MPPI: um modelo de procedência para subsidiar processos de integração

Bruno Tomazela

MPPI: um modelo de procedência para subsidiar processos de integração

Bruno Tomazela

Orientadora*: Profª Drª Cristina Dutra de Aguiar Ciferri*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional.

USP – São Carlos
Janeiro/2010

Agradecimentos

Aos meus pais, Valter e Diva, ao meu irmão Giuliano e à sua esposa Luana, pelo suporte e incentivo incondicionais.

À minha orientadora Cristina D. A. Ciferri, pelo empenho e dedição despendidos sempre com muito boa vontade; além de sua inestimável amizade.

Ao professor Ricardo R. Ciferri pelo acompanhamento do trabalho, seus conselhos e sugestões.

Aos professores Caetano Traina Jr e Agma J. M. Traina pela experiência compartilhada.

A todos os responsáveis pelo Projeto Urano, o qual gerou a motivação deste trabalho.

Ao ICMC e ao GBdI, pela estrutura fornecida.

Ao CNPq, pelo apoio financeiro.

A todos aqueles que direta ou indiretamente contribuíram para a realização deste trabalho, em especial aos companheiros de república e de laboratório.

Resumo

A procedência dos dados consiste no conjunto de metadados que possibilita identificar as fontes e os processos de transformação aplicados aos dados, desde a criação até o estado atual desses dados. Existem diversas motivações para se incorporar a procedência ao processo de integração, tais como avaliar a qualidade dos dados das fontes heterogêneas, realizar processos de *auditoria* dos dados e de atribuição de *autoria* aos proprietários dos dados e reproduzir decisões de integração.

Nesta dissertação é proposto o MPPI, um **m**odelo de **p**rocedência para subsidiar **p**rocessos de **i**ntegração. O modelo enfoca sistemas nos quais as fontes de dados podem ser atualizadas somente pelos seus proprietários, impossibilitando que a integração retifique eventuais conflitos de dados diretamente nessas fontes. O principal requisito do MPPI é que ele ofereça suporte ao tratamento de todas as decisões de integração realizadas em processos anteriores, de forma que essas decisões possam ser reaplicadas automaticamente em processos de integração subsequentes.

O modelo MPPI possui quatro características. A primeira delas consiste no mapeamento da procedência dos dados em operações de *cópia*, *edição*, *inserção* e *remoção*, e no armazenamento dessas operações em um repositório de operações. A segunda característica é o tratamento de operações de sobreposição, por meio da proposta das políticas *blind*, *restrict*, *undo* e *redo*. A terceira característica consiste na identificação de anomalias decorrentes do fato de que fontes de dados autônomas podem alterar os seus dados entre processos de integração, e na proposta de quatro tipos de validação das operações frente a essas anomalias: validação completa, da origem, do destino, ou nenhuma. A quarta característica consiste na reaplicação de operações, por meio da proposta dos métodos VRS (do inglês *Validate and Reapply in Separate*) e VRT (do inglês *Validate and Reapply in Tandem*) e da reordenação segura do repositório, os quais garantem que todas as decisões de integração tomadas pelo usuário em processos de integração anteriores sejam resolvidas automaticamente e da mesma forma em processos de integração subsequentes.

A validação do modelo MPPI foi realizada por meio de testes de desempenho que investigaram o tratamento de operações de sobreposição, o método VRT e a reordenação segura, considerando como base as demais características do modelo. Os resultados obtidos mostraram a viabilidade de implementação das políticas propostas para tratamento de operações de sobreposição em sistemas de integração reais. Os resultados também mostraram que o método VRT proporcionou ganhos de desempenho significativos frente à coleta quando o objetivo é restabelecer resultados de processos de integração que já foram executados pelo menos uma vez. O ganho médio de desempenho do método VRT foi de pelo menos 93%. Ademais, os testes também mostraram que reordenar as operações antes da reaplicação pode melhorar ainda mais o desempenho do método VRT.

Abstract

Data provenance is the set of metadata that allows for the identification of sources and transformations applied to data, since its creation to its current state. There are several advantages of incorporating data provenance into data integration processes, such as to estimate data quality and data reliability, to perform data audit, to establish the copyright and ownership of data, and to reproduce data integration decisions.

In this master's thesis, we propose the MPPI, a novel data provenance model that supports data integration processes. The model focuses on systems in which only owners can update their data sources, i.e., the integration process cannot correct the sources according to integration decisions. The main goal of the MPPI model is to handle decisions taken by the user in previous integration processes, so they can be automatically reapplied in subsequent integration processes.

The MPPI model introduces the following properties. It is based on mapping provenance data into operations of copy, edit, insert and remove, which are stored in an operation repository. It also provides four techniques to handle overlapping operations: blind, restrict, undo and redo. Furthermore, it identifies anomalies generated by sources that are updated between two data integration processes and proposes four validation approaches to avoid these anomalies: full validation, source validation, target validation and no validation. Moreover, it introduces two methods that perform the reapplication of operations according to decisions taken by the user, called the VRS (Validate and Reapply in Separate) and the VRT (Validate and Reapply in Tandem) methods, in addition to extending the VRT method with the safe reordering optimization.

The MPPI model was validated through performance tests that investigated overlapping operations, the VRT method and the safe reordering optimization. The tests showed that the techniques proposed to handle overlapping operations are feasible to be applied to real integration systems. The results also demonstrated that the VRT method provided significant performance gains over data gathering when the goal is to reestablish previous integration results. The performance gains were of at least 93%. Furthermore, the performance results also showed that reordering the operations before the reapplication process can improve even more the performance of the VRT method.

Lista de Figuras

2.1 Aspectos da procedência dos dados. 5

3.1 Relação integrada em sua versão inicial (Adaptada de Archer et al. (2009)). 19
3.2 Relação integrada em sua versão final (Adaptada de Archer et al. (2009)). 19
3.3 Árvore de procedência para o atributo {104, Feridos} (Archer et al., 2009). 21
3.4 Exemplo de operações (Adaptado de Buneman et al. (2006b)). 22
3.5 Execução das operações (Adaptado de Buneman et al. (2006b)). 22
3.6 Técnicas de armazenamento (Adaptado de Buneman et al. (2006b)). . . . 23
3.7 Exemplo de um LDB, adaptado de (Benjelloun et al., 2008). 25
3.8 Exemplo de *x-relation* e *x-tuple*, adaptado de (Benjelloun et al., 2008). . . 25
3.9 Exemplo de um banco de dados implementado no Sistema Trio, adaptado de (Benjelloun et al., 2008). 27
3.10 Exemplo de tabela de vendas (Shiri e Taghizadeh-Azari, 2005). 30
3.11 Exemplo de um XML com itens à venda (Shiri e Taghizadeh-Azari, 2005). 30
3.12 Exemplo de uma DRT (Shiri e Taghizadeh-Azari, 2005). 30

4.1 Integração de fontes de dados no tempo t_1. 36
4.2 Reaplicação das decisões antes do processo de integração no tempo t_n. . . . 36
4.3 Exemplo de fontes com dados inconsistentes. 38
4.4 Fontes João e Luis sincronizadas. 39
4.5 Fontes João e Maria sincronizadas. 39
4.6 Fontes Luis e Maria sincronizadas. 39
4.7 Exemplo de um conjunto de operações. 42
4.8 Exemplo de uma operação de sobreposição (operação *13*). 44
4.9 Inserção de uma operação de sobreposição pela política *blind*. 45
4.10 Tratamento de uma operação de sobreposição pela política *restrict*. 45
4.11 Inserção de uma operação de sobreposição pela política *undo*. 47
4.12 Inserção de uma operação de sobreposição pela política *redo*. 50
4.13 Exemplo utilizado para validação: integração das fontes Luis e João. 53
4.14 Exemplo de validação no primeiro caso. 53
4.15 Exemplo de validação no segundo caso. 54
4.16 Exemplo de validação no terceiro caso. 54
4.17 Exemplo de validação no quarto caso. 55
4.18 Exemplo de validação no quinto caso. 55
4.19 Exemplo de validação no sexto caso. 56
4.20 Exemplo de validação no sétimo caso. 56
4.21 Fluxo das operações no método VRS. 58
4.22 Transitividade no processo de integração p_1. 60
4.23 Resultado do processo de integração p_1. 60
4.24 Validação das operações na reaplicação das operações de p_1. 60

4.25 Resultado da reaplicação das operações de p_1. 61
4.26 Operação que altera a chave no processo de integração p_2. 61
4.27 Resultado do processo de integração p_2. 61
4.28 Validação das operações na reaplicação das operações de p_2. 62
4.29 Resultado da reaplicação das operações de p_2. 62
4.30 Arquitetura do método VRT. 63
4.31 Exemplo de um repositório de operações não ordenado, fonte Luis carregada quatro vezes. 65
4.32 Exemplo de um repositório de operações reordenado, fonte Luis carregada três vezes. 65

5.1 Tratamento de operações de sobreposição - Escalabilidade do número de operações no repositório. 71
5.2 Tratamento de operações de sobreposição - Escalabilidade do número de operações transitivas. 72
5.3 Escalabilidade do número de operações realizadas. 74
5.4 Escalabilidade do número de fontes. 75
5.5 Simulação de caso real. 76
5.6 Número de operações por conjunto de fontes e porcentagem das operações. 77
5.7 Escalabilidade do número de operações. 78
5.8 Tempo para reordenar o repositório. 78
5.9 Escalabilidade do número de fontes. 79
5.10 Tempo para reordenar o repositório. 79

Lista de Tabelas

3.1 Comparação entre os trabalhos correlatos. 32

4.1 Tipos de validação e anomalias geradas. 57
4.2 Comparação entre os trabalhos correlatos e o modelo MPPI. 67

Lista de Algoritmos

1 Restrict (R, Op, TipoSobreposição) 46
2 DetectarSobreposicao (R, Op, OpSob, TipoSobreposicao) 47
3 Undo (R, Op) 48
4 EncontrarOpTransDiretas (R, Op, OperacoesTransitivas) 48
5 EncontrarOpTransIndiretas (R, OperacoesTransitivas) 49
6 Redo (R, Op) 51
7 VRS (R, Fontes) 59
8 VRT (R, Fontes) 64
9 ReordenacaoSegura (R) 68

Sumário

1 Introdução **1**
1.1 Organização do Trabalho . . . 3

2 Procedência dos dados **5**
2.1 Quais dados de procedência armazenar? . . . 6
2.1.1 Tipos de dados de procedência . . . 6
2.1.2 Granularidade dos Dados . . . 8
2.2 Como coletar os dados de procedência? . . . 9
2.2.1 Coleta manual e automática . . . 9
2.2.2 Abordagens *Lazy* x *Eager* . . . 10
2.3 Como armazenar os dados coletados? . . . 11
2.4 Como consultar os dados armazenados? . . . 13
2.4.1 Tipos de consulta . . . 13
2.4.2 Estratégias para consulta e visualização dos dados . . . 14
2.5 Considerações Finais . . . 15

3 Trabalhos Correlatos **17**
3.1 O Sistema CHIME . . . 17
3.1.1 Operações . . . 17
3.1.2 Árvore de Procedência . . . 20
3.2 O Sistema CPDB . . . 21
3.2.1 Operações . . . 21
3.2.2 Armazenamento da Procedência . . . 22
3.3 O Sistema Trio . . . 24
3.3.1 A Teoria ULDB . . . 24
3.3.2 Implementação do Sistema Trio . . . 26
3.4 O Sistema ELIT . . . 28
3.4.1 Características do Sistema ELIT . . . 28
3.4.2 Exemplo de Armazenamento dos Dados . . . 29
3.4.3 Consultando os Dados de Procedência . . . 29
3.5 Considerações Finais . . . 31

4 O Modelo MPPI **33**
4.1 Contextualização e Motivação . . . 35
4.1.1 Cenário de Integração de Dados . . . 35
4.1.2 Exemplo Corrente . . . 37
4.2 Operações . . . 40
4.2.1 Descrição das Operações . . . 40
4.2.2 Relacionamento entre Operações . . . 43

4.3 Operações de Sobreposição 43
4.3.1 Política *Blind* 45
4.3.2 Política *Restrict* 45
4.3.3 Política *Undo* 46
4.3.4 Política *Redo* 49
4.3.5 Considerações sobre as Políticas de Sobreposição 51
4.4 Validação 52
4.4.1 Os Sete Casos de Validação 52
4.4.2 Tipos de Validação 56
4.5 Reaplicação de Operações 57
4.5.1 Método VRS 58
4.5.2 Motivação para Proposta de um novo Método 59
4.5.3 Método VRT 62
4.5.4 Reordenação Segura 63
4.6 Considerações Finais 65

5 Testes de Desempenho **69**
5.1 Tratamento de Operações de Sobreposição 70
5.1.1 Escalabilidade do Número de Operações no Repositório 70
5.1.2 Escalabilidade do Número de Operações Transitivas 72
5.2 Reaplicação pelo Método VRT 73
5.2.1 Escalabilidade do Número de Operações Realizadas 74
5.2.2 Escalabilidade do Número de Fontes 75
5.2.3 Simulação de Caso Real 76
5.3 Reordenação do Repositório 77
5.3.1 Escalabilidade do Número de Operações 77
5.3.2 Escalabilidade do Número de Fontes 78
5.4 Considerações Finais 79

6 Conclusões **81**
6.1 Principais Contribuições 82
6.2 Trabalhos Futuros 83

Capítulo

1

Introdução

Esta dissertação tem como objetivo propor o modelo MPPI, um modelo de procedência dos dados para subsidiar processos de integração de dados de fontes heterogêneas.

A integração de dados de fontes heterogêneas tem sido foco de interesse tanto da área acadêmica (Halevy et al., 2006) quanto da indústria (Halevy et al., 2005). Dois problemas de integração diretamente relacionados a este trabalho são a integração de esquemas e a integração de instâncias. Na integração de esquemas, visa-se principalmente a geração de mapeamentos semânticos entre esquemas de fontes de dados heterogêneas, da forma mais automática possível (Kang e Naughton, 2003; Madhavan et al., 2005; McCann et al., 2005). Na integração de instâncias, visa-se a resolução de conflitos existentes entre os dados de fontes heterogêneas, considerando que as diferentes fontes podem conter dados complementares, repetidos e/ou inconsistentes, além da identificação automática de quais entidades referem-se a uma mesma entidade do mundo real e do agrupamento dessas entidades similares (Bhattacharya e Getoor, 2007; Chen et al., 2007; Dong et al., 2005; Kalashnikov e Mehrotra, 2006; On et al., 2006). Em relação à integração de dados, o enfoque desta dissertação é a investigação da procedência dos dados na resolução de conflitos em nível de instância.

A procedência dos dados consiste no conjunto de metadados que possibilita identificar as fontes de dados e os processos de transformação aplicados aos dados, desde a criação até o estado atual destes. Sinônimos de procedência na literatura são proveniência e linhagem dos dados. Em inglês, procedência é comumente chamada de *provenance*, *lineage* e *pedigree* (Benjelloun et al., 2008; Buneman et al., 2000; Glavic e Ditt, 2007; Munroe et al., 2006).

São várias as motivações do ponto de vista de sistemas de integração para se armazenar a procedência dos dados (Fileto et al., 2003; Freire et al., 2008; Nascimento e Hara, 2008;

Simmhan et al., 2005; Tan, 2004). Uma primeira motivação refere-se a verificar o *histórico* dos dados, por meio do armazenamento da informação sobre as transformações que foram efetuadas nos dados durante o processo de integração. Outra motivação consiste em assegurar a *qualidade* dos dados integrados. Por exemplo, no processo de integração pode-se inferir que um dado obtido de uma fonte confiável tem maior probabilidade de estar correto do que um dado de uma fonte desconhecida ou não-confiável.

A procedência também pode enfocar a realização de processos de *auditoria* dos dados e de atribuição de *autoria* aos proprietários dos dados. Conhecendo-se quais processos geraram os dados e quais fontes forneceram esses dados, é possível identificar falhas no processo de integração e concluir que os dados gerados por uma determinada transformação estão incorretos. Outra aplicação para a procedência é *retificar* fontes de dados que estão incorretas. Uma vez validado um dado integrado, todas as fontes que participaram da geração daquele dado podem ter seus próprios dados atualizados com base no resultado da integração.

Entretanto, nem sempre é possível alterar as fontes de dados, em geral por questões de propriedade. Por exemplo, suponha que Currículos Lattes sejam fontes de dados em um sistema de integração de dados de cunho acadêmico. Embora o processo de integração possa identificar que o mesmo artigo está declarado inconsistentemente em diferentes currículos, ele não tem permissão para alterar esses currículos. Nesses casos, as fontes fornecem sempre os mesmos dados inconsistentes, até que os seus proprietários as retifiquem. Portanto, outra motivação para o armazenamento da procedência dos dados é *reproduzir* decisões de integração. Para manter a consistência entre processos de integração distintos, é necessário que, para um mesmo conjunto de dados inconsistentes, os processos de integração gerem como resultado sempre os mesmos dados integrados.

Baseado nas diversas motivações relacionadas ao armazenamento da procedência dos dados, nesta dissertação é proposto o MPPI, um **m**odelo de **p**rocedência para subsidiar **p**rocessos de **i**ntegração. O modelo enfoca sistemas nos quais as fontes de dados podem ser atualizadas somente pelos seus proprietários, impossibilitando que a integração retifique eventuais conflitos de dados diretamente nessas fontes. O principal requisito do MPPI é que ele ofereça suporte ao tratamento de todas as decisões de integração realizadas em processos anteriores, de forma que essas decisões possam ser reaplicadas automaticamente em processos de integração subsequentes.

As principais características do modelo MPPI são:

- **Procedência baseada em operações**: a procedência dos dados considerada no modelo é vista como operações resultantes de processos de integração que o usuário realiza sobre os dados heterogêneos;

- **Tratamento de operações de sobreposição**: o modelo provê métodos para identificar situações nas quais uma operação gera uma inconsistência já resolvida

por alguma outra operação, ou seja, quando as operações são de sobreposição. O modelo também permite ao usuário escolher a forma como operações de sobreposição devem ser tratadas;

- **Validação das operações**: o modelo provê formas de validar as operações tal que o resultado final da reaplicação corresponda ao resultado gerado originalmente pelo usuário; e
- **Reaplicação de operações**: o modelo provê métodos que garantem que todas as decisões de integração tomadas pelo usuário em processos de integração anteriores sejam resolvidas automaticamente e da mesma forma em processos de integração subsequentes.

A validação do modelo MPPI foi realizada por meio de testes de desempenho utilizando 120 currículos Lattes de docentes do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, os quais foram usados como fontes a ser integradas. O processo de integração realizado pelo usuário, o qual gera as operações a serem tratadas pelo modelo proposto, foi feito usando a ferramenta Reconciliador de Dados Acadêmicos (Tomazela et al., 2008). Nos testes, foram investigadas as quatro características do modelo proposto quanto à sua eficiência em reaplicar as operações já realizadas pelo usuário antes do início de um processo de integração subsequente.

1.1 Organização do Trabalho

Além deste capítulo introdutório, esta dissertação é organizada em mais cinco capítulos.

No Capítulo 2 são descritos os principais conceitos relacionados à procedência dos dados. Nesse capítulo são detalhados os aspectos desafiadores da procedência e são abordados trabalhos sobre procedência que têm sido propostos na literatura.

No Capítulo 3 são detalhados quatro sistemas que provêm modelos de procedência dos dados. O primeiro deles, denominado CHIME (Archer et al., 2009), trata de procedência em ambientes de integração de dados. O sistema CPDB (Buneman et al., 2006a,b), por sua vez, é direcionado à procedência em bancos de dados curados manualmente. O terceiro sistema, chamado Trio (Agrawal et al., 2006; Benjelloun et al., 2008, 2006; Mutsuzaki et al., 2007; Widom, 2005), é direcionado a bancos de dados com suporte à procedência e incerteza dos dados. Por fim, o sistema ELIT (*Exploration and LIneage Tracing*) (Shiri e Taghizadeh-Azari, 2005) visa o rastreamento da procedência em sistemas de integração baseados em mediadores.

No Capítulo 4 é apresentada uma visão geral do modelo MPPI. Nesse capítulo também são detalhadas as soluções propostas para cada uma das características do modelo: a procedência baseada em operações, o tratamento de operações de sobreposição, a validação e a reaplicação das operações.

No Capítulo 5 é definido o ambiente de testes utilizado na investigação do modelo MPPI e são descritos os resultados dos testes de desempenho realizados.

A dissertação é finalizada no Capítulo 6, no qual são ressaltadas as suas principais contribuições, bem como sugestões para trabalhos futuros.

Capítulo

2

Procedência dos dados

Este capítulo descreve o estado da arte no tema procedência dos dados, discutindo os principais aspectos relacionados a essa área de pesquisa. O capítulo também aborda trabalhos sobre procedência que têm sido propostos na literatura, considerando cenários distintos e funcionalidades específicas.

A Figura 2.1 ilustra os aspectos relacionados à procedência dos dados e destaca a sequência na qual ocorrem. O primeiro aspecto a ser considerado trata-se da definição de quais dados devem ser armazenados bem como a granularidade desses dados. Após a definição dos dados a serem armazenados, é preciso estabelecer uma estratégia de coleta dos dados de procedência. Na sequência, é necessário armazenar os dados para torná-los acessíveis futuramente. Por fim, é necessário tornar os dados de procedência disponíveis para que os usuários possam consultar os dados armazenados.

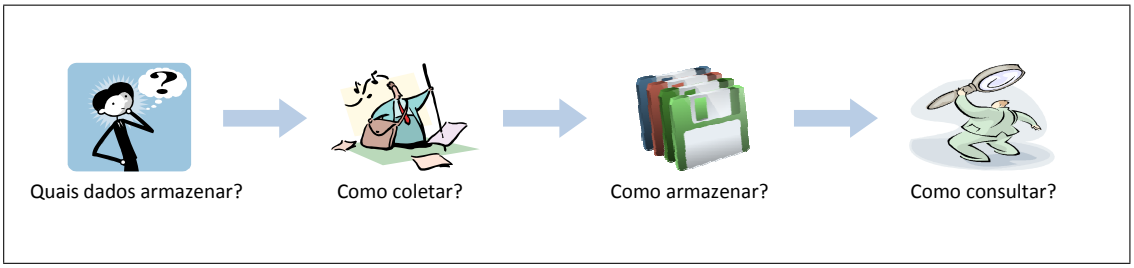


Figura 2.1: Aspectos da procedência dos dados.

Este capítulo está organizado da seguinte maneira. Nas Seções 2.1, 2.2, 2.3 e 2.4 são descritos, respectivamente, os quatro aspectos da procedência: quais dados armazenar, como coletar, como armazenar, e como consultar. O capítulo é finalizado na Seção 2.5 com as considerações finais.

2.1 Quais dados de procedência armazenar?

Primeiramente é necessário definir quais dados devem ter sua procedência monitorada. Isso deve levar em consideração o custo despendido para manutenção desses dados e o benefício alcançado posteriormente. Após definidos os dados que serão monitorados, o próximo passo a ser considerado em um projeto de armazenamento da procedência dos dados é decidir:

- Que tipos de dados de procedência devem ser armazenados (Seção 2.1.1); e
- Qual a granularidade dos dados sobre a procedência (Seção 2.1.2).

Além disso, os dados sobre procedência podem ser vistos como resultados de operações, as quais podem ter diferentes tipos de dados e granularidades. A definição das operações é específica de cada aplicação, tal como descrito no Capítulo 3.

2.1.1 Tipos de dados de procedência

Devido ao fato da área de pesquisa sobre procedência dos dados ser relativamente nova, ainda não há um consenso sobre a taxonomia a ser utilizada para classificar os tipos de dados que podem ser armazenados. As taxonomias existentes na literatura são: *source* e *transformation provenance*, *provenance meta-information* e *process meta-information*, *prospective* e *retrospective provenance*, *why-* e *where-provenance*, e *when-how-what components*.

Segundo Glavic e Ditt (2007), a procedência dos dados pode ser armazenada com dois enfoques principais: (1) identificar as fontes dos dados e (2) identificar os processos de transformação aplicados aos dados. O primeiro enfoque é definido como *source provenance* e o segundo como *transformation provenance*. *Source provenance* ainda é subdividido em três partes: *input source*, *original source* e *contributing source*. Trata-se de uma divisão que separa, respectivamente, (1) as fontes de dados que foram consultadas para a geração de um dado, (2) os itens de dados que realmente contribuíram para a geração de um dado, e (3) o conjunto mínimo de itens de dados necessários para a geração de um dado. *Source* e *transformation provenance* são complementares, sendo que esses dois enfoques podem co-existir em um mesmo sistema de procedência dos dados.

Para ilustrar os conceitos de *source* e *transformation provenance*, considere uma aplicação que armazena dados oriundos de fontes heterogêneas em um único banco de dados integrado. Desde que essas fontes podem conter dados redundantes, possíveis heterogeneidades decorrentes dessa redundância têm que ser resolvidas no processo de integração. Quando um usuário consulta o banco de dados integrado, os dados obtidos como resposta são aqueles que foram considerados como corretos no processo de integração. Sendo assim, o usuário também pode estar interessado na qualidade do

resultado obtido. Se um determinado dado contido na resposta foi coletado e inserido sem nenhum tipo de transformação, a informação sobre a sua fonte pode fornecer um meio de assegurar a sua qualidade. Esse tipo de procedência é denominado *source provenance*. Caso contrário, é possível validar a transformação realizada por meio de informações tais como quais transformações automáticas ou manuais foram efetuadas sobre os dados originais. O processo de integração é um exemplo de transformação que pode ser efetuada sobre os dados. Esse tipo de procedência é denominado *transformation provenance*.

Del Rio e Silva (2007) definem os conceitos *provenance meta-information* e *process meta-information*. *Provenance meta-information* diz respeito à informação sobre a fonte de dados que originou o dado, sendo diretamente relacionado ao conceito de *source provenance*. Já o conceito *process meta-information* refere-se aos processos que originaram um dado, e constitui um conceito semelhante ao de *transformation provenance*. Porém, *process meta-information* é mais abrangente no sentido que captura todo o ambiente envolvido na criação do dado, e não apenas as transformações aplicadas aos dados originais. Por exemplo, adicionalmente à transformação aplicada, podem ser armazenados também o usuário que realizou o processo, bem como a data e as ferramentas utilizadas.

Zhao et al. (2006) definem os conceitos *prospective* e *retrospective provenance*. Esses conceitos estão relacionados aos conceitos de *source* e *transformation provenance*, respectivamente. Contudo, as definições feitas por Zhao et al. (2006) são mais abrangentes. *Prospective provenance* é mais geral do que *source provenance* porque armazena informações suficientes para reproduzir um dado, ao invés de identificar somente a fonte que o originou. *Retrospective provenance* é mais geral do que *transformation provenance* porque guarda informações também sobre o ambiente de execução, e não somente sobre as transformações que foram realizadas.

Buneman et al. (2001) definem os conceitos *why-* e *where-provenance*. A motivação para tal classificação é a distinção entre a fonte de origem de um determinado dado (i.e., *where-provenance*) e porque o dado se encontra no banco de dados (i.e., *why-provenance*). Enquanto *why-provenance* armazena todas as fontes de dados que contribuíram para a geração de um dado, *where-provenance* armazena os dados das fontes que realmente contribuíram para a geração de um dado. *Why-* e *where-provenance* definem conceitos relacionados aos conceitos *original* e *contributing source*, respectivamente.

A taxonomia *when-how-what components* é proposta em Widom (2005). O componente *when* indica o momento de criação do dado. O componente *how* considera diversas maneiras de inserção de dados no banco, tais como por meio de consultas, por aplicações externas que acessam o banco, por atualizações efetuadas nos dados, por meio de importação de dados de outra fonte ou por carregamento em lote. O terceiro componente, *what*, indica quais dados das fontes foram utilizados para derivar o dado atual.

2.1.2 Granularidade dos Dados

Os dados sobre a procedência podem ser armazenados em diversos níveis de detalhe, ou seja, em diversas granularidades. Granularidade dos dados e nível de detalhe são conceitos inversamente proporcionais, ou seja, quanto maior a granularidade, menor o nível de detalhe e quanto menor a granularidade, maior o nível de detalhe. A título de exemplificação, considere um banco de dados relacional. Os dados sobre procedência podem ser armazenados em nível de tabela (maior granularidade), de tupla (média granularidade) ou então de atributo (menor granularidade) (Glavic e Ditt, 2007; Widom, 2005). O conceito de granularidade aplicado à procedência é o mesmo encontrado na teoria de *data warehouse* (Chaudhuri e Dayal, 1997; Kimball e Ross, 2002; Wu e Buchmann, 1997).

Três aspectos diferentes estão relacionados à granularidade dos dados. O primeiro aspecto refere-se ao espaço de armazenamento dos dados, o qual é inversamente proporcional à granularidade dos dados coletados. Sendo assim, quanto menor a granularidade, maior o espaço necessário para armazenar os dados. O segundo aspecto, custo da coleta dos dados, também é inversamente proporcional à granularidade (Simmhan et al., 2005). Já o terceiro aspecto refere-se à variedade de consultas que podem ser respondidas. Quanto maior o nível de detalhe, mais consultas podem ser respondidas sobre a procedência dos dados, em especial perguntas de caráter mais específico.

Por exemplo, em um SGBD (Sistema Gerenciador de Banco de Dados), se a granularidade estiver em nível de tupla, podem ser respondidas consultas em nível do SGBD como um todo, em nível de tabela ou então em seu nível mais detalhado, as tuplas. Porém, a quantidade de dados de procedência em nível de tupla requer maior espaço de armazenamento e maior tempo de coleta do que dados de procedência em nível de tabela ou do SGBD como um todo. Dessa forma, se a granularidade for armazenada em nível de tupla, maior será o tempo de coleta e o espaço de armazenamento necessário, entretanto, uma maior variedade de consultas poderá ser respondida.

Outro ponto a ser destacado é que os dados sobre a procedência podem ser considerados uma combinação dos conceitos de *log* e de versionamento. Por se tratar de um versionamento, dados antigos são mantidos para que se possa rastrear a história de um dado. Contudo, existem alguns pontos a serem considerados para se realizar o versionamento. Se a informação de procedência desejada for muito rica em detalhes, o crescimento do banco de dados de procedência pode se tornar um fator proibitivo para o armazenamento de dados históricos. Surge, então, a necessidade de se elaborar políticas de manutenção da procedência sobre dados obsoletos (Goble, 2002; Muniswamy-Reddy et al., 2006). Por exemplo, depois que um dado for removido, o sistema de procedência deve manter sua informação armazenada? Por quanto tempo? Quais dados sobre a

procedência devem ser mantidos? Todos? Essas são algumas questões em aberto que precisam de estudos mais aprofundados.

Portanto, a granularidade é um ponto que deve ser definido cuidadosamente em um projeto sobre procedência, a fim de evitar a necessidade de uma maior quantidade de espaço de armazenamento e manter o tempo de coleta no menor custo possível. Assim, é preciso escolher os dados de maior relevância para manter a procedência.

2.2 Como coletar os dados de procedência?

Depois de identificados os dados a serem armazenados, é necessário elaborar uma estratégia para a coleta dos mesmos. A coleta dos dados de interesse pode ser realizada manual ou automaticamente (Seção 2.2.1), usando as abordagens *lazy* ou *eager* (Seção 2.2.2).

2.2.1 Coleta manual e automática

A coleta dos dados de procedência pode ser feita com ou sem a intervenção do usuário. A estratégia a ser adotada varia caso a caso. Por exemplo, em bancos de dados curados manualmente, o usuário pode buscar dados de diversas fontes, utilizando mais do que uma ferramenta de busca. Desse modo, a coleta da procedência sobre esses dados pode ser feita diretamente pelo usuário. Porém, a tarefa de coleta da procedência é secundária frente ao interesse do usuário em realizar pesquisas sobre assuntos específicos. Sendo assim, atribuir ao usuário a tarefa de coleta dos dados acarreta a introdução de potenciais pontos de erros (Buneman et al., 2006a). Por isso, estratégias automáticas são preferidas às manuais.

Nos casos em que a aplicação oferece suporte à coleta de dados, a procedência pode ser implementada de maneira transparente para o usuário. Os trabalhos de Munroe et al. (2006), Del Rio e Silva (2007) e Muniswamy-Reddy et al. (2006) abordam o problema de coleta de dados de maneira automática. A principal diferença entre esses trabalhos refere-se ao fato da coleta ser realizada pelas aplicações utilizadas pelos usuários, por serviços externos às aplicações, ou pelo próprio sistema operacional.

Munroe et al. (2006) propõem o *Provenance Incorporating Methodology* (PrIMe), uma metodologia de engenharia de *software* para o desenvolvimento de aplicações que ofereçam suporte à coleta da procedência dos dados. Essa metodologia divide o processo de desenvolvimento em três fases. A primeira fase consiste em identificar os casos de uso da procedência e os itens de dados associados a cada caso. Na segunda fase identificam-se os atores envolvidos nos casos de uso e suas interações com a aplicação, além de quais itens de dados são considerados em cada interação entre os atores. Por ator entende-se toda entidade que interage com a aplicação, tais como um *web service* ou uma pessoa. O

terceiro e último passo é fazer as devidas alterações na aplicação de modo que ela ofereça suporte à coleta da procedência. A vantagem dessa metodologia é que a aplicação é capaz de coletar a procedência internamente e de maneira transparente para o usuário. Em contrapartida, se os passos propostos não forem aplicados durante o desenvolvimento da aplicação, a alteração da aplicação para oferecer suporte à procedência pode se tornar custosa a ponto de inviabilizar o processo.

Del Rio e Silva (2007) consideram a coleta da procedência por meio de serviços externos à aplicação. Mais especificamente, os autores fazem uso da *PLM Service Wrapper* (PSW). Esse serviço coleta informações de entrada e saída das aplicações que participam de um *workflow*, interceptando mensagens e eventos gerados pelas aplicações. Essa abordagem não introduz complexidade extra à aplicação. Porém, os dados que são internos à aplicação e não fazem parte dos dados de entrada/saída, não podem ter a procedência coletada.

Muniswamy-Reddy et al. (2006) propõem um sistema para a coleta da procedência em nível de sistema operacional e seu sistema de arquivos. A procedência é coletada em nível de arquivos, traduzindo sequências de comandos do sistema operacional em informações sobre procedência. A informação da procedência coletada consiste em uma descrição dos processos executados para gerar um arquivo. Assim como a coleta por meio de serviços externos à aplicação, implementar a procedência em nível de sistema operacional tem a vantagem de tornar a coleta transparente tanto para o usuário final quanto para o desenvolvedor de aplicações. Porém, uma desvantagem dessa abordagem é que a procedência é coletada com maior granularidade (em nível de arquivos do sistema de arquivos), pois a coleta de informações mais específicas depende do fato das aplicações oferecerem suporte à procedência.

2.2.2 Abordagens *Lazy* x *Eager*

A abordagem *lazy* consiste em coletar a procedência de um dado apenas quando essa for requisitada. Essa abordagem considera que existe uma consulta e uma fonte de dados, além de assumir que ambas (a consulta e a fonte) estão disponíveis no momento da coleta da procedência. Em geral, essa abordagem é utilizada em SGBDs da seguinte maneira: dado uma consulta Q e sua saída d, gera-se outra consulta cuja finalidade é obter os itens de dados que em conjunto com a consulta Q resultaram no dado d (Tan, 2004). Nessa abordagem, o consumo de memória é pouco, ou quase nenhum, uma vez que as informações são calculadas sempre que necessário. Porém, a informação sobre procedência recuperada é restrita somente ao contexto do SGBD, não sendo possível recuperar informações externas ao mesmo.

A abordagem *eager* consiste em armazenar a procedência dos dados conforme o dado passa de um sistema para outro ou sofre transformações. Essa abordagem também é referenciada na literatura por anotação. No momento em que a procedência é requisitada,

pode ser que a fonte de dados não esteja mais acessível e/ou a consulta não tenha sido documentada e ainda assim, na abordagem *eager*, é possível recuperar a procedência dos dados. Essa vantagem é decorrente do fato de que, nessa abordagem, a procedência é armazenada conforme os dados são gerados. Ademais, essa abordagem permite um nível de detalhe maior que a abordagem *lazy*. Contudo, as desvantagens da abordagem *eager* consistem no aumento da complexidade das aplicações em manter anotações sobre os dados e no espaço adicional necessário para o armazenamento de tais anotações.

Para ilustrar as abordagens *lazy* e *eager*, considere um sistema de procedência de dados que coleta dados em nível de sistema operacional. Esse sistema coleta a procedência conforme os processos são executados e dados (por exemplo, arquivos) são gerados. Dependendo do nível de detalhe coletado, o ambiente (*software* e *hardware*) no qual o processo foi executado pode alterar com o passar do tempo. Nesse caso, com o emprego da abordagem *lazy*, quando a informação sobre a procedência for requisitada, ela não mais refletirá o ambiente em que o processo realmente foi executado. Já com o emprego da abordagem *eager* a informação sobre a procedência será corretamente recuperada, uma vez que ela é coletada no momento em que o processo é executado e os dados são gerados (Muniswamy-Reddy et al., 2006).

Os trabalhos que abordam o problema de coleta de dados de maneira automática (i.e., Munroe et al. (2006), Del Rio e Silva (2007) e Muniswamy-Reddy et al. (2006)) enquadram-se como abordagens *eager*. Nesses trabalhos, os dados de procedência são coletados conforme são gerados.

2.3 Como armazenar os dados coletados?

Assim como toda a área de procedência, aspectos relacionados ao armazenamento dos dados ainda se encontram em desenvolvimento. Devido ao fato da área de procedência abranger praticamente todas as formas de manipulação e armazenamento de dados, os trabalhos existentes na literatura enfocam a procedência em nível de sistema operacional (Muniswamy-Reddy et al., 2006; Simmhan et al., 2005), em nível de aplicativo (Bose e Frew, 2004), em nível de serviço externo ao aplicativo (Del Rio e Silva, 2007) e em nível de SGBD (Buneman et al., 2006a,b; Glavic e Alonso, 2009; Widom, 2005). Ademais, Buneman et al. (2006) fazem algumas considerações adicionais sobre como armazenar os dados de procedência.

Considerando o cenário no qual os dados são coletados em nível de sistema operacional, mais especificamente no sistema de arquivos, Simmhan et al. (2005a) argumentam que a informação sobre a procedência pode ser armazenada dentro do arquivo ou em um banco de dados de procedência, por exemplo. Uma forma de armazenar a procedência dentro do arquivo é incluir metadados no cabeçalho do mesmo. Dessa forma, a procedência fica fortemente acoplada ao dado e o acompanha quando esse é copiado/movido para outros

locais. Porém, nessa abordagem, fica difícil procurar por dados que satisfaçam a algum critério de procedência, uma vez que a informação encontra-se espalhada por diversos arquivos.

O trabalho de Muniswamy-Reddy et al. (2006) propõe um sistema de procedência que coleta os dados diretamente no sistema de arquivos. O gerenciamento dos dados é feito em nível do *kernel* do sistema operacional. Nessa abordagem, a procedência é armazenada separadamente dos dados, facilitando o gerenciamento da procedência. Contudo, a manutenção dos dados de procedência quando arquivos são criados, alterados ou removidos é mais complexa do que se a procedência estivesse dentro do arquivo.

Em nível de aplicativo, a procedência dos dados pode ser gerenciada em formatos de arquivos específicos. Por exemplo, para o formato XML, pode-se adaptar seu XML *schema* para o armazenamento da procedência dos dados, ou pode-se criar um XML *schema* próprio para os dados da procedência. O trabalho de Bose et al. (2004) é um exemplo de trabalho que propõe um XML *schema* próprio para armazenar a procedência de dados advindos de pesquisas ambientais.

Del Rio e Silva (2007) fazem uso da *Proof Markup Language* (PML) para armazenar a informação sobre a procedência. A PML é uma linguagem com base na *Web Ontology Language* (OWL), projetada para facilitar interoperabilidade e confiabilidade na *web* semântica (da Silva et al., 2006). Essa linguagem deve ser utilizada em conjunto com o PSW, apresentado na Seção 2.2, pois o PSW formata os dados coletados na linguagem PML. Porém, testes realizados no trabalho de Del Rio e Silva (2007) mostraram que a interpretação de arquivos nessa linguagem é muito cara computacionalmente, prejudicando assim a escalabilidade do sistema.

SGBDs constituem outro cenário bastante comum para uso de técnicas de procedência dos dados. Com a crescente disponibilidade de informações na internet, bancos de dados curados manualmente têm sido mantidos a fim de reunir dados precisos sobre determinados assuntos, assim como os relacionados à bioinformática (Buneman et al., 2006a,b). Nesse cenário, a informação da procedência pode ser armazenada juntamente com o banco de dados em questão, sendo implementado, por exemplo, como um esquema do banco de dados. Dessa maneira, a informação da procedência pode ser facilmente vinculada aos dados. Contudo, essa abordagem aumenta a complexidade do banco de dados, uma vez que introduz mais relações e relacionamentos.

No SGBD Trio (Widom, 2005), cada banco de dados possui uma relação dedicada ao armazenamento da procedência, denominada *lineage*. Essa relação armazena uma referência para a tupla alvo (que está em alguma das relações do banco de dados), um identificador para a maneira como a tupla foi derivada, um identificador de como a tupla foi derivada, e uma referência aos dados, ou os próprios dados, que derivaram a tupla. A granularidade dos dados no sistema Trio pode ser feita em nível de tupla ou relação, conforme os requisitos da aplicação.

Uma abordagem alternativa à criação de uma relação dedicada é criar um banco de dados separado para os dados de procedência. Essa estratégia é utilizada em (Zhao et al., 2006) e (Buneman et al., 2006b). Assim, o projeto do banco de dados fica modularizado, sendo um banco para os dados e outro para a procedência. Ademais, a complexidade do banco de dados fica restrita à aplicação. Como nem sempre um SGBD é a melhor escolha para todos os casos, uma vantagem dessa abordagem está no fato de que o SGBD para procedência pode ser escolhido conforme a necessidade. Contudo, a manutenção da procedência dos dados se torna mais complexa, uma vez que não é possível fazer uma ligação direta entre o dado e sua procedência.

Independentemente do nível no qual a procedência é tratada (i.e., em nível de sistema operacional, em nível de aplicativo, em nível de serviço externo ao aplicativo ou em nível de SGBD), um aspecto importante a ser considerado refere-se a como armazenar os dados de forma a diminuir o espaço de armazenamento necessário (Anand et al., 2009; Buneman et al., 2006b, 2004; Chapman et al., 2008; Heinis e Alonso, 2008). Em (Buneman et al., 2006b) são propostas quatro técnicas para armazenamento, denominadas *naïve provenance*, *transactional provenance*, *hierarchical provenance* e *transactional-hierarchical provenance*. Essas técnicas são detalhadas na Seção 3.2, referente ao capítulo de trabalhos correlatos.

2.4 Como consultar os dados armazenados?

Com os dados sobre a procedência armazenados, pode-se investigar estratégias para consultar esses dados. Na Seção 2.4.1 são descritos os tipos de consulta existentes na literatura, enquanto que na Seção 2.4.2 são relacionados trabalhos existentes com relação a esses tipos de consulta.

2.4.1 Tipos de consulta

Existem dois tipos de consulta que podem ser realizados sobre os dados de procedência:

- tipo rastreamento: pesquisar os dados, e a partir do resultado obtido verificar a sua procedência; ou
- tipo filtro: recuperar os dados que satisfazem a um determinado critério de procedência (e.g., dados que foram criados/inseridos no banco no mês de outubro).

Cada um dos dois tipos de consulta enfoca diferentes aspectos na busca dos dados. Enquanto o primeiro visa buscar os dados e depois, se necessário, garantir a qualidade dos mesmos, o segundo oferece um método mais direto para buscar dados com base em sua qualidade/procedência. Assim sendo, os dois tipos de consultas consistem em abordagens complementares e podem co-existir em um mesmo sistema de procedência.

A viabilidade ou não de um sistema permitir os dois tipos de consulta depende principalmente de como os dados estão armazenados. Os sistemas de procedência que armazenam os dados em nível de arquivo do sistema operacional, mais especificamente dentro do arquivo como metadado, dificultam consultas do tipo filtro. Em consultas desse tipo para esse cenário, todos os arquivos do sistema de arquivos devem ser consultados, o que demanda grande quantidade de recursos e tempo, prejudicando a escalabilidade do sistema (Muniswamy-Reddy et al., 2006).

2.4.2 Estratégias para consulta e visualização dos dados

Alguns trabalhos existentes na literatura que propõem métodos para consulta dos dados de procedência envolvem o acesso aos dados por meio de uma interface ou API (*Application Programming Interface*), de modo que as aplicações possam acessar a procedência e tratar os resultados da maneira mais adequada (Muniswamy-Reddy et al., 2006). Outros desenvolvem ferramentas para consultas em conjunto com linguagens próprias para acesso aos dados, por exemplo (Widom, 2005). Com relação à visualização, essa pode ser feita por meio de grafos que descrevem a história de um dado. Del Rio e Silva (2007) propõem um trabalho que utiliza grafos como forma de consulta e visualização dos dados de procedência. Todos esses trabalhos oferecem suporte aos tipos de consulta rastreamento e filtro.

Muniswamy-Reddy et al. (2006) propõem um método de consulta da procedência por meio de uma ferramenta, e uma interface de acesso similar à de um banco de dados. A ferramenta proposta permite a navegação pelos arquivos do sistema operacional, tal qual um navegador de arquivos, e permite que o usuário recupere a procedência de cada arquivo. O usuário também pode procurar por arquivos utilizando uma consulta aos dados de procedência. Nesse sistema os dados de procedência estão centralizados em um sistema próprio de armazenamento, ao invés de estarem no próprio arquivo.

No sistema Trio, Widom (2005) propõe uma extensão da linguagem SQL (*Structured Query Language*), denominada TriQL, para consultar os dados sobre procedência. A abordagem de armazenamento da procedência proposta no sistema Trio permite que os tipos de consulta rastreamento e filtro sejam realizados facilmente. A consulta do tipo rastreamento, que para SGBDs trata-se da obtenção dos dados de procedência de uma tupla de uma relação R do banco, pode ser feita com uma operação de junção da relação R com a relação que armazena a procedência. Já a consulta do tipo filtro, que para SGBDs trata-se de selecionar dados que satisfazem a um critério de procedência, pode ser feita com uma operação de seleção na tabela de procedência, seguida de uma operação de junção com a relação que armazena os dados.

Del Rio e Silva (2007) propõem o sistema *Probe-It!* para visualização dos dados de procedência em *workflows*. O objetivo é unir as técnicas de procedência e visualização dos

dados e oferecer um sistema que facilite a análise e o entendimento de grandes quantidades de dados de procedência. Esse sistema separa as tarefas de coleta e armazenamento dos dados de procedência da tarefa de consulta e visualização da procedência. Para uso do sistema, assume-se que existe um banco de dados de procedência acessível como fonte, a qual deve ser passada como parâmetro pelo usuário do sistema. A principal característica desse trabalho é gerar visualizações por meio de grafos para as justificativas dos dados obtidos como resultados de consultas. Ademais, essas visualizações podem ser comparadas entre diferentes resultados de consultas.

Existem ainda trabalhos que atuam em frentes como otimização de consultas sobre dados de procedência (Kementsietsidis e Wang, 2009) e realização de outros tipos de consulta, além de rastreamento e filtro (Chapman e Jagadish, 2009). Como exemplo de outro tipo de consulta, pode-se desejar saber por que determinado dado não se encontra no resultado de um processo.

2.5 Considerações Finais

Neste capítulo foi descrito o estado da arte no tema procedência dos dados. Foi realizada uma classificação dos aspectos que devem ser levados em consideração para a proposta de um modelo de procedência dos dados. Esses aspectos foram classificados em: (i) quais dados de procedência armazenar, considerando a granularidade e os tipos de dados de procedência, além de destacar o fato de que os dados sobre procedência podem ser vistos como resultados de operações; (ii) como coletar os dados de procedência, considerando coleta manual e automática bem como as abordagens *lazy* e *eager*; (iii) como armazenar os dados coletados; e (iv) como consultar os dados armazenados, considerando os tipos de consulta e as estratégias para consulta e visualização.

Para cada um dos aspectos identificados neste capítulo, foram referenciados diversos trabalhos que têm sido propostos na literatura. Entretanto, esses trabalhos foram apresentados resumidamente, desde que o objetivo do capítulo foi apresentar uma visão geral sobre a área de pesquisa em procedência dos dados. Em contrapartida, o Capítulo 3 detalha quatro sistemas que provêm modelos de procedência dos dados, e que são mais relacionados ao modelo MPPI proposto nesta dissertação. Além disso, no Capítulo 4 é descrito o relacionamento das características do modelo MPPI com os aspectos classificados neste capítulo.

Capítulo

3

Trabalhos Correlatos

Neste capítulo são detalhados quatro sistemas que provêm modelos de procedência dos dados O sistema CHIME, descrito na Seção 3.1, trata de procedência em ambientes de integração de dados. O sistema CPDB (Seção 3.2), por sua vez, é direcionado à procedência em bancos de dados curados manualmente. Já o sistema Trio, o qual é direcionado a bancos de dados com suporte à procedência e incerteza dos dados, é resumido na Seção 3.3. O sistema ELIT (Seção 3.4) visa o rastreamento da procedência em sistemas de integração baseados em mediadores. O capítulo é finalizado na Seção 3.5, com as considerações finais.

3.1 O Sistema CHIME

O sistema CHIME (*Capturing Human Intension Metadata with Entities*) (Archer et al., 2009) enfoca a integração de dados em seu nível mais detalhado, ou seja, em nível de atributo. O usuário primeiro integra um conjunto de fontes autônomas e heterogêneas em uma única relação utilizando como base um conjunto de operações (Seção 3.1.1). Na sequência, o usuário utiliza essa relação integrada para realizar consultas a fim de extrair informações sobre os dados que coletou (Seção 3.1.2).

3.1.1 Operações

No sistema CHIME, a procedência dos dados é coletada por meio de operações que o usuário realiza durante o processo de integração. O conjunto de operações disponibilizado pelo sistema é composto por inserção, cópia, resolução de entidades, resolução de atributos e confirmação dos dados. O usuário pode inserir novos dados na relação integrada, copiar

dados de outras fontes e resolver eventuais conflitos de tuplas repetidas por meio da resolução de entidades, além de resolver conflitos semânticos entre atributos de uma mesma relação por meio da resolução de atributos. Ademais, o usuário pode confirmar dados das fontes como sendo corretos. As operações são coletadas automaticamente e armazenam quais dados foram considerados durante a integração, bem como qual dado foi considerado como correto. Portanto, a procedência do sistema CHIME pode ser classificada como *why-* e *where-provenance* e *eager*.

O exemplo a seguir ilustra essas operações. Um usuário deseja obter informações sobre os incidentes mais atuais ocorridos em uma determinada guerra e, após coletar alguns dados de fontes distintas, o usuário possui a relação de incidentes ilustrada na Figura 3.1. Nessa relação, um incidente é identificado por um *id* (*IncidID*), duas coordenadas para localização geográfica (*MapaL1* e *MapaL2*), o número de incidentes registrados (*Incidentes*), a munição utilizada nos incidentes (*Munição*), a quantidade de ocorrências médicas gerada pelo incidente (*Ocorrências Médicas*), outras duas coordenadas para localização geográfica (*Lat* e *Long*), e a data de ocorrência do incidente (*Data*). A relação da Figura 3.1 foi gerada da seguinte forma. A coluna *Data* foi inserida pelo usuário. As colunas *MapaL1*, *MapaL2*, *Incidentes* e *Munição* foram copiadas de uma primeira fonte de dados, ao passo que as colunas *Ocorrências Médicas*, *Lat* e *Long* foram copiadas de uma segunda fonte de dados. Essas colunas copiadas foram relacionadas por meio da coluna *IncidID*.

Analisando os dados coletados, o usuário descobre que as coordenadas geográficas *MapaL1* e *Lat* têm o mesmo significado, mas estão representadas em unidades diferentes. Para resolver essa redundância, o usuário agrupa essas duas colunas em uma única coluna por meio de uma operação de resolução de atributos, nomeando a nova coluna como *Latitude*. Analogamente, os atributos *MapaL2* e *Long* são agrupados em uma única coluna *Longitude*. Ademais, o usuário é informado que o número de incidentes e de ocorrências médicas tem por objetivo contar o número de feridos no incidente e, portanto, as agrupa em uma única coluna *Feridos*. Em uma operação de resolução de entidades, o usuário identifica que as tuplas com *IncidID* igual a 101 e 103 descrevem o mesmo incidente e, portanto, ele as agrupa em uma única tupla com *IncidID* igual a 101. Analogamente, o usuário agrupa as tuplas com *IncidId* igual 102 e 104 em uma única tupla com *IncidID* igual a 102. Por fim, o usuário confirma que o número de feridos da tupla com *IncidId* igual a 104 é correto. O resultado obtido pelo usuário após a integração é dado pela relação ilustrada na Figura 3.2.

Todas as operações realizadas pelo usuário são armazenadas pelo sistema CHIME em duas relações. A primeira relação, denominada R com respectiva instância r, possui o seguinte esquema: (i) *KeyVal*, um identificador único gerado pelo próprio sistema que atua como chave candidata de R; (ii)$\{Attr_1, ColVisible_1, Attr_2, ColVisible_2, ..., Attr_n, ColVisible_n\}$, onde cada *Attr* é um nome de atributo definido pelo usuário e cada

IncidID	MapaL1	MapaL2	Incidentes	Munição	Ocorrências Médicas	Lat	Long	Data
101	34.3998	70.4986	3	ied	3	3423.99	7029.92	03mar
102	34.3996	70.4985	7	art	6	3423.96	7029.90	05mar
103	34.3998	70.4985	4	ied	3	3423.99	7029.91	03mar
104	34.3996	70.4985	7	msl	6	3423.96	7029.90	04mar
105	34.3994	70.4988	12	ied	12	3423.94	7029.88	06mar

Figura 3.1: Relação integrada em sua versão inicial (Adaptada de Archer et al. (2009)).

IncidID	Latitude	Longitude	Munição	Feridos	Data
101	34.3998	70.4986	ied	4	03mar
104	34.3996	70.4985	art	6	04mar
105	34.3994	70.4988	ied	9	07mar

Figura 3.2: Relação integrada em sua versão final (Adaptada de Archer et al. (2009)).

$ColVisible_x$ é um atributo booleano que indica se $Attr_x$ está visível para o usuário e é passível de operações no sistema CHIME; e (iii)*RowVisible*, um atributo booleano que indica se a tupla está visível para o usuário e é passível de operações. Algumas colunas podem não estar visíveis para o usuário devido a uma operação de resolução de atributos, na qual a coluna foi integrada com outra(s) coluna(s). Já as tuplas podem não estar visíveis para o usuário devido a uma operação de resolução de entidades, na qual a tupla foi integrada com outra(s) tupla(s).

A segunda relação, denominada H com respectiva instância h, possui o seguinte esquema: (i) *SeqNum*, um inteiro monotônico tal como um *timestamp*; (ii) *KeyVal*, uma chave estrangeira referenciando R; (iii) *AttrName*, um atributo com domínio {$Attr_1$, ..., $Attr_n$} de R; (iv) *AttrVal*, valor dado ao atributo *AttrName* na tupla com chave *KeyVal*; (v) *Action*, um atributo com domínio {A, E, Θ, X}, que indica, respectivamente, as operações de resolução de atributos, resolução de entidades, atualização e confirmação do usuário; (vi) *Preferred*, um atributo com domínio {*Left*, *Right*, *Both*} que indica qual tupla ancestral, ou atributo, foi escolhida como correta para o item (*KeyVal*, *AttrName*) pelo usuário durante uma operação de resolução de entidades ou de atributos; (vii) *LeftParent*, um atributo com domínio {dom(*AttrName*) ∪ dom(*KeyVal*) ∪ {*NULL*}}, que indica uma tupla ancestral para resolução de entidades ou uma coluna ancestral para resolução de atributos, ou *NULL*; (viii) *RightParent*, um atributo com domíno {dom(*AttrName*) ∪ dom(*KeyVal*) ∪ {*NULL*}}, que indica o outro ancestral usado na resolução de inconsistências; e (ix) *Source*, uma cadeia de caracteres que indica a fonte que forneceu o dado ou se ele foi inserido pelo usuário.

3.1.2 Árvore de Procedência

O usuário pode utilizar a relação integrada da Figura 3.2 para realizar consultas a fim de extrair informações sobre os dados que coletou. Como os dados foram coletados e integrados de diferentes fontes, pode ser interessante averiguar a origem dos dados para estabelecer um nível de confiança no resultado. Por exemplo, o usuário pode fazer perguntas do tipo:

- Para incidentes duplicados, que foram integrados em um único incidente, quais eram os IDs dos incidentes originais?
- Quantas fontes de dados foram encontradas que corroboram a ocorrência do incidente com ID 112?
- Quais nomes o atributo Long teve ao longo do processo de integração?

O sistema CHIME responde a esses tipos de pergunta por meio de uma árvore construída a partir das operações coletadas. Uma árvore de procedência T_p(V,E) é definida para cada dado integrado como sendo um grafo acíclico dirigido com um conjunto de vértices V e um conjunto de arestas E. Um vértice v $\in$ V in T_p corresponde (i) ao estado corrente do dado de interesse, se ele for raiz de T_p; (ii) a um estado anterior (ancestral) do dado de interesse, se v não é raíz nem folha de T_p; ou (iii) a uma fonte de uma operação de atualização ou confirmação, se v é folha de T_p.

Cada árvore T_p é representada por duas relações, *Node* e *Edge*. A relação *Node* possui o seguinte esquema: (i) *Nodenum*, uma chave candidata para *Node*; (ii) *KeyValue*, uma chave estrangeira referenciando *KeyVal* em H; (iii) *AttrName*, um atributo com domínio {$Attr_1$, ..., $Attr_n$} de R, igual ao valor de *AttrName* da tupla correspondente em h; (iv) *AttrVal*, o valor de *AttrVal* na tupla correspondente em h; e (v) *Source*, o valor de *Source* na tupla correspondente em h. A relação *Edge*, por sua vez, possui o seguinte esquema: (i) *Descendant*, uma chave estrangeira referenciando *Node*; (ii) *Ancestor*, uma chave estrangeira referenciando *Node*; (iii) *Action*, atributo com domínio {A, E, Θ, X}, igual ao valor do atributo *Action* na tupla de h correspondente ao atributo *KeyValue* do nó referenciado pelo atributo *Descendant*; e (iv) *Preferred*, um atributo booleano com valor '*true*' se o valor do nó ancestral (com chave *Ancestor*) foi escolhido como correto para o valor do nó descendente (com chave *Descendant*), ou '*false*' caso contrário. Esse atributo é utilizado em operações de resolução de entidades ou atributos.

A Figura 3.3 ilustra a árvore construída para responder a questões de procedência do atributo *Feridos* da tupla com *IncidID* igual a 104, cujo valor atual é 6. Nessa figura, os rótulos das arestas representam as operações realizadas pelo usuário, sendo A para resolução de atributos, E para resolução de entidades, X para confirmação do usuário e Θ para atualização e/ou confirmação do usuário. Cada nó exibe o estado dos valores no decorrer da integração. Um nó que não foi escolhido como correto tem a respectiva aresta

representada por uma linha pontilhada. Utilizando essa árvore, o usuário pode verificar que a última operação realizada para gerar o valor 6 para o atributo *Feridos* do incidente em questão foi uma operação de resolução de atributos envolvendo os atributos *Incidentes* e *Ocorrências Médicas*, na qual considerou-se como correto o valor proveniente do atributo *Ocorrências Médicas*, o qual foi confirmado pelo usuário.

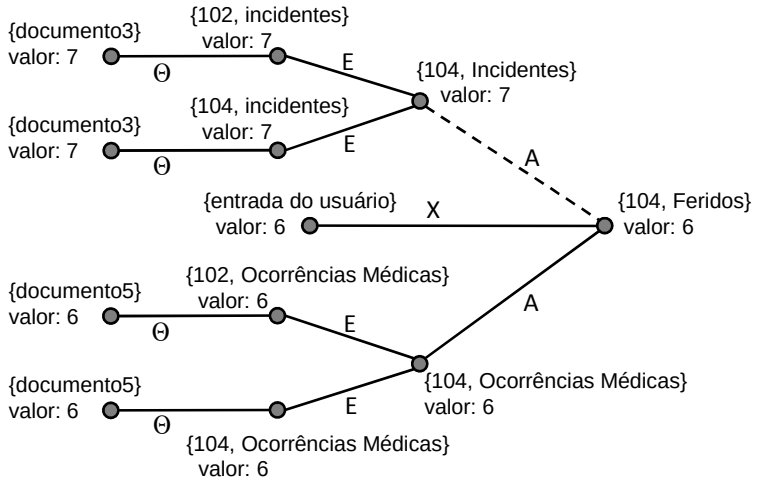


Figura 3.3: Árvore de procedência para o atributo {104, Feridos} (Archer et al., 2009).

3.2 O Sistema CPDB

O sistema CPDB (*Copy-Paste DataBase*) (Buneman et al., 2006a) é um sistema baseado em operações que tem como principal objetivo otimizar o armazenamento e o gerenciamento da procedência. Em suma, quatro técnicas são propostas: *naïve provenance*, *transactional provenance*, *hierarchical provenance* e *transactional-hierarchical provenance*. Essas otimizações consistem em diminuir o nível de detalhe da procedência o máximo possível, reduzindo o espaço de armazenamento necessário, bem como o espaço de busca para rastreamento da procedência. Na Seção 3.2.1 são descritas as operações introduzidas pelo modelo, ao passo que na Seção 3.2.2 são descritas as quatro técnicas propostas.

3.2.1 Operações

No sistema CPDB, a procedência é coletada com base nas operações de inserção, remoção e cópia de dados. As operações justificam o valor de um atributo, além de armazenarem a origem dos dados, o que caracteriza a procedência como *why-* e *where-provenance*. O modelo leva em consideração que as fontes de dados são organizadas em estruturas de árvores, tal como documentos XML, e os dados são identificáveis por atributos chave. Sempre que uma operação é realizada, o sistema coleta automaticamente seus dados. Portanto, trata-se de um sistema com abordagem *eager* para coleta da procedência.

A Figura 3.4 ilustra uma sequência de operações. Um exemplo de remoção é dado pela operação (1), na qual o usuário remove o nó alcançado pela aresta c5 da fonte T. Já um exemplo de cópia é dado pela operação (2), na qual o usuário copia o valor do nó alcançado pelas arestas a1/y da fonte S_1 para o nó alcançado pelas arestas c1/y da fonte T. Por fim, um exemplo de inserção é dado pela operação (5), na qual o usuário insere um nó com valor 10 na fonte T, o qual pode ser alcançado pelas arestas c2/y.

```
(1)   delete  c5          from T;
(2)   copy    S1/a1/y     into T/c1/y;
(3)   insert  { c2 : {} } into T;
(4)   copy    S1/a2       into T/c2;
(5)   insert  { y : 10 }  into T/c2;
(6)   insert  { c3 : {} } into T;
(7)   copy    S1/a3       into T/c3;
(8)   copy    S2/b3/y     into T/c3/y;
(9)   insert  { c4 : {} } into T;
(10)  copy    S2/b2       into T/c4;
(11)  insert  { y : 12 }  into T/c4;
```

Figura 3.4: Exemplo de operações (Adaptado de Buneman et al. (2006b)).

A Figura 3.5 ilustra graficamente a execução das operações da Figura 3.4. As árvores S_1 e S_2 são as fontes de dados, enquanto as árvores T e T' ilustram a fonte destino antes e depois das operações, respectivamente. Nós sem preenchimento não foram alterados, enquanto que nós pretos foram inseridos ou removidos. Os demais nós provêm de S_1 ou S_2, conforme ilustrado.

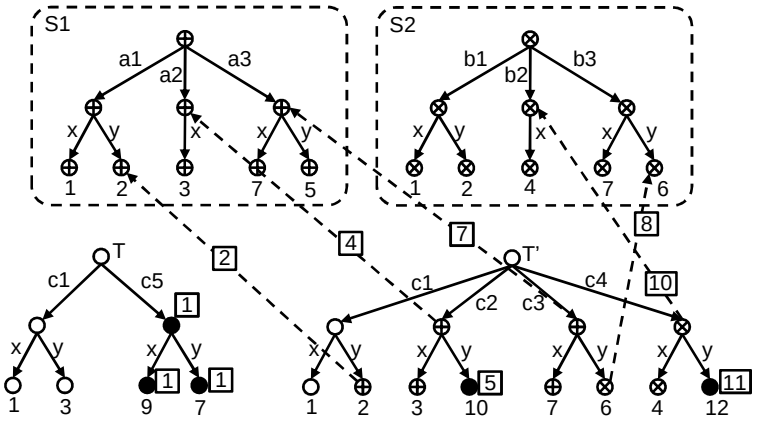


Figura 3.5: Execução das operações (Adaptado de Buneman et al. (2006b)).

3.2.2 Armazenamento da Procedência

No sistema CPDB, a procedência é armazenada separada dos dados, tendo sua própria relação, a qual é denominada Prov e possui o seguinte esquema: (i) *Tid*, que identifica a transação na qual a operação foi definida; (ii) *Op*, que identifica umas das três operações permitidas; (iii) *Loc*, que identifica a fonte destino e o seu dado afetado pela operação;

e (iv) *Src*, que identifica a origem e o dado utilizado em uma operação de cópia. Para operações de inserção e remoção, *Src* é nulo, sendo representado por $\perp$.

Para otimizar o armazenamento e o gerenciamento da procedência, o sistema CPDB introduz quatro técnicas. Para ilustrar cada uma das técnicas, as operações da Figura 3.4 são instanciadas em relações com esquema Prov na Figura 3.6. A primeira técnica (Figura 3.6a), denominada *naïve provenance*, não considera nenhum tipo de otimização, armazenando o máximo de dados possível, conseguindo dessa maneira o maior nível de detalhe.

(a) Prov

Tid	Op	Loc	Src
121	Delete	T/c_5	$\perp$
121	Delete	$T/c_5/x$	$\perp$
121	Delete	$T/c_5/y$	$\perp$
122	Copy	$T/c_1/y$	$S_1/a_1/y$
123	Insert	T/c_2	$\perp$
124	Copy	T/c_2	S_1/a_2
124	Copy	$T/c_2/x$	$S_1/a_2/x$
125	Insert	$T/c_2/y$	$\perp$
126	Insert	T/c_3	$\perp$
127	Copy	T/c_3	S_1/a_3
127	Copy	$T/c_3/x$	$S_1/a_3/x$
127	Copy	$T/c_3/y$	$S_1/a_3/y$
128	Copy	$T/c_3/y$	$S_1/b_3/y$
129	Insert	T/c_4	$\perp$
130	Copy	T/c_4	S_2/b_2
130	Copy	$T/c_4/x$	$S_2/b_2/x$
131	Insert	$T/c_4/y$	$\perp$

(b) Prov

Tid	Op	Loc	Src
121	Delete	T/c_5	$\perp$
121	Delete	$T/c_5/x$	$\perp$
121	Delete	$T/c_5/y$	$\perp$
121	Copy	$T/c_1/y$	$S_1/a_1/y$
121	Copy	T/c_2	S_1/a_2
121	Copy	$T/c_2/x$	$S_1/a_2/x$
121	Insert	$T/c_2/y$	$\perp$
121	Copy	T/c_3	S_1/a_3
121	Copy	$T/c_3/x$	$S_1/a_3/x$
121	Copy	$T/c_3/y$	$S_1/b_3/y$
121	Copy	T/c_4	S_2/b_2
121	Copy	$T/c_4/x$	$S_2/b_2/x$
121	Insert	$T/c_4/y$	$\perp$

(c) HProv

Tid	Op	Loc	Src
121	Delete	T/c_5	$\perp$
122	Copy	$T/c_1/y$	$S_1/a_1/y$
123	Insert	T/c_2	$\perp$
124	Copy	T/c_2	S_1/a_2
125	Insert	$T/c_2/y$	$\perp$
126	Insert	T/c_3	$\perp$
127	Copy	T/c_3	S_1/a_3
128	Copy	$T/c_3/y$	$S_1/b_3/y$
129	Insert	T/c_4	$\perp$
130	Copy	T/c_4	S_2/b_2
131	Insert	$T/c_4/y$	$\perp$

(d) HProv

Tid	Op	Loc	Src
121	Delete	T/c_5	$\perp$
121	Copy	$T/c_1/y$	$S_1/a_1/y$
121	Copy	T/c_2	S_1/a_2
121	Insert	$T/c_2/y$	$\perp$
121	Copy	T/c_3	S_1/a_3
121	Copy	$T/c_3/y$	$S_1/b_3/y$
121	Copy	T/c_4	S_2/b_2
121	Insert	$T/c_4/y$	$\perp$

Figura 3.6: Técnicas de armazenamento (Adaptado de Buneman et al. (2006b)).

A segunda técnica (Figura 3.6b), denominada *transactional provenance*, propõe agrupar os dados de procedência em transações. Por meio das transações é possível eliminar dados desnecessários. Por exemplo, na Figura 3.6a a operação com *Tid* igual a 126 é sobreposta pela operação com *Tid* igual a 127. Quando agrupadas em uma única transação na Figura 3.6b, a operação com *Tid* igual a 126 é descartada por não fazer parte do resultado final dessa transação.

A terceira técnica (Figura 3.6c), denominada *hierarchical provenance*, introduz o conceito de pai/filho para os dados e propõe que os dados da procedência de um dado filho podem ser recuperados pela procedência de seu pai. Dessa maneira, não é preciso duplicar para o dado filho a procedência que está no dado pai. Por exemplo, na Figura

3.6a, as operações com *Tid* igual a 121 estão relacionadas a dados pai/filho. Já na Figura 3.6c, apenas a operação referente ao dado pai é mantida e, portanto, existe apenas uma operação com *Tid* igual a 121. A vantagem dessa técnica é que não há perda de operações, ao contrário da segunda técnica.

A quarta técnica (Figura 3.6d), denominada *transactional-hierarchical provenance*, é uma combinação da segunda e da terceira técnicas. Essa quarta técnica armazena os dados de procedência da maneira mais concisa dentre as quatro técnicas propostas, embora ocorra perda de operações. Os exemplos descritos nas técnicas *transactional provenance* e *hierarchical provenance* podem ser verificados na Figura 3.6d.

3.3 O Sistema Trio

SGBDs convencionais armazenam os dados sem considerar a procedência e a precisão dos mesmos. Ou seja, se um dado está no banco, assume-se que ele está correto e não é possível saber qual a sua fonte ou o porquê de ele estar armazenado. Para tratar esses problemas, o objetivo do SGBD Trio é oferecer suporte ao tratamento de dados incertos, utilizando para isso a procedência e a precisão dos dados (Agrawal et al., 2006; Benjelloun et al., 2008, 2006; Mutsuzaki et al., 2007; Widom, 2005). O SGBD Trio usa a teoria ULDB (*Uncertainty-Lineage Database*), a qual consiste em uma extensão do modelo relacional e que está sendo desenvolvida pelos mesmos autores do sistema Trio. Essa teoria é descrita na Seção 3.3.1. Características da implementação do Sistema Trio são detalhadas na Seção 3.3.2.

3.3.1 A Teoria ULDB

Benjelloun et al. (2006, 2008) propõem uma teoria de banco de dados denominada ULDB. ULDB é a fusão da teoria de banco de dados com suporte à procedência com a teoria de banco de dados com suporte a dados incertos. O objetivo da teoria ULDB é fornecer um meio para tratar dados incertos, utilizando como base a procedência dos dados. Antes de descrever detalhes da teoria ULDB, é necessário explicar os conceitos utilizados por Benjelloun et al. (2006, 2008) para descrever a procedência dos dados e os dados incertos.

A procedência dos dados utilizada por Benjelloun et al. (2006, 2008) é classificada como *where provenance*, pois considera os dados que contribuíram na derivação de outro dado. Benjelloun et al. (2006, 2008) definem LBD (*Lineage DataBase*) como sendo um banco de dados com suporte à procedência. Um LDB é uma tripla (R, S, f), onde R é o conjunto de relações, S é o conjunto de identificadores únicos em R e f é uma função de procedência de S em 2^S. A Figura 3.7 ilustra um LDB. A relação *acusação* é derivada das relações *testemunha* e *motorista*. À frente de cada tupla da relação *acusação*

está ilustrada a sua função de procedência f, ou seja, quais tuplas foram utilizadas no processo de derivação. Por exemplo, a tupla com identificador *41* foi derivada a partir das tuplas com identificadores *21* e *31*. No exemplo corrente, $R = \{testemunha, motorista, acusação\}$, $S = \{21, 22, 23, 31, 32, 33, 34, 41, 42, 43, 44\}$, $f(41) = \{21, 31\}$, $f(42) = \{22, 32\}$, $f(43) = \{21, 33\}$ e $f(44) = \{23, 34\}$.

Testemunha

ID	Pessoa	Carro
21	Amy	Mazda
22	Amy	Toyota
23	Betty	Honda

Motorista

ID	Pessoa	Carro
31	Jimmy	Mazda
32	Jimmy	Toyota
33	Billy	Mazda
34	Billy	Honda

Acusação

ID	Testemunha	Motorista	
41	Amy	Jimmy	$f(41) = \{21, 31\}$
42	Amy	Jimmy	$f(42) = \{22, 32\}$
43	Amy	Billy	$f(43) = \{21, 33\}$
44	Betty	Billy	$f(44) = \{23, 34\}$

Figura 3.7: Exemplo de um LDB, adaptado de (Benjelloun et al., 2008).

Para o tratamento de dados incertos, são definidos os conceitos de *x-tuple* e *x-relation*. Um *x-tuple* é um multiconjunto[1] de tuplas, chamadas de alternativas. Tuplas alternativas são possíveis valores para uma mesma tupla, e são mutuamente exclusivas. Ademais, um *x-tuple* pode ser marcado como *maybe*, ou seja, a pertinência da tupla no banco é incerta, tanto quanto seus dados. Consequentemente, um *x-relation* é um multiconjunto de *x-tuples*. A Figura 3.8 ilustra os conceitos *x-relation* e *x-tuple*. As alternativas de uma mesma tupla são separadas por "||" e uma interrogação à frente de um *x-tuple* identifica-o como *maybe*. Por exemplo, *testemunha* é um *x-relation*, que possui um *x-tuple* com as alternativas (Amy, Mazda) e (Amy, Toyota), e que está marcado como *maybe*.

Testemunha

ID	(Pessoa, Carro)	
21	(Amy, Mazda) || (Amy, Toyota)	?
23	(Betty, Honda)	

Figura 3.8: Exemplo de *x-relation* e *x-tuple*, adaptado de (Benjelloun et al., 2008).

Uma consequência direta de se utilizar o conceito de *x-tuple* é que a relação, agora denominada *x-relation*, pode possuir mais de uma instância em um dado momento, devido às tuplas alternativas. Uma instância de um *x-relation* é construída da seguinte maneira: para *x-tuples* que não são marcados como *maybe*, deve-se escolher exatamente uma de

[1]Diferentemente de um conjunto, um *multiconjunto* pode conter elementos repetidos.

suas alternativas, e para *x-tuples* que são marcados como *maybe*, pode-se escolher uma ou nenhuma de suas alternativas.

Com base nos conceitos LDB, *x-relation* e *x-tuple*, Benjelloun et al. (2006, 2008) definem a teoria ULDB, a qual consiste na adaptação da definição de LDB para o uso do conceito *x-relation* em detrimento do conceito *relação*. Dessa forma, um ULDB é uma tripla (R', S, f), onde R' é um conjunto de *x-relation*, S é o conjunto de identificadores únicos em R' e f é uma função de procedência de S em 2^S. Em um LDB, um identificador corresponde a um valor i que identifica a tupla no banco de dados. Já em um ULDB, um identificador é formado por um par (i, j), onde i identifica uma tupla e j identifica uma de suas alternativas. Nas explicações a seguir, $s_{(i,j)}$ denota um identificador de uma alternativa de um *x-tuple* e t_i um identificador de um *x-tuple*.

Tal como introduzido anteriormente, o conceito *x-tuple* permite que um *x-relation* possua mais de uma instância em um dado momento. A escolha de uma alternativa para cada tupla de um ULDB gera uma instância de um LDB. Porém, a função de procedência f de cada tupla em um LDB deve ser consistente, ou seja, não podem existir tuplas cuja procedência não pertence à instância. Considere $D = (R', S, f)$ um ULDB e $s_{(i,j)}$ o conjunto de identificadores únicos em R'. Um possível LDB D' de D é obtido da maneira a seguir. Seja $S_k \subseteq S$ tal que:

1. Se $s_{(i,j)} \in S_k$, então para todo $j' \neq j$, $s_{(i,j')} \notin S_k$.
2. $\forall\ s_{(i,j)} \in S_k$, $f(s_{(i,j)}) \subseteq S_k$.
3. Para qualquer *x-tuple* t_i tal que não existe um $s_{(i,j)} \in S_k$, as seguintes condições devem ser satisfeitas: (i) t_i é um *x-tuple* marcado como *maybe*, e (ii) $\forall\ s_{(i,j)} \in t_i$, então $f(s_{(i,j)}) = \emptyset$ ou $f(s_{(i,j)}) \not\subseteq S_k$.

A primeira condição impõe que tuplas alternativas sejam mutuamente exclusivas em uma instância de um LDB, ou seja, no máximo uma alternativa de uma tupla pode estar em uma possível instância. Já a segunda condição obriga que, se uma alternativa pertencer a uma possível instância, então as tuplas que formam a sua procedência também devem fazer parte da instância. Por último, a terceira condição diz que um *x-tuple* não precisa contribuir com uma tupla para a instância caso ele esteja marcado como *maybe* e nenhuma de suas alternativas possua em sua procedência tuplas que pertencem à instância.

3.3.2 Implementação do Sistema Trio

O Sistema Trio é a implementação de um SGBD com suporte à procedência dos dados e dados incertos, que usa como base a teoria ULDB. Sua primeira versão, denominada *TrioOne*, foi implementada como uma camada intermediária entre um SGBD convencional e as aplicações de usuário.

Os conceitos *x-relation* e *x-tuple* foram implementados por meio de relações e tuplas convencionais. Cada alternativa de um *x-tuple* é tratada como uma tupla de uma relação em um SGBD convencional. Toda relação no *TrioOne* é adaptada para simular um *x-relation* por meio do acréscimo de três atributos: *aid*, *xid*, *num*. O atributo *aid* identifica uma alternativa de um *x-tuple*, *xid* especifica o *x-tuple* ao qual a alternativa pertence e *num* é um número não negativo que indica se a tupla está ou não marcada como *maybe*. Assim, uma relação $T(A_1, A_2,..., A_3)$ é transformada em $T(aid, xid, num, A_1, A_2,..., A_3)$.

A procedência é armazenada separadamente dos dados, tendo sua própria tabela. Para cada relação T é criada uma relação $lin\_T(aid, src\_aid, src\_table)$. Uma tupla (a_1, a_2, T_2) pertencente à relação lin_T significa que a alternativa a_1 possui a alternativa a_2 da relação T_2 em sua procedência. Cada tupla inserida no banco de dados gera uma entrada na tabela de procedência. Dessa forma, a coleta de dados no Sistema Trio é classificada como *eager*.

A Figura 3.9 ilustra um banco de dados implementado no Sistema Trio para as mesmas relações da Figura 3.7. Assim como na Figura 3.7, a relação *acusação* é derivada das relações *testemunha* e *motorista*. Na Figura 3.9, entretanto, cada uma dessas relações possui as colunas adicionais *aid*, *xid* e *num*. Na relação *testemunha* existem duas alternativas para a tupla com $xid = 21$. Já na relação *lin_acusação* é armazenada a procedência de cada tupla, ou seja, o resultado da função f de todas as tuplas da relação *acusação*. Por exemplo, a primeira tupla de *lin_acusação* refere-se ao fato da alternativa *411* ter como parte de sua procedência a alternativa *211*, que pertence à relação *testemunha*. As tuplas que possuem $num > 1$ correspondem a *x-tuples* marcados como *maybe*.

Testemunha

aid	xid	Pessoa	Carro	num
211	21	Amy	Mazda	3
212	21	Amy	Toyota	3
221	22	Betty	Honda	1

Motorista

aid	xid	Pessoa	Carro	num
411	41	Jimmy	Mazda	1
421	42	Jimmy	Toyota	1
431	43	Billy	Mazda	1
441	44	Billy	Honda	1

Acusação

aid	xid	Testemunha	Motorista	num
411	41	Amy	Jimmy	3
421	42	Amy	Jimmy	3
431	43	Amy	Billy	3
441	44	Betty	Billy	1

Lin_Acusação

aid	src_aid	src_table
411	211	Testemunha
411	311	Motorista
421	212	Testemunha
421	321	Motorista
431	211	Testemunha
431	331	Motorista
441	221	Testemunha
441	341	Motorista

Figura 3.9: Exemplo de um banco de dados implementado no Sistema Trio, adaptado de (Benjelloun et al., 2008).

3.4 O Sistema ELIT

Shiri e Taghizadeh-Azari (2005) endereçam o problema de se obter a procedência dos dados em sistemas de integração baseados em mediadores. Um mediador é uma camada intermediária entre os aplicativos do usuário e as fontes de dados. Quando um aplicativo necessita obter um dado, ele acessa a fonte utilizando as funcionalidades do mediador. Para isso, o mediador deve possuir todas as informações necessárias para acessar as fontes de dados, tais como o endereço do servidor e o esquema do banco de dados. O mediador pode consultar diversas fontes de dados para responder a uma consulta de um determinado aplicativo. Contudo, o aplicativo apenas faz a requisição para o mediador e não precisa ter conhecimento das fontes consultadas. Nesse tipo de integração, não há o armazenamento de dados no mediador. Sempre que uma consulta é realizada, as fontes de dados devem ser acessadas (Wiederhold, 1992).

Em um sistema de integração de dados baseado em mediadores, diversos fatores podem dificultar e até impossibilitar o rastreamento da procedência após uma consulta. Pode ser difícil obter a procedência se, após uma consulta, o esquema do banco de dados de uma das fontes for modificado. Pode ser impossível obter a procedência se, após uma consulta, a fonte não estiver mais disponível ou se os dados utilizados na resposta tiverem sido removidos ou atualizados.

Para possibilitar o rastreamento da procedência dos dados após uma consulta, Shiri e Taghizadeh-Azari (2005) propõem o Sistema ELIT (*Exploration and LIneage Tracing*). Esse sistema oferece suporte à procedência por meio da coleta de dados durante o processamento de consultas no mediador. O sistema ELIT pode trabalhar com diversas fontes de dados ao mesmo tempo, as quais podem ser heterogêneas. Outro aspecto é que o ELIT utiliza um único modelo de dados lógico para realizar consultas nas diversas fontes. Na Seção 3.4.1 são descritas as características do sistema, enquanto que na Seção 3.4.2 é ilustrado um exemplo de armazenamento dos dados nesse sistema. A consulta aos dados de procedência é descrita na Seção 3.4.3

3.4.1 Características do Sistema ELIT

Quatro características importantes do Sistema ELIT referem-se à granularidade dos dados, ao que coletar, a como coletar e a como armazenar. A granularidade adotada é em nível de instância. Contudo, o sistema também precisa ter a definição do esquema de cada fonte para que possa relacionar um dado à sua origem. Por exemplo, o dado *nome_item* foi extraído do campo *nome* da tabela *item* na fonte *filial1*, sendo que *nome*, *item* e *filial1* fazem parte do esquema da fonte.

Com relação ao que coletar, os autores classificam o Sistema ELIT como *source provenance* ou, mais especificamente, como *why-* e *where-provenance*. A fim de possibilitar

o rastreamento da procedência após a consulta, o sistema coleta todos os dados originais que foram usados para responder a consulta, bem como todos os resultados intermediários e o resultado final da consulta. Um dado original refere-se a um dado armazenado em uma fonte de dados.

Para a coleta dos dados de procedência, primeiramente é necessário ter *a priori* a definição dos esquemas das fontes. Essa primeira parte deve ser obtida manualmente e passada como parâmetro de entrada do sistema. Em seguida, quando uma consulta é realizada, os dados utilizados na resposta são coletados nas fontes pelo mediador. O sistema coleta automaticamente esses dados, sendo que o processo de coleta é transparente tanto para o usuário quanto para o aplicativo usado para realização da consulta. Esse tipo de coleta de dados é classificado como *eager*.

Para o armazenamento dos dados são considerados os dois tipos de dados coletados pelo sistema, sendo (i) os metadados relativos ao esquema de cada fonte e (ii) os dados utilizados no processamento da consulta. Os dados e os metadados são armazenados em um banco de dados relacional, sendo que os metadados são armazenados em seu próprio esquema, denominado *Integrated Metadata Schema* (IMDS), e os dados em uma tabela denominada *Data Reference Table* (DRT). Um detalhe importante é que, como as fontes podem conter dados repetidos, os dados da DRT são armazenados utilizando o conceito de multiconjunto, para que a procedência possa ser corretamente rastreada.

3.4.2 Exemplo de Armazenamento dos Dados

Para ilustrar o armazenamento dos dados na DRT, considere a aplicação de vendas extraída de (Shiri e Taghizadeh-Azari, 2005). A Figura 3.10 ilustra uma relação que armazena dados relativos às vendas, enquanto que a Figura 3.11 é um documento XML que armazena os itens a venda. Já a Figura 3.12 é um exemplo de DRT que armazena dados tanto da tabela de vendas quanto do arquivo de itens, construída para uma consulta que requer dados integrados dessas duas fontes de dados heterogêneas. Para cada item e venda utilizados no processamento da consulta é inserida uma tupla na DRT. Como pode ser visto na Figura 3.12, a DRT armazena o nome da tabela (SGBD), o número do registro, o nome do atributo e o valor do atributo. Embora o objetivo da DRT seja armazenar dados, ela também armazena alguns metadados (e.g., nome da tabela) para facilitar o rastreamento da procedência.

3.4.3 Consultando os Dados de Procedência

A especificação da consulta aos dados das fontes é feita utilizando o esquema do IMDS por meio de uma linguagem similar ao SQL. São aceitas consultas que envolvem as operações de seleção, projeção e junção. O sistema mapeia a consulta para as diversas fontes de acordo com funções de mapeamento.

Vendas

id_filial	id_item	qtd_total	preço
2	1	800	5
2	2	2000	2
2	4	800	35
3	3	1500	45
3	4	600	60
4	3	2100	50
4	4	1200	70
4	5	200	30
1	1	1000	4
1	2	3000	1

Figura 3.10: Exemplo de tabela de vendas (Shiri e Taghizadeh-Azari, 2005).

```
...
<xs:complextType name = “ITEM”>
<xs:attribute name = “id_item”      type = “ID”      use = “required” />
<xs:attribute name = “nome_item”    type = “string”  use = “required” />
<xs:attribute name = “categoria”    type = “string”  use = “required” />
...
...
<ITEM id_item = “3” nome_item = “cola”          categoria = “papelaria” />
<ITEM id_item = “3” nome_item = “caneta”        categoria = “papelaria” />
<ITEM id_item = “3” nome_item = “camiseta”      categoria = “vestimenta” />
<ITEM id_item = “3” nome_item = “calça”         categoria = “vestimenta” />
<ITEM id_item = “3” nome_item = “pote”          categoria = “cozinha” />
...
```

Figura 3.11: Exemplo de um XML com itens à venda (Shiri e Taghizadeh-Azari, 2005).

nome_tabela	num_registro	nome_atributo	valor
...	...	...	...
Vendas	1	id_filial	2
Vendas	1	id_item	1
Vendas	1	qtd_total	800
Vendas	1	preço	5
Vendas	2	id_filial	2
Vendas	2	id_item	21
Vendas	2	qtd_total	2000
Vendas	2	preço	2
...	...	...	...
ITEM	1	id_item	1
ITEM	1	nome_item	cola
ITEM	1	categoria	papelaria
ITEM	2	id_item	21
ITEM	2	nome_item	caneta
ITEM	2	categoria	papelaria
...	...	...	...

Figura 3.12: Exemplo de uma DRT (Shiri e Taghizadeh-Azari, 2005).

Quando uma consulta é submetida ao sistema, primeiramente ele recupera todos os dados originais D das tabelas que fazem parte da cláusula *from* do *select* e os armazena

na DRT. Em seguida, aplica todas as condições da cláusula *where* aos dados da DRT. Os dados que permaneceram na DRT são retornados como resultado da consulta. Como os dados da DRT estão armazenados no mediador, é possível obter a procedência dos dados retornados como resposta da consulta, independentemente do que aconteça às fontes (e.g., alteração ou remoção de dados, alteração dos esquemas ou indisponibilidade das fontes).

Para o sistema identificar os dados originais D que afetam a resposta, são aplicadas funções de transformação inversas às aplicadas na geração do resultado R. Para a construção de R, podem ter sido gerados resultados intermediários (T_1,...,T_n), sendo um para cada condição da cláusula *where*. Nesse caso, as funções inversas são aplicadas começando por R para gerar o último resultado intermediário (i.e., T_n). Em seguida, é aplicada a função inversa que gerou T_n para gerar T_{n-1}. Isso é realizado repetidamente até chegar aos dados originais (i.e., D).

A resposta do rastreamento da procedência consiste em uma tabela com as tuplas que contribuíram para o resultado da consulta. O formato da saída é uma tripla (D, COL, V), onde D identifica a fonte de dados (e.g., *ITEM*), COL é o item de dado em D (e.g., *ITEM.nome_item*), e V é o valor de COL (e.g., *caneta*).

3.5 Considerações Finais

Neste capítulo foram descritos quatro sistemas que rastreiam a procedência dos dados, denominados CHIME, CPDB, ELIT e Trio. A Tabela 3.1 compara esses trabalhos, usando como base os quatro aspectos da procedência identificados no Capítulo 2: (i) quais dados armazenar, ou, o que coletar; (ii) como coletar; (iii) como armazenar; e (iv) como consultar. Essa tabela também destaca o objetivo de cada trabalho.

Apesar das vantagens introduzidas por esses sistemas, eles não têm como objetivo oferecer suporte ao tratamento de todas as decisões tomadas pelo usuário em processos de integração anteriores, de forma que essas decisões possam ser reaplicadas automaticamente em processos de integração subsequentes. A reaplicação das decisões do usuário introduz três desafios adicionais:

- A necessidade de que todas as decisões tomadas pelo usuário em processos de integração anteriores sejam resolvidas automaticamente e da mesma forma em processos de integração subsequentes;
- O tratamento das decisões do usuário que porventura invalidem decisões previamente tomadas por este; e
- A investigação da possibilidade das fontes heterogêneas terem alterado os seus dados entre dois processos de integração subsequentes, de forma a invalidar decisões previamente tomadas pelo usuário.

Tabela 3.1: Comparação entre os trabalhos correlatos.

Aspectos		CHIME	CPDB	Trio	ELIT
Objetivo		*Rastrear resultados de integração*	*Otimizar armazenamento*	*Procedência para SGBDs*	*Rastrear resultados de integração*
O que coletar?	tipo	*why-* e *where-provenance*	*why-* e *where-provenance*	*where-provenance*	*where-provenance*
	granularidade	atributo	atributo / objeto	tupla	tupla
	operações	cópia, inserção, resolução de atributos, resolução de entidades, confirmação	cópia, inserção, remoção	não trata	não trata
Como coletar?	tipo	automática	automática	automática	automática
	abordagem	*eager*	*eager*	*eager*	*eager*
Como armazenar?	tipo	*naïve*	*naïve*, *transactional, hierarchical, transactional-hierarchical*	*hierarchical*	*naïve*
	local	separada	separada	junto	separada
Como consultar?	tipo	rastreamento, filtro	rastreamento	rastreamento, filtro	rastreamento

Esses desafios motivam o desenvolvimento de um novo modelo para subsidiar processos de integração de dados de fontes heterogêneas com a finalidade de reaplicação das decisões do usuário. O modelo MPPI, proposto com essa finalidade, é detalhado no Capítulo 4.

Capítulo

4

O Modelo MPPI

Nesta dissertação é proposto o modelo MPPI (**m**odelo de **p**rocedência para subsidiar **p**rocessos de **i**ntegração), o qual enfoca sistemas nos quais as fontes de dados podem ser atualizadas somente pelos seus proprietários, impossibilitando que a integração retifique eventuais conflitos de dados diretamente nessas fontes. O principal requisito do modelo MPPI é que ele ofereça suporte ao tratamento de todas as decisões de integração realizadas em processos anteriores, de forma que essas decisões possam ser reaplicadas automaticamente em processos de integração subsequentes.

No modelo MPPI, cada fonte de dados é vista como uma coleção de objetos. Cada objeto é composto por um conjunto de atributos, sendo que um subconjunto desses atributos deve ser obrigatoriamente utilizado para identificar o objeto, ou seja, o objeto deve possuir um conjunto de atributos chave.

O modelo MPPI tem como base as seguintes características:

- **Operações**: as decisões que o usuário toma para integrar dados heterogêneos são transformadas em operações de *cópia*, *edição*, *inserção* e *remoção* no modelo MPPI, as quais são armazenadas em um **repositório de operações**. As operações podem se relacionar de duas maneiras: por meio de transitividade e de alteração da chave do objeto;

- **Tratamento de operações de sobreposição**: durante a coleta das operações no processo de integração, quando uma operação sobrescreve o resultado de uma operação já existente, alguma decisão deve ser tomada para manutenção do repositório, desde que a nova operação pode introduzir inconsistências. O modelo MPPI oferece quatro políticas para tratamento de operações de sobreposição: *blind*, *restrict*, *undo* e *redo*;

- **Validação**: operações válidas são aquelas que levam os dados originais ao resultado integrado gerado pelo usuário. A validação auxilia na manutenção do repositório, uma vez que apenas operações válidas são mantidas no repositório. O modelo MPPI permite quatro tipos de validação: sem validação, validação da origem, validação do destino e validação completa; e

- **Reaplicação**: principal objetivo do modelo. Refere-se ao fato de que todas as decisões de integração já realizadas anteriormente devem ser resolvidas automaticamente e da mesma forma em processos de integração subsequentes. Portanto, consiste em métodos para reaplicar operações válidas nas fontes de dados, para que o usuário não precise retomar decisões de integração em cenários nos quais ele não pode alterar diretamente a fonte de dados. O modelo MPPI introduz dois métodos para realizar a reaplicação das operações: VRS (do inglês *Validate and Reapply in Separate*) e VRT (do inglês *Validate and Reapply in Tandem*).

Com relação aos quatro aspectos de procedência descritos no Capítulo 2, as características do modelo MPPI relacionam-se com esses aspectos como segue. O aspecto *o que coletar* leva em consideração quais atributos formam a procedência de cada operação. O modelo MPPI propõe um conjunto de operações que podem ser realizadas no processo de integração (i.e., *cópia*, *edição*, *inserção* e *remoção*), sendo que todas as operações são armazenadas em um mesmo repositório e possuem o mesmo conjunto de atributos. As operações rastreiam as alterações nos dados, além de armazenarem a origem dos dados em cada operação, caracterizando a procedência como *why-* e *where-provenance*.

Quanto ao aspecto *como coletar*, no modelo MPPI é considerada a abordagem automática. O usuário pode se concentrar na realização da integração, enquanto a procedência é coletada de forma transparente. Ainda nesse aspecto, o modelo proposto trata operações de sobreposição, as quais fazem parte das políticas de manutenção do repositório.

O aspecto *como armazenar* é caracterizado pelo emprego das operações sobre objetos, as quais são armazenadas separadas dos dados, ou seja, existe um repositório de operações. Esse tipo de armazenamento tem as vantagens descritas na Seção 2.3. Uma vez que o processo de integração não possui permissão para escrever nas fontes, anotações de procedência também não podem ser inseridas nas fontes e, portanto, esse é um motivo adicional para o armazenamento separado.

Com relação ao aspecto *como consultar*, o modelo MPPI permite dois tipos de consulta aos dados de procedência. O primeiro é a consulta tipo filtro, na qual dada uma operação, recuperam-se os dados sobre os quais ela atua, enquanto o segundo é a consulta tipo rastreamento, na qual recuperam-se as operações que atuam sobre um determinado dado. O modelo MPPI emprega esses tipos de consulta no desenvolvimento dos métodos de

reaplicação de operações, os quais podem ser desenvolvidos tanto baseados em consultas tipo rastreamento quanto em consultas tipo filtro.

Esse capítulo está estruturado da seguinte forma. A Seção 4.1 descreve o cenário motivacional para a proposta do modelo MPPI, além de introduzir um exemplo de integração que é usado ao longo do capítulo. As Seções 4.2 a 4.5 detalham cada uma das características do modelo MPPI: a Seção 4.2 introduz as operações propostas e define os relacionamentos que podem existir entre elas; a Seção 4.3 define operações de sobreposição e introduz as quatro políticas usadas pelo modelo MPPI para tratá-las; a Seção 4.4 descreve o processo de validação; e a Seção 4.5 apresenta os dois métodos propostos para realizar a reaplicação das operações. O capítulo é concluído na Seção 4.6 com as considerações finais.

4.1 Contextualização e Motivação

Esta seção tem como objetivo descrever o cenário motivacional para a proposta do modelo MPPI. A Seção 4.1.1 ilustra o cenário de integração considerado no trabalho, enquanto que a Seção 4.1.2 ilustra um exemplo que será utilizado para descrever as características do modelo proposto, e que é usado ao longo deste capítulo.

4.1.1 Cenário de Integração de Dados

O modelo MPPI enfoca o cenário de integração de dados de fontes heterogêneas. O usuário pode interagir com o processo de integração para tomar decisões sobre eventuais inconsistências, ou seja, qual dado deve ser considerado correto, dentre dados inconsistentes. A inclusão do usuário no processo de integração deve-se à necessidade de se obter um alto grau de precisão no resultado final.

A Figura 4.1 ilustra o cenário de integração considerado no trabalho. Inicialmente, existem n fontes a serem integradas. Um processo de integração semi-automático, realizado nesse cenário por meio da ferramenta Reconciliador de Dados Acadêmicos, identifica e exibe para o usuário as inconsistências encontradas entre as fontes, permitindo ao usuário decidir qual fonte possui o dado correto. O usuário toma decisões de integração realizando operações de transformação nos dados originais das fontes, tais como cópia, edição, inserção e remoção de dados entre as fontes. Essas operações são então coletadas e armazenadas em um repositório de operações. Após resolver as inconsistências, todas as fontes estão integradas. Considera-se nesse trabalho que o resultado do processo de integração são cópias locais das fontes e que as fontes originais não podem ser diretamente atualizadas pelo processo de integração por falta de permissão de escrita. Ou seja, o processo de integração possui permissão apenas para leitura das fontes originais.

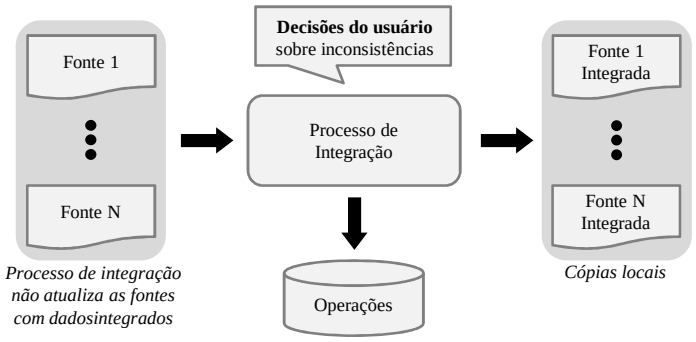


Figura 4.1: Integração de fontes de dados no tempo t_1.

Dessa maneira, todas as decisões de integração que o usuário realizou no processo de integração n são perdidas quando o processo de integração $n + 1$ for executado, pois as fontes irão fornecer os mesmos dados inconsistentes, além de novos dados. Para que o usuário não tenha que retomar as mesmas decisões em processos distintos, os dados de procedência armazenados pelo modelo proposto devem permitir a reaplicação automática de decisões já realizadas em processos anteriores. O uso das decisões em integração posteriores define o conceito de **reaplicação de operações**.

A Figura 4.2 ilustra a reaplicação das operações que o usuário realizou em processos anteriores. Quando as fontes são passadas para o processo de integração, esse primeiro invoca o processo de reaplicação de operações para que ele atualize as fontes de dados. Dessa forma, o processo de integração começa como se o usuário já tivesse tomado as decisões que tomou nos processos de integração anteriores, considerando todas as alterações que ocorreram nas fontes desde o último processo de integração. Portanto, o processo de integração no tempo t_n utiliza como entrada as fontes geradas pelo processo de reaplicação, ao contrário do processo no tempo t_1, no qual as fontes são passadas diretamente para o processo de integração.

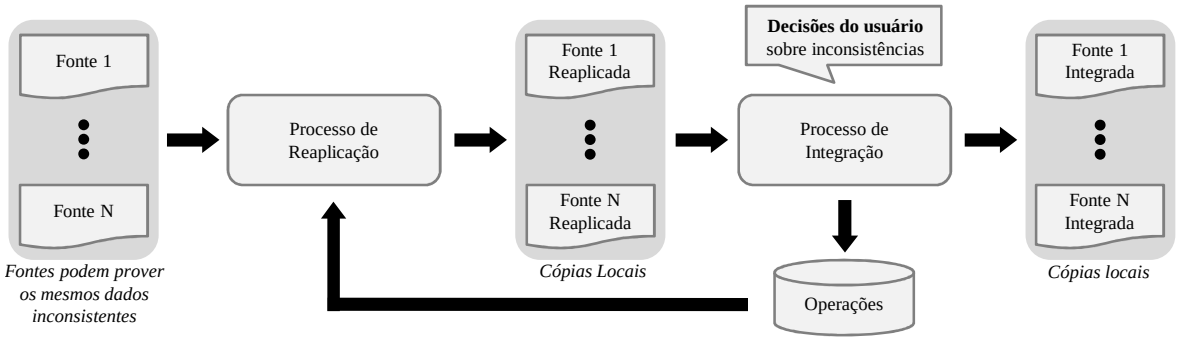


Figura 4.2: Reaplicação das decisões antes do processo de integração no tempo t_n.

Quando o usuário for participar do processo de integração no tempo t_n $(n > 1)$, ele apenas terá que tomar decisões a respeito de novas incosistências que surgiram desde o processo de integração no tempo t_{n-1}. Nesse sentido, o processo de integração torna-se

incremental, evitando que o usuário retome decisões de integração. Evitar essa retomada de decisões tem duas consequências diretas:

- O tempo gasto para integrar os dados consiste apenas no tempo necessário para resolver novas inconsistências; e
- As decisões tomadas pelo usuário mantêm-se consistentes ao longo dos processos de integração.

O primeiro item diz que o tempo de integração pelo usuário diminui de um processo para outro, se novas inconsistências não surgirem. Em contrapartida, se o usuário precisar retomar todas as decisões, o tempo de integração pelo usuário apenas aumenta de um processo para outro. Já o segundo item estabelece que, uma vez que o usuário toma a decisão apenas uma vez, ele não pode tomar decisões inconsistentes a respeito de uma mesma inconsistência de dados, por exemplo, escolhendo um dado como sendo correto em um processo e escolher outro dado como sendo correto em outro processo. Isso geraria inconsistência temporal dos dados sem que houvesse um motivo evidente para isso. Entretanto, se o usuário considerar que a decisão que ele tomou anteriormente está incorreta, ele pode alterar esse dado novamente, assim como qualquer outro dado.

O modelo MPPI proposto nesta dissertação tem como requisito central garantir a característica de reaplicação automática de decisões de integração, valendo-se das vantagens introduzidas por essa característica.

4.1.2 Exemplo Corrente

Nessa seção são feitas algumas considerações a respeito dos dados e de sua estrutura, junto com a definição de um exemplo que é utilizado para descrever as características do modelo proposto, as quais são detalhadas ao longo deste capítulo.

Considera-se que as fontes armazenam seus dados em estruturas de árvore, sendo que os nós internos representam objetos do mundo real e os nós folhas representam os atributos dos objetos. Cada fonte possui uma coleção de objetos, os quais são identificados por uma chave composta por um subconjunto de seus atributos. Outra característica é que um objeto (objeto pai) pode ser composto por outros objetos (objeto filho ou subobjeto). A identificação de um subobjeto é feita de maneira hierárquica, ou seja, a chave do objeto pai faz parte da chave de um subobjeto.

Utilizando essa estruturação para os dados das fontes, é definido o exemplo corrente, ilustrado na Figura 4.3. Nessa figura são ilustrados três objetos de três fontes distintas, as quais contêm dados curriculares a respeito de três docentes: João, Luis e Maria. Esses objetos são referências bibliográficas comuns aos três docentes. O objeto *artigo* possui cinco atributos, *titulo*, *ano*, *congresso*, *paginas* e *local*, sendo sua chave o atributo *titulo*. Além desses atributos, o objeto *artigo* possui subobjetos *autor*. Cada objeto *autor* é

composto pelos atributos *nome* e *ordemCitacao*, sendo que esse objeto é identificado pelo atributo *nome*, mais o atributo *titulo* do *artigo* a que pertence. O atributo *titulo* é incorporado à chave do objeto *autor* devido à identificação hierárquica de subobjetos.

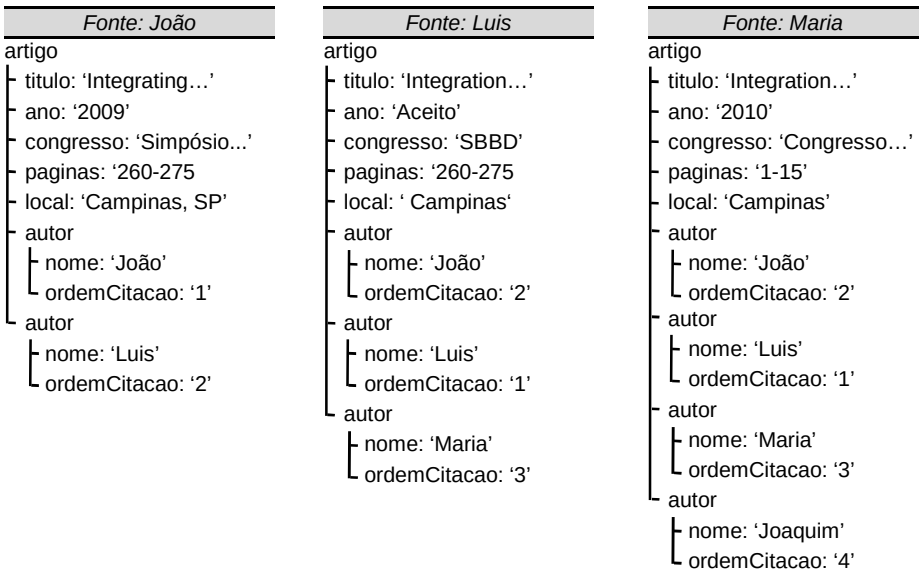


Figura 4.3: Exemplo de fontes com dados inconsistentes.

Como pode ser observado na Figura 4.3, esses três objetos possuem valores diferentes para alguns atributos, embora descrevam o mesmo artigo. Por exemplo, o atributo *ano* possui o valor '*2009*' na fonte João, '*Aceito*' na fonte Luis e '*2010*' na fonte Maria. A diferença entre os valores dos atributos mostra um exemplo de inconsistência de dados entre as fontes.

Essas fontes são então integradas par a par utilizando a ferramenta Reconciliador de Dados Acadêmicos, a qual oferece suporte à reconciliação semi-automática de currículos de docentes diferentes. A ferramenta: compara dados de objetos acadêmicos; sincroniza a representação visual dos dados exibidos desses objetos, emparelhando os que correspondem a objetos similares e desemparelhando os que não têm correspondentes similares; e permite que o usuário reconcilie os dados sincronizados. Objetos são considerados correspondentes quando seus atributos chave são similares.

As Figuras 4.4, 4.5 e 4.6 ilustram a sincronização par a par dos objetos representados na Figura 4.3. Quando dois objetos são considerados correspondentes mas possuem inconsistências, as inconsistências são ressaltadas para que o usuário as identifique visualmente. Por exemplo, na Figura 4.4, a inconsistência identificada no atributo *ano* é marcada em cinza claro. Em contrapartida, atributos marcados em cinza escuro representam valores consistentes, tal como o atributo *titulo*. Com as fontes sincronizadas, o usuário pode então realizar operações a fim de torná-las consistentes. Essas operações, as quais fazem parte da proposta do modelo MPPI, são introduzidas na Seção 4.2.

Figura 4.4: Fontes João e Luis sincronizadas.

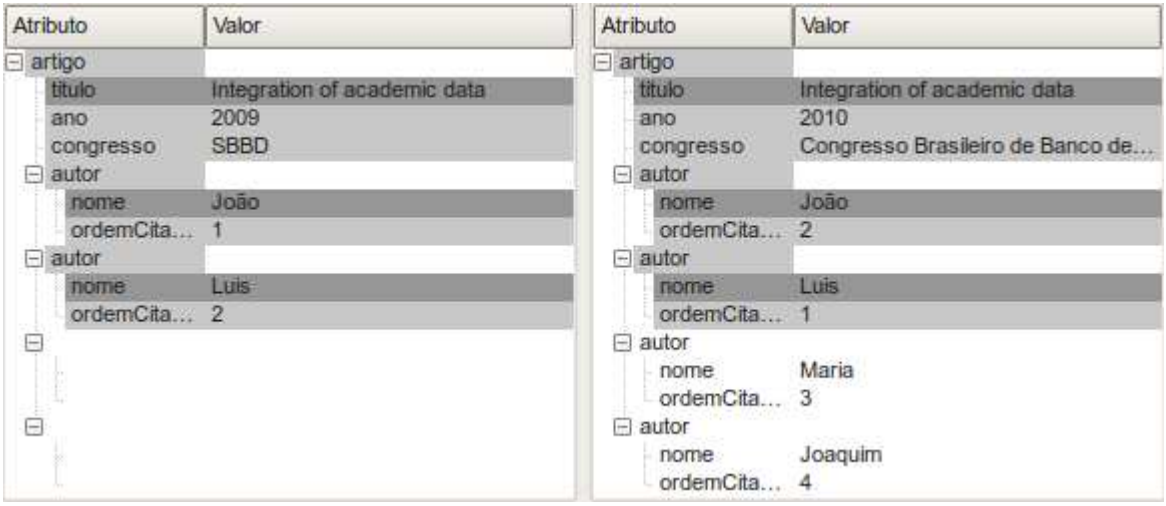


Figura 4.5: Fontes João e Maria sincronizadas.

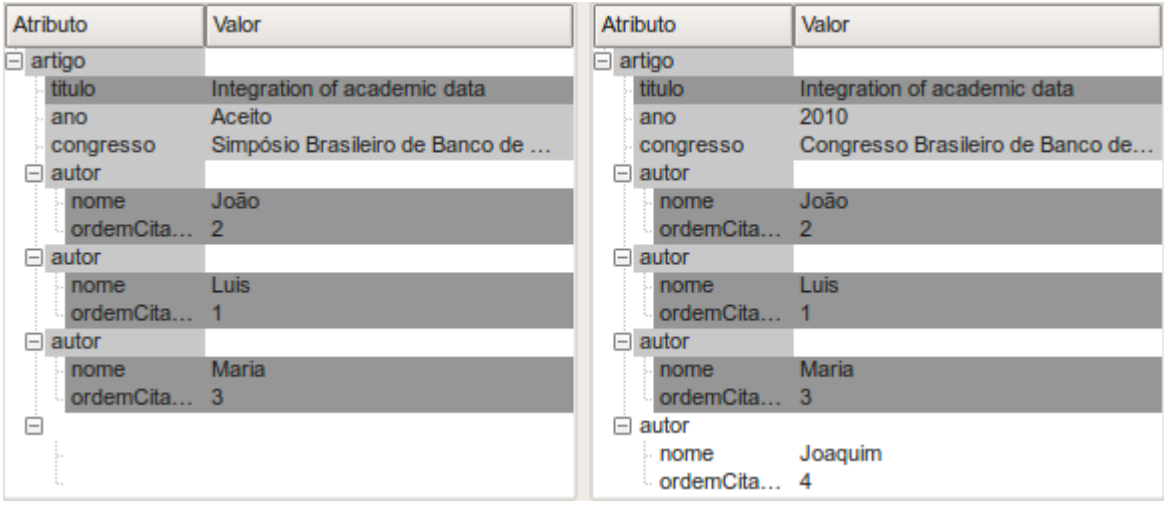


Figura 4.6: Fontes Luis e Maria sincronizadas.

4.2 Operações

A primeira característica do modelo MPPI é ser um modelo baseado em operações. Na Seção 4.2.1 são propostas as operações que os usuários podem efetuar sobre os dados durante o processo de integração, de forma que a procedência das transformações nesses dados sejam rastreadas pelo modelo MPPI. Na Seção 4.2.2 são definidos os relacionamentos que podem existir entre as operações.

4.2.1 Descrição das Operações

O modelo MPPI introduz quatro operações: *cópia*, *edição*, *inserção* e *remoção*. Essas operações são armazenadas no **repositório de operações**, em uma relação *Operação*. O repositório de operações possui as seguintes características:

- Ele é um conjunto ordenado de operações, identificado pelo símbolo R, sendo que a ordem das operações é dada pela ordem temporal em que elas foram inseridas;
- A i-ésima operação é recuperada no repositório utilizando a sintaxe de índice $R[i]$;
- O índice de uma operação Op pode ser recuperado utilizando a função $Indice(Op)$;
- Uma operação Op_1 é menor que uma operação Op_2, representado por $Op_1 < Op_2$, quando $Indice(Op_1) < Indice(Op_2)$; e
- O número de operações em um repositório é determinado por $|R|$.

A relação *Operação*, por sua vez, possui o seguinte esquema:

- *id*: identifica a operação no repositório;
- *op*: identifica o tipo da operação realizada e possui domínio {*cp, ed, in, rm*} para *cópia*, *edição*, *inserção* e *remoção*, respectivamente.
- *origem*: identifica a fonte de dados considerada como correta na operação;
- *destino*: identifica a fonte de dados considerada como incorreta na operação;
- *idObjeto*: identifica o conjunto de atributos chave do objeto, bem como seus valores.
- *atributo*: consiste no atributo que está sendo considerado na operação;
- *valorOrigem*: consiste no valor do *atributo* na *origem*; e
- *valorDestino*: consiste no valor do *atributo* no *destino*.

Algumas considerações sobre esse esquema são descritas a seguir. No processo de integração par a par, cada uma das fontes que participa de uma inconsistência recebe um nome, de acordo com o papel que desenvolve na integração. Se o resultado da fonte é considerado como sendo o correto naquela inconsistência, então a fonte é denominada fonte origem da operação (ou apenas *origem*), pois forneceu o dado para a resolução da inconsistência. Em contrapartida, a outra fonte é denominada fonte destino da operação (ou apenas *destino*). Já *idObjeto* representa um caminho a ser percorrido na estrutura de árvore da fonte de maneira a recuperar o objeto utilizado na operação. Por fim, com o intuito de facilitar a leitura, a expressão "operação com *id* igual a x" é escrita apenas como "operação x" ao longo deste capítulo.

A Figura 4.7 ilustra um conjunto de operações em um determinado repositório. As operações propostas pelo modelo MPPI seguem o esquema da relação *Operação*, e possuem as seguintes características:

- *Cópia*: no momento da integração de duas fontes, pode ocorrer que objetos correspondentes possuam valores inconsistentes para alguns de seus atributos. Nesse caso, o usuário pode copiar de uma fonte para outra o valor que considerar como correto. A *cópia* é uma operação binária, pois atua sobre o *destino* dado uma *origem*. As operações *2, 5, 7, 8, 9, 10, 11* e *12* da Figura 4.7 ilustram operações de *cópia*. Por exemplo, a operação *7* significa que o *valorOrigem* '*3*' do *atributo* '*ordemCitacao*' do '*autor*' identificado por '*nome=Maria*' do '*artigo*' identificado por '*titulo=Integration...*' da *origem* '*Luis*' está sendo copiado para o mesmo '*autor*' do *destino* '*Joao*', sobrescrevendo seu *valorDestino* '*null*';

- *Edição*: no momento da integração de duas fontes, pode ocorrer que objetos correspondentes possuam valores incorretos (independentemente de serem consistentes ou inconsistentes). Nesse caso, o usuário pode editar o valor de alguma das fontes (ou de ambas as fontes) para corrigí-lo. Como a *edição* é uma operação que atua sobre um *destino* sem que seja necessária uma *origem*, ou seja, é uma operação unária, o atributo *origem* não é utilizado. Assim, *valorOrigem* é o valor resultante da edição, enquanto que o atributo *valorDestino* atua como o valor que foi sobrescrito pela edição. As operações *1* e *3* da Figura 4.7 ilustram operações de *edição*. Por exemplo, a operação *3* significa que o *valorDestino* '*4*' do *atributo ordemCitacao* do '*autor*' identificado por '*nome=Joaquim*' do '*artigo*' identificado por '*titulo=Integration...*' do *destino* '*Maria*' está sendo editado para o *valorOrigem* '*null*';

- *Inserção*: no momento da integração de duas fontes, pode ocorrer de um objeto estar presente apenas em uma fonte. Para tornar as fontes consistentes, o usuário pode inserir o objeto faltante na outra fonte. Uma operação de *inserção* é composta por operações de *cópia*, sendo uma para cada atributo não chave do objeto. Para

a *inserção*, os atributos *atributo*, *valorOrigem* e *valorDestino* não são utilizados e, portanto, possuem valor nulo. A operação de *inserção* cria um objeto no destino contendo valores apenas para os atributos chave, os quais são recuperados por meio do atributo *idObjeto*. As operações de *cópia* derivadas de uma inserção preenchem os valores dos demais atributos do objeto no destino. A *inserção* é uma operação binária, uma vez que insere um objeto da *origem* no *destino*. A operação *6* da Figura 4.7 ilustra uma operação de *inserção*, enquanto que a operação *7* ilustra a operação de *cópia* derivada dessa inserção. Por exemplo, a operação *6* significa que o '*autor*' identificado por '*nome=Maria*' do '*artigo*' identificado por '*titulo=Integration...*' da *origem* '*Luis*' está sendo inserido no *destino* '*João*'; e

- *Remoção*: no momento da integração, o usuário pode identificar que um objeto está incorretamente inserido em uma determinada fonte. Nesse caso, o usuário pode remover esse objeto incorreto da fonte. Uma operação de *remoção* é composta por operações de *edição*, as quais modificam o valor de cada atributo não chave do objeto para nulo, sendo que uma operação de *remoção* só é permitida se os valores de todos os atributos não chave do objeto forem nulos. A remoção é uma operação unária, pois remove um objeto do *destino* sem precisar de uma *origem*. Portanto, na *remoção*, os atributos *atributo*, *valorOrigem* e *valorDestino* não são utilizados. A operação *4* da Figura 4.7 ilustra uma operação de remoção, enquanto a operação *3* ilustra a operação de edição derivada dessa remoção. Por exemplo, a operação *4* significa que o '*autor*' identificado por '*nome=Joaquim*' do '*artigo*' identificado por '*titulo=Integration...*' está sendo removido do *destino* '*Maria*'.

Operação

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	João	artigo[titulo=Integrating...]	titulo	Integration...	Integrating...
2	cp	João	Luis	artigo[titulo=Integration...]	ano	2009	Aceito
3	ed	null	Maria	artigo[titulo=Integration...] / autor[nome=Joaquim]	ordemCitacao	null	4
4	rm	null	Maria	artigo[titulo=Integration...] / autor[nome=Joaquim]	null	null	null
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio...	SBBD
6	in	Luis	João	artigo[titulo=Integration...] / autor[nome=Maria]	null	null	null
7	cp	Luis	João	artigo [titulo=Integration...] / autor[nome=Maria]	ordemCitacao	3	null
8	cp	Maria	João	artigo [titulo=Integration...] / autor[nome=João]	ordemCitacao	2	1
9	cp	Maria	João	artigo [titulo=Integration...] / autor[nome=Luis]	ordemCitacao	1	2
10	cp	Luis	Maria	artigo [titulo=Integration...]	ano	2009	2010
11	cp	João	Maria	artigo[titulo=Integration...]	paginas	260-275	1-15
12	cp	Luis	Maria	artigo[titulo=Integration...]	congresso	Simpósio...	Congresso...

Figura 4.7: Exemplo de um conjunto de operações.

4.2.2 Relacionamento entre Operações

Quando uma operação a afeta o resultado de uma operação b, as operações a e b são denominadas operações relacionadas. As operações propostas podem estar relacionadas de duas maneiras diferentes:

- Operações transitivas; e
- Operações que alteram a chave.

Definição 1. *Operação transitiva:* Quando uma operação b depende do resultado da operação a, e $a < b$, então a operação b é transitivita à operação a.

Definição 2. *Operação transitiva direta:* a operação b é transitiva direta à operação a quando $a(destino, idObjeto)$ é igual a $b(origem, idObjeto)$ e $a < b$. Representado por $a \rightarrow b$.

As operações *5* e *12* da Figura 4.7 são exemplos de operações transitivas diretas.

Definição 3. *Operação transitiva indireta:* se $a \rightarrow b$ e $b \rightarrow c$, então c é transitiva indireta a a. Representado por $a \rightarrow_n c$. Se $c \rightarrow d$, então d é transitiva indireta a a e b, e assim por diante.

Definição 4. *Operação que altera a chave:* quando uma operação altera a chave de um objeto.

A operação *1* da Figura 4.7 é exemplo de uma operação que altera a chave do objeto destino da operação.

Operações transitivas e que alteram a chave são empregadas pelo modelo MPPI no tratamento de operações de sobreposição e na reaplicação das operações, tal como descrito nas Seções 4.3 e 4.5, respectivamente.

4.3 Operações de Sobreposição

Operações de sobreposição ocorrem quando a inserção de uma nova operação no repositório sobrepõe outra operação já existente no repositório. Nesse caso, o resultado gerado pela aplicação da operação de sobreposição pode introduzir inconsistências. A seguir são feitas algumas definições de termos relacionados às operações de sobreposição.

Definição 5. *Operação de sobreposição:* uma operação que sobrepõe o resultado de outra operação.

Definição 6. *Operação de sobreposição do destino:* quando os atributos (*destino*, *idObjeto*, *atributo*) de uma operação a aparecem como (*destino*, *idObjeto*, *atributo*) em uma operação b ($a < b$). Representado por $a \leftarrow_d b$.

Definição 7. *Operação de sobreposição da origem:* quando os atributos (*origem*, *idObjeto*, *atributo*) de uma operação a aparecem como (*destino*, *idObjeto*, *atributo*) em uma operação b ($a < b$). Representado por $a \leftarrow_o b$.

Definição 8. *Operação sobreposta:* operação que tem seu resultado sobreposto por uma operação de sobreposição.

A Figura 4.8 ilustra a operação *13* a ser inserida no repositório da Figura 4.7. Trata-se de uma operação de sobreposição da origem pois o valor do atributo *congresso* do *destino* dessa operação (João) está sendo alterado e o valor desse mesmo atributo está sendo utilizado como *origem* em uma operação já existente no repositório (operação *5*). Em outras palavras, as fontes João e Luis estão se tornando inconsistentes porque o valor do atributo *congresso* da fonte João está sendo alterado para '*XXIV Simpósio...*' na operação *13*, embora o usuário já tenha tomado uma decisão para tornar esse atributo consistente com valor '*Simpósio...*' nessas fontes na operação *5*. O fato da operação *13* sobrepor a operação *5* torna a operação *12* inconsistente com a operação *5*, pois $5 \rightarrow 12$, deixando as fontes João e Maria inconsistentes.

Não só as fontes da operação *5* estão se tornado inconsistentes, como também todas as fontes das operações $\{a \mid 5 \rightarrow a \ \cup \ 5 \rightarrow_n a\}$. Ou seja, todas as operações transitivas diretas e indiretas da operação de sobreposição deixam de ser consistentes com a origem da operação *5*. O caso é simétrico para operação de sobreposição do destino.

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio...	SBBD
...							
12	cp	Luis	Maria	artigo[titulo=Integration...]	congresso	Simpósio...	Congresso...
13	ed	null	João	artigo[titulo=Integration...]	congresso	XXIV Simpósio...	Simpósio...

Figura 4.8: Exemplo de uma operação de sobreposição (operação *13*).

Portanto, durante a coleta das operações no processo de integração, quando uma operação sobrescreve o resultado de uma operação já existente, alguma decisão deve ser tomada para manutenção do repositório, desde que a nova operação pode introduzir inconsistências. Com o intuito de auxiliar o usuário no momento da integração, o modelo MPPI introduz quatro políticas para o tratamento de operações de sobreposição: *blind* (Seção 4.3.1), *restrict* (Seção 4.3.2), *undo* (Seção 4.3.3) e *redo* (Seção 4.3.4). Na Seção 4.3.5 são feitas algumas considerações a respeito dessas políticas.

4.3.1 Política *Blind*

Na política *blind*, operações de sobreposição não são tratadas, sendo que o sistema não toma conhecimento da ocorrência da sobreposição pois nenhuma verificação é feita. Ou seja, é permitida a ocorrência de operações de sobreposição e tanto a operação sobreposta quanto suas respectivas operações transitivas permanecem no repositório. Dessa forma, nenhum tempo adicional para tratamento da operação de sobreposição é necessário na inserção dessa operação no repositório.

A Figura 4.9 ilustra a inserção da operação de sobreposição *13* da Figura 4.8 utilizando a política *blind*. Mesmo sobrepondo a operação *5*, a operação *13* é inserida no repositório, tornando as fontes das operações *5* e *12* inconsistentes.

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio…	SBBD
...							
12	cp	Luis	Maria	artigo[titulo=Integration…]	congresso	Simpósio…	Congresso…
13	ed	null	João	artigo[titulo=Integration...]	congresso	XXIV Simpósio...	Simpósio...

Figura 4.9: Inserção de uma operação de sobreposição pela política *blind*.

4.3.2 Política *Restrict*

Na política *restrict*, a inserção de operações de sobreposição no repositório não é permitida. Para tanto, para cada nova operação a ser inserida, é feita uma verificação visando garantir que nenhuma operação será sobreposta pela nova operação. A nova operação somente é inserida quando não sobrepuser nenhuma operação já existente.

A Figura 4.10 ilustra que a política *restrict* impede a inserção da operação de sobreposição *13* da Figura 4.8 no repositório, desde que essa operação sobrepõe a operação *5*. Dessa forma, as fontes das operações *5* e *12* continuam consistentes.

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio…	SBBD
...							
12	cp	Luis	Maria	artigo[titulo=Integration…]	congresso	Simpósio…	Congresso…
~~13~~	~~ed~~	~~null~~	~~João~~	~~artigo[titulo=Integration...]~~	~~congresso~~	~~XXIV Simpósio...~~	~~Simpósio...~~

Figura 4.10: Tratamento de uma operação de sobreposição pela política *restrict*.

O algoritmo da política *restrict* (Algoritmo 1) recebe como parâmetros um repositório R e uma operação Op a ser inserida em R, e gera como saída o repositório R atualizado e o indicador $TipoSobreposicao$, o qual sinaliza para a aplicação a ocorrência de uma operação sobreposta. Na linha 1, ele verifica a existência de uma operação que será sobreposta pela operação Op, no repositório R. Caso nenhuma operação sobreposta seja encontrada (linha 2), então a operação é adicionada ao final do repositório (linha 3).

Algoritmo 1 Restrict (R, Op, TipoSobreposição)

Entrada: R /* repositório */
Op /* operação a ser inserida */
Saída: R /* repositório atualizado */
$TipoSobreposicao$

```
1: DetectarSobreposicao(R, Op, OpSob, TipoSobreposicao)
2: se TipoSobreposicao = 'nenhum' então
3:    R ← R ∪ {Op}
4: fim se
```

O Algoritmo 2 verifica se alguma operação é sobreposta por outra operação. Esse algoritmo recebe como parâmetros um repositório R e uma operação Op (possivelmente de sobreposição), e gera como saída uma operação sobreposta $OpSob$ e o tipo de sobreposição $TipoSobreposicao$ que ocorreu. O algoritmo percorre todas as operações do repositório R (linhas 3 a 15) em busca de alguma operação $R[i]$ cujo *idObjeto* e *atributo* sejam iguais aos da operação Op (linha 4). Em seguida, se alguma operação $R[i]$ for encontrada, é verificado se o *destino* da operação Op é igual à *origem* ou ao *destino* de $R[i]$. Se for igual à *origem*, então o tipo de sobreposição é '*origem*' (linhas 5 a 7), enquanto que se for igual ao *destino*, o tipo de sobreposição é '*destino*' (linhas 9 a 11). Em ambos os casos, a operação $R[i]$ é a operação sobreposta $OpSob$. Caso não seja encontrada uma operação sobreposta, a operação sobreposta $OpSob$ é retornada como nula (linha 1) e o tipo de sobreposição como '*nenhum*' (linha 2).

4.3.3 Política *Undo*

A política *undo* permite a ocorrência de operações de sobreposição, porém realiza um tratamento especial para essas operações. Tanto para operações de sobreposição da origem quanto do destino, a operação sobreposta é removida do repositório, e o resultado da aplicação das operações transitivas diretas e indiretas à operação sobreposta é desfeito. Em seguida, as operações transitivas diretas e indiretas também são removidas do repositório.

A Figura 4.11 ilustra a inserção da operação de sobreposição *13* da Figura 4.8 pela política *undo*. A operação *5* é removida do repositório, bem como a operação *12*, uma vez que $5 \rightarrow 12$. Dessa forma, as fontes das operações *5* e *12* voltam a ficar inconsistentes.

Algoritmo 2 DetectarSobreposicao (R, Op, OpSob, TipoSobreposicao)

Entrada: R /* repositório */
Op /* operação de sobreposição */
Saída: $OpSob$ /* operação sobreposta */
$TipoSobreposicao$

```
1:  OpSob ← null
2:  TipoSobreposicao ← 'nenhum'
3:  para i = 0 até |R| faça
4:    se (R[i].idObjeto = Op.idObjeto e R[i].atributo = Op.atributo) então
5:      se (R[i].origem = Op.destino) então
6:        OpSob ← R[i]
7:        TipoSobreposicao ← 'origem'
8:        retornar
9:      senão se (R[i].destino = Op.destino) então
10:       OpSob ← R[i]
11:       TipoSobreposicao ← 'destino'
12:       retornar
13:     fim se
14:   fim se
15: fim para
```

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
~~5~~	~~cp~~	~~João~~	~~Luis~~	~~artigo[titulo=Integration...]~~	~~congresso~~	~~Simpósio...~~	~~SBBD~~
...							
~~12~~	~~cp~~	~~Luis~~	~~Maria~~	~~artigo[titulo=Integration...]~~	~~congresso~~	~~Simpósio...~~	~~Congresso...~~
13	ed	null	João	artigo[titulo=Integration...]	congresso	XXIV Simpósio...	Simpósio...

Figura 4.11: Inserção de uma operação de sobreposição pela política *undo*.

O algoritmo da política *undo* (Algoritmo 3) recebe como parâmetros um repositório R e uma operação Op a ser inserida em R, sendo que ao final do algoritmo o repositório R está atualizado de acordo com essa política. Inicialmente, procura-se por uma possível operação sobreposta por Op no repositório R (linha 1), utilizando o Algoritmo 2. Se uma operação sobreposta foi encontrada (linhas 2 a 14), procura-se pelas operações transitivas diretas (linha 3) e indiretas (linha 4) da operação sobreposta. Em seguida, cada operação transitiva tem seu resultado desfeito e é removida do repositório (linhas 5 a 8). Na política *undo*, se o tipo de sobreposição for '*destino*', a operação sendo inserida recebe o *valorDestino* da operação sobreposta (linhas 9 a 11). Em seguida, a operação sobreposta é desfeita e removida do repositório (linhas 12 e 13). Por fim, a nova operação (Op) é inserida no repositório (linha 15).

O algoritmo da política *undo* utiliza dois outros algoritmos para encontrar e recuperar todas as operações transitivas diretas e indiretas da operação sobreposta. O primeiro

Algoritmo 3 Undo (R, Op)

Entrada: R /* repositório */
Op /* operação a ser inserida */
Saída: R /* repositório atualizado */

1: $DetectarSobreposicao(R, Op, OpSob, TipoSobreposicao)$
2: **se** $TipoSobreposicao$ <> '$nenhum$' **então**
3: $OpTrans \leftarrow EncontrarOpTransDiretas(R, OpSob)$
4: $OpTrans \leftarrow EncontrarOpTransIndiretas(R, OpSob, OpTrans)$
5: **para toda** *operacao* d **em** $OpTrans$ **faça**
6: $Desfazer$ *a operacao* d /* realizado pela aplicação */
7: $R \leftarrow R - \{d\}$
8: **fim para**
9: **se** $TipoSobreposicao =$ '$destino$' **então**
10: $Op.valorDestino \leftarrow OpSob.valorDestino$
11: **fim se**
12: $Desfazer$ *a operacao* $OpSob$ /* realizado pela aplicação */
13: $R \leftarrow R - \{OpSob\}$
14: **fim se**
15: $R \leftarrow R \cup \{Op\}$

algoritmo (Algoritmo 4) encontra todas as operações transitivas diretas a uma operação. Esse algoritmo recebe como parâmetros um repositório R e uma operação Op, e tem como resultado todas as operações transitivas diretas à operação Op, as quais são armazenadas em um conjunto ordenado de operações denominado $OperacoesTransitivas$. Inicialmente as $OperacoesTransitivas$ é um conjunto vazio (linha 1). Todas as operações $R[i]$ do repositório com índice i maior que o índice da operação Op são percorridas (linhas 2 a 5). Caso a operação $R[i]$ tenha como *origem* a mesma fonte utilizada como *destino* em Op e os valores dos atributos *idObjeto* e *atributo* também sejam os mesmos, a operação $R[i]$ é adicionada ao conjunto $OperacoesTransitivas$.

Algoritmo 4 EncontrarOpTransDiretas (R, Op, OperacoesTransitivas)

Entrada: R /* repositório */
Op /* operação sobreposta */
Saída: $OperacoesTransitivas$ /* operações transitivas diretas */

1: $OperacoesTransitivas \leftarrow \emptyset$
2: **para** $i \leftarrow Indice(Op) + 1$ **até** $|R|$ **faça**
3: **se** ($R[i].origem = Op.destino$ **e**
$R[i].idObjeto = Op.idObjeto$ **e**
$R[i].atributo = Op.atributo$) **então**
4: $OperacoesTransitivas \leftarrow OperacoesTransitivas \cup \{R[i]\}$
5: **fim se**
6: **fim para**

O segundo algoritmo (Algoritmo 5) recupera as operações transitivas indiretas a uma operação. Esse algoritmo recebe como parâmetros um repositório R e um conjunto $OperacoesTransitivas$, o qual representa as operações transitivas a alguma operação a. O resultado desse algoritmo é o conjunto $OperacoesTransitivas$ contendo as operações transitivas diretas e indiretas à operação a. Inicialmente todas as operações do conjunto $OperacoesTransitivas$ são percorridas (linhas 1 a 7). Em seguida, todas as operações $R[j]$ com índice j maior que o índice da operação $OperacoesTransitivas[i]$ são percorridas (linhas 2 a 5). Caso a operação $R[j]$ tenha como *origem* a mesma fonte utilizada como *destino* em $OperacoesTransitivas[i]$ e os valores dos atributos *idObjeto* e *atributo* também sejam os mesmos, a operação $R[j]$ é adicionada ao final do conjunto $OperacoesTransitivas$. Quando uma operação transitiva indireta é adicionada ao conjunto $OperacoesTransitivas$, ela também é considerada pelo laço das linhas 1 a 7, para que as operações transitivas indiretas em n passos também possam ser recuperadas.

Algoritmo 5 EncontrarOpTransIndiretas (R, OperacoesTransitivas)

Entrada: R /* repositório */
$OperacoesTransitivas$ /* operações transitivas diretas */
Saída: $OperacoesTransitivas$ /* operações transitivas diretas e indiretas */

1: **para** $i \leftarrow 0$ **até** $|OperacoesTransitivas|$ **faça**
2: **para** $j \leftarrow Indice(OperacoesTransitivas[i]) + 1$ **até** $|R|$ **faça**
3: **se** $(R[j].origem = OperacoesTransitivas[i].destino$ **e**
$R[j].idObjeto = OperacoesTransitivas[i].idObjeto$ **e**
$R[j].atributo = OperacoesTransitivas[i].atributo)$ **então**
4: $OperacoesTransitivas \leftarrow OperacoesTransitivas \cup \{R[j]\}$
5: **fim se**
6: **fim para**
7: **fim para**

4.3.4 Política *Redo*

A política *redo* permite a ocorrência de operações de sobreposição, porém realiza um tratamento especial para essas operações. Quando uma operação de sobreposição da origem é inserida no repositório, a operação sobreposta é movida para o final do repositório. Quando uma operação de sobreposição do destino é inserida no repositório, a operação sobreposta é removida do repositório. O resultado das operações transitivas diretas e indiretas à operação sobreposta é refeito, considerando o novo valor da operação de sobreposição. Em seguida, as operações transitivas diretas e indiretas são movidas para o final do repositório, pois agora dependem da operação de sobreposição inserida. Diferentemente da política *undo*, a política *redo* mantém no repositório as operações transitivas diretas e indiretas.

A Figura 4.12 ilustra a inserção da operação de sobreposição *13* da Figura 4.8 pela política *redo*. Nesse caso, a operação de sobreposição *13* é inserida no repositório e as operações *5* e *12* são refeitas utilizando o novo *valorOrigem* para a fonte João. Ademais, após a operação de sobreposição *13*, as operações *5* e *12* são movidas para o final do repositório pois $13 \rightarrow 5$ e $13 \rightarrow_n 12$. Dessa forma, as fontes da operações *5* e *12* permanecem consistentes ao final do processo, bem como as fontes de suas operações transitivas.

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
~~5~~	~~cp~~	~~João~~	~~Luis~~	~~artigo[titulo=Integration...]~~	~~congresso~~	~~Simpósio...~~	~~SBBD~~
...							
~~12~~	~~cp~~	~~Luis~~	~~Maria~~	~~artigo[titulo=Integration...]~~	~~congresso~~	~~Simpósio...~~	~~Congresso...~~
13	ed	null	João	artigo[titulo=Integration...]	congresso	XXIV Simpósio...	Simpósio...
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	XXIV Simpósio...	SBBD
12	cp	Luis	Maria	artigo[titulo=Integration...]	congresso	XXIV Simpósio...	Congresso...

Figura 4.12: Inserção de uma operação de sobreposição pela política *redo*.

O algoritmo da política *redo* (Algoritmo 6) recebe como parâmetros um repositório R e uma operação Op a ser inserida em R, sendo que ao final do algoritmo o repositório R está atualizado de acordo com essa política. Inicialmente, a operação Op é inserida no final do repositório (linha 1). Em seguida, procura-se por uma operação sobreposta (linha 2) utilizando o Algoritmo 2. Caso uma operação sobreposta $OpSob$ do tipo *origem* tenha sido encontrada (linhas 3 a 6), a operação $OpSob$ recebe como *valorOrigem* o *valorOrigem* da operação de sobreposição Op (linha 4) e a operação $OpSob$ é refeita (linha 5) e movida para o final do repositório (linha 6), pois é transitiva à operação de sobreposição Op. Caso a operação sobreposta seja do tipo '*destino*' (linhas 7 a 10), o operação Op recebe como *valorDestino* o *valorDestino* da operação sobreposta $OpSob$ (linha 8) e a operação $OpSob$ é removida do repositório (linha 9).

Caso tenha sido encontrada uma operação sobreposta de qualquer tipo (linhas 11 a 19), as operações transitivas diretas (linha 12) e indiretas (linha 13) da operação sobreposta são recuperadas utilizando os Algoritmos 4 e 5, respectivamente, e armazenadas no conjunto $OpTrans$. Em seguida, todas as operações no conjunto $OpTrans$: (i) recebem como *valorOrigem* o *valorOrigem* da operação de sobreposição Op (linha 15); (ii) são refeitas considerando o novo *valorOrigem* da operação (linha 16); e (iii) são movidas para o final do repositório, pois são transitivas à operação de sobreposição Op.

Algoritmo 6 Redo (R, Op)

Entrada: R /* repositório */
Op /* operação a ser inserida */
Saída: R /* repositório atualizado */

1: $R \leftarrow R \cup Op$
2: $DetectarSobreposicao(R, Op, OpSob, TipoSobreposicao)$
3: **se** $TipoSobreposicao =$ '$origem$' **então**
4: $OpSob.valorOrigem \leftarrow Op.valorOrigem$
5: $Refazer\ a\ operacao\ OpSob$ /* realizado pela aplicação */
6: $Mover\ OpSob\ para\ o\ fim\ do\ repositorio$
7: **senão se** $TipoSobreposicao =$ '$destino$' **então**
8: $Op.valorDestino \leftarrow OpSob.valorDestino$
9: $R \leftarrow R - \{OpSob\}$
10: **fim se**
11: **se** $TipoSobreposicao <>$ '$nenhum$' **então**
12: $OpTrans \leftarrow EncontrarOpTransDiretas(R, OpSob)$
13: $OpTrans \leftarrow EncontrarOpTransIndiretas(R, OpSob, OpTrans)$
14: **para toda** $operacao\ d$ **em** $OpTrans$ **faça**
15: $d.valorOrigem \leftarrow Op.valorOrigem$
16: $Refazer\ a\ operacao\ d$ /* realizado pela aplicação */
17: $Mover\ d\ para\ o\ fim\ do\ repositorio$
18: **fim para**
19: **fim se**

4.3.5 Considerações sobre as Políticas de Sobreposição

Nessa seção foram analisadas operações que sobrepõem o resultado de outras operações, denominadas operações de sobreposição. Para tratar operações de sobreposição, o modelo MPPI oferece quatro políticas:

- *Blind*: operações de sobreposição são permitidas, mas não são tratadas;
- *Restrict*: operações de sobreposição não são permitidas;
- *Undo*: operações de sobreposição são permitidas, sendo que as operações afetadas pela operação de sobreposição são desfeitas e removidas do repositório; e
- *Redo*: operações de sobreposição são permitidas, sendo que as operações afetadas pela operação de sobreposição são refeitas, utilizando o valor definido pela operação de sobreposição.

A proposta dessas quatro políticas tem como objetivo garantir flexibilidade ao modelo MPPI, possibilitando que o usuário possa adotar diferentes abordagens dado as necessidades de sua aplicação. As políticas *restrict* e *redo* são as políticas mais adequadas para processos de integração, uma vez que garantem que todas as inconsistências resolvidas

pelos usuários são mantidas ao final do processo. Em contrapartida, as políticas *blind* e *undo* não oferecem essa garantia: a primeria porque não trata operações de sobreposição e a segunda porque retorna as fontes ao estado anterior à alteração feita pelo usuário. Embora não garantam a consistência das fontes ao final do processo, essas políticas são propostas para cobrir casos em que operações de sobreposição não precisam ser tratadas (*blind*), e casos em que uma sobreposição implica que as operações sobrepostas e suas respectivas operações transitivas devem ser revistas pelo usuário a fim de garantir que o novo valor da operação de sobreposição é apropriado para todas as fontes (*undo*).

4.4 Validação

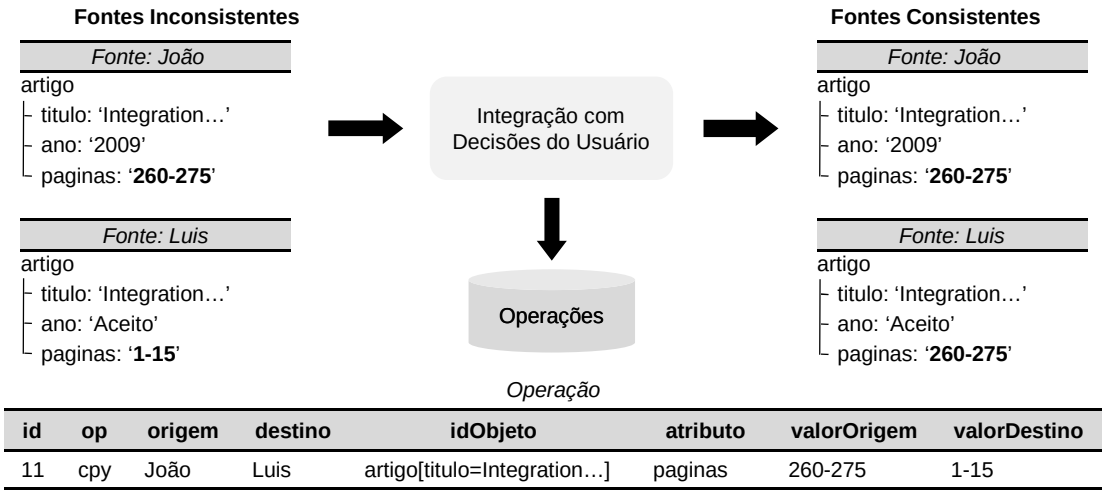
A validação das operações é um processo realizado durante a **reaplicação de operações**. O processo de validação consiste em verificar se as fontes, tanto origem quanto destino, ainda possuem os mesmos dados desde o momento em que as operações foram definidas até o momento no qual elas são reaplicadas automaticamente. Tal processo tem como objetivo garantir que as operações sendo reaplicadas refletem exatamente as decisões tomadas pelo usuário no momento de definição das operações.

A Figura 4.13 ilustra as fontes Luis e João do exemplo introduzido na Seção 4.1.2 em uma versão simplificada, na qual apenas alguns atributos são considerados. Essas fontes possuem valores inconsistentes para o atributo *paginas*, sendo o valor '*260-275*' para João e '*1-15*' para Luis. Devido a essa inconsistência, o usuário interage com o processo de integração para tomar uma decisão, e copia o valor '*260-275*' do atributo *paginas* do objeto com *titulo* igual a '*Integration...*' de João para Luis, deixando as fontes consistentes. Como resultado dessa interação, o modelo MPPI insere automaticamente a operação de cópia com *id* igual a *11* no repositório, guardando dessa forma a decisão do usuário sobre a resolução de tal inconsistência. No processo de integração subsequente, a reaplicação das operações irá tratar automaticamente a inconsistência do exemplo corrente, a qual já foi resolvida pelo usuário.

Uma vez que as fontes são autônomas e podem ser atualizadas entre dois processos de integração, o modelo MPPI garante que a decisão do usuário será reaplicada tal como foi definida e, para isso, as operações são validadas antes de serem reaplicadas. A Seção 4.4.1 descreve diferentes casos de validação plausíveis de ocorrerem em situações reais, enquanto que a Seção 4.4.2 detalha os tipos de validação propostos.

4.4.1 Os Sete Casos de Validação

Existem sete casos possíveis para a combinação dos valores das fontes entre dois processos de integração, o que pode gerar operações válidas e inválidas. O modelo MPPI define operações válidas da seguinte forma.



Operação

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
11	cpy	João	Luis	artigo[titulo=Integration...]	paginas	260-275	1-15

Figura 4.13: Exemplo utilizado para validação: integração das fontes Luis e João.

Definição 9. *Operação válida na origem:* quando a origem da operação continua com o mesmo valor do processo de integração anterior.

Definição 10. *Operação válida no destino:* quando o destino da operação continua com o mesmo valor do processo de integração anterior.

Operações inválidas na origem e no destino representam situações contrárias às Definições 9 e 10, respectivamente.

O **primeiro caso** de validação ocorre quando ambas as fontes consideradas em uma operação continuam com os mesmos dados no processo de integração subsequente (Figura 4.14a). Ou seja, as fontes continuam inconsistentes tal como no processo de integração anterior. Nesse caso, tanto origem quanto destino são válidos. A reaplicação da operação '*11*' torna as fontes consistentes de acordo com a decisão anterior do usuário, como ilustrado na Figura 4.14b.

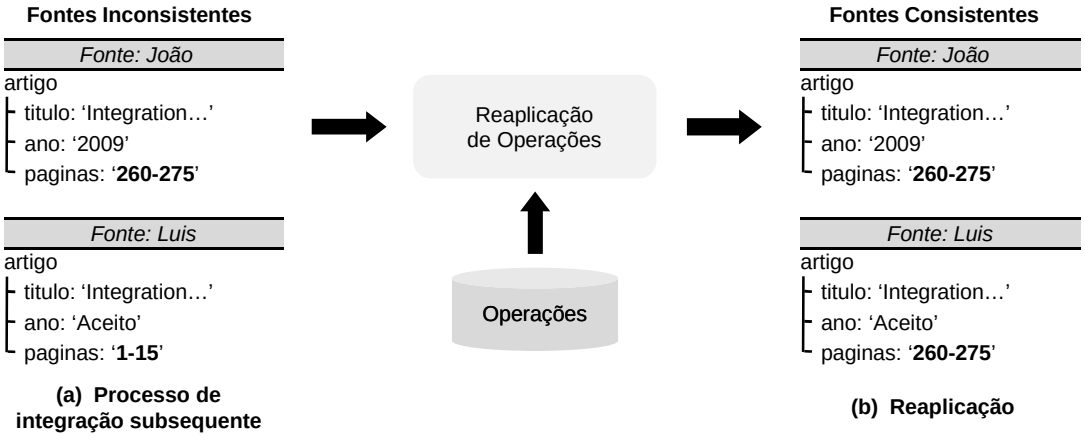


Figura 4.14: Exemplo de validação no primeiro caso.

O **segundo caso** ocorre quando a fonte que atua como origem na operação foi alterada, porém a origem se tornou consistente com o destino (Figura 4.15a). Nesse caso, a operação passa a ser inválida na origem. A reaplicação da operação '*11*' torna as fontes inconsistentes (Figura 4.15b), gerando o problema denominado **anomalia das fontes consistentes**.

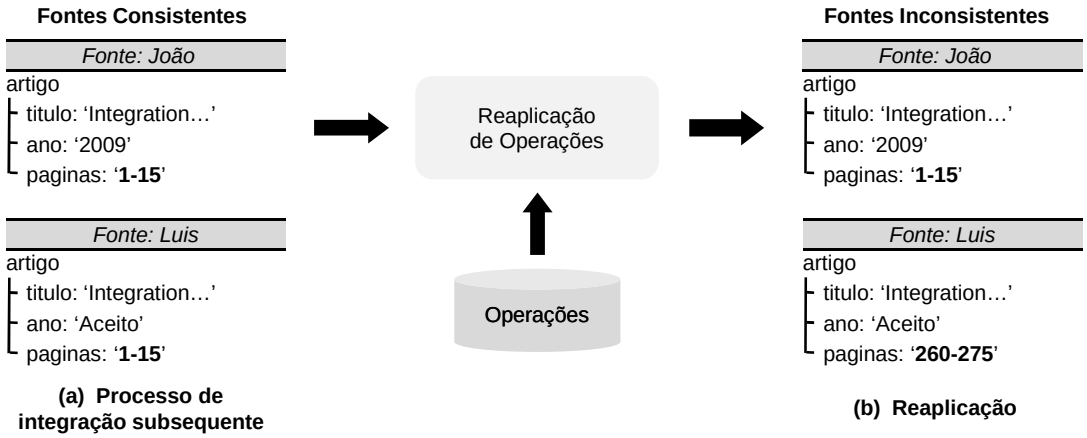


Figura 4.15: Exemplo de validação no segundo caso.

O **terceiro caso** é uma variação do segundo, no qual a origem foi alterada mas continua inconsistente com o destino (Figura 4.16a). Nesse caso, a operação também passa a ser inválida na origem. A reaplicação da operação '*11*' deixa as fontes inconsistentes (Figura 4.16b).

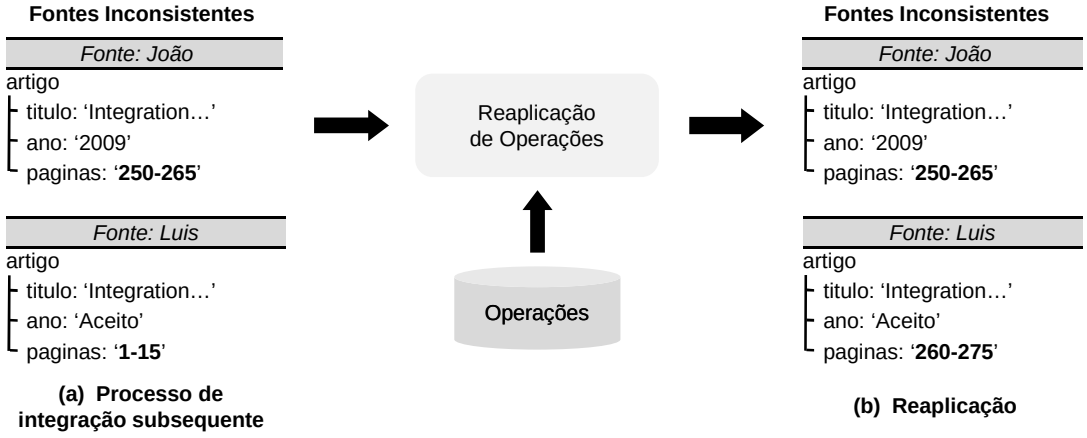


Figura 4.16: Exemplo de validação no terceiro caso.

O **quarto caso** trata operações cujo destino foi alterado de maneira a se tornar consistente com a origem (Figura 4.17a). Nesse caso, a operação passa a ser inválida no destino. Com a reaplicação da operação '*11*', as fontes continuam consistentes.

O **quinto caso** é uma variação do quarto, sendo que o destino foi alterado mas continua inconsistente com a origem (Figura 4.18a). A operação passa a ser inválida no destino.

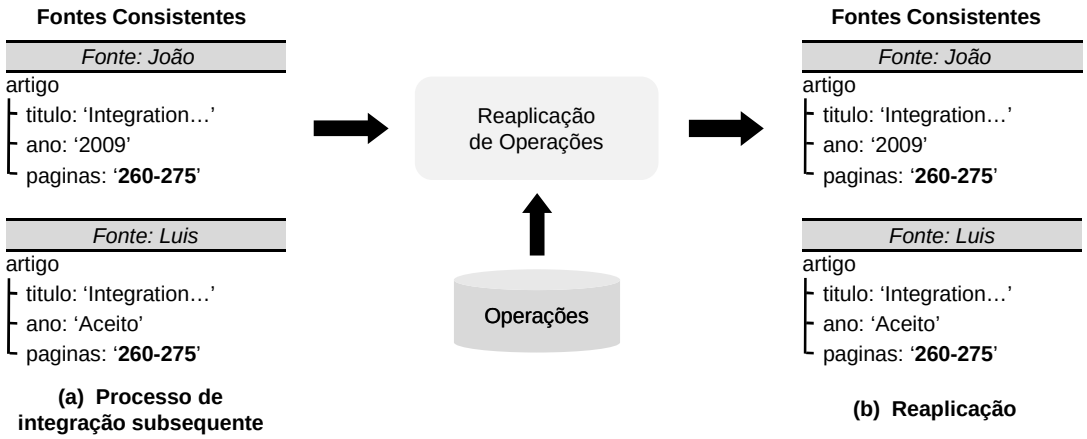


Figura 4.17: Exemplo de validação no quarto caso.

Com a reaplicação da operação '*11*', o destino passa a ter o valor da origem (Figura 4.18b). Apesar das fontes se tornarem consistentes após a reaplicação da operação, a alteração feita no destino pelo usuário é perdida, gerando o problema denominado **anomalia do destino sobrescrito**.

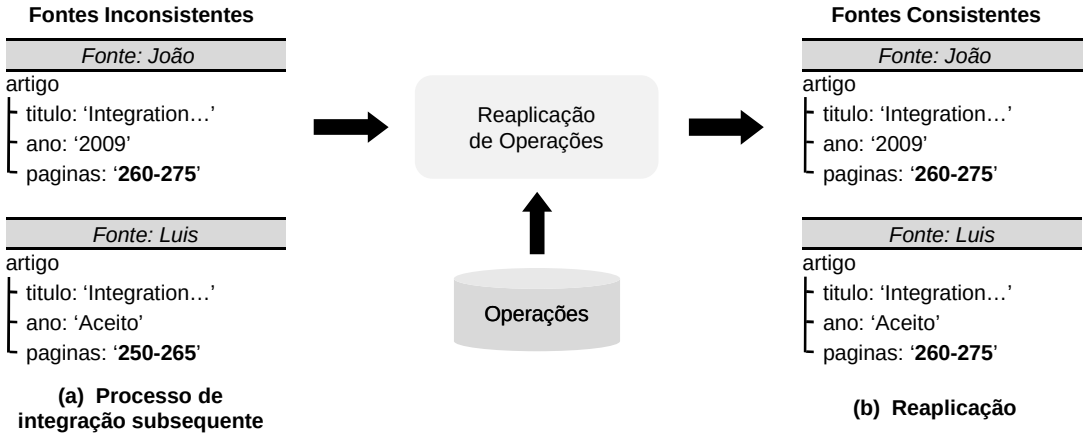


Figura 4.18: Exemplo de validação no quinto caso.

No **sexto caso** tanto a origem quanto o destino foram alterados, sendo que a alteração tornou as fontes consistentes, ou seja, origem e destino foram alterados para o mesmo valor (Figura 4.19a). A operação passa a ser inválida na origem e no destino. Nesse caso, a reaplicação da operação deixa as fontes inconsistentes (Figura 4.19b), gerando tanto a anomalia das fontes consistentes quanto a anomalia do destino sobrescrito.

O **sétimo caso** ocorre quando alterações na origem e no destino ainda deixam as fontes inconsistentes (Figura 4.20a). Analogamente ao sexto caso, a operação passa a ser inválida na origem e no destino. Portanto, a reaplicação da operação deixa as fontes inconsistentes (Figura 4.20b), além de gerar a anomalia do destino sobrescrito.

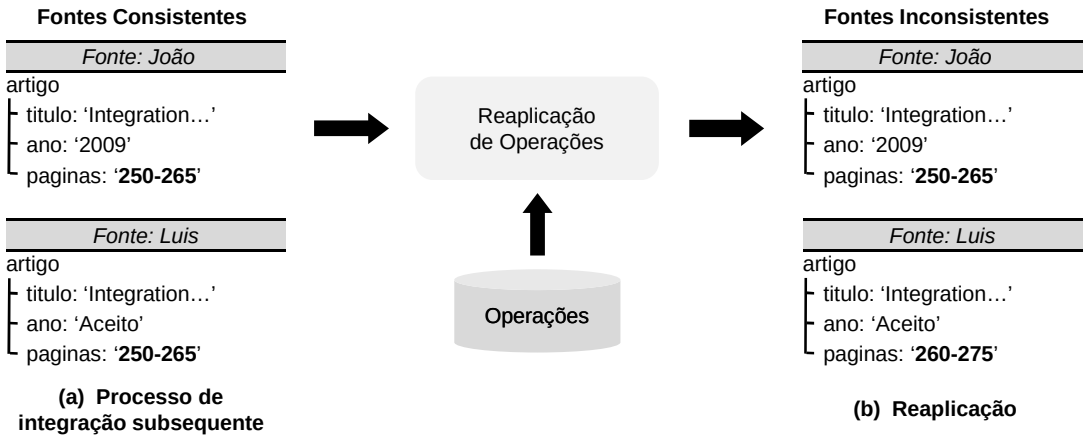


Figura 4.19: Exemplo de validação no sexto caso.

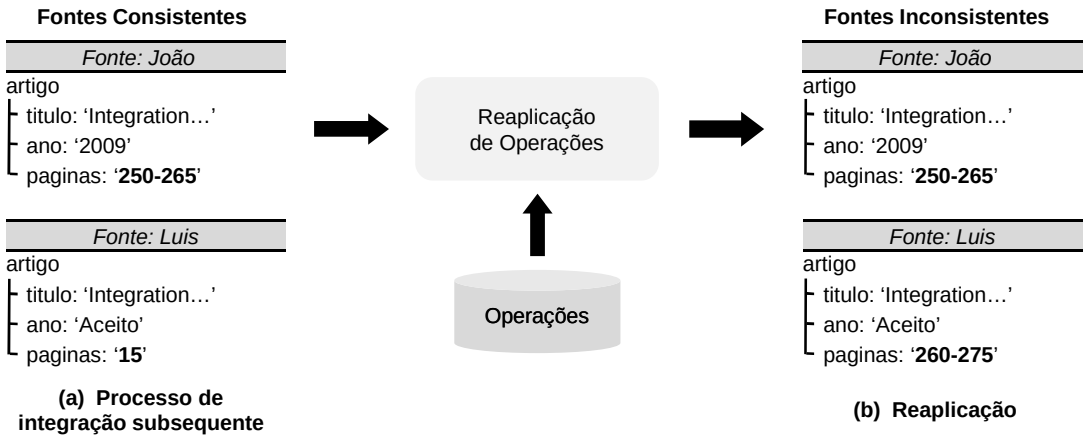


Figura 4.20: Exemplo de validação no sétimo caso.

4.4.2 Tipos de Validação

Por motivos de flexibilidade, o modelo MPPI permite que o usuário escolha qual tipo de validação deve ser considerado durante a reaplicação das operações. O usuário pode optar por um dentre quatro tipos de validação:

- Sem validação;
- Validação da origem;
- Validação do destino; e
- Validação completa;

O tipo **sem validação** não realiza qualquer tipo de validação e está sujeito tanto à anomalia das fontes consistentes quanto à anomalia do destino sobrescrito. A **validação da origem** valida apenas a origem e, dessa forma, evita a anomalia das fontes consistentes. Por outro lado, a **valiação do destino** valida apenas o destino e, dessa forma, evita a

anomalia do destino sobrescrito. Por último, a **validação completa** realiza validação tanto da origem quanto do destino e, dessa forma, evita ambas as anomalias. O usuário deve ter conhecimento dessas anomalias quando da decisão de qual tipo de validação aplicar. A Tabela 4.1 relaciona os tipos de validação e as anomalias.

Tabela 4.1: Tipos de validação e anomalias geradas.

Validação	Anomalia das fontes consistentes	Anomalia do destino sobrescrito
sem validação	x	x
origem	-	x
destino	x	-
completa	-	-

Nos tipos de validação da origem, do destino e completa, quando uma operação é inválida, ela não é reaplicada e é removida do repositório. Operações removidas do repositório podem ser armazenadas como *backup* por motivos de manutenção do histórico dos processos de integração. Entretanto, operações removidas não voltam para o repositório, mesmo que alterações nas fontes as tornem válidas novamente. Essa restrição é imposta para que o repositório seja um conjunto finito e gerenciável de operações, bem como para que a reaplicação das operações acarrete o menor custo adicional possível no processo de integração. Caso operações inválidas pudessem voltar para o repositório quando se tornassem válidas novamente, o repositório apenas cresceria e o custo de validar as operações seria sempre crescente. Portanto, para evitar esse custo adicional, operações inválidas não são verificadas durante o processo de validação e reaplicação.

4.5 Reaplicação de Operações

O modelo MPPI introduz dois métodos para realizar a reaplicação das operações, diferenciando-se pelo tipo de consulta que realizam: tipo rastreamento e tipo filtro. O método VRS é caracterizado por validar as operações antes de reaplicá-las, centrando a reaplicação nas fontes de dados e realizando consultas tipo rastreamento para recuperar as operações que atuam nas fontes. Em contrapartida, o método VRT é caracterizado por validar e reaplicar as operações simultaneamente, centrando a reaplicação nas operações e realizando consultas tipo filtro para recuperar as fontes envolvidas na operação.

O método VRS é introduzido na Seção 4.5.1. Na Seção 4.5.2 são discutidas as motivações para a proposta do método VRT, o qual é descrito na Seção 4.5.3. Na Seção 4.5.4 é descrita uma otimização no método VRT que faz a reordenação das operações do repositório. Os métodos VRS e VRT são apresentados considerando a validação completa, desde que esse tipo de validação não introduz anomalias.

4.5.1 Método VRS

O método VRS (do inglês *Validate and Reapply in Separate*) é caracterizado por primeiro validar as operações para depois reaplicá-las, e tem como base o uso de consultas tipo rastreamento para recuperação das operações. A validação simula a execução do processo de integração no sentido de que apenas operações que seriam aplicadas pelo processo de integração são reaplicadas pelo VRS. Portanto, a validação garante que a reaplicação não gere resultados incorretos.

A Figura 4.21 ilustra o fluxo das operações no método VRS, o qual é constituido por duas etapas. Na primeira etapa, todas as operações são validadas na origem. Essa primeira etapa filtra operações inválidas na origem e passa para a segunda etapa apenas operações válidas. A segunda etapa consiste em validar as operações no destino e, caso a operação continue válida, reaplicá-las.

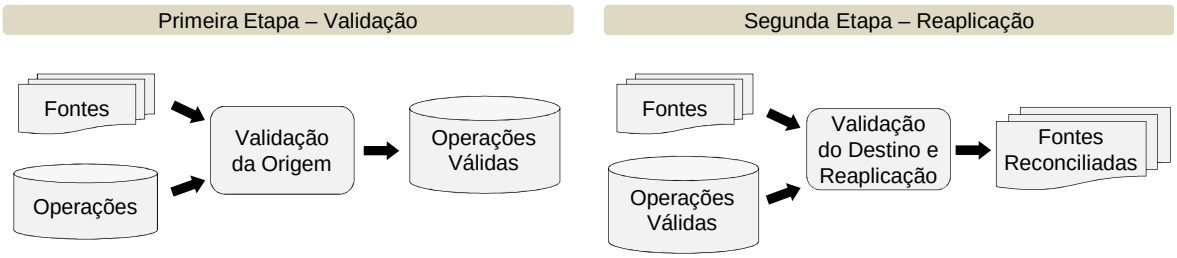


Figura 4.21: Fluxo das operações no método VRS.

O algoritmo do método VRS (Algoritmo 7) recebe como parâmetro um repositório R e um conjunto *Fontes* de fontes de dados, e gera como saída o conjunto *Fontes* atualizado de acordo com a reaplicação das operações válidas. Esse algoritmo utiliza consultas tipo rastreamento para reaplicação das operações, fazendo duas iterações sobre o conjunto *Fontes*. Na primeira iteração (linhas 1 a 8), para cada *FonteOrigem* do conjunto *Fontes*, é recuperado o conjunto *Ops* de operações que têm como origem a *FonteOrigem* (linha 2). Em seguida, para cada operação d do conjunto *Ops*, é verificado se o *valorOrigem* da operação d continua com o mesmo valor na *FonteOrigem*, em relação a quando a operação d foi definida (linhas 3 a 7). Se a *FonteOrigem* possui um valor diferente, a operação d é removida do repositório R (linha 5). Na segunda iteração (linhas 9 a 18), para cada *FonteDestino* do conjunto *Fontes*, é recuperado o conjunto *Ops* de operações que têm como destino a *FonteDestino* (linha 10). Em seguida, para cada operação d do conjunto *Ops*, é verificado se o *valorDestino* de d continua com o mesmo valor na *FonteDestino*, em relação a quando a operação d foi definida (linhas 11 a 17). Se a *FonteDestino* possui um valor diferente, a operação d é removida do repositório R (linha 13). Caso contrário, a operação d é reaplicada na *FonteDestino* (linha 15).

Algoritmo 7 VRS (R, Fontes)

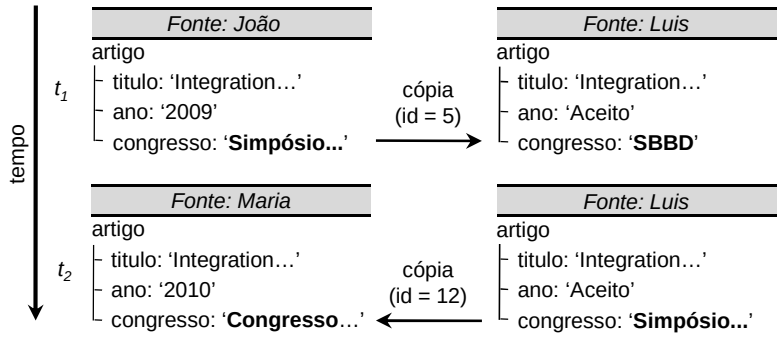
Entrada: R /* repositório */
$Fontes$ /* conjunto de fontes de dados */
Saída: R /* repositório atualizado */
$Fontes$ /* conjunto de fontes atualizadas */

```
1: para toda fonte FonteOrigem em Fontes faça
2:   Ops ← {op ∈ R | op.origem = FonteOrigem}
3:   para toda operacao d em Ops faça
4:     se d.valorOrigem <> valor da FonteOrigem então
5:       R ← R − {d}
6:     fim se
7:   fim para
8: fim para
9: para toda fonte FonteDestino em Fontes faça
10:  Ops ← {op ∈ R | op.destino = FonteDestino}
11:  para toda operacao d em Ops faça
12:    se d.valorDestino <> valor da FonteDestino então
13:      R ← R − {d}
14:    senão
15:      Reaplicar operacao d na FonteDestino /* realizado pela aplicação */
16:    fim se
17:  fim para
18: fim para
```

4.5.2 Motivação para Proposta de um novo Método

O método VRS pode não gerar o resultado correto caso hajam operações transitivas e operações que alteram a chave no repositório de operações. A seguir é ilustrado um exemplo no qual o resultado gerado é incorreto devido às operações transitivas. Considere o processo de integração p_1 ilustrado na Figura 4.22, no qual são geradas as operações transitivas *5* e *12*, ambas operações de cópia. Na primeira operação, no tempo t_1, o usuário copia o valor do atributo *congresso* de João para Luis. Já na segunda operação, no tempo t_2, o usuário copia o atributo *congresso* de Luis para Maria ($t_1 < t_2$). A Figura 4.23 ilustra as fontes após p_1.

No momento da reaplicação das operações de p_1 pelo método VRS (Figura 4.24), quando validando as operações na origem, a operação *5* realizada no tempo t_1 é válida. Entretanto, a operação *12* realizada no tempo t_2 é inválida pois a fonte Luis possui um valor diferente daquele da operação. Sendo assim, apenas a operação do tempo t_1 é passada para a próxima etapa, sendo válida também no destino e, portanto, reaplicada. Como pode ser observado na Figura 4.25, o resultado da reaplicação das operações é diferente daquele gerado pelo usuário, apesar das fontes não terem sido alteradas.



id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio...	SBBD
...							
12	cp	Luis	Maria	artigo[titulo=Integration…]	congresso	Simpósio...	Congresso…

Figura 4.22: Transitividade no processo de integração p_1.

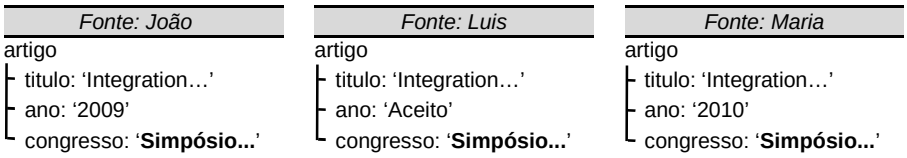
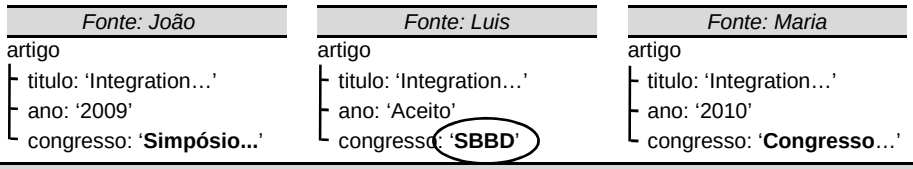


Figura 4.23: Resultado do processo de integração p_1.

Ademais, a operação *12* é removida do repositório uma vez que foi, erroneamente, considerada inválida na origem.



id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio...	SBBD
...							
12	cp	Luis	Maria	artigo[titulo=Integration…]	congresso	Simpósio...	Congresso…

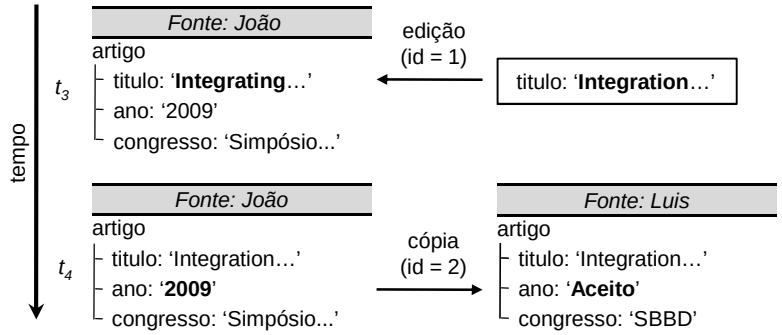
Figura 4.24: Validação das operações na reaplicação das operações de p_1.

Operações que alteram a chave geram um problema similar, ilustrado pelas operações *1* e *2* do processo de integração p_2 (Figura 4.26). Considere que o repositório de operações está vazio antes de p_2, e que a chave do objeto artigo é *titulo*. No tempo t_3, o usuário edita o atributo *titulo* da fonte João de '*Integrating...*' para '*Integration...*' e, no tempo

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
...							
5	cp	João	Luis	artigo[titulo=Integration...]	congresso	Simpósio...	Simpósio…
...							
~~12~~	~~cp~~	~~Luis~~	~~Maria~~	~~artigo[titulo=Integration...]~~	~~congresso~~	~~Simpósio...~~	~~Congresso...~~

Figura 4.25: Resultado da reaplicação das operações de p_1.

t_4, o usuário copia o atributo *ano* de João para Luis ($t_3 < t_4$). A Figura 4.27 ilustra o resultado de p_2.



id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	João	artigo[titulo=Integrating...]	titulo	Integration...	Integrating...
2	cp	João	Luis	artigo[titulo=Integration...]	ano	2009	Aceito
...							

Figura 4.26: Operação que altera a chave no processo de integração p_2.

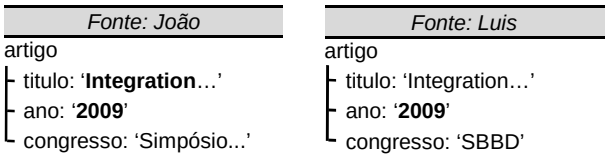
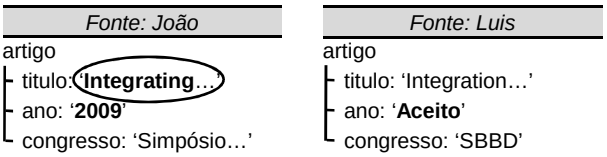


Figura 4.27: Resultado do processo de integração p_2.

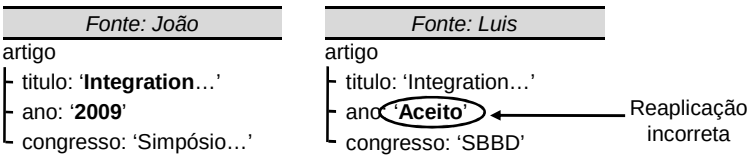
No momento da reaplicação das operações de p_2 pelo método VRS (Figura 4.28), quando validando as operações na origem, a operação de edição do tempo t_3 é válida. Em contrapartida, a operação de cópia do tempo t_4 é inválida pois o objeto destino da operação não existe, uma vez que não é possível encontrar o objeto artigo com chave *titulo* = '*Integration...*' na fonte João. Portanto, apenas a operação *1* é passada

para a segunda etapa, sendo válida também no destino e, portanto, reaplicada. Como pode ser observado na Figura 4.29, o resultado da reaplicação das operações é diferente daquele gerado pelo usuário para o atributo *ano* da fonte Luis, apesar das fontes não terem sido alteradas. Ademais, a operação *2* é removida do repositório uma vez que foi, erroneamente, considerada inválida na origem.



id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	João	artigo[titulo=Integrating...]	titulo	Integration...	Integrating...
2	cp	João	Luis	artigo[titulo=Integration...]	ano	2009	Aceito
...							

Figura 4.28: Validação das operações na reaplicação das operações de p_2.



id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	João	artigo[titulo=Integrating...]	titulo	Integration...	Integrating...
~~2~~	~~cp~~	~~João~~	~~Luis~~	~~artigo[titulo=Integration...]~~	~~ano~~	~~2009~~	~~Aceito~~
...							

Figura 4.29: Resultado da reaplicação das operações de p_2.

Portanto, o método VRS possui duas limitações: (i) resultado incorreto na ocorrência de operações transitivas; e (ii) resultado incorreto na ocorrência de operações que alteram a chave. Para resolver tais limitações, é proposto um segundo método de reaplicação, o qual é descrito na Seção 4.5.3.

4.5.3 Método VRT

Nessa seção é proposto o método validação e reaplicação em conjunto (VRT, do inglês *Validate and Reapply in Tandem*), o qual tem como base o uso de consultas tipo filtro para recuperação das operações. A motivação desse método é realizar os tratamentos dos problemas encontrados no método VRS. A Figura 4.30 ilustra o fluxo de operações no método VRT. Dada uma operação do repositório, a fonte origem é recuperada no conjunto de fontes, para que a operação possa ser validada na origem. Caso a operação seja válida na origem, a fonte destino é recuperada no conjunto de fontes e a operação é validada

no destino. Caso a operação continue válida no destino, a operação é reaplicada e as fontes tornam-se consistentes. Esse processo é executado uma vez para cada operação no repositório.

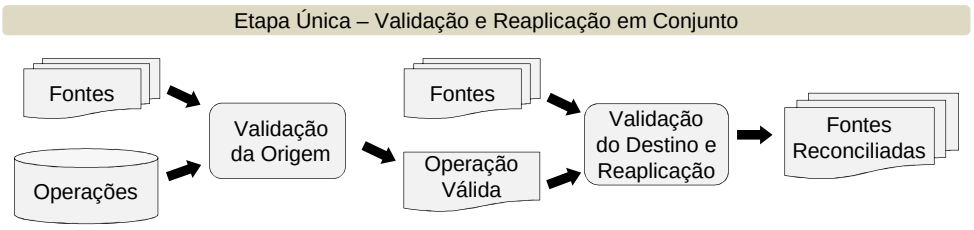


Figura 4.30: Arquitetura do método VRT.

O algoritmo do método VRT (Algoritmo 8) recebe como parâmetros um repositório R e um conjunto *Fontes* de fontes de dados, e gera como saída o repositório R e o conjunto *Fontes* atualizados. Esse algoritmo realiza consulta tipo filtro na procedência para reaplicação das operações, necessitando de uma única iteração sobre as operações do repositório R (linhas 1 a 13). Para cada operação d no repositório, recupera-se a fonte que atua como origem da operação (linha 2). Se o *valorOrigem* da operação d possui um valor diferente na *FonteOrigem* em relação ao momento em que foi definida, a operação d é considerada inválida na origem e por isso é removida do repositório (linhas 3 e 4). Caso contrário, recupera-se a fonte que atua como destino da operação (linha 6). Se o *valorDestino* da operação d possui um valor diferente na *FonteDestino* em relação ao momento em que foi definida, a operação d é considerada inválida no destino e por isso é removida do repositório (linhas 7 e 8). Caso contrário, a operação d é reaplicada na *FonteDestino* (linha 10).

No método VRT, as operações são reaplicadas em ordem temporal. Dessa forma, garante-se que operações transitivas sejam reaplicadas na mesma ordem em que foram definidas. Quando uma operação y, transitiva a x, for considerada para reaplicação, o resultado da operação x já estará reaplicado e, portanto, a operação y será reaplicada. O caso é similar para operações que alteram a chave. Portanto, operações transitivas e operações que alteram a chave não são um problema para o método VRT, sendo que ao final do processo de reaplicação todas as operações válidas estão reaplicadas.

4.5.4 Reordenação Segura

No método VRT, a lista de operações do repositório é percorrida sequencialmente e, entre duas operações, pode ser necessário alternar o carregamento de diferentes fontes para as fontes origem e destino. Para minimizar a quantidade de carregamento das fontes durante a reaplicação, nessa seção é feita a proposta de otimização que faz a reordenação das operações do repositório, chamada de **reordenação segura**. Para ser segura, a

Algoritmo 8 VRT (R, Fontes)

Entrada: R /* repositório */
$Fontes$ /* conjunto de fontes de dados */
Saída: R /* repositório atualizado */
$Fontes$ /* conjunto de fontes atualizadas */

```
1: para toda operacao d em R faça
2:   FonteOrigem ← fonte ∈ Fontes | fonte = d.origem
3:   se d.valorOrigem <> valor da FonteOrigem então
4:     R ← R − {d}
5:   senão
6:     FonteDestino ← fonte ∈ Fontes | fonte = d.destino
7:     se d.valorDestino <> valor da FonteDestino então
8:       R ← R − {d}
9:     senão
10:      Reaplicar operacao d na FonteDestino /* realizado pela aplicação */
11:    fim se
12:  fim se
13: fim para
```

reordenação das operações não deve afetar o resultado gerado pela reaplicação, ou seja, o resultado da reaplicação das operações reordenadas deve ser o mesmo da reaplicação das operações em ordem temporal. Para tanto, é definido o conceito de reordenação segura.

Definição 11. *Reordenação Segura:* Uma reordenação segura é uma reordenação das operações do repositório que não altera a ordem de operações transitivas, de operações que alteram a chave e nem de operações de sobreposição.

A Figura 4.32 ilustra o repositório da Figura 4.31 ordenado de maneira segura. Utilizando a reordenação segura, a operação *5* pode ser reordenada. Portanto, considerando-se o repositório da Figura 4.32, a fonte Luis é carregada três vezes pelo método VRT, enquanto que considerando-se o repositório da Figura 4.31, a fonte Luis é carregada quatro vezes por esse método.

O algoritmo da reordenação segura (Algoritmo 9) recebe como parâmetro um repositório R a ser ordenado e gera como saída o repositório R reordenado. A primeira iteração, linhas 1 a 26, percorre todas as operações $R[i]$ ($i \geq 1$). A segunda iteração procura por operações $R[j]$ ($j < i$) que possuem a mesma fonte origem e destino que a operação i (linhas 3 a 25). O algoritmo primeiro reordena as operações pela origem (linhas 4 a 7), mantendo o índice $j + 1$ como candidato ($IndCand$) para reordenação de R[i] (linha 5) e verificando se a reordenação mantém a ordem de operações transitivas, que alteram a chave e de sobreposição (linhas 8 a 11). Em seguida, o algoritmo reordena pelo destino (linhas 12 a 24). Se for possível, a operação $R[i]$ é reordenada pela origem e destino (linhas 13 a 18), senão é reordenada apenas pela origem (linhas 19 a 23). Caso a reordenação pela origem também não seja possível, a operação $R[i]$ não é reordenada.

Operação

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	João	artigo[titulo=Integrating...]	titulo	Integration...	Integrating...
2	cp	João	**Luis**	artigo[titulo=Integration...]	ano	2009	Aceito
3	ed	null	Maria	artigo[titulo=Integration...] / autor[nome=Joaquim]	ordemCitacao	null	4
4	rm	null	Maria	artigo[titulo=Integration...] / autor[nome=Joaquim]	null	null	null
5	cp	João	**Luis**	artigo[titulo=Integration...]	congresso	Simpósio...	SBBD
6	cp	João	Maria	artigo[titulo=Integration…]	paginas	260-275	1-15
7	in	**Luis**	João	artigo[titulo=Integration…] / autor[nome=Maria]	null	null	null
8	cp	**Luis**	João	artigo[titulo=Integration…] / autor[nome=Maria]	ordemCitacao	3	null
9	cp	Maria	João	artigo[titulo=Integration…] / autor[nome=João]	ordemCitacao	2	1
10	cp	Maria	João	artigo[titulo=Integration…] / autor[nome=Luis]	ordemCitacao	1	2
11	cp	**Luis**	Maria	artigo[titulo=Integration...]	ano	2009	2010
12	cp	**Luis**	Maria	artigo[titulo=Integration…]	congresso	Simpósio...	Congresso…

Figura 4.31: Exemplo de um repositório de operações não ordenado, fonte Luis carregada quatro vezes.

Operação

id	op	origem	destino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	João	artigo[titulo=Integrating...]	titulo	Integration...	Integrating...
2	cp	João	**Luis**	artigo[titulo=Integration...]	ano	2009	Aceito
5	***cp***	***João***	***Luis***	***artigo[titulo=Integration...]***	***congresso***	***Simpósio...***	***SBBD***
3	ed	null	Maria	artigo[titulo=Integration...] / autor[nome=Joaquim]	ordemCitacao	null	4
4	rm	null	Maria	artigo[titulo=Integration...] / autor[nome=Joaquim]	null	null	null
6	cp	João	Maria	artigo[titulo=Integration…]	paginas	260-275	1-15
7	in	**Luis**	João	artigo[titulo=Integration…] / autor[nome=Maria]	null	null	null
8	cp	**Luis**	João	artigo [titulo=Integration…] / autor[nome=Maria]	ordemCitacao	3	null
9	cp	Maria	João	artigo [titulo=Integration…] / autor[nome=João]	ordemCitacao	2	1
10	cp	Maria	João	artigo [titulo=Integration…] / autor[nome=Luis]	ordemCitacao	1	2
11	cp	**Luis**	Maria	artigo [titulo=Integration...]	ano	2009	2010
12	cp	**Luis**	Maria	artigo[titulo=Integration…]	congresso	Simpósio...	Congresso…

Figura 4.32: Exemplo de um repositório de operações reordenado, fonte Luis carregada três vezes.

4.6 Considerações Finais

Neste capítulo foi detalhado o modelo MPPI, um modelo de procedência para subsidiar processos de integração. Inicialmente, foi apresentada uma visão geral do modelo proposto, destacando suas quatro características. Em seguida, cada uma dessas car-

acterísticas foi detalhada. A primeira característica introduziu o fato de que, no modelo MPPI, as decisões que o usuário toma para integrar dados heterogêneos são transformadas em **operações** de *cópia*, *edição*, *inserção* e *remoção* e armazenadas em um repositório de operações. Além disso, foram definidos os conceitos de operação transitiva e de operação que altera a chave. A segunda característica enfocou o **tratamento de operações de sobreposição** por meio da proposta das políticas *blind*, *restrict*, *undo* e *redo*. A terceira característica destacou a necessidade de se realizar a **validação** das operações e definiu as anomalias das fontes consistentes e do destino sobrescrito. Também foram introduzidos os seguintes tipos de validação: sem validação, validação da origem, validação do destino e validação completa. Por fim, o quarto diferencial apresentou dois métodos para realizar a **reaplicação** das operações, o método VRS e o método VRT, além de introduzir uma otimização no método VRT que faz a reordenação segura das operações do repositório.

A Tabela 4.2 mostra a Tabela 3.1 com uma coluna adicional, na qual são comparadas as características do modelo MPPI com as características dos trabalhos correlatos resumidos no Capítulo 3. Como destacado na Seção 3.5, os sistemas CHIME, CPDB, Trio e ELIT não têm como objetivo oferecer suporte ao tratamento de todas as decisões tomadas pelo usuário em processos de integração anteriores, de forma que essas decisões possam ser reaplicadas automaticamente em processos de integração subsequentes, que é o objetivo do modelo MPPI. Portanto, esses sistemas não oferecem funcionalidades para tratamento das operações de sobreposição, validação e reaplicação das operações.

Alguns destaques sobre a Tabela 4.2 são descritos a seguir. O modelo MPPI, assim como os sistemas CHIME e CPDB, é um modelo baseado em operações. As operações de *cópia*, *edição*, *inserção* e *remoção* providas pelo modelo MPPI são adaptadas para o contexto de reaplicação de operações. Elas representam operações elementares, desde que permitem que novas operações possam ser definidas em termos delas. De maneira geral, essas operações oferecem funcionalidades similares às operações de cópia, inserção e remoção do sistema CPDB e de cópia e inserção do sistema CHIME. A operação de resolução de entidades desse último sistema pode ser representada no modelo MPPI, por exemplo, por várias operações de *cópia* e uma operação de *remoção*. Em contrapartida, desde que o enfoque desta dissertação quanto à integração de dados é a investigação da procedência na resolução de conflitos em nível de instância, o modelo MPPI não provê operações relacionadas à integração de esquemas, como é o caso da operação de resolução de atributos do sistema CHIME.

Com relação ao tratamento das operações de sobreposição, foi adicionada uma linha na Tabela 4.2 referente ao aspecto *como coletar*, representando a interação existente entre o usuário e a procedência dos dados durante a coleta da procedência. Na coleta ativa, o usuário pode usar a procedência dos dados para realizar alguma decisão. Por exemplo, o usuário pode ser avisado de que uma operação sendo coletada irá sobrepor uma decisão tomada previamente por ele. O usuário pode decidir, por exemplo, em não

Tabela 4.2: Comparação entre os trabalhos correlatos e o modelo MPPI.

Aspectos		CHIME	CPDB	Trio	ELIT	Nosso
Objetivo		*Rastrear resultados de integração*	*Otimizar armazenamento*	*Rastrear procedência em SGBDs*	*Rastrear resultados de integração*	*Reaplicar operações* (VRS e VRT)
O que coletar?	tipo	*why-* e *where-provenance*	*why-* e *where-provenance*	*where-provenance*	*where-provenance*	*why-* e *where-provenance*
	granularidade	atributo	atributo / objeto	tupla	tupla	atributo
	operações	cópia, inserção, resolução de atributos, resolução de entidades, confirmação	cópia, inserção, remoção	não trata	não trata	cópia, edição, inserção, remoção
Como coletar?	tipo	automática	automática	automática	automática	automática
	abordagem	*eager*	*eager*	*eager*	*eager*	*eager*
	interação	passiva	passiva	passiva	passiva	passiva (*blind*), ativa (*restrict*, *undo* e *redo*)
Como armazenar?	tipo	*naïve*	*naïve*, *transactional, hierarchical, transactional-hierarchical*	*hierarchical*	*naïve*	*naïve* (validação)
	local	separada	separada	junto	separada	separada
Como consultar?	tipo	rastreamento, filtro	rastreamento	rastreamento, filtro	rastreamento	rastreamento, filtro

completar a nova operação, não gerando a operação sobreposta. A coleta ativa é possível de ser realizada pelas políticas *restrict*, *undo* e *redo*. A coleta passiva ilustra a situação contrária, e é provida, no modelo MPPI, pela política *blind*. A interação nos demais trabalhos correlatos foi considerada como passiva.

Quanto ao aspecto *como armazenar*, o modelo MPPI utiliza o tipo *naïve* porque a validação é realizada em nível de atributo, enquanto que as demais otimizações propostas pelo sistema CPDB armazenam as operações em nível de objeto. Ou seja, a validação usa dados de procedência mais detalhados. Em contrapartida, com o passar do tempo as fontes provedoras dos dados inconsistentes tendem a ser atualizadas, o que acarreta a invalidação de operações e a remoção dessas operações inválidas do repositório pelo modelo MPPI.

Além da comparação das características do modelo MPPI com os trabalhos correlatos destacados na Tabela 4.2, o modelo proposto também foi validado por meio de testes de desempenho, os quais são apresentados e discutidos no Capítulo 5.

Algoritmo 9 ReordenacaoSegura (R)

Entrada: R /* repositório */
Saída: R /* repositório reordenado */

```
1:  para i ← 1 até |R| faça
2:    reordenando ← "origem"
3:    para j ← (i - 1) até 0 decremento 1 faça
4:      se reordenando = "origem" então
5:        se R[i].origem = R[j].origem então
6:          IndCand ← j + 1 /* índice candidato para a reordenação */
7:          reordenando ← "destino"
8:        senão se (R[i].origem   = R[j].destino e
                    R[i].idObjeto = R[j].idObjeto e
                    R[i].atributo = R[j].atributo) ou
                   (R[i].destino  = R[j].destino e
                    R[i].idObjeto = R[j].idObjeto) ou
                   (R[i].destino  = R[j].origem e
                    R[i].idObjeto = R[j].idObjeto) então
9:          /* achou operação transitiva, que altera a chave ou de sobreposição */
10:         parar laço interno
11:       fim se
12:     fim se
13:     se reordenando = "destino" então
14:       se R[i].origem = R[j].origem então
15:         se R[i].destino = R[j].destino então
16:           OperacaoReordenada ← R[i]
17:           Incrementar em 1 o indice das operacoes com indice > j
18:           R[j + 1] ← OperacaoReordenada
19:         fim se
20:       senão
21:         OperacaoReordenada ← R[i]
22:         Incrementar em 1 o indice das operacoes com indice ≥ IndCand
23:         R[IndCand] ← OperacaoReordenada
24:       fim se
25:     fim se
26:   fim para
27: fim para
```

Capítulo

5

Testes de Desempenho

O modelo MPPI foi validado por meio de testes de desempenho utilizando 120 currículos Lattes de docentes do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, os quais foram usados como fontes a ser integradas. Os 120 currículos totalizaram 41,8 MB, tendo o maior currículo 1,9 MB e o menor 10,6 KB. Os testes foram realizados com base nos artigos científicos publicados em periódicos desses currículos, sendo que os 120 currículos totalizaram 1782 artigos científicos (considerando repetições), tendo o maior currículo 72 artigos e o menor 1 artigo.

As funcionalidades do modelo MPPI foram implementadas em C++ utilizando o conjunto de bibliotecas do Qt versão 4.5.5 (http://qt.nokia.com/) e compiladas no GNU g++ versão 4.3.3. Os experimentos foram conduzidos em uma máquina executando Linux (32-bits) com um processador Intel Core i7 2,67 GHz, 12 GB de memória primária e 2 TB de memória secundária. A quantidade máxima de memória primária utilizada foi de 1 GB. Nos testes de desempenho, a medida coletada foi o tempo de execução em segundos.

O modelo MPPI é baseado em quatro características, as quais foram testadas da seguinte maneira. A primeira característica, **procedência baseada em operações**, foi investigada em todos os testes realizados, por meio da manipulação das operações de *cópia*, *edição*, *inserção* e *remoção*. Com relação ao **tratamento de operações de sobreposição**, as quatro políticas propostas *blind*, *restrict*, *undo* e *redo* foram investigadas vislumbrando analisar o impacto por elas causado durante a coleta. Os resultados obtidos são discutidos na Seção 5.1. Quanto à **validação das operações**, todos os testes de reaplicação foram feitos utilizando a validação completa, que é a forma de validação mais apropriada a ser usada, uma vez que não sofre das anomalias definidas na Seção 4.4.1. A quarta característica, **reaplicação das operações**, foi investigada considerando-se duas configurações. Enquanto a Seção 5.2 apresenta os

resultados obtidos no emprego do método proposto VRT na reaplicação de operações, a Seção 5.3 analisa o impacto da reordenação segura do repositório na reaplicação das operações. As considerações finais são descritas na Seção 5.4.

5.1 Tratamento de Operações de Sobreposição

Nessa seção são apresentados os testes relativos ao tratamento de operações de sobreposição utilizando as políticas *blind*, *restrict*, *undo* e *redo*, as quais foram introduzidas na Seção 4.3. O objetivo da seção é analisar o impacto causado pelo tratamento dessas operações, isto é, o custo adicional introduzido pelo tratamento das operações de sobreposição durante a coleta. Enquanto a política *blind* não realiza nenhum tratamento para operações de sobreposição, as demais políticas tratam essas operações e por isso introduzem custos adicionais. Assim, a política *blind* é utilizada como medida base na análise do custo adicional introduzido pelas demais políticas. Ou seja, os resultados obtidos pelas políticas *restrict*, *undo* e *redo* são comparados com os resultados obtidos pela política *blind*.

Para testar o impacto das políticas é preciso analisar duas variáveis. A primeira delas é o número de operações armazenadas no repositório, uma vez que é necessário buscar, dentre estas, as operações que são transitivas antes de aplicar uma determinada política. A segunda consiste no número de operações transitivas recuperadas, as quais serão tratadas pela política em questão. Assim, na Seção 5.1.1 é investigado o impacto do número de operações armazenadas no repositório, considerando o número de operações transitivas constante, enquanto que na Seção 5.1.2 é investigado o impacto do número de operações transitivas, considerando o número de operações armazenadas no repositório constante. A operação de *edição* foi utilizada como operação de sobreposição e operação sobreposta. Para as operações transitivas, foram utilizadas operações de *cópia*.

5.1.1 Escalabilidade do Número de Operações no Repositório

Os gráficos da Figura 5.1 têm como objetivo mostrar o impacto do número de operações existentes no repositório ao tratar uma operação de sobreposição. O gráfico da Figura 5.1a ilustra o caso em que todas as operações transitivas são diretas, enquanto que o gráfico da Figura 5.1b ilustra o caso em que todas as operações transitivas são indiretas.

Foram utilizados 30 documentos como fontes de dados para coletar uma operação de sobreposição que desencadeou o tratamento de 30 operações transitivas. O teste foi realizado variando-se o número de operações no repositório, começando com 500 operações e terminando com 10.000 operações, com um incremento de 500 operações entre um teste e outro. As operações transitivas foram espalhadas uniformemente entre as operações do repositório.

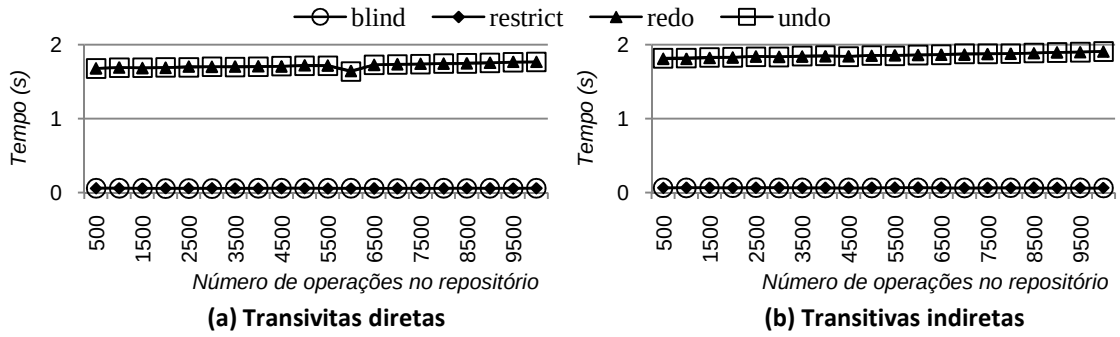


Figura 5.1: Tratamento de operações de sobreposição - Escalabilidade do número de operações no repositório.

Para todas as políticas *blind*, *restrict*, *redo* e *undo*, conforme aumentou o número de operações armazenadas, o tempo permaneceu constante. Portanto, o desempenho dessas políticas não foi alterado pelo aumento no número de operações no repositório.

Os tempos gastos pelas políticas *blind* e *restrict* foram praticamente os mesmos, sendo a diferença média de desempenho de 0,13% para operações transitivas diretas e de 0,03% para operações transitivas indiretas. Essa pequena diferença é devido à organização das operações no repositório. O algoritmo implementado para a política *restrict* termina assim que encontra uma operação sobreposta (Algoritmo 1). Como as operações de sobreposição e suas operações transitivas estão espalhadas uniformemente pelo repositório, a operação sobreposta encontra-se logo no início do repositório, e portanto, o algoritmo *restrict* precisa percorrer poucas operações até encontrar a operação sobreposta.

As políticas *redo* e *undo* também apresentaram resultados de desempenho muito próximos. Esse comportamento é devido ao fato dessas políticas possuírem funcionalidades similares, embora produzam resultados finais diferentes. Os algoritmos dessas políticas requerem a busca por operações transitivas no repositório e, na sequência, a alteração de dados nas fontes (Algoritmos 3 e 6).

Comparando-se as políticas *redo* e *undo* com a política *blind*, os resultados mostraram que a política *blind* apresentou uma redução média de 96,59% para operações transitivas diretas e de 96,53% para operações transitivas indiretas. Essa diferença é decorrente da forma na qual as operações transitivas são tratadas pelas diferentes políticas, evidenciando que é mais caro tratar as operações transitivas do que buscá-las no repositório. Entretanto, apesar da diferença ser percentualmente alta, o tempo gasto pelas políticas *redo* e *undo* foi menor que 2 segundos, o que garante a aplicabilidade das quatro políticas em situações reais. A escolha da política a ser usada depende, portanto, das necessidades da aplicação.

5.1.2 Escalabilidade do Número de Operações Transitivas

Os gráficos da Figura 5.2 têm como objetivo mostrar o impacto do número de operações transitivas existentes no repositório ao tratar uma operação de sobreposição. O gráfico da Figura 5.2a ilustra o caso em que todas as operações transitivas são diretas, enquanto que o gráfico 5.2b ilustra o caso em que todas as operações transitivas são indiretas.

Foram utilizados 30 documentos como fontes de dados para coletar uma operação de sobreposição em um repositório com 1.000 operações. Os testes foram realizados variando-se o número de operações transitivas à operação sobreposta, começando com nenhuma operação transitiva e terminando com 30, com um incremento de 5 operações transitivas entre um teste e outro. As operações transitivas foram espalhadas uniformemente entre as operações do repositório.

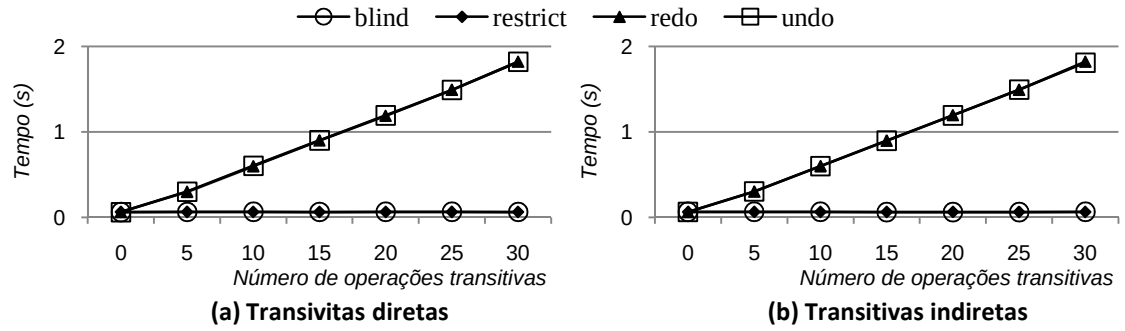


Figura 5.2: Tratamento de operações de sobreposição - Escalabilidade do número de operações transitivas.

Para as políticas *blind* e *restrict*, conforme aumenta o número de operações transitivas, o tempo permanece constante. Os tempos gastos por essas políticas são praticamente os mesmos, sendo a diferença média entre eles de 0,07% para operações transitivas diretas e de 0,72% para operações transitivas indiretas. Portanto, o desempenho dessas políticas não foi alterado pelo aumento do número de operações transitivas. A mesma explicação da Seção 5.1.1 sobre a similaridade dos tempos das políticas *blind* e *restrict* se aplica a esse caso.

As políticas *redo* e *undo* tiveram um comportamento bastante similar entre si, devido às explicações já discutidas na Seção 5.1.1. Para essas políticas, o tempo requerido para coleta aumentou linearmente com o aumento do número de operações transitivas. O aumento foi linear porque o tratamento de uma operação transitiva não influencia o tratamento de outras operações transitivas, ou seja, o tempo de tratamento de uma operação transitiva é o mesmo, independentemente do número de operações transitivas a serem tratadas. Pode-se concluir, portanto, que o aumento no número de operações transitivas não degradou o desempenho das políticas *redo* e *undo*.

Comparando-se a política *redo* com a política *blind*, observa-se que a política *blind* teve uma redução média de tempo de 78,83% para operações transitivas diretas e de 77,63% para operações transitivas indiretas. Com relação à política *undo*, a política *blind* teve uma redução média de tempo de 78,69% para operações transitivas diretas e de 78,08% para operações transitivas indiretas.

Analisando-se o comportamento constante das políticas *blind* e *restrict* e linear das políticas *redo* e *undo* em relação ao aumento no número de operações transitivas, pode-se afirmar que as quatro políticas propostas são aplicáveis em situações reais.

5.2 Reaplicação pelo Método VRT

Nessa seção são apresentados os resultados obtidos no emprego do método proposto VRT (introduzido na Seção 4.5.3) na reaplicação de operações. O objetivo da seção é investigar a aplicabilidade e a viabilidade do método, considerando dois fatores de escalabilidade, sendo o número de operações a ser reaplicadas e o número de fontes envolvidas na integração.

Como base para comparações, foi utilizado o processo de integração usado para coleta das operações. Ou seja, os resultados obtidos pelo método VRT são comparados com os resultados obtidos pela coleta. A forma como foi feita a coleta é descrita a seguir. Nesta dissertação, a coleta requer a interação com o usuário por meio da ferramenta Reconciliador de Dados Acadêmicos. Entretanto, para realização de testes de desempenho, o processo de integração foi realizado automaticamente. Ou seja, a integração de um par de documentos foi feita automaticamente de forma a simular as ações do usuário no processo de integração. Dessa forma, o tempo de coleta consiste no tempo de processamento da integração, sem envolver o tempo que o usuário leva para tomar decisões. Em uma situação em que o usuário toma as decisões, os gráficos que representam os tempos de coleta devem ser multiplicados por um valor que representa o tempo médio para o usuário tomar uma decisão a respeito de uma inconsistência. Portanto, o tempo da coleta consiste no melhor caso do tempo de coleta em uma situação real, dificilmente obtida por causa da interação com o usuário.

Nos testes de desempenho, os tempos coletados referem-se aos tempos de processamento para coleta e reaplicação das operações. Durante a coleta, foi utilizada a política *blind* para tratamento de operações de sobreposição. Na reaplicação, foi utilizada a política de validação completa.

Essa seção está organizada da seguinte forma. As Seções 5.2.1 e 5.2.2 consistem em testes para averiguar a escalabilidade do método VRT quanto ao aumento do número de operações coletadas e quanto ao aumento do número de fontes de entrada, respectivamente. Nesses testes, cada uma das operações de *edição*, *cópia*, *inserção* e *remoção* foi investigada separadamente. A Seção 5.2.3 ilustra um teste no qual essas

quatro operações foram misturadas segundo uma determinada porcentagem com o intuito de simular situações mais próximas às reais.

5.2.1 Escalabilidade do Número de Operações Realizadas

Os gráficos da Figura 5.3 têm como objetivo mostrar a influência do número de operações no tempo de coleta e reaplicação das operações de *edição*, *cópia*, *inserção* e *remoção*. Nesse conjunto de testes foram utilizados 30 currículos como fontes de dados para coletar de 500 a 10.000 operações, com um incremento de 500 operações entre um teste e outro.

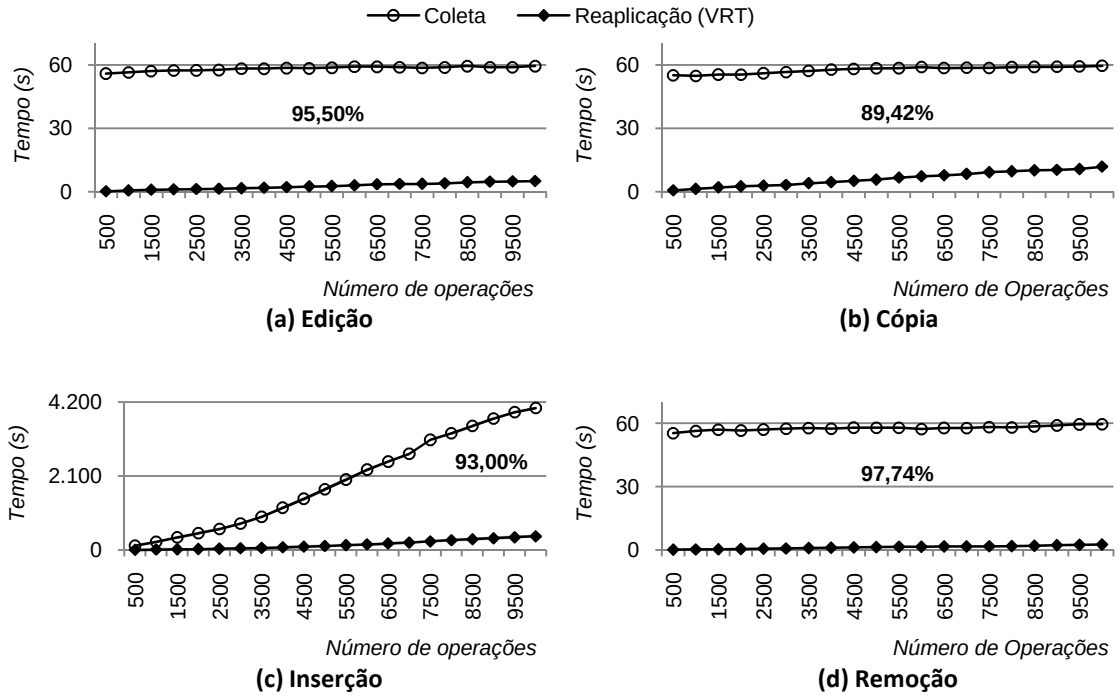


Figura 5.3: Escalabilidade do número de operações realizadas.

A reaplicação provida pelo método VRT teve melhor desempenho para todas as operações, quando comparado com a coleta. Para a *edição*, os resultados mostraram que reaplicar as operações pelo método VRT reduziu em média 95,50% o tempo de re-executar o processo de integração. Essa análise é semelhante para os gráficos das Figuras 5.3b, 5.3c e 5.3d. Nesses gráficos, os ganhos de desempenho do método VRT foram, respectivamente, de 89,42% para a *cópia*, 93,00% para a *inserção*, 97,74% para a *remoção*. Esses ganhos de desempenho significativos mostram a viabilidade de uso do método VRT.

Para as quatro operações, o tempo de processamento da coleta foi próximo aos 60 segundos. A operação de *inserção* foi uma exceção devido à sua característica de adicionar novos objetos às fontes. Como o processo de integração utilizado na coleta possui complexidade quadrática no número de objetos das fontes, com o aumento do número de operações, a coleta ultrapassou a casa dos 60 segundos obtido nas demais operações, atingindo tempo máximo de execução próximo a 4200 segundos.

A reaplicação provida pelo método VRT teve um aumento linear em relação ao número de operações realizadas. Portanto, pode-se verificar que o aumento no número de operações não degradou o desempenho do método VRT. Para exemplificar os tempos gastos, o maior tempo de execução da reaplicação foi próximo a 5 segundos para a operação de *edição*, 12 segundos para a *cópia*, 390 segundos para a *inserção* e 3 segundos para a operação de *remoção*. A operação de inserção leva mais tempo para ser executada por dois motivos. Primeiro, uma operação de *inserção* gera diversas operações de *cópia*, as quais também precisam ser reaplicadas. Segundo, uma operação de *inserção* requer a manipulação da estrutura do documento para construção de um novo objeto. Esses dois fatores fizeram com que o tempo de reaplicação para a operação de *inserção* fosse maior que das demais operações.

5.2.2 Escalabilidade do Número de Fontes

Os gráficos da Figura 5.4 têm como objetivo mostrar a influência do número de fontes no tempo de coleta e reaplicação das operações de *edição*, *cópia*, *inserção* e *remoção*. Foram coletadas 120 operações utilizando de 15 a 120 currículos como fontes de dados, com um incremento de 15 fontes entre um teste e outro.

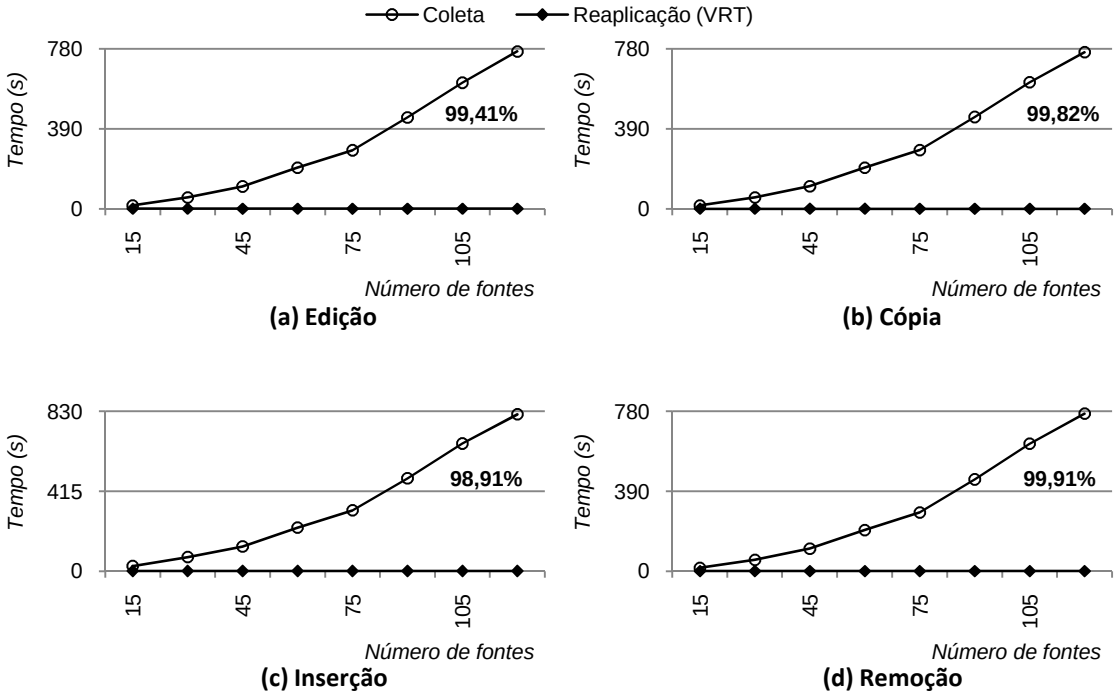


Figura 5.4: Escalabilidade do número de fontes.

A reaplicação provida pelo método VRT teve melhor desempenho para todas as operações, quando comparado com a coleta. O tempo de processamento da coleta teve crescimento quadrático, pois cada fonte é comparada com as demais em busca de inconsistências. Por outro lado, a reaplicação teve um comportamento constante com o

aumento do número de fontes, tendo o maior tempo de execução próximo a 0,5 segundo para a operação de *edição*, 0,15 segundos para a *cópia*, 1,5 segundos para a *inserção* e 0,06 segundos para a *remoção*. Esse comportamento constante mostrou que o aumento no número de fontes sendo integradas não influenciou o desempenho do método VRT.

Os resultados dos testes de desempenho descritos nessa seção também mostraram que o ganho médio de desempenho da reaplicação foi bastante significativo: 99,41% para a operação de *edição* (5.4a), 99,82% para a *cópia* (5.4b), 98,91% para a *inserção* (5.4c) e 99,91% para a remoção (5.4d).

5.2.3 Simulação de Caso Real

O gráfico da Figura 5.5 ilustra o tempo de execução da coleta e da reaplicação para uma simulação que alterna a execução de operações de *cópia*, *edição*, *inserção* e *remoção* de acordo com porcentagens esperadas em uma situação real. Uma vez que o objetivo da integração é resolver inconsistências entre fontes, a operação com maior porcentagem esperada é a de *cópia*. Em seguida, espera-se que a operação de *edição* tenha a segunda maior porcentagem pois ao resolver inconsistências, ambas as fontes podem conter valores incorretos e o usuário edita uma das fontes para o valor correto e executa a cópia para a outra fonte. A operação de inserção ocorre a uma taxa menor que a cópia e a edição uma vez que, no cenário considerado, é esperado que a inserção de novos objetos ocorra com pouca frequência. Por fim, a remoção é a operação com menor porcentagem, pois é esperado que, embora os objetos estejam inconsistentes, eles de fato pertençam àquela fonte. A tabela ilustrada na Figura 5.6a contém o número total de operações coletadas por número de fontes de entrada, enquanto que a tabela da Figura 5.6b ilustra as porcentagens de cada operação em relação ao total coletado.

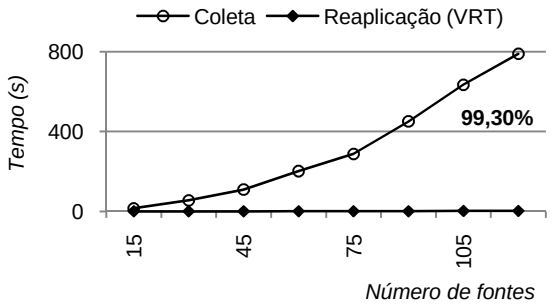


Figura 5.5: Simulação de caso real.

Os resultados obtidos mostraram um comportamento semelhante aos resultados da Seção 5.2.2: a coleta teve crescimento quadrático com o aumento do número de fontes, enquanto a reaplicação teve um crescimento linear. Nos testes de simulação de casa real, o ganho médio de desempenho do método VRT sobre a coleta foi ainda maior, devido à variação no número de operações coletadas. O método VRT reduziu em média

Número de fontes	Número de operações
15	120
30	238
45	270
60	317
75	335
90	399
105	687
120	841

(a) Número de fontes e operações

Operação	Porcentagem (%)
Cópia	75
Edição	15
Inserção	9
Remoção	1

(b) Porcentagem de cada operação

Figura 5.6: Número de operações por conjunto de fontes e porcentagem das operações.

99,30% o tempo de re-executar o processo de integração. Por exemplo, a coleta levou 788 segundos para executar 120 operações em 120 fontes, enquanto que o método VRT levou 2,5 segundos para realizar as mesmas 120 operações em 120 fontes.

5.3 Reordenação do Repositório

Nessa seção são apresentados testes relativos ao impacto da reordenação segura do repositório na reaplicação das operações. A reordenação segura foi introduzida na Seção 4.5.4. Nesses testes são considerados repositórios totalmente desordenados e repositórios totalmente ordenados. O objetivo dos testes é ilustrar o pior e o melhor caso de reordenação. Portanto, os resultados apresentados nos gráficos dessa seção ilustram, respectivamente, o tempo máximo (i.e., repositório não-ordenado) e o tempo mínimo (i.e., repositório ordenado) que o método VRT pode levar para reaplicar operações.

A Seção 5.3.1 investiga o impacto da reordenação com relação à escalabilidade do número de operações, enquanto que a Seção 5.3.2 ilustra esse impacto com relação à escalabilidade do número de fontes.

5.3.1 Escalabilidade do Número de Operações

O gráfico da Figura 5.7 ilustra o impacto da ordenação do repositório no tempo de reaplicação das operações quando se varia o número de operações no repositório. Nesse conjunto de testes foram utilizados 30 currículos como fontes de dados. Para o repositório totalmente desordenado, o crescimento do tempo de reaplicação foi linear. Esse crescimento linear deve-se ao fato de que, a cada operação, existe a necessidade de se trocar as fontes sendo manipuladas. Em contrapartida, quando o mesmo repositório está totalmente ordenado, a reaplicação teve um comportamento praticamente constante, desde que ocorre o menor número de trocas de fontes possível.

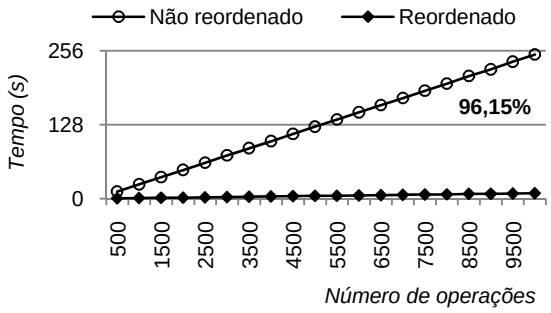


Figura 5.7: Escalabilidade do número de operações.

Os resultados mostraram que o ganho médio de desempenho obtido no tempo de reaplicação das operações devidamente ordenadas foi 96,15% em relação às operações desordenadas. Por exemplo, com o repositório totalmente desordenado, o método VRT levou 249,08 segundos para executar 10.000 operações, enquanto que com o repositório totalmente ordenado o método VRT levou 9,5 segundos para realizar as mesmas 10.000 operações. Como pode ser observado, quanto melhor reordenadas estão as operações, melhor é o desempenho do método VRT.

O tempo para reordenar o repositório considerado nesse teste é ilustrado pelo gráfico da Figura 5.8. O algoritmo apresentou complexidade quadrática no seu pior caso, ou seja, quando o repositório está totalmente desordenado, pois cada operação é comparada com todas as operações que a antecedem no repositório em busca de um lugar para reordená-la. Entretanto, devido ao ganho de desempenho significativo proporcionado pela reordenação frente ao reduzido tempo para reordenação do repositório, pode-se concluir que a proposta da reordenação segura apresentada na Seção 4.5.3 melhora o desempenho do método VRT sempre que o repositório puder ser ordenado.

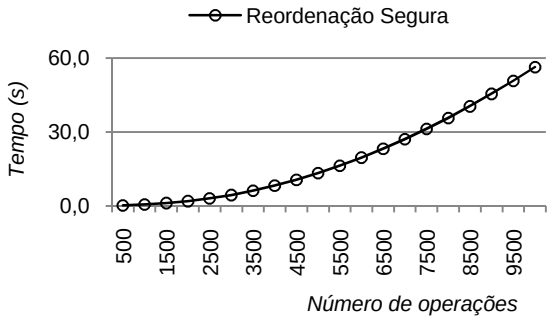


Figura 5.8: Tempo para reordenar o repositório.

5.3.2 Escalabilidade do Número de Fontes

O gráfico da Figura 5.9 ilustra o impacto da ordenação do repositório no tempo de reaplicação das operações quando se varia o número de fontes no repositório. Nesse

conjunto de testes foi utilizado um repositório com 120 operações. Os resultados mostraram que o ganho médio de desempenho no tempo de reaplicação de um repositório totalmente ordenado para um repositório totalmente desordenado foi de 94,58%. Por exemplo, com o repositório totalmente desordenado, o método VRT levou 3,01 segundos para executar 120 operações em 15 fontes, enquanto que o método VRT levou 0,16 segundos para realizar as mesmas 120 operações devidamente reordenadas.

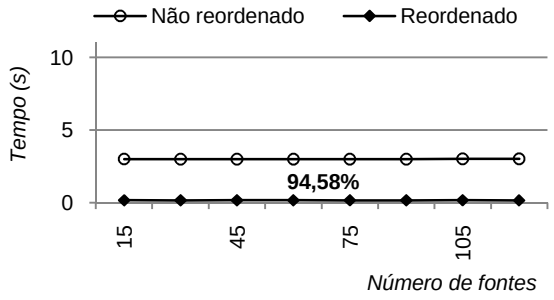


Figura 5.9: Escalabilidade do número de fontes.

Os resultados também mostraram que o tempo de reordenação manteve-se constante, desde que o número de operações no repositório se manteve constante. O tempo para reordenação foi de 0,02 segundos, tal como ilustrado pelo gráfico da Figura 5.10. Portanto, pode-se concluir que o fato do repositório estar reordenado não influenciou os tempos de reaplicação variando-se o número de fontes manipuladas.

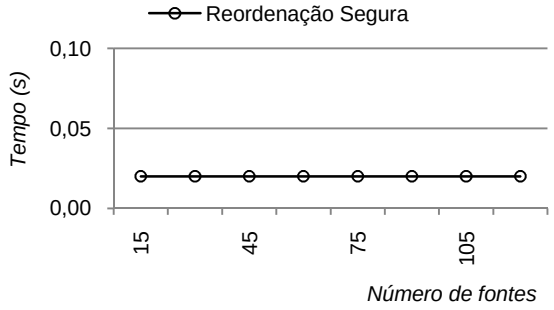


Figura 5.10: Tempo para reordenar o repositório.

5.4 Considerações Finais

Esse capítulo descreveu os testes de desempenho realizados para investigar as características do modelo MPPI, as quais consistem em procedência baseada em operações, tratamento de operações de sobreposição, validação das operações e reaplicação de operações.

O tratamento de operações de sobreposição foi testado variando-se o número de operações no repositório, bem como o número de operações transitivas associadas à

operação sobreposta. Os resultados mostraram que as políticas propostas para tratamento de operações de sobreposição (i.e., *blind*, *restrict*, *redo* e *undo*) obtiveram, no pior caso, comportamento linear com a variação dos parâmetros testados, podendo, portanto, serem implementadas em um sistema de integração sem degradar o seu tempo para coleta de operações.

A reaplicação de operações provida pelo método VRT foi testada variando-se o número de operações coletadas e o número de fontes integradas, considerando todas as operações propostas de *edição*, *cópia*, *inserção* e *remoção* e utilizando a validação completa. Os resultados mostraram que o método VRT proporcionou ganhos de desempenho significativos frente à coleta quando o objetivo é restabelecer resultados de processos de integração que foram já executados pelo menos uma vez. Os testes também mostraram que reordenar o repositório antes da reaplicação pode melhorar ainda mais o desempenho do método VRT, sempre que o repositório for passível de reordenação. Portanto, o método VRT se mostrou um método apropriado para recuperar resultados de processos de integração quando as fontes não podem ser diretamente atualizadas pelo processo de integração.

Capítulo

6

Conclusões

Esta dissertação enfoca a procedência dos dados na resolução de conflitos em nível de instância em processos de integração. O cenário de integração considerado possui duas características principais, as quais consistem na participação do usuário na resolução dos conflitos e o fato de que o processo de integração não pode alterar as fontes de dados por falta de permissão de escrita. Dessa forma, as fontes podem continuar a fornecer os mesmos dados inconsistentes em processos de integração subsequentes, independentemente das decisões de integração tomadas pelo usuário em processos anteriores. Portanto, o usuário deve resolver as mesmas inconsistências em diferentes processos de integração, o que pode levar a decisões diferentes para uma mesma inconsistência. Ademais, nesse cenário o tempo gasto pelo usuário na resolução de conflitos tende a crescer de um processo para outro, uma vez que novas inconsistências podem surgir entre os processos.

Esta dissertação propõe o MPPI, um modelo de procedência dos dados para subsidiar processos de integração. O modelo MPPI possui quatro características. A primeira delas consiste no mapeamento da procedência dos dados em operações de *cópia*, *edição*, *inserção* e *remoção*, bem como a definição de operações transitivas, que alteram a chave e de sobreposição. A segunda característica é o tratamento de operações de sobreposição, por meio da proposta das políticas *blind*, *restrict*, *undo* e *redo*. A terceira característica consiste na identificação das anomalias das fontes consistentes e do detino sobrescrito, decorrentes do fato de que fontes de dados autônomas podem alterar os seus dados entre processos de integração. Tais anomalias motivaram a definição de quatro tipos de validação das operações, sendo validação completa, da origem, do destino, ou nenhuma. A quarta característica consiste na definição dos métodos VRS e VRT para reaplicação das operações, além da proposta da reordenação segura para otimizar o desempenho do método VRT.

A validação do modelo MPPI foi realizada por meio de testes de desempenho que investigaram o tratamento de operações de sobreposição, o método VRT e a reordenação segura. Nos testes, a medida coletada foi o tempo de execução em segundos.

Com relação ao tratamento de operações de sobreposição, os testes levaram em consideração dois fatores de escalabilidade: o número de operações no repositório e o número de operações transitivas à operação sobreposta. Os resultados obtidos mostraram a viabilidade de implementação das políticas *blind*, *restrict*, *undo* e *redo* em sistemas de integração reais. Por um lado, o tempo de execução permaneceu constante com o aumento do número de operações armazenadas. Por outro lado, o tempo requerido para coleta, no pior caso, aumentou linearmente com o aumento do número de operações transitivas. Ademais, o tempo máximo gasto para tratar uma operação de sobreposição foi apenas de 2 segundos, considerando 30 operações transitivas. Essa quantidade de operações transitivas está além do esperado para o cenário de integração considerado.

O método VRT foi comparado com a coleta das operações decorrentes de um processo de integração, considerando como fatores de escalabilidade o número de operações no repositório e o número de fontes sendo integradas. Essa comparação teve como objetivo mostrar que reaplicar as operações é mais eficiente do que executar o processo de integração novamente. Os ganhos obtidos com a reaplicação foram muito significativos. Quanto ao número de operações no repositório, o método VRT obteve ganho médio de desempenho igual a 95,50% para a operação de *edição*, 89,42% para a *cópia*, 93,00% para a *inserção* e 97,74% para a *remoção*. Com relação ao aumento do número de fontes a serem integradas, o método VRT obteve ganho médio de desempenho igual a 99,41% para a operação de *edição*, 99,82% para a *cópia*, 98,91% para a *inserção* e 99,91% para a *remoção*.

Por fim, a influência da reordenação segura das operações no desempenho do método VRT foi investigada considerando os mesmos fatores de escalabilidade que os usados para investigar esse método. Nos testes realizados, foram considerados repositórios totalmente desordenados e repositórios totalmente ordenados. Os resultados de desempenho obtidos mostraram que reordenar o repositório antes da reaplicação pode melhorar ainda mais o desempenho do método VRT. Com o aumento do número de operações no repositório, a reordenação proporcionou um ganho médio de desempenho do método VRT de 96,15%. Quanto ao aumento do número de fontes, a reordenação também proporcionou um significativo ganho médio de desempenho do método VRT, sendo este de 94,58%.

6.1 Principais Contribuições

As principais contribuições da realização deste trabalho são:

- Proposta do modelo MPPI, um modelo de procedência dos dados para subsidiar processos de integração;
- Identificação de anomalias que podem ser geradas pela reaplicação das operações, caso as fontes tenham sido modificadas desde o momento em que as operações foram definidas até o momento no qual elas são reaplicadas automaticamente;
- Redução no tempo de integração gasto pelo usuário entre dois processos de integração nos quais as fontes de dados possuem permissão apenas para leitura, obtido pela reaplicação de operações; e
- Eliminação de decisões conflitantes tomadas pelo usuário a respeito de uma mesma inconsistência de dados entre dois processos de integração, obtido pela reaplicação de operações.

Além do modelo MPPI, durante o mestrado também foi realizada a finalização da ferramenta Reconciliador de Dados Acadêmicos (Tomazela et al., 2008), a qual originou um artigo publicado no Simpósio Brasileiro de Banco de Dados.

6.2 Trabalhos Futuros

Trabalhos futuros relativos à extensão do modelo MPPI incluem:

- Definição de novas operações vislumbrando integração de esquemas (Kang e Naughton, 2003; Madhavan et al., 2005; McCann et al., 2005), desde que o modelo MPPI enfoca a integração de instâncias. Uma operação de integração de esquemas pode envolver, por exemplo, a união de atributos em um único atributo, estabelecendo as devidas correspondências entre seus domínios. A definição de novas operações visa aumentar o escopo de abrangência do modelo MPPI;
- Definição de novas operações vislumbrando integração cujo objetivo é gerar agrupamentos de entidades similares (Bhattacharya e Getoor, 2007; Chen et al., 2007; Dong et al., 2005; Kalashnikov e Mehrotra, 2006; On et al., 2006). Uma atividade complementar sendo realizada é o desenvolvimento da ferramenta VER (*Visual Entity Reconciler*), a qual tem por objetivo ajudar os usuários a melhorar a precisão de agrupamentos de entidades similares e a gerar entidades integradas. O objetivo desse trabalho futuro consiste em adaptar o modelo MPPI para coletar a procedência dos dados enquanto o usuário interage com a ferramenta;
- Propagação de decisões de integração. Por exemplo, uma decisão de integração tomada sobre um subobjeto *autor* de um objeto $artigo_1$ muitas vezes tem que ser tomada novamente quando o mesmo subobjeto *autor* aparece em outro objeto

$artigo_2$. Neste caso, é importante que os conflitos resolvidos para o autor do $artigo_1$ sejam resolvidos automaticamente não somente para o autor do $artigo_2$, mas também para todos os objetos nos quais aquele autor aparece. Essa funcionalidade visa reduzir o tempo de integração gasto pelo usuário e impedir que ele tome decisões diferentes sobre um mesmo problema de inconsistência; e

- Extensão do modelo MPPI para a integração de bancos de dados biológicos (BDBs). BDBs armazenam dados referentes aos organismos vivos, tais como sequências de nucleotídeos e aminoácidos, mapas genéticos, estruturas tridimensionais de proteínas, identificação e anotação de genes e literatura associada, além de quaisquer informações derivadas destes ou necessárias para a compreensão dos mesmos (Baxevanis e Ouellette, 2005; Cohen, 2004; Gusfield, 1997; Korf et al., 2003). A extensão do modelo MPPI com esta finalidade tem como objetivo enfocar também problemas de integração de dados biológicos, desde que esses problemas tornam-se cada vez mais importantes com o aumento exponencial do volume de dados sequenciados e com o aumento do número de BDBs heterogêneos.

Referências Bibliográficas

Agrawal, P.; Benjelloun, O.; Sarma, A. D.; Hayworth, C.; Nabar, S.; Sugihara, T.; Widom, J. Trio: a system for data, uncertainty, and lineage. In: *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, 2006, p. 1151–1154.

Anand, M. K.; Bowers, S.; McPhillips, T.; Ludäscher, B. Efficient provenance storage over nested data collections. In: *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, New York, NY, USA: ACM, 2009, p. 958–969.

Archer, D. W.; Delcambre, L. M. L.; Maier, D. A framework for fine-grained data integration and curation, with provenance, in a dataspace. In: *Workshop on the Theory and Practice of Provenance*, 2009.

Baxevanis, A. D.; Ouellette, B. F. F. *Bioinformatics: a practical guide to the analysis of genes and proteins.* 3 ed. Wiley-Interscience, 540 p., 2005.

Benjelloun, O.; Das Sarma, A.; Halevy, A.; Theobald, M.; Widom, J. Databases with uncertainty and lineage. *The VLDB Journal*, v. 17, n. 2, p. 243–264, 2008.

Benjelloun, O.; Sarma, A. D.; Halevy, A.; Widom, J. Uldbs: databases with uncertainty and lineage. In: *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, VLDB Endowment, 2006, p. 953–964.

Bhattacharya, I.; Getoor, L. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data*, v. 1, n. 1, p. 5, 2007.

Bose, R.; Frew, J. Composing lineage metadata with xml for custom satellite-derived data products. In: *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM'04) - Volume 00*, IEEE Computer Society, 1007130 275, 2004.

Buneman, P.; Chapman, A.; Cheney, J. Provenance management in curated databases. In: *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, New York, NY, USA: ACM, 2006a, p. 539–550.

BUNEMAN, P.; CHAPMAN, A.; CHENEY, J.; VANSUMMEREN, S. A provenance model for manually curated data. In: MOREAU, L.; FOSTER, I. T., eds. *IPAW*, Springer, 2006b, p. 162–170 (*Lecture Notes in Computer Science*, v.4145).

BUNEMAN, P.; KHANNA, S.; TAJIMA, K.; TAN, W.-C. Archiving scientific data. *ACM Transactions on Database Systems (TODS)*, v. 29, n. 1, p. 2–42, 2004.

BUNEMAN, P.; KHANNA, S.; TAN, W. C. Data provenance: Some basic issues. In: *FST TCS 2000: Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, London, UK: Springer-Verlag, 2000, p. 87–93.

BUNEMAN, P.; KHANNA, S.; TAN, W. C. Why and where: A characterization of data provenance. In: *ICDT '01: Proceedings of the 8th International Conference on Database Theory*, London, UK: Springer-Verlag, 2001, p. 316–330.

CHAPMAN, A.; JAGADISH, H. V. Why not? In: *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, New York, NY, USA: ACM, 2009, p. 523–534.

CHAPMAN, A. P.; JAGADISH, H. V.; RAMANAN, P. Efficient provenance storage. In: *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA: ACM, 2008, p. 993–1006.

CHAUDHURI, S.; DAYAL, U. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, v. 26, p. 65–74, 1997.

CHEN, Z.; KALASHNIKOV, D. V.; MEHROTRA, S. Adaptive graphical approach to entity resolution. In: *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, New York, NY, USA: ACM, 2007, p. 204–213.

COHEN, J. Bioinformatics - an introduction for computer scientists. *ACM Computing Surveys (CSUR)*, v. 36, n. 2, p. 122–158, 2004.

DA SILVA, P. P.; MCGUINNESS, D. L.; FIKES, R. A proof markup language for semantic web services. *Information Systems*, v. 31, n. 4-5, p. 381–395, 2006.

DEL RIO, N.; SILVA, P. P. Probe-it! visualization support for provenance. In: *Advances in Visual Computing*, v. Volume 4842/2007 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 732–741, 2007.

DONG, X.; HALEVY, A.; MADHAVAN, J. Reference reconciliation in complex information spaces. In: *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, New York, NY, USA: ACM, 2005, p. 85–96.

FILETO, R.; MEDEIROS, C. B.; LIU, L.; PU, C.; ASSAD, E. D. Using domain ontologies to help track data provenance. In: *Simpósio Brasileiro de Banco de Dados*, 2003, p. 84–98.

FREIRE, J.; KOOP, D.; SANTOS, E.; SILVA, C. T. Provenance for computational tasks: A survey. *IEEE Computing in Science & Engineering*, v. 10, n. 3, p. 11–21, 2008.

GLAVIC, B.; ALONSO, G. Perm: Processing provenance and data on the same data model through query rewriting. In: *Proceedings of IEEE 25th International Conference on Data Engineering (ICDE)*, 2009, p. 174–185.

GLAVIC, B.; DITT, K. R. Data provenance: A categorization of existing approaches. In: *The German Database Conference*, Aachen, Germany, 2007, p. 227–241.

GOBLE, C. Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In: *Workshop on Data Derivation and Provenance*, Chicago, 2002.

GUSFIELD, D. *Algorithms on strings, trees, and sequences: computer science and computational biology.* 1 ed. Cambridge University Press, 534 p., 1997.

HALEVY, A. Y.; ASHISH, N.; BITTON, D.; CAREY, M.; DRAPER, D.; POLLOCK, J.; ROSENTHAL, A.; SIKKA, V. Enterprise information integration: Successes, challenges and controversies. In: *International Conference on Management of Data (SIGMOD)*, 2005, p. 778–787.

HALEVY, A. Y.; RAJARAMAN, A.; ORDILLE, J. Data integration: The teenage years. In: *International Conference on Very Large Data Bases (VLDB)*, 2006, p. 9–16.

HEINIS, T.; ALONSO, G. Efficient lineage tracking for scientific workflows. In: *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA: ACM, 2008, p. 1007–1018.

KALASHNIKOV, D. V.; MEHROTRA, S. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst.*, v. 31, n. 2, p. 716–767, 2006.

KANG, J.; NAUGHTON, J. F. On schema matching with opaque column names and data values. In: *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, New York, NY, USA: ACM, 2003, p. 205–216.

KEMENTSIETSIDIS, A.; WANG, M. On the efficiency of provenance queries. In: *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, Washington, DC, USA: IEEE Computer Society, 2009, p. 1223–1226.

KIMBALL, R.; ROSS, M. *The data warehouse toolkit*, v. 1. 2nd ed. New York: Wiley Computer Publishing, 2002.

KORF, I.; YANDELL, M.; BEDELL, J. *BLAST*. 1 ed. O'Reilly, 339 p., 2003.

MADHAVAN, J.; BERNSTEIN, P. A.; DOAN, A.; HALEVY, A. Corpus-based schema matching. In: *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, Washington, DC, USA: IEEE Computer Society, 2005, p. 57–68.

MCCANN, R.; ALSHEBLI, B.; LE, Q.; NGUYEN, H.; VU, L.; DOAN, A. Mapping maintenance for data integration systems. In: *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, VLDB Endowment, 2005, p. 1018–1029.

MUNISWAMY-REDDY, K.-K.; HOLLAND, D. A.; BRAUN, U.; SELTZER, M. Provenance-aware storage systems. In: *Proceedings of the Annual Technical Conference on USENIX'06 Annual Technical Conference*, Boston, MA: USENIX Association, 1267363 4-4, 2006.

MUNROE, S.; MILES, S.; MOREAU, L.; VÁZQUEZ-SALCEDA, J. Prime: a software engineering methodology for developing provenance-aware applications. In: *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, New York, NY, USA: ACM, 2006, p. 39–46.

MUTSUZAKI, M.; THEOBALD, M.; DE KEIJZER, A.; WIDOM, J.; AGRAWAL, P.; BENJELLOUN, O.; SARMA, A. D.; MURTHY, R.; SUGIHARA, T. Trio-one: Layering uncertainty and lineage on a conventional dbms (demo). In: *Proceedings of the Third Biennial Conference on Innovative Data Systems Research*, 2007, p. 269–274.

NASCIMENTO, A. M.; HARA, C. S. A model for xml instance level integration. In: *Simpósio Brasileiro de Banco de Dados*, 2008, p. 46–60.

ON, B.-W.; ELMACIOGLU, E.; LEE, D.; KANG, J.; PEI, J. Improving grouped-entity resolution using quasi-cliques. In: *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, Washington, DC, USA: IEEE Computer Society, 2006, p. 1008–1015.

SHIRI, N.; TAGHIZADEH-AZARI, A. Lineage tracing in mediator-based information integration systems. In: *Advanced Distributed Systems*, v. Volume 3563/2005 de *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 267–282, 2005.

SIMMHAN, Y. L.; PLALE, B.; GANNON, D. A survey of data provenance in e-science. *SIGMOD Record*, v. 34, n. 3, p. 31–36, 2005.

TAN, W.-C. Research problems in data provenance. *IEEE Data Engineering Bulletin*, v. 27, n. 4, p. 45–52, 2004.

TOMAZELA, B.; DE AGUIAR CIFERRI, C. D.; JUNIOR, C. T. Reconciliando dados de cunho acadêmico. In: *SBBD '08: Proceedings of the 23rd Brazilian symposium on Databases*, Porto Alegre, Brazil, Brazil: Sociedade Brasileira de Computaç ao, 2008, p. 283–297.

WIDOM, J. Trio: A system for integrated management of data, accuracy, and lineage. In: *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR '05)*, 2005.

WIEDERHOLD, G. Mediators in the architecture of future information systems. *Computer*, v. 25, n. 3, p. 38–49, 1992.

WU, M.-C.; BUCHMANN, A. P. Research issues in data warehousing. In: *BTW*, p. 61–82, 1997.

ZHAO, Y.; WILDE, M.; FOSTER, I. T. Applying the virtual data provenance model. In: *International Provenance and Annotation Workshop*, v. 4145, Chicago, IL, USA, p. 148–161, 2006.