INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Dynamic Execution Engine (DEE) Graphical Editor

JOSÉ PEDRO SANTOS FONSECA COSTA
Outubro de 2020

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Dynamic Execution Engine (DEE) Graphical Editor

José Pedro Santos Fonseca Costa

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Professor Paulo Gandra de Sousa

Porto, outubro 2020

Dedicatória

Aos meus pais e à Rita.

Resumo

Os últimos anos têm dado lugar a um grande desenvolvimento tecnológico, sendo que os sistemas informáticos cada vez mais se tornam essenciais à vida humana. Esta transformação digital requer que as empresas que se encontram no mercado tecnológico desenvolvam capacidades nos seus sistemas para incluir cada vez mais o utilizador final no desenvolvimento de novas funcionalidades.

No setor da manufatura, é essencial que as grandes indústrias possuam sistemas de apoio à produção dos seus produtos. Um dos sistemas cada vez mais essenciais é o MES. O MES fornece dados reais relativos às diferentes linhas de produção de uma indústria dando aos engenheiros de processo uma visão detalhada sobre o atual estado da linha de produção.

Sendo que cada indústria requer um sistema MES e que cada indústria apresente não só processos de produção como produtos completamente distintos, o sistema tem de estar munido de recursos de extensão para que este se possa adaptar à realidade de cada indústria.

A presente dissertação visa a evoluir um recurso de extensão ao sistema MES disponibilizando-o de forma mais simplificada ao utilizador final dando a possibilidade de ser o próprio utilizador a estender o sistema sem que tenha necessidade de conhecimentos na área de IT.

Este projeto irá integrar o sistema MES de uma das empresas mais conceituadas a nível mundial neste tipo de sistemas.

Palavras-chave: *Manufacturing Execution System; Dynamic Execution Engine; Drag and drop; Code Block programming.*

Abstract

The last few years have given way to a great technological development, making computer systems increasingly become essential to human life. This digital transformation requires companies in the technological market to develop capabilities in their systems to increasingly include the end user in the development of new features.

In the manufacturing sector, it is essential that large industries have support systems to produce their products. One of the increasingly essential systems is the MES. The MES provides real data relating to the different production lines in an industry giving to process engineers a detailed view of the current state of the production line.

Since each industry requires an MES system and each industry presents not only production processes but completely different products, the system must be equipped with extension resources so that it can adapt to the reality of each industry.

This dissertation aims to evolve an extension resource to the system, placing this resource in a more simplified way to the end user, giving the possibility to be the user himself to extend the system without having the need of knowledge in the IT area.

This project will integrate the MES system of one of the most respected companies worldwide in this type of systems.

Keywords: Manufacturing Execution System; Dynamic Execution Engine; Drag and drop; Code Block programming.

Agradecimentos

Começo por agradecer a todos os meus colegas e amigos que fizeram parte do meu percurso académico por todo crescimento e aprendizagem ao longo destes anos.

Ao Instituto Superior de Engenharia e em especial ao Departamento de Engenharia Informática pelo todo o conhecimento fornecido durante a minha vida académica.

Ao Professor Paulo Gandra, não só pela orientação e concelhos dados, mas também por embarcar pela segunda vez nesta aventura comigo.

À Critical Manufacturing pela oportunidade de realizar este projeto.

Ao Engenheiro José Pedro Silva e ao Engenheiro Samuel Rodrigues pela orientação técnica no desenvolvimento deste projeto.

Aos meus pais pela educação, apoio e suporte durante toda a minha vida. A pessoa que eu sou hoje é o fruto do trabalho deles.

À Rita, essencial à minha vida desde 2013. Se hoje estou a escrever este documento é graças ao seu apoio e motivação dados ao longo destes meses, mesmo quando eu próprio já tinha deitado a tolha ao chão só tu eras capaz de me fazer voltar à luta e assim o fizeste. Obrigado.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

AHP	*Analytic Hierarchy Process*
DEE	*Dynamic Execution Engine*
MES	*Manufacturing Execution System*
DSL	*Domain specific language*

1 Introdução

1.1 Contexto

Atualmente, na área de tecnologias de informação, existem inúmeros sistemas que suportam todos os tipos de negócio, desde os mais complexos aos mais simples, havendo para cada área uma diversidade de sistemas com diferentes comportamentos, mas com um objetivo semelhante. A dependência de um sistema de software numa área de negócio é cada vez maior e cada vez mais os utilizadores encontram novos requisitos e funcionalidades que gostariam de ver ser suportadas pelo seu sistema. Isto leva a uma constante evolução dos sistemas, sendo necessário um esforço por parte dos fornecedores de software para que os requisitos do seu cliente sejam satisfeitos. Em alguns sistemas já é possível observar uma grande capacidade de configuração, levando a que os utilizadores desse sistema não necessitem de requerer junto dos fornecedores novos requisitos. No entanto, nem sempre é fácil para o utilizador configurar e modelar o sistema para que este seja capaz de responder às suas necessidades, dependendo da complexidade do sistema, por vezes o utilizador poderá ter a necessidade de ter conhecimentos em áreas especificas para conseguir, sozinho, obter o sistema que pretende.

Neste trabalho é pretendido que, com base num sistema já existente, seja construída uma solução para que um utilizador, sem conhecimentos em áreas que não sejam especificas do seu negócio, seja capaz de modelar o seu sistema para responder às suas necessidades.

1.2 Problema

Atualmente, o sistema MES da Critical Manufacturing opera em diferentes continentes e segmentos. O MES ajuda a produzir dispositivos médicos (próteses, equipamentos de radioterapia, aparelhos de raios X), porcelana, bicicletas, chips de memória, etc. Isso só é possível devido aos recursos de extensibilidade do MES.

O MES suporta diferentes níveis de extensibilidade, dependendo da(s) camada(s) que o cliente ou parceiro precisa de estender. E aqui entra o Dynamic Execution Engine (DEE). DEE permite

que o código do cliente execute antes e / ou após cada operação disponibilizada pela API do MES. Embora a maioria das operações seja a mesma em cada fábrica, na verdade existem muitos ajustes que cada cliente requer que seja implementado para acomodar os seus requisitos e necessidades específicas.

Atualmente, para criar esses ajustes, é necessário escrever código (C #). O MES já fornece um editor integrado na interface do utilizador, com verificação de sintaxe e preenchimento automático em tempo real, mas isso ainda requer alguma atividade técnica e conhecimento de programação para atingir os objetivos necessários.

1.3 Objetivos

O objetivo desta proposta de tese de mestrado é criar uma forma mais amigável de criar Ações DEE, sem exigir conhecimentos de programação do utilizador. Um dos caminhos a seguir será a criação de um designer gráfico de código de ações DEE. O designer deverá ter uma interface amigável e intuitiva para o utilizador, onde poderá definir as ações que o sistema irá realizar, adotando uma estratégia *drag-and-drop*.

1.4 Dynamic Execution Engine (DEE) Graphical Editor

Dynamic Execution Engine (DEE) Graphical Editor representará uma nova forma de criar ações DEE no sistema MES. Tem como principal objetivo permitir a utilizadores, sem conhecimento em linguagens de programação, possam construir ações DEE apenas com base nas regras de negócio da sua fábrica eliminando desta forma a necessidade de contratação de recursos com conhecimentos na área de IT para executar este tipo de tarefa.

A solução será a disponibilização de uma interface gráfica iterativa de forma a que o utilizador apenas se foque nas suas regras de negócio e não nas regras de programação da linguagem em questão. Desta forma o utilizador poderá utilizar a interface gráfica de modo a ajustar o sistema para os seus requisitos de uma forma simples.

A interface gráfica irá disponibilizar ao utilizador vários blocos de programação pré-definidos permitindo a ligação entre os mesmos para que possa ser contruído o fluxo da ação DEE de forma a executar operações no sistema. Tais blocos irão ser colocados dentro do fluxo da ação DEE pelo utilizador utilizando uma estratégia *drag-and-drop*.

Aquando a conclusão da ação DEE por parte do utilizador, os blocos de programação e suas ligações serão interpretados e transformados em código C# aproveitando a atuais funcionalidades de compilação e execução de ações DEE do MES.

2 Contexto e Estado da Arte

2.1 Contexto

Esta secção tem como objetivo contextualizar, os principais temas da presente dissertação face ao problema exposto, nomeadamente o MES da Critical Manufacturing.

2.1.1 Sistema MES

No contexto industrial Manufacturing Execution Systems (MES) tem um importante papel na gestão e controlo dos materiais produzidos, fornecendo dados em tempo real que quando analisados levam ao desencadeamento de processos de otimização do fluxo de produção, controlo e qualidade do material.

A sua capacidade de integração com sistemas ERP e softwares que atuam na linha de produção preenchem uma das principais lacunas no controlo e gestão de uma fábrica. Com este tipo de sistema, os gestores, conseguem acompanhar todo o ciclo de vida de uma ordem de produção, podendo desta forma indicar prazos mais realistas aos seus clientes.

Do ponto de vista dos engenheiros da linha de produção o MES oferece capacidades de gestão e manutenção de recursos físicos, gestão de equipas de produção, gestão e planeamento de ordens de produção.

Do ponto de vista do operador na linha de produção o MES tem parte importante no processo de construção do produto final disponibilizando instruções e processos de controlo que irão influenciar a qualidade do produto final.

2.1.2 Critical Manufacturing MES

Um dos produtos e serviços fornecidos pela Critical Manufacturing é o MES. Atualmente, com uma grande diversidade de clientes em áreas de negócio distintas o sistema tem a capacidade

de se adaptar a qualquer tipo de produção. Isto deve-se à uniformização e identificação de processos comuns utilizados nos variados tipos de indústria que constituem a base do MES da Critical Manufacturing.

No que diz respeito aos processos específicos de um determinado tipo de indústria o sistema apresenta uma variada lista de recursos de extensibilidade que podem ser utilizados tanto pelo cliente como pelas equipas de implementação da Critical Manufacturing na modelação de requisitos para o determinado tipo de indústria, como o Dynamic Execution Engine (DEE).

2.1.3 Dynamic Execution Engine (DEE).

Dynamic Execution Engine (DEE) é um dos principais recursos de extensibilidade do MES da Critical Manufacturing. Permite às equipas de implementação e aos clientes executar ações no sistema de forma a dar resposta aos requisitos específicos da sua indústria.

Atualmente, as ações DEE são definidas em C# sendo carregadas, compiladas em memoria e executadas em *runtime*. Estas podem ser executadas antes e/ou após a execução de um método ou serviço publico da API do MES dando desta forma a possibilidade às equipas de implementação e aos clientes de geração de dados extra ou acréscimo de validações extra à aquelas que já existem na base do MES da Critical Manufacturing, tal como representado na Figura 1.

Figura 1 - Critical Manufacturing Dynamic Execution Engine

O sistema dispõe de um editor integrado de código C# onde os utilizadores podem construir as suas ações DEE. O editor contém funcionalidades de preenchimento automático e verificação de sintaxe em tempo real e está orientado a utilizadores com conhecimentos na linguagem de programação C#.

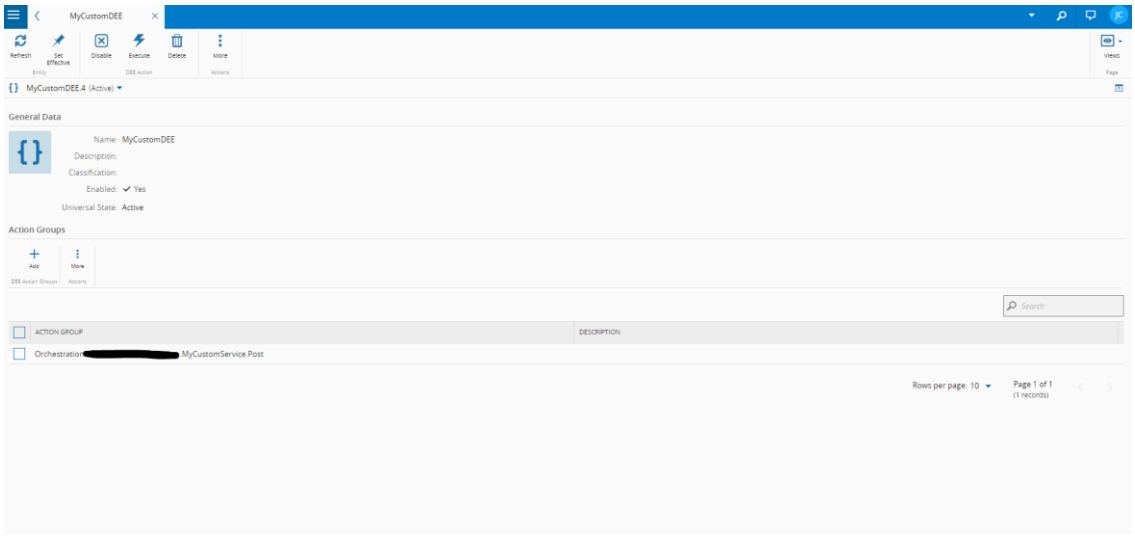


Figura 2 - Exemplo *Action Groups*

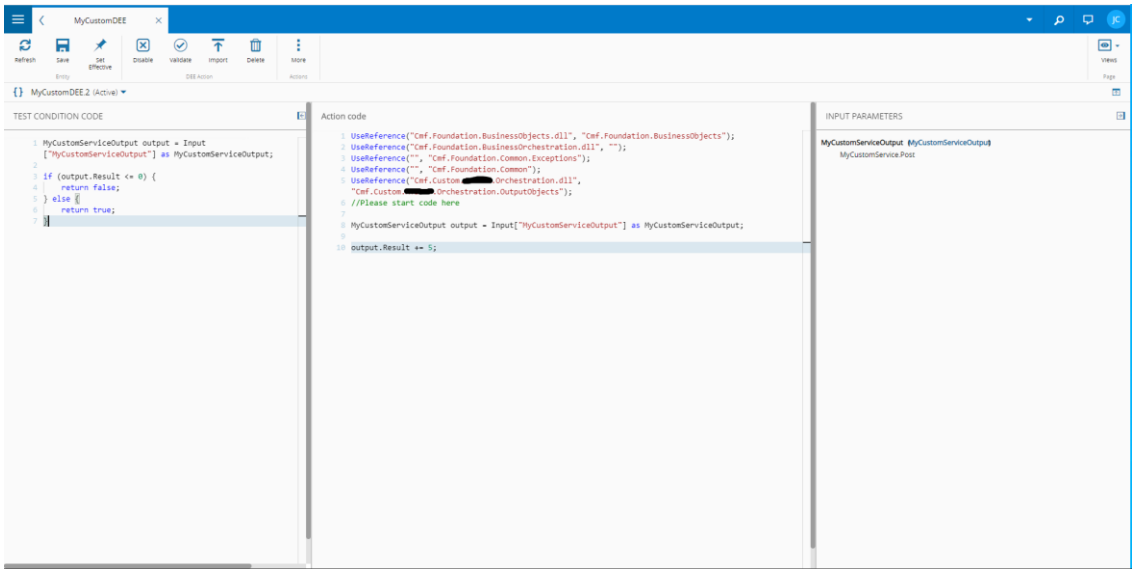


Figura 3 - Exemplo de uma Ação DEE

Na Figura 2 e 3, podemos observar um exemplo de uma Ação DEE. Uma ação DEE é constituída por dois componentes o *Test Condition Code* e *o Action code*. O *Test Condition Code* tem como objetivo indicar ao Sistema se o *Action Code* deverá ser executado. O *Action Code* contém a lógica da regra de negócio. Neste exemplo o *Test Condition Code* e o *Action Code* irão ser executados após o método MyCustomService tal como é indicado pelo *Action Group*. O objetivo desta ação DEE é adicionar cinco unidades ao resultado do método. O método retorna o parâmetro MyCustomServiceOutput que poderá ser referenciado na Ação DEE. Sendo assim o *Test Condition* verifica se o resultado do método é superior a zero, se for verdade o *Action Code* irá ser executado. No *Action Code* existe a logica de negócio em si, podemos observar que irão ser adicionadas cinco unidades ao resultado do método.

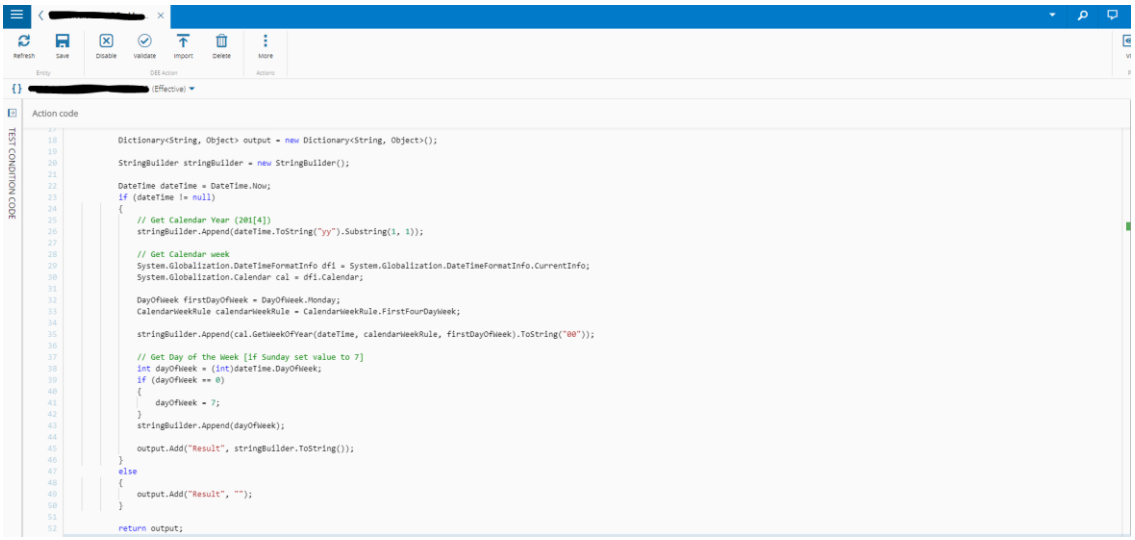


Figura 4 - Exemplo de ação DEE complexa

2.2 Block-based programming

Block-based programming é cada vez mais utilizado por instituições educacionais para introduzir os alunos à prática de programação e ao mundo de ciências da computação (Weintrop, 2019). Devido ao seu conjunto de funcionalidades, este tipo de programação consegue fazer com que o aluno apenas se foque na resolução do problema com as possibilidades que tem retirando toda a importância da sintaxe nas linguagens de programação.

2.3 Análise de abordagens existentes

Nesta secção são analisadas uma abordagem existente que utiliza o conceito de *Block-based Programming*, Blockly e Scratch.

2.3.1 Blockly

O presente capítulo tem como objetivo descrever a analise realizada a biblioteca Blockly assim como as provas de conceito realizadas e suas conclusões.

2.3.1.1 Análise

Blockly (Blockly, 2019) é uma biblioteca que oferece um editor gráfico de código que pode ser integrada em aplicações Web e móveis. O editor faz uso de blocos gráficos interligados para representar conceitos de código como variáveis, expressões lógicas, etc. Permite ao utilizador aplicar conceitos de programação sem se preocupar com problemas de sintaxe.

As principais características da biblioteca encontram-se descritas abaixo:

- A biblioteca encontra-se escrita em JavaScript, sendo todas as suas partes executadas no cliente sem qualquer dependência para o servidor;

- É compatível com a maioria dos browsers;

- Existe a possibilidade de ser customizada e estendida tendo a hipótese de adicionar blocos de programação próprios para serem utilizados pelos utilizadores;

- Os Blocos de código poderão ser exportados para diferentes tipos de linguagens de programação como JavaScript e Python.

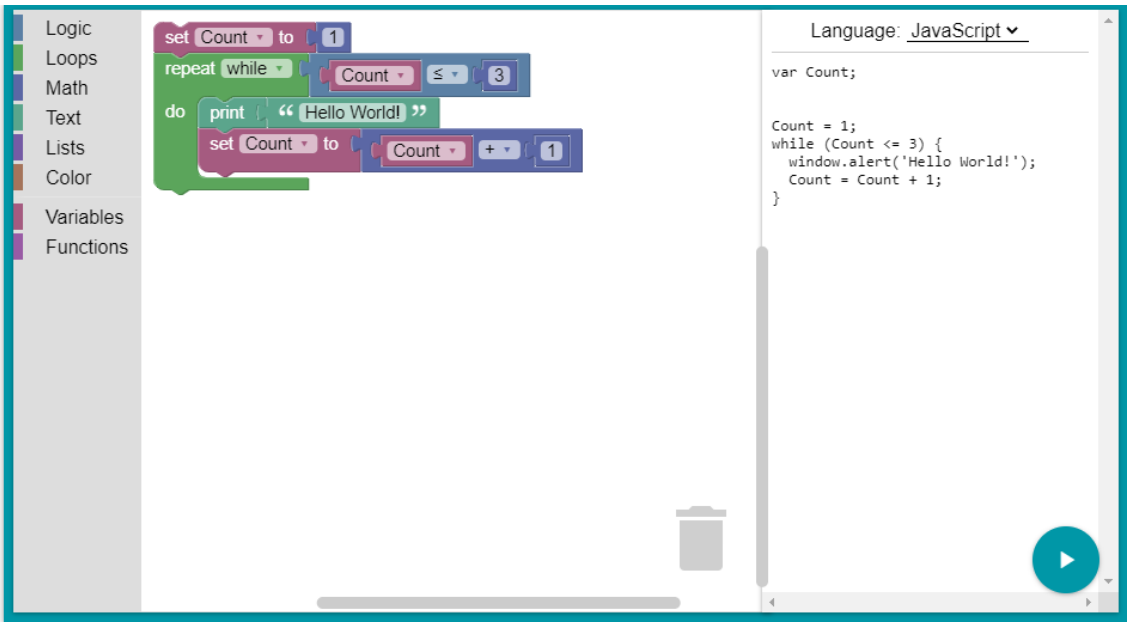


Figura 5 - Exemplo de código gerado na linguagem de programação JavaScript

A definição de um bloco pode ser realizada utilizando objetos JSON ou funções JavaScript. A utilização de funções JavaScript para definir novos blocos deverá ser preferencialmente utilizada apenas quando não é possível realizar a sua definição em um objeto JSON ou quando a instanciação de um novo bloco ocorre dinamicamente ou em *runtime*. Para bloco é possível definir cor, parâmetros de input e output, tipo de ligação, etc.

```json
{
  "type": "string_length",
  "message0": 'length of %1',
  "args0": [
    {
      "type": "input_value",
      "name": "VALUE",
      "check": "String"
    }
  ],
  "output": "Number",
  "colour": 160,
  "tooltip": "Returns number of letters in the provided text.",
  "helpUrl": "http://www.w3schools.com/jsref/jsref_length_string.asp"
}
```

Figura 6 - Exemplo de definição de um bloco em JSON

```javascript
Blockly.Blocks['string_length'] = {
  init: function() {
    this.appendValueInput('VALUE')
        .setCheck('String')
        .appendField('length of');
    this.setOutput(true, 'Number');
    this.setColour(160);
    this.setTooltip('Returns number of letters in the provided text.');
    this.setHelpUrl('http://www.w3schools.com/jsref/jsref_length_string.asp');
  }
};
```

Figura 7 - Exemplo de definição de um bloco em JavaScript

Após a definição de um bloco é necessário criar um gerador de código para esse bloco. Esse gerador é escrito em uma função JavaScript e tem como objetivo transformar o bloco na linguagem de programação escolhida. Cada função tem a capacidade de aceder aos parâmetros de input e output que se encontram na definição do bloco.

```javascript
Blockly.JavaScript['string_length'] = function(block) {
  // String or array length.
  var text = Blockly.JavaScript.valueToCode(block, 'VALUE',
      Blockly.JavaScript.ORDER_FUNCTION_CALL) || '\'\'';
  return [text + '.length', Blockly.JavaScript.ORDER_MEMBER];
};
```

Figura 8 - Exemplo de gerador para JavaScript do bloco "string_length"

Com este tipo de configuração a biblioteca permite a criação de novos blocos de código do mais variado tipo, desde a funções nativas como determinar o tamanho de uma *string* até blocos que possam representar regras de negócio complexas. O fato de permitir que sejam adicionados blocos em *runtime* e que a sua definição possa ser escrita utilizando objetos JSON permite a que possa haver algum tipo de automatismo na criação dos blocos.

Neste ponto é possível identificar algumas possíveis limitações à biblioteca, são elas:

- Inexistência do gerador de código para a linguagem de programação C#
- Construção de blocos aparentemente realizada por injeção de código o que poderá ser um processo bastante demoroso.

2.3.1.2 Prova de conceito

Para que a biblioteca possa dar resposta a um dos requisitos do projeto, as suas limitações descritas no subcapítulo anterior terão de ser ultrapassadas. No presente subcapítulo irá ser descrita a prova de conceito realizada de forma a comprovar que tais limitações poderão ser colmatadas com alguma implementação. A prova de conceito irá também fornecer indicadores de tempo de desenvolvimento necessário, usabilidade e flexibilidade do editor gráfico.

Tal como descrito na secção 2.3.1.1, a biblioteca Blockly é toda ela escrita em JavaScript e executada no cliente como tal foi criado um protótipo que passou pela integração desta biblioteca na interface gráfica do MES sendo o primeiro passo foi incluir o package na lista de dependências da GUI HTML5 do MES.

Após adicionar a dependência, foi criada uma página HTML para que fosse possível a instanciação do componente gráfico do Blockly. A instanciação implica a injeção do editor em um elemento HTML.

```
this.workspace = Blockly.inject('blocklyDiv',
{toolbox: this.toolbox.nativeElement });

this.workspace.addChangeListener(this.updateFunction.bind(this));
```

Figura 9 - Injeção do editor da biblioteca Blockly

A par da injeção do editor, é permitido também alguma configuração tal como a *toolbox* permitindo definir, no momento da inicialização, quais os blocos que estarão disponíveis.

```
<xml id="toolbox" #toolbox style="display: none">
    <block type="controls_if"></block>
    <block type="controls_whileUntil"></block>
    <block type="logic_compare"></block>
    <block type="logic_operation"></block>
    <block type="logic_boolean"></block>
    <block type="variables_get"></block>
    <block type="variables_set"></block>
    <block type="text_append"></block>
</xml>
```

Figura 10 - Exemplo de toolbox utilizada

Após a compilação do protótipo, a aplicação MES HTML é iniciada com o objetivo de observar e avaliar qual o resultado desenvolvido no protótipo até ao momento. Na Figura 9 é possível ver o resultado do mesmo.

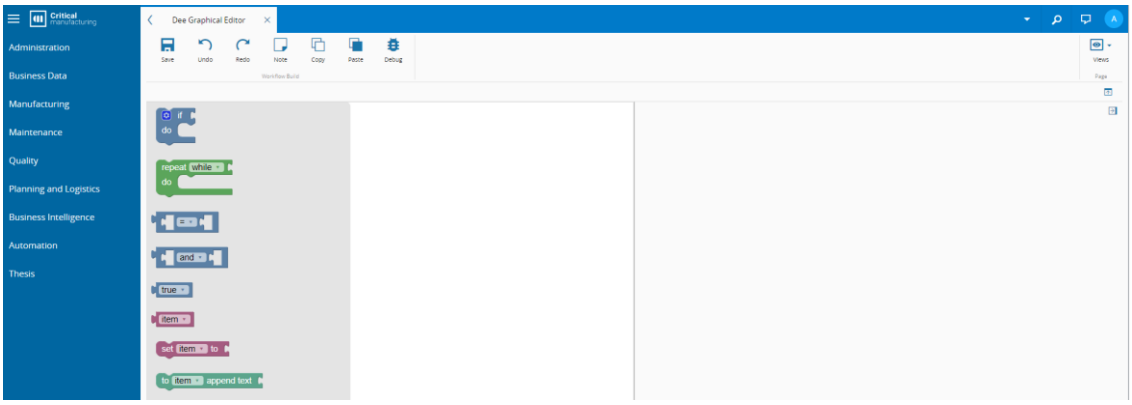


Figura 11 - Estado da página HTML após instanciação da biblioteca Blockly

A partir de este estado, é possível começar a colecionar alguns dados para os indicadores como o tempo de desenvolvimento e a usabilidade.

Até ao momento, no indicador do tempo de desenvolvimento, esta biblioteca traz excelentes mais-valias visto que o componente *drag-and-drop* requerido neste projeto é disponibilizado de uma forma simples com um baixo custo de desenvolvimento de software.

No campo da usabilidade, o editor apresenta uma boa experiência na sua utilização, dando uma boa perceção ao utilizador que ao arrastar os blocos eles devem conectar-se entre si como se tratasse de um puzzle criando o fluxo da aplicação. O fato de os blocos se assemelharem a peças de puzzle permite que o utilizador tenha uma noção mais claro como o seu programa se irá comportar.

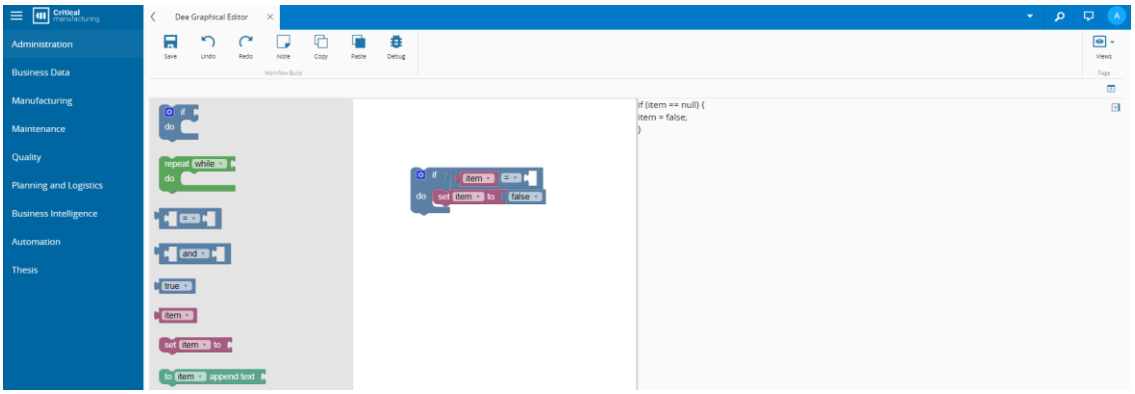


Figura 12 - Exemplo de programa

A biblioteca Blockly, até ao memento, contem integrada geradores para código JavaScript, Python, Lua, PHP e Dart. Sendo as Ações DEE escritas em código C#, a biblioteca deverá permitir a injeção de geradores de código customizáveis, neste caso específico o gerador para código C#, dando início à avaliação desta biblioteca no campo da extensibilidade.

A melhor estratégia de adicionar um novo gerador à biblioteca Blockly passa por clonar o repositório disponível no GitHub. O repositório inclui toda o código fonte e ferramentas de compilação. Após feita a analise da estrutura do repositório, é facilmente percetível a forma

como é adicionado um gerador, para tal e seguindo a estrutura da biblioteca, é adicionado um ficheiro JavaScript à pasta *generators* que irá conter todo o código necessário à instanciação e configuração do novo gerador. Para além do ficheiro que configura o gerador é colocada também uma pasta que irá conter a geração do código para cada um dos blocos como representa a Figura 12.

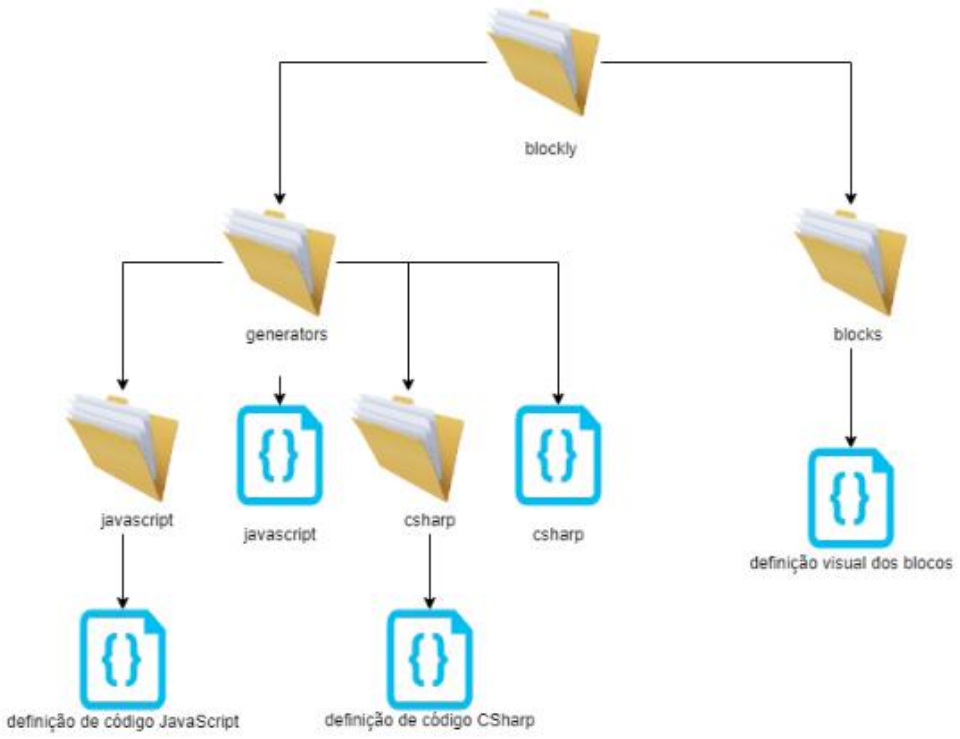


Figura 13 - Estrutura repositório Blockly

A criação de um novo gerador, para alem da instanciação do mesmo, implica a alguma configuração como por exemplo a definição das palavras reservadas da linguagem em questão para que estas não sejam utilizadas na construção do programa.

```
Blockly.CSharp = new Blockly.Generator('CSharp');

Blockly.CSharp.addReservedWords(
    //http://msdn.microsoft.com/en-us/library/x53a06bb.aspx
    'abstract,as,base,bool,break,byte,case,catch,char,checked,class,' +
    'const,continue,decimal,default,delegate,do,double,else,enum,event, ' +
    'explicit,extern,false,finally,fixed,float,for,foreach,goto,if,implicit, ' +
    'in,int,interface,internal,is,lock,long,namespace,new,null,object,operator,' +
    'out,override,params,private,protected,public,readonly,ref,return,sbyte, ' +
    'sealed,short,sizeof,stackalloc,static,string,struct,switch,this,throw,true, ' +
    'try,typeof,uint,ulong,unchecked,unsafe,ushort,using,virtual,void,volatile,while'
    );
```

Figura 14 - Criação de um novo gerador de código

Após a configuração do gerador é então necessário criar a tradução, bloco para código, para a linguagem em questão. Seguindo a arquitetura do repositório, são criados vários ficheiros separados por categoria contendo a o código em C# que é representado por determinado bloco.

```
Blockly.CSharp.load_entity = function(block) {
  var value_entity = Blockly.CSharp.valueToCode(block, 'entity', Blockly.CSharp.ORDER_ATOMIC);
  var value_levelstoload = Blockly.CSharp.valueToCode(block, 'levelsToLoad', Blockly.CSharp.ORDER_ATOMIC);

  var code = null;
  if (isNaN(value_levelstoload)) {
    code = value_entity + '.Load();\n';
  } else {
    code = value_entity + '.Load('+ value_levelstoload +');\n';
  }

  return code;
};
```

Figura 15 - Exemplo código representado pelo bloco load_entity

Na Figura 13, podemos observar um exemplo da definição do um bloco que representa a invocação do método Load(). Visualmente o bloco apresenta dois inputs, entity e levelsToLoad, que são utilizados na geração do código.

Neste ponto da avaliação à biblioteca é possível determinar, no campo da flexibilidade, que é permitido de forma rápida e simples a criação de novos geradores de código assim como também novos blocos de código.
Do ponto de vista do tempo de desenvolvimento, é possível determinar que a construção da definição visual e a definição de código de cada bloco poderá ser um processo demoroso e desgastante, no entanto a criação de uma DSL aplicando uma transformação sobre o modelo de domínio do MES poderá permitir a criação automática e dinâmica das definições necessárias para representar um bolco na linguagem de programação C#, visto que a extensibilidade da biblioteca o permite fazer.

2.3.1.3 Conclusão

Fim da análise e da prova de conceito realizada à biblioteca, conclui-se que a integração da biblioteca Blockly poderá ser determinante no desenvolvimento do projeto visto que um dos seus principais objetivos é o desenvolvimento de um editor gráfico com a estratégia *drag-and-drop*.

A biblioteca apresenta um editor gráfico com um bom nível de usabilidade, oferecendo funcionalidades de fácil compreensão ao utilizador final, reduzindo ao mesmo tempo o tempo de desenvolvimento do projeto.

Ao nível de configuração do Blockly, gerador de código C# e criação de novos blocos, a avaliação determinou que a sua construção poderá ser complicada e demorosa visto que cada bloco necessita de uma configuração visual e também a sua tradução para código C#. No entanto poderá ser desenvolvida uma DSL que represente a estrutura integral de um bloco, graficamente e logicamente, aplicando uma transformação sobre o modelo de domínio do MES com o objetivo de criar a definição de um Bloco automaticamente e dinamicamente. A DSL deverá estar englobada em uma ferramenta que será disponibilizada com o MES.

Quanto à complexidade logica de uma ação DEE, não é expectável que no final do projeto existam todos os blocos de código que darão resposta a todos os requisitos dos utilizadores, no entanto a ferramenta a desenvolver deverá permitir aos utilizadores que criem os seus próprios blocos de código de forma a estarem disponíveis para a construção da sua ação DEE.

A conclusão deste capítulo irá ser crucial no desenho da solução.

2.3.2 Scratch

Scratch (Scratch) é uma plataforma educacional orientada para os jovens e crianças, mas poderá ser utilizada por indivíduos de qualquer idade. A plataforma dispõe de um editor gráfico onde os utilizadores podem programar os seus próprios jogos e animações utilizando o conceito de *Block-based Programming*. Fazendo uso da biblioteca Blocky (Blockly, 2019), a plataforma adicionou e customizou blocos de código para ir ao encontro do seu objetivo, a criação de jogos e animações

2.4 Domain Specific Language

Uma *domain-specific language* (DSL) é uma linguagem de programação ou linguagem que oferece, por meio de notações e abstrações, poder expressivo e geralmente restrito a um problema de domínio em particular (Deursen, 2000).

É na linguagem que está concentrado todo o conhecimento de domínio enquanto todo o conhecimento de programação se encontra no compilador, permitindo assim expor os

problemas de domínio em uma linguagem de fácil compreensão permitindo a especialistas de domínio validar e desenvolver programas com base na DSL definida. Este tipo de abordagem permite a construção de software, utilizando processos simples de mapeamento do modelo conceptual em código (Cazzola, 2010).

Este tipo de abordagem poderá ser crucial no desenvolvimento do projeto, visto que uma das principais características do editor gráfico será a programação baseada em blocos, blocos esses que deverão ter uma representação logica de código de programação.

3 Análise de Valor

3.1 Modelo New Concept Development

O modelo NCD, representado na figura 2, interliga os principais elementos do Fuzzy Front End, encontrando-se dividido em 3 áreas: motor, elementos e fatores

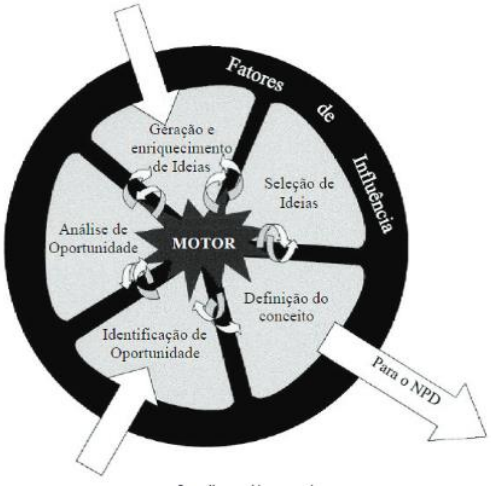


Figura 16 -Modelo New Concept Development

Na Figura 2, é representado que o processo inicia na geração de ideias ou na identificação de oportunidades e termina com a definição de conceitos prosseguindo para o processo NPD.

O modelo, é constituído por cinco elementos chave:

1. **Identificação de Oportunidade –** DEE foi concebido para que possam ser implementadas funcionalidades tanto pela Critical Manufacturing como pelos seus clientes. No entanto os clientes apontam como maior dificuldade o fato de ser necessário ter conhecimento na linguagem de programação C# para que possam modelar o sistema de forma a cumprir os seus requisitos.

2. **Análise de Oportunidade –** Após ter sido analisado diferentes abordagens para diminuir a dificuldade de criação de ações DEE e com base no estado atual do sistema conclui-se que fornecer um editor gráfico baseado em *Block-based Programing* irá facilitar aos clientes a construção de ações DEE.
3. **Geração e Enriquecimento de Ideias –** Após a análise de oportunidade foram identificadas duas alternativas de implementação da oportunidade:
    - Ideia 1: desenvolver uma ferramenta desktop com um editor gráfico em que o resultado será um artefacto a ser interpretado pelo MES ou
    - Ideia 2: desenvolver um modulo, também com um editor gráfico, como parte integrante do MES, para que possam ser utilizadas todas as funcionalidades já existentes.
4. **Seleção de Ideias –** Após reunir todas as ideias é realizado um debate para determinar qual a abordagem a seguir tendo em conta fatores como orçamento, tecnologia já existente no sistema e estado atual do sistema.
5. **Definição de Conceito –** Nesta fase é realizada uma apresentação ao *Product Owner* do sistema com o objetivo de apresentar o a ideia a seguir apresentando argumentos convincentes como a valorização do sistema atual de forma a levar ao investimento no projeto.

3.2 Valor, valor para o cliente e valor percecionado

3.2.1 Valor

O Valor tem uma definição subjetiva que depende da forma de como cada indivíduo avalia o impacto da aquisição de um produto ou serviço.

No âmbito deste projeto, a implementação da funcionalidade em causa será realizada para a Critical Manufacturing, valorizando desta forma o seu sistema de MES, tendo como objetivo final melhorar a construção de ações DEE para os seus atuais e futuros clientes.

3.2.2 Valor para o cliente

O valor para o cliente consiste na combinação entre os benefícios e sacrifícios associados a um produto ou serviço (Woodall, 2003).

Desta forma, a Critical Manufacturing terá de analisar as vantagens e desvantagens de integrar este novo modulo no MES.

3.2.3 Valor percecionado

A perceção de valor depende da expectativa do cliente possui de um determinado serviço ou produto.

O valor percecionado para o cliente no contexto deste projeto é elevado, pois remove a dependência no desenvolvimento de Ações DEE para a Critical Manufacturing, podendo desta forma o cliente investir em funcionalidades mais complexas e a Critical Manufacturing alocar recursos em outros projetos.

Na Tabela 1 é apresentada a relação entre os sacrifícios e benefícios para o cliente final.

Tabela 1 - Relação Sacrifícios/Benefícios

Domínio/Âmbito	Produto	Serviço	Relação
Benefícios	- Qualidade do Produto - Extensibilidade do Produto - Customização	- Flexibilidade - Usabilidade	- Menos dependência no desenvolvimento de novos requisitos
Sacrifícios	- Preço	- Tempo de adaptação ao produto	

3.3 Proposta de Valor

A proposta de valor representa todos os elementos que valorizam o produto e que beneficiam os seus clientes. As descrições das vantagens do produto terão de ser descritas de forma percetível para o cliente para que este possa perceber as vantagens do produto em relação aos seus concorrentes.

Sendo o Designer gráfico de ações DEE um modulo integrante do sistema MES da Critical Manufacturing este irá valorizar o atual sistema permitindo aos clientes desenvolver ações DEE sem a necessidade de conhecimentos em programação removendo essa dependência para a Critical Manufacturing.

De acordo com o parágrafo anterior identifica-se que o valor será a valorização do sistema de MES da Critical Manufacturing em relação aos sistemas concorrentes. O valor para o cliente será a disponibilização de uma forma mais iterativa para a construção de ações DEE, podendo

desta forma, o cliente implementar os seus próprios requisitos sem ter de requerer à Critical Manufacturing.

3.4 Modelo de Negócio Canvas

A geração de valor é um dos principais pilares de suporte de um negócio e para que tal acontece, a empresa deverá ser estruturada para que esse objetivo seja alcançado. O modelo Canvas permite avaliar o comportamento das várias áreas de um negócio sendo possível avaliar o impacto que cada uma delas têm.

Foi criado o seguinte modelo de Negócio Canvas de forma a suportar o negócio.

Tabela 2 - Modelo de Negócio Canvas

<table>
<tr><td rowspan="2">Parceiros Chave

Critical Manufacturing</td><td>Atividades Chave

Customização do produto por parte do cliente</td><td rowspan="2">Proposta de Valor

Interface iterativa</td><td>Relação com o Cliente

Equipa dedicada à customização do cliente</td><td rowspan="2">Segmento de Cliente

Indústria de Manufatura</td></tr>
<tr><td>Recursos Chave

Equipa de desenvolvimento</td><td>Canais de Distribuição

Contacto direto</td></tr>
<tr><td colspan="3">Estruturas de Custo

Colaboradores

Licenças de Software

Hardware</td><td colspan="2">Fluxo de Receitas

Venda do produto

Valorização do produto</td></tr>
</table>

3.5 Analytic Hierarchy Process (AHP)

O método AHP tem como objetivo auxiliar a tomada de uma decisão onde são envolvidas matérias com um elevado grau de complexidade. A tomada de decisão é basiada na privatização de critérios qualitativos e quantitativos atribuídos a cada uma das propostas (Saaty, 1990).

Com base nas ideias propostas na secção 3.1, foi contruída a arvore hierárquica de decisão.

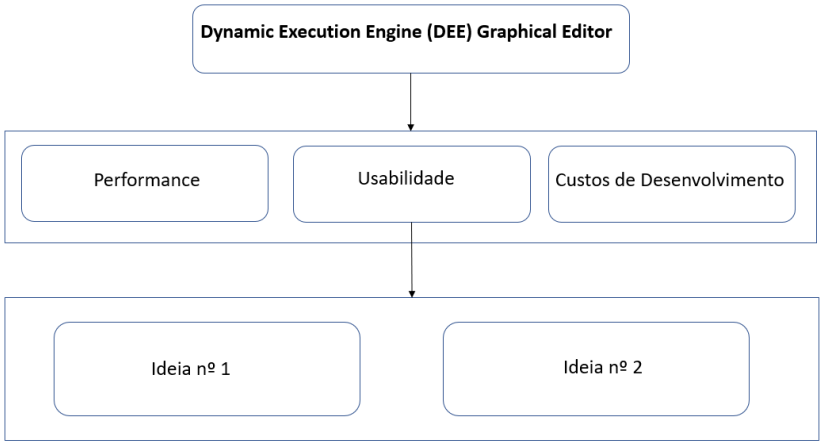


Figura 17 - Árvore hierárquica de decisão

Na primeira camada da arvore encontra a funcionalidade a ser desenvolvida pela Critical Manufacturing, sendo que esta funcionalidade contêm uma interface gráfica para o utilizador foram definidos os seguintes critérios:

- **Performance –** Verificar se a solução tem um bom nível de desempenho ao uso pela parte do utilizador;
- **Usabilidade –** Verificar nível de satisfação do utilizador ao utilizar a interface gráfica;
- **Custo de Desenvolvimento –** Custos e recursos necessários no desenvolvimento da solução.

Definidos os critérios, de seguida é atribuída um nível de importância com base na escala definida por Saaty (Saaty, 1990).

Tabela 3 - Escala fundamental – níveis de importância de comparações (Saaty, 1990)

Nível de Importância	Definição	Explicação
1	Igual Importância	As duas atividades contribuem igualmente para o objetivo.
3	Fraca Importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra.
5	Forte Importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra.
7	Importância Muito Forte	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância Absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Na seguinte tabela é atribuído o peso para cada critério acima definido, segundo a escala de AHP.

Tabela 4 - Tabela de avaliação

Critério de avaliação	Performance	Usabilidade	Custos de Desenvolvimento
Performance	1	2	6
Usabilidade	1	1	6
Custos de Desenvolvimento	1/6	1/6	1
Total	2.17	3.17	13

Pela avaliação realizada na tabela acima, o critério Performance é o que terá maior importância na decisão de qual ideia adotar. No entanto o critério Usabilidade também terá um grande impacto na decisão visto este ter como base o desenvolvimento de uma interface gráfica.

Tabela 5 - Matriz normalizada a partir do método de avaliação AHP

Critério de avaliação	Performance	Usabilidade	Custos de Desenvolvimento
Performance	0.46	0.63	0.46
Usabilidade	0.46	0.31	0.46
Custos de Desenvolvimento	0.08	0.06	0.08
Total	1	1	1

Tabela 6 - Pesos finais associados aos critérios de avaliação AHP

Critério de avaliação	Peso
Performance	0.52
Usabilidade	0.41
Custos de Desenvolvimento	0.07

Pela Tabela 6 consta-se que o critério Performance terá mais importância na decisão da ideia, no entanto o critério Usabilidade terá também um grande impacto na decisão. Ao analisar as ideias descritas no ponto 3.1 e aplicando o peso dos critérios de apoio à escolha, a ideia que apresenta maior performance e também maior usabilidade é a ideia nº2 pois irá ser integrada no sistema de MES da Critical Manufacturing tirando proveito de toda a sua infraestrutura.

4 Análise e Design

No presente capítulo, vão ser abordados os aspetos técnicos no desenvolvimento da solução assim como o levantamento de requisitos funcionais e não funcionais, decisões de design tomadas da arquitetura do sistema e da interface gráfica.

4.1 Requisitos Funcionais

Os requisitos funcionais descrevem, de forma detalhada, as funcionalidades que o sistema irá disponibilizar para que seja alcançado o objetivo para o qual foi desenvolvido.

A visualização, edição e execução de ações DEE apenas se encontra disponível para o grupo de administradores do sistema, sendo este o único interveniente das funcionalidades deste projeto. Na figura abaixo é demonstrado, através de um diagrama de casos de uso, as funcionalidades pretendidas para o sistema.

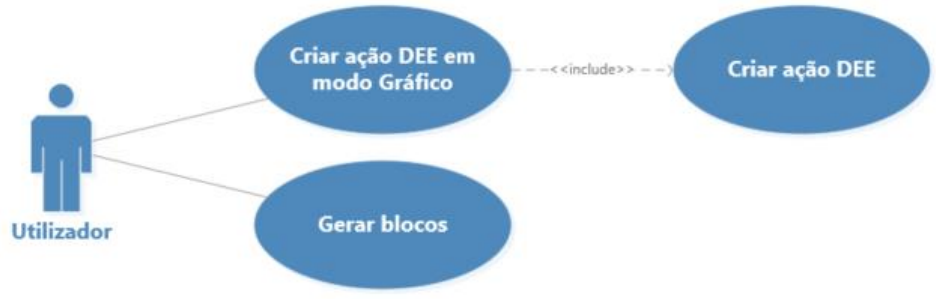


Figura 18 - Diagrama de casos de uso

Nas seguintes tabelas, é apresentada uma breve descrição para cada caso de uso retratado no diagrama acima.

Tabela 7 - Caso de Uso Criar ação DEE em modo gráfico

Caso de Uso	Criar ação DEE em modo gráfico
Descrição	O Utilizador deverá ser capaz de criar uma ação DEE em modo gráfico.
Ator	Utilizador
Pré-Condições	O utilizador deverá estar autenticado no sistema e ter um perfil que permita a criação de ações DEE.
Pós-Condições	Uma nova ação DEE deverá existir no sistema com o nome e conteúdo que o utilizador definiu.

Tabela 8 - Caso de Uso Criar ação DEE

Caso de Uso	Criar ação DEE
Descrição	O sistema deverá criar uma ação DEE valida no Sistema
Ator	Sistema MES
Pré-Condições	O código presente na Ação DEE deverá ser compilável.
Pós-Condições	A ação DEE deverá conter o novo conteúdo que o utilizador definiu.

Tabela 9 – Gerar Blocos

Caso de Uso	Gerar Blocos
Descrição	O utilizador deverá ser capaz de gerar blocos de código automaticamente.
Ator	Utilizador
Pré-Condições	N/A
Pós-Condições	Os blocos deverão ser conter uma definição gráfica e lógica valida.

4.2 Requisitos Não Funcionais

4.2.1 Usabilidade

A norma ISO 9241-11:1998 (ISO 9241-11, 1998) define o termo Usabilidade como: “capacidade de um produto poder ser usado por utilizadores específicos para alcançar determinados objetivos com eficiência, eficácia e satisfação num contexto específico de utilização”.

Mais detalhadamente a definição é seguida por três indicadores:

• Eficácia: a exatidão e integridade com que os utilizadores alcançam objetivos específicos;

• Eficiência: os recursos que se têm de dispensar para que os objetivos pretendidos sejam alcançados;

• Satisfação: a medida pela qual os utilizadores consideram que o uso do produto é aceitável.

No âmbito deste projeto, a usabilidade da funcionalidade terá um papel importante na avaliação da solução. A interface gráfica deverá de ser de fácil manuseamento permitindo ao utilizador usufruir de todas as funcionalidades disponíveis sem dificuldade.

4.3 MES Dynamic Execution Engine

Este subcapítulo tem como objetivo descrever as operações atualmente disponíveis no MES Dynamic Execution Engine assim como este modulo encontra-se integrado no sistema MES dando uma visão de como a biblioteca Blockly irá ser integrada no sistema. Será também apresentado a estrutura e lógica da atual interface gráfica assim como a sua lógica de negócio.

4.3.1 Funcionalidades disponíveis

Atualmente este modulo apresenta uma página HTML onde é possível ter acesso à vista de detalhe, vista de Código e vista de histórico.

Na vista de detalhe para alem de ser possível ver as informações básicas da Ação DEE, é também possível definir os grupos de ação onde a DEE irá ser executada. A página apresenta também alguns botões de ação onde é possível marcar a Ação DEE como efetiva, desabilitar a Ação para não ser executada, executar a ação selecionada e apagar a mesma.

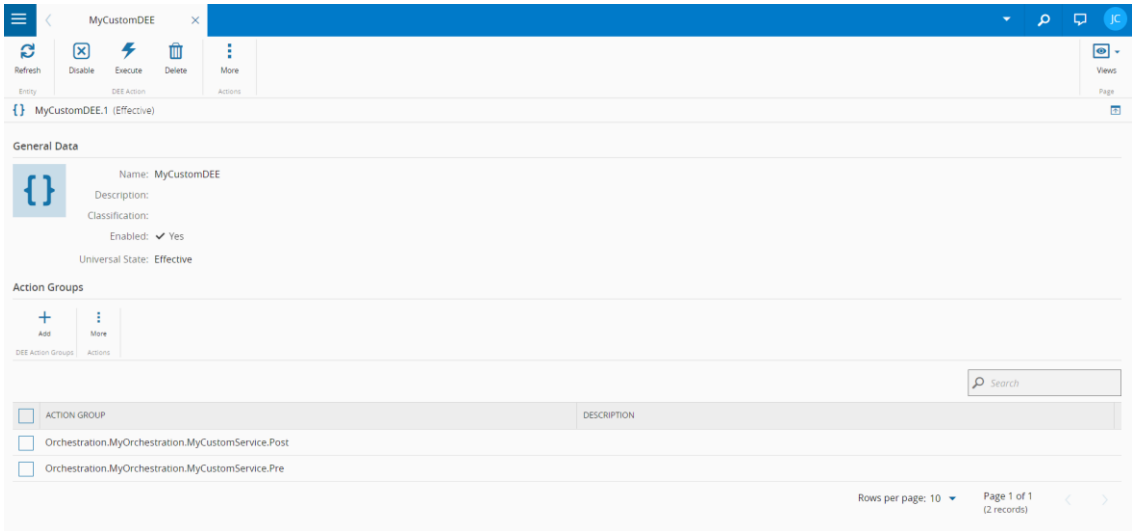


Figura 19 - Vista de detalhe Ação DEE

Na vista de código, para alem dos botões disponíveis na vista de detalhe, contem também botões para validação da DEE, esta funcionalidade compila o código e informa o utilizador se o código escrito na DEE é compilável e o botão onde é possível importar um ficheiro ".cs" que contenha a Ação DEE. No centro da pagina temos três painéis principais, *Test Condition Code*, que contêm um editor de código C# onde é escrito o código que indicará se a Ação DEE irá ser executada, *Action Code*, também contêm um editor de código C# onde irá ser escrito o código

que a Ação DEE irá executar e “Input Parameters” onde contem a informação sobre o tipo de dados que são passados por contexto para a Ação DEE.

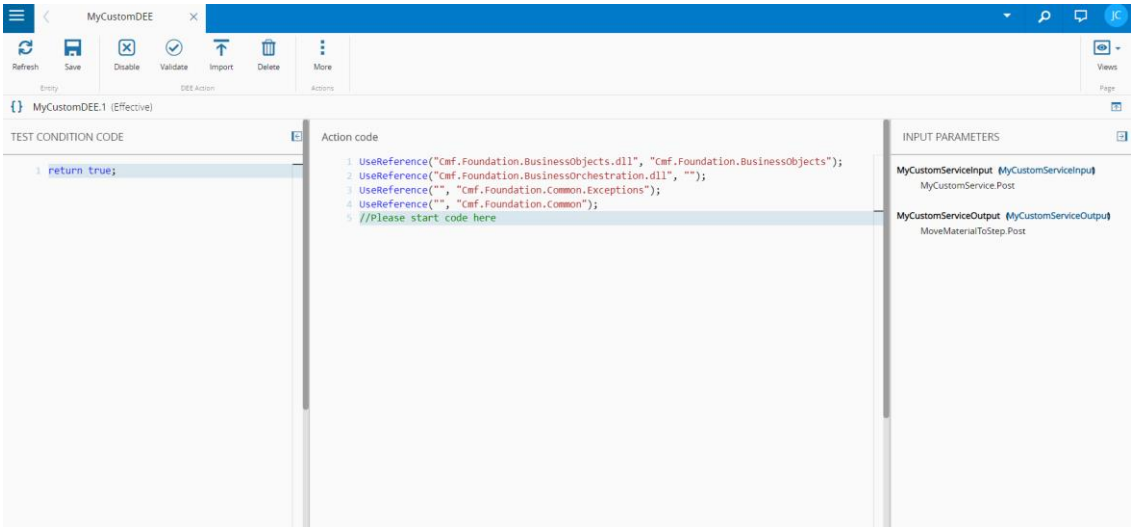


Figura 20 - Vista Código Ação DEE

Na vista de histórico podemos encontrar todas as versões da Ação DEE.

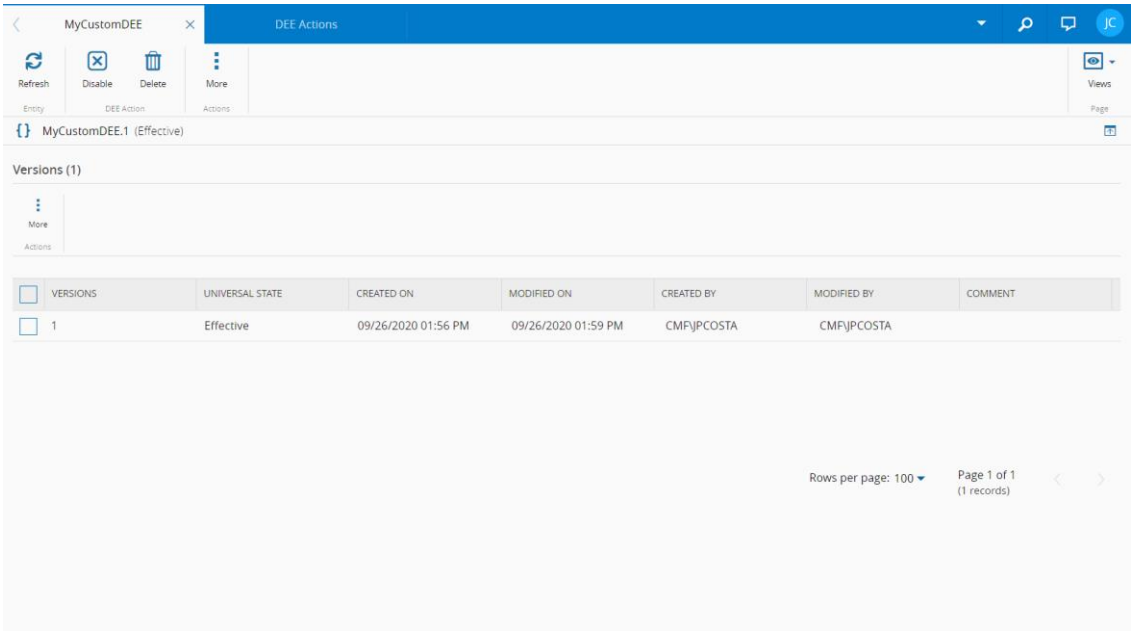


Figura 21 - Vista histórico Ação DEE

4.3.2 DEE lógica de negócio

Tal como anteriormente mencionado, o Dynamic Excution Engine é uma funcionalidade já existente no MES, sendo que o seu modulo assenta sobre a atual arquitetura do sistema. O modulo integra as 3 camadas que constituem o sistema:

- Camada de apresentação – contêm toda a logica de apresentação da funcionalidade incluindo o editor de código C# para a elaboração da ação DEE. Será nesta camada que o Blockly irá ser integrado.
- Camada de negócio – alberga todas as regras de negócio do sistema. Especificamente para este modulo, é disponibilizado operações de criação, edição, execução e validação de uma ação DEE.
- Camada Analítica/Base de Dados – Onde se encontram estruturadas todas as entidades que suportam o sistema MES.

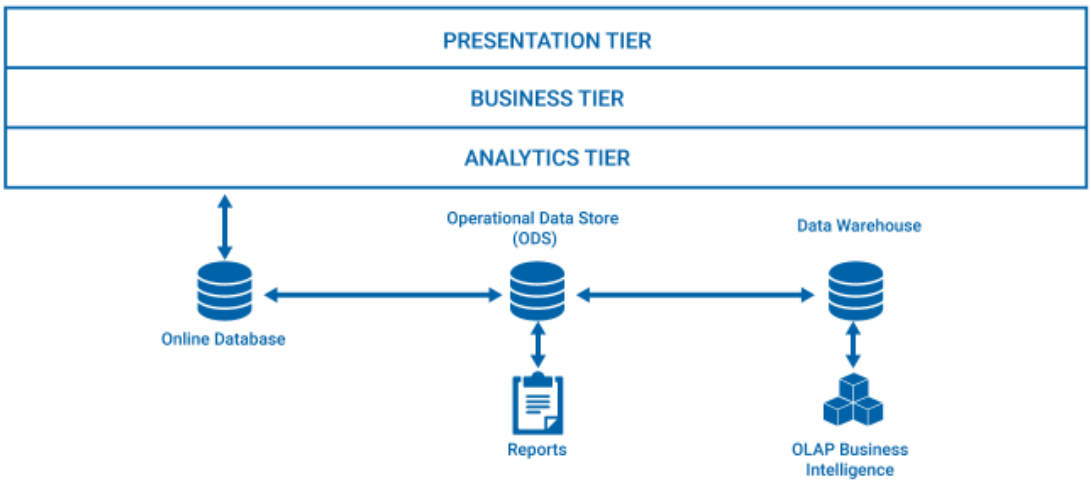


Figura 22 - Arquitetura em Camadas MES (Critical Manufacturing, 2020)

Relativamente ao modelo relacional que dá o suporte ao modulo DEE, este é constituído por três entidades:

- T_Action – representa uma Ação DEE no sistema, incluindo o código que a mesma irá executar.
- T_ActionGroup – Representa o *trigger* que levará à execução de uma ação DEE.
- T_ActionGroup_Action – representa a relação entre as entidades *Action* e *ActionGroup* com o objetivo de representar as ações DEE que irão ser executadas num determinado *trigger.*

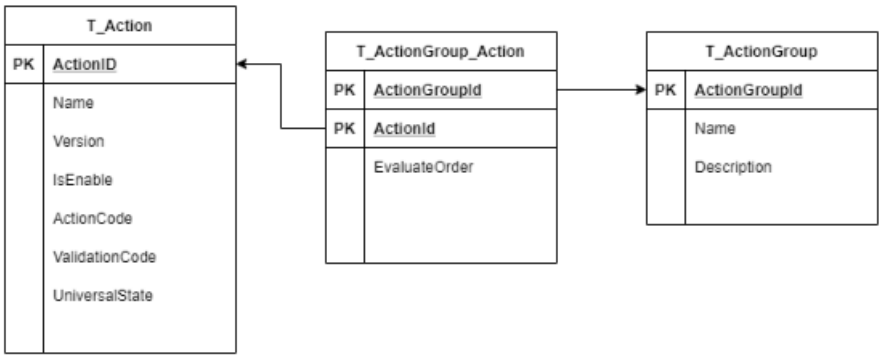


Figura 23 - Modelo relacional DEE

4.4 Blockly

Este subcapítulo tem como objetivo descrever de que forma a biblioteca Blockly irá ser integrada no sistema MES assim como irão ser descritos todos os componentes auxiliares à biblioteca.

4.4.1 Blockly integrado no MES HTML5 GUI

A biblioteca Blockly é totalmente escrita em JavaScript e não depende de um servidor para ser executada. Como tal, esta biblioteca irá ser integrada como uma libraria externa na GUI de HTML5 do MES. Sendo que existe a necessidade de estender esta biblioteca para responder aos requisitos do projeto, será criado um package onde constará todo o código necessário para estender a biblioteca.

Visualmente, este novo editor gráfico, deverá integrar a atual página HTML já existente para criação e edição de Ações DEE como uma nova vista disponível para o utilizador. Como tal, o package "cmf.core.admin.dee" que alberga toda a logica referente ao modulo de DEE, terá agora uma nova dependência para o package "cmf.core.blockly" onde constará toda a logica de extensão da biblioteca Blockly.

4.4.2 Geração automática de Blocos

Atualmente, no sistema MES, já existem algumas ferramentas com o propósito de apoio ao desenvolvimento e também à execução do sistema. Estes tipos de ferramentas são entregues

a cada versão do MES fazendo parte do seu sistema. Porém, estas ferramentas não são executadas pelo utilizador final do MES, mas sim por administradores de sistemas ou por equipas de implementação. Assim sendo esta nova ferramenta irá fazer parte do leque de ferramentas já existentes integrando no sistema MES.

De acordo com as a avaliação realizada à biblioteca e suas conclusões descritas nos subcapítulos 3.2.1 e 3.3, existe a necessidade de criar uma ferramenta de apoio ao desenvolvimento deste projeto com o objetivo gerar dinamicamente e automaticamente a definição gráfica e logica dos blocos de código que irão fazer parte a biblioteca.

A ferramenta, com o nome "BlocksGenerator", terá de ter a capacidade de extrair todos os métodos públicos que todo o sistema MES referencia e gerar os artefactos necessários para que estes sejam interpretados e executados pelo Blockly.

Relativamente à geração dos artefactos, estes terão como apoio, uma DSL que representará a linguagem de domínio do Blockly que irá contemplar toda a informação necessária para que seja aplicada uma transformação de forma a gerar os artefactos com a definição dos blocos. Desta forma os utilizadores desta ferramenta poderão criar o próprio modelo de acordo com a DSL definida com o objetivo de criar blocos de código sem que tenha de escrever, por código, a definição do mesmo.

4.5 Arquitetura de Software

Esta secção tem como objetivo descrever a arquitetura de software, assim como o processo já existente e todos os novos componentes que irão se desenvolvidos.

4.5.1 Vista lógica

A vista logica foca-se na principal funcionalidade que o projeto irá disponibilizar ao utilizador final demonstrando as suas partes integrantes e interações.

Como tal, de forma a representar todas as partes integrantes e interações da funcionalidade, foi criado o diagrama de sequência ilustrado na Figura 24 relativo à ação de criação de uma ação DEE no sistema.

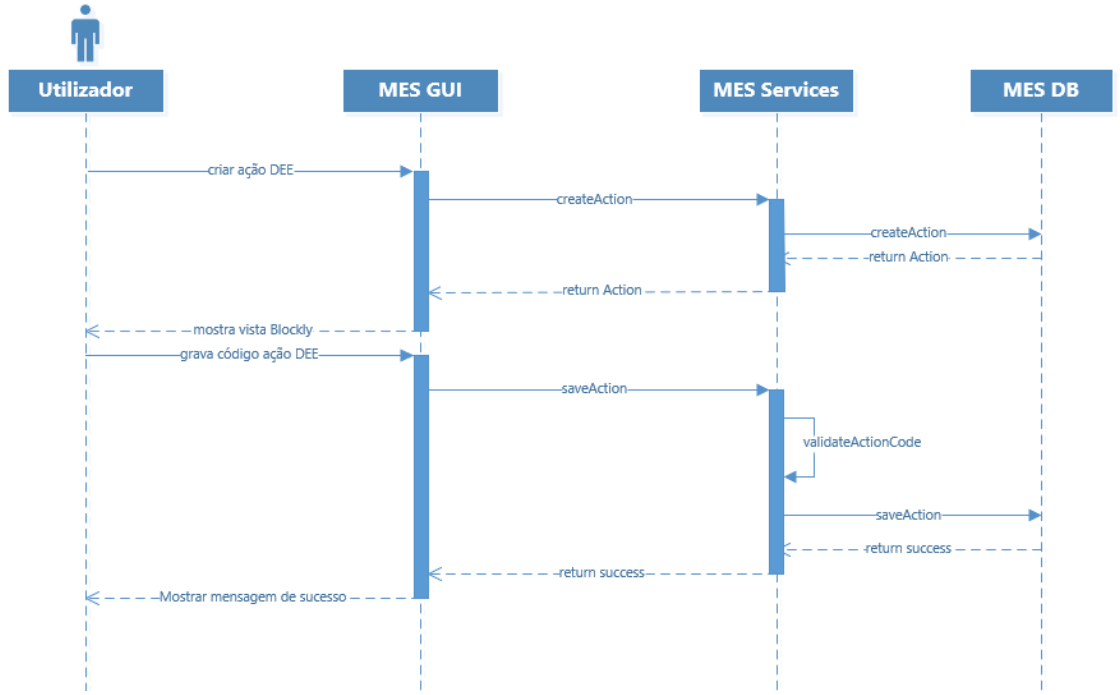


Figura 24 - Diagrama de sequência de alto nível representativo da criação de uma ação DEE

Pelo diagrama de sequência ilustrado em cima, é possível constatar que a primeira interação com a interface gráfica é feita com o objetivo de criar uma ação DEE no sistema. Após a criação, é retornada a resposta de sucesso para a interface gráfica de forma a que esta seja redirecionada para a vista de edição, onde constará o editor gráfico, de forma a que o utilizador construa a ação DEE. Após a construção, relativa à regra de negócio da ação DEE, o utilizador interage com a interface gráfica de forma persistir o seu estado. Por sua vez, o código construído, será validado e compilado sendo e seguida persistido na base de dados.

O componente BlocksGenerator terá como principal responsabilidade a geração automática de blocos. Como *output* a ferramenta deverá gerar um artefacto para que este seja consumido pela biblioteca Blockly contendo a definição logica e gráfica de cada bloco. Na Figura 25 é representado através de um diagrama de sequência todas as partes integrantes e interações da ferramenta.

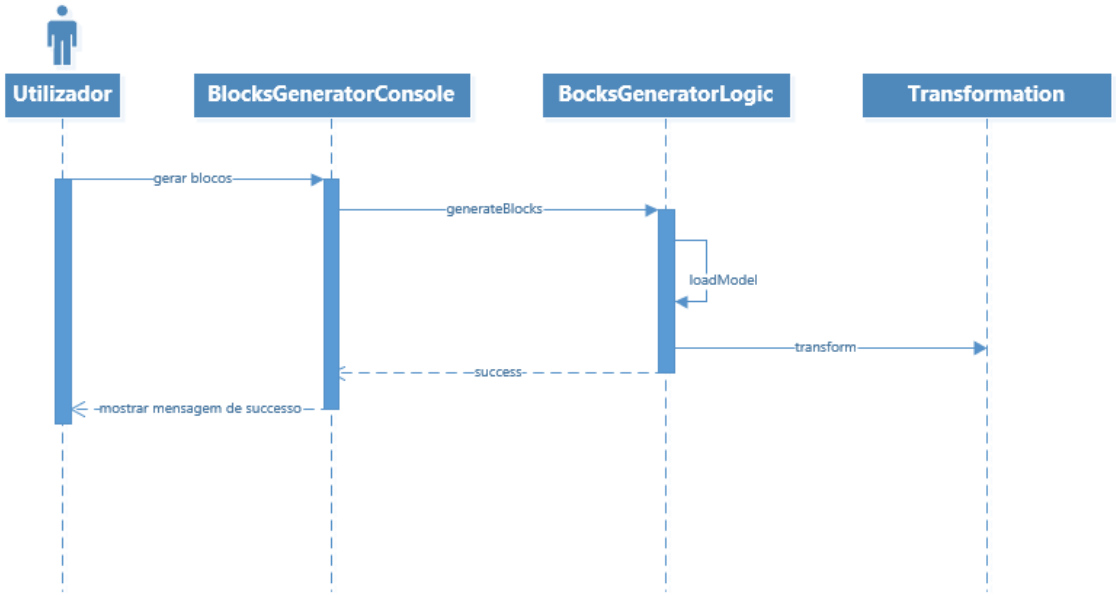


Figura 25 - Diagrama de sequência de alto nível representativo da geração automática de blocos

Pela figura acima, consta-se que a ferramenta disponibilizará uma consola para que o processo de geração de blocos seja iniciado pela o utilizador. Esta, dará início ao processo na camada logica que por sua vez será responsável pelo carregamento do modelo ao qual será aplicada uma transformação com o objetivo de gerar os artefactos contendo a definição dos bolcos.

4.5.2 Vista de implementação

No presente subcapítulo serão ilustrados e descritos todos os módulos constituintes do projeto assim como os seus componentes e relações. Para isso, são ilustrados dois diagramas de componentes de forma a dar uma perspetiva de alto nível da arquitetura do sistema.

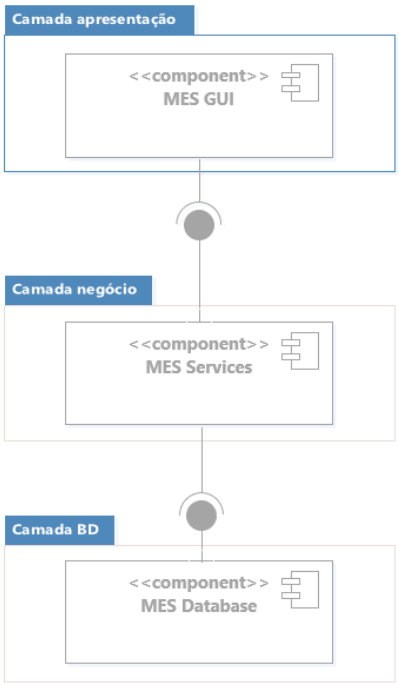


Figura 26 - Diagrama de componentes de alto nível da arquitetura do MES

Na Figura 26, é possível observar a separação de camadas do sistema MES, são elas: camada de apresentação, camada de negócio e camada de base de dados.

Na camada de negócio são encontradas todas das regras de negócio constituintes do sistema.

A camada de base de dados, é responsável pela persistência e processamento de dados.

A camada de apresentação contêm toda a logica de apresentação do sistema. Representa a interação do utilizador com o MES, dando início ao processo da maior parte das funcionalidades do sistema, comunicando exclusivamente com a camada de negócio. Será nesta camada que a maior parte do projeto irá ser integrada, como tal é ilustrada em detalhe na Figura 26 a integração do projeto com a camada de apresentação.

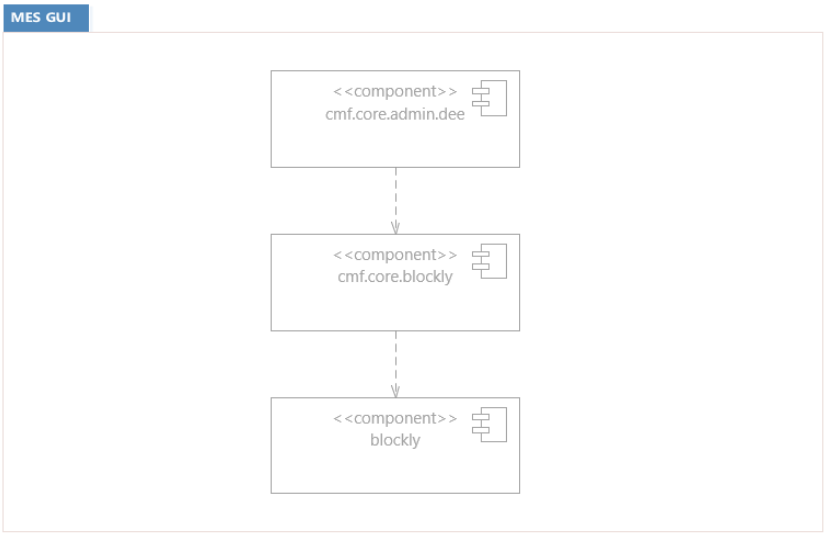


Figura 27 - Diagrama de componentes de alto nível das partes integrantes do projeto

Focando apenas nos componentes essenciais do projeto, constata-se pela figura acima ilustrada que o projeto irá ser suportado por 3 componentes assim como as suas dependências. O detalhe de cada um dos componentes será descrito no capítulo 6.

5 Construção

Após a definição e análise da arquitetura e requisitos, este capítulo tem como objetivo descrever o desenvolvimento técnico no contexto deste projeto. O capítulo irá documentar o desenvolvimento da integração da biblioteca Blockly no MES HTML assim como o desenvolvimento da ferramenta BlocksGenerator.

5.1 Editor Gráfico

Este subcapítulo tem como objetivo descrever a integração da biblioteca Blockly no sistema MES assim como o desenvolvimento de todos os módulos que são suporte a esta integração. O subcapítulo é dividido pelos diferentes módulos desenvolvidos com o objetivo de facilitar a compreensão do desenvolvimento.

5.1.1 Modulo blockly

A biblioteca Blockly é *open source* e possível de estender, dando a possibilidade a programadores de adicionar funcionalidades à mesma desde que respeitem as diretrizes da biblioteca, tendo inclusive ferramentas de compilação do código.

Como tal, a biblioteca, foi adicionada como dependência em constante desenvolvimento do sistema MES, isto é, o seu código fonte é incluído como modulo do MES para que possam ser adicionadas as funcionalidades descritas no subcapítulo 3.2.1. com a perspetiva de enriquecer a biblioteca e disponibilizar as mesmas na comunidade Blockly.

Seguindo as conclusões retiradas na avaliação da biblioteca inicia-se o desenvolvimento do gerador de código C#. Para isso é adicionado é adicionado o ficheiro JavaScript à pasta *generators*, que irá conter toda a logica relativa ao gerador de código. Para alem de conter a

instanciação do gerador, este ficheiro também contém as configurações especificas da linguagem de programação C#.

```
Blockly.CSharp = new Blockly.Generator('CSharp');

Blockly.CSharp.addReservedWords(
    //https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/
    'abstract,as,base,bool,break,byte,case,catch,char,checked,class,' +
    'const,continue,decimal,default,delegate,do,double,else,enum,event, ' +
    'explicit,extern,false,finally,fixed,float,for,foreach,goto,if,implicit, ' +
    'in,int,interface,internal,is,lock,long,namespace,new,null,object,operator,' +
    'out,override,params,private,protected,public,readonly,ref,return,sbyte, ' +
    'sealed,short,sizeof,stackalloc,static,string,struct,switch,this,throw,true, ' +
    'try,typeof,uint,ulong,unchecked,unsafe,ushort,using,virtual,void,volatile,while'
    );
```

Figura 28 - Exemplo de instanciação do gerador de código C# e configuração das palavras reservadas

Após a definição do gerador é dado início à construção dos blocos estáticos da linguagem C#, isto é, a definição logica de blocos com funções nativas da linguagem de programação. Para isso, é adicionado na mesma pasta, alguns ficheiros JavaScript, separados por categoria funcional, que contem a tradução do bloco para código C# para cada um dos blocos que já contem a sua definição gráfica.

```
Blockly.CSharp.controls_if = function() {
  // If/elseif/else condition.
  var n = 0;
  var argument = Blockly.CSharp.valueToCode(this, 'IF' + n,
      Blockly.CSharp.ORDER_NONE) || 'false';
  var branch = Blockly.CSharp.statementToCode(this, 'DO' + n);
  var code = 'if (' + argument + ') {\n' + branch + '}';
  for (n = 1; n <= this.elseifCount_; n++) {
    argument = Blockly.CSharp.valueToCode(this, 'IF' + n,
        Blockly.CSharp.ORDER_NONE) || 'false';
    branch = Blockly.CSharp.statementToCode(this, 'DO' + n);
    code += ' else if (' + argument + ') {\n' + branch + '}\n';
  }
  if (this.elseCount_) {
    branch = Blockly.CSharp.statementToCode(this, 'ELSE');
    code += ' else {\n' + branch + '}\n';
  }
  return code + '\n';
};
```

Figura 29 - Extrato de código contendo a tradução para código C# de um bloco que representa as condições If/else if/else

A função presente na Figura 25 representa a definição logica de um bloco de código com identificador “controls_if”, isto é, o seu retorno deverá conter o código C# correspondendo à sua definição gráfica.

Figura 30 - Resultado da definição gráfica do bloco "controls_if"

```
if (false) {
}
```

Figura 31 - Resultado da definição lógica do bloco "controls_if"

5.1.2 Modulo cmf.core.blockly

A construção deste modulo teve como objetivo abstrair toda a lógica correspondente à biblioteca Blockly expondo de forma tipificada apenas as configurações necessárias à instanciação do componente assim como customizações adicionais à biblioteca como alteração do estilo e adição de blocos de código em *runtime*.

De forma a facilitar a integração por componentes que irão invocar este modulo, foram criadas algumas estruturas de forma a tipificar algumas configurações necessárias à biblioteca. Um dos elementos cruciais a configurar é a ToolBox. Este elemento representa os blocos que estarão visíveis ao utilizador podendo estes estarem organizados por categoria. Este elemento permite também a injeção de botões de ação. A Figura 28 representa a estrutura definida para a configuração da ToolBox.

```typescript
// represents a toolbox item definition
export interface ToolBox {
    // ToolBox items
    items: Array<ToolboxItem>
}

// represents a toolbox item definition
export interface ToolboxItem {
    // block id
    id: string,
    // category id
    category: string,
    // uses a html <button> instead a <block>
    isButton: boolean,
    // callback function for onclick button event
    callback: any,
    // button text
    text: string
}
```

Figura 32 - Estrutura definida para configuração da ToolBox

A ToolBox pode ser representado pelo um elemento HTML que se encontra escondido na página, permitindo desta forma a definição de blocos estáticos, isto é, blocos que deverão estar sempre visíveis independentemente do contexto em que o modulo é invocado. Para permitir a injeção de blocos pelo componente que invoca este modulo, para alem da estrutura demonstrada na Figura 28 foi também criado uma função que injeta os blocos na ToolBox.

```typescript
if (this.toolBoxDefinition && this.toolBoxDefinition.items && this.toolBoxDefinition.items.length > 0) {

    let itemsToAppend = "";

    for (const toolItem of this.toolBoxDefinition.items) {
        let item = "";

        if (toolItem.isButton) {
            item = '<block type=' + toolItem.id + '></block>';
        } else {
            item = '<button text=' + toolItem.text + ' callbackKey=' + toolItem.callback + '></button>';
        }

        if (toolItem.category) {
            const category = this.toolbox.nativeElement.getToolboxItemById(toolItem.category);
            $(category).append(item);
        } else {
            itemsToAppend += item;
        }
    }

    $(this.toolbox.nativeElement).append(itemsToAppend);
}
```

Figura 33 - Extrato de código responsável por adicionar blocos à ToolBox

Após a definição da ToolBox é necessário instanciar a o Blockly. No contexto deste modulo o que se pretende aquando a instanciação do Blockly é a injeção do componente num elemento HTML e a subscrição do evento disparado quando algum bloco é adicionado ou removido ao editor gráfico disponibilizado pela biblioteca.

```
public ngAfterViewInit(): void {

    this.workspace = Blockly.inject('blocklyDiv',
    {toolbox: this.toolbox.nativeElement });

    this.workspace.addChangeListener(this.updateFunction.bind(this));
}

public updateFunction () {

    const code = Blockly.CSharp.workspaceToCode(this.workspace);

    this.onCodeChangeEmitter.emit(code);
}
```

Figura 34 - Extrato de código de inicialização do Blockly

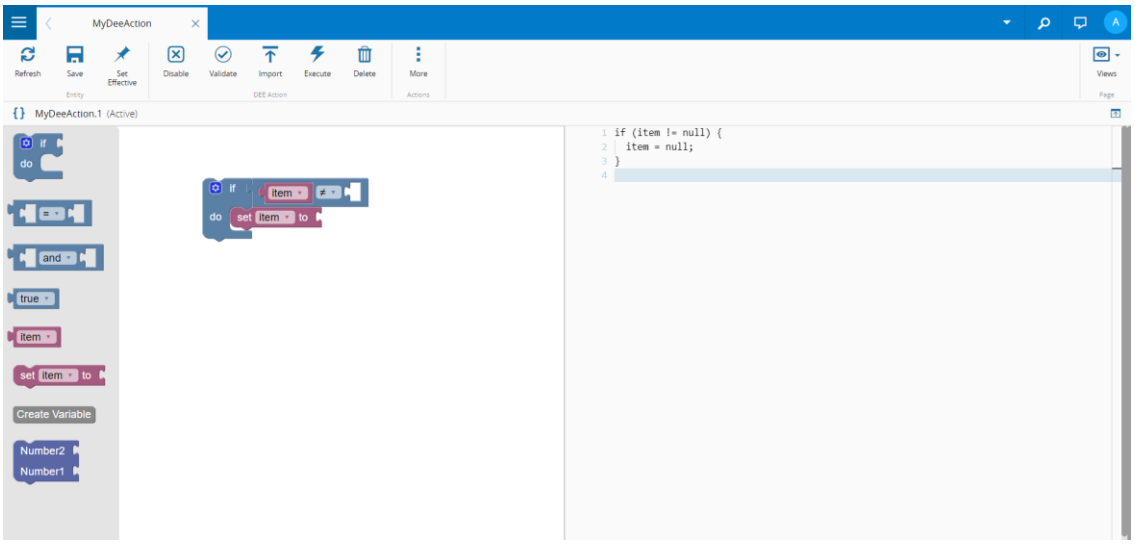
A função que é invocada aquando a alteração do fluxo de blocos no editor gráfico do Blockly é responsável pela invocação de uma das funções da biblioteca que aplica a transformação de blocos para o código C# retornando em texto o código que é representado pelos blocos que se encontram definidos no editor gráfico. Por sua vez a função propaga o código gerado até aos componentes que invocam este modulo.

5.1.3 Dynamic Execution Engine Graphical Editor

Sendo o *Dynamic Execution Engine* uma funcionalidade já existente no MES, faz todo o sentido que o novo editor gráfico faça parte do modulo onde esta funcionalidade se encontra definida, aproveitando todas as funções já implementadas. Para tal, foi adicionada uma nova vista à página HTML já existente.

Figura 35 - Definição de vista Blockly na página da Ação DEE

Esta nova vista leva à parametrização e invocação do modulo cmf.core.blockly onde são configurado todos os blocos que deverão estar visíveis. Desta forma é tirado o partido de todas as funcionalidades existentes na vista de código relativas à edição e validação de uma ação DEE.



5.2 BlocksGenerator

Esta secção visa a descrever e detalhar o desenvolvimento da ferramenta dando uma perspetiva técnica da forma como é que a mesma foi construída. Devido à complexidade do desenvolvimento, o subcapítulo estará dividido nas várias fazes de implementação: BlockGenerator domain-specific-language, transformação model-to-code, carregamento do modelo.

5.2.1 BlockGenerator domain-specific-language

Um dos requisitos principais desta ferramenta é o utilizador poder gerar e adicionar a definição logica e gráfica dos blocos de código de forma a serem elegíveis para a biblioteca Blockly. Para dar resposta aos requisitos, através do Visual Studio Modeling SDK, foi contruída uma linguagem especifica do domínio do Blockly na componente da representação de um bloco graficamente e logicamente.

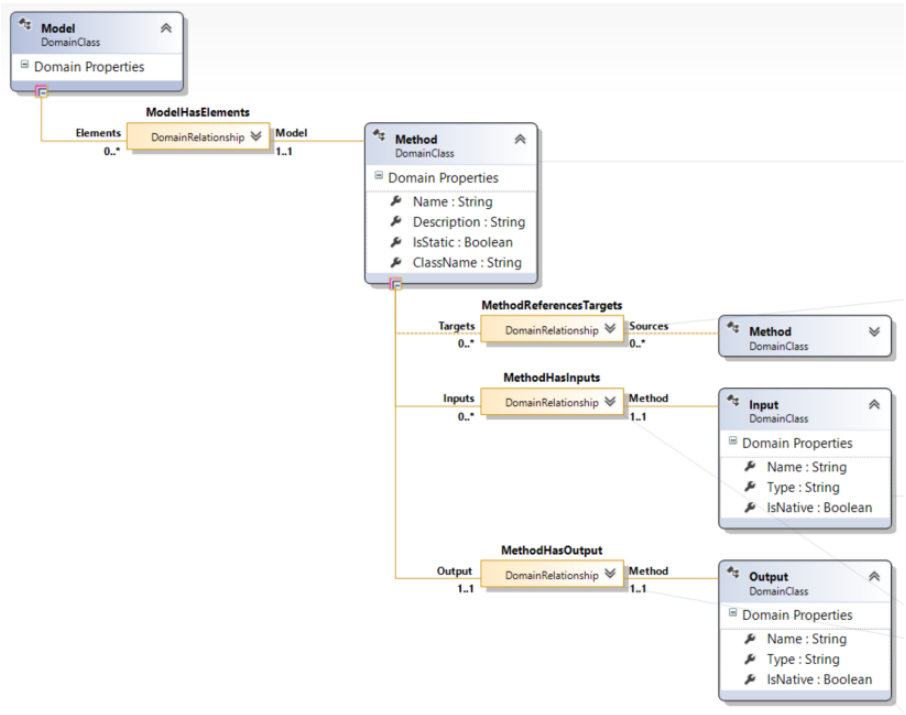


Figura 36 - DSL Blockly

A DSL é composta pelas seguintes classes de domínio:

- Model – Domínio do sistema, agrega todas as partes de interesse do sistema.
- Method – Representa um determinado método referenciado pelo sistema MES. Contêm também informação relativamente à classe onde esta contido assim como a indicação se o método é estático. Apresenta também informação relativa aos seus parâmetros de input e output.
- Input – Representa um argumento de input de um determinado método contendo informação sobre o seu tipo de dados.
- Output – Representa o objeto de retorno de um determinado método contendo informação sobre o seu tipo de dados.

É com esta DSL que iremos iniciar o processo de transformação e criação dos métodos existentes no sistema MES. Importante referir também que com a utilização do Visual Studio Modeling SDK os utilizadores finais desta ferramenta poderão construir instâncias da DSL com o objetivo de criar definições de blocos.

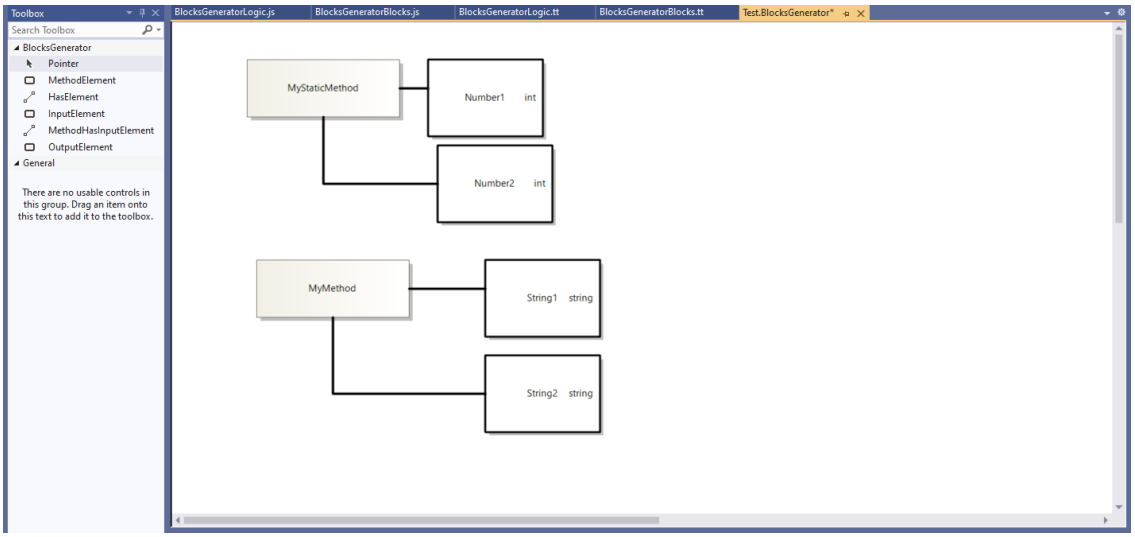


Figura 37 - Criação de instância através do Visual Studio Modeling SDK

```
<?xml version="1.0" encoding="utf-8"?>
<model xmlns:dm0="http://schemas.microsoft.com/VisualStudio/2008/DslTools/Core" dslVersion="1.0.0.0" Id="cfbce240-be32-4acb-ad46-8243a17c9ed5" xmlns="http://schemas.microsoft.com/dsltools/BlocksGenerator">
  <elements>
    <method Id="378a475d-7cff-420f-bd25-927bd75e78a1" name="MyStaticMethod" isStatic="true" className="MyStaticClass">
      <inputs>
        <methodHasInputs Id="db2987f0-54c6-45ae-8747-bf6130c76d19">
          <input Id="622bbd2c-7808-46d0-87e7-c73f33fda39d" name="Number2" type="int" isNative="true" />
        </methodHasInputs>
        <methodHasInputs Id="3cffc83f-250d-445b-ad6f-c5455ef6d560">
          <input Id="7989dc13-39d8-4740-a8bd-0dc43e218b46" name="Number1" type="int" isNative="true" />
        </methodHasInputs>
      </inputs>
      <outputs>
        <methodHasOutputs Id="da56f291-7d50-426e-b77e-75f140421e77">
          <output Id="fbe1a8bf-069f-444c-8737-e33813f0bbe7" name="result" type="int" isNative="false" />
        </methodHasOutputs>
      </outputs>
    </method>
  </elements>
</model>
```

Figura 38 - Resultado da instância criada

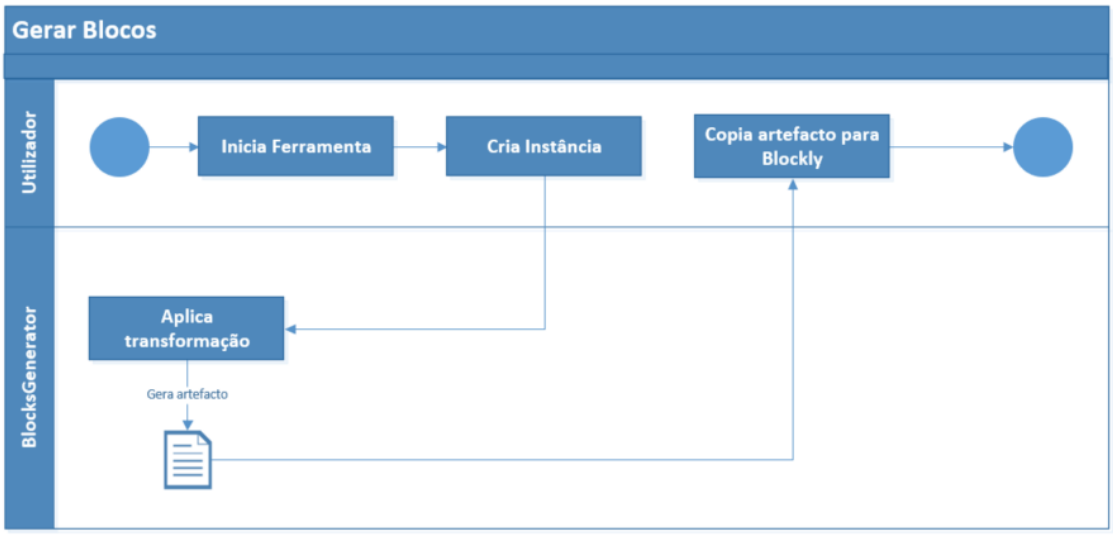


Figura 39 - Diagrama de atividade Gerar Blocos

5.2.2 Transformação model-to-code

Após a definição da DSL e a criação de instâncias da mesma, é necessário aplicar uma transformação à instância de forma a que seja construído os artefactos necessários para a adição de novos blocos à biblioteca Blockly.

Tendo em conta que a estrutura logica da definição de um bloco do Blockly se encontra dividida em dois ficheiros, são eles a definição gráfica e logica, foram criadas dois transformações para gerar os mesmos artefactos:

- BlocksGeneratorBlocks – através de uma instância gera o artefacto com a definição gráfica de um bloco.

- BlocksGeneratorLogic - através de uma instância gera o artefacto com a definição logica de um bloco

```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation" #>
<#@ output extension=".js" #>
<#@ BlocksGenerator processor="BlocksGeneratorDirectiveProcessor" requires="fileName='ModelToBlocksGenerator.BlocksGenerator'" #>
'use strict';
goog.provide('Blockly.Constants.Cmf');
goog.require('Blockly');
goog.require('Blockly.Blocks');
goog.require('Blockly.FieldDropdown');
goog.require('Blockly.FieldLabel');
goog.require('Blockly.Mutator');
Blockly.Constants.Cmf.HUE = 210;
Blockly.defineBlocksWithJsonArray([
<#
  foreach (Method element in this.Model.Elements)
  {
        var message = "";
#>
{
    "type": "<#= element.Name #>",
    "args0": [
<#
  int counter = 1;
  foreach (Input input in element.Inputs)
  {
          message += input.Name + " %" + counter.ToString() + " ";
          counter++;
#>
        {
            "type": "input_value",
            "name": "<#=input.Name#>",
            "check": "<#=input.Type#>",
        },
<#
  }
#>
    ],
    "message0": "<#=message#>",
    "colour": 230,
    "tooltip": "<#=element.Name#>",
    "helpUrl": "",
    "previousStatement": null,
    "nextStatement": null,
},
<#
    }
#>
]);
```

Figura 40 – Transformação BlocksGeneratorBlocks

```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation" #>
<#@ output extension=".js" #>
<#@ BlocksGenerator processor="BlocksGeneratorDirectiveProcessor" requires="fileName='Test.BlocksGenerator'" #>
'use strict';
Blockly.CSharp.logic = {};
<#
    foreach(Method element in this.Model.Elements)
    {
        var finalCall = "";
        var methodName = element.Name;
        if(element.IsStatic){
            finalCall =element.ClassName + "." + element.Name;
        } else {
             finalCall = element.Name;
        }
#>
Blockly.CSharp.<#=methodName#> = function(block) {
  var input = Blockly.CSharp.valueToCode(block, 'input', Blockly.CSharp.ORDER_ATOMIC);
  // TODO: Assemble CSharp into code variable.
  var code = "";
  <#
        var methodArgs = "";
    foreach (Input input in element.Inputs)
    {
            var propName = input.Name;
            if(string.IsNullOrEmpty(methodArgs)) {
                methodArgs += propName;
            } else {
                methodArgs += ", " + propName;
            }

  #>
  var propValue = Blockly.CSharp.valueToCode(block, "<#=propName#>", Blockly.CSharp.ORDER_ATOMIC);
  code += "<#= input.Type#> <#=propName#> = "+propValue +";\n";
  <#
    }
  #>
  code += "<#=finalCall#>(<#=methodArgs#>);\n";
  return code;
};
  <#
    }
  #>
```

Figura 41 – Transformação BlocksGeneratorLogic

Ambas as transformações geram ficheiros JavaScript contendo a logica de injeção de blocos na biblioteca. Após a aplicação da transformação na instância criada e demonstrada na figura 22, obtemos os seguintes artefactos:

```
goog.require('Blockly.FieldLabel');
goog.require('Blockly.Mutator');
Blockly.Constants.Cmf.HUE = 210;
Blockly.defineBlocksWithJsonArray([
{
    "type": "MyStaticMethod",
    "args0": [
        {
            "type": "input_value",
            "name": "Number2",
            "check": "int",
        },
        {
            "type": "input_value",
            "name": "Number1",
            "check": "int",
        },
    ],
    "message0": "Number2 %1 Number1 %2 ",
    "colour": 230,
    "tooltip": "MyStaticMethod",
    "helpUrl": "",
    "previousStatement": null,
    "nextStatement": null,
},
{
    "type": "MyMethod",
    "args0": [
        {
            "type": "input_value",
            "name": "String1",
            "check": "string",
        },
        {
            "type": "input_value",
            "name": "String2",
            "check": "string",
        },
    ],
    "message0": "String1 %1 String2 %2 ",
    "colour": 230,
    "tooltip": "MyMethod",
    "helpUrl": "",
    "previousStatement": null,
    "nextStatement": null,
},
]);
```

Figura 42 - Resultado do ficheiro JavaScript definição gráfica

5.2.3 Carregamento do modelo

De forma a realizar uma geração automática de blocos de todos os métodos e propriedades referenciadas pelo sistema MES com o objetivo de poderem ser utilizados no editor gráfico, a ferramenta BlocksGenerator irá ler e colecionar todos os dados necessários presentes das assemblies referenciadas pelo sistema. Para isso, é passado para a ferramenta o caminho físico onde se encontra a camada de negócio do sistema que contem todas as assemblies referenciadas pelo mesmo. Após verificar que o caminho é valido, a ferramenta faz o carregamento de todas assemblies existentes na pasta.

```
public IEnumerable<Assembly> LoadAll(string baseDirectory, string pattern, string exclusionPattern, IEnumerable<string> additionalAssemblies = null)
{
    List<Assembly> loadedAssemblies = new List<Assembly>();

    DirectoryInfo dirInfo = new DirectoryInfo(baseDirectory);
    var allAssembliesInDirectory = dirInfo.GetFiles(pattern).ToList();
    additionalAssemblies = additionalAssemblies ?? Enumerable.Empty<string>();

    var additionalAssembliesWithoutExtension = additionalAssemblies.Select(a => Path.GetFileNameWithoutExtension(a)).Distinct();
    var allFiles = dirInfo.GetFiles();

    allAssembliesInDirectory.AddRange(allFiles.Where(file => additionalAssembliesWithoutExtension.Contains(Path.GetFileNameWithoutExtension(file.Name))));

    //Get all the assemblies  in the directory that start with Cmf and end with .dll
    foreach (var assemblyFileInfo in allAssembliesInDirectory)
    {
        string assemblyName = assemblyFileInfo.Name.ToLower();
        string assemblyFileNameWithoutExtension = Path.GetFileNameWithoutExtension(assemblyFileInfo.Name);

        var isAdditionalAssembly = additionalAssembliesWithoutExtension.Contains(assemblyFileNameWithoutExtension);

        //Ignoring all the assemblies that that don't match the pattern
        if (!isAdditionalAssembly && IsToIgnore(assemblyName))
        {
            continue;
        }

        // load assembly
        loadedAssemblies.Add(Assembly.LoadFile(assemblyFileInfo.FullName));
    }

    return loadedAssemblies;
}
```

Figura 43 - Extrato de código de carregamento de *assemblies*

Após o carregamento das assemblies a ferramenta começa a colecionar a informação de cada um dos Types presentes na mesma. Neste tipo de objeto podemos obter informação relativa aos métodos existentes assim como os seus parâmetros de input e output.

```
private List<Method> GetMethods(Type type)
{
    List<Method> methodsList = new List<Method>();

    var methods = type.GetMethods();

    foreach (var method in methods)
    {
        Method m = new Method();
        m.Name = method.Name;
        m.IsStatic = method.IsStatic;
        m.ClassName = method.ReflectedType.Name;


        m.Output = new Output();
        m.Output.Name = method.ReturnParameter.Name;
        m.Output.Type = method.ReturnParameter.ParameterType.Name;

        m.Inputs = new List<Input>();

        foreach (var par in method.GetParameters())
        {
            Input i = new Input();
            i.Name = par.Name;
            i.Type = par.ParameterType.Name;
            i.IsClass = par.ParameterType.IsClass;

            m.Inputs.Add(i);
        }

        methodsList.Add(m);
    }

    return methodsList;
}
```

Figura 44 - Extrato de código de leitura de métodos por cada Type

Após a extração da informação relativa aos métodos de cada Type presente em cada uma das *assemblies* carregadas, é aplicada uma transformação à classe Model para obter o artefacto que representará o modelo do sistema MES no contexto da DSL desenvolvida para a geração de blocos.

```
<#@ property name="ModelRoot" type="Model" processor="PropertyProcessor" #>
<?xml version="1.0" encoding="utf-8"?>
<model xmlns:dm0="http://schemas.microsoft.com/VisualStudio/2008/DslTools/Core" dslVersion="1.0.0.0" I
  <elements>
  <#
    foreach(var method in ModelRoot.Methods)
    {

 #>
 <method name="<#=method.Name#>" isStatic="<#=method.IsStatic#>" className="<#=method.ClassName#>" >
 <inputs>
 <#
    foreach(var input in method.Inputs)
    {

 #>
 <methodHasInputs>
    <input name="<#=input.Name#>" type="<#=input.Type#>" isNative="<#=input.IsClass#>" />
  </methodHasInputs>
 <#
    }
  #>
  </inputs>
   <#
    if(method.Output != null)
    {

 #>
  <outputs>
    <methodHasOutputs>
        <output name="output" type="<#=method.Output.Type#>" isNative="<#=method.Output.IsClass#>" />
    </methodHasOutputs>
  </outputs>
  <#
    }
  #>
 </method>
  <#
    }
  #>
  </elements>
</model>
```

Figura 45 - Extrato de código transformação Model to Model

Aquando a geração da instância do modelo, esta poderá ser importado e validado contra a DSL contruída, sendo aplicada a transformação demonstrada na Figura 45 obtendo assim os artefactos necessários para contendo a definição dos blocos.

6 Avaliação da solução

A avaliação da solução desenvolvida tem um papel fundamental para determinar a satisfação dos clientes do MES da Critical Manufacturing quanto à nova funcionalidade implementada.

6.1 Abordagem

Esta secção irá descrever a abordagem utilizada na avaliação da solução.

6.1.1 Grandezas

Sendo o principal objetivo deste projeto o desenvolvimento de um designer gráfico de ações DEE foram definidas as grandezas apresentadas nos subtópicos abaixo.

6.1.1.1 Usabilidade

O designer gráfico deverá providenciar uma interface gráfica de simples utilização e iterativa para que os utilizadores não sintam dificuldades em e se sintam satisfeitos ao utilizar a funcionalidade.

Com isto, a interface gráfica deverá ter um simples layout, seguindo os padrões já existentes no sistema e deverá ajudar o utilizador a construir as ações DEE mostrando exemplos e dando sugestões.

6.1.1.2 Utilidade

Esta nova funcionalidade irá integrar um sistema já existente, portanto é importante avaliar se o que irá ser desenvolvido é útil para os utilizadores finais e se acrescentou algum valor ao sistema já existente.

6.1.2 Hipóteses

De forma a avaliar o projeto são estabelecidas duas hipóteses, descritas abaixo, sendo que ambas vão ao encontro dos objetivos definidos.

6.1.2.1 1ª Hipótese

Com esta nova funcionalidade, espera-se que os administradores do sistema deixem de recorrer a pessoas com conhecimentos na área de tecnologias de informação para construir ação DEE. Com esta hipótese é pretendido avaliar a usabilidade e utilidade da funcionalidade.

- Hipótese a refutar: necessidade de recorrer à Critical Manufacturing ou a pessoas com conhecimentos técnicos para construir uma ação DEE.

- Hipótese a comprovar: Utilizadores com conhecimento do negócio e sem capacidades técnicas são capazes de criar ações DEE.

6.1.2.2 2ª Hipótese

Tendo em consideração que que este projeto será a adição de uma nova funcionalidade no sistema já existente, é importante avaliar se os utilizadores finais não encontram dificuldades no manuseamento da funcionalidade. Esta hipótese vias a testes a usabilidade da funcionalidade.

- Hipótese a refutar: O utilizador encontra dificuldades em perceber o objetivo da funcionalidade e no manuseamento da interface gráfica.

- Hipótese a comprovar: O utilizador consegue perceber e interpretar todas as funcionalidades que a interface gráfica dispõe não encontrando dificuldades no manuseamento da mesma.

6.1.3 Metodologia de avaliação

Este subcapítulo tem como objetivo a descrição das metodologias de avaliação a utilizar no âmbito do projeto.

6.1.3.1 Testes de usabilidade

Antes de a integração da funcionalidade numa das versões oficiais do MES, será realizada uma formação e será disponibilizada toda a documentação a um grupo de pessoas apenas com conhecimentos de negócio.

O objetivo destes testes terá o como principal objetivo perceber se a funcionalidade desenvolvida permite a possibilidade, este grupo de pessoas, de criar ações DEE sem ter formação técnica e especifica para tal.

6.1.3.2 Inquéritos de satisfação

Serão realizados inquéritos de satisfação, não só para o grupo de pessoas descritas na metodologia anterior, mas também para pessoas que irão criar as ações DEE utilizando competências técnicas de forma a perceber até que ponto esta nova funcionalidade consegue cobrir a funcionalidade já existente.

6.2 Avaliações realizadas

Na presente secção é descrito o conteúdo dos questionários realizados a todas as partes interessadas do projeto assim como os resultados obtidos dos mesmos.

6.2.1 Inquéritos de satisfação

Após a fase de implementação do projeto estar concluída, foi dado o acesso a uma máquina virtual onde se encontra instalado o MES com o novo editor gráfico de ações DEE, a um conjunto restrito de pessoas para que possam testar as capacidades desenvolvidas no projeto. Em conjunto com o acesso à máquina virtual, foi também disponibilizado um inquérito de satisfação para avaliar o nível de satisfação do resultado do projeto.

Optou-se pela criação de um inquérito simples com o objetivo de colecionar as opiniões dos utilizadores com as seguintes características:

- Tempo de resposta ao inquérito entre 5 a 10 minutos;
- Constituído por 5 questões no total: 4 de resposta fechada e 1 de resposta aberta;
- 4 das respostas fechadas são constituídas por 5 opções: discordo completamente, discordo, sem opinião, concordo e concordo plenamente.

Nas figuras e tabelas abaixo encontram-se as questões incluídas nos inquéritos, assim como a descrição do questionário.

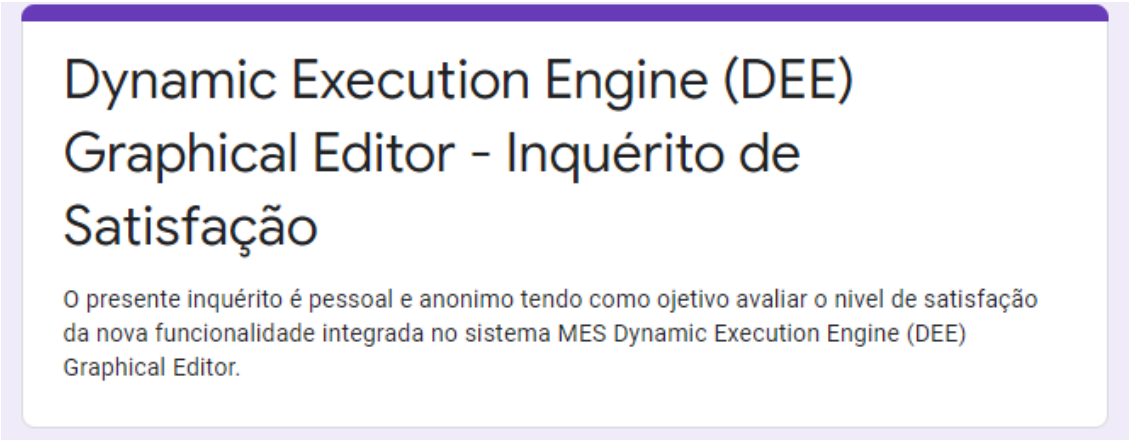


Figura 46 - Descrição do questionário realizado

Tabela 10 - Questões presentes no questionário

Questão Realizada	Tipo de Resposta
Com o novo editor Gráfico foi possível construir uma ação DEE sem recorrer a recursos com conhecimentos técnicos na linguagem de programação C#.	Fechada
É facilmente percetível o modo de funcionamento do novo editor gráfico de ações DEE.	Fechada
Considera que todas as funcionalidades relativas às Ações DEE se encontram implementadas no novo editor gráfico?	Fechada
Foi facilmente possível a definição de novos blocos.	Fechada
O que gostaria de melhorar no novo editor gráfico de ações DEE?	Aberta

6.2.2 Resultados do inquérito

Após o período experimental da funcionalidade, foram realizadas cinco questões com o objetivo de compreender o nível de satisfação desta nova funcionalidade. As respostas às questões colocadas podem ser encontradas nas seguintes figuras.

Com o novo editor Gráfico foi possível construir uma ação DEE sem recorrer a recursos com conhecimentos técnicos na linguagem de programação C#

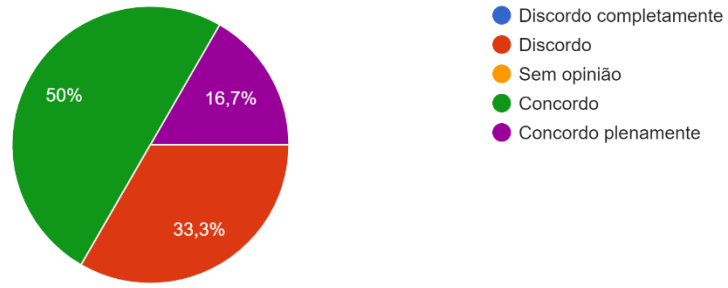


Figura 47 - Questão nº1: Criação de uma ação DEE em modo gráfico

É facilmente percetível o modo de funcionamento do novo editor gráfico de ações DEE.

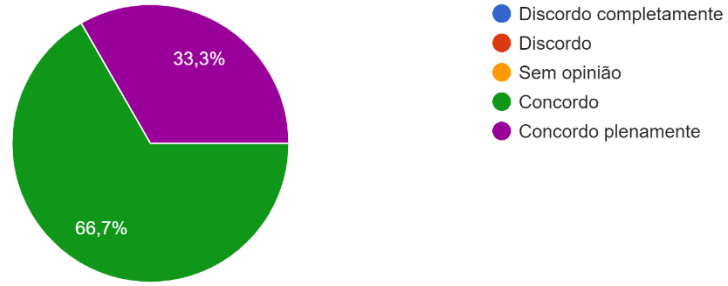


Figura 48 - Questão nº2: Usabilidade do editor gráfico

Considera que todas as funcionalidades relativas às Ações DEE se encontram implementadas no novo editor gráfico?

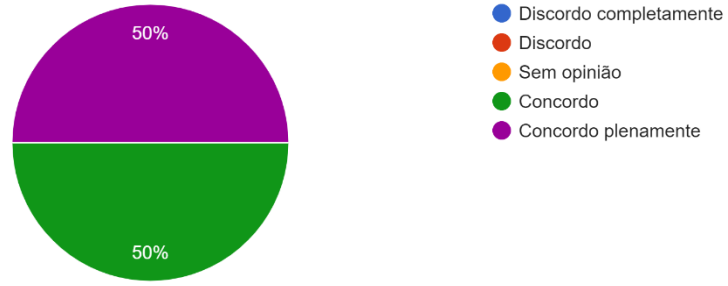


Figura 49 - Questão nº3: Funcionalidades existentes

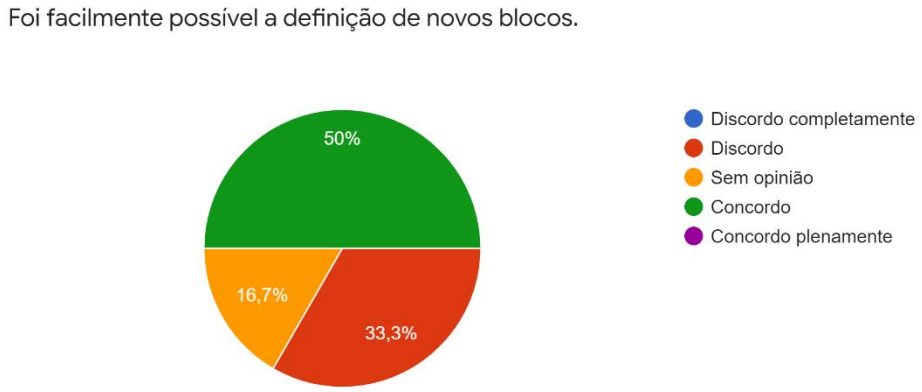


Figura 50 - Questão nº4: Criação de novos blocos

De seguida, encontram-se as respostas obtidas à questão "O que gostaria de melhorar no novo editor gráfico de ações DEE?":

- Nível de abstração dos blocos deveria ser maior;
- Ainda é necessário algum conhecimento técnico na elaboração de uma Ação DEE.

6.2.3 Análise dos resultados obtidos

Após a obtenção dos resultados do inquérito realizado é possível observar que aproximadamente 57% dos inquiridos foi capaz de construir uma ação DEE sem recorrer a conhecimentos técnicos relativos à linguagem de programação C#. Aproximadamente 33% dos inquiridos tiveram alguma dificuldade na construção da ação DEE sendo este número consideravelmente elevado.

Na questão de usabilidade do editor gráfico, os resultados foram bastante positivos sendo que 100% dos inquiridos concorda que o editor gráfico é de fácil utilização.

Quando questionados sobre a implementação das funcionalidades existentes no novo editor gráfico, 50% dos inquiridos concorda que as funcionalidades estão completamente implementadas e outros 50% concordam.

Relativamente à construção de novos blocos 50% concorda que a ferramenta permite facilmente a definição de novos blocos, aproximadamente 17% não tem opinião e aproximadamente 33% afirma ter alguma dificuldade na definição de novos blocos.

Quanto à questão em aberto "O que gostaria de melhorar no novo editor gráfico de ações DEE?" os inquiridos demonstram que o nível de abstração que um bloco apresenta, encontra-se ao

nível da linguagem de programação C# sendo que se fosse apresentado um nível de abstração maior poderia facilitar o processo de construção de uma ação DEE.

6.2.4 Conclusão

Para concluir, com base na análise feita aos resultados do inquérito, é possível afirmar que o novo editor gráfico de ações DEE trouxe melhorias no processo que construção de uma ação DEE para utilizadores que não tem conhecimentos na linguagem de programação C#. No entanto também é possível afirmar que a definição de blocos carece de um nível de abstração maior para que a construção da ação DEE seja mais facilmente.

Sendo o nível de abstração dos blocos de código ser o foco principal para os comentários menos positivos determina-se que uma das próximas ações a tomar será a revisão da DSL definida da ferramenta BlocksGenerator. Ao definir uma DSL mais abstrata em relação á linguagem de programação, sendo possível modelar processos e operações mais comuns, tornará os blocos de código mais complexos, mas mais percetíveis para o utilizador final.

Numa segunda fase do projeto, fase de melhorias, um dos aspetos principais será identificar processos. Com isto pretende-se que operações comuns na indústria da manufatura sejam compiladas em métodos complexos podendo desta forma serem gerados blocos de código que representam muito mais do que uma operação computacional, abstraindo desta forma estas operações da linguagem de programação C# e expondo-a em uma linguagem mais acessível a utilizadores que não tenham conhecimentos nesta tecnologia.

7 Conclusões e trabalho futuro

Neste capítulo são apresentadas as conclusões acerca do projeto desenvolvido. São descritos todos os objetivos alcançados com o desenvolvimento do Dynamic Execution Engine (DEE) Graphical Editor assim como as dificuldades ao longo do projeto e o trabalho futuro a realizar, com o objetivo de melhorar o modulo de acordo com as necessidades dos utilizadores, no sentido de acrescentar valor ao sistema MES. É também realizada uma apreciação final do projeto, bem como uma apreciação a nível pessoal.

7.1 Objetivos alcançados

O maior objetivo do projeto proposto consiste no desenvolvimento de um editor gráfico com recurso à estratégia *drag-and-drop* onde os utilizadores seriam capazes de construir uma regra de negócio sem requerer a recursos com conhecimentos técnicos sobre a linguagem de programação C# evitando erro de sintaxe na escrita de regras de negócio. Devido à escolha da abordagem a adotar, foram também identificados um conjunto de objetivos secundários com o intuito de enriquecer a funcionalidade e o projeto em si como a geração automática da definição de blocos e a possível criação de novos blocos com o recurso à modelação.

Desta forma, considera-se que o principal objetivo assim como os secundários foram cumpridos, na medida em que o Sistema MES oferece uma nova funcionalidade que permite a construção de ações DEE recorrendo a um editor gráfico assim como todas as ferramentas inerentes à gestão da biblioteca Blockly.

A adição deste modulo ao sistema MES vem também acrescentar o enriquecimento tecnológico do mesmo assim como permite aos clientes da Critical Manufacturing possam escrever regras de negócio sem que recorram com tanta frequência à equipa de implementação do projeto.

7.2 Trabalho futuro/melhorias

Apesar de editor gráfico permitir a construção de ações DEE, sem que o utilizador tenha a necessidade de escrever o código C#, o nível de abstração da definição lógica dos blocos encontra-se ao nível do modelo de domínio do sistema MES. Prevê-se que no futuro, seja criado no sistema uma forma de agregar processos lógicos onde seja possível a construção da definição lógica de um bloco com um nível de abstração superior ao atual, formando assim um bloco que contenha um conjunto de chamadas a métodos e não apenas a invocação de um método.

Ao nível do editor gráfico, o seu design poderá ser melhorado, investindo na customização gráfica dos blocos com o objetivo de colocar a biblioteca mais próxima dos padrões atuais da interface gráfica existente.

7.3 Apreciação final e pessoal

Relativamente ao resultado do projeto em si, a minha apreciação é positiva devido ao facto dos produtos resultantes do projeto são executáveis.

Ao longo do projeto, surgiram vários desafios tecnológicos que levaram à melhoria do meu conhecimento sobre a tecnologia utilizada no desenvolvimento do projeto.

O facto de o projeto ter levado à integração de uma nova tecnologia num sistema já existente melhorou a minha capacidade de pesquisa e analise em questões profundamente tecnológicas aumentando também o meu rigor na análise das mesmas.

A elaboração do presente documento veio melhorar a minha capacidade de escrita de documentos relativos a projetos.

Para concluir, considero que o projeto teve um impacto positivo na minha vida académica, profissional e pessoal.

Referências

(ISO 9241-11, 1998) "ISO 9241-11:2018(en), Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts." [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en.

(Blockly, 2019) "Blockly | Google Developers." [Online]. Available: https://developers.google.com/blockly

(Saaty, 1990) T. L. Saaty and J. M. Katz, "How to make a decision: The Analytic Hierarchy Process," 1990.

(Woodall, 2003) "Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis." [Online]. Available: https://www.researchgate.net/publication/228576532_Conceptualising_'Value_for_the_Customer'_An_Attributional_Structural_and_Dispositional_Analysis.

(Scratch) "Scratch - Imagine, Program, Share." [Online]. Available: https://scratch.mit.edu/

(Weintrop, 2019) D. Weintrop, "Block-based programming in computer science education," *Commun. ACM*, vol. 62, no. 8, pp. 22–25, Jul. 2019.

(Critical Manufacturing, 2020) "Critical Manufacturing - Critical Manufacturing MES | Infrastructure Framework for Manufacturing Equipment Integration, Data Analysis and Business Intelligence." [Online]. Available: https://www.criticalmanufacturing.com/en/critical-manufacturing-mes/architecture

(Mernik, 2005) M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, 2005.

(Deursen, 2000) A. van Deursen, P. Klint, and J. Visser. Domain-Specific Languages: An Annotated Bibliography. ACM SIGPLAN Notices, 35(6):26–36, June 2000

(Cazzola, 2010) W. Cazzola and D. Poletti, *DSL Evolution through Composition*. 2010.