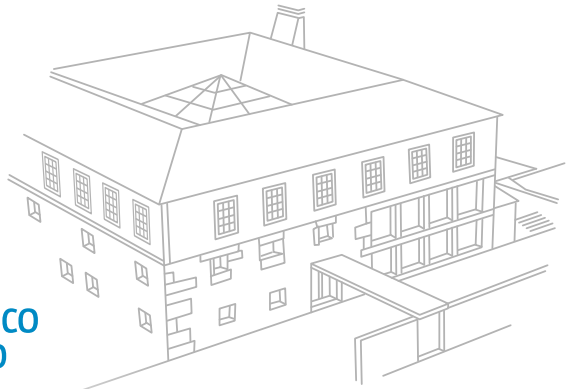


ESTGF | POLITÉCNICO DO PORTO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Implementação de Operações de Álgebra Relacional em JAVA sobre dados em XML

DESIGNAÇÃO DO MESTRADO
Mestrado em Engenharia Informática

AUTOR
Bruno Moisés Teixeira de Oliveira

ORIENTADOR(ES)
Orlando Manuel Oliveira Belo
Vasco Nuno Caio dos Santos

ANO
2011

"Because the people who are crazy enough to think they can change the world, are the ones who do"

Steve Jobs

À minha família, namorada e amigos

Agradecimentos

Gostaria de agradecer a todos aqueles que me apoiaram e contribuíram para a conclusão deste Mestrado, e em particular:

- Ao professor Vasco Santos pela orientação, acompanhamento, amizade, incentivo e permanente disponibilidade demonstrada durante todo o projecto.
- Ao Professor Doutor Orlando Belo pela orientação e disponibilidade demonstrada.
- Aos meus colegas de Mestrado: Rui Silva, Telmo Matos, Fábio Maia e Joaquim Meireles que através do companheirismo, entreajuda e amizade tornaram mais fácil a conclusão desta tese.
- À Escola Superior de Tecnologia e Gestão de Felgueiras – ESTGF, em particular à Professora Doutora Dorabela Gamboa pelas excelentes condições oferecidas e pelo apoio concedido, necessários à conclusão desta dissertação.
- À minha família e namorada, pelo permanente estímulo e natural apoio demonstrado ao longo da realização desta dissertação.

A todos o meu sincero obrigado.

Resumo

Um *Sistema de Data Warehousing* (SDW) armazena dados de uma forma integrada e consistente, o que o torna num repositório de dados ideal para o apoio aos vários tipos de decisão existentes numa organização. No entanto, para manter este repositório devidamente actualizado é necessário aceder a um conjunto variado de sistemas fontes, transformar a informação que deles é extraída, e de acordo com os requisitos do negócio e caraterísticas do modelo de armazenamento de dados de um SDW, proceder à sua adequada alimentação. Estas tarefas, geralmente designadas por Extracção, Transformação e Alimentação, são normalmente complexas e requerem uma grande quantidade de recursos computacionais operando numa janela temporal limitada. O objectivo desta dissertação passa por apresentar uma abordagem não convencional para a execução destas tarefas, em ambientes de execução variados. É apresentada uma proposta de modelação lógica de um processo de ETL baseado em Álgebra Relacional, de modo a implementar um modelo independentemente da plataforma de execução, permitindo o seu posterior mapeamento em primitivas de programação, utilizando a notação *Business Process Model and Notation* (BPMN) como base de modelação gráfica. É apresentado um exemplo de uma infraestrutura que permite o aproveitamento do poder computacional existente numa organização, através da utilização de uma GRID computacional, tendo como suporte o standard de representação de dados presente na linguagem XML e a heterogeneidade de execução da linguagem Java.

Palavras-Chave: Sistemas de Data Warehousing, Processos ETL, Álgebra Relacional, Java, XML, BPMN.

Abstract

A *Data Warehousing System* (DWS) stores data in an integrated and consistent way, making it an ideal data repository to support various types of decisions that exist in an organization. However, to keep this repository updated properly it is necessary to access a variety of source systems, transform the data that is extracted from them according to business requirements and characteristics of the data storage model of the DWS. These tasks, commonly known as Extraction, Transformation and Loading, are typically complex and require a large amount of computer resources in a limited time windows to properly run the ETL process. The purpose of this thesis is to present an unconventional approach to do these tasks in a heterogeneous environment of execution. It presents a proposal for a logic ETL process modelling, based on Relational Algebra in order to implement a model of non-dependent execution platform, allowing their subsequent programming mapping, using the graphical modelling notation *Business Process Model and Notation* (BPMN). An example of an infrastructure is presented that allows the use of the unused computational resources in an organization, through the use of a GRID computing concept, supported by the standard representation of data present in XML and heterogeneity of the Java language.

Keywords: Data Warehousing System, ETL processes, Relational Algebra, Java, XML, BPMN.

Índice

Índice de figuras

Índice de tabelas

Abreviaturas

API	Application Programming Interface
AR	Álgebra Relacional
BI	Business Intelligence
BPMN	Business Process Modelling Notation
DOM	Document Object Model
ETL	Extract, Transform e Load
I/O	Input/Output
JAXP	The Java API for XML Processing
LMD	Linguagem de Manipulação de Dados
MD	Modelação Dimensional
OJXQI	Oracle Java XQuery API
SAX	Simple API for XML
SDW	Sistema de Data Warehousing
SQL	Structured Query Language
StAX	The Streaming API for XML
UML	Unified Modelling Language
XML	Extensible Markup Language
XOM	XML Object Model
XPDL	XML Process Definition Language

XSD	XML Schema
WfMC	Workflow Management Coalition

Capítulo 1

Introdução

1.1 Contextualização

Actualmente vivemos numa era em que a informação é o bem mais precioso para as organizações, num mercado cada vez mais competitivo e cuja luta pela conquista de quota de mercado tem beneficiado as organizações empreendedoras, que apostam em sistemas de informação evoluídos, de modo a optimizar processos de negócio, ajustando os seus produtos às exigências e necessidades do mercado, permitindo a redução de despesa e consequentemente o aumento de lucros.

No mercado moderno, a automação, evolução e disponibilidade dos sistemas informáticos em fornecer informação tem sido um factor crucial na evolução das organizações, nas quais cada vez mais existe um processamento de grandes quantidades de dados que aumentam progressivamente ao longo do tempo, com vista a cobrir as várias áreas de negócio e ambiente que rodeia uma organização. Com o passar dos anos observou-se que pelo facto de uma organização possuir grandes quantidades de dados por si só não chega. É necessário que estes façam sentido e estejam devidamente enquadrados e alinhados com os objectivos da organização, de modo a tornarem-se em informação útil para os gestores que diariamente tomam decisões que afectam a organização e por consequência a vida de muitas pessoas.

Foi neste enquadramento que surgiu o conceito de *Business Intelligence* (BI), um termo que engloba os processos de obtenção, armazenamento e análise e partilha de dados, com o objectivo de utilizar a informação obtida para facilitar o processo de tomada de decisão dentro da própria organização.

O principal objectivo de um sistema de BI é o apoio a vários tipos de decisão dentro de uma organização, seja na tomada de decisões que afectam a organização em períodos de tempo pequenos (decisões operacionais), decisões que afectam o negócio por períodos de tempo longos (decisões tácticas), e decisões que alteram a estratégia da empresa no mercado onde esta opera (decisões estratégicas). Um sistema de BI terá de suportar vários tipos de decisão com vista a satisfazer os diferentes requisitos a nível de informação que as organizações possuem. Uma decisão operacional irá, por exemplo, necessitar de uma análise relativamente a *stocks*, fornecedores e prazos de entrega, uma decisão táctica tem em vista informação relativa a tendência de vendas, estudo de mercado, modelos, clima, etc. Uma decisão estratégica está relacionada com o aparecimento de áreas de negócio que possam revelar mais potencial ou até numa mudança do âmbito de negócio, suportada pela informação analisada.

Neste contexto é fundamental mencionar uma tecnologia crucial em todo este processo, os *Sistemas de Data Warehousing* (SDW). Estes sistemas permitem armazenar grandes quantidades de informação sobre determinada arquitectura com vista à análise avançada dos dados e com o objectivo de servir os requisitos de vários tipos de utilizadores, não só internos mas também externos à organização, servindo de base para ferramentas de análise de dados.

Segundo Inmon [1], um *Data Warehouse* é um tipo de base de dados que gere uma grande quantidade de dados, que deixam de estar orientados aos maiores "sujeitos" ou elementos de maior impacto numa empresa, ou seja, deixam de estar orientados à gestão de uma determinada área (gestão de stocks por exemplo) para estarem orientados a clientes, produtos e vendas, reflectindo uma maior necessidade de armazenamento de informação para apoio à decisão.

A integração de dados provenientes de várias fontes, muitas vezes incompatíveis entre si, permitem a obtenção de dados consistentes e históricos, possibilitando através de uma análise mais profunda e minuciosa, a redução de recursos desnecessários e optimização de processos que originam uma maior produtividade global. A integração de dados permite ainda disponibilizar uma visão unificada aos utilizadores, uma vez que os dados de origem são muitas vezes inconsistentes, derivado dos diferentes formatos de armazenamento que englobam vários problemas como perdas de integridade, dados com diferentes formatos e redundância de dados.

Um dos componentes mais importantes de um SDW é o Gestor de ETL (*Extract*, *Transform* e *Load*), que se encontra associado às tarefas de extracção, transformação e limpeza dos dados provenientes de sistemas fonte que são injectados no SDW, onde são aplicadas operações de transformação que têm como objectivo analisar e assegurar a consistência dos dados, criar índices e vistas sobre as tabelas existentes e transferir os dados que se encontram em armazenamento temporário para o SDW através da área de retenção.

Em termos computacionais, e com a quantidade de dados processados, a exigência de computação é elevada, sendo comum em muitas organizações o aproveitamento de intervalos de tempo de menor carga computacional (tipicamente à noite) para a realização de processos de ETL. Tendo em consideração que a rapidez na obtenção de informação é um factor de impacto na qualidade de uma decisão, então quanto melhor a eficiência e rapidez de conclusão na análise dos dados, maior será a probabilidade de melhorar a qualidade da decisão e consequentemente obter vantagens competitivas. É, por isso, natural que a optimização de processos de ETL seja um dos pontos de maior estudo por parte da comunidade científica [2-5].

1.2 Motivação e objectivos

Uma das maiores problemáticas no desenho de um processo de ETL prende-se com o crescente aumento de volume de dados e consequentemente da complexidade de tratamento dos mesmos, levando a que a escolha da infraestrutura de suporte seja um elemento fundamental, de modo a acompanhar o natural crescimento do SDW. Neste sentido, a utilização de recursos computacionais já existentes nas organizações permitirá reduzir custos de *hardware* e obter um acréscimo significativo do poder computacional que se encontrava inacessível, uma vez que grande parte destes recursos se encontra subaproveitada ou mesmo inactiva durante certos períodos. Naturalmente que o aproveitamento de recursos numa organização acarreta problemas de dependência de plataforma, devido ao facto de estarmos perante uma heterogeneidade de arquitecturas e sistemas operativos, o que implica a utilização de tecnologias multiplataforma que se integrem correctamente em ambientes com essas características. Tendo em conta que a linguagem Java é adequada para a utilização em ambientes multiplataforma e que o suporte de armazenamento de dados XML cada vez mais se afirma como uma alternativa viável ao armazenamento de informação independente da plataforma, é possível criar um conjunto de aplicativos independentes da arquitectura de execução para a implementação e gestão de um ambiente de computação em GRID.

Neste sentido, uma abordagem descentralizada de processamento de dados que satisfaça os requisitos e condicionantes de um sistema ETL, utilizando como suporte a infraestrutura já existente na organização e a sua execução através de tarefas distribuídas suportadas por um ambiente GRID, parece ser uma solução exequível com soluções vantajosas tanto a nível de redução de processamento de dados como de redução de custos de *hardware*.

Outra questão essencial na implementação de um processo de ETL prende-se com a sua modelação, que, por norma, é modelado recorrendo a ferramentas proprietárias ou, então, utilizando uma qualquer outra forma muito *ad-hoc*. Apesar de poderosas, estas ferramentas utilizam uma notação específica e até mesmo metodologias diferenciadas na modelação do processo ETL. Esta diversidade gera naturalmente complicações para o utilizador. Primeiro, sempre o que o utilizador alternar entre aplicações terá de completar o processo de aprendizagem. Segundo, se for necessário migrar o

sistema, os custos serão mais elevados. Tal situação verifica-se essencialmente devido à utilização de metodologias proprietárias, originando incompatibilidade entre aplicações, o que as tornam de difícil manutenção, optimização e compreensão. Por isso, a selecção da ferramenta de ETL é uma questão que não pode ser desconsiderada, pois uma vez seleccionada, a sua migração para outra plataforma torna-se bastante complexa. Alguns autores têm sugerido ao longo dos anos diferentes notações [2, 6, 7], sem no entanto conseguirem fazer prevalecer a sua notação como *standard*, continuando a existir uma lacuna tremenda desde a modelação até ao mapeamento de tais processos em primitivas de programação.

De forma a reduzir tal fosso, e com base no trabalho apresentado, foi proposto em [8] um modelo de especificação de processos de ETL com forte suporte em Álgebra Relacional (AR), dada a sua forte fundamentação no domínio das bases de dados, permitindo reduzir a distância entre a modelação conceptual e a implementação física. As transformações são simplificadas até às suas primitivas mais básicas, ou seja operações de AR, onde através de um grafo de transformações são estabelecidas precedências e dependências, numa modelação independente de plataforma, visando o mapeamento em primitivas de programação, permitindo uma implementação mais acessível.

Com base nos pressupostos referidos anteriormente, definiu-se como objectivos principais para esta dissertação os seguintes pontos:

- Analisar as diversas metodologias de modelação de processos de ETL, identificando as suas vantagens e desvantagens.
- Definir um modelo baseado em Álgebra Relacional de forma a elaborar uma metodologia de modelação de processos de ETL que facilite a implementação física.
- Desenvolvimento de aplicações para transformação de dados, equivalentes às operações de Álgebra Relacional, com vista à sua execução em ambiente distribuído.
- Implementação de um protótipo de representação de um processo de ETL desde a modelação conceptual até à sua implementação física.

Para a concretização de tais objectivos foi necessário estudar as abordagens existentes, recorrendo às diferentes metodologias de modelação de processos de ETL e tendo em consideração as forças e fraquezas de cada modelo definir uma abordagem baseada e fundamentada em Álgebra Relacional.

Uma vez definido o projecto e apresentados os casos de estudo elaborados, é realizada uma avaliação crítica ao trabalho realizado, enumerando os aspectos positivos e negativos. Por fim é apresentada a perspectiva de evolução de forma a complementar o trabalho apresentado.

1.3 Organização do relatório

Além do presente capítulo, esta dissertação íntegra um conjunto de mais quatro capítulos, organizados da seguinte forma:

- **Capítulo 2 - Sistemas de Especificação de ETL.**

  O principal objectivo deste capítulo passa por descrever o ciclo de vida de um processo de ETL, discutindo as várias propostas de modelação lógica e alternativas de especificação de um processo de ETL.

- **Capítulo 3 - Utilização de Álgebra Relacional na Modelação de um Sistema de ETL.**

  Uma vez analisada e estudada a modelação de um processo de ETL, no decorrer deste capítulo é apresentada a Álgebra Relacional, as suas origens, utilização e as suas principais operações, discutindo vantagens e desvantagens.

- **Capítulo 4 - Implementação Física de Tarefas de ETL.**

  Este capítulo tem como principal objectivo apresentar as várias operações alvo desta dissertação, descrevendo o processo de implementação de cada uma delas, assim como da tecnologia utilizada e os sistemas de dados utilizados. É também foco deste capítulo a apresentação e discussão de um exemplo, desde a modulação de um processo de ETL até à sua execução, descrevendo o ambiente computacional para um SDW no qual o processo de ETL produzido possa ser utilizado.

- **Capítulo 5 - Conclusões e trabalho futuro.**

  Neste capítulo é realizada uma análise crítica a todo o trabalho realizado, apresentando vantagens, desvantagens, dificuldades e curiosidades acerca do trabalho realizado. São também discutidos alguns dos aspectos menos positivos que surgiram ao longo da realização deste trabalho de dissertação, sendo apresentadas algumas eventuais soluções para atenuar tais aspectos. Por fim são apresentadas algumas linhas orientadoras para trabalho futuro.

Capítulo 2

Sistemas para a especificação de ETL

2.1 Introdução

O desenho da base de dados de um SDW é uma tarefa bastante complexa que envolve a definição dos requisitos de negócio mais importantes, os dados que devem ser considerados, assim como a infraestrutura necessária para suportar o SDW. No que diz respeito a metodologias de desenvolvimento de um SDW, é fundamental considerar a abordagem de Inmon [9] e Kimball [10] que têm como principal objectivo a criação de uma estrutura capaz de suportar toda a informação necessária de uma organização.

A abordagem de Inmon passa por criar um modelo de dados para toda a informação existente numa organização e utilizá-la para a implementação num SDW, mais concretamente na alimentação das bases de dados de departamentos (usualmente conhecidas por *data marts*), onde existem diferenças em termos de requisitos de negócio mediante o departamento. Inmon utiliza metodologias tradicionais de base de dados recorrendo a digramas ER que contêm tabelas normalizadas na terceira forma normal.

A abordagem de Kimball começa com a identificação dos requisitos da informação associando-os a processos de negócio, resultando na criação de um documento chamado *Data Warehouse Bus Matrix*. Este documento lista todos os processos chave de uma organização com a indicação de como estes processos devem ser analisados, permitindo relacionar os requisitos de estrutura de informação com grupos de utilizadores específicos, por forma a alimentar os diversos data marts da organização de modo a proceder à sua integração no SDW. Kimball utiliza uma técnica chamada de modelação dimensional (MD) para construir um modelo dimensional frequentemente desnormalizado.

A MD tem como objectivo colocar os dados num formato *standard*, de uma forma intuitiva e que permita bons desempenhos em operações de acesso a dados [11]. Cada Modelo Dimensional é composto por uma tabela com uma chave primária composta, chamada de tabela de factos, e por um conjunto de "pequenas" tabelas periféricas, chamadas de tabelas de dimensão. Cada tabela de dimensão tem uma chave primária simples que corresponde a um dos componentes da chave primária composta existente na tabela de factos. Isto é, a chave primária da tabela de factos é construída através das chaves estrangeiras provenientes das tabelas de dimensão. Esta organização baseada em estrela é chamada de *star schema* ou *star join*.

Outra característica fundamental de um Modelo Dimensional é que todas as chaves naturais são substituídas por chaves substitutas (*surrogate keys*), implicando que cada junção entre as tabelas de facto e as tabelas de dimensão é baseada em chaves substitutas. O uso destas chaves no Modelo Dimensional permite que os dados mantenham a sua independência relativamente aos dados utilizados e produzidos nos sistemas fonte que servem de alimentação ao SDW para além de permitir um aumento de performance essencialmente devido ao tipo de dados compacto (tipicamente do tipo inteiro) utilizado na definição das chaves substitutas, diminuindo os tempos de consulta à base de dados.

Ao longo dos anos têm surgido diversos trabalhos fazendo análises comparativas entre a metodologia apresentada por Inmon e a metodologia de Kimball. Alguns desses estudos apresentam vantagens e desvantagens da implementação das metodologias, focando que a escolha para a implementação de um SDW apoia-se essencialmente nas características de uma organização [12]. Embora ambas as metodologias tendam aproximar-se numa fase madura da sua implementação, a metodologia de Kimball tende a obter mais rapidamente um retorno financeiro, permitindo uma evolução incremental com custos financeiros e humanos inferiores comparativamente à metodologia defendida por Inmon [13]. Com base nestas conclusões, a metodologia de Kimball foi a escolhida para a base do trabalho apresentado.

2.2 Ciclo de vida do processo de ETL

O processo de ETL (*Extract-Transform-Load*) é um componente fundamental para a alimentação de um SDW. Um processo de ETL devidamente construído extrai os dados de sistemas fonte, força a implementação da qualidade e consistência de *standards*, estrutura os dados provenientes de várias fontes para que possam ser utilizados como um todo e finalmente entrega esses mesmos dados num formato que um SDW aceite. O desenvolvimento de um processo de ETL é crítico para a implementação de um qualquer SDW, sendo uma linha ténue entre o sucesso e o fracasso na sua implementação. Segundo Kimball [14], o processo de ETL consume 70% dos recursos necessários para a implementação e manutenção de um típico SDW. Especificamente o objectivo do processo de ETL passa por remover dados repetidos e corrigir dados perdidos, ajustando os dados provenientes de

diversas fontes para poderem ser utilizados em conjunto por forma a estruturar a informação de tal modo que esta seja utilizada e interpretada pelo utilizador final.

Para que o SDW possa estar alinhado com a estratégia da organização e servir de base de apoio nas decisões tomadas, é essencial que os requisitos de negócio se encontrem correctamente implementados, na medida que irá influenciar a qualidade da informação gerada pelo sistema. Tendo em consideração tal importância, acaba por ser um contrassenso que actualmente existam poucas metodologias que permitam o planeamento de uma aplicação de ETL de uma forma estruturada e documentada. Kimball [10] discute exactamente este problema e apresenta uma metodologia que consiste num conjunto de passos que permitem descrever de uma forma documentada todo o ciclo de vida de um processo de ETL.

Na obra referenciada, Kimball apresenta dez passos para o desenvolvimento de um processo de ETL. O planeamento de um processo de ETL deve, numa fase inicial, seguir uma abordagem de alto nível, onde se deverá ter em consideração as tecnologias e infraestrutura a utilizar, de modo a ajustar o planeamento à ferramenta de ETL, evitando futuras modificações para adaptar o desenho inicialmente desenvolvido. O primeiro passo no desenvolvimento de um processo de ETL passa por construir um plano de alto nível onde são identificadas as fontes e alvos que recebem os dados. Esta primeira esquematização mantém-se a um nível mais elevado de abstracção, com objectivo de obter uma visão superficial sobre os dados. Frequentemente este primeiro plano é utilizado não só para comunicar com os restantes membros da equipa de desenvolvimento, mas também com intervenientes que não estão familiarizados com pormenores técnicos. De seguida será necessário escolher a ferramenta de ETL que vai suportar a implementação de todo o processo. Existem no mercado diversas ferramentas como a Microsoft Integration Services[1], Oracle Warehouse Builder[2], ou até soluções *open source* como: Talent Open Studio[3] e Pentaho Data Integration[4]. A documentação gerada por este tipo de aplicações é uma das grandes vantagens da sua utilização. A utilização de notações gráficas, suportadas pela maioria das aplicações, permite uma utilização mais acessível e definida, aumentando a performance e a consistência dos processos. No entanto surge a necessidade de existência de um conjunto de regras *standard* que permita uma implementação uniforme nas diversas ferramentas comerciais. Actualmente, sempre que exista uma mudança de ferramenta de suporte ao processo de ETL será necessário começar praticamente do zero, seja na construção do ETL como no processo de aprendizagem, devido às características próprias de cada ferramenta.

Uma vez definida a ideia e os requisitos de um processo de ETL, será necessário desenvolver um conjunto de estratégias base para as actividades mais comuns, como por exemplo determinar o

[1] http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/integration-services.aspx
[2] http://www.oracle.com/technetwork/developer-tools/warehouse/overview/introduction/index.html
[3] http://www.talend.com/products-data-integration/talend-open-studio.php
[4] http://kettle.pentaho.com/

método de extracção de dados de cada fonte, definir a arquitectura de armazenamento da informação, definir políticas que assegurem a qualidade dos dados caso estes não sejam aprovados no controlo de qualidade, gerir as mudanças de atributos das dimensões, avaliar a disponibilidade dos sistemas fonte e definir estratégias de armazenamento temporário de dados em disco de forma a serem utilizados noutras etapas do processo de ETL.

De seguida, será necessário detalhar as transformações necessárias para popular cada tabela no SDW, onde é necessário manter hierarquias de granularidade de dados de dimensão limpas, isto é, garantir que não existem relacionamentos de "um para muitos" entre os dados. É também necessário planear qual a sequência de tabelas que vão ser alimentadas e qual a sequência de transformações a aplicar. Será também necessário construir um documento de especificação completo para o processo de ETL onde por exemplo é definida a estratégia de extracção de dados das fontes. A disponibilidade dos sistemas fonte, o desenho das tabelas incluindo nome de colunas, tipo de dados, chaves e constantes, e a frequência de carregamento de dados são também tarefas típicas realizadas nesta etapa. Uma vez definida a especificação de ETL, o ciclo de vida de um processo de ETL prossegue para a alimentação das tabelas de dimensão com dados históricos. Inicialmente as transformações são realizadas em dimensões mais simples (dimensões nível 1), propagando-se a dimensões que armazenam cada linha de dados referente a determinado ponto do passado (dimensões nível 2). Neste ponto do processo são realizadas operações de conversão de valores a nível de tipo de dados, substituição de valores *NULL*, operações de junção relativas a mais do que uma fonte de dados, e é realizada a validação de relacionamentos "de muitos para um" e de "um para um" de modo a garantir que cada atributo de uma coluna de uma dimensão tem exactamente um valor correspondente noutra coluna de outra dimensão. A partir do momento que os dados se encontram preparados, apenas será necessário carregar esses mesmos dados para as respectivas dimensões.

Uma vez alimentadas as dimensões, será necessário alimentar a tabela de factos, onde deverá existir um especial cuidado com as transformações de conversão dos dados de modo a garantir que as regras da integridade referencial entre a tabela de factos e as tabelas de dimensão são cumpridas. De forma a executar o processo de alimentação da tabela de factos, são realizadas as operações de substituição de chaves, que basicamente trocam todas as chaves naturais por chaves substitutas lidando com qualquer problema de integridade referencial que possa surgir permitindo aos dados serem alimentados na tabela de factos.

Após este processo a preocupação no processo de ETL passa por integrar informação incremental nas tabelas de dimensão identificando quais os dados novos, modificados e eliminados, sendo este um passo extremamente moroso a nível computacional. A alimentação incremental da tabela de factos é também um processo de complexidade elevada devendo ser definido um fluxo de operações lógico automatizado para proceder à actualização da tabela de factos, tendo em consideração as regras que terão de ser aplicadas no decorrer do processo. Por fim os últimos passos passam por construir uma

tabela de agregações, que mantém informação resultado de *querys* de agregação, e automatizar as operações de um processo de ETL, criando tarefas agendadas e tarefas que lidam dados inesperados ou situações de erro, por forma a resolver as anomalias de uma forma dinâmica.

Com base no ciclo de vida apresentado [14], Kimball define 4 passos essenciais de um ETL para a alimentação de um SDW (Figura 1), nos quais os dados são escritos no disco, em paralelo com os dados que são transferidos para os passos posteriores.

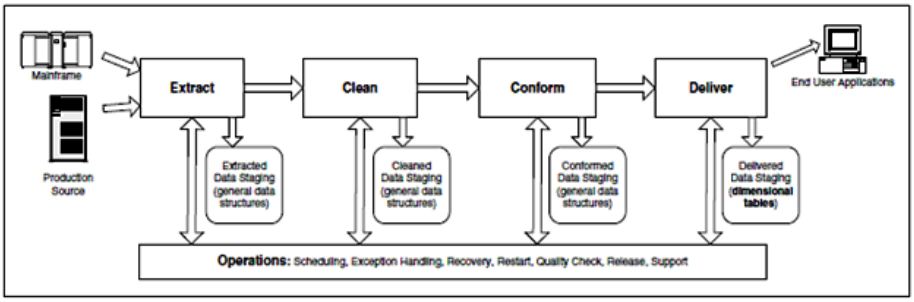


Figura 1 – Os quatro passos essenciais de um processo de ETL segundo Kimball

Os dados são extraídos dos sistemas fonte, através de um processo de extracção, onde os dados provenientes de fontes, como por exemplo bases de dados relacionais, documentos estruturados em XML ou folhas de cálculo, são armazenados em disco sobre uma estrutura mínima de armazenamento. Tipicamente os dados são armazenados em ficheiros simples ou em tabelas relacionais, permitindo simplificar e acelerar o processo ao mesmo tempo que aumenta a sua flexibilidade caso ocorra alguma interrupção. Este conjunto de dados podem ser executados variadíssimas vezes para auxiliar os passos seguintes do processo. Os dados incoerentes, incompletos ou irrelevantes carregados inicialmente são descartados na etapa de limpeza (*cleaning*), na qual a exigência dos requisitos dos dados provenientes das fontes são alinhados com a exigência dos dados aceites neste passo - aqui também se verifica a validade dos dados. Por exemplo se um determinado código postal é válido; e consistência dos dados: verificar se determinado código postal corresponde à cidade em questão. Depois será necessário verificar se a duplicação de dados e as regras de negócio se encontram devidamente implementadas. Este passo assume uma complexidade elevada, sendo muitas vezes necessário armazenar provisoriamente os dados em disco. O passo de uniformização (*Conform*) é necessário sempre que duas ou mais fontes de dados necessitem de ser fundidas. Para a junção de dados de duas fontes será necessário que os tipos de dados sejam compatíveis, assim como os nomes dos atribuídos unificados, de modo a garantir que se mantêm consistentes.

Por fim o passo de alimentação (*Delivering*) é um passo crucial que tem como objectivo estruturar fisicamente os dados no SDW.

Os processos de ETL podem envolver uma complexidade considerável e um mau planeamento pode implicar vários contratempos. O desenho de um processo de ETL deverá ter em consideração a sua

utilização ao longo do seu tempo de vida, no sentido que o volume de dados a processar pode aumentar, sem que, necessariamente, a janela temporal para a execução do processo também aumente. Por isso, deverá ser levada em conta a necessidade de processar um maior conjunto de dados numa janela temporal mais pequena, podendo levar a custos elevados não só na aquisição de *hardware* mas essencialmente na análise de dados e consequentemente no impacto das decisões tomadas.

2.3 Modelação conceptual de um processo de ETL

Tendo em consideração que o processo de ETL é um componente crítico no sucesso da implementação de um qualquer SDW, é natural que o seu desenho seja um factor de extrema importância na implementação deste tipo de processos. O desenho e implementação desta componente envolvem o desenvolvimento de processos complexos, que interagem com a grande maioria das restantes componentes do SDW e baseiam grande parte das suas operações no processamento dos dados que estão armazenados em sistemas transacionais cuja disponibilidade de acesso é condicionada e limitada. É necessário também que a evolução do sistema seja levada em conta, no sentido de acompanhar, o normal desenvolvimento e crescimento de dados num SDW, à medida que mais áreas de negócio vão sendo assimiladas no SDW [15].

O aumento de custos de *hardware* e de manutenção reflectem o crescimento de SDW, tendo em consideração que não só o volume de dados que aumenta mas também o nível de complexidade das operações que naturalmente condiciona a escolha da infraestrutura de suporte, o que faz com que na fase de planeamento se tenha em consideração a evolução da infraestrutura de modo a suportar a execução das operações a realizar, sem que a janela temporal de execução do processo seja condicionada.

2.3.1. A Abordagem de Vassiliadis

Simultaneamente ao planeamento da infraestrutura de suporte ao sistema ETL, é também necessário ter em consideração o desenho das operações de ETL. Esta etapa do processo pode ser realizada seguindo a filosofia utilizada no desenho de bases de dados propostas por Vassiliadis [2], Trujillo [7] e Akkaoui [6]. Através de uma forma bastante estruturada, Vassiliadis apresenta uma notação própria, enquanto que Trujillo e Akkaoui apresentam metodologias baseadas em notações já existentes para a representação de um modelo conceptual do processo de ETL, com o objectivo de o tornar de mais fácil legibilidade, interpretação e actualização, caso seja necessário.

As três notações permitem identificar as fontes de dados, atributos e as transformações necessárias para extracção, transformação e alimentação de dados no SDW. A notação proposta por Vassiliadis baseia-se num conjunto de elementos gráficos apresentados na Tabela 1, que permitem a construção

do modelo conceptual tendo em consideração as restrições e anotações, possibilitando assim a formalização de um processo que é documentado frequentemente de forma muito *ad-hoc*.

Ícone	*Significado*
	Conceito
	Atributo
	Transformação
	Nota
	Parte de
	Providencia

Tabela 1- Notação gráfica usada na modelação conceptual proposta por Vassiliadis [2]

No entanto, na prática, a utilização desta notação pode tornar-se bastante confusa uma vez que as dimensões e as tabelas de factos de um SDW podem ser constituídas por um conjunto elevado de atributos, distribuídos por várias fontes, levando a um aumento da complexidade do diagrama, prejudicando a sua leitura e interpretação. Tendo em consideração este facto, é necessário alguma cautela relativamente à utilização desta notação, principalmente recorrendo a uma divisão das tabelas de factos e dimensões relacionadas em diagramas específicos. Recorrendo à notação de Vassiliadis, é apresentada na Figura 2 um exemplo da notação proposta por Vassiliadis para a modelação conceptual de um processo de ETL de alto nível, que tem como objectivo descrever uma operação de alimentação de uma dimensão (dim_Pessoa) que recorre a dados provenientes de três fontes diferentes.

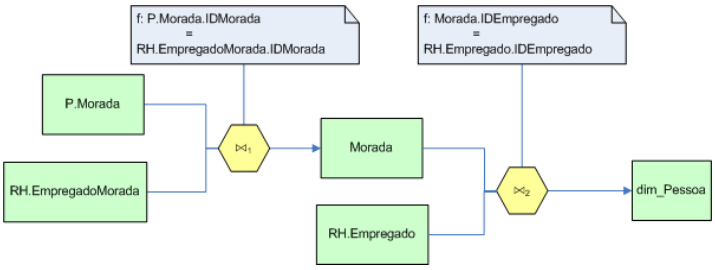


Figura 2 - Exemplo da utilização da notação proposta por Vassiliadis [2] para a modelação conceptual de alto nível

Detalhando o esquema da Figura 2, com a inclusão dos atributos presentes em cada fonte, é possível obter as correspondências de atributos utilizados na alimentação da dimensão dim_Pessoa (Figura 3).

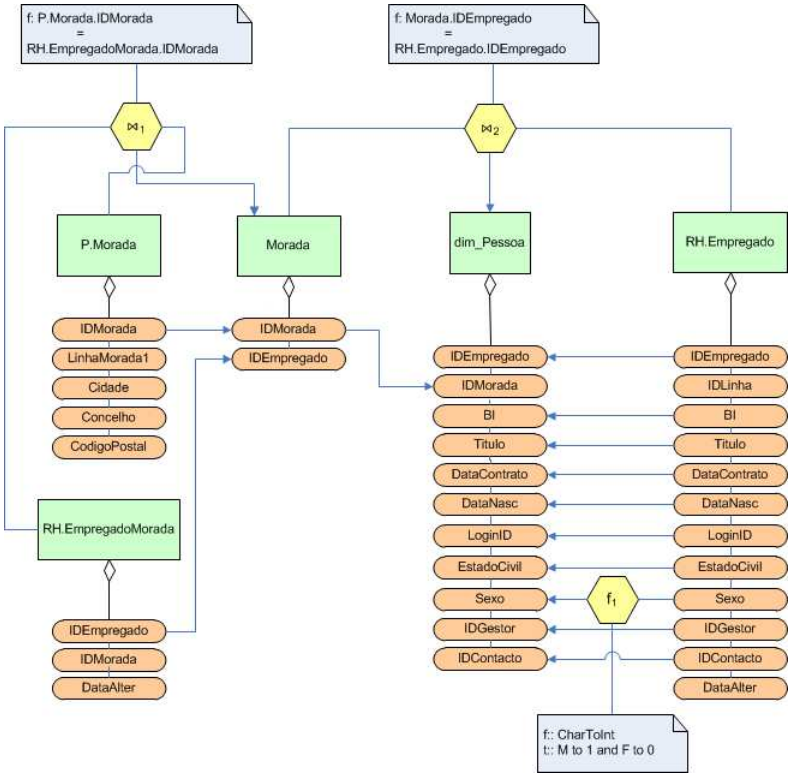


Figura 3 - Modelação conceptual por Vassiliadis [2] com a inclusão de atributos

2.3.2. A Abordagem de Trujillo

Trujillo [6], por sua vez, propõe uma abordagem baseada na notação UML, aplicando conceitos inerentes aos processos de ETL (Tabela 2). Esta notação, embora de mais fácil interpretação dada a popularidade da linguagem UML, recorre em demasia à anotação do diagrama para a explicitação das operações típicas de um processo ETL. Para a representação de processos de nível de dificuldade baixa, a sua clareza é um factor que contribuí para a sua fácil utilização e compreensão.

Na Figura 4 é demonstrado um exemplo de utilização da notação proposta por Trujillo para o processo de ETL descrito anteriormente. No exemplo, é utilizada a operação de Conversão, de forma a representar uma alteração do tipo de dados de um atributo de uma das fontes, e a operação de Junção, para identificar as junções entre pares de fontes com vista à obtenção dos atributos necessários à alimentação da dimensão dim_Pessoa.

Ícone	Mecanismo ETL (Estereótipo)	Descrição
A → B	Conversão	Altera o tipo de dados de atributos
	Junção	Efectua a junção de duas fontes relacionadas entre si em determinados atributos

Tabela 2 - Excerto da notação gráfica usada na modelação conceptual proposta por Trujillo [7]

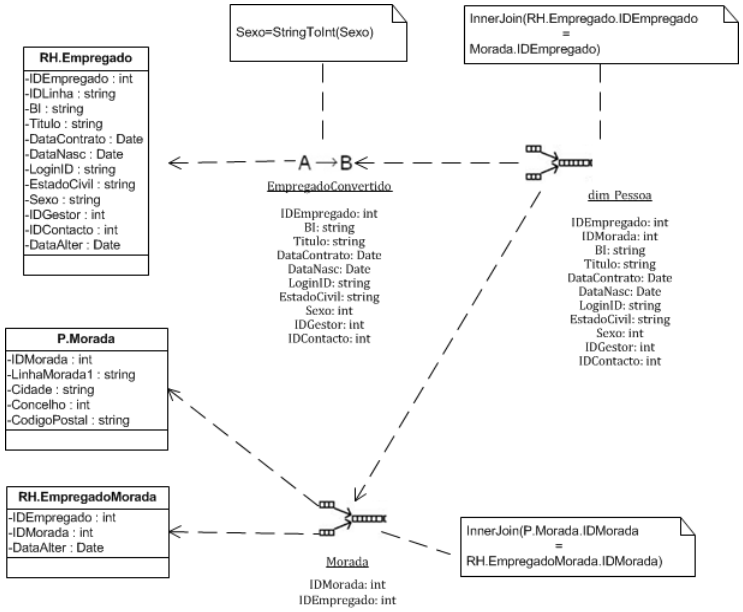


Figura 4 - Exemplo da utilização da notação proposta por Trujillo [7] para a modelação conceptual

2.4 Modelação Lógica de um processo de ETL

Após a elaboração do modelo conceptual para o sistema de ETL é necessário elaborar o respectivo modelo lógico, no qual serão discriminadas cada uma das operações em conjunto com o fluxo de execução de cada uma delas. No modelo conceptual identificam-se quais os dados existentes e as transformações a implementar, enquanto que no segundo se representa a sequência das operações necessárias à execução das transformações apresentadas no modelo conceptual.

2.4.1. A Abordagem de Vassiliadis

Vassiliadis [16] propõe, na sequência do trabalho anteriormente referido, uma notação gráfica que permite descrever as operações para especificar o fluxo da execução de todas as operações envolvidas com o processo de alimentação do SDW. No modelo conceptual identificam-se quais os dados

existentes e as transformações a implementar, enquanto que no modelo lógico se esquematiza a sequência das operações necessárias à execução das transformações apresentadas no modelo conceptual.

Na Figura 5 é apresentado um diagrama segundo a notação de Vassiliadis com a aplicação prática da notação proposta, tendo por base o modelo conceptual presente na Figura 3. As operações estão sequenciadas e podem, por questões de optimização, ser reposicionadas no decorrer dos fluxos que compõem as transformações efectuadas às três fontes de dados.

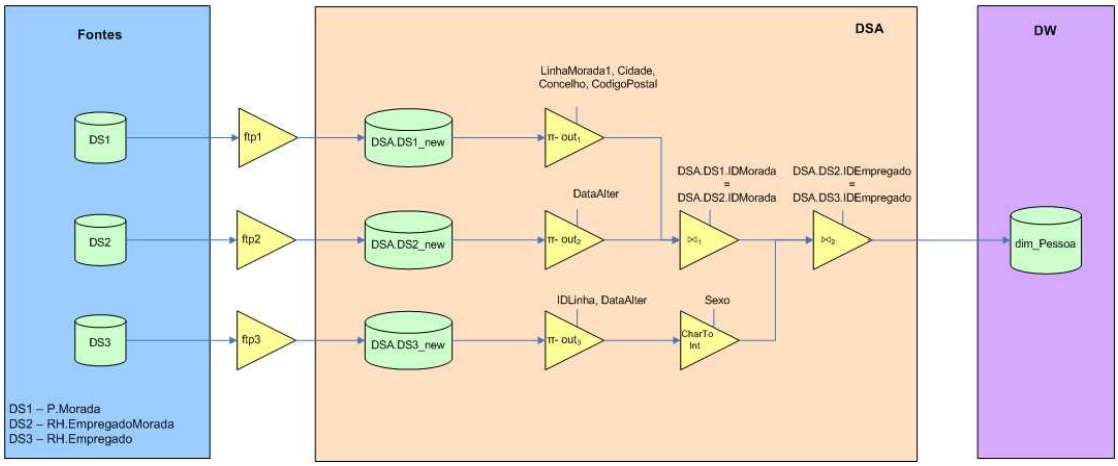


Figura 5 - Exemplo da utilização da notação proposta por Vassiliadis [17] para a modelação lógica

2.4.2. A Abordagem BPMN

A *Business Process Modeling Notation* (BPMN) é um *standard* de modelação de *workflows* de processos de negócio. Esta notação foca-se em dois pontos fundamentais: a gestão e planeamento de um processo de *workflow*, e com a modelação e arquitectura de implementação [18].

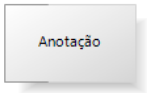
Notação	Nome	Descrição
	Tarefa	Na notação BPMN uma actividade representa o trabalho realizado dentro de um processo de negócio. É representada por um rectângulo arredondado nos cantos. Uma actividade normalmente demora algum tempo e normalmente requer algo de *input*, produzindo algo como *output*.
	Eventos	Um Evento é algo que acontece durante o percurso de um processo de negócio. Estes eventos afectam o fluxo do processo e normalmente têm uma causa (*trigger*) ou um impacto (resultado). Na presente tabela encontram-se descritos o evento de *Start* (circulo verde) e o Evento de *End* (circulo vermelho).
	Gateway paralelo	Os Gateways são elementos de modelação que controlam como o processo diverge ou converge, isto é, representam pontos de controlo para os caminhos presentes no processo. Os Gateways separam ou juntam o fluxo dos processos. O Gateway do tipo Parallel possui duas vertentes: uma de separação para distribuição de caminhos, e uma outra vertente de junção onde espera pela chegada de todos os caminhos de entrada.
	Anotação	Anotações são um mecanismo que permitem adicionar informação adicional, de modo a que o utilizador que se encontra a analisar o diagrama compreenda melhor o seu funcionamento.
	Fluxo de sequência	O fluxo de sequência é representado por uma linha sólida com uma seta preenchida, e é utilizada para demonstrar a ordem (a sequência) das actividades que vão ser realizadas no processo.
	Associação	Uma Associação é utilizada para associar dados, texto e outros artefactos com objectos. As associações são utilizadas para mostrar os *inputs* e *outputs* das actividades.

Tabela 3- Descrição da notação BPMN

Tendo por base [18] e [19], a Tabela 3 apresenta um resumo dos artefactos mais relevantes para o trabalho em questão.

Com base na premissa que um processo de ETL pode ser considerado um tipo particular de processo de negócio, Akkaoui [6] apresenta uma adaptação da notação BPMN para a modelação de processos de ETL. Os autores aplicam a notação BPMN para a modelação de processos de ETL, com a perspectiva de implementação de uma metodologia de modelação *standard*, de forma a reduzir os vários tipos de problemas associados a este tipo de tarefas, utilizando para isso as diversas vantagens de uma *framework*, como a que a notação BPMN disponibiliza.

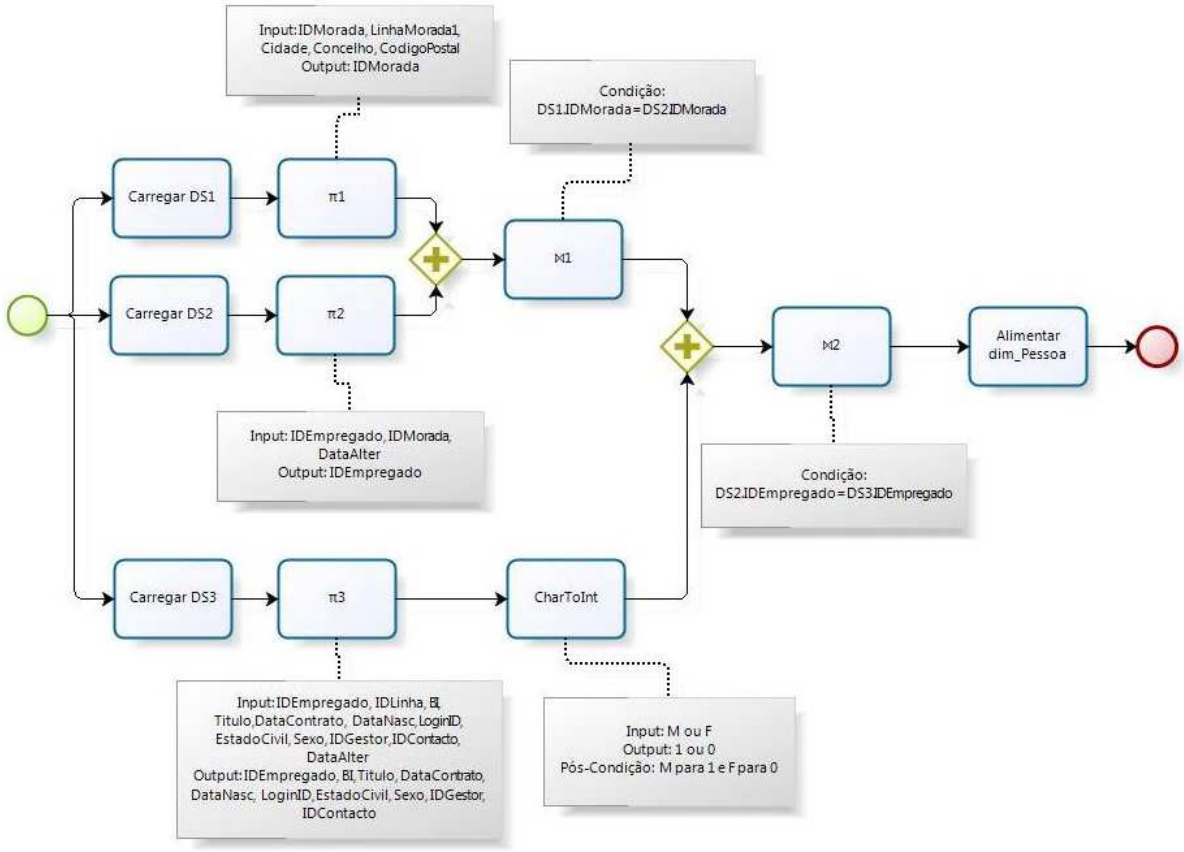


Figura 6 - Conversão do Modelo Lógico de Vassiliadis para a notação BPMN

Na Figura 6 é apresentada a conversão do modelo lógico de Vassiliadis para a alimentação de uma dimensão (dim_Pessoa) utilizando uma versão simplificada da notação BPMN, para a modelação de processos de ETL [6].

Comparativamente à abordagem apresentada por Kimball, as metodologias enunciadas assumem uma outra importância e qualidade na estruturação, simplicidade e compreensão do desenho do processo de ETL. A abordagem apresentada por Kimball assume-se como um procedimento muito *ad-hoc* de especificação de requisitos de um processo de ETL, consistindo em esquemas que têm mais utilidade quando escritos manualmente, do que propriamente utilizando um ambiente computacional. Relativamente à abordagem apresentada por Vassiliadis é possível identificar dois problemas principais: primeiro a notação é criada de raiz para o propósito obrigando os elementos do projecto a aprender as especificidades da notação; segundo porque é uma notação que não possui suporte nem em termos de modelação nem em termos de mapeamento do diagrama para primitivas de programação. A notação de Trujillo resolve em parte esse problema devido ao seu apoio na linguagem UML que para além de reduzir a curva de aprendizagem é uma linguagem simples e com bastante suporte em termos aplicacionais. No entanto para processos de ETL de uma maior complexidade, isto é, com um maior número de transformações, torna-se necessário recorrer em demasia a anotações para complementar o diagrama, devido à não especificação de tarefas na notação proposta por

Trujillo, o que naturalmente faz com que para processos de ETL minimamente complexos, o diagrama se torne de difícil leitura e compreensão.

A proposta de Akkaoui baseia-se numa notação que providencia uma notação de fácil compreensão para várias categorias de utilizadores, analistas de negócio, programadores que são responsáveis pela implementação da tecnologia que suportam esses processos, e gestores de topo que gerem e monitorizam esses mesmos processos. Como é uma das notações mais utilizadas a nível comercial, torna-se uma notação de fácil implementação para vários tipos de utilizadores dentro de uma organização, permitindo reduzir a curva de aprendizagem e facilitando a comunicação com outras organizações. A conversão desta notação para um formato que as linguagens de programação possam interpretar é também um ponto forte, na medida que esta notação suporta a linguagem BPEL [20] e permite a conversão dos diagramas em vários formatos como é o caso do formato *XML Process Definition Language* [21, 22] (XPDL)[5], que é um formato estandardizado pela *Workflow Management Coalition* (WfMC)[6] que permite definir um esquema XSD para a especificação de *workflows*.

A necessidade de uma notação que permita o mapeamento das operações realizadas em todo o processo de ETL para um contexto independente de ambiente de execução torna-se por isso essencial. As operações de transformação dos dados devem ser operações atómicas, isto é cada transformação deverá corresponder a uma operação de Álgebra Relacional por forma a gerar uma decomposição de transformações, permitindo retirar vantagens em termos da sua execução em vários tipos de ambientes de execução, como por exemplo sistemas distribuídos. O sistema de suporte ao armazenamento de dados a processar pelo processo de ETL é também um elemento crucial, pois só mantendo um suporte de dados independente é que se torna possível a sua interpretação e execução em diferentes ambientes de execução.

5 http://www.xpdl.org/

6 http://www.wfmc.org/

Capítulo 3

Utilização da Álgebra Relacional na Modelação de um Sistema de ETL

3.1 Álgebra Relacional

Apresentada por Codd [23] como a base para as linguagens relacionais, a *Álgebra Relacional* (AR) é uma linguagem procedimental de alto nível que é utilizada pelos *Sistemas de Gestão de Bases de Dados* para construir novas relações a partir de uma ou mais relações existentes numa Base de Dados.

A AR define as operações base necessárias para qualquer *Linguagem de Manipulação de Dados*, consistindo numa linguagem cujas operações actuam sobre uma ou mais relações de modo a definir uma outra relação sem alterar a relação original. Tanto os operandos como os resultados são relações, por isso os dados resultantes de uma operação podem ser os dados de entrada de outra operação.

Existem várias operações que se encontram incluídas em Álgebra Relacional. Codd [24] originalmente propôs oito operações. No entanto outras operações foram desenvolvidas posteriormente. As cinco operações fundamentais em Álgebra Relacional são: Selecção, Projecção, Produto Cartesiano, União e Diferença que permitem executar a maioria das operações de extracção de dados.

A Selecção e a Projecção são operações unárias, devido a operarem sobre uma única relação. As restantes operações operam sobre pares de relações e como tal são consideradas operações binárias. Na Tabela 4 foram consideradas R e S, duas relações definidas sobre os atributos $A=(a_1,a_2,\ldots,a_N)$ e $B=(b_1,b_2,\ldots,b_M)$, respectivamente.

No caso das operações unárias, uma relação R possui um estado inicial e com a aplicação da operação surge como resultado uma nova relação. No caso das operações binárias existem duas relações (R e S)

que quando aplicada a respectiva operação, irá produzir uma nova relação resultante da operação aplicada sobre as duas relações.

Apesar de inúmeras propostas por parte de diversos autores [25, 26], e de alguma indefinição relativamente à notação utilizada, a Álgebra Relacional clássica é constituída, segundo Ullman e Widom [27] pela operação de União, Intersecção, Diferença, Selecção, Projecção, Produtos, Junções e a operação de Renomeação (Tabela 4).

Operador	**Notação**	**Função**
Selecção	$\pi_{predicado}$ (R)	Produz uma relação que contém os tuplos de R que satisfazem o predicado
Projecção	$\pi_{a1,...,an}$ (R)	Produz uma relação que contém um subconjunto vertical de R, extraindo os valores de atributos específicos eliminando duplicados
União	R ∪ S	Produz uma relação que contém todos os tuplos de R, ou de S, ou simultaneamente de R e S, duplicados são eliminados. R e S devem ser compatíveis na União
Diferença	R - S	Produz uma relação que contém todos os tuplos de R que não se encontram em S. R e S devem ser compatíveis na União
Produto Cartesiano ou *cross product*	R × S	Produz uma relação que concatena cada tuplo de uma relação R com cada tuplo de uma relação S
Intersecção	R∩S	Produz uma relação que contém todos os tuplos contidos em R e S simultaneamente. R e S devem ser compatíveis na união.
Theta join	$R \bowtie_{C} S$	Produz uma relação que contém os tuplos que satisfazem o predicado F do produto cartesiano de R com S. O predicado F é definido por $R.a_i\ \theta\ S.b_i$ em que θ pode ser um dos seguintes operadores de comparação: <,≤,>,≥,=,≠.
Natural join	R⋈S	Produz uma relação sobre todos os atributos comuns entre R e S em que o predicado contém apenas condições de igualdade(=). Uma ocorrência de cada atributo comum é eliminada da relação resultado.
Renomeação	$\rho_{S(A1,\ldots,An)}(R)$	Produz uma relação idêntica a R mas renomeada segundo S em termos de atributos.

Tabela 4 – Álgebra Relacional clássica segundo Ullman e Widom [27]

Embora não a formalizassem, Ullman e Widom apresentam uma extensão à Álgebra relacional clássica apresentando os operadores de eliminação de duplicados, Agrupamento, Agregação, Ordenação, extensão da projecção e *Outer joins* [27]. No trabalho mencionado é ainda descrita uma abordagem que permite a existência de tuplos duplicados numa relação (*bags* ou *multisets*). A AR clássica proposta por Codd tem como base a existência de *Sets*, que, entre outras propriedades, se refere a relações onde não existe a duplicação de tuplos. No entanto as bases de dados comerciais

raramente utilizam *Sets*, uma vez que as relações permitem a existência de duplicados. Por exemplo, o resultado de uma *query* a uma base de dados pode resultar em duplicados, tornando a relação resultante inválida, segundo as regras do modelo relacional original. A Álgebra Relacional Extendida foi desenvolvida para suportar a existência de tuplos duplicados, operações algébricas sobre atributos e manipulação de dados como inserções, actualizações ou remoções, recorrendo à utilização de *bags* onde para além da existência de tuplos duplicados, a ordem desses mesmos tuplos assume especial importância, de modo a proporcionar um aumento de performance na execução de operações sobre relações.

Embora não suportado como operador primitivo a operação de divisão é também utilizada por autores na definição de Álgebra Relacional. Em [11] é apresentada uma definição não formal de diversos operadores de Álgebra Relacional incluindo a operação de divisão, que juntamente com as operações de Junção e intersecção podem ser expressas através das cinco operações básicas. Existem contudo operadores que não podem ser expressos através das operações básicas, como o operador de ordenação [28]. Embora exista alguma discrepância entre autores na definição de algumas operações, por exemplo na definição da *Outer join* em [27] e [11], é apresentada na Tabela 5 um resumo da descrição das operações de Álgebra Relacional extendida, baseado no trabalho de vários autores [11, 27, 29].

Ao longo do tempo têm sido apresentados alguns melhoramentos [30] relativamente a algumas operações, sendo difícil considerar que operações são consideradas *standard* ou não. Sobre a perspectiva teórica de linguagens de base de dados, ou seja a Álgebra Relacional, têm sido incorporadas operações provenientes da componente prática e comercial da linguagem SQL (*Structured Query Language*), devido ao crescente aumento da complexidade de operações, essencialmente devido ao aumento de complexidade dos processos de negócio, onde as operações elementares apresentadas na Tabela 4 e Tabela 5 não são suficientes para representar a realidade da implementação das linguagens de base de dados. Como tal surgiram novas operações como é o caso da operação de agrupamento e ordenação de tuplos.

Ao aliarmos a necessidade de existência de novas operações à exigência de performance das implementações de bases de dados, é possível concluir que a sequência com que as operações são executadas tem um peso elevado no decorrer do fluxo de execução de um processo que envolva a execução de diversas operações.

No contexto da modelação lógica de processos de ETL, a AR revela-se uma linguagem a considerar, devido à sua rigorosa especificação e essencialmente porque permite a decomposição de operações de transformação típicas de um processo de ETL em operações elementares. Assim é possível criar uma hierarquia de operações para a execução de um processo de ETL em vários tipos de ambiente de execução.

Operador	Notação	Função
Natural Outer join	$R \overset{\circ}{\bowtie} S$	Produz uma relação que contém os tuplos originários de uma Natural *join*, acrescentando os tuplos de R e S que não possuem correspondência. Aos valores dos atributos que não têm correspondência são colocados a *null* (⊥).
Natural Left Outer join	$R \overset{\circ}{\bowtie}_L S$	Produz uma relação que contém os tuplos resultantes de uma Natural *join* acrescentando os tuplos de R que não possuem correspondência com os atributos de S. Os valores dos atributos de R que não têm correspondência com a relação S são definidos com o valor *null*(⊥).
Natural Right Outer join	$R \overset{\circ}{\bowtie}_R S$	Produz uma relação que contém os tuplos resultantes de uma Natural *join* acrescentando os tuplos de S que não possuem correspondência com os atributos de R. Os valores dos atributos de S que não têm correspondência com a relação R são definidos com o valor *null*(⊥).
Grupos	$\gamma_L(R)$	O atributo da relação R onde a operação de agrupamento é aplicada, é um dos atributos por onde R vai ser agrupado. Neste operador pode-se ainda aplicar os operadores de agregação (SUM, AVG, MIN, MAX e COUNT) sobre um atributo de uma tabela. Para a atribuição de um nome para o atributo resultante deverá utilizar-se a indicação → seguido do nome do novo atributo.
Eliminação de duplicados	δ (R)	Produz uma relação com base em R que contém apenas uma cópia dos diferentes tuplos de R
Expansão da projecção	$\pi_L(R)$	A expansão da projecção permite inserir novos atributos numa relação R. A lista L é construída através: • de um atributo simples de R; • uma expressão x→y onde x e y são nomes de atributos, simbolizando a transformação de renomeação de um atributo x de R para y; • uma expressão E→z onde E é uma expressão construída a partir de atributos de R, constantes, operadores aritméticos e operadores de caracteres. z é o nome do novo atributo calculado por E.
Ordenação	$\tau_L(R)$	A operação de ordenação permite a ordenação de tuplos de uma relação R pelas ordem dos atributos especificados em R.

Tabela 5 – Álgebra Relacional Extendida

3.2 Modelação lógica de um processo de ETL utilizando Álgebra Relacional

Com base em AR e de forma a adaptar o modelo lógico da Figura 3 para um modelo de representação de cada operação elementar de AR, foi proposto em [8, 31], uma modelação que através de uma árvore de operações representa uma sequência de execução de um processo de ETL.

O modelo desenvolvido (Figura 7), descreve o fluxo de tarefas sinalizado pelo nodo *Start*, que indica o início do processo. De seguida são representadas as operações que têm como origem o nodo *Start* e que podem ter como *input* de dados fontes diferenciadas. Quando um nodo dá origem a mais do que uma operação então essas operações que divergem do nodo não possuem relação directa entre si,

podendo ser executadas paralelamente, como é o caso das três projecções desencadeadas pelo nodo *Start*.

A decomposição de operações não elementares é uma das características do modelo. A operação de conversão do atributo Sexo que tem como objectivo converter dados do tipo carácter (M ou F) para dados do tipo inteiro (1 ou 0), foi decomposta em operações de AR, dando origem a duas selecções, duas novas projecções e a uma operação de união.

Quando uma operação tem como origem um nodo diferente do *Start* então os dados de entrada dessa operação serão o conjunto de dados transformados pela operação anterior, isto é, o documento resultado de uma operação representa a entrada de dados da operação seguinte, assim como as operações de AR. Quando duas ou mais operações convergem numa nova operação, então será necessário que as operações que a precedem terminem para que a execução da operação actual se possa iniciar. No exemplo apresentado é possível observar isso mesmo através das operações de junção, que necessitam que as operações que as precedem terminem a sua execução.

O fim da execução do processo de ETL é sinalizado pelo nodo *End*. Na Tabela 6 encontram-se descritas, em Álgebra Relacional, cada uma das operações utilizadas na modelação lógica do ETL da Figura 7.

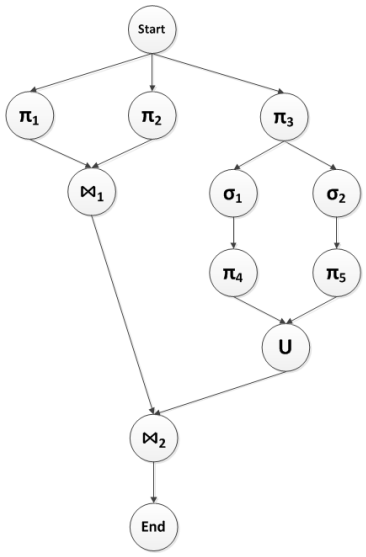


Figura 7 - Modelação lógica de um processo de ETL

A Tabela 6 apresenta a descrição da operação para cada um dos nodos do modelo lógico apresentado.

Uma questão fundamental a ter em consideração é a representação do modelo em primitivas de programação para que possa ser interpretado, de modo a identificar que operações realizar, a sua forma de execução e qual a sua ordem. Para tal, a representação deste modelo, utilizando um formato universal, neste caso XML, torna-se fundamental nesta abordagem. Na Figura 8 é apresentado o esquema XSD de suporte a este modelo lógico, apresentado em [8, 31].

Operação	Representação em Álgebra Relacional
π_1	$\pi_{(IDMorada)}$ (P.Morada)
π_2	$\pi_{(IDEmpregado, IDMorada)}$ (RH.EmpregadoMorada)
π_3	$\pi_{(IDEmpregado, BI, Titulo, DataContrato, DataNasc, LoginID, EstadoCivil, Sexo, IDGestor, IDContacto)}$ (RH.Empregado)
$\bowtie_1$	$R \leftarrow \pi_1 \bowtie \pi_2$
$\boldsymbol{\sigma_1}$	$\sigma_{(Sexo='M')}$ (RH.Empregado)
$\boldsymbol{\pi_4}$	$\pi_{(IDEmpregado, BI, Titulo, DataContrato, DataNasc, LoginID, EstadoCivil, 1->Sexo, IDGestor, IDContacto)}$ (σ_1)
$\boldsymbol{\sigma_2}$	$\sigma_{(Sexo='F')}$ (RH.Empregado)
$\boldsymbol{\pi_5}$	$\pi_{(IDEmpregado, BI, Titulo, DataContrato, DataNasc, LoginID, EstadoCivil, 0->Sexo, IDGestor, IDContacto)}$ (σ_2)
U	$S \leftarrow \pi_4 \cup \pi_5$
$\bowtie_2$	$R \bowtie S$

Tabela 6 - Representação das operações utilizando Álgebra Relacional

O modelo apresentado, tem como objectivo representar em XML o fluxo de dados associado com o modelo em AR apresentado na Figura 7, identificando todas as actividades (tarefas) a serem realizadas, bem como o respectivo fluxo de dados e a estrutura de execução das operações de transformação. Cada operação de AR é, obrigatoriamente, definida pelos atributos:

- *id*, que representa um identificador para a operação;
- *operation* com a indicação da operação a ser utilizada;
- *datasource* com a fonte relativa aos dados a transformar;
- *datadestination*, que permite especificar o armazenamento intermédio permanente de um resultado de uma determinada operação, para posterior consulta;
- *relationships*, que permite indicar as operações que a operação tem dependências, identificadas devidamente pelo seu id;
- *parameters* indica os parâmetros que devem ser enviados para a respectiva operação de transformação, permitindo definir o seu comportamento.

Em AR existem diversas operações unárias e binárias que possuem especificações diferentes em termos da sua configuração, por exemplo, uma operação de projecção necessita de receber como parâmetros os atributos a projectar, uma selecção os predicados de validação dos dados e uma operação de junção necessita de receber como parâmetro as relações para realizar a junção, ou seja, o *output* das operações que a precedem de modo a que a operação de junção possa concluir correctamente.

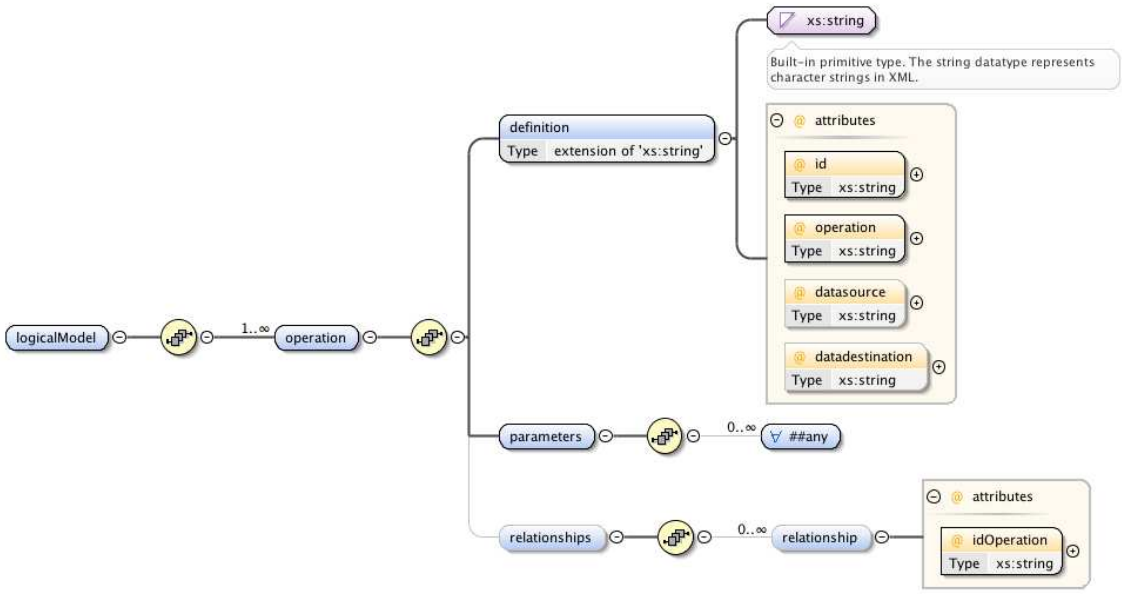


Figura 8 - Proposta de esquema XSD para a representação do fluxo de operações de Álgebra Relacional

Apesar de descritivo em relação às operações e à sua sequência, este modelo acaba por possuir duas importantes lacunas: primeiro por possuir uma notação não *standard* sem suporte aplicacional, segundo porque a descrição das operações terá de ser realizada à parte do diagrama, isto é, numa tabela complementar. Mais importante do que o modo de representação, é a forma de realizar a modelação do processo de modo a operacionalizá-lo.

Com base nas vantagens da utilização da notação BPMN na modelação de processos de ETL é proposta a adaptação da notação BPMN para a representação deste modelo de modo a operacionaliza-lo. A ideia será utilizar a notação BPMN de forma a descrever o Modelo Lógico de operações em Álgebra Relacional, que represente o fluxo de operações, assim como o seu comportamento em termos de execução, de uma forma gráfica e facilmente mapeável para primitivas de programação.

Utilizando a notação BPMN já apresentada para modelação de processos de negócio, é apresentada na Figura 9 a respectiva adaptação da notação para a representação do Modelo Lógico de representação de um processo de ETL em AR recorrendo à notação BPMN.

Assentando numa notação poderosa como a BPMN e com as suas características de representação, a operacionalização do modelo torna-se mais viável, fiável e coerente com a modelação de processos de ETL em SDW [6].

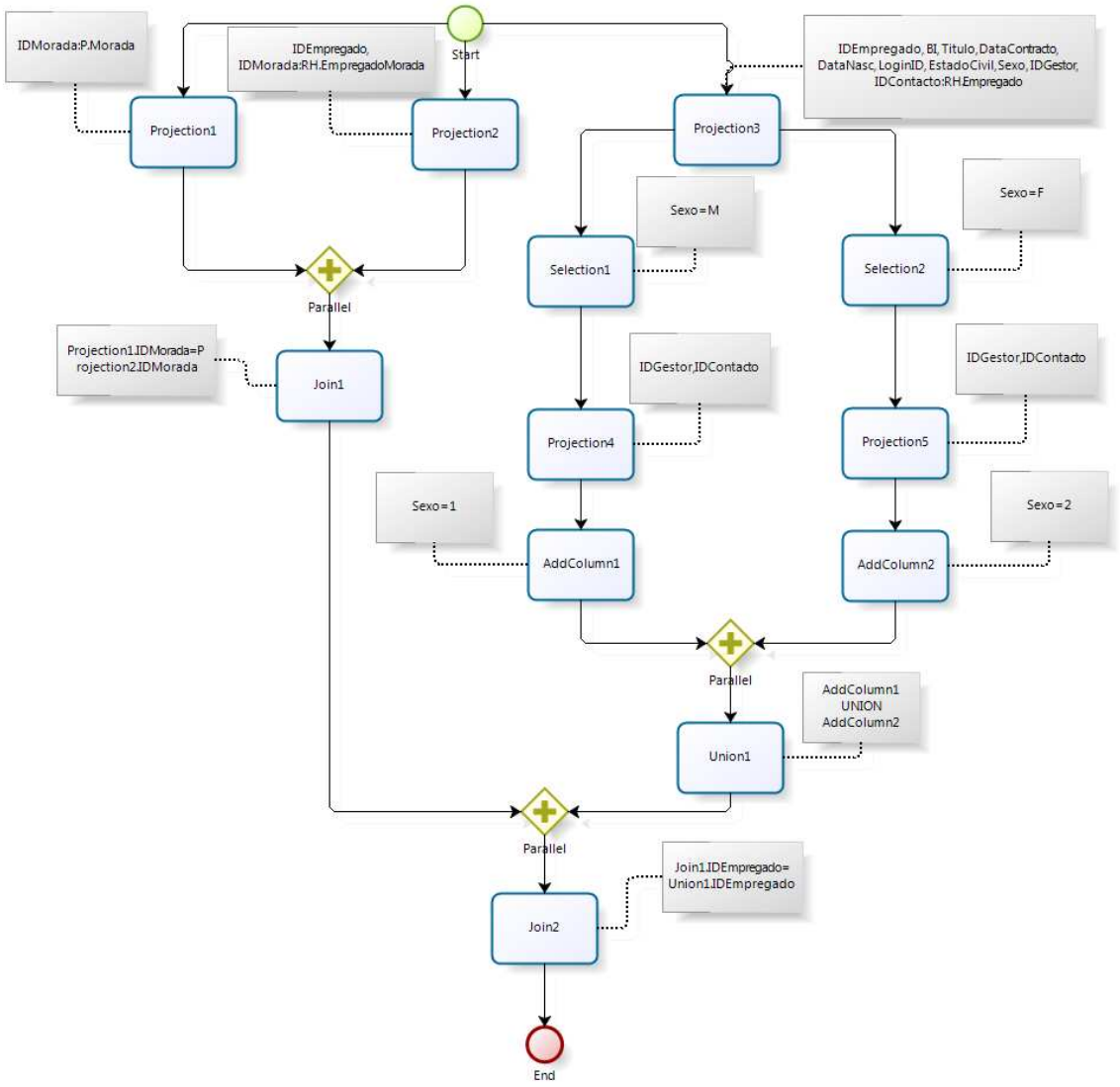


Figura 9 - Modelação lógica de um processo de ETL utilizando a notação BPMN

A notação apresentada baseia-se nos artefactos originais da notação BPMN. Os sinalizadores *Start* e *End* (à semelhança do modelo em AR) indicam o início e fim do *workflow* respectivamente. No artefacto "tarefa" é especificada a operação a realizar, que por uma questão de simplicidade de implementação é representada através do nome da operação correspondente em AR, com uma numeração que indica a sua identificação. O artefacto "*Gateway* Paralelo" pretende sinalizar uma execução em paralelo de operações, onde convergem resultados das operações que o precedem, e cuja saída de dados será o conjunto de entrada de dados que irá alimentar a próxima operação. Um artefacto fundamental em todo este processo é o artefacto "anotação" que no modelo apresentado será o artefacto responsável por indicar a definição da operação a que se encontra associado através do artefacto "tarefa", ao invés de uma tabela complementar. A definição da operação é indicada sobre uma linguagem própria que mediante a operação em questão define um conjunto de regras que complementam a operação de AR. Como é possível observar na Figura 9, o diagrama é composto por diversas operações: Projecção, Selecção, Junção, Adição de coluna e União. Cada uma delas possui

uma definição diferente. No caso da operação de Projecção são indicados os atributos que é pretendido projectar separados por vírgula. No caso da selecção são colocadas as condições tal como são definidas na cláusula *where* da linguagem SQL. Na Junção é definido o critério de junção utilizando o nome das operações que convergem como prefixo para a identificação dos atributos com o respectivo operador de comparação. Por fim a operação de União é definida pelo nome de cada uma das operações que convergem, onde cada elemento da união é identificado pelo nome da operação que lhe dá origem. A fonte de dados de entrada deverá ser especificada colocando o caracter ':' seguido do nome da fonte de dados de entrada. Se uma operação receber como conjunto de dados de entrada, os dados transformados por uma operação precedente, então não será necessário referir qualquer indicação relativamente à fonte de dados.

De modo a permitir a execução do modelo foi desenvolvida uma aplicação em Java que recorrendo à definição em formato XPDL do modelo apresentado na Figura 9, constrói um documento em formato XML baseado no modelo XSD da para permitir a interpretação de todo o *workflow* de operações e possibilitar a sua execução em ambientes computacionais variados. Um dos pontos a considerar é a execução de transformações equivalentes às operações de AR em ambientes computacionais que não possuam o suporte nativo para essas transformações e para o armazenamento de dados, que normalmente são estruturados em modelos de bases de dados relacionais ou modelos de dados de SDW. A utilização de linguagens de programação juntamente com sistemas de suporte de dados multiplataforma favorece a execução de transformações de um processo de ETL em ambientes computacionais diversos, tornando-se um dos pontos de análise do trabalho apresentado.

Capítulo 4

Implementação Física de Tarefas de ETL

4.1 Operações desenvolvidas recorrendo à linguagem Java

Uma vez definida a sequenciação e a decomposição das operações constituintes do modelo lógico apresentado para um processo de ETL, torna-se essencial prosseguir para a modelação física de todo o processo, uma vez que na modelação lógica é descrito todo o fluxo de dados associado ao processo de transformação, sem no entanto serem especificadas características de execução cuja necessidade de representação é essencial. Na modelação lógica de um processo de ETL é definido quais as operações que vão ser executadas, juntamente com a sua definição de transformação. Na modelação física é necessário especificar os requisitos de suporte a nível tecnológico que irá suportar todo o processo. Intrinsecamente surge uma questão óbvia: Como aplicar as respectivas transformações do processo de ETL em ambientes de execução que não possuem suporte aplicacional para essas transformações?

Para responder à questão enunciada é necessário proceder a uma separação de conceitos. No cenário de um SDW os dados são armazenados sobre padrões de armazenamento próprios que aliados à capacidade de execução das transformações, recorrendo a uma linguagem de base de dados, permitem uma total integração e consistência na transformação dos dados. Como a modelação lógica apresentada no presente documento segue uma filosofia não convencional a nível de infraestrutura de suporte, será necessário ter em consideração o suporte aplicacional do modelo para a aplicação das transformações, assim como o sistema de suporte de armazenamento dos dados, uma vez que as transformações sobre os dados devem ser aplicadas num formato universal de representação de dados.

Ao longo dos anos o formato XML tem-se afirmado cada vez mais como um formato de armazenamento de dados. A sua representação hierárquica dos dados tem-se popularizado em diversas

áreas, essencialmente devido à sua simplicidade e portabilidade. Na Figura 10 é possível observar um exemplo de um email representado no formato XML. É comum ver a aplicação do XML na comunicação entre *web services*, transferência de dados entre organizações e até mais recentemente em bases de dados, quer seja no suporte a bases de dados relacionais, quer na criação de bases de dados completamente baseadas neste formato de armazenamento[7]. Adicionalmente é possível recorrer a linguagens que através de *metadata* permitem a definição de regras de construção (XSD[8]), estilos de apresentação da informação (XSL[9]) e até de extracção de dados (XQuery[10] e XPath[11]) de um documento estruturado em XML.

```
<email>
   <para>Rui</para>
   <de>Bruno</de>
   <Assunto>Lembrança</Assunto>
   <body>Não te esqueças de configurar o router!!<body>
</email>
```

Figura 10 – Exemplo de estrutura de um documento XML

Através do formato XML para um armazenamento universal dos dados, é agora possível responder à questão colocada previamente. Recorrendo à definição das operações de AR, é possível a codificação de cada uma das suas operações numa linguagem de programação de forma a replicar o seu comportamento não num conjunto de dados relacionais mas num conjunto de dados hierarquizados em formato XML.

Recorrendo ao conceito de máquina virtual, a linguagem Java assume-se como uma linguagem de programação portável que executa independentemente de plataforma, permitindo assim aplicar o slogan: "*Write once, run anywhere*" [32], que no fundo significa que após a escrita de um aplicativo utilizando a linguagem Java, a sua execução é universal, desde que o sistema operativo que execute o aplicativo tenha suporte para a linguagem Java, o que é bastante comum nos dias de hoje.

Uma vez que existem inúmeros mecanismos em Java que permitem a manipulação de documentos estruturados em XML, Java e XML assumem-se como ingredientes perfeitos de forma a parametrizar o modelo lógico do processo de ETL, servindo de base para definição do documento físico. Isto é, enquanto que no modelo lógico a preocupação era determinar a ordem de execução das operações, no modelo físico será necessário especificar as tecnologias que permitem o suporte à implementação do modelo lógico.

[7] http://exist.sourceforge.net/
[8] http://www.w3schools.com/schema/
9 http://www.w3.org/Style/XSL/
10 http://www.w3.org/TR/xquery/
11 http://www.w3.org/TR/xpath/

Recorrendo a APIs (*Application Programming Interface*) específicas, que representam um conjunto de rotinas e padrões estabelecidos para a utilização das suas funcionalidades, de modo a que as aplicações criadas com base nessa API não necessitem de entrar em detalhes de implementação do software, utilizando apenas as funcionalidades disponibilizadas [33], a linguagem Java fornece um suporte ao processamento de documentos estruturados em XML através de APIs definidas por padrão no módulo JAXP (*Java API for XML Processing*), assim como de APIs desenvolvidas por terceiros que permitem a expansão das capacidades de processamento de documentos XML utilizando a linguagem Java.

Devido à massificação do formato XML e à sua crescente utilização em diversas áreas, a análise de APIs [34-36] para processamento de documentos estruturados em XML, assim como optimizações para essas APIs [37], tem sido um foco da comunidade científica ao longo dos anos. Os modelos de representação de dados implementados, a eficiência na gestão de memória e a performance de execução têm sido os principais pontos de análise. Devido às diferentes arquitecturas de leitura no processamento de documentos estruturados em XML, é possível dividir as APIs em dois grupos distintos: as APIs baseadas em processamento em memória e as APIs baseadas em modo *streaming*.

4.1.1 APIs baseadas em memória

A maioria das APIs baseadas em memória utilizam um modelo comum no tratamento dos dados, onde tipicamente o documento XML é totalmente armazenado em memória num formato de árvore com múltiplos nodos, todos descendendo de um único nodo que representa a raiz da árvore, permitindo a aplicação de vários métodos para localizar e manipular os dados contidos nos nodos. Para cada procura ou algum tipo de manipulação, é necessário iniciar a instrução pelo elemento que se encontra na raiz da árvore e prosseguir na estrutura em hierarquia para aceder aos restantes dados. Como todas as informações estão disponíveis na memória, é possível percorrer a árvore por uma determinada ordem, trocar o posicionamento dos nodos, e realizar combinações de dados de uma forma muito simples e acessível. O processo de *parsing* é normalmente designado como o procedimento de extracção de dados de um documento. No caso das APIs baseadas em memória, este procedimento permite recriar a hierarquia do documento em memória, com base num elemento de raiz (Figura 11).

Devido à sua estrutura de representação de memória, este tipo de APIs permitem tornar o desenvolvimento de aplicações mais acessível, disponibilizando um vasto conjunto de métodos de pesquisa que permitem facilmente realizar operações sobre os nodos constituintes da árvore. No entanto, as APIs baseadas em memória consumem em média quatro a cinco vezes mais memória do que o tamanho do documento a processar. Por exemplo, um documento de 20 *megabytes*, necessita em média (dependendo do modelo de representação) de 100 *megabytes* para ser representado em memória, podendo representar um obstáculo no processamento de documentos de grande dimensão.

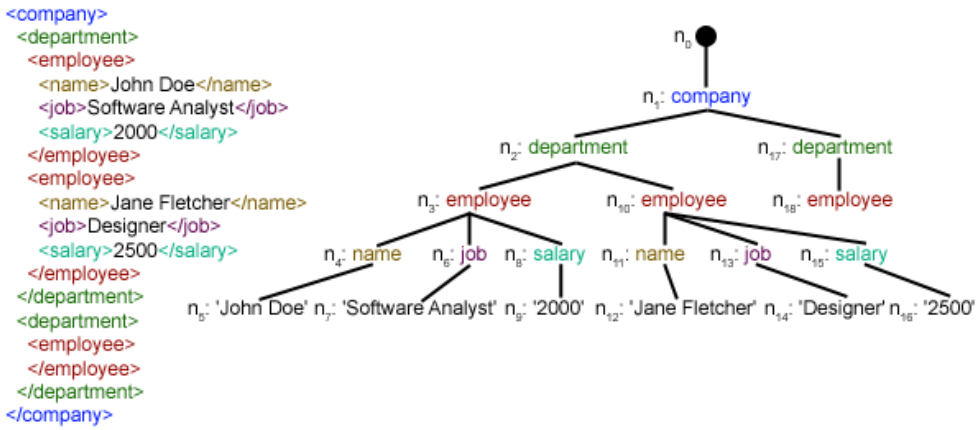


Figura 11 – Exemplo de representação em árvore de um documento XML em memória [12]

De modo a obter uma visão geral sobre o estado da arte das APIs de processamento em memória de documentos XML em Java, foi realizado um estudo de algumas APIs de forma a avaliar qual a API mais eficiente na gestão da memória e a API que permite uma execução mais rápida de um documento XML. Foram estudadas as seguintes APIs: DOM[13],XOM[14], OJXQI[15], VTD[16], jDOM[17], dom4j[18] e Xerces2[19].

Como parte integrante do pacote JAXP, a API DOM é uma colecção de classes que possui um conjunto de métodos Java que permitem a leitura de um documento XML para uma estrutura de representação semelhante à estrutura da Figura 11, assim como todas as restantes APIs baseadas em memória. Apesar de similar à API DOM, a API jDOM implementa um modelo alternativo que não foi construído nem modelado a partir do DOM. De forma a tirar partido das características da linguagem Java como criação de métodos com o mesmo nome, colecção de APIs existentes, *refletion*[20] e *weak references*[21], a API jDom foi desenvolvida especificamente a pensar na linguagem Java, ao contrário de outras APIs, que são desenvolvidas de forma genérica tendo como alvo várias linguagens de programação.

Um método de extracção de dados de um documento XML já mencionado foi a linguagem XQuery, permitindo a criação de um código de alto nível para a extracção de elementos, à semelhança do que acontece com a linguagem SQL para as bases de dados relacionais. Para a utilização da linguagem

[12] http://dbis.informatik.uni-freiburg.de/index.php?project=GCX/example.php
[13] http://download.oracle.com/javase/6/docs/technotes/guides/xml/index.html
[14] http://www.xom.nu/
[15] http://www.oracle.com/technetwork/database/features/xmldb/index-087544.html
[16] http://vtd-xml.sourceforge.net/
[17] http://www.jdom.org/
[18] http://dom4j.sourceforge.net/dom4j-1.6.1/
[19] http://xerces.apache.org/xerces2-j/
[20] http://java.sun.com/developer/technicalArticles/ALT/Reflection/
[21] http://weblogs.java.net/blog/2006/05/04/understanding-weak-references

XQuery será necessário que a API possua suporte nativo da linguagem isto é, que incorpore um motor capaz de interpretar os comandos enviados. A OJXQI (*Oracle Java Xquery API*) é uma API, incorporada nas bases de dados da Oracle, com suporte para a linguagem XQuery, facilitando em muito a aplicação das transformações, que em alguns casos são muito semelhantes em termos de construção aos comando SQL utilizados no processo de ETL, como demonstra a Figura 12.

```
for $i in doc("SalesOrderDetail.xml")/SalesOrderDetail/AdventureWorks2008.Sales.SalesOrderDetail
where $i/SalesOrderID/text() = '43659' and $i/OrderQty/text() > 1
return $i
```

Figura 12 – Exemplo de aplicação de XQuery para a extracção de dados de um documento XML

Actualmente o grande problema das APIs baseadas em memória passa pela criação em memória de objectos que resultam do processo de extracção. A premissa da API VTD passa por optimizar a criação de objectos com um tamanho fixo de números inteiros, representando dados em inteiros de 64 bits (VTD records). O *Virtual Token Descriptor* (VTD) é um formato binário que especifica como atribuir *tokens* (códigos de identificação) aos de uma forma não extractiva. O conceito de *parsing* "Não extractivo" [38] significa que o texto do XML se mantém intacto em memória enquanto os *tokens* são representados exclusivamente utilizando intervalos e tamanhos em *bits* (o conteúdo das *strings* não é copiado)(Figura 13).

O processo contrasta com o método extractivo utilizado por outros modelos de processamento XML (como a DOM e SAX), onde são alocados blocos de memória para proceder à alocação de conteúdos do documento, realizando uma manipulação directa dos dados.

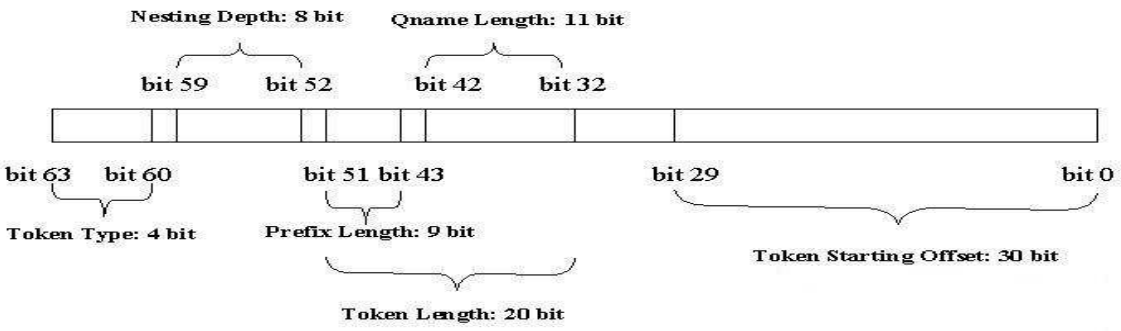


Figura 13 - Representação de um esquema de um registo segundo a API VTD[22]

De modo a analisar a performance de cada uma das APIs no tempo de execução e na gestão de memória do processamento de documentos XML, foi executado o *parsing*, ou seja a leitura, de um excerto de um documento, com o tamanho de 58,0 *megabytes*, simulando as ordens de venda de produtos (DetalhesOrdemVenda.xml), que foi retirado do *Data Warehouse* exemplo da *Microsoft*

[22] http://vtd-xml.sourceforge.net/VTD.html

Adventure Works. A Tabela 7 apresenta um resumo desse estudo, com medições da memória em *megabytes* e tempo de execução em milissegundos utilizado por cada uma das APIs para a execução do documento referido.

Ficheiro	Memória utilizada	Tempo de execução
DOM	262 MB	7156 ms
XOM	193 MB	10690 ms
OJXQI	160 MB	7601 ms
VTD	89 MB	1470 ms
jDOM	327 MB	14330 ms
dom4j	215 MB	8618 ms
Xerces2	290 MB	12216 ms

Tabela 7 – Resumo do estudo das APIs baseadas em memória[23]

Os resultados obtidos baseiam-se numa média aritmética resultado de cinco execuções e cada uma das APIs para o respectivo documento, e demonstram claramente a superioridade da API VTD relativamente às restantes APIs, seja no consumo de memória ou no tempo de execução, provando que o modelo de representação de dados que a API VTD implementa é bastante superior relativamente ao das restantes APIs.

4.1.2 APIs baseadas em leitura em modo *streaming*

As APIs baseadas em modo *streaming* utilizam uma leitura sequencial do documento utilizando recursos mínimos de memória para o processamento do documento. Tipicamente este tipo de APIs utilizam a profundidade do documento XML (número de elementos aninhados no documento) e o máximo de dados armazenados nos atributos de um único elemento para calcular a quantidade de memória mínima a utilizar. De seguida o documento é lido e é apenas carregado para memória uma determinada porção.

O documento XML é lido por um *parser*, que identifica cada elemento XML no momento em que é encontrado, fazendo uma chamada para um método específico enquanto o documento é processado.

A Figura 14 apresenta o modelo conceptual de funcionamento da API SAX para o processamento de um documento XML, que acaba por ser transversal às restantes APIs baseadas em leitura em modo *streaming*.

De uma forma resumida, o *parser* é configurado como uma fonte de entrada ficando associado a um conjunto de métodos de gestão de conteúdo que identificam, por exemplo, o início e fim de documento, elementos, dados dentro de elementos e erros que possam ocorrer durante a leitura do

[23] Testes realizados num Intel Core 2 Duo T5500 1.66GHZ, 2.0 GB DDR2 667 MHZ, Ubuntu 11.04, java version 1.6.0_22 - OpenJDK Runtime Environment com 455 megabytes de memória disponível

documento. Quando o *parser* é executado dispara eventos que são capturados pelos métodos de gestão de conteúdo. Cada vez que o *parser* detecta uma parte importante do documento XML, dispara o método apropriado, de forma a tornar possível a leitura do respectivo bloco de dados.

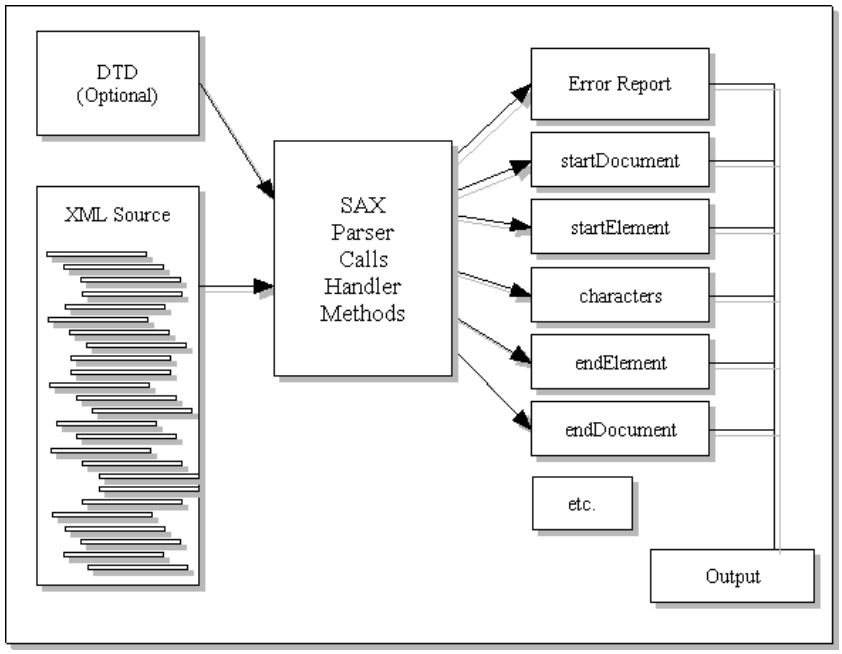


Figura 14 – Modelo conceptual da API SAX[24]

As APIs baseadas em leitura em modo *streaming* são mais adequadas para tratamento de documentos XML de grande dimensão, porque teoricamente conseguem processar documentos de tamanho infinito.

Com base nesta metodologia, as APIs analisadas foram a SAX[25], StAX[26] e a API XOM com a sua implementação baseada em leitura em modo *streaming*. As APIs SAX e StAX integram o pacote JAXP, que é incluído na distribuição do *software* Java.

O JSR (*Java Specification Requests*) 173 define a *Streaming API for XML* (StAX), que permite aceder aos elementos por modo *streaming*, permitindo extrair a informação através dos eventos controlados pela aplicação, diferenciando-se da API SAX que possui um gestor que recebe eventos conforme a conveniência do *parser*. Enquanto que a API StAX permite descartar informação do *parser* conforme as necessidades, na API SAX, o *parser* extrai a informação para aplicação não permitindo o controlo por parte da aplicação. Assim, através da API StAX é possível controlar a informação a analisar, controlando o processo de *parsing*. Para além disso, a API StAX possui duas APIs integradas com níveis de abstracção diferentes: a *cursed-based* API, que é uma API de mais baixo nível que trabalha como um *stream* de eventos; e a *Iterator Based* API que oferece um nível de abstracção superior permitindo processar documentos XML como uma série de eventos de objectos, onde cada objecto

[24] Fonte da imagem: http://www.inf.ufrgs.br/gppd/disc/inf01008/trabalhos/sem01-1/t2/apis_xml_java/
[25] http://www.saxproject.org/
[26] http://stax.codehaus.org/Home a implementação utilizada da API StAX foi a cursor API

comunica uma parte da estrutura do documento XML à aplicação. Para os testes de performance foi considerada a abordagem *cursor-based*.

Uma vez que a memória consumida por este tipo de APIs é reduzida, não representando um ponto crítico do seu funcionamento, foi apenas testada a velocidade de *parsing* (leitura do documento) em milissegundos de cada uma das APIs: Sax, StAX e XOM (Tabela 8) para o mesmo documento utilizado anteriormente (DetalhesOrdemVenda.xml).

Ficheiro	StAX	Sax	XOM
SalesOrderDetail	1271 ms	1275 ms	4611ms

Tabela 8 – Teste de velocidade do *parsing* das APIs baseadas em leitura em modo *streaming*

As APIs StAX e Sax são muito similares no tempo de *parsing*, o que é facilmente expectável, uma vez que o principal ponto que diferencia as duas APIs é a forma como o *parser* controla os eventos processados. Tendo em consideração apenas a leitura de todo o documento, então os resultados são bastante idênticos. A API XOM tem uma performance bastante inferior às restantes APIs.

Para o processamento de documentos simples, e na maioria das áreas de aplicação, as limitações destas APIs não são muito relevantes. Para o trabalho em questão, onde há uma necessidade de tratamento a documentos estruturados em XML de grande dimensão, facilmente se conclui que as APIs baseadas em memória, sobre um ponto de visto superficial, são inadequadas para a realização das tarefas de um processo de ETL, devido à limitação de memória. No entanto é importante estudar estas APIs com o objectivo de verificar as suas vantagens e até a sua aplicação em determinados cenários, de modo a melhorar a performance de execução do processo de ETL segundo o modelo apresentado.

As operações que exigem uma troca de ordem de elementos num documento estruturado em XML, como a operação de ordenação, ou operações que necessitem de realizar um número elevado de comparações entre duas fontes de dados (como é o caso de operações de junção, diferença e produto cartesiano) são operações cujas características de execução se encontram alinhadas com as caraterísticas deste tipo de APIs. Se os dados necessitam de um tratamento não sequencial, então a utilização da memória é um factor crucial de performance, ao invés da realização de várias operações de leitura e escrita em disco para a realização da mesma operação, uma vez que os tempos de acesso a disco são mais elevados que os acessos a memória.

4.1.3 Análise global das APIs apresentadas

Uma vez identificadas as APIs (Figura 9) e a sua metodologia de representação, é importante definir, para o trabalho em questão qual a metodologia mais indicada.

As APIs baseadas em memória são amplamente mais utilizadas em diversas áreas, devido ao facto de na maioria dos casos os documentos a processar serem de pequena dimensão. No entanto para os

casos em que a disponibilidade em memória é comprometedora, ou então o tamanho dos documentos XML a processar são de dimensão elevada (100 *megabytes* ou 1 *gigabyte* por exemplo) as APIs baseadas em leitura em modo *streaming* são as mais indicadas.

API	Modo de leitura
JAXP: Sax	Modo *streaming*
JAXP: StAX	Modo *streaming*
JAXP: DOM	Baseado em memória
XOM	Modo *streaming* e baseado em memória
OJXQI	Baseado em memória
VTD	Baseado em memória
jDOM	Baseado em memória
dom4j	Baseado em memória
Xerces2	Baseado em memória

Tabela 9 – Resumo das APIs analisadas

A escolha de uma API para a codificação das operações tem se ter tomada mediante os requisitos de codificação e do ambiente de execução do trabalho apresentado. Tendo em consideração o grande volume de dados trabalhados num processo de ETL, é facilmente expectável que os documentos a processar facilmente atinjam valores na ordem dos gigabytes. Desta forma e sobre um ponto de vista superficial as APIs baseadas em memória seriam facilmente descartadas, o que não acontece tendo em consideração as vantagens deste tipo de APIs.

No contexto de AR é possível identificar dois tipos de operações:

1. As operações cuja transformação sobre documentos estruturados em XML pode ocorrer à medida que a leitura do documento é realizada, isto é, não é necessário realizar comparações entre elementos do mesmo documento ou de outro documento, nem é necessário modificar a ordem como os elementos surgem fisicamente no documento.
2. As operações que exigem a troca de posição de elementos na hierarquia do documento original ou então que necessitem de realizar múltiplas comparações com elementos do mesmo documento ou então de outro documento.

Por exemplo a projecção pode ser considerada uma operação do tipo 1, na medida que não é necessário alterar a hierarquia do documento para a projecção de elementos. A operação de ordenação e a operação de diferença por exemplo podem ser consideradas do tipo 2. A operação de ordenação necessita de alterar a estrutura de hierarquia, de modo a que as alterações possam ser materializadas,

enquanto que a operação de diferença necessita de múltiplas comparações entre elementos dos documentos que participação na operação.

Ambas as APIs conseguem lidar com este tipo de operações, no entanto a diferença de performance em cada um dos tipos de operações nas diferentes APIs varia.

Na Tabela 10 é possível verificar que as APIs baseadas em leitura em modo *streaming* são as mais indicadas para operações do tipo 1, tendo em consideração que o tempo de execução destas APIs é bastante inferior ao tempo de execução das APIs baseadas em memória. Este fenómeno ocorre essencialmente devido ao custo de recriar em memória toda a estrutura de documento, o que implica um tempo de execução mais elevado. Só após a correcta representação do documento em memória é que possível iniciar o processamento propriamente dito. Como as APIs baseadas em memória não possuem esse procedimento iniciam de imediato a transformação, obtendo o resultado da operação mais rapidamente e com recursos computacionais inferiores.

Operação/API	**StAX**	**Sax**	**VTD**
Projecção	0.070 m	0.084 m	0.17 m
Diferença[27]	156 m	192 m	21,3 m

Tabela 10 – Teste de performance para dois tipos de operação[28]

Para operações do tipo 2, onde se incluem por exemplo as operações de diferença e produto cartesiano, as APIs baseadas em memória são mais rápidas que as APIs baseadas em modo *streaming*.

$$T_1 \leftarrow R - S \quad (1)$$

Para realização de uma operação de diferença entre um documento R e um documento S (1), será necessário subtrair o conteúdo de um documento S ao conteúdo de um documento R. Ou seja para obter o documento resultado da operação de diferença será necessário para cada elemento de R verificar se este se encontra no conteúdo do documento S. Com a utilização das APIs baseadas em modo *streaming* será necessário realizar um número elevado de operações de leitura e operações em disco, porque para cada elemento de um documento R será necessário realizar uma operação de leitura de todo o documento S (no pior cenário). No caso das APIs baseadas em memória será necessário representar em memória os dois documentos e uma vez representados, todas as comparações são realizadas em memória, reduzindo o tempo de execução.

No entanto as APIs baseadas em memória possuem uma limitação associada ao tamanho do documento que conseguem representar em memória. Se pensarmos nas operações que operam sobre

27 Para a operação de diferença foi considerado o documento DetalhesOrdemVenda e uma cópia deste para os dois documentos da diferença, originando um documento vazio.
28 Medições representadas em minutos

mais do que o documento então o tamanho dos dados que estas APIs podem processar reduz-se drasticamente.

De acordo com os resultados obtidos, a API StAX será o principal foco no desenvolvimento das transformações devido a suportar o processamento de documentos de grande dimensão. No entanto é pertinente utilizar uma implementação das operações baseadas em memória utilizando a API VTD para a optimização de todo o fluxo de operações. Desta forma sempre que os documentos possam ser representados em memória e os requisitos das operações sejam compatíveis com as características destas APIs, as operações implementadas segundo a metodologias das APIs baseadas em memória são aplicadas, permitindo assim reduzir o tempo de execução de todo o processo de ETL.

Por exemplo, se para a operação de diferença o tamanho dos documentos permitir a sua representação em memória então a operação codificada na API VTD poderá ser utilizada, caso contrário as operações codificadas na API StAX são as mais indicadas. A utilização da operação de *split* de documentos XML em documentos de menor dimensão por forma a explorar as operações codificadas na API VTD é também um dos pontos considerados neste trabalho, com o desenvolvimento de operações complementares que permitem a divisão (*split*) e a união (*merge*) de documentos XML.

4.1.4 Codificação das operações equivalentes às operações de AR

Uma vez analisados os mecanismos que permitem a transformação de documentos XML utilizando a linguagem Java, e identificados os pontos fortes de cada uma das abordagens, é necessário codificar as operações segundo as definições das operações equivalentes às operações de AR.

A grande maioria das operações de AR apresentadas na Tabela 4 e Tabela 5 foram implementadas recorrendo à API StAX. Foram implementadas as cinco operações base de AR, juntamente com a maior parte das operações de AR extendida: *theta join*, renomeação, junção à esquerda, criação de grupos com a aplicação das operações de agregação e ordenação.

Assim como é necessário definir o comportamento das operações de AR, recorrendo a predicados e expressões, também será necessário especificar para cada uma das operações desenvolvidas o comportamento das mesmas. Por exemplo, indicar numa operação de projecção quais são os atributos a projectar do documento original, de modo a construir um documento resultado que reflita a aplicação da transformação. Desta forma, por exemplo, para uma operação de projecção será necessário definir num documento estruturado em XML os elementos do documento a projectar. A operação de transformação apenas executa se receber por parâmetro: o documento que especifica o comportamento da operação, o documento a transformar e a localização para o armazenamento do documento resultado.

As operações binárias (3) diferenciam-se das operações unárias (2) devido à necessidade de existência de dois documentos para a realização da transformação de modo a gerar o documento resultado.

Sempre que num documento de definição de uma operação binária seja necessário referir elementos dos documentos que alimentam a operação, este atributo deverá ser precedido de um prefixo com o nome do documento a que o atributo se refere.

$$T_2 \leftarrow \pi_{(IDMorada)}(P.Morada) \quad (2)$$

$$T_3 \leftarrow Pessoa \bowtie Morada \quad (3)$$

As operações de união, diferença e produto cartesiano são as únicas operações que não necessitam de documento de definição de operação, recebendo directamente como parâmetro a localização dos documentos a processar.

A operação de projecção extendida foi decomposta numa operação de renomeação de elementos e uma operação de adição de colunas. A operação de adição de elementos permite adicionar:

- colunas com o resultado de operações divisão, multiplicação, soma e subtração sobre outros elementos;
- valores atribuídos estaticamente;
- data actual de sistema;
- inserção de valores auto incrementais nos elementos;
- replicação de colunas;
- concatenação de dados de mais do que uma coluna.

Algumas operações permitem utilizar outras operações de forma a optimizar o seu tempo de execução como é o caso das operações *thetajoin* e *leftjoin*, que caso seja especificado, execução a operação de ordenação sobre os dados antes de realizar a junção, de forma a aumentar a velocidade de execução da operação.

Adicionalmente, foram implementadas operações, que embora não façam parte de AR, simplificam e complementam o modelo apresentado: a operação de *split* que permite dividir um documento XML em partes iguais ou então por percentagem de divisão; a operação de *merge* permite realizar a união de vários documentos recebendo como parâmetro a localização ou um documento XML com a lista dos documentos a realizar a junção; e a operação de replicação de um documento XML. As operações de junção, junção à esquerda e ordenação foram também codificadas para a API VTD de forma a tirar partido da execução de transformações recorrendo à representação de dados em memória.

Como já foi referido, com as excepções da operação de união, diferença e produto cartesiano, todas as restantes operações possuem um documento de definição de operação que é gerado com base na definição da operação indicada no artefacto de anotação da notação BPMN da Figura 9. O utilizador

apenas necessita de preocupar a definição da modelação lógica, na medida que a especificação física do modelo é gerada de uma forma automatizada.

4.2 Análise do Modelo Lógico e Físico

Ao longo do presente documento tem sido apresentado um excerto de um processo de ETL para a alimentação de uma dimensão dim_Pessoa. Recorrendo às modelações propostas por Vassiliadis [2], Trujillo [7] e Akkoiu [6], foi realizada a modelação do processo de ETL mencionado em cada uma das notações de modo a identificar vantagens e desvantagens das propostas apresentadas. Com base nesse estudo foi proposta uma notação de modelação lógica onde o ambiente de execução não tem necessariamente de ser um SDW convencional.

O principal objectivo do modelo apresentado na Figura 7 passa por permitir a modelação de um processo de ETL independentemente da infraestrutura de execução, através da decomposição das transformações em operações elementares, recorrendo às operações de AR que permitem a decomposição num conjunto de operações mais básicas, proporcionando assim a criação de uma árvore de operações onde é representado o respectivo fluxo de execução do processo de ETL.

Cada nodo da árvore representa uma única operação, que recebe como *input* os dados o documento resultado transformado pela operação anterior. Um nodo pode desencadear transformações que podem ser executadas em paralelo (não existindo dependências entre si). No entanto para que a transformação de um determinado nodo possa iniciar a sua execução é necessário que todas as operações que convergem nesse nodo terminem.

Identificadas as fraquezas deste modelo, essencialmente devido à inexistência de ferramentas gráficas de modelação e à necessidade de representação da especificação das operações numa tabela complementar, foi apresentada a conversão do modelo para a notação BPMN, amplamente utilizada na modelação de processos de negócio. Desta forma é possível modelar todo o processo de ETL e definir o comportamento de cada operação recorrendo a anotações, exportando posteriormente toda a definição do diagrama para um documento XPDL[29], permitindo a sua posterior interpretação em qualquer linguagem de programação.

O formato XPDL é um formato estandardizado que permite definir um esquema XML (XSD) para a especificação de *workflows* ou processos de negócio. O XPDL foi desenhado para suportar a definição de um processo, tanto a nível de gráficos como de semântica de um processo, tornando-o no melhor formato para a definição de diagramas em BPMN, facilitando assim a transposição das características de um *workflow* entre uma variedade de aplicações.

29 http://www.xpdl.org/

Devido a este suporte é possível mapear o modelo lógico de representação de um processo de ETL para primitivas de programação. Recorrendo a um aplicativo desenvolvido no âmbito do presente trabalho é possível gerar o documento de representação física baseado na definição XPDL gerada através da notação BPMN, convertendo todo o fluxo de dados. A especificação do comportamento de cada uma das operações é gerada num documento estruturado em XML para cada uma das operações. Neste novo documento é descartada a informação gráfica gerada pela ferramenta de modelação. São também definidas características próprias de implementação física como por exemplo o nome dos executáveis Java que têm de ser invocados para a realização de cada operação.

Em suma, para a construção do modelo físico é necessário:

- identificar toda a sequência de operações;
- gerar todos os documentos de entrada para as operações de transformação codificadas na linguagem Java;
- identificar os documentos de armazenamento final e intermédio(opcional);
- identificar a estrutura de pastas gerada, de forma a suportar todos os ficheiros temporários do processo.

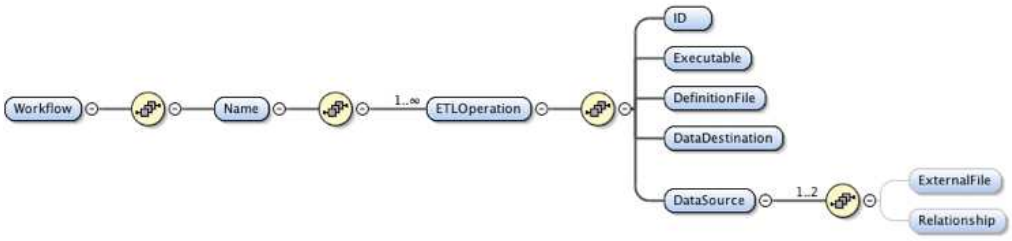


Figura 15 – Esquema XSD para a representação física de um processo de ETL

Com base no esquema XSD da Figura 8, é apresentado na Figura 15 o esquema de representação do documento XML de execução. Neste modelo continuam a ser representadas todas as operações, no entanto para cada operação é definido:

- *ID*, que permite identificar a operação.
- *Executable*, que permite especificar o nome e localização do executável Java para a respectiva operação.
- *DefinitionFile*, que permite especificar o nome e localização do documento com a definição da respectiva operação de transformação.
- *DataDestination*, que similarmente ao Modelo Lógico permite identificar qual o nome e localização do documento que irá armazenar o conjunto de dados resultantes da aplicação da transformação (útil para armazenamento de resultados intermédios).

- *DataSource*, que engloba as tags: *ExternalFile* que permite especificar a localização do documento de entrada, permitindo identificar que a operação vai ser alimentada por um documento externo; *Relationship* que permite identificar o *ID* da operação precedente que vai fornecer o documento de entrada à operação de transformação actual. No máximo só poderão existir duas fontes de entrada (operações binárias) e poderão de ser de diferente natureza, isto é, uma das fontes de entrada poderá ser um documento externo e a outra fonte o resultado do processamento de uma outra operação.

Recorrendo a aplicações codificadas na linguagem Java é possível executar as devidas transformações sobre dados armazenados sobre o padrão de armazenamento XML, permitindo uma execução e representação de dados completamente independente de infraestrutura.

O processo de definição do processo de ETL é simples, com o utilizador apenas a especificar graficamente todas as transformações a ocorrer e a sua sequência, com todo o processo de criação do modelo físico a ser gerado de uma forma automatizada.

Uma vez definido todo o processo de execução será necessário definir um ambiente de execução para o processo gerado. A notação utilizada, linguagem de codificação das operações e o suporte ao armazenamento dos dados são tecnologias que permitem a independência de execução de todo o modelo, tornando viável a execução de todo o processo de ETL em vários tipos de infraestrutura, como o de GRIDs computacionais.

4.3 Descrição do ambiente computacional utilizado para o processamento do processo e ETL

O desenvolvimento científico, é fortemente baseado em computação, e a capacidade de processamento dos computadores actuais, apesar da sua evolução exponencial, continua a revelar-se insuficiente para os objectivos pretendidos pela ciência. Como tal a necessidade de partilhar recursos para atingir um objectivo comum tornou-se uma estratégia fundamental tanto na ciência como a nível comercial.

De modo a obter e aumentar o poder computacional através de vários recursos computacionais, recorreu-se a uma metodologia de trabalho cooperativo, através da distribuição de tarefas a máquinas que se encontram conectadas por um rede de um ou mais computadores, permitindo um trabalho conjunto de uma forma transparente e coerente, como se de uma única máquina se trata-se.

O conceito de GRID computacional assenta na base da cooperação entre máquinas, ou seja, várias máquinas contribuem com os seus recursos para que no conjunto das mesmas, os recursos agregados consigam oferecer garantias em termos de performance e disponibilidade. Estes recursos fornecidos à GRID, podem ser locais, ou recorrendo à internet, abranger uma grande área geográfica. Este tipo de

cooperação em grande escala possibilita um enorme potencial em termos computacionais e permite formas de trabalho conjuntas que à alguns anos atrás se considerava impossível.

Um ponto explorado pelo paradigma de GRID, é que as máquinas que um utilizador comum utiliza para as suas tarefas, possuem na sua maioria recursos em excesso, em relação com o que os utilizadores realmente necessitam. Em diversas situações, apenas é utilizada uma pequena parte da capacidade da máquina, o que implica que grande parte da capacidade da máquina se encontre desperdiçada. É também neste desperdício computacional que a computação em GRID tenta obter benefícios, utilizando recursos que a máquina não necessita para uma determinada função que esteja a desempenhar num dado instante. Este tipo de aproveitamento faz com que as máquinas que integram a GRID possuam uma grande heterogeneidade, ou seja, ao contrário de um *cluster*[30], as máquinas não existem única e exclusivamente para servir o paradigma em que estão inseridas, podendo realizar as tarefas comuns que os utilizadores necessitam, mas aproveitando sempre que possível os recursos não explorados [39].

A heterogeneidade do conceito de GRID não se limita ao tipo de tarefas que as máquinas podem processar, mas também ao tipo de ambientes em que estas operam, ou seja, existe uma abstracção de arquitectura e *software* existente em cada máquina, fazendo com que independentemente das suas características cooperem em conformidade sobre uma tarefa em comum.

As implementações de uma GRID normalmente funcionam através da introdução de uma nova camada de software entre as aplicações e a infraestrutura "física" composta pelo computador, sistema operativo e redes. Assim os utilizadores interagem com os recursos computacionais através desta camada de *software* que esconde a complexidade e diversidade da infraestrutura, oferecendo uma interface uniforme para acesso aos recursos disponibilizados. Esta camada de *software*, esquematizada na Figura 16, é designada de *Middleware*.

O *Middleware* de uma GRID possui um componente chamado de gestor de tarefas que é o componente responsável pela distribuição de tarefas pelos nodos computacionais da GRID, tendo por base a disponibilidade e capacidade computacional de cada nodo numa determinada janela temporal.

[30] Um cluster é formado por um conjunto de computadores, que utiliza um tipo especial de sistema operativo. Muitas vezes é construído a partir de computadores convencionais (*personal computers*), que são ligados em rede e comunicam através do sistema, trabalhando como se fossem uma única máquina de grande porte.

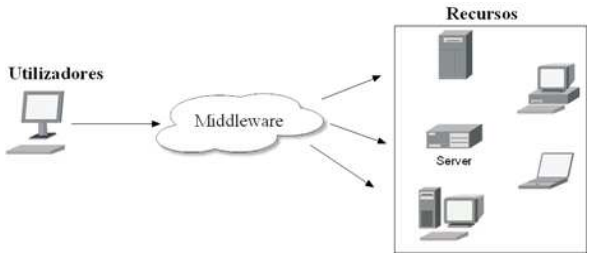


Figura 16 - Interacção do Middleware com uma Grid

O gestor de tarefas tem como objectivo aproveitar os ciclos não utilizados de processamento de um conjunto de máquinas, dividindo os trabalhos pelos recursos disponíveis de acordo com os critérios definidos pelo utilizador (tipo de plataforma, memória, etc), permitindo a submissão de vários trabalhos em simultâneo. Recorrendo ao gestor de tarefas é possível definir critérios de utilização de máquinas, como por exemplo definir o tempo de inactividade de uma máquina, para que só após esse período se encontre disponível para a execução de trabalhos. Outra característica importante é a detecção de falhas na execução de trabalhos. Se por alguma razão uma máquina ficar indisponível, os trabalhos têm de ser automaticamente migrados para uma outra máquina disponível na GRID.

Tipicamente um gestor de tarefas começa por obter informação sobre a disponibilidade dos recursos e com base nesses dados identificar se o recurso cumpre os requisitos mínimos de execução pré-estabelecidos pelo utilizador – Sistema operativo, *software* e configurações de *hardware*. De seguida, o processo de selecção tem como principal objectivo seleccionar os recursos mais apropriados para a submissão de trabalhos. Por fim, devem ser executadas um conjunto de tarefas com o objectivo de executar devidamente os recursos, como: reserva de recursos na GRID, preparação da submissão de trabalhos, monitorização de trabalhos e finalmente tarefas de limpeza resultantes da execução dos trabalhos.

Segundo Ben Segal [40], o escalonamento de tarefas deve ter em consideração a decomposição e distribuição de trabalhos baseado na disponibilidade e proximidade da capacidade computacional e dos dados necessários, tendo em consideração que normalmente são operações críticas devido ao facto de a informação necessitar de ser transferida entre os vários recursos computacionais sobre uma largura de banda limitada.

A existência de uma necessidade de obtenção de poder de processamento, como o caso do processamento de processos de ETL, aliado a uma infraestrutura de rede robusta com recursos computacionais não utilizados ou subaproveitados, a utilização de GRIDs computacionais apresenta-se como uma solução viável.

Para a execução e interpretação do documento XML de definição física de execução do processo de ETL descrito no ponto 4.2, o gestor de tarefas terá de ser capaz de interpretar o documento limitando-se a seguir o fluxo de operações especificado.

O gestor de tarefas terá a responsabilidade de analisar a disponibilidade da GRID, avaliar quais são as máquinas disponíveis que se encontram melhor colocadas para a realização do trabalho e mediante uma previsão da complexidade do trabalho, realizar a sua distribuição pelos nodos computacionais mais adequados. Relativamente aos documentos que representam os resultados intermédios do processo de ETL que surgem como resultado da execução de cada operação, será o gestor de tarefas a gerir toda a sua criação, desde a sua localização até à atribuição do nome, desde que no documento XML de especificação física não esteja especificada nenhuma indicação para que um determinado resultado intermédio resultante de uma operação, tenha de ser armazenado permanentemente sobre determinadas condições.

Adicionalmente o gestor de tarefas poderá dividir os documentos de entrada das operações em documentos de menor dimensão, desde que não exista dependência entre eles, utilizando as operações de *split* e *merge* de documentos XML codificadas e apresentadas anteriormente, permitindo assim ciclos de execução mais curtos nos nodos computacionais, o que pode beneficiar a GRID, mediante a disponibilidade dos nodos numa determinada janela temporal, permitindo tirar partido das operações codificadas seguindo um modelo de representação de uma API baseada em memória.

Em suma, podemos resumir todo o trabalho realizado no âmbito desta dissertação com um único diagrama de acções (Figura 17). Nele podemos observar as principais etapas do processo de transformação, onde são definidos os requisitos do processo, definidas as operações a realizar, construção do modelo lógico utilizando a linguagem BPMN, e o processo de construção dos documentos XML de representação do modelo lógico e físico do processo, de modo a proporcionar a sua execução.

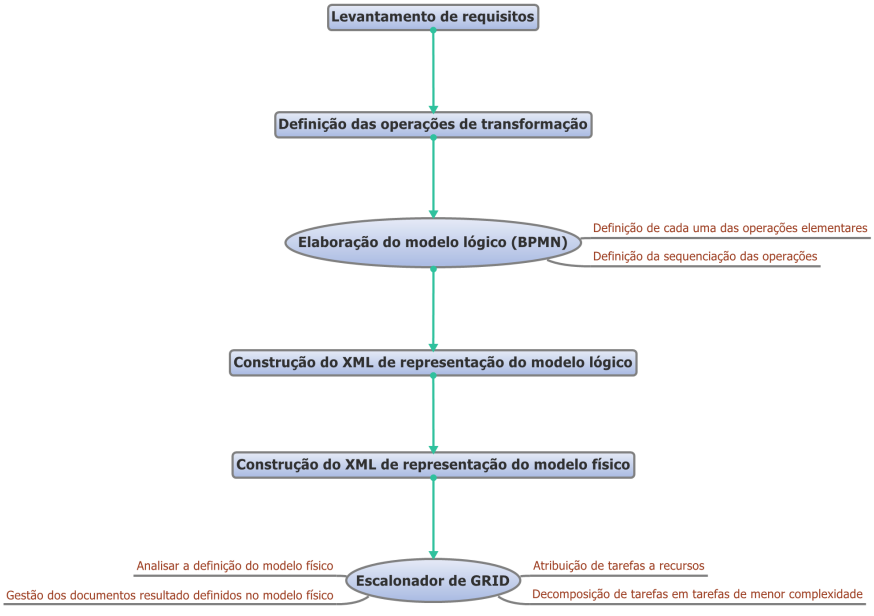


Figura 17 – Resumo das principais etapas do processo de transformação

Capítulo 5

Conclusões e trabalho futuro

5.1 Discussão de resultados

O processo de alimentação de um SDW (ETL) é necessariamente um processo complexo, tendo em conta a elevada quantidade de dados a processar, resultado da complexidade crescente dos requisitos de negócio e essencialmente pela necessidade na análise de dados por parte das organizações para suportar o apoio à tomada de decisão. Se aliarmos a estes factores a complexidade computacional e o tempo despendido nesse processamento, especialmente quando a janela de oportunidade é limitada, estamos perante um processo crítico que influencia o próprio sucesso de implementação de um SDW.

Tendo em consideração a importância do processo de ETL no contexto de um SDW e à sua necessidade de modelação, é possível verificar que existe uma enorme lacuna na definição de *standards* para a modelação conceptual, lógica e física de um processo de ETL. Verificou-se também que muitas das actuais soluções de modelação, baseiam-se em notações e modelos de representação do fluxo de operações proprietários que em nada beneficiam a portabilidade entre as ferramentas de modelação e implementação de processos de ETL. Estas ferramentas implementam uma metodologia integrada, em que a componente de modelação se encontra segundo a metodologia implementada, impossibilitando a migração entre infraestruturas de execução.

O trabalho descrito ao longo do presente documento, teve como objectivo a proposta de uma solução alternativa de execução de processos de ETL em vários tipos de infraestrutura, através da definição de uma proposta de modelação lógica, que recorrendo a AR permite a decomposição das transformações presentes de um processo de ETL em operações elementares. Com esta proposta, é pretendida a redução da distância desde a criação do modelo conceptual do processo de ETL até à sua

implementação física, permitindo o mapeamento do modelo lógico apresentado para primitivas de programação, assumindo uma representação independente da infraestrutura física para a execução de processos de ETL.

5.1.1 Modelação de processos de ETL

Ao longo dos anos têm sido apresentados propostas de modelação por parte de diversos autores. Vassiliadis [4] defende uma proposta de modelação conceptual e lógica própria para a modelação de processos de ETL que apesar de completa acabou por não ser adoptada pelas ferramentas comerciais devido à sua complexidade de compreensão para processos de ETL de grande dimensão. Uma solução para a resolução deste problema passa pela subdivisão do processo de ETL de forma a criar uma separação em várias componentes que permitam uma visão fraccionada, facilitando assim a sua leitura. Um outro problema associado à proposta de Vassiliadis, refere-se à ausência de mecanismos que permitam o mapeamento da notação para primitivas de programação, o que limita a sua implementação física. Por sua vez, Trujillo [7] apresenta uma proposta de modelação conceptual baseada na linguagem UML, que apesar de simples e de fácil compreensão, agrava um dos problemas associados à notação proposta no Vassiliadis: a complexidade do modelo associada a processos de ETL minimamente complexos. A imperceptibilidade do modelo refere-se essencialmente ao excessivo recurso a anotações para especificar as operações, tornando a proposta de modelação realizado por Trujillo inadequada. Por sua vez Akkaoui [6], recorre à notação BPMN, amplamente utilizada na modelação de processos de negócio, para apresentar a sua proposta de modelação conceptual. Na proposta de modelação apresentada, Akkaoui permite não só especificar que operações executar, mas também a sua ordem de execução, permitindo, por exemplo, definir políticas de tratamento de falhas. Assim, o modelo proposto, apresenta-se como uma solução susceptível de ser utilizada no processo de modelação lógica de um processo de ETL.

Com base na proposta de modelação de Akkaoui é apresentada uma proposta de modelação para o modelo lógico de um processo de ETL utilizando os conceitos e especificidades das operações de AR. Desta forma é possível proceder à decomposição das tarefas comuns de um processo de ETL em operações elementares, construindo um grafo de operações que representa a sequência de fluxo de todo o processo de ETL, possibilitando o seu mapeamento para primitivas que permitam a sua implementação e execução em ambientes de execução variados.

A especificação apresentada de modelação lógica baseada em AR, juntamente com a sua conversão na notação BPMN, parece ser uma solução perfeitamente viável, devido não só à sua simplicidade de representação mas essencialmente devido ao suporte existente na conversão de modelos BPMN em primitivas de programação. Recorrendo ao formato de armazenamento XML, é possível representar todas as características do modelo num formato que permita a sua implementação, independentemente da infraestrutura de suporte aplicacional.

Desta forma é possível realizar toda a modelação lógica do processo de ETL recorrendo a operações elementares de transformação (directamente relacionadas com AR), permitindo execução de transformações sobre um conjunto de dados, sem a necessidade de especificar a infraestrutura que irá suportar o processo. Com base neste desconhecimento da infraestrutura de execução, é facilmente perceptível que a implementação física irá necessariamente ter de suportar o armazenamento de dados num formato universal, com o suporte aplicacional às transformações a ter de ser suportado por outros mecanismos, de modo a permitir transformações similares às existentes num ambiente de SDW.

5.1.2 Estudo das APIs Java

Com a definição do modelo lógico, é necessário especificar formatos de armazenamento para os dados e definir a arquitectura aplicacional que permita realizar as transformações sobre os dados, similares às operações de AR.

Neste contexto, foi proposta uma abordagem recorrendo a um formato universal de armazenamento, o XML, e para a construção da camada aplicacional, a utilização da linguagem Java, uma linguagem de programação multiplataforma que permite a criação de um pacote de aplicações independente de sistema operativo e arquitectura.

Foi realizado um estudo aos vários mecanismos que permitem a transformação de documentos estruturados em XML recorrendo à linguagem Java. Foram identificados dois grupos de APIs que permitem a manipulação de documentos XML com metodologias diferenciadas no processamento de documento XML: as APIs que realizam a leitura de todo o documento para memória, recriando em memória toda a estrutura hierárquica presente no documento original; e as APIs que processam um documento XML em modo *streaming*, em que a leitura do documento XML é realizada sequencialmente utilizando uma pequena quantidade de memória, sem que a estrutura do documento seja representada em memória.

Alinhando a metodologia de processamento de documentos XML de cada uma das APIs com as operações de AR, foi possível estabelecer um padrão de performance entre cada metodologia das APIs e as características de transformação das operações de AR.

De acordo com os testes realizados, provou-se que as operações de AR que permitem a transformação do documento à medida que este é processado, como é o caso da operação de projecção ou selecção, são as mais adequadas para a implementação recair numa API cuja leitura é realizada em modo *streaming*. As operações que necessitam de alterar a ordem com que os elementos se encontram estruturados na hierarquia do documento, como o caso da operação de ordenação, ou então operações que necessitem de consultar a estrutura do próprio documento de forma aleatória, como o caso da operação de diferença, indiciam uma utilização mais apropriada para as APIs cujo processamento do

documento em XML é realizado em memória. Verificou-se que a diferença de performance das APIs para os casos apresentados é demasiado relevante para ser ignorada.

No entanto as APIs baseadas em memória, independentemente do seu modelo de representação de dados, possuem um limite de tamanho associado, para o tamanho do documento a processar, no sentido que a memória disponível é limitada. Considerando que a execução de processos de ETL se encontra associado a grandes volumes de dados, a utilização destas APIs encontra-se claramente condicionada.

Por forma a tirar partido da sua potencialidade de processamento, é necessário recorrer à utilização de duas operações complementares desenvolvidas no âmbito deste trabalho: a operação de *split* e a operação de *merge*. A operação de *split* permite, com base num determinado critério, dividir um documento XML em vários documentos de menor dimensão, permitindo reduzir a sua complexidade e consequentemente habilitar a sua utilização para o processamento nas operações codificadas segundo a metodologia das APIs baseadas em memória. Esta estratégia permite, para operações independentes entre si, decompor a complexidade dos documentos, de forma a permitir o possível aumento de performance global na execução do ETL. A operação *merge* permite juntar todos os documentos que resultaram da divisão do documento original, já com a respectiva transformação aplicada.

5.1.3 Modelo físico de um processo de ETL

Com base no modelo lógico para a modelação de processos de ETL e na sua conversão à notação BPMN, é possível realizar um mapeamento para as primitivas de programação de forma a permitir a sua execução. O formato de armazenamento de dados XPDL, suportado pelo notação BPMN para a definição do *workflow* das operações, permite o mapeamento do processo de ETL para o modelo lógico apresentado. Com base na especificação lógica do processo de ETL será necessário aplica-lo para a sua execução, especificando pormenores físicos de execução.

Para isso, foi especificado um modelo de representação física de todo o processo de ETL. Todo o processo de transformação do modelo lógico para o modelo físico é realizado de uma forma automatizada, com a definição das operações codificadas em Java definidas em documentos XML complementares que permitem a definição da especificação de cada uma das operações de transformações que compõem o processo de ETL. Os documentos de definição de cada operação permitem especificar o comportamento de cada operação terá de realizar sobre o documento a realizar a transformação.

No documento de especificação física, para além da definição de todo o fluxo de operações, são definidos pormenores de execução, como a localização do documento de definição, executável a utilizar e os documentos de entrada para cada uma das operações.

Todas estas especificações são definidas recorrendo a um documento estruturado em XML que permitirá a execução de todo o processo num ambiente de execução.

5.1.4 Descrição do ambiente computacional utilizado

A utilização de ambientes de execução externos ao SDW, permite libertar o SDW para outras tarefas, permitindo a distribuição de tarefas constituintes de um processo tipicamente muito complexo, por infraestruturas alternativas, podendo assim aumentar a performance de todo o processo.

Considerando, que as organizações normalmente possuem uma grande quantidade de recursos computacionais, de modo a suportar o seu normal funcionamento, muitas vezes subaproveitados ou inutilizados, é possível concluir que muito do processamento necessário à execução do processo de ETL encontra-se dentro da própria organização, suprimindo a necessidade de aquisição de *hardware* adicional.

O conceito de GRID computacional, recorrendo ao trabalho colaborativo entre máquinas e da decomposição de tarefas permite tirar partido de recursos que se encontram desaproveitados, providenciando um ambiente de suporte que privilegia a metodologia de decomposição de tarefas que na proposta de modelação lógica e física apresentada se encaixa.

Recorrendo ao gestor de trabalhos de uma GRID computacional é possível proceder à distribuição das tarefas do modelo físico de especificação do processos e ETL apresentado pelos nodos computacionais, em que cada operação de transformação corresponderá a um trabalho. Assim, o gestor de tarefas terá a responsabilidade de avaliar a disponibilidade dos vários nodos da GRID, de forma a avaliar quais os recursos computacionais mais apropriados para a realização de cada uma das operações do processo de ETL.

O gestor de trabalhos será o elemento crítico de todo o sistema, na medida que terá de coordenar a execução da hierarquia de operações, ajustando e balanceando todo o processo de modo a reduzir o tempo de execução das tarefas, selecionando os recursos mais adequados, gerindo a transferência de documentos na rede, uma vez que a largura de banda disponível neste tipo de ambientes é limitada. Adicionalmente o gestor de tarefas terá de ser capaz de prever a disponibilidade dos nodos, por forma a reduzir as realocações de trabalho, resultantes de mudanças em termos de disponibilidade.

O ambiente de execução em GRID permite ainda a cooperação entre organizações através da interligação de GRIDs computacionais, permitindo não só o aumento de performance e disponibilidade global da infraestrutura, como também da cooperação e integração de informação entre organizações.

As GRIDs computacionais assumem-se como ambientes de execução viáveis à implementação da modelação de processos de ETL, permitindo uma distribuição da decomposição das operações do processo de ETL, visando o aproveitamento dos recursos existentes dentro de uma organização.

5.2 Trabalho futuro

Tendo em consideração o trabalho descrito, são várias as perspectivas de trabalho futuro consideradas desde a modelação lógica até à modelação física.

Um dos problemas discutidos nas propostas de modelação apresentadas por Vassiliadis e Trujillo foi a utilização as respectivas notações para a modelação de processos de ETL de grande dimensão. De facto, o problema acaba por ser transversal a todas as notações, inclusive a proposta neste documento, na medida que a decomposição das em operações de AR reduz a abstração na representação do processo de ETL, resultando num diagrama muito complexo. No entanto como os nodos representam operações elementares, é possível subdividir a representação de um processo de ETL em ramos de operações, que representam um conjunto de subprocessos, constituindo uma hierarquia. Desta forma é possível não só simplificar a interpretação do diagrama como também aproveitar cada subprocesso e executá-lo no ambiente de execução como se de um único processo se tratasse.

A complexidade do modelo lógico pode também ser reduzido aumentando o nível de abstração entre a modelação lógica e o utilizador, introduzindo um novo modelo conceptual que permita um mapeamento para o modelo lógico. Isto é, ao invés de o utilizador trabalhar numa perspectiva de baixo nível em que têm de ser definidas todas as operações elementares de AR, o utilizador poderia utilizar funções predefinidas, ou seja, operações compostas por várias operações de AR para a realização de uma determinada transformação. Posteriormente o aplicativo responsável pela conversão para modelo físico de execução realizaria a decomposição das operações compostas em operações elementares. Este procedimento reduziria certamente o tamanho do diagrama sobre a metodologia apresentada, reduzindo a sua complexidade de leitura e interpretação.

A evolução e a inclusão de novos artefactos na notação BPMN para o modelo lógico apresentado é também pertinente, possibilitando a especificação de novas regras de modo a enriquecer não só a representação gráfica mas também a qualidade de execução do processo. A inclusão de regras, como requisitos mínimos em termos de poder computacional para um determinado recurso, políticas de recuperação ou até de validação de dados podem ser uma mais valia na representação do modelo utilizando a notação BPMN.

Um factor a considerar é a sequência de execução das transformações. O processo de modelação acaba por ficar dependente da capacidade de o utilizador modelar o sistema de uma forma correcta, isto é, a sequência indicada no processo de modelação será a sequência de execução. Por isso, seria pertinente uma comunicação bidirecional entre a infraestrutura de execução e o processo de

construção do modelo físico de execução, de modo a ajustar a sequência de operações às necessidades da infraestrutura de execução.

Referências

[1] W. H. Inmon, *Building the data warehouse*: John Wiley & Sons, 1993.

[2] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for ETL processes," presented at the Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, McLean, Virginia, USA, 2002.

[3] L. Lunan, "A framework study of ETL processes optimization based on metadata repository," in *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, 2010, pp. V6-125-V6-129.

[4] A. Simitsis, P. Vassiliadis, and T. Sellis, "Optimizing ETL processes in data warehouses," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, 2005, pp. 564-575.

[5] A. Simitsis, "Modeling and Optimization o f Extraction - Transformation - Loading (ETL) Processes in Data Warehouse Environments," Ph. D., COMPUTER SCIENCE, NATIONAL TECHNICAL UNIVERSITY OF ATHENS, 2000.

[6] Z. E. Akkaoui and E. Zimanyi, "Defining ETL worfklows using BPMN and BPEL," presented at the Proceeding of the ACM twelfth international workshop on Data warehousing and OLAP, Hong Kong, China, 2009.

[7] J. Trujillo and S. Luján-Mora, "A UML Based Approach for Modeling ETL Processes in Data Warehouses Conceptual Modeling - ER 2003." vol. 2813, I.-Y. Song, S. Liddle, T.-W. Ling, and P. Scheuermann, Eds., ed: Springer Berlin / Heidelberg, 2003, pp. 307-320.

[8] V. Santos, B. Oliveira, R. Silva, and O. Belo, "Configuring and Executing ETL Tasks on GRID Environments - Requirements and Specificities," in *World Conference on Innovation and Software Development*, Istambul, Turkey, 2011.

[9] W. H. Inmon, C. Imhoff, and R. Sousa, *Corporate information factory*: John Wiley & Sons, 2001.

[10] R. Kimball, *The data warehouse lifecycle toolkit*: Wiley Pub., 2008.

[11] T. M. Connolly and C. E. Begg, *Database systems: a practical approach to design, implementation, and management*: Addison-Wesley, 2005.

[12] M. Breslin, "Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models," *Business Intelligence Journal (2004),* vol. Volume: 9, Pages: 6-20, 2004.

[13] T. Ariyachandra and H. J. Watson, "Which Data Warehouse Architecture Is Most Successful?," *BUSINESS INTELLIGENCE JOURNAL,* vol. 11, 2006.

[14] R. Kimball and J. Caserta, *The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data*: Wiley, 2004.

[15] M. V. Mannino and Z. Walter, "A framework for data warehouse refresh policies," *Decision Support Systems,* vol. 42, pp. 121-143, 2006.

[16] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "On the Logical Modeling of ETL Processes," presented at the Proceedings of the 14th International Conference on Advanced Information Systems Engineering, 2002.

[17] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos, "Conceptual modeling for ETL processes," in *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, ed McLean, Virginia, USA: ACM Press, 2002, pp. 14-21.
[18] S. A. White and D. Miers, *Bpmn Modeling and Reference Guide*: Future Strategies Inc, 2008.
[19] S. A. White. (2006). *Introduction to BPM*.
[20] S. A. White. (2005). *Using BPMN to Model a BPEL Process*.
[21] W. M. Coalition. Available: http://www.wfmc.org/xpdl.html
[22] W. M. P. van der Aalst, "Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language," 2003.
[23] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM,* vol. 13, pp. 377-387, 1970.
[24] E. F. Codd, "Relational completeness of data base sublanguages," in *Database Systems*, 1972, pp. 65-98.
[25] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. M. Matos, "Extending relational algebra and relational calculus with set-valued attributes and aggregate functions," *ACM Trans. Database Syst.,* vol. 12, pp. 566-592, 1987.
[26] P. W. P. J. Grefen and R. A. de By, "A multi-set extended relational algebra: a formal approach to a practical issue," in *Data Engineering, 1994. Proceedings.10th International Conference*, 1994, pp. 80-88.
[27] U. J. D., *A First Course in Database Systems, 2/e*: Pearson Education.
[28] H. García-Molina, J. D. Ullman, and J. Widom, *Database system implementation*: Prentice Hall, 2000.
[29] P. B. Kis and A. Buza, "Expansion of the relational algebra," in *Intelligent Systems and Informatics, 2009. SISY '09. 7th International Symposium on*, 2009, pp. 127-130.
[30] M. Dadashzadeh, "An improved division operator for relational algebra," *Inf. Syst.,* vol. 14, pp. 431-437, 1989.
[31] V. Santos, B. Oliveira, R. Silva, and O. Belo, "Distribuição de Tarefas ETL em Ambientes GRID," in *11ª Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI 2011)*, Lisboa, Portugal 2011.
[32] C. Weekly. (2002) "Write once, run anywhere?".
[33] F.-F. O.-L. D. o. computing. (2010, Setembro de 2011). Available: http://foldoc.org/Application+Program+Interface
[34] E. Perkins, M. Kostoulas, A. Heifets, M. Matsa, and N. Mendelsohn, "Performance Analysis of XML APIs," in *XML 2005 Conference*, 2005.
[35] S. C. Haw and G. S. V. R. K. Rao, "A Comparative Study and Benchmarking on XML Parsers," presented at the The 9th International Conference on Advanced Communication Technology, 2007.
[36] M. L. Noga, S. Schott, W. L\, \#246, and we, "Lazy XML processing," presented at the Proceedings of the 2002 ACM symposium on Document engineering, McLean, Virginia, USA, 2002.
[37] F. Wang, J. Li, and H. Homayounfar, "A space efficient XML DOM parser," *Data Knowl. Eng.,* vol. 60, pp. 185-207, 2007.
[38] J. Zhang. (2008). Available: http://www.codeproject.com/KB/cs/vtd-xml_examples.aspx
[39] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*: Elsevier, 2004.
[40] B. Segal, Robertson, G. L., and C. F., F, "Grid computing: the European Data Grid Project," presented at the 47th IEEE Nuclear Science Symposium and Medical Imaging Conference, Lyons, France, 2000.