

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**

# **Low-Code Data Model Designer for Manufacturing Execution Systems**

**Ana Isabel Ferreira Maia**



Mestrado em Engenharia Informática e Computação

Supervisor: João Carlos Pascoal Faria

July 30, 2022

# **Low-Code Data Model Designer for Manufacturing Execution Systems**

**Ana Isabel Ferreira Maia**

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: A. Miguel Monteiro

External Examiner: Miguel Goulão

Supervisor: João Pascoal Faria

July 30, 2022

# Abstract

A Manufacturing Execution System (MES) is an online integrated computerized system that provides the methods and tools used to accomplish production. They can integrate more accurate and current information into the decision-making process that helps manufacturing decision-makers optimize conditions on the plant floor to improve production output. It also works in real-time to enable the control of multiple elements of the production process.

Critical Manufacturing (CM) MES is a leading Manufacturing Execution System that also acts as an application development framework that allows partners and customers to implement their applications and logic for specific business scenarios.

CM is now in the process of evolving its MES framework to allow the creation of low-code solutions. An essential part of low-code development platforms is defining a data model for each solution.

Currently, several technologies and platforms offer visual data model designers. However, they are oriented to creating a specific data model from scratch. This project aims to create a visual designer integrated into Critical Manufacturing Framework (CMF) that enables the user to extend a preexisting data model by either adding properties and attributes to existing business entities or creating entirely new entities from scratch.

To achieve that goal, the main requirements for a low-code data model designer and the state of the art in data modeling were established, followed by the implementation of a proof-of-concept adapted to the context of the previously mentioned CM framework.

The resulting visual designer is integrated into the CMF and was developed using JointJS as the framework for displaying the data models. The page consists of a canvas where the data model will be drawn, an action bar, and a side panel with the main details of the selected entity type node. The canvas will, by default, display the node of the currently selected main entity type and its direct relations. The data model can be expanded by showing the immediate relations of the entities depicted on the canvas. The page allows the manipulation of the current data model being viewed, such as refreshing it, applying an automatic layout, saving and resetting the current view, adding a new entity type node, etc. In terms of edition, it allows creating and editing an entity type, adding/removing properties, custom properties, and attributes using wizards, and visually creating a relation between two entities.

The validation process showed that the desired requirements were met. The usability study demonstrated that the end-user of this system, as in the deployment services team at Critical Manufacturing, gave an average review of the system, along with some crucial suggestions for future work.

**Keywords:** Manufacturing Execution Systems, Low-Code, Data Model Designer

# Resumo

Um *Manufacturing Execution System* (MES) é um sistema computadorizado *online* que fornece métodos e ferramentas usados para a produção. Estes sistemas podem integrar informações mais precisas e atuais no processo de tomada de decisão que ajuda gestores de produção a otimizar as condições no chão de fábrica para melhorar a produção. Também funciona em tempo real para permitir o controle de vários elementos do processo de produção.

O Critical Manufacturing (CM) MES é um *Manufacturing Execution System* líder de mercado que atua também como uma *framework* que permite a parceiros e clientes a implementação da lógica de negócio de modo a criar soluções personalizadas.

A CM encontra-se no processo de evolução da sua *framework* do MES de modo a permitir a criação de soluções *low-code*. Um fator essencial de plataformas *low-code* é a definição de modelos de dados para cada solução.

Atualmente, várias tecnologias e plataformas oferecem designers de modelos de dados visuais. No entanto, estes sistemas focam-se na criação de modelos de dados de raiz. Este projeto visa criar um designer visual de modelos de dados integrado na Critical Manufacturing Framework (CMF) que permite ao utilizador aumentar um modelo de dados preexistente, adicionando propriedades e atributos a entidades de negócio existentes ou criando entidades de raiz.

Para atingir este objetivo, foram estabelecidos os principais requisitos para um designer de modelos de dados *low-code* e foi apresentado o estado da arte em modelação de dados, seguido da implementação de uma prova de conceito adaptada ao contexto da CMF.

A solução proposta nesta dissertação está integrada no CMF e foi desenvolvida usando o Join-tJS como *framework* para visualizar o modelo de dados. A página consiste numa tela onde será desenhado o modelo de dados, uma *action bar* e um painel lateral com os principais detalhes da entidade associada ao nó selecionado. Ao selecionar uma entidade principal a ser visualizada, por defeito, a tela irá mostrar o nó dessa entidade e das entidades relacionadas com esta. O modelo de dados pode ser expandido de modo a mostrar as relações imediatas das entidades representadas na tela. A página permite a manipulação do modelo de dados a ser visualizado das seguintes formas: atualização, aplicar um *layout* automaticamente, guardar e restabelecer a vista atual, adicionar um novo nó representativo de uma entidade à tela, etc. No que toca à edição, o sistema permite criar e editar uma entidade, adicionar e remover propriedades, propriedades personalizadas e atributos recorrendo a *wizards* e criar visualmente uma relação entre duas entidades.

O processo de validação mostrou que os requisitos desejados foram cumpridos. O estudo de usabilidade demonstrou que o utilizador final deste sistema, o departamento de *deployment services* da CM, deu uma pontuação média ao sistema, juntamente com sugestões cruciais para trabalho futuro.

**Keywords:** Manufacturing Execution Systems, Low-Code, Designer de modelos de dados

# Acknowledgments

I want to thank my Supervisor, Prof. João Pascoal Faria, for all the help, guidance, and support throughout this dissertation process. I would also like to thank Critical Manufacturing for providing a good work environment and my co-supervisor, Micael Queiroz, for his continuous support and valuable advice throughout the development process of this dissertation.

Also, I would like to express my deepest gratitude to my family, friends, and Daniel, who supported me over the years.

Ana Maia

*“We are what we repeatedly do.  
Excellence, then, is not an act, but a habit.”*

Aristotle

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context and Motivation . . . . .	1
1.2	Objectives . . . . .	2
1.3	Document Structure . . . . .	2
<b>2</b>	<b>Problem Analysis</b>	<b>3</b>
2.1	Critical Manufacturing MES . . . . .	3
2.2	Low-Code Features of Critical Manufacturing Framework . . . . .	5
2.3	Data Modeling Status and Needs . . . . .	6
<b>3</b>	<b>State of the Art</b>	<b>9</b>
3.1	Data Models . . . . .	9
3.2	Low-Code Application Development . . . . .	10
3.2.1	Architecture . . . . .	11
3.2.2	Data Modeling in Low-Code . . . . .	11
3.3	Data Model Customization . . . . .	17
3.3.1	Oracle Retail . . . . .	18
3.3.2	SAP PowerDesigner . . . . .	18
3.4	Summary . . . . .	21
<b>4</b>	<b>Proposed Solution</b>	<b>22</b>
4.1	Solution Requirements . . . . .	22
4.2	Architecture and Technologies . . . . .	24
4.3	UI Structure . . . . .	26
4.4	User Interface and Interaction Design . . . . .	28
4.4.1	Data Model Visualization . . . . .	29
4.4.2	Data Model Edition . . . . .	32
4.5	Implementation Challenges and Constraints . . . . .	35
<b>5</b>	<b>Validation</b>	<b>37</b>
5.1	Validation Testing . . . . .	37
5.2	Usability Study . . . . .	41
5.2.1	Objectives and Methodology . . . . .	41
5.2.2	Results and Discussion . . . . .	43
<b>6</b>	<b>Conclusions and Future Work</b>	<b>47</b>
6.1	Conclusions . . . . .	47
6.2	Future Work . . . . .	47

*CONTENTS*

vi

**References**

**49**

**A Appendix**

**52**



# List of Figures

2.1	Magic Quadrant for Manufacturing Execution Systems [24]	4
2.2	Architecture of Critical Manufacturing Framework	5
2.3	Dashboard with pre-defined widgets	6
2.4	Data Connection between widgets	6
2.5	Foreign key "Calendar" in entity type "Plan" page	7
3.1	Modeling notation [31]	10
3.2	Layered architecture of LCDP [26]	11
3.3	Example of OutSystems Data Modeling tool with one-to-one and one-to-many relationships [14]	12
3.4	Example of Entities in Module Tree (OutSystems) [7]	13
3.5	Form to create and edit entity (OutSystems) [13]	13
3.6	Entity diagram of one-to-many association (OutSystems) [7]	14
3.7	On delete rule (OutSystems) [7]	14
3.8	Example of a one-to-many relationship where Order is the Owner (Mendix) [2]	15
3.9	Menu Bar (Mendix) [4]	15
3.10	Edit Attribute dialog box (Mendix) [3]	15
3.11	Edit Association Properties dialog box (Mendix) [1]	16
3.12	Edit Entity Properties dialog box (Mendix) [6]	16
3.13	Entity Properties (Mendix) [6]	17
3.14	Generalization Mendix [6]	17
3.15	Example of executing SQL queries in SAP [27]	19
3.16	Example of Physical Diagram in SAP [27]	19
3.17	Apply PDM changes to database dialog box in SAP [27]	20
3.18	Database synchronization dialog box in SAP [27]	20
3.19	Example of Generalisation in SAP PowerDesigner [27]	21
4.1	Data model visual designer architecture	25
4.2	Original TestNodeEditor	26
4.3	Page Components	27
4.4	Solution UI annotated with "Plan" as main entity type selected	28
4.5	Three types of relations that can exist between entity types	29
4.6	Link with two extra vertices	29
4.7	"Add New Relation" Option in "Created" Entity Type	33
4.8	New foreign key form	34
4.9	Remove relation option after right-click on link	34
5.1	Excerpt of the pull request instructions page	42

A.1	Create New Entity MES Screen . . . . .	52
A.2	Edit Entity MES Screen . . . . .	52
A.3	Manage Entity Type Properties MES Screen . . . . .	53
A.4	Manage Entity Type Attributes MES Screen . . . . .	53
A.5	Default Data Model Visualization, with All Entities Collapsed . . . . .	53
A.6	Data Model Visualization, with All Entities Expanded . . . . .	54
A.7	Data Model Visualization with New Entity Type in "Created" state as Main Entity Type . . . . .	54
A.8	Original page of "Plan" entity type . . . . .	54
A.9	Save current view of "Plan" data model . . . . .	55
A.10	Open saved view of "Plan" data model . . . . .	55
A.11	Refresh "Plan" view . . . . .	56
A.12	Hide "Facility" relation from "Plan" . . . . .	56
A.13	Show related entity types of "Enterprise" . . . . .	56
A.14	"Plan" side panel after edit . . . . .	57
A.15	"Plan" original page after edit . . . . .	57
A.16	New foreign key with invalid name . . . . .	58
A.17	New foreign key with invalid link . . . . .	58
A.18	After delete new foreign key element . . . . .	58
A.19	Entity type after adding property . . . . .	59
A.20	Entity type after adding attribute . . . . .	59

# List of Tables

5.1	Validation testing for each requirement . . . . .	37
5.1	Validation testing for each requirement . . . . .	38
5.1	Validation testing for each requirement . . . . .	39
5.1	Validation testing for each requirement . . . . .	40
5.1	Validation testing for each requirement . . . . .	41
5.2	Participants response to SUS survey . . . . .	43

# Abbreviations

3NF	Third Normal Form
CAGR	Compound Annual Growth Rate
CASE	Computer-Aided Software Engineering
CDM	Conceptual Data Model
CM	Critical Manufacturing
CMF	Critical Manufacturing Framework
DBMS	Database Management Systems
DDL	Data Definition Language
ERP	Enterprise Resource Planning
ER	Entity-Relationship
ERM	Entity-Relationship Model
ICT	Information and Communication Technology
LDM	Logical Data Model
LCDP	Low-Code Development Platform
MES	Manufacturing Execution System
PaaS	Platform-as-a-Service
PDM	Physical Data Model
PLC	Programmable Logic Controller
SCADA	Supervisory Control and Data Acquisition
SQL	Structured Query Language
SUS	System Usability Scale
UI	User Interface

# Chapter 1

## Introduction

In this section, a brief introduction of this dissertation's theme is accomplished. An exploration of the context and motivation for this theme will be performed, as well as the objectives for this dissertation, and the overall document structure.

### 1.1 Context and Motivation

Over the years, companies have made digital transformation a key priority, and digital technology platforms have become the foundation for an increasing share of economic activity, resulting in a changing business environment. Modern IT solutions are also increasingly based on the use of customized solutions.

According to Eurostat, among the enterprises that recruited or tried to recruit Information and Communication Technology (ICT) specialists, 55% reported difficulties filling vacancies in 2019. Lack of applications was reported by the highest share of enterprises (42%) as a difficulty for filling ICT specialists' positions [9].

Low-Code Development Platforms (LCDPs) are provided on the Cloud and emphasize visual interfaces, enabling people without a technological background to create and deploy business apps with relative ease [26]. The motivation of this project is to mitigate the shortage of professional and qualified software developers. By allowing end-users with little experience in a specific programming language to contribute to the software development process, LCDPs enable quick generation and delivery of business applications without sacrificing the productivity of a professional software developer.

This project is being developed in a company context at Critical Manufacturing (CM), founded in Portugal, that focuses on providing automation and manufacturing software for high-tech industries. CM's main product is the Critical Manufacturing MES (Manufacturing Execution System), a manufacturing operations management system. It's framework is currently being evolved to accommodate the creation of low-code solutions.

## 1.2 Objectives

The objective of this dissertation is to develop a visual designer of data models that integrates with the Critical Manufacturing Framework (CMF) based on the ER (Entity-Relationship) paradigm [15].

The resulting solution must provide the ability to create a new data model and extend a data model currently in use. This extension feature must allow the creation of new entities, and the creation of new attributes, custom properties, and properties of entity types, providing the ability to describe relations between entities.

Properties can only be added when an entity type is in the "Created" state. An entity type is in the "Created" state before the object schema is generated. Afterward, only custom properties can be created. Since foreign keys can only be properties, relations can only be created when an entity type is in the "Created" state. Attributes can be added in both states.

Since the low-code platform implemented is in a Web environment and only requires modifying the frontend layer of the CMF, the final solutions architecture is developed in HTML5, Angular 12, and CSS. These technologies are already part of the Critical Manufacturing current framework. The backend and persistence layers were not modified, and connections to the backend were made using Critical Manufacturing's API.

## 1.3 Document Structure

This document is structured as follows. Chapter 1, "Introduction," which is the current chapter, introduces and puts into context the topic of this work. It includes this dissertation's context, motivation, objectives, and structure of the document.

Chapter 2, "Problem Analysis", analyzes the current situation of Critical Manufacturing MES and it's framework, what it is, who it is for, and its needs in terms of data modeling. This chapter serves as a way to contextualize this project's objectives better.

Chapter 3, "State of the Art", analyzes the current status of leading Low-Code Development Platforms in terms of data modeling graphical interfaces. It also analyzes the current data model customization software.

Chapter 4, "Proposed Solution", describes the development process of the proposed solution to this dissertation, along with its requirements, architecture, and resulting solution.

Chapter 5, "Validation", presents the results of the validation process of the proposed solution, which was done by having the final user experiment with the final solution and evaluate its effectiveness.

Chapter 6, "Conclusions and Future Work", is dedicated to concluding the document with a brief overview of the points presented throughout and a reflection of future work that could be done to add value to the proposed solution.

## Chapter 2

# Problem Analysis

This chapter contains an overview of the current MES at Critical Manufacturing and its framework, its architecture, data modeling, and current low-code solutions and deployment procedures to properly understand what this project stands on and how it will contribute to the current MES.

### 2.1 Critical Manufacturing MES

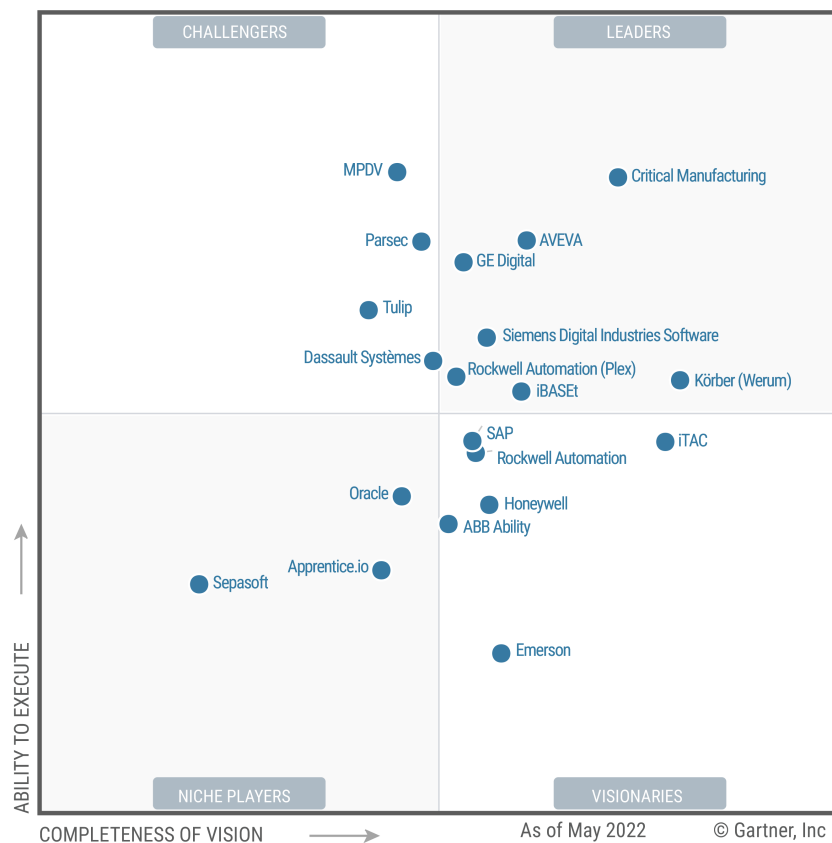
ISA-95 [22], an international standard for integrating enterprise and control systems, presents a functional model that divides the activities necessary for manufacturing systems into five levels.

On the Business Planning and Logistics level (Level 4) are the ERP (Enterprise Resource Planning) systems, responsible for the central functions of accounting and financing, sales and marketing, purchasing, and human resource management [21], enabling the integration of all sectors of an organization. Levels 0, 1, and 2 are responsible for Production Control and Automation, controlled by Programmable Logic Controllers (PLC) systems and robots, and Supervisory Control and Data Acquisition (SCADA) systems used at the control level (level 2).

On the Manufacturing Operation Management level (level 3) are Manufacturing Execution Systems (MES), which fill the communication gap between the manufacturing planning and logistics systems and the control systems used to run equipment on the plant floor [20]. It provides a user interface and data management system, and its primary function is towards manufacturing activities. It fulfills the market's requirements in reactivity, quality, respect of standards, reduction in cost, and deadlines [25].

In 2022, Gartner released their "Magic Quadrant for Manufacturing Execution Systems" which ranks the top MES regarding completeness of vision and ability to execute. "Completeness of Vision" emphasizes offering/product strategy and innovation criteria. "Ability to execute" emphasizes product and sales execution/pricing criteria [24]. Critical Manufacturing is placed among the "Leaders" (Figure 2.1).

The Critical Manufacturing MES helps manufacturers digitize their operations to compete effectively and quickly adapt to changes in demand, opportunity, or requirements.



Source: Gartner (May 2022)

Figure 2.1: Magic Quadrant for Manufacturing Execution Systems [24]

The Critical Manufacturing Framework (CMF) is the foundation on which CM MES is built on.

It consists of three layers (Figure 2.2). The first is the "Presentation Layer", a User Interface (UI) composed of standard components running on a generic shell, developed in HTML5 and Angular.

The second layer is the "Business Layer", developed in .Net. It allows for defining the data model by specifying metadata, properties, references, and relationships, and data access code is automatically generated. The necessary APIs are created just by defining the orchestration code. Pipeline details, exception handling, logging, etc., are all abstracted. Client libraries are automatically generated to call created APIs.

The third layer, "Persistency Layer", developed with Microsoft SQL Server, consists of the database schema and procedures managed by the framework.

The MES is a base product that allows customization to adapt to each client's requirements, providing built-in functionality that covers standard requirements in most applications. This customization is done by CM initially according to the client's needs and can later be modified if



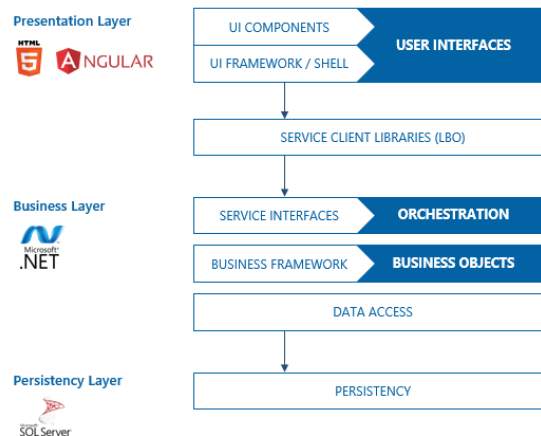


Figure 2.2: Architecture of Critical Manufacturing Framework

needed.

Each client has their own customized version of the MES. For this, a development environment is created for each customization, and CM develops the initial customization with the necessary features required by and for the client.

After the initial customization, the product is deployed to the client, either on the Cloud or on-premises, and the client is free to further customize their own MES to their needs.

## 2.2 Low-Code Features of Critical Manufacturing Framework

Some aspects of the framework have already been implemented with low-code solutions, mainly the dashboard and logic behind it. It allows the creation of custom UIs by reusing predefined widgets and defining data connections between them, also supporting adaptive layouts depending on screen resolution. As figure 2.3 shows, each UI page created can have a set of these widgets that are displayed in certain placement and size (Figure 2.3). The logic and data connection of and between the widgets and the rest of the MES is defined in the "Links" section of the UI page, allowing for a visual representation of the logic behind the page (Figure 2.4).

This canvas of data connection between widgets is powered by JointJS [10], a free, open source library that supports the visualization and interaction with diagrams, graphs and more. Custom elements and links were created in order to customize the look and feel of the canvas and many other functionalities were added.

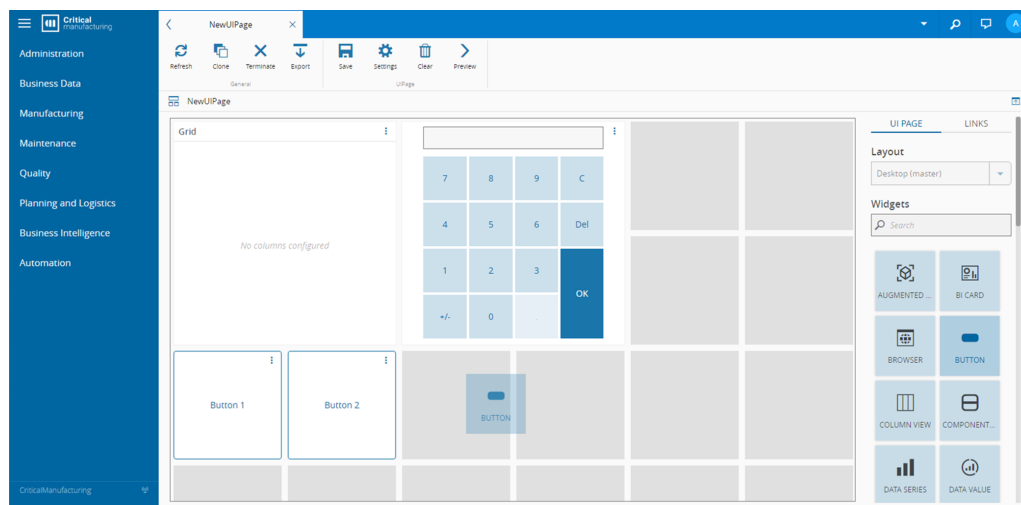


Figure 2.3: Dashboard with pre-defined widgets

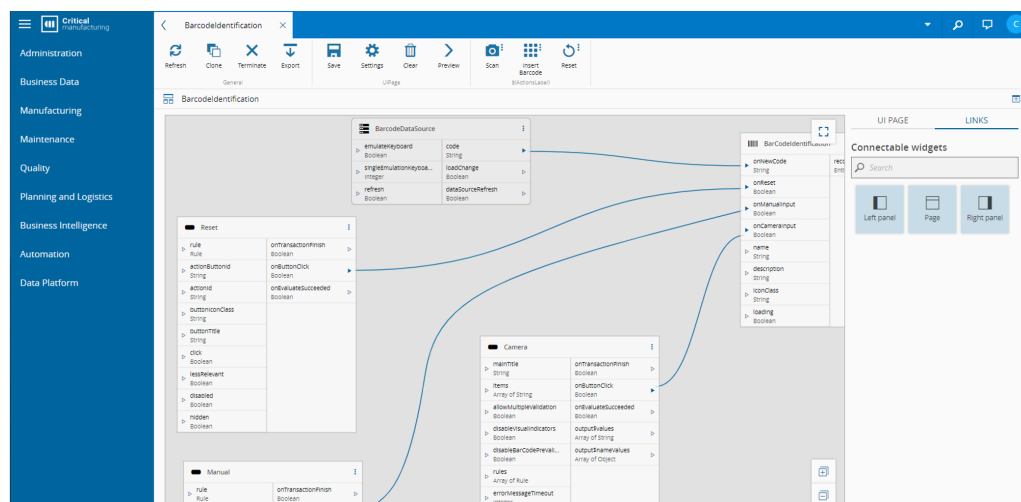


Figure 2.4: Data Connection between widgets

The typical use case for this low-code solution emerges with the need for eventual customization on the client's side, and for the deployment teams, who are responsible for customizing the MES to the clients needs. As the most common need for clients is to modify the dashboard, this was the part that CM developed first in low-code.

## 2.3 Data Modeling Status and Needs

Currently, the data modeling is done using wizards (Figure A.3). To understand how entity types are related, a user must consult the entity type page which is extensive and complex for new clients (Figure A.8).

In figure 2.5, the "Plan" entity type page contains a section with properties, one of these is Calendar, which is a foreign key to the entity type "Calendar". Suppose a user wants to see the entity types that reference "Plan"; it has to search for other entity types with a foreign key that references it by consulting each list of properties on each of their pages. There is no easy and intuitive way to check this.

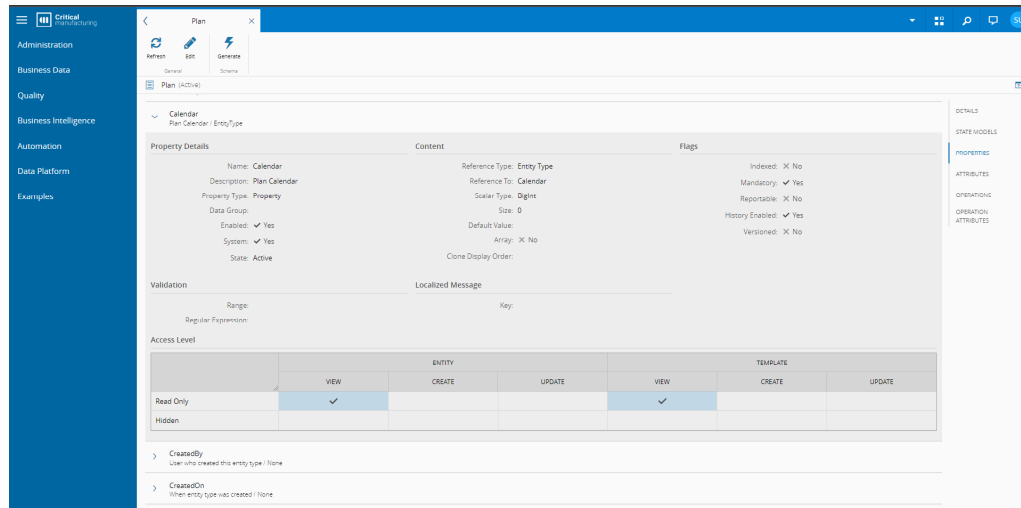


Figure 2.5: Foreign key "Calendar" in entity type "Plan" page

As described in section 1.2, each entity type has a list of properties and custom properties, and may have a list of attributes depending on a toggle the user defines upon the creation of the entity type.

Properties are stored together with the main record and accessed as a property. Custom properties are stored with the main record, but accessed as an attribute. Attributes are properties stored in a separate attribute table.

After creation, an entity type is in the "Created" state, where it is possible to add new properties. An entity type is in this state before the object schema is generated. Since a foreign key can only be a property, new relations are only created while an entity type is in this state.

After generating the schema, it goes to the "Active" state. In this state, adding new properties is no longer possible, and users can only add custom properties. Custom properties can be added by either creating a custom property directly in the properties tab or migrating an existing attribute to a custom property. Therefore, these custom properties will appear in the properties or attributes list on the entity type page, but with the property type "Custom Property". Custom properties can be added either in the "Manage Properties" (Figure A.3) or "Manage Attributes" (Figure A.4) wizards. Attributes can be added in both states.

A list of native entity types is part of the base MES, and new entity types can be added when developing the customized MES version for the client. These native entities cannot be deleted since this could affect other aspects of the MES and create conflicts. Furthermore, all entity types have a list of native properties that cannot be edited or deleted for the same reason.

The relationships between entities and the customization of the data model using the wizards are challenging to understand at first glance. That is where the idea of implementing a low-code solution comes in, allowing people with little to no knowledge of programming languages to modify the current MES data model to their needs.

## Chapter 3

# State of the Art

This chapter contains an overview of current LCDP data modeling graphical interfaces and the leading data model customization software. The latter will focus on ERP systems since they are the most representative, and adopted by leading software companies.

### 3.1 Data Models

For this chapter, it is essential to understand the concepts of the three levels of database design abstraction: conceptual, logical, and physical data models [5].

Conceptual Data Models (CDM) are the most abstract of the three. They help analyze the conceptual structure of an information system by identifying the entities, their attributes, and relationships between them [27].

Logical Data Models (LDM) are a more detailed version of Conceptual Data Models. They help analyze the structure of an information system from an end-user perspective, independent of functional or physical aspects of the database implementation [28]. It is an enriched version of the CDM since the columns in each entity are defined explicitly, and it introduces operational and transactional entities [5].

The Physical Data Model (PDM) represents how the database is (or will be) built, and is a physical manifestation of the LDM into database tables, foreign key constraints, and columns [29], assigning each entity's column a type, length, nullable, etc. [5]

A reference data model for MES, using the ER notation, was proposed by *Zhou et al.* [31], but not in the scope of low-code platforms. The resulting data model included a pre-defined number of entities and accommodated the usual relationships, generalization, attributes, subsets, weak and regular entities, and identification dependencies used to associate regular entities with weak ones (Figure 3.1). A weak entity cannot be identified by its attributes alone, needing a foreign key associated with its attributes to create a primary key.

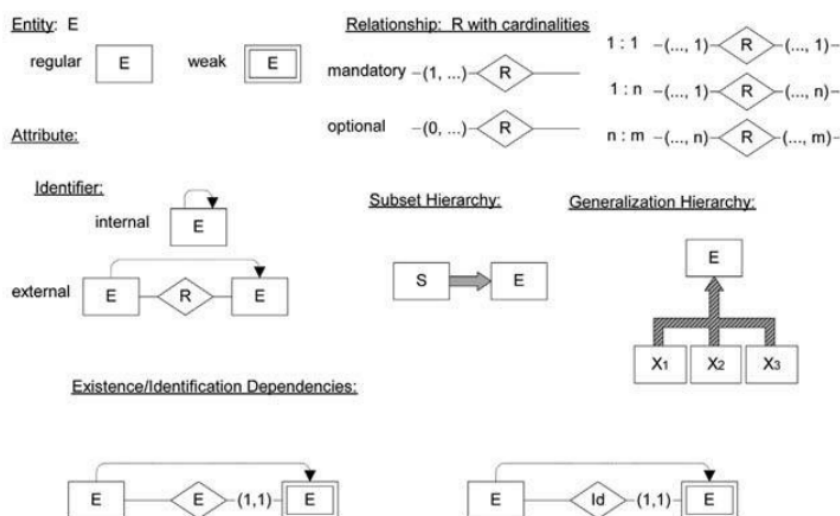


Figure 3.1: Modeling notation [31]

### 3.2 Low-Code Application Development

The term "low-code" was firstly coined by Forrester Research in 2014 (Cambridge, MA, USA) [23]. Low-code Development Platforms (LCDPs) are characterized by their model-driven approach and capitalize on recent developments in cloud computing, on models such as Platform-as-a-Service (PaaS), and proven software design patterns and architectures to ensure effective and efficient development, deployment, and maintenance of the applications. LCDPs enable the rapid development and delivery of software applications through graphical user interfaces, allowing for end-users with no particular IT background to contribute to the development of applications and therefore pulling focus on business logic and knowledge. These platforms also requires little effort in installation, configuring the environments, training, and implementation [30].

The market for low-code development platforms is expected to grow over the years due to its nature of rapid application development and delivery time, and due to the shortage of IT professionals.

According to GlobeNewswire, the global LCDP market is predicted to advance at a pace of 31,1% CAGR (Compound Annual Growth Rate) between 2020 and 2030 [19].

Many companies that focus exclusively on LCDP are now on the market, and other major PaaS providers (such as Google and Microsoft) are now developing their low-code solutions, respectively Google App Maker and Microsoft Power Platform [26].

This section will focus on the two major players (Mendix and OutSystems) since they drive the innovation and execution of low-code development platforms. Both of them use the logical Entity-Relationship (ER) data model.

### 3.2.1 Architecture

In terms of the architecture, low-code platforms are divided into four layers (Figure 3.2). The Application Layer (top layer) consists of the visual interface that users directly interact with. Here lie the toolboxes and widgets necessary to build the application's user interface. This layer also defines authorization and authentication mechanisms to be applied to the specified artifacts. It is also here that users specify the logic and behaviors behind their application.

The Service Integration Layer is used to connect with different services by using APIs and authentication mechanisms. The Data Integration Layer is responsible for operating and homogeneously manipulating data, even from different heterogeneous sources.

At last, depending on the used LCDP, the developed application can be deployed either on dedicated cloud infrastructures or on-premises environments. Containerization and orchestration of applications are handled at this layer and other continuous integration and deployment facilities that collaborate with the Service Integration Layer [26].

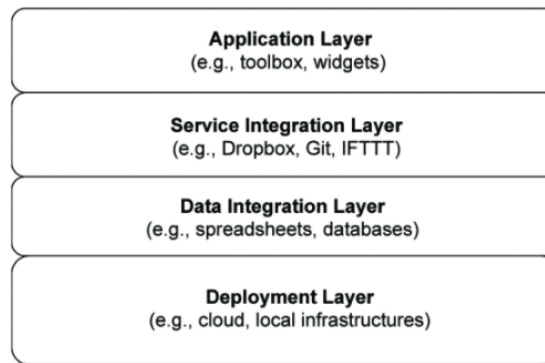


Figure 3.2: Layered architecture of LCDP [26]

### 3.2.2 Data Modeling in Low-Code

Data modeling in developing an LCDP application is usually one of the first steps taken. Users utilize the graphical interfaces provided by this type of platform to configure the data schema by creating the entities, relationships, attributes and defining constraints and dependencies. This configuration is usually done with a drag-and-drop feature, making it intuitive to create and modify data models.

In most low-code platforms, this data structure definition is done in a modeling tool that implements either a variant of the Entity-Relationship Model (ERM) or some proprietary language.

Another common feature is the ability to connect to external data sources using a variety of APIs and to access external relational databases and other persistence technologies. Connection with other files and systems is also possible (e.g., CSV files, Google documents, etc.).

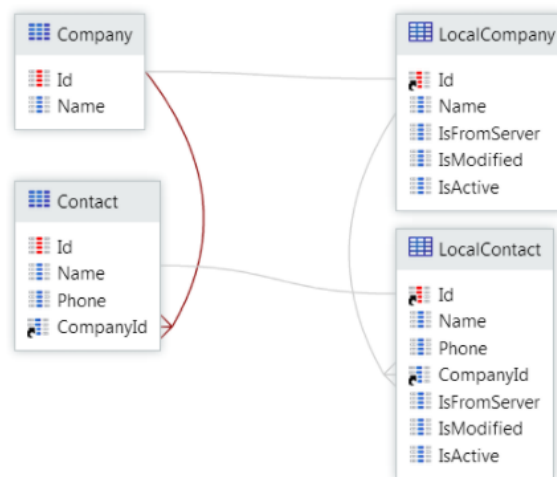


Figure 3.3: Example of OutSystems Data Modeling tool with one-to-one and one-to-many relationships [14]

The ability to store data either in internal database systems or in existing external systems is also present, where the import and export are accomplished according to user-defined patterns [17].

Modifications to external database records are possible since most low-code platforms allow executing SQL queries. Some SQL queries available are Select, Update, Insert, Delete, and Stored Procedure. However, Data Description Language (DDL) queries are also present in LCDPs, but usually not recommended [12] [8] [11].

### 3.2.2.1 OutSystems

OutSystems allows the development of desktop and mobile applications, which can run on the cloud or local infrastructures or hybrid environments. As an LCDP, it requires little to no code writing for development. In its core are two components, the integration studio, for database connection, and the service studio, to specify the behavior of the application being developed. OutSystems supports the development of various types of applications such as billing systems, CRMs, ERPs, operational dashboards, and others [26].

In regard to visual representation of the data model, as presented in Figure 3.3, the Relational Data Model graphically shows us the tables and relationships between them to represent the existing database.

In terms of entities, each entity has a set of attributes and CRUD (Create, Read, Update and Delete) actions associated, organized into a module tree (Figure 3.4).



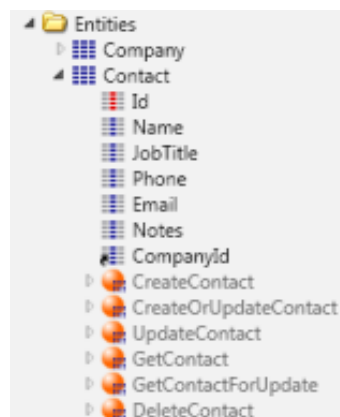
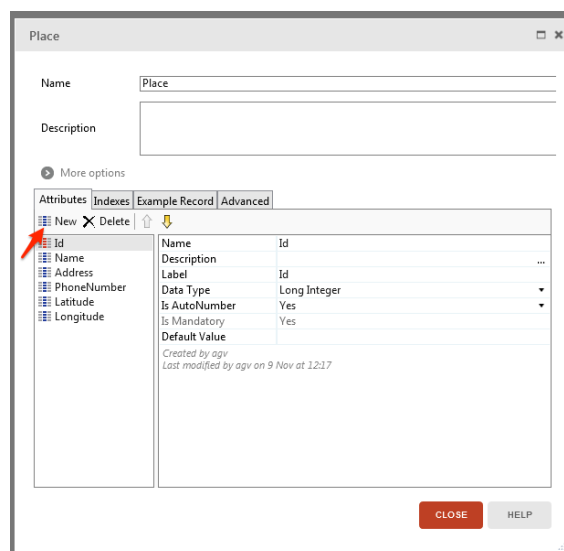


Figure 3.4: Example of Entities in Module Tree (OutSystems) [7]

Adding an entity can be done by either clicking anywhere on the canvas and selecting the "Add Entity" option or right-clicking on the Entities folder and selecting "Add Entity" (Figure 3.5).



Name	Label	Data Type	Is AutoNumber	Is Mandatory	Default Value
Id	Id	Long Integer	Yes	Yes	

Figure 3.5: Form to create and edit entity (OutSystems) [13]

Foreign keys are defined as an entity attribute that references another table that appears as an attribute but with an arrow to differentiate from other attributes. The example in Figure 3.6 shows a one-to-many association where an entity "Contact" keeps a foreign key referencing "Company".

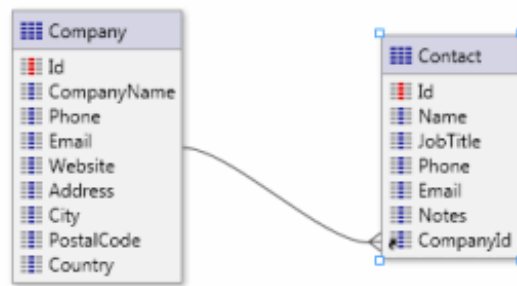


Figure 3.6: Entity diagram of one-to-many association (OutSystems) [7]

The "on delete" rule can be defined in the foreign key that references the other entity that is part of the relationship (Figure 3.7).

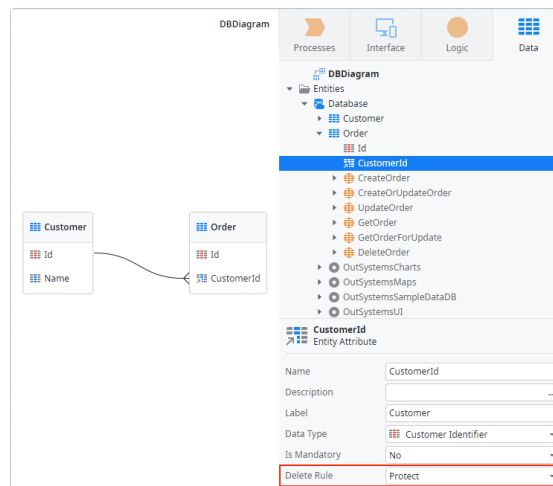


Figure 3.7: On delete rule (OutSystems) [7]

SQL queries can be performed in an action flow, and therefore is not shown in the data modeling. However, it is advised to not perform DDL queries. OutSystems does not support generalization, specialization, or aggregation in the Relational Data Model.

### 3.2.2.2 Mendix

Mendix, much like OutSystems, allows the development of desktop and mobile applications with little to no code writing and where features can be accessed through drag-and-drop capabilities. Mendix's Solution Gallery allows users to start from already developed solutions that might be enough to satisfy their requirements [26].

As for data modeling, Mendix displays a Relational Data Model that shows the entities, their attributes, and the relationship between entities in a visual and interactive format (Figure 3.8).



Figure 3.8: Example of a one-to-many relationship where Order is the Owner (Mendix) [2]

As shown in the Figure 3.8, there can be a concept of "Owner" in each association. Mendix recommends that the Owner be the entity with more records. That means that the "Owner" object refers to the other. While if there is no "Owner" in the association, each side of the association refers to each other. The Owner concept can only be present in one-to-many or many-to-many relationships.

This concept exists so that if the initial design has a one-to-many relationship and later needs to be transformed to a many-to-many relationship with default ownership (the default will be to have an owner, and in this case, it will be the same as in the one-to-many relationship), there is no need to rebuild the database since Mendix can do it automatically [2].

The dashboard also displays a menu bar for quick access to the most important features for data modeling, with the highlight in the Figure 3.9 showing how to add an entity.

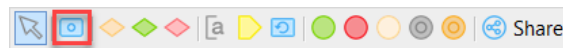


Figure 3.9: Menu Bar (Mendix) [4]

The three following images show the dialog boxes that appear for editing: an attribute (Figure 3.10), an association (Figure 3.11), and an entity (Figure 3.12).

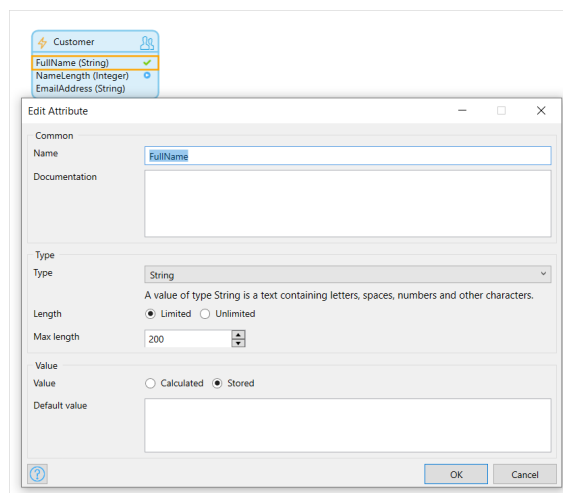
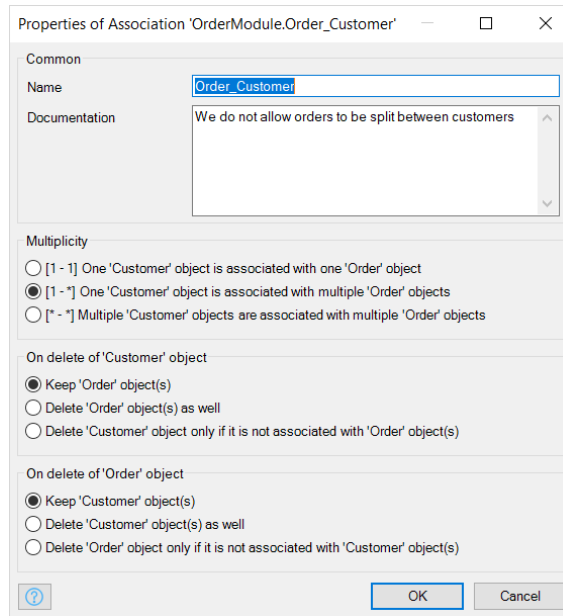


Figure 3.10: Edit Attribute dialog box (Mendix) [3]



Properties of Association 'OrderModule.Order\_Customer'

**Common**

Name:

Documentation:

**Multiplicity**

☐ [1 - 1] One 'Customer' object is associated with one 'Order' object  
☒ [1 - \*] One 'Customer' object is associated with multiple 'Order' objects  
☐ [\* - \*] Multiple 'Customer' objects are associated with multiple 'Order' objects

**On delete of 'Customer' object**

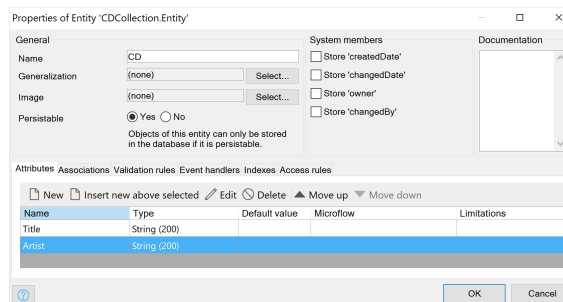
☒ Keep 'Order' object(s)  
☐ Delete 'Order' object(s) as well  
☐ Delete 'Customer' object only if it is not associated with 'Order' object(s)

**On delete of 'Order' object**

☒ Keep 'Customer' object(s)  
☐ Delete 'Customer' object(s) as well  
☐ Delete 'Order' object only if it is not associated with 'Customer' object(s)

OK Cancel

Figure 3.11: Edit Association Properties dialog box (Mendix) [1]



Properties of Entity 'CDCollection.Entity'

**General**

Name:

Generalization:

Image:

Persistable: ☒ Yes ☐ No  
Objects of this entity can only be stored in the database if it is persistable.

**System members**

☐ Store 'createdDate'  
☐ Store 'changedDate'  
☐ Store 'owner'  
☐ Store 'changedBy'

**Documentation**

**Attributes Associations Validation rules Event handlers Indexes Access rules**

Name	Type	Default value	Microflow	Limitations
Title	String (200)			
Artist	String (200)			

OK Cancel

Figure 3.12: Edit Entity Properties dialog box (Mendix) [6]

In the edit entity dialog box, we can also modify the association, create new attributes, change the entity's name, create indexes and access rules, etc. These values can then be viewed in the dashboard of the data model in this format to facilitate consulting of entity properties. 3.13

Entity 'CRM.Customer'	
▼ <b>Access rules</b>	
Access rules	(1 item)
▼ <b>Documentation</b>	
Documentation	
▼ <b>General</b>	
Name	Customer
Generalization	
Image	icon_users
Persistable	Yes
▼ <b>System members</b>	
Store 'createdDate'	No
Store 'changedDate'	No
Store 'owner'	No
Store 'changedBy'	No

Figure 3.13: Entity Properties (Mendix) [6]

Mendix allows for performing SQL queries in external databases with their microflows, same as with OutSystems. Mendix allows for generalization and specialization, however aggregation can only be done using event microflows or delete behaviour. In the image referenced, the Member entity is a generalization of the Student and Professor entities. 3.14

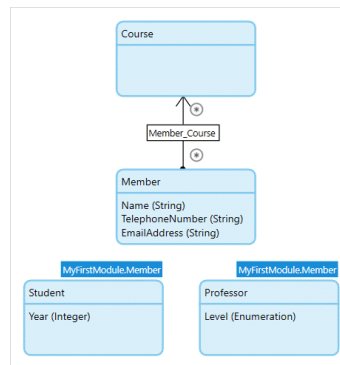


Figure 3.14: Generalization Mendix [6]

### 3.3 Data Model Customization

In this section, an analysis is made of current software that allows for data model customization. Since ERP systems are the most representative, in terms of data modeling customization, we will focus on two of the most used and capable ones at the time of writing, Oracle and SAP, and analyze their data model customization features.

According to Heiko Meyer et al., both ERP and MES systems collect data, and ERP systems already have a large part of master data needed for MES systems however lack the degree of detail required [21].

ERP and MES systems share specific features, for example, material management, planning, and scheduling. ERP focuses on the flow of information and are designed to integrate business processes and functions to provide a more detailed view of the organization and optimize business performance. On the other hand, MES monitors and produces documentation of the manufacturing process, integrating directly with machines on the shop floor to improve its effectiveness [16].

Both ERP tools presented in this analysis incorporate at least the PDM and LDM in their logic, being possible to generate the other models from the one already developed. However, the model that more directly represents the actual database is the PDM. If customization is to be done using one of the other models it is necessary to generate the PDM from that model. For this, an analysis of the physical data layer customization capabilities and how these ERPs allow the developer to make them is done.

### 3.3.1 Oracle Retail

Oracle Retail is an ERP targeted to meet retail needs through a standards-based approach to retail data warehousing, allowing for companies to gain insight into and from their data more efficiently. This software can be used in any application environment and allows for extensions of the model.

The default physical model of Oracle Retail Data Model shares characteristics of a multi-schema "traditional" data warehouse but defines all the data structures in a single schema. This means that it has three layers: staging, foundation, and access.

The Staging layer is used when moving data from the transactional system and other data sources into the data warehouse itself.

The Foundation/Integration Layer is implemented in the Third Normal Form (3NF), which preserves a detailed record of each transaction without any data redundancy. Physical objects for this layer are defined in a single schema, the `ordm_sys` schema.

The Access Layer is traditionally defined as a snowflake or star schema that describes the summarized or dimensional data perspective in the foundation layer.

In order to modify the physical data model, changes to the Foundation Layer have to be made. They can either reflect differences between the company logical model needs and the default logical model provided by Oracle Retail Data Model or improve performance. Oracle recommends packaging the changes made to the physical data model as a patch to the `ordm_sys` schema [29].

These changes are made using DDL (Data Definition Language) queries, so no low-code solution was found to modify the data model.

### 3.3.2 SAP PowerDesigner

PowerDesigner, SAP's data modeling tool, provides full support for round trip generation and reverse-engineering between a PDM and a database [27]. Data customization can be done either with SQL queries (Figure 3.15) or with the use of PDM.

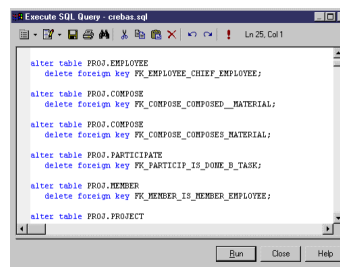


Figure 3.15: Example of executing SQL queries in SAP [27]

SAP uses physical diagrams to provide a graphical view of the database structure, helping in the analysis of its tables, view, and procedures. These diagrams can be created from an existing PDM (Figure 3.16).

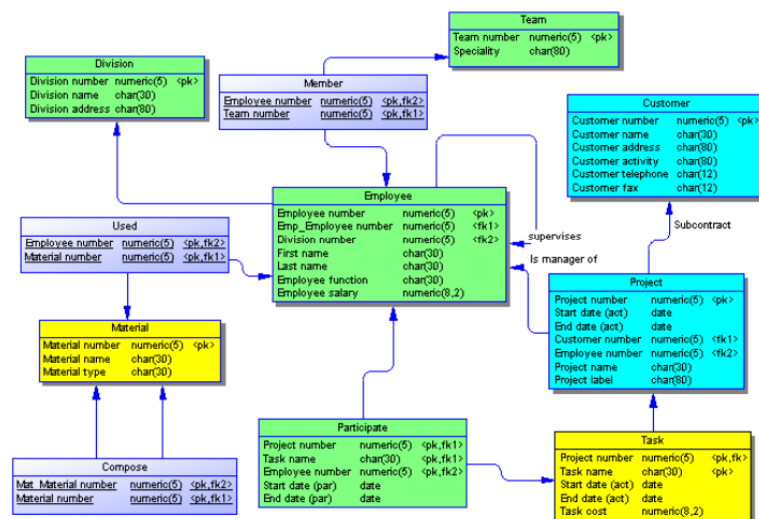


Figure 3.16: Example of Physical Diagram in SAP [27]

Editing of the physical diagram is made graphically, much like the low-code platforms presented in the section above.

After modifying the PDM, it is also necessary to modify the existing database to reflect these changes in the model. The PDM and existing database schema (target model) merge using database synchronization, which allows choosing which objects are added, deleted, or updated in the target.

To accomplish this, in the dialog box (Figure 3.17), a destination Directory and File name for the script file is chosen, along with the type of generation (script or direct generation) to perform.

Through script generation, we obtain a script to be executed on the DBMS (Database Management System) later. Through direct generation, a script is generated and executed on a live

database connection. It is possible to edit the script before it is executed on the database in both options.

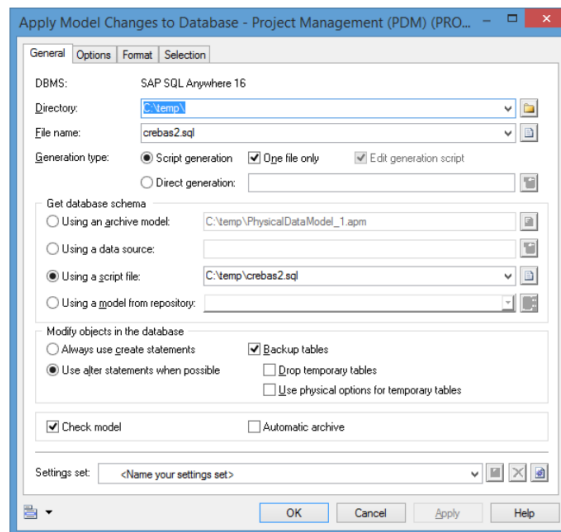


Figure 3.17: Apply PDM changes to database dialog box in SAP [27]

After this, the database synchronization windows opens in order to allow for objects that one wants to include or remove from the model (Figure 3.18).

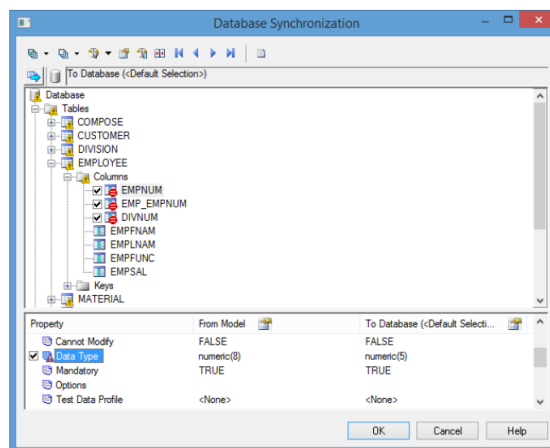


Figure 3.18: Database synchronization dialog box in SAP [27]

SAP also allows reverse-engineer of an already existing database schema into a PDM from one or more script files or a live database. This feature simplifies the customization of the current database since the physical diagram is a low-code solution, and SAP takes care of representing the database in a diagram for easier customization.

Inheritance can be shown in the logical and conceptual data models and are translated into references in the physical data model.



Generalisation can be defined in the CDM and LDM, and is translated to references (foreign key constraints) in the PDM. In the example shown, the Account entity is a generalization of the Savings and Checking entities (Figure 3.19). Does not allow for aggregation in the data models.

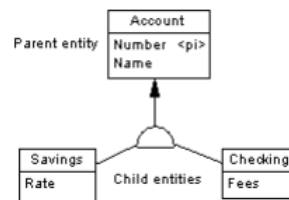


Figure 3.19: Example of Generalisation in SAP PowerDesigner [27]

### 3.4 Summary

According to the research analyzed in this chapter, we can conclude that the overall architecture of a usual LCDP is similar to the Critical Manufacturing Framework (Figure 3.2.1).

The research shed light on techniques used in the current market regarding data model visualization. Among these techniques, some of the ones that stood out were the use of a canvas (where the data model is displayed) within a page, allowing the manipulation of the nodes and links freely throughout the canvas, using a side panel for more detailed information about the data model, the use of wizards to facilitate interaction with the data model (for more complex interactions), etc.

It also explored data model customization software in the current market. While Oracle Retail did not have a low-code aspect, SAP PowerDesigner seems to incorporate it, allowing for the creation and extension of external databases. However, information was not accessible on other Manufacturing Execution Systems and if they contained this data model customization in low-code in their framework. Considering that Manufacturing Execution Systems are usually proprietary software, they are not disclosed to the public, and gathering this information is not feasible.

## Chapter 4

# Proposed Solution

In this section, a detailed description is made of the various aspects of developing the proposed solution. An exploration of the solution's requirements is conducted, followed by a description of the architecture and its different components and dependencies. Furthermore, a demonstration of the resulting solutions' user interface and interaction design is accomplished, dividing it into the two main aspects, visualization and edition of the data model.

For this section, it is essential to note that the final solution consists of a page with an action bar, a side panel, and a canvas. In this canvas, the data model will be displayed, consisting of entity type nodes and links representing relations.

Another important fact is that entity types can be in one of many states. The two most important ones for this dissertation will be the "Created" and "Active" states. An entity type also has properties, custom properties, and attributes. Their addition, edition, and removal also depend on the state of the entity type. The distinction between these categories and states is already addressed in Section 2.3.

### 4.1 Solution Requirements

In the process of developing the solution, a requirements gathering was done. A list of the solution requirements follows:

- R1: The canvas should allow the visualization of the data model in the form of "nodes" and "links" based on the ER (Entity-Relationship) paradigm.

Each "node" should resemble an ER diagram node, much like the ones presented in Chapter 3, which means it should have a header with the entity type name and a body containing its properties' names and types. In the final solution, since each entity type has properties, custom properties, and attributes, these should be divided into sections in the node in order to distinguish between them.

Each link should have an annotation resembling the ER diagrams presented in Chapter 3 (Figures 3.1, 3.3, and 3.8) in order to distinguish what type of relation it represents.

- R2: The canvas must allow the manipulation of the data model elements to customize how they are displayed.
- R3: The data model should be visualized from a single main entity type and display only it and its directly related entity types by default.

This visualization choice proved necessary since the base MES has over 900 entity types, which would render the visualization of the whole data model extraordinarily cumbersome and complex to the user.

- R4: The data model should be presented in a visually understandable and readable way by default.

This will be done by developing a default layout that automatically places the nodes in such a way that minimizes overlapping nodes and links.

- R5: Each node should be able to collapse and extend.

Given the number of properties that some entity types have, "nodes" should be collapsible so as to not overcrowd the canvas. An extended node shows its entity type properties, custom properties, and attributes.

- R6: Provide a way to access the original CMF entity type page if the user needs more detailed information.
- R7: Be integrated into the CMF, maintaining the same technologies and languages, which implies that the proposed solution should use TypeScript, HTML5, and CSS (Figure 2.2).

The choice of language was due to the fact that the proposed solution should only modify the frontend layer, and make calls to the backend using the Critical Manufacturing's API. The backend and persistence layer should not be modified.

- R8: Employ JointJS as the framework used for the canvas.

In order to keep the overall look and feel of CMF, so as to maintain visual coherence, the framework used for the canvas will be JointJS to resemble the previously developed page in the CMF referring to the data connection (Figure 2.4). The whole page should also maintain the overall page format of the CMF.

- R9: Each node should allow showing/hiding relations of an entity type.

Since the default visualization of a data model is based on a main entity type and its direct relations, the ability to show the relations of an entity type that does not yet have them displayed is necessary. Hiding those relations is also important if the user decides no longer needs to visualize them.

- R10: Each node should allow to show/hide relations per property (that belongs to a relation).

Considering that some entity types have many foreign keys or entity types that reference it, the capacity to show and hide the relations associated with a particular entity type property is essential.

- R11: The page must allow the creation of a new entity type.
- R12: The page should allow editing an existing entity type.
- R13: The canvas/page must allow adding and removing properties and custom properties of entity types.
- R14: The canvas/page must allow adding and removing attributes and custom properties of entity types.
- R15: The canvas should provide a way to distinguish what entity types could be modified in terms of adding/removing relations. Since only entity types in the "Created" state can have new foreign keys added and removed.
- R16: The canvas should provide a way to distinguish between "normal" entity types and entity types that serve only as a relation between two other entity types (join tables).
- R17: The canvas must allow visually adding and removing a new relation from an entity type in the "Created" universal state.
- R18: The page should allow saving the current view of the data model by main entity type.
- R19: The canvas should be able to be opened to the previously saved data model by main entity type.
- R20: The page should allow resetting the view of current main entity type data model.
- R21: The canvas must be able to be refreshed, maintaining the same entity type.
- R22: The page should allow an entity type not already displayed in the canvas to be added to it.

The user may want to view the data model of an entity type that is not associated with the main entity type selected. This additional data model could create performance issues but may be necessary for some scenarios. Another use for this requirement is when a user wants to create a new relation with an entity type that is not present in the canvas.

## 4.2 Architecture and Technologies

The proposed solution architecture, as described in the Requirements (Section 4.1), follows the CMF overall architecture (Figure 2.2). It consists of a frontend application that performs API calls to the Host, which consists of a collection of methods and properties associated with the

underlying business layer of Critical Manufacturing to obtain information about the entity types and perform certain modifications on the given entity types, among other uses. With this in mind, this project did not interfere with the backend of CMF.

Also, as mentioned in the Requirements (Section 4.1), the technologies used for the proposed solution followed the ones already used by the CMF: TypeScript, HTML5, and CSS. HTML5 is responsible for the structure of the component, CSS for formatting the style of the structure (HTML), and TypeScript for employing the component's logic. The framework used is JointJS, a diagramming tool built on JavaScript, which was already utilized in other sections of the CMF, specifically the data connection tool (Figure 2.4). It was considered the best choice of framework since it can be integrated with Angular and offers the best features for the desired outcome (Figure 4.1).

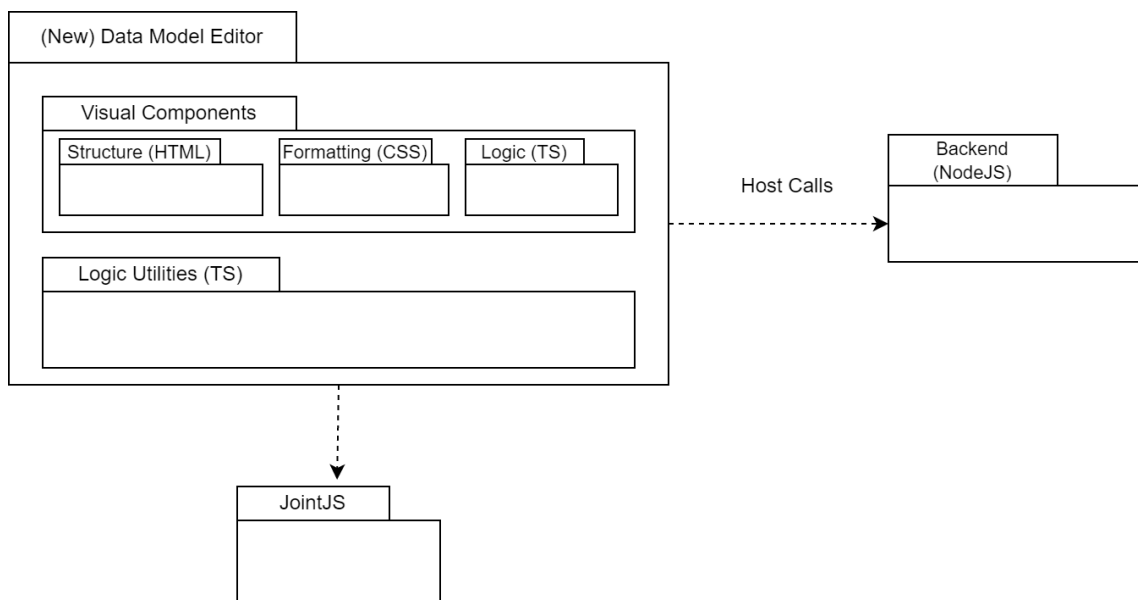


Figure 4.1: Data model visual designer architecture

The page consists of various components representing different visual elements each implemented in a separate source file, and many other utility files that aid in the functioning of the page.

The component that represents the page as a whole is called "TestNodeEditor", which was previously set up with a simplified version of the data connection between widgets canvas of the CMF (Figure 4.2).

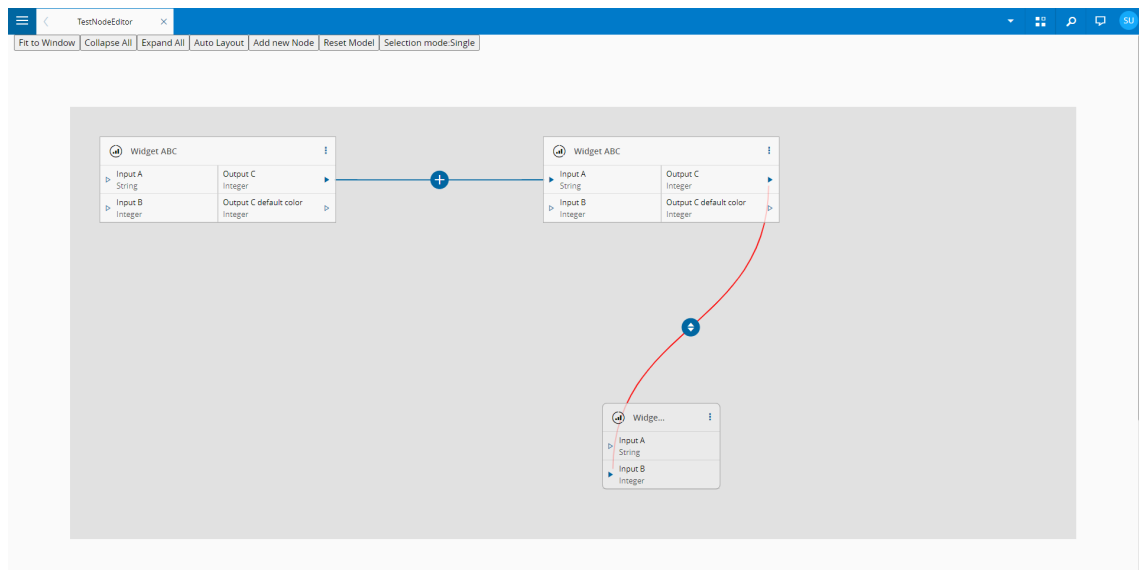


Figure 4.2: Original TestNodeEditor

### 4.3 UI Structure

The final "TestNodeEditor" component consists of a combination of many other components already existent in the CMF. However, most of them had to be modified to perform the actions and display the information needed for the expected result of this dissertation (Figure 4.3).

- "BasePage": The most abstract component is the "BasePage", consisting of a header and an empty body that the parent component must fill with its desired components. The header is collapsible and contains an empty "ActionBar", and a section with the button to collapse it. In order to add the two top buttons of "Main Entity Type" and "Add Entity Type to View", an empty div had to be added to the original "BasePage" in order to place the buttons in the section below the action bar where the button is present. For the body, two components were added "NodeEditor" and "PageSplitter".
- "ActionBar": For the "ActionBar" component, two "ActionGroup" components were added, "General" and "Entity type data model", each with their own "ActionButton"'s. These action button groups were separated, as the name implies, according to either general or the current data-model specific actions and follow the overall structure of the remaining CMF pages, where the "Create" and "Refresh" buttons are usually placed in the "General" action group.
- "PageSplitter": The "PlaceSplitter" component is used to separate the body of "BasePage" into two to add the side panel. To this side panel, a list of details of the selected entity type was added with the use of components already existent in the CMF, and no modifications were made to these.

- "NodeEditor": The "NodeEditor" component represents the canvas. In terms of logic, the "NodeEditor" deals with pointer-events and most of the actions made to the canvas, such as adding/removing nodes, adding/removing links, and most of the logic of the canvas.
- "SimpleGraphNode": The nodes are visually rendered by the "SimpleGraphNode" component. This "SimpleGraphNode" deals with the changes to a particular node and the buttons associated with them, emitting events to "NodeEditor" or "TestNodeEditor" when necessary, which is mostly when a change causes a ripple effect in the rest of the canvas. For example, when trying to show the relations of a given entity, clicking on "Show Related Entity Types", causes changes not only in the node itself but also in the overall canvas and, therefore, in the "NodeEditor", since more links and entity types need to be created and displayed. This behavior is also present between "NodeEditor" and "TestNodeEditor", where the parent (TestNodeEditor) receives emitted events from the child (NodeEditor).

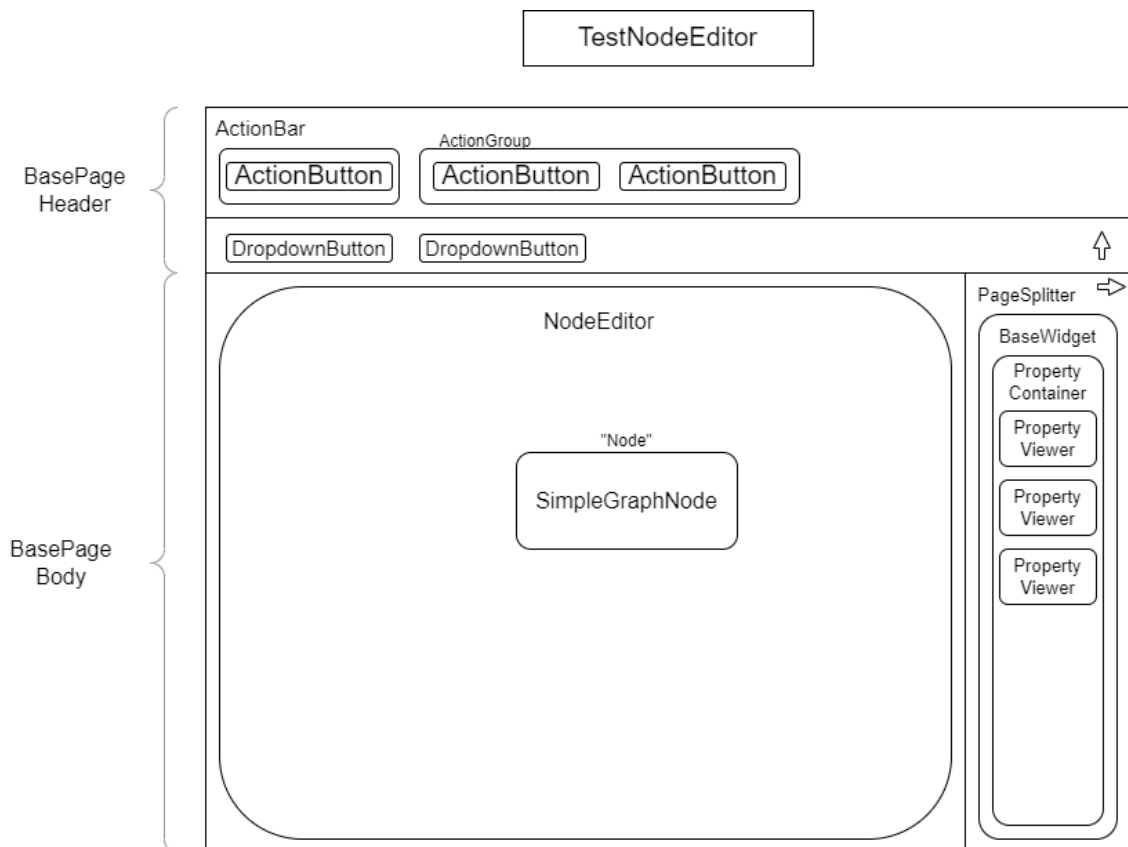


Figure 4.3: Page Components

In terms of files and components without a visual aspect (Utilities), there are four TypeScript files in use: "NodeEditorBase", "NodeEditorModel", "NodeEditorUtils", and "BaseGraphNode".

- "BaseGraphNode": Is an abstraction of the "SimpleNodeGraph". This abstraction is done to reuse some of the "BaseGraphNode"'s utilities in other CMF components.

- "NodeEditorBase": Is a utility file to hold the interfaces for "NodeEditor" events and constants and to be an abstract class of "NodeEditor".
- "NodeEditorModel": Holds the interfaces necessary for "NodeEditor" (such as points, links, and ports) and, since it is used in other components of the CMF, is necessarily separated from "NodeEditorBase".
- "NodeEditorUtils": Is the utility file responsible for instantiating the custom graph components such as the elements (nodes) and links, and for some other general functions, for example, cloning nodes and comparing points. This instantiation of custom graph components also creates custom behaviors, such as when a node is resized (on collapse, expand, or by adding/removing a property). This file instantiates the custom nodes but is only a "skeleton" of the actual node presented in the canvas, the information and the look of each node is, as said above, in the "SimpleGraphNode".

JointJS components and functions are used throughout these files.

## 4.4 User Interface and Interaction Design

This section is divided into two subsections, visualization and edition. As the name implies, visualization will focus mostly on what the user can see and interact with within the page, and edition will focus on what is allowed and possible in terms of edition and addition to the existing data model and entity types.

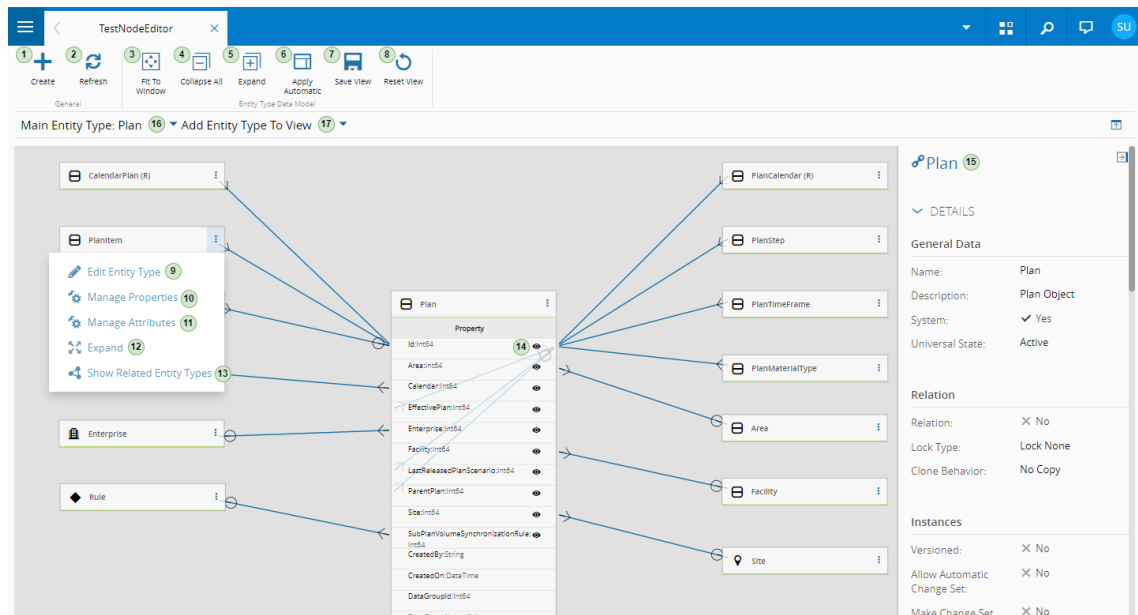


Figure 4.4: Solution UI annotated with "Plan" as main entity type selected



The enumeration in the following sections is a reference to Figure 4.4 which displays most of the components and buttons of the solution.

#### 4.4.1 Data Model Visualization

The user interface consists of an action bar with buttons that perform general actions on the whole canvas, a side panel that displays general information about the currently selected entity type and the canvas that displays "nodes" that represent each entity type, and "links" that represent the relations between nodes. Links have an annotation that represents what type of relation it is. In figure 4.5, there are three types of relations present: One Mandatory to Many Optional, One Mandatory to Many Mandatory, and One Optional to Many Mandatory. Mandatory means it has at least one reference, and Optional can have none. These three examples show the types of annotations that can be seen in the canvas.

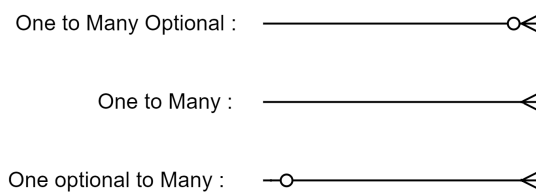


Figure 4.5: Three types of relations that can exist between entity types

Upon first entering the page, the canvas is clean, and no entity type is selected. The canvas allows to move the nodes freely within the bounds of the canvas and to add vertices to the links in order to modify their shape (Figure 4.6). Each link is connected to the property that associates the entity types. Therefore, each relation has to be connected to a foreign key and an Id.

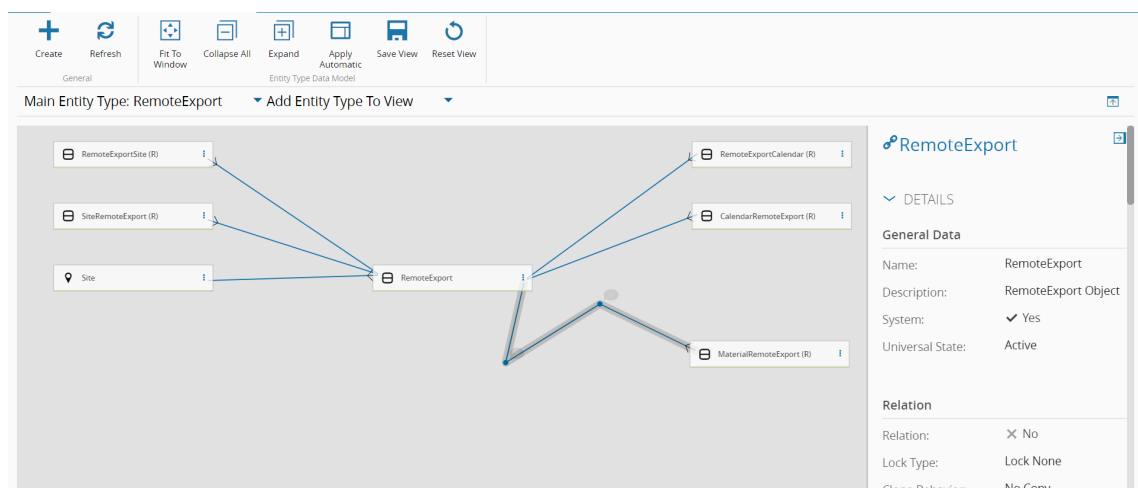


Figure 4.6: Link with two extra vertices

The action bar also displays two dropdown buttons, one where the user selects the main entity type to be displayed (Figure 4.4, Annotation number 16) and the other that lets the user add another entity type to the canvas if it's not already displayed (Figure 4.4, Annotation number 17). Both the side panel and the action bar can be hidden for a bigger and more spacious canvas which can be useful in cases where there are many entity types being displayed or expanded.

#### 4.4.1.1 Action Bar

The action bar contains the following action buttons (numbered in Figure 4.4):

- Refresh (2): Refreshes the entities, their properties, custom properties and attributes, and the links between them if any foreign key is added, keeping the node positions and hidden relations before refresh.
- Fit to Window (3): changes the canvas size and focus to the current data model, positioning it centered in the canvas.
- Collapse All (4): Collapses all the nodes in the canvas, hiding the body and only showing the header of each entity type node.
- Expand All (5): Expands all the nodes in the canvas, showing the body of each node.
- Apply Automatic Layout (6): Returns the entities to their original positions, applying the default layout.
- Save View (7): Saves the current view of the canvas by saving the entities, their properties, custom properties and attributes, position of the nodes, and hidden property relations using the JavaScript "localStorage", a property that allows saving key-values pairs in a web browser with no expiration date.
- Reset View (8): Deletes the previously saved view, if any, and returns to the original view of the main entity type selected, which consists of the entity type and its direct relations, with no hidden properties or entities.

#### 4.4.1.2 Side Panel

The side panel (Figure 4.4, Annotation number 15) shows the main information regarding the entity type of the node that is currently selected. The name displayed at the top of the side panel redirects to the original page of the CMF regarding that entity type, allowing for a complete look into it. The entity type described in the side panel changes to the one corresponding to the currently selected node.

The choice to add a side panel was triggered by the need to have a place where the main information about an entity type is displayed for easier access by the user and to put the link that redirects to the original entity type page in the CMF.

#### 4.4.1.3 Canvas

Each node represents an entity type and displays its properties, custom properties, and attributes. The header contains the entity type name and a dropdown button that shows a list of options to perform on the specific node/entity type. The outline of the header can either be green for "Active" entity types or yellow for "Created" entity types. The entity types that are simply to serve as join tables of two other entity types with a many-to-many relation have an "(R)" in the header in order to distinguish between them and "normal" entity types.

Each node has the following visualization buttons (numbered in Figure 4.4):

- Show/Hide property relations (14): If a property is either an Id or foreign key that references or is a reference to the current entity type, an "eye" will appear that allows showing/hiding the relations associated with that property. Both sides of the relation will then have the "eye" set to hidden, and that relation can be shown again by clicking the "eye" on either side.
- Expand / Collapse (12): changes the view mode of each node from collapsed to expanded and vice versa. In the collapsed mode, it only displays the header, and in the expanded mode, the properties, custom properties, and attributes become visible.
- Show / Hide Related Entity Types (13): Displays the relations of the given entity type that are not yet on the canvas, which consists of entity types that either reference or are referenced by it.

#### 4.4.1.4 Default Layout

One of the challenges in the project was deciding how to generate a default layout that would avoid overlapping nodes and links, and intersections in the latter.

For this purpose, the entity type nodes are displayed in a grid-like layout, where the main entity type is displayed in the center, and the related ones are then displayed either on the left or right of the main node, sequentially. Since all entity types have as their first property their Id, the ones that reference the main entity type (that have a foreign key to the main entity type) are displayed at the top. The other entity types are displayed in order of appearance of their foreign key within the body of the entity type node. This made sure that the data model would avoid intersections (Figure 4.4).

From this point, if the user wants to show the relations of any of the entity types that are present in the canvas, those related entity types will keep the same side as this originating entity, keeping themselves to the right-most or left-most. They will also spawn in the same order as the others, starting with the ones that reference the original entity and then moving on to the ones that are referenced by the original, in the same order as they appear in the body of the node (Figure A.13).

This also guarantees that there are no overlapping or colliding links. However, nodes can overlap if expanded since the default layout considers their positioning when collapsed.

Although a more commonly seen type of connection routing would be one that kept the links at a 90° angle, for example, the Manhattan routing, a linear option was chosen. Since many of the links can connect to the same entity type in the Id property, this can create overlapping in the links that can be avoided by using this linear method of routing.

#### 4.4.2 Data Model Edition

In terms of the edition of the data model, the actions that are allowed to be performed with the currently proposed solution are the creation of a new entity type, the addition and removal of relations from an entity type in the "Created" state, and the addition and removal of properties and attributes using CMF wizards.

All of these changes are reflected in the persistence layer but the user may need to refresh the data model or page in order to visualize them. When the modification is not done by one of the existing CMF wizards (ex: Create Entity wizard), this reflection to the persistence layer is done by making calls to the backend to modify the given entity types.

##### 4.4.2.1 Create Entity Type

The remaining action button in the action bar is the "Create" Button (Figure 4.4, Annotation number 1), that takes us to a screen where the user is able to create and customize a new entity with the desired flags and other details related to the entity type (Figure A.1). This wizard is already contemplated in other pages of the CMF. After creating the entity, it will redirect the user to the solution's page.

The user must refresh the page to see the new entity type in the dropdown and be able to visualize it's data model.

##### 4.4.2.2 Edit Entity Type

In the dropdown button in an entity type node header, there is an option to "Edit Entity Type" (Figure 4.4, Annotation number 9), that opens the wizard that allows editing the given entity type (Figure A.2).

The user must refresh the page to see these changes in the side panel or canvas.

##### 4.4.2.3 Add and remove relation

As mentioned in the sections above, the creation of a new relation between two entities requires that the one that will possess the foreign key to another entity type be in the "Created" universal state. Only those entities will have in the dropdown button the option to "Add new relation". The "Add new relation" button (Figure 4.7, Annotation number 18) adds a new element in the list of properties that has both a text area and two "ports" to help indicate that a connection needs to be made. The text area represents the foreign key name (Figure 4.8).

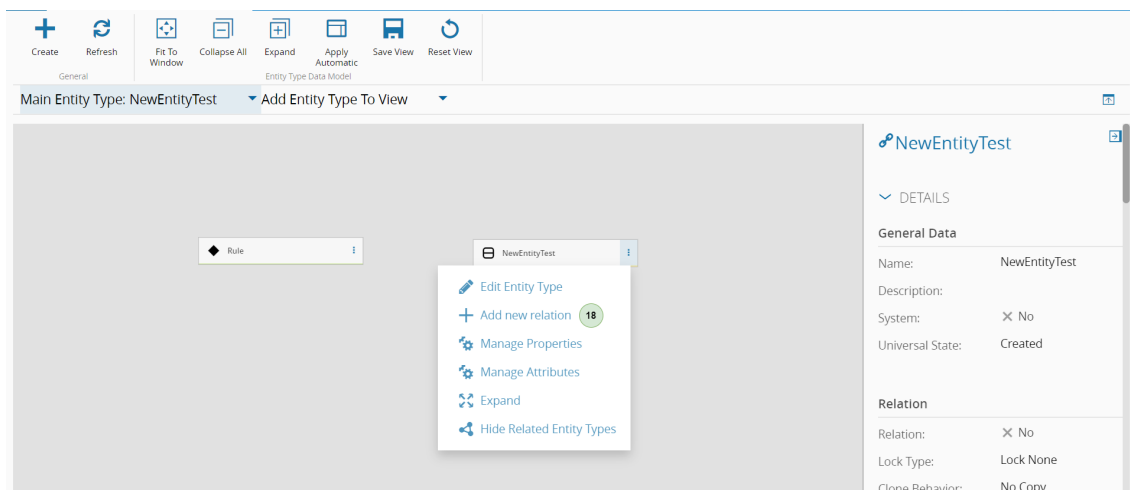


Figure 4.7: "Add New Relation" Option in "Created" Entity Type

In order to connect to another entity type, first, a user must display in the canvas the entity type to be referenced using the "Add entity type to view" dropdown (Figure 4.4, Annotation number 17) . After that, the user must name the new foreign key and choose one of the ports of the newly created entity type property and then connect it anywhere in the other entity type. An entity type can reference itself. For this purpose, simply connect the link from the new entity type property to the entity type itself.

The new property item added contains the following buttons (numbered in Figure 4.8):

- Add the new entity type foreign key (19): Check for the validity of the new foreign key by checking its name and the link created. If the name given is invalid, the text box will appear with a red outline (Figure A.16), and if the link is incorrect or nonexistent, the circle ports will appear red (Figure A.17). If both of them are valid, the new foreign key will be added, and the view will be updated.
- Delete the new entity type foreign key (20): Delete the new foreign key form from the list of properties.

**Remove a relation:** A user can delete any of the relations present in an entity type in the "Created" state. By right-clicking on the relation, the option to remove the relation is presented (Figure 4.9).

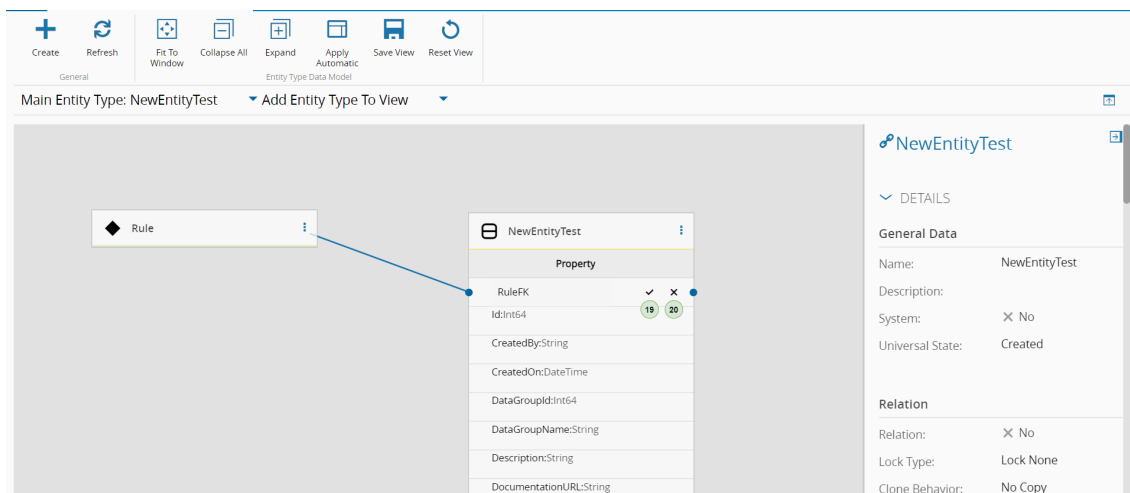


Figure 4.8: New foreign key form

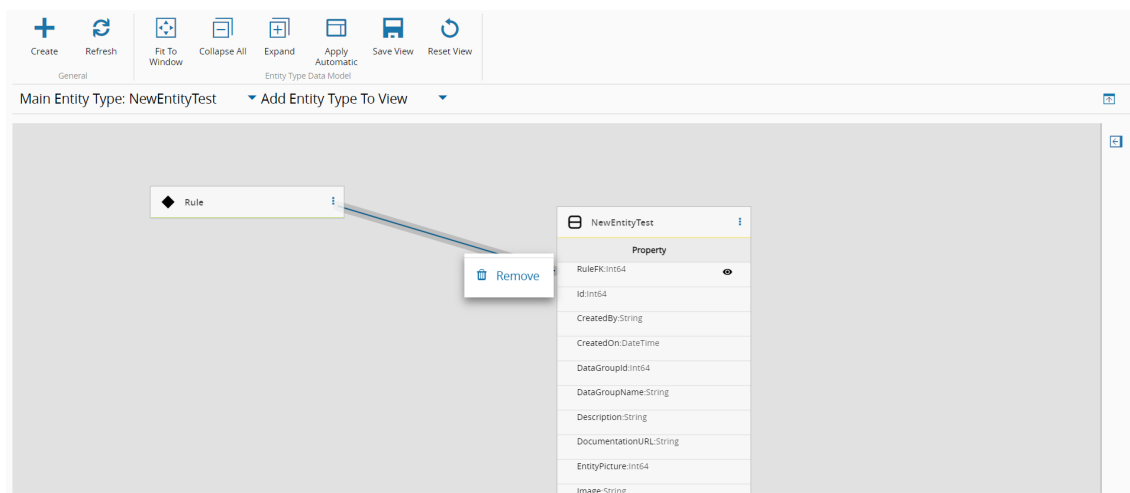


Figure 4.9: Remove relation option after right-click on link

#### 4.4.2.4 Manage Properties and Attributes

The remaining buttons in the header dropdown refer to other possible actions to perform in each entity type (numbered in Figure 4.4):

- Manage Properties (10): Opens the "Manage Properties" wizard of that specific entity type, which allows users to add, remove and edit the properties or custom properties of the entity. Properties can be added when an entity type is in the "Created" universal state, and custom properties in the "Active" state (Figure A.3).

- **Manage Attributes (11):** Opens the "Manage Attributes" wizard of that specific entity type, which allows users to add, remove and edit the attributes and custom properties of the entity. The option to either add an attribute or custom property appears only when the entity type is in the "Active" state, otherwise only attributes can be added. Only appears in entity types that allow attributes (Figure A.4).

These buttons all pop up wizards that were already present in the original entity type page of the CMF. The changes are directly reflected in the persistence layer, but the canvas may need to be refreshed (by clicking the "Refresh" button) to display those changes. These changes can also be displayed by clicking "Reset View" or refreshing the page.

## 4.5 Implementation Challenges and Constraints

One of the main challenges in the development process was the fact that many of the API calls were asynchronous. This created some issues since each relation depends on other entities already being created and displayed, and the positioning of a given node depends on the other nodes also. This was solved by awaiting for the end of a given promise to continue to the next, which does decrease performance but ensured the correct loading of the data model.

Another issue was the slowness caused by the API calls that retrieved the information needed to display and the connection with JointJS when adding a new relation, which often consists of an entity type and a link. This is crucial since it affects user experience. That added to the fact that some entities have many relations (for example, "Calendar" with 519 related entities) can cause serious performance issues and result in a significant load time.

In terms of visualization, the choice to only display one entity type view at a time came from the base MES having, as mentioned in Section 4.1, at the time of writing, more than 900 native entity types present. To view them all as one data model would greatly hinder understanding for the user and the performance of the visualization tool.

In terms of the edition, many constraints arose derived from the way the CMF is designed and what it allows a user to do. The creation of a new entity type entails the user choosing many flags and other information. Since some of these are fixed after creation and cannot be edited later, it renders simply creating an empty node and adding its name and properties, as seen in Chapter 3, redundant since the user will always have to resort to using the "Create Entity Type" wizard or only be able to create entities with the default settings (Figure A.1).

Another challenge arose when deciding how to add new properties, custom properties or attributes to an entity. As observed in Chapter 3, most data model editing tools chose to dynamically add new properties in the node of the canvas itself, and that would later reflect in the data model and persistence layers. This would include the property name and its type. In the CMF, each property or attribute addition entails filling in the name of the property, the reference type, and, in most options, another field. This would greatly overcrowd and complicate the user interface when creating a new property/attribute/custom property since all these options would have to be

displayed in the node body. This led to the decision to add the "Manage Properties" and "Manage attributes" options in the entity type header dropdown button (Figures [A.3](#), [A.4](#)).



## Chapter 5

# Validation

In this chapter, the validation process is described. The validation is divided into two sections, validation testing, and a usability study. The usability study is divided into two parts, the objectives and methodology, and results and discussion.

### 5.1 Validation Testing

For validation testing, a list of test cases is presented per requirement, and each test case is coupled with one or more figures that show examples of the results (See Table 5.1).

In Table 5.1, each of the functional requirements presented in Section 4.1 is presented with its identifier and a short description. For each requirement, a list of test cases is shown with the format: TC(Requirement number).(Test Case number). Moreover, a figure serves as evidence of the results for each test case.

Table 5.1: Validation testing for each requirement

Requirement / Feature	Test cases	Test results / evidences
R1. Visualize data model, based on ER paradigm.	TC1.1 Select main entity type and check if a data model appears based on ER paradigm.	Figure 4.4
	TC1.2 Check that the relations are connected to their correct property (primary or foreign key) and have symbols that correctly represent the different types of relations.	Figure 4.4
	TC1.3 Check if nodes have a header with entity type name and body with the correct entity type properties, custom properties, and attributes.	Figure 4.4
R2. Allow manipulation of data model elements	TC2.1 Check that nodes can be moved.	Figure 4.6

Table 5.1: Validation testing for each requirement

Requirement / Feature	Test cases	Test results / evidences
	TC2.2 Check that vertices can be added to links without changing the properties it is connected to or the type of relation.	Figure 4.6
R3. Show data model from a single main entity type	TC3.1 Select the main entity type and check that the data model displayed in the canvas corresponds to the data model of that selected entity type.	Figure A.5
	TC3.2 Check that the default data model of a given entity type consists of only that entity type and its direct relations, with the correct symbols.	Figure A.5
R4. Default Layout	TC4.1 Check that entities appear by default in the default layout.	Figure A.5
	TC4.2 Click "Apply Automatic Layout" and check if entities return to the original position.	Figure A.5
R5. Expand/Collapse node(s)	TC5.1 Check that entities appear by default collapsed.	Figure A.5
	TC5.2 Expand an entity and check that all properties and attributes are shown and that all links connect to the correct properties (primary or foreign keys).	Figure 4.4
	TC5.3 Collapse the entity and check that the entity becomes properly collapsed.	Figure A.5
	TC5.4 Expand all entities and check the result.	Figure A.6
	TC5.5 Collapse all entities and check the result.	Figure A.5
R6. Access the original entity type page in CMF	TC6.1 Select the entity type link and check if it redirects to the original entity type page.	Figure A.8
R9. Show/Hide Relations of an entity type	TC9.1 Select an entity type node whose relations are not already displayed and click "Show Related Entity Types". Check if the relations appear and are displayed correctly, and the button switches to "Hide Related Entity Types".	Figure A.13
	TC9.2 Select an entity type whose relations are displayed and click "Hide Related Entity Types". Check if the relations are hidden correctly, not hiding the ones that were present before showing the related entity types, and the button changes to "Show Related Entity Types".	Figure 4.4

Table 5.1: Validation testing for each requirement

Requirement / Feature	Test cases	Test results / evidences
R10. Show/Hide Relations by property	TC10.1 Check if the properties with relations associated have the "eye" button.	Figure 4.4
	TC10.2 Select an entity type property previously hidden and click on the "eye" button. Check if the associated relations which were previously hidden appear and the "eye" on both sides of each relation changes state.	Figure 4.4
	TC10.3 Select an entity type property with at least a relation associated and click the "eye" button. Check if the associated relation is hidden and the "eye" on both sides of each relation changes state.	Figure A.12
R11. Creation of new entity type	TC11.1 Click the "Create" action button and check if the "Create New Entity Type" wizard pops up.	Figure A.1
	TC11.2 After filling out the necessary fields, click in "Create" button on the page and check if the entity type was created correctly.	Figure A.7
R12. Edit an entity type	TC12.1 On the header of an entity type node, click in "Edit Entity Type" button. Check if the "Edit Entity Type" wizard appears.	Figure A.2
	TC12.2 After editing the necessary fields of the entity type, click on "Save". Check if the edition was done correctly and is reflected in the side panel.	Figures: A.14, A.15
R13. Adding/Removing properties and custom properties	TC13.1 Click in "Manage Properties" of an entity type node. Check if the "Manage Entity Type Properties" wizard appears with the right entity type.	Figure A.3
	TC13.2 Add/remove the desired properties (if entity type is in the "Created" state) or custom properties (if entity type is in the "Active" state). Click on "Update" to check if those changes are reflected in the data model after refresh and in the original entity type page.	Figure A.19
R14. Adding/Removing attributes and custom properties	TC14.1 Check if the entity types allowed to have attributes have the "Manage Attributes" button in the dropdown.	Figure 4.4

Table 5.1: Validation testing for each requirement

Requirement / Feature	Test cases	Test results / evidences
	TC14.2 Click in "Manage Attributes" of an entity type node. Check if the "Manage Entity Type Attributes" wizard appears with the right entity type.	Figure A.4
	TC14.3 Add/remove the desired custom properties (only if entity type is in "Active" state) and attributes. Click on "Update" to check if those changes are reflected in the data model after refresh and in the original entity type page.	Figure A.20
R15. Distinguish "Created" and "Active" entity types	TC15.1 Check if the "Active" entity types appear with a green border color and the "Created" ones appear with a yellow/gold border color.	Figure 4.9
R16. Distinguish "normal" and "relation" entity types	TC16.1 Check if the "relation" entity types are marked with an "(R)" after the name (in the node header).	Figure 4.4
R17. Visually add/remove relations from "Created" entity type	TC17.1 Check if the "Created" entity types have the "Add New Relation" button in the dropdown of the node.	Figure 4.7
	TC17.2 After clicking "Add New Relation", check if a new list element appears with two connecting dots for the link and a text area to fill with a foreign key name.	Figure 4.8
	TC17.3 If the link of the new relation is non-existent or invalid, check if the connect dots turn red.	Figure A.17
	TC17.4 If the name of the new foreign key is not valid, check if the textbox's border turn red.	Figure A.16
	TC17.5 If the link and foreign key name are valid, click the "tick" button. Check if a new foreign key property is added to the entity type, referencing the correct entity type, and is also added to the original page.	Figure 4.9
	TC17.6 Click the "cross" button. Check if the list element for the new relation is deleted along with any potencial link already created.	Figure A.18
	TC17.7 Right-click on a relation from a "Created" entity type. Check if the remove option appears.	Figure 4.9

Table 5.1: Validation testing for each requirement

Requirement / Feature	Test cases	Test results / evidences
	TC17.8 Click on remove option. Check if the relation is deleted (link and property).	Figure A.18
R18. Save current view by main entity type	TC18.1 Click the "Save View" button. Check if the view was saved, signaled by the success notification ("Current View Saved").	Figure A.9
R19. Open saved view by main entity type	TC19.1 Select an entity type that has a saved view. Check if the saved view is displayed.	Figure A.10
R20. Reset saved view by main entity type	TC20.1 Click the "Reset View" action button. Check if the view is reset to the default.	Figure A.5
	TC20.2 When refreshing the page or selecting an entity type without a saved view, check if the data model displayed is the default for that entity type.	Figure A.5
R21. Refresh data model by entity type	TC21.1 Click the "Refresh" action button. Check if the data model has been refreshed but maintained the same node positions and hidden properties.	Figure A.11
R22: Add entity type to view	TC22.1 Click the "Add Entity Type to View" button and select an entity type not already displayed. Check if that entity type appears in the canvas.	Figure 4.7

## 5.2 Usability Study

For the usability study, a survey that evaluates the end users' opinions upon partaking in testing the final solution was done. Therefore, this section will be divided into "Objective and Methodology" and "Results and Discussion".

The first section will focus on what type of user was chosen, the testing environment and what resources were given to the users to complete the experiment, and a description of the survey provided.

The latter section will focus on what was gathered from the survey and how that reflects in the final solution to this dissertation.

### 5.2.1 Objectives and Methodology

The pipeline for pull requests in Microsoft DevOps is set up to build and provide an environment for testing the solution automatically.

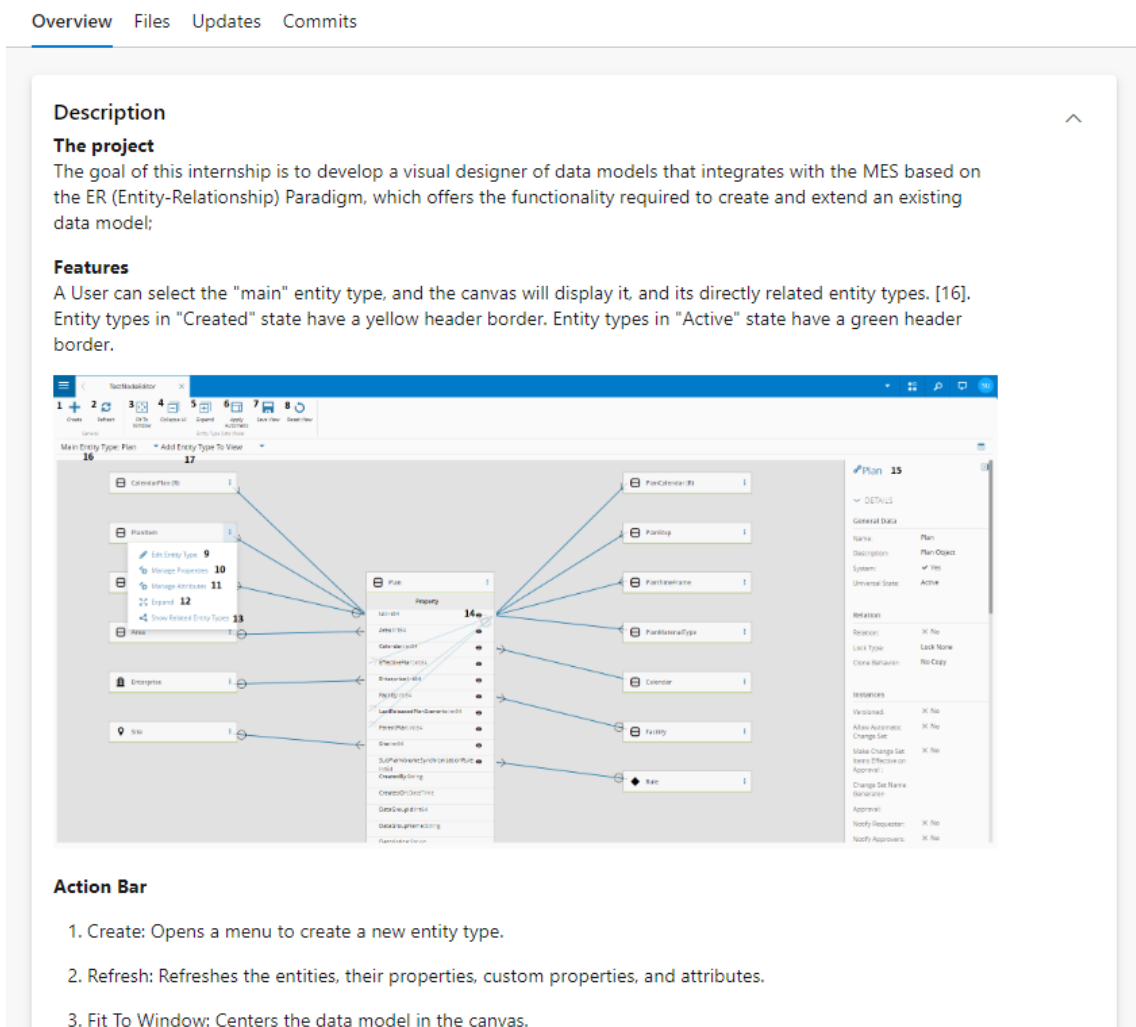


Figure 5.1: Excerpt of the pull request instructions page

The pull request and README were set up with a brief description of the objective of this dissertation, along with a list of features and figures to demonstrate how they work and how each button is associated with each feature. Essentially, a summarized and simplified version of Section 4.4 (Figure 5.1).

At the end of these instructions, there is a link to the environment for testing and a link to the survey. This survey consists of a System Usability Scale (SUS) [18] and the question, "Do you have suggestions for possible improvements or added features to the created solution?".

The System Usability Scale is a standard survey consisting of 10 questions to evaluate the usability of the final solution, measured using a quantitative method of questions from "Strongly Agree" to "Strongly Disagree". The SUS questions are as follows:

- SU01: I think that I would like to use this system frequently.
- SU02: I found the system unnecessarily complex.

- SU03: I thought the system was easy to use.
- SU04: I think that I would need the support of a technical person to be able to use this system.
- SU05: I found the various functions in this system were well integrated.
- SU06: I thought there was too much inconsistency in this system.
- SU07: I would imagine that most people would learn to use this system very quickly.
- SU08: I found the system very cumbersome to use.
- SU09: I felt very confident using the system.
- SU10: I needed to learn a lot of things before I could get going with this system.

To convert the SUS survey to it's standard score (0-100) the following equation was applied to the results:

$$SUS = 2.5(20 + SUM(SU01, SU03, SU05, SU07, SU09) - SUM(SU02, SU04, SU06, SU08, SU10))$$

Since the questions are alternated between positive-tone and negative-tone, this division in sums is necessary. The odd-numbered questions are the positive ones, and the even-numbered the negative-tone ones. Research has been done to better determine the meaning of the final score, but overall a score of 68 is considered average [18].

The people chosen to experiment with the final solution and answer the survey were from the deployment services team at Critical Manufacturing. Seeing as the deployment services team would use this visual designer the most, as they are the primary individuals to modify the customized data models, they proved to be the ones more qualified to answer the survey.

## 5.2.2 Results and Discussion

### 5.2.2.1 Results

The survey counted 5 participants. Their answers to the SUS survey are shown in Table 5.2. A calculation of the overall score of the responses will be made along with a discussion of the results and the participants' suggestions.

Table 5.2: Participants response to SUS survey

SUS Question	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Global Average
SU01	4	3	4	2	2	3
SU02	2	1	1	4	2	2

SU03	4	2	3	3	4	3.2
SU04	4	2	2	2	3	2.6
SU05	4	2	4	3	5	3.6
SU06	1	1	2	3	2	1.8
SU07	3	5	5	4	3	4
SU08	2	3	2	2	2	2.2
SU09	4	5	3	4	4	4
SU10	1	2	1	2	3	1.8
SUS score (0-100)	72.5	70	77.5	57.5	65	68.5

Participants 1, 2, and 4 filled out the last question, where suggestions for the final solution were to be made.

One suggestion all participants made was the need to clarify what each relation type meant. This information regarding the types of relations, and an image exemplifying it, was later added to the pull request description and README file. These suggestions could also be a future addition to the final solution, where, upon hovering, an indication of the type of relation is specified in the canvas.

Participant 1 made the following suggestions:

- Add the ability to move the symbols representing each relation to facilitate visualization.
- The contrast between "Created" and "Active" entity types should be more noticeable.
- Add the ability to remove an entity type from view.
- Dropdown buttons in alphabetical order.
- Remove entity types that are only relations (join tables that exist between many-to-many relations) from dropdowns.

Participant 2 made the following suggestions:

- Give the relationships "names", for example, "Belongs To" and "Owns".
- Place the main attributes/properties inside circles outside the entity.
- Show information about the primary key.

Participant 3 made the following suggestions:

- Add the ability to move the symbols representing each relation to facilitate visualization.
- The contrast between "Created" and "Active" entity types should be more noticeable.
- Zoom scroll is too slow.



- "Refresh" button should be the first element on the action bar.
- New entity type not appearing in Main Entity Type dropdown after creation.

#### 5.2.2.2 Discussion

In terms of the SUS survey, the participants average score amounts to 68.5, considered an average rating.

The participants were torn in terms of using the system frequently. Most of them reported that the various functions in the system were well integrated, that there was not too much inconsistency in the system, and that they would imagine that most people would learn to use this system very quickly. The majority also reported that they did not find it cumbersome to use, felt confident using the system, and did not need to learn a lot to use it. While the majority reported that the system was not unnecessarily complex and that they would not need the support of a technical person to use it, they were torn when it came to thinking that the system was easy to use.

A bigger sample size of participants or a more selective one might see the SUS score average change, increasing the score's reliability. Since the participants were anonymous, there was no knowledge of what type of work they performed within the development services, which could skew their opinion of how frequently they would use this tool. More contact with the participants during the trial phase could also help understand their views on the system's usability.

A critical addition to the solution mentioned by the participants was the ability to remove an entity type from view. This feature will be added in future work. A more significant visual contrast between the entity types "Created" and "Active" will also be added, as will the dropdown button being in alphabetical order.

Showing information about the primary key can be added to future work. However, since the primary key is automatically created and cannot be changed ("Id"), this was not considered a priority when developing the project.

The ability to give each relation an annotation that better represents the type of relation could also be added. For example, as stated in the suggestion, an annotation saying that a relation is of type "Belong To" or "Owns". This feature is not uncommon in data models and would prove helpful to a better understanding of a data model and for this visualization to be even more informative.

The main entity type not appearing in the dropdown is due to the fact that the page needs to be refreshed for it to appear. The tutorial mentioned this. However, it could be mentioned on the page to be more intuitive for the user.

The slowness in the zoom to scroll is theorized to be due to the system's slowness when confronted with an extensive data model. In typical cases, this did not prove to be an issue.

The position of the action button in the action bar was according to other pages in the Critical Manufacturing Framework.

The ability to move the symbols was not very clear. Vertices can be added to links to allow changing their shape, moving the symbols according to this. This feature was described in the tutorial.

Removing the entity types that are only relations could be added. However, since the data models can be expanded by showing the relations of any entity type in the view, this did not seem necessary. Nevertheless, a distinction in the dropdown of entity types that are relations and those that are not could be made.

The ability to place specific attributes and properties outside the entity could be helpful in terms of visualization but would add an extra level of complexity to the final solution and hinder the easy visualization of the data model.

Overall, the usability study results were promising. The resulting score of the SUS is considered average and was lowered mainly due to the ease-of-use aspect, and not all users considered this a tool they would frequently use. The end-users opinions also pointed to some critical factors and additions to the final solution that could significantly increase its usefulness.

## Chapter 6

# Conclusions and Future Work

### 6.1 Conclusions

In order to facilitate the development of the customized MES versions for each client, teams are set up in CM to complete this task. For these teams, the ability to quickly visualize, edit and show the client how each entity type interacts and associates with others is essential. On the other hand, these same features could be helpful to the final client since they are very visual and easily understandable. It is from this necessity that this dissertation project was developed.

The proposed solution resulted in an easy-to-use tool that enabled the visualization of the relations of each entity type according to the ER paradigm, allowing further customization and deepening of the view. When it was logical and possible, the visual designer also made the edition process easier for the end-user and less complex by making it more visual. These characteristics made the tool successful in its goal since it could fulfill the needs presented for this dissertation.

Overall, the objectives set for this dissertation were accomplished. A visual designer that allowed both visualization and edition of data models was developed successfully. The visualization aspect of data models was more emphasized in the final solution due to the limitations in edition caused by the CMF.

The final solution reflected the goals set for this dissertation, and the validation testing corroborated this. In the usability study, most end-users found the final solution promising, although they were torn about whether they would give much use to it. The participants also contributed valuable insight into improvements to the final solution.

### 6.2 Future Work

In the future, an emphasis on performance would be crucial for improving the user experience. This modification could be done by updating it to a different framework.

Since this performance issue is present mainly when displaying an entity type with many relations, a solution could be to allow the user to change the entity type mid-process or warn the user before it starts the displaying process. Another option would be to display a fraction of the

relations of the entity type. Considering that most of the relations are references to that entity type, an alternative would be to hide by default the relations associated with the main entity type Id and only show the relations associated with the foreign keys.

Another possible improvement would be on the automatic layout method. An option for the user to choose between some pre-defined ones that are more efficient or more visually appealing would be of great value.

The remarks made by participants in Section 5.2.2, such as the ability to remove an entity type present in the canvas, to add annotations to each relation, and other minor fixes would also be necessary in future evolution of the solution.

A final thought would be to add the option to view the data model on the original CMF entity type page, as it could be easily accessible and more intuitive to navigate to the data model.

# References

- [1] Association Properties - Studio Pro 9 Guide. <https://docs.mendix.com/refguide/association-properties>. Last accessed 23 February 2022.
- [2] Associations - Studio Pro 9 Guide. <https://docs.mendix.com/refguide/associations>. Last accessed 23 February 2022.
- [3] Attributes - Studio Pro 9 Guide. <https://docs.mendix.com/refguide/attributes>. Last accessed 23 February 2022.
- [4] Create a Basic Data Layer - Studio Pro 9 How-to's. <https://docs.mendix.com/howto/data-models/create-a-basic-data-layer>. Last accessed 23 February 2022.
- [5] Data Modeling: Conceptual vs Logical vs Physical Data Model. <https://online.visual-paradigm.com/knowledge/visual-modeling/conceptual-vs-logical-vs-physical-data-model>. Last accessed 29 February 2022.
- [6] Entities - Studio Pro 9 Guide. <https://docs.mendix.com/refguide/entities>. Last accessed 23 February 2022.
- [7] Entity Relationships - OutSystems. [https://success.outsystems.com/Documentation/11/Developing\\_an\\_Application/Use\\_Data/Data\\_Modeling/Entity\\_Relationships](https://success.outsystems.com/Documentation/11/Developing_an_Application/Use_Data/Data_Modeling/Entity_Relationships). Last accessed 21 February 2022.
- [8] Execute an SQL Statement on an External Database - Studio Pro 9 How-to's | Mendix Documentation. <https://docs.mendix.com/howto/integration/execute-an-sql-statement-on-an-external-database>. Last accessed 24 February 2022.
- [9] ICT specialists - statistics on hard-to-fill vacancies in enterprises. [https://ec.europa.eu/eurostat/statistics-explained/index.php?title=ICT\\_specialists\\_-\\_statistics\\_on\\_hard-to-fill\\_vacancies\\_in\\_enterprises](https://ec.europa.eu/eurostat/statistics-explained/index.php?title=ICT_specialists_-_statistics_on_hard-to-fill_vacancies_in_enterprises). Last accessed 15 February 2022.
- [10] Jointjs. <https://www.jointjs.com/>. Last accessed 29 June 2022.
- [11] Salesforce Object Query Language (SOQL) | SOQL and SOSL Reference | Salesforce Developers. [https://developer.salesforce.com/docs/atlas.en-us.soql\\_sosl.meta/soql\\_sosl/sforce\\_api\\_calls\\_soql.htm](https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql.htm). Last accessed 24 February 2022.

- [12] Using SQL to access data | Evaluation Guide. <https://www.outsystems.com/evaluation-guide/using-sql-to-access-data/>. Last accessed 24 February 2022.
- [13] Create an Entity to Persist Data - OutSystems. [https://success.outsystems.com/Documentation/11/Developing\\_an\\_Application/Use\\_Data/Data\\_Modeling/Create\\_an\\_Entity\\_to\\_Persist\\_Data](https://success.outsystems.com/Documentation/11/Developing_an_Application/Use_Data/Data_Modeling/Create_an_Entity_to_Persist_Data), July 2018. Last accessed 22 February 2022.
- [14] Read/Write Data One-to-Many - OutSystems. [https://success.outsystems.com/Documentation/11/Developing\\_an\\_Application/Use\\_Data/Offline/Offline\\_Data\\_Sync\\_Patterns/Read%2F%2FWrite\\_Data\\_One-to-Many](https://success.outsystems.com/Documentation/11/Developing_an_Application/Use_Data/Offline/Offline_Data_Sync_Patterns/Read%2F%2FWrite_Data_One-to-Many), July 2018. Last accessed 21 February 2022.
- [15] Sikha Bagui and Richard Earp. *Database design using entity-relationship diagrams*. Auerbach Publications, 2003.
- [16] Dalibor Berić, Darko Stefanović, Bojan Lalić, and Ilija Ćosić. The implementation of erp and mes systems as a support to industrial management systems. *Int. J. Ind. Eng. Manag.*, 9(2):77–86, 2018.
- [17] Alexander C Bock and Ulrich Frank. In search of the essence of low-code: an exploratory study of seven development platforms. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 57–66. IEEE, 2021.
- [18] James R Lewis. The system usability scale: past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018.
- [19] Research and Markets. Global \$187 Billion Low-Code Development Platform Market to 2030. <https://www.globenewswire.com/news-release/2020/11/10/2123468/28124/en/Global-187-Billion-Low-Code-Development-Platform-Market-to-2030.html>, November 2020. Last accessed 20 February 2022.
- [20] Michael McClellan. *Applying manufacturing execution systems*. CRC Press, 1997.
- [21] Heiko Meyer, Franz Fuchs, and Klaus Thiel, editors. *Manufacturing execution systems: optimal design, planning, and deployment*. McGraw-Hill, New York, 2009.
- [22] International Society of Automation. Isa95, enterprise-control system integration. <https://www.isa.org/standards-and-publications/isa-standards/isa-standards-committees/isa95>. Last accessed 29 June 2022.
- [23] Clay Richardson, John R Rymer, Christopher Mines, Alex Cullen, and Dominique Whittaker. New development platforms emerge for customer-facing applications. *Forrester: Cambridge, MA, USA*, 15, 2014.
- [24] Christian Hestermann Rick Franzosa. Gartner magic quadrant for manufacturing execution systems. *Gartner report*, May 2022.
- [25] B Saenz de Ugarte, A Artiba, and R Pellerin. Manufacturing execution system—a literature review. *Production planning and control*, 20(6):525–539, 2009.

- [26] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE, 2020.
- [27] SAP. *Data Architecture: SAP PowerDesigner Documentation Collection*, December 2019.
- [28] G. Tillmann. *Usage-Driven Database Design: From Logical Data Modeling through Physical Schema Definition*. Apress, 2017.
- [29] Thomas Van Raalte. Oracle retail data model implementation and operations guide, release 11.3.
- [30] Robert Waszkowski. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381, 2019.
- [31] Bing-hai Zhou, Shi-jin Wang, and Li-feng Xi. Data model design for manufacturing execution system. *Journal of Manufacturing Technology Management*, 2005.

# Appendix A

# Appendix

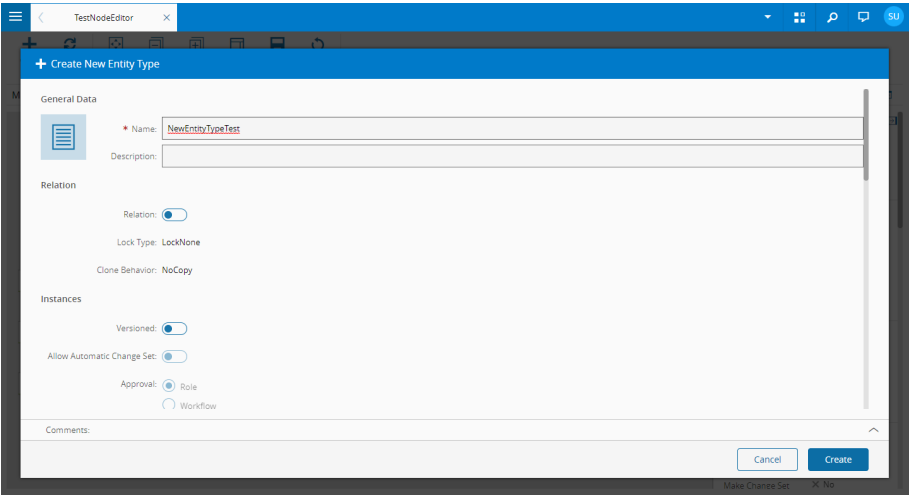


Figure A.1: Create New Entity MES Screen

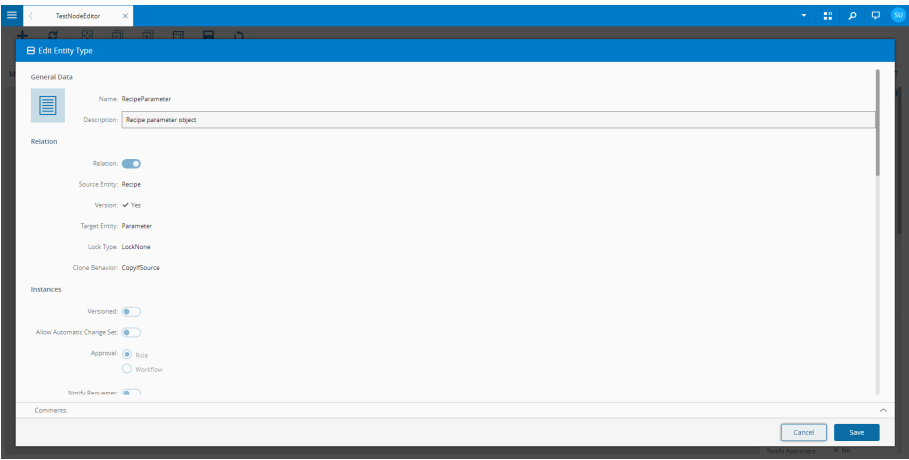


Figure A.2: Edit Entity MES Screen



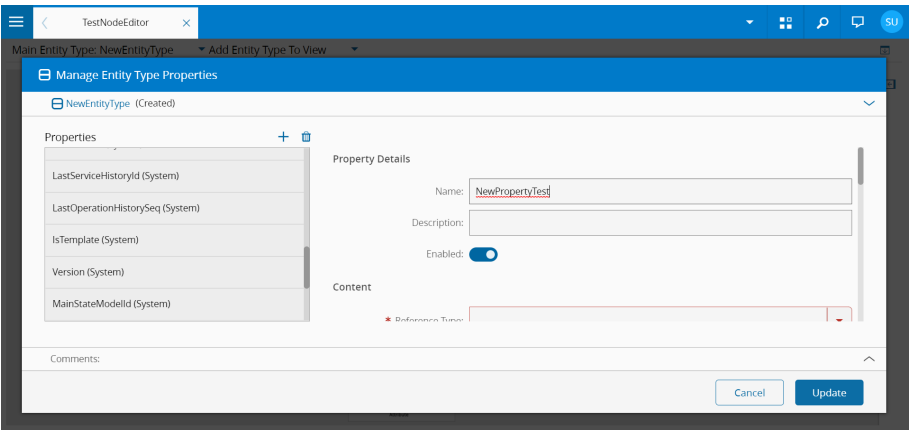


Figure A.3: Manage Entity Type Properties MES Screen

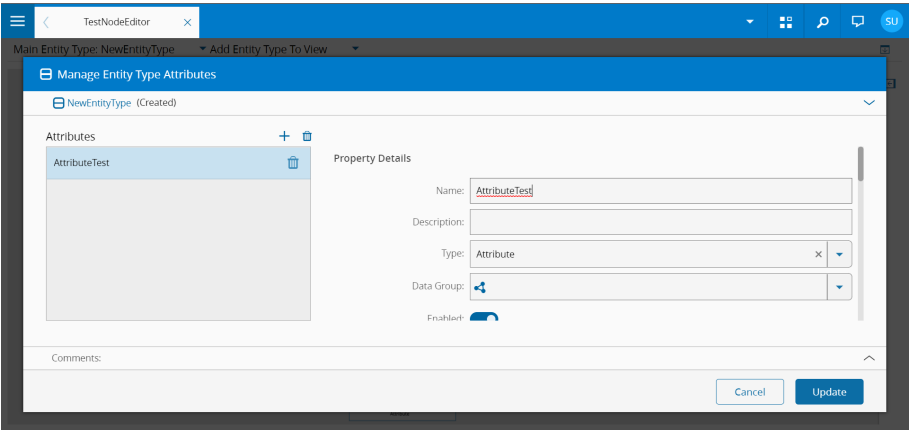


Figure A.4: Manage Entity Type Attributes MES Screen

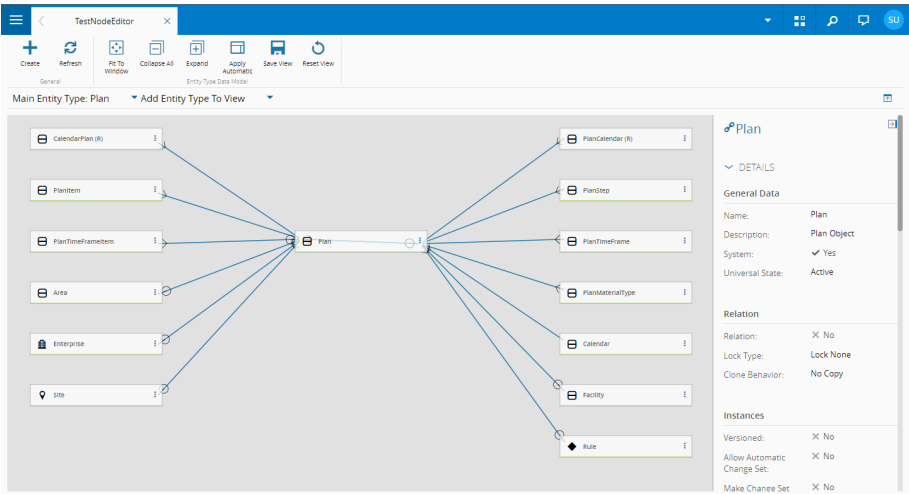


Figure A.5: Default Data Model Visualization, with All Entities Collapsed

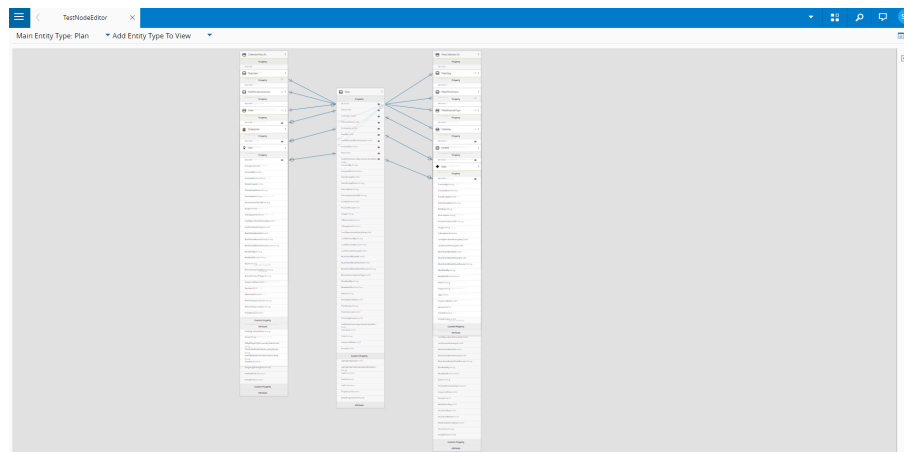


Figure A.6: Data Model Visualization, with All Entities Expanded

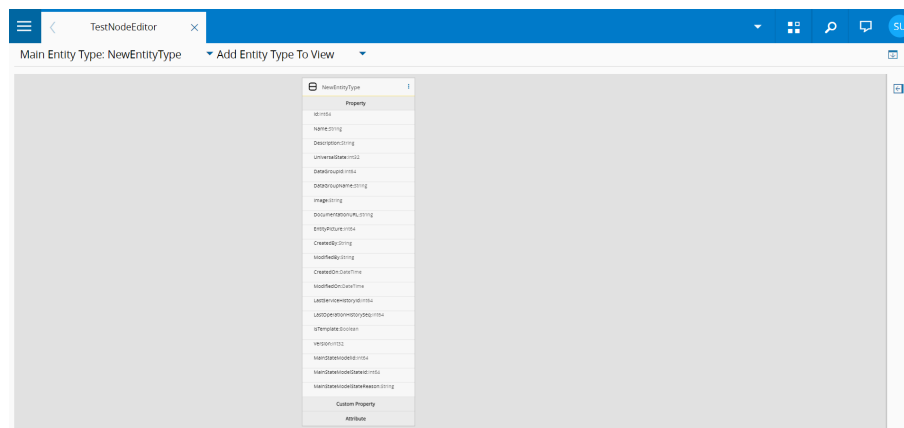


Figure A.7: Data Model Visualization with New Entity Type in "Created" state as Main Entity Type

TestNodeEditor

Plan

Refresh

Edit

Generate

General

Schema

Plan (Active)

DETAILS

General Data

Name: Plan

Description: Plan Object

System: ☒ Yes

Universal State: Active

Relation

Relation: ☒ No

Lock Type: Lock None

Clone Behavior: No Copy

Instances

Versioned: ☒ No

Allow Automatic Change Set: ☒ No

Make Change Set Items: ☒ No

Effective on Approval: ☒ No

Change Set Name Generator: Approval:

Notify Requester: ☒ No

Notify Approver: ☒ No

Use Change Set Approval: ☒ No

Context: ☒ No

Enable Automatic Approval: ☒ No

Display Change Set on Clone: ☒ No

Allow Delete Instances: ☒ Yes

Enable Default Revision Dates: ☒ No

Enable Version Creation: ☒ Yes

STATE MODELS

PROPERTIES

ATTRIBUTES

OPERATIONS

OPERATION ATTRIBUTES

Flags

Information

Figure A.8: Original page of "Plan" entity type

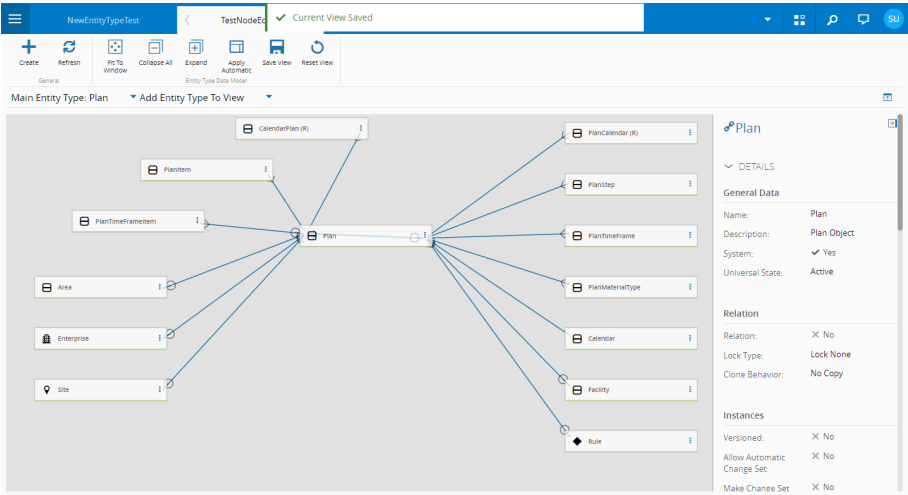


Figure A.9: Save current view of "Plan" data model

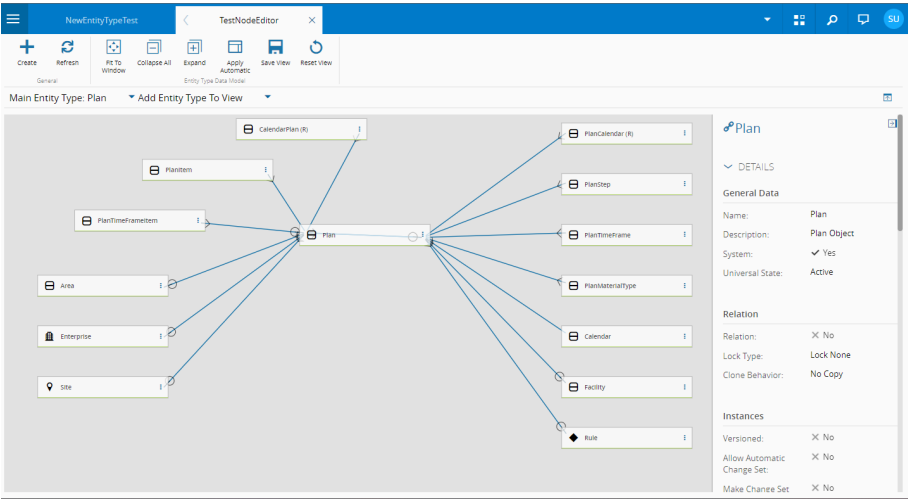


Figure A.10: Open saved view of "Plan" data model

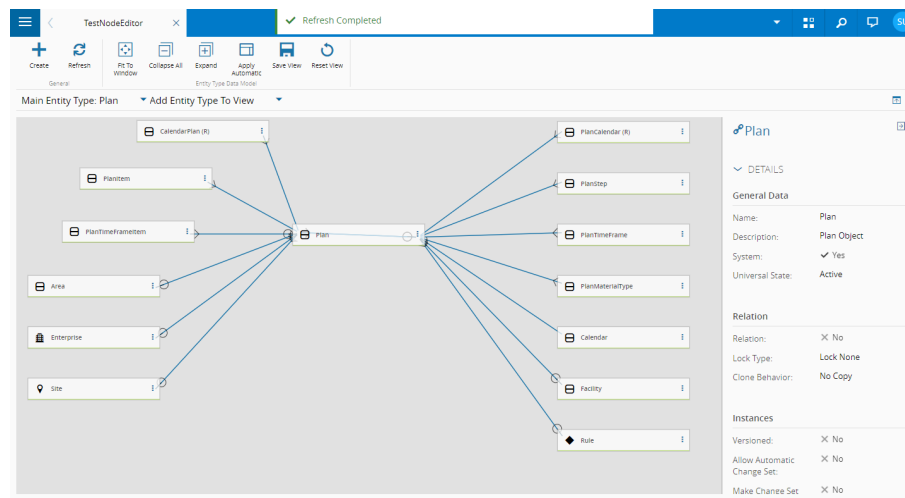


Figure A.11: Refresh "Plan" view

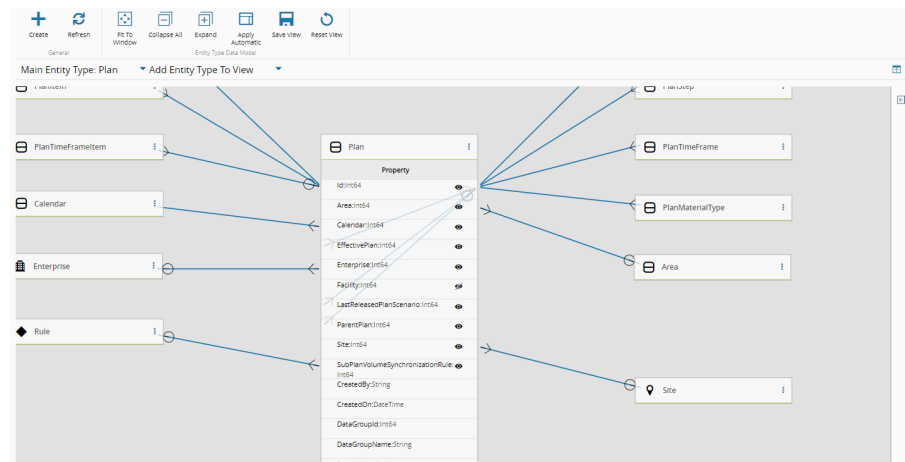


Figure A.12: Hide "Facility" relation from "Plan"

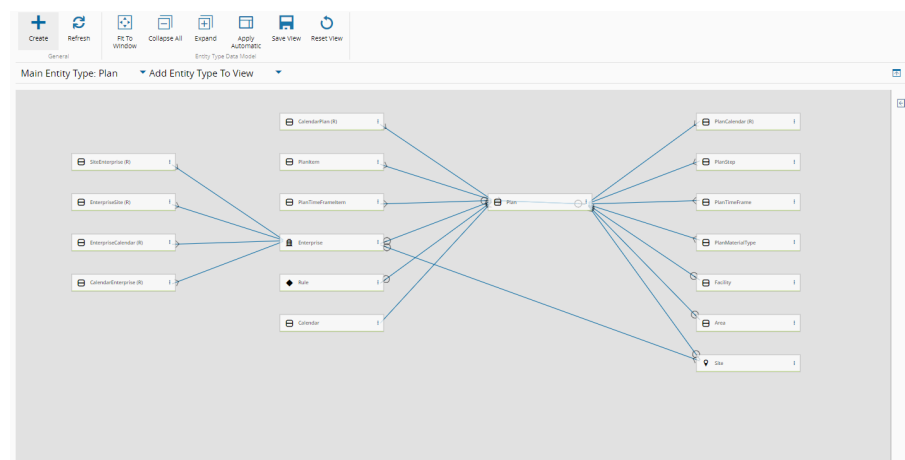


Figure A.13: Show related entity types of "Enterprise"

Plan

DETAILS

General Data

Name:

Plan

Description:

Plan Object After Edit

System:

✓ Yes

Universal State:

Active

Relation

Relation:

✗ No

Lock Type:

Lock None

Clone Behavior:

No Copy

Instances

Versioned:

✗ No

Allow Automatic Change Set:

✗ No

Make Change Set

✗ No

Figure A.14: "Plan" side panel after edit

Refresh

Edit

Generate

General

Schema

Plan (Active)

DETAILS

General Data

Name: Plan

Description: Plan Object After Edit

System: ✓ Yes

Universal State: Active

Relation

Relation: ✗ No

Lock Type: Lock None

Clone Behavior: No Copy

Instances

Versioned: ✗ No

Allow Automatic Change Set: ✗ No

Make Change Set Items: ✗ No

Effective on Approval:

Change Set Name Generator:

Approval:

Notify Requester: ✗ No

Notify Approver: ✗ No

Use Change Set Approval: ✗ No

Context:

Enable Automatic Approval: ✗ No

Display Change Set on Clone: ✗ No

Allow Delete Instances: ✓ Yes

Enable Default Revision Dates: ✗ No

Enable Version Creation: ✓ Yes

DETAILS

STATE MODELS

PROPERTIES

ATTRIBUTES

OPERATIONS

OPERATION ATTRIBUTES

Flags

Information

Figure A.15: "Plan" original page after edit

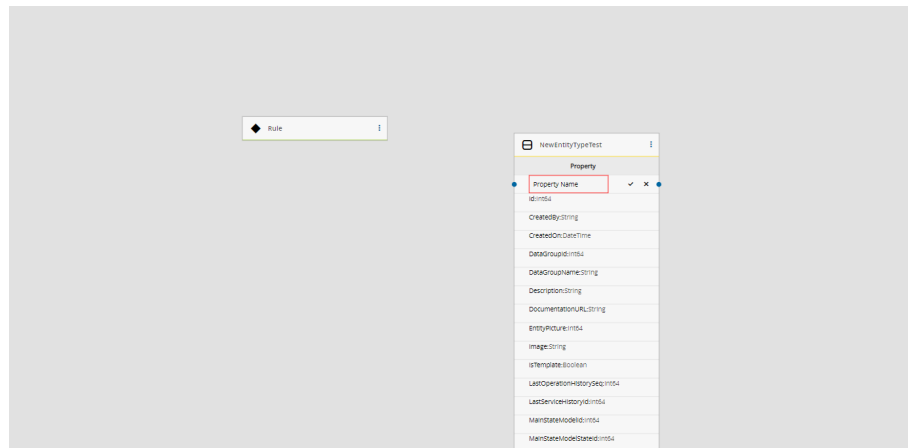


Figure A.16: New foreign key with invalid name

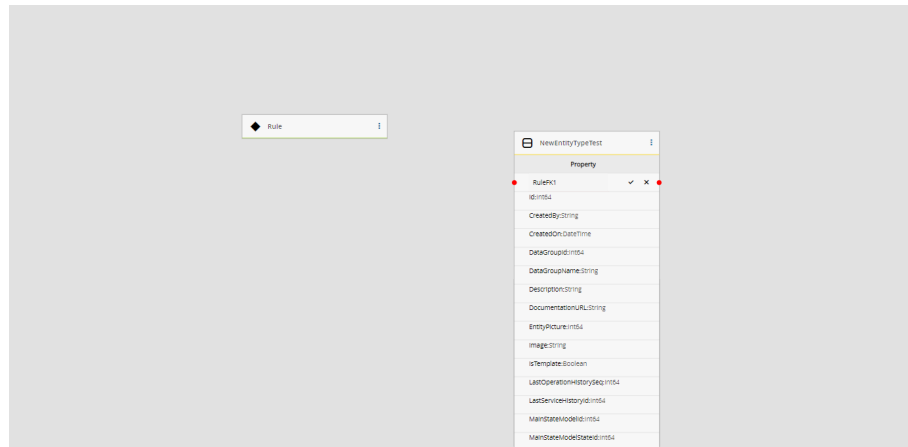


Figure A.17: New foreign key with invalid link

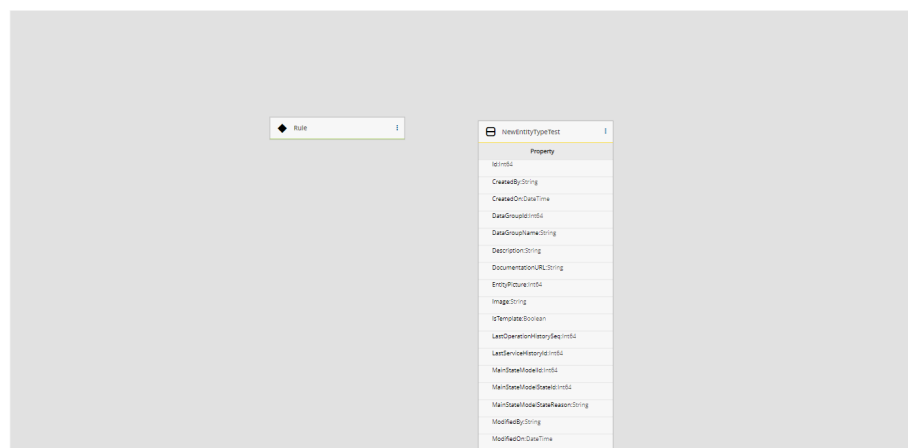


Figure A.18: After delete new foreign key element

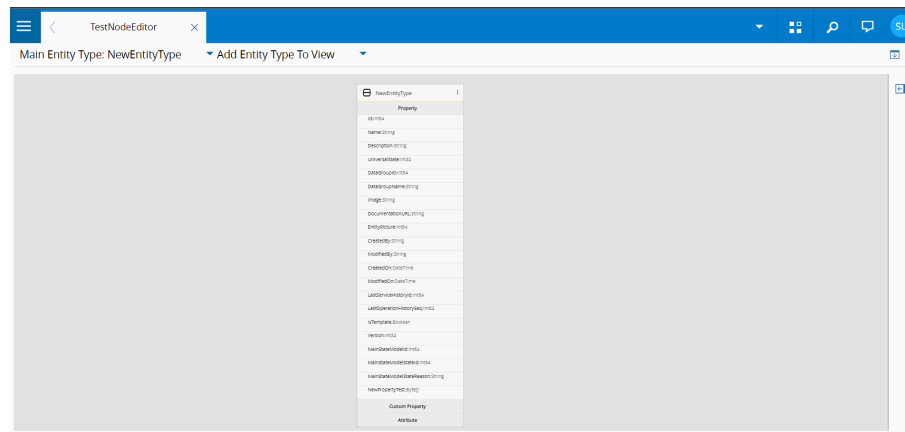


Figure A.19: Entity type after adding property

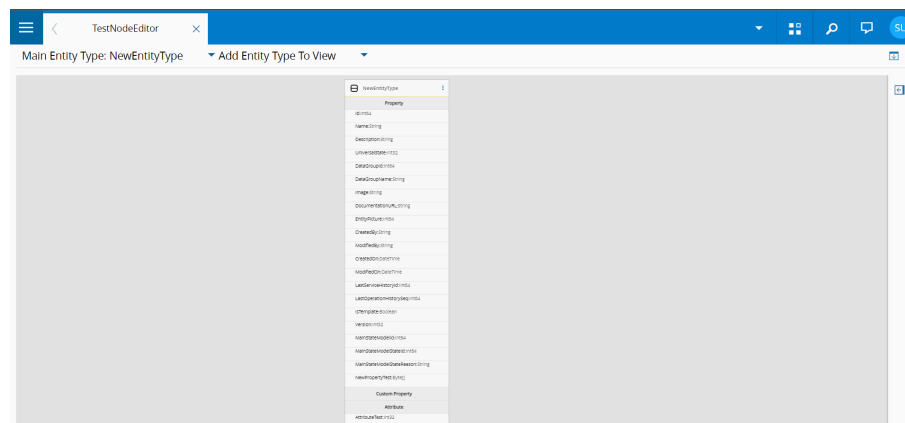


Figure A.20: Entity type after adding attribute