

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
CURSO DE SISTEMAS DE INFORMAÇÃO**

André Luis Zorzo

**ETL 2.0 – Uma proposta de extensão ao processo de extração,
transformação e carga voltada à integração de dados estruturados
e não estruturados**

**Florianópolis
2009**

André Luis Zorzo

**ETL 2.0 – Uma proposta de extensão ao processo de extração,
transformação e carga voltada à integração de dados estruturados
e não estruturados**

Trabalho apresentado ao Curso de Sistemas de Informação da Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do título de Bacharel em Sistemas de Informação.

Prof. José Leomar Todesco – orientador

Prof. Alexandre Leopoldo Gonçalves – coorientador

Prof. Frank Augusto Siqueira – banca

Florianópolis

2009

RESUMO

O rápido crescimento de fontes de informação não estruturadas nas organizações traz muitos desafios para a análise desses dados, que vão desde a coleta e a organização, até a integração com fontes estruturadas, visando apoiar de maneira sistêmica os processos de tomada de decisão. Entretanto, a tecnologia predominante atualmente para elaboração de ambientes/aplicações decisórios é o Data Warehouse (DW), que analisa dados estruturados. Uma nova geração de DW propõe a utilização também dos dados não estruturados, mas não aborda de maneira adequada o processo de Extração, Transformação e Carga (ETL – *Extraction, Transformation and Load*). Este trabalho propõe uma extensão ao processo de ETL tradicional, integrando as duas visões (estruturada e não estruturada) em um modelo genérico para apoiar a tomada de decisão. Para atingir esses objetivos, foram desenvolvidas extensões a uma ferramenta de ETL de código aberto. Posteriormente, a ferramenta foi utilizada sobre dados reais para suportar o processo de ETL, produzindo como resultado um DW. Analisando-se os dados inseridos no DW a partir do processo completo, foi possível encontrar informações de correlação entre entidades pertencentes a ambas as classificações de dados. A principal contribuição do trabalho reside na extensão ao processo de ETL tradicional e na proposição, ainda que inicial, de um modelo de DW genérico para análise de relações entre entidades que promovem suporte a diversos cenários de tomada de decisão.

Palavras-chave: Extração, Transformação e Carga. Data Warehouse. Integração de Dados. Fontes de Informação Estruturada e Não Estruturada. Extração de Informação.

ABSTRACT

The fast growth of non structured information sources in enterprises brings many challenges, which go from crawling to organizing, and to the integration with structured sources, aiming to support decision making. However, the currently predominant technology for decision making systems/applications creation is the Data Warehouse (DW), which analyses structured data. A new generation of DW also considers the use of non structured data, but it does not approach in an adequate way the Extraction, Transformation and Load (ETL) process. This work proposes an extension to the traditional ETL process, integrating the two visions (structured and non structured) in a generic model to support decision making. To reach these objectives, extensions to an open source ETL tool have been developed. Later, the tool was used over real data to support the ETL process, producing a DW as the result. Analyzing the inserted data in the DW from the complete process, it was possible to find information and correlation from entities belonging to both data classifications. The main contribution from this work is in the extension of the traditional ETL process and in the proposal, despite initial, of a generic model of DW to analyze the relation between entities that promote support to the diverse scenarios of decision making.

Keywords: Extraction Transformation and Load. Data Warehouse. Data Integration. Structured and Non Structured Information Sources. Information Extraction.

LISTA DE FIGURAS

Figura 1 - Fluxo dos dados em um processo de extração, transformação e carga e algumas operações que são executadas durante o mesmo.	11
Figura 2 - Metodologia aplicada para o desenvolvimento do trabalho	14
Figura 3 - Modelo de dimensões compartilhadas.....	25
Figura 4 - Exemplo de transformação sendo construída com a ferramenta Kettle	36
Figura 5 - Arquitetura de um DW 2.0: diferentes tipos de dados, suas estruturas básicas e como os dados se relacionam.....	39
Figura 6 - Ciclo de vida dos dados em um DW 2.0	42
Figura 7 - Relacionamento dos metadados na organização	43
Figura 8 - Proposta para o processo de ETL 2.0.....	45
Figura 9 - Modelo do DW	48
Figura 10 - Configuração do plugin “Collector”	56
Figura 11 - Configuração do LuceneOutputPlugin	59
Figura 12 - Carga da dimensão entidade (usuários)	63
Figura 13 - Carga do fato relação.....	65
Figura 14 - Análise de relação entre o termo bankrupt e pessoas	68
Figura 15 - Análise de relação entre o termo bankrupt e departamentos.....	68
Figura 16 - Análise da frequência dos termos bankrupt, profit e payment distribuídos por tempo	70

LISTA DE TABELAS

Tabela 1 - Lista dos arquivos necessários para o plugin Collector.....	55
Tabela 2 - Lista dos arquivos necessários para o plugin LuceneOutputPlugin	58
Tabela 3 - Lista dos arquivos necessários para o plugin LuceneInputPlugin	61
Tabela 4 - Exemplo de valores utilizados na carga da dimensão tempo.....	62
Tabela 5 - Exemplo de valores dos termos utilizados na carga da dimensão entidade	64
Tabela 6 - Exemplo de valores utilizados na carga do fato frequência.	66

SUMÁRIO

1	INTRODUÇÃO	10
1.1	OBJETIVO GERAL.....	11
1.2	OBJETIVOS ESPECÍFICOS	12
1.3	JUSTIFICATIVA.....	12
1.4	METODOLOGIA	13
1.5	ORGANIZAÇÃO DO TRABALHO	15
2	FUNDAMENTAÇÃO TEÓRICA	16
2.1	DATA WAREHOUSE	16
2.1.1	<i>Histórico</i>	<i>16</i>
2.1.2	<i>Variações de Data Warehouses.....</i>	<i>18</i>
2.1.3	<i>Business Intelligence (BI).....</i>	<i>21</i>
2.2	PROCESSO DE ETL	26
2.2.1	<i>Modos de processo de ETL</i>	<i>27</i>
2.2.2	<i>Dados estruturados e não estruturados</i>	<i>28</i>
2.2.3	<i>Representação de conhecimento.....</i>	<i>29</i>
2.2.4	<i>Extração de Informação</i>	<i>31</i>
2.2.5	<i>Ferramentas atuais</i>	<i>35</i>
2.3	A NOVA GERAÇÃO DE DW (DW 2.0)	38
2.3.1	<i>Ciclo de vida de um DW 2.0.....</i>	<i>40</i>
2.3.2	<i>Metadados e DW 2.0</i>	<i>42</i>
2.3.3	<i>DW 2.0 e dados não estruturados.....</i>	<i>44</i>
2.4	CONSIDERAÇÕES FINAIS	44
3	PROCESSO PROPOSTO	45
3.1	PROCESSO DE ETL	45
3.2	COLETA.....	46
3.3	BASE DE TERMOS.....	46
3.4	INTEGRAÇÃO.....	47
3.5	MODELO DO DW	47
3.5.1	<i>Dimensão Classe</i>	<i>48</i>
3.5.2	<i>Dimensão Entidade.....</i>	<i>49</i>

3.5.3	<i>Dimensão Tempo</i>	49
3.5.4	<i>Fato Frequência</i>	49
3.5.5	<i>Fato Relação</i>	50
3.6	ANÁLISE	50
3.7	CONSIDERAÇÕES FINAIS	51
4	APRESENTAÇÃO DOS RESULTADOS	53
4.1.	FONTES DE INFORMAÇÃO PRIMÁRIAS E SECUNDÁRIAS.....	53
4.1.1	<i>Base de dados utilizada</i>	53
4.1.2	<i>Modelo do DW</i>	54
4.2.	COMPONENTES TECNOLÓGICOS DESENVOLVIDOS	54
4.2.1	<i>Collector</i>	55
4.2.2	<i>Lucene Output Plugin</i>	57
4.2.3	<i>Integração (LuceneInputPlugin)</i>	60
4.3.	APRESENTAÇÃO SOBRE O PROCESSO ETL.....	62
4.3.1	<i>Dimensão Tempo</i>	62
4.3.2	<i>Dimensão Classe</i>	62
4.3.3	<i>Dimensão Entidade</i>	62
4.3.4	<i>Fato Relação</i>	64
4.3.5	<i>Fato Frequência</i>	66
4.4.	DISCUSSÃO DOS RESULTADOS	66
4.4.1	<i>Primeira análise</i>	67
4.4.2	<i>Segunda análise</i>	69
4.5.	CONSIDERAÇÕES FINAIS	70
5	CONCLUSÕES	71
5.1	TRABALHOS FUTUROS	72
	REFERÊNCIAS.....	73
	APÊNDICE 1 - BASE DE TERMOS.....	76
	APÊNDICE 2 - CÓDIGO-FONTE DO COLLECTORPLUGIN.....	77
	APÊNDICE 3 - CÓDIGO-FONTE DO LUCENEINPUTPLUGIN.....	106
	APÊNDICE 4 - CÓDIGO-FONTE DO LUCENEOUTPUTPLUGIN.....	121
	APÊNDICE 5 - UPDATEFATOFREQUENCIA.JAVA	153

1 INTRODUÇÃO

O processo de Extração, Transformação e Carga (ETL – *Extraction, Transformation and Load*) pode ser considerado o estágio mais importante na construção de um Data Warehouse (KIMBALL, 2004). Nessa etapa, podem ser gastos 40% do custo para construção total de um Data Warehouse (BERNSTEIN; HAAS, 2008). Existem diversos estudos sobre o processo de ETL, que de certa forma é uma técnica já bem conhecida. Porém, o uso de dados não estruturados nesse processo dificilmente é encontrado na maioria desses estudos.

Cada vez mais as fontes de dados não estruturadas têm sido utilizadas para apoio à tomada de decisão. Coletar os dados de fontes não estruturadas e organizá-los em uma estrutura que possa apoiar a extração de informações úteis para a tomada de decisão é o desafio da próxima geração de Data Warehouse, chamada DW 2.0.

A primeira geração de Data Warehouses (DW), preocupada em analisar os dados operacionais das grandes organizações de forma que pudesse apoiar o processo de tomada de decisão, definiu o processo de ETL (*extraction, transformation and load*) somente para os dados estruturados, o que já cumpria com os seus objetivos. Entretanto, a nova geração de DW, conhecida como DW 2.0, prevê o uso dos dados não estruturados nesse mesmo processo.

Na figura apresentada a seguir observam-se o processo de ETL e algumas operações que podem ser executadas sobre os dados para adequá-los a um sistema de DW. A utilização dos dados não estruturados nesse processo não é tarefa fácil, visto que não existe uma forma comum e simples nem mesmo para a própria extração desses dados. Ainda é necessária a integração entre os dados estruturados e não estruturados para permitir a análise de ambos em conjunto.

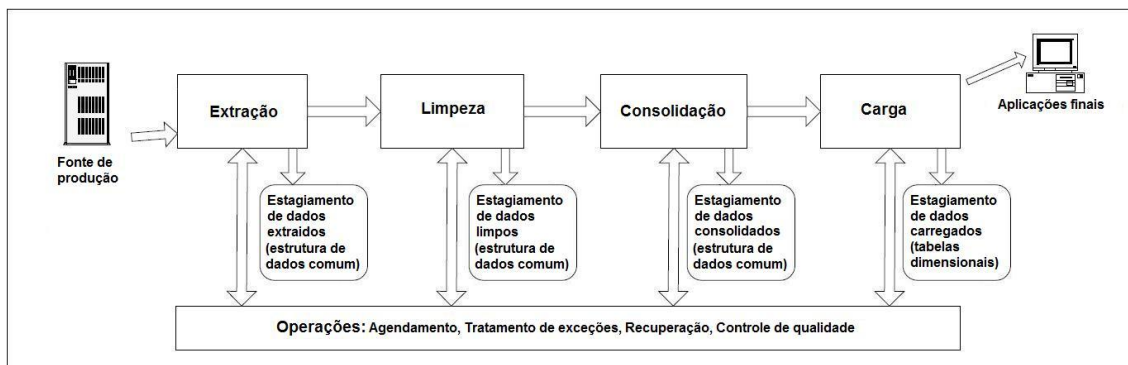


Figura 1 - Fluxo dos dados em um processo de extração, transformação e carga e algumas operações que são executadas durante o mesmo.

Fonte: adaptada de Kimball (2004)

A maioria das ferramentas com o objetivo de apoiar o processo de ETL disponíveis no mercado não provê meios para auxiliar no processo de extração de padrões a partir de informações textuais. Por esse motivo, o uso de documentos não estruturados para auxiliar a tomada de decisão é pouco difundido, em comparação com os sistemas de *Business Intelligence* tradicionais (TRUJILLO; SONG, 2008).

Existem diversos estudos quanto à análise de dados não estruturados, mas dificilmente esses estudos preocupam-se especificamente com a etapa de transformação dos dados para que possam ser feitas análises sobre estes. Inclusive, um dos maiores desafios para o processo de ETL sobre dados não estruturados consiste em executar o mesmo fluxo utilizado sobre os dados estruturados até chegar ao DW (DAYAL, 2009).

Considerando-se essas dificuldades é que se propõe uma extensão ao processo de ETL atual para que sejam considerados os dados não estruturados no mesmo processo. Com isso, será possível tratar esses dados da mesma forma que os dados estruturados.

1.1 Objetivo geral

O objetivo geral deste trabalho é propor uma extensão ao processo de ETL (Extração, Transformação e Carga) que integre dados estruturados e não estruturados visando apoiar a tomada de decisão.

1.2 Objetivos específicos

- Apresentar as principais abordagens teóricas do processo ETL para dados não estruturados.
- Identificar e especificar atividades necessárias para tratamento dos dados não estruturados no processo de ETL.
- Utilizar técnicas da análise de informação textual para auxiliar o processo de aquisição dos dados que irão compor o modelo dimensional.
- Implementar componentes adicionais para uma ferramenta de ETL *open source* que possibilite a execução das etapas de integração entre dados estruturados e não estruturados.
- Avaliar o processo por meio de um estudo de caso que demonstre a viabilidade de se integrarem dados estruturados e não estruturados em um Data Warehouse.

1.3 Justificativa

Ferramentas que apoiam e melhoram a tomada de decisão de negócios são conhecidas como ferramentas de Business Intelligence (BI) (ELBASHIR; COLLIER; DAVERN, 2008). A primeira definição de BI, que surgiu em 1958, descreve um sistema que utiliza processamento de dados para autoabstrair e autocodificar documentos e criar perfis de interesse para cada “ponto de ação” em uma organização. Tanto documentos internos quanto externos são automaticamente abstraídos, caracterizados por um padrão e enviados automaticamente ao seu ponto de ação apropriado (GRIMES, 2008).

Porém, ao longo dos anos, essas ferramentas trataram apenas dos dados operacionais das organizações, ignorando o foco inicial de BI, o qual se concentrava na extração, na categorização e na classificação de padrões textuais em vez de aglomerar dados numéricos. O motivo disso é óbvio: as organizações acumularam muitos dados que eram (e são) ligados diretamente às operações de negócios.

Cada vez mais as fontes de dados não estruturados têm sido utilizadas para a descoberta de conhecimento e apoio à tomada de decisão. As organizações

possuem gigantescos volumes de textos, e outras informações não estruturadas estão disponíveis na internet. No foco original de BI, os documentos eram ignorados.

As ferramentas de BI utilizam como sua principal fonte de análise bancos de dados especializados, tais como Data Warehouses (DW) (ELBASHIR; COLLIER; DAVERN, 2008). Mas os DWs também não estão preparados para trabalhar com as fontes de informação não estruturadas (INMON, 2008).

Segundo Inmon (2008), coletar os dados de fontes não estruturadas e organizá-los em uma estrutura que possa apoiar a extração de informações úteis para a tomada de decisão constituem o desafio da próxima geração de Data Warehouse. As soluções de mercado atuais tratam, em geral, o processo de ETL somente dos dados estruturados. Algumas ferramentas atuais, como a *SAP Data Integration e Informatica 9*, permitem a extensão de suas funcionalidades, o que possibilita ao processo de ETL a utilização de fontes de dados não estruturados. Entretanto, isso ainda está em fase inicial, inclusive em ferramentas *open source*, ponto com o qual o presente trabalho objetiva contribuir.

1.4 Metodologia

O trabalho dividiu-se em quatro etapas, mostradas na Figura 2. Primeiramente, foi feita uma revisão da literatura sobre os desafios para se integrarem dados estruturados e não estruturados ao processo de ETL, propondo-se uma extensão ao processo de ETL para permitir a integração dos dois tipos de dados. Posteriormente, foram desenvolvidas extensões para uma ferramenta de código aberto de modo que o processo proposto pudesse ser executado. Por fim, foram executados todos os passos do processo de ETL com dados reais para posterior análise dos resultados.

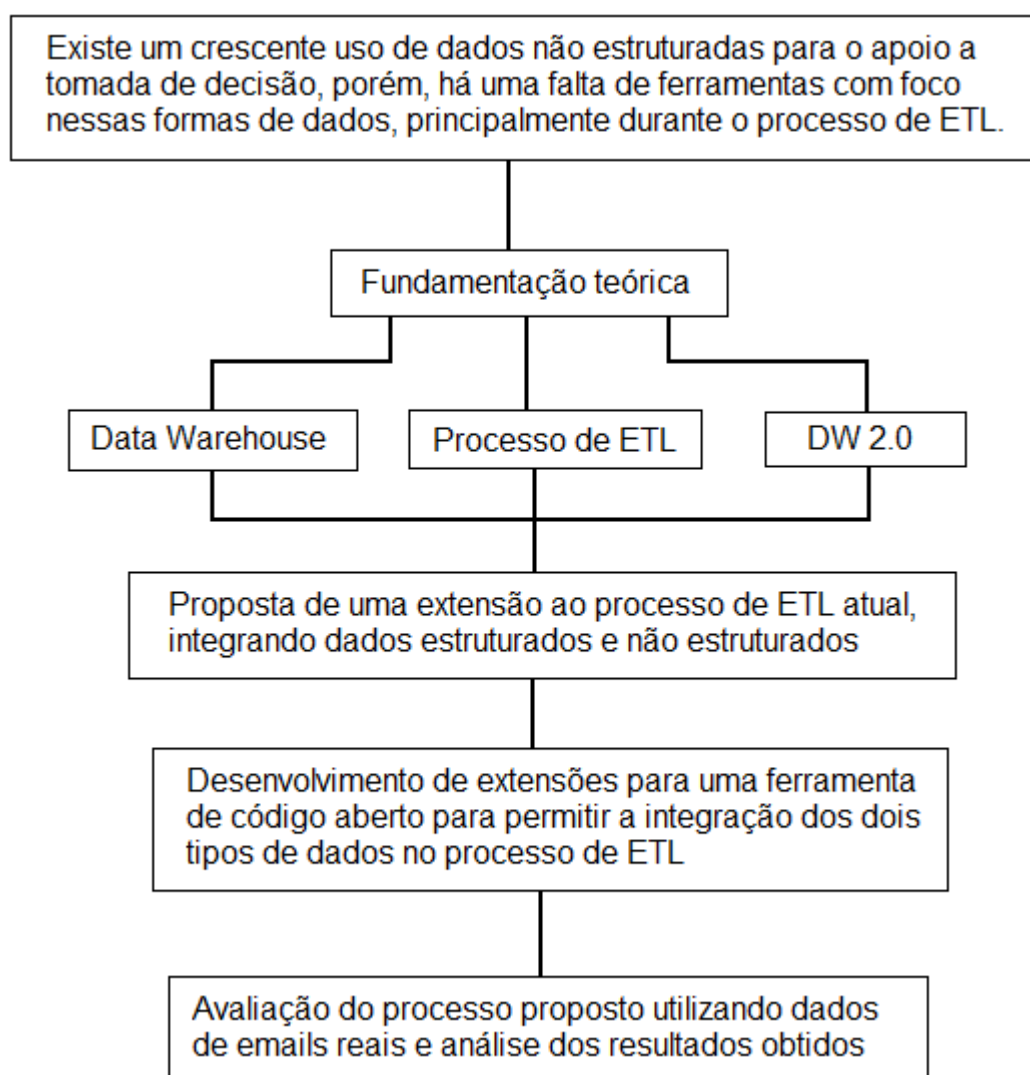


Figura 2 - Metodologia aplicada para o desenvolvimento do trabalho

Fez-se uma análise dos desafios da integração de dados estruturados e não estruturados no processo de ETL e de possíveis soluções de mercado que solucionem o problema. Essa análise foi feita para se obter o conhecimento necessário à concepção de um modelo que pudesse descrever o processo de ETL integrando dados estruturados e não estruturados, e ao desenvolvimento de uma aplicação que permitisse avaliar esse modelo.

Após o estudo inicial sobre o processo de ETL, foi possível a definição de uma extensão ao modelo para possibilitar a integração de dados estruturados e não estruturados.

Com o modelo especificado, escolheu-se uma ferramenta de código aberto para ser desenvolvida uma aplicação-modelo que pudesse avaliar a qualidade do modelo proposto. Foi necessário desenvolver extensões para permitir que o

processo ETL fosse feito de forma a integrar dados estruturados e não estruturados nessa ferramenta.

Com a aplicação do modelo e com as extensões prontas, foram executadas todas as etapas do processo proposto, sendo feitas as análises necessárias nos dados gerados para avaliar o modelo.

1.5 Organização do trabalho

Este capítulo apresentou as motivações, os objetivos e as justificativas relacionadas ao assunto abordado no presente trabalho.

No capítulo dois, é feita uma revisão da literatura referente ao assunto tratado, em que se relacionam os principais conceitos envolvidos, tais como Data Warehouse, Business Intelligence, modelo de dados e representação de conhecimento.

No capítulo três, propõem-se um processo de ETL e um modelo de um DW que visam solucionar alguns dos problemas levantados nesse capítulo. É feita uma descrição completa dos modelos e dos passos para a construção do DW. Ainda no mesmo capítulo, é apresentada uma análise do modelo proposto.

No capítulo quatro, faz-se uma avaliação do modelo descrito no capítulo três, a qual simula uma situação real. No fim do capítulo, são mostrados os resultados dos testes efetuados.

No capítulo cinco, são relatadas as conclusões desta pesquisa, assim como trabalhos futuros que podem ser realizados a partir da proposta aqui apresentada.

2 Fundamentação teórica

A era da produção e do acúmulo de dados deu lugar à era da integração dos dados (BRAZHNIK; JONES, 2005). O desafio de se extrair informação de grandes volumes de dados já foi vencido. Essa abundância de dados coletados pela ciência e pelos negócios deve ser unida e integrada para que seja possível a extração de informação e a descoberta de conhecimento.

Um Data Warehouse é um sistema de apoio à tomada de decisão e o faz a partir do resultado de consultas analíticas a grandes volumes de dados de negócio que foram previamente agrupados para aperfeiçoar essas consultas. Originalmente, esses dados eram carregados de bancos de dados operacionais, mas, como visto por Brazhnik e Jones (2005), o objetivo atual é integrar todas as fontes de dados para descoberta de conhecimento e, por consequência, para apoio à tomada de decisão.

Para a construção de um Data Warehouse, é necessário executar o processo de Extração, Transformação e Carga (ETL), o qual é responsável pela: a) extração dos dados de suas diversas fontes (tradicionalmente estruturadas); b) limpeza; c) customização para adequação ao modelo de dados; e d) inserção de fato dos dados (VASSILIADIS, 2001). Segundo Inmon (2002), esse processo é considerado um dos mais críticos na criação de um Data Warehouse.

Nas seções seguintes, serão abordadas essas questões levando-se em consideração a literatura acadêmica, principalmente sobre Data Warehouses e sobre o processo de ETL. Por fim, será realizado um estudo sobre a nova geração de Data Warehouses (DW 2.0), o qual visa o uso de dados estruturados e não estruturados.

2.1 Data Warehouse

2.1.1 Histórico

De acordo com Inmon (2002), um Data Warehouse é um banco de dados para apoio à tomada de decisões que deve ser integrado, não volátil, variável em

relação ao tempo e organizado por assuntos. Um dos aspectos mais importantes de um DW é a granularidade, que se refere ao nível de detalhamento ou sumarização dos dados. Uma das vantagens de um DW é o fato de os dados serem armazenados cronologicamente, gerando um histórico de determinado domínio de análise.

Talvez a parte mais complicada do processo de criação de um DW seja a integração dos dados. O problema é que muitas organizações têm sistemas legados que utilizam os dados apenas para seus fins operacionais. É necessária a adequação desses sistemas para que possam fornecer uma visão integrada do negócio.

Existem dois principais métodos de construção de um Data Warehouse: (1) o modelo *top-down*, apresentado por Inmon (2002), e (2) o modelo *bottom-up*, apresentado por Kimball (2004).

2.1.1.1 Top-down (Inmon)

Segundo Inmon (2002), um Data Mart é uma extração de dados do Data Warehouse voltada para o atendimento das necessidades específicas de um departamento da organização.

Inmon (2002) defende uma abordagem *top-down* para o desenvolvimento de um DW. Também chamado de Data Warehouse monolítico, o DW é projetado e desenvolvido com toda a organização em mente. Então, após o desenvolvimento completo do DW, pode-se criar Data Marts específicos para cada departamento.

2.1.1.2 Bottom-up (Kimball)

Segundo Kimball (2004), um Data Mart é um conjunto de dados relacionados a um assunto do negócio ou a um departamento da organização. Desse modo, a união dos Data Marts forma o Data Warehouse.

Kimball (2004) defende uma abordagem *bottom-up* para o desenvolvimento de um DW. Desse modo, cada assunto deve ser desenvolvido de forma separada e, após a criação de Data Marts para cada assunto da organização, deve ser feita a integração dos sistemas.

2.1.2 Taxonomia de Data Warehouses

Ao longo dos anos, algumas organizações mudaram o sentido dos Data Warehouses para suprir necessidades próprias, gerando novas versões modificadas da arquitetura originalmente proposta.

2.1.2.1 Data Warehouse Ativo (*Active DW*)

Um Data Warehouse Ativo é um DW em que podem ser feitos processamento e atualizações on-line. Uma das características do Data Warehouse Ativo é o processamento de transações de alto desempenho. Os pontos fracos desse modelo são:

- dificuldade de manter a integridade entre dados e transações: quando uma transação não é executada corretamente, é muito difícil encontrar a informação que deve ser retornada ao estado anterior;
- capacidade: para garantir um bom tempo de resposta para as operações on-line, deve existir uma capacidade de suprir as necessidades de horários de pico, o que tende a deixar recursos preciosos sem ser utilizados em horários mais leves, resultando em um alto custo operacional;
- processamento estatístico: sempre existe um problema com processamento estatístico pesado conflitando com a utilização de recursos do sistema; e
- custo: o ambiente Data Warehouse Ativo é custoso pelo fato de a capacidade extra ficar inutilizada esperando horários de pico e pela noção de que todos os detalhes devem ser armazenados, mesmo que a probabilidade de acesso tenha diminuído há muito tempo.

2.1.2.2 Data Warehouse Federado (*Federated DW*)

Na abordagem Data Warehouse federado, o objetivo é criar um DW integrando as bases de dados dos sistemas antigos e permitir que os dados possam ser acessados de forma simultânea. As organizações podem decidir usar essa abordagem pelo grande trabalho necessário para a integração dos dados.

A ideia de um Data Warehouse federado é muito atrativa por evitar o trabalho de integração dos dados. Segundo Inmon, Strauss e Neushloss (2007), o uso de um Data Warehouse federado para evitar a integração dos dados é mais uma ilusão do que uma solução. Existem muitos problemas fundamentais, como, por exemplo:

- desempenho muito baixo: muitos motivos podem levar um data warehouse federado a ter um baixo desempenho, uma vez que os dados estão sendo acessados tanto pelos sistemas legados quanto pelo próprio DW;
- falta de integração dos dados: não há realmente uma integração nos dados. Isso pode gerar inconsistências, por exemplo, moedas com o mesmo nome mas com valores diferentes podem ser adicionadas à mesma consulta durante a “federação” dos dados;
- técnicas complexas: os diferentes sistemas não foram feitos para cooperar com trocas de informações, e deve ser feito um trabalho complexo para garantir essa união de dados;
- dados históricos limitados: os únicos dados históricos disponíveis para os Data Warehouses federados são os dados dos sistemas legados, que têm preocupação principalmente com o desempenho. Devido a isso, apenas uma pequena parte dos dados chega ao Data Warehouse federado;
- consultas de dados não repetíveis: é impossível garantir a repetição de uma consulta em um Data Warehouse federado. Os dados das bases legadas são alterados de acordo com o fluxo operacional; e
- granularidade dos dados herdada: o Data Warehouse federado está “preso” à granularidade utilizada nos sistemas legados.

2.1.2.3 Data Warehouse Esquema-Estrela (Star Schema DW)

Segundo Inmon, Strauss e Neushloss (2007), a abordagem do esquema-estrela requer a criação de tabelas de fato e tabelas de dimensões, que são tabelas com características que tornam o modelo otimizado a consultas. As tabelas de fato contém os dados sumarizados na granularidade definida na criação do DW e essas tabelas por sua vez têm uma relação com as tabelas de dimensões, que contém descrições sobre os dados, que podem ser usados para agrupar e filtrar os dados das tabelas de fato.

Com a utilização dessa abordagem, consegue-se a maioria dos benefícios que um Data Warehouse real pode oferecer, mas ainda existem alguns problemas fundamentais:

- fragilidade: Data Warehouses que utilizam a abordagem esquema-estrela tendem a ser frágeis. Se os requisitos se mantiverem os mesmos durante o tempo, não há problema, mas, assim que os requisitos forem alterados, devem ser feitas modificações massivas no esquema-estrela existente ou ele deve ser refeito;
- extensibilidade limitada: esquemas-estrela são difíceis de serem estendidos;
- um público: normalmente um público acha um esquema-estrela ótimo, enquanto todos os outros usuários não pensam assim. O objetivo de um Data Warehouse é satisfazer um grande público;
- proliferação de esquemas-estrela: devido ao fato de um único esquema-estrela não satisfazer uma grande quantidade de usuários, é comum a criação de vários esquemas-estrela. Quando isso acontece, é inevitável a diferença de granularidade entre eles, e a integração dos dados torna-se um problema; e
- involução: para resolver o problema de vários esquemas-estrela, os dados de cada esquema-estrela devem ser colocados no seu menor nível de granularidade. Isso derrota a teoria inicial da criação dos esquemas-estrela.

2.1.2.4 Data Mart

A abordagem consiste em criar vários Data Marts para depois transformá-los em um Data Warehouse. Contudo, tal abordagem tem os seguintes pontos fracos (Inmon, 2007):

- não há consenso nos resultados: os vários departamentos de uma organização podem dar respostas diferentes para a mesma pergunta. Com a abordagem de vários Data Marts, isso continua acontecendo;
- proliferação de extração: quando um Data Mart é criado, a extração de dados é aceitável, mas quanto mais Data Marts são criados, maior é o impacto da extração de dados sobre os sistemas legados;

- propagação de mudanças: quando existem múltiplos Data Marts, e é necessário efetuar alguma mudança. Pode ser necessário efetuar essas mudanças por outros ou até por todos os Data Marts. Com isso, a chance de erros por serem feitas mudanças de formas diferentes aumenta consideravelmente; e
- não extensível: quando há a necessidade de se criar mais um Data Mart, torna-se difícil a reutilização de informações ou processos dos Data Marts anteriores. Nesse caso, normalmente deve-se fazer o trabalho desde o início.

2.1.3 Business Intelligence (BI)

Segundo Pinheiro (2008), os objetivos de um ambiente de Business Intelligence estão presentes desde o início da utilização dos sistemas computacionais: auxiliar as organizações a controlar e a otimizar melhor seus processos. Porém, os sistemas de banco de dados e as diversas aplicações que manipulam os dados de uma determinada organização otimizam e controlam tais processos com uma granularidade pequena, ou seja, com o foco totalmente voltado para as transações referentes aos processos operacionais e aos usuários que controlam esses processos. Dessa forma, uma classe de usuários não é atendida adequadamente por um ambiente desse tipo – essa classe é composta dos usuários que são responsáveis pelas tomadas de decisão nas organizações, estabelecendo estratégias de negócio, planos de marketing, campanhas promocionais, etc. Para superar essa barreira, foram idealizados e projetados os ambientes de DW, nos quais o foco é o armazenamento das informações de uma determinada organização pertinentes ao processo de tomada de decisão.

O conceito de Business Intelligence vai além disso – um ambiente dessa natureza irá possibilitar explorações analíticas e gerenciais por parte de usuários de negócio para aperfeiçoar e otimizar o processo de tomada de decisão. A criação de um ambiente de inteligência de negócios, ou inteligência analítica, tem como principal objetivo criar uma infraestrutura tecnológica capaz de atender às necessidades de negócio das empresas. Entende-se que a orientação para os desenvolvimentos no ambiente de BI deva sempre partir da área de negócio, que

demanda necessidades de visões e contextos relacionados com assuntos que auxiliem o processo de tomada de decisão (PINHEIRO, 2008). Essa é também a visão de Kimball (2008), que afirma que os requisitos de negócio devem sempre orientar a concepção de uma arquitetura de BI.

Segundo Moss e Atre (2003), Business Intelligence ou BI constitui uma arquitetura e uma coleção de aplicações de suporte à decisão, operacionais integradas e bases de dados que proveem à comunidade de negócios acesso aos dados de negócios. Esses dados são acessados e analisados pelos gestores das organizações a partir de ferramentas OLAP (Online Analytical Processing), que são ferramentas de consultas a Data Warehouses que permitem a visualização dos dados de forma multidimensional. O usuário deve executar consultas OLAP com base no seu conhecimento do domínio, porém, em muitos casos, os dados podem conter informações e/ou relacionamentos sobre os quais o usuário não tinha conhecimento *a priori*. Nesses casos, são usadas as técnicas de Data Mining e Text Mining, que têm a função de extrair informações úteis previamente desconhecidas de fontes de dados.

2.1.3.1 OLAP (Online Analytical Processing)

Segundo Inmon (2002), OLAP é uma tecnologia, ao passo que Data Warehouse é uma arquitetura de infraestrutura. Existe uma relação simbiótica entre ambos. Em um caso comum, o Data Warehouse serve como fundação para os dados que vão seguir para a ferramenta OLAP – alimentando subconjuntos dos dados detalhados para o processamento analítico, aonde será sumarizada e agregada.

Já segundo Moss e Atre (2003), OLAP refere-se a uma tecnologia que cria novas informações de negócios a partir de uma robusta coleção de transformações de negócios e cálculos executados sobre os dados existentes. Moss e Atre (2003) ainda apontam algumas vantagens das ferramentas OLAP:

- o foco no processamento analítico dos dados, especificamente os aspectos multidimensionais dos dados que são suportados pelas ferramentas OLAP. Objetos de negócios são representados como dimensões (ex.: produto, consumidor, departamento), que são naturalmente inter-relacionados através de áreas de assuntos funcionais

(ex.: vendas) e são muitas vezes hierárquicas (ex.: produtos pertencem a categorias, departamentos pertencem a divisões);

- analistas de negócios navegam por essas dimensões por meio de “drill down”, “roll up”, “drill across”, etc. Eles podem executar uma operação de “drill down” para acessar os detalhamentos dos dados e podem executar um “roll up” para visualizar os dados sumarizados. Também podem executar um “roll up” através de níveis hierárquicos de dimensões ou para uma característica específica ou elementos de dados (colunas) das dimensões. Um “drill across” por dimensões pode acessar os dados de dimensões inter-relacionadas. Além disso, serviços computacionais poderosos proveem funções como ordenação, médias, retorno de investimento, entre outras.

Segundo Anzanello (2002) e Pinheiro (2008), o termo OLAP foi citado pela primeira vez por E. F. Codd quando ele definiu as doze regras às quais essas aplicações deveriam atender. De acordo com Anzanello (2002), a visão conceitual multidimensional dos negócios de uma empresa foi umas das regras citadas, a qual se tornou a característica fundamental no desenvolvimento dessas aplicações. A visão multidimensional consiste de consultas que fornecem dados a respeito de medidas de desempenho, decompostas por uma ou mais dimensões dessas medidas. Esses dados podem também ser filtrados pela dimensão e/ou pelo valor da medida.

Análises OLAP multidimensionais apresentam os dados de acordo com vários níveis de detalhamento (granularidade). O processo agrega os dados de acordo com o nível de detalhamento com funções tais como, soma, média, máximo, mínimo (RAVAT; OLIVIER; TOURNIER, 2007).

2.1.3.2 Data Mining

Os bancos de dados têm cada vez mais dados acessíveis, motivo pelo qual ler e interpretar esses dados torna-se mais difícil a cada dia. Devido a essas dificuldades, foram desenvolvidas técnicas e algoritmos para descoberta de informações previamente desconhecidas, mas potencialmente relevantes sobre uma coleção de dados. Assim surgiu a área de *Descoberta de Conhecimento em Dados (Knowledge Discovery from Data - KDD)* (WIVES, 2002).

Data Mining é um processo de extração de informações potencialmente úteis, porém anteriormente desconhecidas a partir de bases de dados (CHEN; HAN; YU, 1996). Essas informações podem ser recuperadas a partir de uma base de dados, informações, relações entre dados, o que torna possível uma análise dos dados que anteriormente não poderia ser realizada. O conhecimento descoberto pode ser aplicado a diversas áreas, e várias aplicações atualmente utilizam técnicas de Data Mining para entender o comportamento do usuário, melhorar os seus serviços e aumentar as oportunidades nos negócios.

Processos de mineração de dados são comumente aplicados a um DW durante o seu uso. Essas técnicas tornam-se parte da etapa de análise dos dados.

2.1.3.3 Text Mining e Document Warehouse

Segundo Cody et al. (2002), um componente vital para o sucesso das organizações modernas é a habilidade de se obter vantagem a partir das informações disponíveis. Esse desafio torna-se cada vez mais difícil com o aumento constante das massas de dados tanto internas quanto externas à empresa. Assim como o KDD ajuda a descoberta de conhecimento sobre dados estruturados, a área de *Descoberta de Conhecimento em Textos (Knowledge Discovery from Text - KDT)* ajuda a descoberta de conhecimento sobre dados não estruturados textuais.

Text Mining pode ser definido como um processo que utiliza técnicas de recuperação de informação, extração de informação e processamento de linguagem natural (*Natural Language Processing - NLP*) sobre dados não estruturados e os relaciona utilizando algoritmos de Data Mining e estatística (HOTHO; NURNBERGER; PAASS, 2005).

Muitas das técnicas de KDT são na verdade técnicas de KDD aplicadas sobre dados extraídos de textos. Para que isso seja feito, muitas vezes os dados não estruturados devem ser armazenados em uma estrutura otimizada para consultas, possibilitando a análise – um DW supre essa demanda de forma natural. Essas estruturas são conhecidas como *Document Warehouse*.

Segundo Cody et al. (2002), pode-se empregar técnicas usadas em BI com dados em documentos textuais utilizando-se um modelo dimensional em que a granularidade da tabela fato é o documento e as dimensões armazenam atributos desse documento. Sem nenhum processamento adicional, essa representação é

uma tabela fato “sem fatos”, já que não há, ainda, medidas associadas a esse fato. O processo de popular o Document Warehouse possui algumas complexidades além do processo de ETL comum. Dependendo da fonte, muitos documentos possuem metadados que podem ser naturalmente utilizados para popular as dimensões, tais como autor, data de publicação, etc. No entanto, existem dimensões de interesses não presentes nos metadados, casos em que se deve utilizar técnicas de classificação para popular a mesma. Usando-se essa metodologia, a maioria das técnicas disponíveis para cubos de dados estarão também disponíveis para cubos de documentos.

Cody et al. (2002) sugerem um modelo que utiliza dimensões compartilhadas que funcionam como ligação entre os dados e os documentos (Figura 3). Dessa forma, é possível descobrir documentos relevantes em relação a algum produto, em algum local, em certa data.

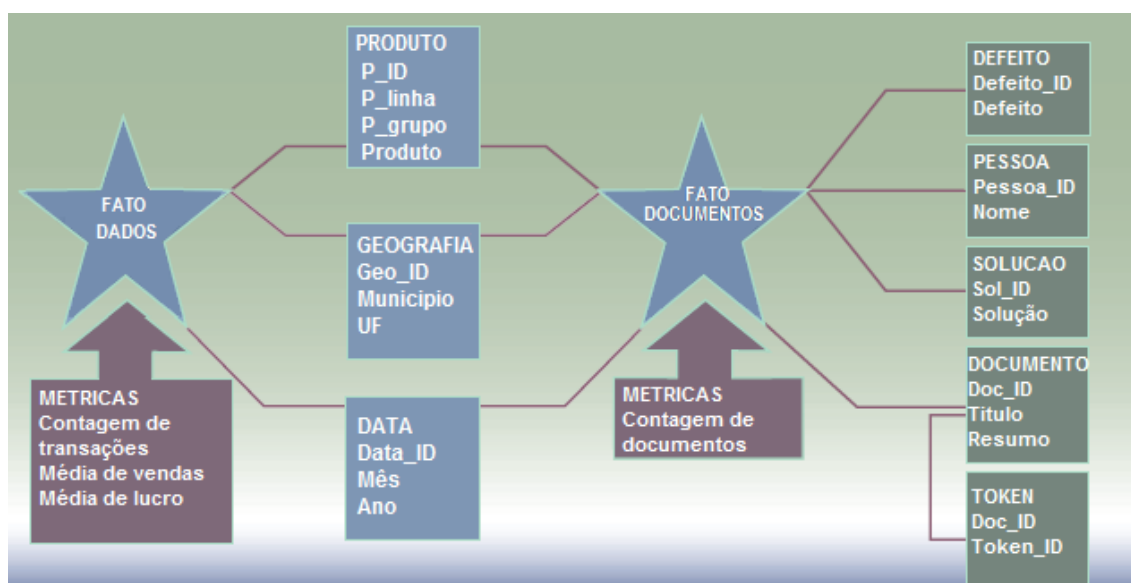


Figura 3 - Modelo de dimensões compartilhadas.
Fonte: adaptado de Cody et al. (2002)

Considerando também as soluções atuais de integração entre dados estruturados e não estruturados no contexto de DW, encontramos o *Contextualized Warehouse*, que é um novo tipo de sistema de suporte à decisão que proporciona aos usuários a obtenção de informações estratégicas combinando fontes de dados estruturados e documentos não estruturados (PÉREZ; BERLANGA; ARAMBURU, 2009).

2.2 Processo de ETL

Segundo Inmon, Strauss e Neushloss (2007), *Extração, Transformação e Carga (Extraction, Transformation and Load – ETL)* são os três passos para a construção de um DW. O processo de ETL deve recolher os dados de várias aplicações legadas e transformá-los para que possam trabalhar de forma integrada. O grande problema é que esses dados geralmente não são desenvolvidos para trabalhar em conjunto, pois existem diferenças de estrutura, formato, cálculos, definições dos dados e outras diferenças semânticas.

Segundo Kimball (2004), todo sistema de ETL deve armazenar dados de várias formas, permanentes ou semipermanentes, por isso muitas vezes esses sistemas são chamados de área de estagiamento. Em muitos projetos de ETL, é necessário escolher como fazer essa área de estagiamento. A grande dúvida é se os dados devem ser armazenados em disco ou em memória. As duas questões mais importantes que definem o equilíbrio entre essas duas opções são:

- levar os dados da origem ao alvo final o mais rápido possível; e
- ter a possibilidade de recuperar os dados caso haja alguma falha sem reiniciar o processo do início.

A decisão para estagiar os dados depende do ambiente e dos requisitos do negócio. Todo Data Warehouse contém uma área de estagiamento de alguma forma. Para decidir se isso será feito em memória ou em disco, deve-se considerar as seguintes razões:

- recuperabilidade: em muitos sistemas, é uma boa prática armazenar os dados assim que são colhidos da fonte e após cada uma das transformações mais importantes. Dessa forma, não é necessário entrar no sistema-fonte novamente caso ocorra algum erro. Isso é especialmente importante com dados na web, que podem não estar disponíveis novamente;
- backup: usualmente, os Data Warehouses contêm volumes massivos de dados, o que impossibilita seu *backup* completo. Se os arquivos de carga forem guardados, o Data Warehouse pode ser recuperado caso haja algum problema;

- auditoria: com os dados salvos em áreas de estagiamento fica muito mais fácil fazer uma auditoria do processo de ETL apenas comparando-se os dados de entrada com os dados de saída, levando-se em consideração as modificações no código ETL, principalmente se o Data Warehouse sobrescreve seus dados. Isso torna cada vez mais difícil saber que alterações os dados sofreram no processo de ETL, o que pode torná-los não confiáveis.

Processos ETL podem operar em dois modos básicos: (1) on-line (tempo real),(2) processamento em *batch* (ou em lote).

2.2.1 Modos de processo de ETL

2.2.1.1 Modo On-line

Quando o processo de ETL é executado no modo on-line, o tempo a partir do momento da execução da transação no sistema legado é medido em pequenas unidades de tempo, como milissegundos, até que a transação seja refletida no ambiente do Data Warehouse. Frequentemente, as transações parecem ser geradas simultaneamente nos dois ambientes.

O problema do ETL em tempo real é que a velocidade é considerada a principal fonte do sucesso. Devido à necessidade de melhorar a velocidade, poucas transformações podem ser efetuadas nos dados.

2.2.1.2 Modo Batch

No modo batch (*processamento em lote*), as atualizações nos ambientes legados são armazenadas em memória ou marcadas como modificadas, então, em algum momento conveniente (talvez durante a noite), o lote das transações legadas é rodado no processo de ETL. Isso quer dizer que, desde o tempo de a transação ser processada no ambiente legado até o DW conhecê-la, 24 horas ou mais podem ter transcorrido. A grande vantagem dessa abordagem é que existe um tempo maior para se realizarem transformações significativas nos dados.

2.2.2 Dados estruturados e não estruturados

Existem dois tipos de dados: (1) estruturados, que são facilmente lidos por máquinas e têm uma estrutura bem definida; e (2) não estruturados, que são dados armazenados sem um formato ou estrutura-padrão, dificilmente lidos por máquinas.

Segundo Inmon, Strauss e Neushloss (2007), dados estruturados são aqueles armazenados sempre da mesma forma e *layout*, normalmente enviando-se transações. Alguns exemplos seriam os dados gerados por transações bancárias, reservas de passagens aéreas, etc. Dados estruturados são convenientemente armazenados em bancos de dados, que contêm chaves, atributos, índices e tabelas para representar os dados.

Dados não estruturados existem em duas formas básicas: (1) textual e (2) não textual. Dados não estruturados textuais podem ser e-mails, conversas por telefone, páginas web, etc. Dados não estruturados não textuais podem ser imagens, gráficos, fotografias, raios-x, diagramas, etc. Os dados em formato de texto livre não possuem estrutura alguma. Porém, em outros formatos, os documentos podem ter alguma espécie de estrutura inferida, como, por exemplo, um livro de receitas, em que cada receita é um dado diferente de outra receita. Ou seja, existem inícios e fins implícitos dentro do documento (DOAN, 2008; WEI, 2006).

Existem também dados semiestruturados, que são dados textuais mas que contêm certa repetição ou estrutura predefinida no formato dos dados, como, por exemplo, um livro de receitas, que constitui um dado textual, mas cada receita segue um mesmo formato predefinido (com ingredientes e formas de preparo) para descrever o prato.

Segundo Bernstein (2008), o ato de integrar e publicar informações de documentos não estruturados é conhecido como *Document Managing* (gerenciamento de documentos). Esse processo pode simplesmente tornar os documentos disponíveis em um portal web, combinar informações desses documentos em um novo documento ou até mesmo combinar essas informações com outras em um banco de dados relacional.

O objetivo de um DW 2.0 é integrar os dados estruturados, semiestruturados e não estruturados em um Data Warehouse. De acordo com Inmon, Strauss e Neushloss (2007), esse é o maior desafio na construção de um DW, visto que dados

não estruturados são difíceis de serem interpretados por uma máquina, o que torna a extração de informações relevantes desses dados mais complexa.

Uma das ferramentas mais importantes para a coleta de dados não estruturados é conhecida como *crawling*. Um *crawler* é um componente e software que iterativamente coleta o conteúdo de documentos (GOMES; SILVA, 2008). Usualmente essas ferramentas são utilizadas no contexto da web, coletando informações de sites e seguindo seus links para coletar mais documentos.

Devido à grande quantidade de informações contidas em dados não estruturados, é cada vez mais difícil utilizar técnicas de *crawling* para recuperar documentos relevantes entre muitos. Por essa razão, utilizam-se muitos mecanismos de coleta baseados em técnicas inteligentes, tais como *focused crawling* (ZHENG; KANG; KIM, 2008).

Existem diversas técnicas de *crawling*, e o uso dessas técnicas com certeza apoiaria o processo de criação de um DW integrando dados estruturados e não estruturados. Porém, apenas a coleta dos dados, mesmo com técnicas como *focused crawling*, não é suficiente em alguns casos, portanto, será necessário o uso de técnicas de extração de informação para realmente poder retirar as informações desejadas dos documentos recuperados.

2.2.3 Representação de conhecimento

No processo de extração de informação, normalmente são utilizadas técnicas que consideram alguma estrutura de representação do conhecimento para auxiliar na identificação de padrões/termos/conceitos presentes em conteúdo não estruturado. Entre essas estruturas, estão as taxonomias, os tesauros e as ontologias, os quais são necessários na maioria das vezes para guiar o processo de extração ou recuperação de informação. Garshol (2004) define essas técnicas como “classificação baseada em sujeitos”, o que quer dizer que a definição dos objetos é levada em consideração para a criação das regras ou hierarquias que definem o significado de cada objeto. Por outro lado, a extração de informação é também uma ferramenta para a explicitação de padrões e auxílio na construção e na manutenção dessas estruturas formais de conhecimento. Dependendo da abordagem, pode-se primeiramente utilizar tais técnicas para se criar uma taxonomia, por exemplo. Mais detalhes são apresentados na seção 2.2.4.

2.2.3.1 Taxonomias

Segundo Garshol (2004), o termo *taxonomia* foi iniciado por Carl von Linné, que desenvolveu um sistema de hierarquia para classificar as formas de vida no século 18. Então taxonomia pode ser definida como uma classificação baseada em assuntos que organiza os termos em um vocabulário controlado em uma hierarquia. Através dos tempos, o termo foi utilizado para estruturas mais complexas.

2.2.3.2 Tesouros

Segundo Jing (1994), um tesouro é uma lista de itens (frases ou palavras) mais uma série de relações entre esses itens. Existem três problemas básicos relacionados ao uso de tesouros para recuperação de informação. São os seguintes:

- construção: há dois tipos de tesouros: (1) manuais e (2) automáticos. Os manuais exigem o tempo dos especialistas para o desenvolvimento, e os automáticos dificilmente terão a mesma qualidade;
- acesso: dada uma busca específica, o tesouro deve ser acessado e usado de alguma maneira que melhore ou expanda a busca. Isso pode onerar a busca que está sendo realizada; e
- avaliação: após a construção do tesouro, é importante saber o quão bom ele é. Tesouros manuais são avaliados no que se refere a sonorização, cobertura da classificação e seleção de itens. A avaliação de tesouros gerados automaticamente é geralmente feita via uma expansão de uma busca para que se possa verificar se a performance é melhorada.

Segundo Igarashi (2005), um tesouro deve conter sinônimos e antônimos para cada palavra, e termos com significados muito próximos. Um tesouro pode ser utilizado durante o processo de armazenamento do documento para controlar o vocabulário, ou seja, substituindo um termo por um termo-padrão que seja sinônimo. Ou, ainda, pode ser utilizado durante o processo de consulta, o que aumenta o número de documentos retornados sem diminuir a relevância da busca.

Igarashi (2005) ainda cita que um tesouro pode ser desenvolvido a partir da coocorrência de termos, já que os tesouros focam na relação entre os termos. Essa seria uma técnica de geração automática de tesouros.

2.2.3.3 Ontologias

Segundo Uschold e Gruninger (1996), ontologia é o termo usado para se referir ao entendimento compartilhado de um domínio de interesse. Podem ser utilizadas para definir padrões entre conceitos, com o objetivo de solucionar problemas como comunicação pobre entre pessoas de áreas distintas, interoperabilidade e reuso entre diferentes sistemas. Uschold e Gruninger (1996) também dizem que uma ontologia necessariamente engloba ou representa algum tipo de visão de mundo com relação a um domínio dado. Essa visão de mundo é muitas vezes concebida como uma série de conceitos (ex.: entidade, atributos, processos), suas definições e seus relacionamentos. Isso é referido como “*conceitualização*”.

De acordo com Guarino (1998), podemos nos referir a uma ontologia no sentido filosófico como sendo um sistema de categorias considerando certa visão de mundo. Mesmo no caso mais simples, uma ontologia descreve uma hierarquia de conceitos relacionados.

Conforme apontam Uschold e Gruninger (1996), é possível encontrar diversas definições de usos para as ontologias, mas, considerando todas elas, é possível subdividir as áreas em que as ontologias podem ser usadas: “Comunicação”, “Interoperabilidade” e “Engenharia de Sistemas: Especificação, Confiabilidade e Reusabilidade”.

Ontologias são especialmente utilizadas em desenvolvimentos para a web semântica. Ferramentas de engenharia de linguagem obtêm uma representação formal possivelmente até raciocinar sobre seus domínios. Além disso, essas ferramentas podem ser usadas para popular uma ontologia com instâncias descobertas em textos ou até mesmo construir ontologias dinamicamente (BONTCHEVA, 1998).

2.2.4 Extração de Informação

Segundo Cowie e Wilks (2005), Extração de informação (EI) é o nome dado a qualquer processo que estrutura e combina dados selecionados, encontrados ou explicitamente nomeados em um ou mais textos. O objeto de saída final do processo

de extração varia; para os autores, entretanto, em todos os casos, ele pode ser transformado com o objetivo de popular algum tipo de base de dados.

Bernstein (2008) define extração de informação como “um termo geral para uma série de técnicas que produzem informações estruturadas a partir de textos livres”. É importante citar entre essas técnicas o reconhecimento de entidades, extração de relacionamento. Segundo Bernstein (2008), quando um fragmento de texto é reconhecido como um conceito, aquele fato pode ser marcado com tags XML identificando o seu conceito ou sendo adicionado a um índice, ou até mesmo a uma tabela relacional.

A extração de informação não deve ser confundida com a tecnologia de Recuperação de Informação (RI), em que a partir de uma consulta do usuário são recuperados documentos que podem ser relevantes. O usuário então seleciona entre os documentos retornados aqueles em que pode encontrar a informação necessária. A diferença entre as duas tecnologias poderia ser resumida como: RI recupera documentos relevantes de coleções e EI extrai informações relevantes a partir de documentos (WILKS; GAIZAUSKAS, 1998). As duas técnicas são então complementares.

2.2.4.1 Dados não estruturados e extração de Informação

De acordo com Brazhnik (2005), no passado o principal objetivo era ter arquivos massivos de dados, o que fez da integração de dados uma atividade principal. Os desafios técnicos de gerenciar esses volumes massivos de dados fizeram com que o foco mudasse para o valor informacional desses dados. Os dados acumulados devem ser unidos e integrados para que se possa retirar informações úteis que auxiliem no processo de tomada de decisão.

Segundo Martinez (2007), as tecnologias atuais de Data Warehouses e OLAP podem ser aplicadas eficientemente para analisar grandes montantes de dados estruturados que as organizações produzem. Essas organizações também produzem muitos documentos e usam a web como a sua maior fonte de informação externa. Apesar de esses documentos não poderem ser analisados pelas tecnologias OLAP atuais, principalmente porque não são estruturados e contêm uma grande quantidade de texto, eles podem conter informações valiosas que podem ajudar as organizações em seu negócio.

Conforme Jhingran (2006), existem duas formas de integrar dados estruturados com dados não estruturados. A primeira estende arquiteturas de Business Intelligence com informações estruturadas e dimensões derivadas de dados não estruturados. A segunda estende técnicas de busca indexando todos os dados em um índice de texto.

Segundo Losee (2006), dados estruturados e não estruturados podem ser integrados em uma única lista para busca e recuperação. Losee (2006) propõe um método que organiza os dados de acordo com um código que contém metadados sobre o documento, o que pode indicar uma possível similaridade entre esses dados. Para a inserção desse código, é necessária a integração dos dados a partir de um ponto comum. Os passos para essa integração são realizados no presente trabalho.

Inmon, Strauss e Neushloss (2007) dizem que o primeiro passo para se utilizarem dados não estruturados em um projeto de Data Warehouse é ler esse texto para então processá-lo, de forma que fique em um formato legível pelo Data Warehouse. Assim que os dados forem processados, eles devem ser integrados ao sistema estruturado para que se possa utilizar a tecnologia analítica atual.

Segundo Inmon, Strauss e Neushloss (2007), o processo de integrar o texto que deve ser feito antes de inseri-lo na base de dados tem várias facetas. Muitos passos devem ser executados para essa preparação. A seguir, é apresentada uma lista das facetas mais comuns com suas descrições:

- edições simples: retirar diferenças entre maiúsculas e minúsculas, retirar pontuações, fontes, etc., para que as análises futuras não sejam impedidas devido a detalhes de formatação;
- remoção de “stop-words”: stop-words são palavras que ajudam na construção das frases mas não são necessárias para o entendimento do conteúdo, como artigos, preposições, etc. Ex.: “de”, “o”, “a”;
- substituição de sinônimos: um passo opcional é o de substituir todas as palavras com o mesmo significado por uma palavra-padrão, o que faz com que as buscas se tornem ainda mais coesas;
- desambiguação: é o contrário de substituição por sinônimos. Algumas palavras são iguais mas podem ter sentidos diferentes dependendo de onde são empregadas. Essa técnica tem o objetivo de distinguir essas palavras;

- união por temas: baseado no número de ocorrências conjuntas de várias palavras que podem ser separadas em grupos, formando temas;
- glossário ou taxonomia externa: com um glossário externo o texto pode ser lido e validado como pertencendo a determinado assunto;
- stemming: é uma técnica que reduz a palavra ao seu radical, fazendo com que palavras com o mesmo significado façam parte do mesmo grupo;
- buscas fonéticas: muitas palavras podem ter a mesma pronúncia mesmo escritas de forma diferente, principalmente nomes. Com algoritmos de busca fonética implementados entradas similares seriam encontradas;
- suporte a buscas diretas e indiretas: buscas diretas são buscas em que é passado um argumento para o motor de busca, que retornará resultados que estejam de acordo com o que foi informado. Em buscas indiretas, são passados argumentos para o motor de busca, o qual retorna resultados que possuam relações com o argumento, mas não necessariamente que contenham o argumento buscado.

Para ser possível efetuar uma análise sobre textos, é necessário resolver o problema de terminologia. O fato é que existem muitas formas/palavras para fazerem referência ao mesmo objeto ou entidade, e dificilmente um padrão é utilizado (WEI, 2006). Deve-se usar técnicas de reconhecimento de entidades ou de *stemming* para encontrar as informações requeridas pelo usuário entre as informações armazenadas no sistema (WIVES, 2002).

Existe uma necessidade de se mapearem essas estruturas para ferramentas analíticas. Em alguns casos, pode ser uma tarefa simples e trivial, mas, em sua maioria, pode ser muito complexo. Outro tipo de dado não estruturado que deve ser tratado, especialmente em um ambiente de DW 2.0, é o tipo de dado chamado de “chave/valor”. Cada chave descreve um campo e está associada a um valor. Por exemplo, um currículo teria a chave *nome* e o valor *André Zorzo*. O sistema poderia ser usado para ler esses dados como símbolos, e não apenas como dados (INMON; STRAUSS; NEUSHLOSS, 2007).

Existem diversas técnicas de *Extração de Informação*, algumas das quais podem ser enquadradas dentro da área de recuperação de informação, já que podem ser vistas como técnicas especiais de indexação (WIVES, 2002). Entre as técnicas que podem ser aplicadas, podemos citar:

- **sumarização:** que identifica as palavras e frases mais importantes de um documento ou conjunto de documentos e gera um resumo ou sumário;
- **clustering:** é um método de descoberta de conhecimento utilizado para identificar correlacionamentos e associações entre objetos, permitindo a identificação de classes; e
- **classificação e categorização:** identifica a que classe ou categoria a que determinado documento pertence, utilizando como base o seu conteúdo.

Quando os dados não estruturados estão prontos para o processamento analítico, o texto pode ser colocado em uma base de dados relacional. Essa base pode então ser acessada e analisada por diferentes ferramentas, como ferramentas de BI. O processo de criação da base de dados relacional com dados não estruturados e a criação de uma ligação entre essas fontes de informação podem ser considerados como extração de informação (WILKS; GAIZAUSKAS, 1998).

2.2.5 Ferramentas atuais

Um processo de ETL não precisa ser programado em uma linguagem específica nem criado para uma única plataforma de hardware. Para que uma organização possa criar a sua própria solução de ETL, ela depende muito do conhecimento sobre o processo de ETL e o desenvolvimento de softwares. Porém, muitas empresas nos dias atuais disponibilizam ferramentas prontas ou quase prontas para o desenvolvimento de um processo ETL, tornando a solução mais fácil para organizações que não dispõem dos conhecimentos necessários para criar as suas próprias ferramentas. Nesta seção, serão mostradas algumas dessas ferramentas, suas vantagens, desvantagens e o foco principal.

Pentaho Data Integration (Kettle) – Pentaho é uma empresa fundada em 2004 que desenvolve ferramentas relacionadas à área de BI. A Pentaho oferece a ferramenta *Data Integration*, também conhecida como *Kettle*, que é utilizada para executar as três partes do processo de ETL – extração, transformação e carga. Suas principais características são:

- **código aberto:** todo o código-fonte pode ser alterado e utilizado a critério do usuário, porém apenas para uso interno da empresa. Para distribuição de uma solução, deve-se comprar uma licença específica;

- leitura de diversas fontes de dados: é possível fazer a extração de arquivos de texto, bancos de dados, planilhas eletrônicas, etc.;
- o processo é construído a partir de *jobs* e *transformations*, que são construídas por uma interface gráfica (Figura 4) que apresenta o caminho percorrido pelos dados e por todas as modificações executadas neles; e
- a ferramenta permite a criação ou a edição dos passos realizados durante o processo a partir de extensões (*plugins*).

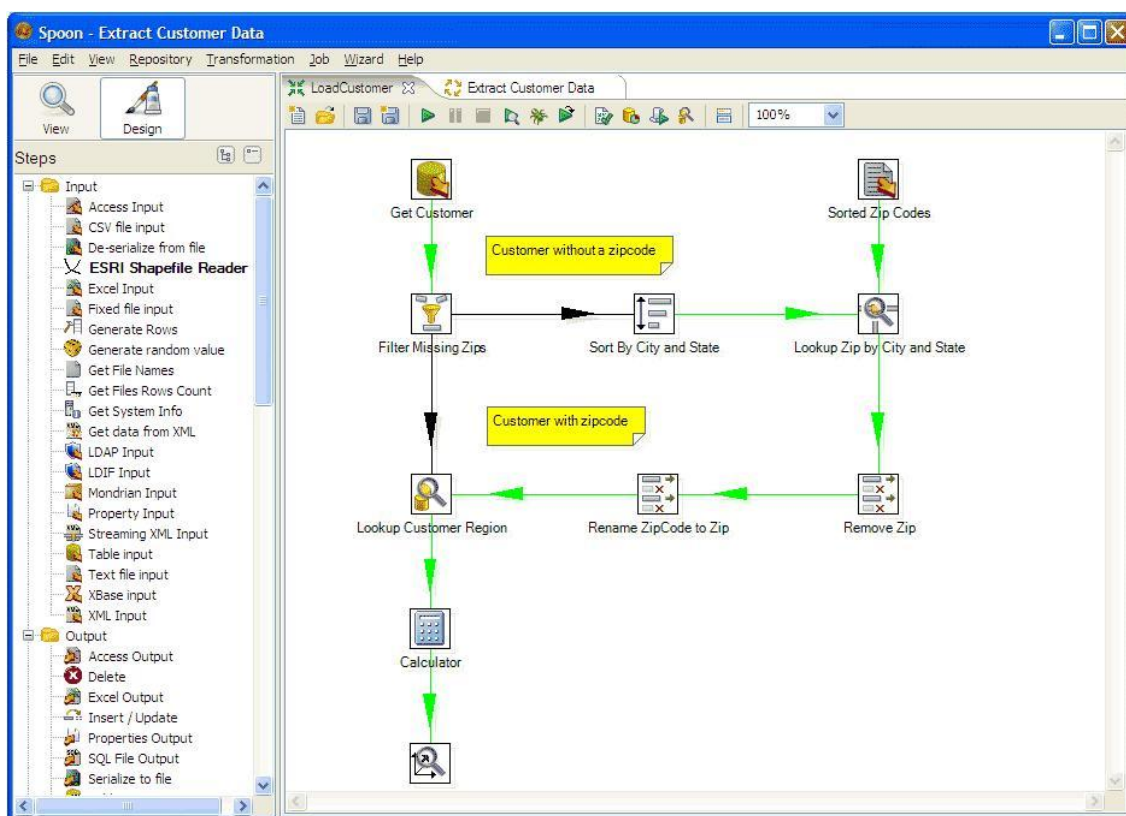


Figura 4 - Exemplo de transformação sendo construída com a ferramenta Kettle

Na Figura 4, pode-se visualizar a interface do Kettle em uma *Transformation* (transformação), que é a forma principal de transformação de dados da ferramenta. Uma transformação é composta de passos (*steps*), que podem ser de entrada (*input*), saída (*output*) ou transformação (*transformation*). Esses passos são relacionados uns com os outros, criando um fluxo por onde os dados passam.

Informatica 9 – Empresa fundada em 1993, focada na ideia de construir Data Warehouses a partir de interfaces gráficas. Oferece a ferramenta PowerCenter, que de forma semelhante à ferramenta Kettle, da Pentaho, pode executar as três partes do processo de ETL. Suas principais características são:

- extração de dados não estruturados: permite o acesso a diversos formatos de arquivos com dados não estruturados. Porém, esses arquivos são coletados de um mesmo repositório e devem conter um formato semelhante;
- o processo é construído a partir de uma interface gráfica semelhante à solução da Pentaho.

SAP Data Integration – SAP é uma empresa alemã focada em softwares para negócios. A SAP tem o produto SAP Data Integration, que na verdade se trata de uma série de ferramentas que juntas permitem a integração de dados para um processo de ETL. Suas principais características são:

- análise textual: a partir da ferramenta *Text Analysis* é possível configurar um *crawler* para recuperar dados de fontes não estruturadas. Esses dados são então transformados em dados estruturados com a ferramenta *Data Integrator*;
- o processo é construído a partir de uma interface gráfica e é separado em *jobs* e *transforms*, sendo *transforms* as operações menores do processo.

Pode-se verificar que todas as ferramentas possuem alguma técnica para extração de dados de documentos não estruturados, porém, em sua maioria, essas ferramentas apenas permitem a leitura de tais documentos, ficando a cargo do usuário um trabalho manual de selecionar quais dados serão resgatados. A única ferramenta que se destacou nesse quesito foi a *SAP Data Integration*, que inclusive permite a configuração de um *crawler* para busca em documentos não estruturados.

A ferramenta Kettle é a única que permite a extensão de suas funcionalidades a partir de *plugins*.

Quanto ao preço, as ferramentas *SAP Data Integration* e *Informatica 9* têm um custo de licença alto, mas a ferramenta da *Kettle* é livre para uso interno de uma organização.

Por fim, verifica-se que todas as ferramentas podem executar de alguma forma o processo proposto, porém vão necessitar de alguma alteração ou configuração adicional. Por isso, foi selecionada a ferramenta Kettle pelo fato de ser de código aberto e permitir a extensão de suas funcionalidades a partir de *plugins*.

2.3 A nova geração de DW (DW 2.0)

Existe uma relação próxima entre extração de informação, Data Mining, ontologias e outras tecnologias que não foram abordadas até o momento neste capítulo (YAO; RAGHAVAN; WU, 2008). Essa relação ficará ainda mais clara com as explicações quanto à nova geração de Data Warehouses, que utiliza muitas dessas tecnologias para apoio à tomada de decisões em uma organização.

Segundo Inmon, Strauss e Neushloss (2007), DW 2.0 é um novo paradigma para Data Warehouses que foca nos tipos básicos de dados, nas suas estruturas e na forma como elas se relacionam para constituir um banco de dados que deve atingir as necessidades por extração de informação das organizações.

A Figura 5 mostra a arquitetura de um DW 2.0. Separando os dados não estruturados (a esquerda) e dados estruturados (a direita), cada um desses tipos de dados é separado de acordo com a probabilidade de serem acessados (*Interactive, Integrated, Near Line e Archival*). Uma camada chamada de *Master Data* permeia todos os dados contendo informações que são utilizadas por todos eles, por fim, existe o repositório de Meta Dados gerais (*Enterprise Meta Data Repository*), esses meta dados contém informações sobre todos os dados do sistema.

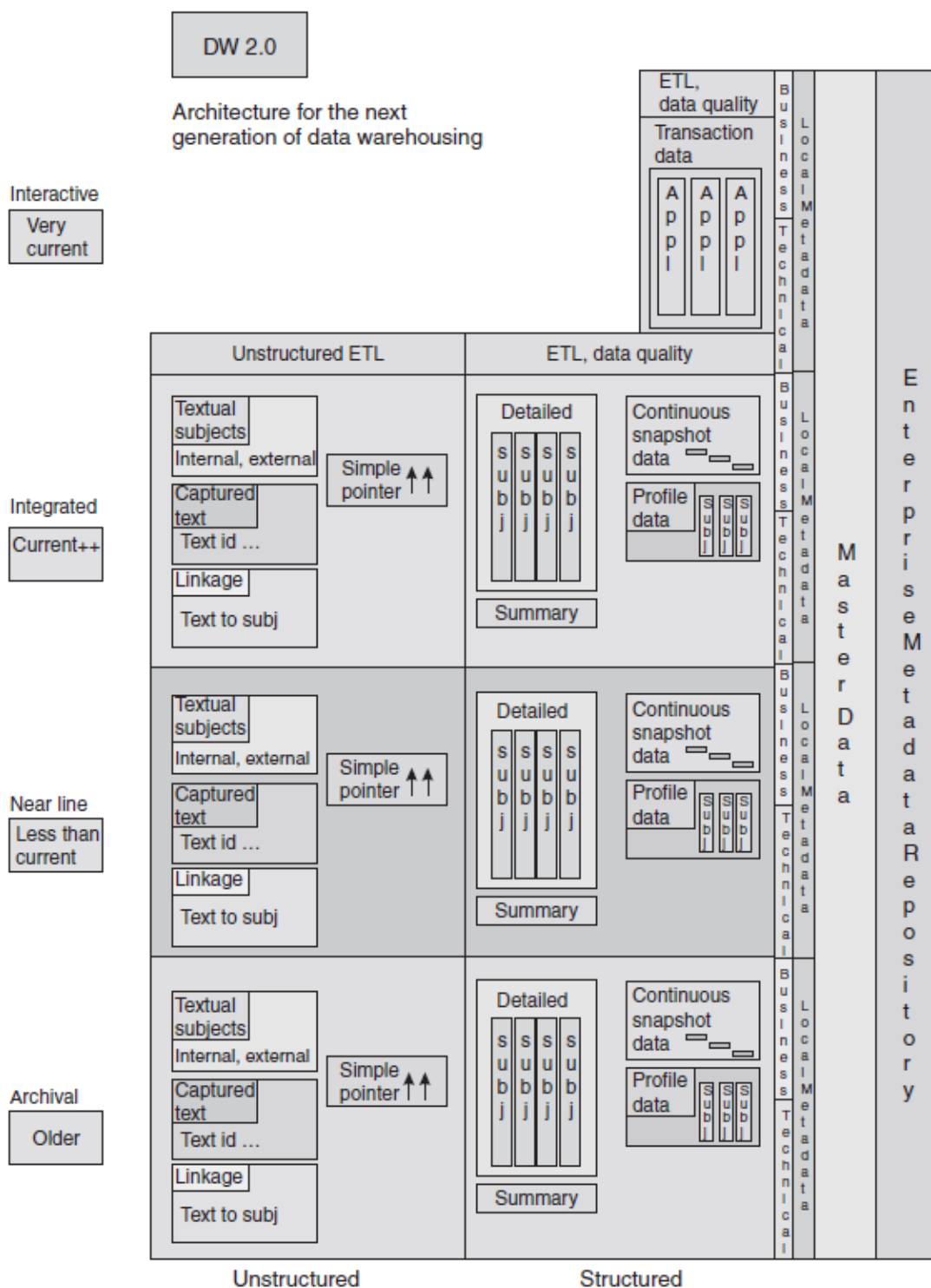


Figura 5 - Arquitetura de um DW 2.0: diferentes tipos de dados, suas estruturas básicas e como os dados se relacionam (Inmon, 2007)

Seguem na sequência algumas das características importantes de um DW 2.0 que possuem um foco mais direcionado aos negócios.

O custo da infraestrutura de um DW na primeira geração aumentava junto com o volume de dados. Na arquitetura DW 2.0, o custo se mantém o mesmo.

A infraestrutura de um DW 2.0 é integrada com metadados. Isso quer dizer que os dados não são perdidos facilmente. Na primeira geração de Data Warehouses, é fácil para uma unidade de dados ou para um tipo de dado serem “perdidos”.

Os dados são acessados facilmente. Em um Data Warehouse de primeira geração, em que os dados são colocados sobre outros dados, logo o volume de dados torna-se um impedimento para o acesso. Dados que devem ser acessados tendem a “se esconder” atrás de uma montanha de dados não necessários. Em um ambiente DW 2.0, os dados são armazenados de acordo com a probabilidade de acesso. O resultado é um acesso aos dados muito mais eficiente do que nas primeiras gerações.

A necessidade para arquivamento é reconhecida. Em Data Warehouses de primeira geração existe pouco arquivamento ou este é inexistente. Como consequência, os dados são armazenados por apenas um curto período de tempo. Em um DW 2.0, os dados são armazenados indefinidamente por quanto tempo for necessário.

Data Warehouses atraem volumes de dados. Os usuários finais de Data Warehouses de primeira geração devem vencer os desafios de acessar grandes volumes de dados. Em um DW 2.0, como os dados estão divididos em seções, o usuário final deve lidar com menos dados.

Com todos esses pontos, um DW 2.0 tem significativas vantagens sobre a primeira geração de Data Warehouses, mas não sem mais desafios. A integração entre dados estruturados e não estruturados, que faz parte da arquitetura de um DW 2.0, será abordada mais à frente neste capítulo.

2.3.1 Ciclo de vida de um DW 2.0

Segundo Inmon, Strauss e Neushloss (2007), diferentemente dos Data Warehouses da primeira geração, a segunda geração de data warehouses possui quatro setores para o ciclo de vida dos dados. O primeiro setor é chamado de setor interativo (*Interactive Sector*). Os dados entram no setor interativo rapidamente. Assim que os dados se estabelecem, eles são integrados e passados para o setor integrado (*Integrated Sector*). Os dados são mantidos no setor integrado até que sua probabilidade de acesso decaia. A queda da probabilidade de acesso dos dados

normalmente vem com a idade. Geralmente após 3 ou 4 anos, a probabilidade de acesso a dados no setor integrado cai significativamente.

Ainda segundo Inmon, Strauss e Neushloss (2007), a partir do setor de integração, os dados podem ser movidos para dois outros setores. Um desses setores é o *Near Line Sector*. De muitas maneiras, o *Near Line Sector* é aparentemente uma extensão do setor integrado. Sendo opcional, os dados não precisam necessariamente passar por ali. Porém, onde existem montantes grandes de dados e onde a probabilidade de acesso é muito diferenciada, o *Near Line Sector* pode ser usado.

O último setor é o setor de arquivos (*Archival Sector*). Dados armazenados no setor de arquivos têm uma probabilidade de acesso muito baixa. Os dados podem ser movidos para o setor de arquivos a partir do *Near Line Sector* ou do setor integrado. Dados no setor de arquivos têm normalmente de 5 a 10 anos ou até mais. A Figura 6 demonstra o ciclo de vida dos dados em um DW 2.0.

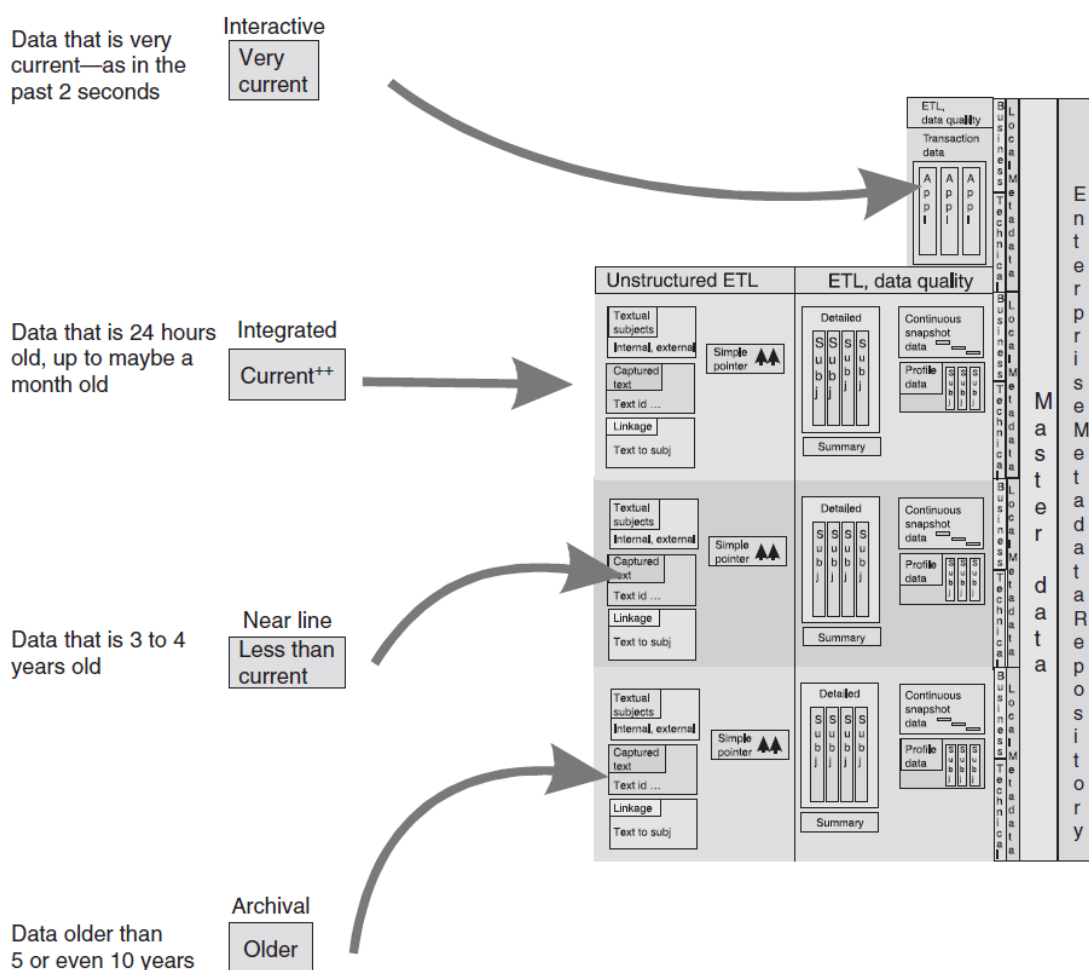


Figura 6 - Ciclo de vida dos dados em um DW 2.0

Também podemos considerar o fluxo dos dados como um avanço no processo de ETL. Nas soluções atuais, os dados seguem um fluxo quase que exclusivamente dos sistemas operacionais para o DW. Na próxima geração, devemos suportar fluxos mais gerais. Por exemplo, a limpeza de dados é uma etapa importante do processo de ETL. O resultado de uma limpeza poderia ser repassado aos sistemas operacionais para melhorar a sua precisão e reduzir os próximos trabalhos de limpeza (DAYAL, 2009).

2.3.2 Metadados e DW 2.0

Uma das principais características de um DW 2.0 é o uso de metadados, que nas primeiras gerações de Data Warehouses não eram utilizados ou eram deixados em segundo plano.

Segundo Inmon, Strauss e Neushloss (2007), os metadados têm um papel especial na implementação de um DW 2.0. Metadados diferentes são necessários para cada setor de um DW 2.0. Existem metadados para o setor interativo, para o setor integrado, para o *Near Line Sector* e para o setor de arquivos.

Há dois tipos de estruturas para metadados, uma para dados estruturados e outra para dados não estruturados. Para dados não estruturados, existem dois tipos de metadados – *enterprise* (ou geral) e *local* (ou específica). Para dados estruturados, existem três níveis – *enterprise*, *local* e *business* ou *technical*.

Metadados locais são aqueles que existem dentro de ferramentas que são úteis para descrever os dados utilizados na ferramenta. Por exemplo, metadados de um processo de ETL são sobre as fontes de dados, os alvos e as transformações utilizadas.

Os metadados *enterprise* têm vários relacionamentos diferentes com os metadados locais. Uma dessas relações é a semântica. Em relacionamentos semânticos, o *enterprise* descreve um termo global para a organização. Então o uso local daquele termo é descrito, incluindo um ponteiro para o sistema local em que o termo é encontrado. Outro relacionamento com metadados locais é a definição de áreas de assuntos. Por exemplo, se tivéssemos três sistemas locais, onde cada um deles mantivesse dados de clientes, um deles pode armazenar nome e endereço, outro idade e preferência de compras, enquanto outro armazena salário e nível de educação. A Figura 7 apresenta o exemplo descrito anteriormente.

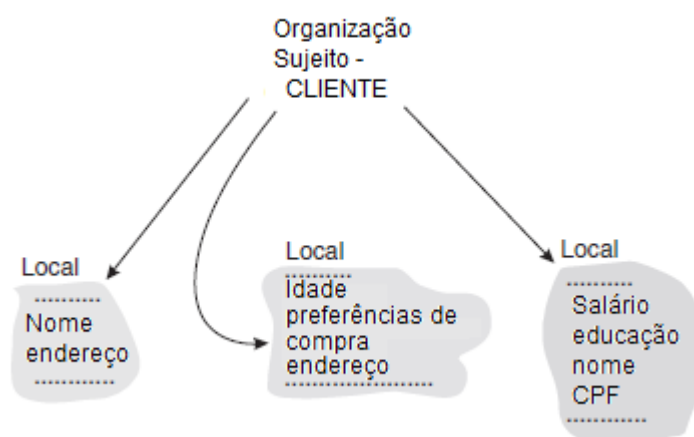


Figura 7 - Relacionamento dos metadados na organização
Fonte: adaptado de Inmon, Strauss e Neushloss (2007)

2.3.3 DW 2.0 e dados não estruturados

De acordo com Inmon, Strauss e Neushloss (2007), estima-se que mais de 80% dos dados das organizações são textos não estruturados. Infelizmente, a tecnologia usada nos computadores atuais é dedicada a manipular dados estruturados. O resultado disso é que informações valiosas não estão sendo usadas para a tomada de decisões nas empresas.

A arquitetura da próxima geração de Data Warehouses reconhece que existem informações valiosas em dados textuais não estruturados. O objetivo é utilizar várias técnicas de recuperação de informação sobre dados textuais a fim de preparar os dados para processamento analítico.

Keith, Kaser e Lemire (2005) propuseram o uso de um (Data) Warehouse of Words (WoW), que seria construído em um processo de ETL, processando o texto e agregando dados de várias fontes. Um WoW armazena seus dados em cubos OLAP, permitindo ao usuário executar consultas em alto nível.

2.4 Considerações finais

Neste capítulo, foram levantadas informações relacionadas ao modelo proposto no capítulo seguinte, enfatizando-se as diferenças na forma de trabalho entre dados não estruturados e estruturados, visto que o foco principal do trabalho está na integração dos dois tipos. Outro assunto amplamente abordado neste capítulo foi o DW, que também faz parte do modelo proposto. Foi despendida atenção especial ao processo de ETL, que será demonstrado no Capítulo 4 para avaliar o modelo proposto.

3 Processo Proposto

Com base na revisão bibliográfica feita no capítulo 2, este trabalho propõe uma extensão ao processo de ETL conhecido atualmente, porém integrando dados estruturados a dados não estruturados com a finalidade de recuperar informações das bases textuais e relacioná-las com dados das bases estruturadas. Esses relacionamentos serão armazenados em um modelo de DW, que tem como objetivo possibilitar análises complexas sobre essas relações, inclusive permitindo o uso de técnicas de Data Mining e Text Mining para obtenção de informações previamente desconhecidas.

3.1 Processo de ETL

O processo proposto, que pode ser visto na

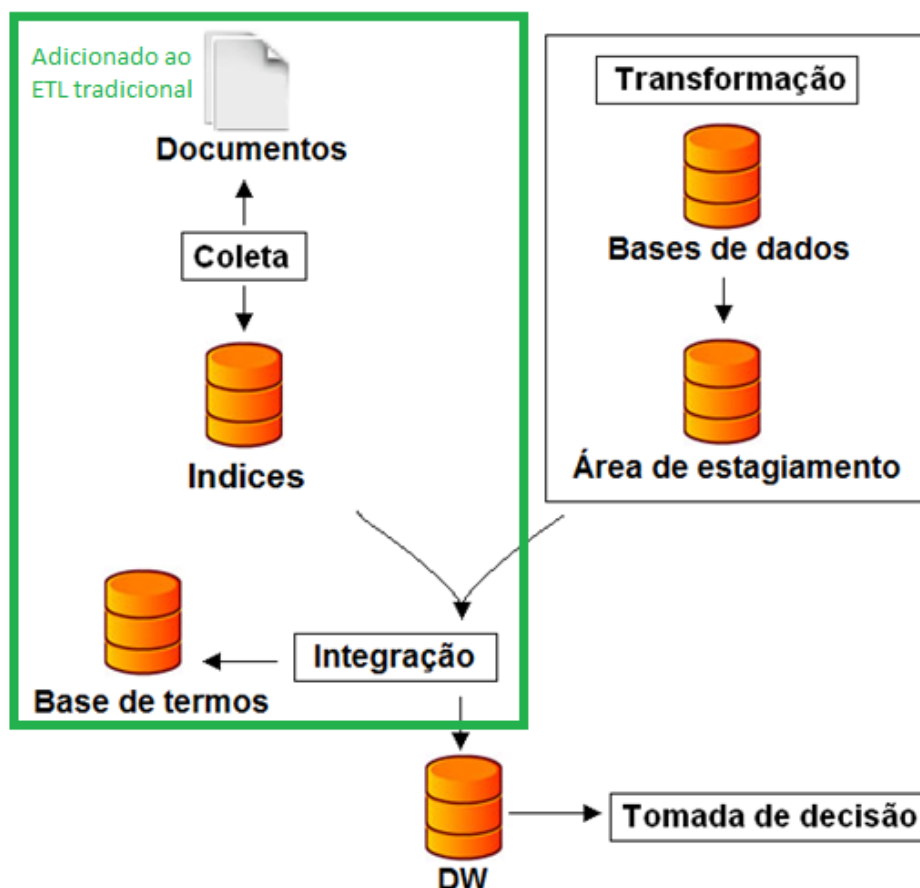


Figura 8, é uma extensão do processo de ETL padrão. As principais diferenças se encontram na inclusão dos documentos não estruturados separadamente das bases estruturadas e da inclusão da base de termos e da fase de integração.

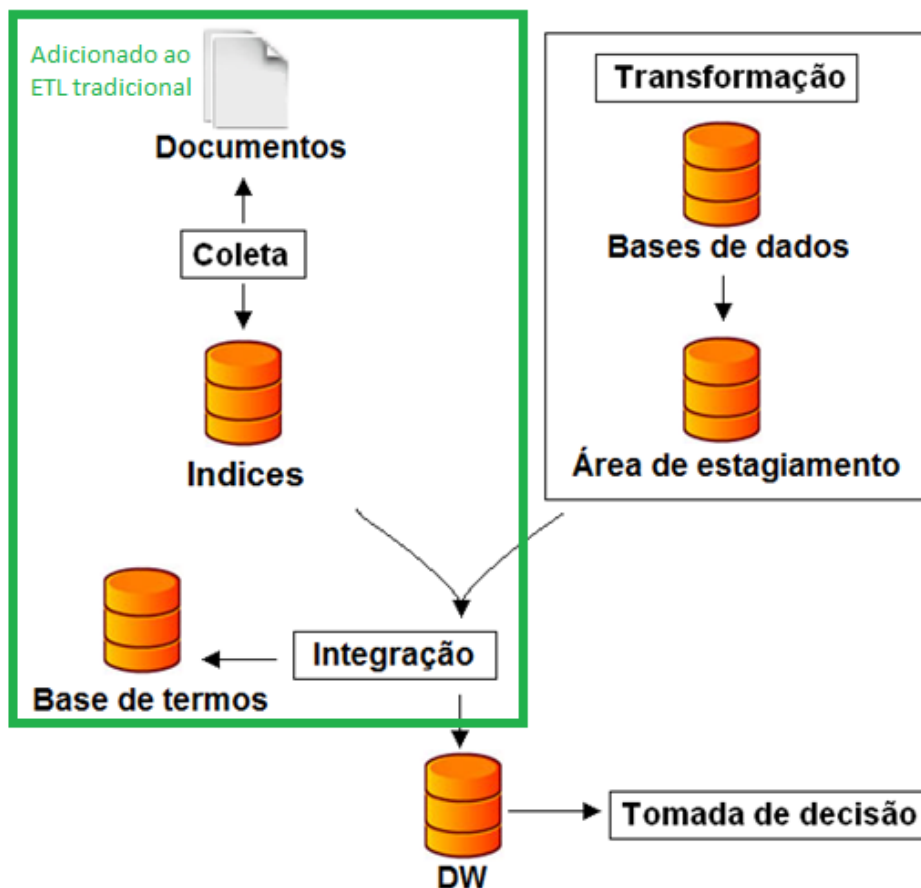


Figura 8 - Proposta para o processo de ETL 2.0

3.2 Coleta

A fase de coleta dos dados é geralmente complexa, e no processo de ETL tradicional já é previsto que os dados podem vir de diferentes fontes de informação. No caso deste trabalho, estão sendo incluídos documentos não estruturados a essa lista. Considerando-se que a coleta dos dados não estruturados deve utilizar uma abordagem diferenciada, adicionou-se o processo chamado de “Collector”, que se refere à forma diferenciada para a coleta dessas informações. No caso deste trabalho, foi utilizado um *plugin* de “*crawling*” que realiza a coleta dessas informações.

Assim que os dados estão prontos e já foram mapeados nas diferentes fontes de informações que serão utilizadas, esses dados precisam ser coletados de fato. Para isso, deve-se utilizar uma ferramenta que possa ler os diferentes tipos de dados e integrá-los em um mesmo formato.

Tanto para dados estruturados quanto para dados não estruturados deve-se utilizar um sistema que irá coletar esses dados e carregá-los para uma área de estagiamento ou para a memória. Essa área de estagiamento deve ser a mesma, de forma que qualquer dado em comum seja integrado já nessa etapa. Pode-se notar que, mesmo com dados em comum entre as duas fontes de informação, não seria possível efetuar qualquer tipo de análise nos dados não estruturados nesse momento.

3.3 Base de termos

Para ser possível efetuar qualquer tipo de análise e recuperar informações das bases textuais com relação às bases estruturadas, é necessário reconhecer padrões que se encontram nos textos, mais especificamente padrões nomeados chamados de entidades. Uma entidade caracteriza-se com um padrão textual que está vinculado (nomeado) a determinada classe, por exemplo, pessoa ou instituição.

Para isso, será utilizada uma base com entidades, ou seja, uma taxonomia/ontologia do domínio de análise que represente as entidades de interesse e, possivelmente, que estão presentes nos documentos coletados.

Essa base deve ser construída a partir de termos que possivelmente tenham alguma relevância em relação ao domínio, o que pode requerer um conhecimento prévio deste. A base pode ser construída manualmente, isto é, um humano pode inserir cada termo que achar relevante, processo extremamente lento e custoso, ou pode-se também construir essa base a partir da própria base de documentos, utilizando-se técnicas de reconhecimento de entidades e/ou de coocorrência de termos (SCHUTZE, 1996).

A base pode ser persistida e acessada a partir de um tesauro, uma ontologia, uma taxonomia ou outro modelo que for conveniente, não existindo um padrão para tal armazenamento.

3.4 Integração

A fase de integração é o que de fato produz o DW. Para o processo de integração, deve-se ter os dados não estruturados e estruturados em uma mesma área de estagiamento. Para que seja possível a análise dos dados não estruturados,

utiliza-se a base de termos para encontrar os termos presentes em cada documento. Para cada termo encontrado, armazenam-se as informações dessa relação no DW, ou seja, cada entidade gera uma série de relações, uma para cada entidade mapeada a partir dos dados estruturados contidos no documento onde foi realizada a consulta.

3.5 Modelo do DW

Com os dados provenientes do passo de integração é construído o DW, que é o artefato final do processo de ETL. O modelo do DW proposto visa permitir a recuperação de informações referentes à correlação de entidades em diferentes documentos.

Na Figura 9, é descrito o modelo entidade-relacionamento que deve ser seguido para que seja possível a análise dos termos. Esse modelo pode ser utilizado para efetuar consultas medindo a frequência dos termos (utilizando-se a medida FREQUENCIA na tabela FATO_FREQUENCIA) com relação ao tempo (inserindo-se uma junção entre a tabela FATO_FREQUENCIA e a tabela DIMENSAO_TEMPO agrupando-se os dados pelo campo DATE) e a relação entre termos e outras entidades mapeadas nos processos anteriores (com a medida FREQUENCIA_CONJUNTA da tabela FATO_RELACAO).

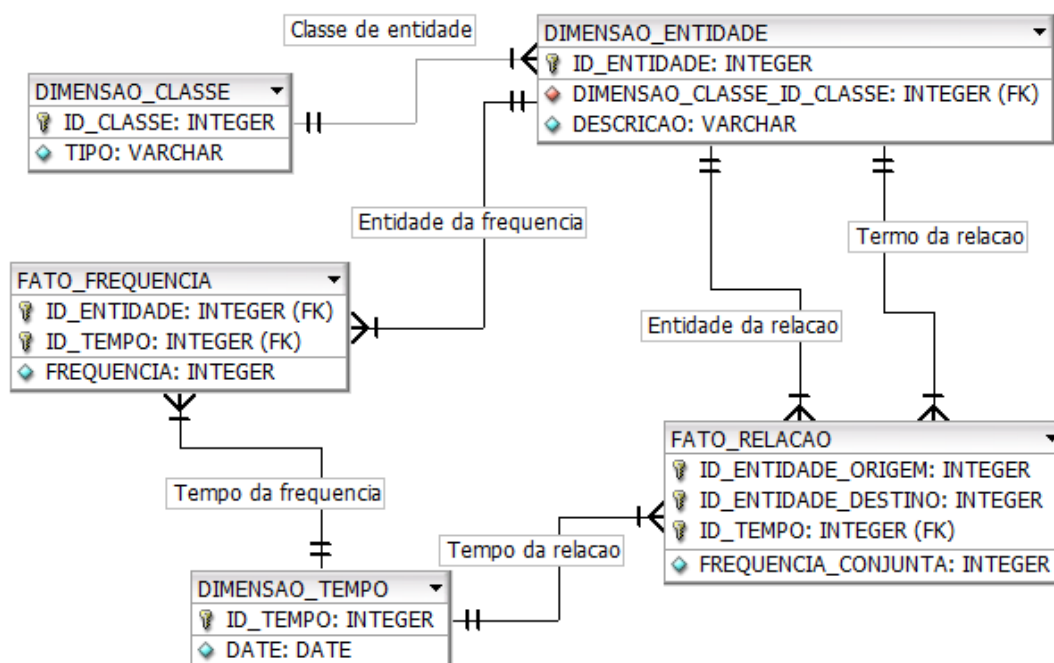


Figura 9 - Modelo do DW

Nas seções seguintes, o modelo será descrito em detalhes, com seções específicas para cada entidade.

3.5.1 Dimensão Classe

A dimensão *classe* deve conter os tipos aos quais as entidades podem pertencer, os projetos, os assuntos, os departamentos ou qualquer outro dado existente na base para o qual se deseja relacionar com os termos.

Uma classe que sempre deve existir é a classe *termo*. Isso é necessário porque os termos são as entidades usadas para encontrar a correlação entre outras entidades.

ID_CLASSE: chave primária identificando o tipo da classe.

TIPO: descrição do tipo da classe.

3.5.2 Dimensão Entidade

A dimensão *entidade* deve conter todas as entidades existentes no modelo, as quais podem pertencer a uma classe definida anteriormente. Por exemplo, poderiam existir as entidades CTC pertencentes à classe *centro*, CFH pertencente à mesma classe ou INE pertencente à classe *departamento*.

Todos os termos definidos na base de termos estarão na dimensão *entidade* e farão parte da classe *termo*.

ID_ENTIDADE: chave primária identificando a entidade.

ID_CLASSE: chave primária da dimensão *classe* correspondente.

DESCRICA0: descrição da entidade.

3.5.3 Dimensão Tempo

A dimensão *tempo* contém os dados referentes ao tempo dos documentos, com a granularidade correspondente.

ID_TEMPO: chave primária identificando o tempo.

ANO, MÊS, DIA: dependendo da granularidade utilizada para definir essa dimensão, podemos ter diferentes atributos relacionados a ela.

3.5.4 Fato Frequência

O fato *frequência* contém os dados referentes à frequência de uma entidade em relação ao tempo. Com informações retiradas dessa tabela, é possível gerar gráficos que mostram a evolução da frequência de determinada entidade no tempo. Por exemplo, poder-se-ia mostrar quantas vezes o termo “Java” foi encontrado nos últimos anos.

ID_ENTIDADE: chave primária da dimensão *entidade* correspondente.

ID_TEMPO: chave primária da dimensão *tempo* correspondente.

FREQUENCIA: quantidade de documentos que contêm essa entidade em determinado tempo.

3.5.5 Fato Relação

O fato *relação* contém os dados referentes à relação entre duas entidades, uma delas sendo necessariamente um termo. Com informações retiradas desta tabela, é possível gerar gráficos de relação conjunta entre diferentes entidades. Por exemplo, poder-se-ia mostrar a frequência em relação ao tempo de diferentes termos no mesmo gráfico.

ID_ENTIDADE_ORIGEM: chave primária da dimensão *entidade* contendo uma das entidades da relação.

ID_ENTIDADE_DESTINO: chave primária da dimensão *entidade* que contém a outra entidade da relação.

ID_TEMPO: chave primária da dimensão *tempo* correspondente.

Cada documento textual contém vários termos e pode pertencer a várias outras entidades. A cada termo encontrado no documento, adiciona-se uma entrada para cada uma das outras entidades às quais o documento pertence, como, por exemplo, uma página da web que pertence à categoria “notícias” e à categoria “fórum” em que encontramos a palavra “Java”. Nesse cenário, inserem-se duas entradas na tabela com o mesmo ID_TERMO e o mesmo ID_TEMPO. A única diferença seria o ID_ENTIDADE, um apontando para a categoria “notícias” e outra apontando para a categoria “fórum”.

Os dados gerados com as tabelas FATO_RELACAO e FATO_FREQUENCIA são diferentes devido à possibilidade de a tabela FATO_RELACAO duplicar a frequência do mesmo termo. Por causa dessa característica, foi escolhida a abordagem em que se utilizam duas tabelas.

3.6 Análise

A fase de análise visa recuperar informações do DW sobre os dados estruturados e não estruturados. Com o DW construído, será possível a visualização de informações antes impossíveis de serem mensuradas. Tanto para dados estruturados quanto para dados não estruturados pode-se relacionar os dois tipos de dados, como, por exemplo, a frequência de um termo no tempo ou a frequência de um termo em um departamento. Isso é possível caso algum conceito presente nos

dados não estruturados possa ser relacionado com dados da base estruturada. Por exemplo, se uma pessoa menciona que está envolvida no projeto P e este projeto faz parte do departamento D, pode-se inferir que essa pessoa está de alguma maneira relacionada com o departamento D, ainda que sem conhecer o tipo do relacionamento. Esse tipo de relação é extremamente útil para sistemas de apoio à tomada de decisão, já que quanto mais informações forem integradas, mais conhecimento pode ser adquirido (PEREIRA; REZENDE; ABREU, 2000; SAVI; AMARAL; ROZENFELD, 2003). Portanto, pode-se realizar dois tipos de análise sobre o modelo proposto:

- Análise da frequência de uma entidade: utilizando-se o fato *frequência*, é possível recuperar informações sobre a frequência de qualquer entidade em relação ao tempo, e com isso fazer a comparação de frequência entre entidades.
- Análise da relação de entidades: utilizando-se o fato *relação* é possível recuperar informações referentes à relação entre entidades e sua frequência conjunta. Essa frequência pode ser utilizada para gerar matrizes de correlação (GONÇALVES, 2006), o que permitiria a análise da relação entre dois documentos (termos em comum) e a relação entre termos (aparecendo em documentos comuns).

Esses conceitos são amplos, e existe uma vasta literatura sobre o assunto, porém, tais conteúdos vão além do escopo deste trabalho.

3.7 Considerações finais

Este capítulo apresentou a proposta de extensão ao processo de ETL, chamado de ETL 2.0. O principal passo do processo é a parte de integração, em que se integram dados estruturados a dados não estruturados. Além desse passo, o processo inclui outros passos necessários para que se atinjam os objetivos do trabalho, entre eles a coleta de dados não estruturados, a criação e o acesso a uma base de termos, a criação do DW de acordo com o modelo definido e, por fim, a análise dos dados já inseridos no DW de acordo com as técnicas de BI descritas na literatura. Nesse sentido, a utilização desse processo visa fornecer uma base de dados pronta para ser consumida por aplicações de apoio à tomada de decisão.

4 APRESENTAÇÃO DOS RESULTADOS

Para a avaliação da proposta do capítulo anterior, foi feito um estudo de caso utilizando-se dados reais que serão apresentados neste capítulo. Primeiramente, serão informadas e descritas as fontes de informação primárias e secundárias que contêm dados estruturados e não estruturados. Em seguida, são apresentados os componentes tecnológicos desenvolvidos. Os passos do processo de ETL são descritos logo após, e por fim é feita uma análise sobre os dados inseridos no DW.

4.1. Fontes de informação primárias e secundárias

Nesta seção, são apresentadas as bases utilizadas para a criação do DW. Uma base não estruturada de e-mails foi integrada a uma base de departamentos fictícia e integrada ao DW com o modelo de acordo com o capítulo anterior.

4.1.1 Base de dados utilizada

Para a validação do trabalho, foi utilizada a base de dados de e-mails *Email Dataset*¹, que contém cerca de 500.000 mensagens entre cerca de 150 pessoas. Essas mensagens possuem uma série de outras informações, mas as informações que foram utilizadas no trabalho são as seguintes:

- data: data de envio da mensagem. Foram armazenados somente o mês e o ano do envio;
- remetente: o e-mail do remetente da mensagem;
- destinatário(s): o(s) e-mail(s) dos destinatários da mensagem;
- conteúdo: o conteúdo da mensagem.

¹ Enron Email DataSet - <http://www.cs.cmu.edu/~enron/>

4.1.2 Modelo do DW

Considerando-se o modelo proposto no capítulo três, algumas considerações devem ser levadas em conta.

A dimensão *Tempo* contém a data de envio da mensagem. Foi utilizado como grão o mês do envio da mensagem, portanto, sendo armazenado o mês e o ano do envio da mensagem.

A dimensão *Classe* tem os tipos “Termo”, “Pessoa” e “Departamento”.

A dimensão *Entidade* armazena todos os termos da base de termos, todas as pessoas e todos os departamentos. Isso inclui o “remetente” e os “destinatários” das mensagens armazenadas na dimensão *Entidade* com o tipo “pessoa”.

Para enriquecer ainda mais os dados dos e-mails, foram criados cinco departamentos fictícios, e cada pessoa foi atribuída a um desses departamentos. Os e-mails enviados por essa pessoa também têm como atributo o departamento ao qual ela está inserida. Com esse atributo a mais, foi possível fazer análises mais complexas sobre os dados. Mesmo que essas informações não reflitam a realidade, ajudam na comprovação do modelo.

A tabela fato *Relação* contém informações sobre a relação, ou seja, a frequência conjunta entre entidades do tipo palavra-chave, departamento ou pessoa. Para cada relação, entre as entidades uma linha foi adicionada à tabela FATO_RELACAO.

A tabela fato *Frequência* contém informações sobre a frequência de determinada entidade em relação ao tempo. Para isso, foi adicionada uma linha para cada entidade em algum determinado tempo. Caso o tempo já exista no momento da coleta de um documento, essa tabela deve ser apenas atualizada com uma soma no campo “*frequencia*”.

4.2. Componentes tecnológicos desenvolvidos

Nesta seção, são descritos os componentes desenvolvidos. Primeiramente, foi desenvolvido um *plugin* para a coleta de informações de dados não estruturados. Em seguida, foram desenvolvidos *plugins* para construir e ler índices textuais. Por

fim, foi desenvolvido um *plugin* para integração de dados estruturados e não estruturados.

4.2.1 Collector

O *plugin* **Collector** foi desenvolvido utilizando-se como base o componente “Retriever” desenvolvido no Instituto Stela. O Retriever é um *crawler open-source* desenvolvido com a tecnologia Java que permite a coleta e a manipulação de documentos a partir de vários tipos de protocolos (ex.: HTTP, smb) e formatos (HTML, XML, PDF, DOC, etc.). Também é possível fazer a coleta de documentos compartilhados a partir de LAN, da Web e de outras fontes. (<http://dev.stela.org.br/retriever/>).

O *plugin* **Collector** foi desenvolvido em tecnologia Java, utilizando-se como exemplo outros *plugins* específicos para o Kettle. Para que o *plugin* possa ser utilizado, deve-se adicionar os arquivos listados na Tabela 1 na pasta *plugins/steps/CollectorPlugin* a partir da raiz da instalação do Kettle. Feito isso, o *plugin* deve aparecer como um passo que pode ser usado em transformações.

Arquivos dentro de <i>plugins/steps/CollectorPlugin</i>	Descrição
dependencies/icu4j-4_0.jar	Biblioteca necessária para o projeto retriever
dependencies/jcifs-1.3.1.jar	Biblioteca necessária para o projeto retriever
dependencies/retriever-0.17.0.jar	Fontes do projeto de código aberto com o coletor
collector.jar	Fontes das classes do <i>plugin</i>
CPI.png	Imagem que será mostrada na interface do Kettle
DefaultFilters.jar	Fontes dos filtros disponibilizados por padrão na interface do <i>plugin</i>
plugin.properties	Propriedades lidas pelo <i>plugin</i> informando quais filtros serão utilizados
plugin.xml	Propriedades lidas pelo Kettle para definir como o <i>plugin</i> é apresentado e quais as dependências de bibliotecas

Tabela 1 - Lista dos arquivos necessários para o *plugin* Collector

O **Collector** foi acoplado como componente de entrada de dados e pode ser encontrado nessa categoria dentro da interface do Kettle. O **Collector** não pode receber dados de outros passos, ou seja, recebe dados de uma fonte externa, nesse caso, de documentos que serão coletados. Para o seu funcionamento, são

necessárias algumas configurações, explicadas em detalhes na Figura 10 e descritas abaixo em mais detalhes.

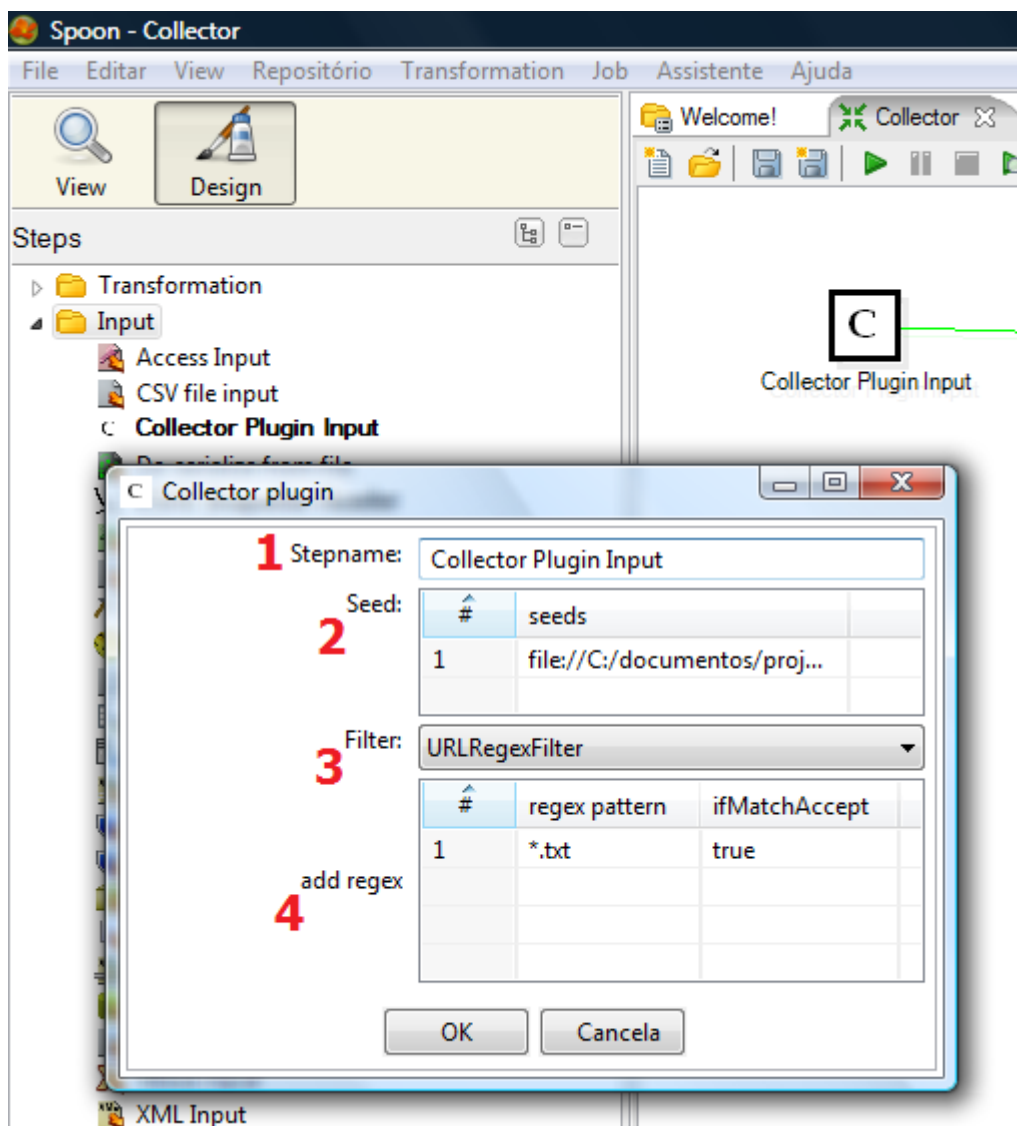


Figura 10 - Configuração do plugin "Collector"

1 – Stepname: nome do passo mostrado na interface do Kettle para essa transformação.

2 – Seed: neste item, são informadas as sementes nas quais o **Collector** irá procurar os dados para coletar. Pode-se adicionar mais de uma semente. O **Collector** vai iniciar a busca por documentos a serem coletados nos caminhos informados como sementes. Os diretórios-filhos dos informados também serão incluídos na busca, e assim por diante, até que não haja mais diretórios para serem buscados. Para cada documento encontrado, o **Collector** envia as informações desse documento para os passos seguintes na transformação.

3 – Filter: neste item, deve-se escolher se será utilizado um filtro para os documentos a serem coletados. As opções são NoFilter (sem filtro), URLRegexFilter (um filtro que compara uma expressão regular com a URL do documento) ou URLLiteralFilter (um filtro que compara um texto com a URL do documento).

4 – Configuração dos filtros: esta opção só está visível ao usuário caso algum filtro esteja selecionado. Nessa opção, são configurados os padrões para a comparação com as URLs dos documentos e se o filtro será utilizado para aceitar ou rejeitar os documentos.

A partir da configuração do *plugin Collector*, criam-se condições para que a ferramenta *open source* utilizada, neste caso Kettle, possa lidar com dados não estruturados. Atualmente, a ferramenta disponibiliza somente entradas para um único documento de texto, tornando inviável o processo de ETL de uma coleção de documentos, caso comum em qualquer organização.

4.2.2 Lucene Output Plugin

Após a criação do *plugin* de coleta, torna-se necessária a persistência desses documentos. Para uma busca eficiente em documentos textuais, é preciso indexá-los.

Para isso, desenvolveu-se um *plugin* para a ferramenta Kettle voltado à indexação de conteúdo textual. Foi utilizada a tecnologia Java, tendo como exemplo outros *plugins* desenvolvidos para o Kettle. Para que esse *plugin* possa ser corretamente utilizado, deve-se disponibilizar e configurar os arquivos listados na Tabela 2 na pasta `plugins/steps/LuceneOutputPlugin`, a partir da raiz da instalação do Kettle. Após isso, o *plugin* está disponibilizado como um passo que pode ser utilizado em transformações.

Arquivos dentro de <i>plugins/steps/LuceneOutputPlugin</i>	Descrição
Analysis.jar	Fontes dos analisadores padrões utilizados pelo <i>Lucene</i> .
dependencies/jcifs-1.3.1.jar	Biblioteca necessária para o projeto retriever
lucene.jar	Fontes das classes do <i>plugin</i>
Lucene-core-2.3.2.jar	Fontes do <i>Lucene</i>
LPO.png	Imagem que será mostrada na interface do Kettle
plugin.properties	Propriedades lidas pelo <i>plugin</i> informando quais analisadores serão utilizados e quais classes

	que os implementam
plugin.xml	Propriedades lidas pelo Kettle para definir como o plugin é apresentado e quais as dependências de bibliotecas

Tabela 2 - Lista dos arquivos necessários para o plugin *LuceneOutputPlugin*

O plugin utiliza a biblioteca de indexação *Lucene*, que é de código aberto e mantida pela *Apache Software Foundation*¹. De modo geral, a biblioteca *Lucene* disponibiliza bibliotecas para indexação de documentos e busca sobre os índices.

O *LuceneOutputPlugin* foi desenvolvido como um *Output* (Saída) de dados e pode ser encontrado nessa categoria de passos na interface do Kettle. O *LuceneOutputPlugin* não pode enviar dados a outros passos, só pode ser utilizado como saída. Para o seu funcionamento, são necessárias algumas configurações, as quais são explicadas em detalhes na Figura 11.

¹ Projeto Apache Lucene - <<http://lucene.apache.org/>>.

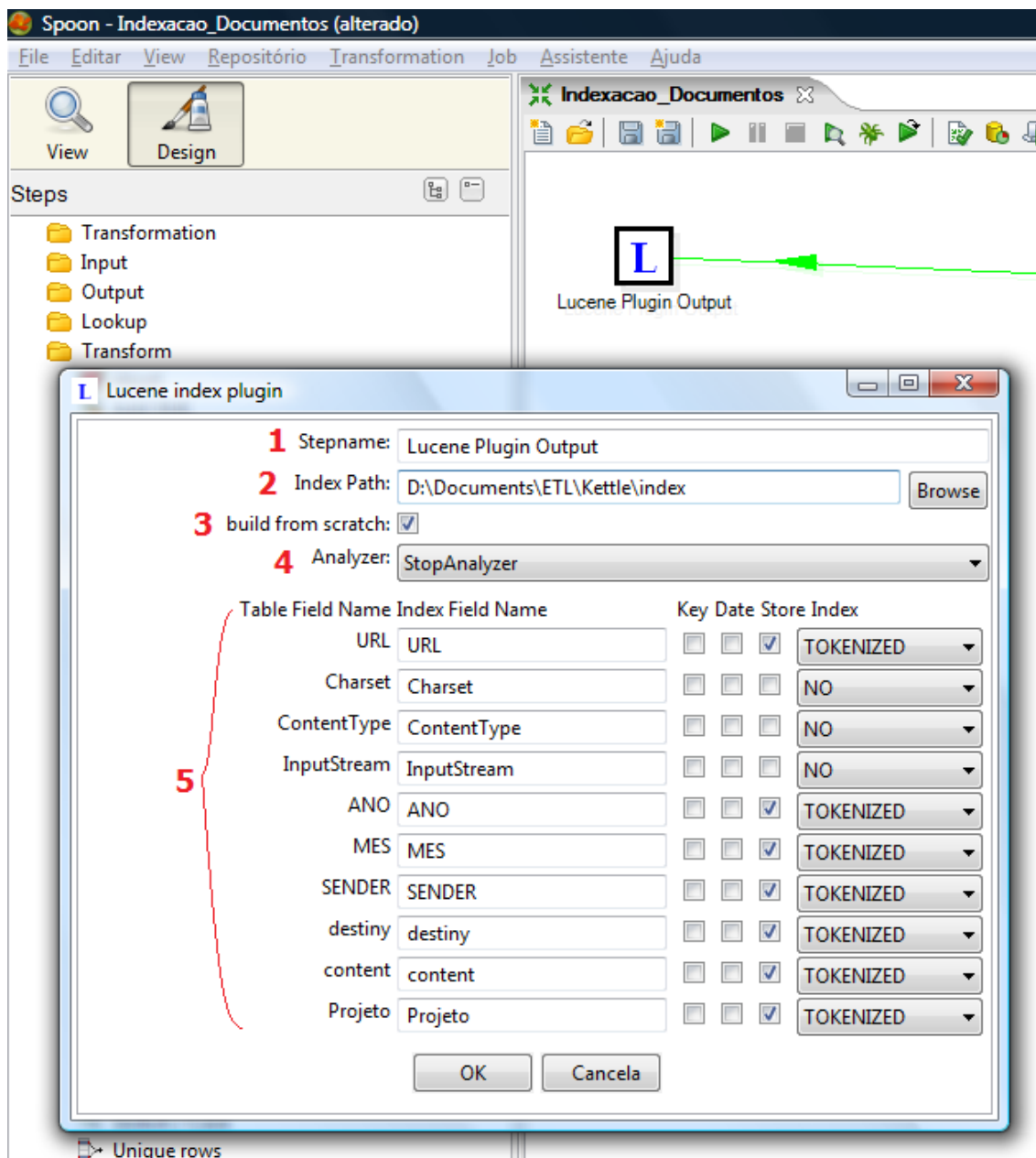


Figura 11 - Configuração do LuceneOutputPlugin

1 – Stepname: nome do passo mostrado na interface do Kettle para essa transformação.

2 – Index Path: o caminho onde serão criados os arquivos contendo o índice Lucene.

3 – Build from scratch: indica se o índice deve ser criado completamente, se estiver selecionado, ou se será incrementado um índice já existente, caso não esteja selecionado.

4 – Analyzer: deve-se selecionar o analisador que será usado para indexar os documentos. Esse analisador define a utilização de algumas técnicas de recuperação de informação, como *stemming*, remoção de *stop-words* e outras. Mais informações podem ser encontradas na documentação do projeto Lucene.

5 - Configuração dos campos indexados: cada coluna que está sendo enviada a partir dos passos anteriores é automaticamente identificada como um campo do índice Lucene. Nessa etapa da configuração, pode-se definir qual o nome de cada campo e suas características, entre elas:

5.1 – Key: se o campo será utilizado como chave para indexações futuras, somente um campo pode ser do tipo chave. Caso uma indexação seja feita posteriormente sobre o mesmo índice, os documentos com a mesma chave de um documento já indexado serão apenas atualizados e não serão adicionados ao índice.

5.2 – Date: se o campo é a data de controle, somente um campo pode ser do tipo *date*. Caso uma indexação seja feita posteriormente sobre o mesmo índice, os documentos que tenham o valor no campo *date* anterior à última data indexada nesse mesmo campo não serão indexados.

5.3 – Store: indica se o campo estará disponível para leitura em seu formato original. Isso é necessário algumas vezes, já que os analisadores executam algoritmos de recuperação de informação, tais como *stemming*, que tornam os termos indexados diferentes dos termos originais.

5.4 – Index: são oferecidas três opções (NO, TOKENIZED e UN_TOKENIZED). NO apenas armazena o valor original do campo, TOKENIZED indica que os termos serão indexados separadamente, sendo utilizado o analisador escolhido para executar alguns processos sobre esses termos. UN_TOKENIZED indica que o campo será indexado, mas sem utilizar os processos do analisador.

4.2.3 Integração (LuceneInputPlugin)

Para a execução do processo de integração, não foi possível a utilização apenas dos passos pré-configurados na ferramenta Kettle. Para contornar essa dificuldade, foi desenvolvido o *plugin* de integração, que objetiva a integração dos dados estruturados e não estruturados de modo a gerar como saída o modelo DW.

O *plugin* foi desenvolvido utilizando a tecnologia Java, tendo como exemplo outros *plugins* criados para o Kettle. Para que o *plugin* possa ser utilizado, deve-se adicionar os arquivos listados na Tabela 3, na pasta `plugins/steps/LuceneInputPlugin`, a partir da raiz da instalação do Kettle. Feito isso, o *plugin* deve aparecer como um passo que pode ser utilizado em transformações.

Arquivos dentro de <i>plugins/steps/LuceneOutputPlugin</i>	Descrição
LuceneInput.jar	Fontes das classes do <i>plugin</i>
Lucene-core-2.3.2.jar	Fontes do <i>Lucene</i>
IPT.png	Imagem que será mostrada na interface do Kettle
plugin.xml	Propriedades lidas pelo Kettle para definir como o <i>plugin</i> é apresentado e quais as dependências de bibliotecas

Tabela 3 - Lista dos arquivos necessários para o *plugin* LuceneInputPlugin

O *plugin* *LuceneInputPlugin* foi desenvolvido como um *Input* e pode ser encontrado nessa categoria dentro da ferramenta Kettle. Ele faz a leitura de um índice da ferramenta Lucene e envia dados para outro passo do Kettle. Não pode ser usado como *output* e não pode receber os dados de outro passo. O *plugin* faz a leitura de índices gerados com a biblioteca Lucene.

O *plugin* *LuceneInputPlugin* não possui interface gráfica. Para cada transformação diferente em que ele foi utilizado, alguns parâmetros foram modificados no código, sendo o *plugin* recompilado. Caso o projeto seja continuado, poderia ser criada uma interface gráfica que permitisse a configuração do índice a ser lido e quais as informações devem ser passadas para os próximos passos.

O *plugin* *LuceneInputPlugin* lê a lista de termos pré-selecionados e faz uma busca em todos os documentos indexados. Para cada documento encontrado, ele envia para os próximos passos uma linha com as informações dos relacionamentos entre cada entidade e o termo buscado. Por exemplo, se o termo “java” for encontrado em um documento de e-mail com o remetente “andre”, que faz parte do departamento “ine”, e o destinatário “zorzo”, serão enviadas para o próximo passo da transformação duas linhas de dados com as informações do relacionamento entre a entidade “andre” e o termo “java”, outra linha com o relacionamento da entidade “zorzo” com o termo “java” e uma terceira linha com o relacionamento da entidade “ine” com o termo “java”.

4.3. Apresentação sobre o processo ETL

Para o processo de ETL, foi utilizada a ferramenta Kettle. Para a carga de cada uma das dimensões, foi criada uma transformação. Nessa seção, são explicados todos os passos que foram seguidos para a criação das dimensões e tabelas de fato.

4.3.1 Dimensão Tempo

Para a carga da dimensão *tempo*, os dados foram analisados manualmente, e foi verificado o intervalo de tempo em que foram enviadas as mensagens. Foi gerado um comando SQL para criar a tabela, a qual foi inserida manualmente utilizando-se a aplicação “MySQL Query Browser”. A dimensão foi criada contendo todos os meses dos anos entre 1970 e 2007, o que resultou em 456 possíveis datas. O comando SQL utilizado foi “INSERT INTO dimensao_tempo (MES, ANO) VALUES (?, ?)”, sendo os valores apresentados na Tabela Tabela 4.

Mês	Ano
Jan	2000
Feb	2000
Mar	2000

Tabela 4 - Exemplo de valores utilizados na carga da dimensão tempo

4.3.2 Dimensão Classe

A dimensão *classe* foi iniciada manualmente a partir de um comando SQL com os três possíveis tipos para as entidades. “Termo”, “Departamento” e “Pessoa”. O comando SQL utilizado foi “INSERT INTO dimensao_classe (TIPO) VALUES ("Termo"), ("Departamento"), ("Pessoa")”.

4.3.3 Dimensão Entidade

A dimensão *entidade* foi criada em três passos. Primeiro, foi feita a carga das entidades do tipo *Pessoa* contendo os e-mails dos usuários da lista. Em seguida,

realizou-se a carga das entidades da classe *Departamento* da qual cada usuário estaria participando e, por último, a carga de entidades da classe *Termo*, utilizadas no processo de sumarização das informações não estruturadas.

4.3.3.1 Carga dos usuários

Para realizar a carga dos usuários, foi criada uma *transformation* para a ferramenta *Kettle*. Essa transformação e seus passos podem ser vistos na Figura 12.

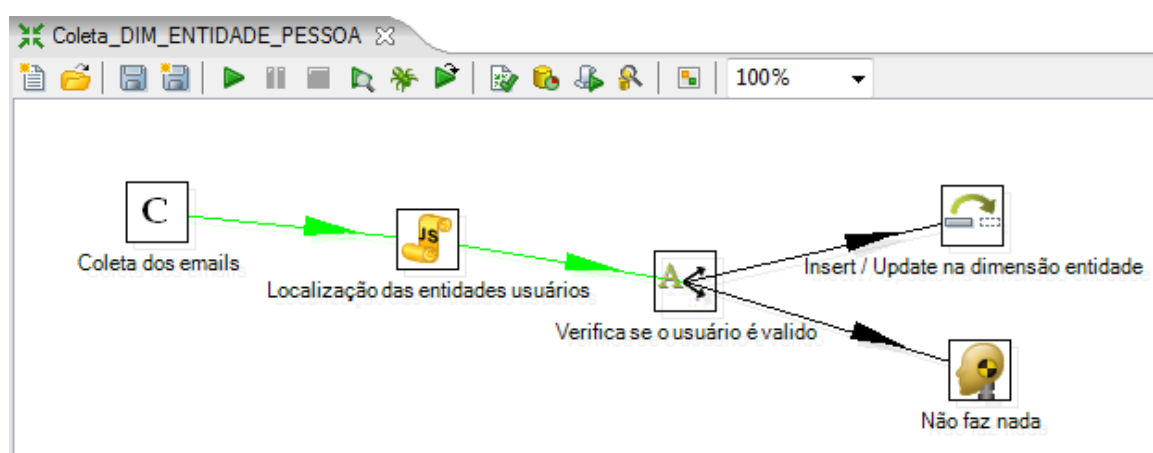


Figura 12 - Carga da dimensão entidade (usuários)

Coleta dos e-mails – Este passo apenas coleta todos os documentos da base e passa os e-mails sem nenhuma alteração para os próximos passos.

Localização das entidades usuários – Este passo faz o *parse* dos documentos e separa apenas os e-mails dos usuários que façam parte do domínio “@enron.com”, ou seja, qualquer e-mail que possa ser encontrado utilizando-se a expressão regular “*@enron.com”, e envia para os próximos passos. Caso o e-mail não seja válido, é enviado um valor nulo para ser tratado pelo passo seguinte.

Verifica se o usuário é válido – Este passo apenas direciona o *output* do usuário validado no passo anterior. Caso receba o valor nulo, o usuário não é válido, portanto apenas envia os dados para o passo “*Não faz nada*”. Se for válido (qualquer outro valor), envia para o passo “*Insert / Update na dimensão entidade*”.

Insert/Update na dimensão entidade – Este passo insere o e-mail encontrado na base. Caso o e-mail já tenha sido inserido, ele é apenas ignorado.

A transformação foi realizada uma vez para cada departamento, de forma a não sobrecarregar o sistema. No total, foram encontrados e inseridos na tabela “dimensao_entidade” 6.371 e-mails.

4.3.3.2 Carga dos departamentos

As entidades representando os cinco departamentos fictícios foram carregadas manualmente por meio da execução de um comando SQL com o id da classe dos departamentos e também com o nome destes. O comando SQL utilizado foi “INSERT INTO dimensao_entidade (id_classe, descricao) VALUES (3, 'Alpha'), (3, 'Bravo') , (3, 'Charlie') , (3, 'Delta') , (3, 'Eco)’”.

4.3.3.3 Carga dos termos

Os termos utilizados para relacionar as entidades foram gerados a partir de uma análise manual sobre alguns dos e-mails da base. Foram escolhidos termos sem qualquer processo de extração de informações ou entidades, inclusive, sem qualquer ajuda computacional. Essa base de termos foi carregada na tabela “dimensao_entidade” utilizando-se o script SQL “INSERT INTO dimensao_entidade (id_classe, descricao) VALUES (1, “termo”), substituindo o “termo” por diversos valores, alguns apresentados na Tabela Tabela 5. A lista completa dos termos utilizados pode ser encontrada no Apêndice A.

Id_classe	Descrição
1	bankrupt
1	bankruptcy
1	oil

Tabela 5 - Exemplo de valores dos termos utilizados na carga da dimensão entidade

4.3.4 Fato Relação

A tabela “fato_relacao” foi gerada a partir de uma transformação da ferramenta *Kettle*. Essa transformação e seus passos podem ser vistos na Figura 13.

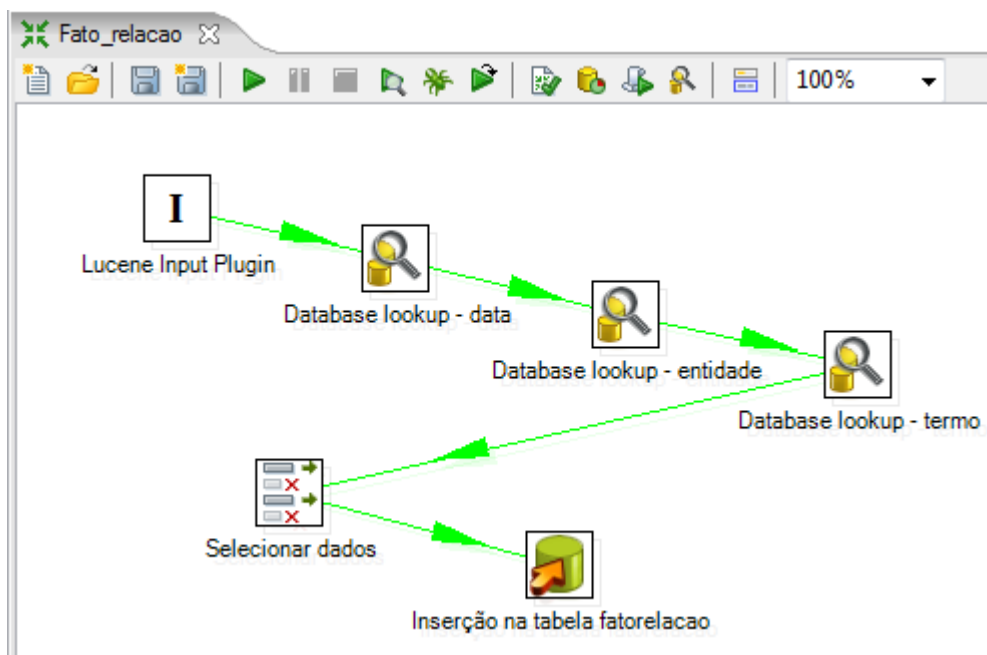


Figura 13 - Carga do fato relação

Lucene Input Plugin – Este passo consulta os índices de todos os departamentos pelos termos armazenados na base de termos selecionados. Para cada documento que contenha o termo procurado, são enviados para os outros passos da transformação uma linha mencionando o termo ao departamento e uma linha para cada usuário referente ao documento, relacionando-o ao termo buscado.

Database lookup – data – Neste passo, é buscado na dimensão *tempo* o valor da chave correspondente ao tempo do e-mail. Por exemplo, para o ano 2007, o mês dezembro corresponde à chave 453.

Database lookup – entidade – Neste passo, é buscado na dimensão *entidade* o valor da chave correspondente à entidade que se deseja relacionar a determinado termo, ou seja, o departamento ou algum dos usuários. Por exemplo, o Departamento Alpha possui o id 17220, e o usuário `enron.security@enron.com` possui o id 13270.

Database lookup – termo – Neste passo, é buscado na dimensão *entidade* o valor da chave correspondente ao termo buscado no e-mail. Por exemplo, o termo “bankrupt” tem o id 17230.

Selecionar dados – Neste passo, somente selecionamos as colunas que vão ser enviadas ao passo seguinte para poder inserir corretamente na tabela `fato_relação`.

Inserção na tabela FATO_RELACAO – É neste passo que de fato faz-se a inserção na tabela. São armazenados `id_entidade_origem`, `id_entidade_destino` e `id_tempo`.

4.3.5 Fato Frequência

Para a carga da tabela `fato_frequencia`, foram inseridas as linhas representando todas as relações entre termos e tempos. A coluna frequência tem o valor 0 para todas essas relações. O script SQL utilizado foi “INSERT INTO `fato_frequencia` (`id_entidade`, `id_tempo`, `frequencia`) VALUES (?, ?, ?)”, sendo os valores utilizados apresentados na Tabela 6.

Id_entidade	Id_tempo	Frequência
17225	1	0
17225	2	0
17302	1	0

Tabela 6 - Exemplo de valores utilizados na carga do fato frequência.

Após a execução do script SQL gerando a tabela em seu estado inicial, foi programada a classe “*UpdateFatoFrequencia.java*” na linguagem Java para executar a atualização do campo *frequência*. Essa classe consulta os índices dos e-mails pelos termos da base de termos relacionados, e para cada documento que contenha a palavra buscada, é procurado o id do tempo e do termo buscado nas tabelas `dimensao_tempo` e `dimensao_entidade`, respectivamente. Com as chaves do tempo e do termo buscado, é somado 1 ao campo *frequência* correspondente a essas chaves.

4.4. Discussão dos resultados

A partir da criação e da carga do modelo de dados, considera-se a integração de dados estruturados e não estruturados. Pode-se então iniciar o processo de análise das informações agora disponíveis no DW. As análises realizadas aqui promovem uma visão inicial da potencialidade do modelo no suporte a aplicações de BI que auxiliem na tomada de decisão.

4.4.1 Primeira análise

Nesta análise, foi realizada uma consulta para verificar a relação entre pessoas e departamentos a um determinado termo, no caso “*bankrupt*”. A partir do resultado, espera-se uma visão mais clara sobre as pessoas mais associadas a determinado termo.

De modo geral, essa informação pode ser útil em aplicações de BI, como, por exemplo, a análise de competências essenciais para a organização. Isso se deve ao fato de que, dada uma possível necessidade de competência (a necessidade é mapeada por meio de termos de interesse presentes no DW) visando suportar projetos e/ou a visão estratégica, pode-se ter um conhecimento da distribuição dessa competência na organização. Caso tal competência não esteja ou esteja parcialmente presente, tornam-se necessários investimentos ou parcerias que possam impulsionar determinado negócio. Também pode ser útil para identificar claramente quem são as pessoas mais envolvidas com determinado assunto, como no caso da empresa Enron, ou mesmo na possível identificação de pessoas-chave em um determinado foco de atuação.

Nesse sentido, foram selecionadas a partir da tabelas FATO_RELACAO as pessoas que mais estavam ligadas aos termos de interesse descritos acima. Como resultado, a análise demonstra a relação entre entidades (classe Pessoa) com os termos “*bankrupt*”. Essa relação representa o fato de uma pessoa ter recebido ou enviado um e-mail com o termo em seu conteúdo ou título. A Figura 14 apresenta quais pessoas provavelmente estavam mais envolvidas com a situação da Enron quanto ao pedido de concordata, que coincide com a maioria dos e-mails da base utilizada.

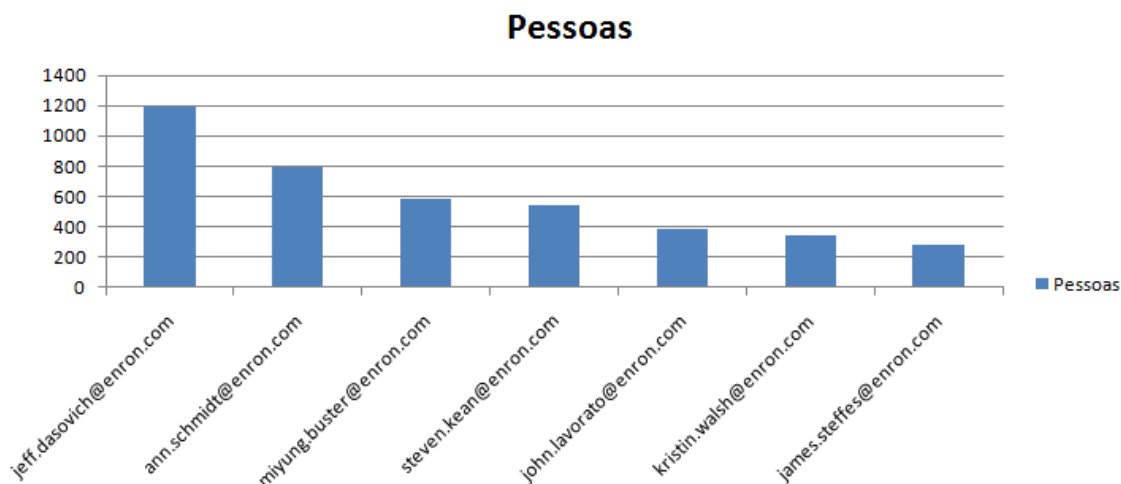


Figura 14 - Análise de relação entre o termo bankrupt e pessoas

A Figura 15, similar à análise anterior, apresenta quais departamentos (nesse caso fictícios) foram mais mencionados conjuntamente nos termos de análise. Nota-se claramente uma maior incidência para os departamentos Bravo e Charlie.

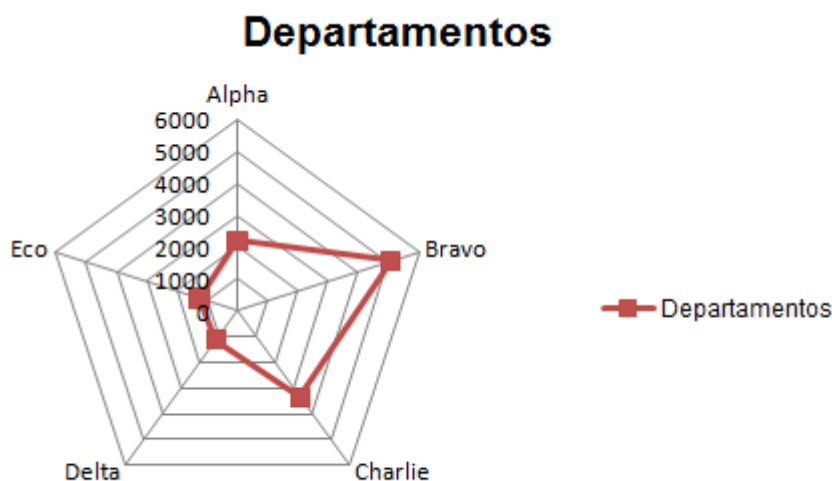


Figura 15 - Análise de relação entre o termo bankrupt e departamentos

Em uma base de dados estruturada, um assunto qualquer poderia estar relacionado a um departamento, indicando que esse seria o departamento responsável pelo assunto. Com essa modelagem, pode-se inferir que as pessoas que trabalham nesse departamento são as mais envolvidas com esse termo. Já com a análise realizada na Figura 15, verifica-se que, apesar de o departamento Bravo ser o mais relacionado ao termo “*bankrupt*”, o e-mail jeff.dasovich@enron.com, que

é a pessoa mais relacionada ao mesmo termo, de acordo com a Figura 14, não pertence a esse mesmo departamento.

Nesse sentido, conclui-se que uma pessoa que não pertença a um determinado departamento responsável por um assunto (aqui, representado pelo termo “*bankrupt*”) pode ser a pessoa que mais entende, ou que mais lida com esse assunto. Com essa técnica, poderiam ser identificadas dentro de uma organização as pessoas que estão mais envolvidas com questões específicas, servindo de subsídio para a tomada de decisão.

4.4.2 Segunda análise

Após a análise de relação entre termos e entidades, realizou-se uma avaliação da distribuição de frequências por mês para os termos “*bankrupt*”, “*payment*” e “*profit*”. Esse tipo de análise, assim como a anterior, visa prover subsídios à organização na busca pelo entendimento de suas fraquezas e pontos fortes no que se refere a competências e possíveis conhecimentos necessários à condução do negócio.

Cabe mencionar que o fato de determinado termo aparecer com maior ou menor intensidade na análise, ou mesmo nem aparecer, não indica necessariamente qualquer fraqueza em uma determinada área. Entretanto, isso cria subsídios para uma potencial decisão de investimento, podendo inclusive se propor a reavaliação dos sistemas de informação de modo que estes consigam capturar o que realmente ocorre no fluxo de informação, estruturada e não estruturada, da organização.

Para demonstrar a análise dos três termos mencionados acima, foram selecionadas as informações temporais e de frequência a partir das tabelas FATO_FREQUENCIA e a descrição dos termos a partir da tabela DIMENSAO_ENTIDADE. O resultado é apresentado na Figura 16.

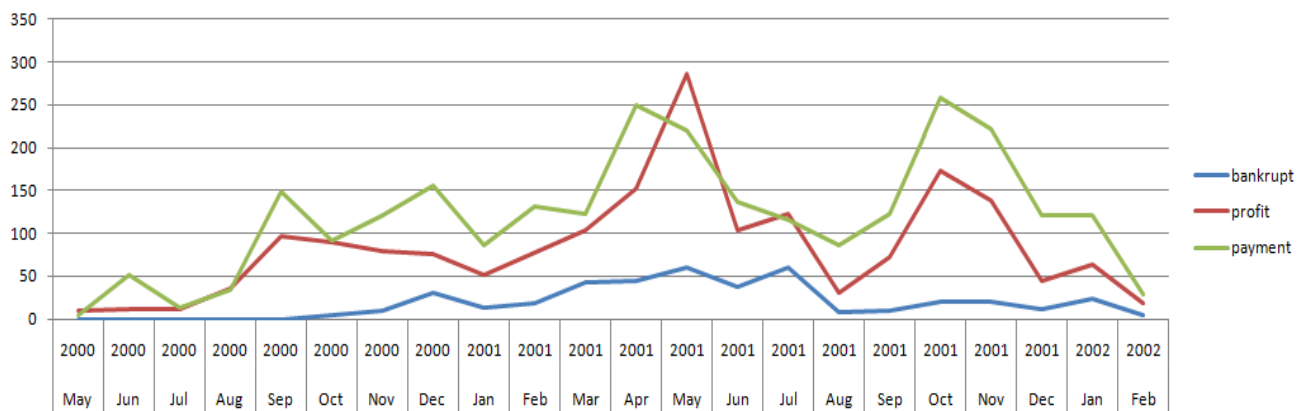


Figura 16 - Análise da frequência dos termos *bankrupt*, *profit* e *payment* distribuídos por tempo

O gráfico ilustrado na Figura 16 indica que os termos *profit* e *payment* possuem alguma correlação, pois tendem a assumir uma distribuição semelhante ao longo dos meses de análise. Considerando-se que foi em agosto de 2001 a primeira vez que a administração da Enron foi apontada como falha e possivelmente corrupta publicamente, pode-se ver um aumento do uso das palavras *payment* e *profit* entre os e-mails internos da empresa, talvez por algum receio de seus funcionários.

4.5. Considerações finais

Este capítulo apresentou o desenvolvimento do modelo proposto e possíveis análises a partir da integração de dados não estruturados e estruturados visando à avaliação do trabalho. Para tal, definiram-se inicialmente as fontes primárias e secundárias utilizadas. Em seguida, foram detalhados os componentes tecnológicos desenvolvidos e o modelo de dados que promoveram suporte ao processo de ETL 2.0. Por fim, foram realizadas análises básicas visando demonstrar o potencial da integração de informações estruturadas e não estruturadas e como estas podem criar novas dimensões quando se busca uma visão mais sistêmica que auxilie nos processos de tomada de decisão de uma organização.

5 CONCLUSÕES

Este trabalho demonstrou os conhecimentos necessários para a criação de um Data Warehouse e as principais técnicas utilizadas em suas várias etapas. Com uma ênfase no processo de ETL, foram identificados os desafios para o futuro desse mercado, principalmente no que diz respeito à integração entre os dados estruturados e não estruturados.

A fundamentação teórica revisou a literatura acadêmica com artigos que tratam de Data Warehouses, Business Intelligence, extração de informação, uso de extração de informação em conjunto com Data Warehouses, Text Mining e análise de textos. Foram identificadas as principais abordagens para a integração de dados estruturados e não estruturados e as atividades necessárias ao tratamento dos dados não estruturados para posterior análise.

Após a fundamentação teórica, foi proposta uma extensão ao processo de ETL para integrar dados não estruturados aos dados estruturados de forma genérica. Esse modelo foi descrito e aplicado a um modelo de dados relacional, em que seria possível a análise dos dados.

O processo proposto foi avaliado utilizando-se dados reais. Para a avaliação, foi necessária a extensão de uma ferramenta de mercado atual de código aberto. Foram desenvolvidas extensões para essa ferramenta em forma de *plugins*, e essas extensões mostraram a viabilidade de a ferramenta tratar dados não estruturados em um mesmo processo.

Com a utilização das extensões desenvolvidas, realizou-se o processo de ETL sobre os dados estruturados e não estruturados. Foi possível efetuar uma análise sobre esses dados, mostrando que o processo de integração dos dados é viável e útil à tomada de decisão. A utilização dos dados não estruturados trouxe mais informações para a tomada de decisão. Quanto mais informações forem utilizadas para analisar os processos de uma organização, mais fáceis e precisos serão os processos de tomada de decisão. Por esses motivos, considerou-se que os objetivos do trabalho foram alcançados.

5.1 Trabalhos futuros

Alguns estudos adicionais poderiam ser derivados a partir do que foi proposto neste trabalho. Esses estudos residem principalmente em três possíveis frentes: (1) extensão do modelo, (2) identificação de entidades e (3) análise dos dados.

Primeiramente, vislumbra-se a extensão do modelo de dados proposto tornando-o mais genérico e robusto para análises de correlação direta e indireta. Uma correlação indireta é mensurada pela frequência conjunta entre duas entidades, enquanto que a indireta analisa o perfil de duas entidades quaisquer. Além disso, seria adequado que o modelo propiciasse uma visão de tema ou domínio, permitindo assim que determinado estudo considerasse somente um conjunto restrito de entidades, e não todo o conteúdo da dimensão entidade.

Outro ponto importante é quanto à seleção dos termos. Para o trabalho, os termos retirados das fontes de informações não estruturadas foram escolhidos de forma arbitrária após uma avaliação manual da fonte de informação. Como melhoria a esse processo, seria adequada a utilização de uma etapa voltada à identificação de entidades para encontrar possíveis termos relevantes e suas classes.

Por fim, mas não de maneira exaustiva, vislumbra-se a utilização de técnicas mais elaboradas de mineração de textos, tais como correlações diretas e indiretas, de modo a explicitar conhecimento não trivial a partir do DW. As correlações diretas são obtidas com a análise da frequência conjunta, enquanto que a indireta é obtida por meio da avaliação do perfil da entidade. Projeções de redes, sugestão de conteúdos, análises de tendência e previsões são exemplos de análises mais elaboradas possíveis de serem obtidas se o modelo proposto for estendido.

REFERÊNCIAS

ANZANELLO, Cynthia Aurora. **OLAP: conceitos e utilização**. 2002.

BERNSTEIN, Philip A.; HAAS, Laura M. **Information integration in the Enterprise**. 2008.

BRAZHNIK, Olga; JONES, John. **Anatomy of data integration**. 2005.

BONTCHEVA, Kalina et al. **Evolving GATE to meet new challenges in language engineering**. 1998.

CHEN, Ming-Syan; HAN, Jiawei; YU, Philip S. **Data Mining: an overview from Database Perspective**. 1996

CODY, William et al. **The integration of business intelligence and knowledge management**. 2002.

COWIE, Jim; WILKS, Yorick. **Information Extraction**. 2005.

DAYAL, Umeshwar et al. **Data integration flows for business intelligence**. 2009.

DOAN, AnHai et al. **Information extraction challenges in managing unstructured Data**. 2008.

ELBASHIR, Mohamed Z.; COLLIER, Philip A.; DAVERN, Michael J. **Measuring the effects of business intelligence systems: the relationship between business process and organizational performance**. 2008.

GARSHOL, Lars Marius. **Metadata? Thesauri? Taxonomies? Topic Maps! - Making sense of it all**. 2004.

GRIMES, Seth. **Text technologies in the mainstream, text analytics solutions, applications, and trends**. 2008.

GUARINO, N. **Formal ontology in information systems: proceedings of the First International Conference**, 1998.

GOMES, Daniel; SILVA, Mario J. **The Viuva Negra Crawler**. 2006.

GONÇALVES, Alexandre L. **Um modelo de descoberta de conhecimento baseado na correlação de elementos textuais e expansão vetorial aplicado à engenharia e gestão do conhecimento**. 2006.

HOTHO, Andreas; NURNBERGER, Andreas; PAASS, Gerhard. **A brief survey of Text Mining**. 2005

IGARASHI, Wagner. **Construção automática de vocabulários e cálculo de aderência curricular**: uma aplicação aos fundos setoriais. 2005.

INMON, William; STRAUSS, Derek; NEUSHLOSS, Genia. **DW 2.0: the architecture for the next generation of Data Warehousing**, 2007

INMON, William. **Building the Data Warehouse**, 2002.

JHINGRAN, Anant. **Enterprise Information Mashups: Integrating Information, Simply**. 2006.

JING, Yufeng; CROFT, W. Bruce. **An association thesaurus for Information Retrieval**, 1994.

KEITH, Steven; KASER, Owen; LEMIRE, Daniel. **Analyzing Large Collections of Electronic Text Using OLAP**. 2005.

KIMBALL, Ralph. **The Data Warehouse ETL Toolkit**. 2004.

_____. **The Data Warehouse Lifecycle Toolkit**. 2008.

LOSEE, Robert. **Browsing mixed structured and unstructured data**. 2006.

MARTINEZ, Juan Manuel Pérez et al. **Contextualizing Data Warehouses with documents**, 2008.

MOSS, Larissa T.; ATRE, Shaku. **Business Intelligence Roadmap - the complete project lifecycle for decision-support applications**. 2003.

PEREIRA, Ricardo Oliveira; REZENDE, Denis Alcides; ABREU, Aline França. **Gestão do conhecimento com apoio dos recursos de sistemas de informação e tecnologias emergentes**. 2000.

PÉREZ, Juan Manuel; BERLANGA, Rafael; ARAMBURU, Maria José. **A relevance model for a data warehouse contextualized with documents**. 2009.

PINHEIRO, Carlos André Reis. **Inteligência analítica: mineração de dados e descoberta de conhecimento**. 2008.

RAVAT, Franck; OLIVIER, Teste; TOURNIER, Ronan. **OLAP aggregation función for textual Data Warehouse**. 2007.

SAVI, Antonio Francisco; AMARAL, Daniel Capaldo; ROZENFELD, Henrique. **Proposta de um conceito de um sistema de apoio à gestão do conhecimento empregando gestão de projetos e modelagem de empresas**. 2003.

SCHUTZE, Hinrich; PEDERSEN, Jan O. **A cocurrence-based thesaurus and two applications to information retrieval**. 1996.

TRUJILLO, Juan; SONG, Il-Yeol. **New trends in Data Warehousing and OLAP**, 2008.

USCHOLD, Mike; GRUNINGER, Michael. **ONTOLOGIES: principles, methods and applications**. 1996.

VASSILIADIS, Panos et al. **Aktos: towards the modeling, design, control and execution of ETL processes**. 2001.

WEI, Ching-Song D. et al. **Integration of structured and unstructured text data in a clinical information system**. 2006.

WILKS, Yorick; GAIZAUSKAS, Robert. **Information extraction: beyond Information Retrieval**. 1998.

WIVES, Leandro Krug. **Tecnologias de descoberta de conhecimento em textos aplicadas à inteligência competitiva**. 2002.

YAO, JingTao; RAGHAVAN, Vijay V.; WU, Zonghuan. **Web information fusion: a review of the state of the art**. 2008

ZHENG, Hai-Tao; KANG, Bo-Yeong; KIM, Hong-Gee. **An ontology-based approach to learnable focused crawling**. 2008.

INSTITUTO STELA. **Retriever**. Disponível em:
<<http://dev.stela.org.br/retriever/index.html>>. Acesso em: 11 jun. 2009.

PENTAHO, PENTAHO CORPORATION, Kettle. Disponível em:
<<http://kettle.pentaho.org>>. Acesso em: 11 jun. 2009.

Enron, Enron Email Dataset. Disponível em:
<<http://www.cs.cmu.edu/~enron>>. Acesso em: 3 out. 2009

Apêndice 1 - Base de termos

<i>admission</i>	<i>alternative</i>	<i>amateur</i>
<i>anthrax</i>	<i>arabia</i>	<i>bankrupt</i>
<i>bankruptcy</i>	<i>biotechnology</i>	<i>blackout</i>
<i>brazil</i>	<i>consumer</i>	<i>cost</i>
<i>costly</i>	<i>data</i>	<i>database</i>
<i>decision</i>	<i>demand</i>	<i>design</i>
<i>doctor</i>	<i>documentation</i>	<i>east</i>
<i>employee</i>	<i>energy</i>	<i>enron</i>
<i>enterprise</i>	<i>environment</i>	<i>europe</i>
<i>european</i>	<i>facility</i>	<i>facing</i>
<i>feedback</i>	<i>fire</i>	<i>firewall</i>
<i>fuel</i>	<i>google</i>	<i>hardware</i>
<i>human</i>	<i>information</i>	<i>internet</i>
<i>intranet</i>	<i>issues</i>	<i>jobs</i>
<i>office</i>	<i>oil</i>	<i>organization</i>
<i>logistics</i>	<i>management</i>	<i>marketing</i>
<i>microsoft</i>	<i>middle</i>	<i>network</i>
<i>networking</i>	<i>payment</i>	<i>planning</i>
<i>president</i>	<i>product</i>	<i>production</i>
<i>profit</i>	<i>project</i>	<i>quality</i>
<i>quantity</i>	<i>resource</i>	<i>restore</i>
<i>resume</i>	<i>saudi</i>	<i>screwed</i>
<i>software</i>	<i>solar</i>	<i>south</i>
<i>stock</i>	<i>supply</i>	<i>system</i>
<i>tactical</i>	<i>technology</i>	<i>urgent</i>
<i>warehouse</i>	<i>waste</i>	<i>wind</i>
<i>yahoo</i>		

Apêndice 2 - Código-fonte do CollectorPlugin

NullFilter.java

```

package info.stela.ekp.iscollector.filter;

import net.sourceforge.retriever.fetcher.Resource;
import net.sourceforge.retriever.filter.Filter;

/**
 * This class implements the Null Object pattern concerning the <code>Filter</code> interface.
 */
public class NullFilter implements Filter {

    /**
     * Always returns true.
     * @param url
     * @return true
     */
    public boolean acceptBeforeFetch(final String url) {
        return true;
    }

    /**
     * Always returns true.
     *
     * @param resource
     * @return true
     */
    public boolean acceptAfterFetch(final Resource resource) {
        return true;
    }
}

```

URLLiteralFilter.java

```

package info.stela.ekp.iscollector.filter;

import info.stela.ekp.iscollector.ISCollectorGUIAnnotation;

import java.util.HashMap;
import java.util.Map;

import net.sourceforge.retriever.fetcher.Resource;
import net.sourceforge.retriever.filter.Filter;

/**
 * <p>
 * Class used to filter URLs that literally match some string.
 * </p>

```

```

*/
public class URLLiteralFilter implements Filter {

    private final Map<String, Boolean> patterns = new HashMap<String, Boolean>();

    /**
     * <p>
     * Adds a pattern, which is a string that will be used in the filtering process.
     * </p>
     *
     * <p>
     * If the URL of a resource literally matches the pattern, then it will be filtered.
     * </p>
     *
     * @param pattern The pattern to be used in the filtering process.
     */
    public void addPattern(final String pattern) {
        this.addPattern(pattern, true);
    }

    /**
     * <p>
     * Adds a pattern, which is a string that will be used in the filtering process.
     * </p>
     *
     * <p>
     * Depending on the value of the <code>ifMatchAccept</code> parameter, the filter class
     * will act differently. If the parameter's value is true, than URLs matching the pattern
     * will be rejected. URLs will otherwise successfully pass the filter if the parameter's
     * value is false and they match the pattern.
     * </p>
     *
     * @param pattern The pattern to be used in the filtering process.
     */
    @ISCollectorGUIAnnotation(name = "add pattern", unique = false, parameters = {"pattern", "ifMatchAccept"})
    public void addPattern(final String pattern, final boolean ifMatchAccept) {
        this.patterns.put(pattern, ifMatchAccept);
    }

    /**
     * Always return true because the resource has already gone through the
     * <code>acceptBeforeFetch(String)</code> method.
     */
    public boolean acceptAfterFetch(final Resource resource) {
        return this.acceptBeforeFetch(resource.getURL().toExternalForm());
    }

    /**
     * @see net.sourceforge.retriever.filter.Filter#acceptBeforeFetch(String)
     */
}

```

```

public boolean acceptBeforeFetch(final String url) {
    if (this.patterns.size() <= 0) return true;

    if (this.patterns.keySet().contains(url)) {
        return this.patterns.get(url);
    } else {
        return false;
    }
}
}

```

URLRegexFilter.java

```

package info.stela.ekp.iscollector.filter;

import info.stela.ekp.iscollector.ISCollectorGUIAnnotation;

import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import net.sourceforge.retriever.fetcher.Resource;
import net.sourceforge.retriever.filter.Filter;

/**
 * Class used to filter URLs using regular expressions.
 */
public class URLRegexFilter implements Filter {

    private Map<String, Boolean> regexes = new HashMap<String, Boolean>();

    /**
     * Adds a regular expression to be used in the filtering process.
     *
     * The URL that pass by the filter must match one of the added regular
     * expressions to be accepted.
     *
     * @param regex Regular expression to be used during the filtering process.
     */
    public void addRegex(final String regex) {
        this.addRegex(regex, true);
    }

    /**
     * Adds a regular expression to be used in the filtering process.
     *
     * If the boolean argument is true, then URLs matching the regex will be
     * accepted, otherwise the URLs will be discarded.
     *
     * @param regex Regular expression to be used in the filtering process.
     */

```



```

    * @param ifMatchAccept decide whether the URL that matches the regex
    *         must be discard or not.
    */
    @ISCollectorGUIAnnotation(name = "add regex", unique = false, parameters = {"regex pattern" , "ifMatchAccept"})
    public void addRegex(final String regex, final boolean ifMatchAccept) {
        this.regexes.put(regex, ifMatchAccept);
    }

    /**
     * Filter a resource before the fetching process, when it's only a URL.
     *
     * @param url The URL used in the filter process.
     * @return True if the URL was accepted and won't be filtered and false otherwise.
     */
    public boolean acceptBeforeFetch(final String url) {
        if (this.regexes.isEmpty()) return true;
        final Set<String> regexes = this.regexes.keySet();
        boolean result = false;
        for (String regex : regexes) {
            if (url.matches(regex)) {
                if (this.regexes.get(regex)) {
                    result = true;
                } else {
                    return false;
                }
            }
        }
        return result;
    }

    /**
     * Always return true because the resource has already gone through the
     * <code>acceptBeforeFetch(String)</code> method.
     */
    public boolean acceptAfterFetch(final Resource resource) {
        return this.acceptBeforeFetch(resource.getURL().toExternalForm());
    }
}

```

CollectorPlugin.java

```

package info.stela.ekp.iscollector;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.lang.reflect.InvocationTargetException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;

```

```

import net.sourceforge.retriever.Retriever;
import net.sourceforge.retriever.analyzer.Analyzer;
import net.sourceforge.retriever.fetcher.Resource;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.exception.KettleException;
import org.pentaho.di.core.exception.KettleStepException;
import org.pentaho.di.core.row.RowMeta;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStep;
import org.pentaho.di.trans.step.StepDataInterface;
import org.pentaho.di.trans.step.StepInterface;
import org.pentaho.di.trans.step.StepMeta;
import org.pentaho.di.trans.step.StepMetaInterface;

public class CollectorPlugin extends BaseStep implements StepInterface
{
    private CollectorPluginData data;
    private CollectorPluginMeta meta;

    public CollectorPlugin(StepMeta s, StepDataInterface stepDataInterface, int c, TransMeta t, Trans dis) {
        super(s, stepDataInterface, c, t, dis);
    }

    public boolean processRow(StepMetaInterface smi, StepDataInterface sdi) throws KettleException {
        meta = (CollectorPluginMeta)smi;
        data = (CollectorPluginData)sdi;

        if (first){
            first = false;

            data.outputRowMeta = new RowMeta();
            meta.getFields(data.outputRowMeta, getStepname(), null, null, this);

            this.runCrawler();
            setOutputDone();
            return false;
        }

        return true;
    }

    public boolean init(StepMetaInterface smi, StepDataInterface sdi) {
        meta = (CollectorPluginMeta)smi;
        data = (CollectorPluginData)sdi;

        return super.init(smi, sdi);
    }

    public void dispose(StepMetaInterface smi, StepDataInterface sdi) {
        meta = (CollectorPluginMeta)smi;
    }
}

```

```

        data = (CollectorPluginData)sdi;

        super.dispose(smi, sdi);
    }

    // Run is were the action happens!
    public void run() {
        logBasic("Starting to run...");

        try {
            while (processRow(meta, data) && !isStopped())
                ;
        } catch (Exception e) {
            logError("Unexpected error : "+e.toString());
            logError(Const.getStackTracker(e));
            setErrors(1);
            stopAll();
        } finally {
            dispose(meta, data);
            logBasic("Finished, processing "+getLinesWritten()+" rows");
            markStop();
        }
    }

    private void runCrawler() {
        final Retriever crawler = new Retriever();

        // Add seeds to crawler
        ArrayList<String> seeds = meta.getSeeds();
        for (String seed : seeds) {
            try {
                crawler.addSeed(new URL(seed));
            } catch (final MalformedURLException e) {
                e.printStackTrace();
            }
        }

        //set crawler filters and call filter methods
        crawler.setFilter(meta.getFilter());
        for (FilterMethod filterMethod : meta.getFilterMethods()) {
            try {
                filterMethod.getMethod().invoke(meta.getFilter(), filterMethod.getParameters());
                System.out.println("called method - "+filterMethod.getMethod());
                System.out.println("with parameters :");
                for (Object param : filterMethod.getParameters()) {
                    System.out.println(param);
                }
            } catch (IllegalArgumentException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
    }

```

```

        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
    }

    // At each crawled resource, its URL is printed
    crawler.setAnalyzer(new Analyzer() {
        private int i = 0;
        public Object analyze(final Resource resource) {
            if(crawler.isRunning()){
                try {
                    //
                    inputStreamToBytes(resource.getInputStream());
                    inputStreamToString(resource.getInputStream());
                    resource.getCharset(),
                    byte[]    byteArray    =    isDirectory(resource)?    null:
                    String    byteArray    =    isDirectory(resource)?    null:
                    Object[]    row    =    {resource.getURL().toExternalForm(),
                                            resource.getContentType(), byteArray};
                    putRow(data.outputRowMeta, row);
                    logBasic(++i + " " + resource.getURL().toExternalForm());
                } catch (KettleStepException e) {
                    e.printStackTrace();
                }
                if(isStopped()){
                    crawler.stop();
                }
            }
            return null;
        }
    });

    private boolean isDirectory(final Resource resource) {
        return new File(resource.getURL().getPath()).isDirectory();
    }

    private byte[] inputStreamToBytes(InputStream in) {
        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream(1024);
            byte[] buffer = new byte[1024];
            int len;

            while((len = in.read(buffer)) >= 0)
                out.write(buffer, 0, len);

            in.close();
            out.close();
            return out.toByteArray();
        } catch (final IOException e) {
            e.printStackTrace();
            return new byte[0];
        }
    }
}

```

```

        private String inputStreamToString(InputStream in) {
            try {

                OutputStream out = new ByteArrayOutputStream(1024);
                byte[] buffer = new byte[1024];
                int len;

                while((len = in.read(buffer)) >= 0)
                    out.write(buffer, 0, len);

                in.close();
                out.close();
                return out.toString();
            } catch (final IOException e) {
                e.printStackTrace();
                return "";
            }
        }
    });

    crawler.start();
}

```

CollectorPluginData.java

```

package info.stela.ekp.iscollector;

import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.trans.step.BaseStepData;
import org.pentaho.di.trans.step.StepDataInterface;

public class CollectorPluginData extends BaseStepData implements StepDataInterface
{
    public RowMetaInterface outputRowMeta;

    public CollectorPluginData()
    {
        super();
    }
}

```

CollectorPluginDialog.java

```

package info.stela.ekp.iscollector;

import java.lang.reflect.Method;
import java.util.ArrayList;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.ScrolledComposite;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;

```

```

import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.ShellAdapter;
import org.eclipse.swt.events.ShellEvent;
import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layout.FormData;
import org.eclipse.swt.layout.FormLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.RowMetaAndData;
import org.pentaho.di.core.exception.KettleValueException;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.ui.core.widget.ColumnInfo;
import org.pentaho.di.ui.core.widget.TableView;
import org.pentaho.di.ui.trans.step.BaseStepDialog;
import org.pentaho.di.trans.step.BaseStepMeta;
import org.pentaho.di.trans.step.StepDialogInterface;

public class CollectorPluginDialog extends BaseStepDialog implements StepDialogInterface {
    public static final int MARGIN = Const.MARGIN;
    public static final String[] TRUE_FALSE_COMBO = {"true", "false"};
    public int middle;
    public Control attach;

    private CollectorPluginMeta input; //input
    private ArrayList<String> seeds; //seeds from input
    private ArrayList<FilterMethod> filterMethods; //methods from input

    private ScrolledComposite wScrolledComposite; //holds composite
    private Composite wComposite; //holds all UI elements
    private ModifyListener lsMod; //Modify Listener used to tell kettle that something has changed in the dialog
    private ArrayList<FilterDialogMethod> filterDialogMethods = new ArrayList<FilterDialogMethod>();

    // seeds
    private TableView wSeedsTable;
    private FormData fdSeedsTable;
    private Label wSeed;
    private FormData fdSeed;

    // filter combo box
    private Label wFilter;
    private Combo wFilter;

```

```

private FormData fdlFilter, fdFilter;

public CollectorPluginDialog(Shell parent, Object in, TransMeta transMeta, String sname) {
    super(parent, (BaseStepMeta)in, transMeta, sname);
    input=(CollectorPluginMeta)in;
    this.seeds = input.getSeeds();
}

public String open() {
    Shell parent = getParent();
    Display display = parent.getDisplay();

    shell = new Shell(parent, SWT.DIALOG_TRIM | SWT.RESIZE | SWT.MIN | SWT.MAX);
    props.setLook(shell);
setShellImage(shell, input);
    shell.setLayout(new FormLayout());
    shell.setText(Messages.getString("CollectorPluginDialog.Shell.Title"));

wScrolledComposite = new ScrolledComposite(shell, SWT.BORDER | SWT.V_SCROLL);
    FormData fdComposite = new FormData();
    fdComposite.top = new FormAttachment(shell, 3);
    fdComposite.left = new FormAttachment(0, 3);
    fdComposite.right = new FormAttachment(100, -3);
    fdComposite.bottom = new FormAttachment(100, -3);
    wScrolledComposite.setLayoutData(fdComposite);
    wComposite = new Composite(wScrolledComposite, SWT.NONE);
    props.setLook(wComposite);
    wComposite.setLayout(new FormLayout());

    middle = props.getMiddlePct();

    lsMod = new ModifyListener() {
        public void modifyText(ModifyEvent e) {
            input.setChanged();
        }
    };
    changed = input.hasChanged();

    // Stepname line
    wStepname=new Label(wComposite, SWT.RIGHT);
    wStepname.setText(Messages.getString("CollectorPluginDialog.Stepname.Label"));
props.setLook( wStepname );
    fdlStepname=new FormData();
    fdlStepname.left = new FormAttachment(0, 0);
    fdlStepname.right= new FormAttachment(middle, -MARGIN);
    fdlStepname.top = new FormAttachment(0, MARGIN);
    wStepname.setLayoutData(fdlStepname);
    wStepname=new Text(wComposite, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
    wStepname.setText(stepname);
props.setLook( wStepname );
    fdStepname=new FormData();

```

```

fdStepname.left = new FormAttachment(middle, MARGIN);
fdStepname.top = new FormAttachment(0, MARGIN);
fdStepname.right= new FormAttachment(100, -MARGIN);
wStepname.setLayoutData(fdStepname);
this.attach = wStepname;

this.drawSeeds();
this.drawFilter();

// Final buttons
wOK=new Button(wComposite, SWT.PUSH);
wOK.setText(Messages.getString("System.Button.OK"));
wCancel=new Button(wComposite, SWT.PUSH);
wCancel.setText(Messages.getString("System.Button.Cancel"));

BaseStepDialog.positionBottomButtons(wComposite, new Button[] { wOK, wCancel}, MARGIN, attach);

this.addListeners();

// Set the shell size, based upon previous time...
setSize();
wScrolledComposite.setMinSize(wComposite.computeSize(SWT.DEFAULT, SWT.DEFAULT));
wScrolledComposite.setExpandHorizontal(true);
wScrolledComposite.setExpandVertical(true);
wScrolledComposite.setContent(wComposite);
wScrolledComposite.getVerticalBar().setIncrement(10);/"faster" scrollbar

getData();
input.setChanged(changed);

shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch()) display.sleep();
}
return stepname;
}

private void drawSeeds() {
    // Seeds Label
    wSeed = new Label(wComposite, SWT.RIGHT);
    wSeed.setText(Messages.getString("CollectorPluginDialog.Seed.Label"));
    props.setLook(wSeed);
    fdlSeed = new FormData();
    fdlSeed.left = new FormAttachment(0, 0);
    fdlSeed.right = new FormAttachment(middle, -MARGIN);
    fdlSeed.top = new FormAttachment(attach, MARGIN);
    wSeed.setLayoutData(fdlSeed);
    // Seeds TableView
    ColumnInfo[] colinf=new ColumnInfo[] {
        new ColumnInfo("seeds", ColumnInfo.COLUMN_TYPE_TEXT, false) };
    wSeedsTable=new TableView(transMeta, wComposite,

```



```

        SWT.FULL_SELECTION | SWT.SINGLE | SWT.BORDER, colinf, 3, lsMod, props);
fdSeedsTable=new FormData();
fdSeedsTable.left = new FormAttachment(middle, MARGIN);
fdSeedsTable.top = new FormAttachment(attach, MARGIN);
fdSeedsTable.right = new FormAttachment(100, -MARGIN);
wSeedsTable.setLayoutData(fdSeedsTable);
this.attach = wSeedsTable;
}

private void addListeners() {
    ModifyListener lsFilterMod = new ModifyListener() {
        public void modifyText(ModifyEvent e) {
            drawFilterParameters();
        }
    };

    lsCancel = new Listener() { public void handleEvent(Event e) { cancel(); } };
    lsOK = new Listener() { public void handleEvent(Event e) { ok(); } };

    wCancel.addListener(SWT.Selection, lsCancel);
    wOK.addListener(SWT.Selection, lsOK );

    lsDef=new SelectionAdapter() { public void widgetDefaultSelected(SelectionEvent e) { ok(); } };

    wStepname.addSelectionListener( lsDef );

    wStepname.addModifyListener(lsMod);
    wFilter.addModifyListener(lsMod);

    wFilter.addModifyListener(lsFilterMod);

    // Detect X or ALT-F4 or something that kills this window...
    shell.addShellListener( new ShellAdapter() { public void shellClosed(ShellEvent e) { cancel(); } });
}

private void drawFilter() {
    // Filter label
    wFilter = new Label(wComposite, SWT.RIGHT);
    wFilter.setText(Messages.getString("CollectorPluginDialog.Filter.Label"));
    props.setLook(wFilter);
    fdlFilter = new FormData();
    fdlFilter.left = new FormAttachment(0, 0);
    fdlFilter.right = new FormAttachment(middle, -MARGIN);
    fdlFilter.top = new FormAttachment(this.attach, MARGIN);
    wFilter.setLayoutData(fdlFilter);
    // Filter Classes Combo box
    wFilter = new Combo(wComposite, SWT.DROP_DOWN | SWT.READ_ONLY);
    props.setLook(wFilter);
    fdFilter = new FormData();
    fdFilter.left = new FormAttachment(middle, MARGIN);
    fdFilter.right = new FormAttachment(100, -MARGIN);
}

```

```

        fdFilter.top = new FormAttachment(this.attach, MARGIN);
        wFilter.setLayoutData(fdFilter);
        ArrayList<String> filters = input.getFilterNames();
        for(String filter : filters){
            wFilter.add(filter);
        }
        this.attach = wFilter;
    }

    private void drawFilterParameters() {
        // Filter Parameters
        this.removeFilterParameters();
        Class<?> filterClass = (Class<?>) input.getFilter(wFilter.getText());
        for(Method method : filterClass.getMethods()){
            if(method.isAnnotationPresent(ISCollectorGUIAnnotation.class)){
                ISCollectorGUIAnnotation anno =
method.getAnnotation(ISCollectorGUIAnnotation.class);
                filterDialogMethods.add(new FilterDialogMethod(this, props, wComposite, anno,
method, transMeta, lsMod));
                this.redrawButtons();
            }
        }
    }

    private void removeFilterParameters() {
        this.attach = wFilter;
        for(FilterDialogMethod filter : filterDialogMethods){
            filter.dispose();
        }
        filterDialogMethods = new ArrayList<FilterDialogMethod>();
        redrawButtons();
    }

    private void redrawButtons() {
        BaseStepDialog.positionBottomButtons(wComposite, new Button[] { wOK, wCancel}, MARGIN, this.attach);
        wComposite.pack(true);
    }

    // Read data from input (CollectorPluginMeta)
    private void getData() {
        wStepname.selectAll();
        wSeedsTable.removeAll();
        for(int i=0; i<seeds.size(); i++){
            wSeedsTable.add(new String[] {seeds.get(i)});
        }
        wSeedsTable.removeEmptyRows();
        wSeedsTable.setRowNums();
        wSeedsTable.optWidth(true);

        wFilter.setText(input.getStringFilter());
        this.filterMethods = input.getFilterMethods();
    }

```

```

for(FilterDialogMethod filterDialogMethod : filterDialogMethods){
    try {
        if(!filterDialogMethod.isUnique()){
            TableView table = ((TableView)filterDialogMethod.getWcFilterControls())[0];
            for(FilterMethod filterMethod : this.filterMethods){
                if(filterMethod.getMethod().equals(filterDialogMethod.getMethod())){
                    Object[] parameterValues = filterMethod.getParameters();
                    String[] parameters = new
String[parameterValues.length];

                    for(int i=0; i<parameters.length; i++){
                        parameters[i] = parameterValues[i].toString();
                    }
                    table.add(parameters);
                }
            }
            table.removeEmptyRows();
            table.setRowNums();
            table.optWidth(true);
        }else if(filterDialogMethod.isUnique()){
            Control[] controls = filterDialogMethod.getWcFilterControls();
            for(FilterMethod filterMethod : this.filterMethods){
                if(filterMethod.getMethod().equals(filterDialogMethod.getMethod())){
                    Object[] parameterValues = filterMethod.getParameters();
                    Class<?>[] parameterTypes =
filterMethod.getMethod().getParameterTypes();

                    for(int i=0; i<controls.length; i++){
                        if(parameterTypes[i].equals(boolean.class)){
                            ((Combo)controls[i]).setText(((Boolean)parameterValues[i]).toString());
                        }else if(parameterTypes[i].equals(String.class)){
                            ((Text)controls[i]).setText((String)parameterValues[i]);
                        }
                    }
                }
            }
        }
    } catch (SecurityException e) {
        e.printStackTrace();
    }
}

private void cancel() {
    stepname=null;
    input.setChanged(changed);
    dispose();
}

private void ok() {
    this.seeds = new ArrayList<String>();
}

```

```

for(String seed : wSeedsTable.getItems()){
    seeds.add(seed);
}
stepname = wStepname.getText(); // return value
input.setSeeds(this.seeds);
input.setFilter(wFilter.getText());

this.filterMethods = new ArrayList<FilterMethod>();
for(FilterDialogMethod filterDialogMethod : filterDialogMethods){
    try {
        if(!filterDialogMethod.isUnique()){
            TableView table = ((TableView)filterDialogMethod.getWcFilterControls())[0];
            table.removeEmptyRows();
            for(int i=0; i<table.nrNonEmpty(); i++){
                RowMetaAndData row = table.getRow(i);
                row.removeValue(0);//remove first colum - id.
                Object[] parameterValues = new Object[row.size()];
                Class<?>[] methodTypes =
filterDialogMethod.getMethod().getParameterTypes();
                for(int j=0; j<methodTypes.length; j++){
                    if(methodTypes[j].equals(boolean.class)){
                        parameterValues[j] =
Boolean.parseBoolean(row.getString(j, ""));
                    }else if(methodTypes[j].equals(String.class)){
                        parameterValues[j] = row.getString(j, "");
                    }
                }
                filterMethods.add(new FilterMethod(filterDialogMethod.getMethod(),
parameterValues));
            }
        } else if(filterDialogMethod.isUnique()){
            Control[] controls = filterDialogMethod.getWcFilterControls();
            Class<?>[] methodTypes =
filterDialogMethod.getMethod().getParameterTypes();
            Object[] parameterValues = new Object[controls.length];
            for(int i=0; i<controls.length; i++){
                if(methodTypes[i].equals(boolean.class)){
                    parameterValues[i] =
Boolean.parseBoolean(((Combo)controls[i]).getText());
                }else if(methodTypes[i].equals(String.class)){
                    parameterValues[i] = ((Text)controls[i]).getText();
                }
            }
            filterMethods.add(new FilterMethod(filterDialogMethod.getMethod(),
parameterValues));
        }
    } catch (SecurityException e) {
        e.printStackTrace();
    } catch (KettleValueException e) {
        e.printStackTrace();
    }
}

```

```

        }
    }
    input.setFilterMethods(filterMethods);
    dispose();
}
}

```

CollectorPluginMeta.java

```

package info.stela.ekp.iscollector;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PropertyResourceBundle;

import net.sourceforge.retriever.filter.Filter;
import net.sourceforge.retriever.filter.NullFilter;

import org.eclipse.swt.widgets.Shell;
import org.pentaho.di.core.CheckResult;
import org.pentaho.di.core.CheckResultInterface;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.Counter;
import org.pentaho.di.core.database.DatabaseMeta;
import org.pentaho.di.core.exception.KettleDatabaseException;
import org.pentaho.di.core.exception.KettleException;
import org.pentaho.di.core.exception.KettleXMLException;
import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.core.row.ValueMeta;
import org.pentaho.di.core.row.ValueMetaInterface;
import org.pentaho.di.core.variables.VariableSpace;
import org.pentaho.di.core.xml.XMLHandler;
import org.pentaho.di.repository.Repository;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStepMeta;
import org.pentaho.di.trans.step.StepDataInterface;
import org.pentaho.di.trans.step.StepDialogInterface;
import org.pentaho.di.trans.step.StepInterface;
import org.pentaho.di.trans.step.StepMeta;
import org.pentaho.di.trans.step.StepMetaInterface;
import org.w3c.dom.Node;

public class CollectorPluginMeta extends BaseStepMeta implements StepMetaInterface {

```

```

private static Map<String, Class<?>> filters;
static {
    filters = new HashMap<String, Class<?>>();
    FileInputStream is;
    try {
        is = new FileInputStream("plugins/steps/Collector Plugin/plugin.properties");
        PropertyResourceBundle pluginP = new PropertyResourceBundle(is);
        Enumeration<String> keys = pluginP.getKeys();
        while (keys.hasMoreElements()) {
            final String key = keys.nextElement();
            String keyData = pluginP.getString(key);
            filters.put(key, Class.forName(keyData));
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private Filter filter;
private ArrayList<FilterMethod> filterMethods = new ArrayList<FilterMethod>();
private ArrayList<String> seeds;

public CollectorPluginMeta() {
    super(); // allocate BaseStepInfo
}

public ArrayList<String> getSeeds() {
    return this.seeds;
}

public void setSeeds(ArrayList<String> seeds) {
    this.seeds = seeds;
}

public Filter getFilter() {
    return this.filter;
}

public Class<?> getFilter(String filter){
    return filters.get(filter);
}

//returns the filter's name as it was declared in plugin.properties
public String getStringFilter(){
    Object[] keys = filters.keySet().toArray();
    for(int i=0; i<keys.length; i++){
        if(this.filter!=null){

```

```

        if(filters.get(keys[i]).equals(this.filter.getClass())){
            return keys[i].toString();
        }
    }
    }
    return "";
}

public void setFilter(Filter filter) {
    this.filter = filter;
}

public void setFilter(String filter){
    if (filters.containsKey(filter)) {
        try {
            this.filter = (Filter)filters.get(filter).newInstance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}

public ArrayList<String> getFilterNames() {
    ArrayList<String> ret = new ArrayList<String>();
    for(String name : filters.keySet()){
        ret.add(name);
    }
    return ret;
}

public void setFilterMethods(ArrayList<FilterMethod> filterMethods){
    this.filterMethods = filterMethods;
}

public ArrayList<FilterMethod> getFilterMethods(){
    return this.filterMethods;
}

public String getXML()
{
    StringBuilder retval = new StringBuilder();

    retval.append("<seeds>").append(Const.CR);
    for(String seed : this.seeds){
        retval.append("<seed>").append(Const.CR);
        retval.append(XMLHandler.addTagValue("seed_name", seed));
        retval.append("</seed>").append(Const.CR);
    }
    retval.append("</seeds>").append(Const.CR);
}

```

```

        retval.append(XMLHandler.addTagValue("filter", this.getStringFilter()));
        retval.append("<filter_methods>").append(Const.CR);
        for(FilterMethod filterMethod : filterMethods){
            retval.append(filterMethod.getXML());
        }
        retval.append("</filter_methods>").append(Const.CR);

        System.out.println(retval.toString());

        return retval.toString();
    }

    public void getFields(RowMetaInterface r, String origin, RowMetaInterface[] info, StepMeta nextStep, VariableSpace
space) {
        ValueMetaInterface vURL = new ValueMeta("URL", ValueMetaInterface.TYPE_STRING);
        vURL.setOrigin(origin);
        r.addValueMeta(vURL);
        ValueMetaInterface vCharset = new ValueMeta("Charset", ValueMetaInterface.TYPE_STRING);
        vCharset.setOrigin(origin);
        r.addValueMeta(vCharset);
        ValueMetaInterface vContentType = new ValueMeta("ContentType", ValueMetaInterface.TYPE_STRING);
        vContentType.setOrigin(origin);
        r.addValueMeta(vContentType);
//        ValueMetaInterface vInputStream = new ValueMeta("InputStream", ValueMetaInterface.TYPE_BINARY);
        ValueMetaInterface vInputStream = new ValueMeta("InputStream", ValueMetaInterface.TYPE_STRING);
        vInputStream.setOrigin(origin);
        r.addValueMeta(vInputStream);
    }

    public Object clone() {
        Object retval = super.clone();
        return retval;
    }

    public void loadXML(Node stepnode, List<DatabaseMeta> databases, Map<String,Counter> counters)
throws KettleXMLException
    {
        try {
            Node seedsNode = XMLHandler.getSubNode(stepnode, "seeds");
            int nrSeeds = XMLHandler.countNodes(seedsNode, "seed");
            this.seeds = new ArrayList<String>();
            for (int i = 0; i < nrSeeds; i++) {
                Node fnode = XMLHandler.getSubNodeByNr(seedsNode, "seed", i);
                this.seeds.add(XMLHandler.getTagValue(fnode, "seed_name"));
                System.out.println("loadXML get seed - "+XMLHandler.getTagValue(fnode,
"seed_name"));
            }

            this.setFilter(XMLHandler.getTagValue(stepnode, "filter"));
            this.filterMethods = new ArrayList<FilterMethod>();

```



```

Node filterMethodsNode = XMLHandler.getSubNode(stepnode, "filter_methods");
int nrMethods = XMLHandler.countNodes(filterMethodsNode, "method");
System.out.println("loadXML get nrMethods = "+nrMethods);
for (int i = 0; i < nrMethods; i++) {
    Node methodNode = XMLHandler.getSubNodeByNr(filterMethodsNode, "method", i);
    String methodName = XMLHandler.getTagValue(methodNode, "method_name");
    Node parametersNode = XMLHandler.getSubNode(methodNode, "parameters");
    int nrParameters = XMLHandler.countNodes(parametersNode, "parameter");
    Object[] parameters = new Object[nrParameters];
    for(int j=0; j<nrParameters; j++){
        Node parameterNode = XMLHandler.getSubNodeByNr(parametersNode,
"parameter", j);
        String parameterType = XMLHandler.getTagValue(parameterNode,
"parameter_type");
        String parameterValue = XMLHandler.getTagValue(parameterNode,
"parameter_value");
        System.out.println("loadXML get parameterValue = "+parameterValue);
        if(parameterType.equals(Boolean.class.toString())){
            parameters[j] = Boolean.parseBoolean(parameterValue);
        }else if(parameterType.equals(String.class.toString())){
            parameters[j] = parameterValue;
        }
    }
    for(Method method : this.filter.getClass().getMethods()){
        if(method.toString().equals(methodName)){
            filterMethods.add(new FilterMethod(method, parameters));
            System.out.println("loadXML get method -
"+method.getName()+"\nwith parameters = ");
            for(Object param : parameters){
                System.out.println(param);
            }
        }
    }
}

} catch(Exception e) {
    throw new KettleXMLException("Unable to read step info from XML node", e);
}

}

public void setDefault(){
    this.seeds = new ArrayList<String>();
    this.seeds.add("http://portal.stela.org.br/");
    this.filter = new NullFilter();
}

public void readRep(Repository rep, long id_step, List<DatabaseMeta> databases, Map<String,Counter> counters)
throws KettleException {
    try {
        int nrfields = rep.countNrStepAttributes(id_step, "seed");
        for (int i = 0; i < nrfields; i++) {

```

```

        this.seeds.add(rep.getStepAttributeString(id_step, i, "seed"));
    }

    this.setFilter(rep.getStepAttributeString(id_step, "filter"));

    int nrMethods = rep.countNrStepAttributes(id_step, "method_name");
    int k=0;
    for(int i=0; i<nrMethods; i++){
        String methodName = rep.getStepAttributeString(id_step, i, "method_name");
        int nrParameters = (int) rep.getStepAttributeInteger(id_step, i, "nrParameters");
        Object[] parameters = new Object[nrParameters];
        for(int j=0; j<nrParameters; j++){
            String parameterType = rep.getStepAttributeString(id_step, k,
"parameter_type");
            String parameterValue = rep.getStepAttributeString(id_step, k,
"parameter_value");

            if(parameterType.equals(Boolean.class.toString())){
                parameters[j] = Boolean.parseBoolean(parameterValue);
            }else if(parameterType.equals(String.class.toString())){
                parameters[j] = parameterValue;
            }
            k++;
        }

        for(Method method : this.filter.getClass().getMethods()){
            if(method.toString().equals(methodName)){
                filterMethods.add(new FilterMethod(method, parameters));
                System.out.println("readRep      get      method      -
"+method.getName()+"\nwith parameters = ");
                for(Object param : parameters){
                    System.out.println(param);
                }
            }
        }
    }

    } catch (KettleDatabaseException dbe) {
        throw new KettleException("error reading step with id_step=" + id_step + " from the repository",
dbe);
    } catch (Exception e) {
        throw new KettleException( "Unexpected error reading step with id_step=" + id_step + " from the
repository", e);
    }
}

public void saveRep(Repository rep, long id_transformation, long id_step) throws KettleException {
    try {
        for (int i=0; i<seeds.size(); i++) {
            rep.saveStepAttribute(id_transformation, id_step, i, "seed", seeds.get(i));
        }
        rep.saveStepAttribute(id_transformation, id_step, "filter", this.getStringFilter());
    }
}

```

```

        int k=0;
        for(int i=0; i<filterMethods.size(); i++){
            FilterMethod filterMethod = filterMethods.get(i);
            Object[] parameters = filterMethod.getParameters();
            rep.saveStepAttribute(id_transformation, id_step, i, "method_name",
filterMethod.getMethod().toString());
            rep.saveStepAttribute(id_transformation, id_step, i, "nrParameters", parameters.length);
            for(int j=0; j<parameters.length; j++){
                rep.saveStepAttribute(id_transformation, id_step, k, "parameter_type",
parameters[j].getClass().toString());
                rep.saveStepAttribute(id_transformation, id_step, k, "parameter_value",
parameters[j].toString());
                k++;
            }
        }
    } catch (KettleDatabaseException dbe) {
        throw new KettleException("Unable to save step information to the repository, id_step=" + id_step,
dbe);
    }
}

public void check(List<CheckResultInterface> remarks, TransMeta transmeta, StepMeta stepMeta, RowMetaInterface
prev, String input[], String output[], RowMetaInterface info)
{
    CheckResult cr;
    if (prev==null || prev.size()==0)
    {
        cr = new CheckResult(CheckResult.TYPE_RESULT_WARNING, "Not receiving any fields from
previous steps!", stepMeta);
        remarks.add(cr);
    }
    else
    {
        cr = new CheckResult(CheckResult.TYPE_RESULT_OK, "Step is connected to previous one,
receiving "+prev.size()+" fields", stepMeta);
        remarks.add(cr);
    }

    // See if we have input streams leading to this step!
    if (input.length>0)
    {
        cr = new CheckResult(CheckResult.TYPE_RESULT_OK, "Step is receiving info from other
steps.", stepMeta);
        remarks.add(cr);
    }
    else
    {
        cr = new CheckResult(CheckResult.TYPE_RESULT_ERROR, "No input received from other
steps!", stepMeta);
        remarks.add(cr);
    }
}

```

```

        }
    }

    public StepDialogInterface getDialog(Shell shell, StepMetaInterface meta, TransMeta transMeta, String name)
    {
        return new CollectorPluginDialog(shell, meta, transMeta, name);
    }

    public StepInterface getStep(StepMeta stepMeta, StepDataInterface stepDataInterface, int cnr, TransMeta transMeta,
Trans disp)
    {
        return new CollectorPlugin(stepMeta, stepDataInterface, cnr, transMeta, disp);
    }

    public StepDataInterface getStepData()
    {
        return new CollectorPluginData();
    }
}

```

FilterDialogMethod.java

```

package info.stela.ekp.iscollector;

import java.lang.reflect.Method;
import java.util.ArrayList;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layout.FormData;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.ui.core.PropsUI;
import org.pentaho.di.ui.core.widget.ColumnInfo;
import org.pentaho.di.ui.core.widget.TableView;

public class FilterDialogMethod {
    private Method method;
    private Label wIMethodName;
    private Control[] wcFilterControls;
    private final boolean isUnique;
    private ArrayList<Label> labels;

    private CollectorPluginDialog collectorPluginDialog;

```

```

private PropsUI props;
private Composite wComposite;
private ISCollectorGUIAnnotation anno;
private TransMeta transMeta;
private ModifyListener lsMod;

public FilterDialogMethod(CollectorPluginDialog collectorPluginDialog, PropsUI props, Composite composite,
ISCollectorGUIAnnotation anno, Method method, TransMeta transMeta, ModifyListener lsMod){
    this.collectorPluginDialog = collectorPluginDialog;
    this.props = props;
    this.wComposite = composite;
    this.method = method;
    this.anno = anno;
    this.isUnique = anno.unique();
    this.transMeta = transMeta;
    this.lsMod = lsMod;

    if(isUnique){
        this.drawFilterRow();
    } else {
        this.drawFilterTable();
    }
}

public boolean isUnique() {
    return isUnique;
}

public Method getMethod() {
    return method;
}

public Control[] getWcFilterControls() {
    return wcFilterControls;
}

public void dispose() {
    this.wMethodName.dispose();
    for(Control control : wcFilterControls){
        control.dispose();
    }
    if(labels!=null){
        for(Label label : labels){
            label.dispose();
        }
    }
}

private void drawFilterRow() {
    Group wMethodGroup = new Group(wComposite, SWT.NONE);
    props.setLook(wMethodGroup);
}

```

```

FormData fdComposite = new FormData();
fdComposite.top = new FormAttachment(this.collectorPluginDialog.attach, 0);
fdComposite.left = new FormAttachment(this.collectorPluginDialog.middle, CollectorPluginDialog.MARGIN);
fdComposite.right = new FormAttachment(100, -CollectorPluginDialog.MARGIN);
wMethodGroup.setLayoutData(fdComposite);
GridLayout gridLayout = new GridLayout(6, false);
gridLayout.verticalSpacing = CollectorPluginDialog.MARGIN;
gridLayout.horizontalSpacing = CollectorPluginDialog.MARGIN;
gridLayout.marginWidth = 0;
gridLayout.marginHeight = 0;
wMethodGroup.setLayout(gridLayout);

labels = new ArrayList<Label>();
Class<?>[] parameters = method.getParameterTypes();
wcFilterControls = new Control[parameters.length];
for(int i=0; i<parameters.length; i++){
    Label wParameter = new Label(wMethodGroup, SWT.RIGHT);
    wParameter.setText(anno.parameters()[i]);
    props.setLook(wParameter);
    wParameter.setLayoutData(new GridData(GridData.HORIZONTAL_ALIGN_FILL |
GridData.GRAB_HORIZONTAL));
    labels.add(wParameter);

    if(parameters[i].getSimpleName().equals("boolean")){
        wcFilterControls[i] = new Combo(wMethodGroup, SWT.DROP_DOWN |
SWT.READ_ONLY);

        props.setLook(wcFilterControls[i]);
        ((Combo)wcFilterControls[i]).add("true");
        ((Combo)wcFilterControls[i]).add("false");
    }else if(parameters[i].getSimpleName().equals("String")){
        wcFilterControls[i] = new Text(wMethodGroup, SWT.SINGLE | SWT.LEFT |
SWT.BORDER);

        props.setLook(wcFilterControls[i]);
    }
    wcFilterControls[i].setLayoutData(new GridData(GridData.HORIZONTAL_ALIGN_FILL |
GridData.GRAB_HORIZONTAL));
}
this.collectorPluginDialog.attach = wMethodGroup;

wMethodName = new Label(wComposite, SWT.RIGHT);
wMethodName.setText(anno.name());
props.setLook(wMethodName);
FormData fdlFilterControl = new FormData();
fdlFilterControl.left = new FormAttachment(0, 0);
fdlFilterControl.right = new FormAttachment(this.collectorPluginDialog.middle,
CollectorPluginDialog.MARGIN);
fdlFilterControl.top = new FormAttachment(this.collectorPluginDialog.attach, 0, SWT.CENTER);
wMethodName.setLayoutData(fdlFilterControl);
}

private void drawFilterTable() {

```

```

Class<?>[] parameters = method.getParameterTypes();
ColumnInfo[] colinf = new ColumnInfo[parameters.length];
for(int i=0; i<parameters.length; i++){
    if(parameters[i].getSimpleName().equals("boolean")){
        colinf[i] = new ColumnInfo(anno.parameters()[i],
ColumnInfo.COLUMN_TYPE_CCOMBO, CollectorPluginDialog.TRUE_FALSE_COMBO, true);
    }else if(parameters[i].getSimpleName().equals("String")){
        colinf[i] = new ColumnInfo(anno.parameters()[i], ColumnInfo.COLUMN_TYPE_TEXT,
false);
    }
}
wcFilterControls = new Control[1];
Control wcFilterTable = new TableView(transMeta, wComposite,
SWT.FULL_SELECTION | SWT.SINGLE | SWT.BORDER, colinf, 3, IsMod, props);

FormData fdFilterTable = new FormData();
fdFilterTable.left = new FormAttachment(this.collectorPluginDialog.middle, CollectorPluginDialog.MARGIN);
fdFilterTable.right = new FormAttachment(100, -CollectorPluginDialog.MARGIN);
fdFilterTable.top = new FormAttachment(this.collectorPluginDialog.attach,
CollectorPluginDialog.MARGIN);
wcFilterTable.setLayoutData(fdFilterTable);
this.setWcFilterControl(wcFilterTable);

this.collectorPluginDialog.attach = wcFilterTable;
//draw label
wMethodName = new Label(wComposite, SWT.RIGHT);
wMethodName.setText(anno.name());
props.setLook(wMethodName);
FormData fdlFilterControl = new FormData();
fdlFilterControl.left = new FormAttachment(0, 0);
fdlFilterControl.right = new FormAttachment(this.collectorPluginDialog.middle,
CollectorPluginDialog.MARGIN);
fdlFilterControl.top = new FormAttachment(this.collectorPluginDialog.attach, 0, SWT.CENTER);
wMethodName.setLayoutData(fdlFilterControl);
}

private void setWcFilterControl(Control wcFilterControl){
    this.wcFilterControls = new Control[1];
    wcFilterControls[0] = wcFilterControl;
}
}

```

FilterMethod.java

```

package info.stela.ekp.iscollector;

import java.lang.reflect.Method;

import org.pentaho.di.core.Const;
import org.pentaho.di.core.xml.XMLHandler;

public class FilterMethod {

```

```

private Method method;
private Object[] parameters;

public FilterMethod(Method method, Object[] parameters) {
    this.method = method;
    this.parameters = parameters;
}

public Method getMethod() {
    return method;
}

public void setMethod(Method method) {
    this.method = method;
}

public Object[] getParameters() {
    return parameters;
}

public void setParameters(Object[] parameters) {
    this.parameters = parameters;
}

public String getXML() {
    StringBuilder retval = new StringBuilder();

    retval.append("<method>").append(Const.CR);
    retval.append(XMLHandler.addTagValue("method_name", method.toString()));
    retval.append("<parameters>").append(Const.CR);
    for(Object parameter : parameters){
        retval.append("<parameter>").append(Const.CR);
        retval.append(XMLHandler.addTagValue("parameter_type",
parameter.getClass().toString()));
        retval.append(XMLHandler.addTagValue("parameter_value",
parameter.toString()));
        retval.append("</parameter>").append(Const.CR);
    }
    retval.append("</parameters>").append(Const.CR);
    retval.append("</method>").append(Const.CR);

    return retval.toString();
}
}

```

ISCollectorGUIAnnotation.java

```

package info.stela.ekp.iscollector;

import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

```



```

@Retention(RetentionPolicy.RUNTIME)
public @interface ISCollectorGUIAnnotation{
    String name();
    boolean unique();
    String[] parameters();
}

```

Messages.java

```

package info.stela.ekp.iscollector;

import org.pentaho.di.i18n.BaseMessages;

public class Messages
{
    public static final String packageName = Messages.class.getPackage().getName();

    public static String getString(String key)
    {
        return BaseMessages.getString(packageName, key);
    }

    public static String getString(String key, String param1)
    {
        return BaseMessages.getString(packageName, key, param1);
    }

    public static String getString(String key, String param1, String param2)
    {
        return BaseMessages.getString(packageName, key, param1, param2);
    }

    public static String getString(String key, String param1, String param2, String param3)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4, String param5)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4, param5);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4, String param5,
String param6)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4, param5, param6);
    }
}

```

```

    }
}

```

plugin.properties

```

## Add or remove filters
## Do not use spaces.
## MyFilter=myfilter.jar|my.filter.package.MyFilterClassName
URLRegexFilter=info.stela.ekp.iscollector.filter.URLRegexFilter
URLLiteralFilter=info.stela.ekp.iscollector.filter.URLLiteralFilter
NoFilter=info.stela.ekp.iscollector.filter.NullFilter

```

plugin.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<plugin
  id="Collector Input"
  iconfile="CPI.png"
  description="Collector Plugin Input"
  tooltip="This is a collector plugin input step"
  category="Input"
  classname="info.stela.ekp.iscollector.CollectorPluginMeta">
  <libraries>
    <library name="collector-0.0.1.jar"/>
    <library name="dependencies/icu4j-4_0.jar"/>
    <library name="dependencies/jcifs-1.3.1.jar"/>
    <library name="dependencies/retriever-0.17.0.jar"/>
    <library name="DefaultFilters.jar"/>
  </libraries>

  <localized_category>
    <category locale="en_US">Input</category>
  </localized_category>
  <localized_description>
    <description locale="en_US">Collector Plugin Input</description>
  </localized_description>
  <localized_tooltip>
    <tooltip locale="en_US">This is a collector plugin input step</tooltip>
  </localized_tooltip>
</plugin>

```

Apêndice 3 - Código-fonte do LuceneInputPlugin

LuceneInputPlugin

```

package input;

import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import org.apache.lucene.analysis.StopAnalyzer;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.Searcher;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.exception.KettleException;
import org.pentaho.di.core.exception.KettleStepException;
import org.pentaho.di.core.row.RowMeta;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStep;
import org.pentaho.di.trans.step.StepDataInterface;
import org.pentaho.di.trans.step.StepInterface;
import org.pentaho.di.trans.step.StepMeta;
import org.pentaho.di.trans.step.StepMetaInterface;

public class LuceneInputPlugin extends BaseStep implements StepInterface {
    private LuceneInputPluginData data;
    private LuceneInputPluginMeta meta;

    private String indexFiles = "D:/andre/Documents/Faculdade/TCC/ETL/Kettle tests/index_alpha";
    Searcher searcher = null;

    private static String[] wordList;/*
                                                * = {"test", "oil", "japan", "brasil",
                                                * "report"};
                                                */

    static {
        File file = new File(
            "D:/andre/Documents/Faculdade/TCC/ETL/Kettle tests/selectedWordList.txt");
        FileInputStream fis = null;
        BufferedInputStream bis = null;
        DataInputStream dis = null;
    }

```

```

try {
    fis = new FileInputStream(file);
    bis = new BufferedInputStream(fis);
    dis = new DataInputStream(bis);

    wordList = new String[15800];
    int count = 0;
    ;
    while (dis.available() != 0) {
        wordList[count] = dis.readLine();
        count++;
    }
    fis.close();
    bis.close();
    dis.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}

public LuceneInputPlugin(StepMeta s, StepDataInterface stepDataInterface,
    int c, TransMeta t, Trans dis) {
    super(s, stepDataInterface, c, t, dis);
}

public boolean processRow(StepMetaInterface smi, StepDataInterface sdi)
    throws KettleException {
    meta = (LuceneInputPluginMeta) smi;
    data = (LuceneInputPluginData) sdi;

    if (first) {
        first = false;

        data.outputRowMeta = new RowMeta();
        meta.getFields(data.outputRowMeta, getStepname(), null, null, this);

        try {
            searcher = new IndexSearcher(indexFiles);
        } catch (IOException e) {
            logBasic("Unable to open index file: " + indexFiles);
        }
    }

    // doing the query
    Query query;
    Hits hits;
    QueryParser qp = new QueryParser("content", new StopAnalyzer());

    String ano;

```

```

String mes;
String from;
String project;
String to;

int wordCount = 0;

for (String word : wordList) {
    if (word != null) {
        wordCount++;
        logBasic("wordCount = " + wordCount + " - word = " + word);
        if (!isStopped()) {
            try {
                query = qp.parse(word);

                hits = searcher.search(query);

                if (hits.length() > 0) {
                    // for each document found, send several outputs
                    for (int i = 0; i < hits.length(); i++) {
                        mes = hits.doc(i).get("MES");
                        ano = hits.doc(i).get("ANO");
                        from = hits.doc(i).get("SENDER");
                        project = hits.doc(i).get("Projeto");
                        to = hits.doc(i).get("destiny");

                        int com = from.indexOf(".com");
                        from = from.substring(0, com + 4);
                        com = to.indexOf(".com");
                        if (com + 4 < to.length()) {
                            to = to.substring(0, com + 4);
                        }

                        if (from.length() > 0) {
                            sendRow(mes, ano, from, "2", word,
"1");
                        }
                        if (project.length() > 0) {
                            sendRow(mes, ano, project, "3",
word, "1");
                        }
                        if (to.length() > 0) {
                            sendRow(mes, ano, to, "2", word,
"1");
                        }
                    }
                }
            } catch (ParseException e) {
                logBasic("Unable to parse: " + word);
                break;
            } catch (IOException e) {

```

```

        logBasic("IO Error.");
        break;
    }
}

}

setOutputDone();
close(searcher);

return false;
}

private void sendRow(String mes, String ano, String entity,
                    String entityType, String term, String termType)
                    throws KettleStepException {
    Object[] row = { mes, ano, entity, entityType, term, termType };
    // logBasic("mes/ano= "+mes+"/"+"ano+" entidade= "+entity+" word = "+term);

    // Object[] outputRow = RowDataUtil.addValueData(row,
    // data.outputRowMeta.size()-5, mes);
    // outputRow = RowDataUtil.addValueData(outputRow,
    // data.outputRowMeta.size()-4, ano);
    // outputRow = RowDataUtil.addValueData(outputRow,
    // data.outputRowMeta.size()-3, entity);
    // outputRow = RowDataUtil.addValueData(outputRow,
    // data.outputRowMeta.size()-2, entityType);
    // outputRow = RowDataUtil.addValueData(outputRow,
    // data.outputRowMeta.size()-1, term);

    putRow(data.outputRowMeta, row); // copy row to possible alternate
// rowset(s).
}

private void close(Searcher searcher2) {
    try {
        searcher2.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public boolean init(StepMetaInterface smi, StepDataInterface sdi) {
    meta = (LuceneInputPluginMeta) smi;
    data = (LuceneInputPluginData) sdi;

    return super.init(smi, sdi);
}

public void dispose(StepMetaInterface smi, StepDataInterface sdi) {
    meta = (LuceneInputPluginMeta) smi;

```

```

        data = (LuceneInputPluginData) sdi;

        super.dispose(smi, sdi);
    }

    //
    // Run is were the action happens!
    public void run() {
        logBasic("Starting to run...");
        try {
            while (processRow(meta, data) && !isStopped())
                ;
        } catch (Exception e) {
            logError("Unexpected error : " + e.toString());
            logError(Const.getStackTracker(e));
            setErrors(1);
            stopAll();
        } finally {
            dispose(meta, data);
            logBasic("Finished, processing " + linesRead + " rows");
            markStop();
        }
    }
}

```

LuceneInputPluginData

```

package input;

import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.trans.step.BaseStepData;
import org.pentaho.di.trans.step.StepDataInterface;

public class LuceneInputPluginData extends BaseStepData implements StepDataInterface
{

    public RowMetaInterface outputRowMeta;

    public LuceneInputPluginData()
    {
        super();
    }
}

```

LuceneInputPluginDialog

```

package input;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.events.ShellAdapter;
import org.eclipse.swt.events.ShellEvent;

```

```

import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layout.FormData;
import org.eclipse.swt.layout.FormLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.row.ValueMetaAndData;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStepMeta;
import org.pentaho.di.trans.step.StepDialogInterface;
import org.pentaho.di.ui.core.dialog.EnterValueDialog;
import org.pentaho.di.ui.trans.step.BaseStepDialog;

public class LuceneInputDialog extends BaseStepDialog implements StepDialogInterface
{
    private LuceneInputDialogMeta input;
    private ValueMetaAndData value;

    private Label    wValName;
    private Text     wValName;
    private FormData fdValName, fdValName;

    private Label    wValue;
    private Button   wbValue;
    private Text     wValue;
    private FormData fdValue, fdbValue, fdValue;

    public LuceneInputDialog(Shell parent, Object in, TransMeta transMeta, String sname)
    {
        super(parent, (BaseStepMeta)in, transMeta, sname);
        input=(LuceneInputDialogMeta)in;
        input.getTableFields();
    }

    public String open()
    {
        Shell parent = getParent();
        Display display = parent.getDisplay();

        shell = new Shell(parent, SWT.DIALOG_TRIM | SWT.RESIZE | SWT.MIN | SWT.MAX);
        props.setLook( shell );
        setShellImage(shell, input);

        ModifyListener lsMod = new ModifyListener()
        {

```



```

        public void modifyText(ModifyEvent e)
        {
            input.setChanged();
        }
    };
    changed = input.hasChanged();

    FormLayout formLayout = new FormLayout ();
    formLayout.marginWidth = Const.FORM_MARGIN;
    formLayout.marginHeight = Const.FORM_MARGIN;

    shell.setLayout(formLayout);
    shell.setText(Messages.getString("DummyPluginDialog.Shell.Title")); //$NON-NLS-1$

    int middle = props.getMiddlePct();
    int margin = Const.MARGIN;

    // Stepname line
    wStepname=new Label(shell, SWT.RIGHT);
    wStepname.setText(Messages.getString("DummyPluginDialog.StepName.Label")); //$NON-NLS-1$
    props.setLook( wStepname );
    fdStepname=new FormData();
    fdStepname.left = new FormAttachment(0, 0);
    fdStepname.right= new FormAttachment(middle, -margin);
    fdStepname.top = new FormAttachment(0, margin);
    wStepname.setLayoutData(fdStepname);
    wStepname=new Text(shell, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
    wStepname.setText(stepname);
    props.setLook( wStepname );
    wStepname.addModifyListener(IsMod);
    fdStepname=new FormData();
    fdStepname.left = new FormAttachment(middle, 0);
    fdStepname.top = new FormAttachment(0, margin);
    fdStepname.right= new FormAttachment(100, 0);
    wStepname.setLayoutData(fdStepname);

    // ValName line
    wValName=new Label(shell, SWT.RIGHT);
    wValName.setText(Messages.getString("DummyPluginDialog.ValueName.Label")); //$NON-NLS-1$
    props.setLook( wValName );
    fdValName=new FormData();
    fdValName.left = new FormAttachment(0, 0);
    fdValName.right= new FormAttachment(middle, -margin);
    fdValName.top = new FormAttachment(wStepname, margin);
    wValName.setLayoutData(fdValName);
    wValName=new Text(shell, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
    props.setLook( wValName );
    wValName.addModifyListener(IsMod);
    fdValName=new FormData();
    fdValName.left = new FormAttachment(middle, 0);
    fdValName.right= new FormAttachment(100, 0);

```

```

        fdValName.top = new FormAttachment(wStepname, margin);
        wValName.setLayoutData(fdValName);

        // Value line
        wValue=new Label(shell, SWT.RIGHT);
        wValue.setText(Messages.getString("DummyPluginDialog.ValueToAdd.Label")); //$NON-NLS-1$
        props.setLook( wValue );
        fdValue=new FormData();
        fdValue.left = new FormAttachment(0, 0);
        fdValue.right= new FormAttachment(middle, -margin);
        fdValue.top = new FormAttachment(wValName, margin);
        wValue.setLayoutData(fdValue);

        wbValue=new Button(shell, SWT.PUSH| SWT.CENTER);
        props.setLook( wbValue );
        wbValue.setText(Messages.getString("System.Button.Edit")); //$NON-NLS-1$
        fdbValue=new FormData();
        fdbValue.right= new FormAttachment(100, 0);
        fdbValue.top = new FormAttachment(wValName, margin);
        wbValue.setLayoutData(fdbValue);

        wValue=new Text(shell, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
        props.setLook( wValue );
        wValue.addModifyListener(IsMod);
        fdValue=new FormData();
        fdValue.left = new FormAttachment(middle, 0);
        fdValue.right= new FormAttachment(wbValue, -margin);
        fdValue.top = new FormAttachment(wValName, margin);
        wValue.setLayoutData(fdValue);

        wbValue.addSelectionListener(new SelectionAdapter()
        {
            public void widgetSelected(SelectionEvent arg0)
            {
                ValueMetaAndData v = (ValueMetaAndData) value.clone();
                EnterValueDialog evd = new EnterValueDialog(shell, SWT.NONE, v.getValueMeta(),
v.getValueData());

                ValueMetaAndData newval = evd.open();
                if (newval!=null)
                {
                    value = newval;
                    getData();
                }
            }
        });

        // Some buttons
        wOK=new Button(shell, SWT.PUSH);
        wOK.setText(Messages.getString("System.Button.OK")); //$NON-NLS-1$
        wCancel=new Button(shell, SWT.PUSH);
        wCancel.setText(Messages.getString("System.Button.Cancel")); //$NON-NLS-1$

```

```

BaseStepDialog.positionBottomButtons(shell, new Button[] { wOK, wCancel}, margin, wValue);

    // Add listeners
    IsCancel = new Listener() { public void handleEvent(Event e) { cancel(); } };
    IsOK = new Listener() { public void handleEvent(Event e) { ok(); } };

    wCancel.addListener(SWT.Selection, IsCancel);
    wOK.addListener (SWT.Selection, IsOK );

    IsDef=new SelectionAdapter() { public void widgetDefaultSelected(SelectionEvent e) { ok(); } };

    wStepname.addSelectionListener( IsDef );
    wValName.addSelectionListener( IsDef );

    // Detect X or ALT-F4 or something that kills this window...
    shell.addShellListener( new ShellAdapter() { public void shellClosed(ShellEvent e) { cancel(); } });

    // Set the shell size, based upon previous time...
    setSize();

    getData();
    input.setChanged(changed);

    shell.open();
    while (!shell.isDisposed())
    {
        if (!display.readAndDispatch()) display.sleep();
    }
    return stepname;
}

// Read data from input (TextFileInputInfo)
public void getData()
{
    wStepname.selectAll();
    if (value!=null)
    {
        wValName.setText(value.getValueMeta().getName());
        wValue.setText(value.toString()+" (" +value.toStringMeta()+")"); //$NON-NLS-1$ //$NON-NLS-2$
    }
}

private void cancel()
{
    stepname=null;
    input.setChanged(changed);
    dispose();
}

private void ok()

```

```

        {
            stepname = wStepname.getText(); // return value
            value.getValueMeta().setName(wValName.getText());
            dispose();
        }
    }
}

```

LuceneInputPluginMeta

```

package input;
import java.util.List;
import java.util.Map;

import org.eclipse.swt.widgets.Shell;
import org.pentaho.di.core.CheckResult;
import org.pentaho.di.core.CheckResultInterface;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.Counter;
import org.pentaho.di.core.database.DatabaseMeta;
import org.pentaho.di.core.exception.KettleDatabaseException;
import org.pentaho.di.core.exception.KettleException;
import org.pentaho.di.core.exception.KettleXMLException;
import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.core.row.ValueDataUtil;
import org.pentaho.di.core.row.ValueMeta;
import org.pentaho.di.core.row.ValueMetaAndData;
import org.pentaho.di.core.row.ValueMetaInterface;
import org.pentaho.di.core.variables.VariableSpace;
import org.pentaho.di.core.xml.XMLHandler;
import org.pentaho.di.repository.Repository;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStepMeta;
import org.pentaho.di.trans.step.StepDataInterface;
import org.pentaho.di.trans.step.StepDialogInterface;
import org.pentaho.di.trans.step.StepInterface;
import org.pentaho.di.trans.step.StepMeta;
import org.pentaho.di.trans.step.StepMetaInterface;
import org.w3c.dom.Node;

public class LuceneInputPluginMeta extends BaseStepMeta implements StepMetaInterface
{
    private ValueMetaAndData value;

    public LuceneInputPluginMeta()
    {
        super(); // allocate BaseStepInfo
    }

    /**
     * @return Returns the value.
     */
}

```

```

public ValueMetaAndData getValue()
{
    return value;
}

/**
 * @param value The value to set.
 */
public void setValue(ValueMetaAndData value)
{
    this.value = value;
}

public String getXML()
{
    String retval = "";

    retval+=" <values>"+Const.CR;
    if (value!=null)
    {
        retval+=value.getXML();
    }
    retval+=" </values>"+Const.CR;

    return retval;
}

public void getFields(RowMetaInterface r, String origin, RowMetaInterface[] info, StepMeta nextStep, VariableSpace
space)
{
    ValueMetaInterface vMes = new ValueMeta("Mes", ValueMetaInterface.TYPE_STRING);
    vMes.setOrigin(origin);
    r.addValueMeta(vMes);

    ValueMetaInterface vAno = new ValueMeta("Ano", ValueMetaInterface.TYPE_STRING);
    vAno.setOrigin(origin);
    r.addValueMeta(vAno);

    ValueMetaInterface vFrom = new ValueMeta("Entity", ValueMetaInterface.TYPE_STRING);
    vFrom.setOrigin(origin);
    r.addValueMeta(vFrom);

    ValueMetaInterface vEntityType = new ValueMeta("EntityType", ValueMetaInterface.TYPE_STRING);
    vEntityType.setOrigin(origin);
    r.addValueMeta(vEntityType);

    ValueMetaInterface vContent = new ValueMeta("Term", ValueMetaInterface.TYPE_STRING);
    vContent.setOrigin(origin);
    r.addValueMeta(vContent);

    ValueMetaInterface vTermType = new ValueMeta("TermType", ValueMetaInterface.TYPE_STRING);

```

```

        vTermType.setOrigin(origin);
        r.addValueMeta(vTermType);
    }

    public Object clone()
    {
        Object retval = super.clone();
        return retval;
    }

    public void loadXML(Node stepnode, List<DatabaseMeta> databases, Map<String,Counter> counters)
        throws KettleXMLException
    {
        try
        {
            value = new ValueMetaAndData();

            Node valnode = XMLHandler.getSubNode(stepnode, "values", "value");
            if (valnode!=null)
            {
                System.out.println("reading value in "+valnode);
                value.loadXML(valnode);
            }
        }
        catch(Exception e)
        {
            throw new KettleXMLException("Unable to read step info from XML node", e);
        }
    }

    public void setDefault()
    {
        value = new ValueMetaAndData( new ValueMeta("valuename", ValueMetaInterface.TYPE_NUMBER), new
Double(123.456) );
        value.getValueMeta().setLength(12);
        value.getValueMeta().setPrecision(4);
    }

    public void readRep(Repository rep, long id_step, List<DatabaseMeta> databases, Map<String,Counter> counters)
        throws KettleException
    {
        try
        {
            String name = rep.getStepAttributeString (id_step, 0, "value_name");
            String typedesc = rep.getStepAttributeString (id_step, 0, "value_type");
            String text = rep.getStepAttributeString (id_step, 0, "value_text");
            boolean isnull = rep.getStepAttributeBoolean(id_step, 0, "value_null");
            int length = (int)rep.getStepAttributeInteger(id_step, 0, "value_length");
            int precision = (int)rep.getStepAttributeInteger(id_step, 0, "value_precision");

            int type = ValueMeta.getType(typedesc);

```

```

        value = new ValueMetaAndData(new ValueMeta(name, type), null);
        value.getValueMeta().setLength(length);
value.getValueMeta().setPrecision(precision);

        if (isnull)
        {
            value.setValueData(null);
        }
        else
        {
ValueMetaInterface stringMeta = new ValueMeta(name, ValueMetaInterface.TYPE_STRING);
            if (type!=ValueMetaInterface.TYPE_STRING) text=ValueDataUtil.trim(text);
            value.setValueData( value.getValueMeta().convertData(stringMeta, text));
        }
    }
    catch(KettleDatabaseException dbe)
    {
        throw new KettleException("error reading step with id_step="+id_step+" from the repository", dbe);
    }
    catch(Exception e)
    {
        throw new KettleException("Unexpected error reading step with id_step="+id_step+" from the
repository", e);
    }
}

public void saveRep(Repository rep, long id_transformation, long id_step) throws KettleException
{
    try
    {
        rep.saveStepAttribute(id_transformation,          id_step,          "value_name",
value.getValueMeta().getName());
        rep.saveStepAttribute(id_transformation,          id_step,          0,          "value_type",
value.getValueMeta().getTypeDesc());
        rep.saveStepAttribute(id_transformation,          id_step,          0,          "value_text",
value.getValueMeta().getString(value.getValueData()));
        rep.saveStepAttribute(id_transformation,          id_step,          0,          "value_null",
value.getValueMeta().isNull(value.getValueData()));
        rep.saveStepAttribute(id_transformation,          id_step,          0,          "value_length",
value.getValueMeta().getLength());
        rep.saveStepAttribute(id_transformation,          id_step,          0,          "value_precision",
value.getValueMeta().getPrecision());
    }
    catch(KettleDatabaseException dbe)
    {
        throw new KettleException("Unable to save step information to the repository, id_step="+id_step,
dbe);
    }
}
}

```

```

    public void check(List<CheckResultInterface> remarks, TransMeta transmeta, StepMeta stepMeta, RowMetaInterface
prev, String input[], String output[], RowMetaInterface info)
    {
        CheckResult cr;
        if (prev==null || prev.size()==0)
        {
            cr = new CheckResult(CheckResult.TYPE_RESULT_WARNING, "Not receiving any fields from
previous steps!", stepMeta);
            remarks.add(cr);
        }
        else
        {
            cr = new CheckResult(CheckResult.TYPE_RESULT_OK, "Step is connected to previous one,
receiving "+prev.size()+" fields", stepMeta);
            remarks.add(cr);
        }

        // See if we have input streams leading to this step!
        if (input.length>0)
        {
            cr = new CheckResult(CheckResult.TYPE_RESULT_OK, "Step is receiving info from other
steps.", stepMeta);
            remarks.add(cr);
        }
        else
        {
            cr = new CheckResult(CheckResult.TYPE_RESULT_ERROR, "No input received from other
steps!", stepMeta);
            remarks.add(cr);
        }
    }

    public StepDialogInterface getDialog(Shell shell, StepMetaInterface meta, TransMeta transMeta, String name)
    {
        return new LuceneInputPluginDialog(shell, meta, transMeta, name);
    }

    public StepInterface getStep(StepMeta stepMeta, StepDataInterface stepDataInterface, int cnr, TransMeta transMeta,
Trans disp)
    {
        return new LuceneInputPlugin(stepMeta, stepDataInterface, cnr, transMeta, disp);
    }

    public StepDataInterface getStepData()
    {
        return new LuceneInputPluginData();
    }
}

```

Messages

```
package input;
```



```
import org.pentaho.di.i18n.BaseMessages;

public class Messages
{
    public static final String packageName = Messages.class.getPackage().getName();

    public static String getString(String key)
    {
        return BaseMessages.getString(packageName, key);
    }

    public static String getString(String key, String param1)
    {
        return BaseMessages.getString(packageName, key, param1);
    }

    public static String getString(String key, String param1, String param2)
    {
        return BaseMessages.getString(packageName, key, param1, param2);
    }

    public static String getString(String key, String param1, String param2, String param3)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4, String param5)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4, param5);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4, String param5,
String param6)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4, param5, param6);
    }
}
```

Apêndice 4 - Código-fonte do LuceneOutputPlugin

DataPersistence

```

package be.ibridge.kettle.lucene;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

/**
 * Enables the flush and retrieve of the serialized objects.
 */
public final class DataPersistence {

    /**
     * Stores a serialized object into a file.
     * @param file path and file name.
     * @param object serialized object
     * @throws Exception exception during flushing process.
     */
    public static void store(final String file, final Object object)
    throws Exception {
        try {
            final FileOutputStream objFileOS = new FileOutputStream(file);
            final ObjectOutputStream objOS = new ObjectOutputStream(objFileOS);
            objOS.writeObject(object);
            objOS.flush();
            objOS.close();
            objFileOS.close();
        } catch (final Exception exc) {
            throw new Exception(
                "Error occurred in the store method in persistence process. Error: "+
                exc.getMessage());
        }
    }

    /**
     * Loads from a serialized object to an object instance.
     * @param file path and file name to be retrieved.
     * @return an object instance.
     * @throws Exception exception during loading process.
     */
    public static Object retrieve(final String file) throws Exception {
        Object objGeneral = null;
        try {
            final File objFile = new File(file);
            if (objFile.exists()) {

```

```

        final FileInputStream objFileIS = new FileInputStream(file);
        final ObjectInputStream objIS = new ObjectInputStream(objFileIS);
        objGeneral = objIS.readObject();
        objIS.close();
        objFileIS.close();
    }
} catch (final Exception exc) {
    throw new Exception(
        "Error occurred in the retrieve method in persistence process. Error: "+
        exc.getMessage());
}
return objGeneral;
}
}

```

LucenePlugin

```
package be.ibridge.kettle.lucene;
```

```
import java.io.IOException;
import java.sql.Timestamp;
import java.util.ArrayList;
```

```
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.Term;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.exception.KettleException;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStep;
import org.pentaho.di.trans.step.StepDataInterface;
import org.pentaho.di.trans.step.StepInterface;
import org.pentaho.di.trans.step.StepMeta;
import org.pentaho.di.trans.step.StepMetaInterface;
```

```
public class LucenePlugin extends BaseStep implements StepInterface {
    private LucenePluginData data;
    private LucenePluginMeta meta;

    private IndexWriter writer;
    private String keyField = "key";
    private Integer keyPos;
    private Integer datePos;
    private ArrayList<LucenePluginField> fields = new ArrayList<LucenePluginField>();
    private Parameter control;
    private Timestamp date;
    private Timestamp initialDate;

```

```

public LucenePlugin(StepMeta s, StepDataInterface stepDataInterface, int c,
    TransMeta t, Trans dis) {
    super(s, stepDataInterface, c, t, dis);
}

public boolean processRow(StepMetaInterface smi, StepDataInterface sdi) throws KettleException {
    meta = (LucenePluginMeta) smi;
    data = (LucenePluginData) sdi;

    Object[] r = getRow(); // get row, blocks when needed!
    if (r == null) { // no more input to be expected...
        setOutputDone();
        if(!first){
            try {
                control.setLastIndexingDate(initialDate);
                control.save();

                writer.optimize();
                writer.close();
                logBasic("writer closed");
            } catch (IOException e){
                e.printStackTrace();
                logBasic("IOException " + e);
            } catch (Exception e) {
                e.printStackTrace();
                logBasic("Exception " + e);
            }
        }
        return false;
    }

    if (first) { //first row
        logBasic("first row");
        first = false;

        fields = meta.getIndexFields();
        logBasic("first row");
        try {
            keyPos = meta.getKey();
            datePos = meta.getDate();

            writer = new IndexWriter(meta.getDirPath(), meta.getAnalyzer(),
meta.isBuildFromScratch());

            logBasic("dir = "+meta.getDirPath()+ " / analyzer = "+meta.getAnalyzer()+
                " / build from scratch = "+meta.isBuildFromScratch());

        } catch (IOException e){
            e.printStackTrace();
        } catch (Exception e){
            e.printStackTrace();
        }
    }
}

```

```

    }

    try {
        Document document = new Document();
        for(int i=0; i<fields.size(); i++){
            if(r[i]==null){
                r[i]="";
            }
            if(!fields.get(i).getStore().equals(Store.NO) || !fields.get(i).getIndex().equals(Index.NO)){
                document.add(new Field(fields.get(i).getName(), r[i].toString(),
fields.get(i).getStore(), Index.TOKENIZED));
            }
        }

        if(keyPos!=null){
            if(datePos!=null){
                date = (Timestamp) r[datePos];
            }else{
                date = initialDate;
            }

            if(control.getLastIndexingDate()==null){
                writer.updateDocument(new Term(keyField, r[keyPos].toString()), document);
            }else if(date.getTime()>control.getLastIndexingDate().getTime()){
                writer.updateDocument(new Term(keyField, r[keyPos].toString()), document);
            }
        } else{
            writer.addDocument(document);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return true;
}

public boolean init(StepMetaInterface smi, StepDataInterface sdi) {
    meta = (LucenePluginMeta) smi;
    data = (LucenePluginData) sdi;

    return super.init(smi, sdi);
}

public void dispose(StepMetaInterface smi, StepDataInterface sdi) {
    meta = (LucenePluginMeta) smi;
    data = (LucenePluginData) sdi;

    super.dispose(smi, sdi);
}

// Run is were the action happens!

```

```

public void run() {
    logBasic("Starting to run...");
    try {
        control = Parameter.create(meta.getDirPath()+"/control/myfile.ctr");
        initialDate = new Timestamp(System.currentTimeMillis());

        while (processRow(meta, data) && !isStopped())
            ;
    } catch (Exception e) {
        logError("Unexpected error : " + e.toString());
        logError(Const.getStackTracker(e));
        setErrors(1);
        stopAll();
    } finally {
        dispose(meta, data);
        logBasic("Finished, processing " + this.getLinesRead() + " rows");
        markStop();
    }
}
}

```

LucenePluginData

```

package be.ibridge.kettle.lucene;

import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.trans.step.BaseStepData;
import org.pentaho.di.trans.step.StepDataInterface;

public class LucenePluginData extends BaseStepData implements StepDataInterface
{

    public RowMetaInterface outputRowMeta;

    public LucenePluginData()
    {
        super();
    }
}

```

LucenePluginDialog

```

package be.ibridge.kettle.lucene;

import java.io.File;
import java.util.ArrayList;

import org.apache.lucene.document.Field;
import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.ScrolledComposite;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;

```

```

import org.eclipse.swt.events.SelectionListener;
import org.eclipse.swt.events.ShellAdapter;
import org.eclipse.swt.events.ShellEvent;
import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layout.FormData;
import org.eclipse.swt.layout.FormLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.DirectoryDialog;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.exception.KettleStepException;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStepMeta;
import org.pentaho.di.trans.step.StepDialogInterface;
import org.pentaho.di.ui.trans.step.BaseStepDialog;

public class LucenePluginDialog extends BaseStepDialog implements StepDialogInterface {

    public static final int MARGIN = Const.MARGIN;
    public int middle;
    public Control attach;

    private LucenePluginMeta input; //LucenePluginMeta class
    private String[] dataSetRowNames; //rows from previous steps

    //local variables
    private ArrayList<LucenePluginField> indexFields;
    private ArrayList<LucenePluginDialogRow> rows = new ArrayList<LucenePluginDialogRow>();

    ScrolledComposite wScrolledComposite; //holds composite
    Composite wComposite; //holds all UI elements

    //directory path UI elements
    private Label wDirPath;
    private Text wDirPath;
    private Button wbDirPath;
    private FormData fdlDirPath, fdDirPath, fdbDirPath;
    //build index from scratch checkbox
    private Label wBuildFromScratch;
    private Button wBuildFromScratch;
    private FormData fdlBuildFromScratch, fdBuildFromScratch;
    //analyzer combo box

```

```

private Label wAnalyzer;
private Combo wAnalyzer;
private FormData fdAnalyzer, fdAnalyzer;
//fields header labels
private Label wTableFieldName, wIndexFieldName, wKey, wDate, wStore, wIndex;

public LucenePluginDialog(Shell parent, Object in, TransMeta transMeta, String sname) {
    super(parent, (BaseStepMeta) in, transMeta, sname);
    this.input = (LucenePluginMeta) in;
    this.indexFields = input.getIndexFields();
    try {
        this.dataSetRowNames = transMeta.getPrevStepFields(stepname).getFieldNames();
    } catch (KettleStepException e) {
        System.out.println("ERROR: Couldn't get last step fields. ");
        e.printStackTrace();
    }
}

public String open() {
    Shell parent = getParent();
    Display display = parent.getDisplay();

    shell = new Shell(parent, SWT.APPLICATION_MODAL | SWT.DIALOG_TRIM | SWT.RESIZE | SWT.MIN |
SWT.MAX);

    props.setLook(shell);
    setShellImage(shell, input);
    shell.setLayout(new FormLayout());
    shell.setText(Messages.getString("LucenePluginDialog.Shell.Title"));

    wScrolledComposite = new ScrolledComposite(shell, SWT.BORDER | SWT.V_SCROLL);
    FormData fdComposite = new FormData();
    fdComposite.top = new FormAttachment(shell, 3);
    fdComposite.left = new FormAttachment(0, 3);
    fdComposite.right = new FormAttachment(100, -3);
    fdComposite.bottom = new FormAttachment(100, -3);
    wScrolledComposite.setLayoutData(fdComposite);
    wComposite = new Composite(wScrolledComposite, SWT.NONE);
    props.setLook(wComposite);
    wComposite.setLayout(new FormLayout());

    changed = input.hasChanged();

    this.middle = props.getMiddlePct();

    //build ui components
    buildHeader(); //build header Labels and buttons
    buildFieldsRows(); //Fields from last step to index rows Labels and buttons
    addButtons(); //Add buttons
    addListeners(); //Add listeners

    setSize();// Set the shell size, based upon previous time...

```



```

wScrolledComposite.setMinSize(wComposite.computeSize(SWT.DEFAULT, SWT.DEFAULT));
wScrolledComposite.setExpandHorizontal(true);
wScrolledComposite.setExpandVertical(true);
wScrolledComposite.setContent(wComposite);
wScrolledComposite.getVerticalBar().setIncrement(10);/"faster" scrollbar

getData();
input.setChanged(changed);

shell.open();
while (!shell.isDisposed()) {
    if (!display.readAndDispatch()){
        display.sleep();
    }
}
return stepname;
}

private void addButtons() {
    wOK = new Button(wComposite, SWT.PUSH);
    wOK.setText(Messages.getString("System.Button.OK"));
    wCancel = new Button(wComposite, SWT.PUSH);
    wCancel.setText(Messages.getString("System.Button.Cancel"));

    BaseStepDialog.positionBottomButtons(wComposite,
        new Button[] { wOK, wCancel }, MARGIN, attach);
}

private void buildFieldsRows() {
    //if there's no previous step, ask the user to have another step.
    if(dataSetRowNames.length==0){
        Label wNoStep = new Label(wComposite, SWT.HORIZONTAL | SWT.CENTER);
        wNoStep.setText(Messages.getString("LucenePluginDialog.NoPrevStep.Label"));
        props.setLook(wNoStep);
        FormData fdNoStep = new FormData();
        fdNoStep.left = new FormAttachment(middle, -20);
        fdNoStep.top = new FormAttachment(attach, MARGIN+10);
        wNoStep.setLayoutData(fdNoStep);
        attach = wNoStep;
    }else{//if there's a previous step, mount labels.
        buildIndexLabels();
    }

    boolean getDefaultFields = false;
    if(dataSetRowNames.length!=indexFields.size()){ //check if previous fields have changed
        indexFields = new ArrayList<LucenePluginField>();
        getDefaultFields = true;
    }
    //add rows and set index names
    /*
    * TODO change LucenePluginDialogRows to tableview as is kettle default output presentation.

```

```

* Add Get fields button
*/
for(int i=0; i<dataSetRowNames.length; i++){
    rows.add(new LucenePluginDialogRow(i, wComposite, props, this, dataSetRowNames[i]));
    attach = rows.get(i).getWField();
    if(getDefaultFields){
        indexFields.add(new LucenePluginField());
        indexFields.get(i).setName(dataSetRowNames[i]);
    }
}
}

private void buildHeader() {
    // Stepname line
    wStepname = new Label(wComposite, SWT.RIGHT);
    wStepname.setText(Messages.getString("LucenePluginDialog.Stepname.Label")); //$NON-NLS-1$
    props.setLook(wStepname);
    fdStepname = new FormData();
    fdStepname.left = new FormAttachment(0, 0);
    fdStepname.right = new FormAttachment(middle, -MARGIN);
    fdStepname.top = new FormAttachment(0, MARGIN);
    wStepname.setLayoutData(fdStepname);
    wStepname = new Text(wComposite, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
    wStepname.setText(stepname);
    props.setLook(wStepname);
    fdStepname = new FormData();
    fdStepname.left = new FormAttachment(middle, 0);
    fdStepname.top = new FormAttachment(0, MARGIN);
    fdStepname.right = new FormAttachment(100, 0);
    wStepname.setLayoutData(fdStepname);

    // Path line
    wDirPath = new Button(wComposite, SWT.PUSH | SWT.CENTER);
    props.setLook(wDirPath);
    wDirPath.setText(Messages.getString("LucenePluginDialog.Path.Button"));
    fdbDirPath = new FormData();
    fdbDirPath.right = new FormAttachment(100, 0);
    fdbDirPath.top = new FormAttachment(wStepname, MARGIN);
    wDirPath.setLayoutData(fdbDirPath);

    wDirPath = new Label(wComposite, SWT.RIGHT);
    wDirPath.setText(Messages.getString("LucenePluginDialog.Path.Label"));
    props.setLook(wDirPath);
    fdDirPath = new FormData();
    fdDirPath.left = new FormAttachment(0, 0);
    fdDirPath.right = new FormAttachment(middle, -MARGIN);
    fdDirPath.top = new FormAttachment(wStepname, MARGIN);
    wDirPath.setLayoutData(fdDirPath);
    wDirPath = new Text(wComposite, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
    props.setLook(wDirPath);
    fdDirPath = new FormData();

```

```

fdDirPath.left = new FormAttachment(middle, 0);
fdDirPath.right = new FormAttachment(wbDirPath, -MARGIN);
fdDirPath.top = new FormAttachment(wStepname, MARGIN);
wDirPath.setLayoutData(fdDirPath);

// Build from scratch checkbox
wBuildFromScratch = new Label(wComposite, SWT.RIGHT);
wBuildFromScratch.setText(Messages.getString("LucenePluginDialog.BuildFromScratch.Label"));
props.setLook(wBuildFromScratch);
fdBuildFromScratch = new FormData();
fdBuildFromScratch.left = new FormAttachment(0, 0);
fdBuildFromScratch.right = new FormAttachment(middle, -MARGIN);
fdBuildFromScratch.top = new FormAttachment(wDirPath, MARGIN);
wBuildFromScratch.setLayoutData(fdBuildFromScratch);
wBuildFromScratch = new Button(wComposite, SWT.CHECK);
props.setLook(wBuildFromScratch);
wBuildFromScratch.setSelection(false);
fdBuildFromScratch = new FormData();
fdBuildFromScratch.left = new FormAttachment(middle, 0);
fdBuildFromScratch.right = new FormAttachment(100, 0);
fdBuildFromScratch.top = new FormAttachment(wDirPath, MARGIN);
wBuildFromScratch.setLayoutData(fdBuildFromScratch);

// Analyzer line
wAnalyzer = new Label(wComposite, SWT.RIGHT);
wAnalyzer.setText(Messages.getString("LucenePluginDialog.Analyzer.Label"));
props.setLook(wAnalyzer);
fdAnalyzer = new FormData();
fdAnalyzer.left = new FormAttachment(0, 0);
fdAnalyzer.right = new FormAttachment(middle, -MARGIN);
fdAnalyzer.top = new FormAttachment(wBuildFromScratch, MARGIN);
wAnalyzer.setLayoutData(fdAnalyzer);
wAnalyzer = new Combo(wComposite, SWT.DROP_DOWN | SWT.READ_ONLY);
props.setLook(wAnalyzer);
fdAnalyzer = new FormData();
fdAnalyzer.left = new FormAttachment(middle, 0);
fdAnalyzer.right = new FormAttachment(100, 0);
fdAnalyzer.top = new FormAttachment(wBuildFromScratch, MARGIN);
wAnalyzer.setLayoutData(fdAnalyzer);
ArrayList<String> analyzers = input.getAnalyzerNames();
for(String analyzer : analyzers){
    wAnalyzer.add(analyzer);
}
this.attach = wAnalyzer;
}

private void buildIndexLabels() {
//labels for index fields
wTableFieldName = new Label(wComposite, SWT.RIGHT);
wTableFieldName.setText(Messages.getString("LucenePluginDialog.TableFieldName.Label"));
props.setLook(wTableFieldName);

```

```

FormData fdLabels = new FormData();
fdLabels.left = new FormAttachment(0,0);
fdLabels.right = new FormAttachment(middle, -MARGIN);
fdLabels.top = new FormAttachment(attach, MARGIN+5);
wTableFieldName.setLayoutData(fdLabels);
//Index label
wIndex = new Label(wComposite, SWT.RIGHT);
wIndex.setText(Messages.getString("LucenePluginDialog.Index.Label"));
props.setLook(wIndex);
FormData fdIndex = new FormData();
fdIndex.right = new FormAttachment(100, -80);
fdIndex.top = new FormAttachment(attach, MARGIN+5);
wIndex.setLayoutData(fdIndex);
//store label
wStore = new Label(wComposite, SWT.RIGHT);
wStore.setText(Messages.getString("LucenePluginDialog.Store.Label"));
props.setLook(wStore);
FormData fdIStore = new FormData();
fdIStore.right = new FormAttachment(wIndex, -MARGIN);
fdIStore.top = new FormAttachment(attach, MARGIN+5);
wStore.setLayoutData(fdIStore);
//date label
wDate = new Label(wComposite, SWT.RIGHT);
wDate.setText(Messages.getString("LucenePluginDialog.Date.Label"));
props.setLook(wDate);
FormData fdIDate = new FormData();
fdIDate.right = new FormAttachment(wStore, -MARGIN);
fdIDate.top = new FormAttachment(attach, MARGIN+5);
wDate.setLayoutData(fdIDate);
//key label
wKey = new Label(wComposite, SWT.RIGHT);
wKey.setText(Messages.getString("LucenePluginDialog.Key.Label"));
props.setLook(wKey);
FormData fdIKey = new FormData();
fdIKey.right = new FormAttachment(wDate, -MARGIN);
fdIKey.top = new FormAttachment(attach, MARGIN+5);
wKey.setLayoutData(fdIKey);
//field name label
wIndexFieldName = new Label(wComposite, SWT.LEFT);
wIndexFieldName.setText(Messages.getString("LucenePluginDialog.IndexFieldName.Label"));
props.setLook(wIndexFieldName);
FormData fdLabels2 = new FormData();
fdLabels2.left = new FormAttachment(middle, 0);
fdLabels2.right = new FormAttachment(wKey, -MARGIN);
fdLabels2.top = new FormAttachment(attach, MARGIN+5);
wIndexFieldName.setLayoutData(fdLabels2);

attach = wTableFieldName;
}

private void addListeners() {

```

```

IsCancel = new Listener() {
    public void handleEvent(Event e) {
        cancel();
    }
};
IsOK = new Listener() {
    public void handleEvent(Event e) {
        ok();
    }
};
ModifyListener IsMod = new ModifyListener() {
    public void modifyText(ModifyEvent e) {
        input.setChanged();
    }
};
SelectionListener IsModButton = new SelectionListener() {
    @Override
    public void widgetDefaultSelected(SelectionEvent arg0) {
        input.setChanged();
    }
    @Override
    public void widgetSelected(SelectionEvent arg0) {
        input.setChanged();
    }
};
IsDef = new SelectionAdapter() {
    public void widgetDefaultSelected(SelectionEvent e) {
        ok();
    }
};

wCancel.addListener(SWT.Selection, IsCancel);
wOK.addListener(SWT.Selection, IsOK);

wStepname.addModifyListener(IsMod);
wDirPath.addModifyListener(IsMod);
wAnalyzer.addModifyListener(IsMod);

wStepname.addSelectionListener(IsDef);
wDirPath.addSelectionListener(IsDef);
for(int i=0; i<dataSetRowNames.length; i++){
    rows.get(i).getWField().addSelectionListener(IsDef);
}

wBuildFromScratch.addSelectionListener(IsModButton);

wDirPath.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {
        DirectoryDialog dialog = new DirectoryDialog(shell);
        if (wDirPath.getText() != null) {
            dialog.setFilterPath(transMeta.environmentSubstitute(wDirPath.getText()));

```

```

        }
        if (dialog.open() != null) {
            wDirPath.setText(dialog.getFilterPath());
        }
    }
});

// Detect X or ALT-F4 or something that kills this window...
shell.addShellListener(new ShellAdapter() {
    public void shellClosed(ShellEvent e) {
        cancel();
    }
});

}

// Read data from input
public void getData() {
    wStepname.selectAll();
    wDirPath.setText(input.getDirPath());
    wAnalyzer.setText(input.getStringAnalyzer());
    wBuildFromScratch.setSelection(input.isBuildFromScratch());
    for(int i=0; i<indexFields.size(); i++){
        rows.get(i).getWField().setText(indexFields.get(i).getName());
        rows.get(i).setKey(indexFields.get(i).isKey());
        rows.get(i).setDate(indexFields.get(i).isDate());
        rows.get(i).getWStore().setSelection(indexFields.get(i).getBooleanStore());
        rows.get(i).getWIndex().setText(indexFields.get(i).getIndex().toString());
    }
}

private void cancel() {
    stepname = null;
    input.setChanged(changed);
    dispose();
}

private void ok() {
    indexFields = new ArrayList<LucenePluginField>();
    for(int i=0; i<dataSetRowNames.length; i++){
        indexFields.add(new LucenePluginField());
        indexFields.get(i).setName(rows.get(i).getWField().getText());
        indexFields.get(i).setKey(rows.get(i).getWKey().getSelection());
        indexFields.get(i).setDate(rows.get(i).getWDate().getSelection());
        indexFields.get(i).setStore(rows.get(i).getWStore().getSelection());
        indexFields.get(i).setIndex(rows.get(i).getWIndex().getText());
    }
    if(!checkErrors()){
        stepname = wStepname.getText();
        input.setDirPath(wDirPath.getText());
        input.setAnalyzer(wAnalyzer.getText());
        input.setBuildFromScratch(wBuildFromScratch.getSelection());
    }
}

```

```

        input.setIndexFields(indexFields);
        input.setChanged();
        dispose();
    }
}

public boolean checkErrors(){
    if(!new File(wDirPath.getText()).exists()){
        MessageBox error = new MessageBox(shell, SWT.ICON_ERROR | SWT.OK);
        error.setMessage(Messages.getString("LucenePluginDialog.Error.PathNotExists"));
        error.open();
        return true;
    }
    for(int i=0; i<indexFields.size(); i++){
        for(int j=0; j<indexFields.size(); j++){
            if(rows.get(i).getWField().getText().equals(rows.get(j).getWField().getText()) && i!=j){
                MessageBox error = new MessageBox(shell, SWT.ICON_ERROR |
SWT.OK);

                error.setMessage(Messages.getString("LucenePluginDialog.Error.DuplicateFieldName"));
                error.open();
                return true;
            }
        }
        // if(rows.get(i).getWIndex().getText().equals(Field.Index.NO.toString()) &&
!rows.get(i).getWStore().getSelection()){
        //     MessageBox error = new MessageBox(shell, SWT.ICON_ERROR | SWT.OK);
        //     error.setMessage(Messages.getString("LucenePluginDialog.Error.NoStoreNoIndex"));
        //     error.open();
        //     return true;
        // }
    }
    return false;
}

public void disableAllKeysBut(int pos) {
    for(int i=0; i<indexFields.size(); i++){
        if (i!=pos) {
            rows.get(i).getWKey().setSelection(false);
            rows.get(i).getWKey().setEnabled(false);
        }
    }
}

public void enableAllKeys() {
    int date = getDate();
    for(int i=0; i<indexFields.size(); i++){
        if(i!=date){
            rows.get(i).getWKey().setEnabled(true);
        }
    }
}

```

```

    }

    public void disableAllDatesBut(int pos) {
        for(int i=0; i<indexFields.size(); i++){
            if (i!=pos) {
                rows.get(i).getWDate().setSelection(false);
                rows.get(i).getWDate().setEnabled(false);
            }
        }
    }

    public void enableAllDates() {
        int key = getKey();
        for(int i=0; i<indexFields.size(); i++){
            if(i!=key){
                rows.get(i).getWDate().setEnabled(true);
            }
        }
    }

    public int getKey(){
        for(int i=0; i<indexFields.size(); i++){
            if(rows.get(i).getWKey().getSelection()){
                return i;
            }
        }
        return -1;
    }

    public int getDate(){
        for(int i=0; i<indexFields.size(); i++){
            if(rows.get(i).getWDate().getSelection()){
                return i;
            }
        }
        return -1;
    }
}

```

LucenePluginDialogRow

```

package be.ibridge.kettle.lucene;

import org.apache.lucene.document.Field;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.FormAttachment;
import org.eclipse.swt.layout.FormData;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;

```



```

import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.pentaho.di.ui.core.PropsUI;

public class LucenePluginDialogRow {
    final private int pos;

    final private LucenePluginDialog dialog;
    final private Composite shell;
    final private PropsUI props;
    final private int checkboxDistance = 10;

    //last step field
    private String field;
    //field
    private Label wField;
    private Text wField;
    private FormData fdlField, fdField;
    //field key checkbox
    private Button wKey;
    private FormData fdKey;
    //field date checkbox
    private Button wDate;
    private FormData fdDate;
    //field Store checkbox
    private Button wStore;
    private FormData fdStore;
    //field index combo box
    private Combo wIndex;
    private FormData fdIndex;

    public LucenePluginDialogRow(final int pos, final Composite shell, final PropsUI props, final LucenePluginDialog
dialog, String field){
        this.pos = pos;
        this.props = props;
        this.field = field;
        this.dialog = dialog;
        this.shell = shell;

        this.createRow();
    }

    public Text getWField() {
        return wField;
    }

    public Button getWKey() {
        return wKey;
    }

    public void setKey(boolean key){

```

```

        wKey.setSelection(key);
        if(key){
            checkKey();
        }
    }

    public Button getWDate() {
        return wDate;
    }

    public void setDate(boolean date){
        wDate.setSelection(date);
        if(date){
            checkDate();
        }
    }

    public Button getWStore() {
        return wStore;
    }

    public Combo getWIndex() {
        return wIndex;
    }

    private void createRow(){
        //Create Index Text Labels
        wField = new Label(shell, SWT.RIGHT);
        wField.setText(field);
        props.setLook(wField);
        fdlField = new FormData();
        fdlField.left = new FormAttachment(0, 0);
        fdlField.right = new FormAttachment(dialog.middle, -LucenePluginDialog.MARGIN);
        fdlField.top = new FormAttachment(this.dialog.attach, LucenePluginDialog.MARGIN);
        wField.setLayoutData(fdlField);
        //Create index drop down lists
        wIndex = new Combo(shell, SWT.DROP_DOWN | SWT.READ_ONLY);
        props.setLook(wIndex);
        fdIndex = new FormData();
        fdIndex.top = new FormAttachment(this.dialog.attach, LucenePluginDialog.MARGIN);
        fdIndex.right = new FormAttachment(100, -LucenePluginDialog.MARGIN);
        wIndex.setLayoutData(fdIndex);
        wIndex.add(Field.Index.NO.toString());
        wIndex.add(Field.Index.TOKENIZED.toString());
        wIndex.add(Field.Index.UN_TOKENIZED.toString());
        //Create store checkboxes
        wStore = new Button(shell, SWT.CHECK);
        props.setLook(wStore);
        wStore.setSelection(true);
        fdStore = new FormData();
        fdStore.top = new FormAttachment(this.dialog.attach, LucenePluginDialog.MARGIN+2);
    }

```

```

fdStore.right = new FormAttachment(wIndex, -checkboxDistance, SWT.LEFT);
wStore.setLayoutData(fdStore);
//Create date checkboxes
wDate = new Button(shell, SWT.CHECK);
props.setLook(wDate);
wDate.setSelection(false);
fdDate = new FormData();
fdDate.top = new FormAttachment(this.dialog.attach, LucenePluginDialog.MARGIN+2);
fdDate.right = new FormAttachment(wStore, -checkboxDistance, SWT.LEFT);
wDate.setLayoutData(fdDate);
wDate.addSelectionListener(new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        if(((Button)e.getSource()).getSelection()){
            checkDate();
        }else{
            uncheckDate();
        }
    }
});
//Create key checkbox
wKey = new Button(shell, SWT.CHECK);
props.setLook(wKey);
wKey.setSelection(false);
fdKey = new FormData();
fdKey.top = new FormAttachment(this.dialog.attach, LucenePluginDialog.MARGIN+2);
fdKey.right = new FormAttachment(wDate, -checkboxDistance, SWT.LEFT);
wKey.setLayoutData(fdKey);
wKey.addSelectionListener(new SelectionAdapter(){
    public void widgetSelected(SelectionEvent e) {
        if(((Button)e.getSource()).getSelection()){
            checkKey();
        }else{
            uncheckKey();
        }
    }
});
//Create Index Field TextBox
wField = new Text(shell, SWT.SINGLE | SWT.LEFT | SWT.BORDER);
props.setLook(wField);
fdField = new FormData();
fdField.left = new FormAttachment(dialog.middle, 0);
fdField.right = new FormAttachment(wKey, -checkboxDistance, SWT.LEFT);
fdField.top = new FormAttachment(this.dialog.attach, LucenePluginDialog.MARGIN);
wField.setLayoutData(fdField);
}

protected void uncheckDate() {
    dialog.enableAllDates();
    if(dialog.getKey() == -1){
        wKey.setEnabled(true);
    }
}

```

```

        wField.setEnabled(true);
        wIndex.setEnabled(true);
        wStore.setEnabled(true);
    }

    protected void checkDate() {
        dialog.disableAllDatesBut(pos);
        wField.setText("date");
        wField.setEnabled(false);
        wKey.setEnabled(false);
        wStore.setSelection(true);
        wStore.setEnabled(false);
        wIndex.setText(Field.Index.NO.toString());
        wIndex.setEnabled(false);
    }

    protected void uncheckKey() {
        dialog.enableAllKeys();
        if(dialog.getDate()!=-1){
            wDate.setEnabled(true);
        }
        wField.setEnabled(true);
        wIndex.setEnabled(true);
        wStore.setEnabled(true);
    }

    protected void checkKey() {
        dialog.disableAllKeysBut(pos);
        wField.setText("key");
        wField.setEnabled(false);
        wDate.setEnabled(false);
        wStore.setSelection(true);
        wStore.setEnabled(false);
        wIndex.setText(Field.Index.UN_TOKENIZED.toString());
        wIndex.setEnabled(false);
    }
}

```

LucenePluginField

```

package be.ibridge.kettle.lucene;

import org.apache.lucene.document.Field;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;

public class LucenePluginField implements Cloneable {

    private String name;
    private Store store;
    private Index index;
    private boolean key;

```

```
private boolean date;

public LucenePluginField(){
    this.name = "";
    this.store = Field.Store.YES;
    this.index = Field.Index.TOKENIZED;
    this.key = false;
    this.date = false;
}

public LucenePluginField(String name, Store store, Index index, boolean key, boolean date) {
    this.name = name;
    this.store = store;
    this.index = index;
    this.key = key;
    this.date = date;
}

public Index getIndex() {
    return index;
}

public void setIndex(Index index) {
    this.index = index;
}

public void setIndex(String index){
    if(index.equals(Field.Index.UN_TOKENIZED.toString())){
        this.index = Field.Index.UN_TOKENIZED;
    }if(index.equals(Field.Index.TOKENIZED.toString())){
        this.index = Field.Index.TOKENIZED;
    }else if(index.equals(Field.Index.NO.toString())){
        this.index = Field.Index.NO;
    }
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public Store getStore() {
    return store;
}

public void setStore(String store) {
    if(store.equals(Field.Store.NO.toString())){
        this.store = Field.Store.NO;
    }
}
```

```

        }else{
            this.store = Field.Store.YES;
        }
    }

    public void setStore(boolean store){
        if(store){
            this.store = Field.Store.YES;
        }else{
            this.store = Field.Store.NO;
        }
    }

    public boolean getBooleanStore(){
        if(this.store == Field.Store.YES){
            return true;
        }else{
            return false;
        }
    }

    public boolean isKey() {
        return key;
    }

    public void setKey(boolean key) {
        this.key = key;
    }

    public boolean isDate() {
        return date;
    }

    public void setDate(boolean date) {
        this.date = date;
    }

    public Object clone() {
        try {
            Object retval = super.clone();
            return retval;
        } catch (CloneNotSupportedException e) {
            return null;
        }
    }
}

```

LucenePluginMeta

```
package be.ibridge.kettle.lucene;
```

```
import java.io.File;
```

```

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PropertyResourceBundle;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.WhitespaceAnalyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.analysis.SimpleAnalyzer;
import org.apache.lucene.analysis.KeywordAnalyzer;
import org.apache.lucene.analysis.StopAnalyzer;
import org.eclipse.swt.widgets.Shell;
import org.pentaho.di.core.CheckResult;
import org.pentaho.di.core.CheckResultInterface;
import org.pentaho.di.core.Const;
import org.pentaho.di.core.Counter;
import org.pentaho.di.core.database.DatabaseMeta;
import org.pentaho.di.core.exception.KettleDatabaseException;
import org.pentaho.di.core.exception.KettleException;
import org.pentaho.di.core.exception.KettleXMLException;
import org.pentaho.di.core.row.RowMetaInterface;
import org.pentaho.di.core.variables.VariableSpace;
import org.pentaho.di.core.xml.XMLHandler;
import org.pentaho.di.repository.Repository;
import org.pentaho.di.trans.Trans;
import org.pentaho.di.trans.TransMeta;
import org.pentaho.di.trans.step.BaseStepMeta;
import org.pentaho.di.trans.step.StepDataInterface;
import org.pentaho.di.trans.step.StepDialogInterface;
import org.pentaho.di.trans.step.StepInterface;
import org.pentaho.di.trans.step.StepMeta;
import org.pentaho.di.trans.step.StepMetaInterface;
import org.w3c.dom.Node;

public class LucenePluginMeta extends BaseStepMeta implements StepMetaInterface {

    private static Map<String, Analyzer> analyzers;
    static {
        analyzers = new HashMap<String, Analyzer>();
        FileInputStream is;
        try {
            is = new FileInputStream("plugins/steps/Lucene Plugin/plugin.properties");
            PropertyResourceBundle pluginP = new PropertyResourceBundle(is);

```

```

Enumeration<String> keys = pluginP.getKeys();
while (keys.hasMoreElements()) {
    final String key = keys.nextElement();
    String[] keyData = pluginP.getString(key).split("\\|");
    try {
        analyzers.put(key, addPath("plugins/steps/Lucene Plugin/"+keyData[0],
keyData[1]));
    } catch (InstantiationException e) {
    } catch (IllegalAccessException e) {
    } catch (ClassNotFoundException e) {
    }
}
} catch (IOException e) {
}
analyzers.put("WhiteSpaceAnalyzer", new WhitespaceAnalyzer());
analyzers.put("StandardAnalyzer", new StandardAnalyzer());
analyzers.put("SimpleAnalyzer", new SimpleAnalyzer());
analyzers.put("KeywordAnalyzer", new KeywordAnalyzer());
analyzers.put("StopAnalyzer", new StopAnalyzer());
}

private String dirPath;
private Analyzer analyzer;
private boolean buildFromScratch;
private ArrayList<LucenePluginField> indexFields;

public LucenePluginMeta() {
    super(); // allocate BaseStepInfo
}

public static Analyzer addPath(String jarPath, String className) throws MalformedURLException,
InstantiationException, IllegalAccessException, ClassNotFoundException{
    //System.load(jarPath);
    File f = new File(jarPath);
    URL u;
    u = f.toURI().toURL();
    ClassLoader cl = URLClassLoader.newInstance(new URL[] { u }, ClassLoader.getSystemClassLoader());
    return (Analyzer) cl.loadClass( className ).newInstance();
}

public String getDirPath() {
    return dirPath;
}

public void setDirPath(String dirPath) {
    this.dirPath = dirPath;
}

public Analyzer getAnalyzer() {
    return analyzer;
}

```



```
public String getStringAnalyzer(){
    return analyzer.getClass().getSimpleName();
}

public void setAnalyzer(Analyzer analyzer) {
    this.analyzer = analyzer;
}

public void setAnalyzer(String analyzer){
    if(analyzers.containsKey(analyzer)){
        this.analyzer = analyzers.get(analyzer);
    }
}

public ArrayList<String> getAnalyzerNames() {
    ArrayList<String> ret = new ArrayList<String>();
    for(String name : analyzers.keySet()){
        ret.add(name);
    }
    return ret;
}

public boolean isBuildFromScratch() {
    return buildFromScratch;
}

public void setBuildFromScratch(boolean buildFromScratch) {
    this.buildFromScratch = buildFromScratch;
}

public void setBuildFromScratch(String buildFromScratch){
    this.buildFromScratch = Boolean.parseBoolean(buildFromScratch);
}

public ArrayList<LucenePluginField> getIndexFields() {
    return indexFields;
}

public void setIndexFields(ArrayList<LucenePluginField> indexFields) {
    this.indexFields = indexFields;
}

public Integer getKey(){
    for(int i=0; i<indexFields.size(); i++){
        if(indexFields.get(i).isKey()){
            return i;
        }
    }
    return null;
}
```

```

public Integer getDate(){
    for(int i=0; i<indexFields.size(); i++){
        if(indexFields.get(i).isDate()){
            return i;
        }
    }
    return null;
}

public String getXML() {
    StringBuilder retval = new StringBuilder();

    retval.append(XMLHandler.addTagValue("dir_path", dirPath));
    retval.append(XMLHandler.addTagValue("analyzer", getStringAnalyzer()));
    retval.append(XMLHandler.addTagValue("build_from_scratch", buildFromScratch));
    retval.append("<fields>").append(Const.CR);
    for(LucenePluginField field : indexFields){
        retval.append("<field>").append(Const.CR);
        retval.append(XMLHandler.addTagValue("name", field.getName()));
        retval.append(XMLHandler.addTagValue("key", field.isKey()));
        retval.append(XMLHandler.addTagValue("date", field.isDate()));
        retval.append(XMLHandler.addTagValue("store", field.getStore().toString()));
        retval.append(XMLHandler.addTagValue("index", field.getIndex().toString()));
        retval.append("</field>").append(Const.CR);
    }
    retval.append("</fields>");
    System.out.println("getxml = " + retval.toString());

    return retval.toString();
}

public void getFields(RowMetaInterface r, String origin,
    RowMetaInterface[] info, StepMeta nextStep, VariableSpace space) {
    //no rows changed
}

public Object clone() {
    LucenePluginMeta retval = (LucenePluginMeta) super.clone();
    retval.indexFields = new ArrayList<LucenePluginField>();

    for (LucenePluginField field : indexFields) {
        retval.indexFields.add((LucenePluginField)field.clone());
    }

    return retval;
}

public void loadXML(Node stepnode, List<DatabaseMeta> databases,
    Map<String, Counter> counters) throws KettleXMLException {
    readData(stepnode);
}

```

```

    }

    private void readData(Node stepnode) throws KettleXMLException {
        try {
            dirPath = XMLHandler.getTagValue(stepnode, "dir_path");
            setAnalyzer(XMLHandler.getTagValue(stepnode, "analyzer"));
            buildFromScratch = "Y".equalsIgnoreCase(XMLHandler.getTagValue(stepnode,
"build_from_scratch"));

            Node fields = XMLHandler.getSubNode(stepnode, "fields");
            int nrfields = XMLHandler.countNodes(fields, "field");
            indexFields = new ArrayList<LucenePluginField>();
            for (int i = 0; i < nrfields; i++) {
                Node fnode = XMLHandler.getSubNodeByNr(fields, "field", i);
                indexFields.add(new LucenePluginField());
                indexFields.get(i).setName(XMLHandler.getTagValue(fnode, "name"));
                indexFields.get(i).setKey("Y".equalsIgnoreCase(XMLHandler.getTagValue(fnode,
"key")));
                indexFields.get(i).setDate("Y".equalsIgnoreCase(XMLHandler.getTagValue(fnode,
"date")));
                indexFields.get(i).setStore(XMLHandler.getTagValue(fnode, "store"));
                indexFields.get(i).setIndex(XMLHandler.getTagValue(fnode, "index"));
            }
        } catch (Exception e) {
            throw new KettleXMLException("Unable to load step info from XML", e);
        }
    }

    public void setDefault() {
        dirPath = "C:/Arquivos index test/index_kettle";
        analyzer = new StandardAnalyzer();
        buildFromScratch = true;
        indexFields = new ArrayList<LucenePluginField>();
    }

    public void readRep(Repository rep, long id_step,
        List<DatabaseMeta> databases, Map<String, Counter> counters)
        throws KettleException {
        try {
            dirPath = rep.getStepAttributeString(id_step, "dir_path");
            setAnalyzer(rep.getStepAttributeString(id_step, "analyzer"));
            buildFromScratch = rep.getStepAttributeBoolean(id_step, "build_from_scratch");

            int nrfields = rep.countNrStepAttributes(id_step, "field");
            for (int i = 0; i < nrfields; i++) {
                indexFields.add(new LucenePluginField());
                indexFields.get(i).setName(rep.getStepAttributeString(id_step, i, "name"));
                indexFields.get(i).setKey(rep.getStepAttributeBoolean(id_step, "key"));
                indexFields.get(i).setDate(rep.getStepAttributeBoolean(id_step, "date"));
                indexFields.get(i).setStore(rep.getStepAttributeString(id_step, i, "store"));
                indexFields.get(i).setIndex(rep.getStepAttributeString(id_step, i, "index"));
            }
        }
    }

```

```

    }
} catch (KettleDatabaseException dbe) {
    throw new KettleException("error reading step with id_step="
        + id_step + " from the repository", dbe);
} catch (Exception e) {
    throw new KettleException(
        "Unexpected error reading step with id_step=" + id_step
        + " from the repository", e);
}
}

public void saveRep(Repository rep, long id_transformation, long id_step)
throws KettleException {
    try {
        rep.saveStepAttribute(id_transformation, id_step, "dir_path", dirPath);
        rep.saveStepAttribute(id_transformation, id_step, "analyzer", getStringAnalyzer());
        rep.saveStepAttribute(id_transformation, id_step, "build_from_scratch", buildFromScratch);

        for (int i=0; i<indexFields.size(); i++) {
            rep.saveStepAttribute(id_transformation, id_step, i, "name",
indexFields.get(i).getName());
            rep.saveStepAttribute(id_transformation, id_step, i, "key", indexFields.get(i).isKey());
            rep.saveStepAttribute(id_transformation, id_step, i, "date", indexFields.get(i).isDate());
            rep.saveStepAttribute(id_transformation, id_step, i, "store",
indexFields.get(i).getStore().toString());
            rep.saveStepAttribute(id_transformation, id_step, i, "index",
indexFields.get(i).getIndex().toString());
        }
    } catch (KettleDatabaseException dbe) {
        throw new KettleException(
            "Unable to save step information to the repository, id_step="
            + id_step, dbe);
    }
}

public void check(List<CheckResultInterface> remarks, TransMeta transMeta,
    StepMeta stepMeta, RowMetaInterface prev, String input[],
    String output[], RowMetaInterface info) {
    CheckResult cr;
    if (prev == null || prev.size() == 0) {
        cr = new CheckResult(CheckResult.TYPE_RESULT_WARNING,
            "Not receiving any fields from previous steps!", stepMeta);
        remarks.add(cr);
    } else {
        cr = new CheckResult(CheckResult.TYPE_RESULT_OK,
            "Step is connected to previous one, receiving "
            + prev.size() + " fields", stepMeta);
        remarks.add(cr);
    }

    // See if we have input streams leading to this step!

```

```

        if (input.length > 0) {
            cr = new CheckResult(CheckResult.TYPE_RESULT_OK,
                "Step is receiving info from other steps.", stepMeta);
            remarks.add(cr);
        } else {
            cr = new CheckResult(CheckResult.TYPE_RESULT_ERROR,
                "No input received from other steps!", stepMeta);
            remarks.add(cr);
        }
    }

    public StepDialogInterface getDialog(Shell shell, StepMetaInterface meta,
        TransMeta transMeta, String name) {
        return new LucenePluginDialog(shell, meta, transMeta, name);
    }

    public StepInterface getStep(StepMeta stepMeta,
        StepDataInterface stepDataInterface, int cnr, TransMeta transMeta,
        Trans disp) {
        return new LucenePlugin(stepMeta, stepDataInterface, cnr, transMeta,
            disp);
    }

    public StepDataInterface getStepData() {
        return new LucenePluginData();
    }
}

```

LuceneUpdate

```

package be.ibridge.kettle.lucene;

import java.io.IOException;

import org.apache.lucene.analysis.StopAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.Field.Index;
import org.apache.lucene.document.Field.Store;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.store.LockObtainFailedException;

public class LuceneUpdate {

    /*
     * Alpha - OK
    */
}

```

```

* Bravo - OK
* Charlie
* Delta - OK
* Eco - OK
*/

static final String projectName = "Charlie";

static final String PROJETO_FIELD = "Projeto";

public static void main(String[] args) throws CorruptIndexException, LockObtainFailedException, IOException,
ParseException {

    IndexSearcher searcher = new IndexSearcher("D:/andre/Documents/Faculdade/TCC/ETL/Kettle
tests/index_charlie");

    IndexWriter indexWriter = new IndexWriter("D:/andre/Documents/Faculdade/TCC/ETL/Kettle
tests/index_charlie 2", new StopAnalyzer(), true);

    QueryParser qp = new QueryParser(PROJETO_FIELD, new StopAnalyzer());
    Query query = qp.parse("Alpha");
    Hits hits = searcher.search(query);

    for (int i = 0; i < hits.length(); i++) {
        Document doc = hits.doc(i);
        doc.removeField(PROJETO_FIELD);
        doc.add(new Field(PROJETO_FIELD, projectName, Store.YES, Index.TOKENIZED));
        indexWriter.addDocument(doc);
        System.out.println("updated document. " + i);
    }

    System.out.println("optimizing.");
    indexWriter.optimize();
    indexWriter.close();
    System.out.println("writer closed.");
}
}

```

Messages

```

package be.ibridge.kettle.lucene;

import org.pentaho.di.i18n.BaseMessages;

public class Messages
{
    public static final String packageName = Messages.class.getPackage().getName();

    public static String getString(String key)
    {

```

```

        return BaseMessages.getString(packageName, key);
    }

    public static String getString(String key, String param1)
    {
        return BaseMessages.getString(packageName, key, param1);
    }

    public static String getString(String key, String param1, String param2)
    {
        return BaseMessages.getString(packageName, key, param1, param2);
    }

    public static String getString(String key, String param1, String param2, String param3)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4, String param5)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4, param5);
    }

    public static String getString(String key, String param1, String param2, String param3, String param4, String param5,
String param6)
    {
        return BaseMessages.getString(packageName, key, param1, param2, param3, param4, param5, param6);
    }
}

```

Parameter

```

package be.ibridge.kettle.lucene;

import java.io.File;
import java.io.Serializable;
import java.sql.Timestamp;

/**
 * A Parameter class keeps the last indexing date in order to improve
 * incremental indexing process.
 */
public class Parameter implements Serializable {

    private static final long serialVersionUID = 3831811521464536531L;

    public static enum STATUS {FREE, BUSY};

```

```

private Timestamp lastIndexingDate = null;

private String controlFile;

private STATUS status = STATUS.FREE;

private Parameter() {
}

public static Parameter create(final String controlFile) throws Exception {
    Parameter parameter = null;
    File file = new File(controlFile);
    if (file.exists()) {
        parameter = (Parameter)DataPersistence.retrieve(controlFile);
    } else {
        parameter = new Parameter();
    }
    parameter.controlFile = controlFile;
    if (parameter.status == null)
        parameter.setStatus(Parameter.STATUS.FREE);
    return parameter;
}

/**
 * Gets the last indexing date.
 * @return Returns the last indexing date.
 */
public Timestamp getLastIndexingDate() {
    return lastIndexingDate;
}

/**
 * Sets the last indexing date.
 * @param lastIndexingDate The last indexing date to set.
 */
public void setLastIndexingDate(final Timestamp lastIndexingDate) {
    this.lastIndexingDate = lastIndexingDate;
}

public void save() throws Exception {
    DataPersistence.store(this.controlFile, this);
}

public void printProperties() {
    if (this.getLastIndexingDate() != null) {
        System.out.println("\nLast Operation Date: " + this.getLastIndexingDate().toString());
    }
}

public boolean isFirstOperation() {

```



```
    if (this.getLastIndexingDate() != null) {
        return false;
    } else {
        return true;
    }
}

public void updateLastOperationDate(final Timestamp dteSearch)
throws Exception {
    if (dteSearch != null) {
        this.setLastIndexingDate(dteSearch);
        this.save();
    }
}

public STATUS getStatus() {
    return status;
}

public void setStatus(STATUS status) {
    this.status = status;
}
}
```

APÊNDICE 5 - UpdateFatoFrequencia.java

UpdateFatoFrequencia.java

```
package input;

import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.apache.lucene.analysis.StopAnalyzer;
import org.apache.lucene.index.CorruptIndexException;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.Searcher;

public class UpdateFatoFrequencia {

    private static String indexFiles = "D:/andre/Documents/Faculdade/TCC/ETL/Kettle tests/index_alpha";

    private static String[] wordList;
    static {
        File file = new File(
            "D:/andre/Documents/Faculdade/TCC/ETL/Kettle tests/selectedWordList.txt");
        FileInputStream fis = null;
        BufferedInputStream bis = null;
        DataInputStream dis = null;
        try {
            fis = new FileInputStream(file);
            bis = new BufferedInputStream(fis);
            dis = new DataInputStream(bis);

            wordList = new String[15800];
            int count = 0;
            ;
            while (dis.available() != 0) {
```

```

        wordList[count] = dis.readLine();
        count++;
    }
    fis.close();
    bis.close();
    dis.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}

public static void main(String[] args) throws CorruptIndexException, IOException, ParseException,
ClassNotFoundException, SQLException {
    Searcher searcher = new IndexSearcher(indexFiles);

    // doing the query
    Query query;
    Hits hits;
    QueryParser qp = new QueryParser("content", new StopAnalyzer());

    String ano;
    String mes;

    int wordCount = 0;

    for (String word : wordList) {
        if (word != null) {
            wordCount++;
            System.out.println("wordCount = " + wordCount + " - word = "
                + word);

            query = qp.parse(word);

            hits = searcher.search(query);

            if (hits.length() > 0) {
                // for each document found, send several outputs
                for (int i = 0; i < hits.length(); i++) {
                    mes = hits.doc(i).get("MES");
                    ano = hits.doc(i).get("ANO");

                    updateLine(mes, ano, word);
                }
            }
        }
    }

    close(searcher);
}

```

```

private static void updateLine(String mes, String ano, String word)
    throws ClassNotFoundException, SQLException {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost/tcc", "root", "admin");
    Statement statement = con.createStatement();

    ResultSet tempoRS = statement.executeQuery("SELECT ID_TEMPO FROM dimensao_tempo d WHERE
ANO = " + ano + " AND MES = " + mes + ";");
    int idTempo;
    if (tempoRS.next()) {
        idTempo = tempoRS.getInt(1);
    }else { return; }

    ResultSet entidadeRS = statement.executeQuery("SELECT ID_ENTIDADE FROM dimensao_entidade d
where descricao = " + word + ";");
    int idEntidade;
    if (entidadeRS.next()) {
        idEntidade = entidadeRS.getInt(1);
    }else { return; }

    ResultSet frequenciaRS = statement.executeQuery("SELECT FREQUENCIA FROM fato_frequencia f
WHERE id_tempo = " + idTempo + " AND id_entidade = " + idEntidade + ";");
    int frequencia;
    if (frequenciaRS.next()) {
        frequencia = frequenciaRS.getInt(1);
    }else { return; }

    statement.executeUpdate("UPDATE fato_frequencia SET FREQUENCIA = " + (frequencia + 1) + " WHERE
id_tempo = " + idTempo + " AND id_entidade = " + idEntidade + ";");

    statement.close();
    con.close();
}

private static void close(Searcher searcher2) {
    try {
        searcher2.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

```