Universidade do Minho
Escola de Engenharia

Janine Marlene Duarte Silva Freitas

Transformação de Especificações ETL em YAWL para Processos Kettle

Outubro 2019

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Janine Marlene Duarte Silva Freitas

Transformação de Especificações ETL em YAWL para Processos Kettle

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de

Professor Doutor Orlando Manuel de Oliveira Belo

Outubro 2019

Transformação de Especificações ETL em YAWL para Processos Kettle

Janine Marlene Duarte Silva Freitas

Dissertação apresentada à Universidade do Minho para obtenção do grau de Mestre em Engenharia Informática, elaborada sob orientação do Professor Doutor Orlando Manuel de Oliveira Belo.

2019

Agradecimentos

Ao meu professor e orientador, Professor Doutor Orlando Belo, por toda a disponibilidade e apoio prestado ao longo desta dissertação.
Aos meus pais e irmã, que sempre me apoiaram, por todo o suporte dado, especialmente nesta última fase.
A todos os amigos que me acompanharam durante o meu percurso académico, o meu muito obrigada por todos os momentos partilhados.
Um especial obrigado ao André, para quem todas as palavras são poucas, por todo o seu apoio e suporte, especialmente nos momentos mais difíceis, sem ele certamente não teria sido possível.
E por fim, a todos aqueles que de alguma forma contribuíram para o meu crescimento pessoal e profissional durante o meu percurso académico.

Resumo

Transformação de Especificações ETL em YAWL para Processos Kettle

YAWL é uma linguagem gráfica para a especificação de processos, com uma semântica bem definida, que permite o desenho, especificação, simulação e validação de sistemas, cujos processos a modelar exijam características específicas de comunicação, concorrência e sincronização entre si. A YAWL é baseada, por um lado, em padrões bem definidos de fluxo de trabalho e, por outro, nas conhecidas Redes de Petri Coloridas. Devido às suas características, vários estudos utilizaram esta linguagem na modelação de sistemas de ETL (*Extract-Transformation-Load*), tentando facilitar e agilizar todo este processo. Este trabalho de dissertação teve como objetivo desenvolver e implementar um sistema de geração de "esqueletos" para sistemas de ETL a partir de um conjunto de especificações YAWL. Nesse sentido, foi idealizado e desenvolvido um sistema capaz de representar e produzir de forma semi-automática a configuração de processos ETL para várias ferramentas de implementação de processos ETL, nomeadamente Kettle, Informatica e Talend, todas elas ferramentas conceituadas no mercado dos sistemas de povoamento de *data warehouses*. Nesta dissertação apresentamos e descrevemos tal sistema, desde as suas fases de fundamentação e conceptualização até às suas fases de teste e exploração.

Palavras-Chave: Sistemas de Data Warehousing, Sistemas de Povoamento de Data Warehouses (ETL), Modelação de Sistemas ETL, Geração de Esqueletos ETL, YAWL, Kettle, Informatica e Talend.

Abstract

Transformation of ETL processes from YAWL to Kettle

YAWL is a graphical language with a well-formed semantics, that allows the design, specification, simulation, and validation of systems where the processes to model have specific characteristics such as, communication, concurrency and synchronization. It is based, on one hand, on well-formed workflow patterns and on the other standards known as the Colored Petri Nets. Due to its characteristics several studies have used this language in the modeling of ETL (Extract-Transformation-Load) systems, trying to facilitate and improve uppon this process. The main goal of this dissertation was to develop and implement a system for the generation of skeletons for ETL systems from YAWL. A system was designed and developed to represent the semi-automatic generation of ETL process for some ETL tools, namely Kettle, Informatica and Talend, all of them renowned tools to populate data warehouses. In this dissertation we present and describe such a system, from its founding and conceptualization phases to its testing and exploration phases.

Keywords: Data Warehousing systems, Systems to populate Data Warehouses (ETL), ETL modeling systems, ETL skeleton generation, YAWL, Kettle, Informatica and Talend.

Índice

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

Vivemos atualmente na era da informação. A quantidade de dados que se armazena diariamente nos sistemas operacionais aumenta de dia para dia, o que torna necessário ter capacidade de resposta para processar tal volume de dados. Neste sentido, têm surgido alguns sistemas capazes de gerir este problema de uma forma efetiva, proporcionando às empresas uma melhor capacidade de organização e extração dos dados. Usualmente, estes sistemas são designados por sistemas de *data warehousing*, que suportam o processamento de informação através da disponibilização de uma plataforma sólida de análise de dados, facilitando o processo de tomada de decisão por parte do sector empresarial. Para isso, necessitam de um elevado número de recursos para que sejam capazes de gerir os volumes de dados envolvidos e fazer a sua disponibilização de forma eficiente para os agentes de tomada de decisão. Todo este processo é garantido através de sistemas de povoamento específicos, vulgarmente designados por *Extract-Load-Transform (ETL)*, que permitem integrar num único espaço – o *data warehouse* – os dados provenientes de diferentes fontes operacionais, garantindo a sua qualidade através da aplicação de limpeza, integração e transformação dos dados. Por isso mesmo, um processo ETL é bastante exigente (e moroso) principalmente na integração com o próprio sistema de *data warehousing*. De modo a tentar minimizar o esforço da sua implementação, recomenda-se o desenho e a modelação conceptual de

um sistema de ETL. O modelo resultante desse processo deve ter em conta os requisitos operacionais, garantindo assim o sucesso do próprio *data warehouse*. Assim, é indispensável a utilização de metodologias para suportar o processo de desenvolvimento de um sistema de ETL.

Apesar da importância deste processo de modelação, continua a não existir uma abordagem convincente para a modelação de processos de ETL, capaz de suportar as diversas fases do seu desenvolvimento e implementação. Tendo isso presente, nesta dissertação, adotou-se e utilizou-se uma linguagem baseada em fluxos de trabalho para especificar um processo de ETL e, posteriormente, converter esta especificação em "esqueletos" semi-automáticos, acelerando assim o processo de implementação, garantindo que os requisitos e todo o processo conceptual fosse devidamente respeitado no processo de implementação.

A *Yet Another Workflow Language* (YAWL) (Foundation, 2010) é uma linguagem que foi desenvolvida por Wil van der Aalst (Eindhoven University of Technology, the Netherlands) e Arthur ter Hosdtede (Queensland University of Technology, Australia) em 2002, baseada, por um lado em redes de Petri Coloridas e por outro em padrões bem conhecidos de fluxo de trabalho. Por ser simples e de fácil acesso, tem vindo a ser explorada como uma ferramenta poderosa que permite especificar os dados relativos à execução de um *model system*. Tais características, fazem com que a YAWL seja uma ferramenta de modelação bastante adequada para a especificação de sistemas de povoamento de *data warehouses*.

1.2 Motivação e objetivos

Aplicar a YAWL para modelar e especificar processos de ETL é algo que tem vindo a ser estudado (Oliveira e Belo, 2004), (Belo, Cuzzocrea e Oliveira, 2014). Esta nova abordagem surge com o objetivo de fornecer uma visão mais genérica que conseguisse abranger todas as fases de desenvolvimento de um sistema ETL. Com isto, pretende-se fazer com que a elaboração e a compreensão de um modelo conceptual para um sistema ETL se torne mais fácil, através de uma simplificação dos usuais métodos de desenvolvimento de sistemas ETL reduzindo, consequentemente, o seu custo de implementação.

Este trabalho de dissertação teve como objetivo principal planear, desenvolver e implementar uma

peça de *software* que fosse capaz de pegar num modelo conceptual de um sistema ETL desenvolvido em YAWL e gerar de forma (semi) automática uma primeira configuração do sistema ETL (Belo *et al.*, 2016) – "esqueleto" – para uma sua possível implementação física, com vista ao seu acolhimento e execução numa ferramenta de construção de sistemas de ETL (*Data Integration* - Kettle). Para tal, será necessário estudar aprofundadamente os ficheiros gerados através de YAWL e pela ferramenta de construção de sistemas de ETL, de modo a que seja possível fazer a geração automática dos "esqueletos" referidos.

1.3 Estrutura do Relatório

Além do presente capítulo, este relatório é composto por mais quatro capítulos. No capítulo 2 será abordado o "estado da arte" da linguagem YAWL expondo alguns dos trabalhos realizados na área. Além disso, aqui também é explicado de forma sucinta como funciona a YAWL e quais são os elementos que a compõem, realçando alguns dos fluxos de trabalho mais utilizados. De seguida, no capítulo 3, apresentar-se-á as diferentes etapas da criação do sistema de geração de processos ETL. É nesta fase que se dará ênfase à importância da modelação conceptual durante a realização de um processo de ETL e onde será exposto o caso de estudo utilizado como base de trabalho nesta dissertação. Com base no caso de estudo escolhido, será possível acompanhar todos os passos do processo, desde a sua implementação, primeiro em YAWL e depois em Kettle, a análise dos ficheiros XML gerados e posteriormente a implementação em Java do sistema. No capítulo 4, depois de verificada a possibilidade de conversão de transformações YAWL em processos Kettle, serão abordadas duas novas ferramentas ETL, o Informatica e o Talend, que servirão também como plataformas de teste para validação do sistema desenvolvido. Por fim, no capítulo 5 será desenvolvida uma análise crítica ao trabalho realizado e apresentadas as devidas conclusões e trabalho futuro. Como curiosidade, na Figura 1 pode-se observar as diferentes etapas desta dissertação expressas na linguagem YAWL.

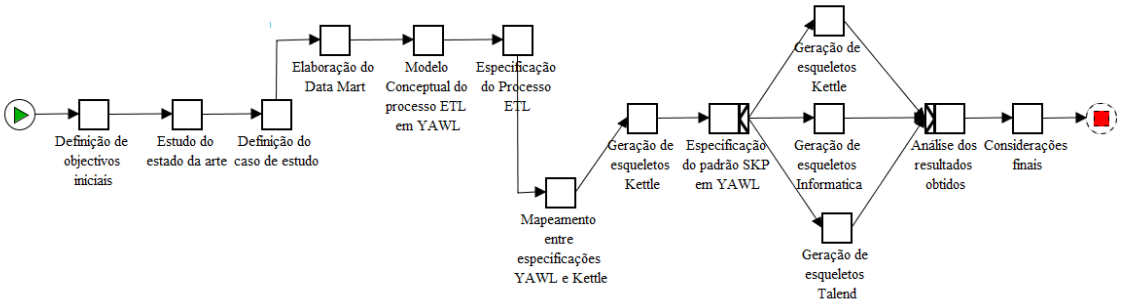


Figura 1 – Esquema da estrutura da dissertação em YAWL.

Capítulo 2

A Linguagem YAWL

2.1 Trabalho Relacionado

Cada vez mais, as organizações procuram melhorar a sua eficiência e capacidade de responder às mudanças do dia-a-dia. É por isso evidente que procuram analisar e melhorar os seus métodos e fluxos de trabalho. Usualmente, as ferramentas e técnicas utilizadas para isso designam-se *Business Process Management (BPM*). Para as organizações, BPM oferece uma oportunidade de conciliar os sistemas informáticos com os requisitos de negócio, de forma a melhorar e a facilitar um dado processo de negócio. É neste sentido que surge a YAWL (Foundation, 2010), capaz de especificar e identificar todos os tipos de dependências de fluxo entre processos ou tarefas. É caracterizada pela facilidade de desenvolvimento de soluções de modelos de fluxo de trabalho e deteção de erros, sendo por isso cada vez mais utilizada não só a nível académico, mas também empresarial.

A linguagem YAWL definida por Van Der Aalst e Ter Hofstede (2005) teve como ponto de partida as redes Petri Coloridas (Petri, 1962) às quais se acrescentaram mecanismos para permitir um suporte mais intuitivo dos padrões de fluxo de trabalho. A YAWL caracteriza-se especialmente pelo facto de ser *open source*, permitindo assim uma fácil extensão. Esta tornou-se uma linguagem bastante apreciada para BPM (Russell e Ter Hofstede, 2009) pois oferece um grande suporte de padrões *workflow*, facilita a sua especificação e consegue identificar dependências durante o controlo de fluxo e requisitos de vários recursos.

Para além de comparações entre *YAWL* e *Business Pattern Model & Notation (BPMN)* (Börger, 2012), (Decker *et al.*, 2002) têm surgido várias propostas de modelação de processos ETL utilizando YAWL. Tem-se estudado a sua posição e competitividade no mercado, comparando com outras soluções já existentes, (Adams, Ter Hofstede e La Rosa, 2011) explicando o porquê da sua necessidade quando comparada, por exemplo, a redes Petri Coloridas (Ter Hofstede *et al.*, 2010). Em (Oliveira, Belo e Cuzzocrea, 2008) e (Belo, Cuzzocrea e Oliveira, 2014), foi feita uma abordagem orientada por padrões para apoiar a modelação de processos ETL, na qual se pretendia identificar e formalizar as tarefas comuns para descrever o seu comportamento, permitindo assim a sua aplicação e reutilização em diferentes cenários de aplicação. Também em (Belo *et al.*, 2014) e (Gomes, 2013) se estudou a modelação de processos ETL, com o objetivo de desenvolver um sistema que permitisse gerar modelos de ETL de forma rápida e eficaz, de maneira a encurtar o tempo e recursos gastos no seu desenvolvimento. O trabalho desenvolvido nesta dissertação, deu continuidade ao trabalho que foi sendo desenvolvido em (Guimarães, 2014) através da concepção e implementação de uma peça de *software* capaz de pegar num modelo conceptual de um sistema ETL e gerar de forma (semi) automática um "esqueleto" para uma sua possível implementação física, na ferramenta *Data Integration* – Kettle (Pentaho Corporation, 2012).

2.2 A YAWL

A YAWL é uma linguagem de fluxo de trabalho desenvolvida em 2002 que teve como base, por um lado, as Redes de Petri Coloridas, que desempenham um bom papel na concorrência e sincronização de modelos, e por outro lado os *workflow patterns* que complementam com um conjunto de processos que especificam os fluxos de trabalho. A YAWL, estende, portanto, as Redes Petri coloridas com a possibilidade de criação de fluxos de trabalho específicos.

Com base nas suas principais características, foi, ao longo dos últimos anos, alvo de vários estudos nas mais diversas áreas. Um dos principais exemplos da sua utilização designa-se YAWL4Film (Ouyang *et al.*, 2008), que é uma ferramenta utilizada para automatização de processos de produção de filmes. Inicialmente foi utilizada para automatizar o processo de *Call Sheet*, que normalmente é um processo bastante demorado e propício a erros. Nos processos modelados com a YAWL verificou-se uma melhoria na sua organização e, consequentemente, o tornou o processo de gravação mais eficiente.

Mais tarde, utilizando como exemplo os comboios *Reti Ferroviarie Italiane,* (Marrella *et al.*, 2011), a YAWL foi utilizada para delegar a execução de subprocessos e atividades para o ambiente *SmartPM execution*, que é capaz de adaptar automaticamente um processo para lidar com mudanças e exceções emergentes. Hoje em dia, a YAWL é uma linguagem de especificação de processos bastante apreciada, pois consegue identificar dependências durante o controlo de fluxo e permite o tratamento de exceções quer surjam durante o desenvolvimento ou não, oferecendo um grande suporte para padrões de controlo de fluxo.

A escolha da YAWL como ferramenta de especificação utilizada nesta dissertação foi feita tendo em conta todas as características e potencialidades acima referidas e à sua simplicidade de representação, o que torna mais simples a compreensão de um processo ETL, com base nos tipos de tarefas utilizadas para representar cada fase do processo. Também, tem a vantagem de ter por base a linguagem XML, permitindo uma fácil extração da informação necessária para a construção dos esqueletos em Kettle, os quais podem ser assim construídos através de ficheiros XML.

A YAWL permite a especificação de processos no seu *Process editor* e a sua execução através do *Engine*. Nesta dissertação daremos ênfase ao seu editor, explicando cada um dos seus elementos e padrões. Os elementos presentes no editor permitem a criação, seleção e posicionamento de objetos durante a especificação. A área de trabalho, em que cada elemento é definido, designa-se *net* e, dependendo do tipo de elemento, existe a possibilidade de criação de uma *sub-net*. Na Tabela 1 apresentamos cada um desses elementos.

Tabela 1 – Descrição dos elementos presentes em YAWL

Elemento	**Representação**	**Descrição**
Input e output condition		Estes dois elementos são criados automaticamente após a criação de um novo ficheiro. São obrigatórios e significam o início de um processo e o seu fim, respetivamente.
Atomic task		Esta tarefa é fundamental no desenvolvimento de uma especificação e representa uma tarefa simples. Na especificação de um processo ETL é utilizada para especificar todos as tarefas básicas, por exemplo extrair dados de uma tabela.
Multiple Instance Atomic Task		Permite executar múltiplas instâncias de tarefas simultaneamente.

Condition Task		Permite a criação de condições, representando um estado que determinada tarefa tem que cumprir para avançar na execução.
Composite Task		A utilização desta tarefa é semelhante à tarefa simples, no entanto esta permite a criação de uma *sub-net* onde é possível criar um novo processo, simplificando a leitura e a visão geral. Durante a especificação de um processo ETL pode ser utilizada para representar conjunto de tarefas semelhantes, como por exemplo, a definição de um padrão.
Multiple Instance Composite Task		Este elemento caracteriza-se por ser uma junção de duas tarefas anteriores. Permite a criação de uma *sub-net* e permite a execução de múltiplas instâncias de tarefas simultaneamente.

Tal como referido anteriormente, uma das principais características da YAWL é o suporte de padrões de fluxo de trabalho, permitindo a simplificação da especificação e a leitura das tarefas. Para isso existem diferentes maneiras visuais de expressar diferentes tipos de processos, desde os mais simples (sequenciais) aos mais complexos (concorrentes/seleção). Para os representar é necessário recorrer às tarefas *Split* e *Join* presentes na YAWL (Figura 2). Na Tabela 2 apresentamos a descrição e utilidade de cada uma delas.

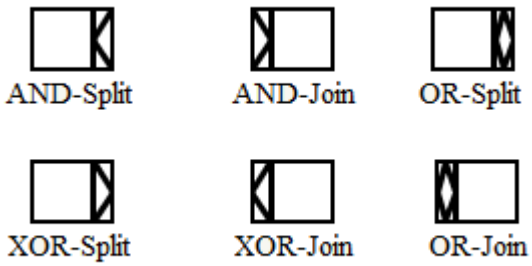


Figura 2 – Exemplo das tarefas *Split* e *Join* existentes em YAWL.

Tabela 2 – Descrição das tarefas *Join* e *Split* existentes em YAWL

Nome da Tarefa	Descrição
AND-Split	Esta tarefa é utilizada quando se pretende iniciar várias tarefas em simultâneo
XOR-Split	Esta tarefa é utilizada quando só existe uma saída possível entre as várias tarefas anteriores.
OR-Split	Utiliza-se quando se pretende que comece uma tarefa, mas não necessariamente todos os fluxos associados.
AND-Join	É utilizado para sincronizar tarefas, uma vez que aguarda que todas as tarefas de entrada terminem para começar.
XOR-Join	Permite que a tarefa seguinte inicie, desde que pelo menos uma tarefa anterior tenha terminado.
OR-Join	Garante que uma tarefa aguarde que todos os fluxos de entrada sejam concluídos ou então que nunca sejam concluídos. Devem ser usados com precaução.

Algumas destas tarefas fazem parte dos padrões de fluxo de trabalho representados em YAWL. Abaixo é possível perceber quais são esses fluxos e qual a utilidade de cada uma das tarefas referidas na sua representação.

Fluxo sequencial

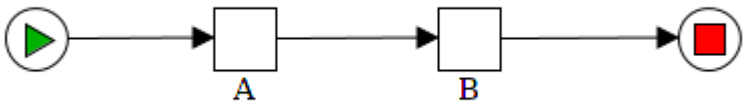


Figura 3 – Exemplo de um processo sequencial.

Tal como o nome indica, neste processo todas as tarefas são executadas de forma sequencial. A primeira tarefa a ser executada é a Tarefa A e, quando terminar é executada a Tarefa B.

Fluxo concorrente

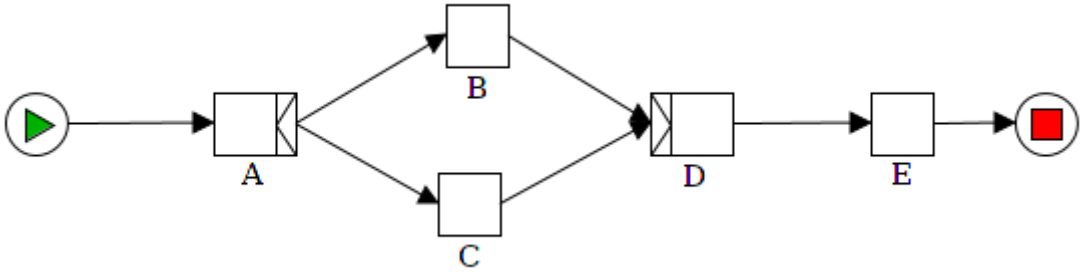


Figura 4 – Exemplo de um processo em paralelo com concorrência e sincronização.

Neste processo a Tarefa A é uma tarefa vazia com a condição *AND-Split*, fazendo com que a Tarefa B e a Tarefa C executem ao mesmo tempo.

Fluxo sincronizado

Este processo surge em resposta ao anterior, uma vez que só depois de existirem tarefas concorrentes é que pode existir a sua sincronização. Neste caso, pode verificar-se que a tarefa E (Figura 4) é executada quando as tarefas que estavam a ser executadas paralelamente terminarem. A tarefa D é, neste exemplo, uma tarefa vazia com a condição *AND-Join*.

Fluxo de escolha única

O fluxo de escolha única, representado pela tarefa B, representa a decisão de escolha entre duas tarefas. Isto significa que, dado uma determinada condição definida pela tarefa A, o processo seguirá ou pela tarefa D ou pela tarefa C, nunca pelas duas em simultâneo como acontece no fluxo explicado anteriormente.

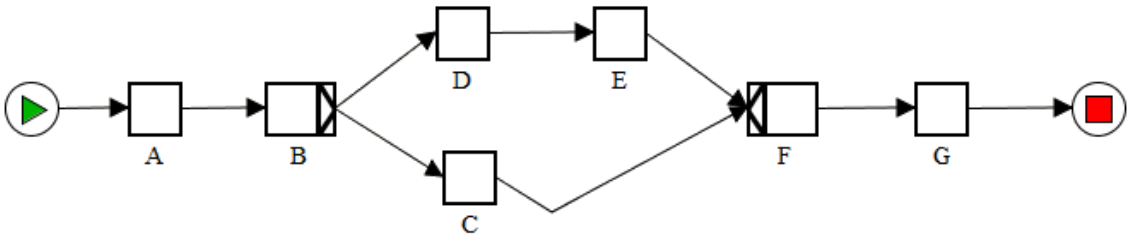


Figura 5 – Exemplo de uma tarefa de escolha única e de junção.

Fluxo de junção

Este tipo fluxo pode surgir, por exemplo, quando existe um processo de escolha. Permite que, independentemente da tarefa executada anteriormente, a tarefa a ser executada seja sempre a mesma. Utilizando como exemplo a Figura 5 a tarefa de junção F permite que a tarefa G não tenha que ser definida mais do que uma vez por existirem duas maneiras distintas de chegar até ela. Neste caso, a sequência poderá ser A-B-D-E-F-G ou, então, A-B-C-F-G sem que exista a necessidade de definir duas vezes a tarefa G, tornando assim a sua implementação mais fácil, a especificação mais simples e a leitura mais clara.

Fluxo condicional

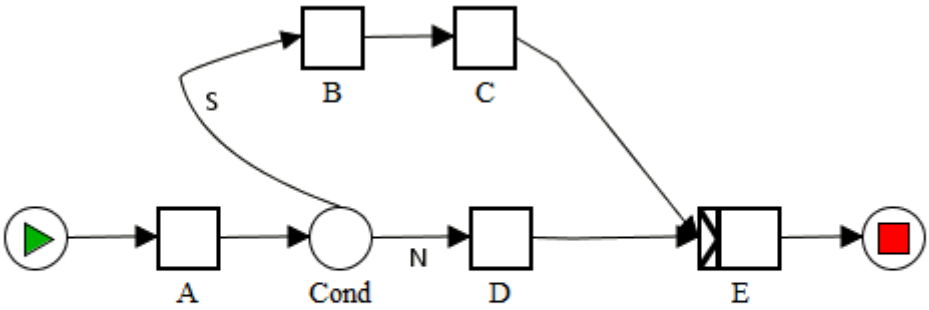


Figura 6 – Exemplo de um fluxo com uma condição.

Neste fluxo, depois de executada a tarefa A, existe uma determinada condição, onde, caso seja cumprida, é executada a tarefa B-C-E. Caso não seja cumprida a condição, o fluxo prossegue para a tarefa D e termina em E. A sua estrutura é semelhante ao processo de escolha única. Outra utilidade da tarefa condicional é a possibilidade de voltar à tarefa anterior, caso a condição assim o exija, como se pode observar na Figura 7, onde, até que seja cumprida a condição, as tarefas B e A serão executadas repetidamente.

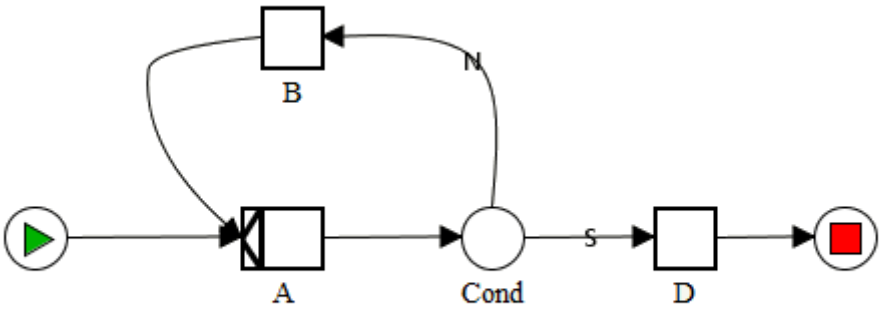
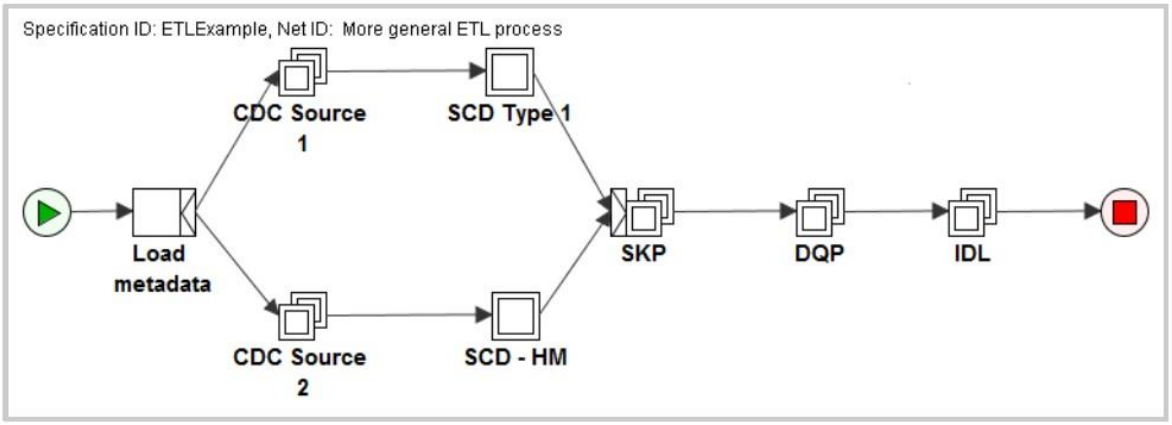


Figura 7 – Outro exemplo de uma tarefa condicional.

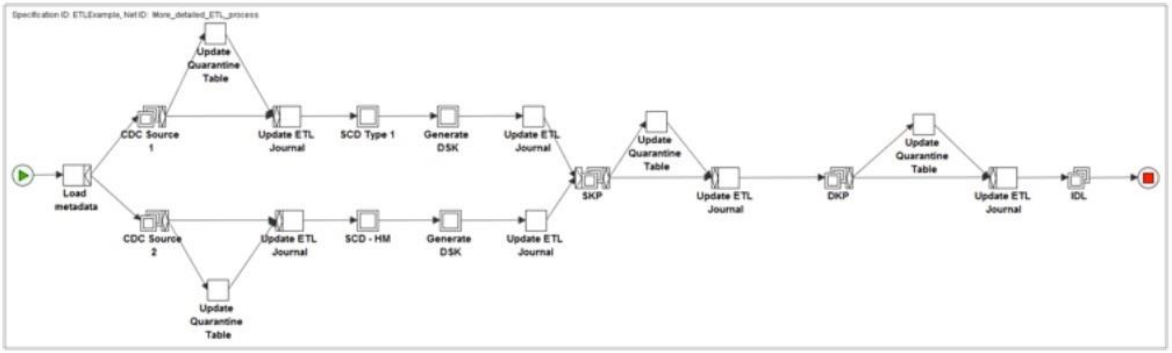
Depois de uma breve explicação sobre a construção de fluxos de trabalho em YAWL, é possível analisar na Figura 8 uma especificação desenvolvida em YAWL, cujo objetivo é a representação de um processo ETL. Na Figura 8.a), é possível identificar facilmente um conjunto de tarefas utilizadas para descrever um processo ETL. Esta maneira abstrata de representação de um processo tão complexo, tem vindo a ser estudada ao longo dos últimos anos, permitindo uma fácil compreensão de todos os processos chave a desenvolver. Desta maneira permite que cada tarefa complexa, também designada como padrão, seja detalhada e especificada num subprocesso à parte.

Na Figura 8.b), é possível observar-se o mesmo processo ETL especificado e representado de uma forma mais detalhada, conjugando no mesmo processo tarefas simples com tarefas complexas.

Ao longo desta dissertação a especificação e representação de processos ETL em YAWL será explorada com mais detalhe.



a) Uma representação abstrata.



b) Uma representação mais detalhada.

Figura 8 – Exemplo de um processo ETL em YAWL extraído de (Oliveira e Belo, 2004).

Capítulo 3

O Sistema de Geração de Esqueletos ETL

O sistema de geração de esqueletos ETL foi desenvolvido de forma a poder gerar, a partir de um modelo conceptual especificado em YAWL, um conjunto de ficheiros XML que pudessem ser interpretados por uma ferramenta para a implementação de processos ETL, mais especificamente a Kettle. A modelação conceptual permite que o processo seja avaliado e validado antes da sua implementação, evitando que sejam feitas alterações ao modelo já numa fase posterior, reduzindo assim o seu esforço de desenvolvimento e consequentemente o seu custo. Conseguir reaproveitar os modelos desenhados conceptualmente em YAWL e transformá-los em processos capazes de ser interpretados por uma ferramenta de construção de processos ETL, não só reduz o tempo e custo de implementação de um processo tão moroso, como reaproveita todo o trabalho desenvolvido durante a fase inicial do processo. Para isso foi desenvolvido um plano de estudo e de desenvolvimento do sistema a implementar. Aproveitando a YAWL como uma linguagem de especificação de processos, pode ver-se na Figura 9 a esquematização do sistema criado para gerar os esqueletos ETL. Para que fosse possível desenhar o processo ETL em YAWL, foi necessário definir um caso de estudo para servir como objeto de estudo prático ao longo desta dissertação. Na secção seguinte apresentaremos esse caso de estudo.

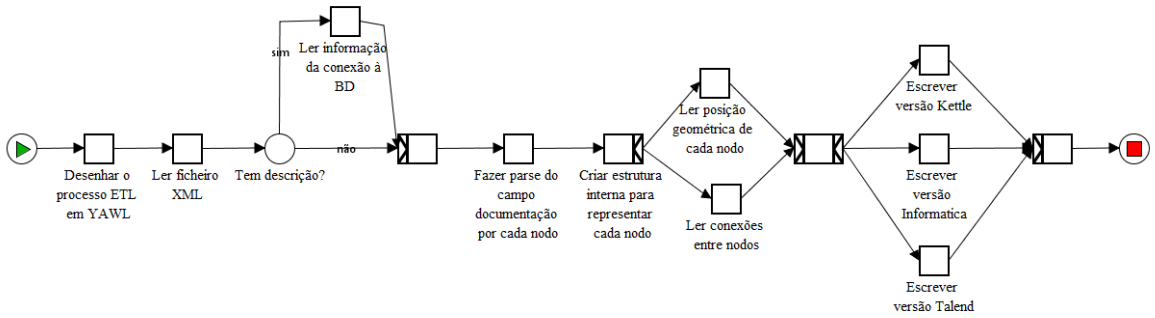


Figura 9 – Representação do sistema de geração de esqueletos ETL.

3.1 Caso de Estudo

Depois de definidos os objetivos e realizado o estado da arte foi necessário definir o caso de estudo que serviria de suporte ao trabalho desta dissertação. Este caso de estudo permitirá simular o processo de desenho e implementação de um sistema ETL. Trata-se de um esquema dimensional de dados relativo a um processo de tratamento de vendas de produtos provenientes de duas fontes de dados distintas. Na Figura 10 podemos ver um esquema do *data mart* (DM), desenhado de acordo com (Golfarelli and Rizzi, 2009), utilizado como caso prático na demonstração de um processo ETL. Como se pode observar, o DM escolhido possui quatro dimensões ("Data", "Cliente", "Funcionário" e "Produto") e uma tabela de factos ("TF_Vendas"). A dimensão "Data" é composta por três atributos dimensionais, "dia", "mês", "ano" e duas hierarquias, representadas da seguinte forma: Data → dia, Data → mês → ano. A dimensão "Produto" possui dois atributos dimensionais, "tipo" e "categoria", e um atributo descritivo, "nome". Os atributos dimensionais formam as seguintes hierarquias: Produto → tipo e Produto → categoria. Por último, as dimensões "Cliente" e "Funcionário" são compostas pelo atributo descritivo "nome" e pelo atributo dimensional "cidade", formando apenas as hierarquias Cliente → cidade e Funcionário → cidade, respetivamente. A tabela de factos "TF_Vendas" agrega a informação do produto vendido por um funcionário, a um determinado cliente numa certa data, integrando como medida o preço unitário, a quantidade vendida e o custo total.

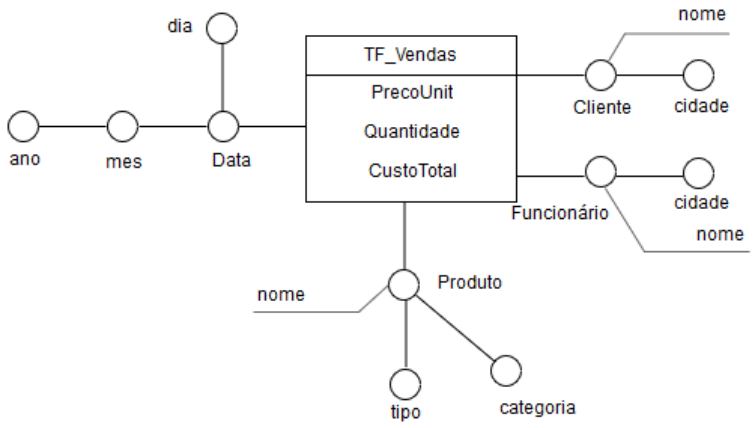


Figura 10 – Esquema do *Data Mart* de Vendas.

3.2 Modelação conceptual de um processo ETL em YAWL

Com base no modelo dimensional definido, esquematizou-se, então, o processo de ETL em YAWL. Este processo foi esquematizado tendo em conta as três diferentes etapas de um processo ETL, nomeadamente extração (*Extract*), transformação (*Transform*) e povoamento (*Load*) (Figura 11). Dentro de cada uma das tarefas múltiplas representadas, estão definidas tarefas concorrentes para cada uma das dimensões definidas, uma vez que não existem dependências entre elas. A tarefa de junção garante que só passam à fase seguinte depois de todas as tarefas estarem terminadas (Figura 12).

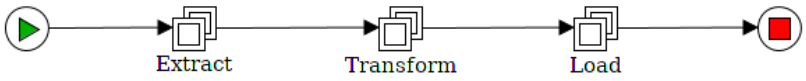


Figura 11 – Representação do Processo ETL em YAWL.

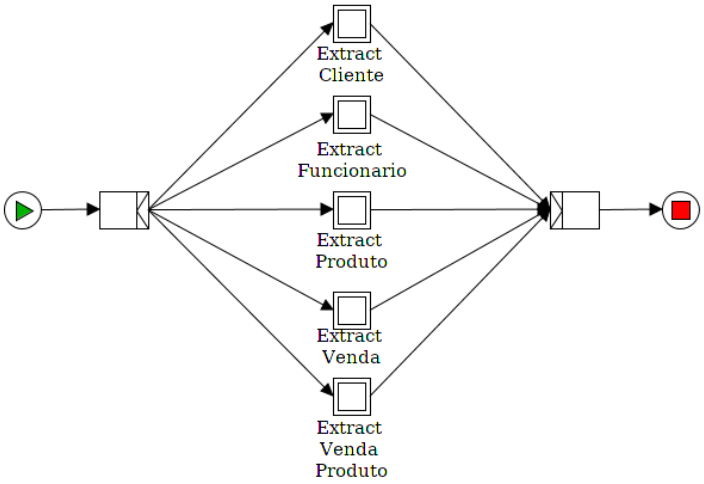


Figura 12 – Representação do processo de extração em YAWL.

Cada etapa do processo representado foi especificada com recurso a tarefas simples, utilizando uma representação o mais completa possível de forma a facilitar o processo de conversão. O processo de extração (Figura 13) é composto por dois fluxos de trabalho independentes, uma vez que os dados que serão extraídos são provenientes de duas fontes de dados distintas.

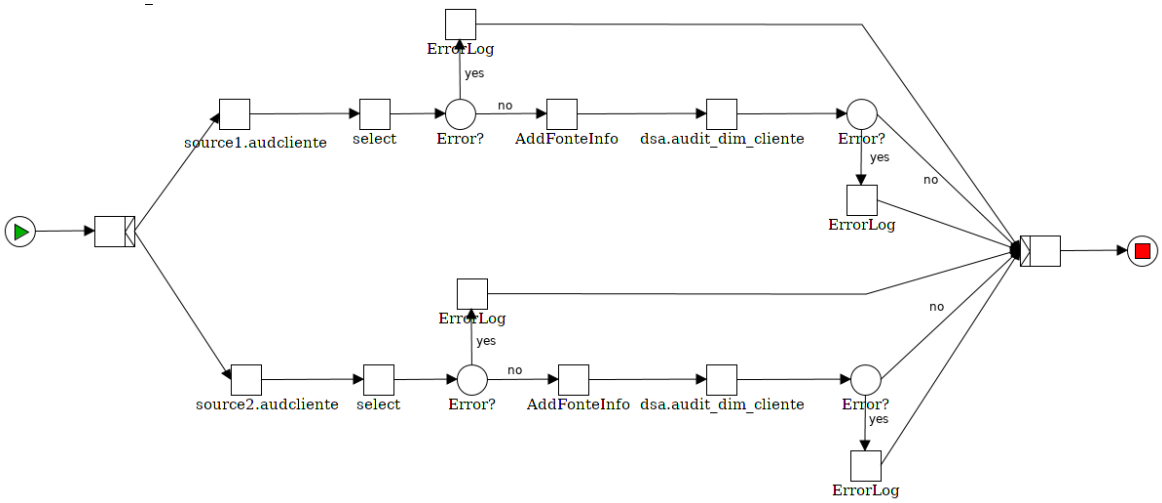


Figura 13 – Representação do processo de extração de um cliente em YAWL.

Numa outra prespectiva, de representação de tarefas sequenciais (Figura 14), é possível observar, por exemplo, as diferentes etapas que compõem a transformação dos dados. É nesta fase que são efectuadas operações de transformação e limpeza dos dados, garantindo que todas as incoerências

serão encaminhadas para ficheiros de quarentena, permitindo assim manter a qualidade da base de dados.

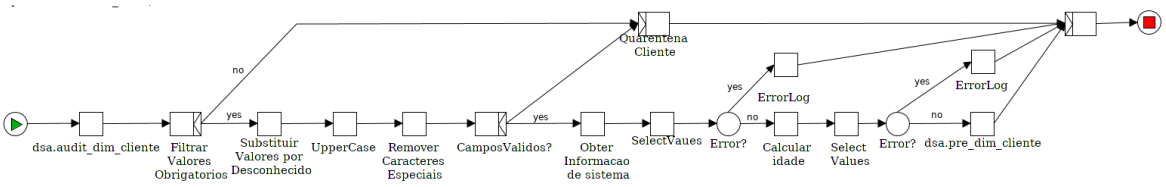


Figura 14 – Representação do Processo de transformação de um cliente.

Alguns desses processos serão explicados em detalhe ao longo desta dissertação, uma vez que o sistema de geração de processos ETL foi desenvolvido com base nesta representação conceptual.

3.3 Desenvolvimento em Kettle

Depois de especificado conceptualmente o processo de ETL, procedeu-se à sua implementação em Kettle de modo a facilitar o processo de geração. Desta forma, foi possível analisar o ficheiro produzido pela ferramenta e comparar com o ficheiro produzido em YAWL e, a partir daí, construir um sistema que fosse capaz de "traduzir" as especificações YAWL para um ambiente ETL. Tal como referido anteriormente, este processo foi desenvolvido tendo em conta a modelação conceptual desenvolvida anteriormente, sendo, por isso, a sua estrutura bastante semelhante, como se pode observar nas imagens seguintes.

Em Kettle existem dois tipos de ficheiros distintos, os *jobs* e as transformações. As transformações, tal como o nome indica, servem para definir processos de transformações entre o ficheiro origem e o ficheiro destino, operando ao nível dos dados. Os *jobs* funcionam como controlo de fluxo de alto nível, executando as transformações. Desta forma, as tarefas múltiplas compostas definidas em YAWL (Figura 11) são definidas como jobs em Kettle (Figura 15) e as restantes definidas como transformações (Figura 16) - as tarefas simples foram definidas dentro de cada ficheiro transformação em Kettle (Figura 17 e Figura 18).

Sendo uma ferramenta de desenvolvimento de processos de ETL dispõe de um conjunto de elementos capazes de especificar cada uma das suas etapas. Na Tabela 3 estão representados os elementos utilizados na especificação dos processos ETL, em Kettle, expostos nesta dissertação.

Tabela 3 – Descrição dos elementos presentes em Kettle

Elemento	**Representação**	**Descrição**
Table input	 Table input	Este elemento permite a leitura de uma tabela da base de dados. Está inserido num conjunto com outro tipo de elementos, designado input, que permitem a leitura de diferentes tipos de ficheiros (csv, json, SAP, Excel…)
Select values	 Select values	Esta tarefa permite que seja feita uma alteração aos campos que prosseguem para o próximo passo no processo de desenvolvimento. Alguns campos podem ser alterados outros removidos. Esta seleção é sempre feita ao nível dos metadados.
Add constants	 Add constants	Permite definir novos campos e atribuir valores por omissão.
Filter rows	 Filter rows	Permite filtrar os dados, quando estes respeitam determinada condição (null, contains, like, regex…).
If field value is null	 If field value is null	Esta tarefa é utilizada quando se pretende dar outro valor a um determinado campo que está nulo. Permite resolver inconsistências futuras na base de dados, normalizando determinado campo com um valor por omissão quando inicialmente estava vazio.
String operations	 String operations	Este elemento permite realizar algumas operações sobre strings, nomeadamente alteração para maiúsculas ou minúsculas, eliminação de espaços no início ou fim da *string*, entre outras.
Replace in string	 Replace in string	É utilizada quando se pretende, principalmente com recurso a expressões regulares, alterar o valor de determinado campo em função do seu valor.

Get System Info		Esta tarefa permite aceder a informação do sistema relacionado não só com a data, mas também informações internas sobre a ferramenta, processadores, versões, etc.
Write to log		Esta tarefa permite a configuração do nível de detalhe que se pretende no ficheiro de log e quais os campos a adicionar ao ficheiro.
Database Lookup		Esta tarefa é utilizada quando é necessário fazer *lookup* a determinada tabela. Permite recolher informação de uma tabela quando existe uma condição que é verificada. Existem dois caminhos distintos no final desta tarefa, erro quando a condição não se verifica e sucesso quando o valor pretendido é encontrado.
Dummy		Tal como o nome indica, esta tarefa apenas facilita, em alguns casos, a representação de um processo, não tendo qualquer valor ou significado.
Table output		Esta tarefa permite a escrita na tabela. É utilizada normalmente no final de um processo de transformações, onde, se armazena o resultado final na tabela correspondente. Tal como acontece com a primeira tarefa aqui descrita, existem várias tarefas do mesmo género desta consoante o formato da tabela/ficheiro de saída.

Figura 15 – Representação do processo ETL em Kettle.

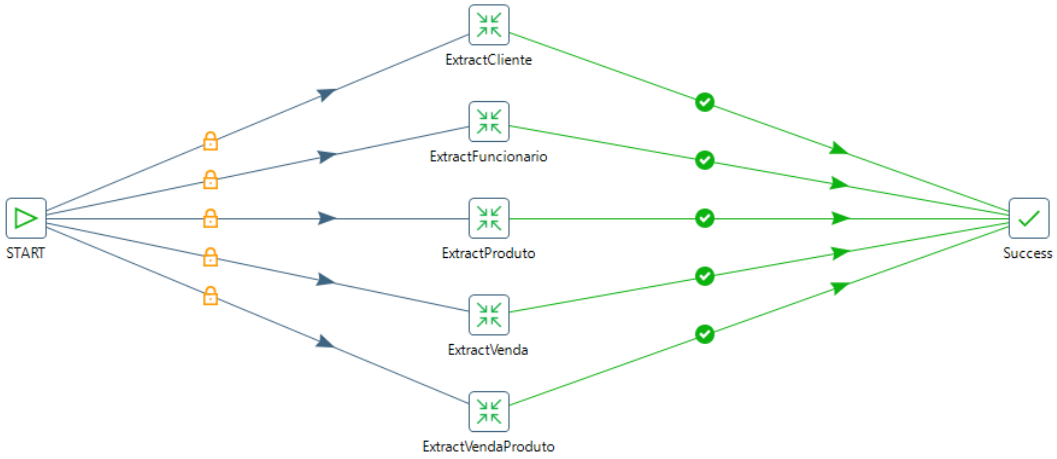


Figura 16 – Representação do processo de extração em Kettle.

Nas representações apresentadas na Figura 17 e na Figura 18 foram utilizadas tarefas específicas Kettle designadas "*dummy*" que contribuem para uma visão mais clara da representação, não tendo qualquer implicação direta na solução desenvolvida.

No processo de extração (Figura 17) foram utilizadas tarefas de *input*, onde foi extraída a metainformação de todos os campos que compõem as tabelas "Source1_Cliente" e "Source2_Cliente". Como nem todos os campos seriam necessários nas fases posteriores, a filtragem dos campos necessários foi feita através da tarefa "*select values*". Por fim, foi adicionada informação relativa à fonte de dados de onde os dados são provenientes e essa informação foi guardada numa tabela de auditoria "Audit_Cliente".

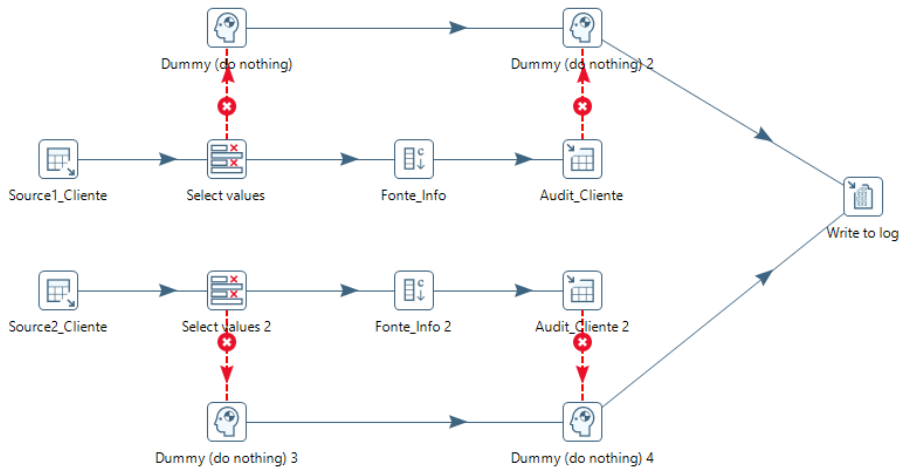


Figura 17 – Representação detalhada do processo de extração de um cliente em Kettle.

Para o processo de transformação de dados (Figura 18), foram realizadas operações de limpeza e transformação, de forma a eliminar incoerências, corrigir dados e realizar as operações necessárias. Todos os valores que não respeitaram alguma das etapas, foram enviados para quarentena para serem analisados posteriormente. Os restantes seguiram para a fase seguinte de processamento.

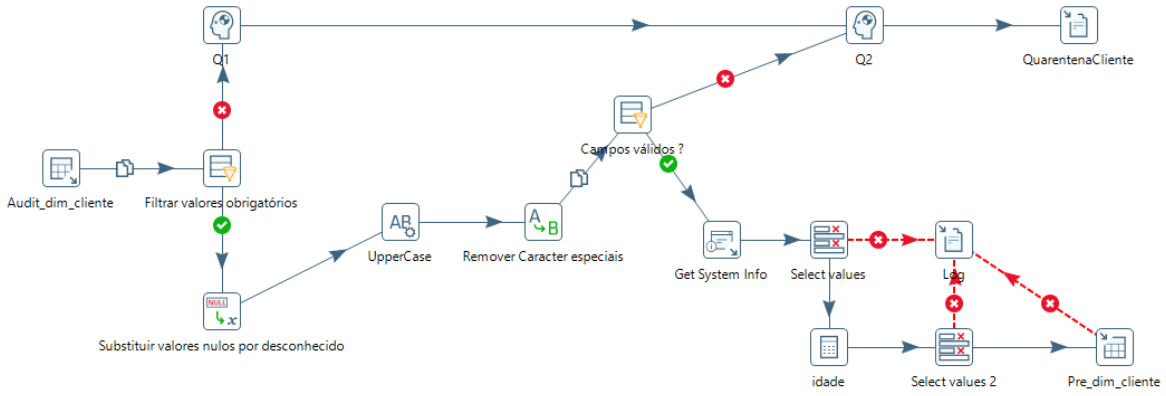


Figura 18 – Representação detalhada do processo de transformação de um cliente em Kettle.

3.4 Análise dos ficheiros XML exportados em YAWL e Kettle

Em YAWL, todas as *nets* e *sub-nets* desenvolvidas no mesmo espaço de trabalho dão origem a um ficheiro XML. Nesse ficheiro, como se pode observar na Figura 19, é possível encontrar informações sobre a especificação desenvolvida, desde a *metadata* associada até à informação sobre cada *sub-net* criada (representado em XML por *decompositions*), como também informação sobre a disposição de cada elemento na área de trabalho (representado por *layout*).

```
<?xml version="1.0" encoding="UTF-8"?>
<specificationSet xmlns="http://www.yawlfoundation.org/yawlschema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="4.0"
xsi:schemaLocation="http://www.yawlfoundation.org/yawlschema
http://www.yawlfoundation.org/yawlschema/YAWL_Schema4.0.xsd">
  <specification uri="ETL_main">
    <documentation></documentation>
    <metaData>
    </metaData>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" />
    <decomposition id="Extract" xsi:type="NetFactsType">
    </decomposition>
    <decomposition id="ExtractCliente" xsi:type="NetFactsType">
    </decomposition>
    <decomposition id="ExtractFuncionario" xsi:type="NetFactsType">
    </decomposition>
    …
  </specification>
  <layout>
    <locale language="en" country="US"/>
    <specification id="ETL_main" defaultBgColor="-526351">
      <size w="58" h="28"/>
      <net id="ExtractCliente" bgColor="-526351">
      </net>
      <net id="ExtractProduto" bgColor="-526351">
      </net>
    …
    </specification>
  </layout>
</specificationSet>
```

Figura 19 – Representação em XML de uma especificação YAWL.

Tal como acima referido, cada *net* ou *sub-net* desenvolvida designa-se por *decomposition* no ficheiro XML. Este elemento é composto pelo seu id que corresponde ao nome do elemento criado em YAWL. Dentro de cada *decomposition* (Figura 20), existe uma *tag* designada por *processControlElements* que tem definido cada elemento criado na respetiva *net* ou *sub-net*. Neste exemplo, podem ver-se não só os elementos *InputCondition* e *OutputCondition* que estão sempre presentes, como também, quais as tarefas associadas que podem ser do tipo *task* ou *condition*, que, como o nome indica, dizem respeito às tarefas simples e às condições representadas.

```
<decomposition id="ExtractCliente" xsi:type="NetFactsType">
    <processControlElements>
        <inputCondition id="InputCondition_4">
        </inputCondition>
        <task id="source1.audcliente">…
        </task>
        <task id="select">…
        </task>
      …
        <condition id="Error">…
        </condition>
        <outputCondition id="OutputCondition_4" />
    </processControlElements>
</decomposition>
```

Figura 20 – Extração XML do elemento *decomposition* em YAWL.

Dentro de cada *task/condition* (Figura 21), é adicionada informação sobre o seu nome e a sua documentação, que será utilizada mais tarde para facilitar o processo de conversão para processos Kettle. No que diz respeito ao *flowsInto*, é definido qual é o elemento seguinte, podendo existir mais que um, como no caso das condições ou tarefas de *split*.

```
<task id="source1.audcliente">
    <name>source1.audcliente</name>
    <documentation>TableInput</documentation>
    <flowsInto>
        <nextElementRef id="select_1" />
    </flowsInto>
    <join code="xor" />
    <split code="and" />
    <resourcing>
        <offer initiator="user" />
```

```
        <allocate initiator="user" />
        <start initiator="user" />
    </resourcing>
</task>
```

Figura 21 – Extração XML do elemento *task* em YAWL.

Na secção *layout* é detalhado tudo o que diz respeito ao aspeto da especificação (Figura 22). Cada *net* tem a sua própria configuração, como exemplificado no extrato do ficheiro principal no início desta secção. As primeiras *tags* correspondentes à *net ExtractCliente* dizem respeito às coordenadas geométricas da tarefa na área de trabalho. Como esta tarefa é composta por uma *sub-net*, cada *container* e cada *flow* irão especificar cada elemento definido no processo de extração de um cliente.

```
<net id="ExtractCliente" bgColor="-526351">
  <bounds x="-494" y="0" w="1720" h="484"/>
  <frame x="0" y="0" w="1133" h="358"/>
  <viewport x="0" y="0" w="1133" h="358"/>
  <container id="select">
  </container>
  <container id="source1.audcliente">
  </container>
  …
  <flow source="source1.audcliente" target="select_1">
  </flow>
  …
</net>
```

Figura 22 – Extração XML da especificação do *layout* em YAWL.

O *container* (Figura 23) é composto pela informação do *vertex*, que diz respeito ao posicionamento e cor do elemento criado e a informação da *label*, que é o posicionamento do texto que descreve o elemento.

```
<container id="source1.audcliente">
    <vertex>
      <attributes>
        <bounds x="240" y="116" w="32" h="32"/>
        <foregroundColor>-4144960</foregroundColor>
      </attributes>
    </vertex>
```

```
    <label>
      <attributes>
        <bounds x="202" y="148" w="109" h="20"/>
      </attributes>
    </label>
</container>
```

Figura 23 – Extração XML do elemento *container* em YAWL.

Por último, o *flow* (Figura 24) diz respeito às ligações que existem entre os diversos elementos, definindo a direção da ligação através da informação da origem e do destino.

```
<flow source="source1.audcliente" target="select_1">
    <ports in="13" out="12"/>
    <attributes>
      <lineStyle>11</lineStyle>
    </attributes>
</flow>
```

Figura 24 – Extração XML do elemento *flow* em YAWL.

Os ficheiros gerados pelo Kettle têm uma estrutura diferente, sendo que, por cada *job* ou transformação existe um ficheiro novo. Tal como explicado anteriormente, os *jobs* funcionam como controlo de fluxos de alto nível, por isso não têm demasiada informação na sua especificação. Para além das informações gerais sobre a *metadata* existem duas *tags* a considerar, as *entries* e os *hops* (Figura 25).

```
<?xml version="1.0" encoding="UTF-8"?>
<job>
    <name>RegularPopulate_main</name>
    <description/>
    <extended_description/>
    <job_version/>
    <directory></directory>
    <created_user></created_user>
    <created_date></created_date>
    <modified_user></modified_user>
    <modified_date></modified_date>
    <entries>
        <entry>
        </entry>
        …
```

```
    </entries>
    <hops>
        <hop>
        </hop>
      …
    </hops>
    <notepads>
    </notepads>
</job>
```

Figura 25 – Extração XML de um ficheiro *job* em Kettle.

Cada *entry* (Figura 26) diz respeito a uma nova tarefa definida na área de trabalho, podendo ser do tipo transformação ou *job*. O que é importante referir na construção deste ficheiro é o *filename*, que é o nome do ficheiro que está associado à tarefa que se está a criar e as coordenadas geométricas definidas pelo *xloc* e *yloc*.

```
<entry>
    <name>ExtractData</name>
    <description/>
    <type>JOB</type>
    <specification_method>filename</specification_method>
    <job_object_id/>
  <filename>&#x24;&#x7b;Internal.Job.Filename.Directory&#x7d;&#x2f;Regular
Populate_extractData.kjb</filename>
    <jobname/>
    <arg_from_previous>N</arg_from_previous>
    <params_from_previous>N</params_from_previous>
    <exec_per_row>N</exec_per_row>
    <set_logfile>N</set_logfile>
    <logfile/>
    <logext/>
    <add_date>N</add_date>
    <add_time>N</add_time>
    <loglevel>Nothing</loglevel>
    <slave_server_name/>
    <wait_until_finished>Y</wait_until_finished>
    <follow_abort_remote>N</follow_abort_remote>
    <expand_remote_job>N</expand_remote_job>
    <create_parent_folder>N</create_parent_folder>
    <pass_export>N</pass_export>
    <parameters>
```

```
        <pass_all_parameters>Y</pass_all_parameters>
    </parameters>
    <set_append_logfile>N</set_append_logfile>
    <parallel>N</parallel>
    <draw>Y</draw>
    <nr>0</nr>
    <xloc>224</xloc>
    <yloc>96</yloc>
</entry>
```

Figura 26 – Extração XML do elemento *entry* em Kettle.

Os *hops* (Figura 27), como o nome indica, definem qual é a origem e o destino, possibilitando as conexões entre tarefas.

```
<hop>
    <from>ExtractData</from>
    <to>TransformData</to>
    <from_nr>0</from_nr>
    <to_nr>0</to_nr>
    <enabled>Y</enabled>
    <evaluation>Y</evaluation>
    <unconditional>N</unconditional>
</hop>
```

Figura 27 – Extração XML do elemento *hop* em Kettle.

As transformações (Figura 28), como descrevem todo o processo, têm um ficheiro XML mais detalhado. Nesse ficheiro é possível encontrar informação sobre as conexões à base de dados, todas as tarefas definidas e respetivas ligações e tratamento de erros.

```
<?xml version="1.0" encoding="UTF-8"?>
<transformation>
    <info>
    </info>
    <connection>
    </connection>
    …
    <order>
        <hop>
        </hop>
      …
    </order>
```

```
    <step>
    </step>
    …
    <step_error_handling>
        <error>
        </error>
      …
    </step_error_handling>
    <slave-step-copy-partition-distribution>
    </slave-step-copy-partition-distribution>
    <slave_transformation>N</slave_transformation>
</transformation>
```

Figura 28 – Extração XML de uma transformação em Kettle.

Para se criarem as conexões é necessário preencher a informação necessária como se pode ver na Figura 29.

```
<connection>
    <name>Source1</name>
    <server>35.242.182.181</server>
    <type>MYSQL</type>
    <access>Native</access>
    <database>source1</database>
    <port>3306</port>
    <username>root</username>
    <password>Encrypted 2be98afc86aa79088ae17aa75ccc3fd89</password>
    <servername/>
    <data_tablespace/>
    <index_tablespace/>
    <attributes>
    </attributes>
</connection>
```

Figura 29 – Extração XML do elemento *connection* em Kettle.

Tal como no ficheiro anterior, também aqui (Figura 30) os *hops* fazem parte da estrutura do ficheiro para representarem as ligações.

```
<order>
    <hop>
        <from>Source1_Cliente</from>
        <to>Select values</to>
        <enabled>Y</enabled>
```

```
    </hop>
    …
</order>
```

Figura 30 – Extração XML do elemento *hops* em Kettle.

Todos os *steps* (Figura 31) têm uma especificação diferente, uma vez que cada um deles diz respeito a uma determinada função. Neste exemplo podemos ver como é definido um *TableInput*. Como cada tipo de operação em Kettle tem uma especificação diferente, foi necessário analisar as transformações mais utilizadas e perceber que tipo de informação é necessária.

```
<step>
    <name>Source1_Cliente</name>
    <type>TableInput</type>
    <description/>
    <distribute>Y</distribute>
    <custom_distribution/>
    <copies>1</copies>
    <partitioning>
        <method>none</method>
        <schema_name/>
    </partitioning>
    <connection>Source1</connection>
    <sql>SELECT&#xd;&#xa;  CC&#xd;&#xa;, NIF&#xd;&#xa;, Nome&#xd;&#xa;,
Telemovel&#xd;&#xa;, DataNascimento&#xd;&#xa;, Cidade&#xd;&#xa;,
Profiss&#xe3;o&#xd;&#xa;, tipoOperacao&#xd;&#xa;, dataOperacao&#xd;&#xa;FROM
audcliente&#xd;&#xa;</sql>
    <limit>0</limit>
    <lookup/>
    <execute_each_row>N</execute_each_row>
    <variables_active>N</variables_active>
    <lazy_conversion_active>N</lazy_conversion_active>
    <cluster_schema/>
    <remotesteps>
        <input>
        </input>
         <output>
        </output>
    </remotesteps>
    <GUI>
        <xloc>112</xloc>
        <yloc>128</yloc>
```

```
        <draw>Y</draw>
    </GUI>
</step>
```

Figura 31 – Extração XML do elemento *step* em Kettle.

3.5 Sistema de geração de esqueletos ETL

Depois de analisados os ficheiros XML, deu-se início ao processo de desenvolvimento de um sistema que fosse capaz de gerar ficheiros Kettle com base na especificação contida num ficheiro YAWL. Para isso, como mencionado anteriormente, durante a especificação YAWL foram definidas algumas regras que ajudaram a simplificar o processo. O primeiro passo foi fazer a definição, nas propriedades da especificação YAWL, que são globais a todo o projeto, das informações relativas às conexões à base de dados (Figura 32). Esta informação é essencial para que, em Kettle, seja possível aceder à estrutura das tabelas e se consiga dar início às transformações.

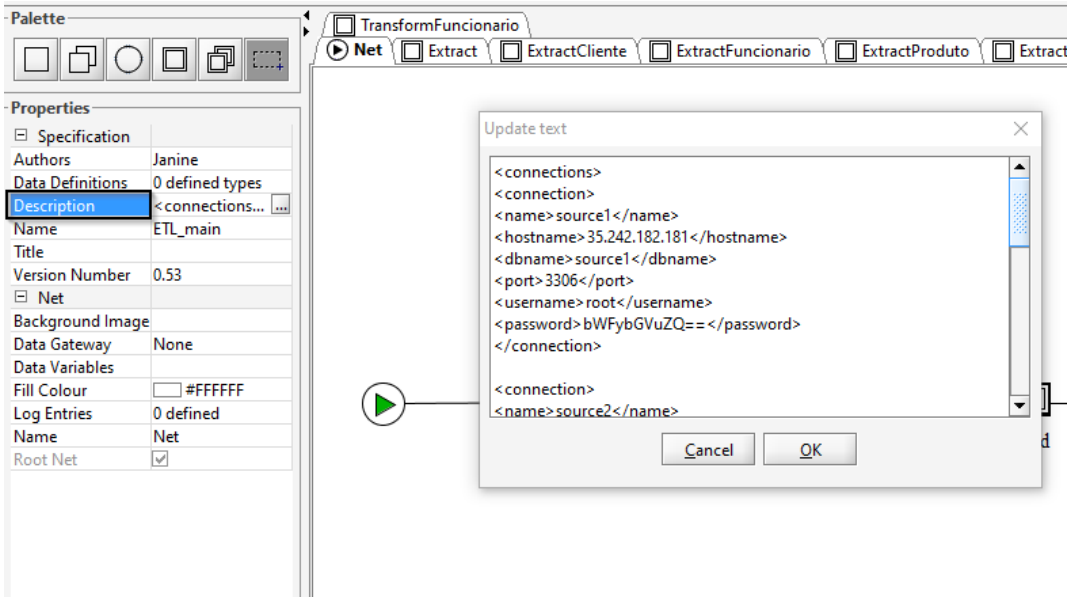


Figura 32 – Definição das conexões à base de dados em YAWL.

O formato escolhido para descrever esta informação foi XML, uma vez que este é a base de todo o processamento do sistema de geração. A informação sobre cada conexão é armazenada numa lista de conexões que servirá mais tarde para ajudar a preencher o ficheiro Kettle (Figura 33). Por cada

conexão definida em YAWL, é armazenada informação relativa ao seu nome, *hostname*, nome da base de dados e informação relativa ao utilizador e respetiva palavra-passe.

```
Node desc = doc.getElementsByTagName("description").item(0);
if (!(desc.getTextContent().isEmpty()) && !(desc.getTextContent().equals("No
description provided"))) {
   DocumentBuilderFactory dbFac = DocumentBuilderFactory.newInstance();
   DocumentBuilder dBuild = dbFac.newDocumentBuilder();
   Document d = dBuild.parse(new
ByteArrayInputStream(desc.getTextContent().getBytes()));
   NodeList conns = d.getElementsByTagName("connection");

   for (int i = 0; i < conns.getLength(); i++) {
      Node conn = conns.item(i);
      if (conn.getNodeType() == Node.ELEMENT_NODE) {
         Element e = (Element) conn;
         String name = e.getElementsByTagName("name").item(0).getTextContent();
         String host = e.getElementsByTagName("hostname").item(0).getTextContent();
         String dbname = e.getElementsByTagName("dbname").item(0).getTextContent();
         String port = e.getElementsByTagName("port").item(0).getTextContent();
         String username =
e.getElementsByTagName("username").item(0).getTextContent();
         String password =
e.getElementsByTagName("password").item(0).getTextContent();
         con = new Connections(name, host, dbname, port, username, password);
      }
      Lconn.add(con);
   }
}
```

Figura 33 – Extração das conexões definidas em YAWL.

Para além das propriedades globais, existem também, em YAWL, propriedades que dizem respeito a cada tarefa. No campo *documentation*, foram guardadas informações acerca do tipo de tarefa que se está a representar usando a nomenclatura do Kettle. Na Tabela 4 podemos ver a descrição de algumas tarefas descritas longo desta dissertação.

Tabela 4 – Informação sobre algumas tarefas definidas no sistema de geração de esqueletos

Tipo de Tarefa	Nomenclatura Kettle	Descrição
Interação com Base de Dados	TableInput	Permite a interação com a base de dados, desde leitura/escrita ou pesquisa de elementos.
	TableOutput	
	DBLookup	
Seleção dos dados	SelectValues	Permite eliminar as colunas não necessárias e fazer a filtragem dos dados com má qualidade
	FilterRows	
	IfNull	
Transformação/Operações sobre os dados	StringOperations	Permite a alteração do valor dos dados
	ReplaceString	
	Constant	
	SystemInfo	
	Calculator	
Erros	Log	Permite o tratamento de erros/ficheiros de Log
	Error	

O nome da tarefa, em YAWL designado por *label*, também é essencial, uma vez que, as tarefas que interagem com a base de dados foram definidas utilizando o formato `<NomeBaseDados>.<NomeTabela>.`

Na Figura 34 é possível observar a função que permite guardar em diferentes variáveis o nome da base de dados e da tabela, separando-os através do caractere ".". Quando o nome da tabela declarado em YAWL é composto por um número, esse número é eliminado uma vez que na base de dados não existe nenhuma tabela constituída por letras e números.

```
if (documentation != null) {
    if (documentation.equals("TableOutput") || documentation.equals("TableInput") ||
documentation.equals("DBLookup")) {
        String[] parts = name.split("\\.");
        db = parts[0];
        table = parts[1];
        String[] tableNum = table.split("_\\d+");
        table = tableNum[0];
```

```
    }
}
```

Figura 34 – Extração do nome da base de dados e tabela através do nome atribuído à tarefa em YAWL.

Na Figura 35 pode observar-se a especificação, em YAWL, de uma tarefa de extração, neste caso de clientes, que servirá de exemplo, facilitando a explicação/demonstração do sistema gerado.

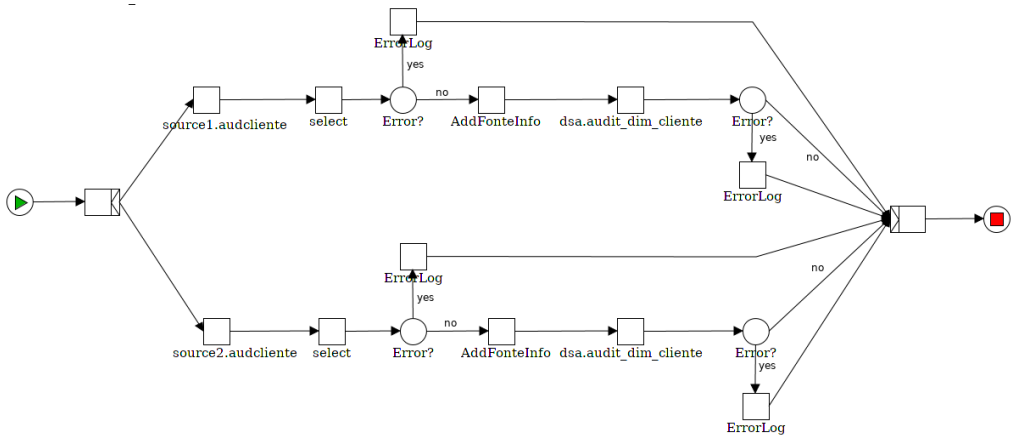


Figura 35 – Extração Cliente em YAWL.

Tal como explicado anteriormente, todas as tarefas que interagem de alguma maneira com a base de dados são definidas usando um formato próprio. Neste caso, para a base de dados designada por "source1" e "source2" e para as tabelas "audcliente", o sistema guarda o nome da base de dados e da tabela, para que, ao converter para ficheiros Kettle seja possível obter informação dos metadados respetivos. Por sua vez, o sistema, sabendo que se trata de um "*tableInput*" irá definir o XML correspondente, permitindo a sua representação em Kettle. Através da informação da base de dados guardada previamente (Figura 36), também é possível obter a informação dos dados presentes nas tabelas (Figura 37).

```
Element sql = doc.createElement("sql");
   sql.appendChild(doc.createTextNode("SELECT * FROM " + e.getTable()));
   step.appendChild(sql);
```

Figura 36 – Definição da *query* utilizada para obter informação sobre a tabela.

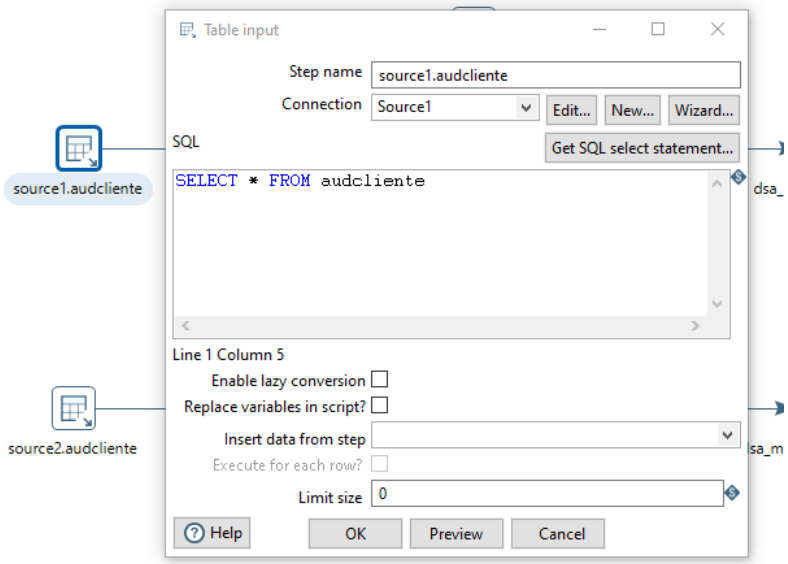


Figura 37 – Exemplo da representação de uma tarefa "*TableInput*".

A tarefa que se segue (Figura 38) permite selecionar quais as colunas da base de dados que se pretendem usar. Desta forma, a informação sobre o nome das colunas que dizem respeito a uma determinada tabela são armazenadas numa lista que será usada para construir o XML correspondente a esta tarefa.

```
List<String> fields = new ArrayList<>();
Statement stmt;
try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT COLUMN_NAME FROM
INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA ='" + dbName + "' AND TABLE_NAME = '" +
table + "';");
    while (rs.next()) {
        String name = rs.getString("COLUMN_NAME");
        fields.add(name);
    }
} catch (SQLException ex) {
    Logger.getLogger(InterfaceDB.class.getName()).log(Level.SEVERE, null, ex);
}
return fields;
}
```

Figura 38 – Definição da *query* utilizada para guardar informação sobre o nome das colunas de uma tabela.

Esta informação é utilizada para compor o ficheiro XML, onde, para cada coluna da tabela é criada uma nova *tag*, designada por "*field*", (Figura 39).
Quando não existem colunas nessa tabela para adicionar, é adicionado uma "field" vazio para garantir a consistência do ficheiro XML (existe sempre um campo "*field*" por omissão).

```
Element fields = doc.createElement("fields");
if (dbFields == null) {
    Element field = doc.createElement("field");
    Element fname = doc.createElement("name");
    fname.appendChild(doc.createTextNode(""));
    field.appendChild(fname);
    Element frename = doc.createElement("rename");
    field.appendChild(frename);
    fields.appendChild(field);
} else {
    for (String f : dbFields) {
        Element field = doc.createElement("field");
        Element fname = doc.createElement("name");
        fname.appendChild(doc.createTextNode(f));
        field.appendChild(fname);
        Element frename = doc.createElement("rename");
        field.appendChild(frename);
        fields.appendChild(field);
    }
}
step.appendChild(fields);
```

Figura 39 – Definição do XML que permite representar o nome de cada coluna da tabela nas tarefas em Kettle.

Na Figura 41 é possível observar a representação desta tarefa, com o preenchimento automático dos campos da tabela em questão. Este preenchimento só é possível porque foi adicionada informação sobre a o nome da base de dados e da tabela a todas as tarefas. Esta informação adicional é propagada para todos os nodos seguintes que não têm qualquer informação sobre a base de dados. Isto significa que, por exemplo, a tarefa "*Select*" não tem definido na sua especificação YAWL qual o nome da base de dados à qual deverá aceder. Mas, como a tarefa antecedente tem essa informação, isto é propagado, passando a tarefa "*Select*" a ter informação necessária para preencher automaticamente o valor das colunas da tabela correspondente. Isto acontece para todas as tarefas, mesmo que não estejam diretamente ligadas a uma tarefa que tenha especificado essa

informação em YAWL. Este processo foi implementado com a função apresentada na Figura 40, que demonstra o algoritmo de travessia de grafos implementado que permitiu percorrer todos os seus "pais" até encontrar a informação necessária.

```
public static void addDBInfo(Net net) {
    Map<String, ControlElement> elems = net.getDecompositions();
    for (ControlElement e : elems.values()) {
        if (e.getDb() == null) {
            Queue<String> queue = new ArrayDeque<>();
            if (e.getFrom() != null) {
                queue.addAll(e.getFrom());
            }
            while (!queue.isEmpty()) {
                String p = queue.element();
                if (elems.get(p).getDb() != null) {
                    e.setDb(elems.get(p).getDb());
                    e.setTable(elems.get(p).getTable());
                    break;
                }
                queue.remove();
                if (elems.get(p).getFrom() != null) {
                    queue.addAll(elems.get(p).getFrom());
                }
            }
        }
    }
}
```

Figura 40 – Algoritmo utilizado para adicionar a informação sobre a base de dados nos nodos onde não existia.

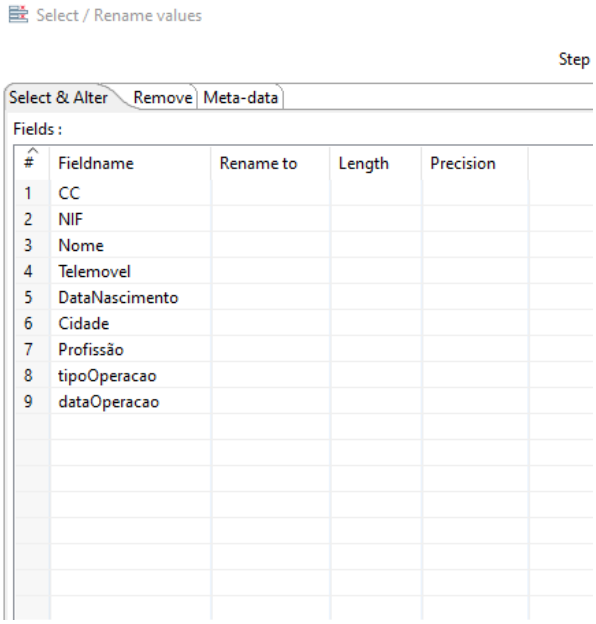


Figura 41 – Representação da tarefa *SelecValues* em Kettle.

O tratamento de erros em Kettle é definido usando uma representação de diferente cor e formato. Para isso, em XML é definida uma zona que se designa por "*step_error_handling*" na qual são aglomerados todos os nodos de erro, especificando, entre outros, qual a sua origem e destino. Na Figura 42 é possível observar a composição XML correspondente. É importante referir que a *tag* "*is_enabled*" deve estar preenchida com o valor "Y" para que seja visível a sua representação na interface.

```
Node stepError = doc.getElementsByTagName("step_error_handling").item(0);
Element error = null;
if (stepError.getNodeType() == Node.ELEMENT_NODE) {
    Element x = (Element) stepError;
    Element aux;
    for (String errorTarget : e.getError()) {
        for (String errorSource : e.getFrom()) {
            error = doc.createElement("error");
            Element source = doc.createElement("source_step");
            source.appendChild(doc.createTextNode(errorSource));
            error.appendChild(source);
            Element target = doc.createElement("target_step");
            target.appendChild(doc.createTextNode(errorTarget));
            error.appendChild(target);
            Element enabled = doc.createElement("is_enabled");
```

```
            enabled.appendChild(doc.createTextNode("Y"));
            error.appendChild(enabled);
            Element nr_valuename = doc.createElement("nr_valuename");
            error.appendChild(nr_valuename);
            Element descriptions_valuename =
doc.createElement("descriptions_valuename");
            error.appendChild(descriptions_valuename);
            Element fields_valuename = doc.createElement("fields_valuename");
            error.appendChild(fields_valuename);
            Element codes_valuename = doc.createElement("codes_valuename");
            error.appendChild(codes_valuename);
            Element max_errors = doc.createElement("max_errors");
            error.appendChild(max_errors);
            Element max_pct_errors = doc.createElement("max_pct_errors");
            error.appendChild(max_pct_errors);
            Element min_pct_rows = doc.createElement("min_pct_rows");
            error.appendChild(min_pct_rows);
        }
        stepError.appendChild(error);
    }
}
```

Figura 42 – Definição XML que permite adicionar o tratamento de erros.

A tarefa que permite adicionar informação da base de dados correspondente, em Kettle é definida como "*Constant*". Entre todos os elementos que compõem o XML, é importante realçar que as principais propriedades são o nome da constante que se pretende criar e qual o seu tipo. Neste exemplo (Figura 43) é possível observar a criação de um novo "*field*", definido como "fonte" do tipo "*Integer*".

```
Element field = doc.createElement("field");
fields.appendChild(field);

Element field_name = doc.createElement("name");
field_name.appendChild(doc.createTextNode("fonte"));
field.appendChild(field_name);

Element field_type = doc.createElement("type");
field_type.appendChild(doc.createTextNode("Integer"));
field.appendChild(field_type);
```

Figura 43 – Definição XML da tarefa "*Constant*".

Para isso, de maneira a diferenciar as duas bases de dados, a "*source1*" ficou definida como fonte '1' e a outra como fonte '2'. Desta maneira, em Kettle, é possível observar (Figura 44) que, nas propriedades da tarefa, a variável fonte é do tipo inteiro e tem como valor o número '1'.

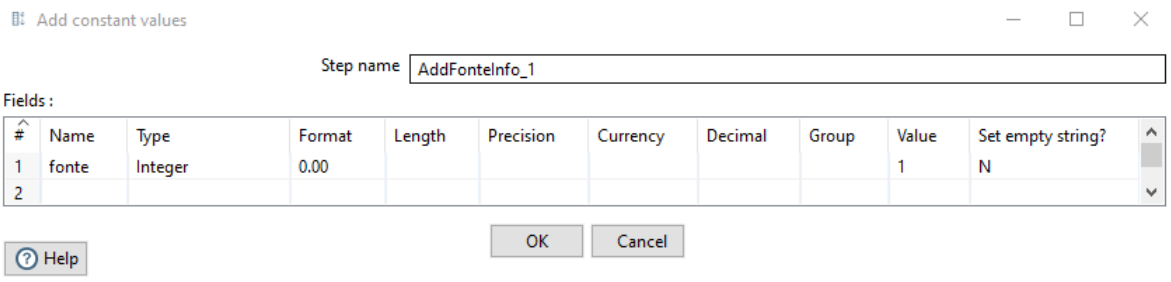


Figura 44 – Representação da tarefa "*Constant*" em Kettle.

Por último, a tarefa que permite a inserção dos valores extraídos na área de retenção é designada por "*TableOutput*". Tal como explicado anteriormente, para a tarefa de "*TableInput*" (Figura 45), através do nome da base de dados e tabela, é possível obter o nome de todas as colunas, permitindo assim a sua representação automática nas propriedades da tarefa.

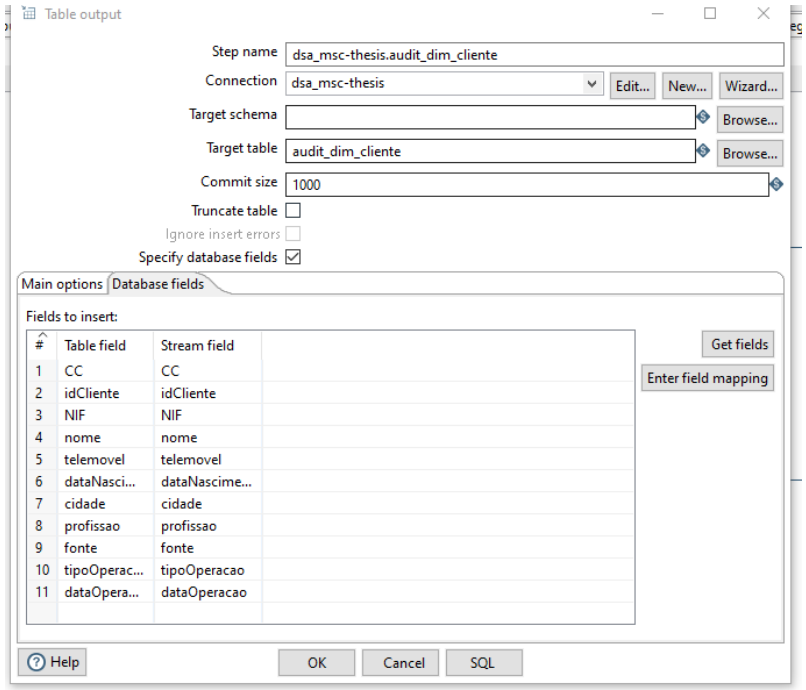


Figura 45 – Representação da tarefa "*tableOutput*" em Kettle.

Depois de analisado, em detalhe, a construção de cada tarefa, na Figura 46 é possível observar o ficheiro que foi gerado automaticamente, tendo por base a representação YAWL definida na Figura 35. Todas as tarefas têm tradução direta, à exceção do tratamento de erros, uma vez que é preciso remover a condição "Error" definida em YAWL e atualizar os valores das suas ligações. Esta atualização é feita sempre que o valor da variável da ligação é "*no*", o que significa que não ocorreu nenhum erro. Para a especificar, foram utilizados apontadores onde, sempre que o nodo seguinte é uma tarefa condição, o seu antecedente passa a apontar para o sucessor da condição. Quando a variável da ligação tem o valor "*yes*" significa que a tarefa seguinte terá de ser um ficheiro de *log* ou de quarentena. Esta informação, como já referido, é preenchida na parte final do XML que diz respeito à maneira como são tratados os erros, especificando qual é o nodo de origem e o nodo de destino. No final, estes erros são representados, em Kettle, pela seta vermelha. Na Figura 46 é possível observar o resultado final da geração do processo de extração, neste caso, de um cliente.

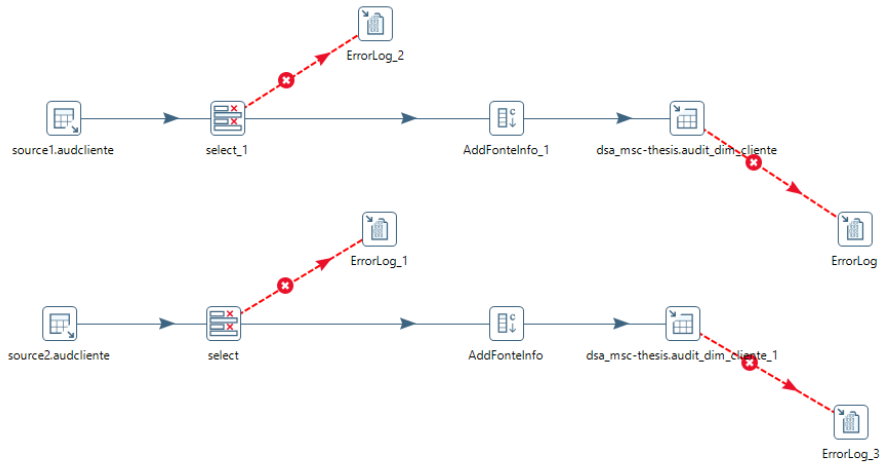


Figura 46 – Extração Cliente em Kettle.

Todas as tarefas de extração de dados das tabelas de auditoria para a área de retenção conseguem ser todas executadas em Kettle, sem ser necessário configurar ou ajustar qualquer pormenor de especificação. Todas as outras etapas que foram representadas em Kettle são consideradas esqueletos, uma vez que é necessário a configuração de algumas propriedades para que seja possível a sua execução. É o caso das tarefas usadas durante o processo de transformação de dados, uma vez que estas tarefas possuem características muito especificas, como, por exemplo, enviar para quarentena valores que não cumpram determinado formato, valores nulos ou cálculos de valores, o que faz com que não seja possível fazer a sua tradução exata.

Capítulo 4

Validação do Sistema Desenvolvido

4.1 Surrogate Key Pipeline

De forma a validar o sistema desenvolvido, foi escolhido, como exemplo de validação, um dos processos típicos de ETL designado por *Surrogate Key Pipeline* (SKP). Usualmente, um processo deste género é responsável pela conversão de chaves naturais pelas respetivas chaves de substituição, antes de se inserir os valores, ditos finais, num *Data Warehouse*. Existem diversas maneiras de implementar este processo. Contudo, neste exemplo serão utilizadas tabelas de *lookup* para obter de maneira eficiente o valor da chave de substituição, com base na chave(s) naturais apresentada(s). Estes valores são gerados, por norma, de forma sequencial e automática, não contendo qualquer significado, ao contrário das chaves naturais que, em alguns casos, para além de puderem conter caracteres não numéricos, podem também conter informação sobre números de conta de clientes, identificações pessoais, etc.. Assim, torna-se necessário criar uma tabela de *lookup* por cada dimensão, permitindo que, no final do processo, a tabela de factos seja povoada com informação sobre as suas dimensões e medidas definidas. Na Figura 48 é possível observar a representação deste processo em YAWL.

Depois de desenvolvida a especificação em YAWL do processo SKP, através do sistema desenvolvido, foi possível obter a sua representação em Kettle. Uma vez validado o processo de conversão de

especificações YAWL para a ferramenta ETL escolhida (Kettle), foi também desenvolvido um processo de conversão, usando este exemplo, para duas das ferramentas ETL mais reconhecidas no mercado, que ocupam o primeiro quadrante do quadro mágico da Gartner (Gartner (Agosto 2019)) no que diz respeito a ferramentas de integração de dados, o Informatica PowerCenter e o Talend (Figura 47).

Figura 47 – Quadrante mágico para ferramentas de integração de dados, fonte Gartner (Agosto 2019).

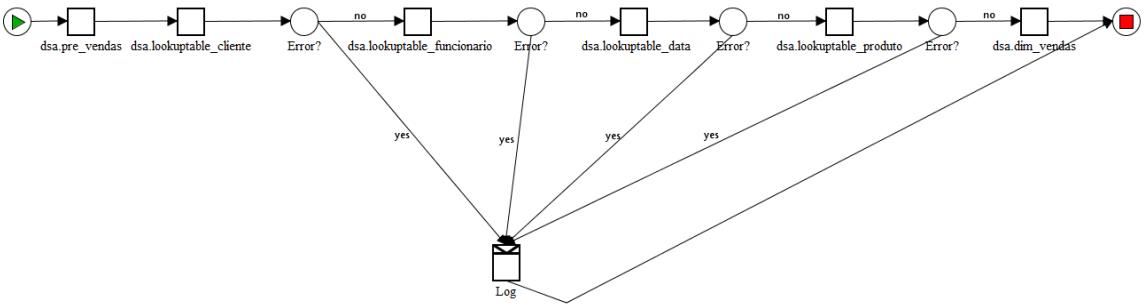


Figura 48 – Representação do processo SKP em YAWL.

4.2 Representação em Kettle

Tal como demonstrado anteriormente, a representação em Kettle é bastante semelhante à especificação em YAWL. Neste caso, em particular, o principal desafio foi o tratamento das condições de erro que, em YAWL são representados por um novo nodo, enquanto que em Kettle não existe essa necessidade. Desta forma, o maior obstáculo aqui foi saber como eliminar as condições de erro e especificar que a conexão existente entre cada *lookup* e a tarefa condição são eliminadas, passando a existir apenas a conexão direta entre as diversas tabelas de *lookup*.

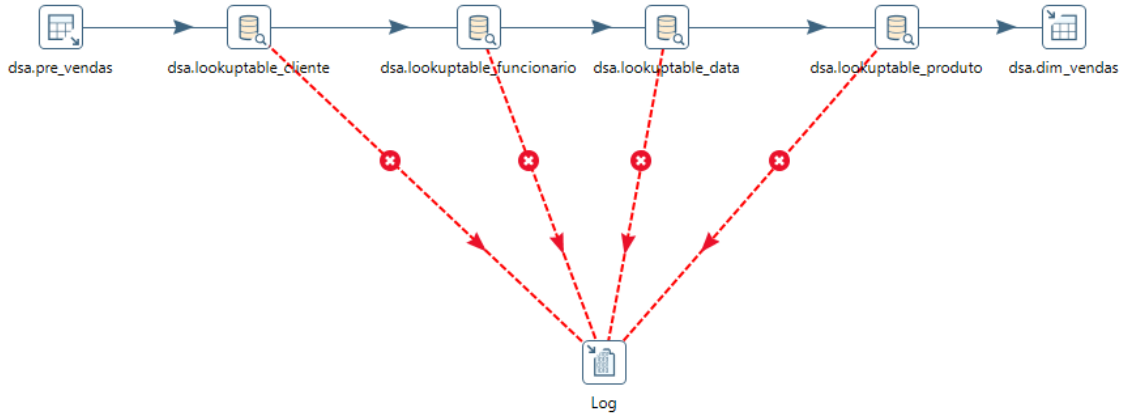


Figura 49 – Representação SKP em Kettle.

4.3 Representação em Informatica PowerCenter

O primeiro passo, para representar o processo SKP utilizando o Informatica, foi perceber quais as transformações que existiam na ferramenta que poderiam ser utilizadas. Facilmente se percebeu que, sendo uma ferramenta ETL convencional, as transformações são muito semelhantes às previamente estudadas para a ferramenta Kettle. A principal diferença é que na ferramenta Informatica, as transformações *lookup* não têm duas opções de saída (com sucesso e erro). O que acontece é que, quando a transformação não retorna qualquer valor (*null*), significa que não foi encontrado na tabela de *lookup* nenhum valor que coincidiu, obrigando assim à criação de uma transformação designada "*RouterTransformation*". Esta transformação funciona como um filtro no qual, sempre que um valor de qualquer uma das transformações de *lookup* retornar um valor nulo,

é preenchido uma nova entrada no ficheiro de log. Por sua vez, se for possível encontrar na tabela de *lookup* o valor pretendido, é preenchida a tabela de factos "vendas". Na Figura 50 pode observar-se a sua representação.

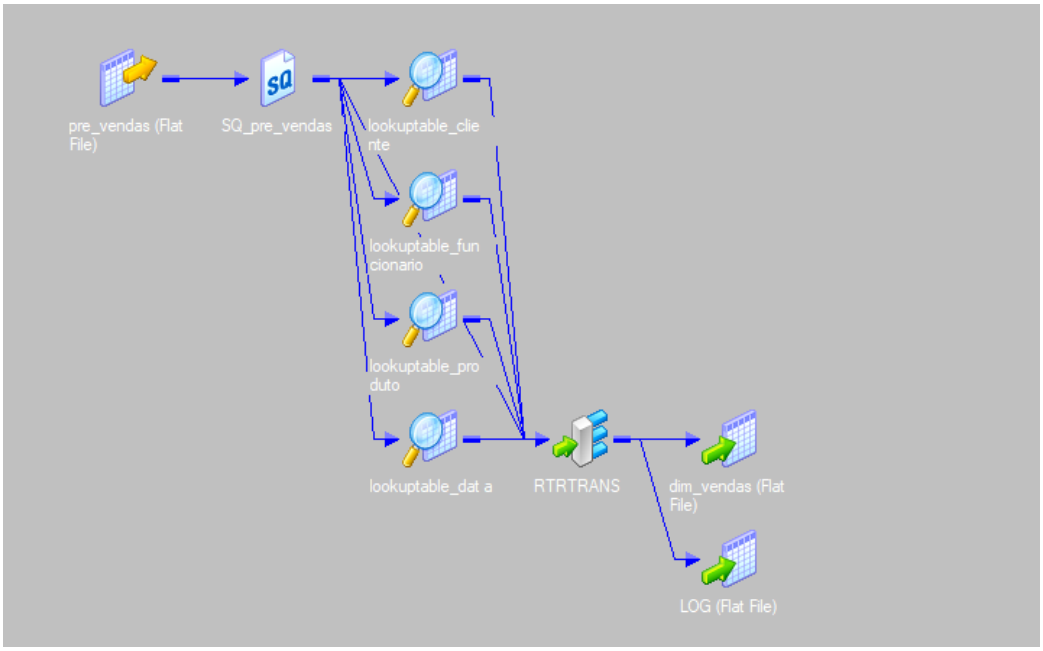


Figura 50 – Representação do processo SKP em Informatica.

4.3.1 Análise do Ficheiro XML em Informatica

O ficheiro XML de um processo definido em Informatica (Figura 51), é composto, essencialmente pelo "*mapping*", que é o espaço no qual são definidas as diferentes etapas do processo, a "*source*" e a "target" nas quais são definidos os metadados dos ficheiros de entrada e de saída. Toda esta informação é armazenada numa pasta, existente ou criada de novo, no repositório no qual o servidor do Informatica está instalado.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<POWERMART CREATION_DATE="05/22/2019 10:05:03" REPOSITORY_VERSION="187.96">
 <REPOSITORY CODEPAGE="UTF-8" DATABASETYPE="Oracle" NAME="repository_eqz_dev"
VERSION="187">
     <FOLDER DESCRIPTION="" GROUP="" NAME="Janine" OWNER="janine" PERMISSIONS="rwx---
r--" SHARED="NOTSHARED" UUID="58619605-5cfa-4348-b580-5c2a3f7c4eef">
```

```
        <MAPPING DESCRIPTION="" ISVALID="YES" NAME="SKP" OBJECTVERSION="1"
VERSIONNUMBER="1">
            <TRANSFORMATION DESCRIPTION="" NAME="lookuptable_cliente"
OBJECTVERSION="1" REUSABLE="NO" TYPE="Lookup Procedure" VERSIONNUMBER="1">
             </TRANSFORMATION>
             …
             <INSTANCE DESCRIPTION="" NAME="lookuptable_cliente"
TRANSFORMATION_NAME="lookuptable_cliente" TRANSFORMATION_TYPE="Lookup Procedure"
TYPE="TRANSFORMATION"/>
             …
             <CONNECTOR FROMFIELD="SK" FROMINSTANCE="pre_vendas"
FROMINSTANCETYPE="Source Definition" TOFIELD="SK" TOINSTANCE="SQ_pre_vendas"
TOINSTANCETYPE="Source Qualifier"/>
             …
        </MAPPING>
        <TARGET BUSINESSNAME="" CONSTRAINT="" DATABASETYPE="Flat File" DESCRIPTION=""
NAME="dim_vendas" OBJECTVERSION="1" TABLEOPTIONS="" VERSIONNUMBER="1"></TARGET>
        <SOURCE BUSINESSNAME="" DATABASETYPE="Flat File" DBDNAME="Flat_File"
DESCRIPTION="" NAME="pre_vendas" OBJECTVERSION="1" OWNERNAME="Janine"
VERSIONNUMBER="1"></SOURCE>
    </FOLDER>
 </REPOSITORY>
</POWERMART>
```

Figura 51 – Representação geral de um processo YAWL em XML.

4.3.2 Geração de Esqueletos para Informatica

Depois de se analisar e conhecer bem a estrutura do ficheiro XML obtido a partir das especificações YAWL, começou-se a desenvolver o sistema de geração de esqueletos para a ferramenta Informatica. Uma vez que a especificação YAWL foi enriquecida com informações necessárias para a sua representação em Kettle, essa estrutura foi mantida e adaptada para o contexto específico da Informatica. Desta forma, sempre que em YAWL surgiam informações "*tableInput*", foi executado o método que permite criar modelos XML para a criação de "*sources*" do Informatica. Para isso, foi criada uma entrada nova na estrutura XML, na *tag* "*Source*", com o tipo da base de dados e o nome do elemento. Neste caso, não foi possível utilizar o tipo de base de dados correto uma vez que no servidor no qual o Informatica estava instalado não existia o *driver* para o tipo de base de dados usado durante a dissertação: o sistema MySQL.

Depois de adicionada o nome do ficheiro de entrada, foi necessário definir cada um dos seus elementos, consoante as colunas existentes na base de dados, tal como se pode observar na Figura 52.

```
<SOURCE BUSINESSNAME="" DATABASETYPE="Flat File" DBDNAME="Flat_File" DESCRIPTION=""
NAME="pre_vendas" OBJECTVERSION="1" OWNERNAME="Janine" VERSIONNUMBER="1">
    <FLATFILE CODEPAGE="US-ASCII" CONSECDELIMITERSASONE="NO" DELIMITED="YES"
DELIMITERS="," ESCAPE_CHARACTER="" KEEPESCAPECHAR="NO" LINESEQUENTIAL="NO"
MULTIDELIMITERSASAND="NO" NULLCHARTYPE="ASCII" NULL_CHARACTER="*" PADBYTES="1"
QUOTE_CHARACTER="NONE" REPEATABLE="NO" ROWDELIMITER="0" SHIFTSENSITIVEDATA="NO"
SKIPROWS="0" STRIPTRAILINGBLANKS="NO"/>
    <TABLEATTRIBUTE NAME="Base Table Name" VALUE=""/>
    <TABLEATTRIBUTE NAME="Search Specification" VALUE=""/>
    <TABLEATTRIBUTE NAME="Sort Specification" VALUE=""/>
    <TABLEATTRIBUTE NAME="Datetime Format" VALUE="A  19 mm/dd/yyyy hh24:mi:ss"/>
    <TABLEATTRIBUTE NAME="Thousand Separator" VALUE="None"/>
    <TABLEATTRIBUTE NAME="Decimal Separator" VALUE="."/>
    <TABLEATTRIBUTE NAME="Add Currently Processed Flat File Name Port" VALUE="No"/>
    <TABLEATTRIBUTE NAME="Source TimeStamp" VALUE=""/>
    <SOURCEFIELD BUSINESSNAME="" DATATYPE="number" DESCRIPTION="dsa" FIELDNUMBER="1"
FIELDPROPERTY="0" FIELDTYPE="ELEMITEM" HIDDEN="NO" KEYTYPE="NOT A KEY" LENGTH="10"
LEVEL="0" NAME="idVenda" NULLABLE="NOTNULL" OCCURS="0" OFFSET="0" PHYSICALLENGTH="10"
PHYSICALOFFSET="0" PICTURETEXT="" PRECISION="10" SCALE="0" USAGE_FLAGS=""/>
    <SOURCEFIELD BUSINESSNAME="" DATATYPE="number" DESCRIPTION="dsa" FIELDNUMBER="2"
FIELDPROPERTY="0" FIELDTYPE="ELEMITEM" HIDDEN="NO" KEYTYPE="NOT A KEY" LENGTH="10"
LEVEL="0" NAME="idProduto" NULLABLE="NOTNULL" OCCURS="0" OFFSET="0"
PHYSICALLENGTH="10" PHYSICALOFFSET="0" PICTURETEXT="" PRECISION="10" SCALE="0"
USAGE_FLAGS=""/>
    ...
</SOURCE>
```

Figura 52 – Definição XML de uma *source* em YAWL.

A função (Figura 53) que permite criar o XML apresentado na Figura 52 é responsável por, para cada coluna da tabela correspondente, guardar qual é o seu tipo de dados. Uma vez que o tipo de dados que o Informatica consegue interpretar não é compatível com o tipo de dados apresentados em MySQL, foi necessário criar uma função que mapeasse cada tipo de dados, de forma a permitir a sua representação.

```
dbFields = InterfaceDB.selectValues(conn.get(dbName), dbName, e.getTable());
```

```
int count = 0;
for (String d : dbFields) {
    String type = InterfaceDB.getDataType(conn.get(dbName), dbName, e.getTable(), d);
    count++;
    String n_type = mapTypes(type);
    Element sourceF = doc.createElement("SOURCEFIELD");
    sourceF.setAttribute("BUSINESSNAME", "");
    sourceF.setAttribute("DATATYPE", n_type);
    sourceF.setAttribute("DESCRIPTION", e.getDb());
    sourceF.setAttribute("FIELDNUMBER", "" + count);
    sourceF.setAttribute("FIELDPROPERTY", "0");
    sourceF.setAttribute("FIELDTYPE", "ELEMITEM");
    sourceF.setAttribute("HIDDEN", "NO");
    sourceF.setAttribute("KEYTYPE", "NOT A KEY");
    sourceF.setAttribute("LENGTH", precision);
    sourceF.setAttribute("LEVEL", "0");
    sourceF.setAttribute("NAME", d);
    sourceF.setAttribute("NULLABLE", "NOTNULL");
    sourceF.setAttribute("OCCURS", "0");
    sourceF.setAttribute("KEYTYPE", "NOT A KEY");
    sourceF.setAttribute("OFFSET", "0");
    sourceF.setAttribute("PHYSICALLENGTH", "10");
    sourceF.setAttribute("PHYSICALOFFSET", "0");
    sourceF.setAttribute("PICTURETEXT", "");
    sourceF.setAttribute("PRECISION", precision);
    sourceF.setAttribute("SCALE", scale);
    sourceF.setAttribute("USAGE_FLAGS", "");
    source.appendChild(sourceF);
}
```

Figura 53 – Função que permite a criação do extrato XML representado na Figura 52.

Na ferramenta Informatica, por cada "*source*" é criada automaticamente um "*Source Qualifier*". Este elemento é responsável por "converter" os tipos de dados que foram importados da "*source*", para o tipo de dados que o "*mapping*" utiliza. Para além desta função, este elemento desempenha o mesmo papel que o "*SelectValues*" no Kettle, permitindo selecionar facilmente quais as colunas da tabela que são necessárias para o processo.

O processo para adicionar a tabela de saída, "*target*", é semelhante ao descrito anteriormente. Depois de definidos os ficheiros de entrada e saída, é necessário especificar todo o processo SKP. Esta definição passa pelo preenchimento da secção "*mapping*" no ficheiro XML. Este processo é

descrito em três fases, a especificação das transformações, das instâncias e dos conectores. As transformações dizem respeito à especificação detalhada de todas as transformações, semelhante ao que foi explicado para a *source* e para o *target*. É importante especificar o tipo de transformação e, a partir daí, adicionar todos os atributos necessários. As instâncias, como próprio nome indica, servem para instanciar todos os objetos presentes, e os conectores, para atribuir ligações entre as diferentes tarefas.

4.4 Representação em Talend

Tal como no processo desenvolvido em Informatica, o método para representar o processo SKP em Talend (Figura 54) seguiu a mesma metodologia. A principal diferença relativamente às duas representações anteriores é que em Talend não existe uma tarefa ou transformação de *lookup*. Neste caso, cada vez que é necessário fazer um *lookup* a uma tabela é usada uma transformação designada por "*tMap*", que, entre as várias funções, é responsável por fazer o *join* entre duas tabelas. Também em Talend não existe previamente definido que cada tarefa tenha uma saída em caso de sucesso e outra em caso de erro. O que acontece é que o tratamento de erros e ficheiros de log são tratados ao nível do "*job*", que é o espaço no qual são definidas todas as transformações. Estes *logs* podem ser apresentados na consola do Talend ou guardados em ficheiros de texto ou em base de dados.

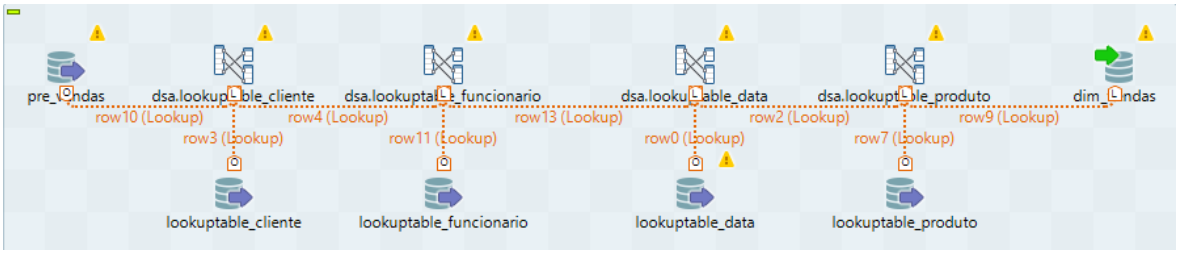


Figura 54 – Representação de um processo SKP em Talend.

4.4.1 Análise do ficheiro XML em Talend

O ficheiro XML de uma representação Talend (Figura 55) é composto essencialmente por três grupos, os "*parameters*", "*node*" e "*connection*". A primeira secção diz respeito a configurações globais que podem ser definidas para o projeto, como por exemplo, a conexão à base de dados ou informação

sobre a localização dos ficheiros de configuração ou *"logs"*. Na secção "*node*" é especificada cada tarefa utilizada. Para além da definição das características específicas de cada tabela, é também adicionada informação sobre os metadados de cada tabela que está a ser utilizada. Por último, a secção "*connection*" permite definir as ligações que existem entre as diferentes tarefas.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<talendfile:ProcessType
xmlns:talendfile="platform:/resource/org.talend.model/model/TalendFile.xsd"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" defaultContext="Default"
jobType="Standard" xmi:version="2.0">
  <context confirmationNeeded="false" name="Default"/>
  <parameters>
  </parameters>
  <node componentName="tMysqlInput" componentVersion="0.102" offsetLabelX="0"
offsetLabelY="0" posX="904" posY="232">
  </node>
…
  <connection connectorName="FLOW" label="row13" lineStyle="8"
metaname="dsa.lookuptable_funcionario" offsetLabelX="0" offsetLabelY="0"
source="dsa.lookuptable_funcionario" target="dsa.lookuptable_data">
  </connection>
…
</talendfile:ProcessType>
```

Figura 55 – Definição XML de um processo Talend.

4.4.2 Geração de Esqueletos para Talend

Depois de analisado o ficheiro XML, procedeu-se, à semelhança dos casos anteriores, à criação de um sistema de geração de esqueletos para Talend. Também neste caso, foi mantida toda a informação adicional existente que servia de base para a criação de ficheiros Kettle, adaptando cada um dos termos a este novo contexto. Tal como já explicado, em Talend não existe uma tarefa que represente uma tabela de *lookup*, por isso, na função apresentada na Figura 56 é possível observar que, sempre que em YAWL foi atribuída a documentação de "*DBLookup*" a uma tarefa, é adicionado ao XML um nodo que representará a tabela ("*tableInput*") e um nodo que permitirá fazer o *join* com a tabela principal ("*tMap*").

```
switch (e.getDocumentation()) {
    case "DBLookup":
        aux = createTMap(doc, e, globalNet);
        x.appendChild(aux);
        aux1 = createInputNode(doc, e, globalNet, "DB_" + e.getName());
        x.appendChild(aux1);
        break;
    case "TableInput":
        aux = createInputNode(doc, e, globalNet, e.getName());
        x.appendChild(aux);
        break;
    case "TableOutput":
        aux = createOutputNode(doc, e, globalNet);
        x.appendChild(aux);
        break;
    default:
        break;
}
```

Figura 56 – Função que demonstra a escolha da estrutura XML a escolher consoante o tipo de documentação definida em YAWL.

Tal como no Informatica, alguns tipos de dados têm representação diferente na base de dados e na ferramenta ETL, o que faz com que seja necessário mapear o seu tipo de dados, para que este possa ser interpretado em Talend (Figura 57).

```
private static String mapTypes(String type) {
String n_type = null;
switch (type) {
    case "int":
        n_type = "id_Integer";
        break;
    case "varchar":
        n_type = "id_String";
        break;
    case "date":
        n_type = "id_Date";
        break;
    case "float":
        n_type = "id_Float";
        break;
```

```
    default:
        break;
}
return n_type;
}
```

Figura 57 – Função usada para mapear os tipos de dados.

Uma vez que em Talend não é possível representar, por exemplo, os elementos condição representados em YAWL, é necessário tratar esses elementos e removê-los. Desta forma é possível criar ligações entre as tarefas que serão efetivamente representadas em Talend (Figura 58).

```
for (ControlElement e : res) {
    for (String hopTo : e.getTo()) {
        if (e.getDocumentation() != null) {
            if (e.getType().equals("condition")) {
                for (String from : e.getFrom()) {
                    if (globalNet.getDecompositions().containsKey(from)) {
                        globalNet.getDecompositions().get(from).setTo(e.getTo());
                    }
                }
            }
        } else {
            to_remove.add(e);
        }
    }
}
```

Figura 58 – Função que reajusta as ligações entre nodos quando surgem tarefas do tipo "*Condition*".

Através da Figura 59, é possível observar, em representação gráfica, o resultado da função anterior, em que, sempre que exista um nodo que não tem documentação ou é uma condição, esse elemento é eliminado e a ligação que existia entre esse nodo e os seus precedentes/antecedentes é reajustada.

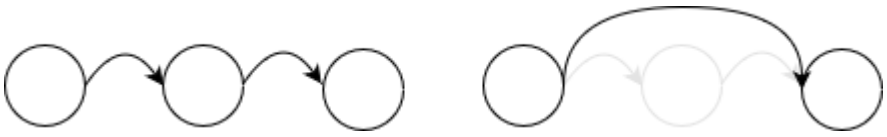


Figura 59 – À esquerda a configuração inicial, à direita a configuração depois de reajustar os nodos para representação em Talend.

Depois de configuradas as ligações, o ficheiro XML está pronto para ser importado em Talend. Ao contrário das outras duas ferramentas, nas quais existe a possibilidade de importar ficheiros num projeto já criado (diretamente na ferramenta), em Talend para fazer a importação é necessário copiar o ficheiro XML para a diretoria local correspondente ao projeto criado. É na pasta "*process*" que são adicionados todos os *jobs* que são criados por cada projeto. Depois de copiado para lá o ficheiro XML, designado por SKP.item, ao abrir o projeto no Talend é possível obter a representação que foi gerada automaticamente pelo gerador de esqueletos (Figura 54). Nesta imagem é possível observar que existe um aviso em cada tarefa gerada, o que significa que a tarefa está incompleta. Isto acontece porque foram gerados apenas esqueletos, obrigando à configuração de alguns detalhes para que possa ser executada. Alguns desses detalhes estão relacionados, por exemplo, com a condição a usar para fazer os "*join*" e quais são os elementos que devem transitar entre cada tarefa representada.

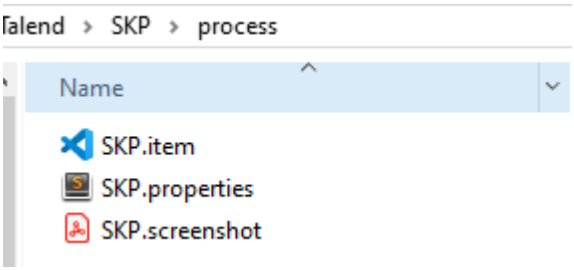


Figura 60 – Estrutura de ficheiros em Talend.

4.5 Comparação dos Resultados Obtidos

Depois de implementadas as diferentes versões do mesmo processo ETL torna-se mais fácil analisar os resultados obtidos com as diferentes implementações e, com isso, comparar as diferentes ferramentas. Para isso, foram tidos em conta alguns dos aspetos que foram determinantes durante todo o processo de construção do gerador de esqueletos ETL, nomeadamente a facilidade de aprendizagem da ferramenta, se é intuitiva ou não, o tipo de ficheiros que gera e a sua complexidade, ou o resultado final e a sua proximidade com a solução desejada e as suas limitações. Como medida de comparação foi escolhida uma escala de valores, que varia entre 1 e 5, na qual 1 significa que é simples ou fácil e 5 que é bastante complexa ou difícil. Como se pode observar na Tabela 5, de uma forma geral, a solução que melhor representa o sistema de geração de esqueletos e que fica mais

próxima da solução ideal é a representação em Kettle. Porém, no que diz respeito à facilidade de aprendizagem da ferramenta, uma vez que o foco desta dissertação foi a ferramenta Kettle, foi a esta ferramenta que dedicámos mais tempo, e, por consequência, seja esta a que se mais destaca. Por sua vez, o Talend surge como a ferramenta mais desafiante em termos de aprendizagem, uma vez que a forma como os componentes são representados e o tipo de configurações que são necessárias não são tão intuitivos e evidentes como acontece nas outras ferramentas.

Como explicado anteriormente, todas as representações podem ser exportadas no formato XML. Tanto em Informatica como em Talend é exportado um único ficheiro no qual estão definidas todas as tarefas e as suas respetivas ligações, enquanto que em Kettle existe um tipo de ficheiro diferente para transformações e para "*jobs*", o que torna a estrutura mais complexa e, por sua vez, a sua interpretação mais difícil. Embora seja mais complexo, a representação em Kettle é a que mais se assemelha à representação em YAWL, o que acaba por facilitar a sua conversão.

Tabela 5 - Comparação entre as diferentes ferramentas estudadas

	KETTLE	**INFORMATICA**	**TALEND**
CURVA DE APRENDIZAGEM	2	3	4
USABILIDADE	1	2	3
COMPLEXIDADE DA ESTRUTURA XML	4	2	1
FACILIDADE DE CONVERSÃO	2	3	3
PROXIMIDADE DO RESULTADO OBTIDO COM O IDEAL	1	3	2

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusão

Durante o desenvolvimento de um processo ETL, a fase de especificação e modelação conceptual deve ter um papel importante, não devendo ser descurada em qualquer aspeto. Apesar de reconhecidas as vantagens e a importância deste processo, a verdade é que, continuam a ser implementados vários sistemas ETL de uma forma bastante *ad-hoc*. Encontrar uma ferramenta que, de uma forma simples e eficaz, consiga especificar conceptualmente um processo ETL é uma tarefa difícil e cada vez mais procurada no meio. Com esta necessidade iminente surgem então várias linguagens e ferramentas que propõem soluções e começam a ganhar alguma popularidade. É neste enquadramento que encontramos a YAWL. Como referido ao longo desta dissertação, a YAWL é uma linguagem de fluxos de trabalho que permite o desenvolvimento e validação de um processo ETL. Conseguir de alguma maneira tirar partido do tempo e trabalho investido durante a modelação conceptual, convertendo essa especificação em processos ETL, é algo que contribui significativamente para incentivar o investimento no desenvolvimento conceptual. É neste sentido que, durante esta dissertação foi, não só reforçada a importância da modelação como também estudado e desenvolvido um sistema capaz de converter especificações YAWL em processos ETL (passíveis de serem implementadas em Kettle, Informatica e Talend).

Numa primeira abordagem, foi desenvolvido o processo de transformação para a ferramenta ETL inicialmente escolhida, o Kettle, garantido que a sua "conversão" além de ser possível é útil. Durante este processo, foram utilizados alguns dos procedimentos mais comuns num sistema de ETL, comprovando, em alguns casos que é possível gerar ficheiros prontos a executar e não apenas esqueletos de processos ETL como inicialmente esperávamos. É claro que, quanto maior a complexidade do processo, mais difícil se torna a sua representação de uma forma completa noutra linguagem. Isto acontece, por exemplo, no processo de transformação e limpeza dos dados, no qual o resultado final é apenas um esqueleto, ficando à responsabilidade da pessoa responsável pela construção do sistema ETL enriquecer as tarefas geradas com as várias necessidades impostas pela implementação. Todos estes pormenores poderiam ser adicionados à especificação YAWL. No entanto, esta deixaria de ser uma abstração do sistema e passaria a confundir-se com a própria implementação. Por essa mesma razão, optou-se por não enriquecer o ficheiro YAWL com este tipo de informação. Depois de confirmada a possibilidade de geração de ficheiros Kettle, o sistema foi validado usando um dos processos mais vulgares (e críticos) de um sistema ETL: o processo SKP. Para além da representação em Kettle, foram concebidas e desenvolvidas outras versões de geração de esqueletos ETL para Informatica e para Talend. Estas duas versões foram utilizadas para validar que o sistema desenvolvido é genérico, permitindo, através do mesmo ficheiro de entrada YAWL, gerar três soluções distintas. Através do estudo destas três ferramentas foi possível perceber que, independentemente da diferença sintática, os resultados obtidos aproximam-se bastante e que, o maior obstáculo na realização de um processo ETL não é a ferramenta que se utiliza, mas sim a definição e especificação do seu modelo conceptual. Neste caso, a YAWL teve um papel fundamental, comprovando a sua utilidade na definição deste tipo de sistemas.

5.2 Trabalho Futuro

Como trabalho futuro, seria interessante adicionar algumas melhorias ao sistema, nomeadamente a criação de avisos nas tarefas que estão incompletas e precisam de ser alteradas. No que diz respeito ao estudo dos processos ETL, seria útil dar continuidade ao estudo que foi feito com o SKP e especificar elementos como o CDC (*Change Data Capture*), SCD (*Slowly Changing Dimension*), entre outros. Seria uma mais valia para o sistema se permitisse de uma forma genérica a representação de processos ETL baseada em padrões, uma vez que permitia uma especificação mais abstrata e de leitura mais simples, conseguindo de uma forma mais clara separar a especificação conceptual da sua representação real. Por outro lado, seria também interessante explorar alguns dos trabalhos que

têm vindo a ser realizados no que diz respeito à representação de padrões ETL, utilizando outras ferramentas de modelação conceptual, como por exemplo BPMN (Oliveira e Belo, 2012) e Reo (Oliveira e Belo, 2013), e adaptá-los para a utilização da linguagem YAWL.

Aplicar o *Reverse Engineering*, seria também um grande desafio, podendo aproveitar-se algum do trabalho aqui realizado uma vez que os ficheiros XML já foram analisados e o objetivo passaria por inverter a lógica que foi criada para gerar, neste caso processos ETL. Seria interessante analisar o quão uma especificação física se pode abstrair de forma a gerar um modelo conceptual.

Bibliografia

Van Der Aalst, W. M. P. and Ter Hofstede, A. H. M. (2005) 'YAWL: Yet another workflow language', *Information Systems*, 30(4), pp. 245–275. doi: 10.1016/j.is.2004.02.002.

Adams, M., Ter Hofstede, A. H. M. and La Rosa, M. (2011) 'Open source software for workflow management: The case of YAWL', *IEEE Software*, 28(3), pp. 16–19. doi: 10.1109/MS.2011.58.

Belo, O. *et al.* (2014) 'Modeling E-Government processes using YAWL -half-way towards their effective real implementation', *ACM International Conference Proceeding Series*, 2014-Janua, pp. 288–291. doi: 10.1145/2691195.2691298.

Belo, O. *et al.* (2016) 'RAID-B2K , transforming BPMN conceptual schemas into Kettle execution primitives', *International Journal of Information and Decision Sciences*, 10 (Special Issue on 'Analytics for Decision Making', Issue Inderscience Enterprises Ltd.), pp. 3–18.

Belo, O., Cuzzocrea, A. and Oliveira, B. (2014) 'Modeling and Supporting ETL Processes via a Pattern-Oriented, Task-Reusable Framework', *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2014-Decem, pp. 960–966. doi: 10.1109/ICTAI.2014.145.

Börger, E. (2012) 'Approaches to modeling business processes: A critical analysis of BPMN, workflow patterns and YAWL', *Software and Systems Modeling*, 11(3), pp. 305–318. doi: 10.1007/s10270-011-0214-z.

Decker, G. *et al.* (2002) 'Lidar measures wake vortices', *Photonics Spectra*, 36(9), p. 35.

Foundation, T. Y. (2010) 'YAWL - User Manual', *The YAWL Foundation*, (November), pp. 1–28. Available at: http://www.yawlfoundation.org/content/yawl-yet-another-workflow-language-1.

Golfarelli and Rizzi (2009) 'Datawarehouse design'.

Gomes, N. M. (2013) *Modelação de Padrões ETL em YAWL*.

Guimarães, H. M. (2014) *Hugo Miguel Teixeira Lopes Guimarães Geração de Esqueletos para Sistemas de ETL a partir de Redes de Petri Coloridas*.

Ter Hofstede, A. H. M. *et al.* (2010) 'Modern business process automation: YAWL and its support environment', *Modern Business Process Automation: YAWL and its Support Environment*, pp. 1–676. doi: 10.1007/978-3-642-03121-2.

Marrella, A. *et al.* (2011) 'Making YAWL and SmartPM interoperate: Managing highly dynamic processes by exploiting automatic adaptation features', *CEUR Workshop Proceedings*, 820, pp. 1–6.

Oliveira, B. and Belo, O. (2004) 'ETL Patterns on YAWL Towards to the specification of platform-independent data warehousing populating processes', *16th International Conference on Enterprise Information Systems (ICEIS'2014), At Lisbon, Portugal*. doi: 10.5220/0004947302990307.

Oliveira, B. and Belo, O. (2012) 'BPMN Patterns for ETL Conceptual Modelling and Validation', pp. 1–10.

Oliveira, B. and Belo, O. (2013) 'Using Reo on ETL Conceptual Modelling A First Approach', pp. 1–6.
Oliveira, B., Belo, O. and Cuzzocrea, A. (2008) 'A Pattern-oriented Approach for Supporting ETL Conceptual Modelling and its YAWL-based Implementation'.

Ouyang, C. *et al.* (2008) 'Camera, Set, Action: A1tomating Film Production via Business Process Management', *QUT ePrints*, 0(0), p. Available at: http://eprints.qut.edu.a1/archive/00013361/.

Pentaho Corporation (2012) 'Pentaho Data Integration User Guide'.

Petri, C. A. (1962) *Kommunikation mit*.

Russell, N. and Ter Hofstede, A. H. M. (2009) 'Surmounting BPM challenges: The YAWL story', *Computer Science - Research and Development*, 23(2), pp. 67–79. doi: 10.1007/s00450-009-0059-7.