UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Aprimorando a Modularidade e a Manutenibilidade no Framework EUL por meio da Seleção Dinâmica de Componentes

Mariany Morais Silva Lima

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

Mariany Morais Silva Lima

Aprimorando a Modularidade e a Manutenibilidade no Framework EUL por meio da Seleção Dinâmica de Componentes

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestra em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Dilvan de Abreu Moreira

USP – São Carlos
Abril de 2024

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

M827a

Morais Silva Lima, Mariany
Aprimorando a Modularidade e a Manutenibilidade no Framework EUL por meio da Seleção Dinâmica de Componentes / Mariany Morais Silva Lima; orientador Dilvan de Abreu Moreira. -- São Carlos, 2024.
98 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Sistemas de Apoio a Decisão. 2. Web Semantica. 3. Interface Web. 4. Interfaces Dinâmicas. I. de Abreu Moreira, Dilvan, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Mariany Morais Silva Lima

Enhancing Modularity and Maintainability in EUL Framework through Dynamic Component Selection

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Dilvan de Abreu Moreira

USP – São Carlos
April 2024

Esse trabalho é dedicado a todas as crianças curiosas de escolas públicas brasileiras.
E a todas as pessoas que sonham com um futuro melhor através da educação.

AGRADECIMENTOS

De todos os agradecimentos que irei fazer, o maior vai para minha mãe, Altair Oliveira, e para meu pai, Joel Vilasio, por me apoiarem incondicionalmente e terem me criado para ser a mulher que sou hoje, antes de qualquer outra coisa. E para o meu marido e melhor amigo, Marcelo Henrique, por ter passado por todo o processo desse trabalho ao meu lado, ter sido minha luz nos momentos ruins e dividir os momentos de alegria.

Agradeço ao meu orientador, Dilvan Moreira, que sem a paciência e a crença este trabalho não teria sido completado. À professora Kamila Rios, que me ajudou a ver novos caminhos a serem seguidos; E as companheiras de laboratório e amigas queridas, Aline Verhalen e Helen Piccoli.

Gostaria de agradecer também a família que São Carlos me deu, Ana Júlia Nociti, Thamis Andrade, Bruno Reitmann, Greyce Michelino, Mariana Valério, Wanner Menezes e Caio Oliveira. Sem vocês eu não teria me divertido tanto e assim, esse trabalho não existiria.

Obrigado também a tantas outras pessoas que fizeram parte da minha vida durante os anos que se seguiram para o fim deste trabalho, mesmo que não pessoalmente nomeadas, vocês também fazem parte da colcha de recordações que vou sempre levar desses anos.

Este trabalho foi financiado em parte pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. E também pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

“O futuro é a escuridão - e isso é a melhor coisa que o futuro pode ser, creio eu.”

(Virginia Woolf)

RESUMO

MORAIS, M. **Aprimorando a Modularidade e a Manutenibilidade no Framework EUL por meio da Seleção Dinâmica de Componentes**. 2024. 98 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

Sistemas de Apoio à Decisão (SADs) facilitam a tomada de decisões em diversas áreas, encapsulando e formalizando o conhecimento específico do domínio. Tradicionalmente, a transferência desse conhecimento para os desenvolvedores de software tem sido um gargalo, retardando o desenvolvimento e a manutenção do sistema. O *End User Language* (EUL) *Framework* foi criado para melhorar essa questão, defendendo a segregação do conhecimento do domínio em uma ontologia e um *script* de Linguagem Específica do Domínio (DSL). Existem apenas alguns itens nos quais a interação entre desenvolvedores e especialistas deve ocorrer, simplificando o processo de desenvolvimento.

Para que isso aconteça, interfaces devem ser geradas automaticamente a partir da ontologia e do *script*. No início da concepção do EUL, um componente central orquestrava a seleção de vários elementos usados para criar as interfaces do SAD. Esse mecanismo dependia de um conjunto rígido de opções, limitando inadvertidamente a versatilidade e adaptabilidade dos sistemas produzidos, levando a um crescimento desordenado do código-fonte da interface do usuário.

Este trabalho propõe um processo de composição de Interface do Usuário (UI) mais flexível para enfrentar essas limitações. Esse processo não apenas deve considerar uma ampla variedade de variáveis para a seleção de componentes de UI, mas também permitir que os usuários introduzam novos componentes de UI sempre que necessário. Isso é alcançado por meio da introdução de um novo seletor dinâmico de componentes a serem usados em tempo de execução, empregando conjuntos de regras associadas aos componentes específicos. Cada conjunto de regras do componente dirá se ele pode ser usado para um tipo de dado específico. As regras são definidas em formato JSON, e consideradas legíveis por humanos. Esse novo processo permite que os componentes sejam compilados separadamente do sistema principal, melhorando a modularidade e a manutenibilidade do *framework*.

Um estudo de caso envolvendo desenvolvedores *web* foi realizado para avaliar se o *framework* proposto teve um desempenho superior na facilidade de adição de componentes. O estudo tinha como objetivo comparar o desempenho do novo framework em termos de tempo de desenvolvimento e compreensão do código em relação ao seu antecessor. Usando questionários, tarefas avaliadas na escala Likert e o método *Think Aloud* para *feedback* em tempo real, os desenvolvedores endossaram (4 em 5) o *framework* reformulado, confirmando as melhorias introduzidas por esta pesquisa

Palavras-chave: Sistemas de Apoio a Decisão, Web Semântica, Interface Web, Interfaces Dinâmicas.

ABSTRACT

MORAIS, M. **Enhancing Modularity and Maintainability in EUL Framework through Dynamic Component Selection**. 2024. 98 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2024.

Decision Support Systems (DSSs) facilitate decision-making across various domains by encapsulating and formalizing domain-specific expertise. Traditionally, the knowledge transfer of this expertise to software developers has been a bottleneck, slowing down system development and maintenance. The End User Language (EUL) Framework was conceived to ameliorate this issue, advocating the segregation of domain knowledge in an ontology and Domain-Specific Language (DSL) script. There are just a few artifacts where the interaction of developers and experts has to take place that streamlines the development process.

For that to happen, interfaces must be automatically generated from the ontology and script. Upon the initial inception of EUL, a central component orchestrated the selection of various elements employed to craft the DSS interfaces. This mechanism relied on a rigid set of options, inadvertently limiting the versatility and adaptability of the systems produced, leading to an unwieldy growth of front-end source code.

This work proposes a more flexible User Interface (UI) composition process to address these limitations. This process should not only consider a wider array of variables for UI component selection but also permit users to introduce new UI components whenever necessary. This is achieved by introducing a new dynamic component selector to choose components to use at runtime, using sets of rules associated with specific components. Each component ruleset will say whether it can be used for a particular datatype. The rules use human-readable JSON format. This new process allows components to be compiled separately from the core system, improving the framework's modularity and maintainability.

A case study involving web developers was undertaken to evaluate whether the proposed framework performed better. The study aimed to compare the performance of the new framework in terms of development time and code comprehension against its predecessor. Using questionnaires, tasks evaluated on the Likert scale, and the Think Aloud method for real-time feedback, the developers overwhelmingly endorsed (4 in 5) the revamped framework, confirming the improvements introduced by this research.

Keywords: Semantic Web, Decision support systems, Web interfaces, Dynamic Interfaces.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACM	*Association for Computing Machinery*
CRUD	*Create, Read, Update and Delete*
DSL	*Domain Specific Language*
DSS	*Decision Support Systems*
EUL	*End User Language*
GUI	*User Graphic Interface*
MorLib	*Morringhan Library*
OWL	*Web Ontology Language*
RDF	*Resource Description Framework*
SAD	Sistemas de Apoio à Decisão
SHACL	*Shapes Constraint Language*
SPA	*Single-Page Application*
UI	*User Interfaces*
UIML	User Interface Markup Language
W3C	*World Wide Web Consortium*

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

1.1 Contextualização e Motivação

Organizar e processar dados e informações para a geração de resultados de valor que apoiem a tomada de decisão em um domínio específico é o princípio dos Sistemas de Apoio à Decisão (SAD), ou *Decision Support Systems* (DSS) (TURBAN; ARONSON; LIANG, 2005).

Os conhecimentos desenvolvidos pelos especialistas de domínio são integrados nos DSSs, conhecimento esse, que não é familiar aos desenvolvedores de software. Assim, os desenvolvedores têm que usar diversas técnicas de levantamento de requisitos para entender o domínio dos especialistas e implementar o DSS corretamente (SCHIUMA; GAVRILOVA; ANDREEVA, 2012). Quanto mais específico for o domínio do especialista, maior será o esforço necessário para o desenvolvedor entendê-lo o suficiente para se certificar que o sistema cumpre seus propósitos.

Por outro lado, especialistas de domínio geralmente não têm conhecimento suficiente em matéria de desenvolvimento de software para desenvolver DSSs eles mesmos. Como exemplo desse problema, podemos apontar o caso dos especialistas em sustentabilidade da Embrapa Meio Ambiente. Uma parte importante do trabalho deles é a criação de DSSs voltados a problemas do meio ambiente. Esses especialistas modelam seus DSSs no papel e contratam empresas de desenvolvimento para torná-los realidade. Eles investem muito tempo treinando os programadores que farão o código do sistema. E, uma vez pronto, eles não têm como verificar a qualidade do código gerado, só sua funcionalidade. Quando necessitam de *upgrades*, enfrentam problemas já que, muitas vezes, têm que contratar uma empresa diferente para a tarefa e a qualidade do código do programa pode atrapalhar a tarefa.

1.2 Desenvolvimentos Anteriores e Desafios

A Embrapa iniciou uma cooperação com o nosso grupo de pesquisa para lidar com esse problema. Em conjunto desenvolveu-se o sistema SustenAgro, um método de avaliação de sustentabilidade nos sistemas produtivos da cana-de-açúcar do centro-sul do Brasil (CARDOSO *et al.*, 2013). O SustenAgro foi implementado como um DSS online para que a comunidade interessada pudesse realizar avaliações de sustentabilidade em cana-de-açúcar de maneira simplificada.

Para a criação do DSS SustenAgro, foi criado o *Framework Decisioner*. Ele usa um método de definição de conhecimento, baseado em uma ontologia,além de uma *Domain Specific Language* (DSL) (FOWLER, 2010) para fazer cálculos. Eles foram desenvolvidos para suportar conceitos de avaliação de sustentabilidade em agricultura. As ontologias no *Decisioner* representam a complexidade do conhecimento dos especialistas de um domínio, integram o conhecimento de diferentes fontes e flexibilizam os modelos perante mudanças. A interface do usuário final do sistema é gerada automaticamente a partir da ontologia e da DSL.

Entretanto, o *framework Decisioner* é específico para o SustenAgro, tornando necessária a formulação de um *framework* mais genérico para ser usado em DSSs de domínios diferentes. Para isso foi criado o *framework End User Language* (EUL), que implementa DSSs como uma *Single-Page Application* (SPA) usando o Vue, um *framework javascript*, que faz a aplicação *web* parecer mais como um sistema *desktop*.

Como o *Decisioner*, o EUL segrega a definição do DSS a basicamente dois artefatos, a ontologia e o *script* DSL, reduzindo a necessidade de desenvolvedores dedicados ao desenvolvimento e manutenção e atualização do sistema. O EUL é capaz de criar todas as interfaces gráficas e funcionalidade de um DSS a partir desses dois artefatos. Mas, ao se usar o EUL para a implementação de outros DSSs, percebeu-se que o conjunto predeterminado de componentes gráficos usados para a geração das interfaces de usuário, *User Interfaces* (UI), poderiam não ser o bastante para criar esses novos DSSs. Por isso a necessidade de uma refatoração foi evidenciada.

1.3 Questões de Pesquisa

As principais questões abordadas neste trabalho são:

- Como podemos criar novos componentes de interfaces de usuário e instanciá-los dinamicamente numa aplicação?

- Como fazer o EUL escolher quais componentes de UI são melhores para editar um tipo de dado?

1.4 Hipóteses

As principais hipóteses relacionadas a essa pesquisa são:

- É possível descrever os tipos de dados de uma ontologia que um componente de interface usa por meio de regras.
- É possível um *framework* interpretar essas regras e instanciar essas interfaces dinamicamente.

1.5 Objetivo

O objetivo deste trabalho foi mostrar que é possível a criação de regras que permitem a escolha dinâmica (em tempo de execução) do melhor componente disponível para mostrar e editar um determinado tipo de dado a partir da modelagem de dados de um DSS, usando ontologias. Essa funcionalidade visa facilitar o desenvolvimento de novas interfaces, diminuindo o tempo de aprendizado do código-fonte do EUL *framework*.

1.5.1 Objetivos específicos

Para alcançar o objetivo apresentado, foram definidos alguns objetivos específicos:

1. Criação de um conjunto básico, mas extensível, de componentes capazes de representar partes das interfaces;
2. Identificar quais variáveis são relevantes para a escolha de componentes para gerar interfaces de usuário;
3. Definir um modelo de regras para apoiar a decisão da escolha de componentes, baseando-se nas variáveis identificadas;
4. Melhorar o acoplamento de novos componentes ao sistema;
5. Conduzir estudos de caso com desenvolvedores *front-end*

1.6 Resultados

Uma nova versão do EUL implementando as ideias apresentadas, foi usada para instanciar uma nova versão do DSS SustenAgro. Os componentes gráficos dessa versão foram implementados usando o sistema de regras. Ela foi testada e tem um comportamento 100% igual a versão anterior (que usava componentes hard coded). Esta versão anterior já tinha sido testada com usuários finais em outro trabalho do grupo, *Enhancing Decision Support Systems*

development with the EUL Framework applied in the Agricultural Sustainability Assessment domain (MENEZES; MOREIRA, 2023).

Também foi realizado um estudo de caso com desenvolvedores que trouxe como resultado a aprovação, de 4 em 5 desenvolvedores, da nova estrutura. O estudo foi feito a partir de uma comparação de acoplamento de novo componente na estrutura que já existia no EUL *Framework* e na nova estrutura proposta neste trabalho.

1.7 Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma:

- Capítulo 2: Apresenta o referencial teórico.
- Capítulo 3: Discute trabalhos relacionados à pesquisa.
- Capítulo 4: Detalha o desenvolvimento da MorLib.
- Capítulo 5: Apresenta um estudo de caso com desenvolvedores *front-end*.
- Capítulo 6: Conclui e sugere trabalhos futuros.

CAPÍTULO

2

REFERENCIAL TEÓRICO

2.1 Considerações iniciais

Neste capítulo, delinea-se os fundamentos teóricos que sustentam este trabalho. Serão abordados conceitos-chave como *Decision Support Systems* (DSS); Web Semântica, ontologias e suas especificações, como *Resource Description Framework* (RDF), *Web Ontology Language* (OWL) e *Shapes Constraint Language* (SHACL), e *Domain Specific Language* (DSL). Esses conceitos serão essenciais para os capítulos subsequentes.

2.2 *Decision Support Systems* (DSS)

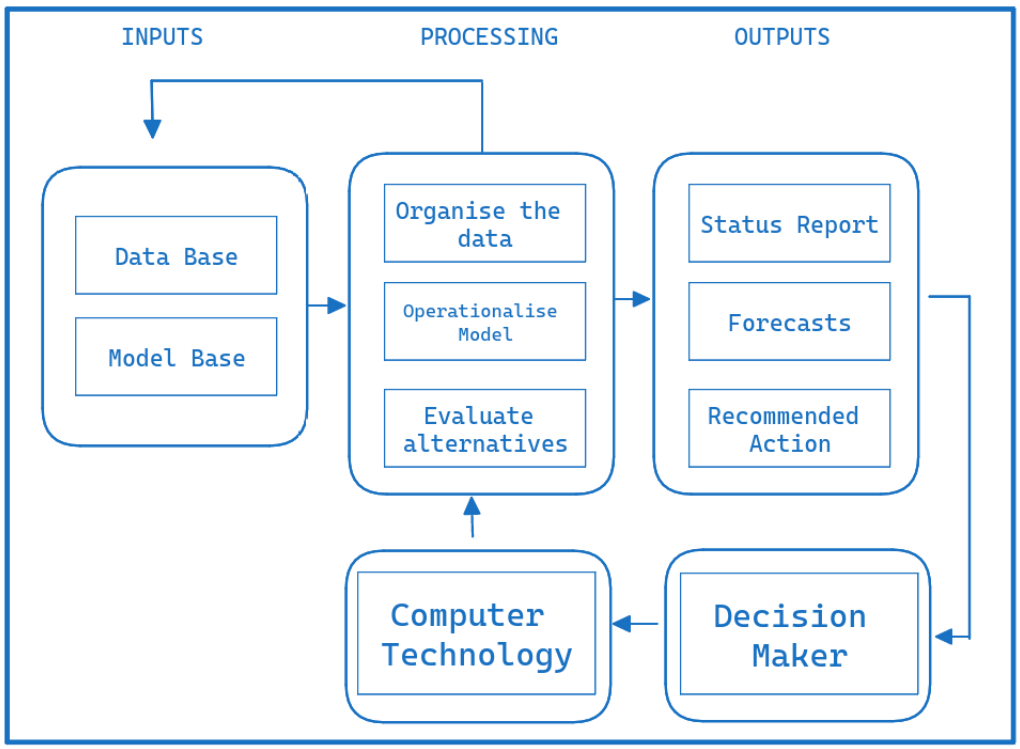
Os Sistemas de Apoio à Decisão, ou *Decision Support Systems* (DSS), são uma área de conhecimento ampla e em contínua evolução. Existem diversas definições para o termo e algumas delas serão apresentadas a seguir.

DSS são ferramentas projetadas para auxiliar na compreensão de processos complexos, facilitar a análise comparativa dos fenômenos envolvidos e apoiar a seleção de alternativas no processo decisório. A eficácia dos DSS deriva da combinação das competências e metodologias humanas com a capacidade computacional de acessar, modelar e avaliar dados em diferentes cenários (HEINZLE; GAUTHIER; FIALHO, 2017).

2.2.1 Arquitetura DSS

Arquitetura é como um projeto ou plano mestre para um sistema. Ela oferece uma abstração para gerenciar a complexidade do sistema e estabelecer um mecanismo de comunicação e coordenação entre seus componentes. Ela define uma solução estruturada para atender a todas as necessidades técnicas e operacionais (JAISWAL, 2019).

Figura 1 – Componentes de um DSS



Fonte: Adaptada de Tweedale, Phillips-Wren e Jain (2016).

A Figura 1 ilustra a arquitetura típica de um DSS, mostrando o fluxo pelo qual o DSS recebe dados, processa-os e, em seguida, retorna resultados para análise pelo tomador de decisão (TWEEDALE; PHILLIPS-WREN; JAIN, 2016).

Os principais componentes da Figura 1 são:

Entradas (*Inputs*) - Dados que alimentam o sistema, incluindo respostas dos usuários e modelos de conhecimento dos especialistas.

Processamento (*Processing*) - Composto pelos modelos e métodos de organização e processamento dos dados.

Saídas (*Outputs*) - Resultados obtidos a partir do processamento das entradas, permitindo a comparação das alternativas de decisão.

2.2.2 Taxonomia DSS

Power (2002) classifica os DSS em cinco categorias principais. Para essa classificação, ele usou a funcionalidade dominante em cada sistema. As categorias ficaram dispostas da seguinte maneira:

***Data-Driven* DSS** - Focado na organização e processamento de grandes volumes de dados estruturados.

***Model-Driven* DSS** - Centrado no gerenciamento de modelos que refletem aspectos da realidade, utilizando ferramentas estatísticas e análises.

***Knowledge-Driven* DSS** - Visa resolver problemas usando regras, fatos, procedimentos ou conhecimento específico do domínio.

***Document-Driven* DSS** - Especializado na gestão de documentos não estruturados.

***Communications-Driven* DSS** - Suporta comunicação, colaboração e tomada de decisão em grupos.

O tipo de DSS usado neste trabalho é o *Knowledge-Driven* DSS, caracterizado por conter conhecimento de um domínio em particular, e utilizar bases de conhecimento que geralmente possuem regras para suportar os conceitos, gerenciar e inferir conhecimentos. Os *Knowledge-Driven* DSS, abordados neste trabalho, são baseados em ontologias. Eles permitem representar, categorizar e relacionar o conhecimento dos especialistas, além de fazer inferências usando esse conhecimento.

2.3 Web Semântica

Em 2001, Berners-Lee *et al.* (2001) propuseram a Web Semântica, como forma de estruturar o conteúdo da web e permitir que o usuário possa realizar tarefas guiado por mecanismos chamados de agentes. Isso seria possível através da estruturação dos dados e da semântica presente em cada documento que esses agentes virtuais conseguissem "compreender". Outro ganho que a proposta traz é que as informações estariam interligadas e com fácil acesso, por serem dados interoperáveis. No mesmo texto de apresentação da Web Semântica, publicado na revista *Scientific American*, os autores relatam que a Web Semântica não é uma Web separada, mas uma extensão da atual, na qual a informação recebe um significado bem definido, permitindo que computadores e pessoas trabalhem em cooperação.

As ontologias exercem um papel fundamental na Web Semântica, pois são elas que permitem a relação entre termos e conceitos de um domínio.

2.3.1 Ontologias

No contexto da Web Semântica, define-se ontologia como um esquema de representação que possibilita conceitualizar e estruturar conhecimento, permitindo a interpretação por computadores, visando compartilhar conhecimento entre humanos e computadores (ALLEMANG; HENDLER, 2011).

Uma ontologia é uma estrutura conceitual e computacional que permite representar o conhecimento, de qualquer domínio, por meio de entidades, classificações, relações semânticas, regras e axiomas. Ela é especificada por meio dos seguintes componentes básicos:

Classes: Representam os conceitos de um domínio e organizam os indivíduos em uma estrutura lógica e hierárquica (NOY; MCGUINNESS *et al.*, 2001).

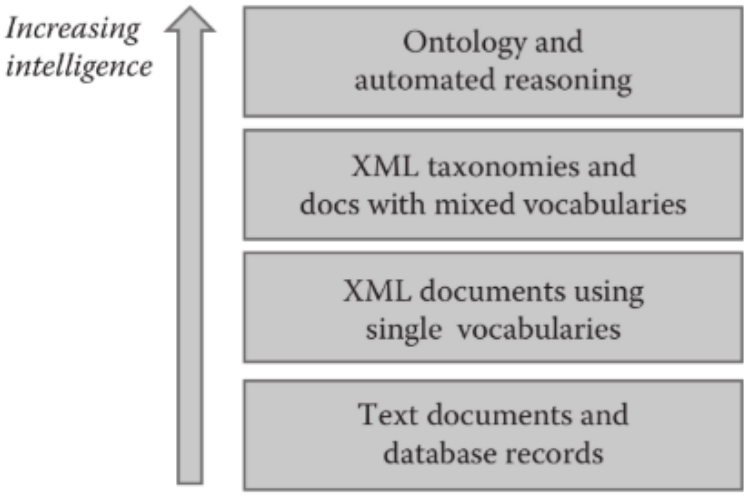
Relações: Simbolizam a interação entre as propriedades presentes nas classes ou indivíduos e os conceitos de um domínio. Elas podem ser transitivas, simétricas ou terem cardinalidade definida.

Axiomas: Estabelecem regras consideradas verdadeiras no domínio, associando relacionamentos entre os indivíduos e fornecendo características descritivas e lógicas para os conceitos.

Indivíduos (ou instâncias das classes): Representam elementos específicos ou dados que, juntamente com a definição de uma ontologia, compõem a base de conhecimento (NOY; MCGUINNESS *et al.*, 2001).

A representação de uma ontologia pode ser feita por lógica de predicados e lógica descritiva, usando padrões definidos pelo W3C, como *Resource Description Framework* (RDF) e *Web Ontology Language* (OWL)(PATEL-SCHNEIDER, 2005). A Figura 2 retrata os níveis de representação de dados na forma de conhecimento processável por máquinas (ALLEMANG; HENDLER, 2011).

Figura 2 – *Smart data continuum:* níveis de representação de dados na forma de conhecimento processável por máquinas.



Fonte: Allemang e Hendler (2011).

No nível mais baixo estão os dados com nenhum significado semântico, que dependem do contexto da aplicação. No segundo nível, são usadas definições de esquemas XML para conseguir independência dos dados da aplicação, ou seja, os dados fluem entre aplicações de um único domínio. No terceiro nível, os dados já podem ser combinados entre diferentes domínios, pois são suficientemente independentes para serem recuperados e combinados com outras fontes. E, finalmente, no quarto nível é possível inferir novos dados a partir dos existentes e compartilhá-los entre aplicações sem a interferência humana (SUGUMARAN; GULLA, 2011).

2.3.2 Resource Description Framework (RDF)

O *Resource Description Framework* (RDF) é uma família de especificações da W3C que foi originalmente projetado como um modelo de metadados e, também, chegou a ser usado como método de descrições conceituais, principalmente para descrever recursos *web*, além de, formalmente, ser um formato de dados de tipo grafo direcionado e rotulado para representar informações na *web*[1].

O objetivo do RDF é fornecer uma representação minimalista do conhecimento na Web (SHADBOLT; BERNERS-LEE; HALL, 2006). O RDF tem uma estrutura formada por triplas (Sujeito, Predicado, Objeto), como mostra a Figura 3.

Figura 3 – Exemplo de grafo RDF

Fonte: Neto *et al.* (2013).

Sujeito: são os recursos, e os mesmos identificados por URIs;

Predicado: cada predicado têm um significado específico e relaciona um sujeito com um objeto. Os mesmos são atributos, características, aspectos ou relações que descrevem o sujeito;

Objeto: outro recurso ou valor que se relaciona com o sujeito através do predicado;

Usando RDF, conseguimos explicitar relações entre dois objetos, mas não conseguimos fazer modelagens específicas nem inferência, para isso são necessárias ontologias descritas no padrão OWL. Ele permite descrever detalhadamente o que um objeto representa e suas relações com outros objetos.

2.3.3 Web Ontology Language (OWL)

A *Web Ontology Language* (OWL) é uma linguagem desenvolvida e endossada pela *World Wide Web Consortium* (W3C). Ela foi projetada especificamente para representar informações sobre categorias de objetos e como eles estão relacionados, tornando-a ideal para representar ontologias.

OWL estende o *Resource Description Framework* (RDF) e fornece um conjunto mais amplo de ferramentas para descrever propriedades, classes e relações. Ela permite representar conhecimento de forma mais detalhada, como especificar características de relações (transitividade, simetria, etc.) e restringir classes com propriedades mais complexas.

[1] https://www.w3.org/TR/rdf-sparql-query/

A OWL 2 é dividida em vários perfis, cada um otimizado para diferentes cenários de uso:

- **OWL 2 EL:** Um perfil destinado a ontologias com uma hierarquia de classes muito grande, mas com restrições expressivas limitadas.
- **OWL 2 QL:** Projetado para aplicações que necessitam de consultas eficientes em dados, mesmo com grandes volumes de instâncias.
- **OWL 2 RL:** Um perfil para aplicações que necessitam de raciocínio eficiente, mas podem operar com uma forma restrita de expressividade.
- **OWL 2 DL:** Mantém a completude computacional, oferecendo uma expressividade balanceada.
- **OWL 2 Full:** A forma mais expressiva da OWL 2, sem garantias de decidibilidade.

A OWL é amplamente utilizada em aplicações que necessitam de processamento de ontologias, inferência e busca de informações. Sua capacidade de representar relações complexas e de fazer inferências sobre essas relações a torna uma ferramenta poderosa para a modelagem de domínios complexos e para a Web Semântica.

2.3.4 Shapes Constraint Language (SHACL)

Apesar da utilidade de OWL em ontologias, ela não é apropriada para descrever e validar dados em RDF. O *Shapes Constraint Language* (SHACL) é uma linguagem, da W3C, para validação de grafos RDF a partir de um conjunto de restrições. SHACL permite especificar restrições em um grafo RDF, essas restrições também sendo um grafo RDF. Esses grafos são chamados de "*Shapes Graph*", e os que serão validados pelo SHACL, são chamados de *Data Graph*. Como os *Shape Graphs* são usados para validar se os *Data Graphs* atendem a um conjunto de condições, eles também podem ser vistos como uma descrição dos *Data Graphs* que atendem as condições. Essas descrições podem ser usadas para diversos propósitos, além da validação, incluindo interfaces de usuário, geração de códigos e integração de dados (KNUBLAUCH; KONTOKOSTAS, 2017).

O SHACL em si é dividido em duas especificações, SHACL *Core* e SHACL-SPARQL (HOGAN; HOGAN, 2020):

SHACL *Core* fornece um vocabulário para a definição das restrições de maneira abstrata (declarativa).

SHACL-SPARQL estende a linguagem para permitir o uso de consultas SPARQL, que servem para especificar restrições que não podem ser expressas em SHACL *Core*.

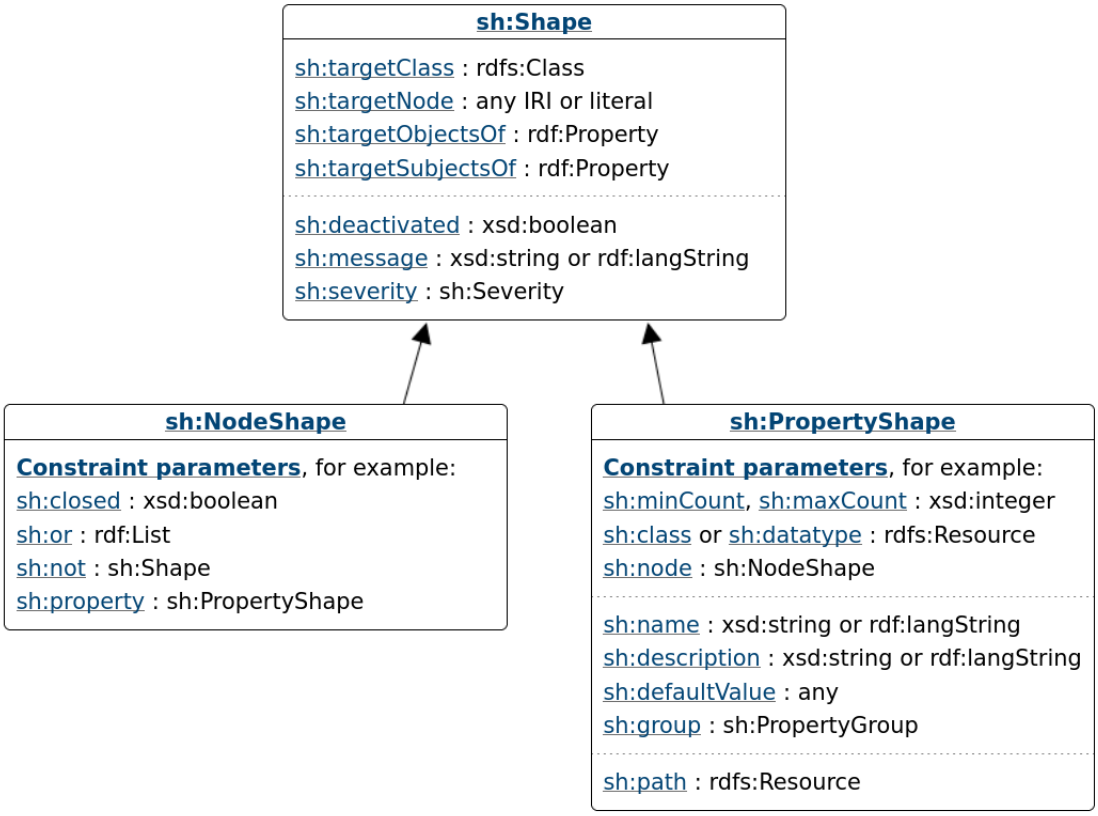


Figura 4 – Diagrama de classes. Fonte: W3C

Fonte: W3C

No geral, SHACL *Core* é informalmente denominado de SHACL. A Figura 4 traz o diagrama do panorama de algumas classes principais dele, de forma simples. Cada caixa representa uma classe e seu conteúdo lista algumas das possíveis propriedades que ela pode ter, além de seus tipos de valores. As setas indicam uma relação de *rdfs:subClassOf*.

Neste trabalho, usa-se o SHACL para a criação de *shape graphs* usados para a descrição de dados do usuário. Essa descrição é usada para a geração automática de interfaces do nosso *front-end*.

2.4 *Domain Specific Languages* (DSL)

As ontologias, particularmente aquelas definidas usando padrões como OWL e validadas por SHACL, são excelentes para representar e validar a estrutura e semântica dos dados. Elas fornecem uma estrutura rica para definir e relacionar conceitos em um domínio específico. No entanto, as ontologias, por sua natureza, são predominantemente declarativas. Elas descrevem "o quê" e "como as coisas são", mas não necessariamente "como fazer algo" ou o aspecto procedural.

Assim usamos as *Domain Specific Languages* (DSLs) para termos esse aspecto procedural. Uma DSL é definida como uma linguagem computacional utilizada em um domínio

particular, com o intuito de realizar tarefas específicas (FOWLER, 2010). DSLs são usualmente declarativas e com enfoque apenas em um domínio de problema em particular (PIERONI *et al.*, 2014).

Enquanto as ontologias definem e validam a estrutura dos dados, as DSLs são projetadas para expressar procedimentos, comportamentos e lógica de domínio específico. No contexto dos DSSs isso é fundamental, já que eles não requerem apenas dados bem definidos, mas, também, uma lógica procedural para analisar, processar e tomar decisões com base nesses dados. Assim, enquanto as ontologias podem servir como a base para a estruturação dos dados em um DSS, as DSLs complementam ao fornecer a capacidade de definir a lógica e os procedimentos específicos do domínio que operam sobre esses dados.

As vantagens de uma DSL são as abstrações específicas de um domínio que são pré-definidas e que representam, de forma direta, os conceitos do domínio da aplicação. Ela aumenta o nível da abstração, gerando códigos concisos e preparando o código para o reúso. Elas também geram documentação suficiente para proporcionar o entendimento do sistema pelo grupo de interesse, pelos especialistas de domínio e pelos desenvolvedores (PIERONI *et al.*, 2014).

As desvantagens ficam por conta da necessidade do especialista de dominio aprender como declarar essas DLS, e com isso poder validar e editar as mesmas. Os especialistas devem possuir maturidade nos problemas de domínio para que a desvantagem fique somente no passo de aprendizado da declaração.

2.5 Considerações Finais

Neste capítulo, foi mostrado o embasamento teórico necessário para o entendimento das técnicas e conceitos utilizados neste trabalho. Todos os conceitos e técnicas perpassam o trabalho como um todo, já que para o mesmo ser implementado todo o sistema teve q ser entendido e estudado. Para o desenvolvimento principal nesse projeto, RDF e SHACL são os principais usados ativamente, eles são os conceitos principais que o a nova biblioteca desenvolvida, usa como base para sua mecânica.

CAPÍTULO

3

TRABALHOS RELACIONADOS

3.1 Considerações Iniciais

A geração automática de interfaces de usuários por meio de *frameworks* já representa um campo especializado na literatura de computação. Quando introduzimos a complexidade adicional de dados originados de ontologias e a descrição de seu funcionamento por meio de Linguagens Específicas de Domínio (DSLs), estamos nos aventurando em um território ainda menos explorado. Dada essa especificidade, amplia-se a revisão bibliográfica visando abranger a geração automática de UIs de forma mais ampla.

Para relacionar trabalhos publicados e relevantes ao tema desta pesquisa, realizou-se uma busca bibliográfica em bases de Bibliotecas Digitais. Entre elas estão o IEEE *Xplore*[1], *Springer*[2], *Scopus*[3] e *Association for Computing Machinery* (ACM)[4]. A busca foi orientada por *strings* específicas, como:

- *("Automatic Generating") AND ("web UI") AND ("DSL")*
- *("model-driven") AND ("Automatic Generating") AND ("UI")*
- *("automatic choosing") AND ("UI")*
- *("owl") AND ("automatic") AND ("GUI") AND ("generator")*
- *("ontology based") AND ("user interface") AND ("automatic generator")*

As *strings* vão desde as mais especificas até as mais gerais, nas mais específicas poucos resultados foram trazidos, e nas mais gerais, chegaram a 16 mil resultados. Com isso a escolha

[1] https://ieeexplore.ieee.org/Xplore/home.jsp
[2] https://link.springer.com/
[3] https://www.scopus.com/home.uri
[4] https://dl.acm.org/

de referências foi dificultada, foi feita uma leitura de títulos e *abstract* em busca de artigos que podiam trazer as referências e com os que foram achados empregou-se a técnica de *snowballing*. Esta técnica se baseia exclusivamente nas relações de citação e referência e não depende de palavras-chave e não utiliza uma base de dados específica. Assim, ao aplicar *snowballing* aos resultados iniciais, é possível descobrir trabalhos relevantes que poderiam ser omitidos em buscas convencionais. Esta abordagem complementar garante uma revisão bibliográfica mais abrangente e robusta (SILVA, 2017). O *Research Rabbit*[5], foi usado para guardar e gerenciar as referências achadas pelo *snowballing*, com isso cerca de 300 *abstracts* foram lidos, e 70 artigos e livros foram separados e usados como referência.

3.2 Geração de UIs Orientado a Modelos

Nos últimos anos, pesquisas sobre a geração de UI, baseadas no conceito de desenvolvimento orientado a modelos produziram contribuições vindas de várias frentes, mas para utilização como trabalho relacionado foram descartadas as que traziam modelos neurais como base das suas contribuições, e foram selecionados os que traziam outros tipos de modelos, como o de tarefas. Estes podem ser divididos em quatro grupos:

3.2.1 User Interface Description Languages (UIDLs)

UIDLs focam em descrever interfaces de usuário de uma maneira que seja independente da tecnologia utilizada no sistema. Alguns trabalhos nesta área incluem o uso de JavaFX (FEDORTSOVA; BROWN, 2014), User Interface Markup Language (UIML) (ABRAMS *et al.*, 1999), UsiXML (LIMBOURG *et al.*, 2004) e XForms (W3C).

A ideia principal é modelar diálogos e formulários usando descrições que são independentes das tecnologias usadas para controles de entrada/saída e a relação entre elementos em uma UI. UIDLs[6] propõem uma nova maneira de construir interfaces, com potencial para substituir o HTML tradicional, projetado para apresentação de documentos e não como uma Interface Gráfica do Usuário, *User Graphic Interface* (GUI), para aplicações complexas.

Enquanto as UIDLs focam em descrições genéricas de interfaces, o EUL *Framework*, visa especificamente a geração automática de interfaces. Ele introduz um seletor dinâmico de componentes que escolhe os componentes a serem usados em tempo de execução, tornando-o mais adaptável e versátil do que as abordagens tradicionais.

[5] https://researchrabbitapp.com/
[6] https://www.uidl.net/doc/reference/pdf/UidlReferenceDocumentation.pdf

3.2.2 Abordagens baseadas em tarefas/conversação

Estas abordagens descrevem aplicações por fluxo de diálogo derivados de modelos de tarefa, por exemplo CAP3 (BERGH; LUYTEN; CONINX, 2011), MARIA (PATERNO; SANTORO; SPANO, 2009) e abordagens baseadas em conversas (POPP *et al.*, 2009). Essa abordagem se concentra em um modelo dos fluxos de diálogo. Para gerar o *frontend* de um aplicativo, as etapas do fluxo de diálogo são associadas a descrições de UI independentes de tecnologia. David Raneburger (RANEBURGER; KAINDL; POPP, 2015), trabalhou uma proposta de transformação de UI, também baseadas em modelo de tarefas, mas como o conceito de regras atreladas.

Em contraste com os primeiros trabalhos citados, o EUL *Framework* se concentra na geração automática de interfaces a partir de ontologias e *scripts* DSL, com um foco especial na seleção dinâmica de componentes de GUI, permitindo uma maior adaptabilidade. Já o trabalho do David Raneburger tem algo em comum com o trabalho descrito aqui, que é o uso de regras para criação/adaptação da GUI.

3.2.3 Abordagens centradas em dados

Estas abordagens podem ser encontradas nos sistemas JANUS (BALZERT *et al.*, 1996), Mecano (PUERTA *et al.*, 1994), MAML (RIEGER; KUCHEN, 2019) e Quid (MOLINA; MOLINA; MOLINA, 2019). Eles usam um modelo de domínio como ponto de partida para a derivação de UIs. O sistema JANUS foi projetado para fornecer interfaces do tipo *Create, Read, Update and Delete* (CRUD) para aplicativos que funcionam em um modelo de domínio persistente. Ele foi incluído como exemplo, mesmo não suportando muitas dinâmicas na sua UI. O projeto MAML usa DSLs como modelos para a criação de UIs dinâmicas em diferentes plataformas, o Quid também usa DLS mas nessa caso são UI especificas para *Web*.

Embora sistemas como JANUS, MAML e Quid usem modelos de domínio como ponto de partida, o EUL vai além, permitindo a introdução de novos componentes de UI conforme necessário e usando conjuntos de regras em formato JSON para determinar a seleção de componentes, que também proporciona uma manutenção mais especifica e centrada.

3.2.4 Abordagens baseadas em ontologias

Elas dependem normalmente de conceitos das abordagens já mencionadas somados ao uso de ontologias para representar as informações sobre UIs. Por exemplo, em analogia às abordagens UIDL, Liu, Chen e He (2005) propôs uma estrutura orientada a ontologias para descrever UIs baseadas em conceitos armazenados em uma base de conhecimento. Em analogia com abordagens baseadas em tarefas, Gaulke e Ziegler (2015) propuseram um modelo de domínio melhorado com dados relacionados a UI e associaram isso a um modelo de tarefa baseado em ontologias.

Quando comparadas com o EUL *Framework*, elas têm as mesmas limitações das outras abordagens cujo os conceitos elas combinam com o uso de ontologias.

3.3 Componentização

A abordagem centrada em dados é a que mais se aproxima com o trabalho aqui desenvolvido, seguida por aquelas que se baseiam em ontologias, já que a maioria dos nossos dados estarão no formato de ontologias e suas instâncias. Isso faz com que nosso sistema seja totalmente dependente dessas estruturas.

Um componente de software é caracterizado como uma unidade de composição que representa porções de software com requisitos, interfaces e dependências de contexto claramente definidos.(SZYPERSKI; GRUNTZ; MURER, 2002)

Neste trabalho, interfaces complexas são compostas pela combinação de componentes mais básicos. A escolha dos componentes mais apropriados é feita em função do tipo de dado que é esperado do usuário. O sistema tem regras para associar tipos de dados aos componentes de interface, por isso, o uso de ontologias para também representar interfaces é uma vantagem, pois mantém o sistema semanticamente conciso.

3.4 DSS SustenAgro

O DSS SustenAgro é um protótipo de DSS, desenvolvido e testado de forma colaborativa entre o grupo de pesquisa e a Embrapa, que trabalha com a avaliação de sustentabilidade em processos produtivos de cana-de-açúcar na região centro-sul do Brasil (JESUS *et al.*, 2019). Ele não é apenas um trabalho relacionado, mas, também, a base para o desenvolvimento do sistema apresentado neste trabalho e para outros sistemas de colegas do grupo de pesquisa.

O SustenAgro utiliza o *Framework Decisioner*, desenvolvido pelo mesmo grupo de pesquisa, que organiza e gerencia os componentes gerais do DSS (SUAREZ, 2017). Além do *Decisioner*, o DSS SustenAgro é composto por uma ontologia de domínio, uma DSL e elementos gráficos da UI. A ontologia do domínio de avaliação de sustentabilidade da produção de cana-de-açúcar na região centro-sul do Brasil é o principal componente do sistema (SUAREZ, 2017).

O *framework Decisioner* é específico para o SustenAgro, que foi projetado exclusivamente para o domínio da sustentabilidade na produção de cana-de-açúcar. Ele não poderia responder às perguntas de pesquisa deste trabalho, já que aqui propõe-se uma interface que seja gerada automaticamente para mais de um tipo de DSS e que permita um processo de composição de UI mais flexível, com mais variáveis para a seleção dinâmica de componentes de UI . Contudo o SustenAgro ainda é relevante, pois é a base para o novo EUL *Framework*.

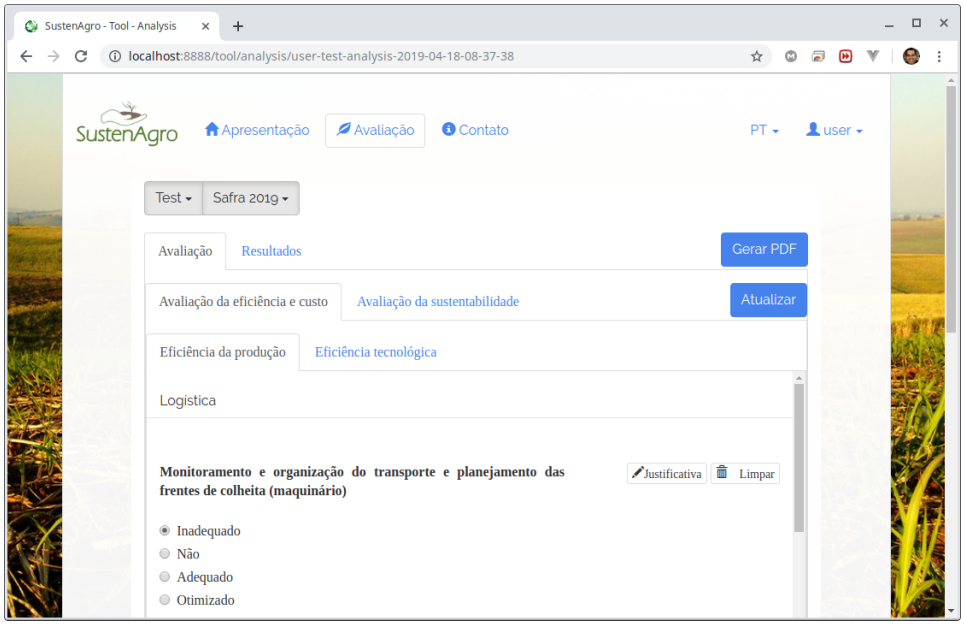


Figura 5 – *Framework Decisioner*

Fonte: Suarez (2017).

3.5 *End-User Language Framework*

O EUL é projetado para gerar automaticamente aplicações *web* para Sistemas de Apoio à Decisão (DSS) com base em descrições fornecidas por DSLs e ontologias, sendo que estas definem os objetos a serem analisados e os tipos de análises possíveis. A partir destas descrições, as interfaces são criadas. Já os *scripts* DSL determinam o comportamento da DSS.

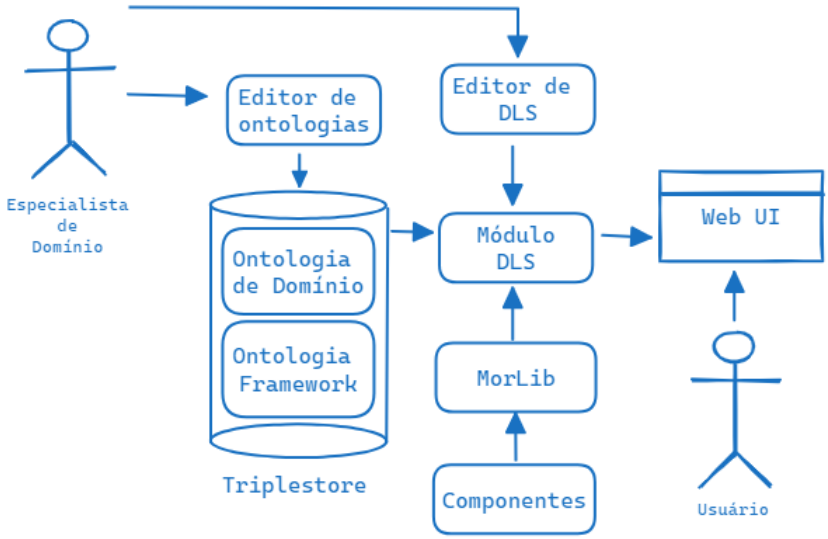


Figura 6 – Arquitetura EUL *framework*

Fonte: Elaborada pelo autor

A figura Figura 6 ilustra a arquitetura fundamental do EUL. Os componentes principais

são:

- **Ontologia Framework:** Define os conceitos essenciais que sustentam o processo de análise, mapeia os tipos de dados e fornece a base para a criação da UI.
- **Ontologia de Domínio:** Criada em colaboração com especialistas do domínio específico da DSS, estabelece conceitos fundamentais, como indicadores, indices e recomendações.
- **Editor de Ontologias:** Uma ferramenta que permite aos especialistas do domínio modificar a ontologia. As ontologias são escritas em YAML, uma linguagem de formatação mais simples que, por exemplo, OWL.
- **Editor DSL:** Um ambiente onde os especialistas podem editar os *scripts* DSL, que determinam o comportamento da DSS. Esses *scripts* fazem as análises nos dados que os usuários provem, para fornecer recomendações.
- **Módulo DSL:** O núcleo do sistema, que acessa as ontologias e os *scripts* para construir a UI. A partir do tipo de cada dado que é usado, um componente de UI é escolhido para lidar com ele e a interface é construída progressivamente.
- **MorLib - Componentes:** É onde ficam as regras e o componente dinâmico que será chamado parar gerar automaticamente a UI dos DSS.

As DSS criadas pelo EUL *Framework* podem ser divididas em duas partes:

- **Client side:** A interface com a qual os usuários interagem. Desenvolvida com VueJS[7], um *framework javascript*, e a biblioteca Vuetify[8] para um design alinhado ao *Material Design* da Google. O uso dessa biblioteca foi uma opção por uma especificação, o *Material Design*, já consolidada no mercado;
- **Server side:** Representa o *backend*, que gerencia a comunicação entre os *web services*, dados do usuário, ontologias e *scripts* DSL. Os dados são armazenados na *triplestore Blazegraph*[9], uma base de dados de grafos com *resoning*. As DSLs são escritas em *Groovy*[10], uma linguagem dinâmica similar ao Java com suporta ao desenvolvimento de DSLs. A Figura 7 mostra um trecho da DSL que está em uso no SustenAgro.

Os usuários alvo de DSS criadas com o EUL *framewok* são os **especialistas de domínio** e os **usuários finais**. Os primeiros usam as ontologias e DSLs para definir as perguntas que serão feitas aos usuários finais, os métodos de avaliação e o formato dos relatórios. Os **usuários finais**

[7] https://vuejs.org/
[8] https://vuetifyjs.com/en/
[9] https://blazegraph.com/
[10] https://groovy-lang.org/

```
// Para cada índice, é possível indicar fórmulas para o cálculo de cada
// atributo. Essas fórmulas podem ser tão complicadas como você queira.
report {

    def weighted = [fromInstance: ui.hasWeight, fromClass: sa.relevance]
    def environment = sum(sa.EnvironmentalIndicator, weighted)
    def economic = sum(sa.EconomicIndicator, weighted)
    def social = sum(sa.SocialIndicator, weighted)
    def sustainability = (environment + social + economic) / 3
    def costProductionEfficiency = sum(sa.ProductionEfficiency)
    def techEfficiencyInTheField = 0.8 * sum(sa.TechnologicalEfficiencyInTheField, weighted)
    def techEfficiencyInTheIndustrial = 0.2 * sum(sa.TechnologicalEfficiencyInTheIndustrial, weighted)
    def efficiency = abs(costProductionEfficiency) * (techEfficiencyInTheField + techEfficiencyInTheIndustrial)

    save('efficiency', efficiency)
    save('sustainability', sustainability)

    cmp 'show-report', [instance: instance()]
    cmp 'show-object', [
            class: 'show-report-tab',
            object: instance(ui.hasObject),
            description: [geo.municipality,
```

Figura 7 – Trecho de DSL

Fonte: Elaborada pelo autor

fornecem dados e recebem informações para apoiar suas decisões. Além disso, os desenvolvedores desempenham um papel crucial, auxiliando os especialistas com a ontologia e personalizando a DSS conforme necessário.

3.6 Considerações Finais

O capítulo apresenta os trabalhos relacionados ao EUL *Framework*, no geral, e também ao desenvolvimento que será descrito neste trabalho, da *Morringhan Library* (MorLib), que é o cerne da nova forma de criação automática das UIs no EUL. O que este trabalho mais se diferencia dos outros é que aqui estamos propondo uma criação de interface a partir de *widgets* que são agregadas até a UI estar completa. Assim, vamos criar a interface completa para uma DSS olhando somente para os dados que as descrições de SHACL, principalmente, fornecem para o sistema.

O objetivo principal desta pesquisa é reduzir a carga de trabalho dos desenvolvedores na criação de DSSs e proporcionar maior autonomia aos especialistas de domínio para fazerem pequenas alterações. Também é um esforço para aprimorar o EUL *Framework*, adicionando o suporte à criação dinâmica de UI. Além do desenvolvimento do EUL *Framework*, houve a re-implementação do DSS SustenAgro assim como a implementação de outro DSS no domínio da medicina.

No próximo capítulo, discutem-se as modificações implementadas no EUL para alcançar esses objetivos.

CAPÍTULO

4

MORRINGHAN LIBRARY

4.1 Considerações Iniciais

A customização de sistemas para novos domínios frequentemente enfrenta desafios, especialmente quando se trata de adicionar novos componentes a um *framework*. Identificou-se que a adição e edição de componentes para a interface do usuário (UI) era uma área que necessitava de melhorias.

Anteriormente no EUL, o componente básico da geração de interface era o *EUlElement.vue*, onde são escolhidos os componentes da UI com base nos tipos dos dados. Identificou-se a necessidade de uma nova versão do *EUlElement.vue* que pudesse fazer essa tarefa de forma dinâmica em tempo de execução.

A meta foi desacoplar os componentes da *build* principal, de forma que quando fossem requisitados os componentes seriam chamados e inseridos de forma dinâmica. Não há necessidade do *framework* principal sofrer mudanças ou ser compilado novamente para alterações e manutenção de componentes.

A seguir explicamos a substituição do *EUlElement.vue*, um componente estático, pela *Morringhan Library*, MorLib, novo componente dinâmico, que usa regras para carregar componentes em tempo real.

4.2 Sistema de Regras

O componente *EUlElement.vue* usava uma estrutura fixa de *if-else*, como exemplifica a Figura 8, para escolha de componentes. Essa estrutura de *if-else* foi substituída por um sistema de regras baseado em JSON. Essas regras são criadas a partir das definições dos próprios componentes, assim o sistema pergunta ao componente, através das regras deste, se ele pode trabalhar com determinado tipo de dados, a reposta retornada a essa pergunta é o nome do

componente em que a regra é cumprida.

```
<EulDataType
    v-else
    v-model="content"
    :element="relation"
    :edit="edit"
    :title="comment | localize">
    <template #default="{value, onInput, required}">
        <EulSHA256Input
            v-if="isSHA256Field"
            :value="value"
            :label="label"
            :required="required"
            :edit="edit"
            @input="payload => onInput(payload)" />
```

Figura 8 – Trecho de código do*EULElement* antigo, para o código na integra Apêndice A, Figura 27

Fonte: Elaborada pelo Autor

Através da ontologia, é possível ter uma descrição detalhada de cada tipo de dado, como essa ontologia é fornecida para a aplicação, os criadores dela têm o controle tanto dos tipos de dados usados como dos componentes que eles vão precisar. O EUL também provê um conjunto variado de componentes básicos prontos para reúso, reduzindo a necessidade a criação de novos componentes. Para estabelecer essas regras, identificou-se quais variáveis eram essenciais na seleção de componentes, variáveis como *type*, *range* e *value* foram escolhidas.

O JSON *Rules Engine*[1], uma biblioteca que interpreta regras descritas em JSON, foi adotada, essa escolha foi influenciada pelo fato de que, no EUL, os dados do sistema já são representados usando o JSON-LD[2], uma serialização de RDF. O JSON *Rules* suporta operações booleanas *ALL* ou *ANY*, incluindo operações recursivas e aninhadas, além de ser possível usar, múltiplos parâmetros de hierarquização para as regras que escolhem os componentes.

A Figura 9 mostra um exemplo de regra sendo definida. Esse exemplo é o que está no repositório[3] da biblioteca, é uma regra bem complicada com vários fatos descritos. No nosso caso, exemplificado na Figura 10, as regras ficaram mais simples e compactas no momento, já que ainda temos poucos componentes implementados. Na Figura 10 então, vemos que o fato que vai, ou não, ser validado é a propriedade '*type*', ter que ser igual ao '*value*' '*ui:sh256*', se

[1] https://www.npmjs.com/package/json-rules-engine
[2] https://json-ld.org/
[3] https://github.com/cachecontrol/json-rules-engine#basic-example

esse fato for validado a regra irá devolver o nome do componente que lida com esse tipo de dado vinculado, que é o '*params*' o disparado pelo '*event*'.

```
let engine = new Engine()

// define a rule for detecting the player has exceeded foul limits.  Foul out any player who:
// (has committed 5 fouls AND game is 40 minutes) OR (has committed 6 fouls AND game is 48 minutes)
engine.addRule({
  conditions: {
    any: [{
      all: [{
        fact: 'gameDuration',
        operator: 'equal',
        value: 40
      }, {
        fact: 'personalFoulCount',
        operator: 'greaterThanInclusive',
        value: 5
      }]
    }, {
      all: [{
        fact: 'gameDuration',
        operator: 'equal',
        value: 48
      }, {
        fact: 'personalFoulCount',
        operator: 'greaterThanInclusive',
        value: 6
      }]
    }]
  },
  event: {  // define the event to fire when the conditions evaluate truthy
    type: 'fouledOut',
    params: {
      message: 'Player has fouled out!'
    }
  }
})
```

Figura 9 – Exemplo de regra

Fonte: *json-rules-engine* no Repositório *Git*

O '*event*' retorna o nome do componente porque, como mostrado na Figura 11, estes são carregados dinamicamente de arquivos *javascript* com o nome do componente, o que acontece em tempo de execução. O EUL só sabe qual componente será montado no momento que ele é escolhido. Como são as regras que fazem a escolha, diminuiu-se drasticamente o tamanho do código, já que não precisamos mais de uma estrutura que implementa todos os casos possíveis.

Toda a estrutura dentro do *EULElement*.vue que estava vinculada à escolha de componentes para cada *DataTypes*, mostrada na Figura 8, foi substituída por um único componente dinâmico, identificado na Figura 11. As várias funções de validação foram substituídas por uma única função (*WhoIm*), que é mostrada na Figura 11b.

```
{
    "conditions": {
        "all": [{
            "fact": "type",
            "operator": "equal",
            "value": "ui:sha256"
        }]
    },
    "event": {
        "type": "sha256-is-acceptable",
        "params": {
            "value": "EulSHA256Input"
        }
    },
    "name": "SHA256_rule"
}
```

Figura 10 – Exemplo de regra implementada - componente SHA256

Fonte: Elaborada pelo autor

```
<component
    :is="whoIm"
    v-bind="{
        type: range,
        value: value,
        label: label,
        required: required,
        edit: edit,
        uri: range === 'xsd:anyURI'
    }"
    v-on="{
        input: payload => onInput(payload)
    }" />
```

(a) Template

```
async whoIm() {
    const engine = require('./engine.js')
    const name = await engine.default({type: this.range})
    return require('./components/'+ name)
}
```

(b) Função de escolha

Figura 11 – Componente Dinâmico

Fonte: Elaborada pelo autor

4.3 Componentes dinâmicos

Agora que este sistema de chamada dinâmica dos componentes foi implementado, foi possível retirar de dentro do EUL os componentes específicos de cada aplicação. E criar um componente dinâmico para ser o controlador intermediário entre o EUL e os componentes.

O componente *EULElement* foi totalmente refatorado para ser o componente que chama o componente dinâmico e faz a adição dos componentes retornados, e em tempo de execução constrói a UI. Com isso os componentes para *DataTypes* ou *ObjectTypes* também puderam ser carregados dinamicamente. Todos os componentes que são 'folhas' da montagem da UI estão fora do EUL e são manipulados então, diretamente pela MorLib.

```
<div v-else-if="isComponentComplex && !isBlank">
    <component
        :is="getComponent"
        v-bind="componentBind"
        v-model="content" />
</div>
```

(a) Complexos/Objetos

```
<EulDataType
    v-else
    v-model="content"
    :element="relation"
    :edit="edit"
    :title="comment | localize">
    <template #default="{value, onInput, required}">
        <component
            :is="getComponent"
            v-bind="{
                'type': range,
                'value': value,
                'label': label,
                'required': required,
                'edit': edit,
                'uri': range === 'xsd:anyURI'
            }"
            v-on="{
                input: payload => onInput(payload)
            }" />
    </template>
</EulDataType>
```

(b) DataType

Figura 12 – Novo *template* - MorLib

Fonte: Elaborada pelo autor

O método anterior para chamada dinâmica do componente, *WhoIm*, que foi mostrado na Figura 11b, agora é o método *getComponent* dentro da MorLib, a Figura 12 ilustra. O método em si não sofreu alterações na sua lógica, mas outros métodos de apoio a ele foram acrescentados para a inclusão do *ObjectType* e da *Feature* na escolha a partir de regras.

4.3.1 Interfaces

Vamos descrever alguns dos componentes de interface da *Morringhan Library*. É importante destacar que existem, conceitualmente, três tipos de interfaces;

1. **Bases**: Representam os *SimpleType*s do RDF, abrangendo os *DataTypes*, como *literal*, *numerical*, entre outros.

2. **ObjectTypes**: Representam os componentes para *ObjectTypes* (do RDF), como *checkbox* e seletor.

3. **Features**: Representam os *ComplexTypes*, tipos específicos criados dentro das ontologias de domínios. Estes são componentes personalizados para os domínios e são feitos por demenada.

Na Figura 13, mostra-se como o *vueJS* renderiza uma interface, nesse caso, do Cadastro de Usina, apresentado na Figura 14. A Figura 13 dá uma ideia melhor de como todas essas

chamadas de componentes funcionam na prática. A componentização da geração da interface faz com que essa grande hierarquia fique menos trabalhosa de ser atualizada, já que as partes estão em lugares separados e bem definidos.

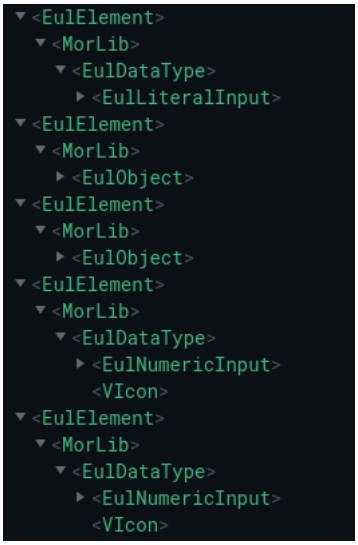


Figura 13 – Hierarquia de chamadas para a construção do formulário de cadastro

Fonte: Elaborada pelo autor

Figura 14 – Formulário de cadastro de Usina

Fonte: Elaborada pelo autor

4.3.2 Componentes Base

O sistema utiliza ontologias como base de dados, descrevendo esses dados para RDF e SHACL. Isso torna a tipagem em *DataTypes* intrínseca. Como temos um sistema que usa ontologias como base de dados e armazenam esses dados em RDF e SHACL, foram escolhidos os tipos *buit-in* do RDF para os componentes bases, tanto os considerados primitivos quanto os mais específicos. Na Figura 31[4]observa-se diagrama do esquema de tipos. Mas além daqueles definidos no próprio RDF, temos tipos que foram criados para a ontologia interna do EUL *framework*, como *code* e senha.

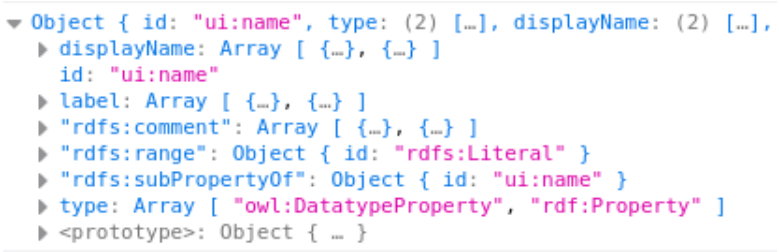


Figura 15 – JSON-LD: gera o componente de literal

Fonte: Elaborada pelo autor

Os componentes e suas regras seguem o mesmo padrão para todos, na Figura 15 temos o JSON-LD que normalmente é gerado no sistema, esse JSON está descrevendo o componente literal, no caso desse componente sua regra irá olhar para sua propriedade *range*, a imagem mostra que o valor vinculado a ela é '*rdfs:Literal*', então quando forem chamadas as regras, a regra que tem esse valor será a única que vai ser validada e com isso a MorLib vai receber o o nome do componente para ser chamado. As outras regras e exemplos de JSON-LD podem ser vistos no Apêndice A. Estes são os componentes existentes:

Literais - Componentes (*EulLiteralInput.vue*) associados a literais RDF. Literais são o tipo básico de dados (DataTypes) em RDF. Eles são strings que podem representar todos os outros tipos de dados. Normalmente este componente trabalha com strings, mas se um tipo de dados, que não tiver um componente de UI próprio, for chamado, ele que será usado. Por exemplo, o tipo primitivo RDF *anyURI* usa este componente, como mostra a Figura 29b, no Apêndice A, assim, pensando que o tipo de resposta mais base num formulário seria entrar com um texto corrido. Na Figura 14 seria o campo de 'Nome';

Numerais - Componente associado aos tipos primitivos numéricos do RDF *decimal*. Sua regra aceita os tipos: *xsd:integer*, *xsd:nonNegativeInteger*, *xsd:nonPositiveInteger*, *xsd:decimal*. Na Figura 14 é o campo de 'Ano da safra';

Datas - Componente associado a vários tipos primitivos do RDF relacionados a datas como, *xsd:date*, *xsd:datetime* etc. Na Figura 14, este componente mostra as datas pedidas para

[4] https://www.w3.org/TR/xmlschema11-2/

o usuário. No caso da regra vinculada, ela abrange todos os tipos RDF para datas que o componentes pode mostrar/editar. A Figura 28, no Apêndice A, mostra que o *range* que está sendo visto é *xsd:sYearMonth*;

Aniversário - Um componente que não está vinculado a um tipo RDF e sim um tipo da ontologia interna do EUL. A sua regra está sendo mostrada na Figura 30b, no Apêndice A.

4.3.3 ObjectType

Além de componentes para *DataTypes*, temos os componentes criados para O*bjectTypes*. O mais importante é o *EULObject.vue* que está associado ao tipo (ou classe) *owl:Thing* (do qual todas as outras derivam em OWL). Similar ao componente para literais (*EulLiteralInput.vue*), ele é usado para todos os objetos (ou instâncias) cuja classe não tem um componente associado.

Este componente tem várias maneiras de mostrar/editar objetos, que variam com a quantidade de objetos existentes e a operação realizada, fazendo a seleção da interface mais específica.

Na Figura 16, temos exemplos de duas interfaces que podem ser geradas, uma interface de *check-box*, na questão 'Sistema de produção agrícola' e a interface de *autocomplete*, 'Município'. Existe só uma opção para 'Sistemas de produção agrícola' (com a possibilidade de inclusão de outras futuramente) daí a escolha do componente da interface *check-box*. Mas existem mais de três mil municípios que podem ser escolhidos, sendo que, nesse caso a escolha pela interface *autocomplete* é mais correta por se adequar à uma escolha com grande quantidade de opções. A regra vinculada a esse componente é mostrada na Figura 32,Apêndice A.

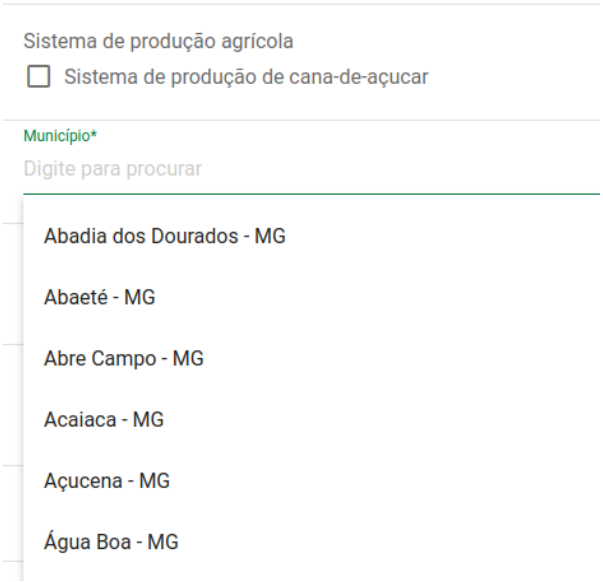


Figura 16 – Foco nos dois componentes de objeto

Fonte: Elaborada pelo autor

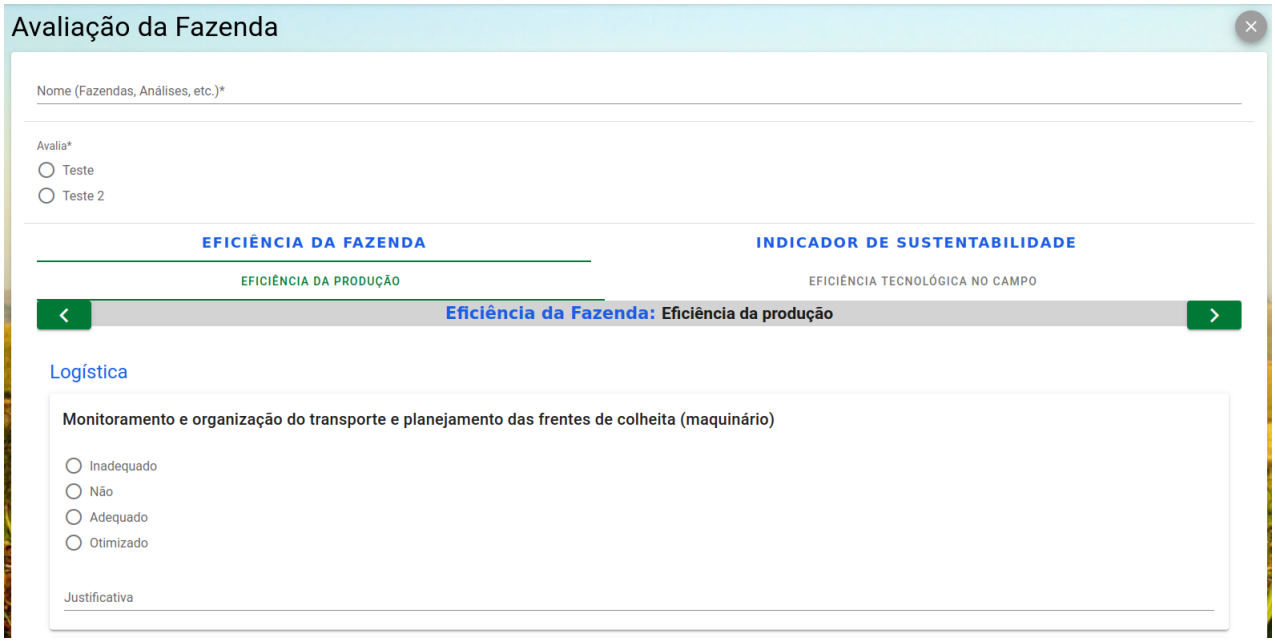


Figura 17 – Avaliação de Fazenda

Fonte: Elaborada pelo autor

4.3.4 Componentes Complexos

Para os *ComplexTypes* existe apenas um exemplo atualmente, que é a *Feature* (EulFeature.vue) criada especificamente para o DSS Sustenagro. A Figura 17 mostra a avaliação de sustentabilidade feita em usinas/fazendas de cana-de-açúcar. Como a *Feature* é um componente complexo, ela não gera somente um elemento da UI, ela gera toda uma estrutura, que pode ser vista na figura (tudo que vem depois do *label select* 'Avalia').

```
{
    "conditions": {
        "all": [{
            "fact": "type",
            "operator": "contains",
            "value": "ui:Feature"
        }]
    },
    "event": {
        "type": "feature-is-acceptable",
        "params": {
            "value": "EulFeature"
        }
    },
    "name": "feature_rule"
}
```

Figura 18 – Regra: *Feature*

Fonte: Elaborada pelo autor

A regra que é associada ao componente, Figura 18 usa o termo '*contains*', diferente das regras anteriores que usavam o '*equal*'. O termo '*contains*' é usado porque a regra recebe uma

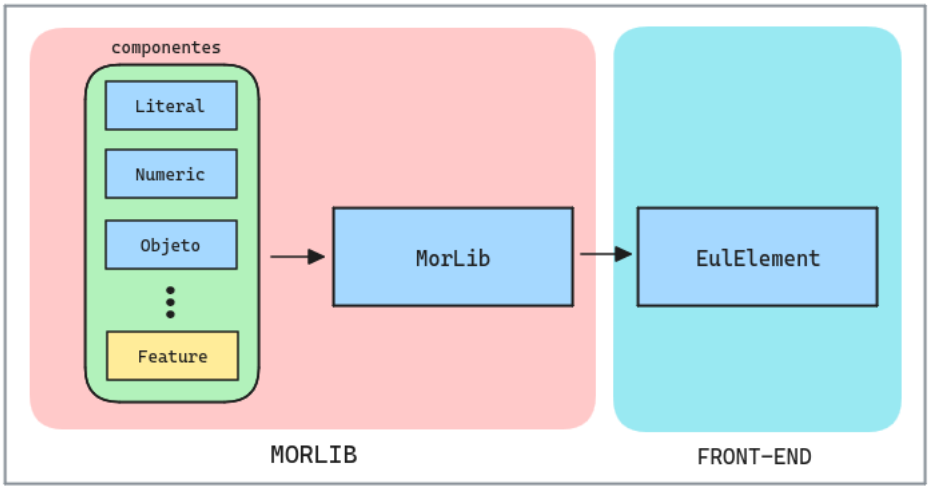


Figura 19 – Comunicação entre MorLib e *EulELement*

Fonte: Elaborada pelo autor

lista de *subClassOf* que podem conter o '*Feature*' *type*. Ela é um componente recursivo e, por isso, é necessário que sua regra se adapte a esse fato, usando assim uma lista e não somente uma variável mas, é interessante notar que essa é a única mudança, já que o próprio motor por trás das regras é que faz a iteração com os valores da lista.

4.3.5 Comunicação EULElement e MorLib

A instanciação dos componentes da biblioteca com o *core* do EUL é feita da mesma forma que um componente simples. O componente MorLib é usado e faz toda a parte de instanciação dos componentes dinâmicos. Por conta dessa divisão entre o *Element* e a MorLib, toda a biblioteca pode ser compilada de forma independente do *core* do EUL. Então a edição de componentes dentro da MorLib não altera os sistemas em produção, somente quando a atualização da biblioteca é implementada.

4.4 Considerações Finais

Ao longo deste capítulo, detalhamos o desenvolvimento e a estruturação da nova biblioteca, MorLib, destacando sua capacidade de selecionar componentes compatíveis com os dados, o que representa uma contribuição significativa deste trabalho.

A introdução de regras para a seleção de componentes não apenas simplifica o sistema, mas também oferece uma flexibilidade na customização das DSS instanciadas pelo EUL *Framework*. Esta abordagem orientada por regras facilita a tarefa dos desenvolvedores, permitindo que eles interajam e modifiquem o sistema de maneira mais modular.

No capítulo seguinte, apresentaremos um estudo de caso realizado com desenvolvedores. O objetivo deste estudo é validar se a nova estrutura é, de fato, mais compreensível e fácil de

usar em comparação com a anterior.

CAPÍTULO

5

ESTUDO DE CASO CONDUZIDO PARA AVALIAÇÃO DE APRENDIZAGEM E MANUTENÇÃO DO SISTEMA

5.1 Considerações Iniciais

Neste capitulo, abordamos um estudo de caso conduzido para avaliar a percepção, por parte de desenvolvedores, das mudanças feitas na arquitetura interna do *front-end* do *framework* EUL. O estudo foi executado em três fases, com atividades práticas e questionários. Nas duas primeiras fases, usuários realizaram tarefas de manutenção no sistema diretamente em código bruto, e depois responderam ao questionário. Na primeira fase, usuários implementaram a manutenção no código *front-end* anterior, e na segunda fase a manutenção foi implementada na nova proposta de *front-end* feita por este trabalho. Aqui é importante mencionar que, na fase 1, os desenvolvedores tiveram alguma experiência com o sistema antigo e, como ele não é totalmente diferente do novo, isso pode ter facilitado seu trabalho na fase 2. Esse é um possível viés desta análise.

Este estudo mostrou que, da perspectiva destes desenvolvedores, a nova abordagem para a arquitetura do sistema foi aprovada com maioria, o que se demonstra, também, a partir da análise de tempo de desenvolvimento.

5.2 Metodologia e Participantes

O estudo foi conduzido com a participação de desenvolvedores voluntários do laboratório Intermídia do ICMC, membros e convidados da comunidade interna. Os testes foram conduzidos presencialmente no laboratório Intermídia, a escolha dos participantes foi feita por conveniência (ETIKAN *et al.*, 2016), além de utilizar-se como critério serem desenvolvedores web com algum

conhecimento de *frameworks javascript*. Todos os participantes foram orientados a revisar e assinar o Termo de Consentimento, conforme apresentado no Apêndice B.

A quantidade de participantes foi limitada pela possibilidade de participar do estudo presencialmente, limitando-se a cinco indivíduos. Cada sessão de avaliação durou aproximadamente entre 1 hora e 1 hora e 10 minutos.

Após passarem pela seleção, o conceito de *Think Aloud* foi explicado aos voluntários. Essa técnica (SOMEREN; BARNARD; SANDBERG, 1994) permite capturar as impressões e percepções dos desenvolvedores em tempo real. Neste estudo, ela foi usada para fazer o levantamento da percepção de facilidade/aprendizado dos desenvolvedores a partir dos comentários feitos.

Além disso, utilizou-se a escala Likert (LIKERT, 1932) nos questionários para obter respostas quantitativas dos participantes. Ela é amplamente utilizada em questionários por permitir que os participantes quantifiquem suas respostas, ao contrário de perguntas com respostas fixas ou abertas (MALIK *et al.*, 2009). Neste estudo foi usada a escala de 1-5.

5.3 Procedimentos e Resultados

Conforme mencionado, o estudo foi segmentado em três fases e sete seções. Inicialmente, foram aplicadas questões de nivelamento para avaliar o grau de familiaridade dos desenvolvedores com conceitos-chave, como JSON, Ontologias, entre outros. A Tabela 1 apresenta uma síntese das respostas. Para facilitar, as questões foram numeradas a partir de Q1 e os desenvolvedores serão referidos em ordem de participação, D1, D2, etc.

Questões de nivelamento						
Questões		Desenvolvedores				
		D1	D2	D3	D4	D5
Q1	Você tem familiaridade com desenvolvimento web?	sim	sim	sim	sim	sim
Q2	Você tem familiaridade com *Framework Js*?	sim	sim	sim	sim	sim
Q3	Você já trabalhou com *Frameworks Js*?	sim	sim	sim	sim	sim
Q4	Você já trabalhou com *Vue Js*?	não	sim	não	sim	não
Q5	Você conhece o conceito de ontologias?	não	não	não	não	não
Q6	Você conhece o paradigma de ontologias para web semântica?	não	não	não	não	não
Q7	Você conhece RDF?	não	não	não	não	não
Q8	Você conhece JSON?	sim	sim	sim	sim	sim

Tabela 1 – Compilado de respostas dos desenvolvedores

A análise das respostas revelou que, embora a maioria dos desenvolvedores estivesse familiarizada com conceitos básicos de desenvolvimento web, muitos não tinham experiência com conceitos mais avançados que estão ligados a este trabalho, como ontologias e RDF. Esta informação foi crucial para calibrar o nível de detalhamento das instruções fornecidas.

O estudo foi, então, estruturado em três fases distintas: a primeira focou na interação com o código do sistema original, a segunda na interação com o novo código reformulado, e a terceira concentrou-se em questões comparativas entre as duas versões. A seguir analisa-se as fases 1, 2 e 3 separadamente.

5.3.1 Fase 1 - Avaliação do EulElement

Nesta fase, os desenvolvedores foram desafiados a realizar tarefas específicas no código e, posteriormente, avaliá-las usando uma escala Likert de 5 pontos, que variava de Muito Difícil, a Muito fácil. Para esse estudo foi escolhida a tarefa de inserção de um novo componente, já pronto, no fluxo do sistema. A decisão de focar apenas na integração, e não na criação do componente, foi tomada para avaliar especificamente a interação dos desenvolvedores com a estrutura de código *front-end* existente.

Após as tarefas serem cumpridas, elas foram avaliadas quanto a facilidade, aprendizado e manutenção da estrutura e códigos que eles haviam interagido. Foi usada uma escala de 5 pontos (de Discordo fortemente a Concordo Fortemente).

5.3.1.1 Tarefas

O cabeçalho que apresentava as tarefas para os desenvolvedores dizia:

"Você irá desempenhar as tarefas a seguir para adicionar o novo componente no fluxo do *framework*. O novo componente se chama *EulDateInput.vue*"Apêndice C mostra o cabeçalho completo

E apresentava as tarefas a serem feitas para completar o objetivo:

- T1 - Adicione o componente no repositório do sistema
- T2 - Adicione o componente no *EulElement.vue*
- T3 - Adicione a validação do componente
- T4 - Valide o componente, suba o sistema novamente

Os desenvolvedores então tinham que realizar as tarefas. Não havia tempo limite e eles podiam fazer da maneira que achassem melhor. Observações: O desenvolvedor D1 pediu ajuda para o *chatGPT* durante o desenvolvimento, para a escrita de uma condicional com muitas respostas possíveis usada na validação do componente. O D2 pediu uma sugestão direta para a pesquisadora de como terminar a validação do componente. As interações com a internet, e com a pesquisadora que estava aplicando o estudo, são vistas com neutralidade, já que estava sendo proposto um ambiente em que o desenvolvedor era requisitado a fazer uma tarefa especifica em código, em um contexto de manutenção.

Tarefas - *EulElement*					
	Respostas				
Tarefas	Muito Difícil	Difícil	Neutro	Fácil	Muito Fácil
T1	1	-	1	-	3
T2	1	1	2	1	-
T3	-	2	2	-	1
T4	1	-	1	2	1

Tabela 2 – Avaliação das tarefas da Fase 1

A Tabela 2 mostra as avaliações dos desenvolvedores para cada tarefa, nela estão quantos dos voluntários escolheram cada resposta. A Figura 20 mostra a representação desses dados em gráfico, as respostas foram mistas.

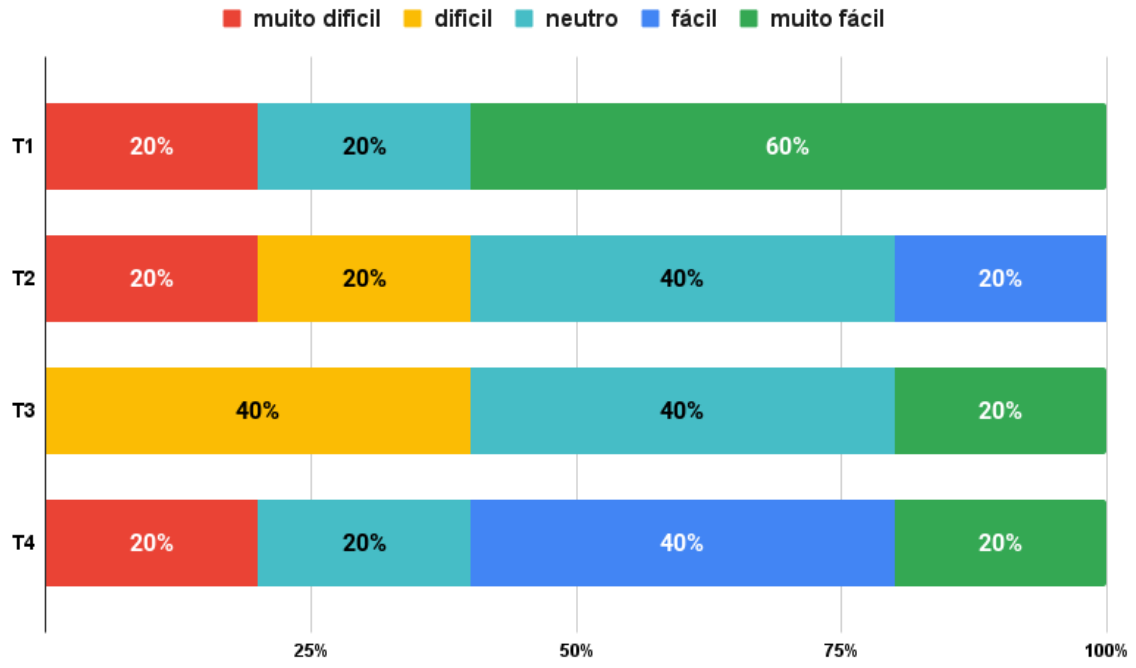


Figura 20 – Gráfico com as respostas das Avaliações das Tarefas da Fase 1

Fonte: Elaborada pelo autor

É importante observar que as respostas para a tarefa T3 tendem para o lado negativo da percepção dos desenvolvedores (Figura 20). Nessa tarefa eles deveriam adicionar a função que seria chamada para a validação do componente adicionado. A estrutura do *EulElement*, conforme mostrada na Figura 27, espera que uma das condições de *if-else* seja cumprida para a escolha de qual componente será mostrado para o usuário na tela final. Isso deveria ser algo fácil já que um *if-else* é uma estrutura básica para programadores. Mas aqui a arquitetura é confusa, e até mesmo uma estrutura simples fica complicada para os desenvolvedores que participaram do estudo.

5.3.1.2 Questionário

Após cumprirem as tarefas, os participantes responderam as seguintes questões relacionadas aos códigos com que eles interagiram:

- Q9 - Você conseguiria dar manutenção no sistema?
- Q10 - As condicionais são entendíveis.
- Q11 - Você considera o componente: *EulElement.vue*, bem estruturado?
- Q12 - O componente cumpre seu papel.
- Q13 - Você recomendaria uma refatoração desse componente?
- Q14 - Você conseguiria acrescentar de forma fácil outra variável que seria responsável pela escolha dos componentes?
- Q15 - Sem a explicação do sistema você teria conseguido fazer as tarefas da mesma maneira.

Na Tabela 3 estão compiladas as respostas dadas pelos desenvolvedores, em um esquema de 5-pontos (de Discordo Fortemente até Concordo Fortemente). A Figura 21 apresenta as respostas num gráfico.

Questões - *EulElement*					
Questões	Respostas				
	Discordo Fortemente	Discordo	Neutro	Concordo	Concordo Fortemente
Q9	-	1	-	3	1
Q10	-	1	1	1	2
Q11	-	-	1	2	2
Q12	-	1	-	-	4
Q13	-	3	-	1	1
Q14	-	-	1	2	2
Q15	2	-	2	-	1

Tabela 3 – Compilado das respostas das questões da Fase 1

Olhando para o gráfico na Figura 21 podemos perceber que as respostas, em geral, são mais neutras para positivas. A questão Q13, que fala sobre uma refatoração, teve uma recepção negativa, isso era esperado. A Q11, que pergunta diretamente sobre o componente em que eles interagiram, tiveram respostas mais positivas novamente. Mas, mesmo com essa tendência por parte dos participantes, uma coisa a se observar é que na Q10 as respostas tendem para o positivo, o que pode ser contradito pelo desempenho mostrado pelos desenvolvedores na tarefa T3.

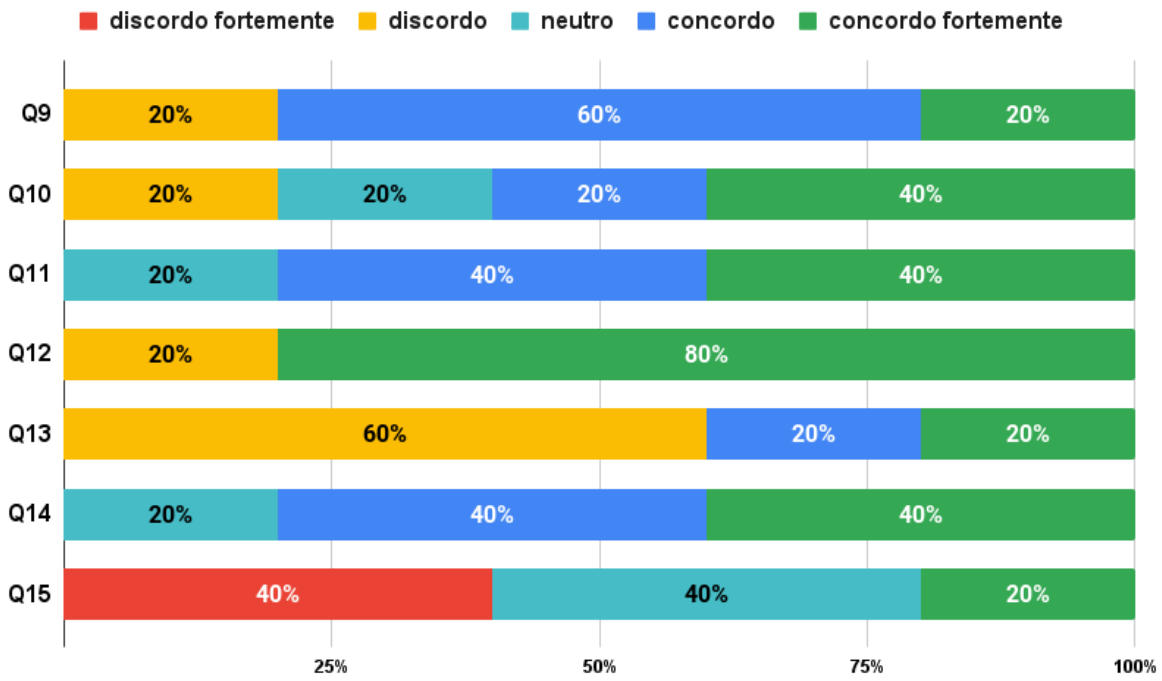


Figura 21 – Gráfico com as respostas das questões da Fase 1

Fonte: Elaborada pelo autor

5.3.2 Fase 2 - Avaliação do MorLib

Esta fase seguiu uma estrutura similar à primeira, mas com foco na nova arquitetura proposta. Os desenvolvedores foram novamente desafiados a realizar e avaliar tarefas para incluir um novo componente no fluxo do sistema. Mas agora tarefas diferentes, já que a estrutura de código da nova versão é distinta da anterior.

Novamente foi usada a escala Likert de 5-pontos para respostas dos desenvolvedores. Para essa nova fase foi feita novamente uma explicação do sistema por parte da pesquisadora.

5.3.2.1 Tarefas

Nessa fase, o cabeçalho das tarefas para adicionar um novo componente era o mesmo da Fase 1. Mas as tarefas eram diferentes:

- T5 - Crie uma regra nova chamada *rEulDateInput.json*
- T6 - Valide essa regra no documento de teste
- T7 - Insira o componente no fluxo do sistema
- T8 - Insira o componente no documento de exportação
- T9 - Faça a *build* da *MorLib* novamente

- T10 - Valide o componente, suba o sistema novamente

Como na Fase 1, não existiam limitações de tempo ou quanto a forma de resolver o problema. Nessa fase nenhum dos voluntários fez uso da internet.

Tarefas - *MorLib*					
	Respostas				
Tarefas	Muito Difícil	Difícil	Neutro	Fácil	Muito Fácil
T5	-	-	1	-	4
T6	-	-	-	1	4
T7	-	-	-	-	5
T8	-	-	-	-	5
T9	-	-	-	-	5
T10	-	-	-	-	5

Tabela 4 – Avaliação das tarefas da Fase 2

Na Tabela 4 vemos as avaliações dos desenvolvedores. A Figura 22 é a representação gráfica. Aqui já podemos observar a diferença de aceitação, dos desenvolvedores, para com a nova estrutura do *front-end*. Apesar de existirem mais tarefas nessa fase, os voluntários avaliaram as mesmas com praticamente 100% para Muito Fácil.

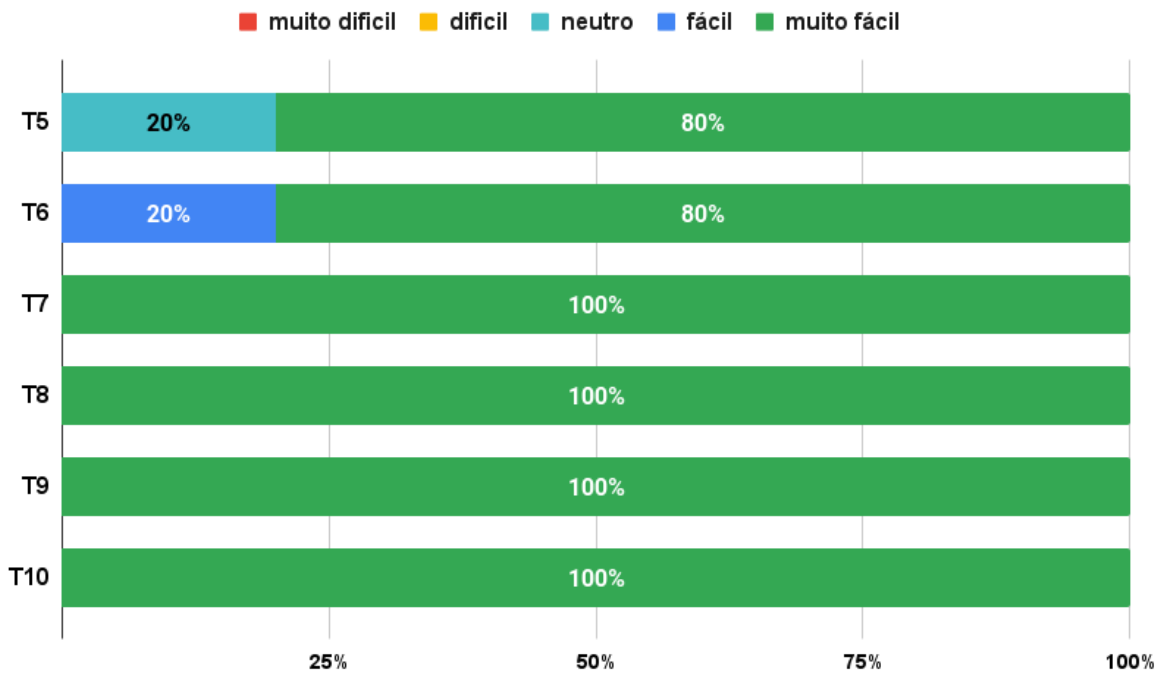


Figura 22 – Gráfico com as avaliações das Tarefas da Fase 2

Fonte: Elaborada pelo autor

As duas primeiras tarefas, T5 e T6, que não tiveram 100% como Muito Fácil, são relacionadas as regras em *JSON*, era esperado já que os desenvolvedores tinham acabado de

ser apresentados ao conceito, e mesmo assim, a aprovação ainda foi alta. O desenvolvedor D5 confirma essa interpretação "Eu acho que as regras são um pouco complicadas de entender porque elas são novas, mas acho que elas têm uma estrutura simples, depois que você entende".

5.3.2.2 Questionário

As questões que os participantes responderam após cumprir as tarefas foram:

- Q16 - Você conseguiria dar manutenção no sistema?
- Q17 - As regras são entendíveis.
- Q18 - Você considera o componente: *MorLib* bem estruturado?
- Q19 - O componente cumpre seu papel.
- Q20 - Você conseguiria acrescentar outra variável que seria responsável pela escolha dos componentes?

Na Tabela 5 estão compiladas as respostas dos desenvolvedores. Na Figura 23 temos o gráfico que as representam.

Questões - *MorLib*					
	Respostas				
Questões	Discordo Fortemente	Discordo	Neutro	Concordo	Concordo Fortemente
Q16	-	-	1	1	3
Q17	1	-	1	-	3
Q18	-	-	-	1	4
Q19	-	-	-	-	5
Q20	-	-	-	-	5

Tabela 5 – Compilado das respostas das questões da Fase 2

Nessa versão do sistema, as questões mostram uma aprovação maior. A única resposta totalmente negativa sobre as regras, a Q17, teve um comentário do D4 "Como as regras são escritas em *JSON*, eu preferiria que houvesse um passo a passo bem explicado de como elas funcionam, eu só copiei as outras, não acho que entendi o que eu estava fazendo". Como comentado anteriormente, alguns dos desenvolvedores tiveram dificuldades com as regras, mas, mesmo assim, foram melhor aceitas à solução com *if-else* da Fase 1. A pergunta Q18, que aborda especificamente a *MorLib*, mostra um alto nível de aceitação, ou seja, a nova estrutura está cumprindo seu papel de uma forma mais aceita pelos desenvolvedores participantes.

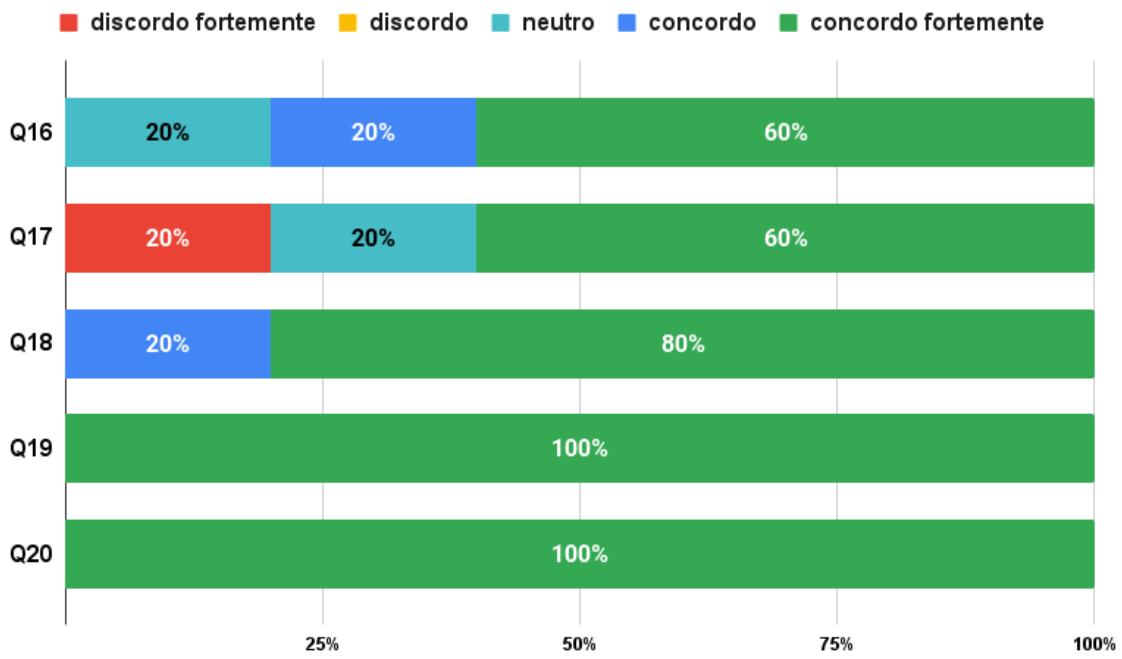


Figura 23 – Gráfico com o compilado das respostas das Questões da Fase 2

Fonte: Elaborada pelo autor

5.3.3 Fase 3 - Comparativo

Nesta fase, os desenvolvedores responderam a um questionário com perguntas mais específicas comparando as duas versões que eles tinham acabado de avaliar/programar. Aqui as perguntas foram mais diretas, mas a maioria das questões ainda usava o esquema de 5-pontos (de Discordo Fortemente a Concordo Fortemente). Mesmo que a intenção fosse ter uma resposta mais direta, ainda temos que levar em conta a subjetividade dos voluntários.

Como estamos falando de comparação, esta fase, além do questionário respondido pelos desenvolvedores, também traz um comparativo dos tempos de execução das fases pelos desenvolvedores.

5.3.3.1 Questionário

A seguir são apresentadas as questões feitas aos desenvolvedores:

- Q21 - Entre a 1ª versão, *EulElement*, e a 2ª, *Morlib*, prefiro a 1ª.
- Q22 - O *EulElement* cumpre seu papel como componente numa estrutura componentizada.
- Q23 - O *EulElement* já estava bom o bastante e a refatoração foi desnecessária.
- Q24 - O *EulElement* é bem comentado.
- Q25 - O *EulElement* não tem problemas na estrutura.

- Q26 - O *EulElement* é entendível e bem organizado.
- Q27 - A *MorLib* cumpre seu papel como componente numa estrutura de componentizada.
- Q28 - A *MorLib* é bem comentada.
- Q29 - A *MorLib* não tem problemas na estrutura.
- Q30 - A *MorLib* é entendível e bem organizado.
- Q31 - O sistema de regras *JSON* é melhor que o *if-else* do *EulElement*.
- Q32 - As regras são fáceis de alterar.

As questões, como pode ser observado, estão divididas entre as duas versões do sistema.

Questões - Comparativo					
	Respostas				
Questões	Discordo Fortemente	Discordo	Neutro	Concordo	Concordo Fortemente
Q21	3	1	-	1	-
Q22	-	1	-	3	1
Q23	3	2	-	-	-
Q24	1	1	2	-	1
Q25	-	1	3	-	1
Q26	-	2	1	1	1
Q27	-	-	-	2	3
Q28	-	-	-	1	4
Q29	-	-	-	2	3
Q30	-	-	-	1	4
Q31	-	1	-	-	4
Q32	-	-	-	1	4

Tabela 6 – Compilado das respostas das questões da Fase 3

Na Tabela 6 temos as respostas compiladas. Já conseguimos ver, somente pelo posicionamento na tabela de onde se encontram as respostas, quais dos grupos de questões tem mais aprovação. Nas Figuras 24 e 25, temos os gráficos que também corroboram com afirmação anterior.

Olhando algumas questões individualmente, como a Q21, que perguntava diretamente sobre a preferência, temos 80% de discordância, quanto a preferência ser a versão anterior do sistema. É importante ressaltar isso, já que se voltarmos na Q13 durante a Fase 1, os desenvolvedores em sua maioria não recomendavam uma refatoração. Mas após a interação com o novo sistema tivemos uma completa mudança de cenário.

A Figura 26 mostra os gráficos dos comparativos das questões Q24 & Q28, Q26 & Q30 e Q22 & Q27. As questões estão relacionadas, respectivamente, aos comentários no código,

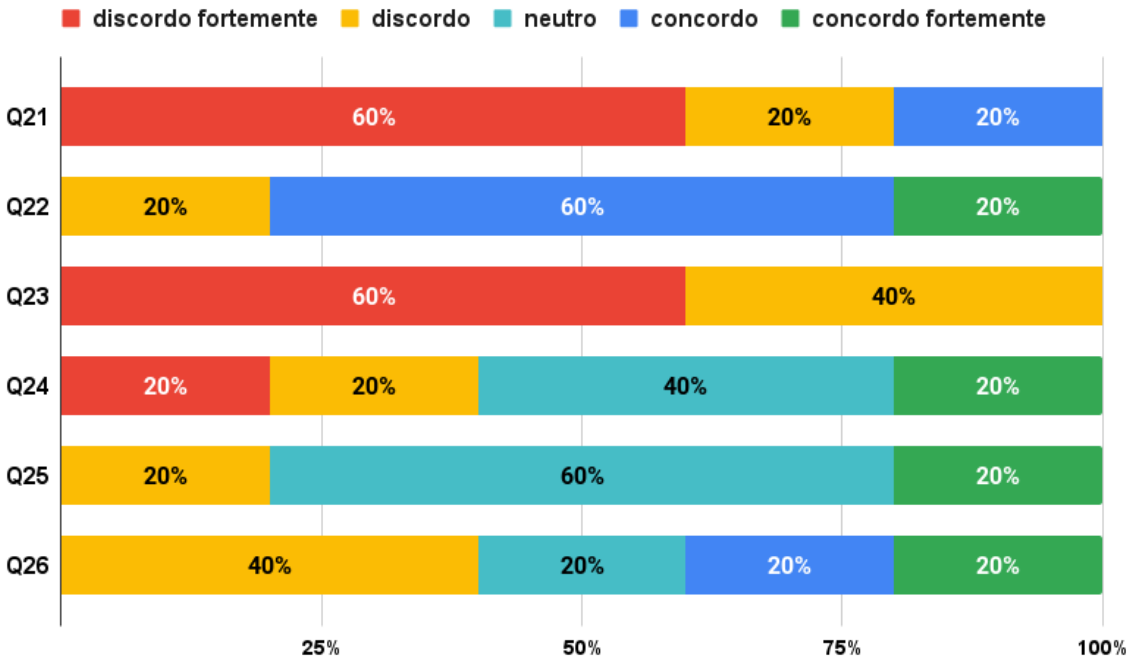


Figura 24 – Gráfico com o compilado das respostas das questões Q21 à Q26

Fonte: Elaborada pelo autor

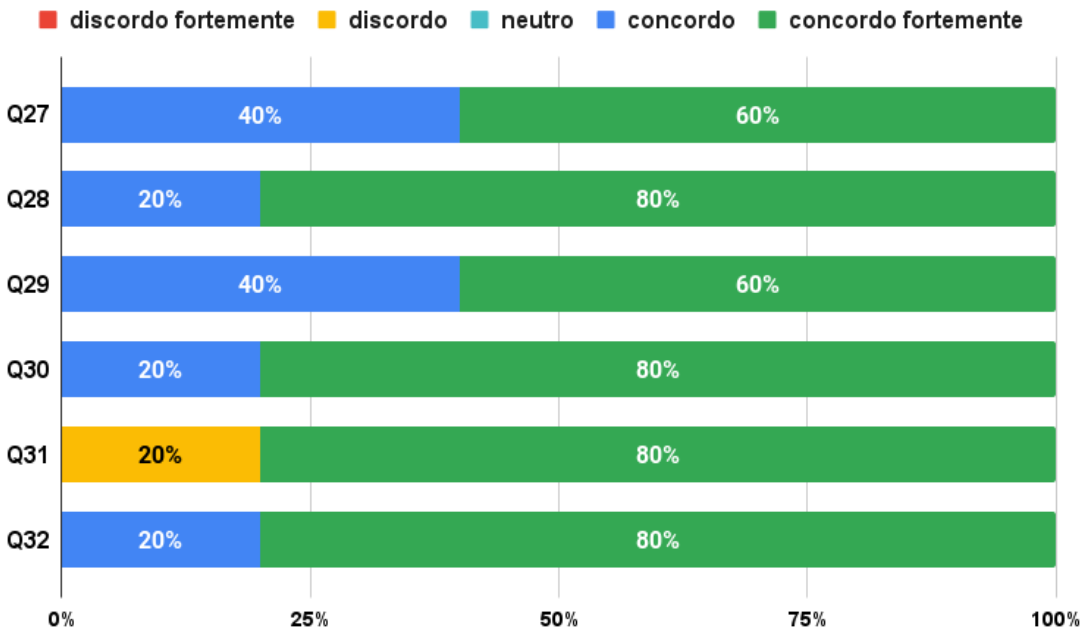


Figura 25 – Gráfico com o compilado das respostas das questões Q27 à Q32

Fonte: Elaborada pelo autor

a estrutura, e se os sistemas estão usando o conceito de componentização. A *MorLib* tem vantagem nas duas primeiras comparações, e, em questão a componentização, os dois sistemas tem resultados positivos. Mas a nova versão ainda se destaca com uma porcentagem maior de

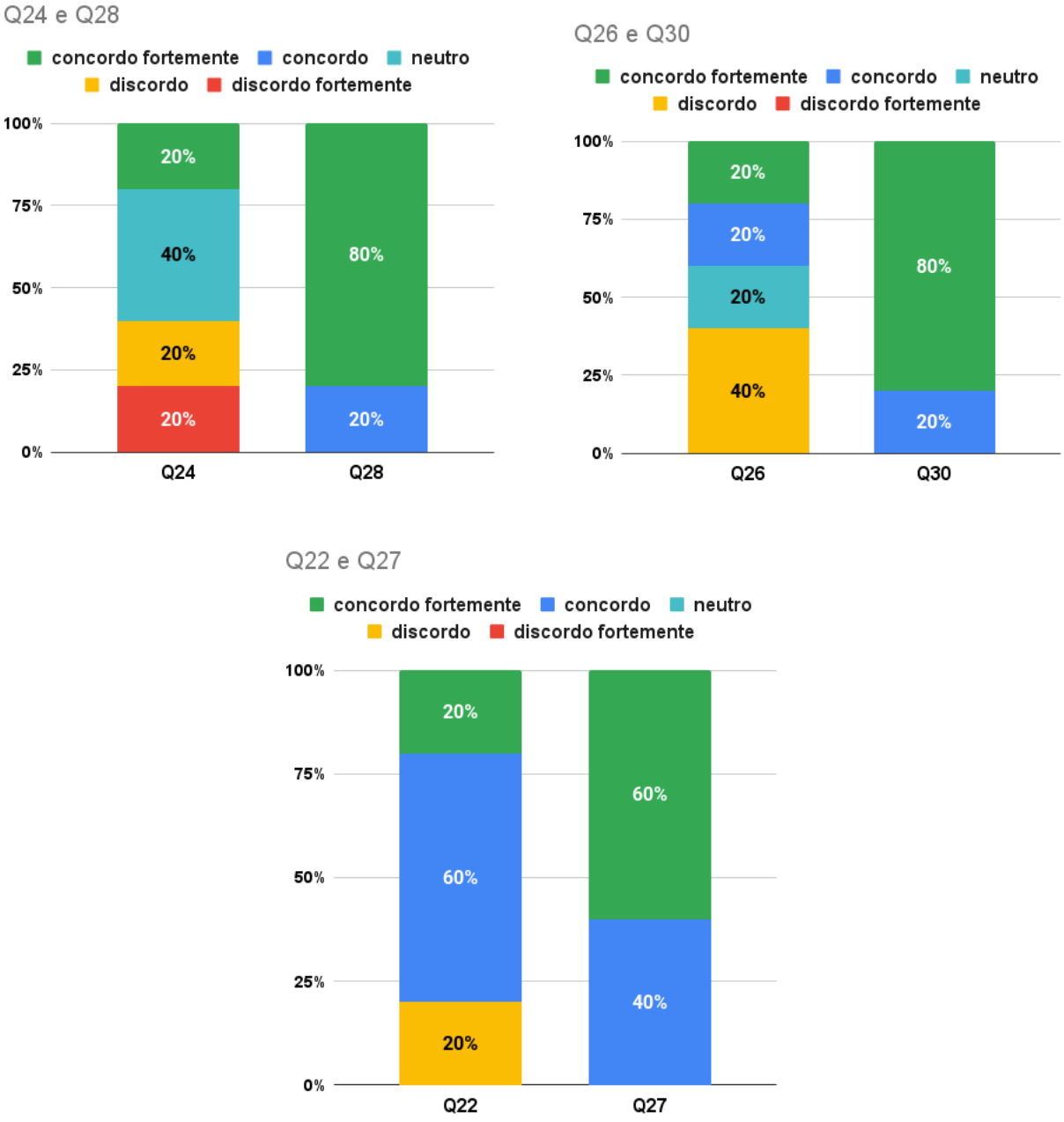


Figura 26 – Gráficos comparando as perguntas feitas em relação as duas versões do sistema

Fonte: Elaborada pelo autor

Concordo Fortemente.

Para finalizar essa fase, mais 3 questões foram feitas aos desenvolvedores. Elas estão separadas porque nesse caso não foi usada a escala de 5-pontos. Só existiam duas opções: "*EulElement*"ou "*MorLib*". As questões eram:

- Q33 - Minha curva de aprendizado foi melhor
- Q34 - Eu prefiro a estrutura, pensando em manutenção

- Q35 - Eu prefiro a estrutura, pensando em aprendizagem

Questões		
	Respostas	
Questões	*EulElement*	*MorLib*
Q33	1	4
Q34	1	4
Q35	1	4

Tabela 7 – Compilado das respostas para as questões Q33 à Q35

Como pode ser observado na Tabela 7, a *MorLib* foi preferida por 4 entre 5 dos desenvolvedores participantes. Essa é uma informação importante, pois, como dito no início deste capítulo, esse estudo foi feito justamente para responder se o novo sistema se sairia melhor em questões de manutenção e aprendizado.

5.3.3.2 Comparativo de Tempo

Para avaliar a eficiência das duas versões, foi feito um comparativo dos tempos de execução das tarefas em ambas as fases. Aqui é importante lembrar novamente que, na fase 1, os desenvolvedores já tinham feito tarefas de inclusão de componentes e isso pode ter facilitado seu trabalho na fase 2, já que os dois sistemas não são 100% diferentes.

Foram medidos o tempo médio de duração das tarefas nas duas primeiras fases, e a quantidade de intervenções feitas pela pesquisadora, na maioria das vezes para responder perguntas dos desenvolvedores. Para as fases 1 e 2, foram medidos os tempos e a quantidade de tentativas para conseguir adicionar o componente.

Tempos de Duração e Interferências						
	Tempo sem Compilações		Tempo Total		Interferências	
Desenvolvedores	*EulElement*	*MorLib*	*EulElement*	*MorLib*	*EulElement*	*MorLib*
D1	18:16	03:13	26:00	05:15	21	7
D2	29:38	05:46	34:05	12:18	14	7
D3	16:51	05:55	21:19	10:25	17	9
D4	11:14	04:36	23:20	07:25	5	3
D5	16:38	09:30	19:32	11:04	3	5
Média	18:31	05:48	24:51	09:17	12	6

Tabela 8 – Tempos de duração, e quantidade de interferências, das fases 1 e 2

Na Tabela 8 temos os "Tempos Totais", que dizem respeito ao tempo que cada desenvolvedor levou para completar o objetivo de inserir o componente no fluxo do sistema, em cada fases. Também temos o "Tempo sem Compilações", em que o tempo das compilações do sistema foram desconsiderados no tempo final, ficando assim um tempo exclusivo de resolução

de problema. O tempo de explicação dos sistemas e preenchimento dos questionários, não foi excluído das duas somatórias.

Considerando as médias de Tempo sem as Compilações, podemos inferir que o tempo para a mesma tarefa, feita na *MorLib*, diminuiu em cerca de 68%. Se falamos em Tempo Total, a diminuição média fica em torno de 62%. Em relação às interferências, tivemos uma diminuição de 50%, relembrando que essa contagem é referente às vezes que a pesquisadora respondeu perguntas dos desenvolvedores, desde perguntas conceituais a práticas. São bons resultados, mesmo levando em conta o possível viés já mencionado.

5.4 Considerações Finais

Através deste estudo de caso, foi possível avaliar a aceitação dos desenvolvedores em relação às mudanças propostas na arquitetura interna do *front-end* do *framework* EUL. Os resultados indicam a preferência pela segunda versão, *MorLib*, tanto em termos de percepção quanto de eficiência. A refatoração proposta neste trabalho mostrou-se bem-sucedida no contexto deste estudo.

Esses resultados reforçam a importância de uma arquitetura bem estruturada e de fácil compreensão, especialmente em ambientes colaborativos onde múltiplos desenvolvedores interagem com o código.

CAPÍTULO

6

CONCLUSÃO

Ao longo deste trabalho, aprimoramos o *front-end* do EUL *framework* através da implementação de um sistema de regras. Esta abordagem não só trouxe flexibilidade ao sistema, mas também foi testada por desenvolvedores, que a preferiram em relação à estrutura anterior.

A introdução de regras, que dão suporte a natureza dinâmica dos componentes, respondeu à questão de pesquisa proposta. Essa nova estrutura permite a inclusão de componentes no fluxo do sistema de maneira mais ágil, facilitando a construção das interfaces de usuário das DSS criadas com o EUL *Framework*. A utilização de regras em JSON proporciona uma ampla gama de combinações, permitindo descrever com precisão os tipos de dados para os quais um componente é mais adequado.

Um dos principais objetivos deste trabalho era simplificar o desenvolvimento de novas interfaces. O tempo necessário para integrar novos componentes ao sistema pode ser reduzido, e o *feedback* dos desenvolvedores, que tiveram contato com o novo sistema, foi positivo.

As principais contribuições deste trabalho incluem a implementação de uma estrutura de regras eficiente, capaz de selecionar os componentes adequados para dados descritos em ontologias, e a consequente facilitação na customização de novas DSS instanciadas pelo EUL *Framework*.

No entanto, enfrentamos desafios ao longo do caminho. Uma dificuldade foi que o EUL *Framework* é um sistema grande e complexo que teve muitos contribuidores durante os anos. Isso torna desafiador manter uma documentação atualizada e garantir que todos os colaboradores, a par dos trabalhos sendo realizados, estejam cientes dos desenvolvimentos em curso.

Outra dificuldade foi a de estabelecer as métricas para se avaliar código bruto de *framework javascript*, foram consultados outros pesquisadores da área de Eng. Software para a procura dessas métricas, mas ainda assim houve grande dificuldade de estabelecê-las.

6.1 Trabalhos Futuros

Olhando para o futuro, identificamos várias oportunidades para aprimoramento:

- Realização de estudos de usabilidade mais completos e rigorosos com usuários.
- A reestruturação do *back-end* do EUL *Framework* para complementar as atualizações feitas no *front-end*.
- A implementação de variáveis que permitam a criação de UIs que atendam a usuários com problemas de acessibilidade específicos, sem afetar a usabilidade para todos.
- O desenvolvimento de uma versão atualizada do editor de ontologias e DSS, proporcionando aos *experts* de domínio maior controle sobre o DSS.

Esperamos que este trabalho sirva como base para futuras inovações na área.

REFERÊNCIAS

ABRAMS, M.; PHANOURIOU, C.; BATONGBACAL, A. L.; WILLIAMS, S. M.; SHUSTER, J. E. Uiml: an appliance-independent xml user interface language. **Computer networks**, Elsevier, v. 31, n. 11-16, p. 1695–1708, 1999. Citado na página 36.

ALLEMANG, D.; HENDLER, J. **Semantic web for the working ontologist: effective modeling in RDFS and OWL**. [S.l.]: Elsevier, 2011. Citado nas páginas 29 e 30.

BALZERT, H.; HOFMANN, F.; KRUSCHINSKI, V.; NIEMANN, C. The janus application development environment-generating more than the user interface. In: **CADUI**. [S.l.: s.n.], 1996. v. 96, p. 183–206. Citado na página 37.

BERGH, J. Van den; LUYTEN, K.; CONINX, K. Cap3: context-sensitive abstract user interface specification. In: ACM. **Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems**. [S.l.], 2011. p. 31–40. Citado na página 37.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. *et al.* The semantic web. **Scientific american**, New York, NY, USA:, v. 284, n. 5, p. 28–37, 2001. Citado na página 29.

CARDOSO, B. O. *et al.* Avaliação da sustentabilidade de sistemas de produção da cana-de-açúcar no estado de são paulo: uma proposta metodológica e de modelo conceitual. Universidade Federal de São Carlos, 2013. Citado na página 24.

ETIKAN, I.; MUSA, S. A.; ALKASSIM, R. S. *et al.* Comparison of convenience sampling and purposive sampling. **American journal of theoretical and applied statistics**, New York, v. 5, n. 1, p. 1–4, 2016. Citado na página 55.

FEDORTSOVA, I.; BROWN, G. Javafx mastering fxml release 8. **JavaFX Doc. http://docs. oracle. com/javase/8/javafx/fxml-tutorial/preface. htm. Abgerufen am**, v. 4, p. 2015, 2014. Citado na página 36.

FOWLER, M. **Domain-specific languages**. [S.l.]: Pearson Education, 2010. Citado nas páginas 24 e 34.

GAULKE, W.; ZIEGLER, J. Using profiled ontologies to leverage model driven user interface generation. In: ACM. **Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems**. [S.l.], 2015. p. 254–259. Citado na página 37.

HEINZLE, R.; GAUTHIER, F. A. O.; FIALHO, F. A. P. Semântica nos sistemas de apoio a decisão: o estado da arte. **Revista da UNIFEBE**, v. 1, n. 8, p. 225–248, 2017. Citado na página 27.

HOGAN, A.; HOGAN, A. **Web of data**. [S.l.]: Springer, 2020. Citado na página 32.

JAISWAL, M. Software architecture and software design. **International Research Journal of Engineering and Technology (IRJET) e-ISSN**, p. 2395–0056, 2019. Citado na página 27.

JESUS, K. R. E. de; TORQUATO, S. A.; MACHADO, P. G.; ZORZO, C. R. B.; CARDOSO, B. O.; LEAL, M. R. L. V.; PICOLI, M. C. A.; RAMOS, R. C.; DALMAGO, G. A.; CAPITANI, D. H. D. *et al.* Sustainability assessment of sugarcane production systems: Sustenagro decision support system. **Environmental Development**, Elsevier, 2019. Citado na página 38.

KNUBLAUCH, H.; KONTOKOSTAS, D. Shapes constraint language (shacl), w3c recommendation 20 july 2017. **URL: https://www. w3. org/TR/shacl**, 2017. Citado na página 32.

LIKERT, R. A technique for the measurement of attitudes. **Archives of psychology**, 1932. Citado na página 56.

LIMBOURG, Q.; VANDERDONCKT, J.; MICHOTTE, B.; BOUILLON, L.; FLORINS, M. Usixml: A user interface description language supporting multiple levels of independence. In: **ICWE Workshops**. [S.l.: s.n.], 2004. p. 325–338. Citado na página 36.

LIU, B.; CHEN, H.; HE, W. Deriving user interface from ontologies: a model-based approach. In: IEEE. **17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)**. [S.l.], 2005. p. 6–pp. Citado na página 37.

MALIK, N. M.; MUSHTAQ, A.; KHALID, S.; KHALIL, T.; MALIK, F. M. Measurable & scalable nfrs using fuzzy logic and likert scale. **arXiv preprint arXiv:0906.5393**, 2009. Citado na página 56.

MENEZES, W. M. d.; MOREIRA, D. d. A. Enhancing decision support systems development with the eul framework applied in the agricultural sustainability assessment domain. 2023. Citado na página 26.

MOLINA, P. J.; MOLINA, P. J.; MOLINA, P. J. Quid: prototyping web components on the web. **Engineering Interactive Computing System**, 2019. Citado na página 37.

NETO, A. L. dos S.; MARCONDES, C. H.; PEREIRA, D. V.; FONSECA, E. R. da; SOUZA, I. V. P. de; BARBOSA, N.; MORAES, R. P. T. de; MARTINS, S. de C. Tecnologias de dados abertos para interligar bibliotecas, arquivos e museus: um caso machadiano using open data technology to connect libraries, archives and museums: a machadian case. **TransInformação**, SciELO Brasil, v. 25, n. 1, p. 81–87, 2013. Citado na página 31.

NOY, N. F.; MCGUINNESS, D. L. *et al.* **Ontology development 101: A guide to creating your first ontology**. [S.l.]: Stanford knowledge systems laboratory technical report KSL-01-05 and ..., 2001. Citado na página 30.

PATEL-SCHNEIDER, P. F. Building the semantic web tower from rdf straw. In: **IJCAI**. [S.l.: s.n.], 2005. p. 546–551. Citado na página 30.

PATERNO, F.; SANTORO, C.; SPANO, L. D. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. **ACM Transactions on Computer-Human Interaction (TOCHI)**, ACM, v. 16, n. 4, p. 19, 2009. Citado na página 37.

PIERONI, R. *et al.* Desenvolvimento de uma dsl para a gerência de configuração de um sistema de gerenciamento integrado de redes. Universidade Federal de São Carlos, 2014. Citado na página 34.

POPP, R.; FALB, J.; ARNAUTOVIC, E.; KAINDL, H.; KAVALDJIAN, S.; ERTL, D.; HORACEK, H.; BOGDAN, C. Automatic generation of the behavior of a user interface from a high-level discourse model. In: IEEE. **2009 42nd Hawaii International Conference on System Sciences**. [S.l.], 2009. p. 1–10. Citado na página 37.

POWER, D. J. **Decision support systems: concepts and resources for managers**. [S.l.]: Greenwood Publishing Group, 2002. Citado na página 28.

PUERTA, A. R.; ERIKSSON, H.; GENNARI, J. H.; MUSEN, M. A. Beyond data models for automated user interface generation. In: **BCS HCI**. [S.l.: s.n.], 1994. p. 353–366. Citado na página 37.

RANEBURGER, D.; KAINDL, H.; POPP, R. Model transformation rules for customization of multi-device graphical user interfaces. **Engineering Interactive Computing System**, 2015. Citado na página 37.

RIEGER, C.; KUCHEN, H. A model-driven cross-platform app development process for heterogeneous device classes. In: **Proceedings of the 52nd Hawaii International Conference on System Sciences**. [S.l.: s.n.], 2019. Citado na página 37.

SCHIUMA, G.; GAVRILOVA, T.; ANDREEVA, T. Knowledge elicitation techniques in a knowledge management context. **Journal of Knowledge Management**, Emerald Group Publishing Limited, 2012. Citado na página 23.

SHADBOLT, N.; BERNERS-LEE, T.; HALL, W. The semantic web revisited. **IEEE intelligent systems**, IEEE, v. 21, n. 3, p. 96–101, 2006. Citado na página 31.

SILVA, C. R. Q. Critérios para priorização de estudos primários identificados por snowballing com conjunto inicial gerado por string de busca. Universidade Federal de São Carlos, 2017. Citado na página 36.

SOMEREN, M. V.; BARNARD, Y. F.; SANDBERG, J. The think aloud method: a practical approach to modelling cognitive. **London: AcademicPress**, Citeseer, v. 11, p. 29–41, 1994. Citado na página 56.

SUAREZ, J. F. G. **Ontologias e DSLs na geração de sistemas de apoio à decisão, caso de estudo SustenAgro**. Tese (Doutorado) — Universidade de São Paulo, 2017. Citado nas páginas 38 e 39.

SUGUMARAN, V.; GULLA, J. Applied semantic web technologies: Overview and future directions. In: **Applied Semantic Web Technologies**. [S.l.]: Auerbach Publications, 2011. p. 17–34. Citado na página 30.

SZYPERSKI, C.; GRUNTZ, D.; MURER, S. **Component software: beyond object-oriented programming**. [S.l.]: Pearson Education, 2002. Citado na página 38.

TURBAN, E.; ARONSON, J.; LIANG, T. **Decision Support Systems and Intelligent Systems**. Pearson/Prentice Hall, 2005. ISBN 9780130461063. Disponível em: <https://books.google.com.br/books?id=NfMJAQAAMAAJ>. Citado na página 23.

TWEEDALE, J. W.; PHILLIPS-WREN, G.; JAIN, L. C. Advances in intelligent decision-making technology support. In: **Intelligent Decision Technology Support in Practice**. [S.l.]: Springer, 2016. p. 1–15. Citado na página 28.

APÊNDICE

A

CÓDIGOS E IMAGENS

A.1 Imagens e Códigos do Capítulo 4

```
<EulDataType
    v-else
    v-model="content"
    :element="relation"
    :edit="edit"
    :title="comment | localize">
    <template #default="{value, onInput, required}">
        <EulSHA256Input
            v-if="isSHA256Field"
            :value="value"
            :label="label"
            :required="required"
            :edit="edit"
            @input="payload => onInput(payload)" />

        <EulImageInput
            v-else-if="isImageField"
            :value="value"
            :label="label"
            :required="required"
            :edit="edit"
            @input="payload => onInput(payload)" />

        <EulFileInput
            v-else-if="isFileField"
            :value="value"
            :label="label"
            :required="required"
            :edit="edit"
            @input="payload => onInput(payload)" />

<EulBirthdayInput
    v-else-if="isBirthday"
    :value="value"
    :label="label"
    :required="required"
    :edit="edit"
    @input="payload => onInput(payload)" />

<EulCodeInput
    v-else-if="isCode"
    :value="value"
    :label="label"
    :required="required"
    :edit="edit"
    @input="payload => onInput(payload)" />

<!--                    <EulOntoJSONInput-->
<!--                        v-else-if="isOntoJSON"-->
<!--                        :value="value"-->
<!--                        :label="label"-->
<!--                        :required="required"-->
<!--                        :edit="edit"-->
<!--                        @input="payload => onInput(payload)" />-->

<!--                <div v-else-if="isOntoJSON" />-->

<EulDateInput
    v-else-if="isDateField"
    :type="range"
    :value="value"
    :label="label"
    :required="required"
    :edit="edit"
    @input="payload => onInput(payload)" />

                <EulNumericInput
                    v-else-if="isNumericField"
                    :type="range"
                    :value="value"
                    :label="label"
                    :required="required"
                    :edit="edit"
                    @input="payload => onInput(payload)" />

                <EulBooleanInput
                    v-else-if="range === 'xsd:boolean'"
                    :value="value"
                    :label="label"
                    :required="required"
                    :edit="edit"
                    @input="payload => onInput(payload)" />

                <EulLiteralInput
                    v-else
                    :value="value"
                    :label="label"
                    :required="required"
                    :edit="edit"
                    :uri="range === 'xsd:anyURI'"
                    @input="payload => onInput(payload)" />
            </template>
        </EulDataType>
```

Figura 27 – *DataTypes* no *EULElement*.vue

Fonte: Elaborada pelo autor

```
▼ Object { id: "http://purl.org/biodiv/sustenagro#beginningOfPlantingDate", type: (3) […], order: {…},
    id: "http://purl.org/biodiv/sustenagro#beginningOfPlantingDate"
  ▶ label: Array [ {…}, {…} ]
  ▶ order: Object { type: "xsd:decimal", "$value": "6" }
  ▶ "rdfs:range": Object { id: "xsd:gYearMonth" }
  ▶ "rdfs:subPropertyOf": Object { id: "http://purl.org/biodiv/sustenagro#beginningOfPlantingDate" }
  ▶ type: Array(3) [ "owl:FunctionalProperty", "owl:DatatypeProperty", "rdf:Property" ]
  ▶ <prototype>: Object { … }
```


Figura 28 – JSON-LD: gera o componente de data

Fonte: Elaborada pelo autor


```
{
    "conditions": {
        "any": [
            {
                "fact": "type",
                "operator": "equal",
                "value": "xsd:integer"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:nonNegativeInteger"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:nonPositiveInteger"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:decimal"
            }
        ]
    },
    "event": {
        "type": "numerical-is-acceptable",
        "params": {
            "value": "EulNumericInput"
        }
    },
    "name": "numerical_rule"
}
```


(a) Regra: Numerais


```
{
    "conditions": {
        "all": [{
            "fact": "type",
            "operator": "equal",
            "value": "xsd:anyURI"
        }]
    },
    "event": {
        "type": "literal-is-acceptable",
        "params": {
            "value": "EulLiteralInput"
        }
    },
    "name": "literal_rule"
}
```


(b) Regra: Literal

Figura 29 – Regras em JSON, *numeral* e *literal*

Fonte: Elaborada pelo autor

```
{
    "conditions": {
        "any": [
            {
                "fact": "type",
                "operator": "equal",
                "value": "xsd:date"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:time"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:dateTime"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:dateTimeStamp"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:gYear"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:gMonth"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:gDay"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:gYearMonth"
            },{
                "fact": "type",
                "operator": "equal",
                "value": "xsd:gMonthDay"
            }
        ]
    },
    "event": {
        "type": "date-is-acceptable",
        "params": {
            "value": "EulDateInput"
        }
    },
    "name": "date_rule"
}
```

(a) Regra: Datas

```
{
  "conditions": {
    "all": [{
    "fact": "type",
    "operator": "equal",
    "value": "ui:birthdate"
  }]
},
  "event": {
    "type": "birthday-is-acceptable",
    "params": {
      "value": "EulBirthdayInput"
    }
  },
  "name": "Birthday_rule"
}
```

(b) Regra: Aniversário

Figura 30 – Regras em JSON, *date* e *birthdate*

Fonte: Elaborada pelo autor

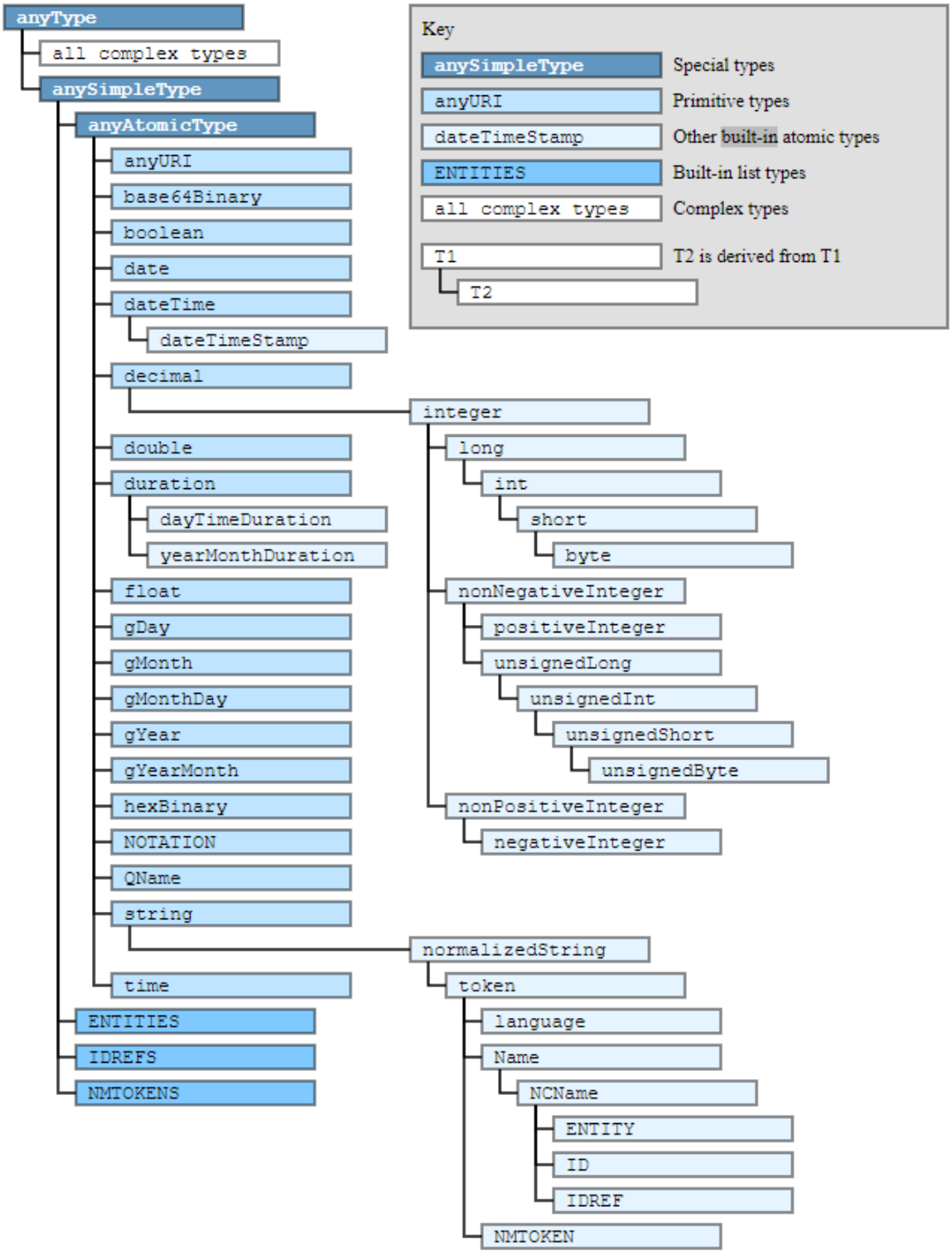


Figura 31 – Esquema de tipos.

Fonte: W3C

```
{
    "conditions": {
        "all": [{
            "fact": "type",
            "operator": "contains",
            "value": "owl:ObjectProperty"
        }]
    },
    "event": {
        "type": "object-is-acceptable",
        "params": {
            "value": "EulObject"
        }
    },
    "name": "object_rule"
}
```

Figura 32 – Regra: *ObjectType*

Fonte: Elaborada pelo autor

APÊNDICE

B

TERMO DE CONSENTIMENTO

Termo de Consentimento

Você está sendo convidado(a) a participar, como voluntário(a), desta pesquisa. Caso concorde em participar, favor consentir ao final desta leitura. Sua participação não é obrigatória e, a qualquer momento, poderá desistir e retirar seu consentimento.

Sua recusa não trará nenhum prejuízo à sua relação com os pesquisadores ou com a instituição. Você receberá uma cópia deste termo onde consta o *e-mail* dos responsáveis do grupo de pesquisa, podendo tirar suas dúvidas sobre o projeto e sua participação.

Nome da pesquisa: Refatoração do *Front-end* do *Framework* EUL: Componentização e Geração Automática de Interfaces de Usuário

Grupo de pesquisadores responsáveis: Mariany Morais, Dilvan de Abreu Moreira (Membros do laboratório Intermídia do Instituto de Ciências Matemáticas e de Computação, Universidade São Paulo).

Contato com o grupo: mariany.morais@usp.br, dilvan@icmc.usp.br

Objetivo do teste: Explorar e avaliar a manutenibilidade do *Framework* EUL. Validar a refatoração e entender o comportamento do desenvolvedor dentro do sistema, tentando identificar se houve melhoras. Como método de coleta de dados, será utilizado o *Think-Aloud*, onde o desenvolvedor irá verbalizar sua experiência durante as tarefas requisitadas, ou seja, "pensará alto" para que possa ser avaliado seu comportamento. Logo depois será aplicado um questionário para medir as escalas de manutenibilidade e compreensão do sistema de acordo com a perspectiva do desenvolvedor.

Procedimento do estudo: Se concordar em participar, você receberá uma introdução rápida ao *front-end* do *framework* EUL e deverá realizar as tarefas e responder as questões para avaliação, propostas pelos pesquisadores. Os dados serão utilizados para comparar as avaliações do sistema agora, e pré refatoração. A entrevista será gravada e depois transcrita, para o método

de *Think-Aloud*, e para verificação de tempo gasto durante as tarefas.

Confidencialidade da pesquisa: O questionário final não pede identificação, com o intuito de preservar a privacidade dos participantes e utilizar somente os dados obtidos de suas respostas.

Consentimento: Eu entendo que minha participação no referido estudo é voluntária e que a todo o momento tive a opção de não continuar caso desejasse, independente do motivo e sem precisar dar nenhuma explicação.

APÊNDICE

C

ESTUDO DE CASO - COMPLETO

Estudo de Caso EUL Framework - MorLib

Coleta de dados para pesquisa.

* Indica uma pergunta obrigatória

Termo de Consentimento

Termo de Consentimento

Você está sendo convidado(a) a participar, como voluntário(a), desta pesquisa. Caso concorde em participar, favor consentir ao final desta leitura. Sua participação não é obrigatória e, a qualquer momento, poderá desistir e retirar seu consentimento.

Sua recusa não trará nenhum prejuízo à sua relação com os pesquisadores ou com a instituição. Você receberá uma cópia deste termo onde consta o e-mail dos responsáveis do grupo de pesquisa, podendo tirar suas dúvidas sobre o projeto e sua participação.

Nome da pesquisa: Refatoração do Front-end do Framework EUL: Componentização e Geração Automática de Interfaces de Usuário

Grupo de pesquisadores responsáveis: Mariany Morais, Dilvan de Abreu Moreira (Membros do laboratório Intermídia do Instituto de Ciências Matemáticas e de Computação, Universidade São Paulo).

Contato com o grupo: mariany.morais@usp.br, dilvan@icmc.usp.br

Objetivo do teste: Explorar e avaliar a manutenibilidade do Framework EUL. Validar a refatoração e entender o comportamento do desenvolvedor dentro do sistema, tentando identificar se houve melhoras. Como método de coleta de dados, será utilizado o Think-Aloud, onde o desenvolvedor irá verbalizar sua experiência durante as tarefas requisitadas, ou seja, “pensará alto” para que possa ser avaliado seu comportamento. Logo depois será aplicado um questionário para medir as escalas de manutenibilidade e compreensão do sistema de acordo com a perspectiva do desenvolvedor.

Procedimento do estudo: Se concordar em participar, você receberá uma introdução rápida ao front-end do framework EUL e deverá realizar as tarefas e responder as questões para avaliação, propostas pelos pesquisadores. Os dados serão utilizados para comparar as avaliações do sistema agora, e pré refatoração.
A entrevista será gravada e depois transcrita, para o método de Think-Aloud, e para verificação de tempo gasto durante as tarefas.

Confidencialidade da pesquisa: O questionário final não pede identificação, com o intuito de preservar a privacidade dos participantes e utilizar somente os dados obtidos de suas respostas.

Consentimento: Eu entendo que minha participação no referido estudo é voluntária e que a todo o momento tive a opção de não continuar caso desejasse, independente do motivo e sem precisar dar nenhuma explicação.

1. Você concorda? *

   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

Questionário de controle

2. Você tem familiaridade com desenvolvimento web? *

   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

3. Você tem familiaridade com FrameworkJs? *

   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

4. Você já trabalhou com Frameworks Js? *

   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

5. Você já trabalhou com VueJs? *

   Marcar apenas uma oval.

   - ◯ Sim
   - ◯ Não

6. Você conhece o conceito de ontologias? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

7. Você conhece o paradigma de ontologias para web semântica? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

8. Você conhece RDF? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

9. Você conhece JSON? *

Marcar apenas uma oval.

- ◯ Sim
- ◯ Não

Tarefas - Eul

Você irá desempenhar as tarefas a seguir para
adicionar o novo componente no fluxo do framework. O novo componente se chama EulDateInput.vue e tem como tipos:

xsd:gYearMonth
xsd:date
xsd:time
xsd:dateTime
xsd:dateTimeStamp
xsd:gYear
xsd:gMonth
xsd:gDay
xsd:gYearMonth
xsd:gMonthDay

1. Adicione o componente no repositório do sistema
2. Adicione o componente no EulElement
3. Adicione a validação do componente
4. Valide o componente, suba o sistema novamente

10. 1. Adicione o componente no repositório do sistema *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

11. 2. Adicione o componente no EulElement.vue *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

12. 3. Adicione a validação do componente *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

13. 4. Valide o componente, suba o sistema novamente *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

Pós questões

Responda com base nas tarefas e na explicação sobre o sistema que foi dada

14. Você conseguiria dar manutenção no sistema? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

15. As condicionais são entendíveis. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

16. Você considera o componente: EulElement, bem estruturado? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

17. O componente cumpre seu papel. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

18. Você recomendaria uma refatoração desse componente? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

19. Você conseguiria acrescentar de forma fácil outra variável que seria responsável pela escolha dos componentes? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

20. Sem a explicação do sistema você teria conseguido fazer as tarefas da mesma maneira. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

Tarefas

Você irá desempenhar as tarefas a seguir para adicionar o novo componente no fluxo do framework. O novo componente se chama EulDateInput.vue e tem como tipo específico "xsd:gYearMonth"
O componente será disponibilizado para você.

1. Crie uma regra nova chamada rEulDateImput.json
2. Valide essa regra no documento de teste
3. Insira o componente no fluxo do sistema
4. Insira o componente no documento de exportação
5. Faça a build da morringhan novamente
6. Valide o componente, suba o sistema novamente

A cada tarefa, responda se achou Muito Fácil - Muito Difícil, cumprir a mesma

21. 1. Crie uma regra nova *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

22. 2. Valide essa regra *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

23. 3. Insira um componente no fluxo do sistema *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

24. 4. Insira o componente no documento de exportação *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

25. 5. Faça a build da Morringhan novamente *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

26. 6. Valide o componente, suba o sistema novamente *

Marcar apenas uma oval.

	1	2	3	4	5	
Muit	◯	◯	◯	◯	◯	Muito Fácil

Pós questões

Responda com base nas tarefas e na explicação sobre o sistema que foi dada

27. Você conseguiria dar manutenção no sistema? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

28. As regras são entendíveis. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

29. Você considera o componente: MorLib bem estruturado? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concorodo Fortemente

30. O componente cumpre seu papel *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

31. Você conseguiria acrescentar outra variável que seria responsável pela escolha dos componentes? *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

Comparativo

Agora que você fez tarefas nos dois sistemas, responda:

32. Entre a 1ª versão, EulElement, e a 2ª, Morlib, prefiro a 1ª. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

33. O EulElement cumpre seu papel como componente numa estrutura componentizada *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

34. O EulElement já estava bom o bastante e a refatoração foi desnecessária. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

35. O EulElement é bem comentado. *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

36. O EulElement não tem problemas na estrutura *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

37. O EulElement é entendível e bem organizado *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

38. A MorLib cumpre seu papel como componente numa estrutura de componentizada *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

39. A MorLib é bem comentada *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

40. A MorLib não tem problemas na estrutura *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

41. A MorLib é entendível e bem organizado *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

42. O sistema de regras JSON é melhor que o if/else do EulElement *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

43. As regras são fáceis de alterar *

Marcar apenas uma oval.

	1	2	3	4	5	
Disc	◯	◯	◯	◯	◯	Concordo Fortemente

44. Minha curva de aprendizado foi melhor *

Marcar apenas uma oval.

- ◯ No EulElement
- ◯ Na MorLib

45. Eu prefiro a estrutura, pensando em manutenção *

Marcar apenas uma oval.

- ◯ Do EulElement
- ◯ Da MorLib

46. Eu prefiro a estrutura, pensando em aprendizagem *

Marcar apenas uma oval.

- ◯ No EulElement
- ◯ Na MorLib


ICMC
USP
SÃO CARLOS