Data sources Converter

André Alexandre dos Santos Vicente
nº: 2181858

André José Rasteiro Carvalho
nº 2181840

André Alberto Paulo Duivenvoorden
nº 2181841

Fábio Da Silva Baptista
nº 2181832

Resumo

Atualmente as aplicações tendem a interligar vários componentes diferentes para as suas implementações, consequentemente surge a unidade curricular Integração de Sistemas.

Nesse sentido, foi desenvolvido um projeto com o propósito de converter o tipo de dados inseridos pelos utilizadores, efetuar regras de processamento e por fim devolver os dados inseridos num tipo de dados diferentes.

Apesar das dificuldades surgidas ao longo do processo, conseguimos realizar todas as funcionalidades impostas para o desenvolvimento deste projeto.

Palavras-chave: componentes, converter, tipo de dados e funcionalidades

I. INTRODUÇÃO

O presente relatório foi realizado no âmbito da unidade curricular de Integração de Sistemas, do curso de Engenharia Informática, tem como objetivo a apresentação e descrição do projeto desenvolvido ao longo do semestre.

Com a realização deste trabalho pretende-se compreender conceitos abordados nas aulas e exploração de novas funcionalidades neste conceito.

Este relatório tem como propósito clarificar aos utilizadores todo o funcionamento da aplicação realizada.

A aplicação tem como funcionalidade principal a tradução de dados, mais especificamente receber dados que o utilizador fornece, aplicar regras de processamento, e depois devolver os dados noutro formato.

Para esta aplicação, nós escolhemos três métodos diferentes onde a aplicação irá obter dados, um pedido a uma RESTFul API, a leitura de um ficheiro XML, e a captura de mensagens num sistema de mensagens, que poderá introduzi-los numa outra RESTFul API, ou em uma tabela num ficheiro HTML.

As fontes e destinos que escolhemos para implementar na aplicação, foram decididos devido ao possível desafio em cada implementação.

II. OBJETIVOS

Com este projeto pretende-se não só aplicar os conhecimentos adquiridos ao longo do semestre, mas também explorar um pouco mais as tecnologias abordadas e o que é possível fazer com as mesmas.

Para tal foi-nos proposto desenvolver uma ferramenta que permita ler diferentes dados de diferentes fontes, disponibilizando-os noutro formato. É importante que um utilizador sem experiência consiga fazer este processo autonomamente.

A. Tecnologias utilizadas

Na realização deste projeto, como referido anteriormente, foram utilizadas diversas tecnologias, das quais se destacam:

- *Markup Languages* - estas foram utilizadas não só com formato de dados de entrada e de saída, mas também na comunicação entre entidades responsáveis pela captura e devolução dos dados.
- Web Services - mais especificamente serviços *RESTFul* utilizados tanto como em fontes de dados, como método de exportação dos mesmos.
- *Messaging Systems* - este sistema foi utilizado apenas como método de captura de dados.

Como base para este projeto foi também utilizada a .NET Framework, a linguagem C# e a ferramenta GIT para o controlo de versões.

B. Bibliotecas utilizadas

No decorrer deste projeto foi necessário utilizar bibliotecas para auxiliar não só no tratamento de dados, mas também no sistema de mensagens.

No âmbito do tratamento de dados recorremos à biblioteca *NewtonsoftJson*, que nos permite não só manipular dados em formato *JSON*, como também transformar dados do tipo *XML* para o tipo *JSON*.

A biblioteca *M2Mqtt* foi principalmente utilizada na captura de mensagens que circulam nos diversos canais de um sistema de mensagens.

III. ARQUITETURA DO SISTEMA

Este projeto está dividido em três partes distintas, sendo a primeira parte o conjunto de entidades que dizem respeito à entrada e tratamento dos dados, a segunda diz respeito ao conjunto das entidades responsáveis pela saída dos dados e por último a atividade principal, que se responsabiliza não só pela configuração de cada uma dessas entidades (entrada e/ou saída), como também pela disponibilização da configuração das mesmas em formato visual. Como é possível observar na *Figura 1*.

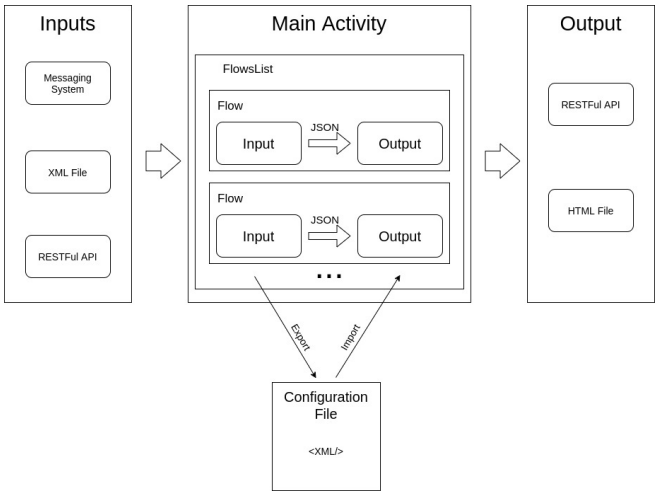

Figura 1 - Arquitetura da aplicação

A. *Atividade Principal*

a. *Componente Visual*

Para este componente foi criado um design o mais intuitivo possível, para que esta aplicação seja fácil de usar por um utilizador com pouca experiência. A *Figura 2* mostra a interface principal.

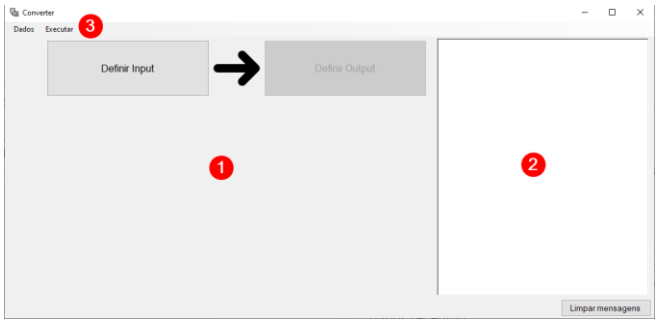

Figura 2 - Interface aplicação

A interface está dividida em 3 zonas:

1) Contém a lista dos *Flows* (será esclarecido na A.b Componente Lógica) que foram definidos e configurados pelo utilizador para interligar diferentes sistemas;
2) Consola responsável por informar o utilizador de eventuais problemas que acontecem durante a execução dos *Flows*, e também algumas mensagens informativas ao estado dos mesmos;
3) Menu de acesso rápido que permite ao utilizador importar/exportar as configurações que fez e executar todos os *Flows*.

Além da interface principal existem também interfaces auxiliares relacionadas com os métodos de entrada e saída de dados. Estas interfaces encarregam-se de preencher os diversos atributos das entidades responsáveis, estas que serão abordadas mais à frente neste relatório.

Na *Figura 3* estão presentes as interfaces responsáveis pela configuração dos métodos de entrada de dados.

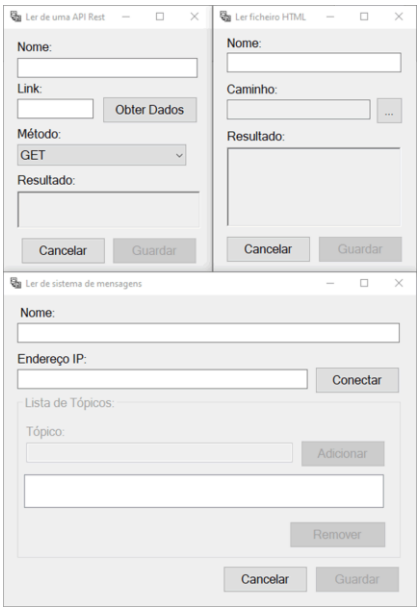

Figura 3 - Interfaces auxiliares para entrada de dados

Na *Figura 4* estão presentes as interfaces responsáveis pela configuração dos métodos de saída dos dados

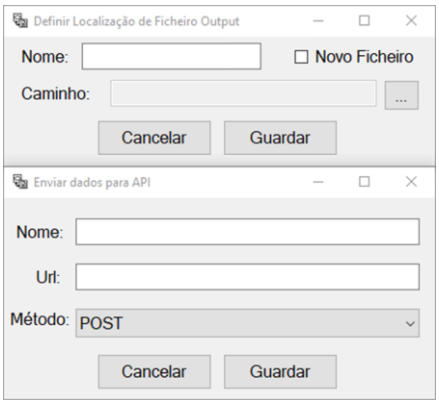

Figura 4 - Interfaces auxiliares para saída de dados

b. *Componente Lógica*

A atividade principal tem também uma componente lógica, que tem como principal objetivo fazer a interligação do processo de captura dos dados, com o processo de disponibilização dos mesmos. Este método é feito através de uma entidade auxiliar *Flow*, que interliga as entidades *Input* e *Output* que representam, respetivamente, o processo de captura e o processo de disponibilização dos dados, os quais serão abordados mais à frente neste relatório. Por sua vez, o conjunto destas entidades auxiliares é armazenado numa estrutura de dados, em conjunto com um número identificador do processo para que seja possível a interligação entre a componente visual com cada um dos *Flows*.

Esta atividade é também responsável pela importação e exportação de configurações de cada uma das entidades envolvidas no processo. Esta importação/exportação é feita para um ficheiro de texto tendo como base uma das *Markup Languages* referidas anteriormente, mais especificamente a linguagem *XML*. No ficheiro é guardado a lista de *Flows* onde cada *Flow* é constituído por um elemento *Input* e por um elemento *Output*, sendo que estes elementos armazenam, no seu interior, dados necessários para o seu correto funcionamento.

B. *Entrada de Dados*

O processo de captura dos dados é feito por três entidades distintas, cada uma destas entidades é responsável por fazer a captura e conversão dos dados para uma *Markup Language*, neste caso o *JSON*, de modo a facilitar o processo de comunicação com as entidades responsáveis pela saída dos dados.

É de notar também que todas as entidades seguem uma estrutura padrão, à qual são posteriormente adicionados atributos e métodos mediante as suas necessidades. Esta estrutura comum é composta por um método que é responsável por executar todas as ações necessárias à captura e conversão dos dados, e por dois atributos, tipo de entrada e nome.

a. ApiRestInput

A entidade *ApiRestInput* tem como único objetivo, efetuar um pedido *HTTP* a um *Web Service RESTFul* e devolver o resultado do mesmo em formato *JSON*. Para tal é necessário definir qual o endereço do servidor que fornece os dados e também qual o método a utilizar (*GET*, *PUT*, *PATCH*, *POST* ou *DELETE*). Tendo em conta que o formato devolvido pelo pedido é o mesmo que o formato utilizado na comunicação entre as entidades, não sendo necessário efetuar qualquer conversão do tipo dados.

b. BrokerInput

A entidade *BrokerInput* tem como função não só conectar-se a um *Messaging System* como também capturar as mensagens provenientes do mesmo. Para tal é necessário definir, tal com a entidade anterior, o endereço do servidor do serviço e também quais os tópicos a serem subscritos no *Messaging System*. Posteriormente, após a ligação com o serviço ser efetuada, as mensagens recebidas são convertidas para formato *JSON*, sendo que o objeto resultante dessa conversão agrega apenas o nome do tópico onde a mensagem foi recebida e o conteúdo da mesma.

c. XmlFileInput

A entidade *XmlFileInput* tem por base a leitura de um ficheiro de texto em formato *XML*. Para tal é necessário definir qual o caminho do ficheiro a ser lido para, posteriormente, o conteúdo do ficheiro ser convertido para o formato *JSON*. Este processo de conversão é auxiliado pela biblioteca *NewtonsoftJson* que nos fornece um algoritmo de conversão de dados do tipo *XML* para dados do tipo *JSON*.

C. *Saída de Dados*

Após a captura e processamento dos dados é necessário devolvê-los ao utilizador num formato por ele escolhido. Para isso surgem duas novas entidades que se responsabilizam por receber os dados em formato *JSON* e devolvê-los no formato escolhido.

a. ApiRestOutput

À semelhança da entidade *ApiRestInput*, esta entidade também é responsável por efetuar um pedido *HTTP* a um *Web Service RESTFul*, mas com a nuance de nesse pedido ser enviado o conteúdo recebido por parte das entidades responsáveis pela captura dos dados. É de notar que sendo a comunicação entre as entidades feita em formato *JSON*, não existe a necessidade de voltar a processar os dados, visto que, estes são enviados no pedido *HTTP* no mesmo formato.

b. FileHtmlOutput

A entidade *FileHtmlOutput* tem como objetivo a criação de um ficheiro de texto do tipo *HTML*, apresentando assim os dados em forma de tabela.

Esta entidade opera em dois modos diferentes, o modo de reescrita e o modo de adição. O modo de reescrita. caso o ficheiro já exista, os dados nele presentes serão apagados para que os novos dados sejam escritos. No modo de adição, os novos dados são acrescentados no fim desse ficheiro.

Esta entidade começa por criar a estrutura base de um documento *HTML* e adicionar no corpo do documento, uma tabela onde a estrutura varia mediante o tipo de dados fornecidos. Para a construção dessa tabela é utilizada uma função recursiva que se responsabiliza por decidir, consoante o tipo de dados fornecidos, qual o elemento *HTML* a ser utilizado para apresentar esse tipo de dados.

IV. PROCESSO DE DESENVOLVIMENTO

O processo de desenvolvimento deste projeto foi dividido em três fases, a planificação, o desenvolvimento das entidades responsáveis pela entrada e saída de dados e por último, mas não menos importante a interligação com a componente visual.

Inicialmente foi necessário esquematizar e definir que entidades iriam ser necessárias para o processo de captura e exportação dos dados. Do qual resultou diagrama de classes da *Figura 5*.

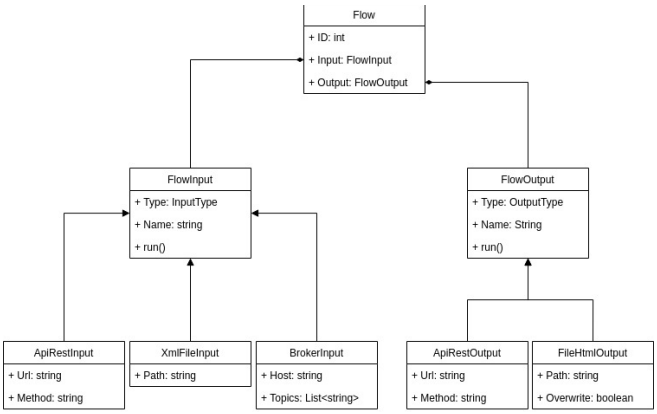

Figura 5 - Diagrama UML de classes

No diagrama podemos verificar que a entidade principal é a entidade *Flow*. Esta entidade é responsável por interligar um método de captura de dados com um método de exportação do mesmo, tendo também um identificador único de forma a facilitar o mapeamento com a componente visual.

Um pouco mais abaixo no diagrama encontramos a entidade *FlowInput* que se responsabiliza pela captura dos dados. Com descendentes desta entidade vamos ter as entidades *ApiRestInput*, *XmlFileInput* e *BrokerInput*.

Ao lado da entidade *FlowInput* temos a entidade *FlowOutput*. Esta entidade é responsável pela exportação dos dados tendo com seus descendentes, as entidades *ApiRestOutput* e *HtmlFileOutput*.

V. Dados utilizados para Desenvolvimento

Durante o desenvolvimento do projeto foram utilizadas várias fontes para obter os dados. Algumas foram disponibilizadas na entrega do projeto.

Para a sua utilização deverá fazer o seguinte:

Inputs
ApiRest
Dados da aplicação *Shush* (projeto desenvolvido em TAES): https://us-central1-taes-shush-b.cloudfunctions.net/getAllData
Dados *JSON* aleatórios: https://jsonplaceholder.typicode.com/posts
Dados locais:
Utilizar o servidor *NodeJS* local localizado na pasta Dummy Data Sources\fake-http-server-nodejs
Utilizar o endereço: GET http://127.0.0.1:8080
XmlFile
Ficheiros *XML* localizados na pasta Dummy Data Sources\XMLFiles
BrokerInput
Iniciar o projeto dummy-MQTTPublisher disponibilizado na solução
Outputs
ApiRest
Servidor local NodeJS: POST http://127.0.0.1:8080 (Utilizar o mesmo servidor do ApiRestInput)

VI. Conclusões e trabalho futuro

Após o desenvolvimento desta aplicação chegámos à conclusão que esta ferramenta, apesar de parecer muito simples, é bastante complexa de implementar devido à grande variedade de formatos de dados.

Estes tipos de aplicações podem ser bastante úteis em situações reais, em que é necessário efetuar a recolha de dados de vários tipos de sistemas e transformá-los num formato único, facilitando assim a sua análise.

Como trabalho futuro poderiam ser adicionadas mais formas de obter dados e de os devolver, aumentando assim a versatilidade da aplicação e por consequência atingindo um maior número de utilizadores.

Em suma, consideramos que tiramos um bom aproveitamento deste trabalho, angariando assim mais experiência na área de Integração de Sistemas e satisfatoriamente conseguimos realizar todas as funcionalidades impostas para o desenvolvimento deste projeto.

Apêndice

No decorrer do desenvolvimento da aplicação fizemos a divisão de tarefas da seguinte forma:

- Ambos os *Outputs* (*APIRestOutput* e *FileHtmlOutput*) foram desenvolvido do André Vicente, também alguns elementos visuais;
- Os *Inputs XmlFileInput* e *BrokerInput*, e o mecanismo da consola foram responsabilidade do André Carvalho, com alguns elementos visuais;
- O *Input ApiRestInput*, importação das configurações do ficheiro *XML* e outros formulários de dados foram responsabilidade do Fábio Baptista;
- A parte visual da *Dashboard*, a exportação das configurações para ficheiro *XML*, e a interligação entre todos os componentes (*input*/*output*) foi responsabilidade do André Paulo.