AcCORD: um modelo colaborativo assíncrono para a reconciliação de dados

Dayse Silveira de Almeida

Dayse Silveira de Almeida

AcCORD: um modelo colaborativo assíncrono para a reconciliação de dados

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutora em Ciências - Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Cristina Dutra de Aguiar Ciferri

USP – São Carlos
Março de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Almeida, Dayse Silveira de
A447a AcCORD: um modelo colaborativo assíncrono para a reconciliação de dados / Dayse Silveira de Almeida; orientadora Cristina Dutra de Aguiar Ciferri. -- São Carlos, 2016.
135 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016.

1. Reconciliação de dados. 2. Resolução de conflitos. 3. Integração de dados. 4. Compartilhamento de dados. 5. Procedência de dados. I. Ciferri, Cristina Dutra de Aguiar, orient. II. Título.

Dayse Silveira de Almeida

AcCORD: asynchronous collaborative data reconciliation model

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação - ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Cristina Dutra de Aguiar Ciferri

USP – São Carlos
March 2016

Resumo

Reconciliação é o processo de prover uma visão consistente de dados provenientes de várias fontes de dados. Embora existam na literatura trabalhos voltados à proposta de soluções de reconciliação baseadas em colaboração assíncrona, o desafio de reconciliar dados quando vários usuários colaborativos trabalham de forma assíncrona sobre as mesmas cópias locais de dados, compartilhando somente eventualmente as suas decisões de integração particulares, tem recebido menos atenção. Nesta tese de doutorado investiga-se esse desafio, por meio da proposta do modelo AcCORD (*Asynchronous COllaborative data Reconciliation moDel*). AcCORD é um modelo colaborativo assíncrono para reconciliação de dados no qual as atualizações dos usuários são mantidas em um repositório de operações na forma de dados de procedência. Cada usuário tem o seu próprio repositório para armazenar a procedência e a sua própria cópia das fontes. Ou seja, quando inconsistências entre fontes importadas são detectadas, o usuário pode tomar decisões de integração para resolvê-las de maneira autônoma, e as atualizações que são executadas localmente são registradas em seu próprio repositório. As atualizações são compartilhadas entre colaboradores quando um usuário importa as operações dos repositórios dos demais usuários. Desde que diferentes usuários podem ter diferentes pontos de vista para resolver o mesmo conflito, seus repositórios podem estar inconsistentes. Assim, o modelo AcCORD também inclui a proposta de diferentes políticas de reconciliação multiusuário para resolver conflitos entre repositórios. Políticas distintas podem ser aplicadas por diferentes usuários para reconciliar as suas atualizações. Dependendo da política aplicada, a visão final das fontes importadas pode ser a mesma para todos os usuários, ou seja, um única visão global integrada, ou resultar em distintas visões locais para cada um deles. Adicionalmente, o modelo AcCORD também incorpora um método de propagação de decisões de integração, o qual tem como objetivo evitar que um usuário tome decisões inconsistentes a respeito de um mesmo conflito de dado presente em diferentes fontes, garantindo um processo de reconciliação multiusuário mais efetivo. O modelo AcCORD foi validado por meio de testes de desempenho que avaliaram as políticas propostas, e por entrevistas a usuários que avaliaram não somente as políticas propostas mas também a qualidade da reconciliação multiusuário. Os resultados obtidos demonstraram a eficiência e a eficácia do modelo proposto, além de sua flexibilidade para gerar uma visão integrada ou distintas visões locais. As entrevistas realizadas demonstraram diferentes percepções dos usuários quanto à qualidade do resultado provido pelo modelo AcCORD, incluindo aspectos relacionados à consistência, aceitabilidade, corretude, economia de tempo e satisfação.

Palavras-chave: Reconciliação de dados, resolução de conflitos, integração de dados, compartilhamento de dados, procedência dos dados.

Abstract

Reconciliation is the process of providing a consistent view of the data imported from different sources. Despite some efforts reported in the literature for providing data reconciliation solutions with asynchronous collaboration, the challenge of reconciling data when multiple users work asynchronously over local copies of the same imported data has received less attention. In this thesis we investigate this challenge. We propose AcCORD, an asynchronous collaborative data reconciliation model. It stores users' integration decision in logs, called repositories. Repositories keep data provenance, that is, the operations applied to the data sources that led to the current state of the data. Each user has her own repository for storing the provenance. That is, whenever inconsistencies among imported sources are detected, the user may autonomously take decisions to solve them, and integration decisions that are locally executed are registered in her repository. Integration decisions are shared among collaborators by importing each other's repositories. Since users may have different points of view, repositories may also be inconsistent. Therefore, AcCORD also introduces several policies that can be applied by different users in order to solve conflicts among repositories and reconcile their integration decisions. Depending on the applied policy, the final view of the imported sources may either be the same for all users, that is, a single integrated view, or result in distinct local views for each of them. Furthermore, AcCORD encompasses a decision integration propagation method, which is aimed to avoid that a user take inconsistent decisions over the same data conflict present in different sources, thus guaranteeing a more effective reconciliation process. AcCORD was validated through performance tests that investigated the proposed policies and through users' interviews that investigated not only the proposed policies but also the quality of the multiuser reconciliation. The results demonstrated the efficiency and efficacy of AcCORD, and highlighted its flexibility to generate a single integrated view or different local views. The interviews demonstrated different perceptions of the users with regard to the quality of the result provided by AcCORD, including aspects related to consistency, acceptability, correctness, time-saving and satisfaction.

Keywords: Data reconciliation, conflict resolution, data integration, data sharing, data provenance.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Reconciliação de dados é o processo de prover uma visão consistente dos dados importados de diferentes fontes. Esse problema tem sido investigado considerando um único usuário ou múltiplos usuários trabalhando sobre uma única cópia dos dados [Köpcke et al. 2010, Cao et al. 2013]. No entanto, o problema de reconciliação de dados quando múltiplos usuários trabalham de maneira assíncrona sobre cópias distintas dos dados tem recebido pouca atenção na literatura. Nesse contexto, por um lado, os usuários podem colaborar para compartilhar as suas decisões sobre a solução de conflitos de dados. Por outro lado, os usuários também podem discordar sobre quais são os valores corretos. Assim, o objetivo do processo de reconciliação de dados consiste tanto em prover uma única visão consistente para todos os usuários, quanto prover visões distintas para cada um deles. Ou seja, quando conflitos entre os dados são detectados, permite-se que todos os usuários concordem com o valor correto para o dado em um processo de integração colaborativo, ou permite-se que eles discordem, mas que trabalhem cooperativamente para compartilhar as suas decisões.

A integração de dados envolve questões em dois níveis: nível de esquema e nível de instância. No nível de esquema, o objetivo é resolver heterogeneidades estruturais, ou seja, visa-se principalmente a geração de mapeamentos semânticos entre esquemas de fontes de dados heterogêneas [Unal and Afsarmanesh 2010, Nguyen et al. 2011, Bhattacharjee and Jamil 2012]. No nível de instância, são dois os principais problemas: resolução de entidades e resolução de conflitos. A resolução de entidades consiste em identificar quais entidades de fontes distintas referem-se à mesma entidade no mundo real, enquanto que a resolução de conflitos refere-se ao problema de se resolver conflitos nos valores dos atributos de uma mesma entidade providos por diferentes fontes [Dorneles et al. 2007, Borges et al. 2008, Benjelloun et al. 2009, Köpcke et al. 2010,

Cao et al. 2013]. O foco adotado nesta pesquisa de doutorado é no problema de resolução de conflitos.

Processos de integração consomem muito tempo e normalmente envolvem intervenção manual. Desta forma, quando vários usuários importam dados das mesmas fontes, ou subconjuntos dessas fontes, compartilhar decisões e trabalhar colaborativamente pode ser uma estratégia interessante para economizar tempo. Como o trabalho colaborativo não deve interferir nos processos de integração de cada usuário, uma sincronização remota sobre cada atualização pode não ser conveniente [Kermarrec et al. 2001]. Sistemas síncronos são altamente interativos, e impedem que os usuários trabalhem em um modo desconectado. Assim, por questões de flexibilidade, uma reconciliação assíncrona é desejável para vários tipos de aplicação.

A reconciliação de dados colaborativa e assíncrona é caracterizada por um alto grau de independência, no qual os usuários trabalham de maneira autônoma e são fracamente conectados em um determinado momento. Um exemplo de tal ambiente colaborativo são os sistemas *e-health*. Esses sistemas permitem tratamentos colaborativos entre provedores de serviços de saúde, tais como farmácias, hospitais e laboratórios, os quais podem curar, revisar e estender os dados compartilhados de maneira autônoma e independente [Hossain et al. 2014]. Outros cenários de aplicação incluem curação e compartilhamento de dados em bioinformática [Taylor and Ives 2006, Ives et al. 2008] e dados bibliográficos [Tomazela et al. 2013].

A independência e a assincronicidade de processos de integração individuais envolvem dois principais desafios. O primeiro desafio está relacionado ao fato de como as decisões para resolver conflitos são armazenadas para serem trocadas entre os colaboradores. Ou seja, para compartilhar essas decisões, informações sobre as atualizações feitas pelos usuários devem ser armazenadas. Tais informações são chamadas de procedência dos dados e consistem no conjunto de metadados que possibilita identificar as fontes de dados e as transformações aplicadas sobre os dados, desde a criação até o estado atual desses [Benjelloun et al. 2008, Cheney et al. 2009, Yang et al. 2012, Ram and Liu 2012, Mahmood et al. 2013]. O segundo desafio está relacionado à discordância de alguns colaboradores frente às alterações realizadas no processo de integração. Tendo em vista esses desafios, é necessário resolver conflitos entre atualizações de colaboradores, principalmente quando esses tomam decisões distintas sobre o mesmo conflito de valor de atributo.

São poucos os trabalhos da literatura que propõem a utilização da procedência dos dados como estratégia para ajudar a identificar e solucionar conflitos no processo de integração de dados. Do ponto de vista de sistemas de integração, são várias as motivações para se armazenar a procedência dos dados [Tan 2004, Fileto et al. 2003, Simmhan et al. 2005,

Freire et al. 2008, Nascimento and Hara 2008, Huang and Cheng 2011, Beneventano et al. 2011, Demsky 2011, Ikeda et al. 2012]. Uma primeira motivação consiste em verificar o histórico dos dados, armazenando-se informações sobre as transformações que foram efetuadas nos dados durante o processo de integração. Essas informações podem ser usadas para reproduzir resultados do processo de integração ou para validar o processo realizado. Outra motivação é assegurar a qualidade dos dados integrados, pois, no processo de integração pode-se inferir, por exemplo, que dados obtidos de fontes confiáveis têm maior probabilidade de estarem corretos. 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. Outra motivação é a otimização de consultas em ambientes heterogêneos, também baseada na confiabilidade das fontes e autoria. Esta tese de doutorado baseia-se no uso da procedência dos dados em processos de reconciliação, considerando o problema de conflito de valores de atributos em nível de instância, a autoria e a confiabilidade das fontes, além da confiabilidade nas decisões dos colaboradores. A procedência também é usada para a retificação de dados incorretos em diferentes fontes.

Nesta tese, é proposto o modelo AcCORD (*Asynchronous COllaborative data ReconcIliation moDel*). AcCORD é um modelo colaborativo assíncrono para reconciliação de dados no qual as atualizações dos usuários são mantidas em um repositório de operações na forma de dados de procedência. Cada usuário tem o seu próprio repositório para armazenar a procedência e a sua própria cópia das fontes. Ou seja, quando inconsistências entre fontes importadas são detectadas, o usuário pode tomar decisões para resolvê-las de maneira autônoma, e as atualizações que são executadas localmente são registradas em seu próprio repositório. Atualizações são compartilhadas entre colaboradores importando cada repositório dos demais usuários. Desde que usuários podem ter diferentes pontos de vista, os repositórios podem estar inconsistentes. Assim, nesta tese também são propostas políticas para resolver conflitos entre repositórios. Políticas distintas podem ser aplicadas por diferentes usuários para reconciliar as suas atualizações. Dependendo da política aplicada, a visão final das fontes importadas pode ser a mesma para todos os usuários, ou seja, um única visão global integrada, ou resultar em distintas visões locais para cada um deles.

1.1 Motivação

Considere um conjunto de fontes de dados no domínio de dados bibliográficos nomeadas de acordo com o proprietário, tais como *Ana, Bruno e Carlos*, como mostrado na Figura 1.1. Usuários

$U_1, \ldots, U_n$ estão integrando essas fontes individualmente, mantendo cópias locais das fontes e as atualizando quando dados sobrepostos são detectados. Para isso, os usuários usam uma ferramenta de integração de dados em nível de instância para auxiliá-los no processo de integração e coletar automaticamente a procedência.

Fonte: Ana

Artigo
- Título: "AcCORD: Asynchronous COllaborative data ReconcIliation moDel"
- Ano: "2013"
- Revista: "Journal of the Brazilian Computer Society"
- Página inicial: "5"
- Página final: "20"
- ISSN: "01046000"
- Local: "Porto Alegre, RS, Brasil"
- Autores
  - Autor
    - Nome: "Ana"
    - Ordem citação: "1"
  - Autor
    - Nome: "Bruno"
    - Ordem citação: "2"

Fonte: Bruno

Artigo
- Título: "AcCORD: Asynchronous COllaborative data ReconcIliation moDel"
- Ano: "2015"
- Revista: "JBCS"
- Página inicial: "5"
- Página final: "24"
- ISSN: "01046500"
- Local: "Porto Alegre, RS, Brasil"
- Autores
  - Autor
    - Nome: "Ana"
    - Ordem citação: "1"
  - Autor
    - Nome: "Bruno"
    - Ordem citação: "2"
  - Autor
    - Nome: "Carlos"
    - Ordem citação: "3"

Fonte: Carlos

Artigo
- Título: "Accord: Asynchronous collaborative data reconcIliation model"
- Ano: "2005"
- Revista: "Journal of the Brazilian Comp. Society"
- Página inicial: "1"
- Página final: "234"
- ISSN: "01000000"
- Local: "Porto Alegre-RS"
- Autores
  - Autor
    - Nome: "Ana"
    - Ordem citação: "1"
  - Autor
    - Nome: "Carlos"
    - Ordem citação: "2"
  - Autor
    - Nome: "Bruno"
    - Ordem citação: "3"
  - Autor
    - Nome: "Daniel"
    - Ordem citação: "4"

Figura 1.1: Fontes contendo valores inconsistentes de atributos.

Considere o usuário U_1. Depois de importar as fontes de dados, ele nota que elas possuem várias inconsistências no valores de atributos, tais como no atributo *página final*, *ano* e *página inicial* do artigo intitulado *"AcCORD: Asynchronous COllaborative data ReconcIliation moDel"*, e decide que *Carlos* é a fonte mais confiável. Então ele copia (cp) o valor de *página final* (234) para a fonte *Bruno* sobrescrevendo o valor *24* e então copia da fonte *Bruno* para a fonte *Ana* sobrescrevendo o valor *20*. Essas atualizações são armazenadas no repositório R_1, como mostrado na Figura 1.2.

usuário	id	op	origem	destino	objeto	atributo	valor Origem	valor Destino	*timestamp*
U1	1	cp	Carlos	Bruno	Artigo [Título=AcCORD...]	página final	234	24	08:28:43_ 03/18/2014
	2	cp	Bruno	Ana	Artigo [Título=AcCORD...]	página final	234	20	08:29:01_ 03/18/2014

Figura 1.2: Repositório R_1 com decisões do usuário U_1.

Nesse exemplo, foi utilizada a ideia de dados de procedência baseados em operações do modelo PrInt [Tomazela et al. 2013], na qual cada operação possui o campo *op* que identifica o tipo de operação realizada, o campo *origem* que identifica a fonte que forneceu o valor correto para a atualização, o campo *destino* que identifica a fonte que foi atualizada, um campo para a chave que identifica univocamente o *objeto*, o *atributo* que teve o seu valor atualizado, o campo *valorOrigem* que armazena o valor correto, o campo *valorDestino* que armazena o valor incorreto atualizado para o *valorOrigem* e, por fim, o campo *timestamp* que especifica o momento em que a operação foi criada. Para ajudar a descrever os exemplos ao longo deste texto, foi adicionado nos repositórios um identificador para o usuário (coluna *usuário*) e um identificador para a operação (coluna *id*). Entretanto, o modelo AcCORD proposto nesta tese não é limitado a um modelo específico de procedência baseado em operações.

Depois, U_1 decide editar manualmente (*ed*) o valor da fonte *Carlos*, modificando o valor do atributo *página final* para *235*. Isso gera um novo registro *[id: 3, op: ed, origem: null, destino: Carlos, objeto: Artigo[Título=AcCORD...], atributo: página final, origem: 235, destino: 234, timestamp: 14:45:03_03/18;2014]* para ser adicionado ao repositório R_1. Note que deve-se considerar agora que o repositório é *inconsistente*, dado que existem dois valores distintos sendo escritos na *página final* da fonte *Carlos*. Desde que o repositório reflete as decisões do usuário sobre o tempo, sua última operação é considerada como aquela que contém o valor correto, e decisões anteriores podem ser reaplicadas para propagar a atualização para ambas as fontes *Bruno* e *Ana*. Essas ações são refletidas no repositório rearranjando a ordem das operações no repositório e atualizando as operações, como mostrado na Figura 1.3. Como resultado, o repositório torna-se consistente.

usuário	id	op	origem	destino	objeto	atributo	valor Origem	valor Destino	*timestamp*
U1	3	ed	null	Carlos	Artigo [Título=AcCORD...]	página final	235	234	14:45:03_ 03/18/2014
	1	cp	Carlos	Bruno	Artigo [Título=AcCORD...]	página final	235	24	08:28:43_ 03/18/2014
	2	cp	Bruno	Ana	Artigo [Título=AcCORD...]	página final	235	20	08:29:01_ 03/18/2014

Figura 1.3: Visão consistente do R_1 depois que a fonte *Carlos* foi atualizada.

Agora considere um conjunto de usuários $U_1, \ldots, U_n$, cada um importando o mesmo conjunto de fontes e resolvendo os mesmos conflitos nos dados de forma independente. Dessa forma, cada usuário manipula o seu próprio repositório de dados, conforme descrito anteriormente. Entretanto, os usuários podem estar realizando o processo de integração de forma colaborativa. Desde que todos concordam em colaborar, eles trocam as atualizações quando exportam os seus repositórios $R_1, \ldots, R_n$. Então, cada usuário U_i tem agora acesso não somente ao seu próprio repositório R_i, mas pode importar para R_i as atualizações de todos os demais colaboradores. Todos eles podem se beneficiar por aplicar as atualizações importadas na sua própria cópia das fontes, e então reduzir a sua carga de trabalho para o processo de integração. Dado um usuário, isso pode ser feito se as decisões tomadas pelos outros usuários não conflitarem com suas próprias decisões ou com as decisões de outros colaboradores. Caso contrário, cada colaborador tem que decidir como resolver o conflito.

Considere novamente o usuário U_1 e seu repositório depois de importar os repositórios dos colaboradores $U_2, \ldots, U_6$, mostrado na Figura 1.4. Nessa figura é mostrado como cada usuário resolveu o conflito no atributo *página final* do *Artigo[Título=AcCORD...]*. Observe que uma resolução de conflitos baseada no tempo, entre múltiplos usuários trabalhando independentemente pode não ser adequada nesse contexto. No exemplo corrente, se a última decisão for considerada como correta, seria escolhida a decisão tomada por U_2, desde que os *timestamps* dos registros são os mais recentes. U_2 considerou o valor provido por *Bruno* como sendo o valor correto, e assim a sua decisão deveria ser propagada para as cópias das fontes de U_1. Entretanto, essa pode não ser a ação pretendida pelo usuário U_1, que decidiu localmente que a fonte *Carlos* deve prevalecer sobre *Bruno*. Surge, então, a necessidade da proposta de um modelo que utiliza diferentes políticas para se adaptar ao contexto da aplicação dos usuários colaborativos, de forma a fornecer como resultado uma decisão mais satisfatória para esses usuários.

1.2 Contribuições

Em um processo de reconciliação colaborativo e assíncrono, políticas, tais como baseada em votação, ou aquela que prioriza as decisões locais, são necessárias. Mais importante, é necessário ter um modelo genérico que possa ser aplicado em diferentes cenários, de acordo com o resultado final pretendido pelo usuário. Ou seja, os usuários podem querer gerar colaborativamente uma única visão integrada para cada um, que estarão consistentes em um determinado momento; ou, criar individualmente distintas visões locais.

Nesta tese, esses desafios são tratados e são feitas as seguintes contribuições:

usuário	id	op	origem	destino	objeto	atributo	valor origem	valor destino	*timestamp*
U1	1	cp	Carlos	Bruno	Artigo [Título=AcCORD...]	página final	234	24	08:28:43_ 03/18/2014
	2	cp	Bruno	Ana	Artigo [Título=AcCORD...]	página final	234	20	08:29:01_ 03/18/2014
	3	cp	Bruno	Ana	Artigo [Título=AcCORD...]	ano	2015	2013	08:29:17_ 03/18/2014
U2	1	cp	Bruno	Daniel	Artigo [Título=AcCORD...]	página final	24	15	13:41:47_ 03/18/2014
U3	1	ed	*null*	Daniel	Artigo [Título=AcCORD...]	página final	22	15	12:24:55_ 03/18/2014
	2	cp	Daniel	Emanuel	Artigo [Título=AcCORD...]	página final	20	1352	12:32:47_ 03/18/2014
U4	1	cp	Daniel	Fábio	Artigo [Título=AcCORD...]	página final	15	42	10:13:45_ 03/18/2014
	2	cp	Daniel	Gabriel	Artigo [Título=AcCORD...]	página final	15	25	10:14:21_ 03/18/2014
	3	cp	Fábio	Hugo	Artigo [Título=AcCORD...]	página final	15	32	10:15:42_ 03/18/2014
U5	1	cp	Bruno	Gabriel	Artigo [Título=AcCORD...]	página final	24	25	09:45:57_ 03/18/2014
	2	cp	Gabriel	Ivan	Artigo [Título=AcCORD...]	página final	24	72	09:50:02_ 03/18/2014
U6	1	cp	Bruno	Hugo	Artigo [Título=AcCORD...]	página inicial	5	13	07:25:56_ 03/18/2014

Figura 1.4: Repositório R_1 com as decisões de todos os usuários.

- Introdução do AcCORD, um modelo colaborativo assíncrono para a reconciliação de dados, no qual as atualizações feitas pelos usuários são armazenadas em *logs*, chamados *repositórios*. Repositórios mantêm a procedência dos dados, ou seja, as operações aplicadas nas fontes de dados que levaram o banco de dados local ao seu estado corrente.

- Proposta de políticas para resolver possíveis conflitos que resultem do processo de reconciliação colaborativo e assíncrono. Essas políticas podem ser destinadas às aplicações que geram uma visão integrada ou distintas visões locais como resultado do trabalho colaborativo. As políticas são baseadas nas seguintes estratégias: remoção das operações realizadas por diferentes usuários cujas decisões conflitam entre si; priorização das decisões tomadas por um determinado usuário; ordem cronológica da tomada da decisão; priorização da decisão tomada pela maioria dos usuários; confiança nas fontes de dados; e, confiança nos usuários que tomaram as decisões.

- Proposta de um método de propagação de decisões de integração em nível de um único usuário para evitar que o usuário tenha que tomar a mesma decisão para o mesmo atributo de um objeto em diferentes fontes, economizando seu tempo. Esse método usa os dados de procedência para retificar as fontes de dados que estão incorretas, com base no processo de integração de dados em nível de único usuário.

A validação do modelo AcCORD e das políticas propostas foi feita por meio de testes que investigaram o gerenciamento de operações conflitantes, a fim de verificar as consequências do aumento no número de usuários, o número de operações não tratadas e removidas, o que causa a perda de decisões dos usuários, e a remoção de operações do repositório local, refletindo a perda de decisões do usuário que está realizando o processo de reconciliação. Os resultados obtidos mostraram a eficácia de cada política proposta, frente às necessidades dos usuários de gerar uma única visão integrada ou diferentes visões usando resultados de integração colaborativos. Portanto, a escolha da política depende do ambiente e da aplicação na qual ela será utilizada, podendo ser mais ou menos restritiva. Os resultados mostraram também a flexibilidade do modelo AcCORD o qual pode ser usado tanto para compartilhamento de dados, quanto para integração em ambiente multiusuário. Além disso, os resultados relacionados ao método de propagação de decisões mostraram a sua necessidade a fim de facilitar o trabalho do usuário e poupar o seu tempo, bem como a eficiência e a eficácia do método proposto. Resultados preliminares relacionados à proposta do modelo AcCORD e a algumas das políticas de reconciliação foram publicados no evento internacional International Conference on Enterprise Information Systems (ICEIS), o qual possui classificação CAPES Qualis Ciência da Computação B1 [Almeida et al. 2015].

1.3 Estrutura do Trabalho

Esta tese de doutorado está organizada da seguinte forma:

- No Capítulo 2 é descrita a fundamentação teórica necessária ao entendimento da tese. Nesse capítulo, são discutidos conceitos relacionados à integração de dados em nível de instância e em nível de esquema, e à procedência de dados. Também é descrito o modelo PrInt, o qual é usado para capturar os dados de procedência usados nesta pesquisa de doutorado.
- No Capítulo 3 são resumidos trabalhos correlatos encontrados na literatura e detalhados aqueles que possuem foco voltado aos sistemas de integração e compartilhamento de dados. Além disso, são discutidas as deficiências desses trabalhos e enfatizadas as motivações para o desenvolvimento do modelo AcCORD.
- No Capítulo 4 são descritas as contribuições desta tese de doutorado. São detalhadas as características das operações armazenadas no repositório, é introduzido o modelo AcCORD para trabalho colaborativo, e são descritas as políticas para a reconciliação de dados em ambiente multiusuário e o método proposto para a propagação de dados em nível de um único usuário.

- No Capítulo 5 são detalhados os resultados obtidos em testes experimentais a fim de validar a eficácia das políticas propostas. Para isso, foram utilizados dados de processos de integração considerando dois cenários. No primeiro cenário considerou-se dados em menor escala de volume, baseados em decisões de integração feitas por usuários reais. No segundo cenário, considerou-se um volume maior de dados, os quais foram gerados sinteticamente usando como base estatísticas dos dados reais. Além desses cenários, foi considerado um experimento baseado na percepção dos usuários em relação ao objetivo e aos resultados gerados pelas políticas. Por fim, um experimento foi realizado para o método de propagação de decisões de integração a fim de verificar tanto a percepção do usuário em relação à necessidade de um método automático para propagar decisões, quanto a eficiência do método.
- Finalmente, no Capítulo 6 são destacadas as conclusões obtidas nesta pesquisa de doutorado, as contribuições e os trabalhos futuros.

Capítulo 2

Integração e Procedência de Dados

Neste capítulo é descrita a fundamentação teórica necessária para o entendimento desta tese de doutorado. Nas Seções 2.1 e 2.2 são descritos conceitos básicos relacionados à integração e à procedência dos dados, respectivamente. Desde que o trabalho desenvolvido neste doutorado utiliza como base o modelo PrInt, na Seção 2.3 são descritas as principais características e o funcionamento desse modelo, bem como as formas de coleta e armazenamento dos dados de procedência em forma de repositório de operações, e os tipos de consulta usadas pelo modelo. O capítulo é finalizado na Seção 2.4, com as considerações finais.

2.1 Integração dos Dados

Integração dos dados é um problema enfrentado por aplicações que precisam acessar várias fontes de dados autônomas e heterogêneas [Hsiao and Kamel 1989, Sheth and Larson 1990, Hsiao 1992, Aguiar 1995, Halevy et al. 2005, Halevy et al. 2006, Roth et al. 2010a, Roth et al. 2010b, Nguyen et al. 2011]. Ela é importante, por exemplo: (i) para o desenvolvimento de projetos científicos de grande escala cujos dados são produzidos independentemente pelos pesquisadores; (ii) para instituições de pesquisa que precisam produzir relatórios institucionais a partir das publicações de seus pesquisadores que estão espalhadas em uma variedade de fontes, tais como as bases de publicações DBLP e *ISI Web of Knowledge*, os Currículos Lattes desses pesquisadores e os sistemas corporativos da instituição; e (iii) para o desenvolvimento de portais Web que integram dados de fontes pré-existentes.

Em qualquer modelo de dados é importante distinguir a descrição do banco de dados dos próprios dados. A descrição especificada durante o projeto do banco de dados é chamada de esquema, enquanto que os dados armazenados no banco de dados consistem nas instâncias. Assim, a integração de dados pode ocorrer tanto em nível de esquema (Seção 2.1.1) quanto em nível de instância (Seção 2.1.2), sendo que primeiramente devem ser resolvidos os conflitos em nível de esquema para que depois os conflitos em nível de instância possam ser também resolvidos.

2.1.1 Integração em Nível de Esquema

Em nível de esquema, a necessidade de se especificar correspondências entre esquemas de fontes de dados heterogêneas que se referem a uma mesma entidade surge devido à não uniformização desses esquemas, ou seja, ao uso de representações distintas do mesmo conceito. Por exemplo, tabelas que não possuem os mesmos atributos em um modelo relacional, atributos que se referem ao mesmo conceito mas possuem nomes diferentes ou, dados armazenados em diferentes estruturas. Dessa forma, o processo de integração depende da identificação de similaridades e conflitos entre os elementos dos diferentes esquemas. Esses conflitos entre esquemas podem ser divididos em grupos, tais como conflitos de nome, conflitos semânticos e conflitos estruturais.

Conflito de nome refere-se aos nomes utilizados para representar os elementos existentes nos esquemas a serem integrados. Diferentes nomes podem ser aplicados ao mesmo elemento constituindo-se o problema dos sinônimos. Por exemplo, uma fonte pode armazenar a data em atributos com nome *data*, enquanto outra fonte pode armazenar a mesma informação com o nome *timestamp*. Outro problema pode ocorrer quando o mesmo nome é aplicado a diferentes elementos, estabelecendo-se o problema dos homônimos. Por exemplo, o atributo *nome* pode ser utilizado para representar o nome de um empregado no esquema *Empregado*, e também pode ser utilizado no esquema *Departamento* para representar os nomes dos departamentos.

Conflito semântico ocorre quando o mesmo elemento é modelado em diferentes esquemas, representando conjuntos que se sobrepõem, ou seja, o conjunto de instâncias do elemento em um esquema é mais abrangente que o conjunto de instâncias do elemento em outro esquema. Os estudantes de uma universidade, por exemplo, podem ser representados em uma fonte como *estudante*, e em outra fonte como $estudante\_grad$ e $estudante\_pos$, especificando os diferentes tipos de estudantes da instituição, da graduação e da pós-graduação.

Conflito estrutural surge quando diferentes estruturas são usadas por diferentes fontes para representar o mesmo conceito. Por exemplo, quando atributos que representam o mesmo conceito são armazenados em diferentes tipos de dados, como a data em uma fonte, podendo ser armazenada no formato *dia/mês/ano*, enquanto que em outra fonte a data pode ser armazenada apenas como *mês/ano*. O atributo *sexo* pode ser armazenado em uma fonte no formato de *string*, feminino ou masculino, em uma segunda fonte como caractere F ou M, e em uma terceira fonte, como valor booleano 0 ou 1. Outro exemplo é quando o elemento *Pessoa* é representado por uma fonte como uma entidade, como os atributos *nome*, *cpf* e endereço, e em outra fonte é representado como atributo.

Uma abordagem descrita na literatura para solucionar o problema de integração de esquemas é por meio de mapeamentos de esquemas. Mapeamentos especificam como instâncias de um esquema fonte devem ser traduzidas dentro de instâncias de um esquema objetivo. Alguns estudos propõem ferramentas práticas e/ou algoritmos para geração de mapeamentos de esquemas [Ives et al. 2008, Fagin et al. 2009, Marnette et al. 2011]. Nesse caso, os sistemas utilizam como entrada uma especificação abstrata do mapeamento, usualmente feita de correspondências entre os dois esquemas, e geram os mapeamentos, normalmente na forma de tgds (*tuple generating dependencies*) [Beeri and Vardi 1984], além dos *scripts* executáveis necessários para realizar a tradução.

Em contrapartida, estudos teóricos sobre integração e troca de dados formalizaram a noção do problema e, nesse contexto, o foco não é a geração de mapeamento, e sim a caracterização de suas propriedades e soluções [Lenzerini 2002, Fagin et al. 2005]. O problema de troca de dados envolve um esquema fonte F e um esquema objetivo O, em que F e O são disjuntos. Como O pode ser um esquema criado independentemente, ele tem seu próprio conjunto de restrições $\sum_O$. Além disso, existe um conjunto de dependências fonte-para-objetivo que modelam a relação entre os esquemas fonte e objetivo, e especificam como e quais dados da fonte devem aparecer no objetivo. Esse conjunto de dependências fonte-para-objetivo $\sum_{FO}$ é da forma $\forall x(\Phi_F(x) \longrightarrow \chi_O(x))$, em que $\Phi_F(x)$ é uma fórmula sobre F e $\chi_O(x)$ é uma fórmula sobre O.

Para materializar uma instância J sobre o esquema objetivo O dado uma instância I sobre o esquema fonte F, e dado que J satisfaz às dependências do objetivo $\sum_F$ e I e J satisfazem às dependências fonte-para-objetivo $\sum_{FO}$, pode haver muitas soluções ou nenhuma para a instância J. Então, várias questões conceituais e técnicas surgem referentes à semântica de troca de dados como, quando a solução existe, quais soluções devem ser materializadas e quais propridades elas devem possuir para refletir os dados das fontes de maneira mais correta possível, e se uma solução "boa" pode ser eficientemente computada.

Os sistemas de integração de dados normalmente são LAV (*local-as-view*) ou GAV (*global-as-view*). Considere um sistema de integração como uma tripla $< G, F, M >$, em que G é um esquema global, F é um esquema fonte e M é um conjunto de asserções relacionando elementos do esquema global com elementos do esquema fonte [Lenzerini 2002]. Em um sistema LAV, cada asserção em M refere-se a um elemento do esquema fonte F para uma consulta (uma visão) sobre o esquema global G. Em um sistema GAV, cada asserção em M refere-se a um elemento do esquema global G para uma consulta (uma visão) sobre o esquema fonte F. Como as dependências fonte-para-objetivo $\sum_{FO}$ relacionam uma consulta sobre o esquema fonte F com uma consulta sobre o esquema objetivo O, a troca de dados não é um sistema LAV e nem GAV. Em vez disso, ela é considerada um GLAV (*global-and-local-as-view*) [Fagin et al. 2005]. Portanto, em um sistema de troca de dados o esquema objetivo é criado independentemente com suas próprias restrições e, em sistemas de integração, um esquema global é assumido para ser a fonte reconciliada, uma visão virtual de uma coleção de fontes heterogêneas e assim, não possui restrições.

Resolvidas as questões relativas à integração de esquemas de fontes heterogêneas, é necessário comparar essas fontes e integrá-las em nível de instância.

2.1.2 Integração em Nível de Instância

No nível de instância, são dois os principais problemas: ambiguidade na identificação de entidades e conflito de valores de atributos.

A ambiguidade na identificação de entidades surge porque entidades do mundo real são usualmente identificadas por valores de seus atributos, ao invés de por identificadores únicos. A resolução de entidades consiste, portanto, em identificar quais representações em fontes distintas, possivelmente inconsistentes, referem-se à mesma entidade no mundo real. Por exemplo, para se integrar publicações, é necessário identificar que a entidade e_1 = ({“Distributed query processing in a relational database system”}, {“169-180”}, {“Robert S. Epstein”; “Michael Stonebraker”; “Eugene Wong”}) de uma fonte refere-se à entidade e_2 = ({“Distrited query processing in a relational database system”}, {“169-179”},{“Epstein, R.S.”; “Stonebraker, M.”; “Wong, E.”}) de uma outra fonte.

A pesquisa sobre ambiguidade na identificação de entidades é extensa, e tem sido denominada na literatura como resolução de entidades, detecção de duplicações e reconciliação de referências [Dorneles et al. 2004, Dong et al. 2005, Gal et al. 2005, Aleman-Meza et al. 2006, Kalashnikov and Mehrotra 2006, Karger and Jones 2006, On et al. 2006, Bleiholder and Naumann 2006, Chen et al. 2007, Dorneles et al. 2007,

Borges et al. 2008, Benjelloun et al. 2009, Bleiholder and Naumann 2009, Kleb and Abecker 2010, Zhu et al. 2010, Shu et al. 2011, Whang and Garcia-Molina 2012, Ferreira et al. 2012, Getoor and Machanavajjhala 2012, Tomazela et al. 2013]. Basicamente, as técnicas existentes na literatura visam a geração de agrupamentos de entidades que têm certo grau de similaridade entre si e que, portanto, têm alta probabilidade de serem a mesma entidade do mundo real.

O conflito de valores de atributos refere-se ao fato de que diferentes fontes podem possuir valores conflitantes para atributos de uma mesma entidade do mundo real (isto é, que possuem um certo grau de similaridade nos valores de seus atributos e foram agrupadas conjuntamente). No exemplo anterior, os valores dos atributos {"Distributed query processing in a relational database system"} da entidade e_1 e {"Distrited query processing in a relational database system"} de e_2, bem como os valores dos atributos {"169-180"} de e_1 e {"169-179"} de e_2 possuem valores conflitantes. Para cada agrupamento pode ser útil gerar uma entidade integrada que represente todas as entidades similares, contendo apenas dados integrados que sejam os mais corretos. Portanto, o valor de cada atributo da entidade integrada deve ser escolhido para ser um dos valores encontrados nas entidades similares, para resolver o conflito de valores de atributos. Assim, a resolução de conflitos refere-se ao problema de se resolver conflitos nos valores dos atributos providos por diferentes fontes, também chamada na literatura de fusão de dados [Bleiholder and Naumann 2006, Borges et al. 2008, Benjelloun et al. 2009, Bleiholder and Naumann 2009, Cao et al. 2013]. O foco adotado nesta pesquisa de doutorado é no problema de resolução de conflitos.

2.2 Procedência dos Dados

Quatro aspectos devem ser considerados para o desenvolvimento de modelos de procedência dos dados: quais dados de procedência devem ser coletados, como a coleta é realizada, como deve ser feito o armazenamento desses dados de procedência e, como devem ser consultados os dados de procedência armazenados.

O primeiro deles investiga "quais dados de procedência armazenar", e inclui a definição de quais tipos de dados de procedência devem ser armazenados e qual a granularidade desses dados [Lebo et al. 2012]. Deve-se levar em consideração o custo para armazenar esses dados e os resultados obtidos posteriormente.

Com relação aos tipos de dados, a procedência recebe diferentes classificações na literatura, como *source* e *transformation provenance* [Glavic and Ditt 2007], *process* e *provenance meta-information* [Del Rio and Silva 2007], *prospective* e *retrospectiveprovenance* [Zhao et al. 2006], *where* e *why-*

provenance [Buneman et al. 2001] e *when-how-what* [Widom 2005]. De forma mais específica ou mais genérica, essas classificações referem-se às informações sobre as fontes de dados e sobre os processos de transformação pelos quais os dados foram submetidos.

Com relação à granularidade, os dados sobre a procedência podem ser coletados em diversos níveis de detalhe. Em um banco de dados relacional, por exemplo, eles podem ser armazenados em nível de tabela (maior granularidade), tupla (média granularidade) ou atributo (menor granularidade) [Glavic and Ditt 2007]. O nível de granularidade possui algumas consequências, como o espaço de armazenamento necessário, o qual é inversamente proporcional à granuralidade dos dados coletados, ou seja, quanto menor a granularidade, maior o espaço de armazenamento requerido. Uma outra consequência é o custo da coleta, também inversamente proporcional à granularidade. Uma terceira consequência refere-se à quantidade de consultas que podem ser respondidas sobre a procedência dos dados.

Os dados sobre procedência, de diferentes tipos de dados e granularidades, podem ser obtidos como resultados de operações. A definição das operações é específica de cada aplicação, tal como descrito nas Seções 2.3, 3.1, 3.2.

Após definir os tipos de dados a serem armazenados, deve-se estabelecer uma estratégia para "como coletar os dados de procedência". Essa coleta pode ser feita com a intervenção do usuário, de maneira manual [Buneman et al. 2006b], ou automaticamente [Munroe et al. 2006, Muniswamy-Reddy et al. 2006, Del Rio and Silva 2007, Archer et al. 2009]. A coleta de dados de procedência automática pode ser realizada pela aplicação do usuário, por serviços externos à aplicação ou pelo próprio sistema operacional.

Quanto ao momento em que a coleta é feita, a abordagem *lazy* coleta a procedência de um dado apenas quando ela é requisitada, considerando que existe uma consulta e uma fonte e que elas estão disponíveis no momento da coleta. Já a abordagem *eager* armazena a procedência conforme o dado passa de um sistema para outro ou sofre transformações [Tan 2004]. Na abordagem *lazy* o consumo de memória é baixo pois as informações são calculadas quando requisitadas. No entanto, essa abordagem é restrita, devido à necessidade da fonte e da consulta estarem disponíveis. Na abordagem *eager* é possível recuperar os dados de procedência mesmo que a fonte não esteja disponível e a consulta não tenha sido registrada, pois as informações de procedência são armazenadas conforme os dados são gerados. Entretanto, a abordagem *eager* aumenta a complexidade das aplicações, para se gerar e manter a procedência, além da necessidade de espaço adicional para armazenar esses dados.

Na sequência, é necessário "armazenar os dados de procedência" para torná-los acessíveis futuramente. A procedência pode ser armazenada junto com o dado ao qual ela se refere [Widom 2005],

ou separadamente [Zhao et al. 2006, Buneman et al. 2006a]. Além disso, desde que dados de procedência são volumosos, eles devem ser estruturados visando diminuir o espaço de armazenamento [Buneman et al. 2006a, Chapman et al. 2008, Heinis and Alonso 2008, Anand et al. 2009, Abawajy et al. 2013]. Para otimizar o armazenamento e o gerenciamento, Buneman et al. [Buneman et al. 2006b] introduzem quatro técnicas. A primeira delas, denominada *naïve provenance*, armazena o máximo de dados, sem nenhuma otimização, possibilitando o maior nível de detalhe. A segunda técnica, denominada *transactional provenance*, permite agrupamento de dados de procedência em transações, eliminando dados desnecessários. A terceira técnica, denominada *hierarchical provenance*, conceitua a hierarquia pai e filho e propõe que os dados de um filho podem ser recuperados pela procedência do pai. Dessa forma, não é preciso duplicar os dados para o filho. A quarta técnica, *transactional-hierarchical provenance*, combina a segunda e a terceira, armazenando os dados de procedência de maneira mais concisa, ocorrendo no entanto, perda de dados de procedência.

Além disso, os dados de procedência podem ser armazenados como *logs*, e deve-se considerar questões de versionamento dos dados. Assim, dados antigos devem ser mantidos para que se possa rastrear o histórico de um dado. No entanto, se a granularidade for baixa, o fator de crescimento do banco de dados de procedência também é alto e proibitivo para armazenar dados antigos. Surge então a necessidade de se verificar como deve ser feita a manutenção de dados obsoletos. Por exemplo, é preciso determinar se diante da remoção de um dado, é preciso manter a sua procedência armazenada, quais dados de procedência devem ser mantidos, e por quanto tempo.

Informações de procedência normalmente são usadas pelas próprias aplicações, mas em algumas circunstâncias pode ser necessário expor a procedência aos usuários [Yang et al. 2012]. Seja para uso da aplicação ou do usuário, um último desafio é tornar os dados de procedência disponíveis para consulta, ou seja, definir "como consultar os dados de procedência armazenados". Consultas do tipo rastreamento consistem em consultar os dados e a partir do resultado obtido, verificar a sua procedência (por exemplo, definir como o relatório foi gerado). Consultas do tipo filtro consistem em consultar os dados que satisfazem algum critério de procedência (por exemplo, gerar um relatório com dados advindos apenas de Currículos Lattes) [Huang and Cheng 2011]. Os dois tipos de consulta são complementares e podem existir em um mesmo sistema.

Além dos quatro aspectos descritos para se desenvolver modelos de procedência, pode-se considerar também a representação das informações de procedência. O projeto OPM (*Open Procedence Model*) [Moreau et al. 2011] visa padronizar a representação das informações de procedência. Esse modelo define as entidades *artefato*, *processo* e *agente*, e os relacionamentos entre essas entidades.

Os relacionamentos são definidos para um *artefato* e um *processo*, um *agente* e um *processo*, dois *processos* e dois *artefatos*. Esses relacionamentos são usados para declarar dependências de causalidade entre as entidades definidas no modelo. Um conjunto dessas declarações pode ser usado para construir um grafo de procedência. Um dos principais objetivos do OPM é permitir a troca de informações de procedência entre sistemas. Inferências que podem ser feitas a partir do grafo de procedência também são descritas. Por exemplo, pode-se derivar, usando o grafo de procedência, relacionamentos mais complexos entre processos e artefatos por meio de transitividade.

2.3 O Modelo PrInt

O modelo PrInt (*Provenance model to support Integration processes*) [Tomazela 2010, Tomazela et al. 2010, Tomazela et al. 2013] é um modelo de procedência para subsidiar a integração de dados em sistemas nos quais as fontes de dados podem ser atualizadas somente pelos seus proprietários, impossibilitando que o processo de integração retifique eventuais conflitos de dados diretamente nessas fontes. Como exemplo de fonte proprietária, tem-se as advindas do Currículo Lattes, nas quais é permitido que apenas os proprietários realizem atualizações. Outra característica importante do modelo PrInt é a garantia de 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.

No modelo PrInt, mostrado na Figura 2.1, o usuário interage com o processo de integração para tomada de decisões sobre possíveis inconsistências. Em um primeiro processo de integração, denotado por *ProcIntt1*, fontes heterogêneas são usadas como entrada (Figura 2.1(a)), para as quais o usuário decide como resolver conflitos nos valores dos atributos (Figura 2.1(b)). As ações feitas manualmente pelo usuário para resolução de inconsistências são transformadas em operações de cópia, edição, remoção e inserção, e armazenadas em um repositório de operações (Figura 2.1(c)). O modelo PrInt gera cópias locais das fontes integradas, pois as fontes originais não podem ser atualizadas pelo processo de integração (Figura 2.1(d)). Assim, em um processo de integração subsequente, denotado por *ProcIntSub*, todas as informações de integração obtidas no processo anterior precisam ser reaplicadas automaticamente, para que o usuário não tenha que tomar as mesmas decisões já tomadas anteriormente, já que as fontes originais podem fornecer os mesmos dados inconsistentes (Figura 2.1(e)). Para isso, quando as fontes são passadas para o processo de integração, ele primeiramente invoca o processo de reaplicação de operações para que atualize as fontes de dados, como mostrado na Figura 2.1(f)(g), interagindo a partir desse ponto, com a cópia local das fontes. Dessa

forma, o processo de integração se torna incremental, sendo o *ProcIntSub* apenas para tomada de decisões sobre novas inconsistências originadas nas fontes desde o processo de integração anterior (Figura 2.1(h) e (b)). Como consequência, um menor tempo é gasto no processo de integração.

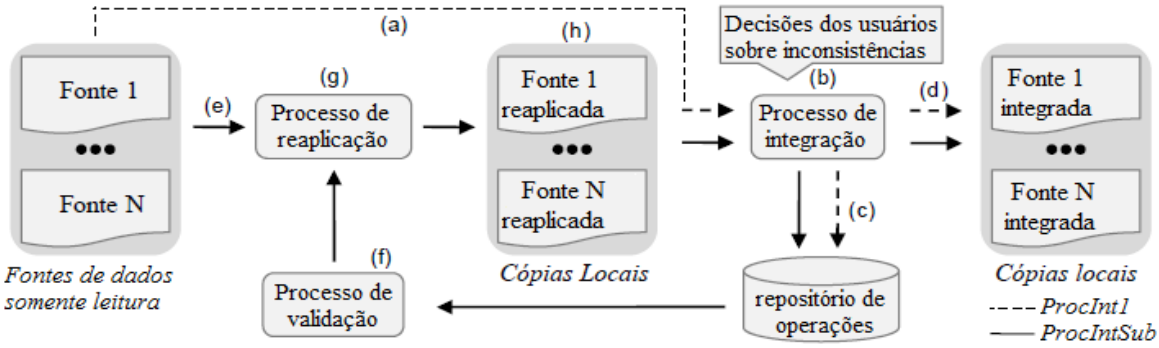


Figura 2.1: Modelo PrInt, adaptado de [Tomazela et al. 2013].

O usuário tem acesso ao processo de integração interagindo com a ferramenta Reconciliador de Dados Acadêmicos [Tomazela et al. 2008]. Essa ferramenta identifica automaticamente e exibe para o usuário as divergências entre as fontes, permitindo ao usuário decidir qual a fonte correta. O detalhamento das operações do modelo PrInt é descrito na Seção 2.3.1. O processo de reaplicação das operações de processos de integração anteriores em processos de integração subsequentes é descrito na Seção 2.3.2. Os aspectos de procedência do modelo são discutidos na Seção 2.3.3.

2.3.1 Operações no Modelo PrInt

O modelo PrInt é baseado nas operações de cópia, edição, inserção e remoção, obtidas por meio de ações do usuário sobre os dados durante o processo de integração. Para essas operações é armazenada a procedência das transformações realizadas sobre os dados.

Considere o cenário em que os currículos de pesquisadores podem ser atualizados apenas por seus proprietários. Cada objeto *Artigo* contém os seguintes atributos: *Título*, *Ano*, *Revista*, *Página inicial*, *Página final*, *ISSN*, *Local publicação*; e o subobjeto *Autores*, composto pelos atributos: *Nome* e *Ordem citação*. O atributo chave é *Título*. Considere que três autores criam seus currículos (fontes), e que eles podem ser distintos ou não, como mostrado na Figura 1.1. Nesse exemplo corrente existem dados inconsistentes entre as fontes. O atributo *Revista*, por exemplo, possui o valor *"Journal of the Brazilian Computer Society"* na fonte *Ana*, *"JBCS"* na fonte *Bruno* e *"Journal of the Brazilian Comp. Society"* na fonte *Carlos*.

Considere agora o repositório de operações mostrado na Figura 2.2, obtido após a integração das fontes. Nesse repositório, cada registro refere-se a uma operação básica, composta pelos seguintes atributos:

id: sequência de números inteiros (I) que identificam os registros do repositório. Dados dois ids de registros, i e j ($i \geq 1$ e $j \geq 1$), $i < j$ se a operação com id i foi executada antes da operação com id j;

op: a operação pode ser uma inserção (*in*) ou remoção (*rm*) de objeto, ou uma simples edição (*ed*) ou cópia (cp) de valor de atributo;

fonteOrigem: fonte da qual se obtém o valor ou objeto, para ser copiado ou inserido na fonte destino. Esse valor é configurado como nulo nas operações de edição e remoção;

fonteDestino: fonte atualizada pela operação;

idObjeto: valor chave que identifica unicamente um objeto no qual a operação é executada;

atributo: atributo do objeto no qual é executada a operação;

valorOrigem: valor prévio do objeto, sobrescrito pelas operações de cópia e edição;

valorDestino: novo valor do objeto.

Uma operação de cópia, representada como cp, é armazenada no repositório após uma ação de cópia do usuário. Como exemplo, as operações com *ids* 2 e 3 na Figura 2.2 representam operações de cópia. Essa ação é feita da fonte considerada correta, chamada de *fonteOrigem*, para a fonte considerada incorreta (*fonteDestino*). A ação de edição é realizada em valores incorretos, independente de estarem consistentes ou não. Ou seja, a edição atua sobre um *fonteDestino*, e o valor do atributo *fonteOrigem* não é utilizado. Por exemplo, as operações com *id* 1 e 4 na Figura 2.2, representam operações de edição. A inserção é a cópia de um objeto presente em apenas uma das duas fontes. Para inserir o objeto faltante, a ação gera no repositório uma operação de inserção (*in*) para o atributo chave e várias operações de cópia (cp), uma para cada atributo não chave do objeto. Nesse caso, os atributos *atributo*, *valorOrigem* e *valorDestino* não são utilizados. As operações com *ids* 11 e 12, representam uma ação de inserção. A ação de remoção (*rm*) remove um objeto incorretamente inserido em uma fonte, ou seja, remove um objeto de um *fonteDestino* sem precisar de uma *fonteOrigem*. Essa ação gera no repositório várias operações de edição (*ed*), modificando o valor de cada atributo não chave para nulo, e uma operação de remoção (*rm*), permitida apenas se todos os atributos não

id	op	fonteOrigem	fonteDestino	idObjeto	atributo	valorOrigem	valorDestino
1	ed	null	Carlos	Artigo[Título=Accord: Asynchronous…]	Título	Artigo[Título=AcCORD: Asynchronous…]	Artigo[Título=Accord: Asynchronous…]
2	cp	Ana	Bruno	Artigo [Título= AcCORD…]	Ano	2013	2015
3	cp	Bruno	Carlos	Artigo [Título= AcCORD…]	Ano	2013	2005
4	ed	null	Carlos	Artigo [Título= AcCORD…] Autor[Nome=Carlos]	Ordem citação	3	2
5	ed	null	Carlos	Artigo [Título= AcCORD…] Autor [Nome=Bruno]	Ordem citação	2	3
6	ed	null	Carlos	Artigo [Título= AcCORD…] Autor [Nome=Daniel]	Ordem citação	null	4
7	rm	null	Carlos	Artigo [Título= AcCORD…] Autor [Nome=Daniel]	null	null	null
8	cp	Bruno	Carlos	Artigo [Título= AcCORD…]	Página final	24	20
9	cp	Bruno	Carlos	Artigo [Título= AcCORD…]	ISSN	01046500	01000000
10	cp	Bruno	Carlos	Artigo [Título= AcCORD…]	Local publicação	Porto Alegre, RS, Brasil	Porto Alegre-RS
11	in	Bruno	Ana	Artigo [Título= AcCORD…] Autor [Nome=Carlos]	null	null	null
12	cp	Bruno	Ana	Artigo [Título= AcCORD…] Autor [Nome= Carlos]	Ordem citação	3	null
13	cp	Carlos	Ana	Artigo [Título= AcCORD…]	Página final	24	20
14	cp	Carlos	Ana	Artigo [Título= AcCORD…]	ISSN	01046500	01046000

Figura 2.2: Repositório gerado pelo modelo PrInt.

chaves forem nulos. Também nesse caso os atributos *atributo*, *valorOrigem* e *valorDestino* não são utilizados. As operações com *ids* 6 e 7, na Figura 2.2, representam uma ação de remoção.

Formalmente, as ações do usuário são mapeadas para operações armazenadas no repositório como descrito a seguir. Considere o conjunto F de fontes e o conjunto V de valores e os elementos f e v tal que $f \in F$ e $v \in V$. Considere também que o conjunto O de objetos é composto por um conjunto A de atributos e um conjunto $O_{so} \subset O$ de subojetos. Os elementos desses conjuntos são representados como $o \in O$ e $a \in A$ e o valor do atributo a no objeto o é denotado por $o.a$. Cada objeto o pode ser unicamente identificado por um conjunto $A_c \subseteq A$ de atributos chave. Os atributos não chave são representados pelo conjunto $A_{nc} \subset A$. Além disso, o conjunto de pares $\{(atributo\text{-}chave = valor)\}$ é representado como A'_c e, dada uma fonte f, $f[A'_c]$ denota o objeto em f identificado por A'_c.

Uma ação de cópia usa como entrada uma fonte f_{origem} e uma fonte $f_{destino}$, os valores dos atributos chaves A'_c do objeto, um atributo a desse objeto, seus valores v_{origem} e $v_{destino}$, sendo mapeada para a operação de cópia no repositório R como descrito na Equação 2.1.

$$\begin{aligned} cp(f_{origem}, f_{destino}, A'_c, a, v_{origem}, v_{destino}) \rightarrow \exists r \in R, id \in I \\ tal\ que\ r = (id, cp, f_{origem}, f_{destino}, A'_c, a, v_{origem}, v_{destino}) \end{aligned} \tag{2.1}$$

Uma ação de edição usa como entrada uma fonte $f_{destino}$, os valores dos atributos chaves A'_c do objeto, um atributo a desse objeto, seu valor corrente $v_{destino}$ e um novo valor inserido pelo usuário $v_{usuario}$, sendo mapeada para uma operação no repositório R como mostrado na Equação 2.2.

$$\begin{aligned} ed(f_{destino}, A'_c, a, v_{destino}, v_{usuario}) \rightarrow \exists r \in R, id \in I \\ tal\ que\ r = (id, ed, null, f_{destino}, A'_c, v_{usuario}, v_{destino}) \end{aligned} \tag{2.2}$$

Uma ação de inserção usa como entrada uma fonte f_{origem} e uma fonte $f_{destino}$, os valores dos atributos chaves A'_c do objeto, sendo mapeada como uma operação de inserção e várias cópias para os atributos não chave do novo objeto criado com os valores obtidos da fonte f_{origem}, como descrito na Equação 2.3.

$$\begin{aligned} &in(f_{origem}, f_{destino}, A'_c) \rightarrow \exists r \in R, id \in I\, tal\ que\ r = (id, in, f_{origem}, f_{destino}, A'_c, null, null, null) \\ &\qquad\qquad e \\ &\forall a \in A_{nc}\, \exists r' \in R, id' \in I\, tal\ que\ r' = (id', cp, f_{origem}, f_{destino}, A'_c, a, f_{origem}[A'_k].a, null) \end{aligned} \tag{2.3}$$

Uma ação de remoção usa como entrada uma fonte $f_{destino}$, os valores dos atributos chaves A'_c do objeto, sendo mapeada como várias operações de edição para definir como *null* todos os atributos não chave do objeto a ser removido e uma operação de remoção, como mostrado na Equação 2.4.

$$rm(f_{destino}, A'_c) \rightarrow \forall a \in A_{nc}\, \exists r' \in R, id' \in Ital\, que\, r' = (id', ed, null, f_{destino}, A'_c, a, null, v_{destino})$$
$$e$$
$$\exists r \in R, id' \in Ital\, que\, r = (id, rm, null, f_{destino}, A'_c, null, null, null) \tag{2.4}$$

Relacionamentos entre as operações

As operações armazenadas no repositório podem se relacionar de duas maneiras: transitividade e alteração de chave. Uma operação *b* é transitiva à operação *a* quando *b* depende do resultado de *a* e *a* ocorreu antes de b. A transitividade pode ser direta, quando o *destino* da operação *a* é igual à *origem* da operação *b*, e as operações são realizadas sobre o mesmo *atributo*. No repositório da Figura 2.2, a operação com *id 3* é transitiva direta à operação com *id 2*, e a operação *13* é transitiva à operação *8*. A transitividade também pode ser indireta. Nesse caso, a operação *a* é transitiva direta à operação *b*, e a operação *b* é transitiva direta à operação *c*, assim, a operação *a* é transitiva indireta à operação *c*.

Suponha a existência de um currículo extra no nosso exemplo corrente, criado pelo autor Daniel, e mostrado na Figura 2.3. A inserção da operação com *id* 15, mostrada na Figura 2.4, gera uma transitividade indireta, sendo a operação 8 transitiva direta à operação 13, e a operação 13 transitiva direta à operação 15. Assim, a operação 8 é transitiva indireta à operação 15.

A operação de alteração de chave ocorre quando uma operação altera a chave de um objeto, como a operação com *id 1* na Figura 2.2.

Outra característica importante tratada pelo modelo PrInt refere-se às operações de sobreposição. Inconsistências podem ser geradas quando uma operação inserida sobrepõe o resultado de outra operação já existente. Quando um atributo é considerado incorreto e alterado em uma operação *b*, sendo que o mesmo já foi considerado incorreto e alterado em uma operação *a*, ocorre uma sobreposição de destino. Considere a operação de *id* 16, mostrada na Figura 2.5. Essa operação sobrescreve o resultado da operação de *id* 2, gerando uma sobreposição do destino. Quando o atributo de uma primeira fonte é considerado incorreto em uma operação *a*, e reconciliado com o atributo correto de uma segunda fonte, e posteriormente, em uma operação *b* esse mesmo atributo da segunda

Fonte: Daniel

Artigo

Título: “AcCORD: Asynchronous COllaborative data ReconcIliation moDel”

Ano: “2003”

Revista: “Journal of the Brazilian Computer Society”

Página inicial: “5”

Página final: “15”

ISSN: “01046500”

Local publicação: “Porto Alegre, RS, Brasil”

Autores

Autor

Nome: “Ana”

Ordem prioridade: “1”

Autor

Nome: “Bruno”

Ordem prioridade: “2”

Autor

Nome: “Carlos”

Ordem prioridade: “3”

Autor

Nome: “Daniel”

Ordem citação: “4”

Figura 2.3: Fonte Daniel.

id	op	fonteOrigem	fonteDestino	idObjeto	atributo	valorOrigem	valorDestino
...							
15	cp	Ana	Daniel	Artigo[Título=AcCORD...]	Página Final	24	15

Figura 2.4: Operação com *id 15*.

fonte é considerado incorreto e alterado, ocorre uma sobreposição da origem. Considere a inserção da operação com *id* 17 no repositório, mostrada na Figura 2.6. A sua inserção gera uma sobreposição

da operação com *id* 9, e assim, além da operação com *id* 9, a operação com *id* 14 transitiva à operação com *id* 9 também fica inconsistente.

id	op	fonteOrigem	fonteDestino	idObjeto	atributo	valorOrigem	valorDestino
...							
16	ed	null	Bruno	Artigo[Título=AcCORD...]	Ano	2009	2015

Figura 2.5: Operação com *id 15*.

id	op	fonteOrigem	fonteDestino	idObjeto	atributo	valorOrigem	valorDestino
...							
17	ed	null	Bruno	Artigo[Título= AcCORD...]	ISSN	01046555	01046500

Figura 2.6: Operação com *id 16*.

Para o tratamento de sobreposições, o modelo PrInt descreve a política *Redo*, em que as operações sobrepostas por sobreposição de origem são movidas para o final do repositório, as operações sobrepostas por sobreposição de destino são removidas do repositório, e todas as operações transitivas são refeitas e movidas para o final do repositório.

Com a inserção da operação 16 e a utilização da política *Redo*, a operação 2 sobreposta no destino, é removida do repositório, e o valor de data do atributo *valorOrigem*, na operação 3, transitiva à 2, é alterado de *"2004"* para *"2009"* e essa operação movida para o final do repositório. Assim, a operação fica consistente com a operação 15, deixando o repositório consistente. A inserção do operação 16 provoca, utilizando a mesma política, a modificação do atributo *valorOrigem* nas operações 9 e 14 de *"01046500"* para *"01046555"*, e o deslocamento dessas para o final do repositório. Assim, a consistência do repositório é mantida.

2.3.2 Reaplicação Provida pelo Modelo PrInt

Quando se inicia um novo processo de integração, é preciso verificar se as fontes possuem os mesmos dados desde que as operações foram definidas, antes de se iniciar o processo de reaplicação. Caso contrário, anomalias podem ser geradas.

Uma anomalia possível é a anomalia das fontes consistentes, em que uma fonte considerada correta em uma operação é alterada pelo autor, ficando consistente com a fonte considerada incorreta. Então, a reaplicação da operação faz as fontes ficarem inconsistentes. Com uma alteração no valor do atributo *Local publicação*, de *"Porto Alegre, RS, Brasil"* para *"Porto Alegre-RS"* na fonte *Bruno* (Figura 1.1), a reaplicação da operação 10 (Figura 2.2) faz as fontes *Bruno* e *Carlos* ficarem inconsistentes. Assim, essa operação deve ser removida do repositório.

Outra anomalia é a anomalia do destino sobrescrito, em que a fonte considerada incorreta pelo usuário em uma operação é alterada pelo autor, mas continua inconsistente com a fonte considerada correta. A reaplicação vai tornar as fontes consistentes, mas o usuário não vai tomar conhecimento da alteração realizada no destino. Uma alteração no valor do atributo *Ano* da fonte *Bruno*, de *"200"* para *"2007"* provoca, ao reaplicar a operação 2, uma anomalia do destino sobrescrito pois as fontes *Ana* e *Bruno* ficam consistentes, mas o usuário não toma conhecimento da alteração.

Esse processo de verificação para determinar se as fontes possuem os mesmos dados desde que as operações foram definidas até o momento da reaplicação, é chamado de validação. A validação feita com fontes usadas como *origem* nas operações é chamada de validação da origem e evita a anomalia das fontes consistentes. A validação realizada com o *destino* nas operações é chamada de validação do destino e evita a anomalia do destino sobrescrito. O modelo PrInt realiza essas validações para garantir uma reaplicação restrita das operações válidas.

Para reaplicar as operações, o modelo PrInt introduz o métod VRT (*Validate and Reapply in Tandem*), no qual as operações são reaplicadas em ordem temporal, ou seja, o modelo PrInt reaplica as operações a cada validação de origem e de destino, como pode ser visto na Figura 2.7. Dessa forma, garante-se que operações transitivas sejam reaplicadas na mesma ordem em que foram definidas. As consultas realizadas por esse método são do tipo filtro, para recuperação das operações e de cada fonte envolvida.

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.

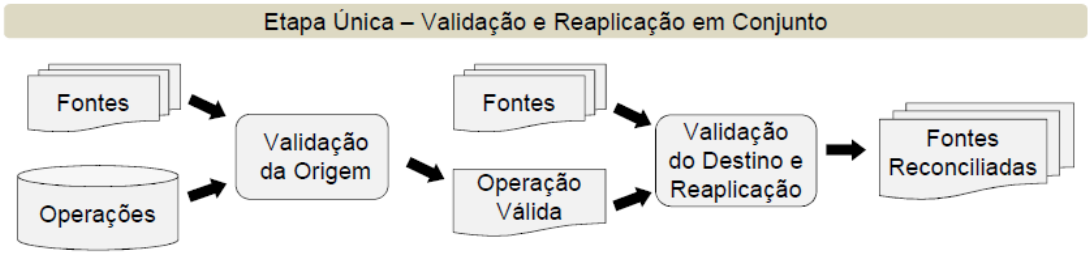


Figura 2.7: Estágios do método VRT [Tomazela 2010].

2.3.3 Aspectos de Procedência

Com relação aos quatro aspectos para desenvolvimento de modelos de procedência, descritos na Seção 2.2, o modelo PrInt possui as características descritas a seguir:

- Quanto aos dados que devem ser coletados, o modelo PrInt leva em consideração os atributos que formam a procedência de cada operação de cópia, inserção e remoção, usadas no processo de integração. Essas operações são armazenadas no mesmo repositório e possui o mesmo conjunto de atributos. A origem dos dados de cada operação é armazenada, bem como as alterações realizadas, que são rastreadas e armazendas, caracterizando a procedência como *why* e *where-provenance* [Buneman et al. 2001].

- Quanto à coleta dos dados de procedência, o modelo PrInt a realiza de maneira automática, tratando as operações que se sobrepõem, como forma de manutenção.

- O modelo PrInt armazena a procedência separada dos dados, em um repositório de operações, pois o processo de integração não tem permissão para escrever nas fontes e portanto, informações de procedência também não podem ser inseridas.

- O modelo emprega os dois tipos de consulta, por filtro e por rastreamento, no desenvolvimento de métodos de reaplicação de operações.

2.4 Considerações Finais

Neste capítulo foram abordados conceitos relacionados à integração de dados e às características e aspectos que devem ser considerados no desenvolvimento de modelos de procedência. Também foi descrito o modelo PrInt, um modelo de procedência de dados para auxiliar processos de integração.

Os conceitos descritos neste capítulo são usados no trabalho desenvolvido nesta tese de doutorado da seguinte forma. O modelo proposto, AcCORD, enfoca a integração de dados em nível de instância, considerando o problema de resolução de conflitos. Adicionalmente, o modelo AcCORD é baseado no uso de procedência dos dados. Por fim, o modelo AcCORD visa o uso de dados de procedência na forma de *log* e, portanto, utiliza como base o repositório de operações do modelo PrInt. As principais diferenças entre o modelo proposto e o modelo PrInt são descritas a seguir. Apesar do modelo PrInt resolver vários desafios discutidos anteriormente, ele não oferece suporte a um ambiente multiusuário. Além disso, ele não vislumbra a propagação automática de decisões de integração tomadas pelo usuário dentro de um mesmo processo de integração. Outra limitação do modelo PrInt é que ele não utiliza nenhum critério de confiabilidade para recomendar ao usuário a fonte ou valor de um dado mais adequado, nem solucionar um conflito. O modelo AcCORD, em contrapartida, não possui essas limitações.

No Capítulo 3 são descritos e comparados trabalhos correlatos ao modelo AcCORD, os quais tratam da integração e compartilhamento de dados.

Capítulo 3

Trabalhos Correlatos voltados à Integração Colaborativa ou Compartilhamento de Dados

Diversos trabalhos são encontrados na literatura na área de integração de dados e compartilhamento ou sincronização de dados, sejam baseados em operações ou em estados. No entanto, dentre esses trabalhos, poucos enfocam no uso da procedência dos dados no processo de integração ou compartilhamento. E, a maioria dos trabalhos que tratam a reconciliação ou sincronização oferecem suporte às aplicações colaborativas assíncronas, mas não enfocam a resolução de conflitos. Devido a isso, neste capítulo são descritos os trabalhos que visam a proposta de modelos de integração ou compartilhamento de dados, usando dados de procedência para auxiliar no processo.

Este capítulo está estruturado da seguinte forma. Na Seção 3.1 é descrito o sistema de compartilhamento de dados colaborativo Orchestra. Na Seção 3.2 é detalhado o sistema Youtopia, o qual é voltado à integração de dados e ao gerenciamento colaborativo. Na Seção 3.3 é descrito o BeliefDB, um banco de dados com anotações de confiança. Na Seção 3.5 são listadas as limitações desses trabalhos e discutidas as motivações para o desenvolvimento do modelo AcCORD.

3.1 Orchestra

O sistema Orchestra [Green et al. 2007, Ives et al. 2008, Taylor and Ives 2010] é voltado para compartilhamento de dados estruturados, de autoria colaborativa, sem a necessidade de padronização de esquemas. Diferentemente dos bancos de dados que impõem restrições globais, tais como um único esquema global, consistência global das instâncias de dados e administração centralizada, o

sistema Orchestra caracteriza-se por ser *peer-to-peer* e não requer servidor central. Assim, cada par no sistema Orchestra controla uma instância local do banco de dados, podendo operar em modo "desconectado" por um período. As edições realizadas localmente são armazenadas e, quando o administrador desse par decide realizar uma troca de atualização, as atualizações locais são publicadas, as atualizações feitas nos demais pares desde a sua última reconciliação são buscadas e mapeadas para o esquema local.

Como as atualizações importadas podem ser conflitantes entre si, e também com as atualizações locais, um processo de reconciliação deve ser realizado. Para isso, é feito um rastreamento da procedência das atualizações e uma filtragem de acordo com a política de confiança local. As atualizações não conflitantes são aplicadas. As atualizações que conflitam com as atualizações locais são rejeitadas e, para cada grupo de atualização não local conflitante, o par recebe a de maior prioridade. Se duas atualizações conflitantes têm a mesma prioridade, elas são adiadas para futura intervenção do usuário. Assim que o par possuir uma instância de dados com as mudanças confiadas incorporadas, as atualizações feitas localmente podem modificar os dados importados.

As características do sistema Orchestra podem ser agrupadas em módulos [Ives et al. 2008], descritos em mais detalhes nas Seções 3.1.1, 3.1.2, 3.1.3 e 3.1.4, as quais tratam, respectivamente, da publicação de dados gerados localmente por um par, dos mapeamentos entre diferentes esquemas, da filtragem de atualizações externas baseada na política de confiança e modificações realizadas localmente e, da reconciliação diante de conflitos existentes nas atualizações advindos de pares distintos. Os estágios para compartilhamento de dados são mostrados na Figura 3.1, segundo a visão de um par.

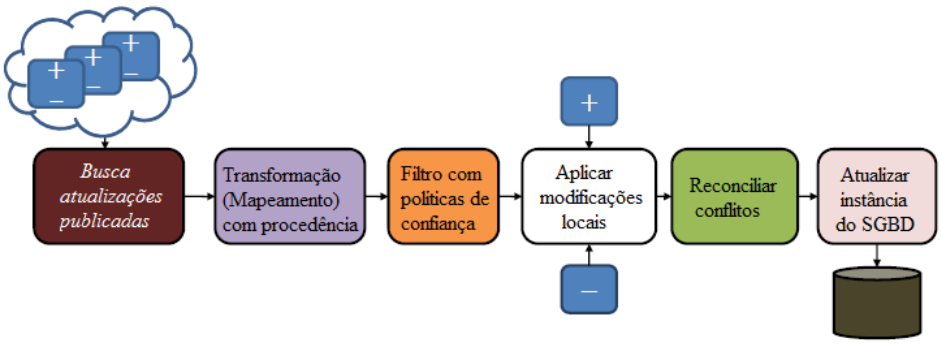


Figura 3.1: Modelo de compartilhamento de dados do sistema Orchestra (adapatado de [Ives et al. 2008]).

3.1.1 Publicando e Arquivando *Logs*

Para que um par compartilhe dados com outros pares no sistema Orchestra, deve-se primeiramente publicar os dados locais. Assim, primeiramente é gerado um *log* com as mudanças locais feitas nos dados. Essas atualizações são armazenadas de modo descentralizado, baseada em replicação *peer-to-peer*, possibilitando que as atualizações sejam armazenadas dentro de transações, e as dependências entre as transações sejam registradas. Para isso, cada transação recebe um *timestamp* global, de acordo com o momento em que ela é publicada. Qualquer dado publicado passa a fazer parte de um registro permanente, garantindo que eles estejam disponíveis no sistema.

Para particionar e replicar os dados entre todos os pares correntemente disponíveis é usada uma Tabela *Hash* Distribuída. Assim, não é necessário nenhum servidor central. Adicionalmente, uma vez que as máquinas no sistema podem ser substituídas, os dados são migrados automaticamente para as novas máquinas.

3.1.2 Busca e Mapeamento de Atualizações

Depois de publicar os seus dados, um determinado par que inicia o processo de troca de atualização busca todas as atualizações de outros pares não vistas desde o último processo de troca de atualização. As atualizações importadas são mapeadas usando mapeamento de esquemas, pois nem todas estão definidas sob o mesmo esquema, usando-se os mesmos identificadores, e as etapas do mapeamento são gravadas como procedência de dados.

Para realizar o mapeamento de esquemas, o sistema Orchestra usa tgds (*tuple generating dependencies*). Por meio de seu uso, são feitas restrições entre as instâncias de dados. Uma tgd é uma asserção lógica do tipo

$$\forall x, y\,(\Phi(x,y) \longrightarrow \exists z\,\Psi(x,z)) \tag{3.1}$$

expressando a existência de tuplas do lado direito, dada uma combinação particular de tuplas satisfazendo uma consulta sobre o lado esquerdo [Ives et al. 2008]. Tgd é um meio comum de especificar restrições e mapeamentos em compartilhamento de dados, e pode ser vista como mapeamento GLAV (*global-local-as-view*), que por sua vez, pode generalizar as formulações de mapeamentos GAV (*global-as-view*) e LAV (*local-as-view*).

Como exemplo, considere os pares X, Y e Z, que possuem esquemas descrevendo periódico, nome do autor e título de publicações, da seguinte forma: X(periodico,autor,titulo), Y(periodico,autor) e Z(autor,titulo). Entre esses pares existem os seguintes mapeamentos:

$$m1 \quad X(periodico, autor, titulo) \longrightarrow Y(periodico, autor)$$

$$m2 \quad X(periodico, autor, titulo) \longrightarrow Z(autor, titulo)$$

$$m3 \quad Y(periodico, autor) \longrightarrow \exists titulo\, Z(autor, titulo)$$

$$m4 \quad Y(periodico, titulo) \wedge Z(autor, titulo) \longrightarrow Y(periodico, autor)$$

Os três primeiros mapeamentos têm um único par, fonte e objetivo, correspondendo ao lado esquerdo e ao direito da implicação. O mapeamento $m3$ possui uma variável existencial $titulo$, cujo valor é desconhecido e não necessariamente único. Relações de múltiplos pares podem ocorrer tanto no lado esquerdo quanto no direito, como é o caso do mapeamento $m4$, o qual define a relação Y baseada em seus próprios dados combinados às tuplas de Z.

Tgds são usadas como uma chamada de procedimento para computar uma instância de dados a partir da qual todas as respostas certas para uma consulta podem ser obtidas, ou seja, uma solução canônica universal. Essa solução é uma MVT (*Multi-Viewpoint Table*) [Ives et al. 2005], englobando o conjunto de possíveis instâncias do banco de dados. No entanto, quando se computa uma solução canônica universal para uma instância do par local, um desafio pode ser encontrado porque a instância pode conter informação incompleta. Isso ocorre porque alguma fonte pode não prover todos os atributos. Em alguns casos, pode ser conhecido que dois (ou mais) valores são realmente o mesmo, apesar desse valor ser desconhecido. Tais casos são, então, representados usando variáveis existenciais no objetivo de um mapeamento tgd, como por exemplo, no mapeamento $m3$. Em procedimentos estilo sequência de busca, esses valores são representados como nulo.

Uma vez que o sistema Orchestra realiza consultas Datalog, funções de Skolem são usadas para representar esses valores desconhecidos. Portanto, para traduzir os mapeamentos, o sistema Orchestra usa uma versão estendida de *Datalog* com suporte para funções de Skolem. Essas funções substituem as variáveis existenciais, como $titulo$. Uma vantagem das funções de Skolem sobre os valores nulos tradicionais é que o mesmo valor Skolem pode ser usado em mais de um lugar, permitindo ao processador de consultas determinar quais tuplas ele vai juntar ou fundir. Assim, para o mapeamento $m3$, a variável $titulo$ poderia assumir valores diferentes, resultando em instâncias distintas.

3.1.3 Filtrando Atualizações e Realizando Modificações Locais

O compartilhamento de dados requer que cada par disponha não somente de uma relação com os dados fontes, mas também um conjunto de atualizações locais, ou seja, a inserção de novos dados e a remoção de dados importados. Para isso, o *log* de atualizações locais de cada par é minimizado,

removendo as inserções e remoções que se anulam [Ives et al. 2005]. Por exemplo: (i) se um dado A é inserido duas vezes, é considerada apenas uma inserção; (ii) se uma atualização insere um dado A, e uma atualização posterior modifica A para B, é considerada uma inserção do dado B e; (iii) se uma atualização insere um dado A, uma atualização posterior modifica A para B, e outra atualização modifica B para C, é considerada uma inserção do dado C.

As atualizações locais de cada relação R são divididas em duas tabelas lógicas: uma de contribuições locais, Rl, incluindo as inserções de dados, e uma de rejeições locais, Rr, incluindo as remoções de dados externos. As regras do *Datalog* são atualizadas para R, adicionando um mapeamento de Rl para R, com a condição $\neg Rr$. Assim, $R = Rl + \neg Rr$.

As inserções e remoções são então propagadas, usando os dados de procedência. No caso de uma remoção, por exemplo, é preciso determinar se tuplas antes vistas ainda são deriváveis caso algumas tuplas bases sejam removidas [Green et al. 2007]. Assim, o modelo de procedência deve identificar não somente as fontes que contribuíram para a geração do dado, mas também como as tuplas são derivadas, incluindo alternativas de derivação separadas entre diferentes mapeamentos.

A procedência é criada e mantida como grafo durante o processo de troca de atualização, como mostrado na Figura 3.2, e usada para avaliar a confiança de uma atualização [Green et al. 2007]. No grafo de procedência, os nós tuplas são representados como retângulos e os nós mapeamentos são representados como elipses. Já os nós 3D, na figura, representam inserções na instância local. Um nó tupla é criado para cada tupla e, no caso dos nós mapeamentos, vários nós podem ser rotulados com o mesmo mapeamento. Tuplas inseridas diretamente pelo usuário na instância local são rotuladas com identificadores globais únicos de procedência. Uma aresta que possui como antecessor um nó tupla e como sucessor um nó mapeamento representa uma entrada. Já uma aresta que possui como antecessor um nó mapeamento e como sucessor um nó tupla, representa uma derivação. Como exemplo, analisando a procedência da tupla *Y(3,2)* na Figura 3.2, nota-se que ela é derivada dos pares *p1* e *p2*, usando como base o mapeamento *m4* e do par *p3* usando como base o mapeamento *m1*.

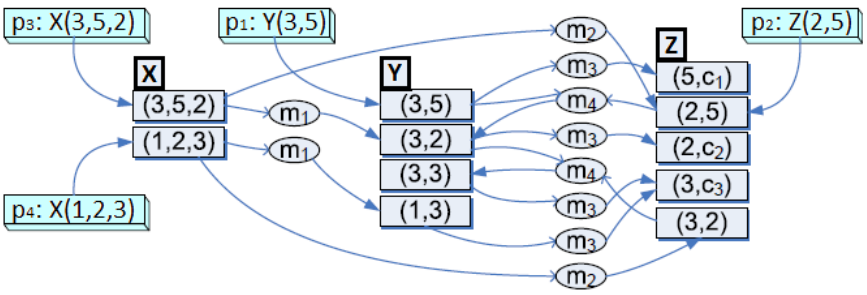


Figura 3.2: Grafo de procedência (adaptado de [Green et al. 2007, Ives et al. 2008]).

As atualizações importadas e mapeadas são filtradas baseando-se na procedência, de acordo com a política de confiança local, pois os pares podem não considerar todas as atualizações com igual autoridade ou qualidade. A política de confiança especifica, para cada par, condições sobre os dados e procedência dos dados, e associam uma prioridade a cada atualização. Uma prioridade 0, por exemplo, especifica uma atualização não confiável que é filtrada durante a troca de atualização.

Qualquer nova modificação feita por usuários locais são adicionalmente consideradas, formando um conjunto de atualizações candidatas. Essas atualizações são agrupadas dentro de transações, e a prioridade da transação é determinada de acordo com a prioridade de suas atualizações. Assim, se qualquer atualização membro de uma transação não é confiável, a transação também não é confiável, e cada transação recebe a maior prioridade de confiança dentre as suas atualizações.

3.1.4 Reconciliação

Transações podem gerar dependências de dados. Como resultado, o processo de reconciliação arbitra entre as possíveis atualizações para determinar um conjunto consistente para aplicar à instância local do banco de dados [Taylor and Ives 2006]. Para isso, o funcionamento do processo de reconciliação ocorre da seguinte maneira [Ives et al. 2008]:

- Todo par recebe um conjunto de atualizações de acordo com suas próprias políticas;
- As atualizações não conflitantes são inseridas;
- Para cada transação conflitante, o par aceita aquela de maior prioridade;
- Se duas transações incompatíveis têm a mesma prioridade, o usuário deve intervir;
- A decisão do usuário pode ser adiada e o sistema continua reconciliando dados que não interagem com transações adiadas;
- As reconciliações subsequentes não podem voltar as instâncias atualizadas no estágio anterior;
- As transações são consideradas em ordem de prioridade, da mais alta para mais baixa;
- Se uma transação for aplicada e depender de uma anterior, a anterior também é aplicada;
- Se qualquer transação em série não puder ser aplicada devido a uma restrição, então nenhuma transação dependente é aplicada.

Ou seja, o conjunto de transações ainda não vistas é recuperado do armazenamento de atualizações, e uma prioridade é atribuída para cada transação. Em ordem decrescente de prioridade, encontra-se a transação com maior prioridade que deve ser aplicada, juntamente com qualquer antecedente necessária para satisfazer dependências de leitura-escrita e escrita-leitura.

3.2 Youtopia

O sistema Youtopia [Kot and Koch 2009] é uma plataforma para gerenciamento colaborativo e integração de dados relacionais. Sua principal característica é uma abstração de troca de atualizações. Ele apresenta um modelo para propagação de mudanças que combina uma sequência de buscas determinística pelos mapeamentos com a intervenção do usuário. Assim, mudanças nos dados são propagadas pelo sistema para satisfazer mapeamentos especificados pelo usuário. Os mapeamentos são feitos usando tgds e ciclos nos mapeamentos são permitidos sem comprometer a corretude.

O sistema Youtopia permite aos usuários inserir, registrar, atualizar e manter dados relacionais em um modo colaborativo. O módulo gerenciador de armazenamento provê a abstração lógica de um repositório, que consiste de um conjunto de tabelas lógicas ou visões contendo os dados. Esses dados são ligados por meio de um conjunto de mapeamentos fornecidos pelos usuários. Assim, do ponto de vista lógico, o Youtopia é um sistema de troca de dados que contém quatro módulos principais, conforme descrito na Figura 3.3. Nessa figura também são resumidas as funcionalidades de cada módulo. Troca de dados é definido como um modelo adequado para sistemas de integração de dados colaborativos, e caracterizado por um alto nível de dinamismo e mudança nos dados e nos mapeamentos. O modelo de troca de dados introduzido pelo sistema Youtopia visa maximizar a cooperação por melhor esforço. Sua abordagem geral é otimista, ou seja, permite novas atualizações para o conjunto de sequências de buscas enquanto velhas sequências de buscas estão esperando assistência humana. O termo *"sequência de buscas"* utilizado neste capítulo refere-se ao estilo de propagação no qual os valores nos mapeamentos são buscados e atualizados sequencialmente.

O sistema Youtopia realiza a propagação de mudanças nos dados contidos nas relações por meio de um conjunto de mapeamentos ou tgds (Seção 3.2.1). Como dados e mapeamentos podem ser inseridos no repositório pelos usuários, inconsistências podem ser geradas e, caso sejam geradas, são corrigidas usando-se sequências de buscas *forward* e *backward* (Seção 3.2.2). Sequências de operações de modificação no banco de dados causadas por uma única operação do usuário são chamadas de atualizações (Seção 3.2.3). Para maximizar a usabilidade do sistema, várias consultas e pesquisas podem ocorrer ao mesmo tempo, e a interferência entre atualizações é coordenada usando-

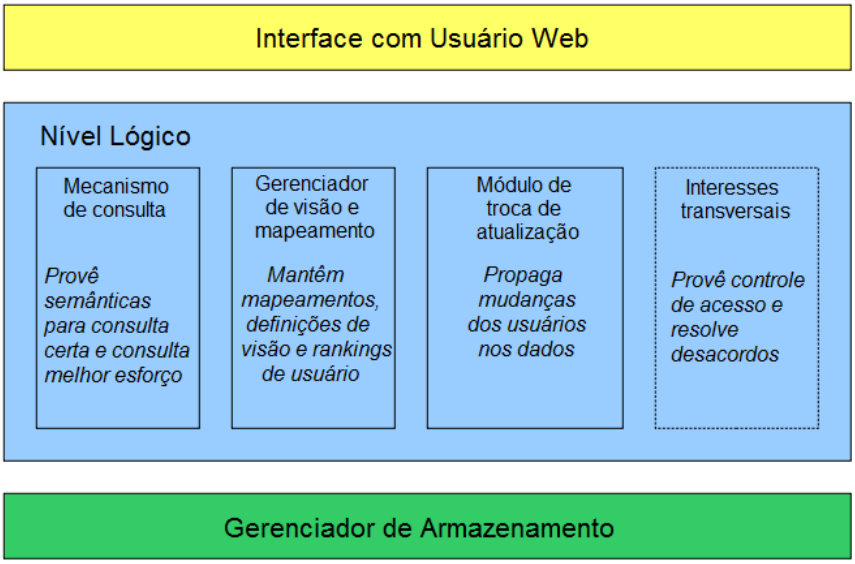


Figura 3.3: Arquitetura do sistema Youtopia (adapatado de [Kot and Koch 2009]).

se técnicas de seriabilidade e controle de concorrência (Seção 3.2.4). O sistema Youtopia pode ser usado na coordenação de dados de usuários, realizada por meio de consultas emaranhadas (Seção 3.2.5).

3.2.1 Troca de Atualização

A propagação de mudanças ocorre por meio do processo de sequência de busca usando tgds. Em especial, o sistema Youtopia permite aos usuários estabelecer e refinar as tgds. Essa criação de mapeamentos é facilitada por um sumário de visões. Usuários experientes podem definir quais visões capturam em seu esquema. Tais visões podem orientar os proprietários de tabelas a inserir seus mapeamentos.

Como dados e mapeamentos são adicionados ao repositório, conflitos são inevitáveis. Para tanto, o sistema Youtopia provê mecanismos para resolução de conflitos pela comunidade e, além disso, oferece suporte para inconsistência de dados em casos em que os conflitos não são resolvidos. Normalmente, na sequência de busca padrão com tgds, o sistema deve corrigir quaisquer violações nos mapeamentos, assim que elas ocorrem, e sem pedir informações adicionais. No entanto, usuários do sistema podem não querer que as violações sejam corrigidas imediatamente, especialmente se não estiverem usando a parte relevante do repositório no momento.

Uma violação em um mapeamento tgd pode ocorrer quando uma tupla é inserida, removida ou modificada pelo usuário não satisfazendo às relações entre os mapeamentos. Toda violação de mapeamento é associada com uma testemunha, ou seja, o conjunto de tuplas do banco de dados que corresponde ao lado esquerdo da tgd, mas não tem um conjunto correspondente de tuplas no lado direito.

3.2.2 Reparando Violações

O sistema Youtopia usa uma variante no procedimento padrão de sequência de buscas para corrigir violações de mapeamento. Assim, três tipos de operações podem iniciar uma sequência de buscas: inserção de tupla, remoção e substituição de *null*. A substituição de *null* refere-se à substituição de todas as ocorrências de um atributo *null* pelo mesmo valor constante. Se uma violação é causada por uma inserção ou substituição de *null*, isso significa que a nova versão da tupla é parte da testemunha da violação. Esse tipo de violação é chamada de violação do lado esquerdo. Caso a violação seja causada pela remoção de uma tupla, isso significa que a tupla excluída era usada para refletir sobre o lado direito alguns valores e assim, nenhuma correspondência à tupla do lado direito pode ser encontrada. Esse tipo de violação é chamada de violação do lado direito.

A correção das violações é realizada por dois tipos de sequências de buscas: *forward* e *backward*. A sequência de busca *forward* corrige violações inserindo no banco de dados a tupla faltante do lado direito para a testemunha. A sequência de busca *forward* corrige as violações por meio da remoção de pelo menos uma das tuplas testemunhas. A escolha do tipo de pesquisa é detectada pelo tipo de violação. Violações do lado esquerdo são reparadas por sequências de buscas *forward*, enquanto que violações do lado direito por sequências de buscas *backward*.

A inserção de tupla não é a única maneira de corrigir uma violação do lado esquerdo. É possível fornecer uma tupla para o lado direito, unificando algumas variáveis (atributos marcados como nulos) com outros valores no banco de dados. Como os usuários têm domínio do conhecimento, eles podem auxiliar na correção das violações. Assim, o modelo de pesquisa do sistema Youtopia é cooperativo e os ciclos nos mapeamentos são permitidos e não causam problemas.

Quando o processo padrão de sequência de buscas *forward* para, é o momento do usuário intervir e auxiliar no processo. O usuário tem acesso a todas as tuplas que foram geradas mas não inseridas banco de dados, chamadas de *tuplas de fronteira positivas*. Diante de uma tupla fronteira t, o usuário pode executar uma das duas *operações de fronteira* descritas a seguir:

- *Expandir* t, ou seja, inserir t no banco de dados;

- *Unificar* t, ou seja, escolher alguma tupla t' que é mais específica que t e executar a unificação da variável entre quaisquer *null* de t e t'. A tupla t desaparece depois da operação.

Ao contrário da sequência de buscas *forward*, uma sequência de buscas *backward* sempre termina, pois não se pode remover mais tuplas do que existe no banco de dados. No entanto, a resolução da sequência de buscas *backward* também requer assistência do usuário para resolver qual tupla do lado esquerdo (caso haja mais de uma) deve ser removida para corrigir uma violação. O sistema Youtopia reconhece que uma das tuplas deve ser removida e marca cada uma dessas tuplas como *tupla de fonteira negativa*, mas não toma a decisão, deferindo-a ao usuário. Diante de um conjunto de tuplas de fronteira negativa, o usuário pode desempenhar a *operação de fronteira negativa*, removendo qualquer subconjunto de tuplas. A execução de uma pesquisa consiste, portanto, de uma sequência determinística, separada por um período de bloqueio e espera por operações de fronteira.

3.2.3 Atualizações

Uma única operação do usuário pode propagar no sistema Youtopia em muitos passos. Assim, uma sequência completa de operações de modificação no banco de dados causada por uma única inserção, remoção ou substituição por *null* inicial é chamada de *atualização*. Isso inclui todas as operações de fronteira realizadas pelo usuário. Uma atualização é positiva se a operação inicial foi uma inserção ou um preenchimento com *null*, e negativa se a operação foi de remoção.

3.2.4 Seriabilidade e Controle de Concorrência

Conforme foi apresentado, o sistema Youtopia requer a assistência humana para a tomada de decisões. Apesar da assistência humana poder ser lenta, o sistema Youtopia foi projetado para minimizar o impacto de usabilidade pelo usuário, e permanece utilizável o maior período de tempo possível. As mudanças são, então, permitidas de maneira assíncrona. Assim, consultas e sequências de buscas não são bloqueadas por sequências de buscas anteriores que possuem violações e estão pendentes de intervenção humana. Como múltiplas pesquisas ocorrem ao mesmo tempo no sistema, é possível que haja interferências, afetando a corretude. Sendo assim, a serialização é usada para prevenir ou detectar e sinalizar para que haja uma correção humana.

A seriabilidade é definida no Youtopia em termos de escalonamentos. Um escalonamento é uma sequência alternada de conjuntos de operações de escrita e de leitura, mas não o contrário. Assim, a serialização de um escalonamento x, denotado por $S(x)$, é um escalonamento obtido após ordenar as

operações em x no número de atualizações, em ordem crescente, mantendo a ordem entre as operações de cada atualização.

A noção de seriabilidade introduzida para coordenar a interferência entre atualizações é baseada em estado final. Ou seja, suponha que existam k atualizações executando no banco de dados. Um escalonamento é serializável por estado final se existe uma ordenação para as atualizações em que o estado final do banco de dados (após executar o escalonamento) é o mesmo estado caso as atualizações sejam executadas sequencialmente. Assim, o escalonamento é serializável para uma dada ordenação se ele é verdadeiro para a ordenação específica desejada.

Essa definição de seriabilidade é geral e aberta à execução de uma grande variedade de soluções. A serialização por estado final é normalmente aplicada para manter as propriedades mais fortes, como a serialização de conflitos. Dois escalonamentos são serializáveis quanto ao conflito se são equivalentes quanto ao conflito, ou seja, a ordem das operações conflitantes é a mesma nos dois escalonamentos.

A abordagem para controle de concorrência no sistema Youtopia também baseia-se em protocolos tradicionais de multiversões, como o protocolo baseado em aborto MVTO (*Multiversion Timestamp Ordering*) e o protocolo baseado em bloqueio MV2PL (*Multiversion Two Phase Locking*). Esses protocolos tradicionais são adaptados para produzir uma solução para aplicar a seriabilidade no modelo de troca de atualização. Como qualquer técnica de controle de concorrência, a alta vazão possui um custo: algumas sequências de buscas podem ser interrompidas e reiniciadas se um conflito ocorrer. Assim, são propostas também, técnicas para reduzir o número de sequências de pesquisas interrompidas.

3.2.5 Coordenação

Além das características descritas anteriormente, o sistema Youtopia possui uma estrutura social, permitindo aos usuários estabelecer uma rede de amigos de confiança para que dados, mapeamentos e pontos de vista definidos pelo usuário possam ser compartilhados em níveis variáveis. Logo, o sistema Youtopia pode ser usado em aplicações que realizam coordenação [Gupta et al. 2011a, Gupta et al. 2011b].

Essa coordenação é oferecida utilizando-se consultas emaranhadas. Isso significa que uma resposta para uma consulta é retornada usando respostas relacionadas compartilhadas por várias consultas no sistema. Uma consulta com uma condição não satisfeita não é rejeitada, ela é registrada no sistema para uma possível execução posterior.

A aplicação desenvolvida pelos autores, que justifica o oferecimento da funcionalidade de coordenação, é um *site* de viagens que permite aos usuários coordenar suas viagens e reservas em hotéis com seus amigos do Facebook, usando a API Expedia.

3.3 BeliefDB

O sistema BeliefDB [Gatterbauer et al. 2009a, Gatterbauer et al. 2009b] é baseado no uso de um banco de dados com tuplas base anotadas com declarações de confiança. Seu objetivo é compartilhar dados em um ambiente no qual uma comunidade de usuários pode criar, revisar e curar um repositório compartilhado. A falta de consenso sobre quais dados estão corretos permite que cada usuário insira as tuplas que acreditam no banco de dados e as razões pelas quais acredita que o banco de dados deva conter tais tuplas. As anotações podem se referir tanto aos dados do banco quanto às outras anotações, e são interpretadas como declarações de confiança. Anotações são metadados adicionais referentes aos dados existentes sem qualquer semântica distinta.

O sistema BeliefDB considera conflitos em nível de instância nos valores dos atributos dentro de um mesmo esquema e, portanto, não investiga o problema de integração de esquemas. Diferentemente de outros sistemas para compartilhamento de dados, como o sistema Orchestra, em que cada par aceita ou rejeita mudanças feitas por outros pares, e cada um possui sua própria visão dos dados em uma instância do banco de dados separada, o sistema BeliefBD permite que as informações conflitantes coexistam em um único banco de dados, e que seus usuários discutam esses conflitos.

Introduz-se, então, o conceito de banco de dados de confiança (*Belief Database*), em que as informações na forma de tuplas são anotadas com declarações de confiança, representando um conjunto de mundos de confiança diferentes, cada um para um tipo de anotação de confiança. Os tipos de anotações de confiança podem ser as confianças de um usuário em particular em um grupo de tuplas, ou anotações de confiança de outros usuários. Um mundo de confiança contém um conjunto de confianças positivas e negativas. Confiança negativa acontece quando um usuário discorda sobre um fato ou anotação, mas não tem sugestão alternativa.

Esses mundos de confiança seguem a hipótese de mundo aberto, e podem conter sobreposições e conflitos entre os dados dos usuários (ou seja, não é necessário satisfazer restrições de chave), diferentemente da hipótese de mundo fechado seguida por bancos de dados convencionais, em que cada tupla que não está no banco de dados é considerada negada. Assim, um mundo de confiança W é representado por duas instâncias de banco de dados, uma contendo as tuplas com anotações de

confiança positivas e outra contendo tuplas com anotações de confiança negativas, da forma $W = (I^+, I^-)$.

A confiança de uma tupla t é positiva para o mundo W se, e somente se, t está na instância de positivos I^+ e, é negativa se, e somente se, está presente explicitamente na instância de negativos I^- ou se existe uma tupla t' na instância positiva com a mesma chave, ou seja,

$$W \models t^+ \; sse \; t \in I^+$$

$$W \models t^- \; sse \; t \in I^- \vee \exists t' \in Tup_i.(t' \neq t \wedge t' \in I^+ \wedge chave(t') = chave(t))$$

Com as seguintes restrições para a consistência do mundo de confiança:

$$\Gamma_1(W) \equiv \Gamma(I^+)$$

$$\Gamma_2(W) \equiv \forall t \in I^+ : t \notin I^-$$

em que Γ_1 é a restrição de chave padrão sobre I^+, e Γ_2 requer que $I^+ \cap I^- = \emptyset$.

Dessa forma, um banco de dados de confiança é uma coleção de mundos de confiança, representando um conjunto de instâncias incompleto e consistente. Dependendo de quais tuplas são ou não compartilhadas, quaisquer duas instâncias podem ser mutuamente disjuntas, sobrepostas, contidas ou parcialmente conflitantes. Especificamente, um banco de dados de confiança é um conjunto de anotações de confiança, no qual as anotações de confiança são definidas em termos de lógica multimodal. Ou seja, uma declaração de confiança é uma expressão da forma

$$\Box_w t^s,$$

em que w é um caminho de confiança não vazio pertencente ao conjunto de usuários que não contêm o mesmo identificador de usuário em posições consecutivas, ou seja, $w \in \hat{U}^*$, $\hat{U}^* = \{w \in U^* | w_{[i]} \neq w_{[i+1]}\}$ e U é o conjunto de identificadores dos usuários representado como $U = \{1, 2, ..., n\}$; t é uma tupla do universo de tuplas e; s é um sinal que indica a confiança positiva ou negativa na tupla, ou seja, $s \in \{'+', '-'\}$.

A lógica multimodal pode ser representada por axiomas e também por estrutura Kripke canônica. Essa estrutura contém estados (nós) associados ao mundo de confiança, e arestas ou relações de acessibilidade associadas com cada usuário, que permitem rastrear a confiança nos dados. Além disso, essa estrutura pode ser representada em modelos relacionais e usada em SGBDs (Sistemas Gerenciadores de Banco de Dados) relacionais padrão, com consultas conjuntivas de confiança traduzidas para a representação relacional, e atualizações no banco dados padrão.

3.4 Hossain et al. 2014

Hossain *et al.* [Hossain et al. 2014] desenvolvem um modelo de sincronização de dados, no qual a resolução de conflitos de atualizações usa o processo automático e a intervenção do usuário. O modelo é aplicado ao contexto de sistemas de saúde colaborativos, os quais permitem o compartilhamento de dados entre provedores de serviço de saúde, tais como médicos, hospitais e laboratórios, a fim de permitir tratamentos colaborativos. Nesses ambientes, os prestadores de serviços são autônomos e, portanto, curam, revisam e estendem os dados compartilhados independentemente. Para colaboração completa, os prestadores de serviços trocam as atualizações de dados. Assim, atualizações de dados (inserção, remoção ou modificação) em um prestador de serviços, isto é, fonte de dados, podem por sua vez, afetar os dados dos outros provedores de serviços.

A sincronização de dados nesse modelo é realizada colaborativamente usando três técnicas, a saber: sincronização automática, sincronização semiautomática e sincronização com envolvimento do usuário. Essas técnicas são resumidas a seguir:

- Na sincronização automática, as fontes determinam a ordem de execução das atualizações conflitantes sem qualquer envolvimento do usuário. A sincronização automática é realizada usando a regra de resolução por absorção.

- Na abordagem semiautomática, a sincronização de dados é realizada por meio da colaboração entre as fontes e o envolvimento dos usuários. Essa abordagem é usada quando duas atualizações são executadas no mesmo número de fontes e a abordagem automática falha para resolver o conflito. A sincronização, nesse caso, é realizada usando a regra de resolução mútua.

- Na abordagem para a sincronização com intervenção do usuário, os usuários estão diretamente envolvidos na resolução de conflitos de atualizações. Nesse caso, os conflitos são resolvidos por superusuários em algumas situações críticas nas quais a abordagem automática não pode encontrar uma decisão para resolução de conflitos entre atualizações.

As abordagens propostas são aplicáveis em sistemas que toleram inconsistências nos dados por um certo período de tempo e uma atualização para um dado em uma fonte não é imediatamente importante em outras fontes. Assim, a *consistência eventual* é garantida. Como as atualizações são executadas localmente e independentemente, protocolos de confirmação (*commit*) não são necessários, desde que eles introduzem bloqueios e, portanto, não são viáveis. No sistema, as atualizações são realizadas localmente e propagadas de maneira assíncrona para fontes conhecidas.

Uma execução consistente de atualizações pode ser obtida assegurando a mesma ordem de execução de atualizações conflitantes para cada fonte conhecida durante a propagação de atualizações. A fim de manter a consistência nos dados em fontes conhecidas, o seguinte procedimento é adotado. Se não há conflito no tempo em que as atualizações são iniciadas em uma fonte, então as atualizações podem ser executadas em qualquer ordem nas fontes conhecidas da fonte sem criar qualquer inconsistência.

Uma abordagem otimista é usada para executar atualizações nas fontes de dados, a fim de permitir o acesso contínuo aos dados durante a execução da atualização. Essa abordagem permite ao usuários ler e atualizar o banco de dados enquanto eles estão desconectados e sincronizam os dados com outras fontes quando eles são reconectados. Assim, as atualizações são propagadas de maneira assíncrona em segundo plano, sem bloquear qualquer requisição de leitura. Os passos para resolver conflitos e sincronizar atualizações são os seguintes:

1. Quando uma fonte detecta um conflito entre duas atualizações que são recebidas de outras fontes, então a fonte interrompe a execução das atualizações e pergunta ao seu pai sobre a ordem de execução. A fonte que propaga uma atualização é a fonte pai e a fonte que recebe a atualização é chamada de filha.

2. Se o pai não tem informação sobre a execução, ele pergunta ao seu pai. Esse processo continua até que uma fonte que conhece a ordem de execução seja encontrada ou a pergunta chegue aos iniciadores da atualização. A primeira fonte que detectou o conflito no mesmo par de atualizações pode já ter propagado a pergunta para os iniciadores e a ordem de execução pode já estar decidida. Então, outras fontes que detectam o mesmo conflito devem receber o resultado de qualquer fonte intermediária ao longo do caminho até o iniciador.

O processo de selecionar a ordem de execução de duas operações conflitantes pelos iniciadores ou por uma fonte intermediária que tem a informação sobre a ordem faz uso de três protocolos de resolução: resolução por absorção, resolução mútua e resolução envolvendo o usuário.

Resolução por absorção A partir da mensagem de conflito, cada iniciador sabe quantas fontes executaram as atualizações desde que a mensagem de conflito percorreu o caminho a partir das fontes onde o conflito foi detectado até os iniciadores. Essa contagem é tratada como nível de execução de uma atualização. O nível de execução de uma atualização A_i é denotado como $nivel(A_i)$. Depois de receber a informação de conflito, ambos os iniciadores aceitam a ordem $< A_2, A_1 >$ se $nivel(A_2) > nivel(A_1)$, caso contrário, a ordem é $< A_1, A_2 >$. Considerando que a

ordem $< A_2, A_1 >$ é selecionada, uma atualização de compensação A_1^- é gerada e enviada para F_i. Quando F_i recebe a informação sobre a ordem e a atualização de compensação, F_i executa A_1^- e executa as atualizações A_1 e A_2 na ordem $< A_2, A_1 >$.

Resolução mútua Se $nivel(A_1) = nivel(A_2)$, então a ordem é determinada com o entendimento mútuo. Depois de receber a informação sobre o conflito, iniciadores aceitam uma ordem específica. O processo de compensação começa como descrito na resolução por absorção.

Resolução envolvendo o usuário Se um conflito é detectado nas fontes S_i e S_j, os superusuários nas fontes decidem a ordem de execução das atualizações. Os usuários são informados sobre um conflito quando um conflito é detectado pelo sistema. Se os usuários não podem decidir a ordem, então eles consultam os usuários dos pais das atualizações. Após o conflito ser resolvido, o processo de compensação começa como descrito na resolução por absorção.

Os autores consideraram como um fator de conflito na avaliação de desempenho do sistema, a razão entre o número de atualizações envolvidas em conflito e o número de atualizações ativas no sistema. Os resultados mostraram que o tempo de execução aumentou com o aumento no número de atualizações e do fator de conflito mantendo o número de fontes fixo, bem como com o aumento no número de fontes e do fator de conflito mantendo o número de atualizações fixo. Os autores também realizaram testes de aceitabilidade com o usuário, medindo a percepção de consistência, de aceitabilidade, de corretude e de risco e concluíram que as abordagens semiautomática e com envolvimento do usuário tiveram maior aceitação. Isso ocorre porque os usuários querem alguma automação que forneça algum benefício para eles, mas se preocupam com os riscos e o menor controle sobre a funcionalidade do sistema.

3.5 Considerações Finais

Neste capítulo foram descritos os trabalhos existentes na literatura para integração e compartilhamento de dados de maneira colaborativa, especialmente quando usam procedência para auxiliar no processo de integração ou compartilhamento. Esses trabalhos são correlatos ao trabalho desenvolvido nesta pesquisa de doutorado. Os sistemas Orchestra, Youtopia, BeliefDB e aquele proposto por Hossain *et al.* foram detalhados individualmente. Esses sistemas introduzem várias características, tais como mapeamento de esquemas, arquivos delta contendo as mudanças locais realizadas nos dados, uso de políticas de confiança, intervenção do usuário para auxiliar no processo de reconciliação dos

dados quando as políticas se mostram ineficazes, e interrupção e reinicialização de uma sequência de atualizações devido a um conflito, quando essas atualizações ocorrem em um modo síncrono.

O objetivo desta tese de doutorado é propor o modelo AcCORD, um modelo colaborativo assíncrono para a reconciliação de dados, no qual as atualizações feitas pelos usuários são armazenadas em repositórios. Repositórios mantêm a procedência dos dados, ou seja, as operações aplicadas nas fontes de dados que levaram o banco de dados local ao seu estado corrente. Adicionalmente, nesta tese são propostas diferentes políticas para resolver possíveis conflitos que resultam do processo de reconciliação multiusuário colaborativo e assíncrono. Essas políticas podem ser destinadas às aplicações que geram uma visão integrada ou distintas visões locais como resultado do trabalho colaborativo. As políticas são baseadas nas seguintes estratégias: remoção das operações realizadas por diferentes usuários cujas decisões conflitam entre si; priorização das decisões tomadas pelo próprio usuário; ordem cronológica da tomada da decisão; priorização da decisão tomada pela maioria dos usuários; confiança nas fontes de dados; e confiança nos usuários que tomaram as decisões. Outra contribuição desta tese consiste em um método de propagação em nível de um único usuário para evitar que o usuário tenha que tomar a mesma decisão de integração para o mesmo conflito de valor de atributo de um objeto em diferentes fontes, economizando seu tempo. Esse método usa os dados de procedência para retificar as fontes de dados que estão incorretas, com base no processo de integração.

Na Tabela 3.1 são mostradas as características de cada trabalho correlato comparativamente às características do modelo AcCORD. A primeira delas refere-se ao sincronismo, e nesse caso, todos os sistemas comparados trabalham em modo assíncrono. Essa característica é essencial para o desenvolvimento da maioria das políticas de resolução de conflitos desde que os valores conflitantes são reconhecidos de uma só vez e, em seguida, o resultado é calculado. Em sistemas síncronos, as operações são processadas uma após a outra, proporcionando um resultado parcial na resolução do conflito que considera apenas os valores existentes até o momento. Assim, os sistemas síncronos permitem apenas políticas que não precisam processar todos os dados de entrada, e políticas como a *política baseada em votação* introduzida pelo modelo AcCORD não seriam possíveis.

A segunda e a terceira características referem-se, respectivamente, à integração de esquemas e à integração de instâncias. O modelo AcCORD não investiga a integração de esquemas. Ele considera que essa funcionalidade já foi previamente realizada. Com relação à integração de instâncias, o modelo AcCORD investiga a resolução de conflitos nos valores dos atributos. Essa resolução de conflitos nos valores de atributos é tratada tanto em nível multiusuário, por meio da aplicação das políticas propostas, quanto em nível de único usuário, por meio do método de propagação proposto. Isso garante

ao modelo AcCORD completude de propagação das decisões de integração quando comparado com os demais trabalhos correlatos.

Com relação à quarta característica, o modelo AcCORD introduz seis políticas que se adaptam tanto ao ambiente de compartilhamento quanto ao de integração cooperativa de dados, de acordo com a conveniência do usuário. Assim, o modelo AcCORD permite tanto a geração de visões locais com diferentes valores para os atributos conflitantes dos usuários, quanto a obtenção de um valor único para cada atributo conflitante. Diferentemente do modelo AcCORD, o sistema Orchestra sempre mantém os dados conflitantes por permitir apenas múltiplos pontos de vista e, assim, os conflitos não são resolvidos, mas reconhecidos e escondidos das consultas dos usuários. O sistema Youtopia propaga as mudanças nos dados fazendo sequências de buscas nos mapeamentos tgds e reparando possíveis violações na consistência. No entanto, tanto a inserção de tuplas quanto as correções nas violações requer a assistência do usuário. Assim, o sistema Youtopia gera um conjunto de tuplas que podem ser inseridas ou unificadas, e reconhece quais as tuplas que devem ser removidas e as marcam, mas as decisões são deferidas para o usuário. Isso gera um um período de bloqueio a espera da resolução de quais tuplas devem ser inseridas, unificadas ou removidas. Já no BeliefDB, os conflitos entre os dados não são resolvidos, mas é permitido que cada usuário insira as tuplas que acreditam no banco de dados e anotações de confiança. No modelo de sincronização de dados de Hossain et al., os conflitos entre as atualizações são resolvidos determinando-se a ordem de execução dessas atualizações automaticamente ou, com a intervenção do usuário. Por oferecer um conjunto variado de políticas para a resolução de conflitos, o modelo AcCORD é mais adaptável às diferentes aplicações. Além disso, o modelo AcCORD oferece um método para propagar as atualizações feitas pelo usuário, de modo que ele possa economizar o tempo que seria utilizado para retificar a mesma inconsistência em outra fonte de dados. Esse método também resolve inconsistências não detectadas pelas políticas de reconciliação devido ao fato da consistência ser definida sobre as fontes de origem e destino em comum. Assim, inconsistências nos dados de um mesmo objeto, cujas origens e destinos nas operações não se sobrepõem são detectadas e corrigidos apenas pelo método de propagação, que é aplicado em modo de um único usuário. Para realizar tais mudanças e tornar os dados consistentes, o modelo AcCORD utiliza um repositório que representa os dados de procedência, e nenhum arquivo delta é utilizado. Assim, operações feitas previamente podem interferir em resultados posteriores, bem como serem modificadas a fim de se manter a consistência.

A última característica refere-se ao uso da procedência de dados, a qual é presente apenas nos sistemas Orchestra e BeliefDB e no modelo AcCORD. Para realizar o processo de reconciliação assíncrono multiusuário, o modelo AcCORD utiliza um repositório que armazena os dados de pro-

cedência referentes às decisões de integração tomadas pelo usuário. Essas decisões de integração são coletadas e armazenadas automaticamente no repositório e são gerenciadas pelo modelo AcCORD sem a necessidade de interferência do usuário no sistema. Em contrapartida, o sistema Orchestra baseia-se no uso de arquivos delta. Assim, é necessário que cada fonte forneça suas atualizações desde o último processo de reconciliação, ou seja, cada fonte deve exportar seu arquivo delta em relação à última troca de dados. Portanto, a aplicabilidade de Orchestra é limitada uma vez que há um certo número de fontes que não são capazes de fornecer o arquivo delta. Já o sistema BeliefDB armazena metadados adicionais na forma de anotações de confiança. Nesse sistema, os usuários compartilham dados utilizando um banco de dados centralizado, no qual inserem tuplas quando acreditam que o banco de dados deva conter tais tuplas, juntamente com as razões pelas quais acredita nisso, chamadas de anotações de confiança. As anotações podem se referir tanto aos dados do banco quanto às outras anotações, e assim o usuário pode interferir diretamente nos dados armazenados no repositório. Além disso, diferentemente do modelo AcCORD, o objetivo do BeliefDB não é resolver as inconsistências entre os valores dos atributos, mas compartilhar dados em um ambiente colaborativo, no qual os usuários podem discutir criando, revisando e curando um repositório compartilhado.

Tabela 3.1: Principais características do trabalhos correlatos

Característica/Sistema	Orchestra	Youtopia	BeliefDB	Hossain et al. 2014	AcCORD
Sincronismo	assíncrono	assíncrono	assíncrono	assíncrono	assíncrono
Integração de esquemas	sim	sim	não	não	não
Integração de instâncias	não	resolução de conflitos	não	resolução de conflitos	resolução de conflitos
Número de políticas de integração/compartilhamento	1	1	1	1	6
Procedência dos dados	sim	não	sim	não	sim

No Capítulo 4 são detalhados o modelo AcCORD, as políticas de resolução de conflitos e compartilhamento de dados e o método de propagação, os quais foram desenvolvidas durante este projeto de doutorado.

Capítulo 4

Modelo AcCORD: um modelo colaborativo assíncrono para a reconciliação de dados

Nesta tese, propõe-se o modelo AcCORD, um modelo colaborativo e assíncrono de reconciliação de dados. O objetivo do processo de reconciliação provido pelo modelo AcCORD consiste tanto em gerar uma única visão consistente e integrada para todos os usuários, quanto em prover visões distintas para cada um desses usuários, de acordo com as suas necessidades particulares. Isso significa que, quando conflitos entre os dados são detectados, permite-se que todos os usuários concordem com o valor correto para o dado em um *processo de integração colaborativo*, ou permite-se que eles discordem, mas que trabalhem cooperativamente para *compartilhar* as suas decisões. Para atingir esse objetivo, o modelo AcCORD mantém as atualizações feitas pelos usuários individualmente em um repositório de operações na forma de dados de procedência. Cada usuário tem o seu próprio repositório para armazenar a procedência e a sua própria cópia das fontes. Ou seja, quando inconsistências entre fontes importadas são detectadas, cada usuário pode tomar decisões para resolvê-las de maneira autônoma, e as atualizações que são executadas localmente são registradas em seu próprio repositório. As atualizações são compartilhadas entre os colaboradores importando cada repositório dos demais usuários. Desde que usuários podem ter diferentes pontos de vista, os repositórios podem estar inconsistentes. Assim, nesta tese também são propostas políticas para resolver conflitos entre repositórios. Políticas distintas podem ser aplicadas por diferentes usuários para reconciliar as suas atualizações. Dependendo da política aplicada, a visão final das fontes importadas pode ser a mesma para todos os usuários, ou seja, um única visão global integrada, ou resultar em distintas visões locais para cada um deles. Além disso, para auxiliar os usuários individualmente, no processo de integração em nível de um único usuário, o modelo AcCORD também incorpora um *método de propagação de*

decisões de integração para evitar que o usuário tenha que tomar a mesma decisão para o mesmo atributo de um objeto em diferentes fontes. Esse método usa o repositório de dados do usuário para retificar as fontes de dados que estão incorretas, com base nas decisões já tomadas para outras fontes.

Este capítulo está estruturado da seguinte forma. Na Seção 4.1 é descrito o cenário de reconciliação de dados no qual o modelo AcCORD atua. Na Seção 4.2 são descritas as características necessárias ao repositório para a aplicação das políticas de reconciliação de dados. Na Seção 4.3 são discutidas as seis políticas propostas para a reconciliação de dados do modelo AcCORD. Na Seção 4.4 investiga-se o *método de propagação de decisões de integração* em nível de único usuário. O capítulo é finalizado na Seção 4.5 com as considerações finais.

4.1 Cenário Proposto

O cenário considerado no modelo AcCORD é mostrado na Figura 4.1. Nesse cenário, usuários colaborativos $U_1, U_2, ..., U_n$ reconciliam dados importados de várias fontes de dados. Cada usuário decide de forma independente e autônoma como resolver conflitos de valores de atributos nos objetos correspondentes de acordo com o seu ponto de vista (Figura 4.1 (a)). Para esse fim, cada usuário pode usar ferramentas para ajudá-lo no processo de integração e para armazenar as atualizações em um repositório de dados de procedência baseado em operações. Neste trabalho, usa-se como ferramenta o modelo PrInt, descrito na Seção 2.3. Como resultado, cada usuário produz um conjunto de visões locais, que são versões integradas das fontes, e um repositório, que contém dados de procedência composto de uma sequência de operações que reflete suas atualizações. Como vários usuários colaborativos trabalham sobre as mesmas fontes, eles mantêm sua própria visão local, e as fontes originais não são atualizadas.

Durante o processo de integração, cada usuário pode ser auxiliado pelo *método de propagação de decisões de integração* (Figura 4.1 (b)). Para cada conflito de dados que compreende mais de duas fontes, esse método permite que o usuário tome uma decisão integração apenas uma vez. Então, o método propaga automaticamente a decisão para as outras fontes e atualiza o repositório do usuário. Para esse fim, o *método de propagação de decisões de integração* obtém a partir do repositório os dados de procedência em nível de único usuário que se referem aos dados do processo de integração, busca as fontes de dados para identificar o mesmo conflito de dado em outras fontes, e armazena no repositório os dados de procedência em nível de único usuário contendo cada decisão de integração propagada. Como resultado da aplicação do método proposto, cada processo de integração torna-se

menos propenso a erros e menos demorado, e garante uma decisão integração consistente para os mesmos dados em conflito.

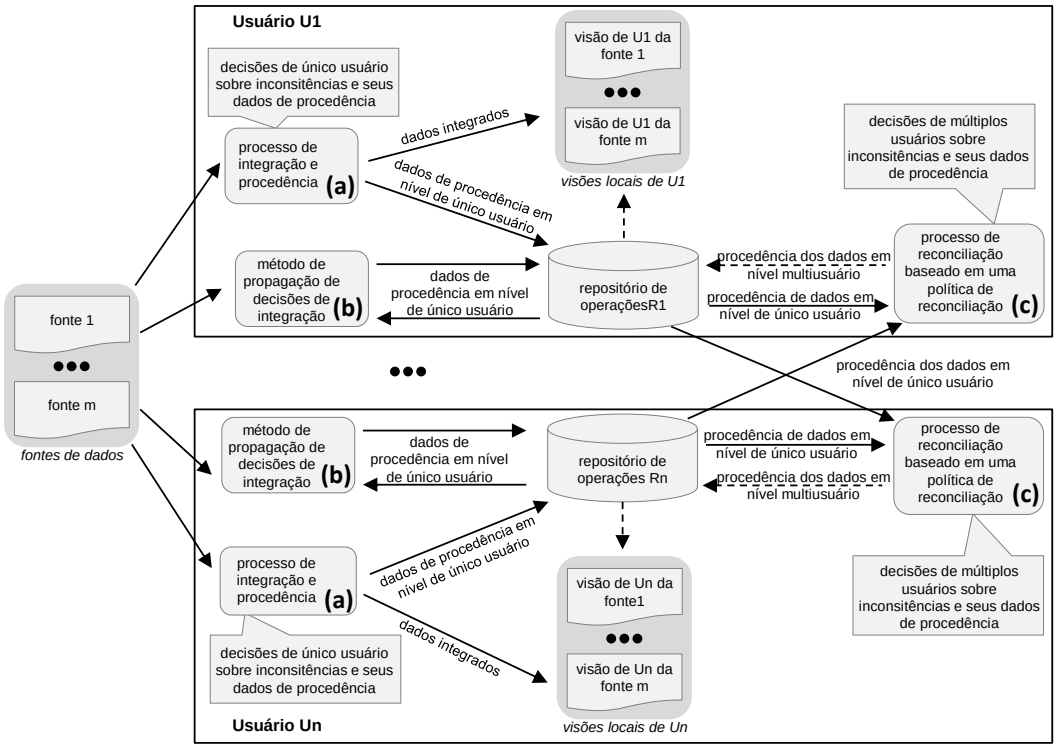


Figura 4.1: O modelo AcCORD.

Depois de terminar os processos de integração e procedência, um usuário, digamos o usuário U_1, decide reconciliar suas atualizações com as atualizações de outros usuários colaborativos (Figura 4.1 (c)) para, assim, realizar o processo de reconciliação considerando as atualizações de múltiplos usuários, que é o foco desta pesquisa de doutorado. Note que o processo de reconciliação realizado pelo usuário U_1 atualiza apenas as operações no repositório do usuário U_1, o processo de reconciliação realizado pelo usuário U_2 atualiza apenas as operações no repositório do usuário U_2, e assim por diante. Note que, no modelo AcCORD, denota-se o processo realizado por um único usuário como um *processo de integração* de valores de atributos, e o processo realizado sobre as decisões tomadas por múltiplos usuários como um *processo de reconciliação*, desde que o primeiro sempre gera um única visão integrada e o segundo pode gerar a mesma visão integrada para todos os usuários, ou distintas visões locais para cada um deles. O processo de reconciliação proposto recebe como entrada vários conjuntos de dados de procedência, cada um representando decisões que um único usuário tomou para resolver inconsistências sobre as fontes, e produz como saída um conjunto de dados de procedência representando a reconciliação de atualizações de todos os usuários. Nós chamamos cada entrada

como “dados de procedência em nível de único usuário”, e a saída como “dados de procedência em nível multiusuário”.

Os dados de procedência em nível multiusuário produzidos pelo processo de reconciliação dependem da política de reconciliação de dados aplicada para resolver conflitos entre atualizações de usuários. Essa política é usada para atualizar o repositório de operações do usuário que está executando a reconciliação, ou seja, o usuário U_1 em nosso exemplo corrente. São propostas nesta tese diferentes políticas, detalhadas nas Seções 4.3.1 a 4.3.6. Os colaboradores podem escolher a mesma política de reconciliação de dados ou diferentes políticas de reconciliação, dependendo do resultado final pretendido. Se os usuários estão trabalhando colaborativamente para obter uma única visão integrada dos dados, então eles devem compartilhar das mesmas atualizações, e então aplicar a mesma política de reconciliação de dados. Por outro lado, quando os usuários querem compartilhar suas atualizações particulares, mas desejam manter sua própria visão dos dados reconciliados, cada um deles pode aplicar diferentes políticas de reconciliação de dados. Adicionalmente, um dado usuário pode decidir não realizar em nível multiusuário, trabalhando somente em nível de único usuário.

Depois de atualizar o repositório de operações, o usuário pode aplicar um modelo de procedência baseado em operações para atualizar suas visões locais de acordo com os novos dados de procedência armazenados no repositório, como o modelo PrInt.

Antes de introduzir as políticas de reconciliação, são detalhadas na Seção 4.2 as características do repositório e restrições aplicadas sobre as operações.

4.2 Repositório de Operações

O modelo AcCORD é baseado em um repositório de operações que armazena dados de procedência. Um repositório é uma sequência de registros, sendo que cada registro contém os seguintes atributos:

- **usuário**: identificador único para o usuário que tomou a decisão e criou a operação;
- **op**: operação que reflete uma decisão do usuário em um processo de integração;
- **usuárioConf**: usuários que confiaram na operação em processos de reconciliação anteriores;
- **origem**: fonte que provê o valor correto de um atributo de uma entidade;
- **destino**: fonte na qual o valor do atributo foi atualizada por *op*;

- **objeto**: valor da chave que identifica uma entidade;
- **atributo**: nome do atributo no qual *op* é realizada;
- **valorOrigem**: valor do atributo em *origem*;
- **valorDestino**: valor do atributo em *destino* antes de ser sobrescrito por *valorOrigem*;
- ***timestamp***: instante de tempo da execução de *op*.

Dos atributos anteriormente listados, o modelo proposto AcCORD requer a existência dos atributos *usuário*, *usuárioConf* e *timestamp*. Os demais atributos dependem do modelo de integração usado como base. Nesta tese de doutorado, foi usado o modelo PrInt, o qual introduz os demais atributos listados, conforme descrito na Seção 2.3.

Operações em um repositório podem conter dependências em nível de único usuário ou conflitos em nível multiusuário. A Definição 1 detalha as dependências entre as operações em nível de único usuário.

Definição 1 (Dependências entre operações)**.** *Uma operação b é dependente de uma operação a quando elas são realizadas pelo mesmo usuário, a ocorre antes de b, elas envolvem o mesmo atributo do mesmo objeto e o destino de a é igual a origem de b. Ou seja,*

- *a(usuário,* ***destino****, objeto, atributo)* $=$ *b(usuário,* ***origem****, objeto, atributo) e* $a < b$.

Além disso, uma operação c é dependente de uma operação a se existe uma operação b que é dependente de a e c é dependente de b. Ou seja,

- *a(usuário,* ***destino****, objeto, atributo)* $=$ *b(usuário,* ***origem****, objeto, atributo),* $a < b$, *e*
- *b(usuário,* ***destino****, objeto, atributo)* $=$ *c(usuário,* ***origem****, objeto, atributo), e* $b < c$.

Intuitivamente, uma operação b depende de a se o valor escrito por a é depois usado por b, propagando-o para outras fontes.

Os exemplos listados e explicados ao longo deste capítulo utilizam como base o exemplo introduzido na Seção 1.1. Para facilitar a leitura do texto, a Figura 4.2 reproduz a Figura 1.4. Aqui, foi acrescentado o campo *usuárioConf*, que representa os usuários que confiaram nessa operação em processos de reconciliação anteriores.

Na Figura 4.2, a operação *2* realizada pelo usuário U_5 depende da operação *1* realizada pelo mesmo usuário. Se U_5 fizer uma operação *3* com os valores *[id: 3, op: cp, origem: Ivan, destino: João, chave: Artigo[Título=AcCORD], atributo: página final, valorOrigem: 24, valorDestino: 36, timestamp: 09:53:26_03/18/2014]*, esta nova operação também depende (por transitividade) da operação *1*.

usuário	id	usuário Conf	op	origem	destino	objeto	atributo	valor origem	valor destino	*timestamp*
U1	1		cp	Carlos	Bruno	Artigo [Título=AcCORD...]	página final	234	24	08:28:43_ 03/18/2014
	2		cp	Bruno	Ana	Artigo [Título=AcCORD...]	página final	234	20	08:29:01_ 03/18/2014
	3		cp	Bruno	Ana	Artigo [Título=AcCORD...]	ano	2015	2013	08:29:17_ 03/18/2014
U2	1	U5	cp	Bruno	Daniel	Artigo [Título=AcCORD...]	página final	24	15	13:41:47_ 03/18/2014
U3	1		ed	*null*	Daniel	Artigo [Título=AcCORD...]	página final	22	15	12:24:55_ 03/18/2014
	2		cp	Daniel	Emanuel	Artigo [Título=AcCORD...]	página final	20	1352	12:32:47_ 03/18/2014
U4	1	U6	cp	Daniel	Fábio	Artigo [Título=AcCORD...]	página final	15	42	10:13:45_ 03/18/2014
	2	U6	cp	Daniel	Gabriel	Artigo [Título=AcCORD...]	página final	15	25	10:14:21_ 03/18/2014
	3	U6	cp	Fábio	Hugo	Artigo [Título=AcCORD...]	página final	15	32	10:15:42_ 03/18/2014
U5	1		cp	Bruno	Gabriel	Artigo [Título=AcCORD...]	página final	24	25	09:45:57_ 03/18/2014
	2		cp	Gabriel	Ivan	Artigo [Título=AcCORD...]	página final	24	72	09:50:02_ 03/18/2014
U6	1		cp	Bruno	Hugo	Artigo [Título=AcCORD...]	página inicial	5	13	07:25:56_ 03/18/2014

Figura 4.2: Repositório R_1 com as decisões de todos os usuários.

A Definição 2 detalha as operações conflitantes em nível multiusuário.

Definição 2 (Operações conflitantes)**.** *Duas operações a e b conflitam quando são realizadas sobre o mesmo atributo do mesmo objeto por usuários diferentes, e se ocorrer uma das seguintes condições: (i) o destino em a é igual ao destino em b; (ii) o destino de a é igual à origem de b, ou; (iii) a origem de a é igual ao destino de b. Ou seja,*

1. *a(**destino**, objeto, atributo) = b(**destino**, objeto, atributo), ou;*
2. *a(**destino**, objeto, atributo) = b(**origem**, objeto, atributo), ou;*
3. *a(**origem**, objeto, atributo) = b(**destino**, objeto, atributo).*

Considere novamente o exemplo na Figura 4.2. A operação *1* do usuário U_2 conflita com a operação *1* de U_1 pois a *origem* (*Bruno*) na operação de U_2 é igual ao *destino* na operação de U_1.

Intuitivamente, enquanto U_2 considera que Bruno tem o valor correto para escrever na fonte *Daniel*, U_1 sobrescreve o valor de Bruno com o valor de *Carlos*. Além disso, a operação *1* do usuário U_2 conflita com a operação *1* do usuário U_3 porque o *destino* em ambas as operações são o mesmo. Aqui, o conflito resulta do fato dos usuários U_2 e U_3 possuírem valores distintos para o mesmo atributo.

4.3 Políticas de Reconciliação

São propostas nesta tese seis políticas para reconciliar dados em nível multiusuário. Como descrito anteriormente, cada política recebe como entrada vários conjuntos de "dados de procedência em nível de único usuário" gerados pelos usuários $U_1, U_2, ..., U_n$ e produz como saída um conjunto de "dados de procedência em nível multiusuário" que é usado para atualizar o repositório de operações do usuário que requereu o processo de reconciliação multiusuário.

O procedimento geral para compartilhar atualizações é o seguinte. Primeiro, um usuário importa um conjunto de "dados de procedência em nível de único usuário" e agrupa as operações de acordo com a *chave* e o *atributo*. Então, os agrupamentos são analisados a fim de se detectar inconsistências. Se a operação em um agrupamento conflita (Definição 2), essas operações conflitantes e suas operações dependentes (Definição 1) podem ser mantidas no agrupamento, removidas do agrupamento ou atualizadas, de acordo com a política escolhida. Por um outro lado, se as operações em um agrupamento não conflitam, elas são mantidas no agrupamento. Depois desse processo, as operações que permanecem nos agrupamento representam os "dados de procedência em nível multiusuário", e compõem a nova versão do repositório de operações do usuário.

As políticas diferem uma da outra em relação à maneira como gerenciam as operações em conflito e suas operações dependentes. Essas políticas são introduzidas nas Seções 4.3.1 a 4.3.6, nas quais são realizadas discussões a respeito de suas características e aplicabilidade.

4.3.1 Política baseada na Visão Local

A *política baseada na visão local* é caracterizada por dar prioridade para atualizações feitas localmente pelo próprio usuário, em relação às outras feitas por outros usuários. Assim, se as atualizações importadas conflitam com as atualizações do usuário local, elas são simplesmente removidas dos agrupamentos.

Considere, por exemplo, que o usuário U_1 está reconciliando seu repositório com os repositórios importados dos usuários $U_2, ..., U_6$. Os conflitos no repositório R_1 depois de importar os repositórios

$R_2, \ldots, R_6$ são gerenciados como segue. Para cada agrupamento, se uma operação o realizada pelos usuários $U_2, ..., U_n$ conflita com uma operação p realizada pelo usuário U_1, a política remove o do agrupamento. Além disso, a política remove todas as operações dependentes de o do agrupamento.

No Algoritmo 1 são mostrados os detalhes da *política baseada na visão local*. O algoritmo recebe como entrada um conjunto consistente de repositórios importados de vários usuários. Primeiramente, as operações de todos os repositórios são agrupadas de acordo com o *objeto* e o *atributo* (linhas 1 a 3). Em cada agrupamento, são buscadas operações conflitantes, de acordo com a Definição 2 (linhas 4 a 21). Se uma operação o conflita com uma operação local p, a operação o é removida do agrupamento. Além disso, as operações dependentes de o (Definição 1) também são removidas para não gerar inconsistências (linhas 7 a 12). O procedimento *getOperacoesDependentes()* é detalhado no Algoritmo 2. Por fim, cada operação que permanece no agrupamento, ou seja, as operações não conflitantes com as operações locais, compõem o repositório local, criando então uma visão local (linhas 22 a 24).

O algoritmo para a *política baseada na visão local* é executado em tempo $O(N + m * |c|^4)$, onde N é o número total de operações de todos os repositórios e $O(N)$ é o tempo para agrupar as operações, m é o número de grupos e $|c|$ é o número de operações de cada grupo. Além disso, o procedimento *getOperacoesDependentes(c, o)* gasta tempo $O(|c|^2)$ para encontrar as operações dependentes.

Operações nas quais o usuário U_1 confiou em processos de reconciliação anteriores são tratadas como pertencentes ao repositório R_1, para que o processo de reconciliação seja incremental. Assim, o campo da operação no repositório usado pela política é o *usuárioConf*, no qual é verificado se o usuário U_1 está presente.

Considere o conjunto de operações a ser inserido no repositório R_1 mostrado na Figura 4.2. A Figura 4.3 mostra o repositório R_1 depois que o usuário U_1 realiza o processo de reconciliação multiusuário aplicando a política baseada na visão local. O repositório R_1 não contém operações dos usuários U_2, U_3, U_4 e U_5, porque elas conflitaram e foram removidas do agrupamento. Observe que a operação do usuário U_6 é mantida no repositório porque ela é aplicada em um atributo diferente, *página inicial*.

Como as operações representam decisões que um determinado usuário toma para resolver inconsistências, a *política baseada na visão local* deve ser usada sempre que esse usuário acredita que as decisões realizadas localmente são mais corretas que aquelas efetuadas por outros usuários. Além disso, essa política garante que as decisões locais nunca sejam sobrescritas por decisões feitas por outros usuários. Então, quando os colaboradores decidem aplicar a política baseada na visão local, as visões locais para cada um deles serão, provavelmente, diferentes, desde que eles tenham tomado

Algoritmo 1: Política baseada na visão local.

Entrada: Dados de procedência em nível de único usuário
Saída : Dados de procedência em nível multiusuário

```
1  forall the repositorio r ∈ R do
2      C ← clustering();
3  endfall
4  foreach cluster c ∈ C do
5      foreach operacao p ∈ c do
6          if ∃o, o ∈ c, conflitando com p then
7              if o ∈ r, r! = repositorioLocal then
8                  OperacoesDependentes ← ∅;
9                  OperacoesDependentes ←
                   OperacoesDependentes + getOperacoesDependentes(c, o);
10                 c ← c − {OperacoesDependentes};
11                 c ← c − {o};
12             endif
13             if d ∈ r, r! = repositorioLocal then
14                 OperacoesDependentes ← ∅;
15                 OperacoesDependentes ← getOperacoesDependentes(c, p);
16                 c ← c − {OperacoesDependentes};
17                 c ← c − {p};
18             endif
19         endif
20     endfch
21 endfch
22 foreach cluster c ∈ C do
23     repositorioReconciliado ← c;
24 endfch
```

Algoritmo 2: Procedimentos que detectam as operações dependentes de uma operação d.

Entrada: Uma operação d de um agrupamento c
Saída : As operações dependentes de d

```
1  getOperacoesDependentes(cluster c, operacao opConflitante)
   booleano resultado ← false;
2  operacao op ← opConflitante.proximoirmaoDOM();
3  while !operation.isNull() do
4      if op.origem = opConflitante.destino ∧ op.chave = opConflitante.chave ∧ op.atributo =
       opConflitante.atributo then
5          OperacoesDependentes ← OperacoesDependentes + op;
6          resultado ← true;
7      endif
8      op ← op.proximoirmaoDOM();
9  endw
10 if resultado = true then
11     getOperacoesDependentesIndiretas();
12 endif
13 endm

   getOperacoesDependentesIndiretas()
14 foreach operacao opDependente ∈ OperacoesDependentes do
15     op ← opDependente.proximoirmaoDOM();
16     while !op.isNull() do
17         if op.origem = opDependente.destino ∧ op.chave =
           opDependente.chave ∧ op.atributo = opDependente.atributo then
18             OperacoesDependentes ← OperacoesDependentes + op;
19         endif
20         op ← op.proximoirmaoDOM();
21     endw
22 endfch
23 retorna OperacoesDependentes;
24 endm
```

usuário	id	usuário Conf	op	origem	destino	objeto	atributo	valor Origem	valor Destino	*timestamp*
U1	1		cp	Carlos	Bruno	Artigo [Título=AcCORD...]	página final	234	24	08:28:43_ 03/18/2014
	2		cp	Bruno	Ana	Artigo [Título=AcCORD...]	final final	234	20	08:29:01_ 03/18/2014
	3		cp	Bruno	Ana	Artigo [Título=AcCORD...]	Ano	2015	2013	08:29:17_ 03/18/2014
U6	1	U1	cp	Bruno	Hugo	Artigo [Título=AcCORD...]	página inicial	5	13	07:25:56_ 03/18/2014

Figura 4.3: Repositório R_1 depois do processo de reconciliação usando a *política baseada na visão local*.

diferentes decisões para resolver o mesmo conflito. Esta política é adequada para compartilhar dados em um trabalho colaborativo.

4.3.2 Política que Remove todos os Conflitos

A *política que remove todos os conflitos* é caracterizada por não priorizar operações conflitantes, independentemente do usuário que requereu o processo de reconciliação.

Ela gerencia as operações conflitantes e suas operações dependentes como segue. Para cada agrupamento, a política remove todas as operações conflitantes e suas dependentes. O algoritmo para esta política é mostrado no Algoritmo 3 e recebe como entrada um conjunto de repositórios com operações consistentes e retorna como saída uma visão integrada desses repositórios. Depois de agrupar as operações de acordo com a *chave* e *atributo*, conflitos são verificados em cada agrupamento. Se quaisquer duas operações p e o conflitam, ambas são removidas do agrupamento juntamente com suas operações dependentes (linhas 4 a 17). Finalmente, cada operação que permaneceu no agrupamento, ou seja, operações não conflitantes, compõem o repositório, criando assim uma visão consistente (linhas 20 a 22).

O algoritmo para a *política que remove todos os conflitos* é executado em tempo $O(N + m * |c|^4)$, onde Né o número total de operações de todos os repositórios e $O(N)$ é o tempo para agrupar as operações, m é o número de grupos, $|c|$ é o número de operações de cada grupo. Além disso, o procedimento *getOperacoesDependentes(c, o)* gasta tempo $O(|c|^2)$ para encontrar as operações dependentes.

Considere o conjunto de operações a ser inserido no repositório R_1 mostrado na Figura 4.2. A Figura 4.4 mostra o repositório R_1 depois que o usuário U_1 requereu o processo de reconciliação multiusuário usando a *política que remove todos os conflitos*. Note que o repositório R_1 contém apenas operações que não conflitam.

Algoritmo 3: Política que remove todos os conflitos.

Entrada: Dados de procedência em nível de único usuário
Saída : Dados de procedência em nível multiusuário

```
1  forall the repositório r ∈ R do
2      C ← clustering();
3  endfall
4  foreach cluster c ∈ C do
5      foreach operacao p ∈ c do
6          if ∃o, o ∈ c conflitando com op then
7              OperacoesDependentes ← ∅;
8              OperacoesDependentes ← getOperacoesDependentes(c, o);
9              c ← c − {OperacoesDependentes};
10             c ← c − {o};
11             OperacoesDependentes ← ∅;
12             OperacoesDependentes ← getOperacoesDependentes(c, p);
13             c ← c − {OperacoesDependentes};
14             c ← c − {p};
15         endif
16     endfch
17 endfch
18 foreach cluster c ∈ C do
19     repositorioReconciliado ← c;
20 endfch
```

usuário	id	usuário Conf	op	origem	destino	objeto	atributo	valor Origem	valor Destino	*timestamp*
U1	3		cp	Bruno	Ana	Artigo [Título=AcCORD...]	ano	2015	2013	08:29:17_ 03/18/2015
U6	1	U1	cp	Bruno	Hugo	Artigo [Títuolo=AcCORD...]	página inicial	5	13	07:25:56_ 03/18/2015

Figura 4.4: Repositório R_1 depois do processo de reconciliação usando a *política que remove todos os conflitos*.

A *política que remove todos os conflitos* remove todas as operações conflitantes e suas operações dependentes quando não é capaz de decidir quais operações conflitantes são corretas. Essa decisão é postergada para que os usuários colaborativos possam discutir sobre potenciais inconsistências e concordar sobre os valores integrados para os dados importados. Dados de procedência relacionados aos valores integrados devem ser adicionados ao repositório de operações do usuário que solicitou o processo de reconciliação. Então, esta política pode ser usada para postergar a decisão sobre quais os valores serão mantidos no banco de dados local, a fim de gerar uma única visão global integrada para todos os usuários. Além disso, quando os usuários colaborativos decidem aplicar a *política que remove todos os conflitos*, as visões locais de cada usuário serão, eventualmente, a mesma.

4.3.3 Política baseada em *Timestamp*

A *política baseada em timestamp*é caracterizada por priorizar a ordem temporal das operações conflitantes.

Ela gerencia as operações conflitantes e suas dependentes como segue. Para cada agrupamento, a política escolhe manter a operação conflitante mais recente, ou seja, a operação conflitante p que tem o mais alto *timestamp*. A política também mantém no agrupamento todas as operações dependentes de p. Cada operação conflitante o remanescente no agrupamento é gerenciada de acordo com seu tipo de conflito. Se o conflito entre p e o ocorre porque a *origem* de o é igual ao *destino* de p (Tipo 2 na Definição 2), o *valorOrigem* de o e de suas operações dependentes são refeitos para o mesmo *valorOrigem* de p. Essas operações atualizadas são movidas para o final do agrupamento. Por um outro lado, se o conflito entre p e o é do Tipo 1 ou 3 na Definição 2, a operação o é removida do agrupamento, desde que não pode ser refeita. Todas as operações dependentes de o são atualizadas alterando o campo *valorOrigem* dessas operações para o mesmo valor de origem de p, e então, elas são movidas para o final de todos os agrupamentos.

Como pode ser observado, na *política baseada em timestamp*, a operação com mais alto *timestamp* é considerada como a operação correta, e para cada operação com a mesma *chave* e *atributo* é

verificado se ela conflita como a operação correta e qual o tipo de conflito. Isso é mostrado nas linhas 5 a 8 no Algoritmo 4.

A Definição 3 detalha a solução para os conflitos entre as operações.

Definição 3 (Solução para os conflitos entre as operações)**.** *Se uma operação o e uma operação correta p conflitam e esse conflito é do tipo (Definição 2):*

Tipo 2: *1. O* valorOrigem *nas operações o e suas dependentes são refeitos de acordo com o* valorOrigem *de p;*

2. A operação o e suas dependentes são movidas para o final do agrupamento.

Tipo 1 ou 3: *1. O* valorOrigem *nas operações dependentes de o são refeitos de acordo com o* valorOrigem *de p;*

2. A operação o é removida do agrupamento, desde que não é possível refazê-la;

3. As operações dependentes de o são movidas para do agrupamento.

De acordo com a Definição 3, os conflitos do tipo 2 no Algoritmo 4 são tratados nas linhas 9 a 16. Já os conflitos dos tipos 1 e 3 são tratados nas linhas 17 a 24. Se nenhum conflito é detectado, então a *origem* na operação *o* é igual à *origem* na operação correta *p*, ou não há sobreposições entre a *origem* e o *destino* de duas operações. Finalmente, cada operação que permanece no agrupamento, ou seja, operações não conflitantes e operações refeitas, compõem o repositório final (linhas 27 a 29).

O algoritmo 4 é executado em tempo $O(N + m * |c|^4)$, onde N é o número total de operações armazenadas em todos os repositórios e $O(N)$ é o tempo para agrupar as operações, m é o número de grupos e $|c|$ é o número de operações de cada grupo. O procedimento *getOperacaoMaiorTimestamp()* é executado em tempo $O(|c|)$, procedimento *detectaConflito(p, o)* é executado tempo $O(1)$ e o procedimento *getOperacoesDependentes(c, o)* é executado em tempo $O(|c|^2)$. Adicionalmente a esse tempo, existe um tempo $O(m * |c|^4)$ para gerenciar as operações conflitantes que não conflitam com a operação mais recente, mas conflitam entre si, não mostrado no algoritmo.

Considere o repositório R_1 mostrado na Figura 4.2. A Figura 4.5 mostra o repositório R_1 depois que o usuário U_1 requereu o processo de reconciliação usando a *política baseada em timestamp*. A operação com mais alto *timestamp* é aquela realizada pelo usuário U_2, portanto, a operação *1* do usuário U_2 é mantida no repositório. Para os demais usuários, a operação *1* do usuário U_1 conflita com a operação *1* do usuário U_2 porque o *destino* na operação do usuário U_1 é igual a *origem* na operação do usuário U_2. Então a operação *1* do usuário U_1 é removida do repositório e sua operação

Algoritmo 4: Política baseado em *timestamp*.

Entrada: Dados de procedência em nível de único usuário
Saída : Dados de procedência em nível multiusuário

```
1  forall the repositório r ∈ R do
2      C ← clustering();
3  endfall
4  foreach cluster c ∈ C do
5      p ← getOperacaoMaiorTimestamp();
6      foreach operation o ∈ c do
7          OperacoesDependentes ← ∅;
8          tipoConflito ← detectaConflito(p,o);
9          if conflictType = '2' then
10             OperacoesDependentes ← d;
11             OperacoesDependentes ←
               OperacoesDependentes + getOperacoesDependentes(c,o);
12             foreach operação d ∈ OperacoesDependentes do
13                 d.valorOrigem ← p.valorOrigem;
14                 Move d para o final de c;
15             endfch
16         endif
17         else if (tipoConflito = '1') ∨ (conflictType = '3') then
18             OperacoesDependentes ←
               OperacoesDependentes + getOperacoesDependentes(c,o);
19             foreach operação d ∈ OperacoesDependentes do
20                 d.valorOrigem ← p.valorOrigem;
21                 Move d para o final de c;
22             endfch
23             c ← c − {o};
24         endif
25     endfch
26 endfch
27 foreach cluster c ∈ C do
28     repositorioReconciliado ← c;
29 endfch
```

dependente *2* é atualizada e movida para o final do repositório. A operação *3* do usuário U_1 não conflita com qualquer outra, e permanece no repositório. A operação *1* do usuário U_3 conflita com a operação *1* do usuário U_2 porque os valores de destino em ambas as operações são o mesmo. Então, a operação *1* do usuário U_3 é removida e sua operação dependente *2* é atualizada e movida para o final do repositório. Além disso, as operações *1* e *2* do usuário U_4 conflitam com a operação *1* do usuário U_2 porque a *origem* nas operações do usuário U_4 são iguais ao *destino* na operação do usuário U_2. Assim, as operações *1*, *2* e a dependente *3*, do usuário U_4 são atualizadas e movidas para o final do repositório. Finalmente, as operações do usuário U_5 não conflitam com a operação *1* do usuário U_2 e a operação do usuário U_6 não conflita com qualquer outra. Operações de ambos os usuários compõem o repositório R_1.

usuário	id	usuário Conf	op	origem	destino	objeto	atributo	valor Origem	valor Destino	*timestamp*
U1	3		cp	Bruno	Ana	Artigo [Título=AcCORD...]	ano	2015	2013	08:29:17_ 03/18/2015
U2	1	U5+ U1	cp	Bruno	Daniel	Artigo [Título=AcCORD...]	final final	24	15	13:41:47_ 03/18/2015
U5	1	U1	cp	Bruno	Gabriel	Artigo [Título=AcCORD...]	página final	24	25	09:45:57_ 03/18/2015
	2	U1	cp	Gabriel	Ivan	Artigo [Título=AcCORD...]	página final	24	72	09:50:02_ 03/18/2015
U6	1	U1	cp	Bruno	Hugo	Artigo [Título=AcCORD...]	página inicial	5 5	13 13	07:25:56_ 03/18/2015
U1	2		cp	Bruno	Ana	Artigo [Título=AcCORD...]	página final	24	20	08:29:01_ 03/18/2015
U3	2	U1	cp	Daniel	Emanuel	Artigo [Título=AcCORD...]	página final	24 24	20 20	12:32:47_ 03/18/2015
U4	1	U6+ U1	cp	Daniel	Fábio	Artigo [Título=AcCORD...]	página final	24	42	10:13:45_ 03/18/2015
	2	U6+ U1	cp	Daniel	Gabriel	Artigo [Título=AcCORD...]	página final	24	25	10:14:21_ 03/18/2015
	3	U6 U1	cp	Fábio	Hugo	Artigo [Título=AcCORD...]	página final	24 24	32 32	10:15:42_ 03/18/2015

Figura 4.5: Repositório R_1 depois do processo de reconciliação usando a *política baseada em timestamp*.

A *política baseada em timestamp* mantém a mais recente atualização feita por qualquer usuário, ou seja, a operação conflitante que tem o mais recente *timestamp*. Ela é motivada pelo mesmo princípio adotado por vários sistemas comerciais e não comerciais, como Google Docs[1] e Wikipedia[2]. Essa política pode ser usada para gerar uma única visão global integrada. Além disso, quando usuários colaborativos decidem aplicar a *política baseada em timestamp*, as visões locais de cada usuário serão eventualmente a mesma.

[1]Google Inc., `https://www.google.com`
[2]Wikimedia, `http://www.wikipedia.org/`

4.3.4 Política baseada em Votação

A *política baseada em votação* é caracterizada por manter a operação conflitante que reflete a maioria das decisões.

Ela gerencia as operações conflitantes e suas dependentes como segue. Para cada agrupamento, a política conta quantos usuários escolheram o mesmo *valorOrigem* para um atributo conflitante. As operações cujo *valorOrigem* foi escolhido pela maioria representam as operações vencedoras e são mantidas no agrupamento. Todas as operações dependentes delas são também mantidas no agrupamento. Similarmente à *política baseada em timestamp*, cada operação *o* conflitante com alguma operação vencedora é gerenciada de acordo com o tipo de conflito na *política baseada em votação*. Se o conflito entre *o* e qualquer operação pertencente ao grupo das operações vencedoras, digamos *p*, é do Tipo 2 na Definição 2, o valor de *valorOrigem* de *o* e suas operações dependentes são atualizados para o mesmo *valorOrigem* de *p*. Essas operações atualizadas são movidas para o final de todos os agrupamentos. Por outro lado, se o conflito entre *o* e qualquer operação pertencente ao grupo das vencedoras é do Tipo 1 ou 3 na Definição 2, a operação *p* é removida do agrupamento, porque não é possível refazê-la. As operações dependentes de *o* têm seu *valorOrigem* atualizado para o *valorOrigem* de qualquer operação no grupo das vencedoras, e são movidas para o final de todos os agrupamento. No caso de não haver vencedoras, todas as operações conflitantes e suas operações dependentes são removidas do agrupamento.

Na *política baseada em votação* (Algoritmo 5), depois que as operações são agrupadas de acordo com a *chave* e o *atributo*, é verificado qual *valorOrigem* ocorre mais vezes, ou seja, ocorre a votação (linha 5). Isso significa que mais usuários concordam que o valor correto para o atributo é um determinado valor. No entanto, pode ocorrer um empate na votação, ou seja, os valores com maior número de votos podem ter a mesma quantidade de votos. Nesse caso, não há vencedor na votação, e todas as operações conflitantes são removidas do agrupamento, similarmente à política que remove todos os conflitos (linhas 6 a 20).

Por outro lado, se existe somente um valor com maior número de votos, é verificado quais operações contribuíram para que aquele valor fosse o vencedor. Essas operações são armazenadas no conjunto *P* (linha 22). Para todas as outras operações no agrupamento, é verificado se existe conflito e qual o tipo de conflito com cada operação em *P* (linhas 24 a 28). Se uma operação conflita com pelo menos uma *p* em *P* porque sua *origem* é igual ao *destino* na operação *p*, o conflito é especificado como Tipo 2 (Definição 2) e a operação pode ser refeita de acordo com o *valorOrigem* de *p* e mantida no repositório. Por outro lado, se o conflito ocorre porque o *destino* em ambas as operações

são iguais, o conflito é especificado como Tipo 1. Finalmente, se o conflito ocorre porque o *destino* na operação *o* é igual a *origem* na operação *p*, o tipo de conflito é tomado como Tipo 3 (linhas 29 a 40). As operações são mantidas, refeitas e mantidas, ou removidas do agrupamento de acordo com a Definição 3 (linhas 41 a 56). Finalmente, o algoritmo retorna como saída um repositório composto por cada operação não conflitante e cada operação tratada de todos os agrupamentos (linhas 60 a 63).

Quando há vencedoras na votação, o algoritmo 5 é executado em tempo $O(N + m * |c|^3 * |w|) + O(m * |c|^4) = O(N + m * |c|^4)$, onde N é o número total de operações de todos os repositórios e $O(N)$ é o tempo para agrupar as operações, m é o número de grupos e $|c|$ é o número de operações de cada grupo. O tempo para executar a votação com o procedimento *realizaVotacao*() é $O(|c|)$. $|w|$ é o número de operações do conjunto de vencedoras e $O(|w|)$ é o tempo requerido para verificar os conflitos com as operações vencedoras (*DetectaConflito(p, o)*). Como $|w|$ não pode ser maior que $|c|$, tem-se o tempo dado. Adicionalmente a esse tempo, existe um tempo $O(m * |c|^4)$ para gerenciar as operações conflitantes que não conflitam com o grupo das operações vencedoras, mas conflitam entre si. Caso não haja operação vencedora, o tempo de execução da política é o tempo para realizar a votação, $O(|c|)$, mais o tempo para gerenciar as operações como no Algoritmo 3 para a *política que remove todos os conflitos*, $O(N + (m * |c|^4)$. Assintoticamente os tempos nos dois casos são semelhantes, no entanto, caso haja operação vencedora, a política precisa gerenciar todas as operações de todos os repositórios em relação aos conflitos com as vencedoras, e posteriormente, gerenciar todas as operações de todos os repositórios, em relação aos conflitos entre si. Caso não haja operação vencedora, a política precisa gerenciar as operações de todos os repositórios, em relação aos conflitos entre si (tempo polinomial em relação ao tamanho do grupo vezes o número de grupos).

Considere o conjunto de operações a ser inserido no repositório R_1 mostrado na Figura 4.2. A Figura 4.6 mostra o repositório R_1 depois que o usuário U_1 requereu o processo de reconciliação multiusuário usando a política baseada em votação. As operações vencedoras são a operação *1* do usuário U_2 e a operação *1* do usuário U_5. Essas duas operações, juntamente com a operação 2 do usuário U_5, que é dependente da operação *1* desse usuário, são mantidas no repositório. O repositório R_1 não contém a operação *1* do usuário U_1 desde que ela foi removida porque conflita com a operação *1* do usuário U_2 e com a operação *1* do usuário U_5 (conflito do Tipo 3 em ambos os casos). Por outro lado, a operação dependente *2* do usuário U_1 é atualizada e movida para o final do repositório. A operação 3 do usuário U_1 é mantida no repositório porque não é uma operação conflitante. Além disso, a operação *1* do usuário U_3 foi removida do repositório porque conflita com a operação *1* do usuário U_2 (conflito Tipo 1). Sua operação dependente é atualizada e movida para o final do repositório. A operação *1* do usuário U_4 conflita com a operação 1 do usuário U_2. Então, essa

Algoritmo 5: Política baseada em Votação.

Entrada: Dados de procedência em nível de único usuário
Saída : Dados de procedência em nível multiusuário

forall the *repositorio* $r \in R$ **do**
  $C \leftarrow$ clustering();
endfall
foreach *cluster* $c \in C$ **do**
  $vencedor \leftarrow realizaVotacao()$;
  if *(!vencedor)* **then**
    /* Similarmente a política que remove todos os conflitos */
    foreach *operacao* $p \in c$ **do**
      if $\exists o, o \in c$ *conflitando com* p **then**
        $OperacoesDependentes \leftarrow \emptyset$;
        $OperacoesDependentes \leftarrow getOperacoesDependentes(c,o)$;
        $c \leftarrow c - \{OperacoesDependentes\}$;
        $c \leftarrow c - \{o\}$;
        $OperacoesDependentes \leftarrow \emptyset$;
        $OperacoesDependentes \leftarrow getOperacoesDependentes(c,p)$;
        $c \leftarrow c - \{OperacoesDependentes\}$;
        $c \leftarrow c - \{p\}$;
      endif
    endfch
  endif
  else
    $P \leftarrow determinaVencedoras()$;
    $TipoConflito \leftarrow \emptyset$;
    foreach *operação* $o \in c - Op$ **do**
      $OperacoesDependentes \leftarrow \emptyset$;
      foreach *operation* $p \in P$ **do**
        $TipoConflito \leftarrow TipoConflito + DetectaConflito(p,o)$;
      endfch
      if $TipoConflito \supset$ *'2'* **then**
        $tipoConflito \leftarrow$ '2';
      endif
      else if $TipoConflito \supset$ *'1'* **then**
        $tipoConflito \leftarrow$ '1';
      endif
      else if $TipoConflito \supset 3$ **then**
        $tipoConflito \leftarrow$ '3';
      endif
      else
        $tipoConflito \leftarrow$ 'nenhum';
      endif
      if $tipoConflito =$ *'2'* **then**
        $OperacoesDependentes \leftarrow o$;
        $OperacoesDependentes \leftarrow OperacoesDependentes + getOperacoesDependentes(c,o)$;
        foreach *operação* $d \in OperacoesDependentes$ **do**
          $d.valorOrigem \leftarrow p.valorOrigem$;
          Move d para o final de c;
        endfch
      endif
      else if $(tipoConflito =$ *'1'*$) \vee (conflictType =$ *'3'*$)$ **then**
        $OperacoesDependentes \leftarrow OperacoesDependentes + geOperacoesDependentes(c,o)$;
        foreach *operação* $d \in OperacoesDependentes$ **do**
          $d.valorOrigem \leftarrow p.valorOrigem$;
          Move d para o final de c;
        endfch
        $c \leftarrow c - \{o\}$;
      endif
    endfch
  endif
endfch
foreach *cluster* $c \in C$ **do**
  $repositorioReconciliado \leftarrow c$;
endfch

operação e a operação dependente *3* são atualizadas e movidas para o final do repositório. A operação *2* do usuário U_4 conflita com a operação *1* do usuário U_2 (conflito do Tipo 2). Então, seu *valorOrigem* é atualizado para o mesmo *valorOrigem* da operação *1* do usuário U_2 e a operação é movida para o final do repositório.

usuário	id	usuário Conf	op	origem	destino	objeto	atributo	valor Origem	valor Destino	timestamp
U1	3		cp	Bruno	Ana	Artigo [Título=AcCORD...]	ano	2015	2013	08:29:17_ 03/18/2015
U2	1	U5+ U1	cp	Bruno	Daniel	Artigo [Título=AcCORD...]	página final	24	15	13:41:47_ 03/18/2015
U5	1	U1	cp	Bruno	Gabriel	Artigo [Título=AcCORD...]	página final	24	25	09:45:57_ 03/18/2015
	2	U1	cp	Gabriel	Ivan	Artigo [Título=AcCORD...]	página final	24	72	09:50:02_ 03/18/2015
U6	1	U1	cp	Bruno	Hugo	Artigo [Título=AcCORD...]	página inicial	5	13	07:25:56_ 03/18/2015
U1	2		cp	Bruno	Ana	Artigo [Título=AcCORD...]	página final	24	20	08:29:01_ 03/18/2015
U3	2	U1	cp	Daniel	Emanuel	Artigo [Título=AcCORD...]	página final	24	20	12:32:47_ 03/18/2015
U4	1	U6+ U1	cp	Daniel	Fábio	Artigo [Título=AcCORD...]	página final	24	42	10:13:45_ 03/18/2015
	3	U6+ U1	cp	Fábio	Hugo	Artigo [Título=AcCORD...]	página final	24	32	10:15:42_ 03/18/2015
	2	U6+	cp	Daniel	Gabriel	Artigo [Título=AcCORD...]	página final	24	25	10:14:21_ 03/18/2015

Figura 4.6: Repositório R_1 depois do processo de reconciliação usando *política baseada em votação*.

A *política baseada em votação* é motivada pelo fato de que se a maioria dos usuários confia em um determinado valor de atributo, então provavelmente esse valor seja correto e deve ser adotado como tal. Quando não é possível decidir qual é o valor correto para um atributo, então as operações conflitantes removidas devem ser retornadas para os usuários colaborativos, semelhantemente à discussão feita na Seção 4.3.2. Finalmente, o uso da *política baseada em votação*por usuários colaborativos representa um cenário de integração, no qual todas as visões locais serão eventualmente a mesma.

4.3.5 Política baseada na Confiança nas Fontes

A *política baseada na confiança nas fontes* é caracterizada por manter a operação conflitante cuja *origem* é a fonte mais confiável.

Primeiramente, as fontes são preprocessadas para verificar quantos objetos existem em cada fonte (numeroObjetos) e o repositório é preprocessado para verificar em quantas operações cada fonte atuou como *origem* (numeroOrigem) e como *destino* (numeroDestino). Assim, a taxa de confiança em cada fonte é calculada de acordo com a fórmula descrita na Equação 4.1. Para determinar o *numeroOrigem*,

o algoritmo não considera as operações dependentes, desde que elas propagam um valor de atributo de outra fonte, nem as operações de edição, porque possuem o campo *valorOrigem* vazio.

$$taxaConfiancaFonte = \frac{(numeroOrigem) - (numeroDestino)}{numeroObjetos} \tag{4.1}$$

A *política baseada na confiança nas fontes* gerencia as operações conflitantes e suas dependentes como segue. Para cada agrupamento, a operação que tem como *origem* a fonte com maior *taxaConfiancaFonte* é mantida no agrupamento. Todas as operações dependentes dela também são mantidas no agrupamento. Cada operação conflitante o e a operação mais confiável no agrupamento, digamos operação p, é gerenciada de acordo com a Definição 3.

O Algoritmo 6 detalha o funcionamento da *política baseada na confiança nas fontes*. Primeiro, as fontes são preprocessadas para verificar quantos objetos existem em cada fonte (linha 1). Segundo, o repositório é preprocessado para se contar em quantas operações cada fonte age como *origem* e em quantas age como *destino* (linha 2). Então, a taxa de confiança é calculada para cada fonte, de acordo com a Equação 4.1 (linha 3). Depois que as operações são agrupadas de acordo com a *chave* e o *atributo*, é verificado qual operação tem como *origem* a fonte com mais alta taxa (linha 8). Note que a operação escolhida não pode ser dependente de qualquer outra, para não propagar um valor incorreto (que não é o valor da fonte mais confiável). Note também, que operações de edição nunca serão escolhidas porque elas não possuem *origem*. As operações são mantidas, refeitas e mantidas, ou removidas do agrupamento de acordo com o tipo de conflito (linhas 9 a 28). No final, o algoritmo retorna como saída um repositório composto por cada operação não conflitante e cada operação tratada de todos os agrupamentos (linhas 30 a 32).

O algoritmo 6 é executado em tempo $O(N + S + m * |c|^4 + |r|)$, onde S é o número de fontes. O procedimento *preprocessaFontes()* é executado em tempo $O(S)$ e o procedimento *preprocessaRepositorioLocal()* é executado em tempo $O(|r|)$ e $|r|$ é o número de operações no repositório do usuário que solicitou o processo de reconciliação. O procedimento *calculaConfiancaFontes()* é executado em tempo $O(S)$ porque a confiança é calculada para cada fonte. A partir de então, a análise feita para o Algoritmo 6 é a mesma feita para o Algoritmo 4.

Considere o conjunto de operações a ser inserido em R_1 mostrado na Figura 4.2. A Tabela 4.1 mostra o cálculo da confiança em cada fonte para valores hipotéticos. Considerando esses valores, a operação com mais alta confiança é aquela realizada pelo usuário U_2 porque tem a *origem* com a maior taxa de confiança e não é dependente de qualquer outra operação (como exemplo, a operação

Algoritmo 6: Política baseada na confiança nas fontes.

Entrada: Dados de procedência em nível de único usuário
Saída : Dados de procedência em nível multiusuário

```
1  preprocessaFontes();
2  preprocessaRepositorioLocal();
3  calculaConfiancaFontes();
4  forall the repositorio r ∈ R do
5      C ← clustering();
6  endfall
7  foreach cluster c ∈ C do
8      p ← getOperacaoTaxaConfiancaFonteMaisAlta();
9      foreach operation o ∈ c do
10         OperacoesDependentes ← ∅;
11         tipoConflito ← detectaConflito(p,o);
12         if tipoConflito = '2' then
13             OperacoesDependentes ← o;
14             OperacoesDependentes ←
               OperacoesDependentes + getOperacoesDependentes(c,o);
15             foreach operação d ∈ OperacoesDependentes do
16                 d.valroOrigem ← p.valorOrigem;
17                 Move d para o final de c;
18             endfch
19         endif
20         else if (tipoConflito = '1') ∨ (conflictType = '3') then
21             OperacoesDependentes ←
               OperacoesDependentes + getOperacoesDependentes(c,o);
22             foreach operação d ∈ OperacoesDependentes do
23                 d.valorOrigem ← p.valorOrigem;
24                 Move d para o final de c;
25             endfch
26             c ← c − {o};
27         endif
28     endfch
29 endfch
30 foreach cluster c ∈ C do
31     repositorioReconciliado ← c;
32 endfch
```

2 do usuário U_1). O repositório R_1 depois que o usuário U_1 requereu o processo de reconciliação multiusuário é o mesmo obtido para a *política baseada em timestamp* e mostrado na Figura 4.5.

Tabela 4.1: Confiança em cada fonte de dado.

Fontes	Cálculo da Confiança	Confiança
Ana	$(0-2)/25$	-0.08
Bruno	$(25-3)/25$	0.88
Carlos	$(1-0)/10$	0.1
Daniel	$(13-2)/20$	0.55
Emanuel	$(0-1)/10$	-0.1
Fábio	$(1-1)/3$	0.0
Gabriel	$(1-2)/3$	-0.33
Hugo	$(1-4)/3$	-1.0
Ivan	$(0-0)/3$	0.0

A *política baseada na confiança nas fontes* é motivada pelo fato de que se uma fonte particular tem o maior número de valores corretos de acordo com a decisão do usuário, então os valores vindos dessa fonte são mais confiáveis para serem corretos, e devem ser adotados com tal. Finalmente, as visões locais para cada usuário serão provavelmente diferentes desde que cada usuário tem suas próprias taxas de confiança para cada fonte, de acordo com suas decisões locais. Essa política é adequada para compartilhar dados entre usuários que trabalham de maneira colaborativa.

4.3.6 Política baseada na Confiança nos Usuários

A *política baseada na confiança nos usuários* é caracterizada por manter no repositório a operação conflitante que possui a maior confiança, de acordo com a taxa confiança no usuário que realizou a operação e nos usuários que confiaram na operação em processos de reconciliação anteriores.

Primeiro, o repositório é preprocessado para verificar a porcentagem de operações criadas pelos usuários, ou seja, o número de operações com um valor particular para o campo *usuário* em relação ao número total de operações no repositório. Assim, a taxa de confiança para cada usuário é calculada de acordo com a fórmula descrita na Equação 4.2.

$$taxaConfiancaUsuario = \frac{count(usuario)}{count(R_i)} \tag{4.2}$$

A política gerencia as operações conflitantes e suas dependentes como segue. Para cada agrupamento, a confiança de cada operação é calculada de acordo a seguinte fórmula:

$$opConf = 0.5 * taxaConfiancaUsuario(usuario) + 0.5 * \sum_{k=1}^{N} taxaConfiancaUsuario(usuarioConf), \tag{4.3}$$

onde N é o número de usuários que confiou na operação em processos de reconciliação anteriores.

A operação com maior *opConf* é mantida no agrupamento. Todas as operações dependentes dela também são mantidas agrupamento. Cada operação conflitante o remanescente no agrupamento é gerenciada de acordo com a Definição 3, com qualquer operação mais alta confiança, digamos, a operação p. Aqui, as operações dependentes de o são atualizadas de acordo com o *valorOrigem* de qualquer uma das operações do grupo confiável.

A *política baseada na confiança nos usuários* (Algoritmo 7) primeiro computa a confiança do usuário que iniciou o processo de reconciliação em cada outro usuário (linha 1), de acordo com a Equação 4.2. Posteriormente, para cada agrupamento é calculada a taxa de confiança para cada operação de acordo com a Equação 4.3 e a operação com a mais alta taxa de confiança é escolhida como a correta (linha 6). As operações são gerenciadas de acordo com o tipo de conflito (linhas 7 a 26). No final, o algoritmo retorna como saída um repositório composto pelas operações não conflitante e cada operação tratada, em todos os agrupamentos (linhas 30 a 32).

O algoritmo 7 é executado em tempo $O(N + m * |c|^4)$. O tempo para a execução do procedimento *computaConfiancaUsuarios()* é $O(N)$, onde N é o número total de operações. O tempo para agrupar as operações é $O(N)$ e o tempo para executar o procedimento *getOperacaoMaiorTaxaConfianca()* é $O(|c|)$. O *foreach* externo é executado m vezes, onde m é o número de grupos. O *foreach* interno é executado $|c|$ vezes, para cada operação no grupo. Para detectar conflitos entre a operação mais confiável e uma outra operação, é gasto tempo $O(1)$. A detecção de operações dependentes é executada em tempo $O(|c|^2)$. Para atualizar as operações dependentes é gasto um tempo $O(|D|)$ onde $|D|$ é o número de operações dependentes. Entretanto, $|D|$ tem no máximo o número de operações de c, ou seja, $|c|$, e $|c|^4 + |c|$ é $O(|c|^4)$. Adicionalmente a esse tempo, existe um tempo $O(m * |c|^4)$ para gerenciar as operações conflitantes que não conflitam com a operação de mais alta confiança, mas conflitam entre si.

Considere o conjunto de operações a ser inserido em R_1 mostrado na Figura 4.2. Considere também que existem 100 operações em R_1 e o número de operações feitas por cada usuário, bem como

Algoritmo 7: *política baseada na confiança nos usuários.*

Entrada: Dados de procedência em nível de único usuário
Saída : Dados de procedência em nível multiusuário

```
1  computaConfiancaUsuarios();
2  forall the repositorio r ∈ R do
3      C ← clustering();
4  endfall
5  foreach cluster c ∈ C do
6      p ← getOperacaoMaiorTaxaConfianca();
7      foreach operacao o ∈ c do
8          OperacoesDependentes ← ∅;
9          tipoConflito ← detectaConflito(p,o);
10         if conflictType = '2' then
11             OperacoesDependentes ← o;
12             OperacoesDependentes ←
               OperacoesDependentes + getOperacoesDependentes(c,o);
13             foreach operação d ∈ OperacoesDependentes do
14                 d.valorOrigem ← p.originValue;
15                 Move d para o final de c;
16             endfch
17         endif
18         else if (tipoConflito = '1') ∨ (tipoConflito = '3') then
19             OperacoesDependentes ←
               OperacoesDependentes + getOperacoesDependentes(c,o);
20             foreach operação d ∈ OperacoesDependentes do
21                 d.valorOrigem ← p.valorOrigem;
22                 Move d para o final de c;
23             endfch
24             c ← c − {o};
25         endif
26     endfch
27 endfch
28 foreach cluster c ∈ C do
29     repositorioReconciliado ← c;
30 endfch
```

sua confiança calculada de acordo com a Equação 4.2, são mostradas na Tabela 4.2. Considerando esses valores, operações feitas pelo usuário U_1 têm taxa de confiança calculada de acordo com a Equação 4.3 de 0.15; a operação feita por U_2 tem taxa de 0.05 porque o usuário U_5 confia nela; as operações feitas por U_3 têm taxa de confiança de 0.1; as operações feitas por U_4 têm taxa de confiança de 0.2; as operações feitas por U_5 têm taxa de confiança de 0.01 e; a operação feita por U_6 tem taxa de confiança de 0.075. Então, as operações mais confiáveis são aquelas feitas por U_4.

Tabela 4.2: Confiança nos usuários.

Fontes	Cálculo da Confiança	Confiança
U1	$30/100$	0.3
U2	$8/100$	0.08
U3	$20/100$	0.2
U4	$25/100$	0.25
U5	$2/100$	0.02
U6	$15/100$	0.15

A Figura 4.7 mostra parte do repositório R_1 depois que o usuário U_1 requereu o processo de reconciliação multiusuário usando *política baseada na confiança nos usuários*. As operações com maior taxa de confiança são aquelas feitas pelo usuário U_4 e, portanto, essas operações são mantidas no repositório. As operações que não conflitam com as operações feitas pelo usuário U_4, mas conflitam entre si, são gerenciadas de acordo com as suas taxas de confiança. Sendo assim, o repositório R_1 não contém a operação *1* do usuário U_1, assim como sua dependente *2*. Elas foram removidas do repositório não por conflitarem com as operações mais confiáveis, mas por conflitar com a operação *1* do usuário U_2, que possui maior taxa de confiança. A operação *3* do usuário U_1 não conflita e é mantida no repositório. Caso duas operações conflitem e possuam a mesma taxa de confiança, as duas são removidas do repositório desde que não é possível decidir qual está correta baseado nos critérios de confiança. A operação do usuário U_2 conflita com as operações *1* e *2* do usuário U_4 e foi removida do repositório. A operação *1* do usuário U_3 conflita com as operações *1* e *2* do usuário U_4, porque o *destino* é igual a *origem* das operações mais confiáveis e então, ela foi removida e sua dependente, a operação *2* foi atualizada e movida para o final do repositório. A operação *1* do usuário U_5 conflita com a operação *2* do usuário U_4 porque o *destino* em ambas são iguais e a operação *1* do usuário U_5 é removida do repositório. Sua operação dependente *2* é atualizada e movida para o final do repositório. A operação do usuário U_6 não conflita com qualquer outra e é mantida no repositório.

A *política baseada na confiança nos usuários* é motivada pelo fato de que operações realizadas por usuários mais confiáveis para o usuário local são mais confiáveis para serem consideradas corretas, e devem ser adotadas como tal. Finalmente, o uso da *política baseada na confiança nos*

usuário	id	usuário Conf	op	origem	destino	objeto	atributo	valorOrigem	valorDestino	*timestamp*
U1	3		cp	Bruno	Ana	Artigo [Título=AcCORD...]	ano	2015	2013	08:29:17_ 03/18/2015
U4	1	U6+ U1	cp	Daniel	Fábio	Artigo [Título=AcCORD...]	página final	15	42	10:13:45_ 03/18/2015
	2	U6+ U1	cp	Daniel	Gabriel	Artigo [Título=AcCORD...]	página final	15	25	10:14:21_ 03/18/2015
	3	U6+ U1	cp	Fábio	Hugo	Artigo [Título=AcCORD...]	página final	15	32	10:15:42_ 03/18/2015
U6	1	U1	cp	Bruno	Hugo	Artigo [Título=AcCORD...]	página inicial	5	13	07:25:56_ 03/18/2015
U3	2	U1	cp	Daniel	Emanuel	Artigo [Título=AcCORD...]	página final	15	20	12:32:47_ 03/18/2015
U5	2	U1	cp	Gabriel	Ivan	Artigo [Título=AcCORD...]]	página final	15	72	10:13:45_ 03/18/2015

Figura 4.7: Repositório R_1 depois do processo de reconciliação usando a *política baseada na confiança nos usuários*.

usuários por todos os usuários colaborativos representa um cenário de compartilhamento, no qual as visões locais de cada um serão provavelmente diferentes, porque cada usuário tem suas próprias taxas de confiança nos demais. No primeiro processo de reconciliação, cada usuário confia somente nele mesmo.

4.4 Propagação de Decisões de Integração

Neste trabalho de doutorado, investiga-se a reconciliação de dados em nível multiusuário, conforme descrito nas Seções 4.1 a 4.3. Entretanto, antes de realizar uma reconciliação em nível multiusuário, cada usuário individualmente pode realizar alterações locais refletindo decisões de integração particulares, as quais são armazenadas nos seus próprios repositórios locais. Durante esse processo de integração em nível de único usuário, quando um usuário toma uma decisão para atualizar um atributo de um objeto para uma fonte e este atributo está com o seu valor inconsistente em outras fontes que contêm o objeto, é necessário que o usuário atualize todas as fontes inconsistentes, uma por vez. Assim, para complementar a proposta do modelo AcCORD, visando poupar o tempo que o usuário gasta para tomar a mesma decisão de integração sobre o mesmo objeto várias vezes, e também para evitar que o usuário tome decisões diferentes para um atributo de um mesmo objeto em fontes diferentes, o modelo AcCORD também introduz uma funcionalidade que é responsável por realizar a propagação de decisões de integração em nível de único usuário, a qual é descrita nesta seção. Essa funcionalidade é oferecida durante o processo de integração em nível de único usuário, conforme mostrado na Figura 4.1 (a).

Considere o exemplo mostrado na Figura 1.1. Caso o usuário decida alterar o ano de publicação do artigo *"AcCORD: Asynchronous COllaborative data ReconcIliation moDel"*, de *"2013"* para *"2016"* na fonte *Ana*, ele deverá alterar esse atributo também nas fontes *Bruno* e *Carlos*. As respectivas operações são, então, armazenadas no repositório. Para poupar o usuário de realizar essa mesma retificação várias vezes, o método proposto propaga o valor atualizado em *Ana* para as demais fontes que possuem o objeto, criando operações e as inserindo no repositório.

Definição 4 (Propagação de Decisões de Integração)**.** *Ocorre em um único processo de integração para um único usuário e envolve toda fonte B que contém um objeto X, dado que o usuário realizou uma operação a em um atributo i do objeto X da primeira fonte A. Para a operação a, é criada a operação de cópia b cujo valor de* origem *e o valor de* valorOrigem *são definidos de acordo com os valores de* destino *e* valorOrigem *da operação a. Ou seja, é criada uma operação b dependente da operação a. O valores de* destino *e* valorDestino *da operação b são definidos de acordo com o proprietário da fonte em B e o valor do atributo i na fonte B.*

Ou seja, desde que as fontes A e B contenham o objeto X, e o usuário realiza a operação a de cópia ou edição no atributo i do objeto X tal que

1. *a(op = "ed", destino = "A", chave = "X", atributo = "i", valorOrigem = usuário(), valorDestino = valorAtributo("A"))*

O método de propagação cria a operação b, tal que

1. *b(op = "cp", origem = "A", destino = "B", chave = "X", atributo = "i", valorOrigem = valorOrigem("A"), valorDestino = valorAtributo("B"))*

Esse método recebe como entrada somente o repositório local de operações e as fontes originais. Para cada operação *a* no repositório, são criadas operações de cópia *b* dependentes da operação *a*. Para a inserção da operação *b* no repositório, é necessário verificar se ela não conflita com qualquer outra operação no repositório cujo *valorOrigem* é diferente.

No método proposto existe a distinção entre diferentes termos, conforme descrito a seguir. A *operação de propagação* é aquela que propaga o valor de um atributo, ou seja, a operação *a* na Definição 4; a *operação criada* é a operação que é criada a partir da operação de propagação, ou seja, a operação *b* na Definição 4; e a operação já presente no repositório é aquela que é usada para verificar se há conflito. No exemplo corrente da Figura 1.1, a *operação de propagação* é aquela baseada na decisão do usuário de editar o valor do atributo *ano* de *"2013"* para *"2016"*. A *operação criada* pelo

método realiza uma cópia no valor do atributo *ano* da fonte *Ana* para a fonte *Bruno*, substituindo o valor *"2015"* para *"2016"*. É verificado então, em todo o repositório, se alguma operação atualiza o valor do atributo *ano* da fonte *Bruno*.

A *operação criada* para propagar o valor do atributo para outra fonte é, então, gerenciada da seguinte maneira:

- Se o valor do atributo na fonte original é o mesmo que o valor em *valorOrigem* na operação de propagação, a operação criada, a qual tem os valores de *valorOrigem* e *valorDestino* iguais, não é inserida no repositório.
- Se a operação criada conflita com alguma operação no repositório e o *valorOrigem* na operação criada é igual ao *valorOrigem* na operação, a operação criada não é inserida no repositório. Isso pode ocorrer por três razões: (i) a operação criada é exatamente a mesma que a operação no repositório; (ii) a operação criada tem o *destino* igual ao destino na operação no repositório, que, por sua vez, obteve o valor correto do usuário (em operações de edição - ed) ou outra fonte (em operações de cópia - cp), ou; (iii) a operação criada tem *destino* igual à *origem* da operação do repositório que, por sua vez, já obteve o valor correto de um operação anterior.
- Se a operação criada conflita como alguma operação no repositório e o *valorOrigem* na operação criada e na operação no repositório são diferentes, é necessário verificar o tipo de conflito. Se o *destino* na operação criada é igual à *origem* na operação no repositório, o tipo de conflito ocorrido é **conflito na origem**. Se o *destino* em ambas as operações são o mesmo, ocorre um **conflito no destino**. Em ambos os casos, é necessário mostrar para o usuário que a operação de propagação e a operação do repositório conflitam. Ou seja, ambas foram criadas ou aceitas por ele, e têm valores inconsistentes uma em relação a outra. O usuário deve então, escolher a operação correta.
  - Se o usuário escolher como correta a operação de propagação, a operação criada é automaticamente inserida no repositório. Se o conflito entre a operação criada e a operação do repositório é na **origem**, a operação do repositório e suas dependentes são refeitas de acordo com o *valorOrigem* da operação de propagação e mantidas no repositório. Se o tipo de conflito entre a operação criada e a operação do repositório é no **destino**, somente as operações dependentes da operação do repositório são refeitas e mantidas no repositório.
  - Se, por outro lado, o usuário escolher como correta a operação realizada ou aceita por ele previamente e já presente no repositório, a operação de propagação é removida do

repositório, bem como todas as operações dependentes e todas as operações criadas automaticamente a partir da operação de propagação. Além disso, todas as operações que foram removidas do repositório, devido a um conflito com a operação de propagação, retornam ao repositório; e todas as operações que tiveram seu *valorOrigem* atualizado de acordo com o *valorOrigem* da operação de propagação têm seu valor refeito de acordo com o valor antigo.

O *método de propagação de decisões de integração* (Algoritmo 8) tenta propagar a decisão para um atributo de um objeto em uma fonte para todas as outras fontes que contêm o objeto. Assim, o objetivo é propagar cada operação do repositório (linha 1) criando novas operações baseadas no *valorOrigem* destas (linha 3). Entretanto, a operação pode ser igual a outra operação no repositório, pode oferecer um valor *valorOrigem* redundante, ou pode conflitar com um outra operação no repositório. Nesses casos, a operação criada não é inserida no repositório (linha 5). Caso a operação criada não conflita com qualquer outra no repositório, ela é inserida (linhas 6 a 8). Se a operação criada conflita com qualquer outra é porque o usuário tomou uma decisão para o atributo da operação de propagação, diferente da decisão tomada para a operação em conflito. Então, as duas operações são mostradas para que ele decida qual é a operação correta (linhas 9 a 10). Se a operação de propagação é decidida como correta, a operação criada é inserida no repositório, e o tipo de conflito entre a operação criada e a operação conflitante é verificado. Se o conflito ocorre porque o *destino* da operação criada é o mesmo que a *origem* da operação conflitante *o*, a operação conflitante e suas dependentes são refeitas de acordo com o *valorOrigem* da operação de propagação *p* (linhas 13 a 19). Se o conflito ocorre porque o *destino* na operação criada e na operação conflitante têm o mesmo valor, as operações dependentes da operação conflitante são refeitos de acordo com o *valorOrigem* da operação de propagação (linhas 20 a 26). Por outro lado, se a operação conflitante é decidida como correta, a operação de propagação e suas dependentes são removidas do repositório. Além disso, as operações criadas de acordo com a operação de propagação são removidas, e as operações possivelmente removidas devido a conflitos anteriores em que o usuário decidiu pela operação de propagação, são retornados para o repositório (linhas 28 a 36).

O algoritmo para o *método de propagação de decisões de integração* é executado em tempo $O(|R|^2 * S * (|R|^2 + |D|)))$ onde $|R|$ é o número de operações nos repositórios, S é o número de fontes, $O(|R|^2)$ é o tempo para encontrar operações dependentes e $O(|D|)$ é o tempo necessário para tratar e mover operações dependentes.

Para exemplificar, considere a Figura 4.8, as operações *1*, *2* e *3* foram realizadas pelo usuário U_7 no *atributo: Volume* de *objeto: AcCORD*. As fontes mostradas na Tabela 4.3 são todas aquelas que

Algoritmo 8: Método de propagação de decisões de integração.

Entrada: Repositório com dados de procedência em nível de único usuário
Saída : Repositório estendido com dados de procedência em nível de único usuário

```
1  foreach operacaoPropagacao p ∈ repositorio r do
2      foreach source s ∈ S do
3          operacaoCriada(op);
4          foreach operation o! = p ∈ repositorior do
5              while tipoConflito(op,o) ≠ 'igual' do
6                  if tipoConflito(op,o) = 'nenhum' then
7                      r ← op;
8                  endif
9                  else if tipoConflito(op,o) = 'origem' ∨ tipoConflito(op,o) = 'destino' then
10                     Mostrar para o usuário a operacaoPropagacao p e a operação o;
11                     if p é correta then
12                         r ← op;
13                         if tipoConflito = 'origem' then
14                             OperacoesDependetes ← o;
15                             OperacoesDependetes ←
                               OperacoesDependetes + getOperacoesDependetes(r,o);
16                             foreach operation d ∈ OperacoesDependetes do
17                                 d.valorOrigem ← p.valorOrigem;
18                             endfch
19                         endif
20                         else if tipoConflito = 'destino' then
21                             OperacoesDependetes ←
                               OperacoesDependetes + getOperacoesDependetes(r,o);
22                             foreach operation d ∈ DependentOperations do
23                                 d.valorOrigem ← p.valorOrigem;
24                             endfch
25                             r ← r − o;
26                         endif
27                     endif
28                     else if o é correta then
29                         OperacoesDependetes ←
                           OperacoesDependetes + getOperacoesDependetes(r, p);
30                         r ← r − p;
31                         foreach d ∈ OperacoesDependetes do
32                             r ← r − d;
33                         endfch
34                         Rollback nas operações feitas de acordo com a operação de propagação
                           p;
35                         Rollback nas operações o e suas dependentes removidas devido a
                           operação de propagação p;
36                     endif
37                 endif
38             endw
39         endfch
40     endfch
41 endfch
```

tem o *objeto: AcCORD*. Assim, cada operação no repositório R_7 deve ser propagada para atualizar o valor do *atributo* em outras fontes.

Inicialmente, a operação *1* é a operação de propagação e a operação *[id: 4, op: cp, origem: Ana, destino: Bruno, chave: Artigo[Título=AcCORD], atributo: Volume, valorOrigem: 3, valorDestino: 2, timestamp: 15:29:17_03/28/2015]* é criada. Essa operação é igual à operação *2* já presente no repositório e, portanto, não é inserida. Então, a operação *4* com os valores: *[id: 4, op: cp, origem: Ana, destino: Carlos, chave: Artigo[Título=AcCORD], atributo: volume, valorOrigem: 3, valorOrigem: 2, timestamp: 15:29:17_03/28/2015]* é criada e inserida no repositório. Na sequência, a operação *5* com os valores: *[id: 5, op: cp, origem: Ana, destino: Daniel, chave: Artigo[Título=AcCORD], atributo: volume, valorOrigem: 3, valorDestino: 1, timestamp: 15:29:17_03/28/2015]* é criada. Essa operação conflita no **destino** com a operação *3* já presente no repositório. Como resultado, as duas operações, *1* e *3* são mostradas para o usuário para que ele decida qual o valor correto, *3* ou *5*. Se o usuário decidir que a operação correta é a *1*, o repositório final R_7, será como mostra a Figura 4.9. Se o usuário decidir que a operação correta é a *3*, o repositório final R_7, será como mostra a Figura 4.10. Posteriormente, a próxima operação de propagação será a operação *2* e as ações a serem tomadas são semelhantes às já descritas.

Tabela 4.3: Fontes que contêm o objeto “AcCORD”.

Fontes
Ana
Bruno
Carlos
Daniel

usuário	id	op	origem	destino	chave	atributo	valor Origem	valor Destino	*timestamp*
U7	1	ed		Ana	Artigo [Título=AcCORD...]	volume	3	1	15:29:17_ 03/28/2015
	2	cp	Ana	Bruno	Artigo [Título=AcCORD...]	volume	3	2	15:30:02_ 03/28/2015
	3	ed		Daniel	Artigo [Título=AcCORD...]	volume	5	1	08:45:23_ 03/29/2015

Figura 4.8: Operações no repositório R_7 realizadas pelo usuário U_7.

usuário	id	op	origem	destino	chave	atributo	valor Origem	valor Destino	*timestamp*
U7	1	ed		Ana	Artigo [Título=AcCORD...]	volume	3	1	15:29:17_ 03/28/2015
	2	cp	Ana	Bruno	Artigo [Título=AcCORD...]	volume	3	2	15:30:02_ 03/28/2015
	3	cp	Ana	Daniel	Artigo [Título=AcCORD...]	volume	3	2	12:30:42_ 03/29/2015
	4	cp	Ana	Carlos	Artigo [Título=AcCORD...]	volume	3	2	12:30:55_ 03/29/2015

Figura 4.9: Operações no repositório R_7 com operação *1* correta.

usuário	id	op	origem	destino	chave	atributo	valor Origem	valor Destino	*timestamp*
U7	3	ed		Daniel	Artigo [Título=AcCORD]	volume	5	1	08:45:23_ 03/29/2015

Figura 4.10: Operações no R_7 com operação *3* correta.

4.5 Considerações Finais

Neste capítulo foi introduzido o modelo proposto AcCORD, o qual tem como objetivo ser um modelo assíncrono para compartilhamento e integração de dados em um ambiente multiusuário. Nesse modelo colaborativo assíncrono, as atualizações dos usuários são mantidas em um repositório e compartilhadas entre colaboradores importando cada repositório dos demais usuários.

Desde que usuários podem ter diferentes pontos de vista, os repositórios podem estar inconsistentes. Assim, nesta tese também são propostas políticas para resolver conflitos entre repositórios. No total foram propostas seis políticas, a saber: *política baseada na visão local*, *política que remove todos os conflitos*, *política baseada em timestamp*, *política baseada em votação*, *política baseada na confiança nas fontes* e *política baseada na confiança nos usuários*. Como resultado, o modelo AcCORD permite que diferentes políticas para integração e compartilhamento de dados que melhor se adequam às necessidades dos usuários e das aplicações sejam usadas. Ou seja, desde que as políticas propostas usam diferentes critérios para a resolução de conflitos entre tomadas de decisão distintas diante de operações inconsistentes, cada usuário pode aplicar o critério mais adequado às suas necessidades. Adicionalmente, as políticas propostas podem ser usadas pelos usuários colaborativos para criar uma única visão integrada do dado ou então para criar distintas visões locais para cada um deles.

Uma funcionalidade complementar investigada e proposta nesta tese de doutorado refere-se à propagação de valores de atributo dentro de um processo de integração em nível de único usuário. O método proposto propaga o dado de uma fonte, para a qual o usuário tomou uma decisão de integração sobre um valor inconsistente do atributo, para todas as outras fontes que contêm o mesmo atributo

do mesmo objeto, poupando o usuário de trabalho adicional e também aumentando a curácia do processo de integração em nível de único usuário, e consequentemente aumentando a curácia do processo de reconciliação em nível multiusuário. Além disso, a detecção de inconsistências por esse método depende do nível de consistência do repositório gerado usando a ferramenta de integração de dados. Neste projeto de doutorado, foi usado o modelo PrInt, cuja definição de consistência permite que operações com valores inconsistentes, mas cuja origem e/ou destino não se sobrepõem, sejam mantidas no repositório. Assim, conflitos entre as operações criadas automaticamente e as operações já presentes no repositório são possíveis, detectados e resolvidos pelo modelo AcCORD usando o*método de propagação de decisões de integração*.

No Capítulo 5 são descritos os resultados experimentais que investigam as principais características das políticas propostas e também do método de propagação de decisões de integração em nível de único usuário.

Capítulo 5

Resultados Experimentais

O modelo AcCORD, as políticas de reconciliação e o *método de propagação de decisões de integração* em nível de único usuário foram validados por meio de experimentos usando dados reais extraídos dos currículos de pesquisadores que trabalham no Departamento de Ciência da Computação da Universidade de São Paulo, Campus São Carlos.

Foram realizados quatro experimentos. Nos dois primeiros experimentos foi considerada uma reconciliação multiusuário usando como base tanto repositórios com dados reais gerados pela tomada de decisão de integração dos usuários sobre as fontes quanto repositórios com dados sintéticos gerados a partir dos dados reais armazenados nas fontes. Nos dois últimos experimentos foi considerada a participação dos usuários a fim de verificar a percepção desses com relação às políticas de reconciliação e ao *método de propagação de decisões de integração*, além de se comparar o tempo que o usuário leva para resolver conflitos de integração com o tempo despendido pelo método automático.

O protótipo do modelo AcCORD foi implementado na linguagem de programação C++ usando o Qt versão 5.5.1 e compilado com MingW versão 4.9.2. Os experimentos foram realizados em um computador com processador Intel Core $i5$-$4200U$ $1.6GHz$ e $6GB$ de memória principal.

Este capítulo está organizado da seguinte forma. Nas Seções 5.1 e 5.2 são descritos os experimentos de reconciliação multiusuário automática, considerando um menor volume de dados e um maior volume de dados, respectivamente. Na Seção 5.3 são mostrados os resultados obtidos na análise qualitativa quanto às políticas de reconciliação de dados. Na Seção 5.4 são mostrados os resultados experimentais quantitativos e qualitativos realizados com relação ao *método de propagação de decisões de integração*. O capítulo é finalizado na Seção 5.5 com as considerações finais.

5.1 Primeiro Experimento: dados reais, pequeno volume de dados

Nesse primeiro experimento foram usados 4 currículos, escolhidos devido ao grande número de publicações em comum. O tamanho do conjunto de dados é de 3,11 MB. O maior currículo tem 1,20 MB e contém 25 objetos, enquanto que o menor tem 238 KB e contém 3 objetos. Cada objeto tem os seguintes atributos: *título*, *ano*, *local*, *local de publicação*, *linguagem*, *mídia*, *tipo de publicação*, *número de páginas*, *ISSN*, *volume*, *número*, 6 atributos para *palavras-chave*, e os *autores* da publicação, que são compostos pelos atributos *nome*, *nome como citado*, e *ordem de citação*.

A geração dos dados reais de tomada de decisão armazenados nos repositórios foi realizada por 16 estudantes de mestrado e doutorado em Ciência da Computação do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, Campus São Carlos, e da Universidade Federal de São Carlos. Cada usuário interagiu com o modelo PrInt descrito na Seção 2.3 para integrar dados importados dos currículos e gerar seu próprio repositório com operações de *cópia*, *edição*, *remoção* e *inserção*, as quais refletiram as suas decisões de integração. Os dados de procedência armazenados em cada repositório seguiram a definição descrita na Seção 4.2. O maior repositório tem 390 KB e contém 749 operações, enquanto que o menor repositório tem 40 KB e contém 79 operações.

Foi considerado o trabalho de reconciliação colaborativa realizado por vários usuários, e que o usuário U_1 importou os conjuntos de "dados de procedência em nível único usuário" dos outros usuários e aplicou uma dentre as políticas de reconciliação propostas. São mostrados nos resultados descritos nesta seção, os efeitos de gerenciar operações conflitantes para o repositório R_1, que é o repositório de U_1. O objetivo desses experimentos foi investigar os efeitos de adotar cada política em relação: (i) ao número total de operações removidas, que causa a perda de decisões dos usuários; (ii) ao efeito do aumento no número de colaboradores; (iii) ao efeito de cada política na remoção de operações no repositório local, refletindo a perda de operações do usuário que está realizando o processo de reconciliação; e (iv) ao tempo de execução gasto pelas políticas. Na Seção 5.1.1 são mostrados os resultados quando o usuário U_1 aplica o processo de reconciliação pela primeira vez. Na Seção 5.1.2 são discutidos os resultados quando o usuário U_1 aplica o processo de reconciliação várias vezes consecutivas. Na Seção 5.1.3 são investigados os comportamentos das políticas baseadas na remoção de todos os conflitos e na votação, quando o usuário U_1 aplica o processo de reconciliação várias vezes consecutivas.

5.1.1 Primeiro Processo de Reconciliação

Nesse teste, o número de usuários colaborativos variou de 2 a 16. Para cada cenário, foi considerado que o usuário U_1 requereu o processo de reconciliação pela primeira vez. Assim, o repositório R_1 sempre armazenou somente as operações do primeiro processo de integração do usuário U_1, ou seja, o repositório R_1 não armazenou operações de qualquer processo de reconciliação anterior.

Na Figura 5.1 é mostrado: (i) o número total de operações armazenadas no repositório R_1 depois que o usuário U_1 importou todos os conjuntos de dados de procedência, mas antes de aplicar a política de reconciliação; e (ii) o número de operações no repositório R_1 depois do usuário U_1 executar o processo de reconciliação, para cada política proposta. Os resultados demonstraram que a *política que remove todos os conflitos* e a *política baseada em votação* removeram mais operações que as demais políticas. Além disso, elas tiveram o mesmo comportamento. O resultado obtido com a *política que remove todos os conflitos* era esperado, porque ela remove todas as operações conflitantes, mas os resultados obtidos com a *política baseada em votação* não eram esperados. Entretanto, analisando os repositórios, foi verificado que não era possível haver operações vencedoras entre as operações conflitantes. Nessa situação, de empate na votação, a *política baseada em votação* tem o mesmo comportamento que a *política que remove todos os conflitos*. Já a *política baseada na confiança nos usuários* removeu menos operações, ou seja, causou a perda de um menor número de decisões de integração dos usuários e, portanto, obteve o melhor resultado.

Na Figura 5.2 é ilustrado o número de operações refeitas durante o processo de reconciliação. A *política baseada na visão local* e a *política que remove todos os conflitos* não possuem a funcionalidade de refazer operações conflitantes e não são mostradas no gráfico. A *política baseada em votação* teve o mesmo comportamento da *política que remove todos os conflitos* e não refez qualquer operação. A *política baseada na confiança nos usuários* refez o maior número de operações na maioria dos casos. Apenas para 4 e 6 usuários que a *política baseada em timestamp* e a *política baseada na confiança nas fontes* obtiveram melhores resultados que a *política baseada na confiança nos usuários*. Porém, a diferença entre o número de operações refeitas entre as políticas foi bem pequena. Para 4 usuários, a *política baseada em timestamp* e a *política baseada na confiança nas fontes* refizeram, respectivamente, 19 e 4 operações a mais do que a *política baseada na confiança nos usuários*. Para 6 usuários, a *política baseada em timestamp* e a *política baseada na confiança nas fontes* refizeram, respectivamente, 1 e 10 operações a mais do que a *política baseada na confiança nos usuários*. A *política baseada na confiança nas fontes* também obteve melhores resultados para 12 usuários, quando comparada com a *política baseada na confiança nos usuários*. Pode-se concluir, portanto, que na maioria dos casos, a *política baseada na confiança nos usuários* proveu o melhor

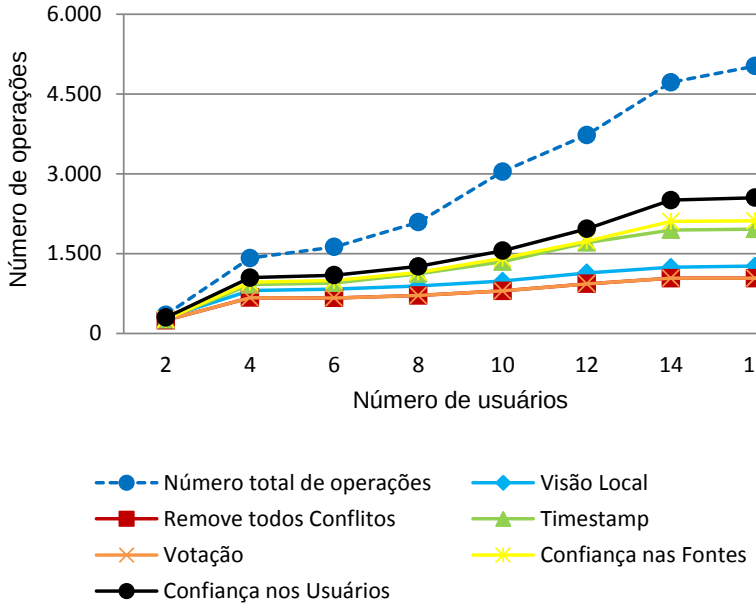


Figura 5.1: Número de operações no repositório R_1 após o usuário U_1 importar os dados dos outros usuários e após realizar o processo de reconciliação, aplicando as diferentes políticas propostas considerando um menor volume de dados.

resultado, refazendo o maior número de operações. Refazer o maior número de operações significa que mais operações geradas pelas decisões de integração tomadas pelos usuários colaborativos encontram-se presentes no repositório final.

Na Figura 5.3 tem-se: (i) o número inicial de operações no repositório R_1 depois que o usuário U_1 importou todos os repositórios de dados, mas antes do usuário U_1 aplicar uma política de reconciliação; e (ii) o número de operações do usuário U_1 no repositório R_1 depois do processo de reconciliação. Este teste teve como objetivo mostrar as perdas das decisões tomadas pelo usuário local U_1. Como esperado, a *política baseada na visão local* não removeu qualquer operação. Adicionalmente, como é o primeiro processo de reconciliação, a confiança nos usuários para a *política baseada na confiança nos usuários* é determinada somente para o usuário local e, portanto, a maior confiança é a do próprio usuário U_1, sendo que suas operações não foram removidas. A *política baseada em timestamp* e a *política baseada na confiança nas fontes* removeram um número médio de operações do usuário U_1, porque essas políticas são capazes de atualizar e manter algumas operações conflitantes e suas dependentes. Pela mesma razão discutida no teste anterior, a *política que remove todos os conflitos* e a *política baseada em votação* tiveram o mesmo comportamento, removendo um grande número de operações do usuário U_1, ou seja, perdendo muitas decisões de integração que es-

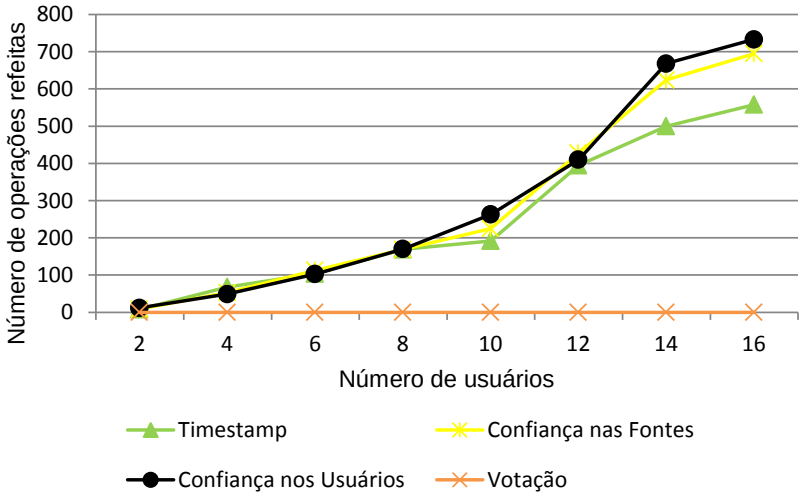


Figura 5.2: Número de operações refeitas após o usuário U_1 solicitar o processo de reconciliação, considerando as diferentes políticas propostas aplicadas à um menor volume de dados.

tavam previamente no repositório R_1, feitas pelo usuário que solicitou o processo de reconciliação, o usuário U_1.

Na Figura 5.4 mostra-se o tempo de execução gasto pelas políticas para manipular as operações dos repositórios. Para as análises de desempenho realizadas neste teste, cada política foi executada 5 vezes e foi calculada a média aritmética e o intervalo de confiança de 95% dos resultados obtidos. A quantidade de execuções foi definida considerando-se que foram obtidos intervalos de confiança suficientemente pequenos e que possibilitaram uma conclusão sobre o desempenho das políticas. De acordo com os resultados, o tempo de execução aumentou linearmente com o aumento do número de operações. Além disso, a *política baseada na confiança nas fontes* teve o tempo aumentado pelo processamento de todas as fontes de dados (Figura 5.4(a)). Nesse experimento, a *política baseada na confiança nos usuários* obteve os melhores resultados. Na Figura 5.4(b) são mostrados os mesmos resultados que na Figura 5.4(a) omitindo-se os resultados obtidos pela *política baseada na confiança nas fontes* para que as diferenças entre as demais políticas ficassem mais nítidas.

A *política baseada na confiança nos usuários* obteve os menores tempos de execução para manipular as operações no repositório e, além disso, removeu o menor número de operações e refez o maior número de operações, tendo assim, o melhor desempenho nesse teste. A *política baseada na confiança nas fontes* tem seu tempo de execução prejudicado pelo processamento das fontes de dados, não realizado pelas demais políticas, mas obteve bons resultados em relação ao número de

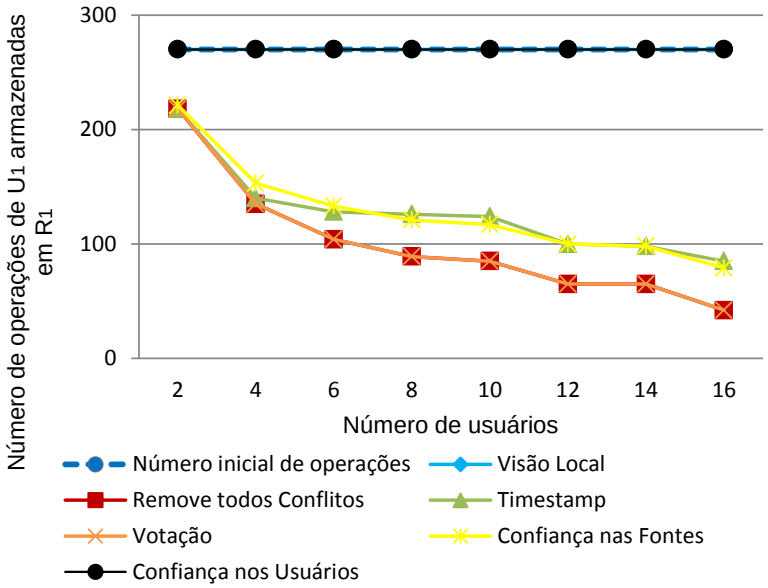
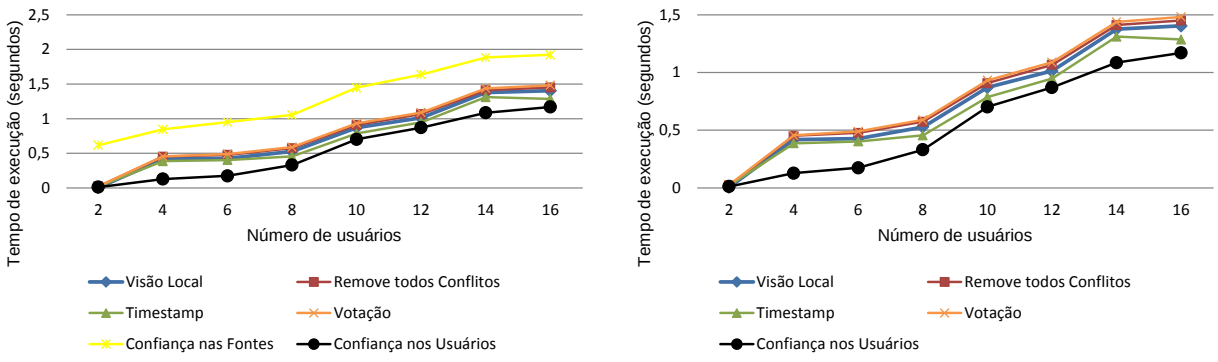


Figura 5.3: Número de operações do repositório R_1 após a aplicação de cada política proposta, considerando um menor volume de dados.

operações removidas e refeitas, juntamente com a *política baseada em timestamp* que, além de apresentar bons resultados em relação a essas características, também apresentou o segundo melhor tempo de execução.



(a) Considerando também ta *política baseada na confiança nas fontes*.

(b) Desconsiderando a *política baseada na confiança nas fontes*.

Figura 5.4: Tempo de execução gasto pelas políticas para manipular as operações dos repositórios, considerando um menor volume de dados.

5.1.2 Processos de Reconciliação Consecutivos

Neste teste, o número de usuários colaborativos foi fixado em 4. Considerou-se que o usuário U_1 executou o processo de reconciliação 5 vezes consecutivamente. Assim, no processo de reconciliação 1 no tempo t_1, o repositório R_1 armazenou operações dos repositórios $R_1, \ldots, R_4$; no processo de reconciliação 2 no tempo t_2, o repositório R_1 armazenou operações dos repositórios $R_1, \ldots, R_4$, tal que esses repositórios também armazenaram as novas operações geradas nos tempos t_1 até t_2; e assim por diante.

Na Figura 5.5 é mostrada a porcentagem de operações removidas no repositório R_1 aplicando cada política. Os resultados mostraram que a *política baseada na confiança nos usuários* garantiu os melhores resultados, desde que ela removeu a menor porcentagem de operações conflitantes durante cada processo de reconciliação, ou seja, mais decisões de integração conflitantes dos usuários colaborativos foram mantidas no repositório após o processo de reconciliação. A *política baseada na visão local*, a *política baseada em timestamp* e a *política baseada na confiança nas fontes* removeram uma porcentagem média de operações, enquanto que a *política que remove todos os conflitos* e a *política baseada em votação* tiveram o mesmo comportamento e removeram uma maior porcentagem de operações que as outras políticas.

A porcentagem de remoções é dada em relação ao número de operações antes de cada processo de reconciliação, como mostrado na Figura 5.6. Os resultados mostraram que para todas as políticas, mesmo para a *política que remove todos os conflitos* e a *política baseada em votação*, o tamanho do repositório cresce depois da execução de cada novo processo de reconciliação. Por exemplo, considere a *política baseada em timestamp*. Antes do primeiro processo de reconciliação, o repositório R_1 tinha 1.417 operações, e no final do quinto processo, o repositório continha 2.026 operações. Já para a *política baseada na confiança nos usuários*, o repositório R_1 continha ao final do quinto processo, 2.367 operações. A diferença representa a contribuição que os outros usuários colaborativos fizeram para o trabalho do usuário U_1 nos processos de reconciliação.

Os resultados também foram analisados de forma a reportar o efeito de cada política considerando somente as operações presentes no repositório R_1 no início de cada processo. O objetivo desse teste é verificar o número de remoções de operações que estavam previamente no repositório do usuário que está solicitando o processo de reconciliação, ou seja, no repositório R_1. Os resultados, mostrados na Figura 5.7, demonstraram que a *política baseada na visão local* e a *política baseada na confiança nos usuários* não removeram quaisquer operações. A *política que remove todos os conflitos* e a *política baseada em votação* obtiveram os piores resultados, removendo uma maior porcentagem de operações

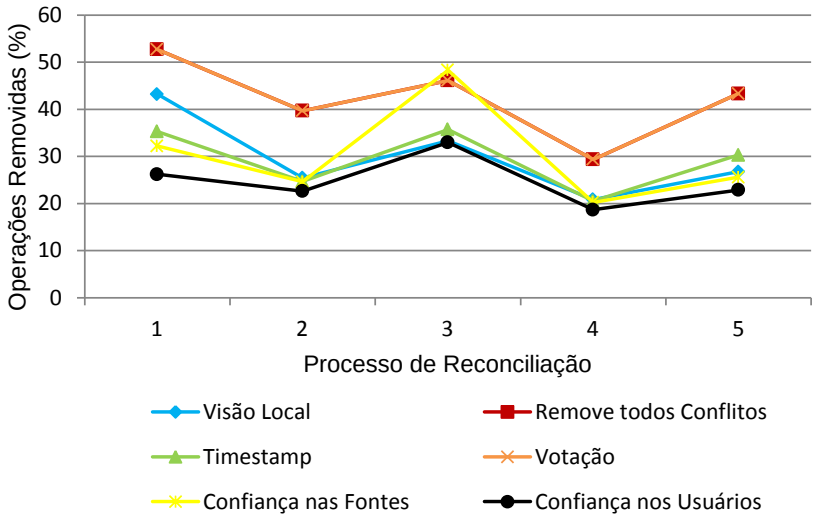


Figura 5.5: Porcentagem de operações removidas considerando cada política e um menor volume de dados.

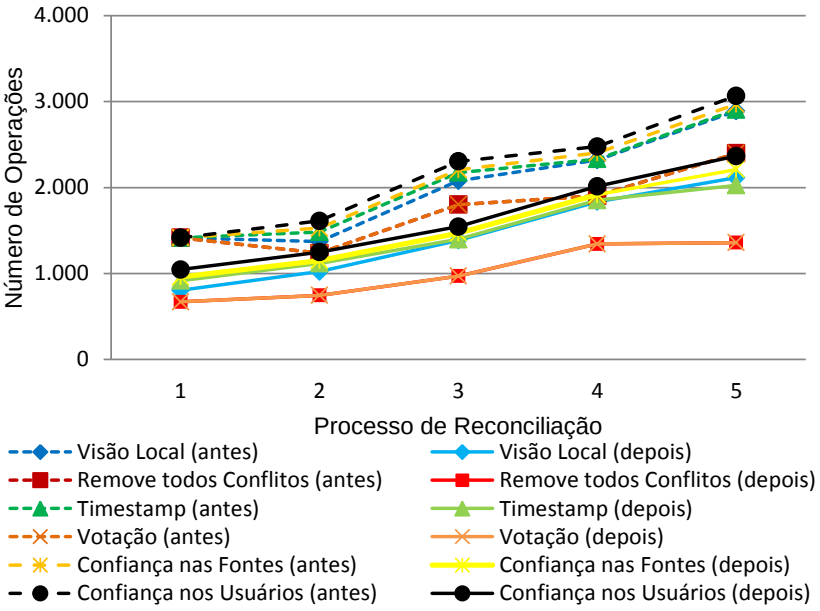


Figura 5.6: Número de operações removidas considerando cada política e um menor volume de dados.

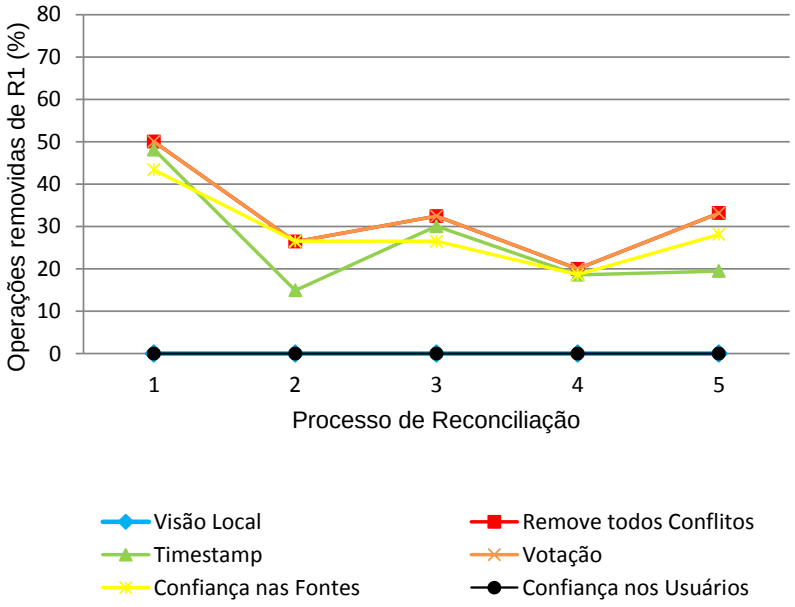
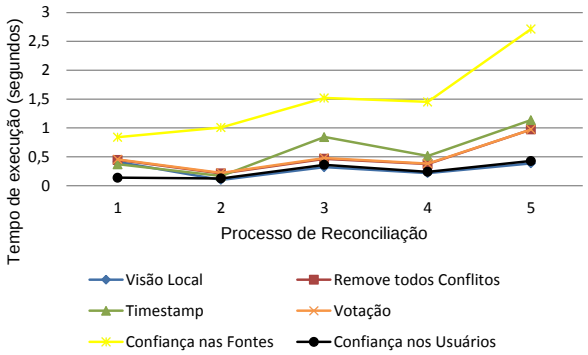


Figura 5.7: Porcentagem de operações do repositório R_1 removidas, aplicando cada política e considerando um menor volume de dados.

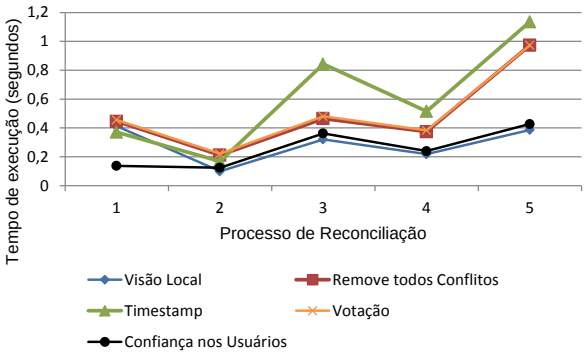
em todos os processos de reconciliação, exceto no segundo, no qual tiveram uma porcentagem de remoções levemente inferior à *política baseada na confiança nas fontes*, sendo $26,42\%$ para a *política que remove todos os conflitos* e a *política baseada em votação*, e $26,53\%$ para a *política baseada na confiança nas fontes*.

Os tempos de execução gastos pelas políticas para manipular as operações são mostrados na Figura 5.8. Novamente, a *política baseada na confiança nas fontes* requereu os maiores tempos de execução pelo processamento de todas as fontes de dados (Figura 5.8(a)). Nesse experimento, a *política baseada na visão local* e a *política baseada na confiança nos usuários* obtiveram os melhores resultados. Na Figura 5.8(b) são mostrados os mesmos resultados que na Figura 5.8(a) omitindo-se os resultados obtidos pela *política baseada na confiança nas fontes* para que as diferenças entre as demais políticas ficassem mais nítidas.

Analisando-se todos os resultados descritos nessa seção e mostrados nas Figuras 5.5 a 5.7, conclui-se que, além de obter bons tempos de execução, a *política baseada na confiança nos usuários* também removeu a menor porcentagem de operações. A *política baseada na visão local* obteve bons resultados para a porcentagem de remoções, especialmente para os processos de reconciliação 2, 3 e 4.



(a) Considerando a *política baseada na confiança nas fontes*.



(b) Desconsiderando a *política baseada na confiança nas fontes*.

Figura 5.8: Tempos de execução de cada política em processos de reconciliação consecutivos considerando um menor volume de dados.

5.1.3 Investigação das políticas baseadas na remoção de todos os conflitos e na votação

Como nos testes descritos na Seção 5.1.2 a *política que remove todos os conflitos* e a *política baseada em votação* tiveram o mesmo comportamento, foi considerado um novo cenário de teste com somente 3 usuários colaborativos a fim de determinar como essas duas políticas diferem. Nesse novo cenário de teste, garantiu-se que haveriam desempates nas decisões de integração quando da aplicação da *política baseada em votação*. Essa seção tem por objetivo descrever os resultados obtidos nos testes realizados, considerando não somente essas duas políticas, mas também as demais políticas propostas para efeitos comparativos.

Na Figura 5.9 são mostradas as porcentagens de operações removidas do repositório R_1 depois de aplicar cada política após 7 processos de reconciliação consecutivos. Os resultados mostraram uma menor porcentagem de remoções para a *política baseada em votação* e uma maior porcentagem de remoções para a *política que remove todos os conflitos*. Apesar de nos dois primeiros processos de reconciliação a *política baseada em votação* e a *política baseada na confiança nos usuários* apresentarem resultados similares, essa tendência não permaneceu nos demais processos de reconciliação subsequentes. Em detalhes, o primeiro processo de reconciliação iniciou-se com 1.338 operações para a *política baseada em votação* e a *política que remove todos os conflitos*, enquanto que o último processo de reconciliação iniciou-se com 1.690 operações para a *política que remove todos os con-*

flitos e 3.159 para a *política baseada em votação*. No final do sétimo processo de reconciliação, o repositório R_1 continha 1.375 operações aplicando a *política que remove todos os conflitos* e 2.911 aplicando a *política baseada em votação*. Esse resultado foi alcançado porque existiram operações vencedoras entre as operações conflitantes considerando a *política baseada em votação*.

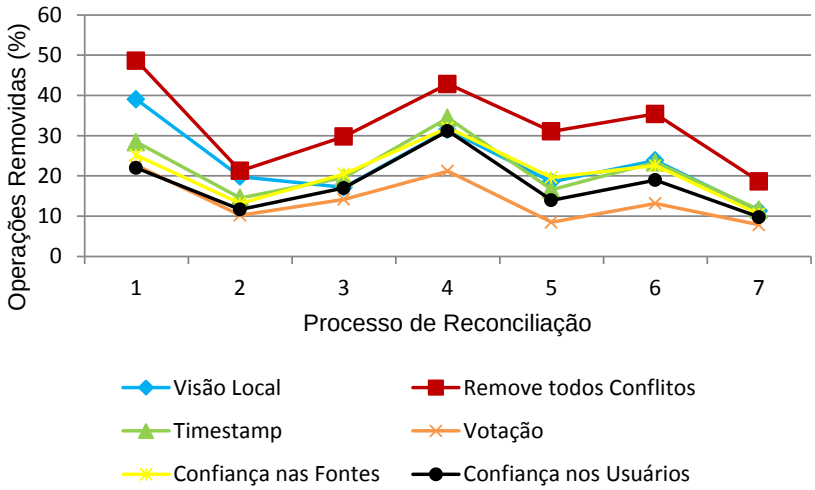


Figura 5.9: Porcentagem de operações removidas aplicando cada política ao trabalho colaborativo de 3 usuários, considerando um menor volume de dados.

Na Figura 5.10 é mostrada a porcentagem de operações refeitas durante o processo de reconciliação. A *política baseada na visão local* e a *política que remove todos os conflitos* não refazem qualquer operação e, portanto, não são mostradas no gráfico. A *política baseada em votação* obteve os melhores resultados refazendo uma porcentagem consideravelmente maior de operações e minimizando o número de perdas de operações. A *política baseada na confiança nos usuários* refez a menor porcentagem de operações, de maneira geral. Já a *política baseada na confiança nas fontes* e a *política baseada em timestamp* refizeram uma porcentagem média de operações.

Os tempos de execução gastos pelas políticas para manipular as operações são mostrados na Figura 5.11. Da mesma forma que os resultados obtidos e mostrados na Figura 5.8, a *política baseada na confiança nas fontes* teve os maiores tempos de execução pelo processamento de todas as fontes de dados. Além disso, a *política baseada na visão local* e a *política baseada na confiança nos usuários* obtiveram os melhores resultados novamente. Como nesse teste houve operações vencedoras na votação, a *política baseada em votação* apresentou maiores tempos de execução que a *política que remove todos os conflitos*, confirmando a análise assintótica mostrada na Seção 4.3.4.

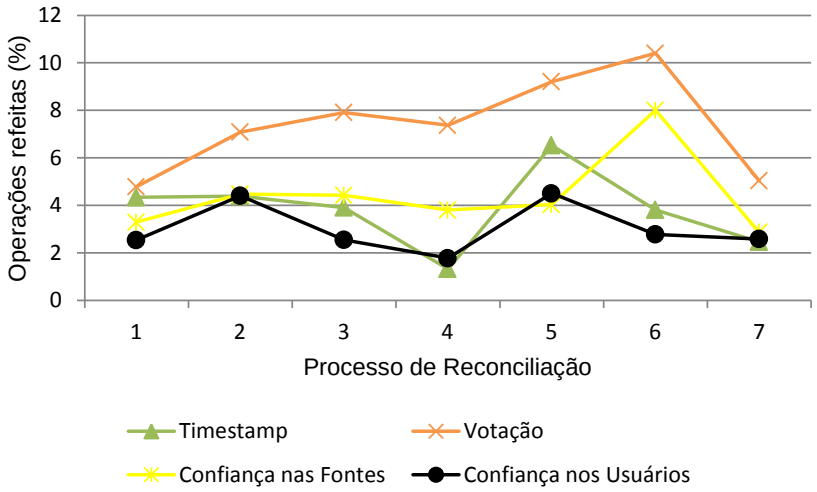


Figura 5.10: Porcentagem de operações refeitas aplicando cada política ao trabalho colaborativo de 3 usuários, considerando um menor volume de dados.

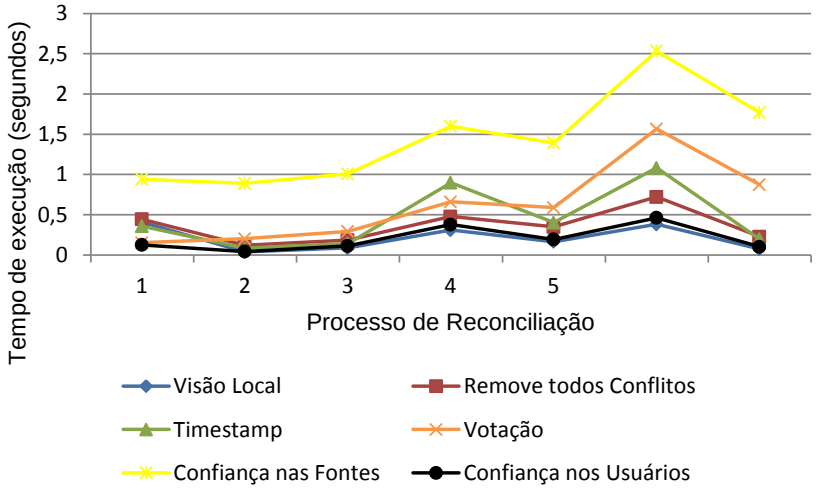


Figura 5.11: Tempo de execução gasto pelas políticas durante o processo de reconciliação de 3 usuários, considerando um menor volume de dados.

5.2 Segundo Experimento: grande volume de dados

Nesse segundo experimento foram usados 120 currículos de pesquisadores do Departamento de Ciência da Computação da Universidade de São Paulo, Campus São Carlos. Os 120 currículos foram divididos em 12 grupos contendo 12 currículos para gerar cada repositório, sendo que 2 desses currículos estavam em comum em dois grupos, com o objetivo de gerar operações conflitantes. Foram gerados 12 repositórios com um total de 27.219 operações. O maior repositório tem 1.626 KB e 2.755 operações, enquanto que o menor repositório tem 1.018 KB e 1.466 operações. Os repositórios foram gerados usando o modelo PrInt. As porcentagens de operações de cada tipo (*edição*, *cópia*, *remoção* e *inserção*) foram obtidas por meio da média arimética das porcentagens dessas operações em cada repositório gerado pelos processos de integração realizados pelos usuários no primeiro experimento.

Assim como no primeiro experimento, nesse segundo experimento foi considerado o trabalho de reconciliação colaborativa realizado por vários usuários, e que o usuário U_1 importou os conjuntos de "dados de procedência em nível único usuário" dos outros usuários e aplicou uma dentre as políticas de reconciliação propostas. No entanto, foi usada uma quantidade maior de operações geradas sinteticamente usando o modelo PrInt. São mostrados nos resultados descritos nesta seção os efeitos de gerenciar operações conflitantes para o repositório R_1, que é o repositório de U_1. O objetivo desse experimento foi investigar os efeitos de se gerenciar um grande volume de dados pelas políticas, em relação: (i) ao número total de operações removidas, que causa a perda de decisões dos usuários; (ii) ao efeito do aumento no número de colaboradores; (iii) ao efeito de cada política na remoção de operações no repositório local, refletindo a perda de operações do usuário que está realizando o processo de reconciliação; e (iv) ao tempo de execução gasto pelas políticas.

Na Seção 5.2.1 são mostrados os resultados quando o usuário U_1 aplica o processo de reconciliação pela primeira vez. Na Seção 5.2.2 são discutidos os resultados quando o usuário U_1 aplica o processo de reconciliação várias vezes consecutivas.

5.2.1 Primeiro Processo de Reconciliação

Nesse teste, o número de usuários colaborativos variou de 2 até 12. Para cada cenário, considerou-se que o usuário U_1 requereu o primeiro processo de reconciliação. Assim, o repositório R_1 sempre armazenou apenas as operações do primeiro processo de integração do usuário U_1, ou seja, o repositório R_1 não armazenou operações relacionadas com qualquer processo de reconciliação anterior.

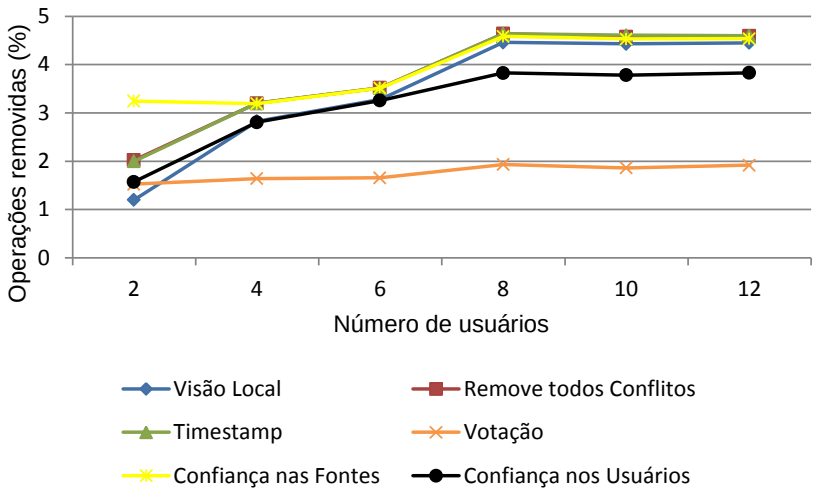


Figura 5.12: Porcentagem de operações removidas considerando cada política e um maior volume de dados.

Na Figura 5.12 é mostrada a *porcentagem de operações removidas* do repositório R_1 depois que o usuário U_1 importou todos os conjuntos de dados de procedência e executou o processo de reconciliação com cada política. Os resultados demonstraram que a *política que remove todos os conflitos*, a *política baseada em timestamp* e a *política baseada na confiança nas fontes* removeram mais operações que as outras políticas. A *política baseada na visão local* também removeu grande porcentagem de operações. Em contrapartida, a *política baseada em votação* removeu uma porcentagem consideravelmente menor de operações e obteve o melhor resultado. Isso significa que a *política baseada em votação* foi a que mais manteve decisões de integração dos usuários colaborativos.

Na Figura 5.13 é mostrada a porcentagem de operações refeitas durante o processo de reconciliação. A *política baseada na visão local* e a *política que remove todos os conflitos* não refazem qualquer operação e, portanto, não são mostradas no gráfico. A *política baseada em votação* obteve os melhores resultados refazendo um número consideravelmente maior de operações e minimizando o número de perdas de operações. A *política baseada na confiança nos usuários* refez o segundo maior número de operações. Já a *política baseada na confiança nas fontes* e a *política baseada em timestamp* refizeram os menores números de operações e, portanto, juntamente com a *política que remove todos os conflitos* geraram a perda de mais operações.

Na Figura 5.14 é representado o tempo de execução gasto pelas políticas para manipular as operações. De acordo com os resultados, o tempo de execução aumentou linearmente com o au-

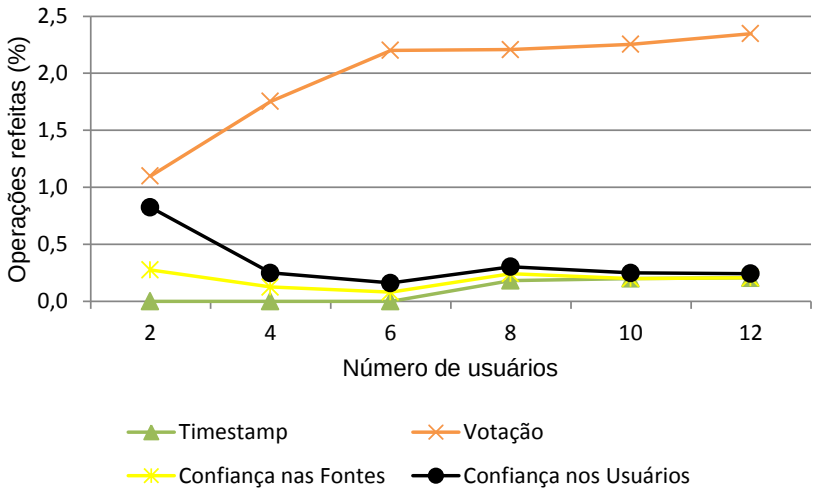
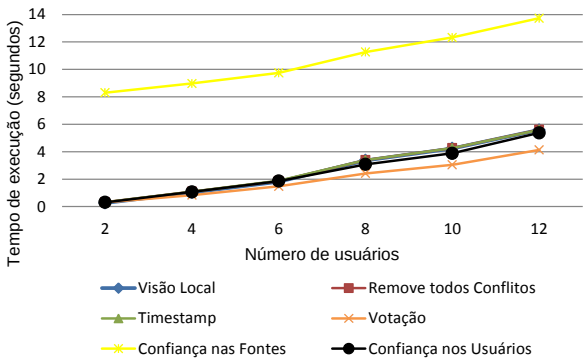


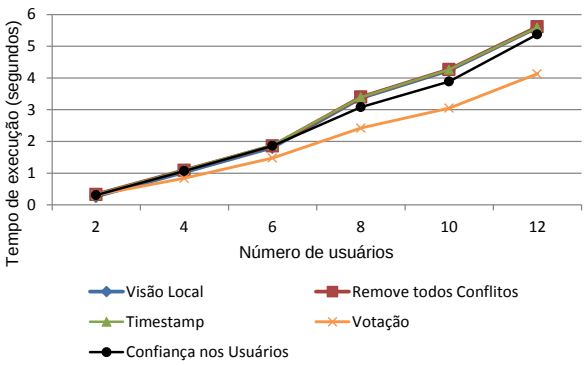
Figura 5.13: Porcentagem de operações refeitas considerando cada política e um maior volume de dados.

mento do número de operações. Além disso, a *política baseada na confiança nas fontes* teve o tempo aumentado pelo processamento de todas as fontes de dados (Figura 5.14(a)). Na Figura 5.14(b) são mostrados os mesmos resultados que na Figura 5.14(a) omitindo-se os resultados obtidos pela *política baseada na confiança nas fontes* para que as diferenças entre as demais políticas ficassem mais nítidas. Nesse experimento, a *política baseada em votação* obteve os melhores tempos de execução.

Comparando-se os resultados descritos na Seção 5.1.1 com os resultados descritos na Seção 5.2.1, o comportamento das políticas se manteve. Sendo assim, a *política baseada na confiança nos usuários* obteve os melhores resultados para número de remoções, número de operações refeitas e tempo de execução, desconsiderando a *política baseada em votação* que, no teste descrito na Seção 5.1.1, apresentou comportamento similar à *política que remove todos os conflitos* devido ao empate nas votações. Da mesma maneira, a *política que remove todos os conflitos* e a *política baseada na visão local* tiveram o maior número de remoções e maior tempo de execução, desconsiderando a *política baseada na confiança nas fontes* que teve novamente o pior tempo de execução. No teste mostrado nesta Seção 5.2.1, a *política baseada em timestamp* apresentou tempo de execução similar à *política que remove todos os conflitos* e à *política baseada na visão local*. Já com os dados reais usados no teste mostrado na Seção 5.1.1, a *política baseada em timestamp* obteve tempos de



(a) Considerando também a *política baseada na confiança nas fontes*.



(b) Desconsiderando a *política baseada na confiança nas fontes*.

Figura 5.14: Tempo de execução gasto pelas políticas para manipular maior volume de operações.

execução ligeiramente melhores que a *política que remove todos os conflitos* e a *política baseada na visão local*.

5.2.2 Processos de Reconciliação Consecutivos

Nesta seção são detalhados os resultados obtidos para a execução de 5 processos consecutivos de reconciliação. Neste teste, o número de usuários colaborativos foi fixado em 3. Considerou-se que o usuário U_1 executou o processo de reconciliação 5 vezes consecutivamente. Assim, no processo de reconciliação 1 no tempo t_1, o repositório R_1 armazenou operações dos repositórios $R_1,\ldots,R_3$; no processo de reconciliação 2 no tempo t_2, o repositório R_1 armazenou operações dos repositórios $R_1,\ldots,R_3$, tal que esses repositórios também armazenaram as novas operações geradas nos tempos t_1 até t_2; e assim por diante.

Na Figura 5.15 é mostrada a *porcentagem de operações removidas* do repositório R_1 depois que o usuário U_1 importou os repositórios de outros 2 usuários e executou 5 processos de reconciliação consecutivos utilizando a mesma política nos 5 processos. Os resultados mostraram que a *política que remove todos os conflitos*, a *política baseada em timestamp* e a *política baseada na confiança nas fontes* removeram mais operações que as outras políticas. A *política baseada na confiança nos usuários* também removeu grande porcentagem de operações. A *política baseada em votação* e a *política baseada na visão local* removeram as menores porcentagens de operações, na média. A *política baseada na confiança nos usuários* iniciou os processos de reconciliação removendo grande

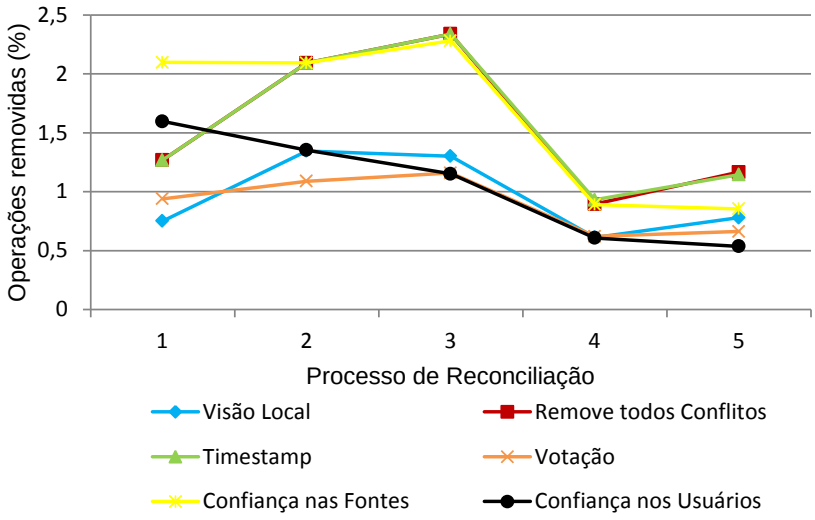


Figura 5.15: Porcentagem de operações removidas considerando cada política e um maior volume de dados.

porcentagem de operações e superou a *política baseada em votação* apenas a partir do terceiro processo de reconciliação. Quanto maior a quantidade de operações removidas, maior a quantidade de decisões de integração dos usuários que são descartadas. Assim, para esse primeiro resultado, a *política baseada em votação* proveu melhores resultados.

Na Figura 5.16 é mostrada a porcentagem de operações refeitas durante o processo de reconciliação. Os resultados são similares àqueles realizados considerando apenas o primeiro processo de reconciliação (Seção 5.2.1). A *política baseada na visão local* e a *política que remove todos os conflitos* não refazem qualquer operação e, portanto, não são mostradas no gráfico. A *política baseada em votação* obteve os melhores resultados refazendo uma porcentagem consideravelmente maior de operações e minimizando o número de perdas de operações. A *política baseada na confiança nos usuários* refez a segunda maior porcentagem de operações. Já a *política baseada na confiança nas fontes* e a *política baseada em timestamp* refizeram as menores porcentagens de operações.

Na Figura 5.17 é mostrado o tempo de execução gasto pelas políticas para manipular as operações. De acordo com os resultados, a *política baseada na confiança nas fontes* teve os maiores tempos de execução, devido ao processamento de todas as fontes de dados (Figura 5.17(a)). A *política baseada em votação* obteve tempos de execução altos, sendo superada apenas no primeiro processo de reconciliação por todas as políticas exceto a *política baseada na visão local* e no quinto processo de reconciliação pela *política que remove todos os conflitos* e a *política baseada em timestamp*. A

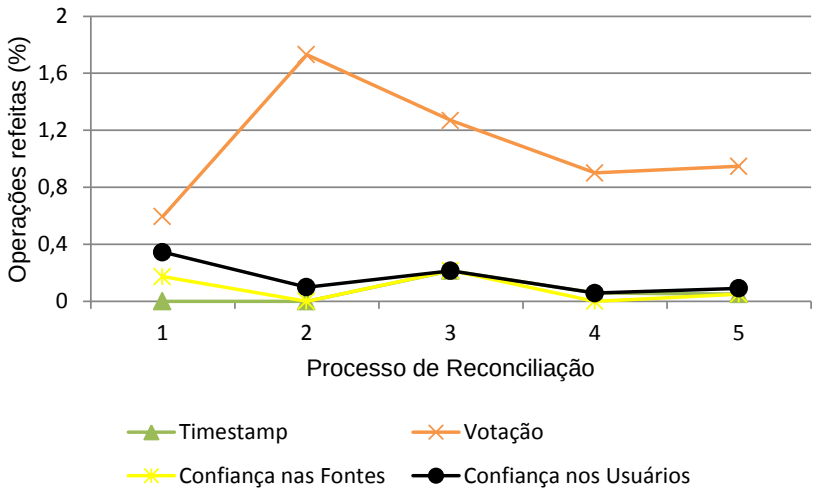
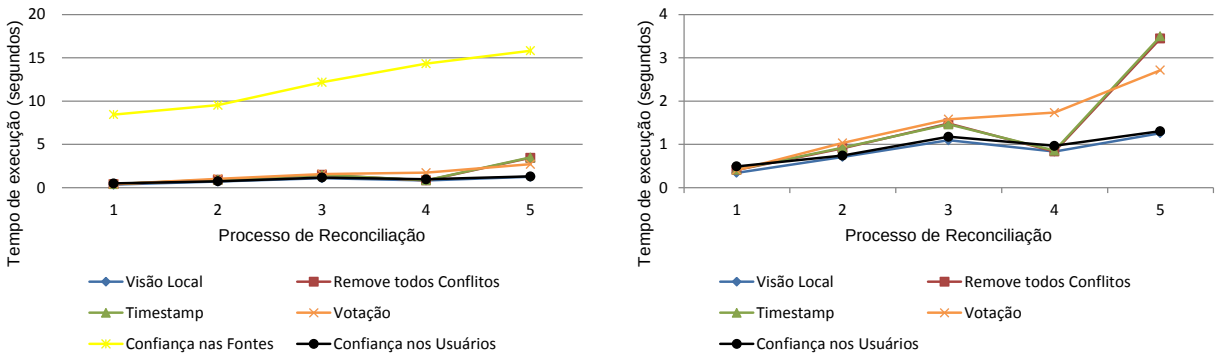


Figura 5.16: Porcentagem de operações refeitas considerando cada política e um maior volume de dados.

política baseada na visão local e a *política baseada na confiança nos usuários* obtiveram melhores tempos de execução, no geral (Figura 5.17(b)).

Considerando os resultados mostrados nesta Seção 5.2.2, nota-se que a *política baseada em votação* obteve os piores tempos de execução, excetuando os tempos de execução obtidos pela *política baseada na confiança nas fontes*, mas em contrapartida, refez consideravelmente maior porcentagem de operações e, juntamente com a *política baseada na confiança nos usuários*, removeu menor porcentagem de operações. A *política baseada na confiança nos usuários*, além de remover menor porcentagem de operações, refez a segunda maior porcentagem de operações e, juntamente com a *política baseada na visão local*, obteve bons tempos de execução.

Comparando-se os resultados descritos na Seção 5.1.3 com os resultados descritos na Seção 5.2.2, nos resultados da primeira, a *política que remove todos os conflitos* removeu a maior porcentagem de operações enquanto que a *política baseada em votação* removeu a menor porcentagem. Já nos resultados descritos na segunda seção, a *política que remove todos os conflitos* juntamente com a *política baseada em timestamp* e a *política baseada na confiança nas fontes* removeram as maiores porcentagens de operações e, a *política baseada em votação* juntamente com a *política baseada na confiança nos usuários* removeram as menores porcentagens de operações. Na Seção 5.1.3, a *política baseada em votação* refaz a maior porcentagem de operações enquanto que a *política baseada na confiança nos usuários* refaz a menor porcentagem, no geral. Já na Seção 5.2.2, a *política baseada*



(a) Considerando também a *política baseada na confiança nas fontes*.

(b) Desconsiderando a *política baseada na confiança nas fontes*.

Figura 5.17: Tempo de execução gasto pelas políticas para manipular um maior volume operações.

em votação refaz a maior porcentagem de operações e a *política baseada na confiança nos usuários* refaz a segunda maior porcentagem de operações, ficando a *política baseada em timestamp* e a *política baseada na confiança nas fontes* com os piores resultados. Quanto ao tempo de execução, em ambos os testes a *política baseada na confiança nas fontes* obteve os maiores tempos, e a *política baseada em votação* os segundos maiores tempos, de maneira geral, enquanto que *política baseada na visão local* e a *política baseada na confiança nos usuários* obtiveram os menores tempos de execução.

5.3 Experimento baseado no Usuário

Como a experiência dos usuários é importante para a reconciliação de dados, é necessário compreender as suas características e necessidades, pois são eles os alvos reais da reconciliação. Para a realização do experimento descrito nesta seção foi apresentada aos 7 usuários, estudantes da graduação do curso de Ciência da Computação da Universidade Federal do Tocantins e pós graduação em Ciência da Computação da Universidade Federal de Uberlândia, uma breve explicação sobre cada uma das políticas propostas e os resultados gerados a partir de um conjunto inicial de operações fictícias feitas por 6 usuários. Posteriormente, foi aplicado um questionário contendo 62 questões, as quais investigaram a opinião de cada usuário em relação àquela política, e as possíveis aplicações para as quais as políticas são adequadas. Essas questões também envolveram investigações relacionadas à percepção de consistência, aceitabilidade, corretude e risco, para as quais cada estudante deveria

marcar o seu nível de concordância em uma escala de 1 a 7, que variava de um completamente desacordo a um completamente acordo. Algumas afirmativas continham ideias contraditórias entre si, a fim de detectar esforço insuficiente para responder à pesquisa de respostas legítimas de usuários.

Algumas afirmativas do questionário são mostradas na Tabela 5.1 e o questionário completo é mostrado no Apêndice A.1. As respostas do usuários foram agrupadas e resumidas a fim de melhor compreender a percepção deles em relação às políticas e também analisar qualitativamente os resultados. Parte do questionário aplicado foi baseado no questionário descrito em [Hossain et al. 2014].

Tabela 5.1: Questionário usado no experimento envolvendo a aceitação do usuário.

Variável de medição	Questionário
Percepção de consistência	A política baseada <na visão local|na remoção de conflitos|em *timestamp*| na votação|na confiança nas fontes|na confiança nos usuários> fornece dados consistentes para o usuário local.
Percepção de aceitabilidade	A política baseada <na visão local|na remoção de conflitos|em *timestamp*|na votação|na confiança nas fontes|na confiança nos usuários> é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.
Percepção de corretude	Eu acho que a política baseada <na visão local|na remoção de conflitos|em *timestamp*|na votação|na confiança nas fontes|na confiança nos usuários> pode fornecer resultados eficazes.
Percepção de risco	A política baseada <na visão local|na remoção de conflitos|em *timestamp*| na votação|na confiança nas fontes|na confiança nos usuários> pode resolver inconsistências sem muita intervenção humana.
Economia de tempo	A resolução de inconsistências feita automaticamente é importante para poupar tempo do usuário.
Satisfação com o resultado	O usuário local ficará bastante satisfeito com o resultado gerado pela política baseada <na visão local|na remoção de conflitos|em *timestamp*|na votação|na confiança nas fontes|na confiança nos usuários>.

Na Tabela 5.2 são mostradas as porcentagens de concordância dos usuários com relação a cada variável de medição de percepção para a *política baseada na visão local*. Quanto à percepção de aceitabilidade da *política baseada na visão local*, $71,43\%$ dos usuários acharam a política inadequada para resolver inconsistências entre as operações dos usuários colaborativos. Isso se deve ao fato do usuário ficar restrito às suas próprias informações sem poder comparar as suas decisões com as decisões de outros. Além disso, caso o usuário local tome uma decisão errada sobre um atributo, uma possível decisão correta tomada por outro usuário seria desconsiderada. Em contrapartida, $28,57\%$ dos usuários tiveram um nível de aceitação razoável sobre a política, afirmando que seria necessário que o usuário tivesse bastante confiança em seus dados ou suas decisões. Com relação à percepção de consistência, $71,43\%$ dos usuários acreditaram que a política não é capaz de fornecer

dados consistentes para o usuário local. Quanto à percepção de corretude e à percepção de risco, a maioria dos usuários acreditaram, respectivamente, que a política não é capaz de fornecer dados corretos para o usuário local e que a política também não é capaz de resolver inconsistências sem muita intervenção humana. Com relação à economia de tempo e à satisfação com o resultado, 71,43% do usuários acreditaram que a resolução de inconsistências feita automaticamente é importante para poupar tempo do usuário, e 85,71% acreditaram que o usuário local ficaria satisfeito com o resultado gerado pela política visto que ele confia que as suas próprias operações são as mais corretas e elas não são jamais removidas do repositório.

Tabela 5.2: Percepção dos usuários quanto à *política baseada na visão local*.

	Nível de concordância		
Variável de medição	1-3	4	5-7
Percepção de aceitabilidade	71,43%	28,57%	0%
Percepção de consistência	71,43%	14,28%	14,28%
Percepção de corretude	85,71%	0%	14,29%
Percepção de risco	28,57%	0%	71,43%
Economia de tempo	14,28%	14,28%	71,43%
Satisfação com o resultado	14,29%	0%	85,71%

Na Tabela 5.3 são mostradas as porcentagens de concordância dos usuários com relação a cada variável de medição de percepção para a *política que remove todos os conflitos*. Quanto à percepção de aceitabilidade, 57,14% dos usuários acharam a política aceitável para resolver inconsistências entre as operações dos usuários pois, além de evitar que dados incorretos permaneçam armazenados no repositório, a política ainda permite que aconteça uma discussão entre os usuários, possibilitando que os dados corretos sejam armazenados posteriormente. No entanto, 42,86% acreditaram que a política é inadequada por gerar muita perda de operações. Quanto à percepção de consistência, a maioria dos usuários acreditaram que a política fornece dados consistentes para o usuário local, sendo que 42,86% dos usuários caracterizaram essa consistência como média e outros 42,86% dos usuários caracterizaram essa consistência como boa. Quanto à percepção de corretude, ocorreu um empate entre as opiniões dos usuários de acordo com os níveis de concordância. Quanto à percepção de risco, a grande maioria dos usuários acreditou que a política não é capaz de resolver inconsistências sem muita intervenção humana. Com relação à economia de tempo, uma minoria dos usuários concordaram que a política é importante para poupar tempo do usuário, desde que ela descarta um alto número de operações e os usuários devem tomar a decisão sobre a inconsistência novamente. Quanto à satisfação com o resultado, 42,86% dos usuários acharam que o usuário local ficará insatisfeito com o resultado da política pois operações já consideradas corretas por ele em processos de reconciliação

anteriores podem ser removidas; e, 42,86% dos usuários não concordaram que o usuário local ficará insatisfeito por essa razão. Já 71,43% dos usuários acreditaram que o usuário local ficaria satisfeito com o resultado gerado pela política, visto que se não é possível determinar qual a operação correta, é melhor descartar todas as operações conflitantes e deixar que algum usuário tome a decisão novamente. Além disso, os usuários acreditaram que a política é adequada para bancos de dados que não podem conter dados incorretos que, portanto, deve ser usada para gerar todas as visões iguais e integradas.

Tabela 5.3: Percepção dos usuários quanto à *política que remove todos os conflitos*.

	Nível de concordância		
Variável de medição	1-3	4	5-7
Percepção de aceitabilidade	42,86%	0%	57,14%
Percepção de consistência	14,28%	42,86%	42,86%
Percepção de corretude	33,33%	33,33%	33,34%
Percepção de risco	14,29%	0%	85,71%
Economia de tempo	42,86%	28,57%	28,57%
Satisfação com o resultado	28,57%	0%	71,43%

Na Tabela 5.4 são mostradas as porcentagens de concordância dos usuários com cada variável de medição de percepção para a *política baseada em timestamp*. Quanto à percepção de aceitabilidade, 42,86% dos usuários acreditaram que a política é aceitável para resolver inconsistências entre as operações dos usuários pois mantém os dados sempre atualizados; 42,86% acreditaram que a política é inadequada pois a atualização mais recente pode alterar o atributo para um valor incorreto; e, 14,28% dos usuários acreditaram que a política é ágil, aparentemente eficaz, mas pode levar os dados aos valores incorretos. Quanto à percepção de consistência, 42,86% dos usuários acreditaram que a política fornece dados consistentes para o usuário local, mas 42,86% acreditaram que não. O mesmo padrão de nível de concordância da percepção de aceitabilidade também foi observado para a percepção de consistência. Quanto à percepção de risco, a maioria dos usuários acreditou que a política é capaz de resolver inconsistências sem muita intervenção humana. Além disso, 57,14% do usuários acreditaram que a resolução de inconsistências feita automaticamente é importante para poupar tempo do usuário. Quanto à satisfação com o resultado, 85,71% dos usuários acreditaram que o usuário local ficará insatisfeito com o resultado gerado pela política caso esteja resolvendo inconsistências nas mesmas fontes de dados concomitantemente com outros usuários. Apenas 28,57% dos usuários concordaram que o usuário local ficará satisfeito com o resultado da política pois desde que ela seja aplicada ao ambiente colaborativo, a última alteração feita é a mais provável de ser correta e, 42,86% não concordaram com esta afirmação. Os usuários acreditaram que essa política seja indi-

cada para ambientes que envolvem notícias pois, nesse caso, dados anteriores estarão ultrapassados e a última notícia é a mais importante para o público em geral.

Tabela 5.4: Percepção dos usuários quanto à *política baseada em timestamp*.

	Nível de concordância		
Variável de medição	1-3	4	5-7
Percepção de aceitabilidade	42,86%	14,28%	42,86%
Percepção de consistência	42,86%	14,28%	42,86%
Percepção de corretude	42,86%	42,86%	14,28%
Percepção de risco	71,43%	14,29%	14,28%
Economia de tempo	28,57%	14,29%	57,14%
Satisfação com o resultado	42,86%	28,57%	28,57%

Na Tabela 5.5 são mostradas as porcentagens de concordância dos usuários com cada variável de medição de percepção para a *política baseada em votação*. Quanto às percepções de aceitabilidade e consistência, todos os usuários acreditaram que a política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários e para fornecer dados consistentes para o usuário local. Os usuários consideraram a política mais democrática, capaz de satisfazer a um número maior de usuários. Quanto à percepção de corretude, a maioria dos usuários acredita que a política é capaz de fornecer dados corretos. Quanto à percepção de risco, 57,14% dos usuários acreditaram que a política não pode resolver inconsistências sem intervenção humana e, 42,86% acreditam que sim. Quanto à economia de tempo, os resultados obtidos foram divergentes. Enquanto, 42,86% dos usuários acreditaram que a resolução de inconsistências feita automaticamente, sem a participação dos usuários, é importante para poupar tempo do usuário, 42,86% acreditaram no contrário. Por fim, 85,71% dos usuários acreditaram que o usuário ficará satisfeito com o resultado da integração provido pela aplicação da política. Os usuários acreditam também que o usuário ficará satisfeito mesmo quando ele não souber o valor para um atributo, pois poderá tirar proveito de operações feitas por outros cuja maioria acredita que tenha o valor correto.

Na Tabela 5.6 são mostradas as porcentagens de concordância dos usuários com cada variável de medição de percepção para a *política baseada na confiança nas fontes*. Quanto à percepção de aceitabilidade, a maioria dos usuários acreditou que a política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários. Adicionalmente, todos os usuários acreditaram que a política fornece dados consistentes para o usuário local e que, além de consistentes, esses dados são os dados corretos. Quanto à percepção de risco, 57,14% dos usuários acreditaram que a política pode resolver inconsistências sem muita intervenção humana. Quanto à economia de tempo, a maioria dos usuários acreditou a resolução de inconsistências feita automaticamente é importante para poupar

Tabela 5.5: Percepção dos usuários quanto à *política baseada em votação*.

	Nível de concordância		
Variável de medição	1-3	4	5-7
Percepção de aceitabilidade	0%	0%	100%
Percepção de consistência	0%	0%	100%
Percepção de corretude	28,57%	0%	71,43%
Percepção de risco	42,86%	0%	57,14%
Economia de tempo	42,86%	14,28%	42,86%
Satisfação com o resultado	0%	14,29%	85,71%

tempo do usuário. Por fim, 85,71% dos usuários acreditaram que o usuário ficará satisfeito com o resultado gerado pela política e, 14,29% acreditaram nisso razoavelmente pois o usuário local pode estar confiando em um fonte que possua dados incorretos.

Tabela 5.6: Percepção dos usuários quanto à *política baseada na confiança nas fontes*.

	Nível de concordância		
Variável de medição	1-3	4	5-7
Percepção de aceitabilidade	14,28%	14,29%	71,43%
Percepção de consistência	0%	0%	100%
Percepção de corretude	0%	0%	100%
Percepção de risco	57,14%	28,57%	14,29%
Economia de tempo	14,29%	0%	85,71%
Satisfação com o resultado	0%	14,29%	85,71%

Na Tabela 5.7 são mostradas as porcentagens de concordância dos usuários com cada variável de medição de percepção para a *política baseada na confiança nos usuários*. Quanto à percepção de aceitabilidade, a maioria dos usuários acreditou que a política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários. Quanto à percepção de consistência, a maioria dos usuários acreditou que a política fornece dados consistentes para o usuário local. Quanto à percepção de corretude, a maioria dos usuários acreditou que a política seja capaz de fornecer os dados corretos e, quanto à percepção de risco, a maioria dos usuários acreditou que a política pode resolver inconsistências sem muita intervenção humana. Com relação à economia de tempo, a maioria dos usuários acreditou que a resolução de inconsistências feita automaticamente é importante para poupar tempo do usuário. Por fim, quanto à satisfação do usuário que requer o processo de reconciliação com o resultado, a maioria dos usuários acreditou que o usuário ficará satisfeito com o resultado gerado pela política desde que o usuário pode priorizar uma operação cuja confiança de outros usuários nos quais ele confia seja alta, em detrimento de uma que ele mesmo já realizou ou já confiou anteriormente.

Tabela 5.7: Percepção dos usuários quanto à *política baseada na confiança nos usuários*.

	Nível de concordância		
Variável de medição	1-3	4	5-7
Percepção de aceitabilidade	0%	28,57%	71,43%
Percepção de consistência	0%	14,29%	85,71%
Percepção de corretude	0%	14,29%	85,71%
Percepção de risco	71,43%	28,57%	0%
Economia de tempo	0%	14,29%	85,71%
Satisfação com o resultado	14,29%	0%	85,71%

De acordo com as percepções dos usuários, a política que teve maior aceitabilidade foi a *política baseada em votação*, e a política considerada mais inadequada para resolver inconsistências sobre os valores dos atributos foi a *política baseada na visão local*. As políticas que obtiveram maiores porcentagem sobre a percepção de consistência foi, novamente, a *política baseada em votação* e a *política baseada na confiança nas fontes*, com 100%, em adição à *política baseada na confiança nos usuários*, com 85,71%. Os usuários acreditaram que as políticas que fornecem como resultado os dados mais corretos são a *política baseada na confiança nas fontes*, a *política baseada na confiança nos usuários* e a *política baseada em votação*, com 100%, 85,71% e 71,43%, respectivamente. Quanto à percepção de risco, é interessante que o usuário tenha uma baixa percepção de risco. Então essa variável de medição é analisada de maneira oposta às demais, ou seja, maior porcentagem daqueles que não concordaram com a percepção de risco e menor porcentagem daqueles que concordaram com a percepção de risco é mais interessante. As políticas que tiveram menor percepção de risco foram a *política baseada em timestamp* e a *política baseada na confiança nos usuários*, com 71,43% dos usuários concordando que essas políticas são capazes de resolver inconsistências sem muita intervenção humana. A política que obteve menor percepção sobre a economia de tempo por parte do usuário que realiza o processo de reconciliação foi a *política que remove todos os conflitos*, pois quanto maior o número de conflitos, maior o número de operações são descartadas, e então, é necessário que o usuário faça novamente a decisão para aquele valor de atributo. Com exceção à *política baseada em timestamp*, as análises realizadas mostraram que o usuário que requereu o processo de reconciliação usando cada uma das demais políticas ficará satisfeito com os resultados obtidos.

5.4 Propagação de Decisões de Integração

Para validação do *método de propagação de decisões de integração*, 3 usuários, estudantes do curso de graduação em Ciência da Computação da Universidade Federal do Tocantins, interagiram com a ferramenta PrInt, a fim de corrigir inconsistências em 4 currículos de pesquisadores que trabalham no Departamento de Ciência da Computação da Universidade de São Paulo, Campus São Carlos. As operações de procedência geradas a partir dessas decisões foram coletadas.

Inicialmente, pediu-se para que os usuários realizassem uma operação de cópia, a partir de uma fonte considerada correta para duas outras fontes, considerando o mesmo atributo do mesmo objeto. Foram geradas portanto, duas operações de cópia no repositório: uma operação de cópia da primeira fonte para a segunda fonte e uma segunda operação de cópia da primeira fonte para a terceira fonte. Os tempos gastos pelos usuários são mostrados na Tabela 5.8. Os tempos foram considerados para cada usuário, a partir da realização da primeira operação de cópia, ou seja, desconsiderou-se o *tempo de pensar* inicial do usuário, bem como a configuração da ferramenta, tal como carregamento das fontes de dados. O *método de propagação de decisões de integração*, no entanto, gastou $0,078$ segundos para propagar a primeira decisão do usuário para a terceira fonte de dados. Ou seja, esse foi o tempo gasto pelo método para criar uma operação de cópia, a partir da primeira operação de cópia feita pelo usuário, armazenada no repositório. Portanto, o *método de propagação de decisões de integração* gastou substancialmente menos tempo para extrair os valores dos atributos da primeira operação e criar a segunda operação, quando comparado ao tempo gasto pelos usuários para criar duas operações de cópia.

Tabela 5.8: Tempo gasto pelos usuários para a tomada de decisão sobre um atributo em três diferentes fontes.

Usuário	Tempo (segundos)
Usuário 1	143s (02m:23s)
Usuário 2	31s (00m:31s)
Usuário 3	83s (01m:23s)

Posteriormente, pediu-se aos usuários que realizassem operações sobre dois objetos presentes nas quatro fontes de dados. O primeiro objeto estava presentes em três das quatro fontes, e o segundo objeto estava presente em duas das quatro fontes. O número de operações presentes no repositório de cada usuário, bem como o tempo gasto para realizar as operações, são mostrados na Tabela 5.9. Posteriormente, o *método de propagação de decisões de integração* foi aplicado sobre o repositório do *Usuário 1*, que continha 28 operações, e criou 10 operações em $1,622$ segundos. Então, algumas

operações do repositório do *Usuário 1* foram apagadas, de forma que apenas uma operação sobre cada atributo diferente de cada objeto ficasse no repositório. Restaram então, 19 operações no repositório do *Usuário 1*. O *método de propagação de decisões de integração* criou 25 operações a partir das 19 operações iniciais, gastando $1,451$ segundos para isso. Portanto, o tempo gasto pelo método para propagar as decisões do usuário para todas as fontes que contêm o objeto atualizado é consideravelmente menor que o tempo gasto pelo usuário para resolver a mesma inconsistência em várias fontes de dados.

Tabela 5.9: Tempo gasto pelos usuários para a tomada de decisão sobre atributos em quatro diferentes fontes.

Usuário	Número de operações	Tempo (segundos)
Usuário 1	28	650s (10m:50s)
Usuário 2	14	155s (02m:35s)
Usuário 3	3	140s (02m:20s)

Após a finalização dos testes, os usuários responderam um questionário contendo 7 questões, mostrado no Apêndice A.2. Inicialmente, os usuários foram questionados sobre o fato de tomar a mesma decisão para o mesmo atributo em 3 diferentes fontes, e sobre a possibilidade de realizar isso em um número maior de fontes. Os usuários classificaram como "fácil" e "prático" realizar essa tarefa, desde que eles utilizaram a ferramenta PrInt, mas acharam que a tarefa poderia ser cansativa para um número maior de fontes e sugeriram que, nesse caso, o usuário poderia escolher um número maior de fontes para serem atualizadas ao mesmo tempo. Questionados sobre o fato de poderem tomar uma decisão para um atributo uma única vez, e essa decisão ser propagada automaticamente para outras fontes que contêm o atributo, os usuários acharam que isso seria muito eficiente e valioso para o processo de integração. No entanto, acreditaram que processos automáticos invalidam a possibilidade de mudança de decisão do usuário, ou seja, o usuário não poderia mudar de ideia em relação ao valor de um atributo. Os usuários não foram informados que poderiam mudar de ideia quanto aos valores dos atributos, e que seriam questionados pelo *método de propagação de decisões de integração* sobre qual o valor correto para o atributo em questão.

Os usuários afirmaram ainda que, ao realizar o processo de correção de inconsistências em dias diferentes com algum intervalo entre eles, poderiam se esquecer do valor atualizado para um atributo em uma fonte, podendo atualizar posteriormente o mesmo atributo em fontes diferentes, para um valor diferente ou em outro formato. Afirmaram também que, após atualizar o mesmo atributo em um grande número de fontes, poderiam se cansar e simplificar os valores utilizados nas atualizações, como utilizando siglas. Por exemplo, após atualizar um *local de publicação* para *"Simpósio Brasi-*

leiro de Banco de Dados" várias vezes, poderiam atualizar o valor do atributo nas fontes seguintes para *"SBBD"*. Em contrapartida, essas situações não ocorreriam no sistema AcCORD. Por garantir uma propagação automática, os mesmos valores seriam sempre propagados e o processo não precisaria ser simplificado. Isso mostra uma maior corretude nos resultados gerados utilizando o *método de propagação de decisões de integração* e uma maior consistência para o estado do repositório.

5.5 Considerações Finais

Neste capítulo foram descritos experimentos realizados com o objetivo de avaliar o modelo AcCORD, as políticas de reconciliação propostas e o *método de propagação de decisões de integração*. Os experimentos foram realizados de forma a enfocar as eficácia e desempenho das políticas de reconciliação multiusuário, e enfocar a percepção dos usuários em relação às características das políticas de reconciliação e *método de propagação de decisões de integração*.

Com relação aos experimentos que visam as características das políticas em relação ao gerenciamento de decisões conflitantes dos usuário, foram consideradas a reconciliação de um menor volume de dados e a reconciliação de um maior volume de dados. Os resultados mostraram que a *política que remove todos os conflitos* removeu o maior número de operações, como esperado, além de não refazer nenhuma operação, que é característica da política. Considerando um cenário com 3 usuários, a *política baseada em votação* obteve operações vencedoras na votação, e mostrou melhores resultados em termos de porcentagem de remoções do que a *política que remove todos os conflitos* e as demais políticas. A *política baseada em votação* removeu de $21,3\%$ a 53.6% menos operações que a *política que remove todos os conflitos*. De maneira geral, a *política baseada na confiança nos usuários* obteve bons resultados, juntamente com a *política baseada em votação*, no que diz respeito à menor porcentagem de remoções de operações do repositório e ao maior número de operações refeitas, e bons resultados, juntamente com a *política baseada na visão local*, no que diz respeito ao menor tempo de execução. A *política baseada na confiança nas fontes* tem o seu desempenho prejudicado em relação às demais políticas devido ao gerenciamento das fontes de dados.

Os resultados demostraram a eficiência do modelo AcCORD, corroborando a hipótese de que reconciliação de dados colaborativa pode prover ganhos para o usuário local, evitando que ele realize um trabalho já realizado por outros colaboradores. Esses ganhos existem mesmo quando o volume de dados é maior, e o número de usuários trabalhando sobre as mesmas fontes de dados, cresce, aumentando o número de conflitos.

Quanto ao experimento baseado na percepção dos usuários, acreditava-se que a *política baseada na visão local* teria a maior aceitabilidade entre os usuários devido ao fato de manter as operações feitas pelo usuário local, deixando-o sempre satisfeito. No entanto, essa foi a política com menor porcentagem de aceitabilidade entre os usuários, pois esses acreditaram na colaboração dos demais usuários. Acreditava-se também que a *política que remove todos os conflitos* não teria grande aceitabilidade entre os usuários, e os resultados mostraram o contrário. As políticas com maior aceitabilidade, no entanto, foram a *política baseada em votação*, a *política baseada na confiança nas fontes* e a *política baseada na confiança nos usuários* que os usuários consideraram bastante semelhante a *política baseada em votação*, porém com pesos diferentes para os usuários confiáveis para o usuário local. Acredita-se que alguns usuários possam ter confundido a percepção de consistência com a percepção de corretude pois, a maioria dos usuários entrevista não teve contato com a disciplina de banco de dados e não foi dada, durante a breve explicação anterior à aplicação do questionário, a definição de consistência utilizada nesta tese. Essa definição refere-se ao fato de não haver operações conflitantes no repositório, e não ao fato de que a operação que permanece no repositório seja a operação correta. Acredita-se que para a *política baseada em votação*, os usuários possam não ter compreendido que a votação é feita automaticamente e por isso uma porcentagem alta na percepção de risco, na qual acreditaram que a política não é capaz de resolver inconsistências sem muita intervenção humana. Isso se diferiu das demais políticas, para as quais a mesma afirmativa quanto à percepção de risco foi apresentada. Também observou-se para essa política, uma porcentagem alta de usuários que não acreditou que o processo sem a intervenção do usuário seja importante para poupar tempo do usuário, sendo que para as demais políticas, os usuários consideraram o contrário. Algumas afirmativas contraditórias entre si foram apresentadas, bem como algumas afirmativas foram repetidas para todas as políticas, a fim de detectar esforço insuficiente por parte do usuário para responder à pesquisa, testar a sua compreensão quanto às políticas e validar o próprio questionário.

No experimento realizado para a validação do *método de propagação de decisões de integração*, demonstrou-se que o método proposto gastou consideravelmente menos tempo do que os usuários para realizar as operações que refletem a mesma decisão em diferentes fontes. Quanto à opinião dos usuários, eles consideraram o *método de propagação de decisões de integração* uma forma eficiente de realizar a tarefa repetitiva e poupar o seu tempo. Por meio de uma análise dos questionários respondidos, pode-se perceber que os usuários confiam em um método automático desde que tenham a possibilidade de alterar os resultados gerados pelo método quando assim o desejarem. Ou seja, os usuários querem alguma automação que forneça algum benefício para eles, mas também querem ter controle sobre os resultados obtidos.

No Capítulo 6 são apresentadas as principais contribuições desta tese de doutorado, as conclusões e os trabalhos futuros.

Capítulo 6

Conclusões e Trabalhos Futuros

Esta pesquisa de doutorado investigou o problema de reconciliação de dados quando múltiplos usuários trabalham de maneira assíncrona sobre cópias locais do mesmo conjunto de dados importados. As principais contribuições são as seguintes. A primeira contribuição foi a proposta do modelo AcCORD, um modelo colaborativo assíncrono para a reconciliação de dados. A segunda contribuição foi a proposta de políticas de reconciliação para resolução de conflitos resultantes de atualizações de múltiplos usuários. A terceira contribuição foi a proposta de um método de propagação de decisões de integração para auxiliar o usuário no processo de integração e evitar inconsistências nas tomadas de decisões. As propostas se baseiam nos seguintes conceitos: (i) procedência dos dados, que representa decisões que os usuários tomaram para resolver conflitos em dados importados; (ii) repositório, que funciona como um *log* e armazena dados de procedência na forma de operações; e (iii) flexibilidade, por prover suporte a aplicações nas quais todos os usuários concordam com a política de resolução de conflitos utilizada no processo de reconciliação, e como resultado provê uma única visão consistente para todos eles, bem como aplicações que permitem os usuários discordarem sobre o valor correto do dado, mas promovem o compartilhamento de dados.

O modelo AcCORD pode ser aplicado a aplicações caracterizadas por um alto grau de independência, nas quais os usuários trabalham de maneira autônoma e não estão conectados a todos os outros em todos os momentos. Exemplos dessas aplicações incluem sistemas de auxílio à saúde, sistemas de curação em dados de bioinformática e compartilhamento de dados bibliográficos. Assim, é oferecido ao usuário a possibilidade de escolher, dentre diversas políticas disponíveis, aquela que melhor atende aos seus objetivos. Outra vantagem do modelo AcCORD refere-se à sua extensibilidade, desde que ele não foi projeto especificamente para usar um determinado modelo de procedência baseado em operação e portanto pode ser aplicado usando diferentes modelos de procedência. As-

sim, os usuários podem escolher a ferramenta mais apropriada para os seus requisitos de integração de dados, e aplicar o modelo AcCORD para a colaboração entre eles.

6.1 Principais Resultados e Contribuições

Os principais resultados deste trabalho são relacionados a seguir:

1. **Modelo AcCORD** - Proposta do modelo AcCORD, um modelo colaborativo assíncrono para a reconciliação de dados, no qual as atualizações feitas pelos usuários são armazenadas em repositórios.

2. **Políticas para integração e compartilhamento de dados** - Propostas de políticas para resolver possíveis conflitos que resultam do processo de reconciliação multiusuário colaborativo e assíncrono. Essas políticas podem ser destinadas às aplicações que geram uma visão integrada ou distintas visões locais como resultado do trabalho colaborativo. As políticas são baseadas nas seguintes estratégias: remoção das operações realizadas por diferentes usuários cujas decisões conflitam entre si; priorização das decisões tomadas pelo próprio usuário; ordem cronológica da tomada da decisão; priorização da decisão tomada pela maioria dos usuários; confiança nas fontes de dados, e; confiança nos usuários que tomaram as decisões.

3. **Método de propagação de decisões de integração** - Proposta de um método de propagação em nível de um único usuário para evitar que o usuário tenha que tomar a mesma decisão para o mesmo atributo de um objeto em diferentes fontes, diminuindo o seu trabalho na integração dos dados e evitando inconsistências na tomada de decisões de integração. Esse método usa os dados de procedência para retificar as fontes de dados que estão incorretas, com base no processo de integração.

A partir dos resultados obtidos durante o desenvolvimento deste projeto, a seguinte publicação foi gerada até a presente data:

- Almeida, D. S., Hara, C. S., and Ciferri, C. D. A. (2015). What if Multiusers Wish to Reconcile Their Data? In Proceedings of the 17th International Conference on Enterprise Information Systems, volume 1, pages 184-195.

6.2 Conclusões

A validação do modelo AcCORD e das políticas propostas foi feita por meio de experimentos que investigaram o gerenciamento de operações conflitantes, o desempenho das políticas e a percepção dos usuários em relação às características de cada política e também do *método de propagação de decisões de integração*. Os testes que investigaram o gerenciamento de operações conflitantes tiveram como objetivo verificar as consequências do aumento no número de usuários e do número de operações, em termos do número de operações não tratadas e removidas (que causa a perda de decisões dos usuários), e da remoção de operações realizadas localmente (que reflete a perda de decisões do usuário que está realizando o processo de reconciliação). O desempenho de cada política foi analisado em termos de tempo de execução.

Os resultados obtidos mostraram a eficácia de cada política, mesmo diante de um maior volume de operações. Portanto, a escolha da política depende das características do ambiente e da aplicação na qual ela será utilizada a política escolhida, podendo ser mais ou menos restritiva. Os resultados mostraram também a flexibilidade do modelo AcCORD que pode ser usado tanto para compartilhamento de dados, quanto para integração de dados em nível multiusuário. Além disso, mostram a eficiência do modelo, o que corrobora para a hipótese de que reconciliação de dados colaborativa pode prover ganhos para o usuário local, evitando que ele realize um trabalho já realizado por outros colaboradores.

Os experimentos baseados no usuário tiveram como objetivo verificar a percepção do usuário para cada política, em relação à consistência, aceitabilidade, corretude, economia de tempo e satisfação. Além disso, foi verificada também a percepção do usuário em relação à necessidade de um método de propagação de decisões automático e a eficiência desse método. Os resultados do experimento baseado no usuário mostraram uma maior aceitação para as políticas de reconciliação que permitem maior colaboração entre os usuários e que adotam como resultado correto aquele baseado na opinião de todos os colaboradores. Os resultados mostraram também a necessidade do *método de propagação de decisões de integração* para evitar inconsistências entre as decisões tomadas, e a eficiência do método para diminuir o tempo despendido no processo de integração.

6.3 Trabalhos Futuros

O objetivo do modelo AcCORD, na sua presente versão, é realizar o processo de reconciliação de dados da forma mais automática possível, de acordo com a política escolhida pelo usuário. Entretanto,

ele pode ser estendido para incorporar uma interação com o usuário. Existem dois níveis de iteração que podem ser incorporados. No primeiro nível, quando o processo de reconciliação estiver sendo realizado, ao invés de apenas descartar as operações conflitantes e as dependentes dessas, pode-se mostrar as operações conflitantes ao usuário e aos usuários que as criaram, para que eles decidam qual é operação a correta de acordo com a qual devem ser atualizadas as suas dependentes para o seu valor, e qual operação deve mesmo ser descartada, como é feito no *método de propagação de decisões de integração*. No segundo nível, pode-se armazenar em um repositório auxiliar todas as operações que foram descartadas, de forma que esse repositório possa ser posteriormente pesquisado pelo usuário. O usuário pode assim, analisar as operações descartadas e tomar alguma decisão a respeito delas, podendo retorná-las ao repositório caso conclua que tenha tomado uma decisão errada previamente.

Outro trabalho futuro é a aplicação do modelo AcCORD em outros cenários além do compartilhamento de dados bibliográficos, realizado nesta tese. Esses cenários incluem a integração de dados de saúde para auxiliar em diagnósticos mais precisos e o compartilhamento de dados agronômicos. Outro possível estudo consiste na aplicação do modelo AcCORD no ambiente *web*, possibilitando maior adesão dos usuários, o que tem como consequência um possível aumento na manipulação dos dados gerando mais operações e mais conflitos entre elas. Considera-se também a aplicação do modelo AcCORD para ambientes P2PDBN (*peer to peer database network*) [Masud and Kiringa 2011].

Outro possível trabalho futuro consiste em adaptar o modelo AcCORD para ser executado em plataformas de processamento intensivo de dados em *clusters* de computadores, tais como baseados em Hadoop *MapReduce*[1] e Spark [Zaharia et al. 2012], de forma a atender requisitos de processamento de *big data* [Han et al. 2011]. O modelo AcCORD também pode ser estendido para ser usado em um ambiente de computação em nuvem [Abadi 2009] de forma a prover um ambiente com características de elasticidade e eficiência sob demanda.

[1]The Apache Software Foundation, `http://hadoop.apache.org`

Referências Bibliográficas

[Abadi 2009] Abadi, D. J. (2009). Data Management in the Cloud: Limitations and Opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12.

[Abawajy et al. 2013] Abawajy, J. H., Jami, S. I., Shaikh, Z. A., and Hammad, S. A. (2013). A framework for scalable distributed provenance storage system. *Computer Standards & Interfaces*, 35(1):179 – 186.

[Aguiar 1995] Aguiar, C. D. (1995). Integração de Sistemas de Banco de Dados Heterogêneos em Aplicações de Planejamento Urbano. Master's thesis, Universidade Estadual de Campinas (Unicamp).

[Aleman-Meza et al. 2006] Aleman-Meza, B., Nagarajan, M., Ramakrishnan, C., Ding, L., Kolari, P., Sheth, A., Arpinar, B., Joshi, A., and Finin, T. (2006). Semantic Analytics on Social Networks: Experiences in Addressing the Problem of Conflict of Interest Detection. In *International World Wide Web Conference (WWW)*, pages 407–416.

[Almeida et al. 2015] Almeida, D. S., Hara, C. S., and Ciferri, C. D. A. (2015). What if Multiusers Wish to Reconcile Their Data? In *Proceedings of the 17th International Conference on Enterprise Information Systems*, volume 1, pages 184–195.

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

[Archer et al. 2009] Archer, D. W., Delcambre, L. M. L., and Maier, D. (2009). A Framework for Fine-grained Data Integration and Curation, with Provenance, in a Dataspace. In *Workshop on the Theory and Practice of Provenance*.

[Beeri and Vardi 1984] Beeri, C. and Vardi, M. Y. (1984). A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741.

[Beneventano et al. 2011] Beneventano, D., Dannoui, A., and Sala, A. (2011). Data lineage in the MOMIS data fusion system. In *2011 IEEE 27th International Conference on Data Engineering Workshops (ICDEW)*, pages 53–58, Los Alamitos, CA, USA.

[Benjelloun et al. 2008] Benjelloun, O., Das Sarma, A., Halevy, A., Theobald, M., and Widom, J. (2008). Databases with uncertainty and lineage. *The VLDB Journal*, 17(2):243–264.

[Benjelloun et al. 2009] Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S., and Widom, J. (2009). Swoosh: a Generic Approach to Entity Resolution. *The VLDB Journal*, 18(1):255–276.

[Bhattacharjee and Jamil 2012] Bhattacharjee, A. and Jamil, H. (2012). A schema matching system for on-the-fly autonomous data integration. *International Journal of Information and Decision Sciences*, 4(2/3):167–181.

[Bleiholder and Naumann 2006] Bleiholder, J. and Naumann, F. (2006). Conflict Handling Strategies in an Integrated Information System.

[Bleiholder and Naumann 2009] Bleiholder, J. and Naumann, F. (2009). Data Fusion. *ACM Comput. Surv.*, 41(1):1:1–1:41.

[Borges et al. 2008] Borges, E. N., Galante, R. M., and Gonçalves, M. A. (2008). Uma Abordagem Efetiva e Eficiente para Deduplicação de Metadados Bibliográficos de Objetos Digitais. In *Simpósio Brasileiro de Banco de Dados (SBBD)*, pages 76–90.

[Buneman et al. 2006a] Buneman, P., Chapman, A., and Cheney, J. (2006a). Provenance management in curated databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 539–550, New York, NY, USA. ACM.

[Buneman et al. 2006b] Buneman, P., Chapman, A., Cheney, J., and Vansummeren, S. (2006b). A Provenance Model for Manually Curated Data. In Moreau, L. and Foster, I. T., editors, *IPAW*, volume 4145 of *Lecture Notes in Computer Science*, pages 162–170. Springer.

[Buneman et al. 2001] Buneman, P., Khanna, S., and Tan, W. C. (2001). Why and Where: A Characterization of Data Provenance. In *ICDT '01: Proceedings of the 8th International Conference on Database Theory*, pages 316–330, London, UK. Springer-Verlag.

[Cao et al. 2013] Cao, Y., Fan, W., and Yu, W. (2013). Determining the relative accuracy of attributes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 565–576.

[Chapman et al. 2008] Chapman, A. P., Jagadish, H. V., and Ramanan, P. (2008). Efficient provenance storage. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 993–1006, New York, NY, USA. ACM.

[Chen et al. 2007] Chen, Z., Kalashnikov, D. V., and Mehrotra, S. (2007). Adaptive graphical approach to entity resolution. In *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 204–213, New York, NY, USA. ACM.

[Cheney et al. 2009] Cheney, J., Chiticariu, L., and Tan, W.-C. (2009). Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases*, 1(4):379–474.

[Del Rio and Silva 2007] Del Rio, N. and Silva, P. P. (2007). Probe-It! Visualization Support for Provenance. In *Advances in Visual Computing*, volume Volume 4842/2007 of *Lecture Notes in Computer Science*, pages 732–741. Springer Berlin / Heidelberg.

[Demsky 2011] Demsky, B. (2011). Cross-application data provenance and policy enforcement. *ACM Transactions on Information and System Security*, 14(1):6:1–6:22.

[Dong et al. 2005] Dong, X., Halevy, A., and Madhavan, J. (2005). Reference reconciliation in complex information spaces. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96, New York, NY, USA. ACM.

[Dorneles et al. 2004] Dorneles, C. F., Heuser, C. A., Lima, A. E. N., da Silva, A. S., and de Moura, E. S. (2004). Measuring similarity between collection of values. In *Web Information and Data Management (WIDM)*, pages 56–63.

[Dorneles et al. 2007] Dorneles, C. F., Heuser, C. A., Orengo, V. M., da Silva, A. S., and de Moura, E. S. (2007). A strategy for allowing meaningful and comparable scores in approximate matching. In *International Conference on Information and Knowledge Management (CIKM)*, pages 303–312.

[Fagin et al. 2009] Fagin, R., Haas, L. M., Hernández, M., Miller, R. J., Popa, L., and Velegrakis, Y. (2009). Conceptual Modeling: Foundations and Applications. chapter Clio: Schema Mapping Creation and Data Exchange, pages 198–236. Springer-Verlag, Berlin, Heidelberg.

[Fagin et al. 2005] Fagin, R., Kolaitis, P. G., Miller, R. J., and Popa, L. (2005). Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89 – 124. ¡ce:title¿Database Theory¡/ce:title¿.

[Ferreira et al. 2012] Ferreira, A. A., Gonçalves, M. A., and Laender, A. H. F. (2012). A Brief Survey of Automatic Methods for Name Disambiguation. *Sigmod Record*, 41(2):15–26.

[Fileto et al. 2003] Fileto, R., Medeiros, C. B., Liu, L., Pu, C., and Assad, E. D. (2003). Using Domain Ontologies to Help Track Data Provenance. In *Simpósio Brasileiro de Banco de Dados*, pages 84–98.

[Freire et al. 2008] Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *IEEE Computing in Science & Engineering*, 10(3):11–21.

[Gal et al. 2005] Gal, A., Anaby-Tavor, A., Trombetta, A., and Montesi, D. (2005). A Framework for Modeling and Evaluating Automatic Semantic Reconciliation. *The VLDB Journal*, 14:50–67.

[Gatterbauer et al. 2009a] Gatterbauer, W., Balazinska, M., Khoussainova, N., and Suciu, D. (2009a). Believe It or Not: Adding Belief Annotations to Databases. *PVLDB*, 2(1):1–12.

[Gatterbauer et al. 2009b] Gatterbauer, W., Balazinska, M., Khoussainova, N., and Suciu, D. (2009b). Believe It or Not: Adding Belief Annotations to Databases. *CoRR*, abs/0912.5241.

[Getoor and Machanavajjhala 2012] Getoor, L. and Machanavajjhala, A. (2012). Entity Resolution: Theory, Practice & Open Challenges. *Proceedings of VLDB Endowment*, 5(12):2018–2019.

[Glavic and Ditt 2007] Glavic, B. and Ditt, K. R. (2007). Data Provenance: A Categorization of Existing Approaches. In *The German Database Conference*, pages 227–241, Aachen, Germany.

[Green et al. 2007] Green, T. J., Karvounarakis, G., Ives, Z. G., and Tannen, V. (2007). Update exchange with mappings and provenance. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 675–686. VLDB Endowment.

[Gupta et al. 2011a] Gupta, N., Kot, L., Bender, G., Roy, S., Gehrke, J., and Koch, C. (2011a). Coordination through querying in the Youtopia system. In *Proc. SIGMOD*, pages 1331–1334. Demo paper.

[Gupta et al. 2011b] Gupta, N., Kot, L., Roy, S., Bender, G., Gehrke, J., and Koch, C. (2011b). Entangled queries: enabling declarative data-driven coordination. In *Proc. SIGMOD*, pages 673–684.

[Halevy et al. 2005] Halevy, A. Y., Ashish, N., Bitton, D., Carey, M., Draper, D., Pollock, J., Rosenthal, A., and Sikka, V. (2005). Enterprise Information Integration: Successes, Challenges and Controversies. In *International Conference on Management of Data (SIGMOD)*, pages 778–787.

[Halevy et al. 2006] Halevy, A. Y., Rajaraman, A., and Ordille, J. (2006). Data Integration: The Teenage Years. In *International Conference on Very Large Data Bases (VLDB)*, pages 9–16.

[Han et al. 2011] Han, J., E, H., Le, G., and Du, J. (2011). Survey on NoSQL database. In *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, pages 363–366.

[Heinis and Alonso 2008] Heinis, T. and Alonso, G. (2008). Efficient lineage tracking for scientific workflows. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1007–1018, New York, NY, USA. ACM.

[Hossain et al. 2014] Hossain, M. S., Masud, M., Muhammad, G., Rawashdeh, M., and Hassan, M. M. (2014). Automated and User involved Data Synchronization in Collaborative E-Health Environments. *Computer in Human Behavior*, 30:485–490.

[Hsiao 1992] Hsiao, D. K. (1992). Federated Databases Systems: Part I - A Tutorial on their Data Sharing. *The VLDB Journal*, 1(1):127–179.

[Hsiao and Kamel 1989] Hsiao, D. K. and Kamel, M. N. (1989). Heterogeneous Databases: Proliferations, Issues, and Solutions. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):45–62.

[Huang and Cheng 2011] Huang, L. and Cheng, H. B. (2011). Query Optimization Based on Data Provenance. *Advanced Materials Research*, 186:586–590.

[Ikeda et al. 2012] Ikeda, R., Cho, J., Fang, C., Salihoglu, S., Torikai, S., and Widom, J. (2012). Provenance-Based Debugging and Drill-Down in Data-Oriented Workflows. *IEEE 28th International Conference on Data Engineering (ICDE)*, 0:1249–1252.

[Ives et al. 2005] Ives, Z., Khandelwal, N., Kapur, A., and Cakir, M. (2005). Orchestra: Rapid, collaborative sharing of dynamic data. In *CIDR*.

[Ives et al. 2008] Ives, Z. G., Green, T. J., Karvounarakis, G., Taylor, N. E., Tannen, V., Talukdar, P. P., Jacob, M., and Pereira, F. (2008). The ORCHESTRA Collaborative Data Sharing System. *SIGMOD Rec.*, 37(3):26–32.

[Kalashnikov and Mehrotra 2006] Kalashnikov, D. V. and Mehrotra, S. (2006). Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Trans. Database Syst.*, 31(2):716–767.

[Karger and Jones 2006] Karger, D. R. and Jones, W. (2006). Data Unification in Personal Information Management. *Communications of the ACM*, 49(1):77–92.

[Kermarrec et al. 2001] Kermarrec, A.-M., Rowstron, A., Shapiro, M., and Druschel, P. (2001). The IceCube Approach to the Reconciliation of Divergent Replicas. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 210–218.

[Kleb and Abecker 2010] Kleb, J. and Abecker, A. (2010). Entity Reference Resolution via Spreading Activation on RDF-Graphs. In *Proceedings of the Semantic Web: Research and Applications, 7th Extended Semantic Web Conference*, pages 152–166.

[Köpcke et al. 2010] Köpcke, H., Thor, A., and Rahm, E. (2010). Evaluation of entity resolution approaches on real-world match problems. *PVLDB*, 3(1):484–493.

[Kot and Koch 2009] Kot, L. and Koch, C. (2009). Cooperative Update Exchange in the Youtopia System. *PVLDB*, 2(1):193–204.

[Lebo et al. 2012] Lebo, T., Wang, P., Graves, A., and McGuinness, D. (2012). Towards Unified Provenance Granularities. In Groth, P. and Frew, J., editors, *Provenance and Annotation of Data and Processes*, volume 7525 of *Lecture Notes in Computer Science*, pages 39–51. Springer Berlin Heidelberg.

[Lenzerini 2002] Lenzerini, M. (2002). Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 233–246, New York, NY, USA. ACM.

[Mahmood et al. 2013] Mahmood, T., Jami, S. I., Shaikh, Z. A., and Mughal, M. H. (2013). Toward the modeling of data provenance in scientific publications. *Computer Standards & Interfaces*, 35(1):6–29.

[Marnette et al. 2011] Marnette, B., Mecca, G., Papotti, P., Raunich, S., and Santoro, D. (2011). ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. *PVLDB*, 4(12):1438–1441.

[Masud and Kiringa 2011] Masud, M. and Kiringa, I. (2011). Transaction processing in a peer to peer database network. *Data & Knowledge Engineering*, 70(4):307–334.

[Moreau et al. 2011] Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., and den Bussche, J. V. (2011). The Open Provenance Model core specification (v1.1). *Future Gener. Comput. Syst.*, 27(6):743–756.

[Muniswamy-Reddy et al. 2006] Muniswamy-Reddy, K.-K., Holland, D. A., Braun, U., and Seltzer, M. (2006). Provenance-aware storage systems. In *Proceedings of the Annual Technical Conference on USENIX'06 Annual Technical Conference*, Boston, MA. USENIX Association. 1267363 4-4.

[Munroe et al. 2006] Munroe, S., Miles, S., Moreau, L., and Vázquez-Salceda, J. (2006). PrIMe: a software engineering methodology for developing provenance-aware applications. In *SEM '06: Proceedings of the 6th international workshop on Software engineering and middleware*, pages 39–46, New York, NY, USA. ACM.

[Nascimento and Hara 2008] Nascimento, A. M. and Hara, C. S. (2008). A Model for XML Instance Level Integration. In *Simpósio Brasileiro de Banco de Dados*, pages 46–60.

[Nguyen et al. 2011] Nguyen, H.-Q., Taniar, D., Rahayu, J., and Nguyen, W. K. (2011). Double-layered schema integration of heterogeneous XML sources. *Journal of Systems and Software*, 1(1):63–76.

[On et al. 2006] On, B.-W., Elmacioglu, E., Lee, D., Kang, J., and Pei, J. (2006). Improving Grouped-Entity Resolution Using Quasi-Cliques. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 1008–1015, Washington, DC, USA. IEEE Computer Society.

[Ram and Liu 2012] Ram, S. and Liu, J. (2012). A Semantic Foundation for Provenance Management. *Journal on Data Semantics*, 1:11–17.

[Roth et al. 2010a] Roth, B., Volz, B., and Hecht, R. (2010a). Data integration systems for scientific applications. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems*, OTM'10, pages 110–118, Berlin, Heidelberg. Springer-Verlag.

[Roth et al. 2010b] Roth, B., Volz, B., and Hecht, R. (2010b). Data Integration Systems for Scientific Applications. In Meersman, R., Dillon, T., and Herrero, P., editors, *On the Move to Meaningful Internet Systems: OTM 2010 Workshops*, volume 6428 of *Lecture Notes in Computer Science*, pages 110–118. Springer Berlin Heidelberg.

[Sheth and Larson 1990] Sheth, A. P. and Larson, J. A. (1990). Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–235.

[Shu et al. 2011] Shu, L., Chen, A., Xiong, M., and Meng, W. (2011). Efficient SPectrAl Neighborhood blocking for entity resolution. In *Proceedings of the 27th International Conference on Data Engineering*, pages 1067–1078.

[Simmhan et al. 2005] Simmhan, Y. L., Plale, B., and Gannon, D. (2005). A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36.

[Tan 2004] Tan, W.-C. (2004). Research Problems in Data Provenance. *IEEE Data Engineering Bulletin*, 27(4):45–52.

[Taylor and Ives 2006] Taylor, N. E. and Ives, Z. G. (2006). Reconciling while tolerating disagreement in collaborative data sharing. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 13–24, New York, NY, USA. ACM.

[Taylor and Ives 2010] Taylor, N. E. and Ives, Z. G. (2010). Reliable storage and querying for collaborative data sharing systems. In *Proceedings of the 26th International Conference on Data Engineering*, pages 40–51.

[Tomazela 2010] Tomazela, B. (2010). MPPI: um modelo de procedência para subsidiar processos de integração. Master's thesis, Universidade de São Paulo, São Carlos, SP.

[Tomazela et al. 2008] Tomazela, B., Ciferri, C. D. A., and Traina Jr, C. (2008). Reconciliando dados de cunho acadêmico. In *SBBD '08: Proceedings of the 23rd Brazilian symposium on Databases*, pages 283–297, Porto Alegre, Brazil, Brazil. Sociedade Brasileira de Computação.

[Tomazela et al. 2010] Tomazela, B., Hara, C. S., Ciferri, R. R., and Ciferri, C. D. A. (2010). PrInt: a provenance model to support integration processes. In *Proceedings of the 19th International Conference on Information and Knowledge Management (CIKM)*, pages 1349–1352.

[Tomazela et al. 2013] Tomazela, B., Hara, C. S., Ciferri, R. R., and Ciferri, C. D. A. (2013). Empowering Integration Processes with Data Provenance. *Data & Knowledge Engineering*, 86:102–123.

[Unal and Afsarmanesh 2010] Unal, O. and Afsarmanesh, H. (2010). Semi-automated schema integration with SASMINT. *Knowledge and Information Systems*, 21(1):99–128.

[Whang and Garcia-Molina 2012] Whang, S. E. and Garcia-Molina, H. (2012). Joint Entity Resolution. In *Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE)*, pages 294–305.

[Widom 2005] Widom, J. (2005). 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)*.

[Yang et al. 2012] Yang, H., Michaelides, D. T., Charlton, C., Browne, W. J., and Moreau, L. (2012). DEEP: A Provenance-Aware Executable Document System. In Groth, P. and Frew, J., editors, *Provenance and Annotation of Data and Processes*, volume 7525 of *Lecture Notes in Computer Science*, pages 24–38. Springer Berlin Heidelberg.

[Zaharia et al. 2012] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA. USENIX Association.

[Zhao et al. 2006] Zhao, Y., Wilde, M., and Foster, I. T. (2006). Applying the virtual data provenance model. In *International Provenance and Annotation Workshop*, volume 4145, pages 148–161. Chicago, IL, USA.

[Zhu et al. 2010] Zhu, J., Fung, G., and Zhou, X. (2010). Efficient web pages identification for entity resolution. In *Proceedings of the International Conference on World Wide Web*, pages 1223–1224.

Parte I

Apêndice

Apêndice A

A.1 Questionário usado no experimento com as políticas de reconciliação

Nome:_____

Função/cargo:_____

Nível de escolaridade/período:_____

Curso:_____ Instituição:_____

Após analisar a descrição das seis políticas de reconciliação de dados descritas, bem como os resultados gerados por essas, responda:

Sobre a *política baseada na visão local*:

1. Qual a sua opinião sobre a *política baseada na visão local*?

2. Analise as afirmativas abaixe e marque o valor de 1 a 7 corresponde a sua opinião, sendo que 1 significa "discordo completamente" e 7 significa "concordo completamente".
   a) A política fornece dados consistentes para o usuário local.

1	2	3	4	5	6	7

   b) A política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.

1	2	3	4	5	6	7

   c) A resolução de inconsistências feita automaticamente (sem a participação dos usuários) é importante para poupar tempo do usuário.

1	2	3	4	5	6	7

   d) Eu penso que a política fornece as operações corretos ou confiáveis durante a resolução de inconsistências de atualizações.

1	2	3	4	5	6	7

   e) Eu acho que a política pode fornecer resultados eficazes.

1	2	3	4	5	6	7

   f) A política pode resolver inconsistências sem muita intervenção humana.

1	2	3	4	5	6	7

   g) O usuário local sempre ficará satisfeito com o resultado gerado pela política, pois suas operações jamais serão removidas.

1	2	3	4	5	6	7

   h) A priorização das operações feitas por outros usuários deixaria o usuário local (U1) insatisfeito, pois ele confia que as suas próprias operações de atualização são as mais corretas.

1	2	3	4	5	6	7

3. Você havia pensado anteriormente a esse momento que o usuário local (U1) pode não saber ou não ter certeza sobre o valor de algum atributo, e que a colaboração de outros usuários pode ser valiosa? Além disso, que a opinião de outro usuário pode estar mais correta que a dele?

4. E agora, o que você pensa sobre a política em questão?

5. Para quais aplicações/ambientes, você considera essa política adequada?

Sobre a *política que remove todas as inconsistências*:

1. Qual a sua opinião sobre a *política que remove todas as inconsistências*?

2. Analise as afirmativas abaixe e marque o valor de 1 a 7 corresponde a sua opinião, sendo que 1 significa "discordo completamente" e 7 significa "concordo completamente".
   a) A política fornece dados consistentes para o usuário local.

1	2	3	4	5	6	7

   b) A política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.

1	2	3	4	5	6	7

   c) A resolução de inconsistências feita automaticamente (sem a participação dos usuários) é importante para poupar tempo do usuário.

1	2	3	4	5	6	7

   d) Eu penso que a política fornece as operações corretos ou confiáveis durante a resolução de inconsistências de atualizações.

1	2	3	4	5	6	7

   e) Eu acho que a política pode fornecer resultados eficazes.

1	2	3	4	5	6	7

   f) A política pode resolver inconsistências sem muita intervenção humana.

1	2	3	4	5	6	7

   g) O usuário local ficará bastante insatisfeito com o resultado gerado pela política, pois suas operações e as operações já consideradas corretas em processos anteriores serão constantemente removidas.

1	2	3	4	5	6	7

   h) O usuário local ficará bastante satisfeito com o resultado gerado pela política, pois ele a aplica em um ambiente que requer consistência forte.

Portanto, se não é possível determinar qual a operação correta, melhor descartar todas e deixar que o usuário tome a decisão novamente.

1	2	3	4	5	6	7

3. Para quais aplicações/ambientes, você considera essa política adequada?

Sobre a *política baseada em timestamp*:

1. Qual a sua opinião sobre a *política baseada em timestamp*?

2. Analise as afirmativas abaixe e marque o valor de 1 a 7 corresponde a sua opinião, sendo que 1 significa "discordo completamente" e 7 significa "concordo completamente".
   a) A política fornece dados consistentes para o usuário local.

1	2	3	4	5	6	7

   b) A política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.

1	2	3	4	5	6	7

   c) A resolução de inconsistências feita automaticamente (sem a participação dos usuários) é importante para poupar tempo do usuário.

1	2	3	4	5	6	7

   d) Eu penso que a política fornece as operações corretos ou confiáveis durante a resolução de inconsistências de atualizações.

1	2	3	4	5	6	7

   e) Eu acho que a política pode fornecer resultados eficazes.

1	2	3	4	5	6	7

   f) A política pode resolver inconsistências sem muita intervenção humana.

1	2	3	4	5	6	7

   g) O usuário local ficará bastante insatisfeito com o resultado gerado pela política, caso esteja resolvendo inconsistências nas mesmas fontes de dados (no mesmo estado corrente), concomitante com (ao mesmo tempo que) outros usuários, pois nesse caso a operação com maior *timestamp* não representa, necessariamente, a correta.

1	2	3	4	5	6	7

   h) O usuário local ficará bastante satisfeito com o resultado gerado pela política pois, desde ela é aplicada a um ambiente colaborativo, a última alteração feita é a mais provável de ser correta.

1	2	3	4	5	6	7

3. Para quais aplicações/ambientes, você considera essa política adequada?

Sobre a *política baseada em votação*:

1. Qual a sua opinião sobre a *política baseada em votação*?

2. Analise as afirmativas abaixe e marque o valor de 1 a 7 corresponde a sua opinião, sendo que 1 significa "discordo completamente" e 7 significa "concordo completamente".
   a) A política fornece dados consistentes para o usuário local.

1	2	3	4	5	6	7

   b) A política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.

1	2	3	4	5	6	7

   c) A resolução de inconsistências feita automaticamente (sem a participação dos usuários) é importante para poupar tempo do usuário.

1	2	3	4	5	6	7

   d) Eu penso que a política fornece as operações corretos ou confiáveis durante a resolução de inconsistências de atualizações.

1	2	3	4	5	6	7

   e) Eu acho que a política pode fornecer resultados eficazes.

1	2	3	4	5	6	7

   f) A política pode resolver inconsistências sem muita intervenção humana.

1	2	3	4	5	6	7

   g) O usuário ficará bastante satisfeito com o resultado gerado pela política, pois a opinião da maioria sobre o valor de um atributo é, normalmente, a correta.

1	2	3	4	5	6	7

   h) O usuário ficará bastante satisfeito com o resultado gerado pela política, pois caso ele não saiba o valor de um atributo , ele poderá tirar proveito de operações feitas por outros cuja maioria acredita que tenha o valor correto.

1	2	3	4	5	6	7

3. Para quais aplicações/ambientes, você considera essa política adequada?

Sobre a *política baseada na confiança nas fontes*:

1. Qual a sua opinião sobre a *política baseada na confiança nas fontes*?

2. Analise as afirmativas abaixe e marque o valor de 1 a 7 corresponde a sua opinião, sendo que 1 significa "discordo completamente" e 7 significa "concordo completamente".
   a) A política fornece dados consistentes para o usuário local.

1	2	3	4	5	6	7

   b) A política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.

1	2	3	4	5	6	7

   c) A resolução de inconsistências feita automaticamente (sem a participação dos usuários) é importante para poupar tempo do usuário.

1	2	3	4	5	6	7

   d) Eu penso que a política fornece as operações corretos ou confiáveis durante a resolução de inconsistências de atualizações.

1	2	3	4	5	6	7

   e) Eu acho que a política pode fornecer resultados eficazes.

1	2	3	4	5	6	7

   f) A política pode resolver inconsistências sem muita intervenção humana.

1	2	3	4	5	6	7

   g) O usuário ficará bastante satisfeito com o resultado gerado pela política, pois se ele sempre faz atualizações nos currículos de outros autores, baseando-se nos dados do currículo A, é provável que ele confie mais na fonte A. Portanto, resolver inconsistências priorizando as operações da fonte A gerará uma boa solução para o usuário U1.

1	2	3	4	5	6	7

3. Para quais aplicações/ambientes, você considera essa política adequada?

Sobre a *política baseada na confiança nos usuários*:

1. Qual a sua opinião sobre a *política baseada na confiança nos usuários*?

2. Analise as afirmativas abaixe e marque o valor de 1 a 7 corresponde a sua opinião, sendo que 1 significa "discordo completamente" e 7 significa "concordo completamente".
   a) A política fornece dados consistentes para o usuário local.

1	2	3	4	5	6	7

   b) A política é adequada para resolver inconsistências entre as atualizações dos diferentes usuários.

1	2	3	4	5	6	7

   c) A resolução de inconsistências feita automaticamente (sem a participação dos usuários) é importante para poupar tempo do usuário.

1	2	3	4	5	6	7

   d) Eu penso que a política fornece as operações corretos ou confiáveis durante a resolução de inconsistências de atualizações.

1	2	3	4	5	6	7

   e) Eu acho que a política pode fornecer resultados eficazes.

1	2	3	4	5	6	7

   f) A política pode resolver inconsistências sem muita intervenção humana.

1	2	3	4	5	6	7

   g) O usuário ficará bastante satisfeito com o resultado gerado pela política, pois ele poderá tirar proveito de operações feitas por outros usuários nos quais ele confia.

1	2	3	4	5	6	7

   i) O usuário ficará bastante satisfeito com o resultado gerado pela política, pois caso ele poderá mudar de opinião caso uma operação *c* feita por outro usuário tenha um nível de confiança mais alto que uma operação *b* feita por ele próprio, dado que outros usuários nos quais ele confia já confiaram na operação *c* anteriormente. (Basicamente, eu confio na operação *b*, mas os meus amigos, nos quais eu confio, confiam mais na operação *c*, deveria eu mudar de opinião?)

1	2	3	4	5	6	7

3. Para quais aplicações/ambientes, você considera essa política adequada?

( ) Eu entendo e concordo que a análise da minha opinião será divulgada na tese de doutorado de Dayse Silveira de Almeida, podendo, posteriormente ser publicada em artigos científicos da área. Os meus dados pessoais, no entanto, não serão divulgados.

_____

Palmas, de fevereiro de 2015.

A.2 Questionário usado no experimento com o método de propagação

Nome:_____

Função/cargo:_____

Nível de escolaridade/período:_____

Curso:_____ Instituição:_____

Após utilizar o PrInt para resolver inconsistências em 4 fontes de dados, responda:

1. O que você achou de atualizar o(s) mesmo(s) atributo(s) em 4 fontes diferentes? Imagine se fosse um número maior de fontes.

2. Defina com uma palavra a sua opinião sobre isso: _____
3. Marque um valor entre 1 e 7 corresponde a intensidade da sua opinião, sendo que 1 significa "nada/muito pouco" e 7 significa "completamente".

1	2	3	4	5	6	7

4. O que você acha de poder tomar a decisão para um determinado valor de atributo apenas uma vez e esse valor ser propagado automaticamente para as outras fontes que contêm o mesmo objeto? Ou seja, você não precisaria tomar a mesma decisão para o mesmo atributo do mesmo artigo, várias vezes. Dada a sua decisão para um atributo de um objeto uma única vez, ela é copiada para todos os outros currículos que contêm o artigo.

5. Defina com uma palavra a sua opinião sobre isso: _____
6. Marque um valor entre 1 e 7 corresponde a intensidade da sua opinião, sendo que 1 significa "nada/muito pouco" e 7 significa "completamente".

1	2	3	4	5	6	7

7. Se fosse tivesse que alterar o mesmo atributo, em várias fontes diferentes, e o fizesse em dias diferentes, você acredita que poderia esquecer o valor que colocou no atributo da fonte 1, e colocar um valor diferente vários dias depois na fonte 7. Ou ainda, você está alterando o local de publicação de um artigo para "*Simpósio Brasileiro de Banco de Dados*". Você acredita que depois alterar esse valor em 20 currículos, você poderá ficar cansado e começar a editar as próximas fontes apenas com "*SBBD*"?