Motores de Regras com IBM ODM

O IBM Operational Decision Manager (ODM) é uma plataforma poderosa para automação de decisões empresariais, integrando regras de negócios e análises em tempo real , o IBM ODM é uma ferramenta essencial para automação de decisões, permitindo separar lógica de negócios da programação.

Este artigo inclui exemplos práticos em linguagens de programação como Java, Python e C#.

IBM ODM é uma ferramenta essencial para automação de decisões, permitindo separar lógica de negócios da programação.

Este guia apresenta um fluxo completo de desenvolvimento, publicação e integração com sistemas externos, cobre desde o desenvolvimento de classes e regras até a integração com APIs REST e IBM Business Automation Workflow (BAW).

Ao adotar práticas como modularização do XOM, personalização do BOM e uso eficiente de APIs REST, as empresas podem garantir flexibilidade e agilidade em seus processos decisórios.

1. Introdução ao IBM ODM

O IBM ODM permite às organizações automatizar decisões críticas com base em regras de negócios claras. Seus principais componentes incluem:

1. **Rule Designer:** Ambiente de desenvolvimento baseado no Eclipse, usado para criar e testar regras localmente.
2. **Decision Center:** Plataforma colaborativa para gerenciar e versionar regras.
3. **Decision Server:** Componente de execução que processa regras publicadas e expõe APIs para integração com outros sistemas.

Este documento orienta o desenvolvimento de motores de regras com IBM ODM, apresentando desde a modelagem até testes e integrações.

2. Estrutura do Projeto ODM

2.1. XOM (Execution Object Model)

O XOM é o modelo técnico que define objetos e suas relações. Ele serve como base para o desenvolvimento das regras e geralmente é implementado em Java.

Exemplo de Classe XOM - Cliente:

```
java
package com.ibm.odm;

/**
 * Classe que representa o cliente envolvido no processo de decisão.
 */
public class Cliente {
    private String nome;
    private int idade;
    private double renda;

    // Getters e Setters
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public int getIdade() { return idade; }
    public void setIdade(int idade) { this.idade = idade; }

    public double getRenda() { return renda; }
    public void setRenda(double renda) { this.renda = renda; }
}
```

Exemplo de Classe XOM - Imovel:

```
java
package com.ibm.odm;

/**
 * Classe que representa o imóvel para financiamento.
 */
public class Imovel {
    private String tipo;
    private double valor;
    private int idade;

    // Getters e Setters
    public String getTipo() { return tipo; }
    public void setTipo(String tipo) { this.tipo = tipo; }

    public double getValor() { return valor; }
    public void setValor(double valor) { this.valor = valor; }

    public int getIdade() { return idade; }
    public void setIdade(int idade) { this.idade = idade; }
}
```

2.2. BOM (Business Object Model)

O BOM traduz o XOM em termos mais acessíveis para analistas de negócios, utilizando linguagem natural.
Exemplo de Verbalização:

- Método getNome: "obter o nome do cliente".
- Método getValor: "obter o valor do imóvel".

Boas práticas:

1. Ajuste os termos no BOM para refletir o idioma e a cultura organizacional.
2. Use nomenclaturas consistentes para facilitar a compreensão por usuários não técnicos.

3. Desenvolvimento e Teste de Regras

3.1. Exemplos de Regras

Regra 1: Validação de idade do cliente

```
text
Se a idade do cliente for menor que 18,
Então reprovar o crédito com o motivo "Cliente menor de idade".
```

Regra 2: Aprovação de crédito

```
text
Se o cliente for elegível,
Então aprovar o crédito com o valor financiado.
```

3.2. Testes de Cenário no Rule Designer

Os testes são realizados simulando entradas no Rule Designer.

Exemplo de script de teste:

```
java
com.ibm.odm.Cliente cliente = new com.ibm.odm.Cliente();
cliente.setNome("João");
cliente.setIdade(17);
cliente.setRenda(3000);

com.ibm.odm.Imovel imovel = new com.ibm.odm.Imovel();
imovel.setTipo("Residencial");
imovel.setValor(200000);
imovel.setIdade(10);

return new Object[] { cliente, imovel };
```

Resultados esperados:

- Cliente menor de idade → Crédito reprovado.
- Cliente elegível → Crédito aprovado com observação.

4. Publicação e Gerenciamento de Regras

4.1. Publicação no Decision Center

Após validar as regras no Rule Designer:

1. Conecte o projeto ao Decision Center.
2. Publique o projeto para disponibilizar as regras no Decision Server.
3. Utilize o controle de versão do Decision Center para gerenciar alterações.

5. Integração com IBM BAW

5.1. Configuração no ODM

1. No Decision Server, gere um arquivo de descrição no formato **OpenAPI JSON**.
2. Importe o arquivo JSON no BAW para configurar o serviço REST.

5.2. Configuração no BAW

1. No editor de processos do BAW, adicione um serviço REST.
2. Configure o endpoint usando o arquivo JSON gerado.
3. Teste a integração diretamente no ambiente do BAW.

6. Integração com APIs REST

6.1. Input e Output de Regras

Input (JSON):

```
json
{
  "cliente": {
    "nome": "João",
    "idade": 17,
    "renda": 3000
  },
  "imovel": {
    "tipo": "Residencial",
    "valor": 200000,
    "idade": 10
  }
}
```

Output (JSON):

```
json
{
  "credito": {
    "aprovado": false,
    "observacoes": ["Crédito reprovado: Cliente menor de idade"]
  }
}
```

7. Integração em Linguagens de Programação

7.1. Java

```
java
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class ODMClient {
    public static void main(String[] args) throws Exception {
        String endpoint = "http://<decision-server-
host>:<port>/decisionservice/<ruleapp>/<operation>";
        String payload = "{"
            + "\"cliente\": {\"nome\": \"João\", \"idade\": 17, \"renda\": 3000},"
            + "\"imovel\": {\"tipo\": \"Residencial\", \"valor\": 200000, \"idade\": 10}"
            + "}";

        HttpURLConnection conn = (HttpURLConnection) new URL(endpoint).openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setDoOutput(true);

        OutputStream os = conn.getOutputStream();
        os.write(payload.getBytes());
        os.flush();

        System.out.println("Resposta HTTP: " + conn.getResponseCode());
    }
}
```

7.2. Python

```
python
import requests

url = "http://<decision-server-host>:<port>/decisionservice/<ruleapp>/<operation>"
payload = {
    "cliente": {"nome": "João", "idade": 17, "renda": 3000},
    "imovel": {"tipo": "Residencial", "valor": 200000, "idade": 10}
}
headers = {"Content-Type": "application/json"}

response = requests.post(url, json=payload, headers=headers)
print(response.json())
```

7.3. C#

```
C#
using System;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;

class Program {
    static async Task Main(string[] args) {
        var client = new HttpClient();
        var url = "http://<decision-server-host>:<port>/decisionservice/<ruleapp>/<operation>";
        var payload = @"{
            ""cliente"": {""nome"": ""João"", ""idade"": 17, ""renda"": 3000},
            ""imovel"": {""tipo"": ""Residencial"", ""valor"": 200000, ""idade"": 10}
        }";

        var response = await client.PostAsync(url, new StringContent(payload, Encoding.UTF8,
"application/json"));
        Console.WriteLine(await response.Content.ReadAsStringAsync());
    }
}
```

O IBM ODM é uma ferramenta essencial para automação de decisões, permitindo separar lógica de negócios da programação.

Este guia apresentou um fluxo completo de desenvolvimento, publicação e integração com sistemas externos.

Ao adotar práticas como modularização do XOM, personalização do BOM e uso eficiente de APIs REST, as empresas podem garantir flexibilidade e agilidade em seus processos decisórios.

EducaCiência FastCode para a comunidade