



**Universiteit
Leiden**
The Netherlands

Opleiding I&E

Interaction to enable a dynamic User Interface frontend
communicating with a backend UML runtime engine

Xue Jun Wang

First supervisor: Dr. Guus Ramackers
Second supervisor: Prof. Dr. Joost Visser

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

dd/mm/yyyy

Abstract

Unified Modelling Language(UML) is a modeling language used to specify, build and document the artifacts of a system. The purpose of this language is to align the interpretation of readers of diagrams, so that every reader knows for example the flow of a process, like in a flowchart.

This research and development is part of the ngUML/Prose to Prototype Software Development Environment project at Leiden Institute of Advanced Computer Science (LIACS). This thesis attempts to extend the User Interface of the ngUML/Prose to Prototype Software Development Environment project to make it possible to enable workflow and application flow between pages. However, these are only static pages where the user enters data. The User Interface needs to be extended to enable workflow and application flow between pages. This is achieved by expanding the current frontend and the backend. The frontend is extended with the functionality to link nodes that are made in the frontend with pages of the backend and the backend is extended with an engine for prototype execution. The result is a dynamic display of pages corresponding in the order of the nodes of the created activity diagram.

Even though the frontend and the backend of the ngUML/Prose to Prototype Software Development Environment project are extended, the application is not ready for business and individual consumption.

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Problem statement	1
1.3	Approach	1
1.4	Result	1
1.5	Thesis overview	2
2	Background and Related Work	3
2.1	Unified Modelling Language	3
2.2	Activity Diagram	3
2.3	Current State	6
2.4	Desired State	10
2.5	Related work	10
2.6	Event-driven applications	10
2.6.1	Usage of Event-driven applications	10
2.6.2	Wireframe	12
3	System Design	15
3.1	Logical Design	15
3.1.1	Requirements	15
3.1.2	Overview	15
3.2	Technical Design	16
4	Worked Examples	17
4.1	Page creation	17
4.2	workflow1	24
5	Conclusions and Further Research	30
	References	32

1 Introduction

1.1 Introduction

Unified Modelling Language (UML) creates visual modeling language for the construction, design and implementation of complex software systems, both structurally and behaviorally, using a collection of diagrams. It is similar to the blueprints used in other fields and consists of different types of diagrams. Broadly speaking, a UML diagram describes the boundary, structure, and behavior of a system and the objects contained within it.

1.2 Problem statement

Currently, there are static pages in which users are able to enter data. The existing research project “prose to prototype” has the functionality to interpret UML class metadata for prototype execution. However, these are only static pages where the user enters data. The User Interface needs to be extended to enable workflow and application flow between pages. This requires extending the prototype with controls and making it communicate with background execution. Also, the backend must be extended to include the new interaction model.

1.3 Approach

This thesis proposes to enable a dynamic user interface communicating with a backend. The final prototype consists of a User Interface that is able to support workflow and application flow between pages. Meaning a frontend that support linking pages within an application or workflow. This requires extending the prototype with controls and making it communicate with background execution.

1.4 Result

The result is software that will be built within the React framework and is able to perform processes at runtime. Meaning, that the deliverable is software, consisting of TypeScript-code including methods and const cooperating based on the metadata extraction as a result of user interaction with the prototype. So the current frontend needs to be extended.

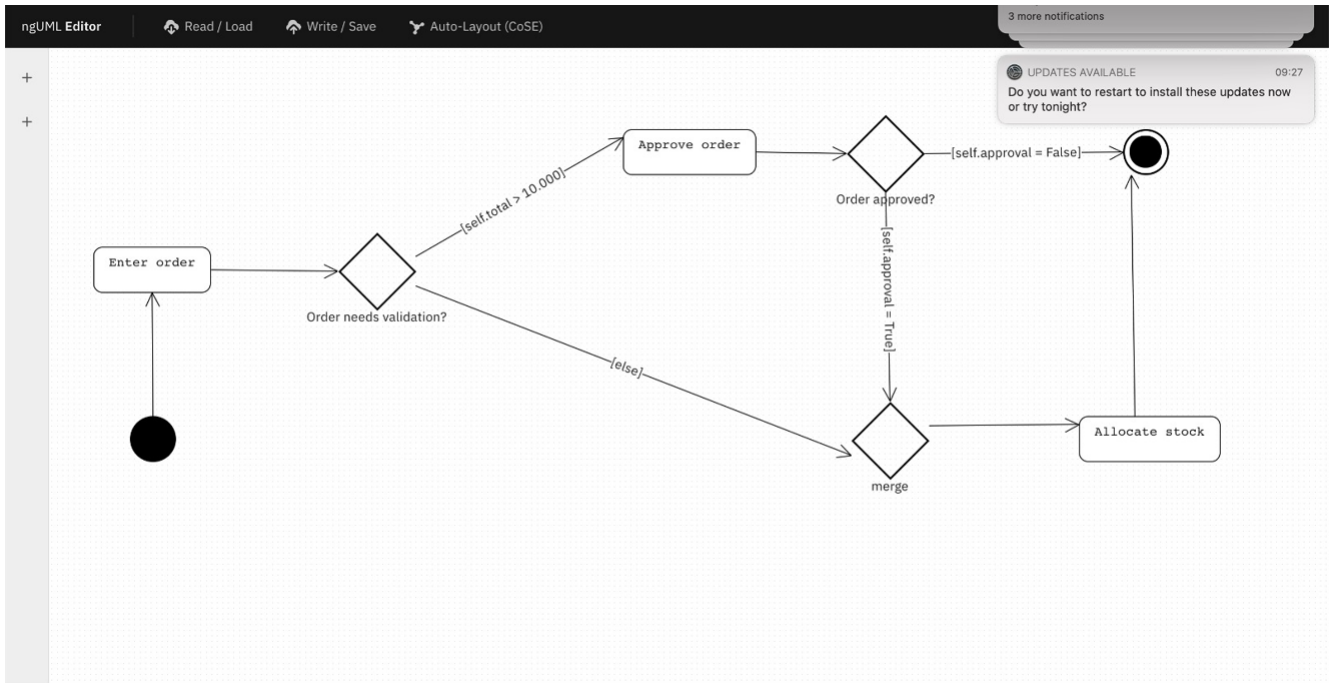


Figure 1: The flowchart

1.5 Thesis overview

Chapter 1 contains the introduction, where the current situation, problem statement, approach and the result will be discussed and explained; Chapter 2 provides the background of the ngUML project and the related work will be discussed; Chapter 3 describes the logical and technical design of the system; Chapter 4 shows and explains two worked example; Chapter 5 discusses challenges regarding this research and further research.

This bachelor thesis is the final assignment for the bachelor's study in Computer Science and Economics at the Leiden Institute of Advanced Computer Science from Leiden University. It is supervised by Guus Ramackers.

2 Background and Related Work

2.1 Unified Modelling Language

Unified Modelling Language(UML) is a modeling language used to specify, build, and document the artifacts of a system. The purpose of this language is to align the interpretation of readers of diagrams, so that every reader knows for examples the flow of a process, like in a flowchart. A flowchart is a schematic representation of a process, in which the process flow is represented by arrows and symbols[Luc]. A process has a clear start (the 'trigger') and leads to a predefined result in several successive steps (the activities).

2.2 Activity Diagram

For this report, the main studies are the component of UML which are Activity Diagrams. Unified Modeling Language is divided into several subsets of diagrams, including structure diagrams, interaction diagrams and behavior diagrams[IBMb]. For activity diagrams, an example can be seen in Figure 2. Along with use case and state machine diagrams, activity diagrams are considered behavior diagrams because they describe what needs to be done in the system being modeled.

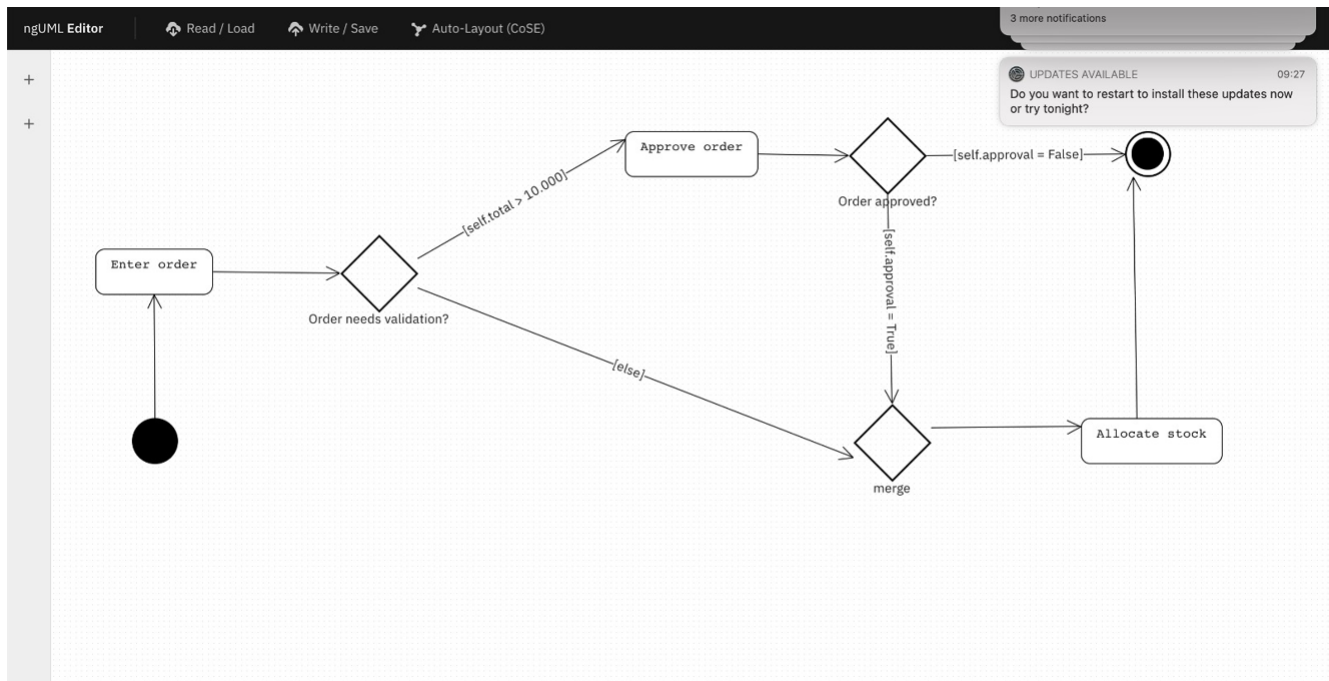


Figure 2: The activity diagram

An activity diagram helps people from the business and development side of an organization come together to understand the same process and behavior. In other words, a UML activity diagram

helps to visualize a particular use case at a more granular level. It is a behavioral diagram that illustrates the flow of activities through a system. UML activity diagrams can also be used to show a flow of events in a business process. They can be used to examine business processes to identify the flow and its requirements.

In an activity diagram, there are different components/nodes which are used to build a diagram[IBM]. A short description of those nodes is shown in Table 1:

Activity diagram nodes


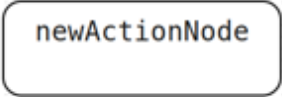
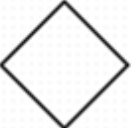




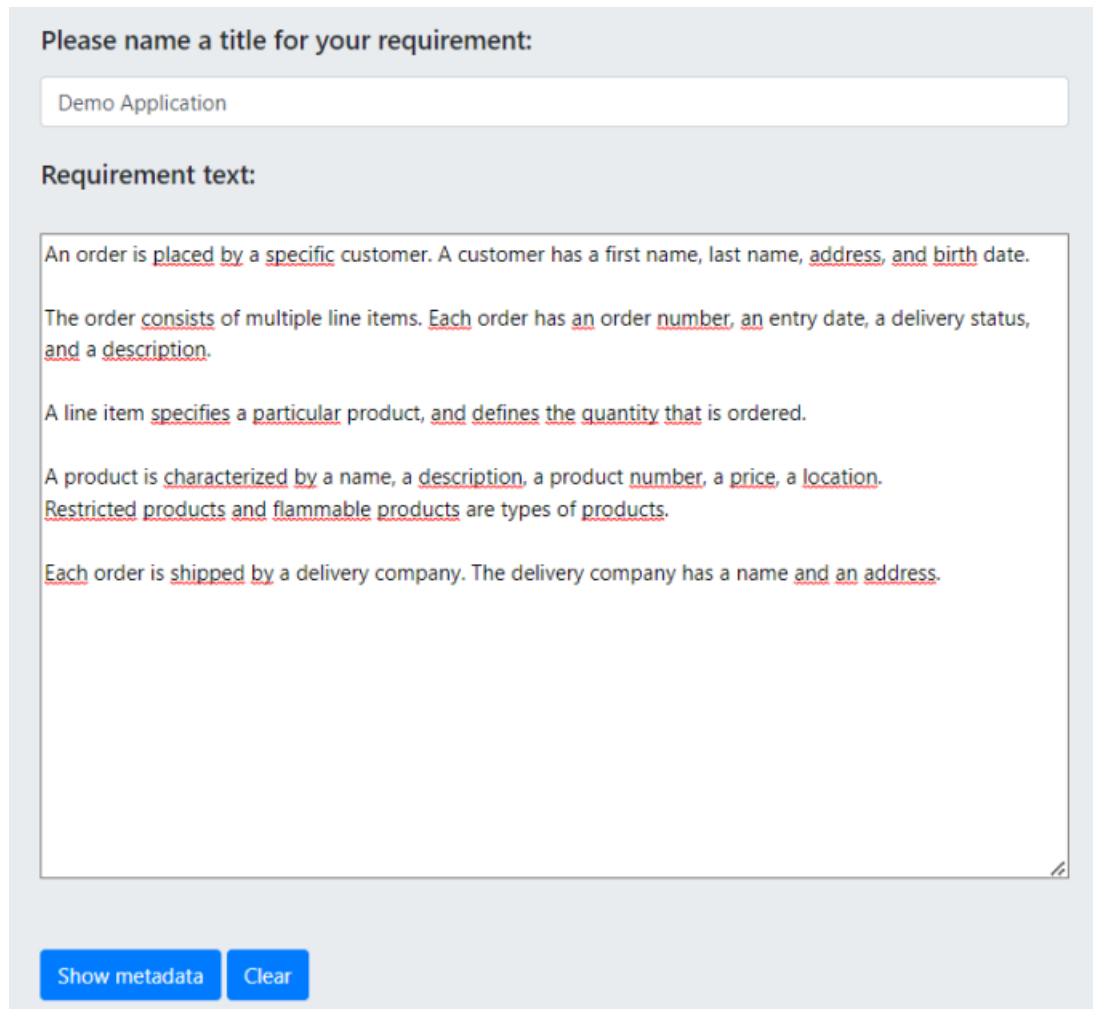
Node	Description
 Initial Node	This node represents the starting point of an activity flow.
 Action Node	This node represents the activities in a flow. It is a rectangular box with a short description of the action/activity in the rectangular.
 Decision Node	This node indicates a decision point and always has at least two branching paths with condition.
 Merge Node	This node represents a point in an activity diagram where several edges merge together into a single edge.
 Fork Node	This node splits one activity stream into several outgoing flows.
 Join Node	This node synchronized several incoming edges into a single outgoing edge.
 End activity Node	This node indicates the final state of an activity and symbolizes the completion of all flows in a process.

Table 1: The Activity diagram nodes and the description of each node

2.3 Current State

The ngUML software extracts metadata and is able to create UML classes from the natural language requirements text the user provides. Based on the class model, the software can create a demo application. In order to run the demo application. First, the requirements need to be defined. In figure 3 the requirements are defined as: *'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.'*



The screenshot shows a web interface for defining requirements. At the top, there is a label "Please name a title for your requirement:" followed by a text input field containing "Demo Application". Below this is a section titled "Requirement text:" which contains a large text area. The text area contains several paragraphs of requirements text, with certain words underlined in red. At the bottom of the text area, there is a small icon of a pencil. Below the text area, there are two buttons: "Show metadata" and "Clear".

Please name a title for your requirement:

Demo Application

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.

Show metadata Clear

Figure 3: the requirement text

After the requirement text is formulated. The backend will use natural language processing to transform the requirement text into a UML class model, this will happen when the button “*show metadata*” is pressed. When the backend is finished translating, a UML model with classes will be generated as illustrated in figure 4.

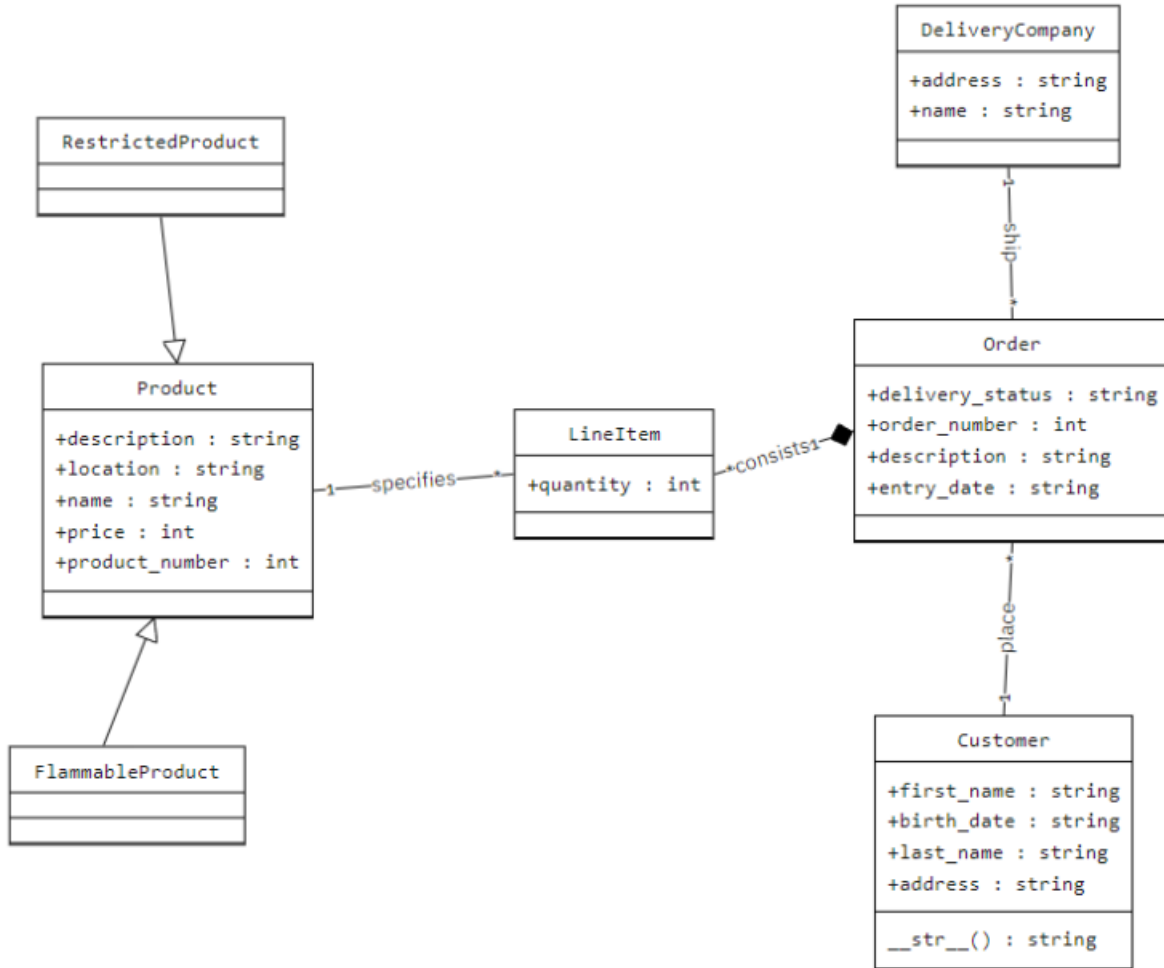


Figure 4: The UML model that is generated based on the requirement text

In the runnable prototype, in the section *domain model* as depicted in figure 5 the attribute values and relations between the attributes can be changed and deleted when an attribute is not needed for the application.

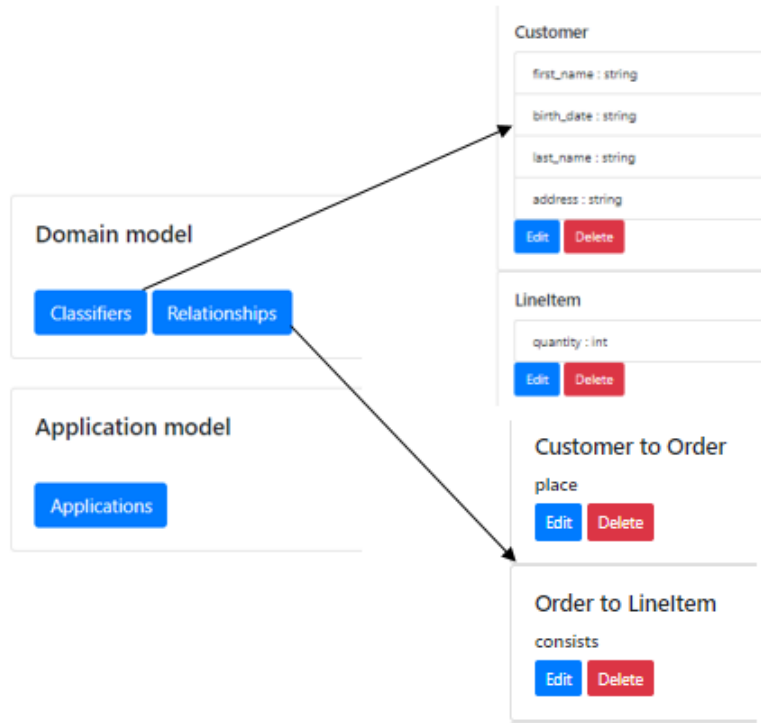


Figure 5: The section domain model, where attributes from each class and relation between classes can be modified

When the modification is done, an application can be created in the section *Application model*. From there we provide each class with data as depicted in figure 6

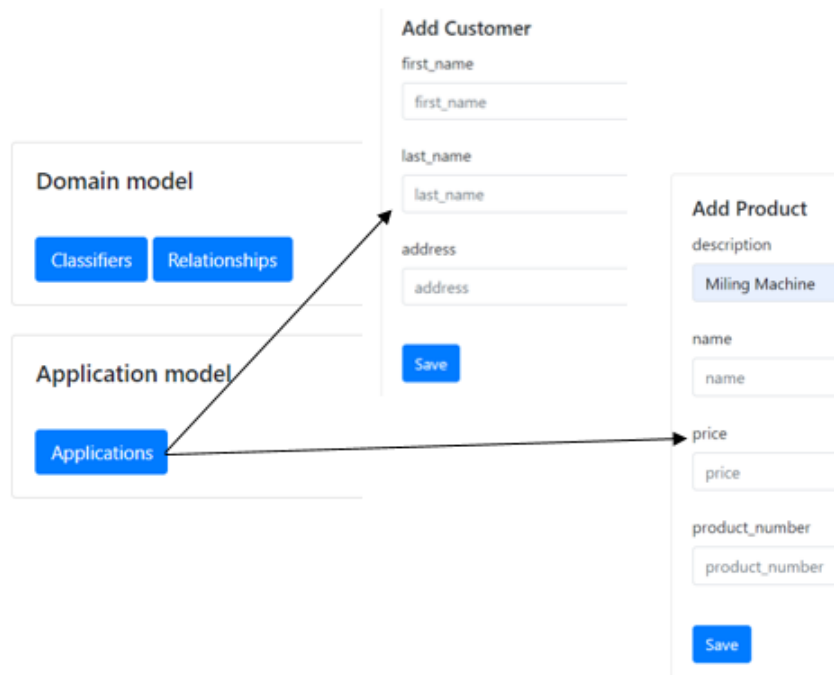


Figure 6: In the application model, an application can be created

When all the classes are provided with data an order list can be created with the data as illustrated in figure 7.

Order list

Id	Customer	DeliveryCompany	delivery_status	order_number	description	entry_date

Add Order

customer: Bilbo Baggins ▼

deliverycompany: ▼

delivery_status: delivery_status

order_number: order_number

description: description

entry_date: entry_date

Save

An 'Add' button is located between the 'Order list' table and the 'Add Order' form, with an arrow pointing from the 'Add' button to the 'order_number' field in the 'Add Order' form.

Figure 7: With the provided data an order list can be created

2.4 Desired State

In order to add the flow between pages that are part of an application or workflow, a workflow between pages must be added. Meaning, in the desired state, a page will be shown for each node in an activity diagram at runtime. For example the activity diagram in Figure 2.

As described above, the software will be built, where for each node, a page will be shown at runtime. As described in the flowchart of Figure 2, a user places an order. The order will go to a decision point where the order will be validated. If the order does not need validation the order will be merged and the stock will be allocated. When the order is for example higher than 10.000 dollars, an approval is needed. When the approval is finished, the order will be merged and the stock will be allocated and if the approval is declined the order will be ignored.

2.5 Related work

In this section, some research in this field of study will be shown and discussed. Namely, Event-driven applications/programming and Wireframe. This section is divided into two parts. The first part, Event-driven applications/programming, will be explained and other research in this field will be shown. The second part, Wireframe, will explain the main components behind Wireframes and other research in this field will be shown. The questions which will be answered to understand the relevance of this paper are: “why are Event-driven applications/programming and Wireframe used?”, “What are the benefits of using Event-driven applications/programming and Wireframe” “What are the drawbacks of using Event-driven applications/programming and Wireframe?”,

2.6 Event-driven applications

2.6.1 Usage of Event-driven applications

“It’s been predicted by Gartner that by 2020 a real-time, event-driven approach will be demanded for 80% of all digital applications used in business”[Chi], but what is an event-driven application? An event-driven application is an application that is driven by input and reacts in real-time to that input with an action. That input is caused by a user for example. The purpose of an event-driven application or using an event-driven User Interface is to ensure that the end-user has a smooth experience while using the application or the user interface[CJV15].

2.6.1.1 Relation to ngUML

In an event-driven program there are 2 components:

- The scheduler
- The event handler

The scheduler is the central point of the application that receives events and passes them down to the corresponding event handler[JM6]. At runtime, the scheduler remains active until it encounters

an end-event that terminates the application. Each event is identified by an ID of the object affected by the event. An example may be the name of a button the user clicks. Additionally, events are generated not only by a user during the execution of a program, but also by messages from the operating system or from an interruption by a device or by system hardware. When the events are generated and received by a scheduler, the scheduler checks the event for the type of the event for example. After it is checked, the scheduler calls for the event handler to deal with it. The handler takes the necessary actions in the form of a visual response or changing the system's state. An event handler can trigger another event that will cause a second handler to deal with the event. A global view of how an event-driven program works is shown in Figure 8.

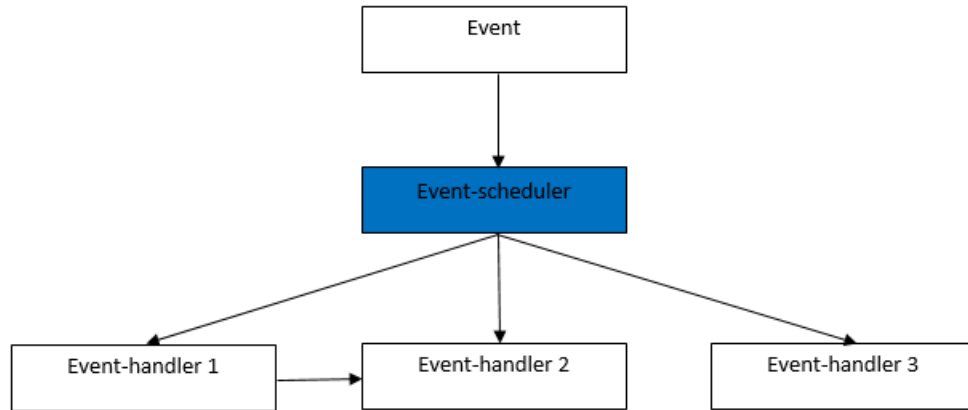


Figure 8: How a simple event-driven program works

The approach for the ngUML software is also based on event-driven applications where the software is able to interpret process specifications at runtime in order to add pages which are part of an application or workflow. As a result, pages must be shown at runtime for each node in an activity diagram. In short, in an event-driven application, there is a main loop that waits for events. When an event is detected, a call-back function (event handler) is triggered. As a result, the flow of the application is determined by actions/events like mouse clicks or signals from other programs.

2.6.1.2 Rationale behind Event-driven applications

As quoted above by Raspal Chima, Gartner predicted that by 2020, 80% of all applications which are used in business are required to be event-driven. Provost et al. [ATL16] has done research on why an event-driven approach is demanded for businesses in this era. Industry 4.0 is characterized by the combination and integration of systems and new technologies and one of those new technologies is the Internet of Things (IoT). The Internet of Things is actually simply a name for the ever-increasing integration of various digital systems on the internet and the connection to exchange data between these systems. As quoted in the journal by Lan et al. [LLS14]: “*IoT (Internet of Things) bridges the physical world and information space. IoT services are environment sensitive and event-driven. The new IoT service architecture should adapt to these features.*”

A business with an event-driven approach in architecture is more flexible. As such, tougher and constantly changing market demands can be withstood. To move along as a company with regard to Industry 4.0, businesses are required to be flexible when integrating new systems and managing big data. As described in the journal from Provost et al. [ATL16], an event-driven architecture enables a firm to integrate new systems and manage big data.

The journal by Provost et al. presents The Line Information System Architecture (LISA). The intent of LISA is to use LISA to tackle the tougher and constantly changing market demands. Moreover, LISA is described by the authors as “an event-driven architecture featuring loose coupling, a prototype oriented information model and formalised transformation services. LISA is designed to enable flexible factory integration and data utilisation. The focus of LISA is on integration of devices and services on all levels, simplifying hardware changes and integration of new smart services as well as supporting continuous improvements on information visualisation and control” [ATL16].

Another reason why event-driven applications are used is described in the paper by Dabek et al. [FDM02]. Namely, an event-driven program is more robust than a thread program. Events are a better means of managing I/O concurrency in server software than threads. Meaning, an event-driven application can run multiple events sequentially while thread programs cannot. This is because events help avoid bugs that occur by the unnecessary CPU concurrency. Furthermore, under heavy load event-driven programs perform more stable than threaded programs.

Also, the Web is constantly changing. As described in the paper of Yuan et al., the Web is becoming a medium for providing a wide array of e-commerce and other information, where services are based solely on distributed software interactions for collaborative work [YL09]. Event-driven architecture is the solution for building such information systems based on software interactions due to their characteristics of modularity, loose couplings, and flexibility.

On the other hand, there are drawbacks to event-driven architecture. An example of this is described in the paper by McClurg et al. [JM6]. In the paper, the authors mention a few drawbacks when using an event-driven architecture. When applications using an event-driven approach in the architecture need a maintenance break, it may be difficult to keep the application running as before the update was implemented. Due to the high degree of concurrency in networks, implementing updates properly is difficult.

Furthermore, a drawback of Event-driven application is the adaptability of Event-driven architecture [YL09]. The paper of Yuan et al. presents a new Event-driven architecture of adaptive integration of Event-driven architecture and discussed why the old Event-driven architecture is no longer up to date. The main reason is the dynamic behavior of the users. Given the rapid changes in the web environment, Event-driven architecture is not able to empower those changes and adapt to those behaviours.

2.6.2 Wireframe

A Wireframe is a visual tool to develop a website or an application. It is like a construction drawing, where an overview is given of the different parts that will be present on a website. With a Wireframe,

the structure of a website and how pages are connected to each other are illustrated. The main components of a Wireframe are usability and user-friendliness. Namely, the following elements are key aspects of Wireframes:

- Structure: coherence between different parts.
- Layout: arrangement between parts.
- Navigation: deals with the menu and links. Meaning, how the menu on the website should look like and what each link should refer to.
- Functionality: elements that determine interaction, ease of use and efficiency of the website.

2.6.2.1 Usage of Wireframes

First, previous research has determined the usage of wireframe. In the first paper presented by Roth et al., the authors describe the potential of wireframe by creating a prototype [RHMQ17]. In order to test the contribution of wireframe to the end product, a wireframe was introduced for a project (the National Oceanic Atmospheric administration's (NOAA) Lake Level Viewer) to improve the user experience. Eighteen participants were selected to use the wireframes. At the end of the participation, for each participant their recommendations were asked for the end product and those recommendations were applied to the end product. As a result, the usage of wireframes leads to an improvement in the end product, which would be unattainable had the experiment never been partaken.

The second paper discussing the usage of wireframe described that wireframe are useful when creating prototypes [SRMSCV13]. Especially in the start phase where a team is united to design and create the first prototype, wireframe are of particular interest. As a result of using Wireframe, developers are able to spend less time and money creating several prototypes when a high proportion of prototypes is discarded. Furthermore, the usage of wireframe saves time with regards to understanding what a prototype should look like. Gutierrez et al. proposed a solution for the common high costs and long development times during the development phase[GLS21]. The solution is the implementation of an algorithm that generates static web pages from hand-drawn drawings. The usage of the wireframe algorithm reduces the cost and time of developing web pages by automating the process of creating Html and CSS code.

The third paper extends the second paper which discussed the usage of wireframe [Bra]. Namely, the paper by Brajnik states that wireframe helps with accessibility assessments. In the conceptual design phase, wireframe carried out usability and accessibility investigations. Because of the wireframe, the developers get a better mapping of the requirements. This leads to a quicker transition from the abstract level of design to a concrete level of design.

The fourth paper covering the why-question reviews the relevance of using visual representations to communicate design ideas between different stakeholders [BKLHI01]. In this paper, Bryan-Kinns et al. studied large multinational companies that are specialised in creating, designing and building e-commerce websites. To discover the importance of visual representations, a range of visual representations was shown to participants. The different forms of visual representations used were

hand-drawn sketches, wireframes, prototypes, digital sketches and other digital prototypes. In order to obtain data, data was collected through interviews and surveys. Bryan-Kinns et al. found that, in order to create, design and build potential e-commerce websites, several different components working together are needed to develop websites. For example, usability engineering, graphic design, coding and management consulting are key factors. As such, communication between components is necessary to discover early problems in the process of designing an e-commerce website and make it easier and cheaper to correct mistakes and make changes. In order to achieve that, visual representations are necessary. They show the navigation structure and the content of the designing process.

2.6.2.2 Drawbacks of wireframes

Yet, there are drawbacks when using wireframes. A study done by Sutipitakwong et al. describes their findings on the analysis of the effectiveness of team communication [SJ20]. In the analysis, two wireframes were used. That is, a tangible wireframe and a digital wireframe were used in order to test the team communication through three main aspects: Design, Features, and Layout. The outcome of the analysis showed that tangible Wireframes are easy to work with which leads to a clearer view of a product's design. The effectiveness of team communication not only depends on the kind of Wireframe that is used, but also depends on the user experience of team members, understanding of goals and the intermediate achievements and the ability to select the right tools that are suitable for the Wireframes to achieve those goals and intermediate achievements. The analysis concluded was that most companies procure the newest Wireframe to improve teamwork, but the quality of the team communication not only depends on the kind of Wireframe that is used, but depends on many factors.

3 System Design

This section describes the design of the system created to enable a dynamic user interface communicating with a backend. This section is divided into two subjects. First, the logical design will be reviewed. Secondly, the technical design will be discussed. In the section of logical design, an analysis and an explanation will be given for the design of the system. In the technical design section, a high-level overview explanation of the system will be given.

3.1 Logical Design

3.1.1 Requirements

For this project, the purpose is to extend the User Interface to make it possible to enable workflow and application flow between pages. This requires extending the prototype with controls and making it communicate with background execution. The project consists of a frontend (the ngUML class modeler) and a backend (the ngUML Django backend). The language of the frontend is Typescript and the framework that is used is REACT. The backend is made with the language Python and the framework that is used is Django.

To achieve the goal to extend the prototype with controls and make it communicate with the backend, the frontend needs to deliver the required data and code to make it possible for the backend to do automatic execution at runtime for processes. Thus, the following requirements must be met:

- For every activity node that is created in the frontend it has to be stored in the backend, but also the data of the activity node must be stored in the backend.
- For every activity node, it needs to be linked with a page to be able to execute the prototype in the backend. As a result, the backend must know in which order to show the pages.

3.1.2 Overview

View the activity diagram given in Figure 2. A user manually creates the activity diagram from Figure 2 in the frontend and gives each of those nodes a pagename, except the initial node. For example, the Enter Order node gets the pagename “EnterOrder”, the decision node gets the pagename “Decision”, the approve order node gets the pagename “Approve” and the allocate stock node gets the pagename “Allocate stock”. When the user is done with providing each node with a pagename. The user goes to the backend to create manually pages. Depending on the number of pagename the user assign, the user will create four pages for this example. The user assigns page 1 to the Enter Order node, page 2 to the decision node, page 3 to the approve order node and page 4 to the allocate stock node. After each activity node is assigned to a page, the backend is provided with the necessary data to execute the prototype.

3.2 Technical Design

To make it possible to enable workflow between pages based on an activity diagram one major thing needs to be accomplished. That is, the connection between an activity node and a page (that can be created in the backend) must be added. The activity diagram will be made in the frontend, so when a node is made by a user the user has the ability to give the node that will be created a pagename. So, the following implementation needs to be done:

- The frontend has the functionality to give a node a pagename.
- The frontend needs to store the user's input and send it to the backend.
- A feature needs to be implemented where the user can connect the nodes of an activity diagram with a page.

Firstly, to create a function where the user can place a page name for a node that is made, an input box is created in the ActivityMenu. The input box is made with the package `blueprintjs/core`. Secondly, the input box takes the user input and gives that input to the property "pagename". The property "pagename" is created by adding the pagename property in the object (INode) and passing that property to the function `addnode`. Thirdly, this function ensures that the data is stored for each node that is created as a copy of the node.

When the data is stored for each node as a copy. The `WriteData` function will send the data to the backend and in the backend the data of the node will then be stored in the database.

Lastly, to link the node with the page that is created in the backend the function `page.link` is made to connect the activity node with the page in the backend. The function checks whether the request it obtained from the `Html` was `get` or `post`. If the request was `get`, the function `page.link` shows the corresponding page. Namely, a web page with a drop-down button where the user can choose from the following pagename, as seen in Figure 22 and Figure 23 is shown. If the request was `post`, the user has clicked the save button. As a result, the id of the pagename connected to the activity node will be linked with the id of the page, where the page currently is located. The linking of the ids will be then visible in the database. To make the functionality to store the two connected ids, a `ManyToMany` relationship was added to the tables of `Page` and `ActivityNode`. That means that `Page` can have multiple `ActivityNode` objects and `ActivityNode` can have multiple `Page` objects. Meaning, that multiple pages that are created in the backend can be connected with multiple activity nodes and multiple activity nodes can be linked with multiple pages.

4 Worked Examples

In this section, two worked examples will be shown to illustrate how the system works. First, a worked example of Bram van Aggelen of the creation of a page will be shown. Second, an illustration of the workflow will be illustrated.

4.1 Page creation

To start the worked example of the creation of pages, the user starts by defining a requirement text. When the requirement text is defined, the user enters the requirement text into the backend metadata extractor (see Figure 3) and presses the “Show metadata” button. By pressing the “Show metadata” button, the backend uses NLP processing to start finding classes, their attributes and the relation between the classes. When the processing in the backend is finished, a class diagram is generated in the ngUML editor as illustrated in Figure 4. Next, some changes need to be made to the class diagram to let it work in the way it is supposed to. So the following changes need to be made:

- order – customer (1)
- order – deliverycompany (1)
- lineitem – product (1)

These changes need to be made because some relations that are modeled as many to many need to be modeled as many to one due to the lack of implementation of many to many relationships, where the (1) in the changes indicates the 1 in the relationships.

Next, go back to the backend and go to “Runnable prototype” and then to “Domain model”, where the classes and the relation between them can be viewed and edited as depicted in Figure 5. When everything is modified correctly, the other part will be used, namely “Application model”. In the “Application model” two applications will be created, called Client and Business (see Figure 9). “Client” refers to the client side and “Business” refers to the business side.

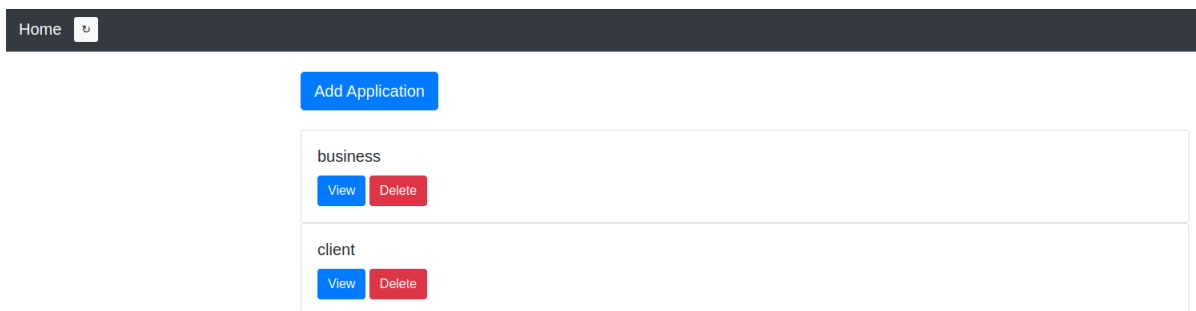


Figure 9: The Business and the Client applications

Following, for each application the corresponding classifiers will be added. In the Business application under “Category”, a new category is created, called “Creation”. In “Creation” two pages will be created, named Product and Delivery company (see Figure 10).

The screenshot shows a web application interface with a dark header bar containing 'Application home', 'Run', and a small icon. Below the header, there are two main sections. The first section is titled 'Product' and contains five input fields: 'description : string', 'location : string', 'name : string', 'price : string', and 'product_number : string'. Below these fields are two buttons: 'Link properties' (blue) and 'Unlink' (red). The second section is titled 'DeliveryCompany' and contains two input fields: 'address : string' and 'name : string'. Below these fields are two buttons: 'Link properties' (blue) and 'Unlink' (red). Below these two sections is a 'Categories' section. It has a title 'Categories' and an 'Add Category' button (blue). Below the title is a list of categories, with 'creation' selected. Below 'creation' are four buttons: 'View' (blue), 'Edit' (blue), 'Add Page' (blue), and 'Delete' (red).

Figure 10: Category with the pages Product and Delivery company

Next is setting up each page. That is possible in the query tab (see Figure 11). From there the main model can be chosen, the toggle for “page for data insert” can be checked and if wanted a user can specify a query to narrow down the results (see Figure 12).

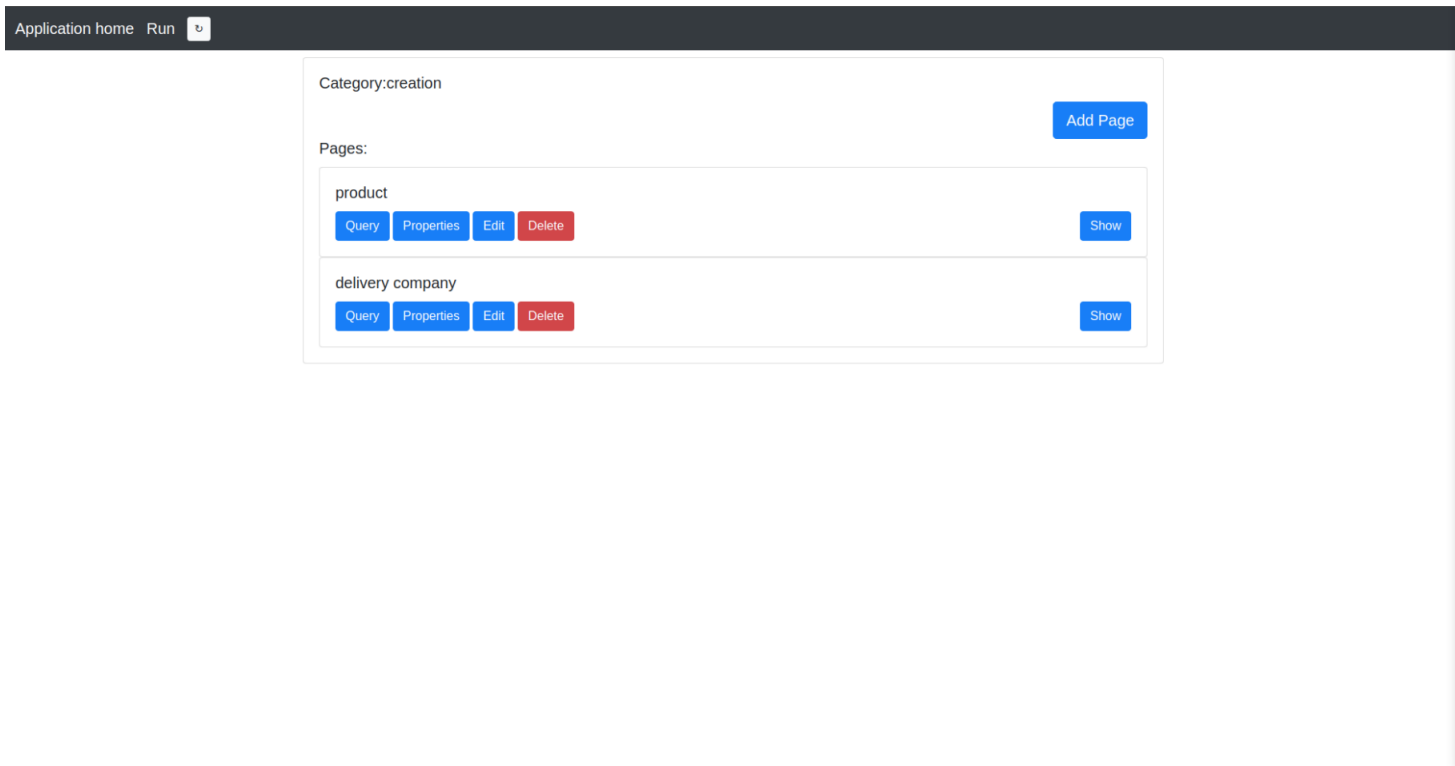


Figure 11: Category with the pages Product and Delivery company

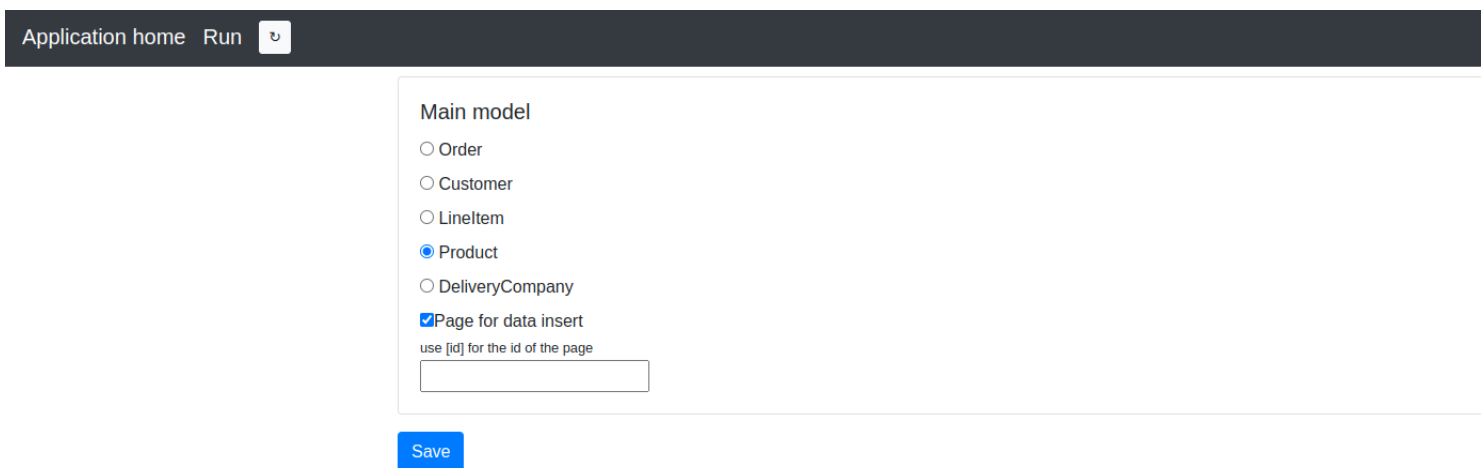
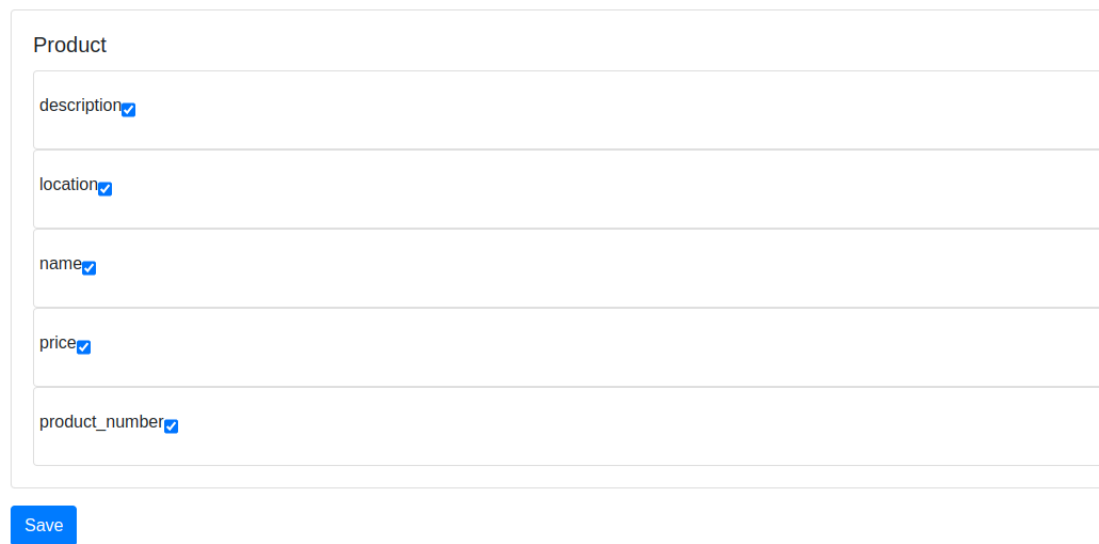




Figure 12: Page for data insert


Following, select “Product” and choose the page creation option. After that has happened, select all of the properties linked to the “Product” classifier as illustrated in Figure 13.





Product

description 

location 

name 

price 

product_number 

Save

Figure 13: Properties of “Product”

Next, the Html structure will be created through the tab “Edit” (see Figure 11). In “Edit” (see Figure 14) there is the ability to use features. The features are: insert data, insert data table, insert input field, insert input model and insert submit button.

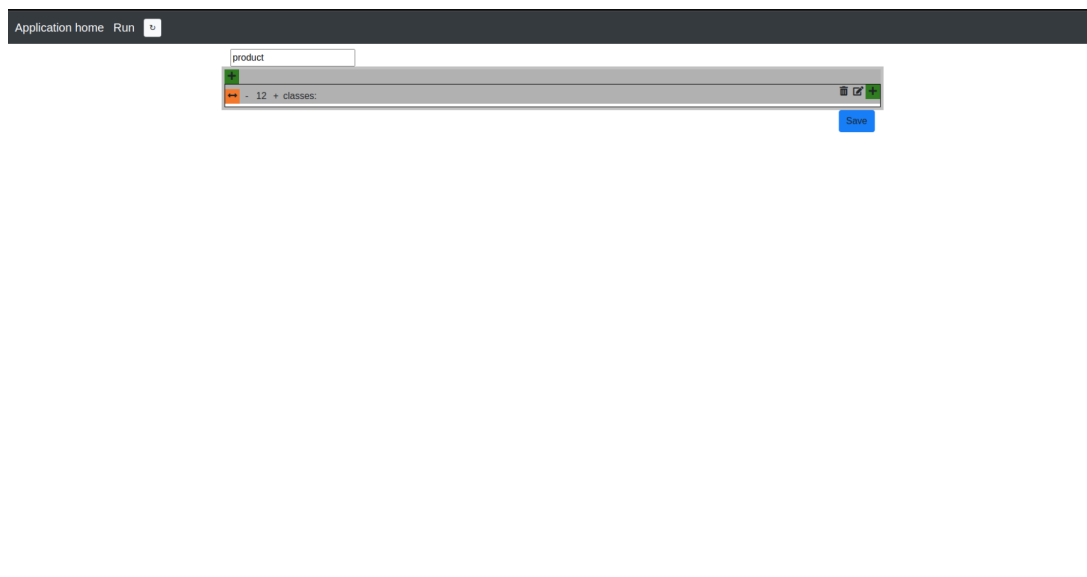


Figure 14: The custom Html editor

For this example, choose under the tab “insert”, “insert input” (see Figure 15). This will show a table where the data of “Product” can be entered, as illustrated in Figure 16.

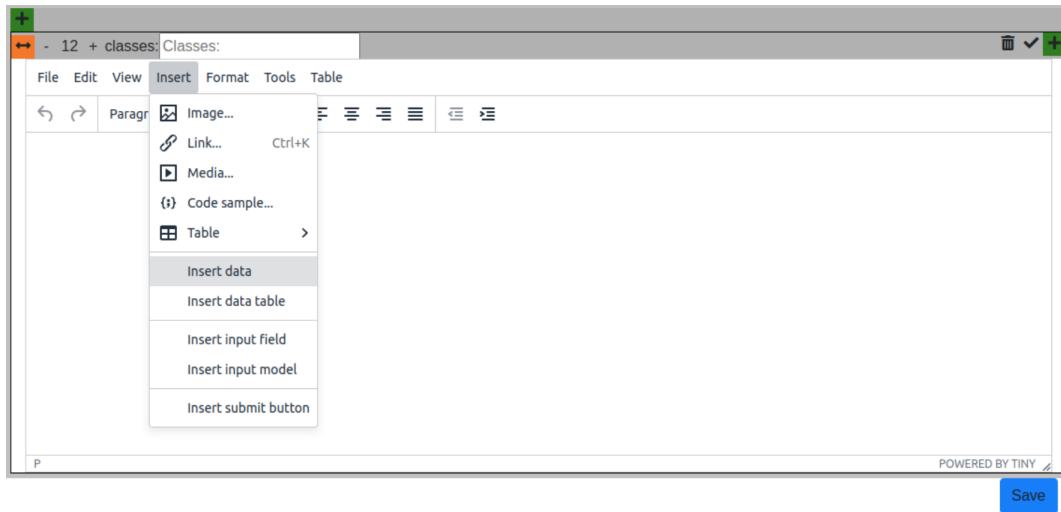


Figure 15: Inserting data in the Html page editor

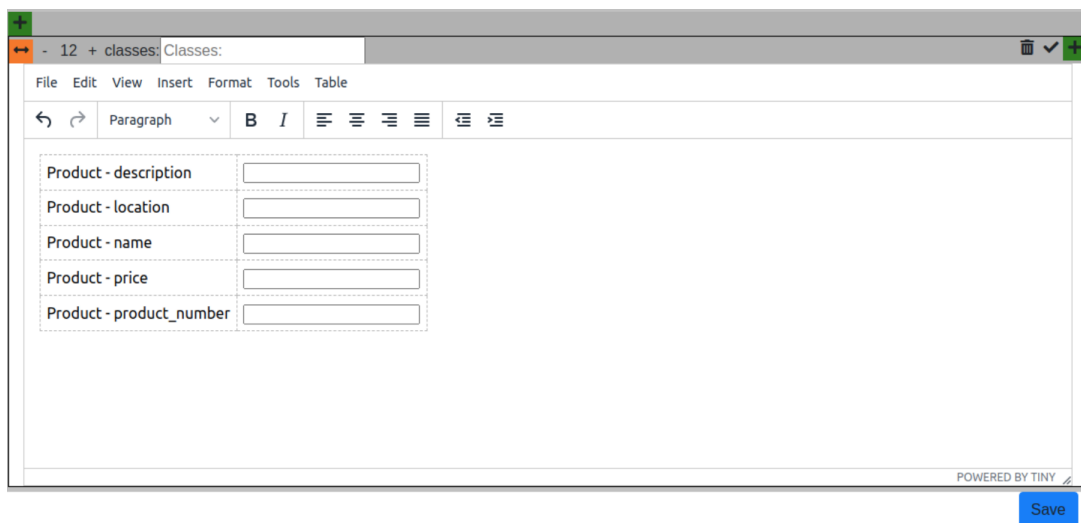


Figure 16: Table where the data of Product can be entered

After the data is entered, a save button will be added to save the page and this will result in the following page (see Figure 17). As seen in Figure 17 there is the ability to add more data with the button “Add more”.

The screenshot shows a web application interface. At the top, there is a dark header bar with the text "Application home" and "Run" next to a small icon. Below the header, on the left, there is a sidebar with a tree view containing the following items: "business", "-creation", "->product", and "->delivery company". The main content area displays a form with the following fields: "Product - description", "Product - location", "Product - name", "Product - price", and "Product - product_number". Each field has a corresponding input box. Below the "Product - product_number" field is a "save" button. To the right of the form, there is a green button labeled "Add more".

Figure 17: The page that is generated after the data is entered and saved with the save button

This process will also be done for the Delivery company afterward. Next, a new category will be created, named “overview” as seen in Figure 18.

The screenshot shows a web application interface titled "Categories". In the top right corner, there is a blue button labeled "Add Category". Below this, there is a list of categories. The first category is "creation", which has four buttons: "View", "Edit", "Add Page", and "Delete". The second category is "overview", which also has four buttons: "View", "Edit", "Add Page", and "Delete".

Figure 18: The creation of the category “Overview”

In “overview”, the page “order-overview” will be created. The main model of the “order-overview” page is “Order”, so the following properties will be selected:

- All of Order
- Name of Customer
- Quantity from LineItem
- Name, price and product_number from Product
- All of Delivery Company

This will generate the page as seen in Figure 19. Finally, the “Client” application will be opened and the Order, Customer, LineItem and Product classifiers were added. After that is done, the category “Order” will be created for “product” in Figure 10. Following, in the category “Order” a page called “create-order ” will be made. For the main model, choose “Order” and insert it as data page. After that link all the possible properties and the final page will be created as seen in Figure 10.

order-overview

- 12 + classes:

Delivery status	Order number	Entry date	Description	Customer name	Delivery company
-----------------	--------------	------------	-------------	---------------	------------------

- 12 + classes:

{{Order-delivery_status}}	{{Order-order_number}}	{{Order-entry_date}}	{{Order-description}}	{{Customer-name}}	{{DeliveryCompany-name}}
---------------------------	------------------------	----------------------	-----------------------	-------------------	--------------------------

- 6 + classes:

{{LineItem-quantity}}	{{Product-name}}	{{Product-product_number}}
-----------------------	------------------	----------------------------

Save

Figure 19: The creation of the page “order-overview”

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:

LineItem - quantity

Product ▾

- 12 + classes:

Save

Figure 20: The final page

4.2 workflow1

All of the steps of the workflow example will be shown and explained in this section. For these worked examples two applications are involved Namely, the ngUML editor and the ngUML Django backend. From here on ngUML editor will be referred to as frontend and the ngUML Django backend referred to as the backend. To start the workflow example, users start by creating an activity diagram. For example the activity diagram in Figure 21.

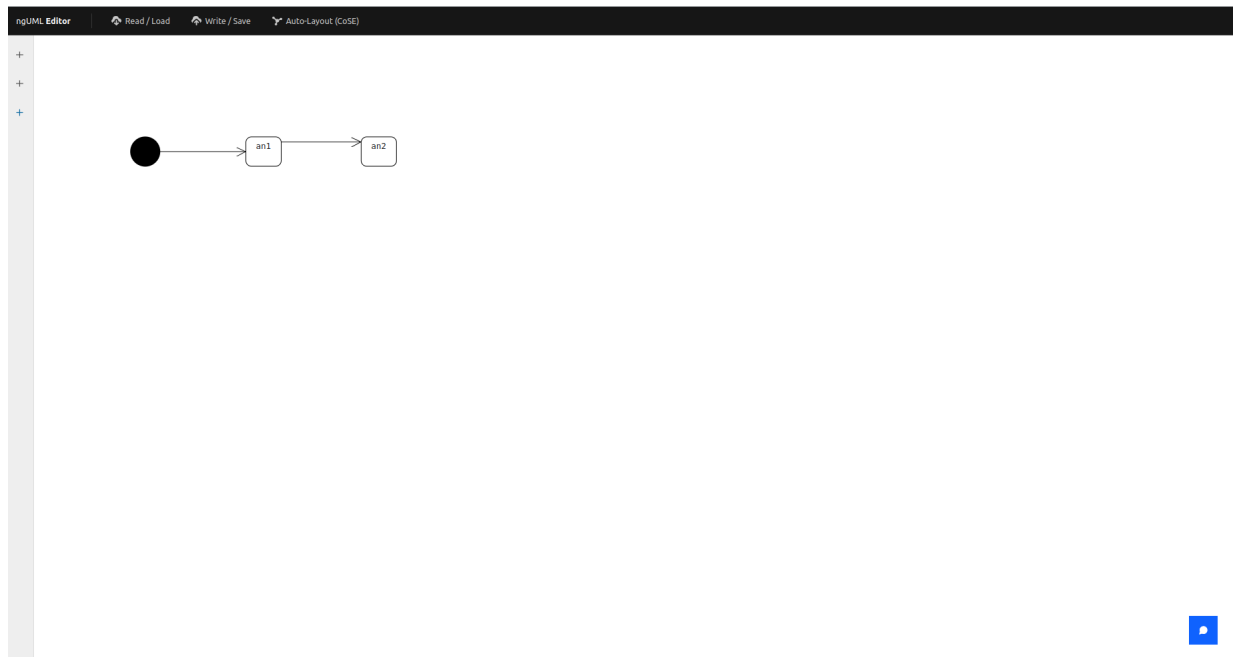


Figure 21: The activity diagram for the worked example

After the activity diagram is created, the user has the ability to assign for each node a page name (see Figure 22 and Figure 23).

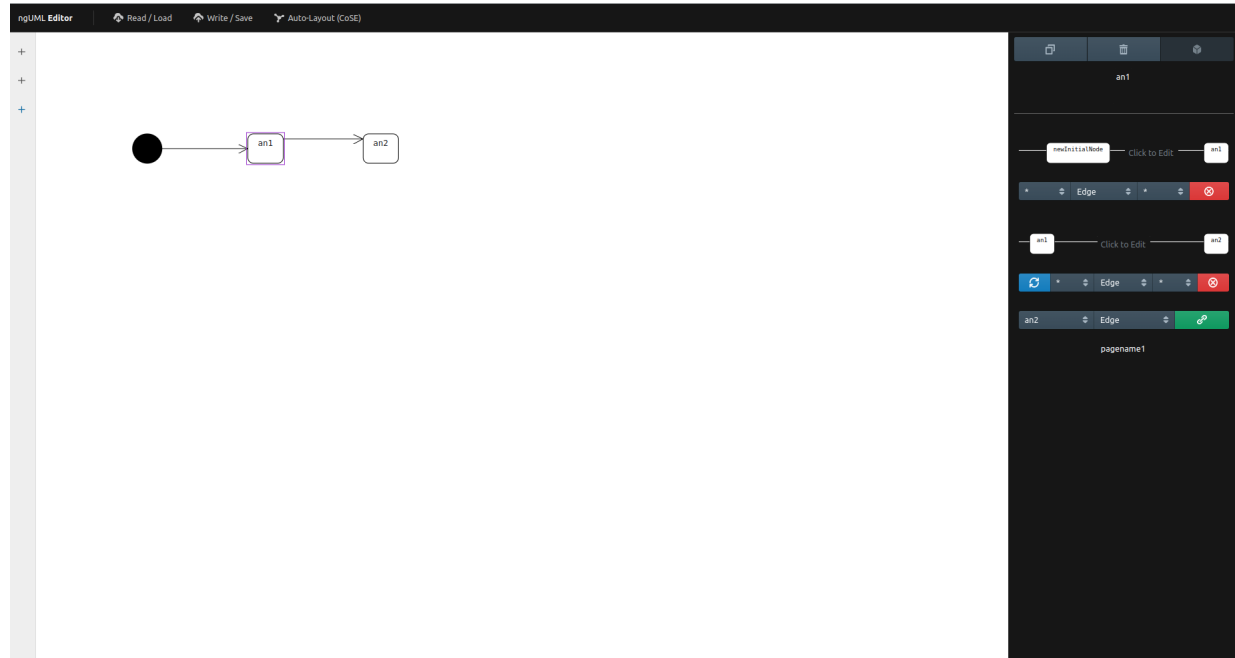


Figure 22: The menu that will pop up when the user clicked on node1 to assign a pagename

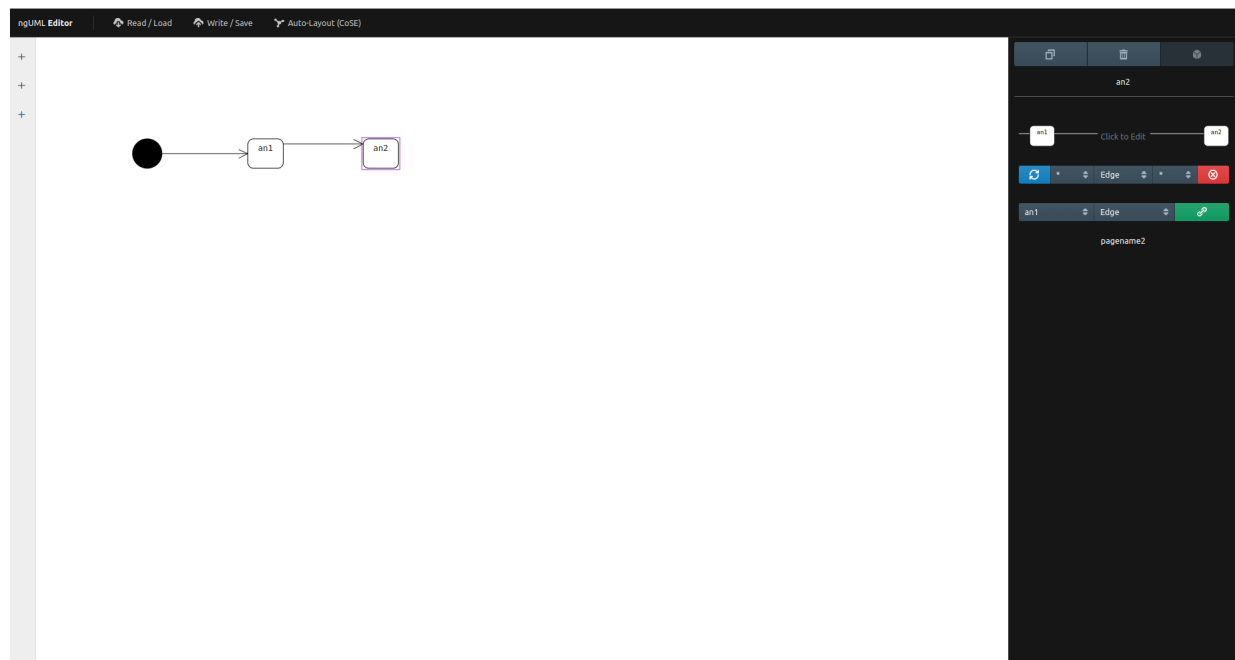


Figure 23: The menu that will pop up when the user clicked on node2 to assign a pagename

So, node1 gets the pagename “pagename1” and node2 gets the pagename “pagename2”, as illustrated in Figure 22 and Figure 23). When that is done, the user presses the “Push to Backend” button to push the node data to the backend. When all the data is sent, the user goes to the backend, then to “Runnable prototype” and to “Application model”. In the “Application model” an application will be created, called “OrderApplication” (see Figure 24).

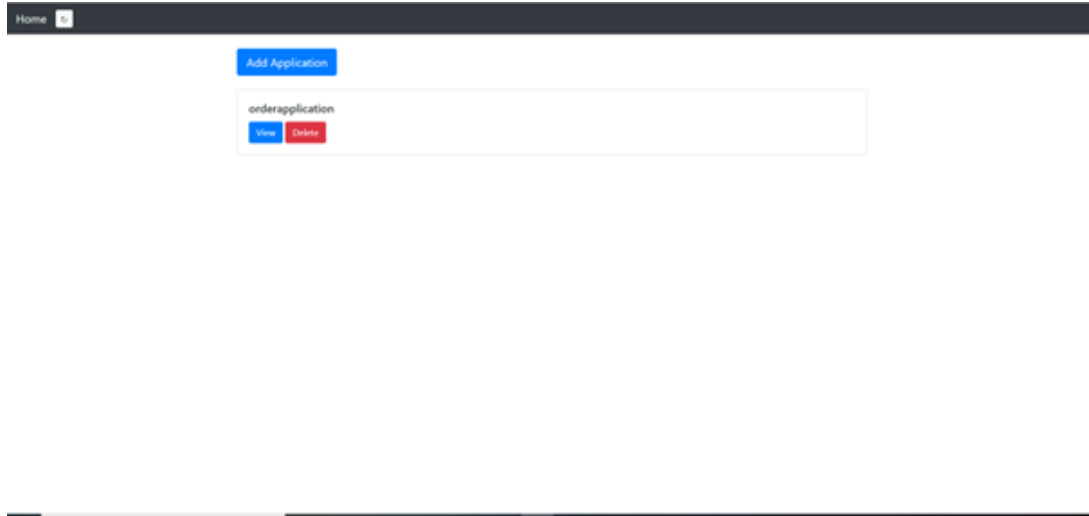


Figure 24: The application for the activity diagram

Following, in the application under “Category”, a new category is created called “cat1”(see Figure 25). In “cat1” two pages are created, namely page1 and page2, as illustrated in Figure 26.

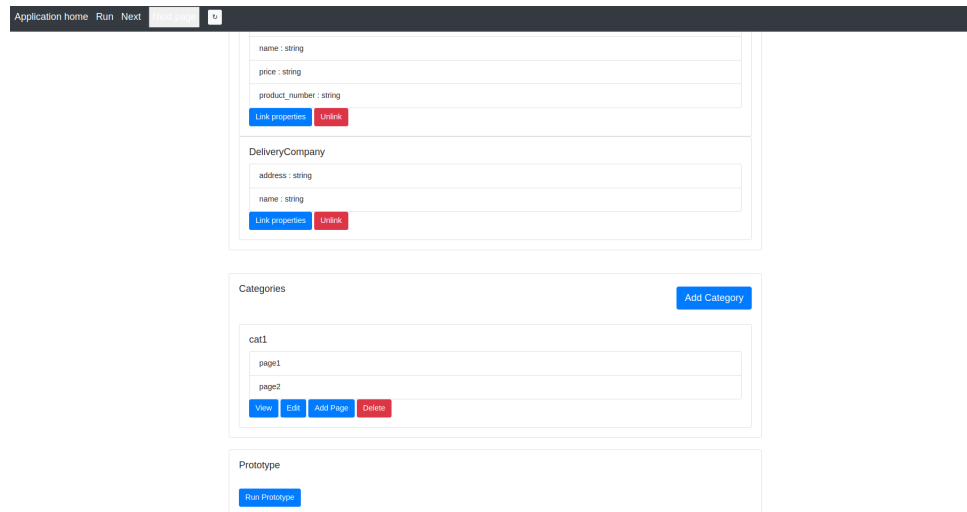


Figure 25: The category for the activity diagram

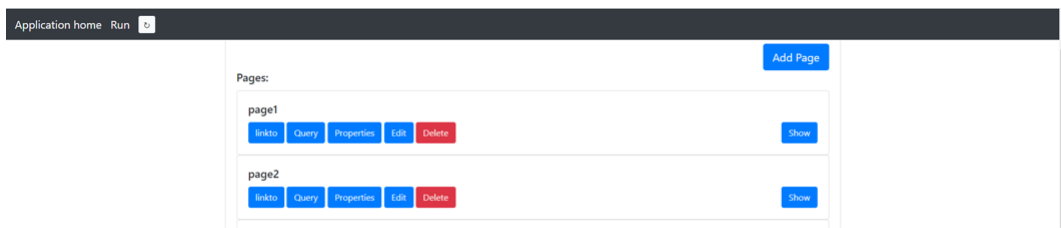


Figure 26: The 2 pages that are created in “cat1”

Next, by pressing the button “link to” a web page will be generated where the user can link the page with a pagename, where the pagename is connected with a node from the frontend. Page1 and page2 were linked to pagename1 and pagename2 as illustrated in Figure 27 and Figure 28).

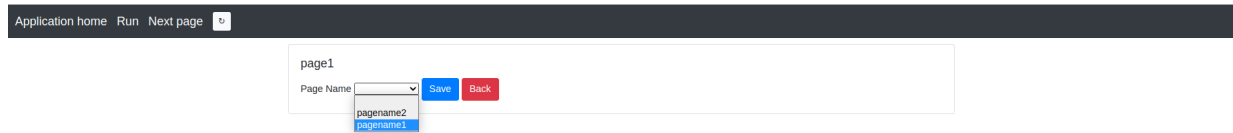


Figure 27: Page1 linked with pagename1

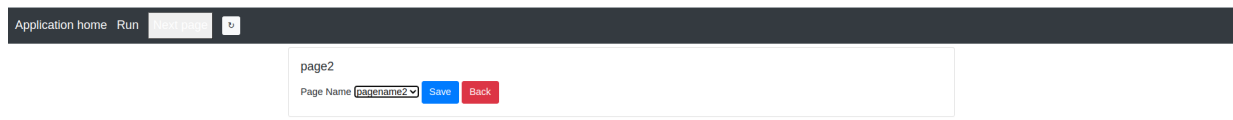


Figure 28: Page2 linked with pagename2

When all the preparations have been made, the backend is ready to run the workflow. In order to run the prototype, the “run prototype” button (see Figure 25) must be clicked. This leads the user to the following page, as seen in Figure 30.

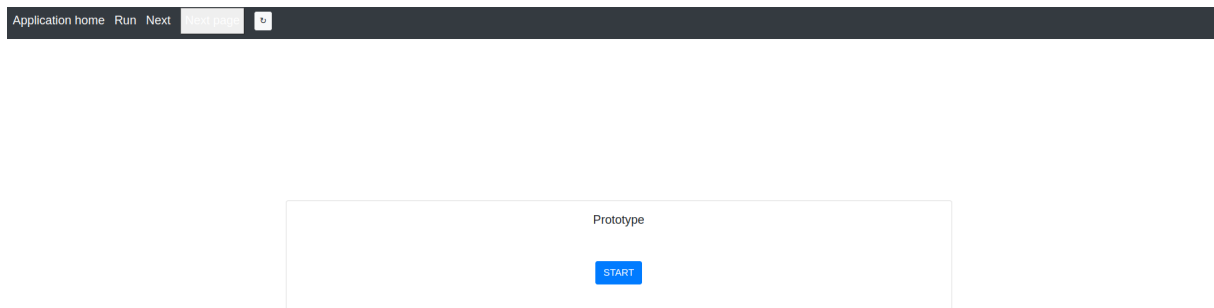


Figure 29: Page to start the prototype execution

After the “run” button is pressed, the first page will be shown. That page is linked to the first action node after the initial node. Since action node1 was linked to pagename1 and pagename1 to page 1, page 1 will be shown for action node1 (Figure 29). When the next button is clicked, the page of next action node that is connected with the previous action node will be displayed, so in this case action node2. Since both the pages for action node1 and 2 are empty, the pages will look like in Figure 29. But the id of the page is still different for both nodes.

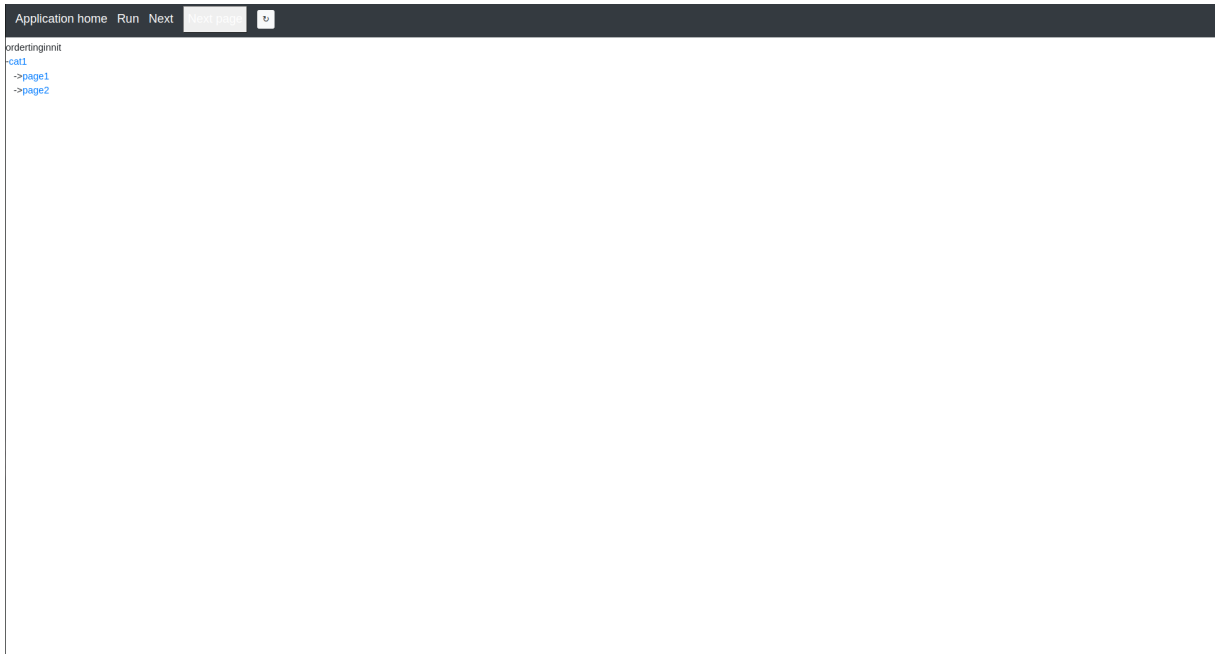


Figure 30: Page that is generated for action node1, this page will also be shown for action node2, because the pages are empty

5 Conclusions and Further Research

This thesis is part of ngUML at Leiden Institute of Advanced Computer Science and presented an approach to automatically perform prototypes at runtime by adding behavioural aspects to static pages in a UML interpreter. This project is built on a current existing project by the ngUML team and was produced in collaboration with Spruijt's work focused on the automatic execution of the flow of pages for an activity diagram.

This paper adds to the previous research about wireframe and event-driven application by creating a runtime prototype executor from metadata obtained by user requirements. This is achieved by extending frontend functionalities. In the frontend, an input box is implemented where the user can give a node a pagename. When all the created nodes are assigned a pagename, the data will be sent to the backend. When the data is sent, the user can connect the page in the backend to the created pagename with a dropdown button. In this button, the user can choose from all pagenames where each pagename comes from all activity nodes in an activity diagram.

Nevertheless, the prototype execution is not ready for real-life or business use. Namely, the currently created feature for the frontend to assign a pagename for a node is only possible for the Action Node and Decision Node. As such, future research should focus on expanding this feature for all other nodes. Another limitation of this project is that in the frontend, that it is not possible to create more than one activity diagram. Meaning, it is not possible to use two initial nodes in the same editor. It is possible to place two initial nodes, but the two initial nodes will not be stored in the backend. So, future research should focus on making it possible to use more initial nodes in the same editor, so more advanced activity diagrams can be made.

Furthermore, future research should focus on making the pagenames editable. For example, when the user has entered a pagename in the input box and pressed enter to submit the pagename, the pagename can no longer be changed. Another problem with the current software for the ngUML editor is that it is only able to handle linear processes. For example, if an activity diagram consists of a parallel process it is still possible to give the nodes a pagename, but the processing of the data of those nodes will not be possible. In the backend, those nodes are stored with an incorrect name and pagename. Future research should extend the current choice of activity process and make it possible to store those nodes as they should be in the database of the backend. The last limitation is the lack of response when an activity diagram is incomplete. For example, currently, it is possible in the frontend to make an activity diagram and push it to the backend, without an initial node or an end activity node. Future research should return a notification to warn the user that the created activity diagram is incomplete.

References

- [ATL16] J. Provost M. Lieder C. Johnsson T. Lundholm A. Theorin, K. Bengtsson and B. Lennartson. An event-driven manufacturing information system architecture for industry 4.0. *International Journal of Production Research*, 55:1297–1311, 2016.
- [BKLHI01] Nick Bryan-Kinns, Magnus Lif, Fraser Hamilton, and Ismail Ismail. *Prototypes in Web-Site Design — Representations with Political Agenda*, pages 92–105. Springer London, London, 2001.
- [Bra] Giorgio Brajnik. Is the uml appropriate for interaction design?
- [Chi] Raspal Chima. Event-driven applications in software development.
- [CJV15] V. Raychev D. Dimitrov C.S. Jensen, A. Møller and M. Vechev. Stateless model checking of event-driven applications. *ACM SIGPLAN Notices*, 50:57–73, 2015.
- [FDM02] F. Kaashoek D. Mazières F. Dabek, N. Zeldovich and R. Morris. Event-driven programming for robust software. *Proceedings of the 10th workshop on ACM SIGOPS European workshop: beyond the PC - EW10*, 2002.
- [GLS21] Cesar Enrique Estrada Gutierrez, Rodrigo D. Lara, and Daniel Subauste. Cloud application for the generation of static websites through the recognition of wireframes using artificial intelligence. 2021.
- [IBMa] IBM. Control nodes in activity diagrams.
- [IBMb] IBM. Uml activity diagrams.
- [JM6] N. Foster J. McClurg, H. Hojjat and P. Černý. Event-driven network programming. *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, page 369–385, 2016.
- [LLS14] B. Wang L. Zhang L. Lan, F. Li and R. Shi. An event-driven service-oriented architecture for the internet of things. *2014 Asia-Pacific Services Computing Conference*, 2014.
- [Luc] Lucidchart. Why use a uml diagram?
- [RHMQ17] Robert E. Roth, David Hart, Rashauna Mead, and Chloë Quinn. Wireframing for interactive & web-based geographic visualization: designing the noaa lake level viewer. *Cartography and Geographic Information Science*, 44(4):338–357, 2017.
- [SJ20] Sutipong Sutipitakwong and Pornsuree Jamsri. Pros and cons of tangible and digital wireframes. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2020.
- [SRMSCV13] Óscar Sánchez Ramón, Jesús Molina, Jesús Sánchez Cuadrado, and Jean Vanderdonckt. Gui generation from wireframes. 09 2013.

- [YL09] Soe-Tsyr Yuan and Mei-Rung Lu. An value-centric event driven model and architecture: A case study of adaptive complement of soa for distributed care service delivery. *Expert Systems with Applications*, 36(2, Part 2):3671–3694, 2009.