**Manual de Manutenção e Implementação de Processos no Novo Layout Variável**

1. **Histórico e Justificativa da Necessidade**

Alguns produtos e operações de negócios da MAPFRE demandam o processamento de quantidades massivas de dados, originárias de inúmeras corretoras de seguro, clientes e parceiros, e com destinação aos diversos sistemas da companhia. Boa parte destas transações são operacionalizadas através da troca de arquivos entre estes atores e a própria MAPFRE, utilizando-se formatos e layouts de arquivos tão diversos quanto a quantidade destes canais de origem.

Dado este cenário, é fator crítico de sucesso que se estabeleça a gestão efetiva sobre os layouts, regras de validação e processamento dos arquivos, através de uma ferramenta dinâmica e eficiente. Esta ferramenta agregará aos sistemas da companhia um canal de processamento de dados em lote, com efetivo controle sobre formatos, layouts e processamento dos dados.

1. **Objetivos deste Documento**

Documentar os requisitos não funcionais levantados, fazendo uma breve descrição destes.

Documentar as decisões de arquitetura da solução explicando, em linhas gerais, as responsabilidades e a interação entre os componentes e como estes implementam os requisitos não funcionais.

Permitir que a solução possa ser mantida e estendida, afim de acoplar e novos modelos de serviços.

Documentar o que está pendente de implementação e o que pode ser melhorado na solução atual.

1. **Requisitos Macro da Solução**
   1. Gerenciamento de Layouts
      1. Ler arquivos XLSX, XLS e TXT massivos.

Ser capaz de processar arquivos XLSX, XLS e TXT, lendo-os com baixo footprint de memória, independente do tamanho do arquivo lido.

* + 1. Mapear Serviços de Destino e Seus Parâmetros

Ser capaz de mapear os serviços de destino dos dados, registrando os parâmetros de entrada, seus formatos e validações básicas.

* + 1. Mapear Colunas de Origem nas Planilhas

Mapear as colunas dos arquivo Excel ou TXT para os parâmetros dos serviços de destino, estabelecendo as conversões e formatações necessárias.

* + 1. Converter Dados

Permitir estabelecer no layout as regras e formatos para conversão de dados.

* + 1. Formatar e *Parsear* Dados

Permitir determinar no layout os formatos para *parseamento* e formatação de dados.

* 1. Execução Processos
     1. Particionamento e Paralelismo

Permitir particionar a carga de processamento em múltiplas linhas de processamento paralelas, na mesma máquina (implementado) ou de forma distribuída (não implementado).

* + 1. Performance.

Criar um mecanismo de controle e execução de processos com o mínimo uso de recursos para particionar, paralelizar e registrar resultados do processamento dos dados.

* + 1. Controle de “*throttling*”.

Permitir determinar efetivamente qual será a quantidade de recursos computacionais utilizados para processamento dos dados, criando um mecanismo estável e previsível.

* + 1. Controle de Processamento.

Permitir estabelecer políticas de cancelamento do processo, em função da quantidade de erros de validação ou de processamento dos dados (parcialmente implementado).

Permitir controlar o processo de execução do arquivo em tempo real, podendo aumentar ou reduzir os recursos disponíveis de acordo com uma janela de dias e horários (não implementado).

* + 1. Registro de Resultados.

Permitir registrar o resultado do processamento dos arquivos, independente do tipo de resultado, sucesso ou falha.

* 1. Conectores para Componentes de Negócio
     1. Processamento em Lote

Criar um modelo de abstração de conectores que permita acoplar novos serviços de processamento de dados ao layout variável.

* + 1. Controle Transacional

Determinar, para cada serviço, a política de controle transacional.

1. **Componentes da Arquitetura do novo LV**

A description...

No diagrama acima as dependências e responsabilidades de cada grupo de componentes estão agrupadas por cor.

**Leitura de Arquivos (componentes destacados em verde):**

Os componentes destacados em verde são a interface principal do framework, que permite que outros componentes ou sistemas acionem o LV. Estes componentes são responsáveis por iterar os registros de arquivos (Excel), utilizando uma implementação de leitura de arquivos XLSX com SAX, o que garante o baixo footprint de memória, mesmo para arquivos Excel com centenas de mega bytes de tamanho.

br.com.mapfre.lv.jobmanager.XLSXSAXHandler: Classe implementação de org.xml.sax.helpers.DefaultHandler, contém a implementação dos eventos de leitura XML SAX, e lógica para agrupar os registros da planilha e acionar os componentes destacados em azul, responsáveis pelo particionamento e paralelização do processamento.

No XML que representa os dados e fórmulas contidos numa planilha, o elemento “<v>...</v>” é onde ficam armazenados os valores das células. Numa célula numérica ou tipo data o valor é representado por um double. Adicionalmente a este double, o excel armazena na célula qual é o tipo do dado (inteiro, monetário, data) e seu formato (dd/mm/aaaa, R$#,## etc). Para armazenar valores literais, como um nome ou endereço, o Excel utiliza uma tabela de strings, que permite que estes literais sejam armazenados sem duplicidade, economizando no tamanho dos arquivos.

Por exemplo:

Para armazenar o valor string “Av. Paulista”: <v>321</v>, onde 321 é a chave para uma tabela de strings que conterá o valor “Av. Paulista”. Toda vez que esta String aparecer na planilha, na verdade o Excel estará armazenando na célula a chave 321 desta tabela.

Para visualizar como funciona a leitura de valores das células, definição de tipo de dado, formatação e tabelas de strings, verificar método:

br.com.mapfre.lv.layoutmodel.AbstractSAXHandler.XLSXSAXHandler.getVal.

Uma vez lido o valor da célula, a classe XLSXSAXHandler aciona um componente implementação de ILayoutParser, destacados no diagrama em cinza. Estes componentes irão detectar qual é o valor e formato esperado, fazer validações e conversões se necessário etc, aplicando o layout definido pelo usuário. Maiores detalhes sobre a aplicação do Layout definido pelo usuário a seguir, em “Interpretação do Modelo de Layout”.

Cada linha lida da planilha é armazenada num objeto do tipo PayloadVO. Um lote de registros é representado por objetos do tipo LotVO, que contém uma coleção de PayloadVOs.

Após ler um conjunto de linhas e completar um lote de registros (LotVO), o XLSXSAXHandler irá acionar um componente implementação de IExecutorMaster, que será responsável pelo processamento do referido lote de registros. Maiores detalhes a seguir em “Particionamento e Paralelização de Processamento”.

**Interpretação do Modelo de Layout (Componentes destacados em Cinza):**

Os metadados para definição de layouts são definidos pelas entidades persistentes destacadas no diagrama em cinza escuro. Neste modelo, cada layout (LayoutModelVO) possui um conjunto de definições de colunas (LayoutColumnVO) e cada coluna deve estar mapeada para um parâmetro de entrada (ServiceParamVO) do acionamento de um serviço (ServiceModelVO).

O método LayoutColumnVO.getObjectValue é o coração da aplicação do layout e das formatações.

A interface ILayoutParser é o contrato principal do componente que aplica regras de layout, tendo como implementação a classe XLSXLayoutParser.

O XLSXLayoutParser lê os metadados referentes ao layout escolhido pelo usuário, contidos em LayoutModelVO, encontra das definições para cada coluna da planilha e aplica as regras de mapeamento, validação e formatação contidas no modelo, já atribuindo o valor a um mapa contido no PayloadVO informado.

**Particionamento e Paralelização de Processamento (Componentes destacados em Azul):**

Estes componentes são responsáveis pela paralelização, particionamento do processamento dos dados e controle da alocação de recursos computacionais.

A interface IExecutorMaster é o contrato de utilização destes componentes e a sua implementação ThreadPoolExecutorMaster utiliza pool de threads para implementar a paralelização e controle do processamento dos lotes de dados.

Este componentes utiliza a classe do pacote concurrent de java java.util.concurrent.ThreadPoolExecutor, que permite ter total controle sobre o tamanho do pool de threads, tamanho da fila de threads e ainda possui um mecanismo interno de semáforo que mantém todo o enfileiramento em “*wait*” até que a fila do pool tenha capacidade de receber mais threads.

A capacidade de manter o enfileiramento do pool em “*wait*” segura também a leitura do arquivo e a alocação de dados em memória. O tunning do pool de threads é fundamental para performance e resiliência da solução. As propriedades a seguir de JobManagerXLSXEventModel são utilizadas no tunning do pool de threads:

// TUNNING DA LEITURA DO ARQUIVO

**private** **int** lotBufferSize=45; // Quantidade de linhas do arquivo para cada lote de registros a serem processados por thread.

// CONFIGURAÇÕES DO POOL DE THREADS \*\*\* PONTO CHAVE DA ARQUITETURA \*\*\*

// IllegalArgumentException - if corePoolSize or keepAliveTime less than zero, or if maximumPoolSize less than or equal to zero, or if corePoolSize greater than maximumPoolSize.

**private** **int** corePoolSize =50; // the number of threads to keep in the pool, even if they are idle.

**private** **int** maximumPoolSize = 60 ; // the maximum number of threads to allow in the pool.

**private** **int** queueCapacity =50 ; // Capacidade máxima de Threads em estado idle. "Trava" a inclusão de novas threads, até que as threads executantes terminem.

**private** **long** keepAliveTime = 4000;// when the number of threads is greater than the core, this is the maximum time that excess idle threads will wait for new tasks before terminating.

**private** TimeUnit threadExecutorTimeoutUnit = TimeUnit.*SECONDS*;

**private** **int** threadExecutorTimeout = 5000; // tempo maximo em segundos que o pool de threads irá esperar o término da execução das threads faltantes, uma vez acionado o método ThreadPoolExecutor.awaitTermination

Este componente é a pedra angular do framework, que garante a máxima eficiência de uso de recursos computacionais e o baixo footprint de memória para processamento de arquivos com centenas de megas ou até gigas de dados.

O ThreadPoolExecutorMaster (ver método “execute”) cria threads do tipo IExecutorSlave, passando o lote a ser processado, a bridge e os demais dados necessários para processamento.

**Conector para Componentes de Negócio (Componentes destacados em vermelho)**:

O objetivo desta família de componentes é fazer a ponte entre o mecanismo de processamento do Layout Variável e os componentes ou tabelas de destino dos dados.

A implementação de IBridge, referente ao serviço de destino do processamento, irá receber o lote de dados (LotVO) como sua carga de trabalho. Na prática este componente deve converter cada PayloadVO, contido no lote de dados, numa chamada para um componente de negócio, da maneira mais efetiva e performática possível.

A implementação de IBridge.execute deve retornar um array de inteiros. Com este array a implementação de IExecutorSlave irá acionar o IProcessMonitor para registro de resultados do lote.

É recomendado que se utilize processamento em lote com JDBC puro, o que reduz drásticamente o overhead de uso do banco de dados, e ainda gera um array de resultados compatível com o retorno esperado pela IBridge, mesmo que haja erro durante o processamento em lote.

Lembrando que a exceção java.sql.BatchUpdateException contém os resultados referentes a cada instrução SQL informada.

**Monitoramento do Processo (Componentes destacados em amarelo):**

A interface deste grupo de componentes é IProcessMonitor, que possui um método para cada evento relevante do processamento de um arquivo de dados e seus lotes, cobringo, inclusive, eventos de falha.

Para maiores detalhes, ver JavaDoc da interface IProcessMonitor.

Sua implementação PersistentProcessMonitor realiza o contrato IProcessMonitor persistindo os registros de eventos de processamento de arquivos e lotes, inclusive tratando situações de erro e de rejeição de lotes de arquivos.

Um único PersistentProcessMonitor deve ser instanciado e passado para as threads de processamento dos dados. Esta classe é thread safe pode ser acionada simultaneamente por qualquer thread do processo, afim de registrar o início, fim ou falhas no processamento de dados.

O PersistentProcessMonitor está preparado para implementar políticas de cancelamento do processamento de um arquivo por quantidade de erros de validação, erros de processamento etc.

O monitoramento possui um modelo persistente capaz de registrar os eventos referentes a arquivos de dados, lotes de registros e faltas encontradas durante o processamento.