

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



# **Human-Computer Interaction in Smart Manufacturing Systems : reactive and adaptive UIs**

**Hugo Martins**

FINAL VERSION

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Gil Gonçalves

Second Supervisor: Paulo Martins

July 31, 2022



# Resumo

Desde a primeira revolução industrial no século dezoito, a indústria tem evoluído de forma exponencial. A evolução mais significativa está a ser vivida neste momento com os conceitos emergentes da Indústria 4.0 e dos novos Sistemas de Manufatura Inteligentes, os Smart Manufacturing Systems (SMS). Estes sistemas têm vários requisitos como a customização em massa, a automação, a flexibilidade, a adaptabilidade e transparência. Apesar dos requisitos dos SMS, continua a ser necessária a interação humana, no processo da gestão produção destes sistemas. Esta interação é maioritariamente feita através de Interfaces de Utilizador (User Interfaces - UI) que devem ser adaptativas e permitir satisfazer os requisitos do sistema.

Esta dissertação tem como objetivo desenvolver uma arquitetura que permita o fácil e rápido desenho destas Interfaces, para que possam ser utilizadas por utilizadores finais do SMS. Esta dissertação tem o objectivo de desenvolver uma arquitectura que permita um desenvolvimento fácil e rápido de UIs e a sua posterior geração para que os utilizadores finais do SMS possam aceder e manipular a informação do sistema. O desenho destas interfaces foi feito usando estruturas de dados baseadas em árvores, intituladas de Estruturas Genéricas (Generic Structures - GS). Estas estruturas genéricas foram modeladas para que fossem capazes de conter a informação necessária para gerar UIs. Esta arquitetura contém ainda informação das entidades e relacionamentos do Modelo Conceptual do SMS, o que permite que relações sejam estabelecidas entre esta informação e os nós das estruturas genéricas das Interfaces.

Foi desenvolvida uma aplicação para possibilitar aos responsáveis por definir as regras e o modelo do SMS, a criação e manipulação destas estruturas genéricas. Foi desenvolvida uma outra aplicação que, usando estas estruturas genéricas permite a geração de UIs disponibilizadas aos utilizadores finais. Esta aplicação é capaz de gerar diferentes tipos de UI de acordo com a estrutura genérica, permitindo utilizadores finais manipular a informação contida no SMS. Estas UIs também permitem a navegação através da informação contida no sistema e a visualização dessa informação e das suas relações de diversas maneiras, que se ajusta às necessidades do utilizador. Esta navegação é possível, uma vez que as UIs geradas são sensíveis ao contexto e contêm um menu de navegação que é gerado de acordo com as necessidades do Utilizador.

Esta arquitectura para desenhar e gerar UI foi desenvolvida no âmbito do projecto de investigação e desenvolvimento m“Continental Factory of the Future”, que tem como objetivo o desenvolvimento um SMS para ser implementado na Fábrica de Antenas em Vila Real.



# Abstract

Since the first industrial revolution in the eighteenth century, the industry has evolved exponentially. The most significant evolution is being experienced now with the emerging concepts of Industry 4.0 and the new Smart Manufacturing Systems (SMS). These systems have several requirements such as mass customisation, automation, flexibility, adaptability and transparency. Despite the requirements of the SMS, human interaction is still necessary for the process of managing the manufacturing in these systems. This interaction is mainly done through User Interfaces (UI), which must be adaptive and have the ability to meet the system requirements.

This dissertation aims to develop an architecture that allows easy and fast design of User Interfaces and their further generation so SMS end users to access and manipulate the system information. The design of these interfaces was done using data structures based on trees that were named Generic Structures (GS). These generic structures were modelled so they could store the necessary information to generate User Interfaces. This architecture also contains information about the entities and relationships of the SMS Conceptual Model, which allows relationships to be established between this information and the nodes of the generic structures.

An application was developed to allow those responsible for defining the model and events of the SMS to create and manipulate these generic structures. Another application was also developed that uses these GS to generate User Interfaces made available to end users. This application can generate different types of UI according to the generic structure, which allows end users to manipulate the information contained in the SMS. These UIs also allow navigation through the system's information and visualisation of that information and its relationships in a diverse fashion, adaptive to the user's needs. These navigation is possible since the generated UIs are context sensitive and contain a navigation menu that will be generated according to the User needs.

The architecture for designing and generating UIs was developed for the research and development project "Continental Factory of the Future", which aims to develop an SMS to be implemented in the Continental Antenna Factory in Vila Real.



# Acknowledgements

Chegando ao fim de mais uma etapa na minha vida, marcada pela conclusão desta dissertação, não posso deixar de agradecer a todos aqueles que me prestaram auxílio durante a sua realização e também durante o resto do meu percurso acadêmico.

À Faculdade de Engenharia e a todos os seus docentes com quem me cruzei durante os últimos cinco anos do meu percurso acadêmico. Agradeço a oportunidade que me foi dada de aprender, de crescer e de me tornar um Engenheiro. Por todo o conhecimento que me foi transmitido, que sei que será necessário para o meu futuro e para que consiga vencer na vida.

Ao meu orientador, o professor Gil Gonçalves, agradeço pelo voto de confiança, pela disponibilidade e pelo apoio que me ofereceu na realização desta dissertação.

Ao meu co-orientador, o professor Paulo Martins, agradeço pela confiança que depositou em mim na área de investigação, que levou a realização desta dissertação. Agradeço também toda a disponibilidade e todo o apoio que me forneceu, que fez com a realização desta dissertação fosse possível.

Aos meus familiares agradeço do fundo do coração, tudo o que sempre deram por mim, que fez de mim a pessoa que sou hoje. Por todo o conhecimento que me transmitiram, ensinamentos e maneira de olhar para a vida, estou eternamente grato. Aos não presentes, mas nunca esquecidos, que estarão sempre comigo. A todos eles devo tudo o que sou, e espero que estejam orgulhosos do que alcancei.

Aos meus amigos de Guimarães, pelos longos anos de amizade que continuarão por muitos mais, por me acompanharem e sempre me motivarem mesmo nas situações mais difíceis, e principalmente por sempre me fazerem sentir especial.

À comunidade académica da Faculdade de Engenharia, agradeço por tudo o que me foi ensinado, pelas oportunidades que me foram dadas e por todas as tradições que me foram passadas. Por me mostrarem que por vezes, somos capazes de fazer coisas que nunca pensamos conseguir.

Por último, mas não menos importante, agradeço profundamente aos meus grandes amigos que fiz nos últimos cinco anos, que apesar de estar longe da minha família e da minha cidade, fizeram com que sempre me sentisse em casa. Obrigado por todas as jantaras, noites de risos, choros e acontecimentos aleatórios, e também pelo constante interesse no meu paradeiro. Graças a vocês estes foram dos melhores anos da minha vida.

Hugo Martins





*"Sometimes we search for one thing  
but discover another"*

Barney Stinson



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Scope . . . . .	2
1.3	Motivation . . . . .	2
1.4	Main Objectives . . . . .	3
1.5	Structure . . . . .	3
<b>2</b>	<b>State of the Art</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	UI in Smart Manufacturing Systems . . . . .	6
2.3	UI main aspects . . . . .	6
2.4	Communication between UI . . . . .	7
2.5	Generic abstract Templates for UI . . . . .	8
2.6	Adaptive UI . . . . .	9
2.7	UI User Adaptability . . . . .	10
2.8	Interface Menus . . . . .	11
2.9	Machine Learning for pattern recognition and values prediction . . . . .	11
<b>3</b>	<b>Proposed Work</b>	<b>13</b>
3.1	Introduction . . . . .	13
3.2	Continental Project . . . . .	13
3.3	Project Requirements and main Functional Areas . . . . .	14
3.3.1	SMS requirements . . . . .	14
3.3.2	Manufacturing Planning and Control Main Functional Areas . . . . .	15
3.3.3	Project's specific Main Functional Areas . . . . .	16
<b>4</b>	<b>Project's Architecture for designing and generating UIs</b>	<b>17</b>
4.1	Introduction . . . . .	17
4.2	SMS simplified Conceptual Model . . . . .	18
4.3	Generic Structures . . . . .	19
4.3.1	Types of Nodes . . . . .	19
4.3.2	Generic Structure Node Attributes . . . . .	20
4.4	SMS DB instances example . . . . .	21
4.5	Architecture to design and generate UI . . . . .	22
4.6	Technologies used . . . . .	23
4.7	Summary and observations . . . . .	24

<b>5</b>	<b>Generic Structures for UI</b>	<b>25</b>
5.1	Introduction . . . . .	25
5.2	Generic Structures for UI . . . . .	26
5.3	Node Pattern . . . . .	26
5.3.1	Select Pattern . . . . .	26
5.3.2	Insert Pattern . . . . .	28
5.3.3	View Pattern . . . . .	29
5.3.4	Delete Pattern . . . . .	30
5.4	Conceptual Model to Store the UI GS . . . . .	31
5.5	Relationships between UI GSs . . . . .	32
5.6	UI GSs Syntax Rules . . . . .	33
5.7	UI Designer . . . . .	34
<b>6</b>	<b>Types of UI and UI Generation</b>	<b>39</b>
6.1	Introduction . . . . .	39
6.2	UI Components . . . . .	40
6.2.1	Label . . . . .	40
6.2.2	Button . . . . .	40
6.2.3	Text Field . . . . .	42
6.2.4	Date/Time Picker . . . . .	42
6.2.5	Toggle . . . . .	43
6.2.6	Table View . . . . .	44
6.3	Type of UI Templates . . . . .	45
6.3.1	Execution UI Template . . . . .	45
6.3.2	Search UI Template . . . . .	46
6.3.3	Select UI Template . . . . .	46
6.4	UI Generation . . . . .	47
6.4.1	Execution UI . . . . .	48
6.4.2	Search UI Generation . . . . .	49
6.4.3	Selection UI . . . . .	50
<b>7</b>	<b>Navigation and Visualisation</b>	<b>53</b>
7.1	Introduction . . . . .	53
7.2	Navigation in Context Sensitive UIs . . . . .	53
7.2.1	Navigation Menu . . . . .	54
7.2.2	Navigation between UI . . . . .	56
7.2.3	Navigation in different Types of UI . . . . .	57
7.3	Generic Diagram UI . . . . .	59
7.3.1	Diagram Cells . . . . .	59
7.3.2	Visualise Information . . . . .	60
7.3.3	Diagram Model . . . . .	61
<b>8</b>	<b>UI Designer and UI generator</b>	<b>63</b>
8.1	Introduction . . . . .	63
8.2	UI Designer . . . . .	63
8.3	UI Generator . . . . .	66
8.3.1	Menus and generation of interfaces . . . . .	67
8.3.2	User Theme Adaptability . . . . .	69

<b>9</b>	<b>Conclusions and Future Work</b>	<b>71</b>
9.1	Generic Structures' Reach and Advantages . . . . .	72
9.2	Contributions . . . . .	73
9.3	Future Work . . . . .	73
9.4	Final Remarks . . . . .	74
	<b>References</b>	<b>75</b>



# List of Figures

2.1	Data acquisition and analytics in a service-oriented production system [1]	7
2.2	Framework for UI customization [2]	9
2.3	User Profile Example [3]	10
4.1	Entities and Relations Diagram of the Simplified SMS DB	18
4.2	GS representation of the simplified SMS DB Model	19
4.3	Simplified SMS DB Entity Tables	21
4.4	SMS DB Instances Virtual GS representation	22
4.5	Diagram of the Architecture to design and generate UI	23
5.1	Simple Examples of GS for UI using Select Pattern Nodes	27
5.2	Examples of GS for UI using Select Pattern Nodes with user interaction	27
5.3	Example of GS for UI using Select Pattern Nodes with a Node pointing to another	28
5.4	Examples of GS for UI using Insert Pattern Nodes	29
5.5	Examples of GS for UI using Insert Pattern Nodes, with better selection	29
5.6	Examples of GS for UI using View Pattern Nodes	30
5.7	Examples of GS for UI using Delete Pattern Nodes	30
5.8	Options to manipulate and create GS	34
5.9	Node addition to the GS design area	35
5.10	Entity attributes selection in UI Designer	35
5.11	GS Node Pattern change	36
5.12	Example of a created UI GS	37
6.1	Label usages for UI generation	41
6.2	Button usages example for UI generation	41
6.3	Different Text Fields Generation Examples	42
6.4	Time and Date Pickers generation Example	43
6.5	Interactive selection elements in Date and Time Pickers	43
6.6	Toggle Button generation Example	44
6.7	Table View generation Example	44
6.8	Execution UI GS Example and Template	45
6.9	Search UI GS Example and Template	46
6.10	Selection UI Pseudo GS Example and Template	47
6.11	Execution UI generation Example and Layout Rules	48
6.12	Search UI generation Example	49
6.13	Search UI Components Area interaction	50
6.14	Search UI Filter and Ordering	50
6.15	Selection UI generation Example	51

7.1	Example of GS for the Menu Search Category . . . . .	54
7.2	Example of GS for the Dependencies Search Category . . . . .	54
7.3	Example of GS for the Detail Search Category . . . . .	55
7.4	Example of GS for the Execute Search Category . . . . .	55
7.5	Navigation Menu Example from a UI with a Supplier Entity for context . . . . .	55
7.6	Example of Navigation between different UI . . . . .	56
7.7	Navigation Menu example in an Execution UI . . . . .	57
7.8	Navigation Menu example in a Search UI . . . . .	58
7.9	Navigation Menu example in a Selection UI . . . . .	58
7.10	Generic Diagram Cell Example . . . . .	60
7.11	Example of a Search Event Execution in the Generic Diagram UI . . . . .	60
7.12	Adding instance cell to the Generic Diagram UI . . . . .	61
7.13	Generic Diagram UI options . . . . .	61
7.14	Example of the visualisation of a Diagram Model . . . . .	62
7.15	Diagram Cells generated from a Diagram Model . . . . .	62
8.1	UI Designer application Options Bar . . . . .	63
8.2	Folder Management Tool . . . . .	64
8.3	List of Search UI GSs . . . . .	65
8.4	Detail of a Search UI GS . . . . .	65
8.5	User Management Interface . . . . .	66
8.6	UI Generator Top bar . . . . .	66
8.7	UI Generator Side Navigation Menu . . . . .	67
8.8	UI Folder Navigation Menu . . . . .	68
8.9	UI Event History . . . . .	68
8.10	User Preferences Interface . . . . .	69
8.11	Some of the different Themes available applied to a generated UI . . . . .	70



# List of Tables

4.1	Type of Nodes and corresponding Visual Representation . . . . .	20
4.3	GS Model Attributes . . . . .	21
5.1	Different patterns for Entity Nodes . . . . .	26
5.2	GS Nodes Attributes common to all Node Types . . . . .	31
5.3	GS Nodes Attributes specific to Entity Nodes . . . . .	31
5.4	GS Nodes Attributes specific to Data Nodes . . . . .	32
7.2	GS Model Attributes . . . . .	59



# Abbreviations

SMS	Smart Manufacturing Systems
UI	User Interfaces
IoT	Internet of Things
CPS	Cyber-Physical Systems
DB	Database
HMI	Human-Machine Interaction
MES	Manufacturing Execution System
APS	Advanced Programming System
CAD	Computer-Aided Design
JSON	JavaScript Object Notation
IDE	Integrated Development Environment
GUI	Graphical User Interface
GS	Generic Structures



# Chapter 1

## Introduction

### Contents

---

<b>1.1 Context</b> . . . . .	<b>1</b>
<b>1.2 Scope</b> . . . . .	<b>2</b>
<b>1.3 Motivation</b> . . . . .	<b>2</b>
<b>1.4 Main Objectives</b> . . . . .	<b>3</b>
<b>1.5 Structure</b> . . . . .	<b>3</b>

---

Smart Manufacturing Systems (SMSs) are Information and Technological Applications that meet the requirements usually associated with traditional Manufacturing Planning and Control (MPC) applications - like Product Data Management (PDM), Material Requirement Planning (MRP) and Capacity Requirement Planning (CRP). SMSs also contain a new set of requirements that emerge from the concepts and new technologies related to Industry 4.0. Connectivity, products and system customisation, integration, synchronisation, and flexibility are some of these requirements. From th set of requirements, the SMS User Interfaces(UIs) customisation and adaptability was the primary goal of this dissertation.

### 1.1 Context

Since the first Industrial revolution in the eighteen century, the manufacturing industry has been evolving into the complex manufacturing process seen today in modern factories. This evolution started with the invention of the steam engine that was used to achieve Mass Production, which later evolved to electrical and digital automated machinery[4]. Currently, we find ourselves in the midst of the fourth industrial revolution named "Industry 4.0". This industrial revolution is set to have a substantial economic impact since, unlike other industrial revolutions, it was predicted and is being analyzed as it unfolds and shapes the future [5].

Smart Manufacturing enforces this new Industrial Revolution. This industrial revolution covers various technologies and, bringing together humans, technology and information, aims to innovate the existing manufacturing industry. In recent years, countries with notable manufacturing

sectors have been developing technologies to implement in Smart Manufacturing. Some of these technologies are IoT (Internet of Things), and CPS (Cyber-Physical Systems), which are widely used in numerous fields [6]. SMS shifted from the typically used paradigm of mass production to one of mass customization that is possible using the technologies developed for Industry 4.0.

Despite aiming to become autonomous, most SMS components and machines always imply input from human users. This interaction with the system is made almost every time through a UI presented in a display. This UI can significantly influence work quality and time efficiency depending on its design. Furthermore, the information displayed is sometimes very complex and could be susceptible to the user's skill level and the easiness that he/she has with computers.

## 1.2 Scope

To fulfil the growing need for customisation and flexibility of the manufacturing processes, UIs tend to be more and more challenging to use. The products that are manufactured must be customisable, and the requirements of the SMS itself increase even more the complexity of these systems. With this level of complexity, explicitly designing and developing all the needed UIs would be very time-consuming and almost impossible in an industrial environment[7]. Changes in the manufacturing system requirements would also lead to recurrent changes and redesign of the numerous existing UI, which would be highly time-consuming if each UI is developed for each specific purpose. To prevent this long process, there is a need for an approach to design Adaptive UIs, where their design is as simple, automated and flexible as possible.

The UIs that will be designed through the work developed for this dissertation are within the scope of the Research and Development project "Continental Factory of the Future". The project's goal is to develop an SMS for an Industrial factory. Therefore the UIs part of the SMS must fulfil the system requirements. The development of the project SMS is divided into teams, each with different tasks. There are teams responsible for developing the SMS UIs, others for defining the SMS conceptual model, the system's events and processes, and those that will test the whole SMS.

This project is halfway through its conclusion, so its work is still under development. Therefore, this dissertation's work is neither final nor representative of the entirety of the work developed for the project in this dissertation's lifespan.

## 1.3 Motivation

With the customization of the SMS and its products, brought by Industry 4.0, SMSs must be able to adapt to the manufacturing and users' requirements. To this extent, the UI used in an SMS must be somewhat adaptive to avoid designing a UI for every system need. Due to the number and complexity of all the components, people and information that are part of an SMS, specifically designing every single UI would be unfeasible and almost economically impossible [8].

The architecture developed in this project aims to allow the design UIs that are adaptive and whose design is straightforward and effortless, making its development far less expensive and

time-consuming. At the same time, the UIs must be straightforward and appealing, so the user is engaged and productivity increases. With this UI, the information displayed can also be customized in some aspects since the user needs might vary throughout time as the system evolves.

This approach for designing UIs alongside the adaptive aspect intrinsic to them will easily allow the development of all the UIs the SMS requires. In the future possibility of a change in the requirements of the SMS, these UIs must easily be altered, deleted, or new ones added. This would save human and financial resources since its implementation would also be much faster than traditional methods, where designers must manually change the desired aspects or develop new UIs.

## 1.4 Main Objectives

The main goal of this dissertation is to develop an architecture that provides the ability to design and generate Adaptive UIs for the project SMS. The generation of these UIs is based on Generic Structures(GS), tree data structures developed in the scope of the project containing the UI information. Two main objectives can derive from the goal of this dissertation, which allows for better compartmentalisation of the work developed and consequent explanation.

- **Develop a fast, simple and accessible Application for designing User Interfaces based on generic structures to meet the requirements of Smart Manufacturing Systems.**

This first objective aims to provide those responsible for developing the SMS's conceptual Model and events with a way to design the desired interfaces fast. This design will be possible by developing an Application that allows Users to manage generic structures that store the system's UI information. The application developed must be simple and efficient and allow an Interface to be designed in a fraction of the time required to do the same through traditional methods.

- **Design and develop an Application to generate different types UIs, through which end users can populate and access the information in the SMS DB according to their needs.**

The generic structures created to answer the first objective will be used to generate Adaptive User Interfaces. This second objective consists of an Application that will be incorporated in the SMS that will be able to generate UI that represents events available in the SMS. The generated UIs will be presented to end users, allowing them to manipulate, access, and browse through the system's information. These UIs must be context sensitive so it can adapt to the user needs.

## 1.5 Structure

The structure of the dissertation is as follows:

- Chapter 1 - Introduction: The present chapter introduces the reader to the dissertation, describes the context of the work developed, brings about the scope and motivation for the dissertation, and also defines the main objectives of the dissertation.
- Chapter 2 - State of the Art: This chapter contains the information found in the literature regarding UI, their role in SMS and Industry 4.0, and methods to design and make them adaptive.
- Chapter 3 - Proposed Work: In this chapter, the project in which this dissertation is inserted is further explained, giving details of the project's goals and fitting the desired outcome of the dissertation into the scope of the project.
- Chapter 4 - Project's Architecture for designing and generating UIs: This chapter explains the architecture developed to design and generate adaptive UI and the Simplified SMS DB Conceptual Model that will be used to design the UI.
- Chapter 5 - Generic Structures for UI: Defines and details the Generic Structures that will store the information needed to generate UI and presents the application developed to create and manage these structures.
- Chapter 6 - Types of UI and UI Generation: Explains how the information stored in generic structures is used to generate UIs. This chapter also details what UI elements were used and how they were combined to develop the different types of UI templates and generate concrete UIs from generic structures.
- Chapter 7 - Navigation and Visualisation: Different alternatives for end users to navigate and visualise the information in the System are presented in this chapter.
- Chapter 8 - UI Designer and UI Generator: Features of the UI Designer and the UI generator are presented in this chapter, like Generic Structures visualisation in the UI Designer, the Menus available in the UI Generator and its user adaptability.
- Chapter 9 - Conclusions and Future Work: Conclusions of the work developed are discussed in this chapter, and the future work that will continue to be developed in the project's remaining duration is presented.



# Chapter 2

## State of the Art

### Contents

---

<b>2.1 Introduction</b>	<b>5</b>
<b>2.2 UI in Smart Manufacturing Systems</b>	<b>6</b>
<b>2.3 UI main aspects</b>	<b>6</b>
<b>2.4 Communication between UI</b>	<b>7</b>
<b>2.5 Generic abstract Templates for UI</b>	<b>8</b>
<b>2.6 Adaptive UI</b>	<b>9</b>
<b>2.7 UI User Adaptability</b>	<b>10</b>
<b>2.8 Interface Menus</b>	<b>11</b>
<b>2.9 Machine Learning for pattern recognition and values prediction</b>	<b>11</b>

---

### 2.1 Introduction

As machines and industrial robots grow in autonomy, the human interaction aspect of an SMS is mainly dedicated to assigning orders and procedures, supervising work being done and interrupting it in case of emergency [9]. Machines used in SMS are more technologically developed, more complex and require higher expertise to be operated, as they can do more intricate procedures, contain more components and have to deal with higher volumes of information [10]. An increase follows this in the number and complexity of the UIs used in an SMS.

The way information is presented to users is critical, so they can quickly assess the information and assign orders or procedures to the machines. The UIs must then be appealing and appear simple to users, as they form their opinion of them almost instantaneously [11], which can influence their performance. This chapter covers the role of interfaces in SMS in section 2.2, as well as a UI's main aspects to be considered upon its design in section 2.3.

In the literature regarding UIs, were found methods to design both UIs and Adaptive UIs. A way to ensure decentralisation of the manufacturing system's information is discussed in section 2.4, and in section 2.5 is analysed an approach to make UI design somewhat autonomous in section

2.5. The interface adaptability to its users and its menu and alerts behaviour is discussed in sections 2.8 and 2.6, respectively. Finally, a pattern recognition method using machine learning to predict user data inputs is analysed in section 2.9

## 2.2 UI in Smart Manufacturing Systems

As a result of the diversity and number of Industry's 4.0 components in SMS, the communication process between systems and other elements of industrial environments is also unavoidably complex, although crucial to achieve interoperability. For instance, a product must be able to communicate with different kinds of machines in order to decide where it must execute a particular operation. This communication between the SMS components ensures the decentralisation of the information represented in the Manufacturing Execution System (MES), one of the main goals of Industry 4.0 [12]. By achieving this decentralisation of information, we ensure that every smart component of the shop floor is aware of its processes, where it came from, and its destination while communicating with the surrounding components. To this degree, and to facilitate this communication between SMS components, where some need human interaction through the display or insertion of information, the development of generic models of representation and standard interfaces is essential to aid information sharing [12].

UIs are also an essential component in Industry 4.0 as they are part of two layers of the architecture proposed in [13]. As part of the informational layer, it is through UI that information regarding the factory (or part of it) and its machines, robots and products is made available to the user. UIs are also part of the integration layer, allowing the operator, by interacting with the UI, to start processes, start production orders, and other instructions allowed by the manufacturing system that controls the production flow of the factory [13].

## 2.3 UI main aspects

When designing the UI, its behavior must depend on the process it represents. Suppose is being designed a UI for monitoring a machine's execution. In that case, this interface should be merely informative and would show detailed information about the machine and all associated tasks and products. On the other hand, when designing interfaces for the resource entry point of the factory, they would be mainly dedicated to allowing the user to input the stock entries into the system, in this case, probably insert information into a database. Another scenario would be an interface dedicated to create manufacturing orders. In contrast, in this case, it would be a mix of both previous examples, where it would be needed to create the order itself and select the material needed to manufacture the final product, as this information is presented to the user. The interfaces for SMSs must then be flexible and easily adapted to method changes and new manufacturing demands [14] to allow them to be used in different sections of a factory.

The main components of a UI can be narrowed down to data insertion elements, action elements, like a button, and information display elements. At its core, an interface is the aggregation

of sets and groups of these elements styled to the client's request and, most times, validated to prevent errors in the system. Each of these abstract elements can then be formed into something concrete by giving them a specific size, position, colour, layout, and shape, amongst many other attributes. These visual aspects of the interface are the main factors influencing the user perception of the UI [15]. On top of this, the complexity of these UI elements could vary immensely. For example, a simple text label and a table or a pie chart are both UI display elements, but for different data types, the first contains one value and the latter several.

Mass customisation is one of the pillars of Industry 4.0. To achieve this, the manufacturing system's processes must be flexible, and design customisation of the product and all its components. To ensure this customisation is done with ease, it helps the user to visualise the product outcome as he personalises the product and chooses specific characteristics [16]. One way to provide this visual aid to the user is allowing the interface to display an image or 3D representation of the product and the changes made in the product with the help of CAD (Computer Aided Design) tools, allowing the user to assess the possible product outcomes in real time.

## 2.4 Communication between UI

In order to ensure the decentralisation of the information represented in the MES, as factory 4.0 dictates, it has to be able to communication between different interfaces in the factory, being of Human-Machine Interaction (HMI) or not.

Some devices used in SMS have specific communications protocols, and their use with different system devices is not allowed by these protocols. Some of these devices already come with integrated UIs, but even those interfaces might not be the best suited for the system. The way devices communicate with each other is key for developing UIs, because the information they exchange will either be the data represented in the UI or the data inserted in the interface by an operator, which the device will then forward to other devices.

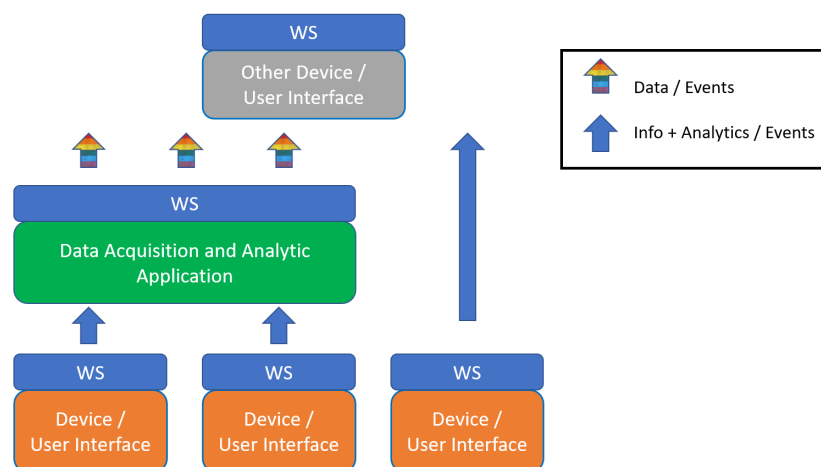


Figure 2.1: Data acquisition and analytics in a service-oriented production system [1]

To best the first concern, a Service Oriented Architecture (SOA) model could be used in an industrial environment, using Web Services at the device level, employing a publisher/subscribe mechanism to allow the device to link to different applications at runtime [1]. The only requirement for the devices would be access to WiFi, as most electronic devices nowadays do. The data acquired by devices and machines and orders dispatched by operators through UIs would then be turned into helpful information made available by Web Services. Other UIs or devices could later use this information in the factory. This interaction between different devices and interfaces can be seen in figure 2.1.

The engineering approach in [1] describes a UI that receives processed information from the shop floor devices, composed of different controllers and types of outputs, later used to define the main aspects of the interface and build the UI.

## 2.5 Generic abstract Templates for UI

In an Industrial environment, there is a need for several different UI in different stations and posts of a factory. In order to make its design easier, there is a need for a methodology to customise them to specific needs. A method that can be used is Model Driven Development, described in [17]. By having abstract models of interfaces, these models can be adjusted to the specific needs of the interface being designing. As mentioned before, a UI can be breakdown into some elementary elements that the user interacts with, like insertion, action (buttons) and visualisation elements.

In [17] is proposed the abstraction of these simple elements as abstractions of UI widgets that, when put together, form an abstract UI through an abstract UI Model representation. Families of abstract UIs could then be developed to the necessities of the manufacturing system. When developing an interface to a specific task, using this abstraction and giving it concrete values and properties quickly generate a concrete UI. This customisation can either be done at development time when designing interfaces for two completely different scenarios or at runtime when developing adaptive interfaces that rely on the process characteristics and the user experience [17].

Model Driven Development, User Centered Design and Object Oriented Customization, can be combined to design UI based on the user's roles and the interface tasks and functions in order to develop a group of use cases [2]. These use cases will later define the interface's abstraction to be developed using abstract UI templates and rearranging its elements to fit the possible use cases. The interface designer will then make final adjustments to customise it to meet the desired requirements and transform it into a concrete and final UI. This customisation framework can be found in figure 2.2.

Others have also used the concept of abstract UIs to develop user-centred interfaces based on interfaces from different manufacturers, allowing the user to customise its components. In some fields, this ability for customisation solved the problem posed by other interfaces, and the use of generic UI with function mapping mechanisms was found to be successful [18].

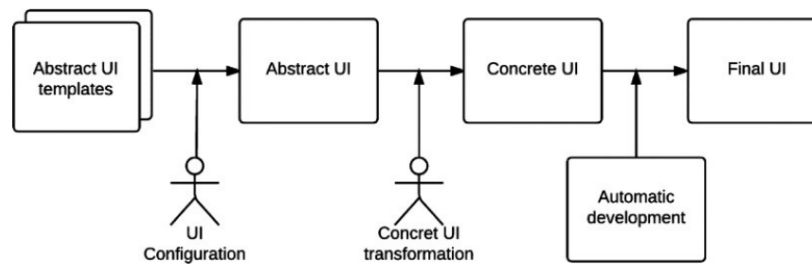


Figure 2.2: Framework for UI customization [2]

## 2.6 Adaptive UI

Aspects concerning the person using the interface should be considered when talking about how the UI should adapt. For example, several user characteristics can influence how the interface adapts to the user, such as age, physical impairments, easiness around computers and experience using said interface [19]. Also, other variables like user performance and user knowledge will influence the interface adaptability[9].

Setting the variables for the interface also requires a method to change according to these variables. For example, the adaptation of the content displayed to the user and the order in which it is displayed [9], and the presented information's colour, font, and size. Following a rule-based system, the interface would adapt according to the user variables' values by associating user variable possible values to the corresponding interface adaptation method.

With an experienced user who knows the system well, every aspect should provide as much information as possible. For example, it could display a global view of the process, production statistics, and even a virtual representation of the system (or part of it) [19]. It could also display its current status and the status of its components because the user would be able to interpret, make a better assessment and have full use of the information given. An experienced user would also be provided with an extensive list of all alarms and warnings in play to be on par with everything going on, even those events that do not impose a hazardous situation or machine malfunction.

On the other hand, more technical information would be hidden when the user is not very experienced, thus not influencing the user's ability to gather or understand information. Furthermore, alarms can be displayed to the user only in case of critical malfunction or similarly dangerous situations. At the same time, a combination of steps to fix what had happened and re-establish the system's normal state would be provided.

Storing a user profile is a good way of keeping track of the evolution of knowledge and experience of the user, as well as preferences and personal information [3]. An interactive tutorial or guide could be provided to teach an inexperienced user how to use the interface [19]. This guide would instruct them about the interface's main aspects and functionalities. At the same time, it ensures that the user is learning and adapting to using the UI and is gaining knowledge about the system at his own pace. The UI would also be able to adapt to this evolution.

The ease with which operators use the UI is critical. It is directly related to productivity, so the interface should adapt its presentation to match the user's characteristics. For example, for a user who is old or has some degree of vision loss, textual content would be displayed in a bigger font and a more contrasting colourway so that the words could be easily perceptible, making the interface less visually tiring to the user.

This method for designing an Adaptive UI would be based on a cyclic process to obtain information from the user, who would provide feedback before or after using the interface through a simple questionnaire. This information would be used to update the user profile so the next time the user uses the interface, it would adapt to the evolution of the user and provide feedback to the interface designer about the interface's said ability to adapt.

## 2.7 UI User Adaptability

The design algorithm of the interface is not always able to adapt perfectly to every user, and even so, there are always specific user preferences that cannot be predicted and that are not adaptive. Therefore, the interface needs to be adaptable by the user, who can manually change specific interface configurations, textual content characteristics like colour, size and font, the way elements are displayed, and which contents are displayed on the interface. By making the UI adaptable, the user chooses an environment to suit his preferences, increasing his productivity and ease with the UI. Moreover, with adaptable UI, letting the user personalise the interface and making him part of the customisation process makes the user more dedicated to the UI [20].

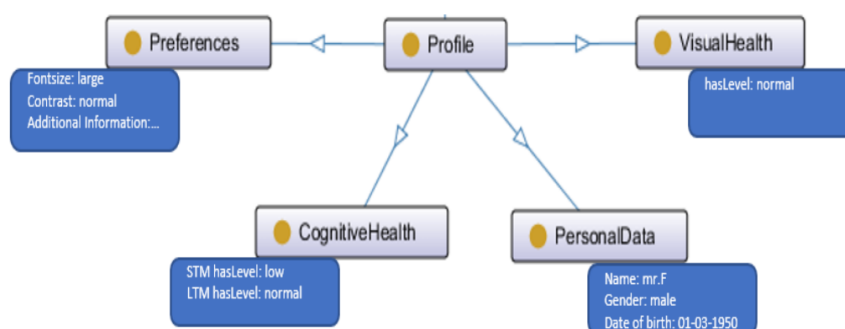


Figure 2.3: User Profile Example [3]

Figure 2.3 shows an example of a User profile, where in addition to previously mentioned user characteristics and various personal data, this information regarding the user preference of the interface has to be stored. Thus, the User Profile is an integral part of designing a Reactive UI that can be both Adaptive to the user and Adaptable by the user. This profile stores meaningful information used as input variables for the interface to be generated most efficiently, updated, and refined each time the interface is used.

Developing a UI with the methods presented above would have a significant upside to the work environment in terms of productivity, as eventually, every worker will be able to use the interface to its full ability. The ease of learning to use it would make new employment more accessible. Finally, these methods would benefit economically as integrating more sophisticated machines would be more straightforward and better accepted by users [19]. Giving the user a sense of control over the interface customisation would enhance user satisfaction amplifying the improvements in production mentioned above [21].

## 2.8 Interface Menus

One aspect of the interface that allows organising the information better, and the capabilities of the UI, is the implementation of menus. Intelligent and good use of these when designing an interface can significantly impact user satisfaction and ease of use of the interface [22].

Menus allow an interface to present different types of information and even users to navigate through different interfaces. For this navigation to be helpful to the user, menus must be displayed to save the user time rather than delay him/her. The order in which the menu options are presented can be done in alphabetical order, by grouping in categories, or by indexation defined in the interface. Nonetheless, the most effective way is for the menu options and information order to be adaptive based on user selection events and adapted by users themselves [22].

However, when talking about devices with smaller screens with menus composed of several levels, there is a limit to the information we can display without making it too small to be perceptible. This problem can be mitigated by using 3D menus that allow for better use of the space the interface has available, like a 3D carousel view where the information is arranged on a sort of a ring that can rotate on its axis[22]. These menus play an advantage, allowing more information regarding menu options description to be displayed alongside the menu option itself.

When developing the UI, the choice of the menu type is essential, as it should be befitting to the intended functions the interface must have. Several aspects must influence this decision, like the menu's number of items and the length and space each one uses. 3D menus allow for better use of space, are more intuitive and are better suited for broader implementations, although they are not always the best solution [22].

## 2.9 Machine Learning for pattern recognition and values prediction

When designing interfaces that mainly require the insertion of data and actions from the user, sometimes this interaction develops into repetitive, time-consuming actions by the operator because when the UI was developed, there was no way of knowing how the operator would interact with it. This repetition can make the interaction not optimal in terms of the time the operator takes to perform this action, meaning he/she would click on several buttons in the same order, time and time again [23].

Standard UI do not register interactions of the user. However, suppose it was to be stored interaction associated with the element of the interface, the timestamp, the content of the element, and the event or events it would trigger. In that case, this information could later be used to predict the user's movements. From the group of possible actions, this data, when acquired, can then be processed into valid sequences of events forming a possible interaction of the user.

Using Machine Learning is proposed in [23], an algorithm to generate the most probable outcome of the interaction before it even happens. This prediction would only happen after a learning phase, where the user would have to use the UI casually for some time. Once trained, the interface would then adapt to present the user with a group of actions generated in the algorithm. The user would also be able to alter the predicted information, if necessary, as the algorithm is not always accurate because some new parameters might come into play. This change would require the interface to learn again from the user and adjust the parameters that allow the UI to adapt.

In tests made in [23] there were significant improvements in the number of actions during the interaction, making this less time-consuming and less likely to lead to errors in the system, as there were fewer buttons clicked and fewer data inserted into the interface.



# Chapter 3

## Proposed Work

### Contents

---

<b>3.1 Introduction</b>	<b>13</b>
<b>3.2 Continental Project</b>	<b>13</b>
<b>3.3 Project Requirements and main Functional Areas</b>	<b>14</b>
3.3.1 SMS requirements	14
3.3.2 Manufacturing Planning and Control Main Functional Areas	15
3.3.3 Project's specific Main Functional Areas	16

---

### 3.1 Introduction

As mentioned before, the work presented in this dissertation was developed as part of the Research and Development Project "Continental Factory of the Future". Once implemented in an actual use case scenario, the UIs designed will be used alongside many other technologies developed in this project's scope.

This chapter presents the company endorsing the project and the factory where it will be implemented. The project will be explained, introducing its essential requirements and fitting the dissertation work into the project's scope. Since it entails the use of many technologies and research in several engineering fields, it is divided into five sub-projects according to the area of research or development. The fourth sub-project will be further detailed since it is the one that requires the development of UIs, and a connection will be established between the objectives of the dissertation and the specific requirements of the project.

### 3.2 Continental Project

Continental is a technology Company Group created in Germany in 1871, with its headquarters based there still to this date. It develops groundbreaking technologies and services primarily for the automobile Industry. Most known for manufacturing tyres for trucks and commercial

vehicles, it also develops several other components and software used by most car manufacturers. This multinational employs over 240 000 people and has several manufacturing plants all over the globe. One of these factories is "Continental Advanced Antenna Portugal", one of Europe's leading car antenna manufacturers that yearly manufacture and exports all over the world over 18 million antennas.

This Antenna factory in Vila Real is coordinating the R&D project where this dissertation work is inserted, "Continental Factory of the Future". The aim of this project is to develop technologies that will allow this factory to adapt to the requirements of the next Industrial revolution, "Industry 4.0". This will be achieved by a joint effort made by three Universities, the University of Porto, the University of Minho and the University of Trás-os-Montes e Alto Douro. The coordination of each sub-project was assigned to one of the three Universities, although all sub-projects have members from all three universities.

Currently, most big corporations face new challenges that test their technology, commercial strategies and organisational structure. In the same way, so does the Continental Antenna factory, where at the technological level, the use of increasingly flexible machinery and equipment alongside powerful product design tools leads to a growth in diversity and complexity of the products that the factory is allowed to manufacture. This product diversity, accompanied by the ability to customise every product, is one of the current challenges factories face. To react to these challenges while not lowering the performance levels obtained in mass production, the manufacturing, decision-making processes and the technology systems that support the manufacturing management need to be adjusted.

These challenges and the necessity to adjust present the new reality factories face upon the arrival of Industry 4.0. In this context, the sub-project SP4 - Management of Industrial Manufacturing, proposes the development of a Smart Manufacturing System (SMS) that will provide solutions to assure production competitiveness and ensure the efficiency of the production system. An essential part of the SMS is the interaction the human operators in factories have with the machines and how they influence the production flow. This interaction is made almost every time through a User Interface.

### **3.3 Project Requirements and main Functional Areas**

This project integrates several functional areas that will allow meeting the SMS requirements. These functional areas were grouped into six main functional areas to deal with a system of this dimension efficiently. Some of these main functional areas are related to specific Manufacturing Planning and Control functional areas, and others englobe all of them and are specific to certain goals of this project.

#### **3.3.1 SMS requirements**

The system that will be developed in this project must gather the following requirements of Smart Manufacturing Systems:

- **REQ1 - Product Diversity and Mass Customisation:** Efficiently handling part's diversity and customisation, the SMS must be able to efficiently deal with the explosion of information that results from the increase in the diversity of possible products and processes caused by mass customisation.
- **REQ2:Connected:** Have the ability to connect all resources and entities that interact and affect the manufacturing system to ensure all the information used in the decision-making process is up to date and reflects the actual state of the manufacturing system.
- **REQ3 - Automated and Integrated:** Be able to allocate, synchronise and monitor the tasks of all resources and entities that interact or affect the manufacturing system, with the least human interaction possible and with great credibility.
- **REQ4 - Agile:** Adapt, in real-time and with minimal human intervention, to changes in demand, product changes, equipment status changes and component availability, through automatic reallocation of tasks, equipment configurations and material flow changes.
- **REQ5 - Flexible and Adaptive:** The manufacturing system must adapt to new products, new equipment, new processes and new organisational procedures.
- **REQ6 - Proactive:** Predict the system state in a temporal horizon lengthier than the current state, providing information on the expected state of the system and anticipating possible problems so they can be avoided.
- **REQ7 - Transparent:** The manufacturing system must be able to be personalised, so it can follow the evolution of the business and adapt to different types of users, equipment and technologies that may appear.

### 3.3.2 Manufacturing Planning and Control Main Functional Areas

As mentioned above, six main functional areas were integrated into this project to have the SMS meet the desired requirements. Four of those are related to Manufacturing Planning and Control functional areas intended to be implemented, which are:

- **FA1 - Product Data Management:** Comprehensive models to manage parts data must be developed based on generic structures that ensure flexibility and adaptation to new products and processes. End users must be able to efficiently manage large amounts of data while at the same time being presented with a simple and easy-to-use UI. The system must also integrate the information from the different functional areas in a single data structure to avoid duplicate data.
- **FA2 - Resources Management:** There is the need to identify e define all entities that are part of the system, from the stations and machinery on the Shop Floor to external entities like clients, suppliers and carriers.

- **FA3 - Manufacturing Planning and Control (Mid-Term):** This mid-term planning and control is responsible for determining what will be produced and purchased and the corresponding quantities. It is also responsible for determining when to purchase or start production of each part to satisfy clients' needs and fully use the system's resources. The mid-term manufacturing planning and control are normally assigned to Enterprise Resource Planning (ERP) Systems within organisations. However, the algorithms and techniques traditionally used in most ERPs are directed to mass production and are not integrated with those used in the shop floor's Manufacturing Monitoring and Scheduling.
- **FA4 - Manufacturing Monitoring and Scheduling (Short-Term):** Includes functions of allocation, sequencing, and monitoring of all activities that will be executed on the Shop Floor to implement a mid-term defined manufacturing plan. In traditional Manufacturing Systems, monitoring and scheduling tasks are executed by the Manufacturing Execution System (MES) and the Advanced Programming System (APS), respectively. These systems are usually not very extensive since they only consider the resources' competencies for a particular activity and its availability. Many other factors influence the allocation and sequencing process, like the availability to transport the resources to and from the activity location or the state or maintenance the system might need.

### 3.3.3 Project's specific Main Functional Areas

The two remaining main functional areas integrated into this project are related to SMS requirements transversal to all functional areas of Manufacturing Planning and Control and are specific to this project SMS. These main functional areas are:

- **FA5 - Integration and Synchronisation:** The SMS must be able to adapt and integrate new kinds of equipment, providing them with the information to perform their tasks. The system also needs to be able to integrate other technologies that might appear in the organisation, synchronising their information with the one present in the SMS.
- **FA6 - Interface Management:** A set of functionalities must provide the ability to customise UIs to adapt them to the organisation's development and different users. A Functional Architecture that includes the system's entities, their relationships and the events available for each entity in the SMS will be implemented. This allows those responsible for management to identify the information they intend to view, how to display it, browse throughout the information contained in the system, define menus and alerts and restrict access to specific data. This will be achieved by designing the UIs required by the system, adjusting them to manufacturing needs and managing users' access to specific interfaces. This dissertation's work was developed within this functional area to implement the SMS being developed for the Continental project. The User Interfaces that will be designed using the architecture developed in this dissertation will help meet some requirements of SMS, namely **REQ5** and **REQ7**.

## Chapter 4

# Project's Architecture for designing and generating UIs

### Contents

---

<b>4.1</b>	<b>Introduction</b>	<b>17</b>
<b>4.2</b>	<b>SMS simplified Conceptual Model</b>	<b>18</b>
<b>4.3</b>	<b>Generic Structures</b>	<b>19</b>
4.3.1	Types of Nodes	19
4.3.2	Generic Structure Node Attributes	20
<b>4.4</b>	<b>SMS DB instances example</b>	<b>21</b>
<b>4.5</b>	<b>Architecture to design and generate UI</b>	<b>22</b>
<b>4.6</b>	<b>Technologies used</b>	<b>23</b>
<b>4.7</b>	<b>Summary and observations</b>	<b>24</b>

---

### 4.1 Introduction

To assure the SMS meets the established requirements, an architecture was developed that allows to efficiently design UIs that give users full use of the SMS functionalities. Each UI must be reactive, meaning it is responsive to the user interaction and the effect of this interaction is perceptible. Additionally, the UIs that will be designed must be adaptive to the User and the context from which they are requested.

The Interfaces developed must be simple and easy to use to increase accessibility and productivity. To cover the SMS requirements, different types of Interfaces must be designed according to the event it portrays. Managing product information, launching manufacturing orders or monitoring production status, and possibly the shop floor are examples of events that need to be executed through a UI.

This chapter explains how the architecture that was developed in the project scope allows storing the information of the entities and relations defined in the SMS's DB and later used to design

the UI. Since the SMS of the Continental factory is very complex, a simplified Conceptual Model for the SMS DB was used. This model represents some aspects of manufacturing management and will be analyzed to display how its entities and relations will be stored.

## 4.2 SMS simplified Conceptual Model

In order to explain the architecture functionalities and provide examples of what UI can be designed and generated, a simplified Conceptual Model of an SMS was developed for this dissertation. This simplification was used since the SMS of the project is still being developed, and its definition is not part of this dissertation's goals. This simplification avoided overflowing the reader with information since the model used represents only a niche of the Entities and relations that will compose the project's SMS Model once it is fully developed. In Figure 4.1 shows the diagram of entities and relations of the simplified SMS DB Conceptual Model.

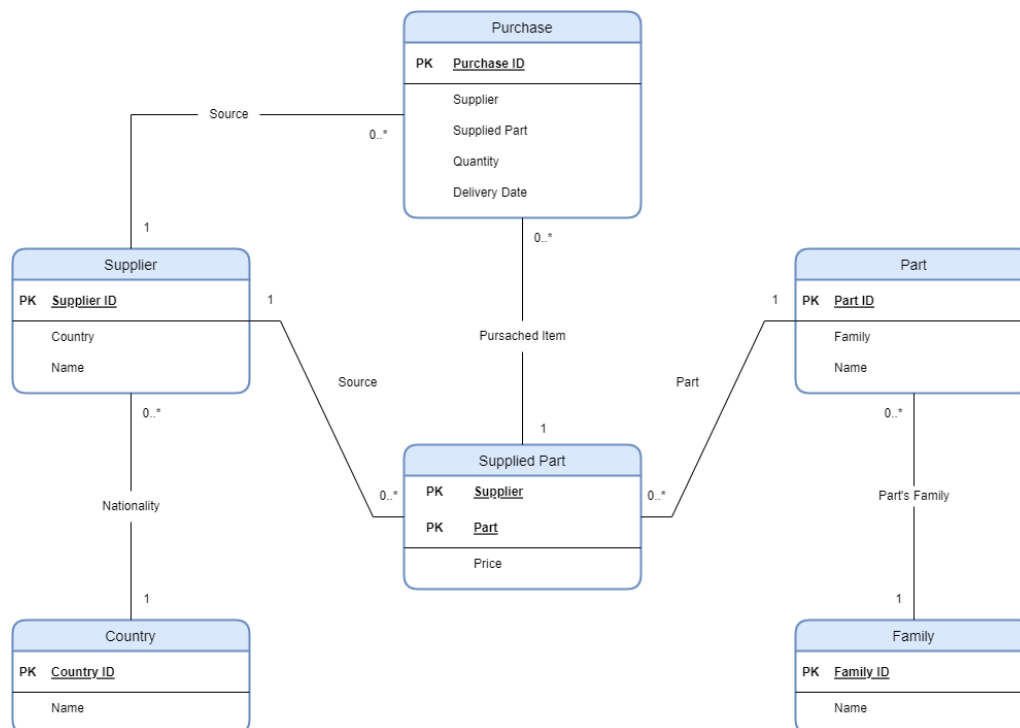


Figure 4.1: Entities and Relations Diagram of the Simplified SMS DB

The simplified model consists of six entities - Family, Part, County, Supplier, Supplied Part and Purchase - that have established relationships between themselves. The diagram in Figure 4.1 shows the name of the relationships between entities displayed through the line that connects them. The same people that designed the SMS DB conceptual Model will simultaneously store its information, which will later be used to generate UI. This architecture created and defined data structures - that were named Generic Structures - responsible for storing all this information regarding the Conceptual Model of the SMS.

### 4.3 Generic Structures

Generic structures were used to model the generation of interfaces to provide as much freedom when designing the interfaces.

GS were used to store information about the conceptual model of the SMS to forestall the wide range of complexity the SMS's entities might have. These GS are based on tree data structures, a mathematical structure that consists of oriented graphs that contain oriented nodes, where every node has a father node except for the root node of the tree, which has no father. On the other hand, each node can have from zero to multiple children. The root node is used to identify each one of these GS.

Each node of the GS has one of three possible types, as shown in Figure 4.2 where the Entities and Relations presented for the simplified SMS DB model have their corresponding GS that contains its information.

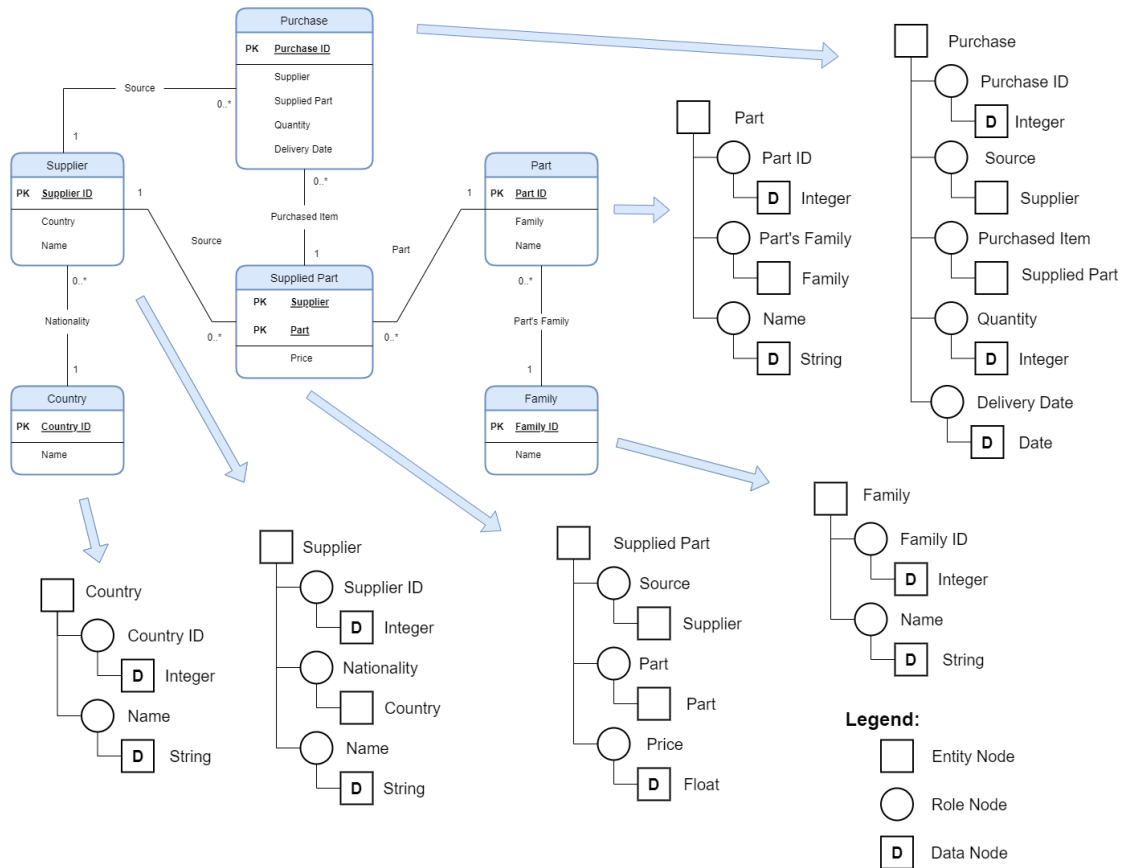


Figure 4.2: GS representation of the simplified SMS DB Model

#### 4.3.1 Types of Nodes

The type of the GS Node will assume one of three values - Entity Node, Role Node or Data Node that can be seen in Figure ???. Each node type has the objective of representing the different aspects

of the SMS DB.

- **Entity Node:** This type of node represents an entity part of the Conceptual Model of SMS DB.
- **Data Node:** Used to represent a primitive data type like Integers or Strings, this type of node has the goal of storing the information regarding an entity's attribute in the SMS DB..
- **Role Node:** As the name suggests, this type of node establishes a relation between its father node and its child node. This type of node represents a relationship between two different entities or an entity's attribute, where the child node of the role will be a data node.

Table 4.1 contains the information regarding the node type and the visual representation used for each node type, as can also be seen in the legend of Figure 4.2. The role node takes the shape of a circle, and other nodes of a square, differentiating data nodes from entity nodes with a D in the centre of the square.




Node Type	Code	Visual Representation
Event/Entity Node	ENTITY	
Role Node	ROLE	
Data Node	DATA	

Table 4.1: Type of Nodes and corresponding Visual Representation

This explanation for each node type can be asserted by looking at the association between the entities and relationships diagram and the generated structure. Analyzing the specific scenario of the Purchase Entity, can be seen that this Entity has relationships with the Supplier Entity and the Part Entity. These relationships are represented by a role Node identifying the relationship that connects both entity nodes, having the Purchase Node as its father and the Part or the Supplier as its child node. Also, for the Purchase Entity, we can analyze how the entity attributes are represented in a GS. A Role Node represents each Entity attribute to assert the attribute name, and a Data Node represents the data type of the attribute.

### 4.3.2 Generic Structure Node Attributes

As previously mentioned, the GS that contain information on how the SMS DB is modelled will be stored in a dedicated DB that will later be used to design the UI. However, the nodes of these GS have other attributes besides their type, which will allow storing the tree structure and the model information. In Table 4.3 we find the model used to store the GS nodes' information.



Attributes	Description	Set of Values
ID	Unique identifier to each GS node	Integer
Root ID	Identifier of the father node (null in case of the root of the structure)	Integer
Type	Code of the corresponding node type	ENTITY, ROLE, DATA
Name	Identifies the SMS DB entity associated with the node, the relation or the data type name, depending on the type of node	String

Table 4.3: GS Model Attributes

#### 4.4 SMS DB instances example

The DB Entity Tables were populated with a couple of instances to understand the Simplified Conceptual Model of the SMS DB. These are presented in Figure 4.3 as well as the entity instances' relationship. Some of these relationships are represented by dotted lines that establish the connection between two entities.

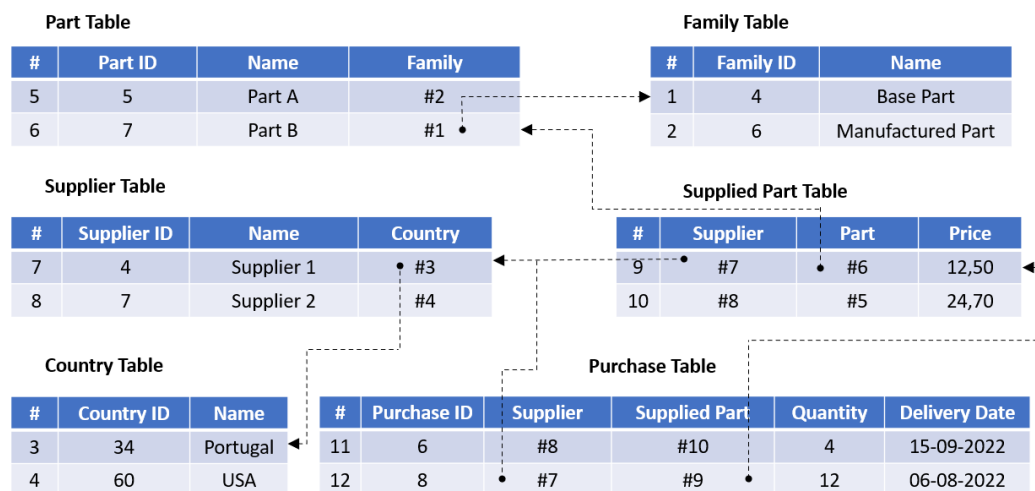


Figure 4.3: Simplified SMS DB Entity Tables

It is also important to note that the SMS in this architecture has a unique numeric identifier for every instance contained in the system. This allows access to that instance regardless of the Entity it represents.

Another advantage of storing the SMS DB conceptual model is the ability to mirror the GS of the system entities into structures containing instance information. The SMS developed in this architecture will be capable of creating a virtual tree representation of the instances contained in

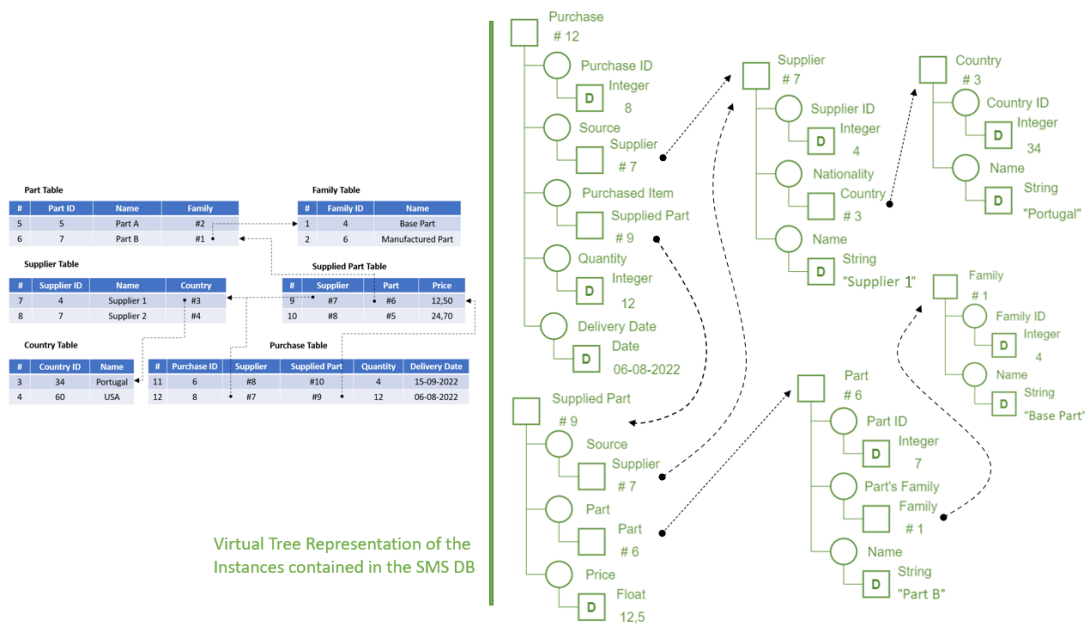


Figure 4.4: SMS DB Instances Virtual GS representation

the Entity Tables of the SMS DB. Figure 4.4 shows a virtual representation of an instance of each Entity of the Simplified Conceptual Model presented.

As seen in the instance GS, the data Nodes take the value of the corresponding attribute value, and the entities have the number identifier of that entity instance. It must be noted that these structures are painted in green to distinguish them from the structures that contain the SMS Model Information. This distinction is because the GS serve only as a virtual representation of the system instances and is not stored in any DB.

## 4.5 Architecture to design and generate UI

The UIs for the SMS will also be developed using the developed architecture, which will allow to quickly design and autonomously generate UIs that will be available to End Users. Rather than individually and specifically developing each UI, in this architecture were used GSs similar to those explained in 4.3, where its nodes also store the information needed to generate the desired UI. Like the GS that stored the SMS conceptual model, these structures will be stored in a separate DB, the UI DB, although in different Tables. To make the generation of UI possible, a model capable of storing these GS must be created and a system that enables the creation of these GS and the generation of the corresponding UI. To this end, two software applications were developed, one meant to design and manage the GSs and another destined for end Users that would generate a UI from a GS, allowing users to interact with the system information. The first will allow project members responsible for defining possible events and operations of the SMS, to quickly design the corresponding UI.

The architecture developed in the project's scope to design and generate UIs is presented in Figure 4.5, with its four main components, the SMS DB, the UI DB, the UI Designer and the UI Generator. The SMS DB contains information regarding all the entities that are part of the SMS and are influenced by it. The UI DB stores GS containing the information necessary to generate UIs and the GS that contain information on the SMS conceptual model, regarding its entities and their relationships. The UI Designer Application will allow users to create or alter UI GS and organize them. This interaction and the UI Designer Application features will be explained and detailed in Section 5.7.

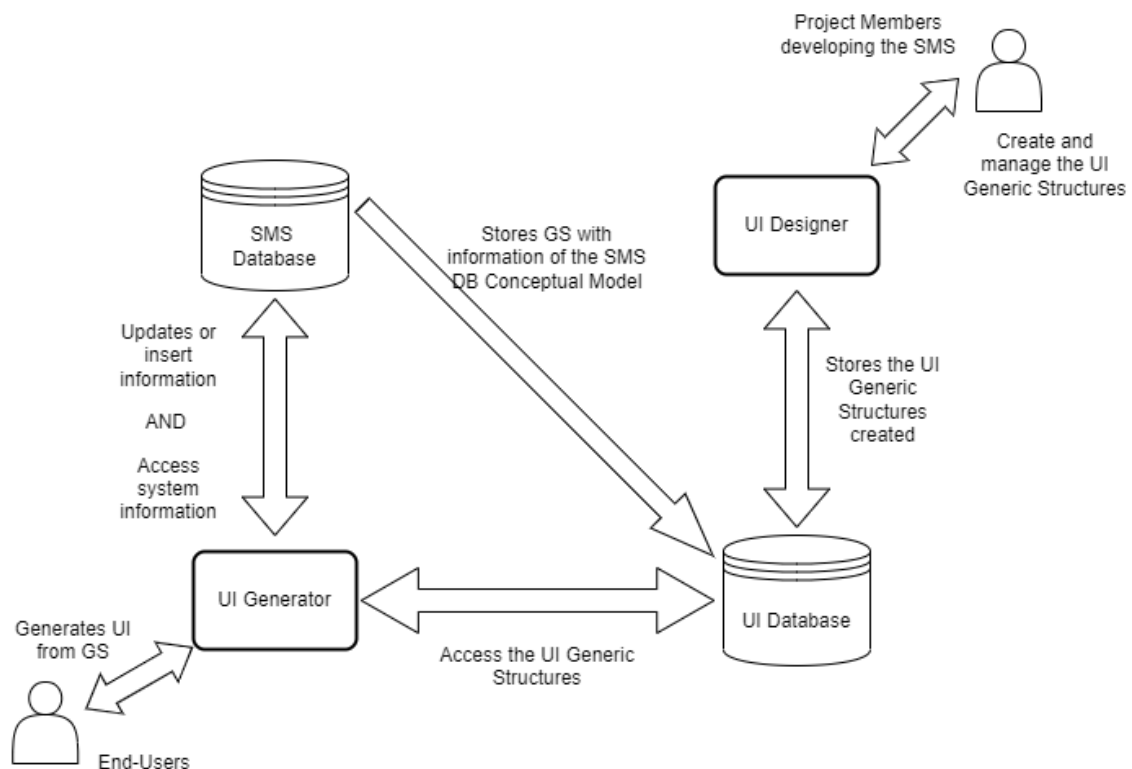


Figure 4.5: Diagram of the Architecture to design and generate UI

The application destined for end Users will use the information contained in the GS of the UI DB and generate Adaptive and Reactive UI. The UI will interact differently with the system DB depending on the interface designed and its related event. Some of these UIs might be designed to alter, insert or delete information from the system DB, while others have the purpose of retrieving data from the system so the User can monitor certain aspects of the SMS.

## 4.6 Technologies used

Since a team of several people is developing this project, this dissertation work was built to incorporate other functionalities of the SMS, either previously developed, in development or already

planned. To this extent, the project's development imposed the following technologies while developing the architecture presented.

#### **Embarcadero Delphi**

Embarcadero Delphi is an Integrated Development Environment(IDE) used to create desktop applications [24] and was used to develop the application to design the GS used to generate UIs. It allows to easily connect to the SMS DB locally, which was done without encryption since it will only be used by project members.

#### **JSON**

JSON (JavaScript Object Notation) is a lightweight approach to format data [25] where an object is composed of key-value pairs where the value can be another JSON object, a JSON array or a primitive data type. The UIs available for end Users need to be more secure since every employee in the factory will be able to use them. To this extent, the SMS DB was accessed through a web server that received an encrypted query and returned an encrypted JSON table.

#### **Java and JavaFX**

Java is an object-oriented programming language mainly used in game design, and desktop application development [26]. The application that allowed the generation of the designed UIs was developed using Java since it is a programming language with several components and libraries already developed. Since the UIs generated are meant to be Graphical User Interfaces (GUI), a library is required that provides the ability to do that. In this case, JavaFX was used since it has specifically developed components for Java and is incorporated in the IDE used to program and compile java code [27]. JavaFx is also the best-suited solution since it has recent updates and is intended to have more in the future as Java progresses, and for other reasons that will be explained in chapter 6.

## **4.7 Summary and observations**

In this dissertation, an architecture to design and generate UIs was developed in the project's scope. The presented GS will be used differently in two scenarios in this architecture. The first one detailed in this chapter is that GS will store the information of the SMS Conceptual Model. The second scenario will use the GS to store information necessary to generate UIs, which will vary according to the GS nodes' type and other node attributes. The combined use of the information from the SMS DB will enable the creation of UIs capable of visualising the system information regarding instances of different entities and their relationship.

Additionally, these GS will also contain information that will make it possible for the generated UI to convey to users the ability to navigate through the system information easily.

# Chapter 5

## Generic Structures for UI

### Contents

---

<b>5.1</b>	<b>Introduction</b>	<b>25</b>
<b>5.2</b>	<b>Generic Structures for UI</b>	<b>26</b>
<b>5.3</b>	<b>Node Pattern</b>	<b>26</b>
5.3.1	Select Pattern	26
5.3.2	Insert Pattern	28
5.3.3	View Pattern	29
5.3.4	Delete Pattern	30
<b>5.4</b>	<b>Conceptual Model to Store the UI GS</b>	<b>31</b>
<b>5.5</b>	<b>Relationships between UI GSs</b>	<b>32</b>
<b>5.6</b>	<b>UI GSs Syntax Rules</b>	<b>33</b>
<b>5.7</b>	<b>UI Designer</b>	<b>34</b>

---

### 5.1 Introduction

This chapter aims to define and explain in detail the Generic Structures used to store information of the UIs that need to be generated for the SMS. These GSs will be stored in the UI DB mentioned in the previous chapter, despite being far more complex than the GS used to store the SMS DB model and stored in different DB tables. First, it will be explained how the UI GSs will be built, the added attributes from the previously mentioned GS, and what rules must be followed to avoid ambiguities in the interface generation. This chapter also aims to explain what type of interfaces can be obtained using this GS for UI and how their attributes will allow navigating within the system information. Finally, it will be introduced the application that provides users the ability to create and manipulate these GS for UI.

## 5.2 Generic Structures for UI

The nodes of the GS used to store information necessary to generate UI also have the attributes presented in Table 4.3. Additionally, they have other attributes that will allow determining what type of UI their information will generate. These GS for UI also represent an event or set of events that can access the SMS information. These events require User interaction and can consist of simple queries to the system entities instances or manipulation - deletion or insertion - of that information.

The nature of these events is defined by one of the added attributes to GS for UI, the Node Pattern, which is the most relevant node attribute next only to the node type. The following sections will detail the possible Patterns a node can take as well as the other node attributes, some of which are specific to distinct node types.

### 5.3 Node Pattern

The GS must be able to represent any event or set of events through a UI available in the system, executed by the UI Generator. These events' influence on the SMS DB is directly related to the node pattern of the structure or substructure that consists of that event. The node pattern can assume one of four possible values - Select, Insert, View, Delete. The symbology used to represent them in GS visually and the value they take in the GS node attribute can be seen in Table 5.1.





Node Pattern	Code	Visual Representation
Insert	INS	
Select/Search	SEL	
Delete	DEL	
View	VIEW	

Table 5.1: Different patterns for Entity Nodes

#### 5.3.1 Select Pattern

This Node Pattern is used to search for instances contained in the system's DB or relationships between entities instances and between entities instances and primitive data values. Nodes with this pattern that are either Entity or Data Nodes require input from the user to select an entity instance or insert a value, respectively. The following figures show examples that display what would be the outcome of events represented by GS that only contain Nodes with the Select pattern. The scenarios analysed considered the SMS DB instances presented in Figure 4.3.

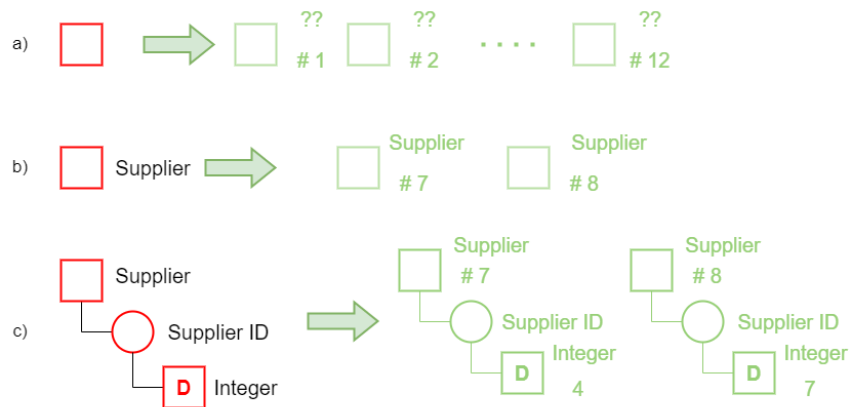


Figure 5.1: Simple Examples of GS for UI using Select Pattern Nodes

Figure 5.1 presents three simple examples that do not require User interaction for the UI GS to obtain the information of the instances in the system's DB presented in the instances' virtual GS in green. Example a) displays the most basic GS that can be made where a node only has its Type and Pattern, resulting in all the instances contained in the DB, regardless of the entity they represent. This is possible due to the numeric identifier unique to each instance in the SMS DB. In example b), the Supplier Entity was associated with the Select Node, and only two instances were obtained, but as before, none of their attributes. In the last of these examples, the UI GS, in addition to the entity Node, had a Role Node for the Supplier ID attribute and the corresponding Data Node containing an Integer Value. The UI GS in c) would provide the user with the same instances as b), but each instance would also provide information about its Supplier ID attribute.

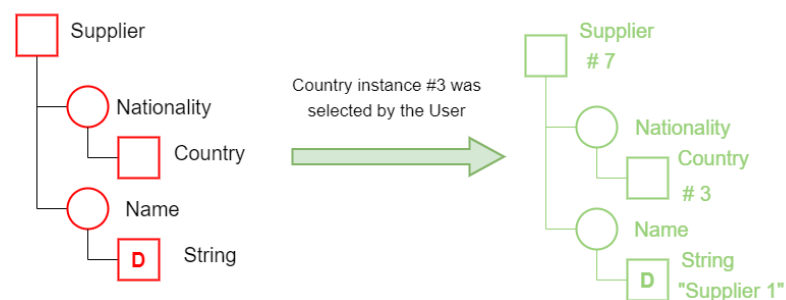


Figure 5.2: Examples of GS for UI using Select Pattern Nodes with user interaction

Since SMS will be far more populated than the Simplified example, it is not always computationally efficient to search all the instances of an entity in the SMS DB since it may produce millions of results. To this extent, the GS nodes have attributes that the generated UI can manipulate to filter the query made to the DB. These attributes are the instances and values particular to Entity and Data Nodes, respectively. It is important to note that how this query is obtained from the GS for UI is out of the scope of this dissertation, and the same applies to those queries used for GS that use nodes of different Patterns.

An example of this scenario is presented in Figure 5.2 where the GS presented would provide

instances similar to the ones obtained in scenario c) from Figure 5.1, but containing the Nationality through the number of the Country instance and the Name, instead of the Supplier ID. However, the scenario presented displays a situation where a user has selected the Country instance #3 for the Nationality. In this scenario, only a Supplier instance fits those requirements, which is the one that will be obtained.

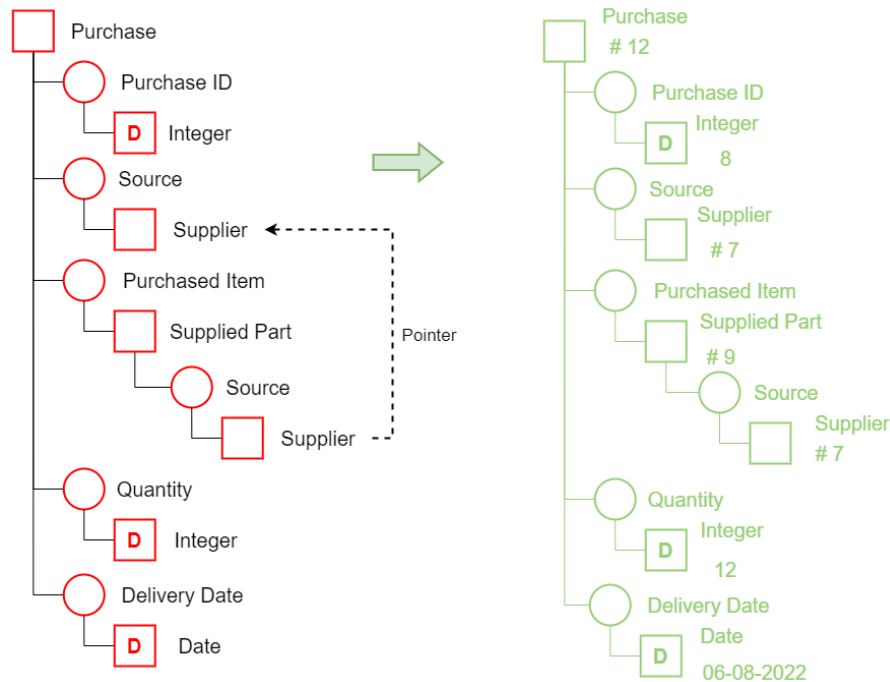


Figure 5.3: Example of GS for UI using Select Pattern Nodes with a Node pointing to another

Figure 5.3 presents a far more complex GS for UI where two Entity Nodes are part of the GS. They need to represent the same instance, but since the GS Nodes cannot have more than one father, two Entity Nodes of the same Entity were created. Nonetheless, they still needed to be connected, which was done by assigning a pointer attribute to the Entity Node that intended to have the same instance as the other. In the example in Figure 5.3 are only obtained instances of Purchases where the Supplier of the delivered part was the same that supplied the purchase, resulting in the instance obtained and seen in the figure.

### 5.3.2 Insert Pattern

These nodes will have the ability to add information to the system. An instance can be created of the type of entity it represents, for the case of Entity Nodes. Also, this pattern can assign to Role Nodes a connection between different instances or values to attributes of an entity's instance.

Figure 5.4 presents UI GSs examples containing Nodes with the Insert Pattern. These types of GS are used to create instances of Entities and, according to the event they represent, assign values to their attributes, or establish relationships with other Entities as desired. In scenario in a), only the instance would be created, ignoring that the Supplier ID attribute is a Primary Key. The



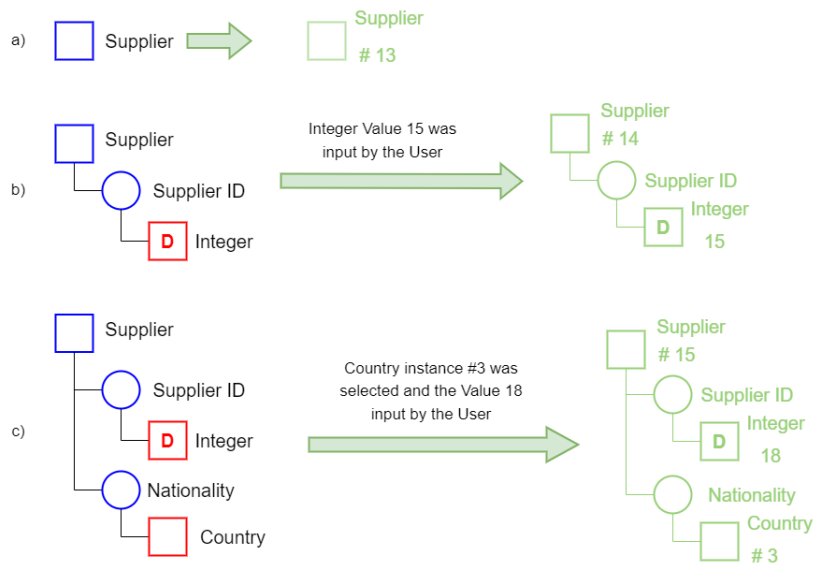


Figure 5.4: Examples of GS for UI using Insert Pattern Nodes

instance created would have all its attributes be null. In b) and c), similar scenarios are presented where the GS was manipulated to assert Values for the Supplier ID. However, in the scenario in c), the user also selected the Country ID.

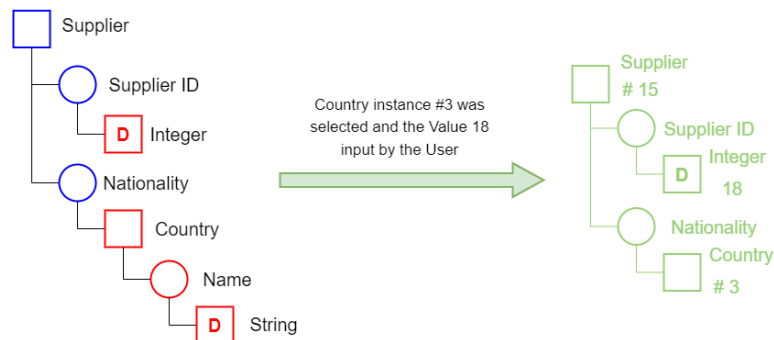


Figure 5.5: Examples of GS for UI using Insert Pattern Nodes, with better selection

In Figure 5.5 is represented a GS for UI that is very similar to the one in scenario c) of Figure 5.4. Although they produce the same result, users' interaction with the generated UI will differ. In the first case, where the Country Entity Node has no children, the user has to choose from a list of instances without any information regarding those instances. On the other hand, when the Country Entity Node has a child Role node and corresponding Data Node, for each Country instance available, the name attribute of that instance would also be available to the user.

### 5.3.3 View Pattern

Similarly to the Select Pattern explained in Section 5.3.1, these nodes are used to search for something that exists in the system. However, the information does not need to exist in the system's

DB for nodes with this pattern, contrary to nodes with the Select Pattern. This restriction applies to every node with this pattern regardless of its type.

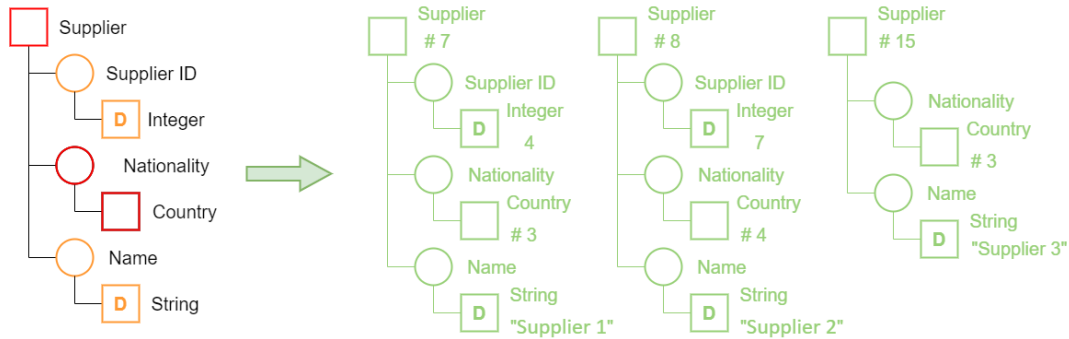


Figure 5.6: Examples of GS for UI using View Pattern Nodes

An example of the distinction between Select Nodes and View Nodes can be assessed from the GS for UI in Figure 5.6. In this GS will only appear the information regarding Supplier instances with a Country assigned to them. If they likewise have values assigned for the Supplier ID and Name, these will be made available to the user. Assuming the examples in Figure 5.4 were executed, and those instances were inserted contained the SMS DB, the one created in a) and b) would not be displayed since they did not have an assigned country entity.

### 5.3.4 Delete Pattern

This pattern is unique compared to the others since it is an extension of another node pattern, the select pattern. This pattern aims to delete entities and relationships from the system DB. However, it extends the select pattern since the user must select the instance it intends to eliminate.

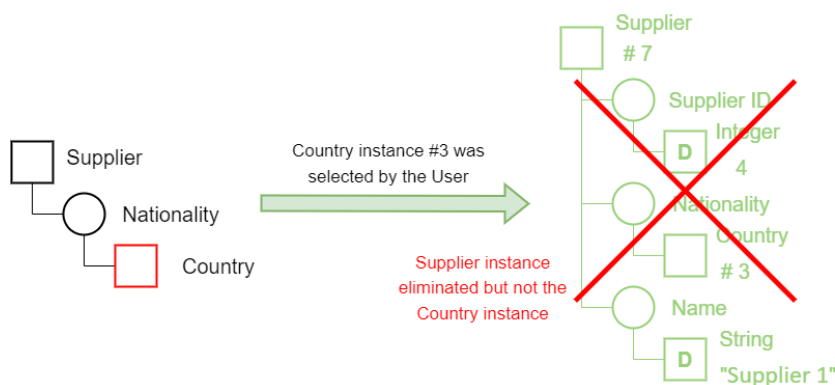


Figure 5.7: Examples of GS for UI using Delete Pattern Nodes

Figure 5.7 is a practical example of this deletion. The presented GS would provide the Supplier instances where the Nationality was the Country #3 if it were composed only of Select nodes. In this scenario, only one instance would be obtained, which is the one presented in the virtual tree on the right side of the figure. However, since the Supplier entity node has a delete pattern, The

Supplier instances obtained will be deleted from the system DB, unlike the Country 3 instance that has a selection Pattern, therefore will not be deleted.

## 5.4 Conceptual Model to Store the UI GS

Asides from the type of node and its pattern, the structure nodes have other attributes that contain information for the interface to be generated. Some of these attributes are used by every single node, while others are only used for certain types of nodes. Table 5.2 contains all the attributes common to all GS Nodes, followed by a description of its purpose.

Attributes	Description	Set of Values
ID	Unique identifier to each GS node	String
Root ID	Identifier of the father node (null in case of the root of the structure)	String
Type	Code of the corresponding node type	ENTITY, ROLE, DATA
Pattern	Code of the corresponding node pattern	INS, SEL, DEL, VIEW
Name	Stores the name of the Entity, relationship or the data type depending on its type	String
Title	Node's distinct title displayed in the interface	String
Order	Node order in the GS the structure relative to its father	Integer

Table 5.2: GS Nodes Attributes common to all Node Types

Attributes	Description	Set of Values
ID	Unique identifier to each GS node	String
Pointer	Pointer that references to an exactly equal node used in the structure whose entity instance must be applied to the node. Stores the ID of the node it is pointing towards	Integer
Instance	Identifies the SMS Entity instance associated with the node	Integer
Master Entity	Used to determine the context of the interface (further explained in 5.5)	Boolean

Table 5.3: GS Nodes Attributes specific to Entity Nodes

Entity Nodes have specific attributes, like the previously mentioned instance attribute that associates the node with a system entity. The Pointer attribute allows the development of more complex UI GSs, and the Master Entity attribute will establish relationships between different GS and events. These attributes will be stored in a different Table not to overcrowd the table containing the vital information of the UI GSs. Each Entity Node will also have a corresponding instance in this table. Attributes specific to Entity Nodes are presented in Table 5.3.

For Data Nodes similar to entities, these must be able to contain system information. The Value attribute serves this purpose since it stores, when necessary, the primitive data value. This value can be of any data type, and the other attributes of these specific types of nodes are used to parse those values to the correct data type and mask the value of this information. Similar to the specific entity attributes, the attributes particular to data nodes are stored in a separate Table, storing the Node ID to establish a connection with the information from the GS Nodes main table. Table 5.4 shows the GS Nodes attributes that are part of this table.

Attributes	Description	Set of Values
ID	Unique identifier to each GS node	String
Data Type	Identifies the type of variables the node represents an integer, float, time, date, timestamp, boolean or string.	\$INT, \$FLOAT, \$TIME, \$DATE, \$DTIME, \$BOOL, \$CHAR
Data Length	Maximum length the value in the node can have	Integer
Data Mask	String representing the mask to apply to the value when displaying it to the user	String
Value	Contains the value of a SMS Entity's attribute	String

Table 5.4: GS Nodes Attributes specific to Data Nodes

Role nodes will have an added String attribute called Inverted Title. This attribute will be used similarly to the Master attribute of Entity Nodes to establish a relationship between UI GSs when its value is not null.

## 5.5 Relationships between UI GSs

The UIs must be context-sensitive, and therefore, users must be able to access events related to the context of the UI. This UI context was defined as the information known about the entity and information of instance being analysed while executing a UI. This context allows Users to navigate through the information contained in the SMS. The context is obtained from these three components:

1. The UI GS root Node Entity.

2. The entity instance information contained or selected in the UI.
3. The type of UI being executed.

The entity of the GS root node is crucial when determining the events that should be available in that interface's context. They are obtained by analysing the system's events' relation with that entity. The root Node characterises each UI GS, and since this root is always an entity Node, the GS is also characterised by this entity. Using the root Entity, users will have access to events with the same entity as the root Node and events whose GS has entity Nodes of the corresponding entity where the master attribute is true. The Master attribute defines which entity masters a specific UI or event GS. For example, an event designed to Create a Supplied Part will be mastered by the Supplier since they will decide which parts they will provide and what their price will be.

Presenting the user with the possible events inherent to the context, it is possible to browse through the information in SMS. However, only providing the user with the possible events would lead to a series of repetitive data insertions since the interface would be generated regardless of the instance being analysed. To mitigate this issue, when events are called while navigating between interfaces, the generated UI will adapt to the instance being analysed using the context information. Depending on the type of UI, the context instance will be gathered differently. It will either be an instance from a list or the instance created or altered in a UI. The UI generation based on the context information will save end Users considerable time through fewer mouse clicks and keyboard strokes.

## 5.6 UI GSs Syntax Rules

Having established the information that has to be stored in each node of the GS for UI, it needs to be defined what kind of UI component each combination of node type and pattern will produce while generating the UI. To avoid errors or ambiguities in the GS created, certain syntax rules must be defined and followed when designing them:

1. The root node of the GS must be an event/entity type node and can have any possible patterns except for the View pattern. This type of node has a higher level of abstraction, meaning it can represent any event or entity with as many levels and components as desired.
2. Data nodes are always leaf nodes of the GS, nodes that do not have any children since they represent primitive data variables.
3. In a User Interface's GS can only exist one root node that will be used to determine what type of UI will be generated.
4. An entity node child must always be a role node and never another entity node, nor a data node.

## 5.7 UI Designer

To create the GSs used to generate UIs, a Desktop application was developed to create and manage these GS for UI. This application is not destined for end-users since they will not have deep knowledge of the SMS Conceptual Model. Instead, this application is destined to be used while developing the SMS, simultaneously creating the necessary GS for UI as the systems events are also created and defined.

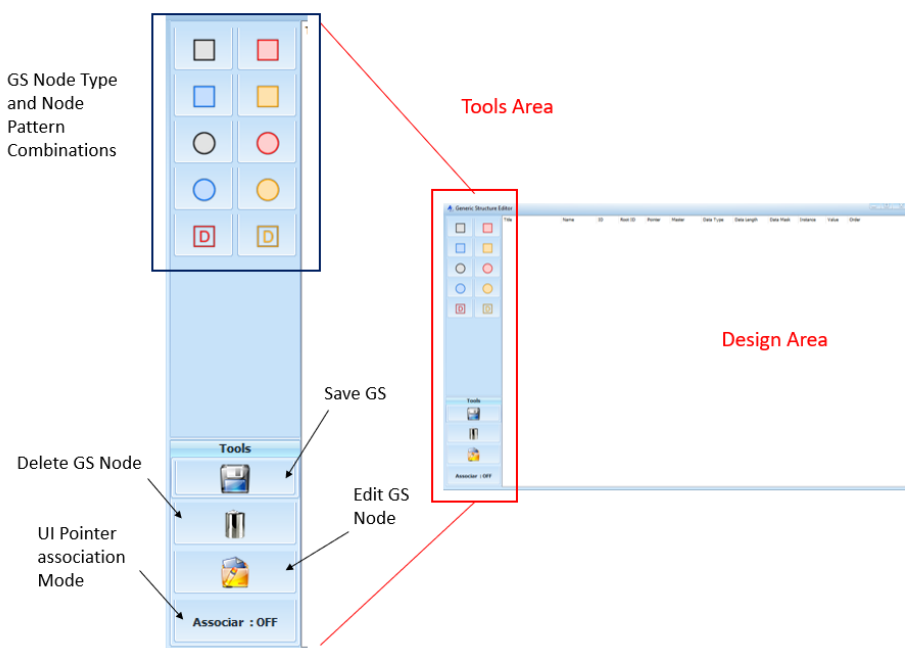


Figure 5.8: Options to manipulate and create GS

The UI Design Application was developed using Embarcadero Delphi as mentioned in 4.6, where a component was used to display a tree structure seen in the blank edit space in Figure 5.8. This component will be populated with the UI GS, displaying its nodes and attributes. To easily design the GS using this application, by dragging the intended node from the Tools area and dropping it on the design area in the appropriate location, adding it to the GS being created or altered.

In Figure 5.8, we can also see the Tools area zoomed in, which allows users to manipulate a GS. This area contains nodes that can be dragged and inserted in the GS and other tools that allow editing the nodes' attributes, deleting nodes, toggling pointing mode on/off, and saving the UI GS. To assign a pointer to a node, the pointing mode must be toggled on and then, by dragging node A to node B, the Pointer attribute in node B is changed to the Node ID value of A.

The nodes added to the design area must represent an entity or relation that is contained in the SMS DB. After dragging a GS Node to the design area, a list of entities or relationships in the system is presented, already adapted to the Node Type and Pattern of the node dragged. An

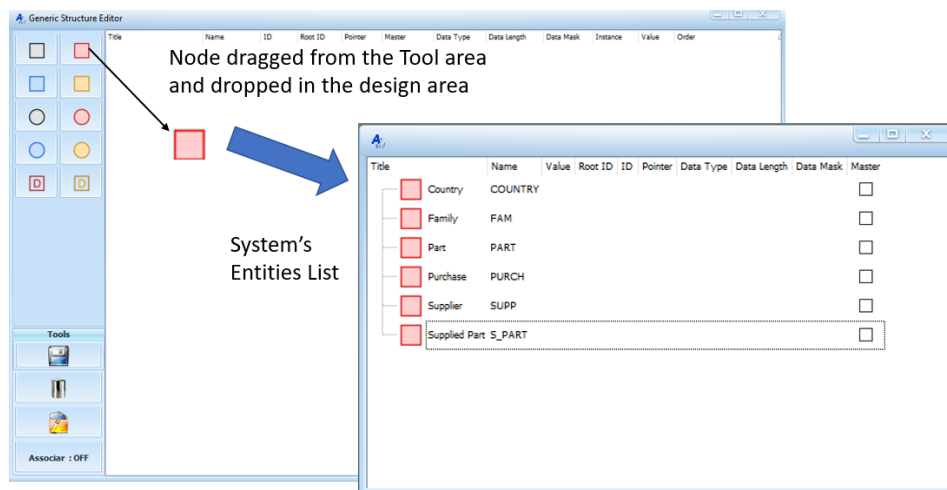


Figure 5.9: Node addition to the GS design area

example of this can be seen in Figure 5.9, where a Select Entity Node was dragged to the design area, and the six entities contained in the simplified SMS were presented in a list.

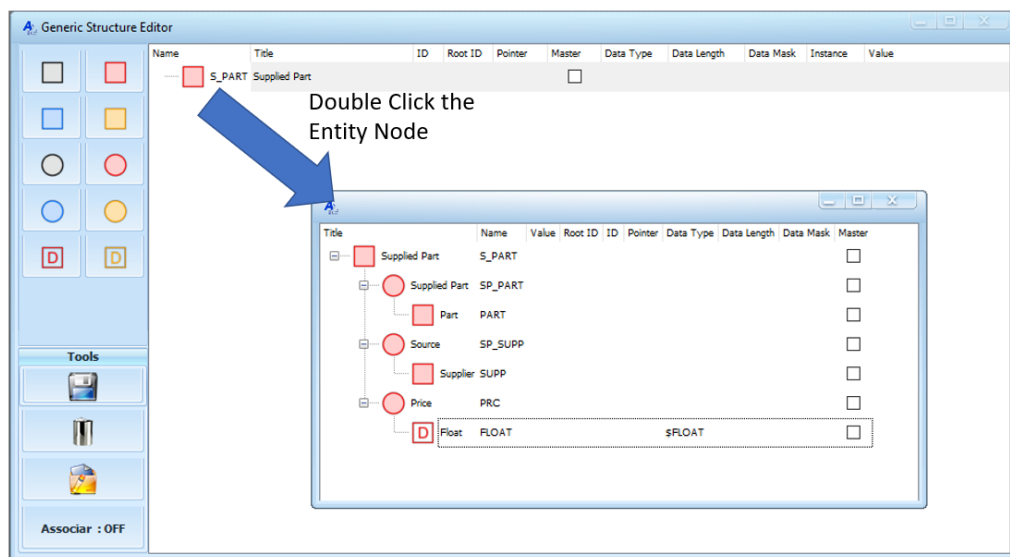


Figure 5.10: Entity attributes selection in UI Designer

To make the design of the UI's GS faster, the UI Designer application provides the entity's relationships and attributes instead of dragging every node necessary for the UI. When double clicked an entity Node, the GS containing its information will be made available so the intended attributes, and relationships can be selected. In Figure 5.10, can be seen that when clicking the Entity Node of the Supplied Part Entity, a GS containing all the attributes and relationships of the Supplied Part entity is made available.

The UI Designer application also allows the user to alter the pattern of the GS Nodes if needed to alter the outcome the GS UI will have in the system. This interaction is made by right-clicking

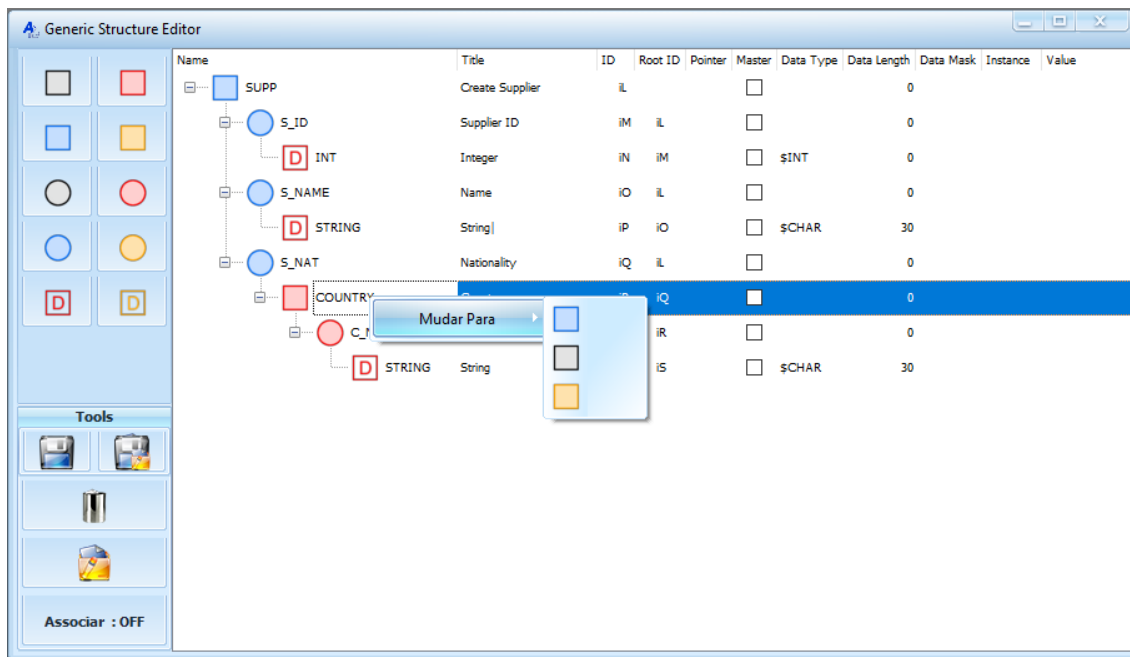


Figure 5.11: GS Node Pattern change

on the node, which pops up a Menu that provides the available patterns to which the node can be transformed. This interaction is portrayed in Figure 5.11.



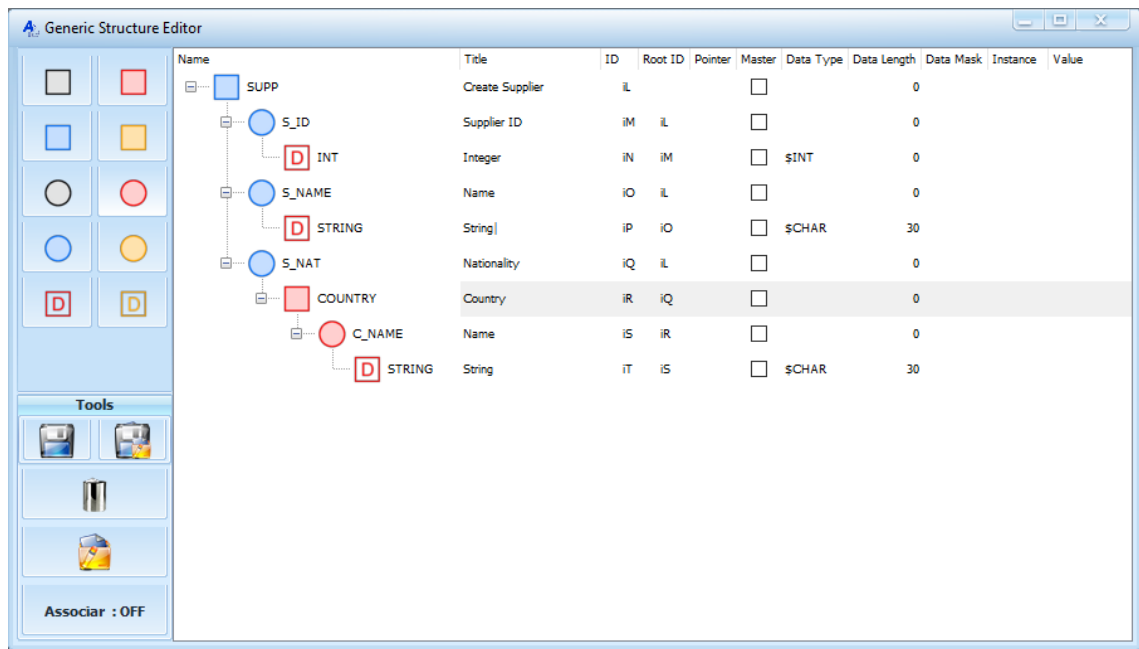


Figure 5.12: Example of a created UI GS

Figure 5.12 shows an example of a generic structure designed in the developed UI Designer application. This event represents the same generic structure presented in Figure 5.5 but was created in the UI Designer. Its nodes have all the necessary attributes and are ready to be stored in the system DB. The Type and Pattern of the node are not displayed since they are implicit by the node's icon and colour.



# Chapter 6

## Types of UI and UI Generation

### Contents

---

<b>6.1</b>	<b>Introduction</b>	<b>39</b>
<b>6.2</b>	<b>UI Components</b>	<b>40</b>
6.2.1	Label	40
6.2.2	Button	40
6.2.3	Text Field	42
6.2.4	Date/Time Picker	42
6.2.5	Toggle	43
6.2.6	Table View	44
<b>6.3</b>	<b>Type of UI Templates</b>	<b>45</b>
6.3.1	Execution UI Template	45
6.3.2	Search UI Template	46
6.3.3	Select UI Template	46
<b>6.4</b>	<b>UI Generation</b>	<b>47</b>
6.4.1	Execution UI	48
6.4.2	Search UI Generation	49
6.4.3	Selection UI	50

---

### 6.1 Introduction

UIs can be generated from the information in the Nodes of the GSs modelled in the previous chapter. To enable this generation, an application was developed that uses the GS modelled to generate UIs for the SMS. These UIs will allow End Users to access and manipulate the information contained in the SMS DB. This application, with the goal of generation Adaptive UIs, was developed using JavaFX since it contains packages that allow the development of GUI using Java.

This chapter aims to present the JavaFX components found, adjusted and developed that will make possible the association between the GS Nodes and UI visual elements. Another objective

of this chapter is to define the type of UIs that will be generated and their base characteristics. Finally, a connection will be established between the type of UI and GS particularities and how the UI generator will use these GS to complete a UI Type Base and generate a final interface available for End Users.

## 6.2 UI Components

The decision to develop the application that generates UIs in JavaFX was influenced by how its components are organized. In JavaFX, each interface has a scene where all the interface's visual elements are stored. Similarly to the GS defined in chapter 5, these elements are stored in a tree structure, where each JavaFX node represents a UI component. This architecture is a huge advantage that will allow to easily transform the GS's nodes into UI components and define its layout.

The appearance and cosmetics of a UI play a huge role in assessing the users' enjoyment and usability, and therefore it influences their productivity. To this extent, attention was paid to making the UIs as simple and engaging to the user as possible by researching for components' libraries that met these requirements by using simple but appealing components. On top of that, CSS (Cascade Style Sheets) files were used to style these components to make them even more appealing and in sync with the whole interface.

The following sections will present the UI components found, adjusted, and those required to be developed to generate a UI from a GS. A relation will also be established between these components and the particular GS node they will be created from. Examples of these components are action, text, input and list components.

### 6.2.1 Label

This UI component was used to display text in the generated interface on three different occasions. It was used mainly to provide Users information regarding the relationship between entities and other entities or their attributes so it will be generated mainly from GS Role Nodes. When the GS root node is a Select Entity Node, a Label is used to convey User information on that entity. Finally, the Title of the GS or the Event Name it represents will also be generated using a Label.

Depending on its purpose, the Label's font, size and colour will be customized, as can be seen in Figure 6.1 where the three situations described are presented and the GS nodes that originated it.

### 6.2.2 Button

A Button is a UI component with the ability to convey an action when clicked by the user, reacting visually as well when clicked. Buttons were used in two different scenarios. The first one is when GS had Select Entity Node, meaning the user will have to select an Instance of that entity, and this

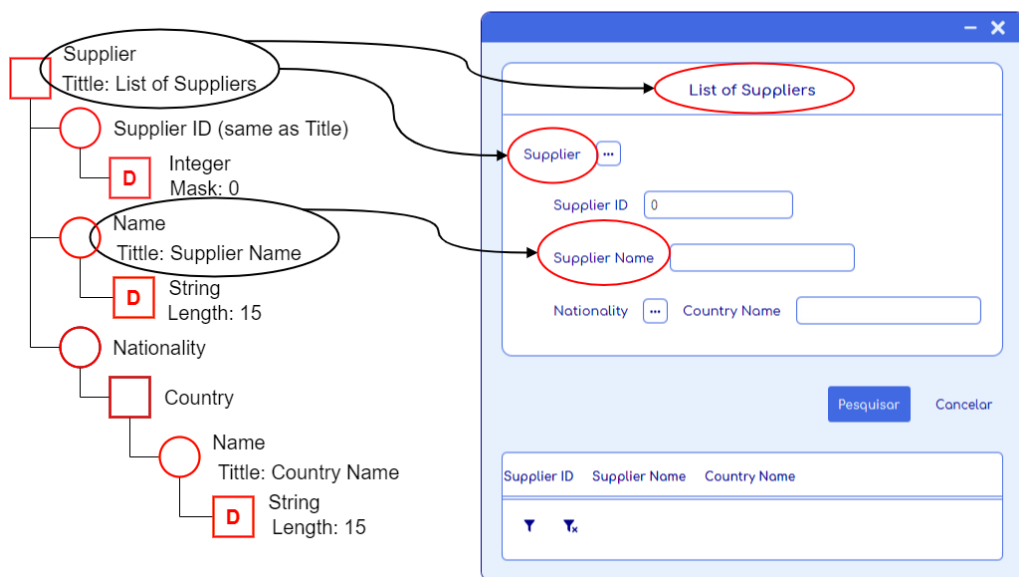


Figure 6.1: Label usages for UI generation

button is generated so the user can begin that selection. Buttons were also used to generate the UI components that allowed users to execute the event related to the GS that originated the UI.



Figure 6.2: Button usages example for UI generation

This UI component was also styled according to its purpose in the interface. Figure 6.2 presents an example of these two applications, where the button with "..." is used for users to select an instance, and the one with the Blue background executes the event.

### 6.2.3 Text Field

The most basic and flexible way of data introduction is inserting text by typing on a keyboard or clicking on a virtual one. A Text Field element contains this functionality and is the easier way of introducing either text or numbers. A Text Field was used to generate UI components to introduce String, Integers and Float type variables. However, the component was adapted and styled according to the variable type, length and mask.

This component is generated from Data Nodes in GSs since these Nodes are meant to represent primitive data Values. The functionalities and restrains of these components will be asserted by the attributes of the Data Node, namely the Data Mask, Data Type, and Data Length attributes.

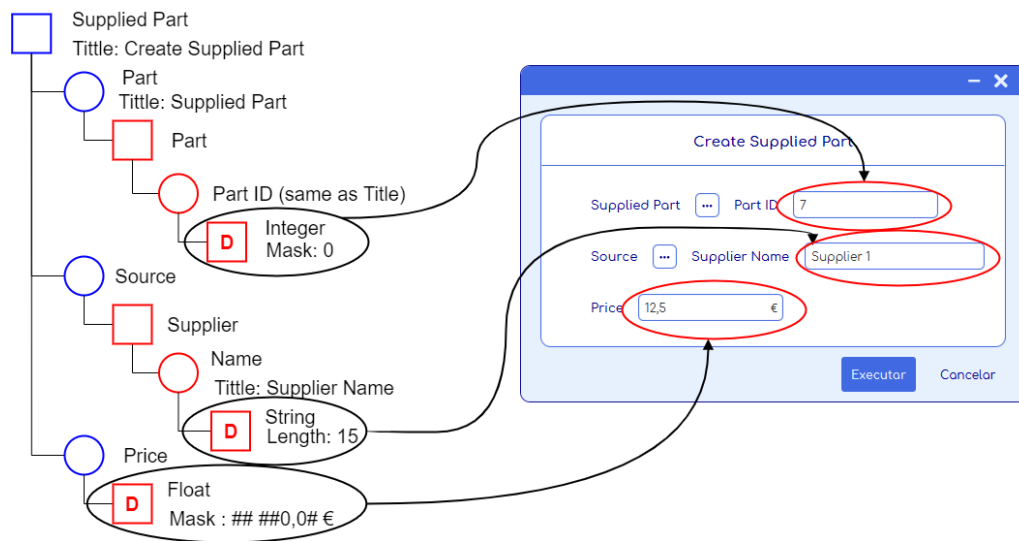


Figure 6.3: Different Text Fields Generation Examples

Figure 6.3 is an example of a UI containing Text Fields representing Integer, Float and String Data Nodes. String Data Nodes are only restrained by their length. For float and integers data types, the Text Field is restrained to digits and decimal or group separators defined by the Data Mask. Can also be seen from the mask assigned to the Float Data type in Figure 6.3 that this mask can also contain a unit value, which is the euro sign (€) in this case.

### 6.2.4 Date/Time Picker

Data Nodes with Date, Time or Date Time data types can also be represented through String values. However, they have specific rules that all Users might not know and can easily be misused. To solve this problem, Date and Time Pickers were used to let Users select the desired Time or Date through an interactive Clock or Calendar. These UI components are called Time and Date Pickers. Although they display the information similarly to Text Fields, their interaction with the user is significantly different.

Figure 6.4 shows GS built to exemplify the outcome of these data types from Data Nodes, and it would not be able to influence the system Data Base since no entity had the characteristics

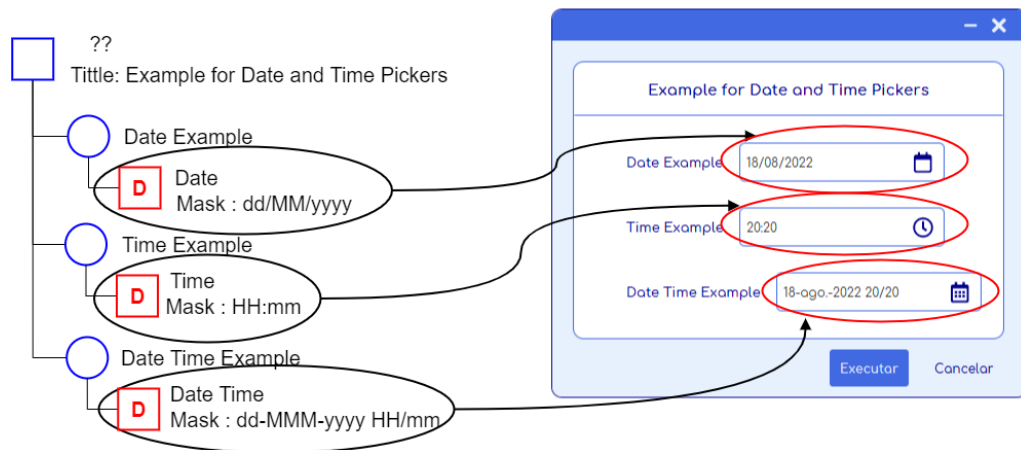


Figure 6.4: Time and Date Pickers generation Example

displayed in that GS. In the pickers example, the mask's effect can also be seen when representing Time and Date values, changing how the month, year, separators, and other aspects of those data types are displayed in the UI.

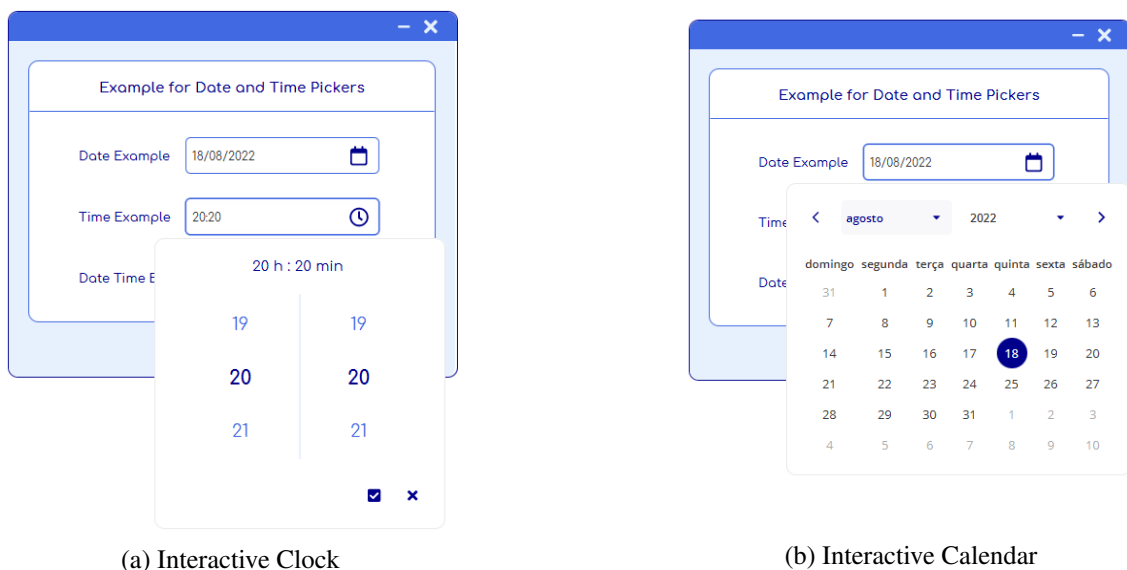


Figure 6.5: Interactive selection elements in Date and Time Pickers

A different icon can be seen in every unique picker, used to distinguish them. Additionally, when clicked, this icon opens the interactive Calendar or Clock that allows Users to select the data value. These interactive selectors can be seen in Figure 6.5.

### 6.2.5 Toggle

With the UI components presented above, there is only one data type that was not mentioned, which lacks a dedicated component, Boolean values. For this data type, was used a Toggle Button,

a component that, with a simple mouse click, can toggle between two values, in this case, True and False.

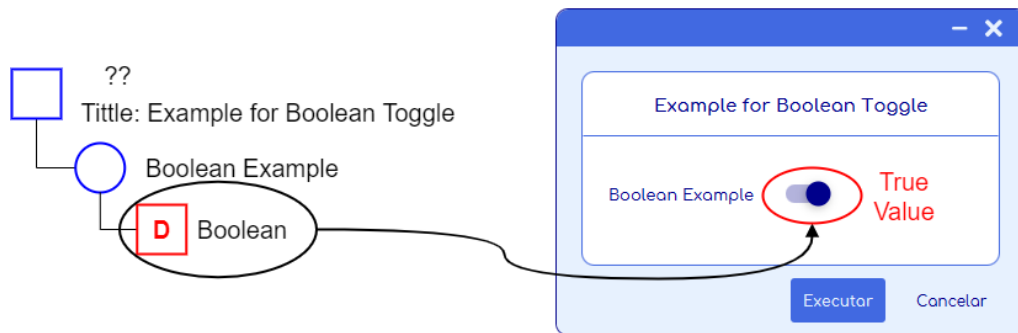


Figure 6.6: Toggle Button generation Example

Figure 6.6, shows a mock GS that was simply built to exemplify the outcome of a UI component generated from a Data Node of boolean data type. Similarly to the GS used to exemplify the Date and Time Pickers, this GS would not be able to influence the system DB. The Toggle Button presented has a True Value.

### 6.2.6 Table View

All the previous components are generated from a single GS Node, although there is still the need for another UI component that will be generated from the whole GS rather than just a particular node. This UI component must provide the user with a list of instances with a variable number of columns, the one used was a Table View.

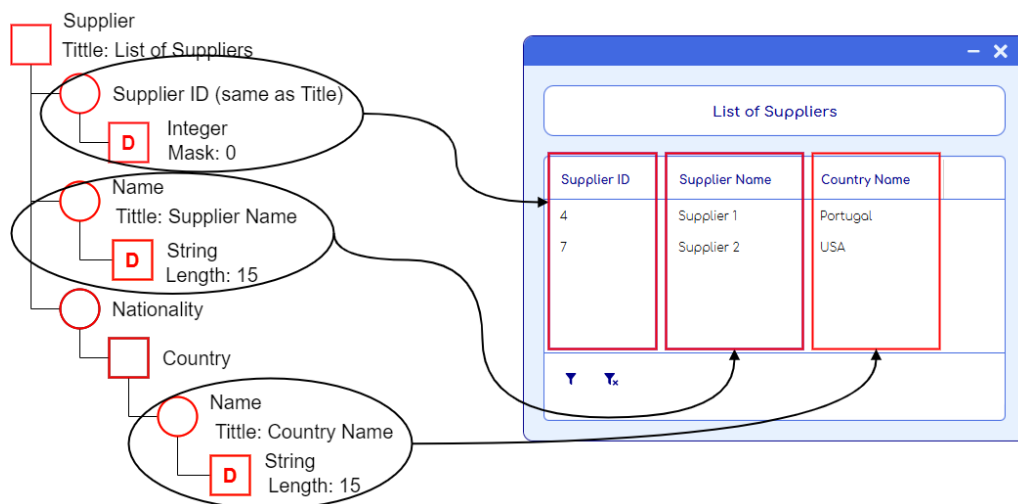


Figure 6.7: Table View generation Example

Table Views will be generated from a UI GS and will have as many columns as the variety of data the UI intends to display. This variety will amount to a number of columns equal to the number of Data Nodes in the GS. An example of this generation can be seen in Figure 6.7, where



a Table View with 3 Columns was generated. The column title is asserted from the data node's father node's title and the column's data type from the data node's data type.

Additionally, the Table View also provides the ability to order the list by each column and apply filters to its content. Each column will also mask its content, depending on the information on the corresponding data node's mask.

## 6.3 Type of UI Templates

The UI elements constructed from the GS nodes will have to be added to a container that will arrange them accordingly. This container varies according to the interface type, and for each one of those, a UI Template was developed, a base UI to be completed with components generated from a GS.

The type of UI is asserted from the combination of the Node patterns that compose a GS and from where it is generated. Two different types of UI can be asserted from the GS stored, Execution and Search UIs, plus a third one used by the other two types of UI, the Selection Interface. The UI asserted from the GS information will depend on whether the GS's event is merely a query to the system DB or a manipulation of its information.

### 6.3.1 Execution UI Template

GSs that simultaneously contain Nodes of select and insert and/or delete Patterns represent Events that intend to alter the information in the System DB. Therefore these GS will generate Execution UIs.

The execution UI has a Label with the event's title that is portrayed to provide context to the user. The bottom contains an execute button to perform the action intended with the GS's event and a cancel button to abandon the event and close the UI.

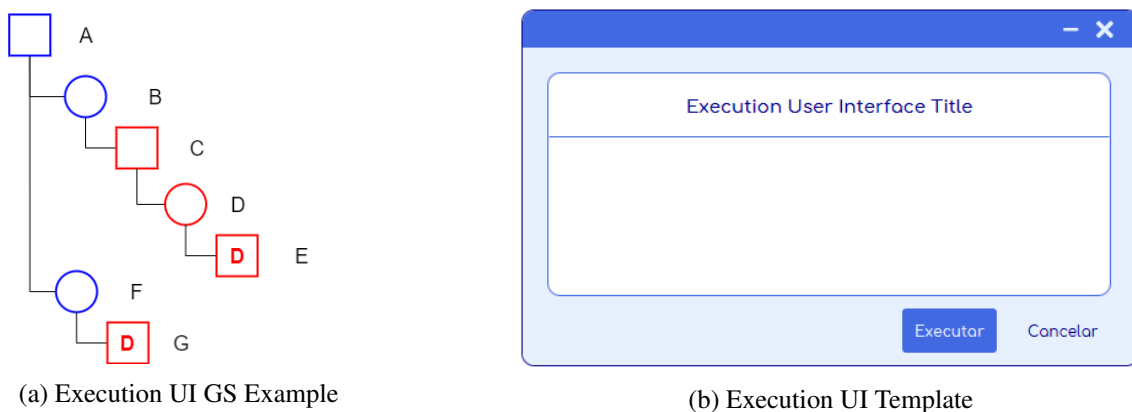


Figure 6.8: Execution UI GS Example and Template

Figure 6.8a contains a GS that generates an Execution Interface, and Figure 6.8b, an Execution UI Template without the components generated from the GS. This UI will block the UI components when executed, in case its execution is successful.

For this type of UI, the context instance is obtained after the UI is executed, and it is the instance that was created or altered when this execution is successful.

### 6.3.2 Search UI Template

Events with the primary purpose of only accessing information in the DB will generate Search UI. These events are translated into GS with Nodes with only the Select or View pattern. The access to the system DB information will return a table of instances that will use a Table View to provide Users with this list.

However, as mentioned in 5.3.1, users have to be able to filter the query made to the system DB. To this extent, GSs that generate this type of UI will be used to generate the Table View that will be populated with the result of the DB access, but also the components needed to filter the information that will be requested from the DB.

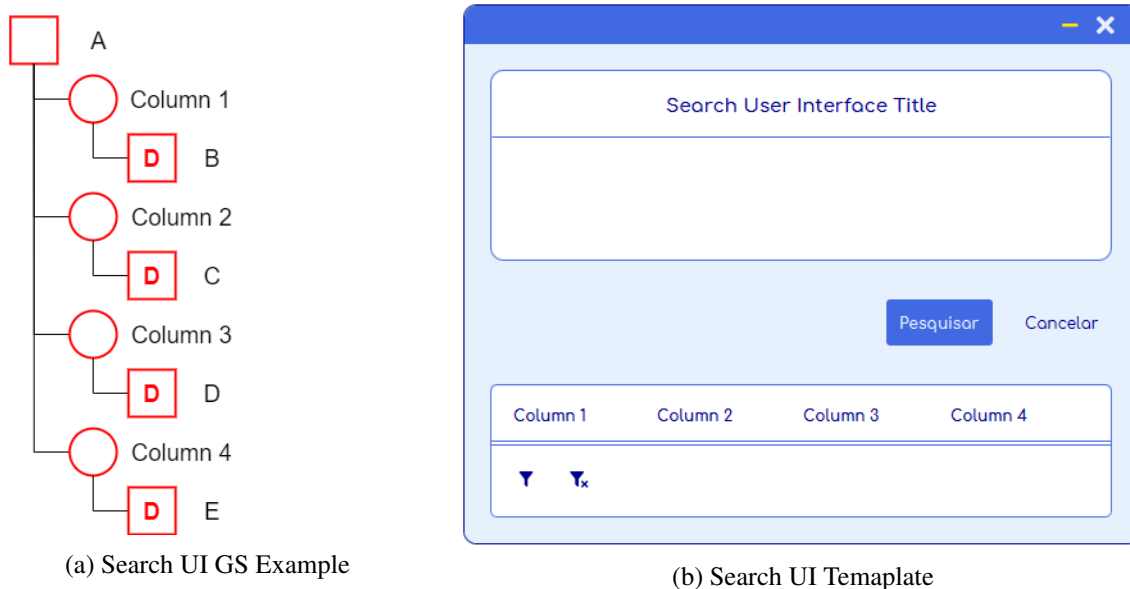


Figure 6.9: Search UI GS Example and Template

Figure 6.9a presents an example of a GS that would generate a Search UI. The generated UI would derive from the Search UI Template in Figure 6.9b that, when executed, would only show the Table View, hiding the remainder of the UI components generated from the GS.

For this type of UI, the context instance is obtained after the UI is executed, and it will assume the value of the instance associated with the selected Table View Row.

### 6.3.3 Select UI Template

This type of UI will not be generated from a particular GS saved in the UI DB. Instead, when an entity selection Button is clicked, the structure that derives from that Entity Node will be used to generate this type of UI.

This type of UI will be used to select an Entity instance requested by another UI. Since the generated UI will have the context information, the interface generated will be similar to the resulting UI of the Search UI after executing it. The only UI component derived from these Pseudo GS, will be a Table View already containing the list of instances.

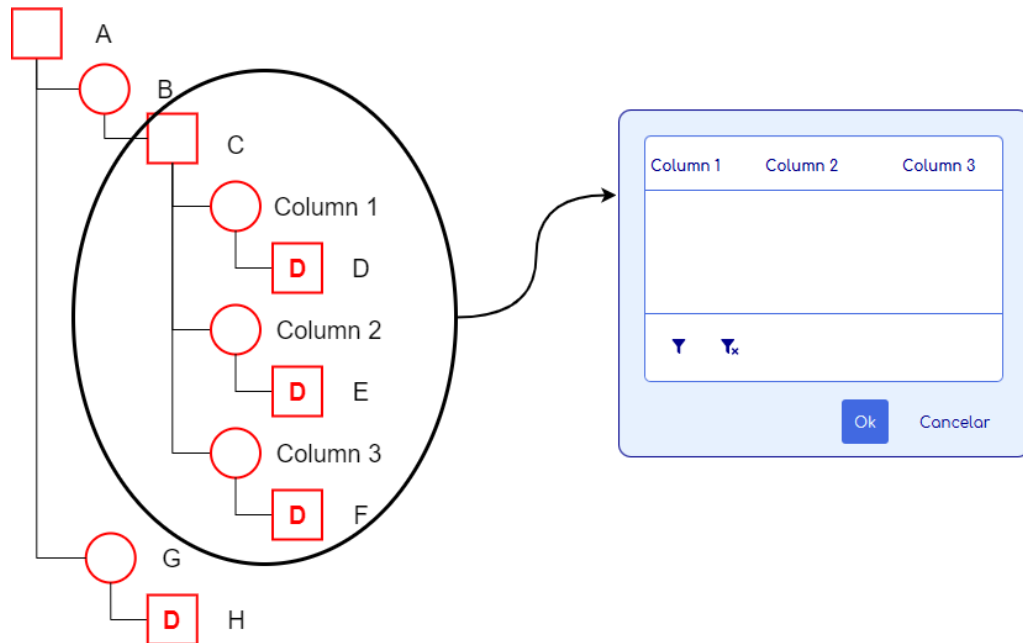


Figure 6.10: Selection UI Pseudo GS Example and Template

Figure 6.10 shows an Example of an Execution UI GS and the sub GS used to generate the Selection UI. It also represents the Template for Selection UI, where in the bottom right is an action Button. However, since these pseudo GS do not have a corresponding event, the action button serves to confirm the selected instance. This selection can also be made by double-clicking the Table View Row of the desired instance. Upon selecting the desired instance, this UI will automatically close and pass the chosen information to the UI that requested it. The instance context will be obtained similarly to the Search UI, but right upon its generation since the Table View is populated immediately.

## 6.4 UI Generation

Having defined the Type of UI that will be generated from UI GS, UI templates will be completed with the UI components generated from GS when generating the final Interfaces. The generation and placement of the UI Components in the UI templates will vary depending on the type of UI template being used. The following sections will explain the layout and generation of each of the three types of UI defined above.

### 6.4.1 Execution UI

The GS creation gives users designing them a high level of freedom that might produce somewhat confusing UI if the layout of their components is not carefully sought out. Since the different GS Nodes can have any Title, the layout of the UI components that will be generated from these Nodes is critical to allow for a clear interpretation of the UI generated. A strategy was used to define the layout of the UI components generated, composed of the following rules:

1. A UI component generated from a Role Node will always be positioned in one of two locations relative to the component of the corresponding father node. (a) Right next to it, in the same horizontal axis, if it is the only child Node; (b) Bellow it with a certain indentation to identify it as something that is part of the father Node Component.
2. UI elements that are meant to insert data (i.e. those that derive from Data Nodes) are always placed right next to their father node to identify the intent of information being inserted by the user.
3. The order in which the elements appear is defined in the GS Node, where a Node's children will be arranged by their Order value.

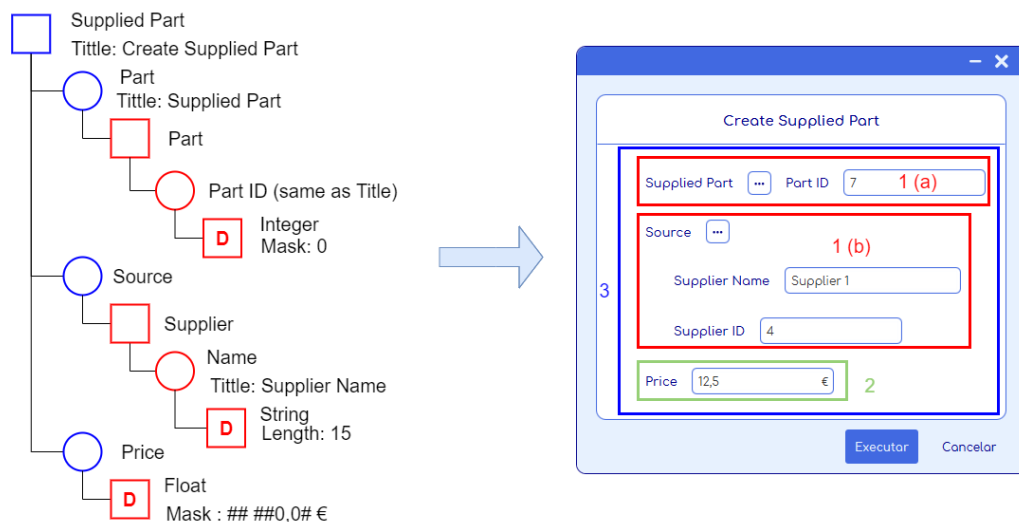


Figure 6.11: Execution UI generation Example and Layout Rules

These layout rules can be observed in Figure 6.11, where examples of those rules are identified and highlighted. The Figure also contains the GS that originated the UI. The example in Figure 6.11 represents an Execution UI generated from a GS that represents an event that enables Users to create Instances of the Supplied Part Entity.

It can be seen from the examples previously presented of generated UIs, that the size of the area containing the UI components generated from GS nodes adapts to its content size. If the GS is so complex that the generated UI components layout area grows so considerably as not to

fit the Screen, this area was designed using a component that enables its content to be scrolled horizontally and vertically. This way, a GS can have an infinite number of nodes, despite the generated UI not being able to display all the corresponding UI components simultaneously.

### 6.4.2 Search UI Generation

As mentioned before, this type of UI has a similar area to Execution UIs, containing the GS Nodes generated UI components. These components enable Users to filter the search query made to the system DB. The layout rules used to fill this particular area are the same as the ones mentioned above, but with an added feature to only show a few of the UI components contained in the area.



Figure 6.12: Search UI generation Example

Figure 6.12 displays the example of a Search UI generated from a UI GS that represents an Event that consists of listing Supplier instances contained in the system DB. In the generated UI, can be seen that not all Components that were supposed to be generated are visible, and only the first few are. Below the components Area and the button to execute the GS event, there is a preview of the Table View containing only its columns. This Table View will be populated once the UI is executed, adding the obtained instances.

Some components are hidden in this type of UI since users might not always intend to analyze and manipulate all the search GS functionalities or even intend to filter the search query. However, all the components are still generated and can become visible by clicking the button at the bottom of this area, showing the rest of the filtering options. This interaction can be seen in Figure 6.13.

After executing the event intended with the GS of this type of UI, without filtering the search query, the layout will change. The Search UI will then show only the event's title and the Table

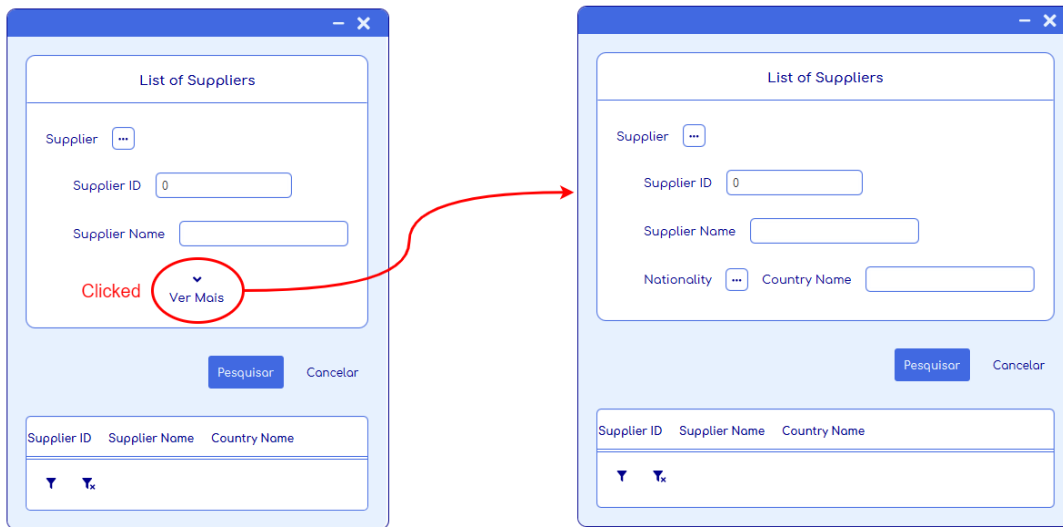


Figure 6.13: Search UI Components Area interaction

View containing the instances obtained from the system DB, hiding the remaining UI components generated previously.

The Table View populated with the system information allows users to order the information by the value in a specific Column or even apply filters to the Table information to narrow down the instances available further. In Figure 6.14 can be seen this interaction with the populated Table View, where the Supplier Name column is ordered decreasingly. On the right, the Table View's component allows the application of specific filters to the desired column.

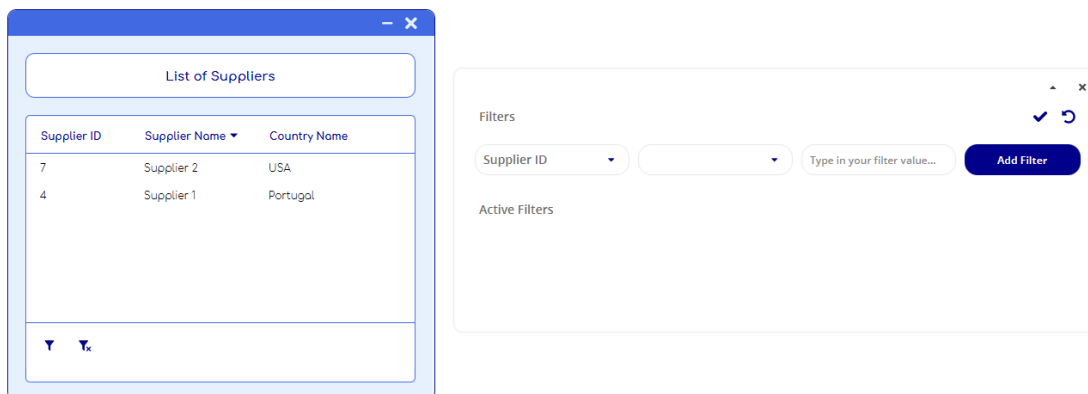


Figure 6.14: Search UI Filter and Ordering

### 6.4.3 Selection UI

Selection UI will be similar to Search UIs obtained after their event is executed. This resemblance is due to Selection UI being generated from a pseudo GS that is part of the GS used to generate the UI requested for the Selection UI generation. In this case, the pseudo GS will use the values and instances users impose in the UI GS when the selection is requested.

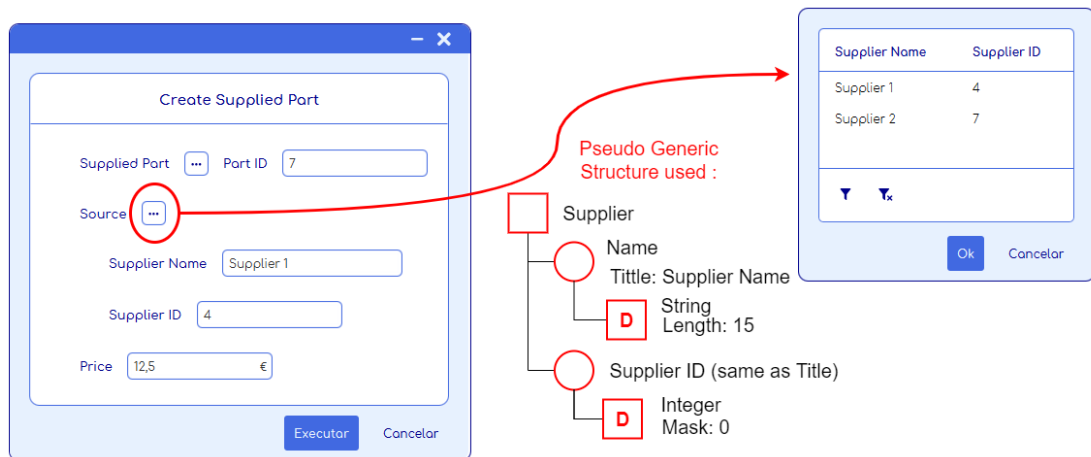


Figure 6.15: Selection UI generation Example

Figure 6.15 shows a scenario where a User intends to Select a Supplier instance when using the Execution UI from Figure 6.11. It can be seen the pseudo GS, part of the Execution GS, that was used to generate the Selection UI when the Node Entity UI Button is clicked.





# Chapter 7

## Navigation and Visualisation

### Contents

---

<b>7.1 Introduction</b>	<b>53</b>
<b>7.2 Navigation in Context Sensitive UIs</b>	<b>53</b>
7.2.1 Navigation Menu	54
7.2.2 Navigation between UI	56
7.2.3 Navigation in different Types of UI	57
<b>7.3 Generic Diagram UI</b>	<b>59</b>
7.3.1 Diagram Cells	59
7.3.2 Visualise Information	60
7.3.3 Diagram Model	61

---

### 7.1 Introduction

Previous Chapters focused on the definition of the GS used to store UI information and the generation of UI using these structures. This Chapter presents how users can navigate between UIs and browse the system's information. It explains how the information in UI GSs defined in Section 5.5 is used to adapt the navigation of each UI.

A different kind of UI was also developed to allow Users to visualise information of several instances and their relationships, all in a single UI, the Generic Diagram UI. Finally will be discussed how the UIs developed will adapt to the user and how the user can adapt the UI to its preferences.

### 7.2 Navigation in Context Sensitive UIs

The UI generation was developed so the obtained Interfaces would have an additional component allowing users to access similar or related events. This component is a Menu that contains the

available UI that represents these events. This menu is context sensitive and uses the information from the context defined in 5.5 to obtain the list of related UIs and allow their generation.

The following sections will explain how this menu is constructed, how users can navigate using this menu, and how these menus will be generated according to the type of UI.

### 7.2.1 Navigation Menu

The Navigation Menu will be created using the Events gathered from the context of the UI. These Events are all related to the entity gathered from the context. The events available are also grouped into five different categories that will be presented in the following paragraphs.

#### A. Search

This category will contain events that have GS that will generate Execution UI. These GS must have an Entity Root Node with the same entity as the UI context. An example of GS that would be a part of this Menu category is presented in Figure 7.1 when the context Entity is the Supplier Entity.

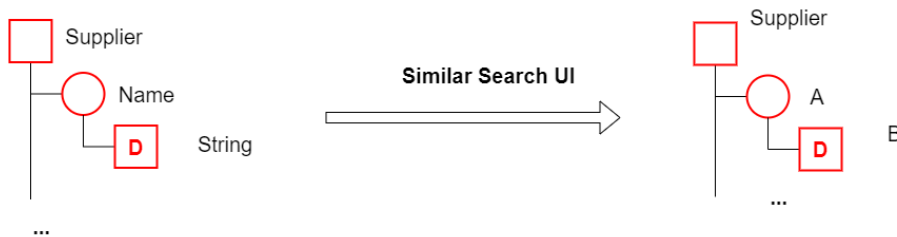


Figure 7.1: Example of GS for the Menu Search Category

#### B. Dependencies

This category contains events from Search UI generated from GS that contain a Role Node with a non-null Inverted Title and an Entity Child Node with the same entity as the one gathered from the UI context. This scenario is described in Figure 7.2 that contains a GS that would be part of this category when the context Entity is the Supplier Entity.

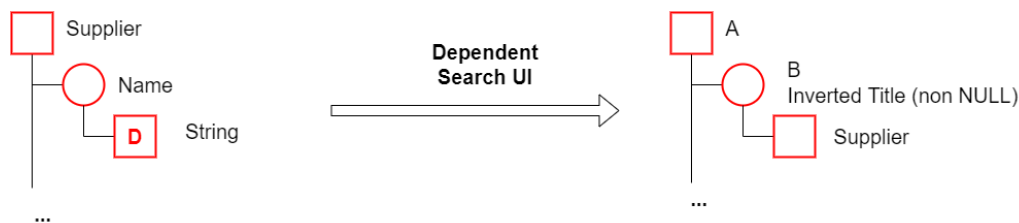


Figure 7.2: Example of GS for the Dependencies Search Category

#### C. Detail

This category contains events from Execution UI whose GS contain an Entity Node with the Same Entity gathered from the UI context. An example of an Execution GS that has a Supplier

Entity Node as Master is presented in Figure 7.3.

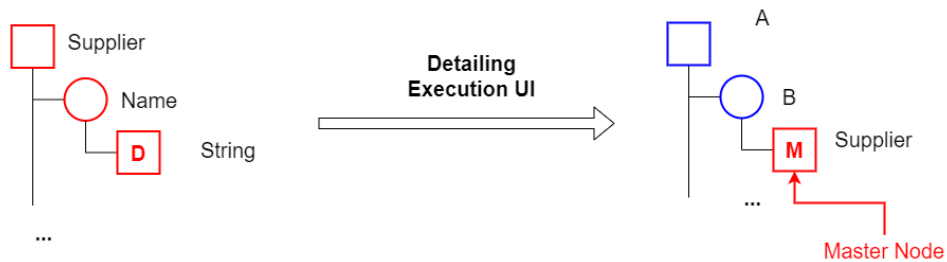


Figure 7.3: Example of GS for the Detail Search Category

**D. Execute**

This category will contain events that have GS that will generate Search UI. These GS must have an Entity Root Node with the same entity as the UI context. This Menu category will contain GS like those seen in Figure 7.4 when the context Entity is the Supplier Entity.

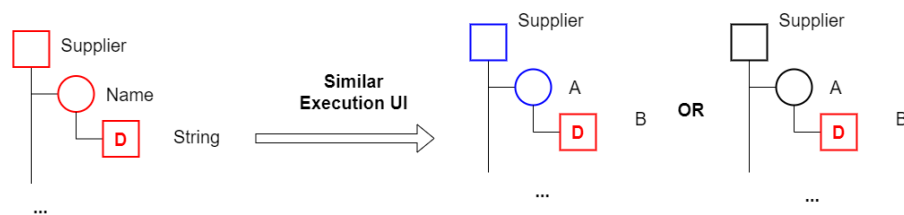


Figure 7.4: Example of GS for the Execute Search Category

**E. Lazy Mode**

This category will contain events that will manipulate the layout and information of the Generic Diagram UI, which will be presented in Section 7.3.



Figure 7.5: Navigation Menu Example from a UI with a Supplier Entity for context

In Figure 7.5, the Search UI presents a list of Suppliers. On the left can be seen the Navigation Menu generated and identified each category and the events available for each category.

## 7.2.2 Navigation between UI

With the events available in this menu, users can browse through the information in SMS DB and have multiple UI opened simultaneously. While using a UI, users can access one or more related UI through the Navigation Menu, which will adapt to the context when generated. Users can execute events that might be required at any moment while executing an event or analysing information from a UI.

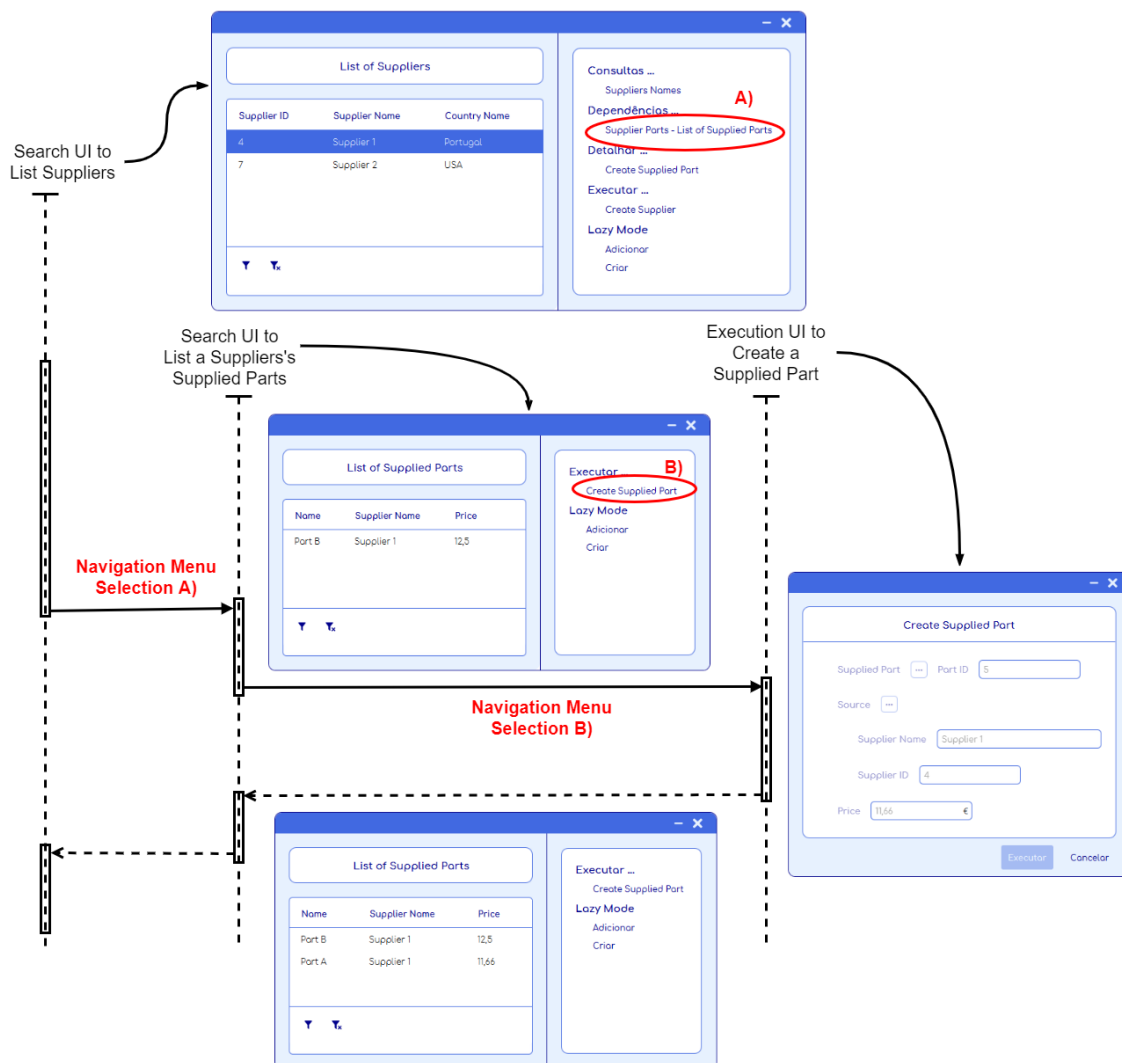


Figure 7.6: Example of Navigation between different UI

In Figure 7.6 contains a Sequence diagram that exemplifies the navigation between different UI. This navigation simulation starts with a Search UI where a list of the Supplier instances was obtained from the system DB. After selecting a Supplier instance and the desired UI from the Navigation Menu, was generated of the Search UI responsible for obtaining the list of Supplied

Part instances. Since the UI that listed Supplies had selected the instance of Name "Supplier 1", this instance was part of the UI context. It resulted in a UI that listed Supplied Parts, where only instances where the Source was the context's Supplier instance were obtained.

Next, in the diagram in Figure 7.6, another UI was generated from the Supplied Parts List UI's Navigation Menu. This time an Execution UI to create a Supplied Part instance. Since the context Supplier instance was already known, the Execution UI corresponding components adapted to the context when generated. When finished creating this new Supplied Part instance, it can be seen at the bottom of Figure 7.6 that the Supplied Parts list was updated, and a new instance appears in that list.

### 7.2.3 Navigation in different Types of UI

Since the different types of UI have different components and how the context information is gathered. The Navigation Menu will be generated for different occasions and use information from different UI components. The following paragraphs describe and exemplify the navigation for each type of UI.

#### Execution UI

For Execution UIs, when generated, the Navigation Menu is not presented since the context does not have any instance. However, after the execution of the UI event, the instance obtained will be used to provide Users with a Navigation Menu. This use case can be seen in Figure 7.7 where a Supplier was created, and in the image can be seen the resulting Execution UI's Navigation Menu.

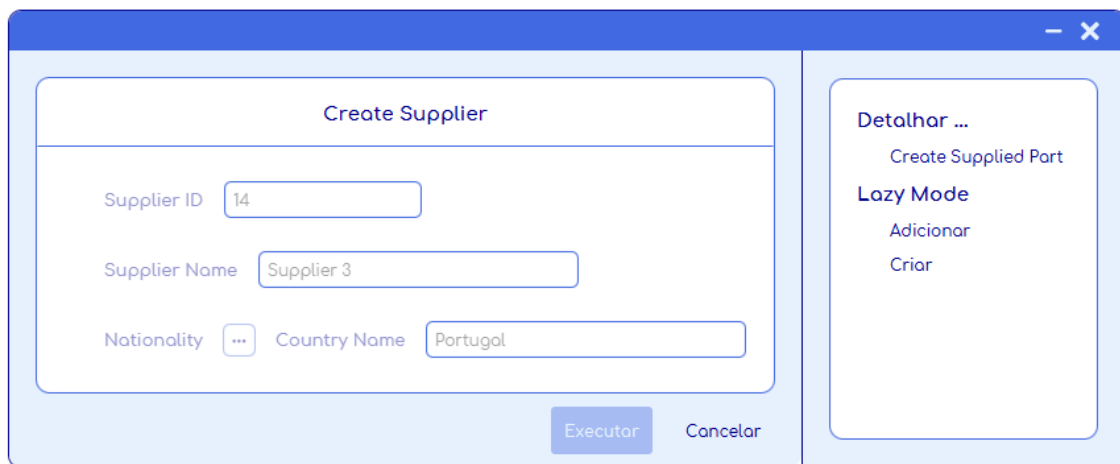


Figure 7.7: Navigation Menu example in an Execution UI

#### Search UI

The navigation Menu of Search UIs can be seen in the examples presented in Figure 7.8. For this type of UI, the navigation will also be shown after the event was executed, since only then will possibly be obtained instances' information from the system DB. In this particular case, the context is constantly changing since the context instance of that entity is the one related to the

selected row in the Search UI Table View. Figure 7.8 contains the Search UI after the search was executed, containing a List of Suppliers after the Navigation Menu has been generated.



Figure 7.8: Navigation Menu example in a Search UI

### Selection UI

Since this type of interface already has an instance list when it is generated, the Navigation Menu is generated simultaneously. An example of the Navigation Menu for a UI that allows Country Instances to be selected can be seen in Figure 7.9.

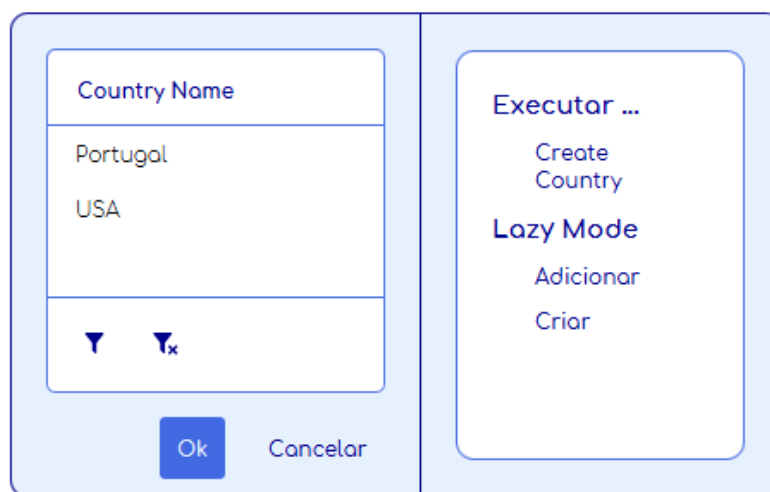


Figure 7.9: Navigation Menu example in a Selection UI

## 7.3 Generic Diagram UI

One of the requirements mentioned for the SMS is the ability for UI to display the several entities instances and relationships between them that might exist in the SMS DB. This visualisation can be obtained with the UIs generated from GS, but with the amount of information that the SMS contains, this might result in an explosion of several UIs. A different approach to display this amount of information had to be developed to avoid this.

To this end, a special kind of UI was designed to display several instances of different entities and their relationships. This UI was named Generic Diagram UI. This UI contains cells representing entity instances or events available in the SMS, used to obtain entity instances. These Diagram Cells are arranged in an invisible grid and are connected between themselves. The UI Generator contains a Generic Diagram UI entitled Lazy Mode.

### 7.3.1 Diagram Cells

The information about the entity's instance must be gathered to create the corresponding UI Diagram cell. The cells contain a Label with the entity's instance's most relevant details and the entity icon. These cells will allow more experienced users to assert their entity immediately.

Attributes	Description	Set of Values
Icon (Entity Nodes)	Icon code	String
Icon Color (Entity Nodes)	Icon RGB Color code (ex: #FFFFFF - white)	String
Tag (Entity and Data Nodes)	Used to decide if this Node information is relevant	Boolean

Table 7.2: GS Model Attributes

In reality, the GSs that contain the information on the Conceptual Model of the SMS have more attributes not mentioned in Section 4.3 because that information was irrelevant when the GSs were introduced. These attributes are presented in Table 7.2.

The Icons were created using Icon libraries, where each library has a Specific set of icon codes. The Tag attribute, when assigned to an entity node, adds the entity Name to the information Label. For data nodes, the value of that instance's corresponding attribute value is added to the information Label. Figure 7.10 shows an example of a Supplier cell and the GS used to save the Supplier Entity.

Users can either delete the cell or open a Navigation Menu when hovering Diagram cells, using that cell's context. This hovering feature can also be seen in Figure 7.10, as well as the Navigation Menu generated for a Supplier instance Diagram Cell.

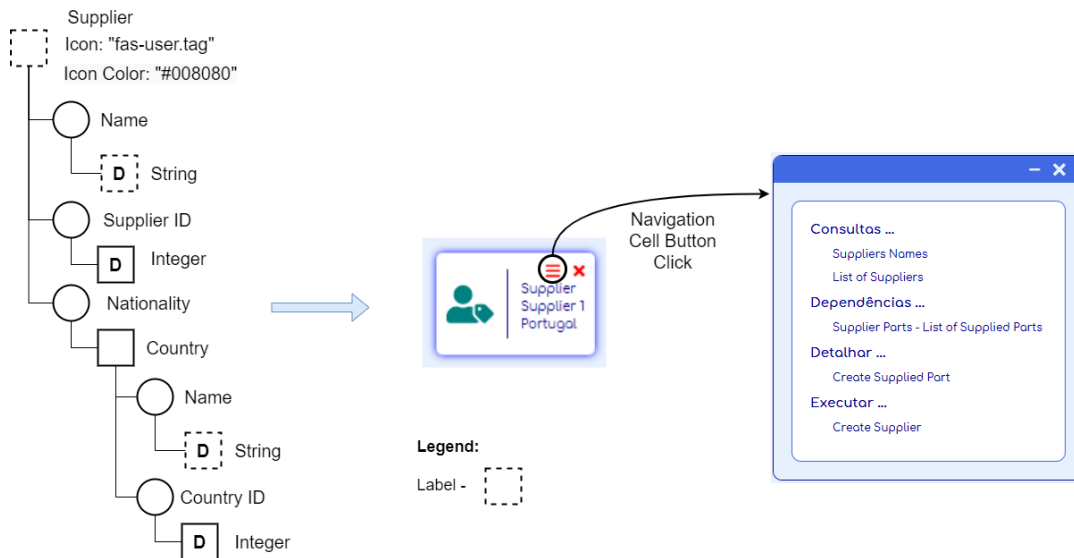


Figure 7.10: Generic Diagram Cell Example

### 7.3.2 Visualise Information

The events in the Navigation Menu of the Diagram Cell will have a different outcome from the Navigation Menus detailed above in Section 7.2. However, this difference will only be applied to events with a GS that would generate a Search UI. Since the context is given by the Diagram cell, with this context information and the event GS, a list of instances from the SMS DB is obtained and transformed into Diagram Cells.

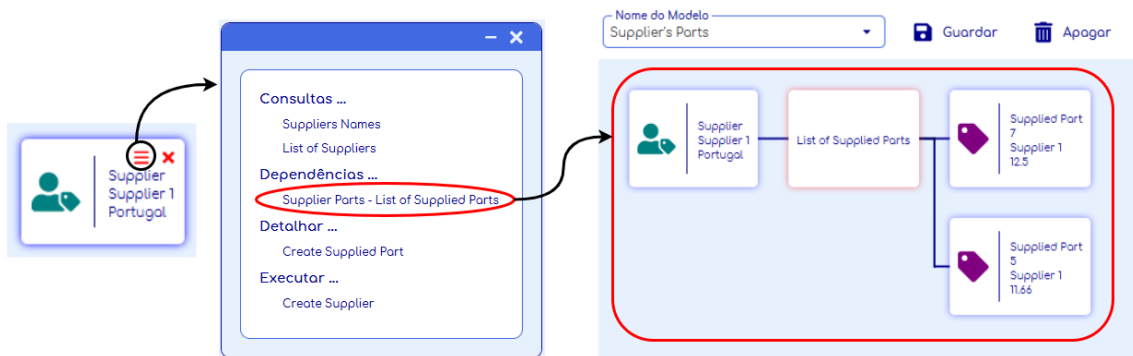


Figure 7.11: Example of a Search Event Execution in the Generic Diagram UI

In these scenarios, a dummy cell representing the search event executed will also be added to the diagram, containing only the Event Title and connecting the instances obtained from the Search event and the one used to give context to this search. A specific Scenario is presented in Figure 7.11, where the List of Supplied Parts is executed using the context provided by the Supplier instance in the Diagram Cell.

The Entity instances can also be added to the Generic Diagram from category D of the Navigation Menu defined in Section 8.3.1, where two options are always available to users, to create



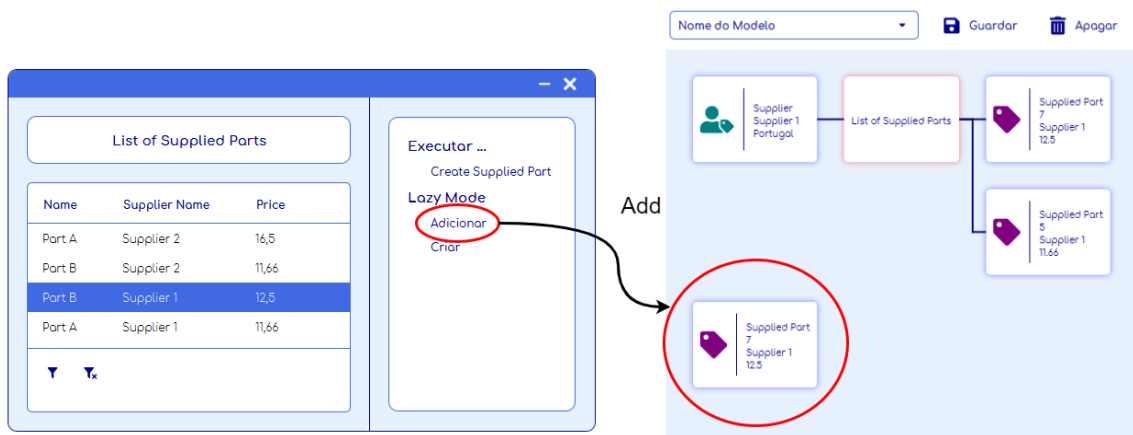


Figure 7.12: Adding instance cell to the Generic Diagram UI

or to add that instance to the UI Generic Diagram. The first eliminates all the Diagrams and adds a cell containing the context's instance. The latter only adds the cell, maintaining the Rest of the Diagram cells. Figure 7.12 portrays the addition of a Supplied Part instance to the Diagram UI.

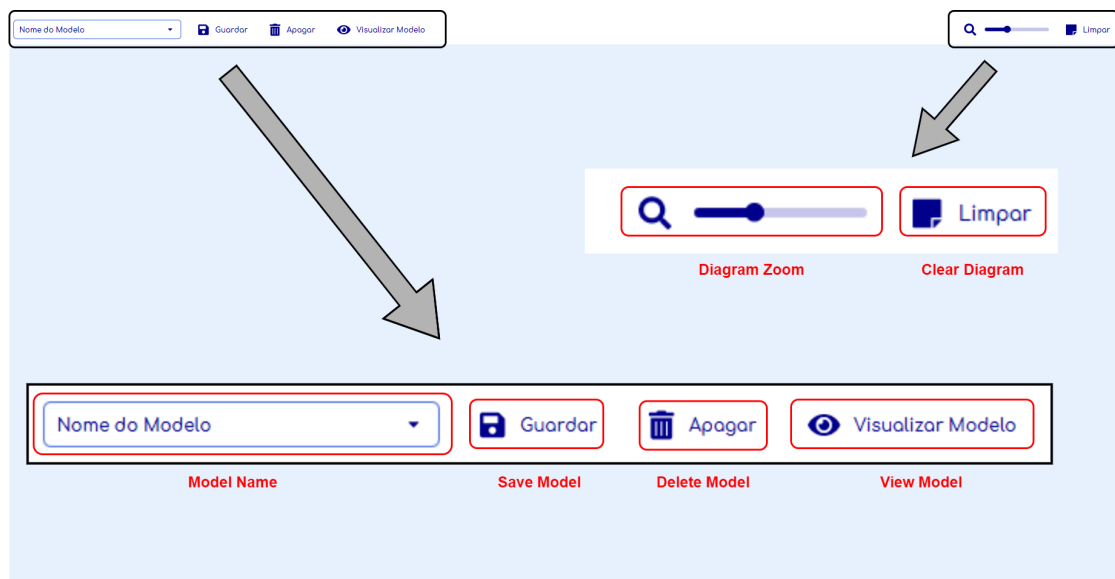


Figure 7.13: Generic Diagram UI options

The Generic Diagram also has some options that allow users to save, delete and view the entities and events model constructed in the UI. The Generic Diagram content can be resized by applying the desired zoom through a horizontal slider. The diagram's content can also be cleared, deleting all the cells in the diagram. These options are presented and highlighted in Figure 7.13.

### 7.3.3 Diagram Model

The sequence of Entities and Events used to populate the Generic Diagram UI can be saved in a Diagram Model with the desired title. The Models saved can later be visualised without instances

information, and the Model sequence can be edited at any time if desired.

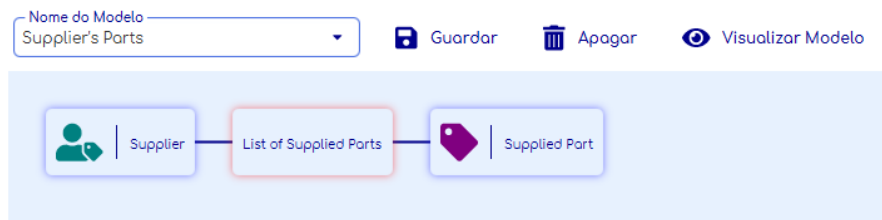


Figure 7.14: Example of the visualisation of a Diagram Model

Figure 7.14 represents an example of the visualisation of a Diagram Model. This Diagram Model was obtained from saving the sequence developed previously in Figure 7.11, where a Supplier Instance was added, and with that instance were obtained its Supplied Parts.

This model only contains a single Supplied Entity Cell as it is only here to provide information on the entity obtained from the event. The model will generate as many cells as instances returned from that event. Similarly, if a second event was applied on top of the Supplied Part Entity, this second event will be applied to every single Supplied Entity obtained from the first event.

The Diagrams Models will later be available in navigation menus where the context entity is the same as the entity that initiates the Diagram Model sequence. In the Diagram Model used to exemplify the Diagram Model Visualisation in Figure 7.14, the starting entity is a Supplier. Therefore when a navigation menu is generated from a context where the entity is a Supplier, the Lazy Mode category of the Navigation Menu will contain the Diagram Model created, as seen in Figure 7.15.

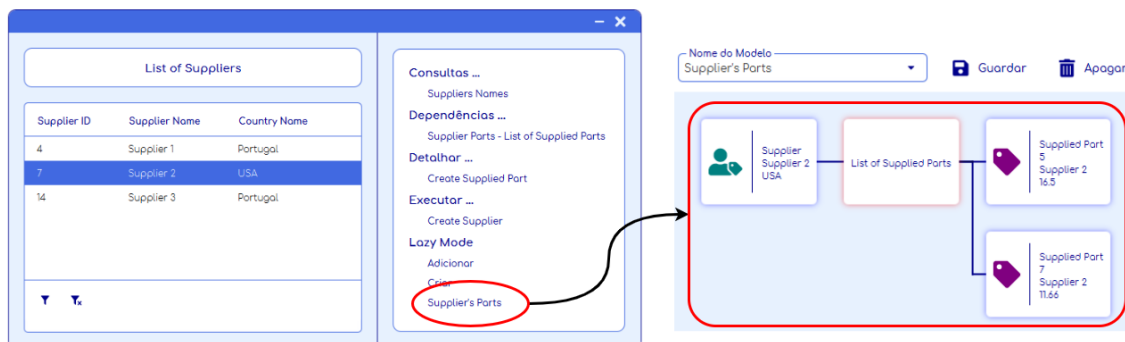


Figure 7.15: Diagram Cells generated from a Diagram Model

When the Diagram Models are used from Navigation Menus, the model is used with the context instance, and all the Generic Diagram UI cells are automatically generated. This scenario is presented in Figure 7.15, where a different Supplier instance is used with the Diagram Model saved before.

## Chapter 8

# UI Designer and UI generator

### 8.1 Introduction

Previous Chapters explained how the UI Designer is used to create GS for UI and how the UI Generator uses these GS to generate UI. However, these desktop applications have other features that were not mentioned before.

This chapter will present the UI Designer features that allow to analyse the GS created and how the UI Generator provides information on the UI available in SMS. It will finally be discussed how Users can adapt the UI generator and the Interfaces Generated to their preferences.

### 8.2 UI Designer

The UI Designer Application was designed with features other than just to create the GSs used to generate UIs. This application also allows users to view a list of the GS created and their detail.

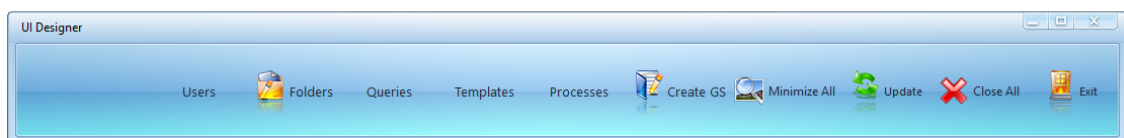


Figure 8.1: UI Designer application Options Bar

The Application features can be accessed through the Options Bar in Figure 8.1. These options allow the following actions:

- **Users** - Access the interface that manages the UI Generator end-users.
- **Folders** - Create or edit folders to organise the GSs.

- **Templates** - Access the list of Execution UI GSs that have an Insert root node.
- **Processes** - Access the List of Execution UI GSs with a Select or Delete root Node.
- **Queries** - Access the List of Search UI GSs.
- **Create GS** - Opens the Interface designer to create or alter a GS.
- **Others** - Exit the Application. Close, minimise or update the Interfaces opened.

One feature of the UI Designer Application is the ability to group the GSs created in folders and sub-folders. The entities of the SMS conceptual Model can be organised in folders whose names can be defined by the user. This folder management is also based on GSs where its nodes are either folders or entities. Figure 8.2 shows an example of a folder structure where the stripped red squares referenced an Entity used to create GSs. This folder management will allow organising how the UIs available in the system will be presented in the UI generator.

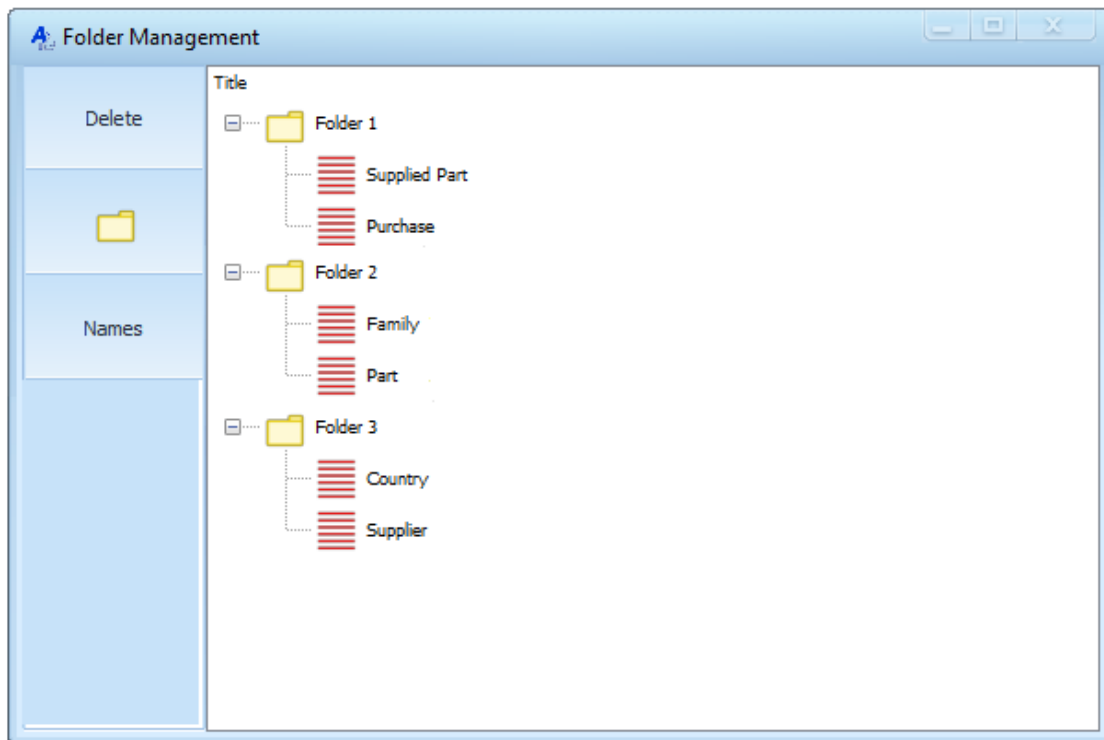


Figure 8.2: Folder Management Tool

To edit a UI GS, the GS root node must be dragged to the Create GS button. Similarly, Folders content can be edited by dragging a folder to the Folder button in the UI Designer Options Bar

Figure 8.3 shows the list of Search UIs developed to implement the Simplified SMS Conceptual Model and exemplify the generation of UI. The GSs are arranged in the folders and sub-folders, where the systems Entities were organised. This organisation is exemplified in the Search UI list where two Search GS, used to obtain instances of the Supplier Entity, are contained in the same folder, "Suppliers List" and "Names of Suppliers".

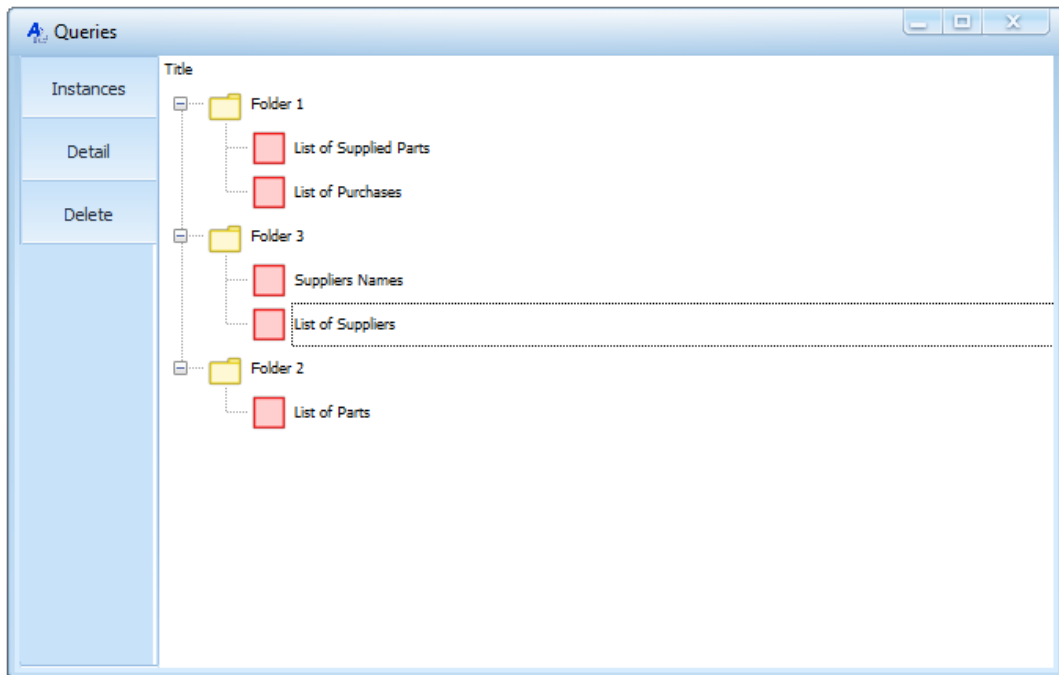


Figure 8.3: List of Search UI GSs

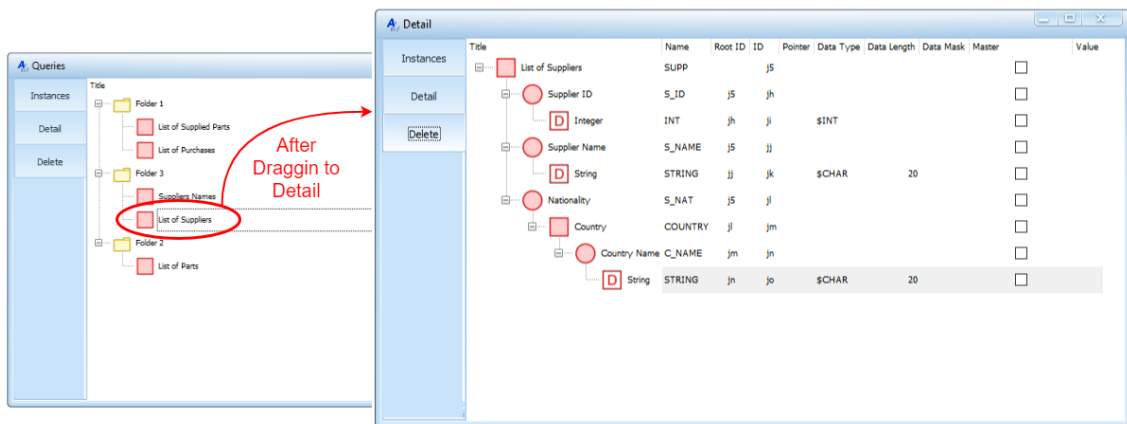


Figure 8.4: Detail of a Search UI GS

The GSs presented in these lists are represented by their root node, which in the case of Search UI is always a Select node. The options on the left of these lists allow users to delete GSs or see their composition in detail. These operations are executed by dragging a GS root node to the desired option. Figure 8.4 shows an example of this, where the UI GS that lists Supplier instances is presented in detail.

This application also allows management of which users can access the UI Generator and the UIs available in the system. Figure 8.5 shows the interface that enables this management.

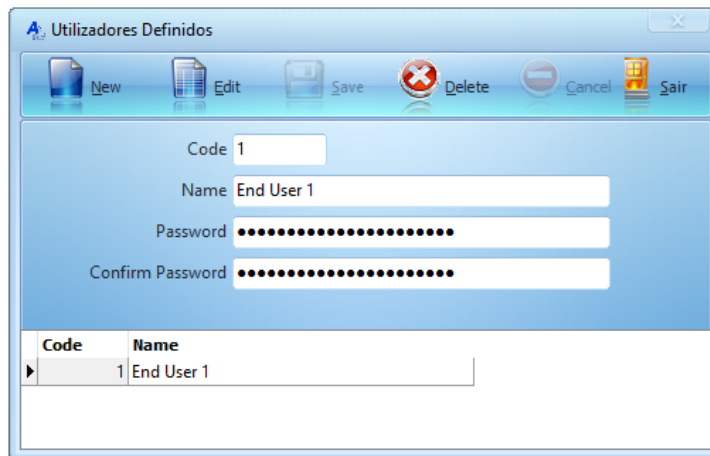


Figure 8.5: User Management Interface

### 8.3 UI Generator

The previous Chapters explained how GSs are used to generate UIs and how Users can navigate between them. However, it was not yet discussed how the UI available in the system are initially presented to Users. This area will be covered in the following Sections, as well as what features allow Users to customise the Application theme.

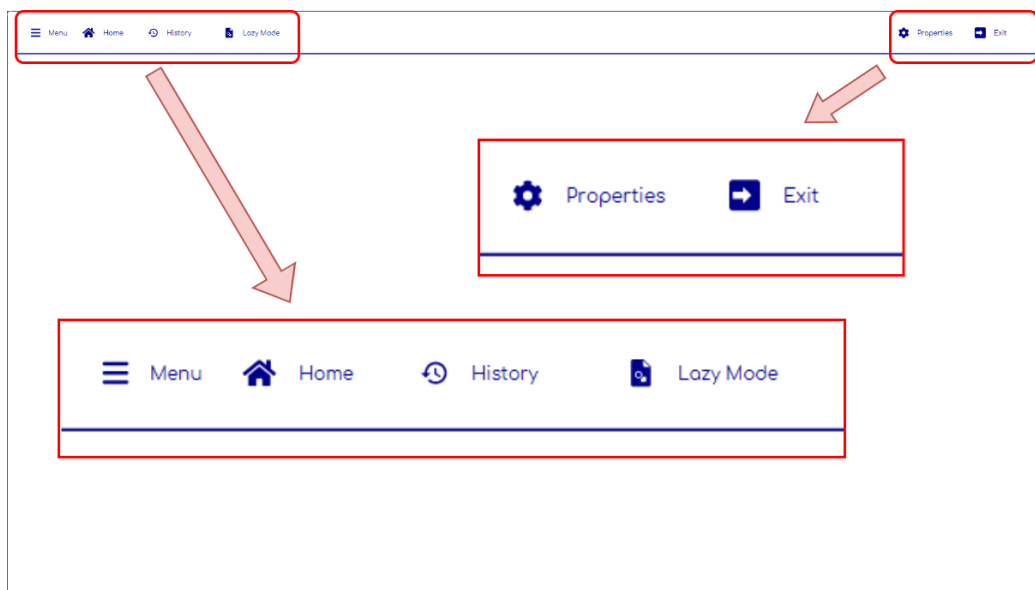


Figure 8.6: UI Generator Top bar

Figure 8.6 presents the UI Designer application with buttons that allow users to access Menus and specific features. The options provided by these buttons are the following:

- **menu** - Opens the side Menu, containing a list of all the UI available.
- **Home** - Changes the application background to the folder Navigation Menu.

- **History** - Changes the application background to the Events History.
- **Lazy Mode** - Changes the application background to the Generic Diagram UI.
- **Properties** - Opens the User Properties Interface.
- **Exit** - Closes the UI Generator application.

### 8.3.1 Menus and generation of interfaces

The folder management in the UI Designer allows the organisation of the system's entities to be managed. This folder organisation is used to generate Menus to make the UI of the SMS available to End Users, as displayed in Figure 8.7.

Since this Folder organisation contains entities of the SMS DB, GS that have root nodes with the same entity are automatically arranged in a specific Folder for that entity. This situation does not apply if only exists a single GS for that entity. The UI Designer provides two different Menus to access the UI available in the system. One is more traditional, found in most Applications, and the other is more intuitive and easy to use.

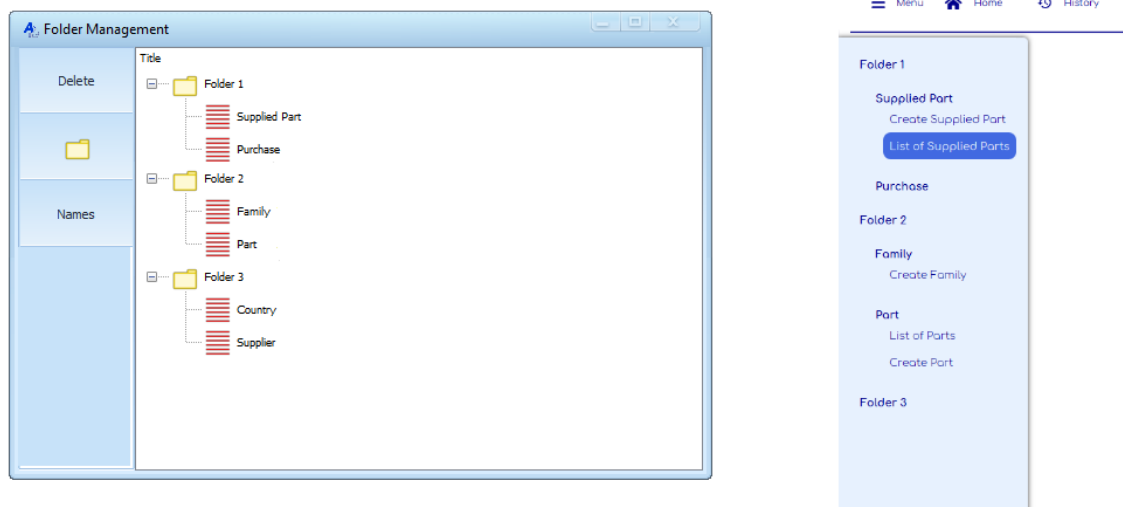


Figure 8.7: UI Generator Side Navigation Menu

The more traditional menu consists of a sidebar navigation menu that can be accessed and hidden on the left side of the screen as the user wishes. This menu contains all the events available in the system. Figure 8.7 contains the folder organisation of the system's entities and the sidebar menu generated from these folders. Only the root folders appear initially, and their content is expanded or hidden when the folder title is clicked. The available UI displayed has a smaller font size, and its background colour changes when hovered.

In Figure 8.7, the Folder 3 content is hidden, while the other two folders are expanded. A sub-folder for Suppliers exists since there were created 3 UI to manipulate this entity. This approach

can cause the size of this side Menu to be lengthy and hard to use, so a different approach was also taken.

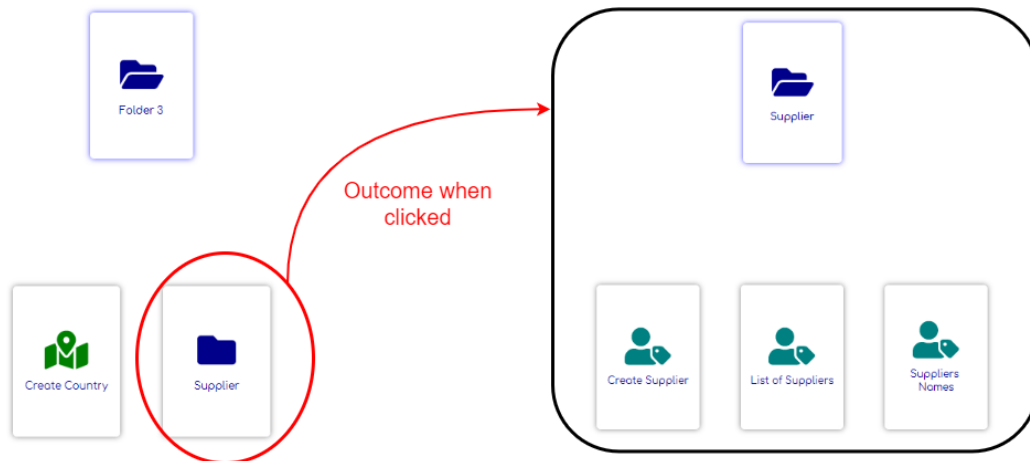


Figure 8.8: UI Folder Navigation Menu

The other menu developed allows navigating folders content one folder at a time. Initially are displayed all root folders, and the layout of this menu changes when a folder is clicked. The UI options of the menu use the Entity icon so Users can easily find the desired UI. An example of this menu navigation and its interaction is presented in Figure 8.8. The content from Folder 1 is displayed, and when the Supplier Folder is clicked, the Menu information changes to the UI that manipulate the Supplier Instance.

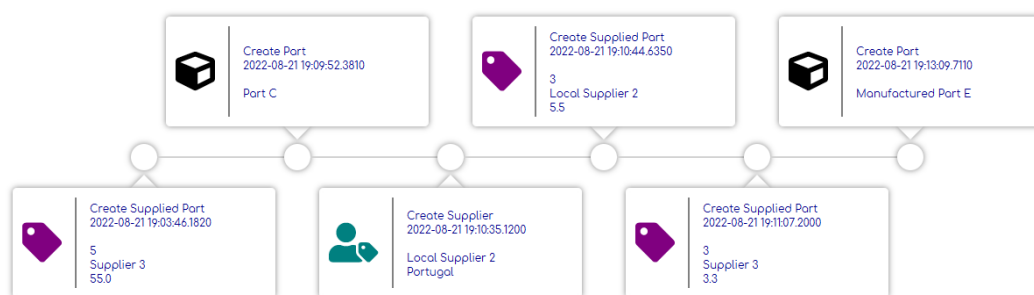


Figure 8.9: UI Event History

Finally, a feature was developed that allows users to visualise the more recent changes made to the SMS DB. This feature was obtained by storing the timestamp when an execution Interface was executed and providing Users with this event history. This history contained a Timeline with cells created using a similar method to the one used to create Generic Diagram Cells, presented in Section 7.3.1. In addition, the timestamp and the UI name were added to the cells Label. Figure



8.9 presents an example of this event history timeline, where its cells, when clicked, also provide Users with a navigation Menu using the cell instance information.

### 8.3.2 User Theme Adaptability

In addition to the adaptive aspects of the generated UI mentioned in previous chapters, the UI generator ensures that the application and the generated UIs can adapt to their end users. This adaptation is based on a user profile contained in the UI DB with specific characteristics imposed for each user and others that users can adjust.

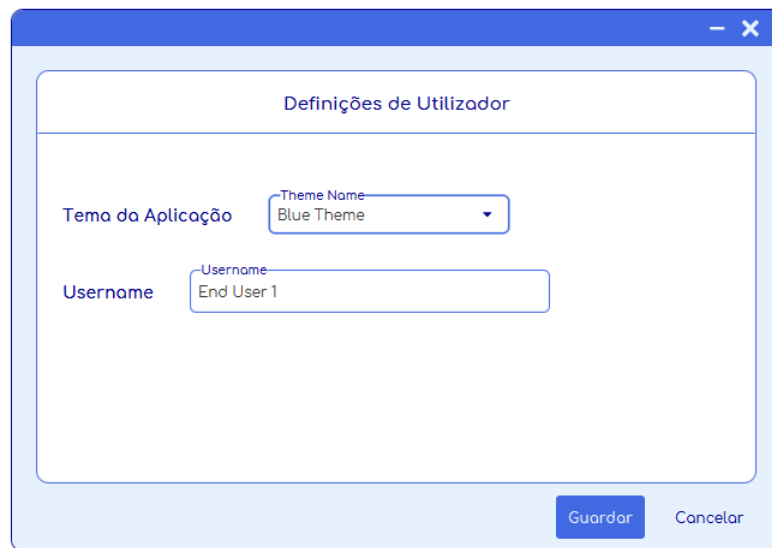


Figure 8.10: User Preferences Interface

End users do not have the power to manipulate what SMS events he has access to. Thus a set of events is assigned to a user according to the role he plays in the organisation. Moreover, the user can change the Theme of the application to adapt the application to his/her preferences and other profile characteristics. Figure 8.10 contains the interface that allows users to manipulate the application's Theme and change its Username.

The application's Theme chosen by the user is applied to every aspect of the application, consisting of a set of colours used to style the components used throughout all the UI generated. In 8.11 can be seen some of the different themes available to the user.

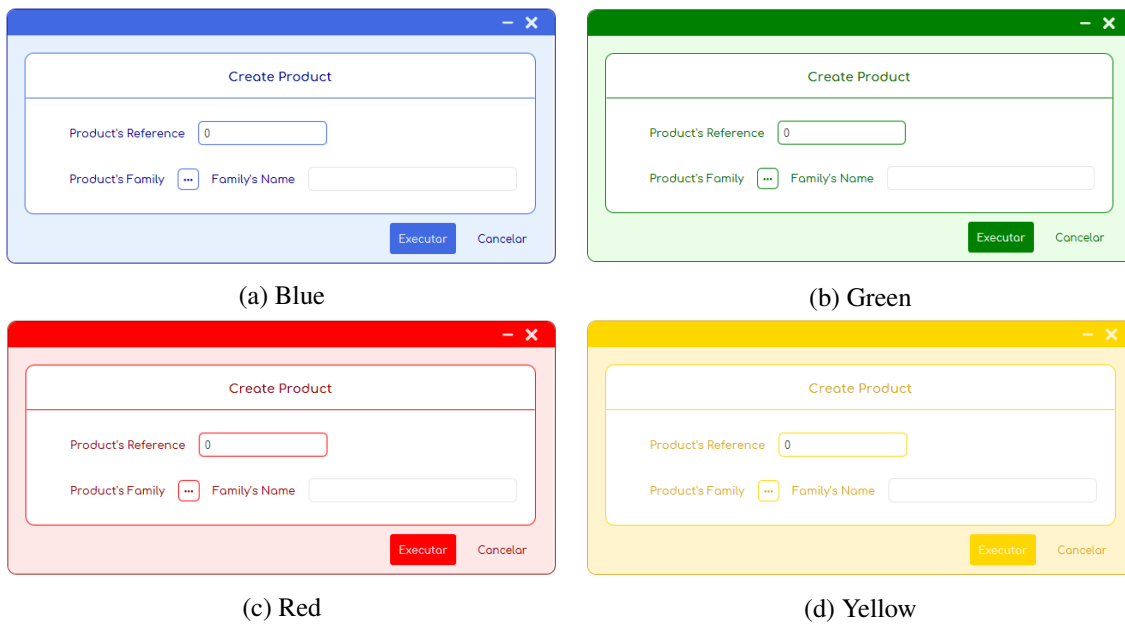


Figure 8.11: Some of the different Themes available applied to a generated UI

## Chapter 9

# Conclusions and Future Work

### Contents

---

<b>9.1</b>	<b>Generic Structures' Reach and Advantages</b>	<b>72</b>
<b>9.2</b>	<b>Contributions</b>	<b>73</b>
<b>9.3</b>	<b>Future Work</b>	<b>73</b>
<b>9.4</b>	<b>Final Remarks</b>	<b>74</b>

---

The two main goals that were proposed for the development of this dissertation were the following :

- Develop a fast and straightforward application to design User Interfaces using Generic Structures, to meet the requirements of SMS.
- Develop an approach to generate UI, using the information contained in Generic Structures, through which Users can populate and access the information in the SMS DB.

The first goal was accomplished using GSs with specific rules and characteristics to store the information of the desired UI. A desktop application was developed to meet that end. This application allowed other project members to create and manipulate these GSs and establish connections between them and the entities available in the SMS.

The second goal of this dissertation was to develop an application that would use the information from the GSs to generate UI that users could interact with. This UI Generator was developed with the following features:

- The generated UIs are based on different Templates according to the type event obtained from the GS.
- The generated UIs contain a context-sensitive Navigation Menu that allows users to navigate between different User Interfaces.
- A Generic Diagram UI that allows users to visualise several different information contained in the system and the relationships between this information.

- Different ways of accessing the UI in the SMS are organised in folders.
- Provide an Event History with the recently executed UIs.
- Allow Users to customise the Theme of the Application.

The architecture developed to achieve this dissertation's two main goals pivoted around the project's main functional area presented in section 3.3.3, **FA6**. This architecture allows us to quickly design the User Interfaces desired for the SMS and adjust them later to follow the manufacturing needs. Organising the UIs in folders makes menus adjustable and decides which interfaces will be available.

The functionalities achieved with this architecture also help meet the SMS requirements presented in section 3.3.1, mainly **REQ5** and **REQ7**. The ability to create or alter the system's UIs fast and easily makes the system flexible since its UIs can follow the system's requirements changes, manufacturing needs and organisation. The UIs developed and generated with this architecture make the system Transparent since the information presented to users is context-sensitive, namely the navigation menus. With context-sensitive UIs, they adapt to the information they are displaying and also to the user needs as they are generated. Additionally, using the Generic Diagram UI and the models saved, users can change what information is displayed in the diagram according to their needs.

This dissertation work is part of an ongoing project that aims to develop an SMS. The applications developed for this project were presented in this dissertation and to other teams involved in the project. These teams' members will use and test the applications, allowing them to design UIs for events of the SMS that require Human-Machine Interaction.

## 9.1 Generic Structures' Reach and Advantages

The relevant information that allows for the generation of UI is stored in the Generic Structures explained in Chapter 5. These GSs are extensive since they can represent most of the possible manipulations a user can bring about in a system's DB. This extensiveness means that UIs whose complexity can vary drastically can be designed. From a simple GS with only three nodes to the opposite level of complexity, having one with dozens of nodes. These more complex GSs can generate very extensive UI, where users can select instances from several entities and input varied types of data.

Due to the extensiveness of the GSs used in the Architecture developed to design and generate UIs, it can be applied not only to the SMS being developed in the project but to other applications that might require human-machine interaction. This extensiveness poses a significant advantage to developing UI individually for each purpose using traditional methods.

This dissertation focused only on the generation of UI based on the information in GS. However, using these GS is possible to store in the same DB Conceptual Model the information about the Conceptual Model of the System itself, the system instances information, and the meaning of its entities, relationships and events that must be implemented. A future challenge would be to

use this unified conceptual model to autonomously generate processes that would use the system events, as long as this implementation posed the same advantages - simplicity and time-efficiency- that were obtained with the architecture developed to design and generate UI.

Another surplus that this Architecture provides is the generation of context-sensitive Navigation Menus, which were possible since the system's conceptual model is contained in the GSs. Generating these Menus in every UI assists end users in navigating the system information and executing the desired events. These navigation menus and the UI Generic Diagram help meet Transparency's SMS requirement. The navigation menus are context-sensitive and will adapt to the information the user is analysing. The Generic Diagram UI will adapt to the information end users intend to visualise on different occasions, using the diagram models saved by them.

## 9.2 Contributions

As explained before, the dissertation work was developed to provide UIs to be used in the SMS being developed for the Continental Antenna Factory in Vila Real as part of the "Continental Factory of the Future" project. Other project members responsible for developing the conceptual model of the SMS and its events can simultaneously easily design the required UI for those events using the UI Designer developed.

The UI examples presented in this dissertation display a simple scenario to showcase the capabilities of the architecture developed to design and generate UIs. From the set of abilities, the generated UIs convey, it can be asserted that this architecture, when applied to the SMS events, will be able to generate the necessary User Interfaces to manipulate the system and its components.

The understanding of this UI design can be used to create the GSs that will generate the UIs required for the SMS being developed for the Continental project. Other project members are refining the UIs that are being designed as they define the intricacies of the SMS and its events. This continuous work is possible since these members have also assisted in the definition of the UI types and elements required and have helped test the Interfaces, their design and generation.

Since this dissertation is part of an ongoing project the UIs development will continue far long after this dissertation end. Therefore, since the presented Interfaces will suffer changes and adjustments in the future, it was not developed a dedicated User Experience validation of the UIs. However other project members used and tested the developed applications and gave constant feedback that allowed the make the UIs appealing and easy to use.

## 9.3 Future Work

Since this project duration is lengthier than the time available for this dissertation, new features were and will be added. Also, different approaches for showing information will be combined with the work developed in this dissertation. The following paragraphs contain some of these features.

### **Shop Floor Monitoring**

An example of functionality that might be adapted from a UI developed in this dissertation is the adaptation of the Generic Diagram UI to monitor the Shop Floor. This monitoring would be achieved by having the grid background displaying a plant of the Shop Floor or part of it. In this scenario, the diagram cells would represent entities of physical components of the factory having information about its location that would be used to place them in the corresponding grid position. This would allow Users to view several aspects of the factory, from every station on the Shop Floor to every machine and sensor in a specific station. If the intention were to plan the production, like an AVG route, for example, the user would be able to establish relations between the grid cells available.

### **Ghant Diagrams and Pivot Tables**

The generated UIs that display the system's instances use a Table View that organises the system information in columns and rows. The information of the instances can instead be displayed in a pivot grid, allowing users to choose the columns, rows and value fields of the pivot grid. The same applies to Gantt diagrams, but only for instances with start and end date attributes.

### **Access Files**

Other features that will be implemented are the ability to visualise and access images, PDFs and other files and also establish a connection between these files and the system's entities. Integration with other technologies, such as CAD software, must be implemented and a generic way to communicate with the system's devices.

To ultimately centre users in the scope of the UI, more methods for user adaptation will be added to the generated User Interfaces. Some examples of these methods are allowing users to choose specific styling options for each UI available and adopting machine learning algorithms to adapt the UIs based on previous events, as seen in the literature found. However, some aspects of the Interfaces will always be set by the organisation and are not customisable.

## **9.4 Final Remarks**

The development of this dissertation and all the work surrounding the Continental project resulted in the acquisition of new information regarding SMS and the emerging concepts of Industry 4.0, which will undoubtedly shape the future of manufacturing and technology.

Working on a project as broad as this allowed to work with several people from other engineering backgrounds, which also provided new knowledge and insight into how manufacturing systems operate.

# References

- [1] N. A. N. Lee, L. E. G. Moctezuma, and J. L. M. Lastra. Visualization of information in a service-oriented production control system. pages 4422–4428. doi:[10.1109/IECON.2013.6699847](https://doi.org/10.1109/IECON.2013.6699847).
- [2] L. Zhang, Q. X. Qu, W. Y. Chao, and V. G. Duffy. Object-oriented user interface customization: Reduce complexity and improve usability and adaptation. volume 10286 LNCS, pages 404–417. Springer Verlag, 2017. doi:[10.1007/978-3-319-58463-8\\_34](https://doi.org/10.1007/978-3-319-58463-8_34).
- [3] E. MacHado, D. Singh, F. Cruciani, L. Chen, S. Hanke, F. Salvago, J. Kropf, and A. Holzinger. A conceptual framework for adaptive user interfaces for older adults. pages 782–787. Institute of Electrical and Electronics Engineers Inc. Cited By :6 Export Date: 30 December 2021. doi:[10.1109/PERCOMW.2018.8480407](https://doi.org/10.1109/PERCOMW.2018.8480407).
- [4] Saurabh Vaidya, Prashant Ambad, and Santosh Bhosle. Industry 4.0 - a glimpse, 2018. doi:[10.1016/j.promfg.2018.02.034](https://doi.org/10.1016/j.promfg.2018.02.034).
- [5] Mario Hermann, Tobias Pentek, and Boris Otto. Design principles for industrie 4.0 scenarios, 2016. doi:[10.1109/hicss.2016.488](https://doi.org/10.1109/hicss.2016.488).
- [6] Hyoung Seok Kang, Ju Yeon Lee, SangSu Choi, Hyun Kim, Jun Hee Park, Ji Yeon Son, Bo Hyun Kim, and Sang Do Noh. Smart manufacturing: Past research, present findings, and future directions. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 3(1):111–128, 2016. doi:[10.1007/s40684-016-0015-5](https://doi.org/10.1007/s40684-016-0015-5).
- [7] M. Bakaev and M. Gaedke. Application of evolutionary algorithms in interaction design: From requirements and ontology to optimized web interface. pages 129–134. Institute of Electrical and Electronics Engineers Inc. doi:[10.1109/EIConRusNW.2016.7448138](https://doi.org/10.1109/EIConRusNW.2016.7448138).
- [8] Maxim Bakaev and Tatiana Avdeenko. *Defining and Optimizing User Interfaces Information Complexity for AI Methods Application in HCI*, book section Chapter 37, pages 397–405. Lecture Notes in Computer Science. 2015. doi:[10.1007/978-3-319-20916-6\\_37](https://doi.org/10.1007/978-3-319-20916-6_37).
- [9] G. G. Yen and D. Acay. Adaptive user interfaces in complex supervisory tasks. *ISA Transactions*, 48(2):196–205, 2009. doi:[10.1016/j.isatra.2008.11.002](https://doi.org/10.1016/j.isatra.2008.11.002).
- [10] Jay Lee, Behrad Bagheri, and Hung-An Kao. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3:18–23, 2015. doi:[10.1016/j.mfglet.2014.12.001](https://doi.org/10.1016/j.mfglet.2014.12.001).
- [11] Gitte Lindgaard, Gary Fernandes, Cathy Dudek, and J. Brown. Attention web designers: You have 50 milliseconds to make a good first impression! *Behaviour Information Technology*, 25(2):115–126, 2006. doi:[10.1080/01449290500330448](https://doi.org/10.1080/01449290500330448).

- [12] Francisco Lobo. The industry 4.0 revolution and the future of manufacturing execution systems (mes). *Journal of Innovation Management*, 2015.
- [13] Bitkom, VDMA, and ZVEI. Implementation strategy industrie 4.0: Report on the results of the industrie 4.0 platform. Report, 2016.
- [14] M. Peruzzini and M. Pellicciari. A framework to design a human-centred adaptive manufacturing system for aging workers. *Advanced Engineering Informatics*, 33:330–349, 2017. doi:10.1016/j.aei.2017.02.003.
- [15] M. Bakaev and T. Avdeenko. Ontology-based approach for engineering web interface design resolutions. Science and Engineering Institute.
- [16] S. Jack Hu. Evolving paradigms of manufacturing: From mass production to mass customization and personalization. *Procedia CIRP*, 7:3–8, 2013. doi:10.1016/j.procir.2013.05.002.
- [17] Wollny S. Botterweck G. Pleuss, A. Model-driven development and evolution of customized user interfaces. In *5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2013*, pages 13–22.
- [18] L. Wu, P. Yu, J. Wei, Y. Xie, and H. Kishida. A method for user-centered interface customization and development of a prototype system. volume 1, pages 197–200. doi:10.1109/WCSE.2010.46.
- [19] V. Villani, L. Sabattini, J. N. Czerniaki, A. Mertens, B. Vogel-Heuser, and C. Fantuzzi. Towards modern inclusive factories: A methodology for the development of smart adaptive human-machine interfaces. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 12-15 Sept. 2017*, 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), page 7 pp. IEEE. doi:10.1109/ETFA.2017.8247634.
- [20] L. Zhang, Q. X. Qu, W. Y. Chao, and V. G. Duffy. Investigating the combination of adaptive uis and adaptable uis for improving usability and user performance of complex uis. *International Journal of Human-Computer Interaction*, 36(1):82–94, 2020. doi:10.1080/10447318.2019.1606975.
- [21] S. L. T. Hui and S. L. See. Enhancing user experience through customisation of ui design. *Procedia Manufacturing*, 3:1932–1937, 2015. doi:10.1016/j.promfg.2015.07.237.
- [22] K. Kim, J. Jacko, and G. Salvendy. Menu design for computers and cell phones: Review and reappraisal. *International Journal of Human-Computer Interaction*, 27(4):383–404, 2011. doi:10.1080/10447318.2011.540493.
- [23] D. Reguera-Bakhache, I. Garitano, R. Uribeetxeberria, C. Cernuda, and U. Zurutuza. Data-driven industrial human-machine interface temporal adaptation for process optimization. volume 2020-September, pages 518–525. Institute of Electrical and Electronics Engineers Inc. doi:10.1109/ETFA46521.2020.9211930.
- [24] Embarcadero. Delphi. URL: <https://www.embarcadero.com/br/products/delphi>.



- [25] JSON. Json. URL: <https://www.json.org/json-en.html>.
- [26] Java. What is java technology and why do i need it? URL: [https://www.java.com/download/help/whatis\\_java.html](https://www.java.com/download/help/whatis_java.html).
- [27] JavaFX. Javafx. URL: <https://openjfx.io/>.