Universidade Federal do Ceará
Centro de Ciências
Pró-Reitoria de Pesquisa e Pós-Graduação
Mestrado e Doutorado em Ciência da Computação

DTX: UM MECANISMO DE CONTROLE DE CONCORRÊNCIA DISTRIBUÍDO PARA DADOS XML

Leonardo Oliveira Moreira

DISSERTAÇÃO DE MESTRADO

Fortaleza
Julho - 2008

Universidade Federal do Ceará
Centro de Ciências
Pró-Reitoria de Pesquisa e Pós-Graduação

Leonardo Oliveira Moreira

DTX: UM MECANISMO DE CONTROLE DE CONCORRÊNCIA DISTRIBUÍDO PARA DADOS XML

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do título de Mestre.

Orientador: Prof. Javam de Castro Machado, DSc

Fortaleza
Julho - 2008

DTX: UM CONTROLE DE CONCORRÊNCIA DISTRIBUÍDO PARA DADOS XML

Leonardo Oliveira Moreira

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do título de Mestre.

Prof. Javam Machado, DSc
Universidade Federal do Ceará

Prof.a Vânia Vidal, DSc
Universidade Federal do Ceará

Prof.a Marta Mattoso, DSc
Universidade Federal do Rio de Janeiro

Aos meus pais, Maurício e Fátima.

AGRADECIMENTOS

A Deus, por estar sempre ao meu lado, dando-me coragem para enfrentar todos os obstáculos da vida.

Ao Prof. Dr. Javam de Castro Machado por ter aceitado o desafio de orientar este trabalho. Obrigado pela confiança e paciência nos momentos mais delicados.

Ao Prof. M.Sc. Flávio Sousa, pelas sugestões, pela referência bibliográfica sugerida, revisão e pelo incentivo sempre presente.

A minha mãe Fátima e ao meu pai Maurício, exemplos de honestidade e dignidade, que sempre incentivaram e apoiaram a minha educação.

Ao meu irmão e melhor amigo Maurício Moreira Neto, por compartilhar comigo todos os momentos alegres e difíceis, dando-me força para enfretar a vida.

Aos meus tios Rui Verlaine e Jamile Moreira, não só pela força e incentivo, mas também pela preocupação, dedicação e acompanhamento.

A minha namorada, Patrícia Mourão, pelo companheirismo, dedicação, atenção, compreensão e extrema paciência.

Aos colegas do Mestrado, pela excelente relação pessoal que criamos e espero que não se perca. Em especial a Ériko Moreira, Maiquel Sampaio, Simão Gurgel, Lincoln Rocha e Heraldo Carneiro, pelos profissionalismos exemplares, pelos apoios nos momentos bons e menos bons e pelas suas amizades.

Aos meus amigos João Sérgio Ramos e Gustavo Henrique Azevedo, pelas suas sinceras amizades, companheirismo e apoio até hoje.

Aos meus familiares, que sempre me deram amor e força, valorizando meus potenciais.

A todas as pessoas que, direta ou indiretamente, contribuíram para a execução desta Dissertação de Mestrado.

À CAPES pelo fornecimento da bolsa de estudos que garantiu o sustento financeiro necessário à realização deste trabalho.

O conhecimento é o processo de acumular dados;
a sabedoria reside na sua simplificação.
—MARTIN H. FISCHER

RESUMO

XML tornou-se um padrão amplamente utilizado na representação e troca de dados entre aplicações na Web. Com isso, grande volume desses dados está distribuído na Web e armazenado em diversos meios de persistência. SGBD relacionais que suportam XML fornecem técnicas de controle de concorrência para gerenciar esses dados. A estrutura dos dados XML, entretanto, dificulta a aplicação dessas técnicas.

Trabalhos estão sendo propostos e fornecem gerenciamento de documentos XML. A maioria destes trabalhos, todavia, não oferecem um controle de concorrência eficiente para dados distribuídos. Outros trabalhos dão suporte ao controle distribuído de dados XML, mas estes possuem protocolos com baixo grau de concorrência e limitações. Para prover um gerenciamento eficaz em ambientes distribuídos, este trabalho apresenta o DTX, como mecanismo para o controle de concorrência distribuído para dados XML, que leva em consideração características estruturais destes dados.

O DTX visa a um gerenciamento eficaz de dados XML e contemplar as propriedades de isolamento e consistência em transações, utilizando um protocolo para controle de concorrência multigranular que aumenta o paralelismo entre as transações e possui uma estrutura otimizada para representação dos dados. A solução proposta possui uma arquitetura modular e flexível, o que facilita sua integração com diferentes estruturas de armazenamento XML, além de poder ser estendido, adicionando novos recursos.

Para validar o DTX, diversos testes foram feitos, comparando o DTX descrito neste trabalho com uma variação do DTX, utilizando um protocolo de maior granulosidade, visando a simular as estratégias dos trabalhos relacionados. Os resultados obtidos atestam a eficácia do DTX, considerando diferentes aspectos em transações distribuídas a dados XML, melhorando o desempenho, ou seja, o tempo de execução destas transações.

Palavras-chave: XML, Transações Distribuídas, Controle de Concorrência

ABSTRACT

XML has become a widely used standard for data representation and exchange among Web applications. Consequently, a large volume of such data is distributed all over the Web and stored using several persistence methods. DBMSs provide concurrency control techniques to manage data. However, the structure of XML data makes it difficult to use these techniques.

Projects are being proposed and they provide management of XML documents. Nevertheless, most of these projects do not provide efficient concurrency control mechanisms for distributed data. Some of them do provide support for distributed control of XML data, but use protocols that have limitations and offer low concurrency levels. In order to provide effective data management in distributed environments, we present DTX, a distributed concurrency control mechanism for XML data that takes into account its structural characteristics.

DTX aims to provide effective management of XML data and contemplate properties such as isolation and consistency in transactions, using a multi-granular concurrency control protocol that increases parallelism among transactions and that has an optimized structure for data representation. DTX has a modular and flexible architecture, allowing for easy integration with any XML storage mechanisms. Moreover, DTX can be extended by adding new features to it.

In order to evaluate DTX, several experiments were conducted comparing DTX as it is with a variation of DTX that uses a fine-grained protocol, in an attempt to simulate existing strategies in related work. Results confirm DTX's effectiveness considering different aspects of distributed transactions on XML data, improving their performance, i.e., transaction execution time.

Keywords: XML, Distributed Transactions, Concurrency Control

SUMÁRIO

LISTA DE ABREVIATURAS

API - Application Programming Interface

SGBD - Sistema Gerenciador de Banco de Dados

SGBDXN - Sistema Gerenciador de Banco de Dados XML Nativo

DOM - Document Object Model

DTD - Document Type Definition

DTX - Distributed Transaction on XML

HTTP - Hyper Text Transfer Protocol

RMI - Remote Method Invocation

S2PL - Strict Two-Phase Locking

URL - Uniform Resource Locator

XML - eXtensible Markup Language

XSD - XML Schema Definition

2PC - Two-Phase Commit

2PL - Two-Phase Locking

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

Esta dissertação descreve um mecanismo que provê um controle de concorrência distribuído para dados XML. O trabalho visa a garantir a consistência e o isolamento global de dados XML em ambiente distribuído, melhorando o desempenho mediante acesso concorrente aos dados fisicamente distribuídos. O presente capítulo relata a motivação para o desenvolvimento deste trabalho, assim como os objetivos que se pretende alcançar. Ao final, será descrito como está organizado o restante desta dissertação.

1.1 MOTIVAÇÃO E CARACTERIZAÇÃO DO PROBLEMA

XML [2] tornou-se um padrão amplamente utilizado na representação e troca de dados em aplicações. Como conseqüência, grande volume de dados deste tipo está distribuído na Web e armazenado em diversos meios de persistência. Com isso, aplicações precisam ter acesso e manipular dados distribuídos. Portanto, torna-se necessária a existência de sistemas eficientes para o gerenciamento destes documentos XML em ambiente distribuído.

Para gerenciar estes dados, os SGBD relacionais implementam protocolos de controle de concorrência distribuídos. A estrutura dos dados XML, entretanto, dificulta a aplicação destes protocolos, afetando o nível de concorrência, tanto em ambientes centralizados como distribuídos.

No contexto centralizado, existe grande quantidade de propostas de controle de concorrência para dados XML. Alguns protocolos são baseados em bloqueios hierárquicos em árvores, e muitos deles de forma descendente, ou seja, os nós desde o ponto inicial da consulta até o final do documento são bloqueados [3], diminuindo a concorrência de consultas.

Protocolos baseados no modelo DOM utilizam diferentes tipos de bloqueio para agrupar nós de níveis distintos, o que ocasiona um número elevado de conflitos e, conseqüentemente, ocorre um número maior de bloqueios em conflito. Protocolos baseados

em bloqueios de caminho aumentam a concorrência, mas utilizam um subconjunto muito limitado da linguagem XPath e métodos dispendiosos para determinar conflitos entre consultas complexas, que inviabilizam sua aplicação em ambientes práticos. Alguns protocolos se servem de estruturas como o *DataGuide* [4] para gerenciar o acesso aos dados e apresentam melhores resultados.

Considerando o ambiente distribuído, há ainda poucos trabalhos como [5] [6] [7]. Estes apresentam baixo nível de concorrência, afetando o tempo de resposta e não apresentam estudos que avaliem explicitamente aspectos de desempenho. Após os resultados obtidos em uma revisão bibliográfica, nota-se na literatura a carência de soluções de concorrência em ambientes distribuídos que levam em consideração a estrutura destes dados.

1.2 OBJETIVO E CONTRIBUIÇÃO

Visando a prover um gerenciamento eficaz em ambientes distribuídos, esta dissertação apresenta o DTX, como mecanismo para garantir a consistência e o isolamento global de dados XML em ambiente distribuído, servindo-se da adaptação de um protocolo para controle de concorrência de baixa granulosidade que considera características estruturais de XML. Aspectos de desempenho são levados em conta para avaliação do trabalho proposto.

O DTX faz uso de técnicas síncronas para garantir a consistência e o isolamento entre os sítios que compõem um sistema, sempre visando a melhorar o desempenho de transações distribuídas sobre dados XML. Além disso, pretende-se que o DTX possa integrar diversos tipos de sistemas que armazenam XML. A principal contribuição desta dissertação é o mecanismo para controle de concorrência distribuído em dados XML, servindo-se de adaptações feitas em um protocolo de controle de concorrência centralizado para o âmbito distribuído. A fim de avaliar a proposta, este trabalho estendeu o *benchmark* XMark, o que constituiu contribuição complementar.

1.3 ESTRUTURA DA DISSERTAÇÃO

Os próximos capítulos deste ensaio estão estruturados da seguinte forma:

- Capítulo 2 - apresenta os conceitos relativos ao gerenciamento de dados XML, abordando assuntos básicos de XML, meios de armazenamento, processamento de consultas, protocolos para controle de concorrência e os trabalhos relacionados à presente investigação.

- Capítulo 3 - descreve em detalhes o DTX. Características, especificação, um cenário de execução e os algoritmos desenvolvidos são explicados.

- Capítulo 4 - serão mostrados aspectos de implementação e a avaliação realizada no trabalho.

- Capítulo 5 - traz as considerações finais da pesquisa.

CAPÍTULO 2

GERENCIAMENTO DE DADOS XML

Este capítulo discute as principais características relacionadas ao gerenciamento de dados XML, destacando as estruturas de armazenamento e processamento de consultas. Nele também são abordados alguns protocolos utilizados para o controle de concorrência a esses dados. Por fim, são apresentados os trabalhos relacionados encontrados na literatura.

2.1 XML

Em razão dessa crescente utilização do XML, torna-se necessária a existência de sistemas eficientes de armazenamento e recuperação de dados nesse formato. Existem várias estratégias para o gerenciamento de dados XML, dentre as quais se pode destacar: SGBD-XNs e os SGBDs XML habilitados.

A flexibilidade na estrutura dos dados XML impõe novos desafios para o processamento de consultas e de transações, de modo que novas técnicas devem ser desenvolvidas, tanto em ambientes centralizados como distribuídos. Isso decorre principalmente do modelo de representação destes dados, onde os documentos XML são representados em grafo, o que adiciona maior complexidade à sua estrutura e à heterogeneidade, visto que um mesmo subelemento pode ser omitido ou repetido várias vezes. Por exemplo, o processamento de consultas a dados XML ainda não dispõe de uma álgebra padrão, o que dificulta a decomposição e a otimização de consultas. Com relação aos protocolos para controle de concorrência, a maioria das soluções existentes proporciona baixo nível de concorrência, bloqueando grande parte dos dados XML.

Uma organização bastante utilizada no gerenciamento de dados XML é o conceito de coleções de documentos XML. Uma coleção é um agrupamento de documentos de acordo com sua relevância semântica [8]. As coleções têm papel similar às tabelas em SGBDs relacionais ou diretórios em um sistema de arquivos, também sendo usados nos SGBDs orientados a objetos. A tecnologia XML apresenta um conjunto de especificações

que possibilita o gerenciamento de dados neste formato, tais como: validação e processamento de documentos, e linguagens de manipulação. Os grandes fabricantes de sistemas estão integrando o formato XML cada vez mais em seus produtos, tais como: SGBDs, *browsers* e ferramentas de desenvolvimento [9].

2.2 FORMAS DE ARMAZENAMENTO

Documentos XML podem ser armazenados de formas diversas, da maneira mais simples, em sistemas de arquivos, passando por SGBDs XML habilitados até SGBDXNs. Essas modalidades existem para serem utilizadas em diferentes contextos. Em aplicações de pequeno porte, onde os documentos são pequenos e raramente modificados, a abordagem de sistemas de arquivos oferece grandes benefícios, por ser o modo mais simples. Para aplicações de médio e grande porte, onde existem grandes volumes de documentos e operações de manipulação, o uso de SGBDs habilitados e SGBDXNs torna-se mais atrativa.

Em SGBDs habilitados, um SGBD tradicional emprega técnicas de mapear dados XML para seu formato, a fim de aproveitar o poder de processamento existente nesse modelo [10]. Como exemplo de SGBDs habilitados, pode-se mencionar os sistemas Oracle 9i e o DB2. Diversas técnicas de mapeamento são propostas pela comunidade acadêmica [11] [12] [13].

Segundo [14], SGBDXNs são sistemas construídos especialmente para o gerenciamento de dados XML, ou seja, possuem a capacidade de definir, criar, armazenar, manipular, publicar e recuperar documentos ou fragmentos de documentos XML. Nesses sistemas, um documento XML é representado como um grafo ou uma estrutura de árvore, onde os nós representam elementos e atributos, e as arestas definem os relacionamentos elemento/subelemento ou elemento/atributo [15]. Os SGBDXNs implementam muitas características presentes em sistemas tradicionais, tais como armazenamento, indexação, processamento de consultas, transação e replicação. São exemplos desse tipo de sistema [6] [14] [15] [16] [17] [18].

2.3 PROCESSAMENTO DE CONSULTAS

O desenvolvimento de técnicas de processamento de consultas sobre dados XML é uma tarefa desafiadora [19]. Isso decorre principalmente das seguintes características:

- modelo de dados - documentos XML são representados por um modelo de dados baseado em grafo, o que adiciona maior complexidade à sua estrutura; e
- heterogeneidade - um documento XML pode ter um mesmo subelemento omitido ou repetido várias vezes.

Sob o ponto de vista de banco de dados, os documentos ou dados XML são semi-estruturados, onde o esquema dos dados de um documento é variável e nem sempre conhecido previamente, diferentemente dos modelos estruturados, onde o perfil dos dados é explicitamente declarado mediante um esquema [8].

É importante também destacar que a flexibilidade do XML que permite consultas sobre documentos, cujos esquemas e índices não são conhecidos previamente, pode resultar na não-realização de otimização e na não-validação de consultas. O uso de um esquema auxilia no processamento de consultas, eliminando as expressões de caminho incompatíveis, as expressões que sempre são vazias, favorecem a remoção de condições redundantes [20] e a detecção de certos tipos de erros em tempo de compilação.

Diferentemente dos modelos de dados tradicionais, o XML ainda não dispõe de uma álgebra padrão. Álgebras são usadas para dar semântica a linguagens de consulta e auxiliar sua otimização. Muitos trabalhos são propostos na definição de uma álgebra. Em [21] é apresentado uma álgebra que captura a semântica de muitas linguagens XML e possui várias operações, tais como junção, projeção, seleção, agregação, funções, entre outras. Jagadish et al. [22] propõem uma álgebra, denominada TAX (*Tree Algebra for XML*), que utiliza as características dos dados XML e possui regras análogas à álgebra relacional.

A álgebra TLC (*Tree Logical Classes*) [23] é uma extensão da álgebra TAX [22]. A TLC permite acesso a conjunto de árvores heterogêneas por meio das árvores APTs (*Annotated Pattern Trees*). As APTs aperfeiçoam a aplicação de padrões de árvores (*pattern trees*) de TAX por permitirem o uso de conjuntos heterogêneos. A idéia de classes lógicas (*Logical Classes*) é usada para rotular nós e resultados de árvores de acordo com o padrão de árvores da álgebra aplicada. TLC é baseada nos conceitos de APTs e classes lógicas e define os seguintes operadores de álgebra: filtro, junção, seleção, projeção, eliminação de duplicadas e funções de agregação. A álgebra TLC é implementada no Timber [14].

Em razão da dificuldade de implementação de uma álgebra padrão e da complexidade das linguagens de consultas, o W3C tomou como base a álgebra proposta em [21] e desenvolveu uma semântica formal [24], que possibilita a identificação de ambigüidades na linguagem e auxilia na verificação formal [25].

Linguagens como a XPath e a XQuery contêm uma semântica formal em seus núcleos para a qual as consultas, submetidas em alto nível, são convertidas [20]. Esta conversão proporciona uma série de benefícios inerentes a muitas linguagens funcionais, tais como verificação forte de tipos e facilidades na utilização de técnicas de otimização, como, por exemplo, prevenção de nós duplicados. Michiels [20] discute o *status* da semântica formal como álgebra e apresenta um comparativo entre elas.

XPath e XQuery não possuem suporte a operações de atualização de documentos, tais como inserção ou remoção [26]. Alguns trabalhos propõem soluções para esse problema. Em [27] é apresentada uma linguagem para atualização de documentos XML denominada XUpdate. Já [28] discute alterações no núcleo da linguagem XQuery, fornecendo suporte a atualização.

2.4 CONTROLE DE CONCORRÊNCIA

A estrutura em grafo dos dados XML dificulta o desenvolvimento de protocolos para controle de concorrência a esses dados [8]. Diversos protocolos [29] [30] [31] [32] [1] [33] [34], desenvolvidos pela comunidade acadêmica, utilizam técnica de bloqueios para realizar o isolamento entre transações. Essas soluções podem ser agrupadas em protocolos que realizam bloqueios em árvores, baseados no modelo DOM e multigranular [35].

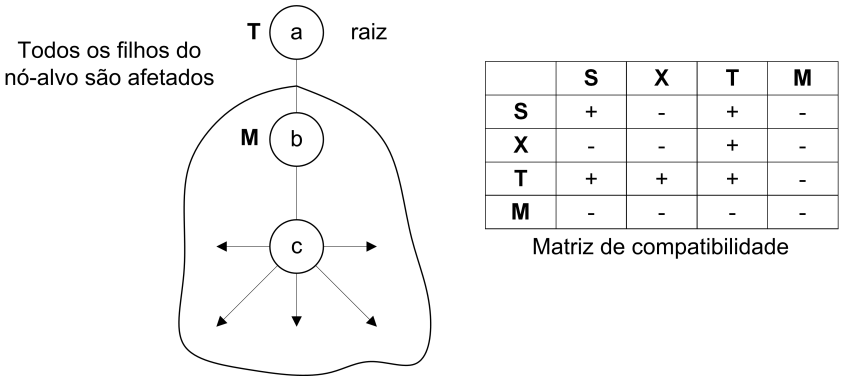
2.4.1 Protocolos Baseados no Bloqueio em Árvores

O primeiro grupo de protocolos implementa o processamento das transações, executando operações de consultas e de modificações de estruturas em árvore. A aplicação dos bloqueios ocorre de forma descendente, ou seja, começando do elemento-raiz e descendo até os elementos-alvos [3].

Em [31] é apresentada uma categoria inserida nestes protocolos, denominada Doc2PL, que bloqueia todo o documento. Esse tipo de protocolo tem o mais baixo

grau de concorrência, pois, enquanto o documento estiver bloqueado em modo exclusivo, nenhuma outra transação poderá utilizá-lo. Neste trabalho é expresso que o SGBDXN Tamino [6] utiliza este protocolo.

Outra variante da categoria de protocolos, denominada Node2PL, adquire bloqueios para os nós ancestrais do elemento-alvo e suas subárvores também são afetadas. Para isto, esta categoria de protocolos utiliza três tipos de bloqueios: de estrututra, de conteúdo e de acesso direto. O primeiro é aplicado de maneira apropriada nos nós-ancestrais dos nós-alvos de uma operação de consulta ou modificação. O segundo é utilizado nas modificações dos nós, ou seja, em alterações dos valores dos atributos ou do valor do próprio nó. Cada elemento do documento XML possui identificador único que permite diferenciá-lo e localizá-lo dentre os outros. Todos os nós possuem identificador único na árvore XML e existem meios para acessar um nó diretamente pelo seu identificador. O terceiro tipo de bloqueio é aplicado para proteger os nós alcançados diretamente pelo identificador [35]. Com os bloqueios de acesso direto, é garantido que nenhuma outra transação fará acesso via identificador a qualquer nó de uma subárvore-alvo de uma modificação.



	S	X	T	M
S	+	-	+	-
X	-	-	+	-
T	+	+	+	-
M	-	-	-	-

Matriz de compatibilidade

Figura 2.1 Exemplo do protocolo Node2PL

Protocolos com essas características possuem muita restrição, por bloquearem o nó-ancestral dos nós-alvos e tendo as subárvores do nó-alvo afetadas pelo bloqueio. A Figura 2.1 mostra a matriz de compatibilidade e o comportamento do protocolo Node2PL diante de um situação de consulta ou modificação de algum filho do nó *b*. Na matriz de compatibilidade, o caracter "+" denota compatibilidade entre bloqueios, já o caracter "-" significa incompatibilidade entre bloqueios. No exemplo da figura, o nó *c*, alvo da consulta, e todos os seus filhos estão sendo afetados por bloqueios. Os bloqueios `S` e `X` referem-se aos bloqueios de conteúdo compartilhado e exclusivo, respectivamente; já os

bloqueios T e M estão relacionados aos bloqueios de estrutura, onde T representa que estes nós dão acesso ao nó-alvo e M indica que os filhos deste nó estão sendo bloqueados para alguma modificação. Existe um melhoramento deste grupo de protocolos, denominado NO2PL, que consiste em bloquear somente os nós-alvos e alguns filhos em operações de atualização, assim diminuindo a granulosidade do bloqueio. A Figura 2.2 ilustra o funcionamento do protocolo diante da inserção do nó *c* antes do nó *d*. Na Figura 2.2, é possível observar que este protocolo segue a mesma matriz de compatibilidade do protocolo anterior e mostra que, para realizar uma modificação no nó *c*, os nós *b* e *d* são bloqueados com o bloqueio M, pois dão acesso ao nó-alvo. Já nos ancestrais de *b* e *d*, são aplicados bloqueios T de estrutura.

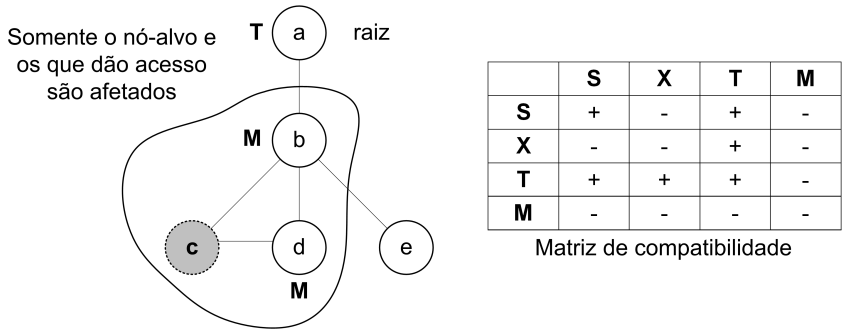
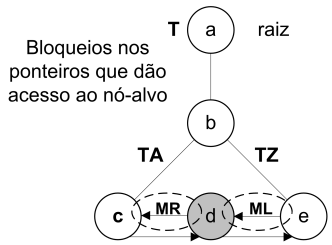


Figura 2.2 Exemplo do protocolo NO2PL

No NO2PL, não é necessário bloquear todos os nós do nível do nó-alvo. Outra categoria deste grupo de protocolos, chamada OO2PL, realiza bloqueios apropriados ao tipo de operação nos ponteiros que ligam ao nó que é alvo da operação. Essas referências podem vir do primeiro/último filho do nó-ancestral e/ou do irmão à esquerda/direita. A Figura 2.3 mostra o comportamento deste protocolo diante da remoção do nó *d*. A matriz de compatibilidade, conforme a figura, foi expandida seguindo os mesmos bloqueios dos protocolos anteriores, ou seja, criando quatro variações dos bloqueios M e T que representam as direções dos ponteiros. O bloqueio TA é utilizado para bloquear o ponteiro de *b* para *c*, pois *c* é o primeiro filho de *b* e este é caminho para o nó *d*. O bloqueio TZ é semelhante ao bloqueio TA, mas é aplicado no ponteiro para o último filho do nó pai. O bloqueio *MR* é aplicado no ponteiro que liga *c* para *d*, pois *c* alcança *d* como irmão à direita e a operação é de modificação. O bloqueio ML é semelhante ao bloqueio MR, mas *e* alcança *d* como irmão à esquerda.

O OO2PL fornece menor granulosidade de bloqueio, mas é necessário gerenciar quatro tipos de bloqueios por nó [31]. Realizando breve análise entre os protocolos

Doc2PL, Node2PL, NO2PL e OO2PL, ocorre que nesta ordem há um aumento do grau de concorrência, ou melhor, paralelismo entre transações. Assim, como descrito no protocolo Node2PL, os bloqueios de acesso direto são aplicados também nos protocolos NO2PL e OO2PL, pois estes são variações do Node2PL.



	TL	TR	TA	TZ	ML	MR	MA	MZ
TL	+	+	+	+	-	+	+	+
TR	+	+	+	+	+	-	+	+
TA	+	+	+	+	+	+	-	+
TZ	+	+	+	+	+	+	+	-
ML	-	+	+	+	-	+	+	+
MR	+	-	+	+	+	-	+	+
MA	+	+	-	+	+	+	-	+
MZ	+	+	+	-	+	+	+	-

Matriz de compatibilidade

Figura 2.3 Exemplo do protocolo OO2PL

Em [29] é apresentado um protocolo baseado em bloqueios hierárquicos em árvores. Apesar de aplicar os bloqueios sobre as árvores, estrutura semelhante à dos documentos XML, esse protocolo possui baixo grau de concorrência, pois os bloqueios afetam toda a subárvore-alvo da consulta. Assim sendo, consultas XPath não podem ser aplicadas eficientemente, pois podem ser executadas sobre partes distintas de um documento que, por sua vez, podem estar bloqueadas desnecessariamente.

Os protocolos desenvolvidos em [31] [32] contemplam as características das linguagens XML, utilizando bloqueios de caminho para aumentar a concorrência e melhorar o desempenho das consultas. Esses protocolos apresentam, no entanto, algumas desvantagens, como a utilização de um subconjunto muito limitado da linguagem XPath e o uso de métodos dispendiosos para determinar conflitos entre as consultas mais complexas, que inviabilizam sua aplicação [8].

Segundo [35], a abordagem do grupo de protocolos de bloqueio em árvores contém uma série de desvantagens. A maior deficiência está relacionada ao tratamento das operações de acesso direto aos elementos XML e por bloquear as subárvores destes elementos.

2.4.2 Protocolos Baseados no Modelo DOM

Protocolos baseados no modelo DOM [30] [36] usam diferentes tipos de bloqueio para agrupar nós de níveis distintos, ocasionando elevado número de conflitos [8]. Em [36] é

apresentado um protocolo baseado no modelo DOM. Este trabalho representa o documento XML como uma extensão da estrutura XML do modelo DOM; com isso, aproveita a manipulação do documento XML por meio da DOM API. Diversos tipos de bloqueios são utilizados para diferentes tipos de operações expressadas por funcionalidades da DOM API. Todos esses bloqueios devem ser obtidos antes de ter acesso ao nó, durante uma navegação ou modificação do documento XML.

No protocolo apresentado em [36], existem bloqueios, denominados bloqueios de nós, que servem para leituras do nó, de nível do documento, de subárvores e bloqueios exclusivos para alterações em nós e filhos. Ao conseguir bloqueios exclusivos nos nós, em seus ancestrais são aplicados bloqueios exclusivos de intenção. Além desses bloqueios, existe um que é utilizado em situações em que se tenciona efetuar uma leitura seguida de uma escrita. Outro grupo de bloqueios, denominado bloqueio de navegação, é utilizado para as operações do modelo DOM que tem acesso a elementos diretamente mediante um identificador. Para detectar e tratar *deadlock*, este protocolo utiliza a técnica de grafo de esperas; caso ocorra *deadlock*, a transação que efetuou menor número de atualizações é cancelada [30].

Esses protocolos são utilizados por vários sistemas e apresentam resultados satisfatórios com operações, de manipulação, sobre o modelo DOM servindo-se dos recursos da DOM API. Não existem, todavia, estudos que comprovem a eficiência desses protocolos quando linguagens de consultas XML, como a XQuery, são submetidas [8]. Isso decorre do fato de que estes protocolos utilizam variações da DOM API para a manipulação dos dados XML.

2.4.3 Protocolos Multigranulares

Os protocolos multigranulares, em geral, bloqueiam todos os ancestrais do nó-alvo da consulta. Para isto, são aplicados bloqueios apropriados a cada tipo de operação, com seus respectivos bloqueios de intenção. Estes servem como indicativo para o gerenciador de bloqueios de que existem nós da subárvore do nó-alvo bloqueados em determinado modo; com isso, conferem agilidade à detecção de conflitos entre bloqueios. Estes protocolos protegem os nós dos acessos diretos pelas regras de compatibilidade de bloqueios dos ancestrais. Em [35] é discutida uma solução para tratamento de acesso direto a nós.

O Natix [15] utiliza uma variante do protocolo multigranular com variados níveis

para o controle de concorrência. A granulosidade do bloqueio pode ser de segmentos do documento, documento inteiro, subárvore e nível de registro. O documento pode ser separado em níveis de registro, utilizando uma matriz de corte do protocolo; isso fornece maior grau de concorrência, pois partes do documento podem ser armazenadas em diferentes registros [37]. Estes são semelhantes às páginas de dados em SGBD tradicionais.

Em [38] é proposto um protocolo, denominado XR-Lock, que possui alto grau de concorrência, enquanto garante a "seriabilidade" de transações operando em um mesmo documento. Este protocolo utiliza um subconjunto da linguagem XPath para realizar consultas. Aplica-se um bloqueio apropriado ao conjunto de nós contido em uma região. Conflitos que violam a "seriabilidade" são detectados a cada operação de leitura ou escrita. Cada nó possui um número identificador, e uma região é um conjunto de nós que possuem semelhanças. O RRC (*Relative Region Coordinate*) é utilizado para descrever o intervalo de bloqueios e usado como coordenada do nó no documento XML. De posse da região do nó, o escalonador pode decidir que subelementos devem ser bloqueados. A Figura 2.4 mostra um exemplo, adaptado de [38], que mostra um documento XML e a tabela de bloqueios aplicados a este documento contendo os nós, as coordenadas do nó em RRC e o seu respectivo bloqueio. O par ordenado RCC é obtido pelo número de *bytes* que seleciona o nó-alvo no documento. Os bloqueios `X` estão relacionados aos bloqueios exclusivos e os bloqueios `S` estão associados aos bloqueios compartilhados. Uma região bloqueada com `S-Lock` pode incluir bloqueios `S-Lock` ou `X-Lock`, mas uma região bloqueada com `X-Lock` só pode possuir bloqueios `X-Lock`, conforme ilustrado no exemplo.

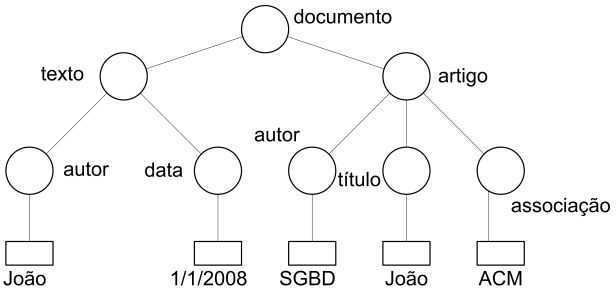


Tabela de bloqueios

OID	Nó	Região	Bloqueio
1	documento	(1, 202)	S-Lock
2	texto	(11, 84)	X-Lock
3	artigo	(85, 191)	S-Lock
4	autor	(9, 39)	X-Lock
5	data	(40, 65)	X-Lock
6	título	(8, 42)	S-Lock
7	autor	(43, 68)	X-Lock
8	associação	(69, 185)	X-Lock

Figura 2.4 Exemplo do protocolo XR-Lock

Em [1] é apresentado o protocolo DGLOCK, que contempla as principais características de concorrência a dados XML. Nesse trabalho, são exibidos resultados que comprovam sua eficiência. Uma desvantagem do DGLOCK é não garantir o critério da "seriabilidade" [33], característica fundamental a protocolos de concorrência e, por isso, pode gerar inconsistências. Ele utiliza bloqueio granular em grafos direcionados acíclicos

representados em *DataGuide* [4]. Para tratar *deadlock*, o DGLOCK utilizada a abordagem de manter um grafo de espera que represente as transações que estão aguardando por bloqueios. Se a transação for inserida no grafo de espera e isso produzir um ciclo, a transação mais recente será abortada.

Este protocolo utiliza três classes de bloqueios: compartilhados, exclusivos e de intenção. Estes últimos identificam que algum nó de sua subárvore está bloqueado por determinado tipo de bloqueio. Estas regras provêem o bloqueio de granulosidade fina e são aplicadas a predicados simples, isto é, em conjunção de igualdade e desigualdade. A incompatibilidade entre bloqueios ocorre também quando bloqueios sobre predicados são permitidos e algum bloqueio requisitado não é conseguido. A Figura 2.5 ilustra dois exemplos, sendo que, em um deles, há uma incompatibilidade entre bloqueios no teste do predicado das transações *t3* e *t4*, pois ambos os predicados trabalham sobre o valor "XML" do elemento descrição, e os bloqueios são incompatíveis. Além destes exemplos, a Figura 2.5 mostra a matriz de compatibilidade deste protocolo. Esta figura foi adaptada do trabalho [1]. Na matriz de compatibilidade do DGLOCK estão descritos cinco bloqueios. Os bloqueios `S` (*Shared*) e `X` (*Exclusive*) representam, respectivamente, os bloqueios compartilhados e exclusivos. Já os bloqueios `IS` (*Intension Shared*) e `IX` (*Intension Exclusive*) ilustram, respectivamente, os bloqueios de intenção de operações compartilhadas e exclusivas. O bloqueio `SIX` representa a combinação dos bloqueios `S` e `IX` em situações de leituras sobre algo que já está sendo bloqueado com `IX` pela mesma transação.

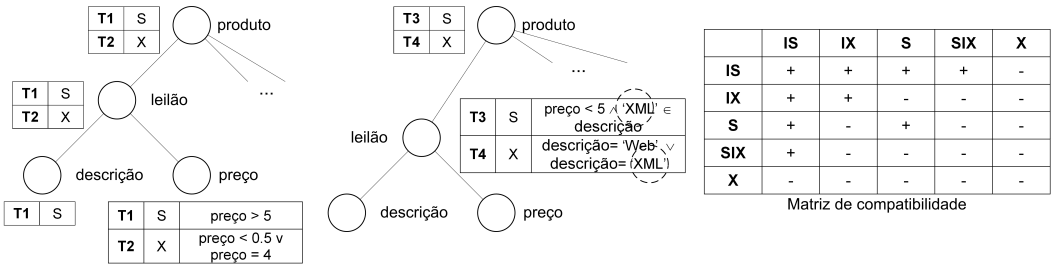


Figura 2.5 Exemplo do protocolo DGLOCK. Adaptado de [1]

Pleshachkov et al. [34] apresentam o XDGL, um protocolo de bloqueios baseado em operações XPath. O XDGL é uma extensão do protocolo DGLOCK, que utiliza uma estrutura *DataGuide* para representar os bloqueios. Esse protocolo pode ser implementado no topo de qualquer sistema que gerencia dados XML, uma vez que possui uma estrutura própria de representação. Por utilizar uma estrutura otimizada para represen-

tar bloqueios, o XDGL possui mais eficiência na gerência destes bloqueios. Este protocolo requer que a transação esteja em acordo com o S2PL [39]. Portanto, a transação adquire e mantém bloqueios até o seu término. São promovidos os princípios de bloqueios em granulosidade múltipla. Assim, como no protocolo DGLOCK, são introduzidos os bloqueios de intenção. Ao contrário de seu protocolo-base, o XDGL garante o critério de "serializabilidade".

O XDGL utiliza um subconjunto da linguagem XPath para recuperar informações dos documentos XML. Para a atualização de dados nos documentos XML, foi definida uma linguagem de atualização. Esta linguagem possui cinco tipos de operações para atualização: inserção, remoção, transposição, renomeação e troca. Detalhes sobre estas operações podem ser observadas no Apêndice A.

Neste protocolo, há duas possibilidades de bloqueios: em árvores e em nós. Em árvores, são gerenciadas partes da estrutura que representa o documento XML, já no bloqueio em nós, são tratados simples elementos da representação. O XDGL utiliza um subconjunto da linguagem XPath para recuperar informações dos documentos XML. XPath define uma sintaxe e semântica em operações de localização. Uma operação de localização é composta de uma linha central que define a direção para a localização dos nós-alvos, um teste de nós que especifica os tipos de nós a serem selecionados e opcionalmente um predicado que define um filtro sob esses nós especificados. Para este protocolo, são tratadas somente operações que visam a localizar nós-filhos, descendentes e atributos.

Os bloqueios em nós e em árvores são totalizados em oito tipos. Os bloqueios `SI` (*shared into*), `SA` (*shared after*) e `SB` (*shared before*) são utilizados como bloqueios compartilhados em operações de inserção. Esses bloqueios previnem qualquer modificação do nó especificado na expressão de caminho por alguma transação concorrente. O bloqueio `X` (*exclusive*) é utilizado no nó a ser modificado. Os nós aninhados podem ser lidos por outras transações. O bloqueio `ST` (*shared tree*) é aplicado em uma subárvore do *DataGuide* para protegê-la de alguma atualização. Já o bloqueio `XT` (*exclusive tree*) trava uma subárvore do *DataGuide* para leituras e modificações. O bloqueio de intenção `IS` (*intention shared*) é obtido em cada ancestral do nó bloqueado em modo compartilhado. Já o `IX` (*intention exclusive*) deve ser aplicado em cada ancestral do nó bloqueado em modo exclusivo.

Ao realizar uma consulta XPath, são aplicados `ST` nos nós-alvos e bloqueios `IS` em

seus ascendentes. Na execução de uma inserção é obtido bloqueio `X` no nó a ser inserido e, em seus ascendentes, são aplicados bloqueios `IX`. No nó que liga ao nó alvo, é aplicado um bloqueio `SI` e `IS` em seus ascendentes. Nos nós-alvos do predicado da expressão de caminho são empregados bloqueios `ST` e `IS` em seus ascendentes.

Na ato de remoção aplicam-se bloqueios `XT` nos nós-alvos da operação e bloqueios `IX` em seus ascendentes. Nos nós que fazem parte do predicado na expressão de consulta, são obtidos bloqueios `ST` e `IS` em seus ascendentes. Ao efetuar uma operação de renomeação, aplicam-se bloqueios `XT` nos nós-alvos e bloqueios `IX` em seus ascendentes. Nos nós que fazem parte do predicado da expressão de caminho, são obtidos bloqueios `ST` e `IS` em seus ascendentes.

Ao se realizar de uma transposição, o nó-alvo da expressão de origem é bloqueado em modo `XT` e seus ascendentes em `IX`. Os nós-alvos do predicado da expressão de origem são bloqueados com `ST` e seus ascendentes com `IS`. O nó-alvo da expressão de destino é criado em sua posição, bloqueado com `X` e seus ascendentes com `IX`. O pai do nó de destino é bloqueado com `SI` e seus ascendentes com `IS`. Ao realizar uma operação de troca, os nós-alvos da troca são bloqueados com `X`. Aos ascendentes dos nós-alvos da troca são aplicados bloqueios `IX`. Os nós que fazem parte do predicado da expressão de caminho são bloqueados com `ST` e seus ascendentes com `IS`. A Figura 2.6 ilustra a matriz de compatibilidade entre bloqueios do protocolo XDGL. O carácter "+" denota compatibilidade, já o "-" expressa incompatibilidade entre bloqueios.

	SI	SA	SB	X	ST	XT	IS	IX
SI	-	+	+	-	+	-	+	+
AS	+	-	+	-	+	-	+	+
SB	+	+	-	-	+	-	+	+
X	-	-	-	-	-	-	+	+
ST	+	+	+	-	+	-	+	-
XT	-	-	-	-	-	-	-	-
IS	+	+	+	+	+	-	+	+
IX	+	+	+	+	-	-	+	+

Figura 2.6 Matriz XDGL de compatibilidade

2.5 ANÁLISE COMPARATIVA

Foi realizada uma análise dos protocolos para controle de concorrência estudados, com o objetivo de identificar o protocolo mais adequado ao contexto deste trabalho. Alguns pro-

tocolos baseados no bloqueio em árvores, como o apresentado em [31], realizam bloqueios em todo o documento, o que inviabiliza sua aplicação em sistemas cujo desempenho é um parâmetro crucial. Em [35], apresentam-se modificações deste tipo de protocolo, onde existem diferentes tipos de bloqueios para determinadas situações. Estas variantes, entretanto, possuem alta granulosidade por bloquear nós-ancestrais dos elementos-alvos e suas subárvores, o que afeta diretamente o paralelismo entre transações. Já em [29], é apresentado um protocolo que bloqueia nós desde o ponto inicial da consulta até o final do documento. Assim como os protocolos anteriores, este possui baixo grau de concorrência.

Protocolos baseados no modelo DOM [30] [36] utilizam diferentes tipos de bloqueios para agrupar nós de níveis distintos, o que ocasiona um número elevado de conflitos [8]. Assim, eleva-se atraso entre transações concorrentes, uma vez que, ao conflitarem, uma transação espera pela liberação de um bloqueio pela outra. Outra restrição destes tipos de protocolos é a dependência das operações sobre o modelo DOM.

Já a categoria de protocolos multigranular [38] [1] [34] possui grau maior de paralelismo entre transações, pelo fato de não bloquear todo o documento ou subárvores, apenas pequenos grânulos do documento são bloqueados. Estes protocolos utilizam técnicas mais eficientes para determinar conflitos entre transações, o que melhora o fator desempenho. O protocolo DGLOCK [1] utiliza uma estrutura otimizada para representar a estrutura XML e os bloqueios, o que diminui o tempo de acesso aos elementos XML e a sobrecarga na gerência dos bloqueios. Este protocolo, porém, não garante o critério de "serializabilidade", o que torna este protocolo inviável aos propósitos deste ensaio. Já o protocolo XDGL [34] é uma extensão do DGLOCK; este protocolo estendido garante o critério de "serializabilidade". Após um estudo detalhado dos protocolos de controle de concorrência centralizados, foi escolhido o protocolo XDGL, visto que cobre todas as características necessárias. A Tabela 2.1 ilustra uma breve comparação entre os protocolos estudados; todos estes protocolos possuem recursos para tratamento de *deadlock*.

Protocolo	**Categoria**	**Serializabilidade**	**Granulosidade**
Doc2PL	Árvores	Sim	Documento XML
Node2PL	Árvores	Sim	Subárvores
taDOM	Modelo DOM	Sim	Nível do documento XML
XR-Lock	Multigranular	Sim	Região de Elementos XML
DGLOCK	Multigranular	Não	Elemento XML
XDGL	Multigranular	Sim	Elemento XML

Tabela 2.1 Tabela comparativa entre protocolos para controle de concorrência

2.6 TRABALHOS RELACIONADOS

O Tamino XMLServer é um SGBDXN, desenvolvido pela Software AG [40], que usa uma evolução do SGBD ADABAS como seu mecanismo de persistência de dados. O controle de concorrência do Tamino é baseado no protocolo 2PL [37]. Seu protocolo possui quatro níveis de granulosidade: de banco de dados, coleção de documentos XML, *doctype* e documento XML. Neste, existem seis tipos de bloqueios divididos em dois grupos: *exclusive lock* e *intention lock*. O Tamino suporta transações distribuídas e utiliza o protocolo 2PC para garantir a consistência e a atomicidade das transações distribuídas.

O Berkeley DB XML [41] é um sistema de banco de dados *open-source* que utiliza XQuery para acessar documentos. Este SGBD armazena dados em *containers* e usa técnicas de indexação em seus conteúdos. Este sistema de banco de dados XML é construído no topo do Berkeley DB original, aproveitando suas características. O Berkeley DB XML possui em sua camada um gerente de documentos, um indexador e processamento de consultas baseado em XQuery.

O protocolo para controle de concorrência padrão, utilizado para os recursos transacionais do Berkeley DB, é baseado em bloqueios; basicamente é implementado o 2PL convencional [7]. A granulosidade do bloqueio pode ser no escopo do documento [42] ou banco de dados. O Berkeley DB permite a execução de transações distribuídas [43], utilizando o protocolo 2PC para a garantia da atomicidade e a consistência. Este sistema não faz a detecção de *deadlock* distribuído; então, fica ao encargo da aplicação realizar o tratamento da ocorrência destes problemas. O gerenciador de transações globais do Berkeley DB é implementado utilizando o padrão XA [44].

Em [5] são descritos um projeto e uma implementação inicial de banco de dados XML distribuído. Neste trabalho, é proposta uma camada arquitetural com infraestrutura de acesso a dados que integram diferentes tipos de bancos de dados que suportam XML. Basicamente, esse mecanismo é composto de três componentes funcionais: um gerenciador de transações distribuídas, um processador de consultas distribuídas e um gerenciador distribuído de esquemas.

O mecanismo proposto fornece aos seus usuários transparência de localização, suporte a transações distribuídas em dados XML e acesso uniforme aos diferentes tipos de bancos de dados XML integrados. Fica ao encargo de cada SGBD, que integra o sistema, garantir o acesso concorrente aos dados. A atomicidade global é garantida pela

plataforma, utilizando o protocolo 2PC para coordenar as transações de acordo com o modelo X/Open [44] de processamento de transações distribuídas. Neste trabalho, é enfatizado o fato de que a atomicidade é garantida quando os SGBD possuem baixo grau de acesso concorrente [5]. O gerenciador de transações distribuídas possui um módulo, denominado monitor de *deadlock* distribuído, que utiliza a técnica de grafo de esperas para detecção de *deadlock*.

Os trabalhos relacionados, descritos antes, apresentam baixo nível de concorrência entre transações, por possuirem alta granulosidade de bloqueio. Essa característica afeta diretamente o desempenho, mais precisamente o tempo de resposta na execução de transações. Dentre os trabalhos, o Berkeley DB é o que possui menor granulosidade, mas o usuário necessita decompor os documentos em fragmentos. Além disso, o Berkeley DB deixa a cargo da aplicação detectar e resolver *deadlock* distribuído.

O trabalho [5] deixa o gerenciamento do controle de concorrência sob a responsabilidade de cada banco que compõe o sistema. É válido ressaltar que a plataforma descrita aceita opcionalmente bancos com granulosidade baixa, mas o trabalho em questão garante o critério de "serializabilidade" quando a granulosidade do bloqueio nos bancos é alta. O Tamino contempla todos os recursos pretendidos em transações distribuídas, incluindo tratamento de *deadlock*. Como os outros trabalhos, no entanto, possui baixo grau de paralelismo, por possuir grossa granulosidade de bloqueio em seu protocolo para controle de concorrência. Além do mais, o Tamino é um sistema comercial e não possui nenhuma flexibilidade quanto a adaptações em seu código-fonte. A Tabela 2.2 mostra uma comparação entre os trabalhos relacionados.

	Tamino	**Berkeley DB XML**	**Pagnamenta 2005**
Granulosidade	BD e Coleção	BD e Documento	Depende dos SGBDs
Protocolo	2PL	2PL	Depende dos SGBDs
Transações Distribuídas	2PC	2PC	2PC
Serializabilidade	Sim	Sim	Nem sempre
Tratamento de *deadlock* distribuído	Sim	Não	Sim

Tabela 2.2 Comparação entre os trabalhos relacionados

2.7 CONCLUSÃO

Este capítulo apresentou a tecnologia para gerenciamento de dados XML, destacando os sistemas de armazenamento e o processamento de consultas. Também foram investigados protocolos para controle de concorrência centralizados a dados XML, que influenciaram na construção do protocolo utilizado no DTX. Também foram abordadas algumas vantagens e desvantagens desses protocolos e os principais trabalhos relacionados. Nenhum destes apresenta protocolos que contenham características que aumentem o grau de concorrência entre transações distribuídas em dados XML, ponto este previsto nesta dissertação.

Acredita-se que a adaptação para o ambiente distribuído de um protocolo para controle de concorrência XML multigranular, que contemple os recursos desejados, é uma solução eficaz para o problema do desempenho em transações distribuídas a dados XML. No próximo capítulo, serão explanadas as características do DTX, sua especificação, um cenário de execução e os algoritmos desenvolvidos.

CAPÍTULO 3

DTX: UM MECANISMO DE CONTROLE DE CONCORRÊNCIA DISTRIBUÍDO PARA DADOS XML

Este capítulo descreve o DTX, um mecanismo para o controle de concorrência distribuído para dados XML, cujo propósito fundamental é garantir a consistência e o isolamento global de dados XML em ambiente distribuído. O DTX utiliza um protocolo para controle de concorrência eficaz e otimizado para esses dados, melhorando o acesso concorrente aos dados. Também são discutidos aspectos arquiteturais, especificação, cenário de execução e os algoritmos desenvolvidos.

3.1 CARACTERÍSTICAS

O objetivo deste trabalho é desenvolver um mecanismo para controle de concorrência distribuído para dados XML, que leva em consideração características estruturais destes dados. O DTX utiliza um controle de concorrência multigranular que aumenta o paralelismo entre as transações e por possuir uma estrutura otimizada para representação dos dados.

A manipulação dos dados XML é feita em memória principal. Para isso, o DTX recupera os documentos XML da memória secundária, faz os devidos processamentos em memória principal para um melhor desempenho, e finalmente atualiza as alterações no meio de armazenamento. A estrutura de armazenamento destes documentos é independente, ou seja, o DTX oferece suporte de comunicação com qualquer abordagem de armazenamento de documentos XML. O DTX opera em dados XML replicados total ou parcialmente. No caso de dados XML totalmente replicados, poderiam ser utilizadas soluções para replicação, como o RepliX [45].

Em seu controle de concorrência, o DTX utiliza algumas adaptações do protocolo

XDGL, com a finalidade de contemplar as propriedades de isolamento e consistência em âmbito distribuído. As adaptações realizadas foram:

- inseriu-se uma infra-estrutura de comunicação no escalonador para interação dos escalonadores, a fim de executar operações remotas, ao mesmo tempo em que adquire os bloqueios necessários. Essa infra-estrutura também é útil para a consolidação e o cancelamento de uma transação distribuída;
- foi realizada a distribuição do gerenciador de bloqueios em cada instância do DTX; com isso, é possível descentralizar o módulo de aquisição e liberação de bloqueios, deixando a cargo de cada instância gerenciar os bloqueios sobre seus dados; e
- alterou-se a maneira pela qual o XDGL faz a detecção e o tratamento de *deadlock*; para isso, foi criado um processo que periodicamente percorre todas as instâncias do DTX e verifica se, na união dos grafos de espera, ocorre um ciclo, ou seja, um *deadlock*. A periodicidade deste processo é configurada pelo usuário do DTX

Para utilizar outro protocolo para controle de concorrência, ele deverá passar pelas adaptações descritas há pouco. Pelo fato de utilizar uma adaptação do protocolo para controle de concorrência, o DTX possui limitações quanto à linguagem de consulta e atualização. O subconjunto da linguagem XPath, utilizada para recuperação de informações no protocolo XDGL, é comum à linguagem de consulta do DTX; quanto à linguagem de atualização, segue a mesma idéia. As regras de bloqueios para o processamento das operações do XDGL são comuns àquelas de bloqueios empregadas no processamento de operações do DTX [46].

No presente trabalho, é usado o modelo de transações distribuídas, baseado em *coordenadores* e *participantes* [47]. Nesta abordagem, o *coordenador* refere-se à instância em que a transação foi iniciada e os *participantes* são as instâncias que possuem os dados ou cópias envolvidos na transação iniciada pelo *coordenador*. Ressalta-se, por oportuno, que o *coordenador* é também um *participante*, no caso em que sua instância contenha dados envolvidos na transação. O DTX utiliza a abordagem síncrona para a execução das transações, servindo-se da técnica de bloqueios. Com a abordagem síncrona, é possível garantir o ordenamento das mensagens de comunicação entre sítios, e também garantir acesso exclusivo a recursos compartilhados. No DTX, cada instância faz a gerência de bloqueios em seus dados locais. Utilizando-se, em seu controle de concorrência, técnicas

de bloqueios, poderá eventualmente ocorrer *deadlock*; então, o DTX implementa a política de detecção de *deadlock*, servindo-se da técnica de união de grafos de espera das instâncias do DTX. Na ocorrência de *deadlock*, assim como no protocolo XDGL original, a transação mais recente envolvida no ciclo é cancelada.

O DTX implementa a seqüência clássica de execução de transações, na qual clientes iniciam uma transação, enviando uma requisição *begin* para, em seguida, fazer requisições de escrita e/ou leitura [39]. A transação é finalizada por uma requisição *commit* ou *abort*. Se a requisição *commit* for enviada e executada com sucesso, as atualizações persistem no meio de armazenamento. Caso contrário, se for enviada uma requisição *abort*, a transação é cancelada e suas modificações são desfeitas. Na utilização de um protocolo para controle de concorrência, que possui o nível de isolamento *read-committed*, as transações concorrentes não vêem as alterações pendentes umas das outras.

3.2 ARQUITETURA

O DTX é dividido em alguns componentes de *software* que se comunicam entre si, implementando o controle de concorrência distribuído para dados XML. Uma visão geral da arquitetura do DTX pode ser observada na Figura 3.1.

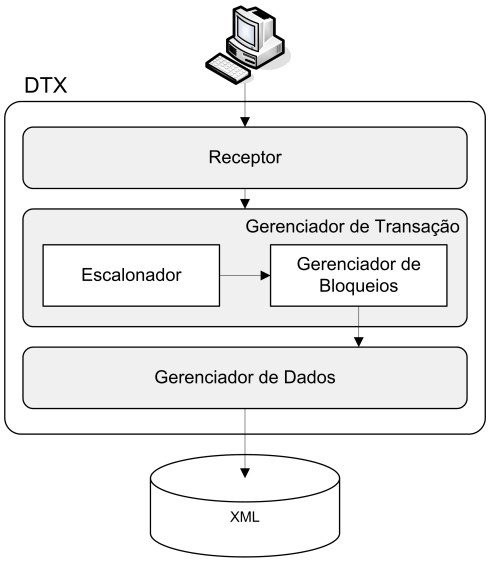


Figura 3.1 Arquitetura do DTX

O receptor é o componente responsável por receber as requisições dos clientes. Ele fornece uma interface simples para processar transações, recuperar seus resultados e

encaminhá-los para o cliente. Outra função do receptor é receber, tratar e encaminhar requisições de outros escalonadores ao escalonador do DTX para realizar uma tarefa distribuída ou de sincronização.

O gerenciador de transação é o macro-componente responsável pela execução das transações; ele é composto de duas partes: o escalonador tem por função escalonar a execução entre operações de transações, utilizando as regras do protocolo para controle de concorrência, detectar/tratar *deadlock*, e executar as operações por meio do gerenciador de nloqueios. O gerenciador de bloqueios contém a estrutura de representação de dados e de bloqueios (i.e., *DataGuide*), utilizada para percorrer de forma otimizada os dados XML; contém ainda as regras para concessão de bloqueios e as operações de manipulação dos dados XML.

O gerenciador de dados é o componente utilizado pelo DTX para interagir com a estrutura de armazenamento dos dados XML. Ele é responsável pela recuperação dos dados XML da estrutura de armazenamento, conversão para uma estrutura própria de representação e fornecimento de meios de atualizar estes dados na estrutura de armazenamento.

3.3 ESPECIFICAÇÃO

Em cada sítio (i.e., nó do sistema) é acoplada uma instância do DTX entre as aplicações e o meio de armazenamento dos dados XML. Essas instâncias se comunicam entre si para executar uma transação distribuída. Para submeter uma transação ao DTX, a aplicação deve realizar uma conexão com a instância DTX. Após isso, a aplicação está apta a enviar uma transação e aguardar seu retorno da mesma com os devidos resultados. Ao término do uso do DTX pela aplicação, esta última deve fechar a conexão estabelecida.

O componente receptor do DTX recebe as transações enviadas pelos clientes. Ao chegar uma transação, o receptor repassa-a para o gerenciador de transação com a finalidade de monitorar sua execução. O gerenciador de transação, por sua vez, repassa a transação para o escalonador, a fim de que ela execute em conjunto com outras transações concorrentes. O escalonador tem por função decidir qual transação irá executar e obter bloqueios necessários para a execução com o gerenciador de bloqueios do sítio corrente. O componente escalonador do sítio, em que a transação foi iniciada, é chamado de *coordenador*. Se a transação contiver alguma operação a ser executada em outros sítios, o

coordenador enviará esta operação para os escalonadores dos sítios de destino; aguarda suas execuções e prossegue para a próxima operação. O escalonador que executa uma operação a mando de outro escalonador é chamado de *participante*. O *coordenador* também tem a responsabilidade de, periodicamente, verificar e tratar *deadlock* distribuído, e consolidar ou cancelar uma transação distribuída. O escalonador, ao conseguir os bloqueios necessários para uma operação, a realiza, interagindo com o gerenciador de bloqueios, que, por sua vez, atualiza o gerenciador de dados.

O critério de *serializabilidade* global é obtido pela utilização de técnicas de bloqueios nos sítios envolvidos em transações distribuídas, uma vez que o protocolo para controle de concorrência, utilizado no DTX, garante este critério em sítio único. Em [48] há uma demonstração de prova do critério de *serializabilidade* transacional global, onde não há um coordenador central para ambientes de grades computacionais. O método de demonstração pode ser aplicado ao DTX também, por possuir características semelhantes, ou seja, não ter um escalonador central.

Para resolver o problema de consolidação das transações, foi adotada uma regra de consolidação. Uma transação só pode consolidar se não depender de nenhuma outra transação ativa. Na solução aqui apresentada, só é permitida a consolidação de transações que efetivaram todas suas operações; para executar uma operação, a transação deve obter os bloqueios necessários em todos os sítios-alvos desta operação. Caso em algum sítio a operação não obtiver todos os bloqueios, a transação, que contiver essa operação, entrará em modo de espera. Com isso, será garantido que uma transação só efetuará a consolidação se não houver dependência de alguma outra transação ativa.

Em [48] os autores descrevem que, para garantir a *serializabilidade*, os escalonadores devem ter conhecimentos sobre o conflito entre transações. No DTX, isso é feito na tentativa de aplicação de bloqueios em operações de transações. Caso em algum sítio não seja possível obter o bloqueio, o sistema é informado de que houve um conflito de bloqueios entre transações. Identificado o fato de que a transação entrou em conflito ao tentar adquirir um bloqueio, então, esta transação é posta no estado de espera.

Para garantir o isolamento e a consistência, o *coordenador* deve obter os bloqueios necessários por parte de cada operação e executá-las em todos os sítios *participantes*. Caso alguma operação necessite ser executada em outros sítios, e em algum destes não obtiver os bloqueios necessários, a transação será posta no modo de espera, e nos sítios, em que a operação conseguiu ser executada, suas ações serão desfeitas. Com isso, será garantido

que uma operação só executará em sua totalidade, ou seja, ela terá que ser realizada em todos os sítios *participantes*. Ao final de uma transação, seja por sucesso ou falha, o *coordenador* deverá, respectivamente, consolidar ou cancelar as operações realizadas em todos os sítios envolvidos na transação distribuída. Caso em algum sítio não seja possível consolidar, a transação será cancelada. Se, no cancelamento de uma transação, não for possível realizar este procedimento em algum sítio, a transação falha.

Com relação à garantia do término de uma transação, o DTX utiliza um processo que periodicamente percorre os escalonadores de todos os sítios, para recuperar seus grafos de espera e verificar se, na união, ocorre um ciclo. Seguindo a regra adotada pelo DTX, a transação mais recente envolvida no ciclo é cancelada. Quando uma transação se consolida, as transações que entraram em modo de espera, aguardando por bloqueios, entram novamente em execução. Com isso, haverá sempre a garantia de que uma transação consolida, cancela ou falha.

A Figura 3.2 ilustra um exemplo de um sistema utilizando o DTX. Este sistema é composto de três sítios: s_1, s_2 e s_3. O sítio s_1 contém uma instância do DTX que atende às requisições do cliente c_1. A instância do DTX do sítio s_1 manipula dados XML oriundos de um SGBD. O sítio s_2 possui uma instância do DTX que trata as requisições do cliente c_2. O módulo DTX do sítio s_2 gerencia dados XML persistidos em um sistema de arquivos. O sítio s_3 contém dois clientes: c_3 e c_4. A instância utilizada pelos clientes c_3 e c_4, gerencia dados XML de um SGBD.

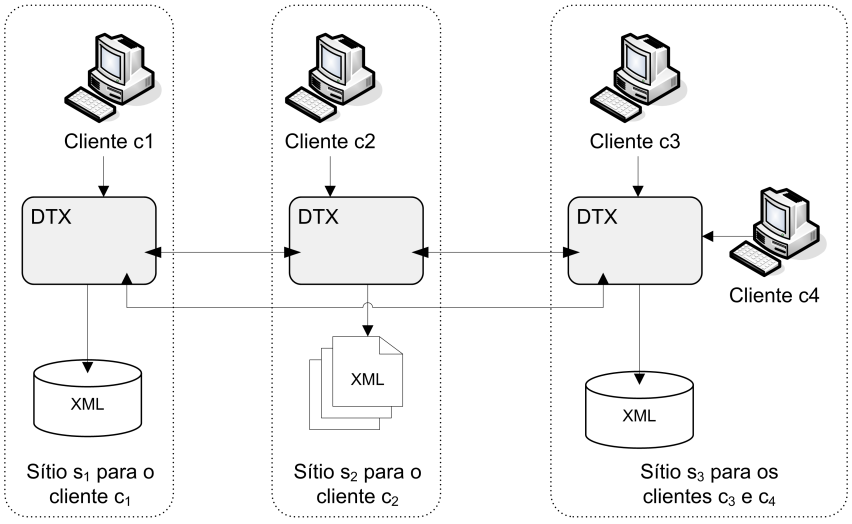


Figura 3.2 Exemplo de configuração do DTX

A Figura 3.2 mostra a comunicação entre as instâncias DTX dos sítios; a inte-

ração é necessária para execução das transações distribuídas quando dados, requeridos pela transação, estiverem localizados em outros sítios. Outras aplicabilidades da comunicação entre instâncias do DTX são: as consolidações, os cancelamentos de transações distribuídas e as detecções/tratamentos de *deadlock* distribuído.

3.4 CENÁRIO

Ambiente

Para demonstrar o funcionamento do DTX, foi ilustrado um cenário de execução. Neste exemplo, existem dois clientes: c_1 e c_2. Estes estão alocados, respectivamente, nos sítios s_1 e s_2. O meio de armazenamento do sítio s_1 é um SGBD XML habilitado; já no sítio s_2, os dados estão persistidos em um SGBDXN.

Para simplificar o entendimento, demonstra-se esse cenário com dois pequenos fragmentos de documentos XML. O primeiro, d_1, contém informações a respeito de clientes de alguma instituição de vendas. O outro, d_2, armazena informações sobre produtos vendidos nesta loja. Os esquemas no formato DTD dos documentos podem ser visualizados na Figura 3.3.

pessoas.dtd

```
<!DOCTYPE pessoas
 <!ELEMENT pessoas (pessoa) >
 <!ELEMENT pessoa (id, nome)>
 <!ELEMENT id (#PCDATA)>
 <!ELEMENT nome (#PCDATA)>
]>
```

produtos.dtd

```
<!DOCTYPE produtos
 <!ELEMENT produtos (produto)>
 <!ELEMENT produto (id, descrição, preço)>
 <!ELEMENT id (#PCDATA)>
 <!ELEMENT descrição (#PCDATA)>
 <!ELEMENT preço (#PCDATA)>
]>
```

Figura 3.3 Esquemas dos documentos

Para a execução do cenário, definem-se três transações: t_1, t_2 e t_3. O cliente c_1 submete a transação t_1, e o cliente c_2 envia as transações t_2 e t_3. A transação t_1 contém duas operações: t_1op_1 e t_1op_2. A t_1op_1 é uma consulta do cliente com identificador de número `4`. O identificador está relacionado aos atributos *id* contidos nas estruturas XML de pessoas e produtos; é uma chave que identifica unicamente cada registro de clientes e produtos. Já a t_1op_2 é uma inserção de um produto, denominado `Mouse`, de preço `10.30` e identificador de número `13`.

A transação t_2 contém duas operações: t_2op_1 e t_2op_2. A primeira é uma consulta

Transação **t1**	Transação **t2**	Transação **t3**
t1op1	***t2op1***	***t3op1***
/pessoas/pessoa[@id=4]	/produtos	/produtos/produto[@id=14]
t1op2	***t2op2***	***t3op2***
inserir <produto id="13">	**inserir** <pessoa id="22">	**inserir** <produto id="32">
<descrição>Mouse</descrição>	<nome>Patrícia</nome>	<descrição>Keyboard</descrição>
<preço>10.30</preço>	</pessoa>	<preço>9.90</preço>
</produto> **em** /produtos	**em** /pessoas	</produto> **em** /produtos

Figura 3.4 Descrição das transações

que recupera todos os produtos da loja. A outra contém uma inserção de uma cliente, chamada `Patrícia`, com identificador `22`. Já a transação t_3, também possui duas operações: t_3op_1 e t_3op_2. A operação t_3op_1 representa uma consulta do registro do produto que possui identificador `14`. A outra, t_3op_2, é uma inserção de um produto denominado `Keyboard`, de preço `9.90` e identificador `32`. Melhor visualização do conteúdo de todas essas transações pode ser obtida na Figura 3.4.

O sítio s_1 gerencia uma cópia do documento d_1; já o sítio s_2 contém uma cópia de todos os documentos: d_1 e d_2. A Figura 3.5 ilustra a arquitetura, alocação dos clientes e documentos nos sítios que compõem este exemplo.

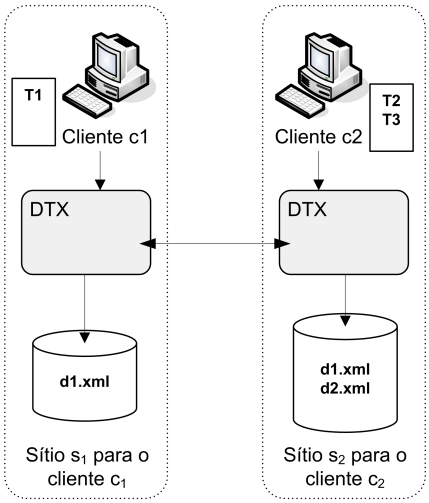


Figura 3.5 Exemplo do cenário

Execução

Suponha-se que, ao iniciar este cenário, os clientes c_1 e c_2 submetam as transações t_1 e t_2 respectivamente. Considere-se que, para este caso, as submissões das transações aconteçam concorrentemente. Cada instância DTX dos sítios locais recebe as transações

pelos seus componentes receptor que, por sua vez, encaminha essas transações para os respectivos gerenciador de transação. O gerenciador de transação envia as transações para o escalonador, que armazena as transações em um fila, para que estas possam ser executadas concorrentemente com outras pendentes.

Suponha-se ainda o caso em que o escalonador do sítio s_1 inicie a execução da t_1op_1. Para executar esta operação, o escalonador deve obter os bloqueios necessários nos dois sítios, pois o documento d_1 está presente neles. Supondo-se que não existam outras transações nas instâncias do DTX, os bloqueios serão obtidos utilizando-se as regras do protocolo XDGL e a operação é processada. Neste instante, a primeira operação da transação t_2 inicia sua execução no sítio s_2. O documento d_2 está presente apenas no sítio s_2; como não existem transações executando sobre o documento d_2, então a operação t_2op_1 adquire os bloqueios, e é realizada. O momento do cenário de execução é ilustrado na Figura 3.6.

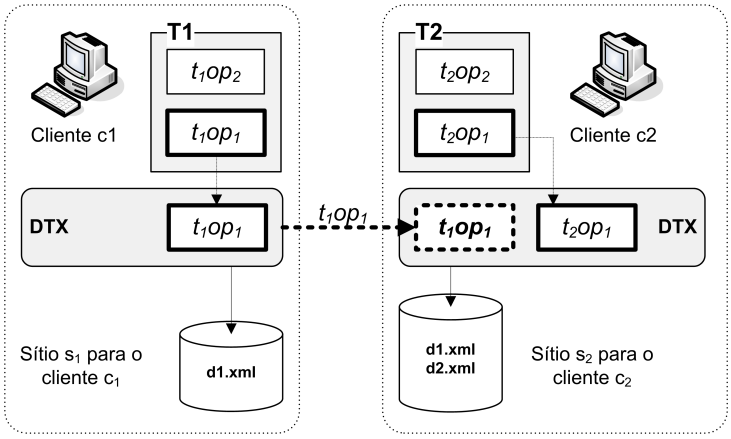


Figura 3.6 Momento do cenário de execução

Já os estados das estruturas de representação dos bloqueios e dados podem ser visualizados na Figura 3.7.

O escalonador do sítio s_1 inicia a execução da operação t_1op_2. Esta operação manipula o documento d_2 e é enviada ao sítio s_2. Não é possível obter bloqueios de inserção para t_1op_2, pois a transação t_2 mantém bloqueios de consulta em d_2. Neste momento, a transação t_1 é posta em estado de espera. O escalonador de s_2 decide, então, executar a operação t_2op_2. O documento d_1, alvo da próxima operação, está presente nos dois sítios. Portanto, é necessário enviar a operação e obter os bloqueios necessários nestes sítios. Não é possível obter os bloqueios de inserção para t_2op_2, pois a transação t_1

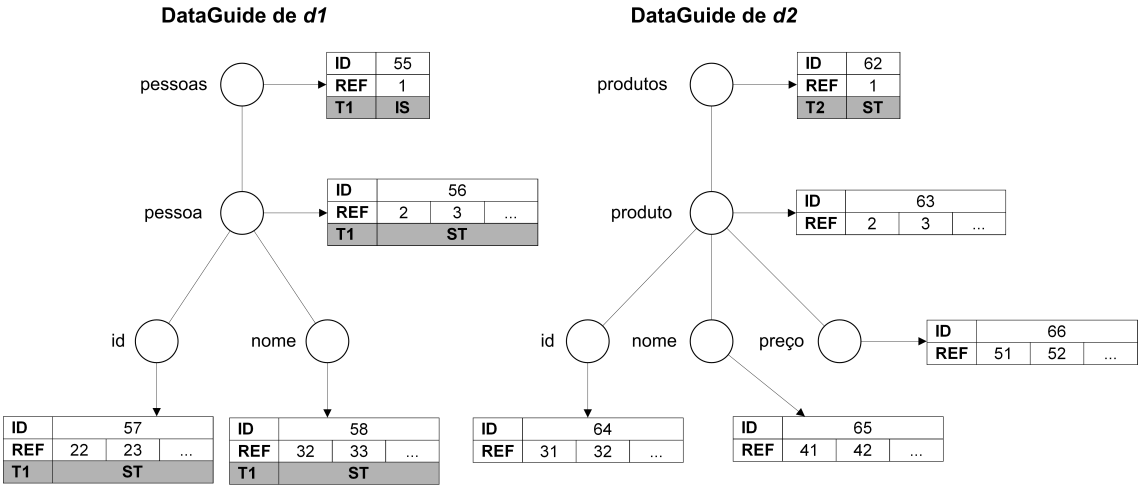


Figura 3.7 *DataGuides* dos documentos

mantém bloqueios de consulta sobre o documento d_1. Com isso, a transação t_2 também entra em modo de espera. É ilustrado na Figura 3.8 o momento de execução do cenário.

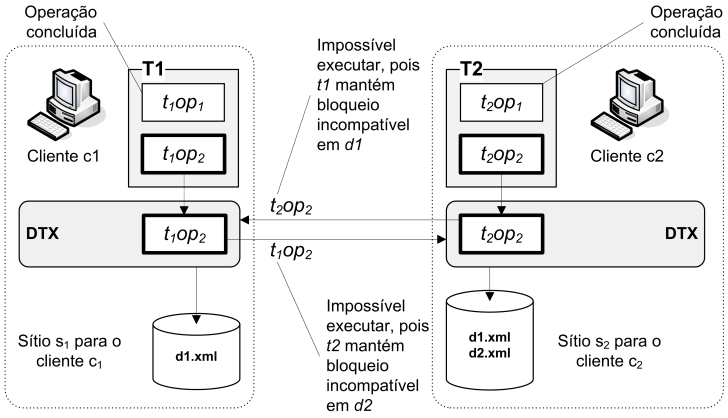


Figura 3.8 Momento do cenário de execução

A Figura 3.9 mostra a situação de incompatibilidade entre bloqueios de transações distintas nos *DataGuides* dos documentos.

Neste instante, visualiza-se uma situação de *deadlock* distribuído. O DTX possui um processo no escalonador, que periodicamente recupera os grafos de espera de todos os sítios e verifica a ocorrência de *deadlock*. Suponha-se que o escalonador do sítio s_1 inicie o processo de verificação de *deadlock* e encontre as transações-alvos do ciclo. Pelas regras do protocolo, a transação mais recente deve abortar; então, a transação t_2 é cancelada, suas modificações desfeitas e seus bloqueios liberados.

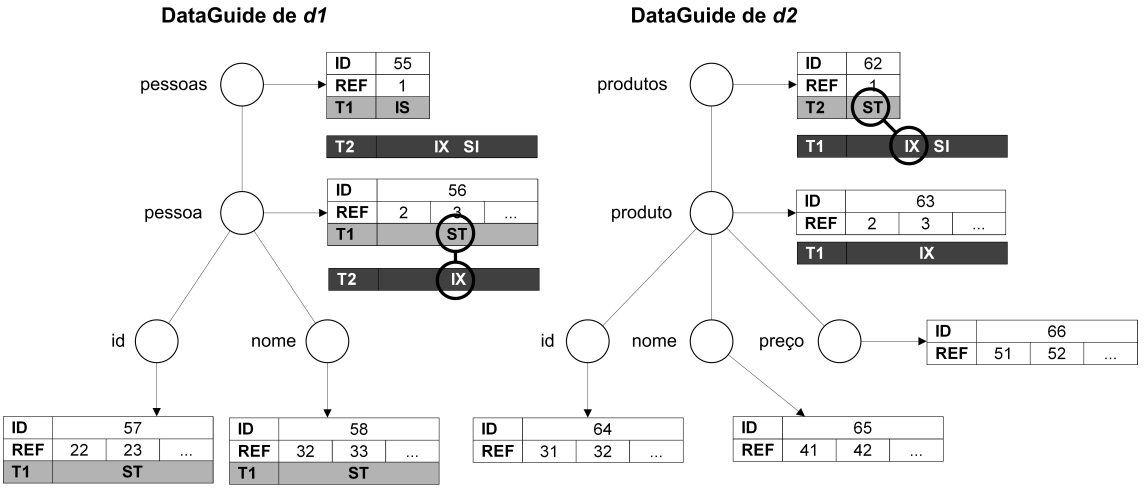


Figura 3.9 Incompatibilidade entre bloqueios

Agora, t_1 pode entrar em execução. O escalonador de s_1 envia novamente a operação t_1op_2 para executar no sítio s_2. Como agora não existem bloqueios sobre o documento d_2, esta operação adquire os bloqueios necessários para a inserção e processa. A transação t_1 não possui mais operações a serem executadas; então, ela inicia o processo de consolidação. Na consolidação, as modificações efetuadas pela transação persistem e seus bloqueios são liberados.

Fica a cargo da aplicação cliente c_2 decidir se coloca novamente a transação t_2 para uma tentativa de execução. Suponha-se o caso em que ele despreza a transação t_2 e decide executar a transação t_3. Como neste exemplo não há transações concorrentes, muito menos bloqueios sobre algum documento, a transação t_3 obtém os bloqueios necessários em todos os sítios e realiza seu processamento.

3.5 ALGORITMOS

Esta seção descreve os principais algoritmos do DTX. O Algoritmo 1 é o procedimento executado pelo escalonador *coordenador* para o processamento das transações. O processo é repetitivo e circular, analisa e recupera a próxima transação disponível na lista de transações (linha 3). As transações disponíveis são aquelas que não se encontram no estado de espera. Após recuperar uma transação, é escolhida a primeira operação que ainda não foi executada (linha 4) para, em seguida, verificar onde esta operação será realizada (linha 5). Se a operação for executada somente no sítio do *coordenador*, a operação será

efetuada no gerenciador de bloqueios local (linhas 6 a 9); caso a operação seja executada em algum outro sítio, esta será enviada e executada em todos os *participantes* que contêm o dado envolvido nesta operação (linha 12 a 22). Se a operação for executada somente no *coordenador* e não obtiver os bloqueios necessários, a transação entra no estado de espera (linha 9). No caso de uma operação que será executada em algum outro sítio (linha 11), esta será enviada a todos os *participantes* (linhas 11 e 12), inclusive para o *coordenador*, caso ele contenha o dado envolvido na operação. A operação enviada para ser executada em outro sítio é chamada de operação remota. É válido ressaltar que o *coordenador* aguarda a execução de todos os sítios para onde a operação foi enviada (linha 14). Caso a operação não adquira bloqueios em algum dos sítios *participantes* (linha 15), o procedimento da linha 16 desfaz as ações em todos os sítios em que a operação executou; após isto, a transação entra no estado de espera (linha 17). Se a operação falhar em algum dos sítios *participantes* ou esta gerar um *deadlock* (linha 19), a transação é cancelada (linha 20). Quando a transação analisada não contiver mais operações a serem executadas (linha 24), esta transação é consolidada (linha 25). Quando uma transação é cancelada, todas as operações são desfeitas em todos os sítios *participantes* e no *coordenador*, e todos os bloqueios sobre os dados envolvidos são liberados. Ao consolidar uma transação, as atualizações são efetivadas e são liberados todos os bloqueios sobre os dados envolvidos na transação.

O Algoritmo 2 descreve o comportamento dos escalonadores ao executar operações remotas nos sítios *participantes*. Ressalta-se que este procedimento também é comum ao *coordenador*. As operações remotas são aquelas que o *coordenador* envia para serem executadas em outros sítios. Todas essas operações são armazenadas em uma lista e recuperadas em modo seqüencial (linha 3). Se houver alguma operação remota a ser executada (linha 4), os bloqueios necessários são adquiridos e a operação é processada no gerenciador de bloqueios do sítio participante (linha 5). Caso a operação não adquira bloqueios necessários, ela é marcada de modo a identificar esta ação pelo *coordenador* (linha 8). Se por algum motivo a operação falhar (linha 10), ela é marcada com o indicador de *abort* (linha 11). Ao término da execução da operação remota, seja por sucesso ou fracasso, o estado da operação é enviado ao *coordenador* (linha 13). Após processar as operações remotas, os *participantes* executam os procedimentos que tratam as mensagens remotas de consolidação e cancelamento das transações distribuídas (linhas 14 e 15).

O Algoritmo 3 descreve o procedimento de aquisição de bloqueios e realização das

Algoritmo 1 - Procedimento executado pelo Coordenador

```
1: procedure process_transactions
2:     loop
3:         transaction ← buffer.next_transaction_avaliable();
4:         operation ← transaction.next_operation();
5:         if operation contain only site of coordinator then
6:             if LockManager.process_operation(operation, wait_for_graph) then
7:                 operation.set_executed(true);
8:             else
9:                 transaction.wait();
10:            end if
11:        else
12:            participants ← sites.get_participants(operation.get_sites());
13:            participants.send_operation(operation);
14:            wait_for_reponses();
15:            if operation.not_adquire_locking() then
16:                undo_operation(operation);
17:                transaction.wait();
18:            else
19:                if operation.aborted() or operation.deadlock() then
20:                    abort_transaction(transaction);
21:                end if
22:            end if
23:        end if
24:        if transaction does not have avaliable operation then
25:            commit_transaction(transaction);
26:        end if
27:    end loop
28: end procedure
```

Algoritmo 2 - Procedimento executado pelos Participantes

```
1: procedure process_transactions
2:     loop
3:         remote_operation ← buffer.next_remote_operation_avaliable();
4:         if remote_operation is valid then
5:             if LockManager.process_operation(remote_operation, wait_for_graph) then
6:                 remote_operation.set_executed(true);
7:             else
8:                 remote_operation.set_adquire_locking(false);
9:             end if
10:            if remote_operation.failed() or remote_operation.deadlock() then
11:                remote_operation.set_abort(true);
12:            end if
13:            send_remote_operation_coordinator(remote_operation);
14:            process_commit_messages();
15:            process_abort_messages();
16:        end if
17:    end loop
18: end procedure
```

operações no gerenciador de bloqueios pelo escalonador. Este procedimento recebe como parâmetro a operação a ser executada e o grafo de esperas para o tratamento e detecção de *deadlock*. Este procedimento percorre todos os elementos do *DataGuide*, baseando-se nos nós utilizados na operação (linha 3). A cada elemento percorrido, é verificada a possibilidade de obtenção de bloqueio para o elemento em questão; caso não seja possível, é retornada a transação que mantém o bloqueio sobre o dado requerido (linha 4). Caso o bloqueio seja adquirido com sucesso (linha 5), o *DataGuide* é atualizado (linha 6). Se houver uma transação conflitante, é adicionada uma aresta que liga a transação conflitante com a transação da operação no grafo de esperas (linha 8). Caso a adição de uma aresta no grafo de esperas gerar um ciclo (linha 9), isto é, ocorrer um *deadlock*, a operação é marcada com a ação de ocorrência de *deadlock* para ser tratada pelo escalonador (linha 10). Após a análise da ocorrência de *deadlock*, as modificações feitas pela operação no *DataGuide* e no gerenciador de bloqueios são desfeitas (linha 12). Este procedimento retorna o resultado da operação em caso de sucesso em sua execução (linha 17), e nulo caso não adquira bloqueios e/ou ocorra *deadlock* (linha 14).

Algoritmo 3 - Procedimento executado pelo Gerenciador de Bloqueios ao executar uma operação

```
 1: procedure process_operation
 2:     result ← null;
 3:     while not go through all the DataGuide elements of the operation do
 4:         transaction_conflited ← DataGuide.process_locking(operation, lock_table, current_node);
 5:         if transaction_conflited is empty then
 6:             result ← DataGuide.update(operation);
 7:         else
 8:             wait_for_graph.add_link(operation.get_transaction(), transaction_conflited);
 9:             if wait_for_graph.is_circle() then
10:                 operation.set_deadlock(true);
11:             end if
12:             DataGuide.undo_operation(operation);
13:             return null;
14:         end if
15:     end while
16:     return result;
17: end procedure
```

O Algoritmo 4 descreve o procedimento que processa a consolidação de uma transação distribuída pelo *coordenador*. Este algoritmo é atômico, o escalonador e o gerenciador de bloqueios param suas atividades até o término deste procedimento. Este algoritmo inicia recuperando todos os sítios envolvidos na transação (linha 2); para cada sítio, é enviada uma mensagem para que este consolide os efeitos realizados pela transação (linha 4). Caso a mensagem enviada ao sítio não seja atendida (linha 5), a transação é

cancelada (linha 6) e o procedimento se encerra (linha 7). Se todos os sítios que foram requisitados efetuarem de maneira correta a consolidação, o escalonador realiza uma comunicação com o gerenciador de bloqueios, para que este persistam as modificações (linha 10) e libere os bloqueios mantidos pela transação (linha 11).

Algoritmo 4 - Procedimento executado pelo *Coordenador* ao consolidar uma transação distribuída

```
 1: procedure commit_transaction
 2:     sites ← transaction.get_sites();
 3:     for site in sites do
 4:         result ← site.send_commit_message(transaction);
 5:         if not result then
 6:             abort_transaction(transaction);
 7:             return false;
 8:         end if
 9:     end for
10:     LockManager.DataManager.persist(transaction);
11:     LockManager.DataGuide.release_lockings(transaction, lock_table);
12:     return true;
13: end procedure
```

O Algoritmo 5 descreve o procedimento que processa a operação de cancelamento de uma transação distribuída pelo *coordenador*. Assim como o algoritmo de consolidação, este algoritmo é atômico; o escalonador e o gerenciador de bloqueios interropem suas atividades até o término deste procedimento. Este algoritmo inicia recuperando todos os sítios envolvidos na transação (linha 2); para cada sítio, é enviada uma mensagem para que este cancele os efeitos realizados pela transação (linha 4). Caso a mensagem enviada ao sítio não seja atendida (linha 5), é remetida a todos os sítios uma mensagem para que todos falhem a transação (linha 6); o *coordenador* também executa o procedimento de falha (linha 9) e o procedimento se encerra (linha 10). Se todos os sítios, que foram requisitados, efetuarem de maneira correta o cancelamento, o escalonador realiza uma comunicação com o gerenciador de bloqueios, para que este desfaça todas as modificações (linha 13) e libere os bloqueios mantidos pela transação (linha 14).

O Algoritmo 6 descreve o procedimento que detecta e trata a ocorrência de *deadlock* distribuído. Este algoritmo é executado periodicamente pelos escalonadores. Primeiramente, é recuperada uma lista de todos os sítios envolvidos no sistema (linha 2). Para cada sítio, é enviada uma mensagem de requisição do seu grafo de espera (linha 4). A cada grafo de espera recebido, o escalonador faz a união com o seu grafo (linha 5). Se, nessa operação de união, o grafo de espera resultante gerar um ciclo (linha 6), a transação mais recente que está envolvida no ciclo é cancelada (linhas 7 e 8) e o procedimento se

Algoritmo 5 - Procedimento executado pelo *Coordenador* ao cancelar uma transação distribuída

```
 1: procedure abort_transaction
 2:     sites ← transaction.get_sites();
 3:     for site in sites do
 4:         result ← site.send_abort_message(transaction);
 5:         if not result then
 6:             for site in sites do
 7:                 result ← site.send_fail_message(transaction);
 8:             end for
 9:             fail_transaction(transaction);
10:             return false;
11:         end if
12:     end for
13:     LockManager.DataManager.undo(transaction);
14:     LockManager.DataGuide.release_lockings(transaction, lock_table);
15:     return true;
16: end procedure
```

encerra, retornando verdadeiro. Se o algoritmo retornar falso, significa que não houve ocorrência de *deadlock* distribuído.

Algoritmo 6 - Procedimento executado periodicamente pelos Escalonadores para detectar e tratar *deadlock* distribuído

```
 1: procedure process_deadlock_detection
 2:     sites ← get_all_sites();
 3:     for site in sites do
 4:         graph ← site.request_wait_for_graph();
 5:         result_graph ← result_graph.union(graph);
 6:         if result_graph.is_circle() then
 7:             transaction ← graph.get_newest_transaction_in_circle();
 8:             abort_transaction(transaction);
 9:             return true;
10:         end if
11:     end for
12:     return false;
13: end procedure
```

3.6 CONCLUSÃO

Neste capítulo, apresentou-se o DTX como um mecanismo de controle de concorrência distribuído para dados XML, cujo objetivo é garantir o isolamento e a consistência em transações distribuídas, com a finalidade de obter melhor desempenho. Inicialmente, foram explorados os pressupostos do DTX e esboçada sua arquitetura. A seguir, foi descrita sua especificação, um cenário de execução e os principais algoritmos.

CAPÍTULO 4

IMPLEMENTAÇÃO E AVALIAÇÃO

Este capítulo descreve os componentes do DTX e apresenta a avaliação realizada, comparando o DTX descrito neste trabalho com uma variação do DTX que utiliza um protocolo baseado em bloqueio em árvores. São explorados os detalhes de implementação e, ao final, traz os resultados obtidos na avaliação, considerando aspectos de desempenho em mecanismos para controle de concorrência.

4.1 ASPECTOS DE IMPLEMENTAÇÃO

Por decisões de projeto, optou-se por desenvolver o DTX servindo-se da linguagem Java [49]. Características como portabilidade, "reusabilidade", processamento distribuído e *multithreading* são propriedades desta linguagem que favorecem um desenvolvimento confortável, confiável e de altos recursos. A portabilidade está relacionada à independência da plataforma de execução, ou seja, não depende de sistema operacional ou *hardware*. Já a reusabilidade está no fato de que, em Java, há diversos componentes, e APIs estão prontas para serem incorporadas à aplicação, aumentando assim a produtividade. Com relação ao processamento distribuído, Java oferece diversos recursos de comunicação entre componentes, tais como: chamada a funcões remotas (Sockets e RMI) e integração com os protocolos conhecidos de internet (TCP/IP, HTTP, Telnet etc). Assim, torna-se mais fácil o desenvolvimento de aplicações derivadas da arquitetura cliente/servidor. Java possui recursos confiáveis para aplicações *multithreading*, servindo-se de primitivas de sincronização baseadas no uso de monitores. Os mecanismos de sincronização, baseados em monitores da linguagem Java, fornecem um ambiente mais fácil e seguro para o desenvolvimento de aplicações multi-tarefas.

No que diz respeito ao sistema de comunicação entre sítios, optou-se pela utilização da comunicação Java Socket. A implementação Socket de Java é o meio de comunicação mais simples e primitivo desta linguagem, o que proporciona melhor desempenho na troca de mensagens. Pelo fato de que o DTX foi escrito na linguagem Java, optou-

se por utilizar Sockets com meio de comunicação, visto que atendem as necessidades e promovem melhor desempenho.

Pelo fato de que o DTX foi desenvolvido na linguagem Java, e a implementação Socket de Java é meio de comunicação mais simples e primitiva desta linguagem, isso proporciona melhor desempenho na troca de mensagens, ao mesmo tempo em que atende as necessidades de comunicação deste trabalho.

A implementação do DTX realizou o desenvolvimento do protocolo XDGL padrão, uma vez que não foi possível obter os fontes deste protocolo. A versão do protocolo XDGL foi implementada seguindo as regras descritas em [46]. Depois, foram inseridas algumas adaptações que proporcionassem o protocolo XDGL funcionar em ambiente distribuído. As adaptações foram descritas no capítulo anterior. Para construir o DTX, alguns componentes de *software* foram criados. A seguir, descrever-se-á, em nível de implementação, cada um destes componentes e suas classes ou serviços associados.

Listener

O Listener é o componente responsável pela interação dos usuários com o DTX. Este componente é composto por uma série de classes para oferecer tal recurso. A Figura 4.1 mostra o diagrama de classes do Listener. A classe `Listener` estende da classe `Thread` do pacote `java.lang`, o método de execução desta `Thread` espera pelas requisições `Socket` de algum cliente, repassando-as para um objeto da classe `ListenerThread` tratar estas requisições individualmente. Ao receber uma requisição de um cliente, o objeto `Listener` cria uma instância da classe `ListenerThread`, passando como parâmetro o objeto `Socket` estabelecido na comunicação entre o cliente e o Listener. Logo em seguinda, o objeto `ListenerThread` é posto em execução.

A classe `ListenerThread` também é uma extensão de `Thread`. Essas interações dos objetos `Listener` e `ListenerThread` fazem com que o DTX atenda requisições concorrentemente entre os clientes, no intuito de aumentar o desempenho e a disponibilidade do serviço. O `ListenerThread` recebe um objeto `DTXTransaction` pelo `Socket` estabelecido na comunicação. O `DTXTransaction` é um objeto criado pelo cliente para representar de forma simplificada uma transação. Este objeto contém todas as operações da transação. Ao receber o objeto `DTXTransaction`, este é convertido para um objeto da classe `Transaction`. A classe `Transaction` é a forma completa de representação de uma transação, contendo métodos e atributos apropriados para o seu gerenciamento.

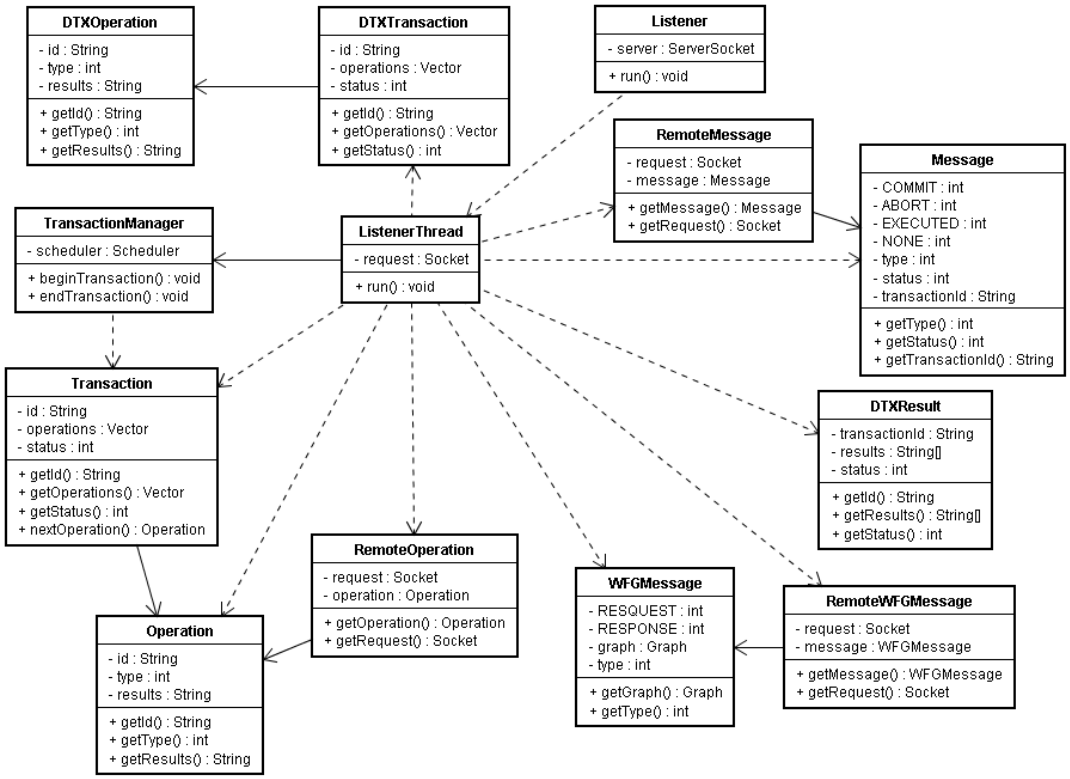


Figura 4.1 Diagrama de classe do Listener

Após a conversão de `DTXTransaction` para `Transaction`, este é enviado para o TransactionManager para ser executado. Ao final da execução, os resultados e o estado do objeto `Transaction` são convertidos para um objeto no formato `DTXResult`. O objeto `DTXResult` é utilizado pelo cliente para visualizar de modo simplificado os resultados e/ou estado da transação. Após a conversão, o `DTXResult` é enviado ao cliente por meio do objeto `Socket`. Ao término da emissão do `DTXResult`, a comunicação é encerrada e o objeto `Socket` é fechado.

O componente Listener não é somente responsável pelo recebimento de requisições dos clientes, mas também requisições de outras instâncias do DTX. Estes outros tipos de requisições podem ser: requisições para execução de operações remotas, requisições para consolidação de transações, cancelamento de transações e cópias do grafo de espera.

O `ListenerThread`, ao receber um objeto do tipo `Operation`, significa que esta operação irá ser executada nesta instância a mando do escalonador de outra instância DTX. Esse objeto é encarado como operação remota; então, ele é convertido para um objeto do tipo `RemoteOperation` e enviado diretamente ao Scheduler. Na conversão do

objeto `Operation` para o `RemoteOperation`, são anexados ao objeto convertido o próprio objeto `Operation` e objeto `Socket` da comunicação com o escalonador requisitante. A importância de manter o objeto `Socket` está na necessidade de retornar os resultados ou estado da operação para o solicitante.

Outros tipos de objetos que o `ListenerThread` pode receber são `Message` e `WFGMessage`. O do tipo `Message` representa troca de mensagens entre escalonadores; estas mensagens podem ser para cancelar ou consolidar uma transação distribuída. Já o objeto do tipo `WFGMessage` representa uma requisição do grafo de espera para a detecção de *deadlock* distribuído. Estes objetos do tipo `Message` e `WFGMessage`, ao chegarem no `ListenerThread`, são convertidos para objetos `RemoteMessage` e `RemoteWFGMessage`, respectivamente, e enviados diretamente ao Scheduler. Assim como feito no objeto `RemoteOperation`, o objeto `Socket` também é mantido nos objetos `RemoteMessage` e `RemoteWFGMessage`, para o mesmo propósito.

Para submeter uma transação ao DTX, o cliente deve criar um objeto do tipo `DTXTransaction`. A classe `DTXTransaction` contém métodos não presentes no diagrama para adicionar diferentes tipos de operação, passando além dos parâmetros da operação, parâmetros para a localização das instâncias do DTX que contêm os dados envolvidos na operação. Como resposta, o cliente recebe um objeto do tipo `DTXResult`, contendo os resultados e o estado da transação que fora enviada ao DTX. Os métodos `getResults` e `getStatus` do objeto `DTXResult` informam ao cliente os resultados e o estado da transação, respectivamente.

TransactionManager

O TransactionManager é o componente que gerencia a execução das transações em cada uma das instâncias do DTX. A Figura 4.2 mostra o diagrama de classes desse componente. Ele recebe os objetos do tipo `Transaction` do Listener. O objeto `TransactionManager`, ao ser criado, inicia o escalonador, ou seja, o Scheduler. A classe `TransactionManager` possui o método `beginTransaction` para iniciar o gerenciamento de uma transação. Este método recebe como parâmetro o objeto `Transaction` e o objeto `ListenerThread`. O objeto `ListenerThread` é necessário para que, ao término da execução da transação, seus resultados sejam enviados ao cliente pelo canal de comunicação; lembrando que cada objeto `ListenerThread` mantém o objeto `Socket` para a comunicação individual com o cliente. Ao ser chamado, o método `beginTransaction` instancia um objeto do

tipo `TransactionManagerThread` para delegar o gerenciamento de uma transação. Este `TransactionManagerThread` inicia a transação, enviado-a ao Scheduler e monitora sua execução. Ao término, o `TransactionManagerThread` invoca o método `endTransaction` da classe `TransactionManager`, passando o objeto `Transaction` e o `ListenerThread`, para finalizar a transação. O método `endTransaction` comunica-se com um objeto do tipo `ListenerThread` passado por parâmetro, para que ele envie os resultados e/ou estado da transação para o cliente por meio do seu objeto `Socket` e finalizar a comunicação.

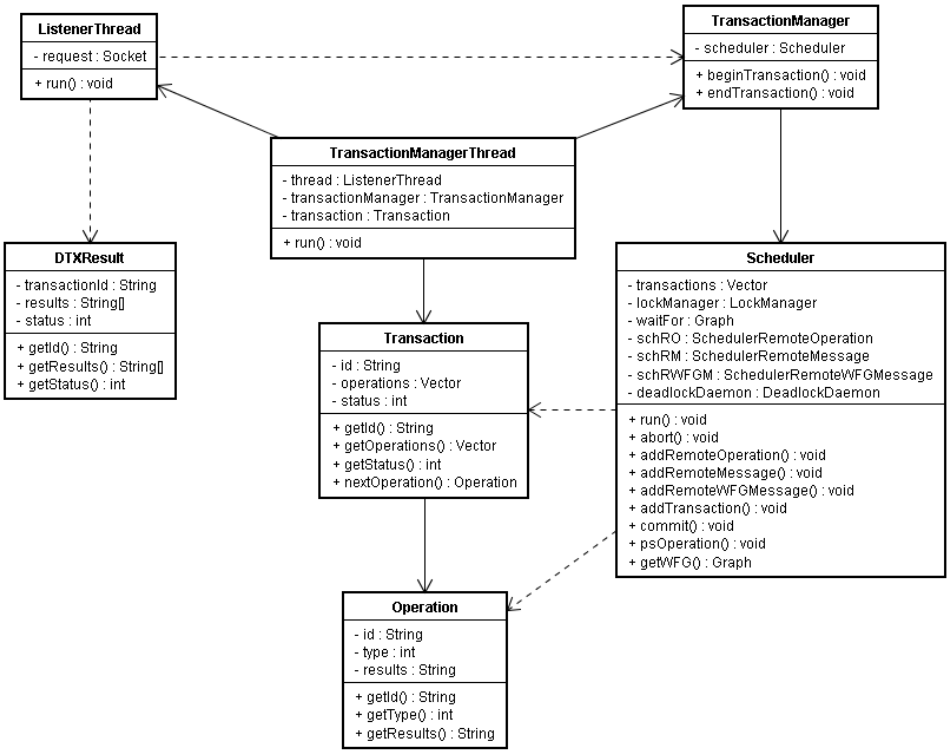


Figura 4.2 Diagrama de classe do TransactionManager

A classe `Transaction` basicamente contém uma lista de operações, um identificador e um atributo que indica o estado da transação. O método `getId` serve para recuperar o identificador da transação, enquanto o método `getOperations` recupera a lista de objetos do tipo `Operation`. O método `nextOperation` recupera a próxima operação a ser executada, ou seja, a próxima operação da seqüência ainda não executada. Outro método é o `getStatus`, que indica o estado da transação. A classe `Operation` é a classe base de todas as operações. Nesta, o método `geType` retorna o tipo de operação. O método `getState` recupera o estado da operação. Os métodos `getHost` e `getPort` da classe `Operation`, não presentes no diagrama, recuperam o endereço e a porta, respectivamente, da instância do DTX em que a operação irá executar.

Scheduler

O Scheduler é a parte do TransactionManager, responsável pelo entrelaçamento entre operações de transações concorrentes e garantir o critério de "serializabilidade". A Figura 4.3 apresenta o diagrama de classe do Scheduler e suas classes associadas. A classe `Scheduler` estende da classe `Thread`. Ao iniciar, o objeto `Scheduler` inicia outras partes do TransactionManager, tais como o LockManager e o detector de *deadlock* distribuído. O objeto `Scheduler` mantém uma lista de transações adicionadas pelo objeto `TransactionManager`. O método `addTransaction` da classe `Scheduler` é utilizado para adicionar uma transação na lista de transações. Como serviços auxiliares, o objeto `Scheduler` inicia outros três objetos do tipo `Thread` que processam as operações remotas, mensagens remotas e requisições dos grafos de espera. Estes objetos são o `SchedulerRemoteOperation`, o `SchedulerRemoteMessage` e o objeto denominado `SchedulerRemoteWFGMessage`, respectivamente.

O objeto `SchedulerRemoteOperation` mantém uma lista de operações remotas, ou seja, uma lista de objetos do tipo `RemoteOperation`, adicionados pelo objeto `Scheduler`. Basicamente, o objeto `SchedulerRemoteOperation` recupera uma operação remota disponível, processa e guarda seus resultados. Similarmente, o objeto do tipo `SchedulerRemoteMessage` recupera uma mensagem remota de sua lista de mensagens remotas, ou seja, uma lista de objeto do tipo `RemoteMessage`, que pode ser uma consolidação ou cancelamento de uma transação distribuída, processa, responde ao solicitante e guarda seus resultados. Já o objeto `SchedulerRemoteWFGMessage` recupera uma requisição de grafos de espera, adiciona na requisição o grafo de espera mantido pelo do objeto `Scheduler` e responde ao solicitante da requisição. Essas operações e mensagens são mantidas e processadas seguindo uma lógica de fila.

A classe `Scheduler` possui métodos para adicionar operações remotas, mensagens remotas e requisições de grafos de espera. O método `addRemoteOperation` é chamado por um objeto do tipo `ListenerThread` ao receber uma requisição de um escalonador de uma outra instância do DTX. Ao ser invocado, o método `addRemoteOperation` adiciona o objeto `RemoteOperation` passado por parâmetro na fila de operações do objeto `SchedulerRemoteOperation`. Já o método denominado `addRemoteMessage` segue a mesma lógica, pois ele é invocado por um objeto do tipo `ListenerThread`, e o objeto `RemoteMessage` passado por parâmetro é adicionado na fila de mensagens remo-

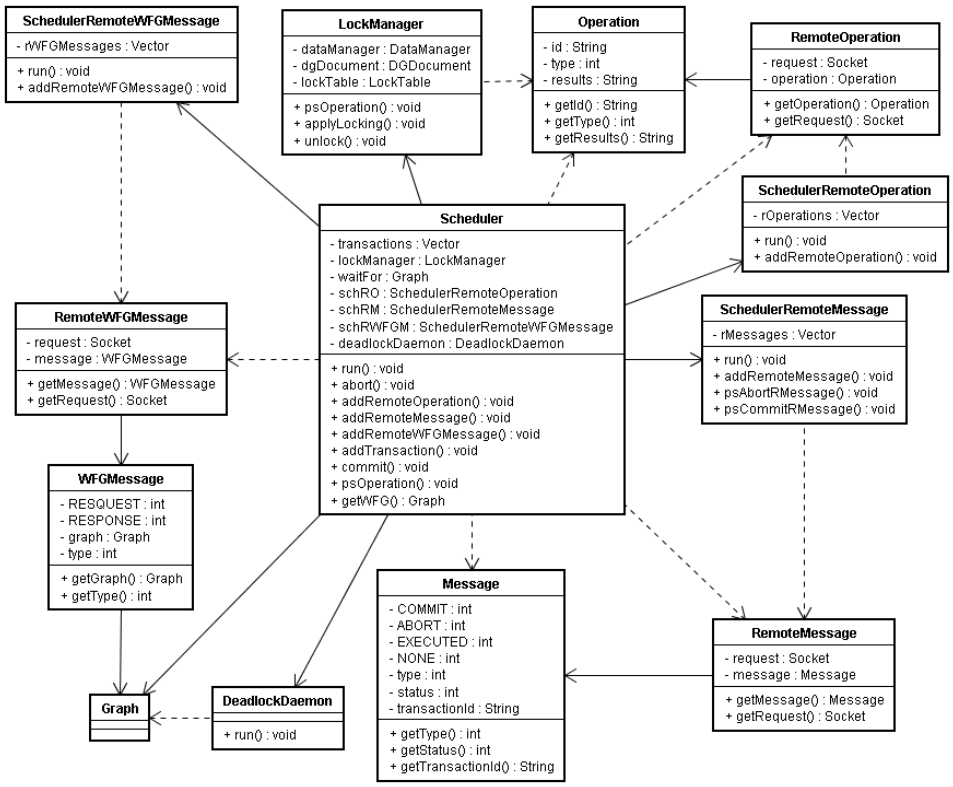


Figura 4.3 Diagrama de classe do Scheduler

tas do objeto `SchedulerRemoteMessage`. O método `addRemoteWFGMessage` do objeto `Scheduler` adiciona uma requisição de grafos de espera na lista de requisições do objeto `SchedulerRemoteWFGMessage`. Quando um escalonador envia uma operação, uma mensagem ou uma requisição de grafos de espera a ser processada por outro escalonador, o `ListenerThread` do escalonador de destino converte os objetos `Operation`, `Message` e `WFGMessage` recebidos para os formatos dos tipos `RemoteOperation`, `RemoteMessage` e `RemoteWFGMessage`, respectivamente, juntamente com os objetos do tipo `Socket` estabelecidos na requisição.

O método de execução do objeto `SchedulerRemoteMessage` consiste em analisar a presença de mensagens remotas e executá-las. Primeiro percorre-se a fila de mensagens remotas e pega-se a que está no começo da fila, analisa-se o tipo de mensagem e faz-se o devido processamento. Para cada tipo de mensagem, é executado um método adequeado. Para isto, existem dois tipos de métodos: `psAbortRMessage` e `psCommitRMessage`. O método `psAbortRMessage` identifica a transação, realiza o cancelamento da transação na instância local e envia uma resposta ao solicitante. Já o método `psCommitRMessage` identifica a transação, efetua a consolidação da transação na instância corrente e emite

uma resposta ao emissor. O método de execução do objeto `SchedulerRemoteWFGMessage` é bastante similiar ao do `SchedulerRemoteMessage`, só que, em vez de processar cancelamentos e consolidações de transação, atende às requisições de grafos de espera. Sua finalidade é gerar uma cópia do grafo de espera mantido pelo objeto `Scheduler` e o enviar ao solicitante da mensagem.

Já o método de execução do objeto `SchedulerRemoteOperation` é aquele em que ele analisa a presença e processa as operações remotas. Para isto, o objeto denominado `SchedulerRemoteOperation` recupera a operação remota que está no começo da fila e que ainda não foi executada. Então, a operação remota é posta em execução. Para isto, o método `psRemoteOperation` é executado, passando o objeto `RemoteOperation` recuperado da fila de operações remotas. Se a operação remota executar com sucesso, ela é removida da fila e seu estado ou resultado é enviado ao solicitante. Caso a operação remota não obtenha algum bloqueio necessário, esta operação é configurada com um indicador que ilustra essa característica e enviada ao *coordenador*. O *coordenador* ao receber a operação, configura a transação solicitante para o estado de espera.

O método de execução do objeto `Scheduler` consiste em processar as transações que estão na lista de transações. A escolha da próxima transação a executar é de maneira seqüencial e circular. A cada transação, ao ser recuperada, uma operação é executada e dá o direito da próxima transação da lista executar uma operação. Ao recuperar uma transação, é resgatada a próxima operação não executada desta transação. Caso a operação seja uma operação remota, a transação entra no modo de espera e um objeto do tipo `Thread` é ativado para o processamento remoto da operação. Caso contrário, o método `psOperation` é executado. A `Thread` responsável pela execução remota da operação envia esta aos escalonadores das instâncias de destino. Caso não seja possível executar esta operação remota em alguma das instâncias, a transação entra no estado de espera ou cancela, dependendo da situação. A transação cancela quando há uma falha, seja por gerar uma possível inconsistência ou por falta de comunicação com alguma instância. A transação entra no modo de espera quando não é possível obter os bloqueios necessários. O método `psOperation` verifica o tipo de operação, e interage com o LockManager para adquirir os bloqueios necessários e executar essa operação. Caso na execução deste método ocorra uma falha, a exceção `AbortTransactionException`, que não está presente no diagrama, é lançada, ou, caso não seja possível obter algum bloqueio necessário, a exceção `LockIncompatibleException`, também não presente no diagrama, é lançada. Estas exceções são tratadas pelo objeto Scheduler. Ao verificar a ocorrência

da exceção `AbortTransactionException`, ele executa o método `abort` para a transação. Já no tratamento da exceção `LockIncompatibleException`, a transação é posta no modo de espera.

Ao término da execução de cada operação, antes de passar para a próxima transação, é verificado se na transação corrente existem mais operações a executar. Caso não haja, o método `commit` é executado para a consolidação distribuída da transação. Este método inicia-se recuperando uma lista de endereços do sítios das instâncias do DTX envolvidas na transação. Para cada sítio, é enviado um objeto `Message`, indicando a transação e sua intenção de consolidar. Para isto, uma conexão `Socket` é estabelecida com a instância de destino e o objeto `Message` é enviado. O objeto `Scheduler` aguarda no método a resposta enviada de cada instância de destino. Caso todas as consolidações sejam efetivadas com sucesso, é verificado se a instância corrente também participa da transação, se participar a consolidação nesta também. Caso todos estes passos ocorram de forma correta, a transação distribuída consolida com sucesso e todos os bloqueios mantidos pelas operações da transação em todas as instâncias são liberados. Caso não seja possível efetuar a consolidação em alguma instância, o método `abort` é iniciado. O método `abort`, assim como o método `commit`, envia uma mensagem de cancelamento da transação a todas as instâncias envolvidas nela. Para isto, ele envia um objeto do tipo `Message`, contendo a transação e um indicador de cancelamento. Na emissão deste objeto, o objeto `Scheduler` estabelece uma comunicação `Socket` com cada instância para enviar este objeto e aguarda uma resposta. Se todos os retornos do objeto Message enviado forem satisfatórios, o método verifica a necessidade de executar o cancelamento na instância local; caso necessário, este é executado. Se tudo ocorrer de forma correta, a transação cancela de maneira global, caso contrário, a transação falha.

Periodicamente, configurado pelo usuário da instância do DTX, o processo de detecção de *deadlock* distribuído é iniciado. Este procedimento inicia uma objeto `Thread` chamado `DeadlockDaemon`; ele contém a localização de todas as instâncias do DTX que compõem o sistema. O objeto `DeadlockDaemon`, servindo-se da localização das instâncias do DTX, envia um objeto do tipo `WFGMessage` para todas essas instâncias para requisitar seus grafo de espera. Esses grafos realizam uma operação de união com o grafo da instância solicitante e os obtidos. Ao verificar a ocorrência de um ciclo no grafo resultante, o objeto `DeadlockDaemon` invoca o método `abort` na transação mais recente envolvida no ciclo, para que esta seja cancelada e o *deadlock* resolvido.

LockManager

O LockManager é a parte do TransactionManager responsável pela execução efetiva das operações, ao mesmo tempo em que adquire os bloqueios necessários a cada uma delas. A Figura 4.4 ilustra o diagrama de classe do LockManager e suas classes associadas. Para gerenciar os bloqueios e navegar de modo mais rápido pelo documento XML original, o objeto `LockManager` mantém um objeto do tipo `DGDocument`, que representa a implementação de um *DataGuide*, e um objeto `LockTable`, que ilustra uma tabela de bloqueios adquiridos pelas transações em execução. A classe `LockManager`, além de conter um objeto `DGDocument` como atributo de classe, possui um ponteiro para o DataManager, responsável pela manipulação efetiva do documento XML original em seu meio de armazenamento.

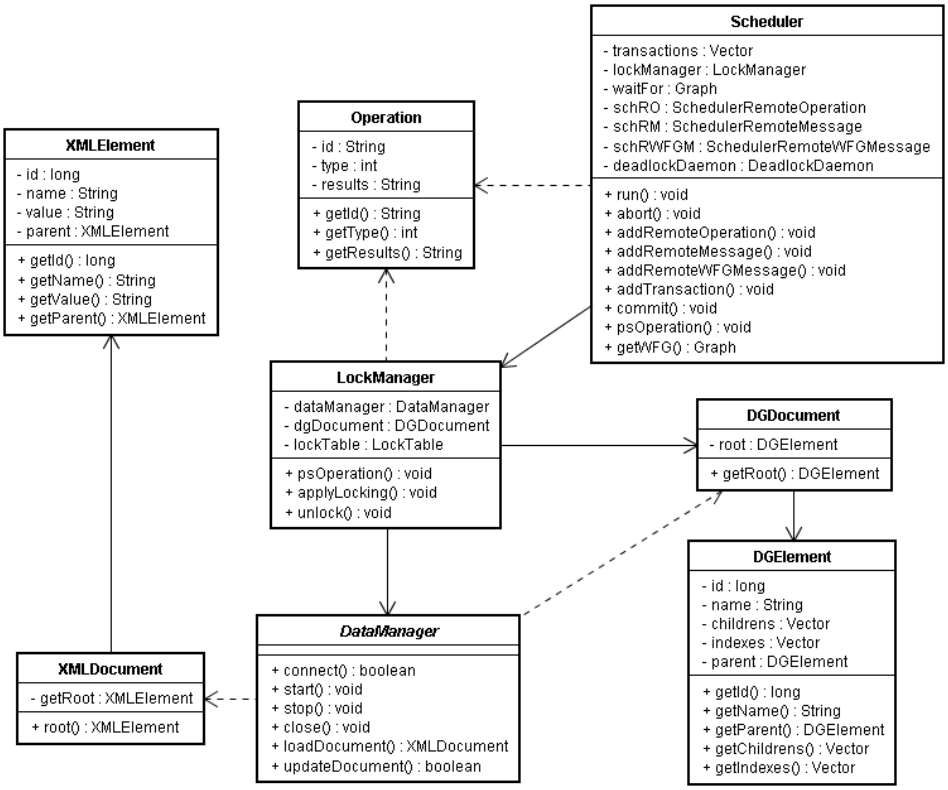


Figura 4.4 Diagrama de classe do LockManager

A classe `DGDocument` possui como parâmetro de criação um `XMLDocument`, que é o formato de representação do documento XML no DTX. Basicamente, o método de criação da classe `DGDocument` transforma um objeto `XMLDocument` em um *DataGuide*. Para isto, existe uma série de métodos privados para realizar essa transformação. Estes

métodos criam um *Strong DataGuide* [4]. Na classe que representa o *DataGuide*, ou seja, a `DGDocument`, possui um ponteiro para o elemento raiz, um contador para gerar identificadores de elementos e os métodos para recuperar estes atributos.

O elemento-raiz e os elementos da estrutura *DataGuide* são representados pela classe `DGElement`. Esta classe não somente representa nós, mas também os atributos. A classe `DGElement` possui como atributos: um identificador do elemento, o nome, uma referência para o elemento ancestral, uma lista de objetos do tipo `DGElement` que representam os filhos deste elemento e uma lista de objetos do tipo `XMLElement`, que são ponteiros para os elementos originais do XML onde o `DGElement` representa. Estas referências aos elementos do tipo `XMLElement` são bastantes úteis para a realização de consultas e atualizações de maneira mais rápida, uma vez que o *DataGuide* é uma estrutura otimizada, ou melhor, sumariada.

O método `psOperation` da classe `LockManager` é o responsável pela execução da operação enquanto adquire os bloqueios necessários. Ele é invocado pelo objeto `Scheduler`, recebe como parâmetro um objeto do tipo `Operation` e retorna o resultado da operação ou lança a exceção `LockIncompatibleException`, caso não obtenha algum bloqueio. Ao iniciar, o método `psOperation` verifica o tipo de operação e executa um procedimento apropriado a cada uma delas. Para qualquer operação, o processo de navegação é iniciado no intuito de encontrar os elementos-alvos da operação. Seguindo as regras do protocolo XDGL, os elementos são percorridos; ao encontrar o elemento-alvo ou elemento-raiz da subárvore da expressão de caminho da operação, este é bloqueado com o bloqueio do tipo apropriado e todos os seus ancestrais com o seu respectivo bloqueio de intenção. Para percorrer os elementos ancestrais aplicando o devido bloqueio, foi criado um método chamado `applyLocking` que utiliza a referência do elemento ancestral do `DGElement` para realizar seu trabalho. Os bloqueios adquiridos pelas transações com a aplicação do método`applyLocking` ficam guardados no objeto `LockTable` da classe `LockManager`.

Para verificar a incompatibilidade entre bloqueios, foi implementada uma classe denominada `XDGLLockCompatibility` que tem essa função. Esta classe possui somente um método estático para verificar a incompatibilidade entre dois tipos de bloqueios. Este método é utilizado pelos métodos da classe `LockManager` que aplicam bloqueios em elementos; caso nestes métodos não seja possível aplicar os devidos bloqueios, a exceção `LockIncompatibleException` é lançada. A classe `LockManager` possui um método denominado `unlock` para desbloquear ou liberar bloqueios mantidos por uma transação.

Este método recebe como parâmetro principal a transação a qual mantém os bloqueios. Na execução do método `unlock`, os bloqueios adquiridos pela transação passada por parâmetro são excluídos do objeto `LockTable`.

DataManager

O DataManager é o componente responsável pela recuperação de dados XML dos sistemas de armazenamento, transformá-los para um formato DTX de representação e provê a persistência deles. A Figura 4.4, além de mostrar o diagrama de classe do LockManager, ilustra também o DataManager. Ele é capaz de interagir com qualquer sistema de armazenamento XML. Para isto, ele foi implementado como uma classe abstrata e os métodos abstratos de conexão, desconexão com os sistemas de armazenamento, carregamento e atualização de documentos devem ser implementados para o sistema de armazenamento utilizado no DTX.

Se o sistema de armazenamento for um SGBD XML habilitado ou um BDNX, o método `connect` deve ser implementado de tal modo que possa estabelecer uma conexão com o sistema de armazenamento. Como parâmetros, podem ser recebidos: o nome do banco de dados, uma URL de conexão, o nome do usuário do banco e a senha de acesso. Existe um método denominado `start` que serve para "inicializar" o banco de dados passado por parâmetro, após ter realizado a conexão com o sistema de armazenamento. Já o método `stop` faz com que o banco de dados "inicializado" com o método `start` seja parado. O método `close` é responsável por fechar a conexão ativa com o sistema de armazenamento.

O método `loadDocument` tem a função de carregar o documento passado por parâmetro e mapeá-lo para o formato DTX de representação. No caso em que o sistema de armazenamento seja um SGBDXN ou um SGBD XML habilitado, este método é utilizado após ter sido realizados uma conexão e a "inicialização" do banco. Se o sistema de armazenamento for em arquivos, não é necessário utilizar os métodos `connect` e `start`, precisando somente utilizar o método `loadDocument`, passando o caminho relativo para encontrar o arquivo do documento XML. O método `updateDocument` é responsável por persistir no sistema de armazenamento a estrutura XML mapeada. Como parâmetros, este método recebe o nome do documento e o objeto `XMLDocument`. Este método segue a mesma lógica do método `loadDocument`, no que diz respeito ao tipo do sistema de armazenamento.

Para facilitar a compreensão das tarefas realizadas pelo DTX, a Figura 4.5 apresenta um diagrama de seqüência simplificado, com os principais componentes do DTX e a interação deles.

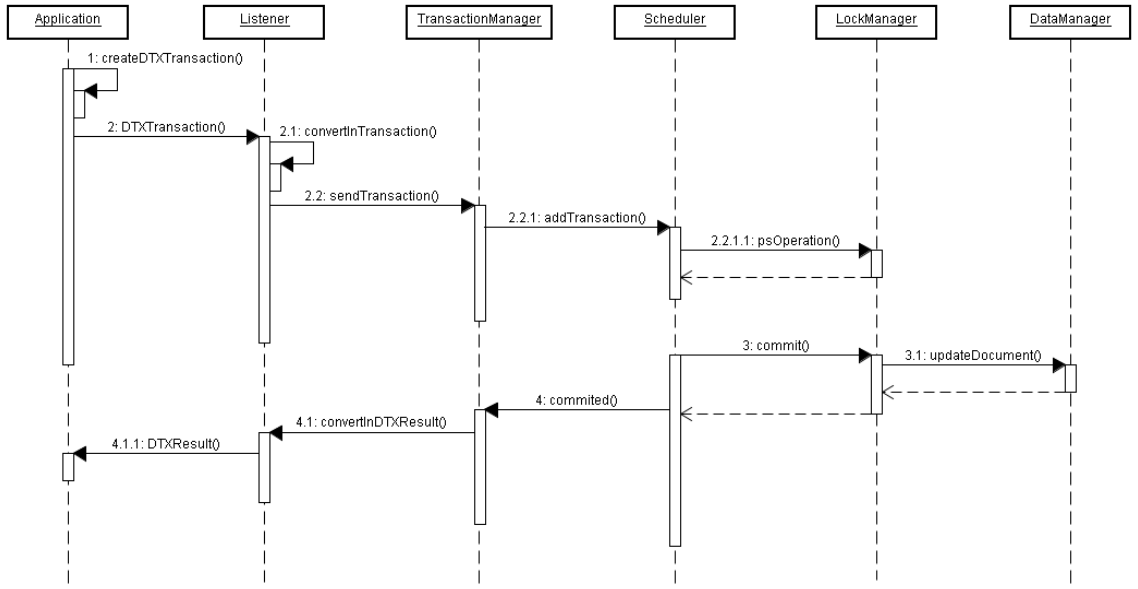


Figura 4.5 Diagrama de seqüência do DTX

O funcionamento do DTX no diagrama de seqüência pode ser exemplificado obedecendo a seqüência de ações e eventos que ocorrem durante uma transação. O processo se inicia quando a aplicação cliente cria o objeto `DTXTransaction`, realiza uma conexão Socket com o DTX através do Listener e envia o objeto criado pelo canal de comunicação. Ao chegar ao objeto `Listener`, o objeto `DTXTransaction` é convertido em um objeto `Transaction` e imediatamente enviado ao objeto `TransactionManager` para realizar a transação.

O objeto `TransactionManager`, por sua vez, inicia o gerenciamento da transação, enviando-a para o objeto `Scheduler` pelo método `addTransaction`. O objeto `Scheduler` inicia sua execução das transações, realizando todas as operações contidas na transação mediante o método `psOperation` contido no objeto `LockManager`. O método `psOperation` executa as operações ao mesmo tempo em que adquire os bloqueios necessários para esta.

Ao completar a execução de todas as operações da transação, o objeto `Scheduler` invoca o método `commit` para a consolidação do objeto `Transaction`. Na consolidação, todos os bloqueios mantidos pela transação são liberados e o meio de persistência é atualizado pelo método `updateDocument` contido no objeto `DataManager`.

Quando a consolidação termina, o objeto TransactionManager realiza o processo de resposta ao cliente. Para isto, ele converte o objeto `Transaction` contendo seu estado e resultados para um objeto `DTXResult`. Após essa conversão, o objeto `DTXResult` é enviado ao cliente por meio do objeto `Listener` que mantém o canal de comunicação estabelecido.

4.2 AVALIAÇÃO

À medida que os sistemas computacionais se tornam mais complexos, a análise de desempenho se torna uma atividade cada vez mais relevante e indispensável. No âmbito desses sistemas, uma ferramenta, para execução de testes-padrão ou *benchmark*, permite realizar um conjunto de testes projetados para comparar o desempenho de um sistema computacional em relação a outros, submetendo-os a uma carga de trabalho semelhante. Uma carga de trabalho corresponde ao conjunto de tarefas e respectivos consumos de recursos, submetidas a um determinado sistema para execução durante um específico intervalo de tempo. Por sua vez, a tarefa é a unidade de execução do sistema computacional. Por exemplo, num cenário de SGBDXN, uma tarefa pode corresponder a uma consulta XPath.

Como expresso anteriormente, o DTX pode ser acoplado a qualquer sistema de armazenamento XML. Durante o desenvolvimento e nos experimentos do DTX, optou-se por utilizar o SGBDXN Sedna [17] pelo fato de ser um sistema *open-source* e com as características de armazenamento e processamento de consultas, necessárias à execução dos experimentos. Além disso, o Sedna possui uma API que fornece acesso simples para aplicações Java, o que facilitou a implementação do método de conexão do DTX com este SGBDXN.

O DTX visa a garantir a consistência e o isolamento global de dados XML em ambientes distribuídos. Por utilizar uma adaptação de um protocolo multigranular e que leva em consideração a estrutura XML, o DTX proporciona melhor o tempo de resposta. Assim, a avaliação no contexto deste trabalho busca analisar o tempo de resposta quando o DTX é utilizado. Além disso, foi verificado o número de *deadlocks* em cada experimento realizado.

Em razão das interfaces e linguagens de acesso aos SGBDXNs, SGBDs habilitados e às tecnologias para manipulação de documentos XML, torna-se complexo desen-

volver experimentos apropriados para verificar o desempenho de sistemas como o DTX [5]. Nesse sentido, vários *benchmarks* para dados XML foram propostos, tais como os apresentados em [50] [51]. Lu et al.[52] ressaltam que nenhum estudo foi encontrado no uso de *benchmarks* que permitam ao usuário identificar o impacto causado pelo tipo de armazenamento no desempenho das consultas. Argumentam que a observação da forma como os dados são armazenados, ou seja, com ou sem a utilização de um esquema, influencia muito na avaliação do sistema.

Para a avaliação do DTX, estendeu-se o *benchmark* XMark [50], adaptando-se suas consultas à linguagem XPath, adicionando-se operações de atualização (Apêndice B), de forma a viabilizar a execução de experimentos. Também se desenvolveu um simulador de clientes, denominado DTXTester, baseado em [8]. O DTXTester visa a realizar experimentos com base no tempo de resposta e número de *deadlock*, criando um conjunto de clientes e submetendo cargas transacionais.

Ambiente de Avaliação

Considere-se um conjunto de sítios S=$\{S_1...S_N\}$. Cada sítio S_i possui um SGBDXN Sedna contendo os documentos XML, adequados a cada experimento, e uma instância do DTX acoplado ao Sedna. Considera-se, ainda, um conjunto de clientes C=$\{C_1...C_M\}$, que contém os aplicativos que dão origem às transações. Para processar uma transação t, um cliente de C conecta-se ao DTX e submete a transação t. Para cada transação t, somente um sítio S_i a inicia (*coordenador*) e, se t possuir operações a serem executadas em outros sítios, o DTX do sítio S_i envia estas para os demais sítios (*participantes*). Na execução dos experimentos, as transações, que foram vítimas de *aborts* durante a ocorrência de *deadlock*, são descartadas.

A concorrência de transações é simulada quando se usam múltiplos clientes. O simulador de clientes, o DTXTester visualizado na Figura 4.6, gera transações de acordo com diversos parâmetros, envia-as para o DTX e coleta os resultados ao final de cada execução. Os parâmetros usados para a geração das transações especificam o número, a porcentagem de transações de atualização, o tamanho da transação e o percentual de operação de atualização por transação deste tipo. O ambiente utilizado para a avaliação foi um *cluster* de 8 PCs conectados através de um *Hub Ethernet*. Cada PC possui um processador de 3.0 GHz, 1 GB de RAM, sistema operacional Windows XP e interface de rede *full-duplex* de 100 Mbit/s. A base de dados foi um documento de 40MB XML,

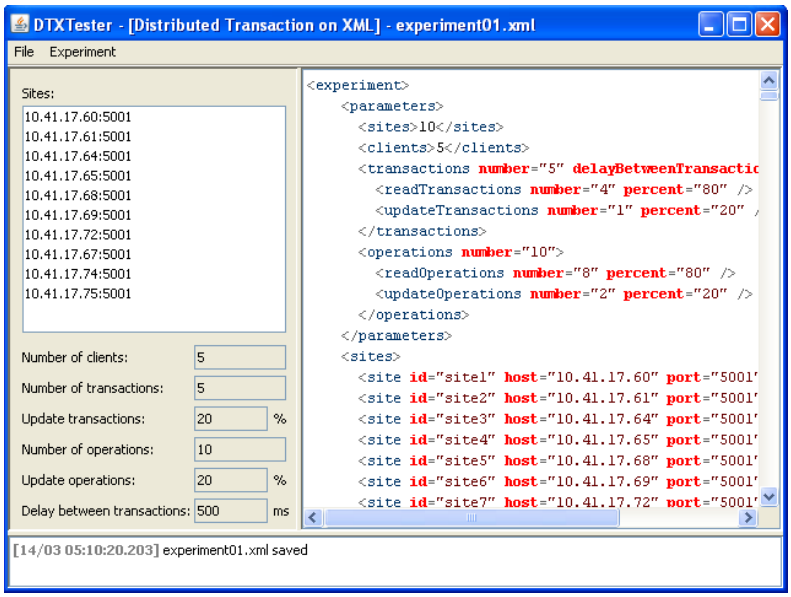


Figura 4.6 DTXTester

gerado pelo XMark.

Para a realização dos experimentos de replicação parcial, a base de dados foi fragmentada de acordo com a abordagem proposta por [53]. Nesta abordagem, os dados são fragmentados, considerando-se a estrutura e o tamanho do documento, de forma que cada fragmento gerado tenha tamanhos similares. Os abordagens de fragmentação, como o PartiX [54], também poderiam ter sido utilizadas.

A abordagem de fragmentação utilizada neste trabalho faz com que todos os sítios tenham volumes de dados semelhantes. Com isso, bases de dados do esquema ilustrado na Figura 4.7 foram geradas, utilizando-se o XMark [50] e dividida em fragmentos de documentos XML. Os fragmentos de dados e suas alocações são descritos na Figura 4.8.

A Figura 4.8 exibe o modo como os dados foram alocados nos respectivos sítios em diferentes cenários. Na Figura 4.8, a primeira coluna representa o número de sítios do cenário, a segunda ilustra o sítio a ser descrito e a terceira seu respectivo conteúdo. Os documentos em negrito mostram que estes estão copiados em outros sítios.

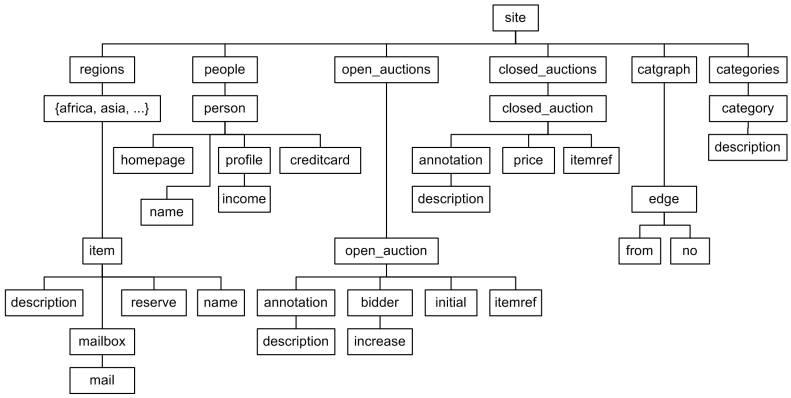


Figura 4.7 Esquema dos dados

Número de sítios	Sítio	Documento(s)
2	Site 01	**africa.xml** asia.xml australia.xml europe.xml samerica.xml namerica.xml
	Site 02	people.xml **africa.xml** closed_auction.xml open_auction.xml categories.xml
4	Site 01	people.xml **africa.xml** categories.xml closed_auction.xml
	Site 02	**africa.xml** asia.xml australia.xml europe.xml samerica.xml
	Site 03	namerica.xml
	Site 04	open_auction.xml

Número de sítios	Sítio	Documento(s)
6	Site 01	people.xml **africa.xml** categories.xml
	Site 02	namerica.xml
	Site 03	open_auction.xml
	Site 04	**africa.xml** asia.xml australia.xml
	Site 05	closed_auction.xml
	Site 06	europe.xml samerica.xml
8	Site 01	people.xml
	Site 02	closed_auction.xml
	Site 03	europe.xml
	Site 04	namerica.xml
	Site 05	open_auction.xml
	Site 06	**africa.xml** samerica.xml
	Site 07	**africa.xml** asia.xml categories.xml
	Site 08	australia.xml

Figura 4.8 Fragmentação e alocação dos dados

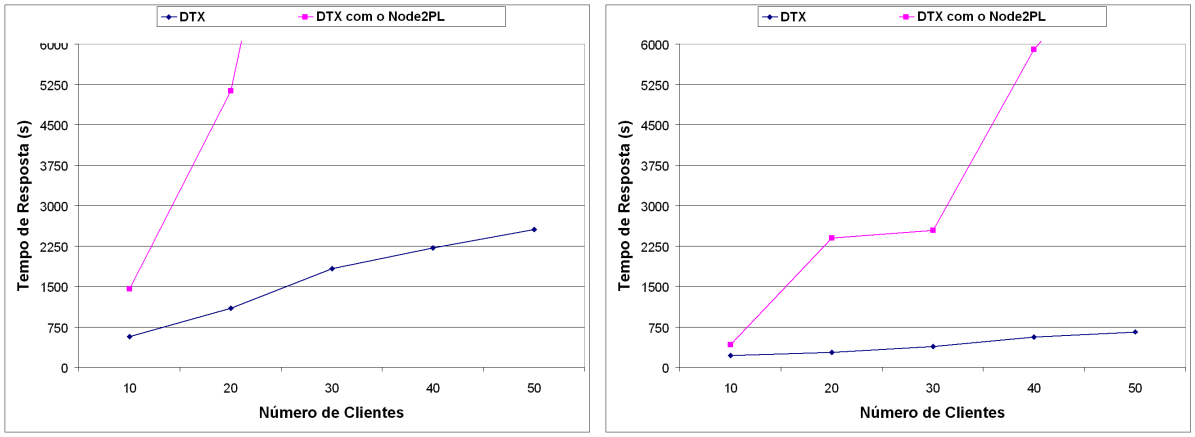
4.3 EXPERIMENTOS

A comparação foi feita entre o DTX utilizando o protocolo Node2PL [35], que possui bloqueio em árvores e o DTX, ou seja, o DTX com as adaptações do protocolo XDGL, ambos acoplados ao SGBDXN Sedna. Pela dificuldade de obter acesso às implementações dos trabalhos relacionados, optou-se por adaptar o DTX e utilizar um protocolo de bloqueio em árvores, uma vez que a maioria dos trabalhos relacionados utiliza protocolos com essa característica, servindo-se da mesma lógica de sincronização entre sítios, persistência, recuperação e linguagem. Portanto, as modificações feitas no DTX foram: a estrutura de representação de bloqueios/documento e as regras de aplicação/liberação de

bloqueios por operação. Durante essas modificações, o DTX mostrou-se bastante flexível à mudança para novos protocolos.

4.3.1 Variação no Número de Clientes

Este experimento verifica o comportamento do DTX quanto à variação do número de clientes, com replicação total e parcial. Neste caso, varia-se o número de clientes de 10 a 50, cada cliente contendo 5 transações de leitura com 5 operações cada qual.



(a) Replicação total (b) Replicação parcial

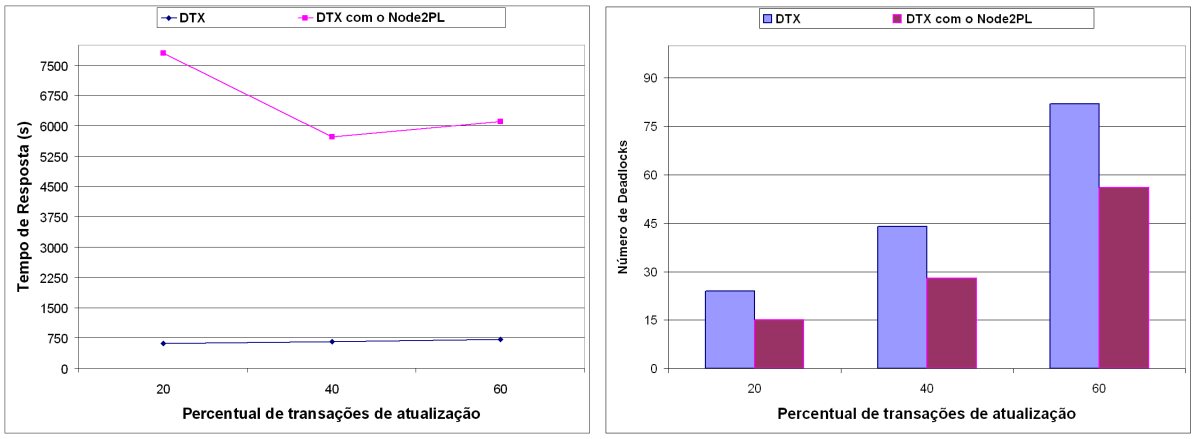
Figura 4.9 Variação no número de clientes

A Figura 4.9 mostra o tempo de resposta resultante deste experimento. Em ambas as abordagens de replicação, o tempo de resposta do DTX se mantém inferior com a adição de mais clientes, e o DTX com o Node2PL apresenta um crescimento do tempo de resposta. A razão para isso é que o protocolo XDGL, utilizado no DTX, possui uma granulosidade bem menor do que a do protocolo de bloqueio em árvores; neste caso, o *overhead* de gerência destes bloqueios é bem inferior.

O tempo de resposta da replicação parcial é inferior ao da replicação total; o motivo disso decorre do fato de que, na replicação total, há um *overhead* de comunicação e sincronização em todos os sítios do sistema, o que atrasa a execução das transações. Com base nestes resultados, optou-se pela realização dos demais experimentos, utilizando-se a abordagem de replicação parcial, já que em ambientes reais os dados XML geralmente estão distribuídos por vários sítios e não se apresentam totalmente replicados.

4.3.2 Variação no Percentual de Atualização

Este experimento mostra o comportamento do DTX contendo uma variação no percentual de transações de atualização. Neste experimento, fixou-se o número de clientes em 50. Cada cliente submete 5 transações com 5 operações. A variação do percentual de transações de atualização ocorre entre 20% a 60%. O percentual de operações de atualização por transação de atualização é de 20%.



(a) Tempo de resposta

(b) Número de *deadlocks*

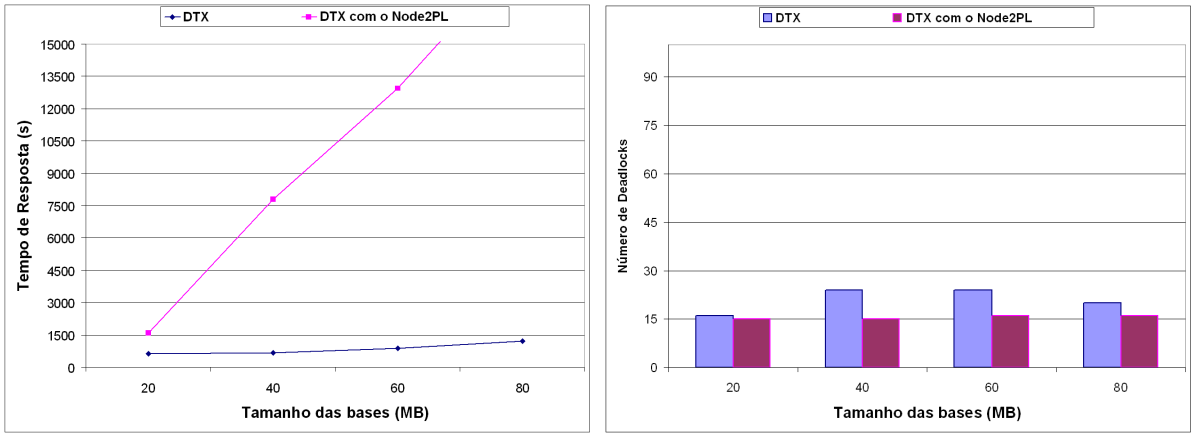
Figura 4.10 Variação no percentual de atualização

Na Figura 4.10 (a), o tempo de resposta do DTX é bem inferior com o crescimento da proporção de atualização, enquanto o DTX com o Node2PL possui um tempo de resposta bem superior. A justificativa, para que o DTX possua um tempo de resposta melhor, está relacionada ao menor *overhead* de gerência de bloqueios, e por usar uma estrutura sumariada dos dados, o que apressa a recuperação e a modificação destes, além de manter uma estrutura de tamanho bem menor do que o documento XML original.

Analisando-se o gráfico (b) desta figura, o número de *deadlocks* obtido pelo DTX foi superior ao do DTX implementado com o protocolo Node2PL. Além disso, com o aumento do crescimento da proporção de atualização, esses números tendem a crescer. Isso é justificado pelo fato de que o DTX consegue mais paralelismo entre transações por utilizar uma granulosidade de bloqueio muito menor do que a da outra abordagem. Isso faz com que um maior número de transações entre em execução concorrentemente no DTX e, eventualmente, pelo fato de requererem dados em comum, entre em *deadlock* em maior quantidade do que o outro protocolo, que é mais restrito e menos concorrente.

4.3.3 Variação no Tamanho das Bases

Este experimento verifica o comportamento do DTX quanto à variação do tamanho da base, servindo-se da replicação parcial. Para isso, fixou-se o número de clientes em 50, cada cliente submetendo 5 transações contendo 5 operações. O percentual de transação de atualização é 20% e o percentual de operações de atualização por transação de atualização é 20%. A variação do tamanho da base foi feita entre 20 a 80MB.



(a) Tempo de resposta

(b) Número de *deadlocks*

Figura 4.11 Variação no tamanho da base

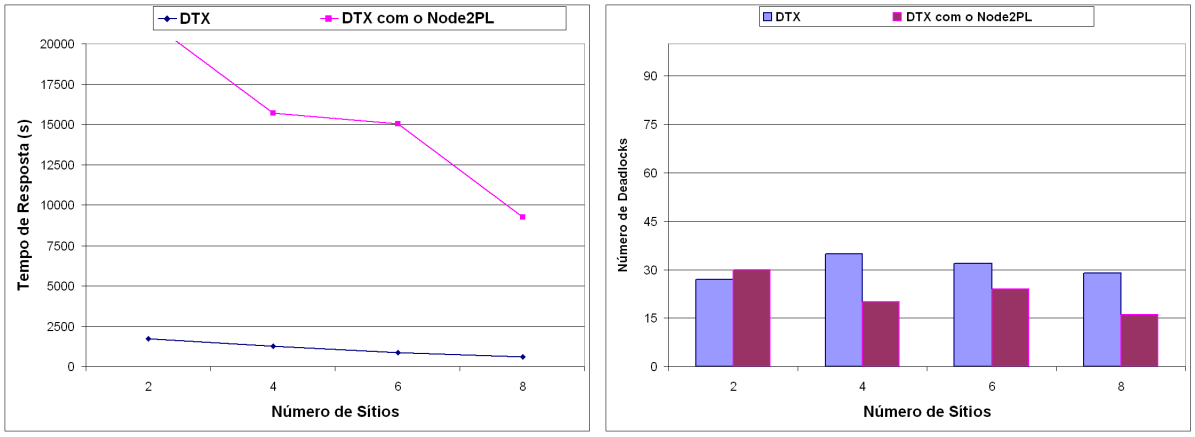
Na Figura 4.11 (a), o tempo de resposta do DTX mostra-se inferior diante do crescimento do tamanho das bases, enquanto o DTX, utilizando o protocolo Node2PL, ilustra um crescimento do tempo de resposta. Um motivo está relacionado ao fato de que o DTX utiliza uma estrutura compacta para representação dos documentos XML. Isso leva a uma simplificação e redução do tamanho da estrutura de mapeamentos dos dados. Apesar do crescimento do volume de dados, a estrutura compacta de mapeamento de dados favorece esses melhores resultados. Outra razão é que, no DTX implementado com o Node2PL, o *overhead* na gerência de bloqueios é bem maior, uma vez que o Node2PL, protocolo utilizado, bloqueia toda a subárvore e ancestrais do nó-alvo. Já o DTX faz uso do protocolo XDGL, que se serve da estrutura compacta *DataGuide* e bloqueia somente os ancestrais do nó-alvo. Portanto, se o documento cresce, o número de bloqueios na abordagem, que utiliza o Node2PL, também aumenta.

Com o crescimento do tamanho da base de dados, o DTX mostrou-se mais deficiente quanto ao número de *deadlock* e isto é visualizado na Figura 4.11 (b). O motivo

disso decorre do fato de que, quando o tamanho da base de dados cresce, o DTX implementado com o Node2PL é mais lento por possuir maior *overhead* de bloqueios, o que favorece maior de tempo de resposta na execução das transações. O maior tempo de resposta diminui o grau de concorrência do protocolo Node2PL, o que ocasiona menor número de conflitos, ou seja, *deadlock*.

4.3.4 Variação no Número de Sítios

Este experimento verifica o comportamento do DTX quanto à variação do número de sítios, servindo-se da replicação parcial. Para isso, fixou-se o número de clientes em 50, cada cliente submetendo 5 transações contendo 5 operações. O percentual de transação de atualização é 20% e o percentual de operações de atualização por transação de atualização é 20%. A base de 40MB foi fragmentada, alocada e carregada no Sedna dos sítios do experimento. A variação entre o número de sítios foi feita entre 2 e 8 sítios.



(a) Tempo de resposta

(b) Número de *deadlocks*

Figura 4.12 Variação no número de sítios

A Figura 4.12 (a) mostra que ambas as abordagens, o tempo de resposta diminui diante do crescimento do número de sítios. Isso está relacionado ao fato de que, com o crescimento do número de sítios, a sobrecarga é menor no sistema como um todo, por utilizar replicação parcial. Além disso, o gerenciador de bloqueios é descentralizado; então, o *overhead* total de bloqueios também é distribuído entre os sítios.

Já a Figura 4.12 (b) exibe o comportamento do número de *deadlocks* diante do crescimento do número de sítios. O DTX apresenta maior deficiência em termos

de *deadlock* do que o DTX com o Node2PL. O motivo disso é o fato de que, com o crescimento do número de sítios, o DTX consegue maior grau de concorrência, por possuir um menor *overhead* de bloqueios e um protocolo multigranular, o que favorece um menor tempo de resposta. Assim, o DTX atinge elevado grau de entrelaçamento de transações, ocasionando um número mais elevado de conflitos, ou seja, *deadlocks*.

4.3.5 Vazão e Grau de Concorrência

O último experimento visa a identificar o número de transações consolidadas por intervalo de tempo. Com isso, pode-se obter aspectos simples de vazão e grau de concorrência. Neste experimento, utilizou-se replicação parcial e fixou-se o número de 50 clientes, cada um submetendo 5 transações contendo 5 operações. O percentual de transação de atualização é 20% e o percentual de operações de atualização por transação de atualização é 20%. A base de dados utilizada neste experimento possui tamanho de 40MB, fragmentada e alocada em 4 sítios. O total de transações submetidas neste sistema é de 250.

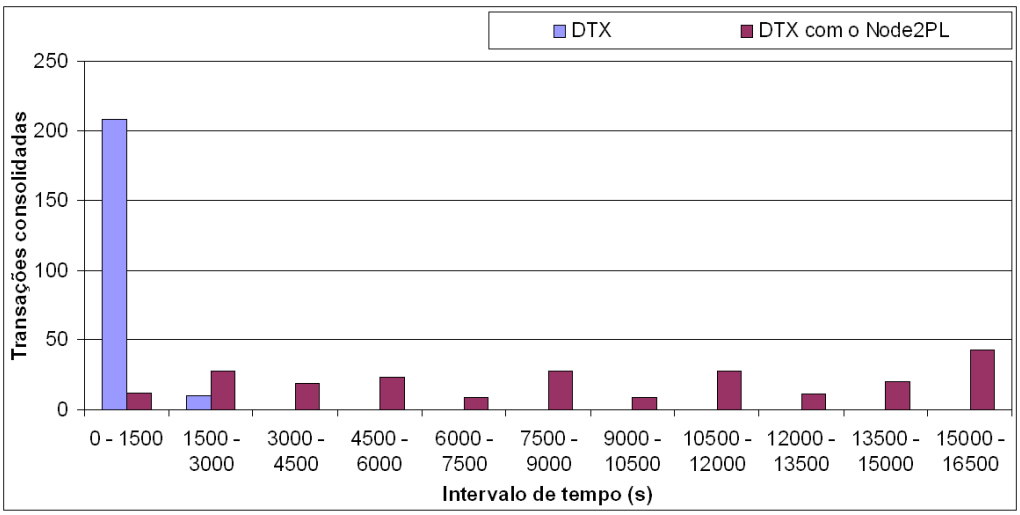


Figura 4.13 Vazão e grau de concorrência

Na Figura 4.13, é possível visualizar o DTX fornecendo maior vazão e grau de concorrência do que o DTX com o Node2PL. Neste experimento, o DTX executou 218 transações em 1553 segundos, já o DTX com o Node2PL processou 230 transações em 16500 segundos. A razão disso é que o DTX possui menor *overhead* na gerência de bloqueios e, por ser multigranular, isso favorece um menor tempo de resposta e que um maior número de transações entrem em execução concorrentemente. Pode-se observar

que o DTX deixou de executar 32 transações, já o DTX com o Node2PL, 20 transações. Com isso, se essas transações forem postas novamente em execução, o tempo de resposta do DTX, ainda sim, seria menor na execução completa do experimento.

4.4 CONCLUSÃO

Este capítulo descreveu a implementação e avaliação do DTX. Foi justificada a escolha dos artefatos utilizados, os detalhes da implementação foram descritos, além dos experimentos realizados e do ambiente de avaliação. Ao final, foram exibidos os resultados obtidos na avaliação do DTX. O capítulo a seguir expressa as conclusões do trabalho.

CAPÍTULO 5

CONCLUSÃO

O presente trabalho apresentou o DTX como um mecanismo que fornece um controle de concorrência distribuído para dados XML, com menor tempo de resposta na gerência eficaz destes dados. O DTX possui arquitetura modular e flexível, o que facilita sua integração a qualquer estrutura de armazenamento XML, além de poder ser estendido quando se adicionam novas características.

5.1 RESULTADOS

Em razão da flexibilidade e da simplicidade do XML, adotá-lo significa boa escolha. Por exemplo, XML pode ser usado na representação de dados na Internet, como formato padrão para troca de informações entre diferentes aplicativos, dentre outros. Com um volume muito grande de documentos XML distribuídos pela Web, surge a necessidade de gerenciamento eficiente destes documentos.

Muitos trabalhos acadêmicos e sistemas estão sendo propostos e oferecem gerenciamentos de documentos XML. A maioria dos trabalhos, porém, não oferece meios para distribuição de dados. Outros ensaios oferecem suporte à distribuição, mas estes possuem protocolos com baixo grau de concorrência.

Nesta dissertação, foi apresentado o DTX, como mecanismo para o controle de concorrência distribuído para dados XML. O DTX contribui para melhor tempo de resposta no processamento de transações distribuídas, ao utilizar um protocolo que leva em consideração características de XML, e possui uma granulosidade baixa.

Avaliou-se o DTX, considerando o tempo de resposta e o número de *deadlock* para executar um conjunto de transações. Pela análise dos resultados, foi possível verificar que o DTX melhorou o tempo de resposta na execução de transações distribuídas em diversas situações, mas mostrou deficiente com relação à ocorrência de *deadlocks*. Foi possível

verificar também que o DTX é flexível, permitindo a adaptação de outros protocolos para controle de concorrência, como o utilizado na avaliação.

5.2 TRABALHOS FUTUROS

Como trabalhos futuros, a pretensão deste pesquisador é, primeiramente, desenvolver soluções mais eficazes de sincronização e balanceamento de carga, considerando o volume de requisições nos sítios e o conteúdo das transações. Com isso, consegue-se um mecanismo ainda mais robusto para o processamento de transações em dados XML. O DTX foi validado com o SGBDXN Sedna. Pode, entretanto, ser utilizado por qualquer estrutura de armazenamento XML. Poder-se-ia, posteriormente, avaliá-lo com outros SGBDXNs, tais como Timber, Natix e eXist.

Pretende-se desenvolver soluções para o DTX contemplar as propriedades de atomicidade e durabilidade e assim oferecer um suporte transacional completo aos seus usuários. Durante a avaliação do DTX, foram observadas consideráveis quantidades de *deadlocks*. Portanto, é necessário um estudo aprofundado desses resultados, bem como da estrutura de funcionamento dos algoritmos, visando a identificar os fatores que possam ter ocasionado esse problema. Para melhor robustez, confiabilidade e segurança ao DTX, pretende-se aplicar estratégias para que o processamento no DTX não seja realizado todo em memória principal. Por fim, pretende-se adicionar outros protocolos de controle de concorrência e verificar o desempenho deles adaptados ao DTX.

No que concerne à avaliação do desempenho, é ainda proposta a avaliação do DTX em ambientes de WAN, com o intuito de identificar a variação no desempenho adicionado em decorrência da latência da rede. Para tanto, é necessário identificar parâmetros e desenvolver cenários que contemplem um ambiente mais geral do que o utilizado nos experimentos aqui apresentados. Tenciona-se, também, verificar e realizar outros experimentos com o DTX, comparando-o com o protocolo 2PC, e assim constatar o impacto e o desempenho da abordagem de comunicação e sincronização proposta pelo DTX.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Torsten Grabs, Klemens Böhm, and Hans-Jörg Schek. Xmltm: efficient transaction management for xml documents. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 142–152, McLean, Virginia, USA, 2002. ACM Press.

[2] W3C. *Extensible Markup Language*, 2008. `http://www.w3.org/XML/`.

[3] Kuen-Fang Jea, Shih-Ying Chen, and Sheng-Hsien Wang. Lock-based concurrency control for xml document models. 2006.

[4] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 436–445, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[5] Francesco Pagnamenta. Design and initial implementation of a distributed xml database. Master's thesis, University of Dublin, Irlanda, 2005.

[6] Tamino. *Tamino XML Server*, 2008. `http://www.softwareag.com/tamino`.

[7] Margo I. Seltzer. Berkeley db: A retrospective. *IEEE Data Eng. Bull.*, 30(3):21–28, 2007.

[8] Flávio Rubens C. Sousa. Replix: Um mecanismo para replicação de dados xml. Master's thesis, Universidade Federal do Ceará, 2007.

[9] Otávio C. Décio. *Guia de Consulta Rápida XML*. Novatec, 2000.

[10] Elaine Castro. Xml-pm: Um método eficiente para identificação de padrões no processamento de consultas a dados xml. Master's thesis, Universidade Federal do Ceará, Fortaleza, 2006.

[11] Daniela Florescu and Donald Kossmann. Storing and querying xml data using a rdbms. *IEEE Data Engineering Bulletin*, 1999.

[12] Jayavel Shanmugasundaram, Eugene Shekita, Jerry Kiernan, Rajasekar Krishnamurthy, Efstratios Viglas, Jeffrey Naughton, and Igor Tatarinov. A general technique for querying xml documents using a relational database system. *SIGMOD Rec.*, 30(3):20–26, 2001.

[13] Albrecht Schmidt, Martin L. Kersten, Menzo Windhouwer, and Florian Waas. Efficient relational storage and retrieval of xml documents. In *Selected papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases*, pages 137–150, London, UK, 2001. Springer-Verlag.

[14] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. Timber: A native xml database. *The VLDB Journal*, 11(4):274–291, 2002.

[15] Thorsten Fiebig, Sven Helmer, Carl-Christian Kanne, Guido Moerkotte, Julia Neumann, Robert Schiele, and Till Westmann. Anatomy of a native xml base management system. *VLDB J.*, 11(4):292–314, 2002.

[16] eXist. *eXist: Open Source Native XML Database*, 2008. `http://exist.sf.net`.

[17] Andrey Fomichev, Maxim Grinev, and Sergey Kuznetsov. Sedna: A native xml dbms. In *SOFSEM 2006: Theory and Practice of Computer Science, 32nd Conference on Current Trends in Theory and Practice of Computer Science*, volume 3831 of *Lecture Notes in Computer Science*, pages 272–281, Merin, Czech Republic, 2006. Springer.

[18] X-Hive. *X-Hive Database*, 2008. `http://www.x-hive.com`.

[19] Michael Brundage. *XQuery: The XML Query Language*. Addison-Wesley, 2004.

[20] Michiels Philippe. Xquery optimization. In *Proc. of the VLDB 2003 PhD Workshop. Co-located with the 29th International Conference on Very Large Data Bases (VLDB 2003)*, number 76 in CEUR Workshop Proceedings, Berlin, Germany, 2003. Morgan Kaufmann.

[21] Mary F. Fernandez, Jérôme Siméon, and Philip Wadler. An algebra for xml query. In *FST TCS 2000: Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 11–45, London, UK, 2000. Springer-Verlag.

[22] H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, and Keith Thompson. Tax: A tree algebra for xml. In *Database Programming Languages, 8th International Workshop, DBPL 2001*, volume 2397 of *Lecture Notes in Computer Science*, pages 149–164, Frascati, Italy, 2001. Springer.

[23] Stelios Paparizos, Yuqing Wu, Laks V. S. Lakshmanan, and H. V. Jagadish. Tree logical classes for efficient evaluation of xquery. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 71–82, New York, NY, USA, 2004. ACM.

[24] W3C. *XQuery 1.0: An XML Query Language*, 2006. `http://www.w3.org/XML/Query`.

[25] Mary F. Fernandez and Jérôme Siméon. Growing xquery. In *European Conference on Object-Oriented Programming*, volume 2743 of *Lecture Notes in Computer Science*, pages 405–430, Darmstadt, Germany, 2003. Springer Berlin / Heidelberg.

[26] Cynthia P. Santiago and Javam C. Machado. i-fox: Um Índice eficiente e compacto para dados xml. In *XIX Simpósio Brasileiro de Bancos de Dados*, pages 191–203, Brasília, Distrito Federal, Brasil, 2004. UnB.

[27] XUpdate. *XML Update Language*, 2008. `http://xmldb-org.sf.net/xupdate/`.

[28] Giorgio Ghelli, Kristoffer Høgsbro Rose, and Jérôme Siméon. Commutativity analysis in xml update languages. In *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, 2007, Proceedings*, volume 4353 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2007.

[29] Stijn Dekeyser and Jan Hidders. Conflict scheduling of transactions on xml documents. In *ADC '04: Proceedings of the fifteenth conference on Australasian database*, pages 93–101, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.

[30] Michael Peter Haustein and Theo Härder. A lock manager for collaborative processing of natively stored xml documents. In *XIX Simpósio Brasileiro de Bancos de Dados*, pages 230–244, Brasília, Distrito Federal, Brasil, 2004. UnB.

[31] Sven Helmer, Carl-Christian Kanne, and Guido Moerkotte. Evaluating lock-based protocols for cooperation on xml documents. *SIGMOD Rec.*, 33(1):58–63, 2004.

[32] Kuen-Fang Jack Jea, Shih-Ying Chen, and Sheng-Hsien Wang. Concurrency control in xml document databases: Xpath locking protocol. In *ICPADS '02: Proceedings of the 9th International Conference on Parallel and Distributed Systems*, pages 551–556. IEEE Computer Society, 2002.

[33] Philip Bernstein and Eric Newcomer. *Principles of transaction processing: for the systems professional.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[34] Peter Pleshachkov, Petr Chardin, and Sergey Kuznetsov. A dataguide-based concurrency control protocol for cooperation on xml data. In *9th East-European Conference on Advances in Databases and Information Systems (ADBIS)*, volume 3631 of *Lecture Notes in Computer Science*, pages 268–282, Tallinn, Estonia, 2005. Springer Berlin / Heidelberg.

[35] Michael Haustein, Theo Härder, and Konstantin Luttenberger. Contest of xml lock protocols. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 1069–1080. VLDB Endowment, 2006.

[36] Michael Peter Haustein and Theo Härder. tadom: A tailored synchronization concept with tunable lock granularity for the dom api. In *ADBIS*, pages 88–102, 2003.

[37] Muhammad Mainul Hossain. *Architectural Approaches of XDBMS Realization*, 2008. `http://wwwlgis.informatik.uni-kl.de/cms/fileadmin/courses/ws0708/Seminar/Thema_8_Ausarbeitung_Hossain.pdf`.

[38] WanSong Zhang, DaXin Liu, and Wei Sun. Xr-lock: locking xml data for efficient concurrency control. *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on*, 5:3921– 3925, 2004.

[39] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems.* McGraw-Hill, 2003.

[40] Harald Schoning. Tamino - a dbms designed for xml. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, page 149, Washington, DC, USA, 2001. IEEE Computer Society.

[41] Oracle. *Berkeley Database XML*, 2008. `http://www.oracle.com/database/berkeley-db/xml/index.html`.

[42] Oracle. Anatomy of an xml database: Oracle berkeley db xml. Technical report, Oracle Corporation, 2006.

[43] Oracle. *Berkeley DB Reference Guide, Version 4.5.20*, 2008. `http://www.oracle.com/technology/documentation/berkeley-db/xml/ref/toc.html`.

[44] Zhengwei Qi, Xiao Xie, Baowen Zhang, and Jinyuan You. Integrating x/open dtp into grid services for grid transaction processing. *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pages 128–134, May 2004.

[45] Flávio R. C. Sousa, Heraldo J. A. Carneiro Filho, Rossana M. C. Andrade, and Javam C. Machado. Replix: Um mecanismo para a replicação de dados xml. In *SBBD*, pages 53–67, 2007.

[46] Peter Pleshachkov and Petr Chardin. A locking protocol for scheduling transactions on xml data. In *Spring Young Researcher's Colloquium on Database and Information Systems SYRCoDIS, St.-Petersburg, Russia*, 2005.

[47] M. Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems.* Prentice-Hall, Inc., 1999.

[48] Can Türker, Klaus Haller, Christoph Schuler, and Hans-Jörg Schek. How can we support grid transactions? towards peer-to-peer transaction processing. In *Conference on Innovative Data Systems Research (CIDR)*, pages 174–185, 2005.

[49] Sun Microsystems. *Java 2 Standart Development Kit*, 2008. `http://java.sun.com`.

[50] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. Xmark: A benchmark for xml data management. In *28th International Conference on Very Large Data Bases*, pages 974–985, Hong Kong, China, 2002. Morgan Kaufmann.

[51] Benjamin Bin Yao, M. Tamer Özsu, and Nitin Khandelwal. Xbench benchmark and performance testing of xml dbmss. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 621, Washington, DC, USA, 2004. IEEE Computer Society.

[52] Hongjun Lu, Jeffrey Xu Yu, Guoren Wang, Shihui Zheng, Haifeng Jiang, Ge Yu, and Aoying Zhou. What makes the differences: benchmarking xml database implementations. *ACM Trans. Inter. Tech.*, 5(1):154–194, 2005.

[53] Hiroto Kurita, Kenji Hatano, Jun Miyazaki, and Shunsuke Uemura. Efficient query processing for large xml data in distributed environments. In *AINA '07: Proceedings of the 21st International Conference on Advanced Networking and Applications*, pages 317–322, Washington, DC, USA, 2007. IEEE Computer Society.

[54] Alexandre Andrade, Gabriela Ruberg, Fernanda Araujo Baião, Vanessa P. Braganholo, and Marta Mattoso. Efficiently processing xml queries over fragmented repositories with partix. In *EDBT Workshops*, pages 150–163, 2006.

[55] Guilherme Figueiredo, Vanessa Braganholo, and Marta Mattoso. A methodology for query processing over distributed xml databases. Technical report, University Federal of Rio de Janeiro, 2007.

[56] Steven Holzner. *Inside XML.* New Riders Publishing, 2000.

[57] Oracle. Oracle berkeley db xml getting started with transaction processing for c++ release 2.3. Technical report, Oracle Corporation, 2006.

[58] Peter Pleshachkov. *Transaction Management in XML Database Management Systems.* PhD thesis, Institute for System Programming of Russian Academy of Sciences, Rússia, 2006.

[59] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts, 4th Edition.* McGraw-Hill Book Company, 2001.

APÊNDICE A

PROTOCOLO XDGL - LINGUAGEM DE ATUALIZAÇÃO

Para a atualização de dados nos documentos XML foi definida uma linguagem de atualização. Esta linguagem possui cinco tipos de operações de atualização: inserção, remoção, transposição, renomeação e troca. Na operação de inserção, é especificado o elemento ou atributo a ser inserido, sua localização por meio de uma expressão de caminho e uma palavra que define ser uma inserção que ocorrerá antes, depois ou no elemento alvo da expressão. Na operação de remoção, é especificada somente a expressão de caminho que leva ao elemento alvo. Na operação de transposição, é informada uma expressão de caminho que leva ao elemento alvo, outra de caminho que informa sua nova posição e uma palavra que especifica se sua localização será antes, depois ou no elemento especificado na segunda expressão. Já na operação de renomeação, é especificada uma expressão de caminho que leva ao elemento requerido e uma palavra que define o novo nome deste elemento. Na operação de troca, os elementos alvos da expressão de caminho são substituídos por um novo nó especificado.

A.1 LINGUAGEM DE ATUALIZAÇÃO

A linguagem de atualização do protocolo XDGL é composta de cinco operações: *insert*, *delete*, *replace*, *move* e *rename*. Estas operações formam um conjunto completo para manipulação de documentos XML. Neste apêndice, serão demonstradas cada funcionalidade destas operações e suas sintaxes definidas no protocolo XDGL.

Nas sintaxes das operações, indicamos por `Expr` uma expressão de caminho definida na linguagem XPath. Essas expressões são escritas seguindo o subconjunto da linguagem XPath implementada pelo protocolo XDGL. Uma expressão construtora chamada de `constructor` especifica a definição de um novo elemento XML, seja atributo ou nó.

Para especificar um nó, a sintaxe de uma expressão construtora é indicada por `element{elem-name}{content}`. O `elem-name` indica o nome do nó e `content` o seu valor. Já para inidicar um novo atributo, a sintaxe da expressão construtora é ilustrada por `attribute{name}{text}`. A palavra `name` indica o nome do atributo e `text` o seu valor.

INSERT

A operação INSERT é utilizada para adicionar um novo elemento XML em uma determinada localização no documento XML. A localização é definida por uma expressão de caminho (`Expr1`) e o novo elemento pode ser adicionado antes, após ou no elemento alvo de `Expr1`. A definição do novo elemento é indicada por uma expressão construtora (`constructor1`).

INSERT `constructor1` (**INTO** | **AFTER** | **BEFORE**) `Expr1`

Exemplo:

INSERT `element{homepage}{http://www.lia.ufc.br/}`
INTO `/site/people/person[@id="person01"]`

DELETE

A operação DELETE tem a finalidade de retirar um elemento ou uma subárvore do documento XML. O elemento ou subárvore a ser removida é indicada por uma expressão de caminho (`Expr1`).

DELETE `Expr1`

Exemplo:

DELETE `/site/people/person[@id="person01"]`

REPLACE

A operação REPLACE é responsável por localizar um elemento ou subárvore e substituí-lo por um novo elemento. A localização dos elementos alvos da operação é indicada por uma expressão de caminho (`Expr1`) e o novo elemento é especificado por uma expressão construtora (`constructor1`).

REPLACE `Expr1` **WITH** `constructor1`

Exemplo:

REPLACE `/site/people/person[@id="person01"]/name`
WITH `element{name}{Leonardo Oliveira Moreira}`

MOVE

A operação MOVE retira um elemento ou subárvore especificada por uma expressão de caminho (`Expr1`), e o coloca em uma posição indicada em outra expressão de caminho (`Expr2`). Durante esta operação, é possível indicar se os elementos ou subárvore serão realocados antes, após ou no elemento alvo de `Expr2`.

MOVE `Expr1` (**INTO** | **AFTER** | **BEFORE**) `Expr2`

Exemplo:

MOVE `/site/people/person[@id="person01"]/creditcard`
INTO `/site/people/person[@id="person03"]`

RENAME

A operação RENAME faz que a definição de um elemento indicado por uma expressão de caminho (`Expr1`) seja modificado por uma definição indicada em `NCNAME`. A palavra `NCNAME` especifica o novo nome do elemento.

RENAME `Expr1` **AS** `NCNAME`

Exemplo:

RENAME `/site/people/person[@id="person01"]/homepage` **AS** `page`

APÊNDICE B

OPERAÇÕES DE ATUALIZAÇÃO - BENCHMARK XMARK

O XMark é um *benchmark* para dados XML desenvolvido pela Universidade de Amsterdam, Holanda. A estrutura do documento modelada é um site de leilões na Internet e os dados são gerados em um único documento. As entidades principais são pessoas, leilões abertos, leilões fechados, item e categoria. O esquema hierárquico é mostrado na Figura B.1.

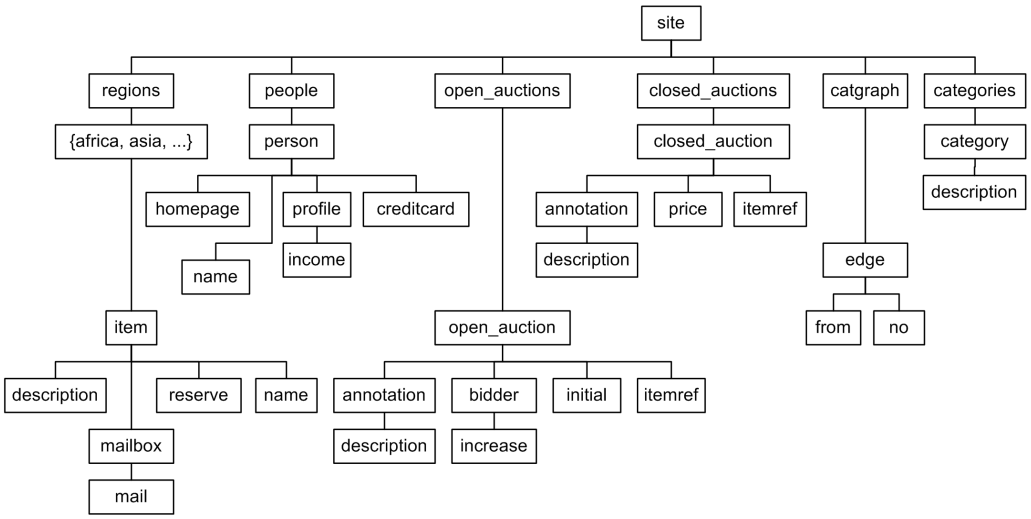


Figura B.1 Estrutura documento gerado pelo benchmark XMark

Os dados são gerados pelo *xmlgen*, um gerador de documentos. O *xmlgen* é independente de plataforma e escalável. As palavras usadas na criação dos documentos são as 17000 palavras mais comuns nas peças de teatro de Shakespeare. Essas palavras são selecionadas randomicamente, gerando um documento, de acordo com um DTD, que pode variar de 10 KB até 100 MB de tamanho.

O XMark especifica 20 consultas em XQuery, cada uma delas explorando um aspecto particular do sistema. Contudo, o XMark não contém nenhuma operação de

atualização. Assim sendo, criamos seis dessas operações para utilização durante os experimentos de avaliação:

- U1. Cria um novo elemento *person* com informações a respeito do nome, e-mail, telefone e página pessoal em sua subárvore, adicionando-a ao conjunto de elementos *person* do site (/site/people/).

  Exemplo:

```
insert
 <person id="person999">
  <name>Leonardo Moreira</name>
  <emailaddress>mailto:leoomoreira@lia.ufc.br</emailaddress>
  <phone>+55 (85) 32140152</phone>
  <homepage>http://www.lia.ufc.br/~leoomoreira</homepage>
 </person>
into /site/people
```

- U2. Insere um lance em um leilão aberto. Essa subárvore contém dados do lance como data, hora, valor de acréscimo e referência para a pessoa que fez o lance, e é adicionada em um determinado leilão aberto em /site/open_auctions/.

  Exemplo:

```
insert
 <bidder>
  <date>04/09/2006</date>
  <time>11:57:28</time>
  <personref person="person11"/>
  <increase>41.50</increase>
 </bidder>
into /site/open_auctions/open_auction[@id="open_auction15"]
```

- U3. Insere uma categoria denominada "category1" no "item0" registrado em site/regions/europe/.

  Exemplo:

```
insert
 <incategory category="category1" />
into site/regions/europe/item[@id="item0"]
```

- U4. Insere um registro de e-mail enviado para Javam Machado por Leonardo Moreira na data 25/12/2007 contendo a palavra "None" na caixa de correio do "item0" cadastrado em site/regions/namerica/.

  Exemplo:

```
insert
 <mail>
  <from>Leonardo Moreira mailto:leoomoreira@lia.ufc.br</from>
  <to>Javam Machado mailto:javam@ufc.br</to>
  <date>25/12/2007</date>
  <text>None</text>
 </mail>
into site/regions/namerica/item[@id="item0"]/mailbox
```

- U5. Cria um novo item. Sua subárvore contendo informações como nome do item, quantidade, localizaçao, etc. é adicionada na região norte americana.

  Exemplo:

```
insert
 <item id="item29">
  <location>United States</location>
  <quantity>1</quantity>
  <name>balthasar bred breathe </name>
  <payment>Cash</payment>
  <description>
   <text>
    mistake treacherous springe <emph> absent lucius </emph> fairly
   </text>
  </description>
  <shipping></shipping>
 </item>
into /site/regions/namerica
```

- U6. Insere um leilão fechado contendo como vendedor o "person999", como comprador o "person998", como produto o "item0", como preço 22.12 e na data 12/02/2008.

  Exemplo:

```
insert
 <closed_auction>
```

```
    <seller person="person999"/>
    <buyer person="person998"/>
    <itemref item="item0"/>
    <price>22.12</price>
    <date>21/02/2008</date>
    <quantity>1</quantity>
    <type>Featured</type>
  </closed_auction>
into /site/closed_auctions
```

Para a avaliação do DTX, utilizamos as seguintes consultas disponíveis no XMark transformadas para a linguagem XPath:

- Q1. Retorna o nome da pessoa com o ID "person0"
- Q15. Retorna as palavras-chave das anotações nos leilões fechados.

Para gerar transações com variadas consultas em diferentes partes da base de dados, as seguintes expressões XPath foram criadas para a realização dos experimentos:

- Q1. Retorna os nomes dos itens pagos em dinheiro na região da austrália.
- Q2. Lista a localização dos itens da região sul-americana que contém a categoria "category0".
- Q3. Lista os nomes dos itens localizados nos Estados Unidos registrados na região da África.
- Q4. Quais os nomes dos itens que possuem duas quantidades e estão registrados na região da Austrália?
- Q5. Lista os nomes dos itens contidos na região da Austrália.
- Q6. Retorna a localização dos itens que foram caracterizados e estão contidos na região da Ásia.
- Q7. Lista os nomes dos itens pagos com dinheiro na região sul-americana.
- Q8. Lista os nomes dos itens pagos com dinheiro na região da África.

- Q9. Retorna a descrição da categoria que possui o nome “health”.
- Q10. Lista o nome das pessoas que moram na cidade de Georgia.
- Q11. Lista o nome das pessoas que moram no país Luxemburgo.
- Q12. Retorna o nome das pessoas que possuem no perfil interesse na categoria 7.