. Conclusion and future work	27
6.1 Conclusion	27
6.2 Future work	28
6.2.1 Extending relationships	28
6.2.2 Linking to activity models	28
6.2.3 Changing filters	28
6.2.4 Changing data types while running	29
6.2.5 Self error correcting	29
6.2.6 Better UX design	29
References	30

Appendix	31
A.1 Standard requirement text	31
A.2 The prompts for the questionnaire	31
A.3 Answers to questionnaire	32

1. Introduction

1.1 Problem description

The process of designing an application is well known, unfortunately it does take a long time. A user indicates that a specific application needs to be made, they might hire a firm to specify their needs and to help them communicate with the eventual developing party, and they have endless meetings to form the specifications. These specifications are then brought to the developing party, who interprets the specifications and starts making a data model and a wireframe. When these things have been made and presented to the client but found not adequate, it might take a very long time for the changes, how small they might be, to be made. By implementing data-driven wireframe prototyping, we hope to be able to change this.

1.2 Research questions

This thesis extends a previous thesis which generates class models from natural language requirements[1] by making the option to create dynamic wireframe based prototypes that animate functionality using related requirements.

The first question was related to the data model used in the previous work[2] by extending it so that the original application remained fully functional and the additions were as smooth as possible. This was necessary as new functionality was added and thus needed to be supported. Eventually, the number of needed alterations was designed and engineered to be functional in a succinct and non-intrusive (requiring no modifications of the underlying system) manner.

The second question was how to implement a front-end system to be able to create wireframes. To help answer this question an evaluation of different open source text editors has been made to ensure good functionality and a way to alter it to our needs.

Lastly and most importantly the question of how to correctly link data to be able to display, insert and edit it. Because the previous work relations were already initialised, this might seem trivial, but the implementation required an efficient graph traversal algorithm design and implementation. This implementation was required to be made available through an internal API for reuse by future extensions such as an NLP rule editor.

1.3 Methodology

To get the needed results, a large part of the thesis is research by design, but part of it is a literature study. The deliverables will thus be a design, a working wireframe environment integrated with the ngUML models and a fully worked example scenario to demonstrate the deliverable. In addition to this validation was performed on a small sample of people to gauge general sentiment.

The architecture of the current NGUML backend looks like this:

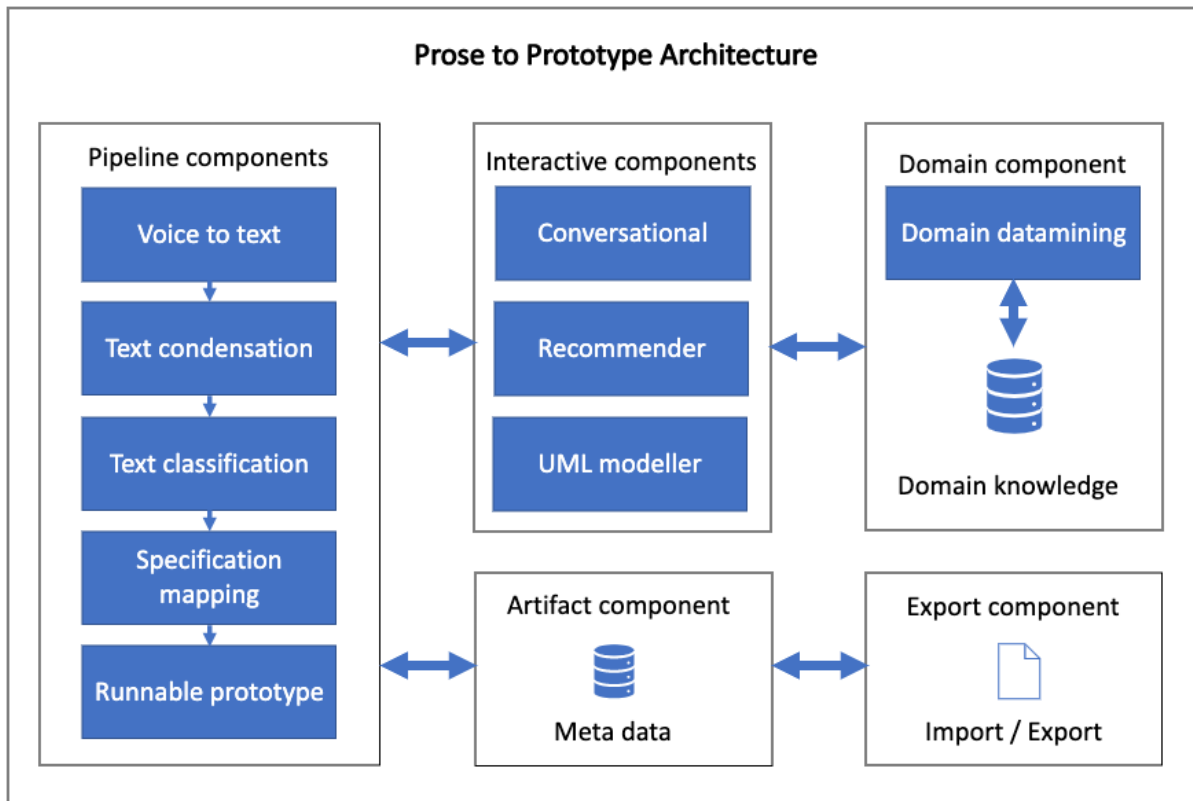


Diagram 1: The current use of all technologies in the ngUML application.

The thesis extends a lot on the base set by Ralph Driessen[2]. This is one of the reasons for using Application as the primary type for both Section and Page. As a result, the design will be an added part of the system. This will include, among others, a system to display any information with the right relationships, the ability to insert data with relationships, and a system to design a wireframe-like frontend.

The literature study part of the thesis relates to the design of the wireframe creation system, model-driven environments, and graph traversal algorithms.

1.4 Academic contribution

Researching the viability of implementing data driven technologies is an academic contribution because it combines the questions asked by multiple fields. These fields are mostly the application/creation of algorithms like a path finding algorithm and the real world application of what to do with changing data types. Lastly, this thesis also continues on the work started by Ralph Driessen in his thesis "UML class models as first-class citizen: Metadata at design-time and run-time"[2].

The first problem that arose was *how can a wireframe UI design be driven from a conceptual model in such a way that the solution is dynamic, in that modifications to metadata immediately result in changes to the UI, and the wireframes manipulate live data?*

To achieve a system where that is possible some other questions had to be answered. Some of these are:

- What is a minimal and effective set of metadata extensions to a UML class model to enable this?
- How can an execution engine support all this functionality in an efficient manner?
- What is the optimal graph traversal algorithm to manipulate complex data graphs in such a context?
- Which types of data migrations can be supported automatically in such a metadata driven environment?

1.5 Structure

Firstly, the background will be given and explained to help understand later mentioned topics. It will mainly focus on Model Driven Environments and UML, which is the implementation of models we use. In section 3, the system design is explained and substantiated. Section 4 explains the worked example. This furthermore can be considered a manual on how to use the product. A few people have tested the product. These people subsequently filled in a survey on the ease of use of the product through some prompts, this is discussed in section 5. Lastly, in section 6, a conclusion will be made and possible future work will be discussed.

2. Background

2.1 OMG UML

In the project, we decided to use the Unified Modeling Language [3] made by the Object Management Group, as this is a well known and defined language. By using this language, the readability of the models improves a lot. This ensures that both the professionals designing the system and the eventual people implementing it understand it and can communicate with each other. Besides that, it is also readable by the eventual users to see how the data links together.

There currently is much demand for applications that implement UML models [4]. Right now that demand is mostly in the research space and the business is lacking implementation. This divide could have many causes. It could be because it is not implemented in the right way for businesses right now or that businesses have no interest in the overhead of having to think in UML but instead choose to go for the implementation that suits them best. These are just some things to keep in mind when using UML like we do here.

2.2 The ngUML base

Extending Ralph Driessen's work[2] helped pick the subset of classes to extend with since he had already picked some for his system. He had chosen a basic section of the UML to be able to create technical designs. Extending on it was easy as he used Django[5], an open source and clear system.

Later other students added new functionality. A large focus of these students has gone into implementing Natural Language Processing (NLP). NLP is a specific application of artificial intelligence focused on understanding/analysing human language[6]. NLP is used in 2 main ways in the application right now.

The first is text-to-speech. It allows people to upload a recording of someone saying the requirement text, which will then translate into the words being said to pass along to the next phase. This allows for a quick, easy and good base to start on regarding the application.

The second part is to analyse the given text. This will go through sentence structures to find which words have relations to others and use that information to make a model that can be put into the system Ralph Driessen[2] created.

These two NLP systems combined lead to a good way of getting a base project. The second part will also be used to get a project up and running in chapter 4, the worked example. Only a few changes need to be made for it to work fully.

Currently, the research that has already been done has been marked under ngUML by the University of Leiden. This research will fall under the same category.

2.3 Evaluation of WYSIWYG editors

Firstly the term WYSIWYG stands for What You See Is What You Get. This is a term used for editors showing a version of what is being edited in the way it should eventually appear.

First, we tried to make our own editor, however eventually we settled on using TINY(MCE) [8] because it is open source and allows writing your own plugins.

Open source editors are there in all sorts of ways. TINY was eventually the one we chose because of its already wide use in other applications and because of how easy it is to implement it and write custom functions for it. The editor itself is not open source but is trusted by many big companies and is easy to set up and free. Lastly, the editor also is protected with free cloud security features. [9] It makes sure there are no XSS (Cross-Site Scripting) attacks. This might not be essential in a mainly development/testing application however, losing work regardless of what context is something that needs to be avoided.

Some other options that were not chosen were:

- CKEditor 4
- Editor.js
- Quill
- Bubble

Listed below are some of the reasons they were not chosen:

- Less secure
- Harder to write custom functions for
- Less functionality
- More of an inline editor than a full HTML editor
- Illogical layout of the editor

Since Tiny seemed to have less of these issues than other editors and the extra security features it was chosen over the other listed options.

2.4 MDD platform evaluation

To understand why this platform was built, first we looked at some other platforms and their good and bad points.

There currently exist other Model Driven Development platforms. These have paved the way for other systems like the one discussed here and, importantly provide an accessible service for people to make something quickly.

Some of the more widely known ones are

- Appian [10]
- Mendix [11]
- The Microsoft power platform [12]

All of the platforms will have their advantages as well as disadvantages. Some of the major disadvantages, though, are running costs as well as a locked-in system. It is usually impossible to easily convert from one of those applications to another, or to a platform people can run themselves.

In the example of Mendix, it is designed as a workflow engine but can also be used as a prototyping tool. Even though the tool is quite open, some parts like talking to other databases can be implemented, but this takes much working around limitations set up by Mendix. So to test a new feature, the database either has to be converted to Mendix and the data imported, or the database has to be copied to Mendix models, filled and only after new features can be added.

One of the advantages of the system designed for this research is not only that it is fully open but also that the possibilities are nigh limitless. The central part of this research is based on making it viable to use as a prototyping platform. However, these features also make it perfect as a verification environment in a more extensive system. The expendability and the verification are features not found or partly implemented in the above named platforms.

2.6 Wireframes and their place in the development of applications

Wireframes are an agile prototyping technique to sketch the skeletal structure of a Website or Web application [13]. Making wireframes is a collaborative and social process that involves all the stakeholders like designers, end users and developers. A wireframe provides a close representation of the final application. Consequently, a wireframe is more intuitive and feels more familiar to end users. This helps the users by providing a visual representation of what the pages will look like and ensuring that they don't visualise it in their head, which can lead to a difference in expectation. By showing exactly what the page will look like users will also more easily notice missing information and functionality.

Besides the advantages it has for the user, it also helps the designers and developers as they will know what types of pages they need to design. For the developer, it makes it easy to know exactly what functionality each page has, and it is a blueprint for exactly what the page should look like. This ensures that the development also is a lot easier and smoother since they know exactly what to do.

This is why a wireframe is chosen over other page representation methods.

2.7 Combination of above mentioned topics

Combining the application previously mentioned in 2.2 with the top choice of WYSIWYG editor from 2.3 and the limitations of 2.4 in mind, the focus was on being able to create a fully functioning prototyping system which can add, alter and display live data gathered from a database. This system has the benefit of being able to run in most systems and situations due to the usage of Docker Containers[14]. This ensures the running environment is the same regardless of the user's operating system. Besides that it prevents situations where some software might not be accessible on certain platforms.

3. System design and implementation

3.1 Design of metadata model

The initial data structure designed and implemented by Ralph Driessen[2] was robust and needed little changing to allow for all the designed functionality to be built. The main changes were the addition of 2 new subclasses and one other new class. These classes are Page and Section for the subclasses and the class Category as the last new class.

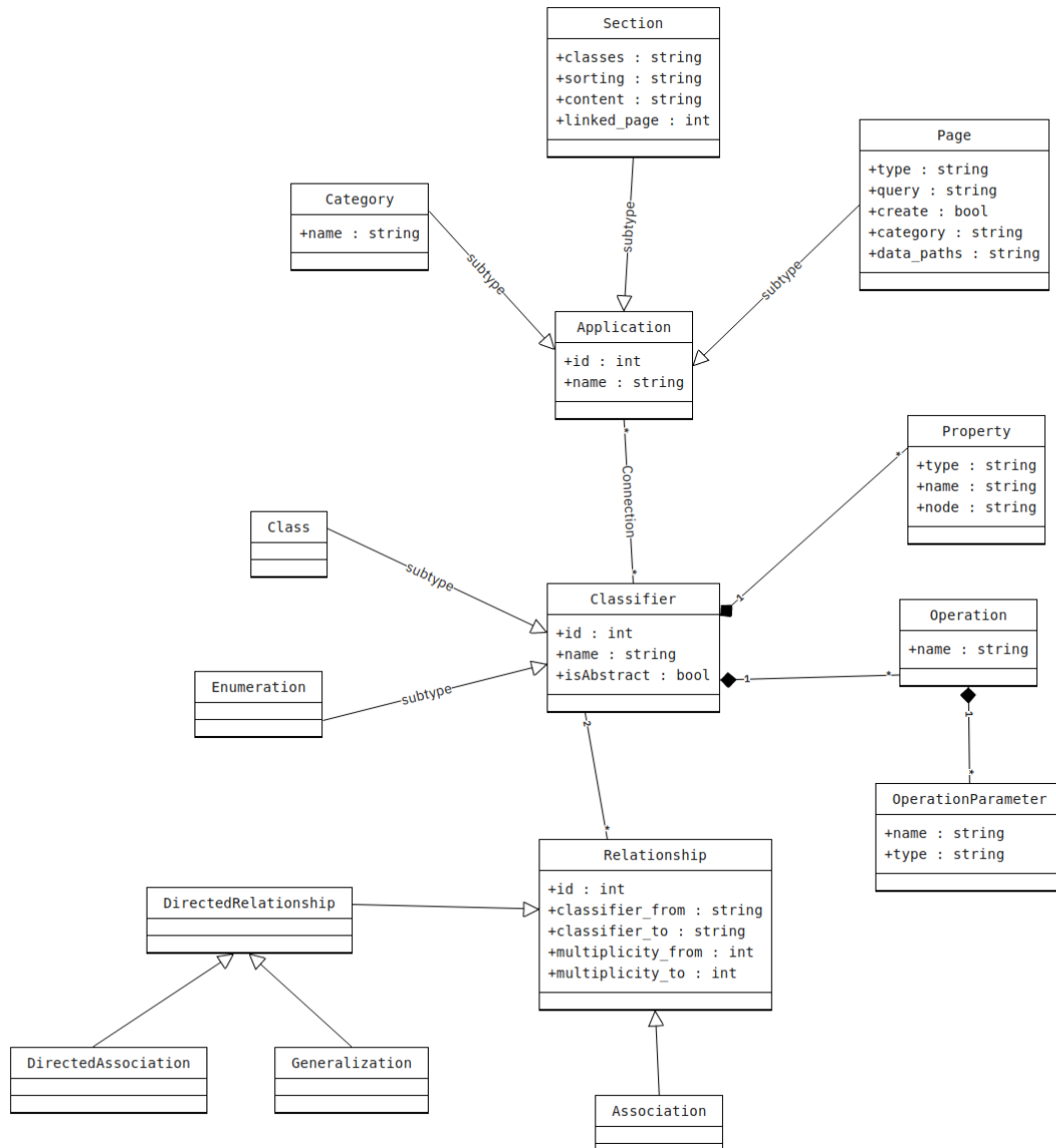


Diagram 2: The current data model.

The choice of going with subclasses instead of new base classes was made to more efficiently use already implemented features and for clearer code because of less duplication.

The Category class is mostly to group pages so they are more easily findable and a menu can be made, which can easily be displayed when viewing the custom pages. It does not use any of the built-in functionality an Application does so it did not make any sense to make the model more advanced.

A Section is a collection of HTML elements or text that can be linked with Classifiers and properties to eventually display the parsed HTML elements. It has a sorting to determine where on the page it will be displayed, classes to add custom CSS classes to the element and is linked to 1 page.

A Page is a collection of Sections that are displayed at once on a single page in the browser. It mostly holds properties that relate to data. Type, query and data_paths are all related to the gathering or holding of data.

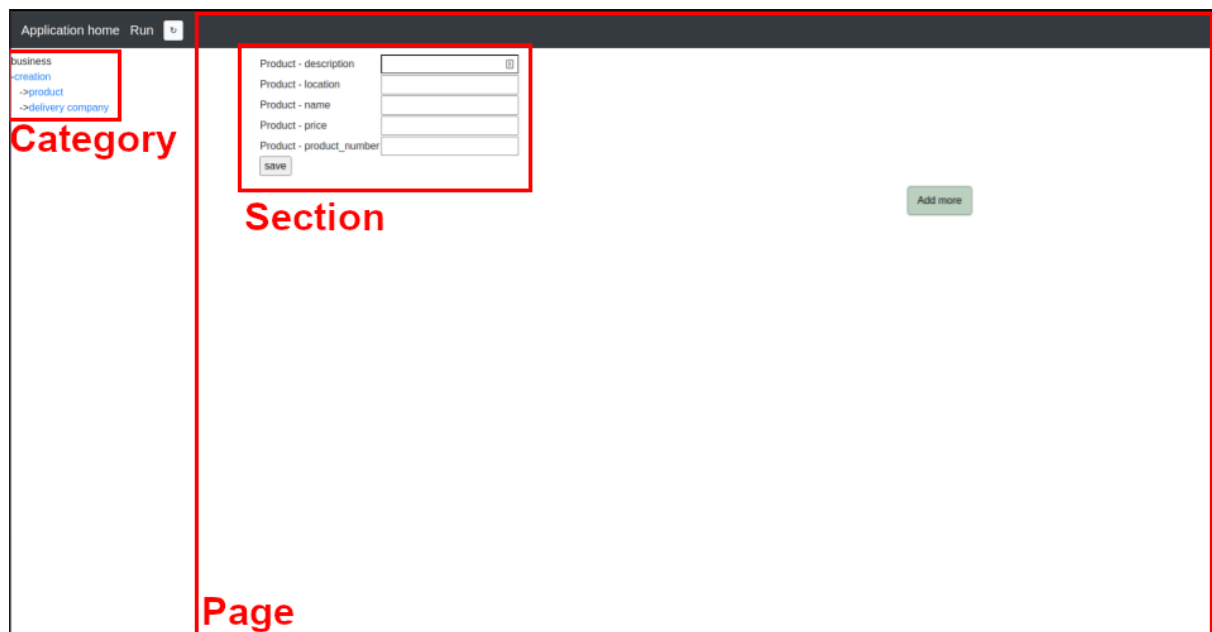


Diagram 3: An example of a Category, Page and Section.

Both Page and Section are subclasses of Application, this is because they both share the properties that are already in use by Application. Application is a class designed by Ralph Driessen [2] it is used mostly to link Classifiers and Properties to. Which in his application was only used to generate a simple application to insert data. It does not allow for any alterations compared to the computer-generated pages. As mentioned above this limitation is overcome by the addition of the combination of Sections and Pages.

Using Application as its main model makes it so that the usage of these properties and functionalities are already included and easy to extend. Most importantly, the link to the classifiers is used to clarify which classifiers and respective properties are used by both the Page and Section respectively. Both also have their respective unique properties.

For Section we have the following unique properties:

Table 1: Section properties

Property	Use of property
----------	-----------------

classes(string)	The extra CSS classes that are added to this page section. This can be used to style certain sections a specific way, to allow for a large amount of flexibility.
sorting(string)	The string denoting where in the page the section should be displayed. An example of this string is 2.1, this means it is the first child element in the second element on the page. 2.1.1 would be their first child. This ensures that the sections are properly displayed on the page. At first, the usage of relationships was thought of however, then people would have less control over the order the sections will be displayed.
content(text)	The LONGTEXT which holds all the HTML for the section. This property holds all the HTML made by the WYSIWYG editor. It holds the strings that need to be replaced with data in the following format: {{ Classifier-Property }}. By using this string over just using Property or Classifier-Property, it is almost ensured that the data will not be inserted in a place where it is not wanted. If the word Property would be used in a different context, we would not want it triggering the replacement.
linked_page(*-1 relationship)	The link to the page the section should be displayed on. This links the section to its parent page. Choosing a relationship was easy as it only needs to be connected to one, and the sorting will denote the place on the page.

For Page we have the following properties:

Table 2: Page properties

Property	Use of property
type(string)	The name of the main Classifier. This is used to gather the classifier's relationships to display the correct classes and properties that can be linked to the page. It is also used to know where to apply the filters later filled in by the user.
query(string)	The filters that will be applied to the main classifier to limit the resulting data.
create(boolean)	A boolean to determine whether or not the page will be used to create data. This boolean is used to ensure that the input fields are always displayed and that there is never any data being loaded. The system automatically loops through results to determine how

	many times a specific Section needs to be displayed. This, however led to a problem where the input fields did not show up or showed up filled with data. To counter this the property was implemented.
category(** relationship)	This links the page to a category. The choice of a many to many relationship was chosen as later we might want to be able to put pages in multiple categories.
data_paths(text)	The JSON string that holds the tree of how all possible Classifiers are accessible. This is the result eventually given by the function described in 3.3.1.

These are all the added properties as of finishing this iteration of the ngUML backend framework.

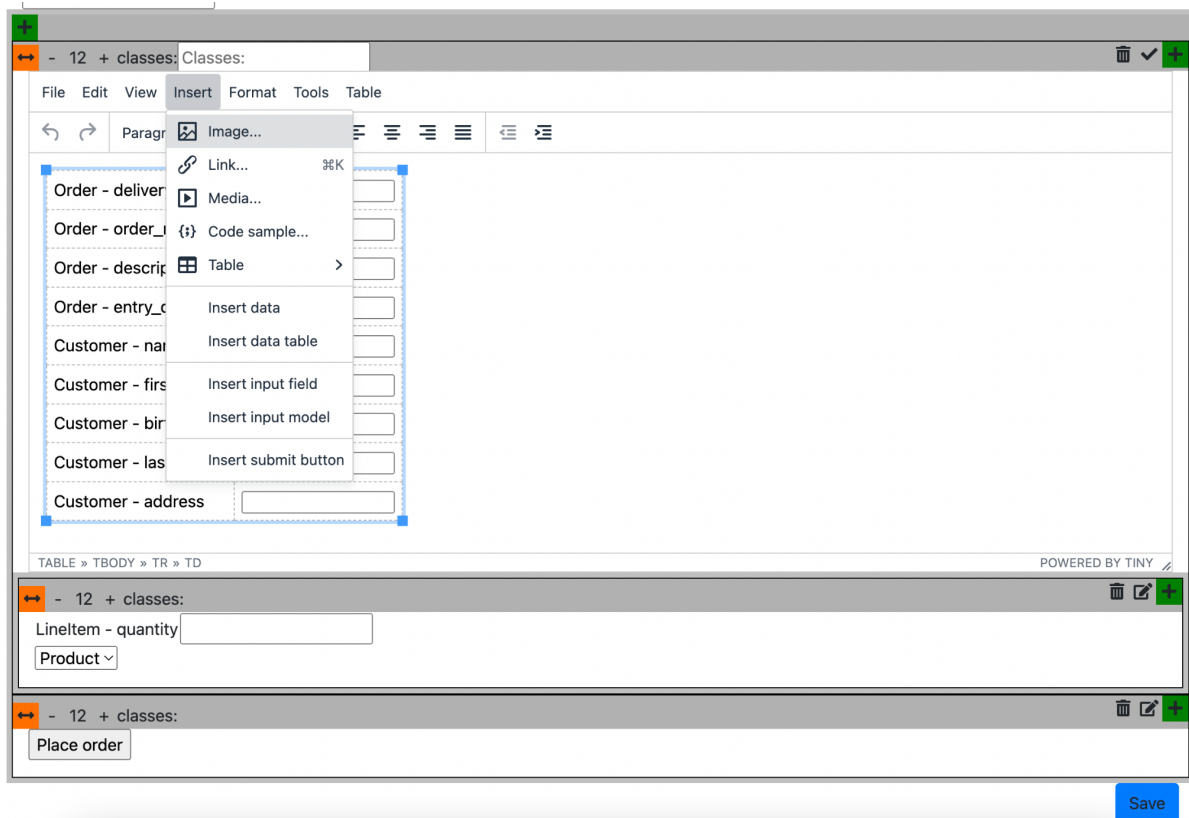
3.2 Front-end

Front-end refers to front-end development. This is anything that a user interacts with on an application [7]. In this research, this term will be used to describe anything a user sees and interacts with. One of the main considerations when making a front-end is the User Experience (also named UX).

One of the main choices of UX for this research was how to insert data into a Section, for this we chose a WYSIWYG editor. A WYSIWYG editor makes it easy to see and show what a page will look like when it is accessed.

For the front-end, we chose to use Django's built-in HTML templating system. This is because the emphasis is on the data not necessarily the speed. We looked at using a system like React as part of our UML Modeler is written in that framework, but decided not to use it as it would be counterintuitive because it is not designed for a system as open as this one.

The system right now works by the ID each page is assigned. Using that information it gathers all the information from the backend to display a page, with data where needed. That ID is also used when someone wants to access other aspects of the page. Other aspects that can be checked are the main model a page is linked to and its filter, the properties that are linked to the page and lastly the HTML content editor.



The current frontend editor for pages.

The HTML editor is an implementation of Tiny, combined with custom plugins and custom javascript code to insert data and other custom objects for the wireframe system.

As shown in the above picture, the following options are custom plugins:

- Insert data
- Insert data table
- Insert input field
- Insert input model
- Insert submit button

These do what their names suggest. The first two are mostly focused on displaying data, whereas the last three are more focused on inserting new data or editing existing data. To get all the selected classifiers and properties a JSON-like structure is made by passing the selected Classifiers to the Django HTML template. These are then looped through, the entire Classifier is added to the list, and then all Properties will be looped through. These are then also added to the JSON list. When the full Classifier is selected, all their Properties will be added to the list of selected Properties to add. So the custom plugins can be seen a bit as an API to make sure the metadata from the Django system is gathered and shown on the right pages at the right places.

Besides the custom features in the WYSIWYG editor, some work was done to ensure Sections can be created, deleted and moved. This was all done using javascript. Which was necessary as the contents of a page must be altered after the page has been loaded.

For adding a section, a default empty div is placed in the children div of the div where the green plus button was pressed.

It is also possible to move sections. This is currently possible in their respective wrappers. Children cannot be moved out to the level of the parent or even higher. This choice was made to prevent an accidental move higher. This is not very easily visible but can lead to a different page outcome.

3.3 Backend implementation

For the data processing in the backend a solution had to be thought of to be able to ensure data is only displayed once when needed, but also to have the possibility to display some information multiple times. Eventually, a tree structure was decided to ensure the best functionality combined with the best data integrity.

To handle changes Django allows users to add, remove and change properties as well as classes because of its migration system[18]. Running these migrations in a thread in the background is done during runtime. The system call to run the migrations is run in a different thread to ensure no caching occurs and is done in the backend using a system call to ensure users do not have to wait for it to finish to do anything else.

3.3.1 Making use of relationships

The looping through the relationships makes use of this adapted version of the Dijkstra algorithm:

```
def loop_through_relationships(relationships, paths, added, new_paths,
currentpath, model):
    for relationship in [rela for rela in relationships if (rela.multiplicity_to
== '1' or rela.multiplicity_from == '1') and ((
        rela.classifier_to.name not in paths or (
            rela.classifier_to.name in paths and
len(paths[rela.classifier_to.name]) > len(currentpath) + 1)) or (
            rela.classifier_from.name not in paths or (
                rela.classifier_from.name in paths and
len(paths[rela.classifier_from.name]) > len(currentpath) + 1)))]):
        if relationship.classifier_to.name == model.name:
            new_currentpath = currentpath + [relationship.classifier_from.name]
        if len(currentpath) > 0 else [
            relationship.classifier_to.name,
relationship.classifier_from.name]
        paths[relationship.classifier_from.name] = new_currentpath
        added.append(relationship.classifier_from)
    else:
        new_currentpath = currentpath + [relationship.classifier_to.name] if
```



```

len(currentpath) > 0 else [
    relationship.classifier_from.name,
    relationship.classifier_to.name]
    paths[relationship.classifier_to.name] = new_currentpath
    added.append(relationship.classifier_to)
    new_paths.append(new_currentpath)

```

What it does is loop through all given relationships and check if either of the multiplicities to the models is 1 as many to many relationships are not yet supported. If this is the case it also checks if either the new model does not yet exist in the list of known models, or the path to get there is smaller.

If this is the case, we change the new path to the model and append it to the list of known models with their paths. After the function is run, we rerun it with the new relationships of all the newly added paths to ensure we cover all the possible relations and the fastest routes to them.

Notice that this algorithm is simply an adapted version of Dijkstra's shortest path algorithm, which uses Django relations and models instead of paths and nodes.

When checking if this was the fastest possible algorithm, it was found out it is Dijkstra's shortest path but adjusted to fit the current Django implementation. In addition to this, Dijkstra is taught to almost everyone studying computer science and is very easily understandable. When researching further, the consensus appears that Dijkstra's shortest path is still the best algorithm to find the shortest path between nodes, even in most real world applications. This was not necessarily found through any one source, but looking at shortest path algorithms, Dijkstra appears to be the one always implemented. There are some cases where other algorithms can outperform Dijkstra [15]. But these cases are very limited according to the research. Upon checking it, this appears to not be one of the situations where that algorithm outperforms Dijkstra.

3.2.2 Using the algorithm

The saved pathing information mentioned in 3.2.1 is used to construct this tree. The tree is made up of a recursively recurring structure which is the following:

```

{data:[list of objects],children:[list of children where key of list is matches
to its respective data]}

```

By using this data structure and linking each classifier and class to every page and section we can smartly show some sections multiple times when there are multiple rows of data. To do this, the following function was written to find the highest possible parent usable for each section. The following function determines the highest parent and reshapes the data accordingly.

```

def get_highest_parent(page, result, children, paths, classifiers):
    if len(classifiers) == 0:
        return None
    if page.type in classifiers:
        return extract_data(result, paths, page.type)
    if len(classifiers) < 2:

```

```

        return extract_data(result, paths, classifiers[0])

path_copy = {}
for classifier in classifiers:
    path_copy[classifier] = copy.deepcopy(paths[classifier])

stop = False
parent = ""
deepest = 0
for classifier in path_copy.values():
    if len(classifier) > deepest:
        deepest = len(classifier)
while not stop:
    for classifier in path_copy.values():
        if len(classifier) == deepest:
            classifier.pop(-1)

    parent = ""
    for classifier in path_copy.values():
        if parent == "":
            parent = classifier[-1]
            continue
        if classifier[-1] != parent:
            stop = False
            break
    stop = True
    deepest = deepest - 1
return extract_data(result, paths, parent)

```

The function first checks if the main model of the page is in the classifier, if this is the case, we simply use that to get the data. Afterwards, it checks if more than 1 classifier is given, if not the one that is given is always the highest.

After that, the paths of each requested classifier are copied into a new dictionary. After which the deepness of the deepest possible classifier is taken, every classifier with that depth will be taken 1 step up. After checking if all classifiers have the same parent, the depth will be decreased by one if no matches are found, and this part of the function is executed until the highest parent has been found.

When the highest parent has been found, we call a different function to return the data in the type listed above. This ensures that certain sections of the page are shown multiple times but with different data to display the most amount of data possible and display it in a manner representative of the end product.

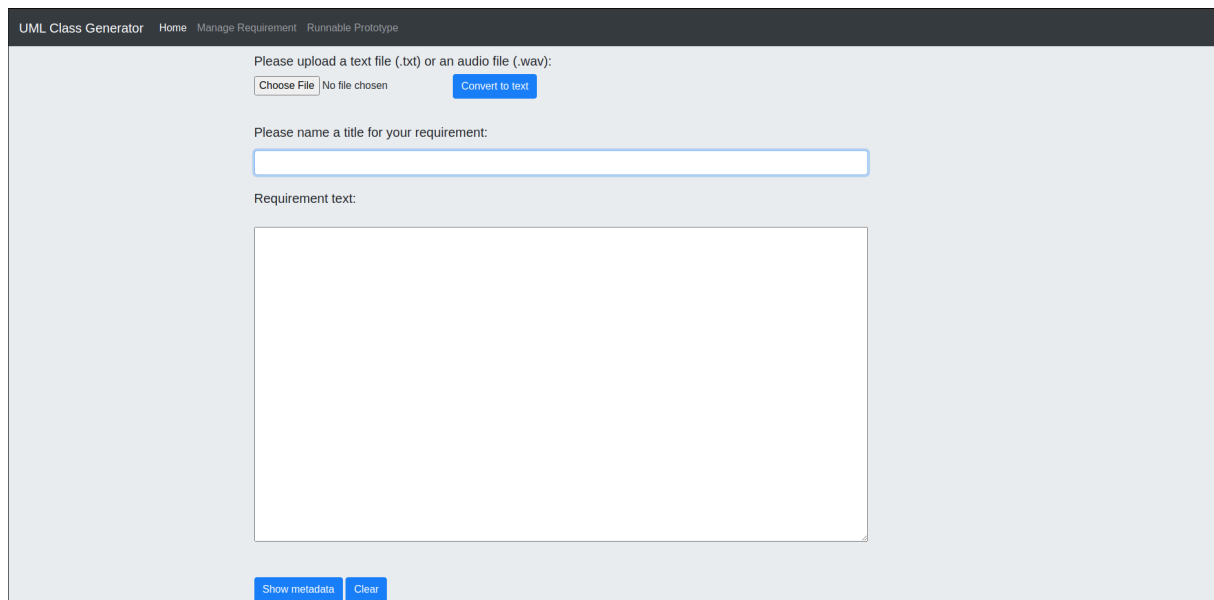
4. Worked Example

This worked example was used as a manual for both the testing and later for new students to get used to and find the full functionality of the application at the time. This is the reason why we and you are used in this part of the research.

4.1 Homepage

The worked example that will be shown is worked out in a custom codebase designed specifically for this application. All of the steps will be both shown and explained. It involves 2 different applications, the NGUML class modeller, which from here on is referred to as the class modeller, and the NGUML Django backend, from here on referred to as the backend.

We will start off by running through the demo script. When getting to the backend, the following screen will show, which allows you to upload a text or audio file or manually put in the requirement text.



The screenshot shows the 'UML Class Generator' application interface. At the top, there is a navigation bar with links: 'UML Class Generator', 'Home', 'Manage Requirement', and 'Runnable Prototype'. The main content area has a light blue background. It starts with the instruction 'Please upload a text file (.txt) or an audio file (.wav):'. Below this, there is a 'Choose File' button, a status indicator 'No file chosen', and a 'Convert to text' button. Further down, it says 'Please name a title for your requirement:' followed by a text input field. Below that is a 'Requirement text:' label and a large text area for input. At the bottom of the form, there are two buttons: 'Show metadata' and 'Clear'.

When the requirement text, found in appendix 1, is input, and the show metadata button is pressed, the NLP processing starts by finding classes and their attributes, after which it tries to find the relationships between the classes. When it is done, it will display the result at the bottom of the page and input them as classes and relationships in the backend.

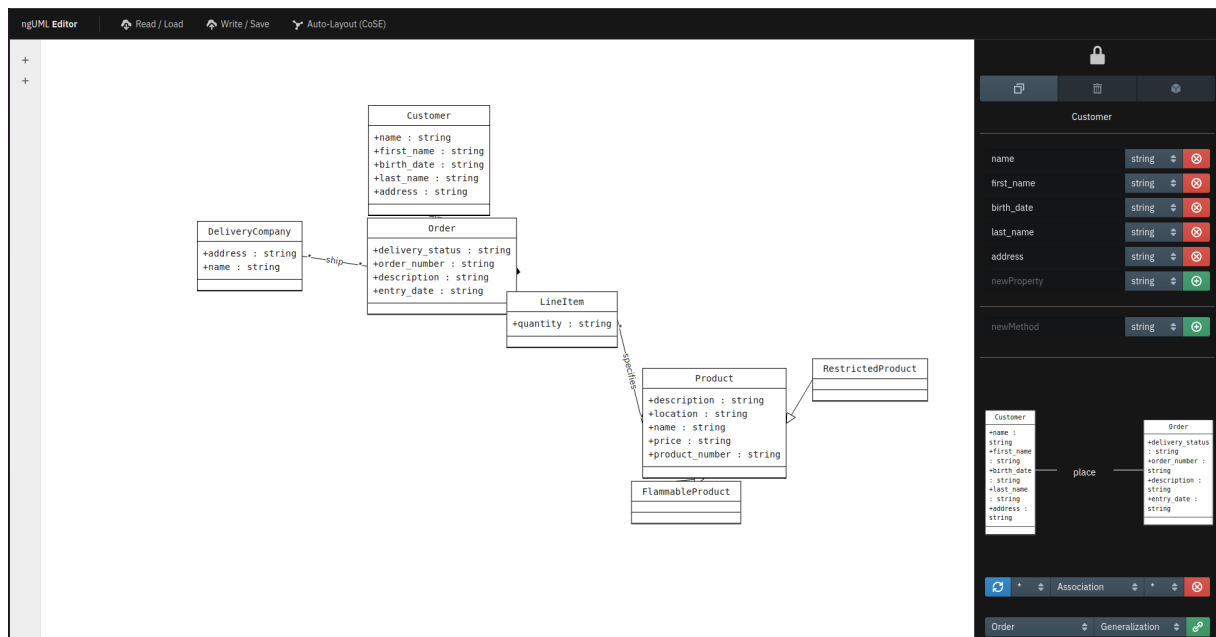
When the backend receives all this information, it will do a system call to ensure the latest changes are input in the database by calling the Django function to make migrations and run migrations. This is all done in the background and does not require user input.

4.2 Changing the data model in the ngUML editor

Next up open the editor, and load in the class model by clicking Load → classes, accepting that all changes will be discarded, and pressing auto layout to make it easily readable. This feature provides an aesthetically pleasing way to get a full overview of the data model easily. Other notable features of

the editor include the changing data types of attributes, adding attributes, adding methods and changing relationships.

The current view is something along these lines:



To get the application fully working in the way it should a few changes need to be made. Most importantly, some relationships need their cardinality changed, from many to many to many to one. This has to do with the lack of implementation of many to many relationships as of this moment.

The following changes need to be made:

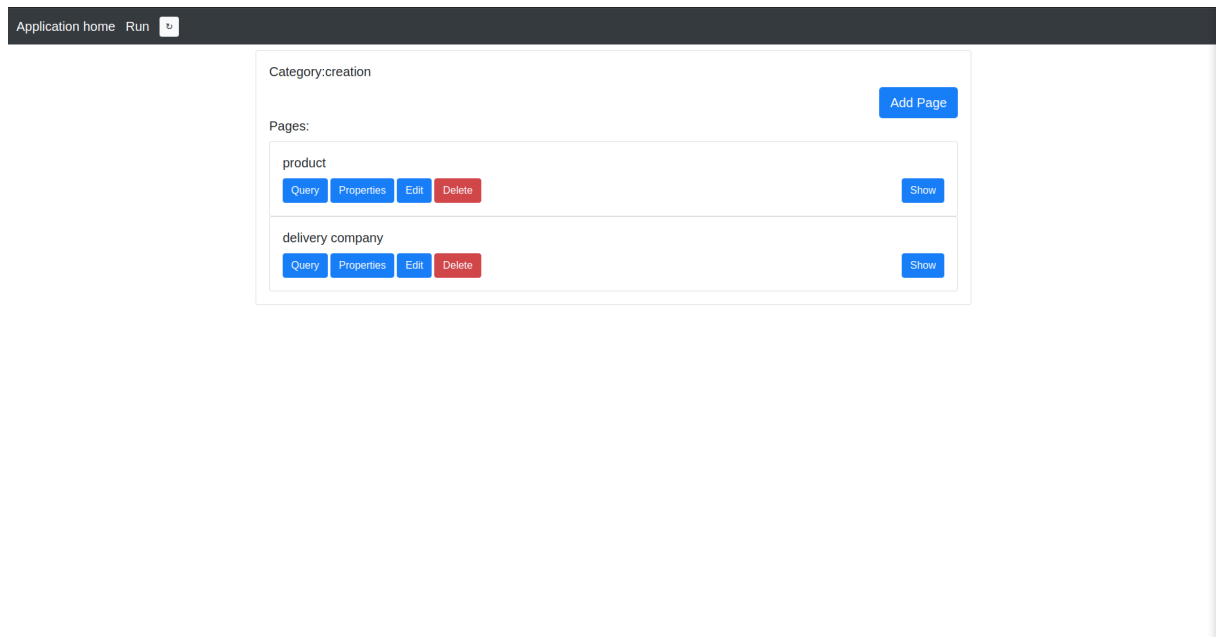
- Order – Customer (1)
- Order – DeliveryCompany (1)
- LineItem – product (1)

Where the (1) indicates the one in the relationship. Once these changes are made and saved by pressing write/save and confirming the changes need to be pushed to the backend the application is now fully ready for use.

4.3 Adding and setting up Applications

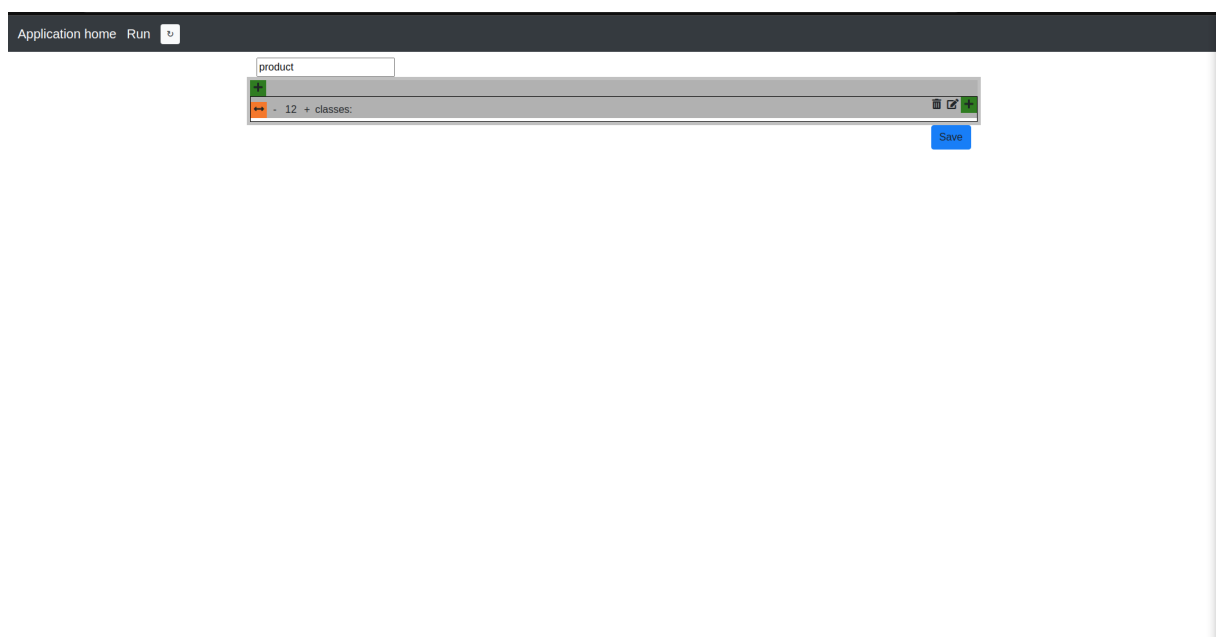
First, head back to the Django application and go to the runnable prototype. Here the classifiers/classes (from now on named classifiers) and relationships can also be viewed and edited. As seen in the modeller, all classifiers and relationships are correct, so the other part of the page, called applications, will be used. To display the capabilities of multiple Applications two will be made, called Client and Business. Upon completion, start by adding the right classifiers to the client and business application respectively, if wanted some attributes can be enabled and disabled.

First up is the business application setup, starting by creating a new category. This can be named anything you want. In this category, we will create all types users cannot alter/add, so for now the name Creation will do. Under that category, we add two pages named Product and Delivery company. Upon creation, the screen should look something like this:

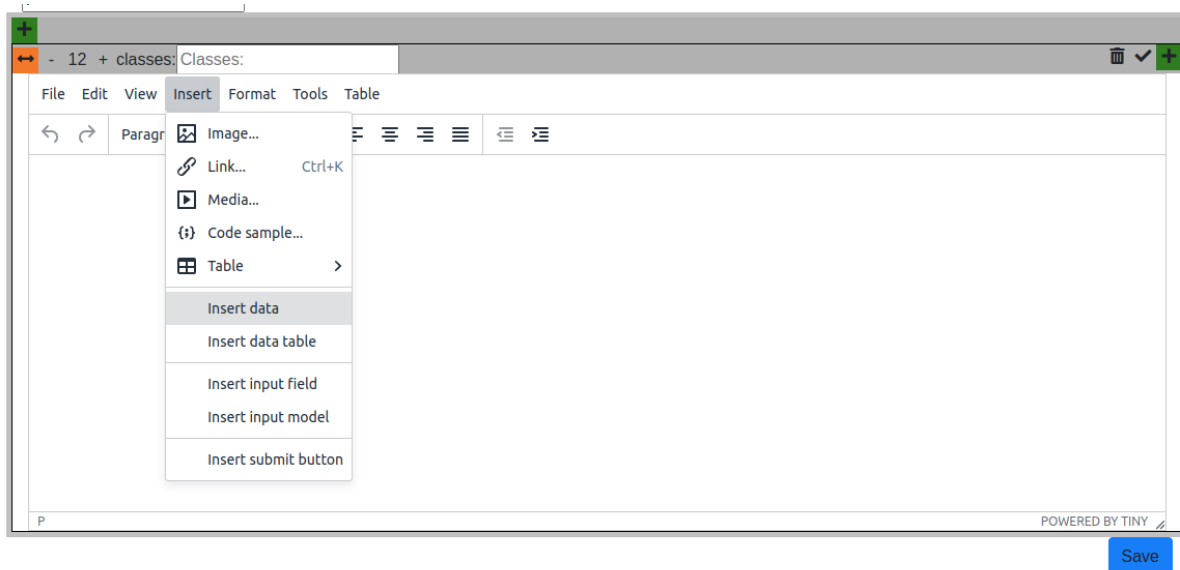


4.4 Creating pages

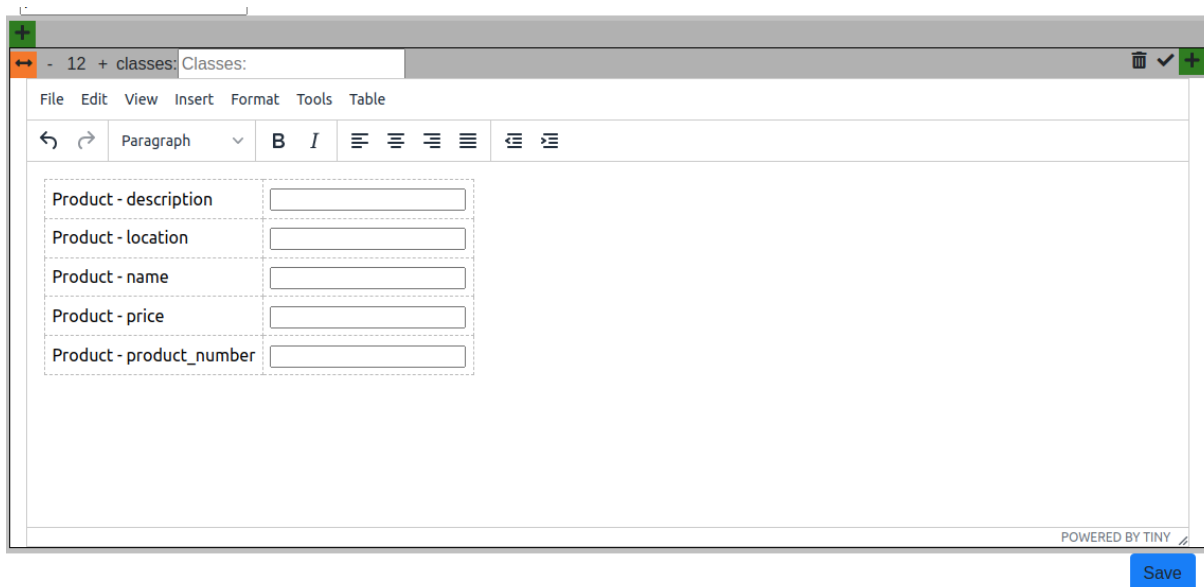
When setting up a page, the way to move is from left to right. The first step is to enter the query tab. The main model can be chosen, the toggle for "page for data insert " can be checked, and if wanted specify a query to narrow down the results. For this page, select Product, and tick the box for page for data creation. Once that is saved, move over to properties and select all of the properties linked to the Product classifier. To do this easily, click on the name Product. Now the most challenging part of creating pages starts. The HTML structure of the page needs to be designed. Upon pressing the Edit button, a screen like this will appear.



This is the starting screen. When the edit button is pressed, it will show and enable the custom HTML editor. It is standard in most ways, but under the insert tab, there are some custom features specifically designed for this application.



These are: insert data, insert data table, insert input field, insert input model and lastly, insert submit button. For now, choose the insert input field and select Product, which will select all fields. Upon pressing insert, a new insert table will be created.



After that, the only thing that needs to be done is to insert a submit button, save the page and view it. When the page is opened, the following page will show.

Now open the Client application, add the Order Customer Lineltem and Product classifiers and make the Order category. In that just make 1 page for now called create-order. For the main model, choose Order and ensure it's marked as a data insert page. For ease of use, link all possible properties. Afterwards, make the page something like this:

create-order

- 12 + classes:

Order - delivery_status

Order - order_number

Order - description

Order - entry_date

Customer - name

Customer - first_name

Customer - birth_date

Customer - last_name

Customer - address

- 12 + classes:

Lineltem - Product

Save

Save

These pages can be used to input and show data, and many more can be added as pleased.

5. Validation

5.1 Validation plan

For validation, a group of 6 developers will be asked to follow a set of prompts and fill in a very short questionnaire about them. The set of people is minimal yet decently diverse. It ranges from someone who has worked in IT for 20 years, mainly as a developer, to fellow students. Some of those students are involved or have been involved in the project and some are entirely new to it. This makes it so that the level of information about the project differs quite a lot, which makes the results very interesting.

The prompts try to test every part of the system to ensure they are doable and work as expected. Some prompts might be graded more difficult, but this will be fine depending on the prompt. Some are harder than others as they would usually be done by a dedicated team person on the designing side and not necessarily something that a possible client does. The result of the tests will be shared here but not used any further, so it can be used in future work as improvements or to steer the project in a better direction.

They will be asked to grade the prompts on the following scale:

1. Very easy
2. Easy
3. Average
4. Hard
5. Very hard

After grading the prompt on that scale they can add their notes on each prompt. The prompts focus on doing the following steps:

1. Adding a property and subsequently showing it on certain pages.
2. altering a method and adding a similar one to a different class.
3. changing a page to add the functionality of changing the selected object to a different one.
4. make a new page with a specific filter.

By doing the prompts, users will test the most critical parts of the system to ensure they work and work as expected

As the system does not allow for multiple completely separate systems for a part of the validation, a server had to be set up. This, thankfully, was quite easy by running everything in Docker containers, and then on the server itself running an NGINX[16] install. Every docker container had its internal ports kept the same, but the external ports mapped to 8001-8003. This allowed us to port forward a specific subdomain to a specific Docker container via its port.

For the editor, the idea was the same but unfortunately, it did not work so we ended up using the NPM package “serve”[17] in combination with React and NPMs built-in “build” features. However, when port forwarding was attempted, it was unable to load properly. Thus we simply left the ports open to the outside world and gave each of the three testers that used the server their own specific port ranging from 3001-3003.

The other two validators used branches that were set up for the testing on their own machines as they already work on the project and thus knew how to set it all up.

5.2 Validation by people involved with the project

The validation by people who were or are involved in the project was mixed. Two people answered all prompts as expected. The hard prompts were graded high on the scale of how difficult it was and the easy prompts as quite easy. This would suggest that the application is well put together and the features for each type of user are well implemented.

One person however, graded everything a bit to a lot harder than expected, this would suggest they know the application less well, and this leads to the main feedback given.

The main general feedback was that the functionality was good. However, the UX (User Experience) could be improved. This could be showing an example of how a for example a `__str__` method should be implemented or the way the filter property needs to be formatted and filled. This is something that the worked example of this thesis should do. However, it appears it does not do it quite well enough. Maybe a more general manual with more information or a video/workshop could help resolve this issue.

5.3 Validation by outsiders

The validation group of people outside of the project involves people who have had no earlier interaction with any part of the project. This was done by 1 Data Science student, 1 Aerospace engineer and a developer who has worked as a website developer for close to 20 years. The testing did not go flawlessly for these people as some Docker containers had crashed multiple times during the validation. This led to repeated white screens for the users.

The users were generally aligned to the people known with the project though they were a bit more divided on the toughness scale. Generally, they did confirm roughly what the insiders said which was that some of the prompts were hard to very hard, and some were more doable. This is roughly as expected as some of the prompts were more focussed on the developers compared to the end users or testers.

The main takeaway was again that the UX is not great but the functionality is there. Again, this would suggest a better manual, a video, a workshop or a UX redesign could help improve the usability.

6. Conclusion and future work

6.1 Conclusion

This research has put forward a roadmap for a dynamic data-driven wireframe approach to prototyping based on evolving design time metadata.

The roadmap defines an approach based on Model Driven Engineering using the Unified Modeling Language and combines this with graphical UI driven prototyping using 'live' user data. This approach is implemented in the Django based LIACS Prose to Prototype / ngUML Software Development environment.

It uses the Django based ngUML tool and expands on it by adding 3 classes, a new front-end interface and a graph-traversal algorithm to group and manipulate data.

What is a minimal and effective set of metadata extensions to a UML class model to enable this?

The minimum is adding Category, Page and Section to the existing datamodel. Each of these holds an important part of the datamodel to allow for the new features. A detailed explanation is located in Chapter 3.1

How can an execution engine support all this functionality in an efficient manner?

By structuring the data in such a way that everything is linked correctly. Both in terms of pure data saved in the backend and how the parts of the pages (Sections) are held together. The more detailed answers to this question can be found in chapters 3.2 and 3.3

What is the optimal graph traversal algorithm to manipulate complex data graphs in such a context?

The fastest graph traversal algorithm in this application is Dijkstra. It is widely considered to be the best and fastest algorithm to walk around a graph. Considering the relations in the application can all be mapped in such a way they form a graph it is the best algorithm for the job. This question is answered in chapter 3.3.1.

Which types of data migrations can be supported automatically in such a metadata driven environment?

As of right now, the changing of datatypes, the adding of new properties and the deletion of properties is possible in the current environment. These are the ones automatically possible in the Django framework. This is also discussed in chapter 3.3.

Then the main question which was *how can a wireframe UI design be driven from a conceptual model in such a way that the solution is dynamic, in that modifications to metadata immediately result in changes to the UI, and the wireframes manipulate live data*

The answer is, by using a system in which the datamodel is central but also has an HTML parser which can make and edit wireframes which can be filled with the live data from the system. It should use a system which implements the lessons learnt from the above named subquestions.

As you can see, most of the important questions were answered, however, some remain not answered enough or new questions came up, these will be discussed now in the future work chapter.

6.2 Future work

During the project some things that were unfortunately unable to be done or were left out of the scope for other reasons. Some of these things are an interesting continuance of this work and will be named in the following sections.

6.2.1 Extending relationships

As of right now it's only possible to use one to many or one to one relationships in the system however adding support for many to many relationships would greatly benefit the application as it would not only extend viability but also would simulate the real world more closely. Unfortunately adding this was not in the scope of the project as it would add too much extra time to the research and building the feature into the application.

6.2.2 Linking to activity models

At the start of the project this was part of the scope. This is why the ngUML editor was changed and activity models were added, however soon after it was determined that this unfortunately would take too long to design and implement. Adding this would make the system more versatile and add a lot of custom features that users (mostly developers/ UI developers) could implement. By adding full functionality for activity almost all features you could think of could be implemented. For example it would allow for a full workflow to be created and executed without having to leave the application. This would mostly help users understand exactly the new functionality of a new system and the full workflow of any action. This will ensure that users can easily tweak small issues even during the design stage.

6.2.3 Changing filters

One of the things that is currently a limiting factor are the filters, as they can currently only be placed on the main model. Something that might be possible is to extend this to the relationships as well. Using the algorithm to build the data tree as described in 3.3 with some slight alterations, the application of filters could be added.

Another nice to have feature would be to allow some standard functions like a function for a date to be used. This so a timed overview like a quarterly report page can be created very easily.

These things were not in the scope of work but could be nice to have to make the application more viable for certain users.

6.2.4 Changing data types while running

This is one of the big points one of my supervisors asked to be looked at. As the system uses Django in the background, changing the data model also changes the database. What would happen if we were to change the datatype while running? Would the existing data that does not comply with the new data type be thrown away? Would it display weird symbols from converting from int to string, or a large number the other way around? This would be a part that could be looked at and choices would have to be made. This could be something that might be combined with the next point under the umbrella of ensuring run and uptime.

6.2.5 Self error correcting

One of the things that came out of the validation process was that the application could completely shut down when an operation is not done correctly or a relationship is added twice. One of the possible things to add to the system would be a way to check the code being added for python validity and to see if properties are not yet selected and added before adding them. While this would add extra calculation overhead and would make the program a little slower, it would be greatly beneficial to ensure that the system does not go down when a client is trying to use the system, and thus, be improve the usability of the solution.

6.2.6 Better UX design

From the answers to the prompts it became very clear that the UX design needs to be improved. This would improve the usability of the application a lot. One of the main things the participants of the questionnaire ran into was that it was very hard at times to find what exactly they wanted to do. Changing the product with this in mind would be greatly beneficial.

References

- [1] Tang, Tiantian, From Natural Language to UML Class Models: An Automated Solution Using NLP to Assist Requirements Analysis, Thesis Master ICT in Business, LIACS, Leiden University, 2021. <https://theses.liacs.nl/1819>
- [2] Driessen, R.: UML class models as first-class citizen: Metadata at design-time and run-time. Informatica en Economie thesis <https://theses.liacs.nl/1836> .
- [3] Object Management Group. Unified modeling language. <https://www.omg.org/spec/UML/2.5.1/About-UML/> 2017
- [4] Ciccozzi, F., Malavolta, I. & Selic, B. Execution of UML models: a systematic review of research and practice. *Softw Syst Model* 18, 2313–2360 (2019). <https://doi.org/10.1007/s10270-018-0675-4>
- [5] <https://www.djangoproject.com/>
- [6] <https://monkeylearn.com/natural-language-processing/>
- [7] What is a front-end developer
<https://www.freecodecamp.org/news/front-end-developer-what-is-front-end-development-explained-in-plain-english/>
- [8] Tiny <https://www.tiny.cloud/>
- [9] Tiny | security <https://www.tiny.cloud/docs/advanced/security/>
- [10] Appian. <https://www.appian.com/platform>.
- [11] Mendix. <https://www.mendix.com/platform>.
- [12] Microsoft power platform. <https://powerplatform.microsoft.com/nl-nl/>.
- [13] Arnowitz, J., Arent, M., Berger, N.: *Effective Prototyping for Software Makers*. Interactive Technologies, Elsevier Science (2010)
- [14] Docker <https://www.docker.com>
- [15] Golden, B. (1976). Shortest-path algorithms: A comparison. *Operations Research*, 24(6), 1164-1168.
- [16] NGINX <https://www.nginx.com/>
- [17] serve <https://www.npmjs.com/package/serve>
- [18] Django migrations <https://docs.djangoproject.com/en/4.0/topics/migrations/>

Appendix

A.1 Standard requirement text

An order is placed by a specific customer. A customer has a first name, last name, address, and birth date.

The order consists of multiple line items. Each order has an order number, an entry date, a delivery status, and a description.

A line item specifies a particular product, and defines the quantity that is ordered.

A product is characterized by a name, a description, a product number, a price, a location.

Restricted products and flammable products are types of products.

Each order is shipped by a delivery company. The delivery company has a name and an address.

A.2 The prompts for the questionnaire

Dutch	English
Voeg onder Product de property stock_level toe en laat deze terugkomen op de order-overview en product aanmaak pagina van de Business applicatie. Vergeet na het aanmaken van de property niet de info op te slaan.	Under product add the property stock_level and display it on the order-overview and product creation page of the Business application. Don't forget to save after creating the property
Pas de method __str__ van Product aan en voeg de property stock_level. Voeg daarna voor Delivery Company de method __str__ van type string toe. Vergeet na het toevoegen de veranderingen niet op te slaan. Kopieer beide methods na het aanmaken in de questionnaire.	Alter the method __str__ of Product and add the property stock_level to it. Afterwards make the method __str__ of type string for Delivery Company. Don't forget to save after making these changes. Copy both methods after creation into the questionnaire
Voeg op de order-overview pagina de mogelijkheid toe om een delivery company toe te voegen.	On the order-overview page, add the possibility to add a delivery company
Maak een nieuwe pagina aan in de Business applicatie genaamd day-overview en zorg dat alleen de Orders van vandaag zichtbaar zijn. Om een filter in te vullen druk je op de query knop op het pagina overzicht in de categorie.	Make a new page in the Business applicaiton named Day-overview and make sure only the Orders of today are visible. To apply a filter, press the query page int he page overview of the category.
TIP(gebruik deze alleen als het nodig is): gebruik voor de property de naam order_date en vul handmatig de datum van vandaag in.	TIP(use this only when necessary): use the property order_date and manually fill the date of today.

A.3 Answers to questionnaire

Answers to prompt 1		
Answer (1-5)	Note in Dutch	Note in English
5	<p>Ten eerste hadden we de property stock_level aangemaakt. Hierna hadden we in de Product de stock_level aangevinkt bij de "Link properties". Verder hadden we bij de Categories -> product -> view -> properties de stock_level aangevinkt en bij edit de data tabel inserted. Ook hadden we bij Categories -> order -> view -> properties de stock_level aangevinkt. Uiteindelijk hebben we nog steeds niet de stock_level op de order-overview en product aanmaak pagina van de Business applicatie gekregen.</p> <p>Overigens, wanneer we bij de properties één property stock_level: int hadden en één property stock_level: string, crashte het gehele systeem</p>	<p>At first we made the property stock_level. After we checked it at product under link properties. Besides that we also selected stock level under Categories -> product -> view -> properties and inserted it in the data table. We also checked stock_level in Categories -> order -> view -> properties. Eventually we still did not get the stock_level on the order-overview or new product page of the Business Application.</p> <p>Besides that, when we had 1 property named stock_level with type int and one property named stock_level with type string the entire system crashed</p>
2		
1		
2		
2		
4	De bewoording van "order-overview en product aanmaak" was een aan de vage kant en daardoor niet goed te vinden	The wording of "order-overview and product creation" was a bit vague and hence not easy to find

Answers to prompt 2		
Answer (1-5)	Note in Dutch	Note in English
4	Het aanpassen van de method van de stock_level was eenvoudig. Het maken van de str voor de Deliver Company is eenvoudig als je exact weet wat je moet doen, maar als een gebruiker zonder verstand dit zou moeten doen spreekt het niet voor zich voor diegene wat hij moet doen.	The editing of the method of the stock_level was easy. The creation of the str (method) is easy if you know exactly what to do, but a user with no knowledge would not know what to do.
5	DB print van methods: 1 __str__ string return self.name 4 2 __str__ Type.String return self.name 7	DB print van methods: 1 __str__ string return self.name 4 2 __str__ Type.String return self.name 7
4	Eerst geprobeerd: return self.name + "(" + str(self.stock_level) + ")". Geeft bij opslaan wit scherm? return self.name in DeliveryCompany werkt wel.	First tried : self.name + "(" + str(self.stock_level) + ")". Gives a white screen at saving? Return self.name in DeliveryCompany does work
1	return self.stock_level + " : " + self.name	return self.stock_level + " : " + self.name
3	return self.stock_level return self.name	return self.stock_level return self.name
1	Voor Product: def __str__(): return self.stock_level Voor Delivery Company: def __str__(): return f"{self.name} at {self.address}" --- Comment: method editor geeft niet aan of de property die je probeert te returnen valid is	Voor Product: def __str__(): return self.stock_level Voor Delivery Company: def __str__(): return f"{self.name} at {self.address}" --- Comment: method editor does not indicate whether the property you're trying to return is valid.

Answers to prompt 3		
Answer (1-5)	Note in Dutch	Note in English
2	Eenvoudig om te doen via de page viewer.	Easy to do via page viewer.
4		
1		
1	werkte verbazend makkelijk en goed	Worked surprisingly easy and well
4		
1	Na het saven op de Query en/of Properties pagina zou het fijn zijn als deze je terug brengen naar de vorige pagina	After saving on the Query or Properties page it would be nice to be send back to the previous page.

Answers to prompt 4		
Answer (1-5)	Note in Dutch	Note in English
4	Redelijk lastig voor iemand die geen verstand hiervan heeft.	Reasonably difficult to do for someone with no knowledge of this.
5	Taal van de query is onbekend, en de opmaak van de querybox is vrij chaotisch	The language of the query is unknown and the formatting of the querybox is quite chaotic
5	als ik een pagina aanmaak en probeer iets op te slaan krijg ik steeds een wit scherm. Geen idee wat verschil tussen 'query' of properties bijvoorbeeld.	If i make a page and try to save it, i keep getting a white screen. No idea what the difference between 'query' or properties are for example.
5	Geen idee in welk formaat de query moet. Als dit verduidelijkt wordt en het werkt daarna meteen zou het wel erg makkelijk zijn.	No idea what the format for the query is. If that would be and it would work it would be very easy.
5	Verwacht een SQL (o.i.d.) interface, of juist iets drag-and-drop like. Begrijp niet hoe dit werkt met alleen radio buttons en een enkel tekstvlak.	(I'm) expecting a SQL (or similar) interface or something drag-and-drop like. Dont understand how this works with just radio buttons and a textfield
4	Bij deze prompt was het niet duidelijk of het de juiste uitkomst had	At this prompt it was unclear if the outcome was right

General feedback	
Dutch	English
Veel potential voor errors als je niet weet wat je aan het doen bent.	A lot of potential for errors if you do not know what you are doing
Mooie functionaliteit, niet heel gebruiksvriendelijk om in te stellen zeker zonder navigatieknoppen	Good functionality but not very user friendly to set up, especially without navigation buttons
Het principe is goed, ik denk dat alles er in zit, maar het is nog niet vanzelfsprekend hoe alles werkt. Waarschijnlijk denk ik moeilijker dan nodig is. Betere UX en handleiding zou dat al op kunnen lossen.	The principle is good, i think everything is in it, but it is not as self explanatory how everything works. Im probably thinking harder than needed. Better UX and a manual could solve this.