STUDIA INFORMATICA 2006
Volume 27 Number 3 (68)

Krzysztof JANKIEWICZ
Politechnika Poznańska, Instytut Informatyki

PRZEGLĄD I ANALIZA METOD ZARZĄDZANIA WSPÓŁBIEŻNYM DOSTĘPEM DO BAZ DANYCH DOKUMENTÓW XML

Streszczenie. Dotychczasowe prace w zakresie przetwarzania dokumentów XML koncentrowały się na zagadnieniach: optymalizacja zapytań, transformacja dokumentów, sposoby przechowywania, kompresja dokumentów, normalizacja. Niewiele natomiast zrobiono w zakresie zapewnienia poprawności współbieżnego dostępu i aktualizacji dokumentów XML. Celem tego artykułu jest przegląd i analiza metod zarządzania współbieżnym dostępem do baz danych dokumentów XML.

Słowa kluczowe: bazy danych, XML, współbieżny dostęp, transakcje

SURVEY AND ANALYSIS OF CONCURRENCY CONTROL METHODS FOR XML DATABASE SYSTEMS

Summary. Existing work on processing of XML documents has been concentrated on: query optimization, storage problems, documents transformation, compressing methods, normalization. There are only few papers concurrency control in accessing and modifying XML documents, which are stored in XML database systems. The aim of this paper is to present survey and analysis of concurrency control methods for XML database systems.

Keywords: XML, database systems, concurrency control, transactions

1. Wstęp

Przetwarzanie dokumentów XML stanowi obecnie jeden z podstawowych nurtów badawczych w zakresie technologii przetwarzania danych. Popularność XML wynika z jego prostoty i elastyczności. Te cechy spowodowały, że XML stał się także popularnym i dominującym standardem wymiany danych o złożonej, zmiennej i nieokreślonej strukturze. Dzięki

tym cechom z powodzeniem jest wykorzystywany w dziedzinach: nauka, finanse, wymiana informacji, medycyna, grafika, kartografia, multimedia itp. Rosnące znaczenie i popularność XML'a powodują, że w ostatnim czasie bardzo wiele komercyjnych systemów zarządzania obiektowym i postrelacyjnym systemem baz danych rozszerzyło swoją funkcjonalność o mechanizmy pozwalające na przechowywanie i przetwarzanie dokumentów XML. Jednocześnie powstają liczne specjalizowane rozwiązania pod postacią XML-owych baz danych. Dotychczasowe prace w zakresie przechowywania i przetwarzania dokumentów XML koncentrowały się głównie na zagadnieniach związanych z: optymalizacją zapytań [10, 27, 7], transformacją dokumentów, metodami przechowywania dokumentów XML [1, 17, 16, 25], wykorzystaniem baz danych relacyjnych oraz obiektowych do przechowywania i przetwarzania dokumentów XML [22, 20, 21], kompresją dokumentów XML [18, 8, 23, 2]. Jednakże bardzo niewiele zrobiono w zakresie zarządzania współbieżnym dostępem do baz danych dokumentów XML. Jest to o tyle zaskakujące, że zarządzanie współbieżnym dostępem do danych stanowi jeden z podstawowych mechanizmów systemów zarządzania bazami danych (SZBD).

Współbieżny, niekontrolowany dostęp do baz danych dokumentów XML, podobnie jak w przypadku klasycznych relacyjnych i obiektowych baz danych, może prowadzić do niespójności bazy danych [26, 3]. Problem współbieżnego dostępu był szeroko rozważany w literaturze [3, 4]. W ramach tych prac opracowano kryterium poprawności współbieżnego dostępu do baz danych, szereg algorytmów zarządzania współbieżnym dostępem do danych. Podstawowe ogólnie przyjęte jest kryterium konfliktowej uszeregowalności. Opracowane algorytmy reprezentują trzy podstawowe podejścia: blokowania, uporządkowania i optymistyczne, które zostały z powodzeniem zaimplementowane w obiektowych relacyjnych i postrelacyjnych SZDB. Jak wspomnieliśmy powyżej, bardzo niewiele zrobiono w zakresie zarządzania współbieżnością w XML-owych bazach danych. Stosowane dotychczas mechanizmy oparte o mechanizm blokowania oferują bardzo niski poziom współbieżności, a co za tym, idzie niską efektywność przetwarzania. Zachodzi zatem oczywista potrzeba opracowania nowych metod zarządzania współbieżnym dostępem do baz danych dokumentów XML, zapewniających poprawność współbieżnej aktualizacji, efektywność oraz uwzględniających specyfikę dokumentów XML i różnorodność wykorzystywanych interfejsów. Dotychczas przedstawiono bardzo niewiele propozycji rozwiązania tego problemu: [13, 14, 12, 9, 6, 5, 24, 15]. Rozwiązania te różnią się od siebie poziomem współbieżności, stosowanym interfejsem dostępu do dokumentu. Brak jest krytycznej analizy zaproponowanych rozwiązań. Istnieje potrzeba porównania tych algorytmów.

Celem tego artykułu jest przedstawienie dotychczas zaproponowanych algorytmów zarządzania współbieżnym dostępem oraz ich krytyczna jakościowa analiza. W pracy przedstawio-

no klasyfikację tych algorytmów, zdefiniowano kryteria oceny, przeprowadzono analizę algorytmów w oparciu o wprowadzone kryteria.

Struktura artykułu jest następująca. W rozdziale drugim omówiono podstawowe standardy związane z dokumentami XML i wykorzystywane przy ich przetwarzaniu. Rozdział trzeci dotyczy zarządzania współbieżnym dostępem, przedstawiono w nim przykład motywacyjny oraz pokrótce omówiono klasyczne protokoły zarządzające współbieżnym dostępem w bazach danych. Rozdział czwarty przedstawia algorytmy zarządzania współbieżnym dostępem do dokumentów XML. Rozdział piąty jest krytyczną analizą propozycji przedstawionych w rozdziale czwartym. Rozdział szósty jest podsumowaniem.

2. XML, DOM, XPath

Extensible Markup Language (XML) jest prostym formatem tekstowym opartym na znacznikach, wywodzącym się z języka SGML. Zaprojektowany początkowo jako format elektronicznej publikacji informacji szybko stał się powszechnie wykorzystywanym formatem wymiany informacji.

Użytkownik wewnątrz dokumentu XML definiuje znaczniki, które pozwalają na wyznaczanie struktury dokumentu, hierarchii poszczególnych składowych itp. Dla przykładu, wewnątrz znaczników *<person>...</person>* można zawrzeć informacje dotyczącą pojedynczej osoby. Informacja ta może składać się z elementów prostych – tekstu, lub może posiadać strukturę złożoną, wykorzystującą kolejne znaczniki np. *<name>*, *<lastname>*, *<birthday>*. Znacznik otwierający *<person>* i zamykający *</person>*, wraz z zawartością, nosi nazwę elementu. XML zezwala także na wykorzystywanie atrybutów umieszczanych wewnątrz znaczników otwierających elementy. Standard XML oprócz definiowania elementów umożliwia także tworzenie atrybutów. Atrybuty definiuje się wewnątrz znaczników otwierających elementy. Pełnią one taką samą rolę jak elementy – pozwalają na przechowywanie informacji i definiowanie jej struktury oraz zależności. Różnica polega na tym, że atrybut może wystąpić w ramach elementu tylko raz i jest zawsze prostą wartością tekstową.

W dokumentach XML istnieją wyróżnione typy atrybutów ID i IDREF, będące odpowiednikami klucza podstawowego i klucza obcego w modelu relacyjnym. Atrybuty typu ID pełnią rolę identyfikatora elementu i są unikalne w obrębie całego dokumentu. Wykorzystywane są przy odwołaniach do elementów z innych części dokumentu. Atrybuty typu IDREF pełnią rolę referencji do elementu i wykorzystują do tego celu wartości użyte w atrybutach typu ID.

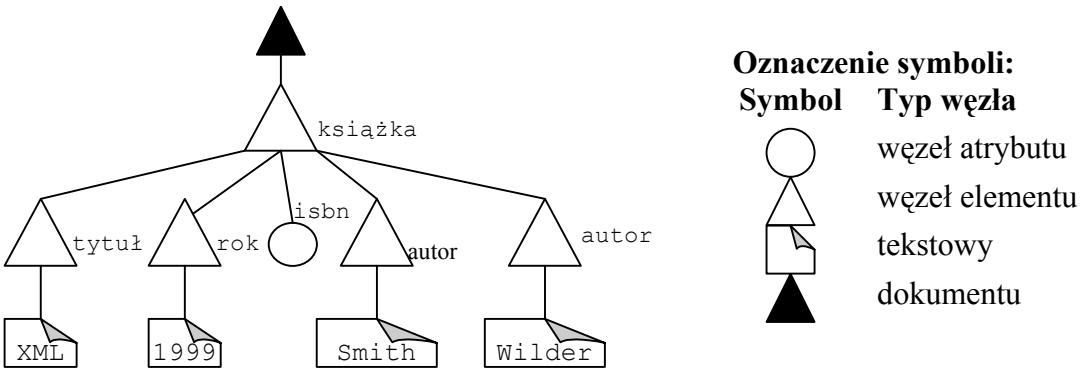
Przykładowy dokument XML został przedstawiony na rys. 1.

```
<książka isbn="KD-12345-XY">
  <tytuł>XML</tytuł>
  <rok>1999</rok>
  <autor>Smith</autor>
  <autor>Wilder</autor>
</książka>
```

Rys. 1. Dokument XML
Fig. 1. XML document

Dla dokumentów XML opracowano wiele standardów umożliwiających ich przetwarzanie. Do najpopularniejszych należą DOM i XPath, które jednocześnie stanowią podstawowe interfejsy dostępu do dokumentów XML.

Standard DOM (Document Object Model) jest jednym z najstarszych standardów związanych z przetwarzaniem XML. Udostępnia on niezależny od języka programowania standardowy zbiór obiektów służących do reprezentacji dokumentów oraz standardowy interfejs dostępu i manipulacji nimi. Dzięki temu za pomocą standardu DOM możemy odwzorować dokumenty w ich obiektową reprezentację, odczytywać i modyfikować ich zawartość, nawigować po ich strukturze i zmieniać ją. Reprezentacja dokumentów XML w standardzie DOM przypomina strukturę drzewiastą, stąd też często używa się wobec niej określenia drzewo DOM. Na rys. 2 przedstawiono drzewo DOM zbudowane w oparciu o dokument XML z rys. 1.



Rys. 2. Drzewo DOM
Fig. 2. DOM Tree

W ramach standardu DOM możemy wyróżnić procedury i funkcje odczytujące zawartość dokumentu XML oraz dokonujące jego modyfikacji.

Do pierwszych możemy zaliczyć metody operujące na strukturze dokumentów: `firstChild` – odczytuje pierwszy element podrzędny węzła, `lastChild` – odczytuje ostatni element podrzędny węzła, `previousSibling` – odczytuje bezpośrednio poprzedzający element, `nextSibling` – odczytuje bezpośrednio następny element, `getNodeById` – odczytuje węzeł na podstawie jego identyfikatora (wartości atrybutu ID), `getElementByTagname` – odczytuje węzły posiadające określoną wartość znacznika, oraz metody odczytujące zawartość elemen-

tów: `Text` – odczytuje zawartość tekstową elementu, `nodeName` – odczytuje nazwę znacznika elementu, `getAttribute` – pobiera atrybut na podstawie jego nazwy.

Do drugiego zbioru możemy zaliczyć metody wykonujące zmiany w strukturze dokumentu: `insertBefore` – wstawia nowo utworzony węzeł przed już istniejący, `replaceChild` – usuwa istniejący węzeł, zastępując go nowo utworzonym, `removeChild` – usuwa istniejący węzeł, `appendChild` – dodaje nowoutworzony węzeł na koniec listy węzłów podrzędnych, oraz metody modyfikujące zawartość elementów: `appendData` – dodaje ciąg znaków na koniec zawartości węzła, `deleteData` – usuwa fragment z zawartości węzła, `insertData` – wstawia tekst do zawartości węzła, `replaceData` – zamienia tekst wewnątrz węzła, `setAttribute` – ustawia wartość atrybutu.

Standard DOM jest interfejsem proceduralnym, to znaczy, że użytkownik w sposób proceduralny definiuje, w jaki sposób chce wykonać określoną modyfikację lub odczyt. Innym podejściem do definiowania modyfikacji i odczytów jest podejście deklaratywne. Zdefiniowano wiele standardów związanych z XML'em, wykorzystujących podejście deklaratywne. Przykładem standardu wykorzystującego podejście deklaratywne jest standard XPath.

XPath jest standardem, który pozwala na adresowanie fragmentów dokumentów XML. Do adresowania wykorzystywane są wyrażenia ścieżkowe podobne do tych, które są wykorzystywane do definiowania ścieżek katalogów w systemie plików. Wyrażenia ścieżkowe XPath mogą wykorzystywać wyrażenia regularne, funkcje itp. Każde wyrażenie XPath składa się z tzw. kroków rozdzielonych ukośnikami. Każdy z kroków zawiera:

- opcjonalny specyfikator współrzędnych, służący do określenia miejsca w drzewie dokumentu, począwszy od którego wyszukiwane będą węzły,
- test węzła, służący do określenia, które węzły są wyszukiwane w obszarze drzewa określonym przez specyfikator współrzędnych, oraz
- opcjonalny predykat, dodatkowo zawężający test węzła, powodujący wybór tylko tych węzłów, które spełniają podany warunek.

Format każdego kroku wyrażenia XPath jest następujący:

```
specyfikator_współrzędnych::test_węzła[predykat]
```

Tabela 1 przedstawia przykłady wyrażeń Xpath, adresujących obiekty w dokumencie XML z rys. 1.

Tabela 1

Przykłady wyrażeń XPath

Wyrażenie XPath	Zaadresowane fragmenty dokumentu
`/książka@isbn`	`KD-12345-XY`
`//autor`	`<autor>Smith</> <autor>Wilder</>`
`/książka/rok/text()`	`1999`
`/książka[rok=1999]/autor[1]`	`<autor>Smith</>`

Podstawowe elementy składni wyrażeń XPath są następujące:

`/` – oznacza bezpośrednią zależność, znak `/` na początku wyrażenia XPath oznacza korzeń dokumentu,

`//` – oznacza dowolną liczbę poziomów nieokreślonych elementów występujących w hierarchii dokumentu,

`*` – oznacza element o dowolnym znaczniku,

`[]` – pozwalają na zdefiniowanie warunków nakładanych na elementy, atrybuty itp.,

`@` – pozwala na odwołanie do atrybutu.

XPath jest standardem bardzo szeroko wykorzystywanym w przetwarzaniu dokumentów XML. Stanowi on podstawę języka transformacji dokumentów XSLT (ang. *Extensible Stylesheet Language Transformations*), umożliwiającego transformację dokumentów XML w dowolny obiekt tekstowy, ale także występuje w językach zapytań takich jak XQuery, czy językach modyfikacji danych np. XUpdate czy Lexus.

W bardzo wielu systemach informatycznych standard DOM i standard XPath występują równolegle jako dwie alternatywne metody dostępu do dokumentów, zależne od preferencji i potrzeb użytkownika.

3. Zarządzanie współbieżnością

3.1. Przykład motywacyjny

Jak wspomnieliśmy, niekontrolowany współbieżny dostęp do tych samych danych może prowadzić do niespójności bazy danych. Dotyczy to również XML-owych baz danych. Dla ilustracji rozważmy następujący przykład.

Dany jest dokument XML z rys. 1 oraz dwóch użytkowników A i B, którzy wykonują następujące transakcje. Użytkownik A w transakcji T_1 sprawdza czy dokument opisuje książkę wydaną w roku 1999, jeśli tak, to usuwa istniejących autorów książki, wstawiając w ich miejsce element `<autor>Pantano</autor>`. Jednocześnie użytkownik B w transakcji T_2 weryfikuje czy współautorem książki jest `Smith` i jeśli tak, to zmienia rok wydania książki na `2000`. Załóżmy następującą współbieżną realizację transakcji T_1 i T_2:

T_1: czyta element `/książka/rok` (sprawdzając czy jego wartość to `1999`,
jest to warunek konieczny wykonania przez transakcję T_1 dalszych operacji)
T_2: czyta element `/książka/autor` (sprawdzając czy jego wartość to `Smith`,
jest to warunek konieczny wykonania przez transakcję T_2 dalszych operacji)
T_1: usuwa elementy `/książka/autor`
T_1: wstawia element `/książka/autor` z zawartością `Pantano`
T_2: modyfikuje element `/książka/rok` (ustawiając jego wartość na `2000`)

Rys. 3. Współbieżna realizacja transakcji
Fig. 3. Concurrent execution of transactions

Realizacja obu transakcji pozostawia dokument z rys. 1 w następującej postaci:

```
<książka isbn="KD-12345-XY">
  <tytuł>XML</>
  <rok>2000</>
  <autor>Pantano</>
</>
```

Rys. 4. Dokument XML po współbieżnej realizacji transakcji
Fig. 4. XML document after concurrent execution of transactions

Zauważmy, że przedstawiona realizacja jest niepoprawna, to znaczy, prowadzi do niespójności bazy danych. Zgodnie z ogólnym kryterium uszeregowalności współbieżna realizacja zbioru transakcji T jest poprawna, jeżeli jest równoważna dowolnej sekwencyjnej realizacji zbioru transakcji T. Jak łatwo zauważyć, przedstawiona na rys. 4 realizacja transakcji T_1 i T_2 nie jest równoważna żadnej z dwóch możliwych realizacji sekwencyjnych transakcji T_1 i T_2 (rys. 5 i rys. 6).

T_1: czyta element `/książka/rok` (sprawdzając czy jego wartość to `1999`, jest to warunek konieczny wykonania przez transakcję T_1 dalszych operacji)
T_1: usuwa elementy `/książka/autor`
T_1: wstawia element `/książka/autor` z zawartością `Pantano`
T_2: czyta element `/książka/autor` (sprawdzając czy jego wartość to `Smith`, jest to warunek konieczny wykonania przez transakcję T_2 dalszych operacji)
T_2: modyfikuje element `/książka/rok` ustawiając jego wartość na `2000`) – operacja nie będzie miała miejsca, ponieważ warunek konieczny do jej przeprowadzenia nie jest spełniony

Rys. 5. Sekwencyjna realizacja T_1,T_2
Fig. 5. Serial execution T_1,T_2

T_2: czyta element `/książka/autor` (sprawdzając czy jego wartość to `Smith`, jest to warunek konieczny wykonania przez transakcję T_2 dalszych operacji)
T_2: modyfikuje element `/książka/rok` (ustawiając jego wartość na `2000`)
T_1: czyta element `/książka/rok` (sprawdzając czy jego wartość to `1999`, jest to warunek konieczny wykonania przez transakcję T_1 dalszych operacji)
T_1: usuwa elementy `/książka/autor` – operacja nie będzie miała miejsca, ponieważ warunek konieczny do jej przeprowadzenia nie jest spełniony
T_1: wstawia element `/książka/autor` z zawartością `Pantano` – jw.

Rys. 6. Sekwencyjna realizacja T_2,T_1
Fig. 6. Serial execution T_2,T_1

Stan bazy danych, uzyskany w wyniku realizacji sekwencyjnej przedstawionej na rys. 5, jest zgodny ze stanem przedstawionym na rys. 7.

```
<książka isbn="KD-12345-XY">
  <tytuł>XML</>
  <rok>1999</>
  <autor>Pantano</>
</>
```

Rys. 7. Stan bazy danych po sekwencyjnej realizacji transakcji T_1 i T_2
Fig. 7. Database state after serial sequential of T_1 and T_2

Zauważmy, że stan bazy danych, uzyskany w wyniku tej realizacji sekwencyjnej, nie jest zgodny ze stanem realizacji współbieżnej.

W przypadku przedstawionym na rys. 6 stan bazy danych, uzyskany w wyniku realizacji sekwencyjnej, jest zgodny ze stanem przedstawionym na rys. 8:

```
<książka isbn="KD-12345-XY">
  <tytuł>XML</>
  <rok>2000</>
  <autor>Smith</>
  <autor>Wilder</>
</>
```

Rys. 8. Stan bazy danych po sekwencyjnej realizacji transakcji T_2 i T_1
Fig. 8. Database state after serial sequential of T_2 and T_1

Zauważmy, że również w tym przypadku stan bazy danych, uzyskany w wyniku tej realizacji sekwencyjnej, nie jest zgodny ze stanem realizacji współbieżnej.

Jak zatem widać, nieuszeregowalna, współbieżna realizacja transakcji w XML-owej bazie danych jest analogiczna do przypadku klasycznych relacyjnych i obiektowych baz danych. Zachodzi zatem oczywista konieczność opracowania i implementacji mechanizmów zarządzania współbieżnym dostępem do XML-owych baz danych.

Większość rozwiązań zaproponowanych dla XML wykorzystuje klasyczne mechanizmy wypracowane na gruncie klasycznej teorii zarządzania współbieżnością [3, 4]. Stąd, zanim przejdziemy do przedstawienia zaproponowanych rozwiązań w zakresie zarządzania współbieżnością w XML-owych bazach danych, krótko scharakteryzujemy i przedstawimy poniżej podstawowe podejścia w zakresie zarządzania współbieżnym dostępem, opracowane dla systemów relacyjnych baz danych.

3.2. Algorytmy zarządzania współbieżnością

W zakresie zarządzania współbieżnością wyróżnia się, najogólniej, trzy zasadnicze podejścia:

- metody blokowania,
- metody porządkowania transakcji wg znaczników czasowych,
- metody optymistyczne.

Obecnie krótko scharakteryzujemy poszczególne podejścia.

3.2.1. Metody blokowania

Najpopularniejszym podejściem w zakresie zarządzania współbieżnością jest podejście wykorzystujące blokady. W literaturze zaproponowano szereg algorytmów blokowania, które różnią się założeniami odnośnie struktury transakcji, struktury bazy danych. Cechą wspólną tych algorytmów jest to, że każda transakcja, która wykonuje operacje na bazie danych, zakłada odpowiednie blokady na odczytywanych lub modyfikowanych obiektach. Każda ope-

racja jest poprzedzona żądaniem odpowiedniego typu blokady. Typ blokady determinowany jest typem operacji, jaką transakcja zamierza wykonać. Blokady w zależności od typu mogą być ze sobą w konflikcie, uniemożliwiając transakcji przeprowadzenie zamierzonej operacji. W takich przypadkach transakcja oczekuje zazwyczaj na zdjęcie blokady konfliktowej, po czym kontynuuje przetwarzanie. Do określania konfliktu pomiędzy poszczególnymi typami blokad wykorzystywane są tzw. tabele kompatybilności blokad. Tabela 2 przedstawia przykładową tabelę kompatybilności blokad.

Tabela 2

Tabela kompatybilności blokad

		Blokady żądane	
Blokady założone		r	w
	r	+	-
	w	-	-

Z tabeli tej wynika, że blokady typu `r`, nazwane blokadami odczytu, lub blokadami współdzielonymi, nie są konfliktowe, natomiast blokady, wśród których występuje choć jedna blokada typu `w`, zwana blokadą zapisu lub blokadą wyłączną, są konfliktowe.

Przykładem protokołu opartego na blokadach jest *protokół blokowana dwufazowego* (2PL). Jest on najczęściej wykorzystywany w komercyjnych SZBD. Charakteryzuje się występowaniem dwóch faz: fazy zakładania blokad oraz fazy ich zdejmowania. Fazy te są rozłączne i występują dokładnie w takiej kolejności. Bardzo istotnym zagadnieniem w przypadku protokołów opartych na blokadach jest wykrywanie i rozwiązywanie zakleszczeń. Dochodzi do nich w przypadku, gdy ze względu na założone blokady dwie transakcje oczekują na wzajemne zakończenie.

Algorytmy blokowania gwarantują uszeregowaną realizację współbieżnych transakcji, co więcej, jak wykazały badania, algorytmy blokowania charakteryzują się bardzo dobrą efektywnością.

3.2.2. Metody porządkowania transakcji wg znaczników czasowych

Alternatywnym podejściem zaproponowanym w literaturze jest podejście wykorzystujące do zarządzania współbieżnością znaczniki czasowe transakcji (ang. T/O – *time ordering*).

W podejściu tym każda z transakcji w momencie inicjacji otrzymuje unikalny znacznik czasowy (ang. *timestamp*). Znaczniki czasowe są wykorzystywane do szeregowania dostępu do danych. Kolejność dostępu do danych jest określona porządkiem znaczników czasowych transakcji, tzn. zgodnie z metodą T/O, transakcje o wcześniejszych znacznikach czasowych realizują dostęp do danych przed transakcjami o późniejszych znacznikach czasowych. W przypadku naruszenia kolejności dostępu do danych wynikającej z porządku znaczników czasowych transakcji następuje odrzucenie transakcji. Innymi słowy, jeżeli transakcja T_i o znaczniku czasowym $TS(T_i)$ realizuje dostęp do danej po transakcji T_j o znaczniku czaso-

wym $TS(T_j)$ takim, że $TS(T_i) < TS(T_j)$, to transakcja T_i jest odrzucana i restartowana ponownie.

Algorytmy T/O gwarantują uszeregowaną realizację współbieżnych transakcji. Jak wykazały badania, efektywność metod T/O jest niższa od metod blokowania.

3.2.3. Metody optymistyczne

Trzecim podejściem zaproponowanym w literaturze jest tzw. podejście optymistyczne (ang. *optimistic aproach*), którego podstawowym założeniem jest, że prawdopodobieństwo konfliktu pomiędzy transakcjami jest w praktyce bardzo małe. W przedstawionych powyżej podejściach, metodach blokowania i znaczników czasowych zanim określona operacja się zostanie wykonana system weryfikuje, czy może to doprowadzić do naruszenia poprawności bazy danych. Z weryfikacją związany jest narzut czasowy, który ogranicza efektywność, gdy konflikty występują bardzo rzadko. W metodach optymistycznych stosowane są trzy fazy: faza odczytu, faza walidacji i faza zapisu. Podczas fazy odczytu transakcja odczytuje zatwierdzone dane z bazy danych. Natomiast modyfikacje wykonywane są tylko na lokalnej kopii danych przechowywanej indywidualnie dla każdej transakcji. Faza walidacji służy do weryfikacji, czy nie zostanie naruszona uszeregowalność w przypadku, gdy zmienione dane zostaną z lokalnej kopii przeniesione do bazy danych. Jeśli walidacja się powiedzie, wówczas podczas fazy zapisu zmiany przenoszone są do bazy danych. W przeciwnym przypadku modyfikacje zostają odrzucone, a transakcja jest restartowana.

Jak wykazano, metoda ta gwarantuje uszeregowaną realizację współbieżnych transakcji. Jednakże w przypadku, gdy podstawowe założenie braku konfliktów nie jest spełnione, efektywność metod optymistycznych drastycznie spada w stosunku do metod opartych na blokowaniu i metod opartych o znaczniki czasowe.

4. Algorytmy zarządzania współbieżnym dostępem do XML-owych baz danych

Hierarchiczna i semistrukturalna natura dokumentów XML powoduje, że metody zarządzania współbieżnym dostępem do danych przedstawione w poprzednim punkcie, opracowane dla relacyjnych i posterelacyjnych SZBD są niewystarczające dla potrzeb XML-owych baz danych. Stąd, w ciągu ostatnich kilku lat, z uwagi na zwiększające się znaczenie standardu XML, obserwujemy zainteresowanie problemem zarządzania współbieżnością dostępu do baz danych dokumentów XML.

W ostatnim czasie zaproponowano kilka algorytmów zarządzania współbieżnym dostępem do dokumentów XML. Algorytmy te można sklasyfikować w dwojaki sposób: ze wzglę-

du na zakładany przez nie standard dostępu do dokumentu oraz ze względu na wykorzystywany mechanizm zarządzania współbieżnością. Krótko omówimy teraz obie klasyfikacje.

Zgodnie z pierwszą klasyfikacją wyróżniamy następujące klasy algorytmów:

- oparte na standardzie DOM,
- oparte na standardzie XPath.

Algorytmy oparte na standardzie DOM zakładają dostęp do dokumentów XML za pomocą operacji standardu DOM API. W przypadku tych rozwiązań mamy do czynienia z blokadami na różnych poziomach dokumentu. Blokady dotyczą pojedynczych składowych dokumentu: elementów, atrybutów, a także zależności pomiędzy nimi (nadrzędny-podrzędny, następny-poprzedni). Algorytmy te, jako model dokumentu, wykorzystują drzewo DOM. Algorytmy oparte na standardzie XPath zakładają dostęp do dokumentów XML za pomocą wyrażeń XPath. W tym drugim przypadku sytuacje konfliktowe są rozpoznawane za pomocą porównywania ścieżek lub blokad zakładanych na wykorzystywanym przez te algorytmy modelu danych.

Zgodnie z drugą klasyfikacją, opartą na wykorzystywanym mechanizmie zarządzania współbieżnością, wyróżniamy następujące klasy algorytmów:

- oparte na blokadach,
- oparte na metodzie znaczników czasowych,
- oparte na podejściu optymistycznym.

W przypadku rozwiązań komercyjnych baz danych, w chwili obecnej, w zakresie zarządzania współbieżnym dostępem do dokumentów XML, dominują metody wykorzystujące mechanizm blokad. Nie spotyka się, jak dotąd, implementacji algorytmów opartych na znacznikach czasowych lub podejściu optymistycznym – pozostają one tylko w sferze propozycji. Metody, wykorzystujące mechanizm blokad, różnią się typami blokad oraz metodami ich zakładania.

4.1. Algorytmy zarządzania współbieżnym dostępem oparte na standardzie DOM

4.1.1. Algorytmy blokowania drzewa DOM

Algorytmy blokowania w procesie zarządzania współbieżnym dostępem do dokumentów XML zakładają odpowiednie blokady na różnych poziomach pojedynczego dokumentu. Oznacza to możliwość blokowania całych dokumentów, pojedynczych elementów, atrybutów.

W pracy [14] zaproponowano 4 algorytmy zarządzania współbieżnym dostępem do dokumentów XML oparte na idei algorytmu blokowania dwufazowego. Jak wspomnieliśmy już wcześniej, tradycyjny algorytm 2PL wykorzystuje dwa typy blokad: współdzielone S i wyłączne X. Operacje do odczytu wymagają blokad współdzielonych, podczas gdy operacje

modyfikujące wymagają blokad wyłącznych. Operacje standardu DOM można podzielić ogólnie na dwa rodzaje: operacje odczytu i operacje modyfikacji. Pierwsze pozwalają na poruszanie się po dokumencie XML, drugie na jego modyfikację. Pierwsze są odpowiednikiem operacji odczytu zaś drugie odpowiednikiem operacji modyfikacji w systemach relacyjnych baz danych. W związku z tym, w pracy [14] wprowadzono dwa typy blokad: T i M, odpowiednio dla obu rodzajów operacji. Tabela 3 przedstawia tabelę kompatybilności blokad.

Tabela 3
Tabela kompatybilności blokad

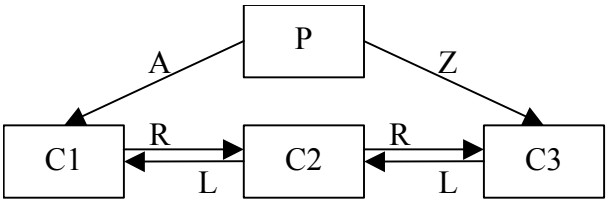
	T	M
T	+	–
M	–	–

Przed wykonaniem każdej operacji odczytu lub modyfikacji niezbędne jest założenie odpowiedniej blokady. W pierwszej fazie weryfikuje się istnienie blokad konfliktowych. W przypadku istnienia blokad konfliktowych wykonywanie transakcji jest wstrzymywane aż do zakończenia transakcji konfliktowej (transakcji konfliktowych). Po uzyskaniu wszystkich blokad transakcja osiąga punkt akceptacji (ang. *commit point*) i przechodzi do fazy zdejmowania blokad.

4.1.1.1. Algorytm Doc2PL

Pierwszym z zaproponowanych w pracy [14] algorytmów jest Doc2PL. W algorytmie tym ziarnem blokowania jest cały dokument XML. W przypadku odczytu dokumentu zakładana jest blokada do odczytu. W przypadku jakiejkolwiek modyfikacji zakładana jest blokada do modyfikacji na poziomie dokumentu. Rozwiązanie to jest szeroko stosowane w aktualnych implementacjach baz danych XML. Jego podstawową zaletą jest prostota implementacji. Doc2PL zakłada bowiem co najwyżej jedną blokadę na dokument dla pojedynczej transakcji.

Kolejne protokoły przedstawione w pracy [14] (Node2PL, NO2PL i OO2PL) wykorzystują zaproponowany przez autorów pojęciowy model dokumentu. Konstrukcja tego modelu oparta jest o drzewo DOM. W modelu tym (rys. 9), oprócz węzłów reprezentujących poszczególne elementy dokumentu XML, analogicznie jak w przypadku drzewa DOM, występują również wskaźniki odzwierciedlające występujące pomiędzy elementami zależności. Na rys. 9 przedstawiono przykładowy pojęciowy model dokumentu XML i jego omówienie.



Rys. 9. Model dokumentu dla Node2PL, NO2PL i OO2PL
Fig. 9. Document model for Node2PL, NO2PL and OO2PL

Węzły P, C1, C2, C3 reprezentują poszczególne elementy dokumentu XML. Węzły C1, C2, C3 są węzłami podrzędnymi dla węzła P, oznacza to, że reprezentują elementy wewnętrzne elementu P. Wskaźniki A i Z reprezentują odpowiednio zależności: pierwszy potomek i ostatni potomek. Natomiast wskaźniki R i L reprezentują odpowiednio zależności: następny, poprzedni. Wskaźniki odpowiadają operacjom DOM. Rozważmy dla przykładu dokument XML przedstawiony na rys. 10.

```
<książka>
  <tytuł>XML</>
  <rok>2000</>
  <autor>Smith</>
</>
```

Rys. 10. Przykładowy dokument XML
Fig. 10. Example of XML document

Wówczas P oznacza w modelu pojęciowym element `książka`, a C1, C2, C3 odpowiednio elementy: `tytuł`, `rok` i `autor`.

4.1.1.2. Algorytm Node2PL

W tym algorytmie ziarnem blokowania jest pojedynczy węzeł przedstawionego powyżej modelu dokumentu. W przypadku operacji odczytu (modyfikacji) zakładana jest blokada do odczytu (modyfikacji) na węźle nadrzędnym. Przykładowo, odczyt węzła C1 pociąga za sobą założenie blokady T węzła P. W przypadku modyfikacji węzła C2 lub dodawania nowego węzła C4 zakładana jest blokada M na węźle P. Jego współbieżność w porównaniu do algorytmu Doc2PL jest większa. Wadą tego algorytmu jest ograniczenie stopnia współbieżności w przypadku współbieżnego dostępu dwóch transakcji do węzłów, które mają tego samego poprzednika. Przykładowo rozważmy dwie transakcje T1 i T2. Transakcja T1 wstawia węzeł C1' pomiędzy węzły C1 i C2. Tymczasem transakcja T2 odczytuje zawartość węzła C3. Ze względu na blokadę węzła P, założoną przez transakcję T1, wykonywanie transakcji T2 zostanie zawieszone – niekompatybilność blokad T i M na węźle P. Rozwiązaniem tego problemu jest propozycja algorytmu NO2PL.

4.1.1.3. Algorytm NO2PL

W przypadku algorytmu NO2PL ziarnem blokady, podobnie jak w Node2PL, jest pojedynczy węzeł modelu dokumentu. W przeciwieństwie do algorytmu Node2PL algorytm NO2PL zakłada blokady na tych węzłach, których wskaźniki były przeglądane lub modyfikowane. Przykładowo, w odniesieniu do przedstawionego wyżej scenariusza współbieżnej realizacji transakcji T1 i T2 zgodnie z algorytmem NO2PL transakcja T1, wstawiająca węzeł C1' pomiędzy węzły C1 i C2, zakłada blokadę T na wierzchołku P oraz blokady M na węzłach C1 i C2. Wynika to z konieczności odczytu wskaźnika A dla węzła P oraz zmodyfikowania wskaźnika R dla węzła C1 i wskaźnika L dla węzła C2. Transakcja T2, odczytująca zawartość C3, zakłada blokadę T na węźle P. Liczba blokad NO2PL jest większa od liczby

blokad zakładanych przez algorytm Node2PL, gdyż ten drugi nigdy nie zakłada blokad na liściach dokumentu.

4.1.1.4. Algorytm OO2PL

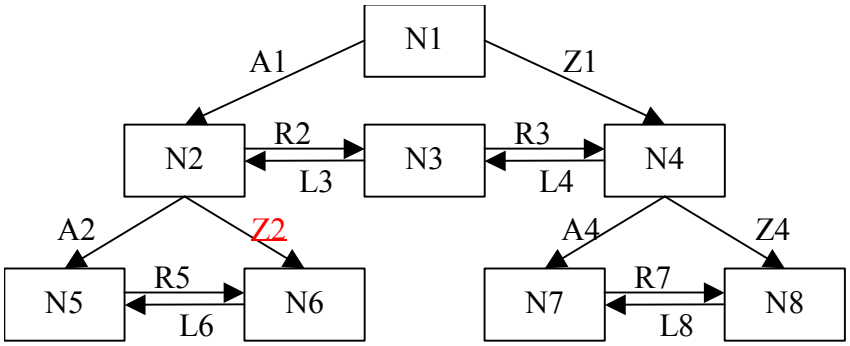
Najbardziej złożonym algorytmem, bazującym na przedstawionym pojęciowym modelu dokumentu, a jednocześnie gwarantującym największy stopień współbieżności, jest algorytm OO2PL. Ziarnem blokowania w OO2PL jest pojedynczy węzeł. Blokady T i M w algorytmie OO2PL zostały rozbite na cztery podtypy w zależności od odczytywanego lub modyfikowanego wskaźnika (A, Z, L, R). Stąd rozróżniamy następujące typy blokad: TA, TZ, TL, TR, MA, MZ, ML, MR. W algorytmie przed wykonaniem każdej operacji konieczne jest założenie odpowiednich blokad uzależnionych od wykorzystywanych wskaźników do poruszania się po dokumencie lub modyfikowania dokumentu. Tabela 4 przedstawia tabelę kompatybilności algorytmu OO2PL.

Tabela 4

Tabela kompatybilności metod dla OO2PL

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

Algorytm OO2PL różni się od przedstawionych wcześniej algorytmów większą ilością zakładanych blokad, zapewniających jednocześnie większy stopień współbieżności wykonywanych transakcji. Dla ilustracji przedstawmy scenariusz współbieżnej realizacji transakcji T1 i T2 ,operujących na przykładowym modelu dokumentu przedstawionym na rys. 11.



Rys. 11. Model przykładowego dokumentu
Fig. 11. Model of sample document

Dana jest transakcja T1 usuwająca węzeł N3. Natomiast transakcja T2 chce odczytać węzeł N7. Załóżmy, że T1 realizuje dostęp do węzła N3, przechodząc przez wskaźniki A1 i R2. Natomiast transakcja T2 realizuje dostęp do wierzchołka N7 przez wskaźniki Z1 i A4. W przypadku algorytmu Doc2PL transakcja T1 zakłada blokadę M na cały dokument. Stąd

transakcja T2 zostaje zablokowana ze względu na niekompatybilność blokad. W przypadku algorytmu Node2PL transakcja T1 zakłada blokadę M na węzeł N1, który jest nadrzędny w stosunku do usuwanego. Natomiast transakcja T2, przechodząc do węzła N7, musi założyć blokady T na węzły N1 i N4. Stąd, ze względu na niekompatybilność blokad T i M, na wierzchołku N1 transakcja T2 jest zablokowana. W przypadku algorytmu NO2PL transakcja T1 wymaga założenia blokad T na węźle N1 oraz M na węzłach N2 i N4 ze względu na konieczność modyfikacji wskaźników R2 i L4 dla węzłów N2 i N4. Transakcja T2 wymaga założenia blokad T na węźle N1 oraz blokady T na węźle N4. Pierwsza z blokad jest kompatybilna z blokadą T założoną na węźle N1 przez transakcję T1, druga natomiast jest blokadą konfliktową z blokadą M założoną na węźle N4 przez transakcję T1. Stąd transakcja T2 wstrzymuje swoje działanie, aż do czasu zakończenia się transakcji T1. W przypadku algorytmu OO2PL transakcja T1 wymaga założenia blokad TA na węźle N1, MR na węźle N2 oraz ML na węźle N4. Transakcja T2 wymaga założenia blokady TZ na węźle N1 i blokady TA na węźle N4. Obie blokady zakładane przez transakcję T2 są kompatybilne z blokadami założonymi przez transakcje T1, wynika to z tabeli kompatybilności blokad (tabela 4).

Opisane powyżej algorytmy pozwalają na kontrolę współbieżności w zakresie zmian struktury dokumentów. Nie obejmują swym zakresem zmiany zawartości dokumentu, która nie ma wpływu na strukturę np. zmiany zawartości pojedynczego prostego elementu. Dlatego też autorzy algorytmów rozszerzają je o dodatkowe blokady S i X zakładane odpowiednio przy odczycie i modyfikacji zawartości pojedynczych węzłów (elementów). Uwzględniając powyższe rozszerzenie, tabela kompatybilności blokad dla algorytmów Doc2PL, Node2PL i NO2PL wygląda jak tabela 5.

Tabela 5

Tabela kompatybilności blokad dla protokołów Doc2PL, Node2PL, NO2PL

	S	X	T	M
S	+	–	+	–
X	–	–	+	–
T	+	+	+	–
M	–	–	–	–

Generalną zasadą jest to, że nowo wprowadzone blokady S i X są kompatybilne z blokadami T oraz niekompatybilne z blokadami M. Taka sama zasada dotyczy również algorytmu OO2PL, w którym blokady Tx są kompatybilne z blokadami S i X, natomiast blokady Mx są z nimi w konflikcie. Kompatybilność blokad Tx wynika z faktu, iż przejście przez modyfikowane węzły nie ma wpływu na ich zawartość. Niekompatybilność blokad Mx wynika natomiast z faktu, iż transakcja, która chce założyć S i X, musi założyć niekompatybilne z blokadami Mx blokady typu Tx. Blokady Tx są konieczne, aby uzyskać dostęp do odczytywanych lub modyfikowanych węzłów.

Przedstawione powyżej algorytmy zakładają, że dostęp do danego węzła (elementu dokumentu) wymaga przejścia przez ścieżkę, łączącą dany węzeł z korzeniem dokumentu.

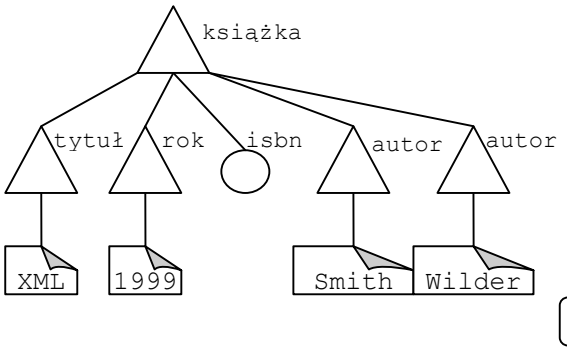
W rzeczywistości dostęp do określonego elementu można uzyskać również bezpośrednio np. za pomocą jego identyfikatora. Przykładem takiego dostępu niech będzie metoda interfejsu DOM `getElementById`. Załóżmy dla ilustracji, że transakcja T1 usuwa węzeł wraz z jego elementami podrzędnymi. Natomiast transakcja T2 realizuje dostęp do elementów podrzędnych usuwanego węzła z wykorzystaniem ich identyfikatorów. W takim przypadku przedstawione wyżej algorytmy nie gwarantują poprawnego rozwiązania zaistniałej sytuacji. Dlatego autorzy proponują rozszerzenie zakładanego zbioru blokad. Dla każdego dokumentu przechowywany jest zbiór ID – zbiór identyfikatorów elementów w nim zawartych. Wprowadzone zostają dodatkowe blokady współdzielone IDS i wyłączne IDX zakładane na elementach zbioru ID. Za każdym razem, gdy transakcja nawiguje za pomocą identyfikatorów do węzła z określonym ID, zakładana jest blokada IDS na tym identyfikatorze. W momencie, gdy węzeł jest usuwany, oprócz standardowych blokad omówionych wcześniej, wymagane jest także założenie blokad IDX na identyfikatorach wszystkich jego węzłów podrzędnych.

4.1.2. Algorytm blokowania drzewa taDOM

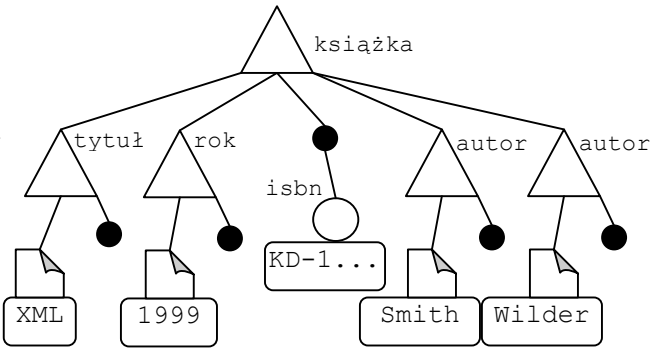
W roku 2003 Haustein i Härder zaproponowali algorytm blokowania taDOM. Jest to algorytm, który, podobnie jak wcześniej omówione propozycje, zakłada dostęp do dokumentów w bazie danych XML za pomocą standardu DOM API [12]. Autorzy na potrzeby algorytmu zaproponowali nowy pojęciowy model dokumentu, będący rozszerzeniem drzewa DOM, pod nazwą taDOM. Zanim przejdziemy do przedstawienia algorytmu, krótko omówimy zaproponowany model taDOM.

4.1.2.1. Drzewo taDOM

Drzewo taDOM jest rozszerzeniem drzewa DOM. Węzły atrybutów, które standardowo są bezpośrednimi następnikami węzła elementu, w przypadku drzewa taDOM mają jeden wspólny element, grupujący tzw. korzeń atrybutów. Korzeń atrybutów jest dołączony do węzła elementu niezależnie od tego, czy dany element posiada atrybuty czy też nie. Dodatkową modyfikacją jest dodanie do każdego węzła tekstowego (jak i atrybutu) węzła podrzędnego, określanego mianem węzła, ciągu znaków. Dla ilustracji na rys. 12 przedstawiono drzewo DOM dla dokumentu z rys. 1. Na rys. 13 przedstawiono drzewo taDOM z wyróżnionymi elementami dodatkowymi. Na rysunkach pominięto węzeł dokumentu.



Rys. 12. Drzewo DOM dokumentu
Fig. 12. DOM Tree of document



Rys. 13. Drzewo taDOM dokumentu
Fig. 13. taDOM Tree of document

Symbole zastosowane na rysunkach 12 i 13 oznaczają typy węzła:

- ○ – atrybutu,
- △ – elementu,
- ▭ – ciągu znaków – nowy w taDOM,
- ● – korzeń atrybutów – nowy w taDOM,
-  – tekstowy,
- ▲ – dokumentu.

Dodatkowe elementy w drzewie taDOM zostały wprowadzone dla potrzeb mechanizmu synchronizacji dostępu i nie są widoczne dla użytkownika poruszającego się po dokumencie. Znaczenie dodatkowych typów węzłów jest następujące. Korzeń atrybutów jest wykorzystywany wówczas, gdy transakcja wykonuje operację pobrania listy atrybutów elementów. Węzły ciągu znaków są wykorzystywane podczas modyfikacji i odczytu zawartości odpowiednich węzłów tekstowych.

4.1.2.2. Algorytm blokowania taDOM

Idea algorytmu opiera się na modelu dokumentu taDOM, podziale operacji wykorzystywanych w standardzie DOM na trzy podstawowe grupy: operacje nawigacyjne, operacje modyfikujące i operacje zapytań, oraz wprowadzeniu trzech rodzajów blokad: blokad węzłów, blokad nawigacyjnych i blokad logicznych.

Znaczenie poszczególnych rodzajów blokad jest następujące. Podczas wykonywania operacji na węzłach drzewa taDOM, każda transakcja zakłada na nich blokady. Blokady te są zależne od typu wykonywanych operacji i nazywane są blokadami węzłów. Jeśli transakcja wykonuje sekwencję operacji nawigacyjnych, przechodząc przez określoną sekwencję węzłów, to po ponownym wykonaniu tej samej sekwencji operacji otrzymana sekwencja węzłów musi być identyczna. Aby to zapewnić, wprowadzone są blokady nawigacyjne. Jeśli transakcja odczytuje za pomocą metody `getElementByTagName` zbiór węzłów posiadających określoną nazwę znacznika, to ponowne wykonanie tej samej metody musi dać w efekcie taki

sam zbiór węzłów. Oznacza to, że w ramach dokumentu nie mogą zostać utworzone, w tym samym czasie, elementy o nazwie znacznika użytej przy wywołaniu tej metody. W tym celu wykorzystywane są blokady logiczne.

4.1.2.3. Blokady węzłów

Podczas przeglądania lub modyfikacji dokumentu XML algorytm taDOM wymaga założenia odpowiedniej blokady na odwiedzanym węźle. Węzeł, do którego transakcja uzyskała dostęp, jest określany mianem węzła roboczego. Dostęp do węzła uwarunkowany jest założeniem blokad na węźle roboczym oraz na węzłach znajdujących się na ścieżce pomiędzy węzłem roboczym a korzeniem dokumentu. Tabela 6 przedstawia tabelę kompatybilności blokad węzłów.

Tabela 6

Tabela kompatybilności dla blokad węzłów

		Blokady istniejące						
		IX	NR	CX	LR	SR	U	X
Blokady żądane	IX	+	+	+	+	–	–	–
	NR	+	+	+	+	+	–	–
	CX	+	+	+	–	–	–	–
	LR	+	+	–	+	+	–	–
	SR	–	+	–	+	+	–	–
	U	+	+	+	+	+	–	–
	X	–	–	–	–	–	–	–

Znaczenie poszczególnych typów blokad jest następujące:

NR – odczyt węzła; podczas odczytu węzła są zakładane blokady NR, także na wszystkie jego węzły nadrzędne.

LR – odczyt poziomu; blokada zakładana na dany poziom węzłów (np. dla metody `getChildNodes`).

SR – odczyt poddrzewa; blokada zakładana przy dostępie do poddrzewa węzła roboczego.

X – modyfikacja węzła; blokada zakładana przy modyfikacji węzła.

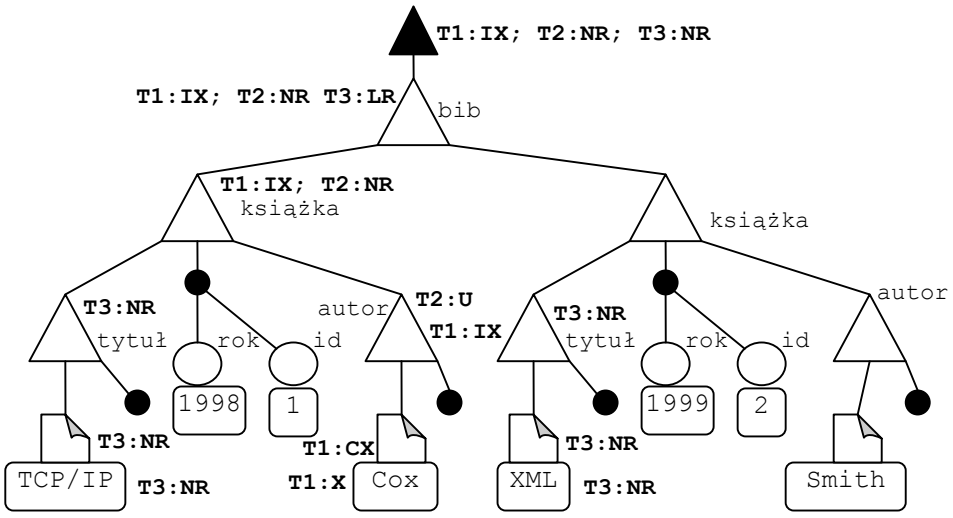
CX – modyfikacja węzła potomnego; zakładana w przypadku, gdy modyfikowany jest węzeł potomny.

IX – modyfikacja następnika; zakładana na węzeł w przypadku, gdy modyfikowany jest jeden z jego następników, wyłączając z tego węzeł potomny.

U – potencjalna modyfikacja węzła; zakładana przy odczycie węzła z deklaracją jego późniejszej modyfikacji.

Rozważmy następujący przykład współbieżnej realizacji trzech transakcji T1, T2 i T3. Załóżmy, że transakcja T1 modyfikuje zawartość elementu `autor` pierwszej książki (`Cox`). Modyfikacja ta wymaga założenia blokady X na węźle ciągu znaków, a także CX na węźle nadrzędnym i IX na wszystkich węzłach poprzedzających węzeł nadrzędny. Załóżmy, że transakcja T2 współbieżnie usuwa węzeł autora zawierający ciąg znaków `Cox`. Realizacja tego żądania wymaga założenia blokad X na węźle `autor` oraz blokad CX na węźle nadrzęd-

nym i IX na węzłach poprzedzających węzeł nadrzędny. Żądanie to zostanie odrzucone ze względu na niekompatybilność blokady X na węźle `autor` z blokadą IX transakcji T1. W efekcie powoduje to założenie blokady U na węźle `autor` oraz blokad NR na węzłach nadrzędnych. Uniemożliwi to późniejszym transakcjom odczyt tego węzła. Załóżmy dodatkowo, że transakcja T3 odczytuje listę tytułów książek. Żądanie to wymaga blokady LR na węźle `bib` w celu uzyskania dostępu do wszystkich węzłów podrzędnych. Zastosowanie blokady LR eliminuje konieczność zakładania blokad na każdy węzeł podrzędny. Następnie, dla każdego węzła `książka`, zakładane są blokady NR na ścieżce węzłów (elementów) prowadzących do węzła `tytuł`. Rysunek 14 ilustruje blokady zakładane przez transakcje T1, T2 i T3.

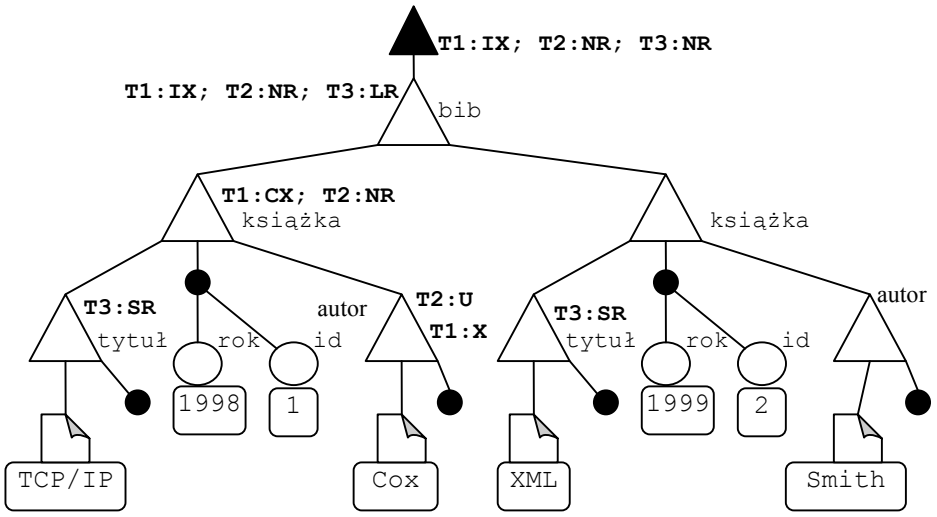


Rys. 14. Blokady węzłów założone przez transakcje T1, T2 i T3
Fig. 14. Node locks set by T1, T2 and T3 transactions

4.1.2.4. Ziarnistość i eskalacja blokad

Jak łatwo zauważyć, przedstawiony mechanizm charakteryzuje się dużą eskalacją blokad. Może to prowadzić do znacznych narzutów czasowych związanych z ich obsługą przez zarządcę blokad. Stąd, autorzy algorytmu proponują strojenie ziarnistości blokad węzłów i ich eskalacji. Strojenie ziarnistości blokad jest realizowane za pomocą parametru określającego dopuszczalną głębokość blokady. Określa on maksymalną liczbę poziomów, począwszy od korzenia dokumentu, na których dopuszcza się wystąpienie blokady. Jeżeli transakcja wykonuje operacje na węzłach znajdujących się poniżej parametru określającego dopuszczalną głębokość blokad, wówczas nie zakłada ona blokad na tych węzłach. Zamiast tego odpowiednie blokady, uzależnione od typu wykonywanych operacji, zakładane są na węźle nadrzędnym, znajdującym się na głębokości określonej za pomocą omawianego parametru. W przypadku operacji odczytu zakładana jest blokada SR, a przy operacjach modyfikacji blokada X. Rozwiązanie to nie tylko zmniejsza liczbę zakładanych blokad, ale także pozwala transakcji poruszać się po fragmencie dokumentu bez konieczności zakładania całego szeregu

blokad. Drzewo taDOM, dla rozważanego przykładu i wartości parametru głębokości blokady równego 3, zostało przedstawione na rys. 15.



Rys. 15. Blokady węzłów założone przez transakcje T1, T2 i T3 dla parametru głębokości równego 3

Fig. 15. Node locks set by T1, T2 and T3 transactions for parameter lock depth = 3

Problem nadmiernej eskalacji jest rozwiązywany następująco. Wprowadzono dwa parametry: parametr progu eskalacji i parametr głębokości eskalacji. Zarządca blokad przegląda drzewo taDOM w zadanych interwałach. W przypadku znalezienia poddrzewa z liczbą blokad węzłów przekraczają zadany próg eskalacji dla pojedynczej transakcji, istniejące blokady są zamieniane na odpowiadającą im blokadę zakładaną na korzeniu poddrzewa. Wyjątkiem jest sytuacja, gdy blokada docelowa jest w konflikcie z blokadą już istniejącą. Głębokość eskalacji definiuje maksymalną wysokość poddrzewa (licząc od liści), jaka ma być analizowana przez zarządcę blokad.

Stosowanie powyższych rozwiązań minimalizuje liczbę blokad węzłów i narzut związany z ich zarządzaniem kosztem zmniejszenia współbieżności równolegle wykonywanych transakcji.

4.1.2.5. Blokady nawigacyjne

Mechanizm blokad węzłów nie gwarantuje, że w trakcje nawigacji po dokumencie realizowanej przez transakcje, inne transakcje nie zmienią struktury dokumentu. W związku z tym wprowadzono pojęcie wirtualnych krawędzi nawigacyjnych oraz blokady nawigacyjne. W drzewie taDOM rozróżniamy następujące krawędzie nawigacyjne: pomiędzy sąsiadującymi węzłami posiadającymi wspólny węzeł nadrzędny, pomiędzy węzłem nadrzędnym a pierwszym jego węzłem podrzędnym, pomiędzy węzłem nadrzędnym a jego ostatnim węzłem podrzędnym. Podczas nawigacji po dokumencie, wykorzystującej krawędzie nawigacyjne, transakcja wymusza zakładanie blokad na każdej z nich. Autorzy proponują następujące typy blokad:

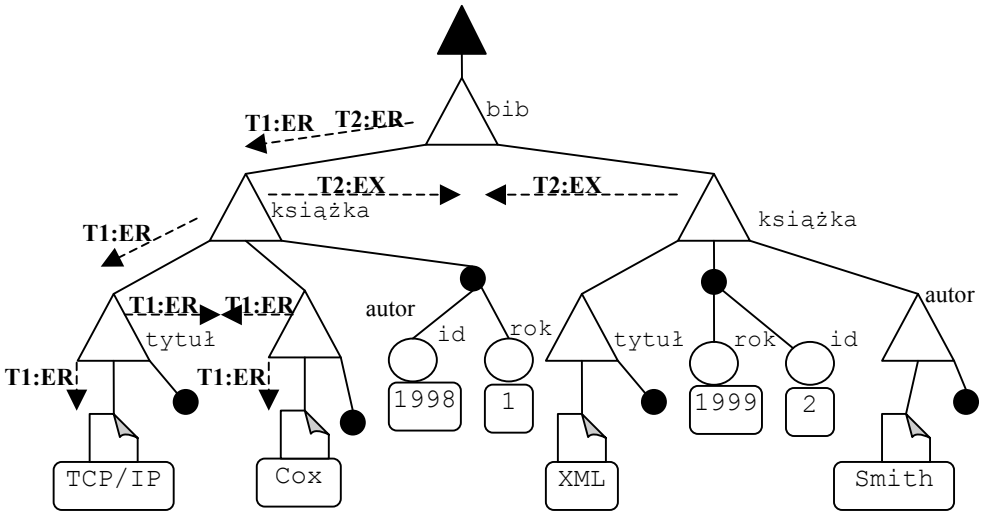
ER – zakładana podczas wykorzystania krawędzi do odczytu (np. dla metod DOM API: `nextSiblingEdge`, `firstChildEdge` itp.).
EX – zakładana przed modyfikacją krawędzi (np. podczas usunięcia węzła lub dodania węzła – wtedy, gdy docelowy węzeł krawędzi zostaje zmieniony).
EU – intencjonalna blokada zakładana jako deklaracja późniejszej modyfikacji krawędzi – analogiczna do blokady węzłów U.

Tabela 7 przedstawia tabelę kompatybilności blokad nawigacyjnych.

Tabela 7
Tabela kompatybilności blokad nawigacyjnych

	ER	EU	EX
ER	+	–	–
EU	+	–	–
EX	–	–	–

Rozważmy następujący przykład współbieżnej realizacji trzech transakcji T1, T2 i T3. Transakcja T1 odczytuje zawartość elementów `tytuł` i `autor` pierwszej z książek, natomiast transakcja T2 zamierza dodać element bezpośrednio za pierwszym elementem `książka`. Realizacja obu transakcji jest następująca. Transakcja T1 odczytuje węzeł `bib`, a następnie trzykrotnie przechodząc do pierwszego elementu podrzędnego (za pomocą metody `getFirstChild`) dociera do węzła ciągu znaków, opisującego tytuł pierwszej książki. Kolejną operacją wykonywaną przez transakcje T1 jest przejście do węzła `autor`, następującego bezpośrednio po węźle `tytuł` (za pomocą metody `getNextSibling`), oraz odczytanie podrzędnego węzła ciągu znaków, przechowującego nazwę autora (za pomocą `getFirstChild`).



Rys. 16. Blokady nawigacyjne założone przez transakcje T1 i T2
Fig. 16. Navigational locks set by T1 and T2 transactions

Transakcja T2 odczytuje węzeł `bib`, a następnie przechodzi do elementu `autor` – pierwszego elementu podrzędnego (za pomocą metody `getFirstChild`). Następnie wykonuje operację dodania elementu za elementem `autor` (za pomocą metody `insertAfter`). Blokady

nawigacyjne założone przez transakcje T1 i T2 na wirtualnych krawędziach nawigacyjnych zostały przedstawione na rys. 16.

4.1.2.6. Blokady logiczne

Niestety blokady węzłów, uzupełnione o blokady nawigacyjne, w dalszym ciągu nie gwarantują poprawności współbieżnego dostępu do dokumentów XML. Dla przykładu, w standardzie DOM API istnieje możliwość wykonywania operacji znajdujących wszystkie elementy posiadające określoną wartość znacznika (metoda `getElementByTagName`) lub posiadające określoną wartość atrybutu ID (metoda `getElementById`). Blokady węzłów oraz blokady nawigacyjne nie gwarantują niezmienności odpowiedzi na powyższe typy zapytań. W związku z tym wprowadzono blokady logiczne. W odróżnieniu do wcześniejszych dwóch rodzajów blokad, blokady logiczne nie są związane z określonymi elementami (węzłami lub krawędziami) modelu danych. Blokady logiczne dotyczą całego dokumentu, na którym wspomniane operacje są wykonywane. Typy blokad logicznych są następujące: R – blokada do odczytu, U – blokada intencjonalna do modyfikacji, X – blokada do modyfikacji. Tabela 8 przedstawia tabelę kompatybilności blokad logicznych.

Tabela 8
Tabela kompatybilności blokad logicznych

	R	U	X
R	+	-	-
U	+	-	-
X	-	-	-

Do zarządzania i przechowywania blokad logicznych są wykorzystywane trzy specjalne tabele: LocksTagnameQuery, LocksAttributeQuery i LocksIDQuery. Ich znaczenie i wykorzystanie jest następujące.

Wykonanie metody pobierającej elementy, które posiadają określoną wartość znacznika (`getElementByTagName`), wymaga założenia blokady R i zapisania odpowiedniej informacji w tabeli LocksTagnameQuery. Informacja ta składa się z identyfikatora transakcji, typu blokady, identyfikatora węzła, na rzecz którego metoda została wykonana oraz nazwy znacznika.

Tabela 9
Tabela LocksTagnameQuery

ID transakcji	typ blokady	ID węzła	nazwa znacznika
…	R/U/X	…	…

Dodanie nowego węzła do dokumentu wymaga założenia blokady typu X dla nazwy znacznika nowego węzła, czyli umieszczenia odpowiedniego wpisu w tabeli LocksTagnameQuery. Niekompatybilność blokad X i R gwarantuje, że lista węzłów uzyskiwana za pomocą metody `getElementByTagName` będzie niezmienna. Usunięcie węzła nie wymaga założenia blokad logicznych. Niekompatybilność operacji usunięcia węzła z innymi operacjami weryfikowana jest za pomocą blokad węzłów.

Wykonanie metody sprawdzającej czy dany węzeł posiada atrybuty czy też nie (`hasAttributes`) wymaga założenia blokady R i zapisania tego faktu w tabeli LocksAttributeQuery. Informacja o blokadzie składa się z identyfikatora transakcji, typu blokady, identyfikatora węzła, na rzecz którego metoda została wykonana oraz ewentualnej nazwy atrybutu.

Tabela 10

Tabela LocksAttributeQuery

ID transakcji	typ blokady	ID węzła	nazwa atrybutu
…	R/U/X	…	…

Wstawienie nowego oraz usunięcie istniejącego atrybutu wymaga umieszczenia blokady X w tej samej tabeli. Niekompatybilność blokad X i R gwarantuje powtarzalność odpowiedzi uzyskanej w wyniku działania metody `hasAttributes`.

Wykonanie metody wyszukującej element o określonym identyfikatorze (`getElementById`) wymaga założenia blokady R i zapisania tego faktu w tabeli LocksIDQuery. Informacja o blokadzie składa się z identyfikatora transakcji, typu blokady oraz identyfikatora elementu.

Tabela 11

Tabela LocksIDQuery

ID transakcji	blokada	ID
…	R/U/X	…

Wstawienie nowego elementu lub usunięcie istniejącego wymaga blokady X, dotyczącej identyfikatora wstawianego lub usuwanego elementu. Niekompatybilność blokad X i R gwarantuje powtarzalność odpowiedzi uzyskanej w wyniku działania metody `getElementById`.

Rozważmy przykładową realizację czterech transakcji T1, T2, T3 i T4. Transakcja T1 poszukuje elementów o identyfikatorze równym 4, a następnie poszukuje elementów o identyfikatorze równym 3. Transakcja T2 poszukuje elementów `tytuł` dla węzła `książka` o identyfikatorze 13. Tabele blokad logicznych, po wykonaniu powyższych operacji, zostały przedstawione w tabelach 12 i 13. Następnie, transakcja T3 usiłuje wstawić nowy węzeł o identyfikatorze 4 – wymaga to założenia blokady X na identyfikatorze 4. Z uwagi na niekompatybilną blokadę transakcji T1 operacja zostaje wstrzymana. Transakcja T4, która zamierza dodać nowy element `tytuł` do elementu `książka` o identyfikatorze 13, także zostaje wstrzymana ze względu na istniejącą, niekompatybilną blokadę R założoną przez transakcję T2.

Tabela 12

Tabela LocksIDQuery po wykonaniu operacji transakcji T1 i T2

TAID	blokada	ID
1	R	4
1	R	3

Tabela 13

Tabela LocksTagnameQuery po wykonaniu operacji transakcji T1 i T2

TAID	blokada	ID węzła	znacznik
2	R	13	tytuł

Przedstawiony w pracy [12] algorytm taDOM jest algorytmem opartym na systemie blokad, ściśle powiązanym z interfejsem DOM API. W zależności od wykorzystywanej metody interfejsu DOM API, autorzy proponują zakładanie odpowiedniego zbioru blokad, gwarantujących poprawną realizację współbieżnego dostępu do dokumentów XML. Blokady wykorzystywane przez algorytm taDOM są zakładane na modelu danych, który jest rozszerzoną wersją drzewa DOM, zwanego drzewem taDOM. Algorytm dzieli blokady na trzy rodzaje: blokady węzłów, blokady nawigacyjne oraz blokady logiczne. Operacje na węzłach dokumentu wymagają zakładania blokad na odpowiednich węzłach drzewa taDOM. Jeśli transakcja wykonuje sekwencję operacji nawigacyjnych zakłada również blokady nawigacyjne. Natomiast w przypadkach bezpośredniego dostępu do węzłów dokumentu, w oparciu o nazwy znaczników bądź ich identyfikatory, zakładane są blokady logiczne. Dla każdego z typów blokad ustalone są tabele kompatybilności. Celem blokad jest zapewnienie uszeregowania transakcji i wyeliminowanie anomalii takich jak niepowtarzalne odczyty lub fantomy. Ponadto, aby ułatwić zarządzanie dużą liczbą blokad, autorzy proponują wykorzystanie metod minimalizujących liczbę blokad węzłów i narzut związany z ich zarządzaniem kosztem zmniejszenia stopnia współbieżności równolegle wykonywanych transakcji.

4.1.3. Algorytmy porządkowania transakcji wg znaczników czasowych

W metodach blokowania porządek uszeregowania transakcji wynika, najogólniej mówiąc, z kolejności uzyskiwania blokad przez transakcje. W przypadku metody porządkowania transakcji wg znaczników czasowych (T/O), porządek uszeregowania transakcji jest zdefiniowany a priori i wynika z porządku znaczników czasowych transakcji. W metodzie T/O, z każdą transakcją T jest związany jednoznaczny (unikalny) znacznik czasowy transakcji oznaczany przez TS(T). W przypadku, gdy kolejność dostępu do danych nie odpowiada porządkowi znaczników czasowych transakcji, wówczas transakcja naruszająca ten porządek jest wycofywana i restartowana ponownie z nowym znacznikiem czasowym. W literaturze zaproponowano dwa algorytmy zarządzania współbieżnym dostępem do bazy danych dokumentów XML – algorytm XTO i algorytm XCO – oparte na metodzie T/O [13].

4.1.3.1. XTO

Idea algorytmu XTO zaproponowanego w pracy [13] oparta jest na dwóch koncepcjach: utrzymywaniu wielu wersji dokumentu i tabeli zawierającej sposoby rozwiązywania sytuacji konfliktowych, występujących w przypadku współbieżnego dostępu transakcji do tego samego węzła dokumentu.

Utrzymywanie wielu wersji dokumentu jest realizowane następująco. Z każdym węzłem dokumentu związany jest status oraz dwie etykiety czasowe: etykieta czasowa wstawienia oraz etykieta czasowa usunięcia. Etykiety czasowe przechowują informacje o znacznikach czasowych transakcji, które dokonały odpowiednio wstawienia lub usunięcia węzła. W przy-

padku operacji usunięcia węzła otrzymuje on status usunięty i dodatkowo zapisywana jest etykieta czasowa usunięcia, sam węzeł nie jest usuwany z dokumentu. W przypadku wstawienia węzła otrzymuje on status wstawiony i zapamiętywana jest etykieta czasowa jego wstawienia. Po zakończeniu transakcji, utworzone przez transakcje etykiety czasowe są usuwane. Dodatkowo usuwane są także węzły usunięte. Dzięki etykietom czasowym operacji wstawiania i usuwania jest możliwe odtworzenie dokumentu z dowolnego momentu w czasie pomiędzy pierwszą i ostatnią niezatwierdzoną modyfikacją. Ponadto, dla każdego węzła utrzymywana jest etykieta odczytu. Etykiety odczytu są konieczne do weryfikacji sytuacji konfliktowych. Ich utrzymywanie polega na ich aktualizacji w przypadku, gdy węzeł jest odczytywany przez transakcję posiadającą późniejszy znacznik czasowy niż znacznik umieszczony na etykiecie czasowej odczytu. W ten sposób, dla każdego węzła pamiętany jest znacznik czasowy najnowszej transakcji, która go odczytywała.

Tabela rozwiązań sytuacji konfliktowych stanowi podstawowy mechanizm wykorzystywany przez algorytm zarządzania współbieżnym dostępem do danych. Tabela 14 przedstawia tabelę rozwiązań sytuacji konfliktowych. Op1 i op2 oznaczają odpowiednio operacje transakcji T1 i T2 realizowane na tym samym węźle, gdzie op1 oznacza operację wykonaną, natomiast op2 operację wykonywaną. Wyróżniamy 3 typy operacji wykonywanych na węzłach T – odczyt, D – usunięcie i I – wstawianie węzła.

Tabela 14

Tabela rozwiązań sytuacji konfliktowych w algorytmie XTO

op1	op2	TS(T1) < TS(T2)	TS(T2) < TS(T1)
T	T	brak konfliktu	brak konfliktu
D	T	ignoruj (kaskadowe wycofanie)	odczytaj
I	T	ignoruj (kaskadowe wycofanie)	ignoruj
T	D	zaznacz usunięcie	wycofaj T2
D	D	przypadek niemożliwy	wycofaj T2
I	D	blokuj T2 a potem zaznacz usunięcie	przypadek niemożliwy
T	I	przypadek niemożliwy	przypadek niemożliwy
D	I	przypadek niemożliwy	przypadek niemożliwy
I	I	przypadek niemożliwy	przypadek niemożliwy

Tabela rozwiązań sytuacji konfliktowych jest odpowiednikiem macierzy kompatybilności blokad w algorytmie blokowania i zawiera sposób rozwiązania sytuacji konfliktowych. Dla zilustrowania działania mechanizmu rozwiązywania sytuacji konfliktowych przeanalizujemy niektóre reguły przedstawione w tabeli 14.

TT – jeżeli obie operacje op1 i op2 są operacjami transakcji T1 i T2 przechodzącymi przez ten sam węzeł, to nie występuje problem konfliktu dostępu, niezależnie od znaczników czasowych transakcji T1 i T2.

TI, DI, II – jeżeli operacja op2 jest operacją wstawiającą węzeł do dokumentu, to op1, która jest operacją wcześniejszą, nie mogła wykonywać operacji na nieistniejącym jeszcze węźle.

DT – jeżeli operacja op1 usunęła węzeł, a zatem zostaje on zaznaczony jako usunięty znacznikiem czasowym, a następnie op2 próbuje go odczytać, to należy wówczas rozważyć dwie sytuacje. Jeżeli TS(T1) < TS(T2) (czyli znacznik czasowy T1 jest mniejszy niż znacznik czasowy T2 – T1 rozpoczęła się wcześniej niż T2), wówczas usunięty węzeł jest ignorowany przez operację op2. W przypadku, gdy TS(T2) < TS(T1) transakcja T2 wciąż może odczytać usunięty węzeł. W pierwszej sytuacji, w której usunięty węzeł zostaje zignorowany, jeśli transakcja T1 zostanie wycofana, wówczas transakcja T2 również będzie musiała się wycofać, wynika to faktu, że nie mogła zignorować węzła, który ostatecznie nie został usunięty.

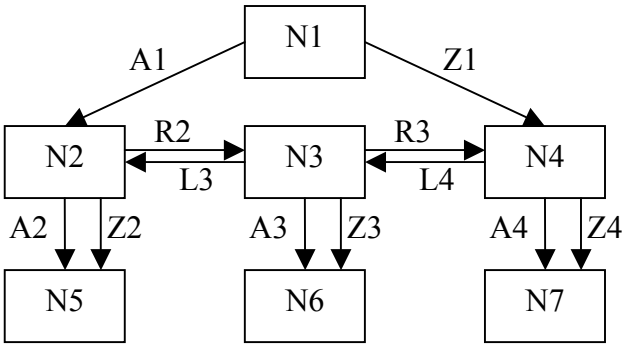
TD – jeżeli transakcja T2 chce usunąć węzeł odczytany przez starszą transakcję T1 (TS(T1) < TS(T2)), wówczas nie występuje problem konfliktu dostępu, T2 musi tylko dokonać odpowiednich zmian w etykietach czasowych węzła. Jeśli jednak T1 była młodsza, wówczas T2 musi zostać wycofana – transakcja T1, bowiem nie może odczytać węzła usuniętego przez starszą transakcję T2. Inną możliwością jest wycofanie transakcji T1. Wymaga to jednak posiadania informacji o wszystkich transakcjach młodszych niż T2, które odczytały usuwany węzeł i wycofania każdej z nich. Nie jest to możliwe, ponieważ na poziomie węzła jest przechowywany znacznik czasowy tylko najnowszej transakcji, która odczytywała węzeł. Rozwiązaniem w tym przypadku byłoby przechowywanie, dla każdego węzła, listy znaczników czasowych transakcji, które dokonały operacji odczytu.

Protokół XTO jest stosunkowo mało kosztowny – potrzebne są tylko dwa znaczniki czasowe i status dla każdego węzła. Dodatkowym kosztem byłoby zastosowanie list wykonanych operacji odczytów zgodnie z propozycją zawartą w regule TD. Jak wynika między innymi z reguły DT, algorytm XTO nie zapobiega kaskadowemu wycofywaniu transakcji. Transakcje muszą być zatwierdzane zgodnie ze znacznikami czasowymi, co może powodować ich buforowanie do czasu zatwierdzenia transakcji z mniejszym znacznikiem czasowym. Zaletą algorytmu XTO jest większy stopień współbieżności niż w przypadku algorytmu OO2PL, który wśród algorytmów blokowania drzewa DOM charakteryzuje się największym stopniem współbieżności. Można to zauważyć na następującym przykładzie (rys. 17).

W przykładzie zastosowanie algorytmu OO2PL doprowadzi do zakleszczenia. Wynika to z faktu, iż transakcja T1 nie będzie mogła wykonać operacji (9) odczytu węzła N4 w wyniku wcześniejszej operacji (6) usunięcia tego węzła przez transakcję T2. Z drugiej strony transakcja T2 nie będzie mogła się zakończyć ze względu na niekompatybilność operacji (10) odczytu węzła N2 z operacją (5) usunięcia tego węzła przez transakcję T1.

Zastosowanie, w przypadku tych transakcji, algorytmu XTO pozwoli im zakończyć się z powodzeniem. Transakcja T1 poprzedza T2 (TS(T1) < TS(T2)), powoduje to, że przypadek operacji 9 i 6 dzięki regule DT zakończy się odczytem węzła N4 przez transakcję T1. Natomiast przypadek operacji 10 i 5 na podstawie tej samej reguły DT zakończy się zignorowaniem przez transakcję T2 węzła N2 usuniętego przez transakcję T1 i odczytem węzła N3.

	T1	T2
1	sd→N1	
2		sd→N1
3	nthP(1)→N2	
4		nthM(1)→N4
5	del→N2	
6		del→N4
7	sd→N1	
8		sd→N1
9	nthM(1)→N4	
10		nthP(1)→N3
11	nthP(1)→N7	
12		nthP(1)→N6



Rys. 17. Przykład realizacji dwóch transakcji T1 i T2
Fig. 17. Example of concurrent execution of both transactions T1 and T2

Algorytm XTO jest jednym z nielicznych algorytmów opartych na znacznikach czasowych stosowanych do zarządzania współbieżnym dostępem do dokumentów XML. Podstawową wadą tego algorytmu jest możliwość wystąpienia kaskadowego wycofywania transakcji oraz konieczność zatwierdzania transakcji zgodnie z kolejnością posiadanych przez nie znaczników czasowych.

4.1.4. Dynamiczne szeregowanie zatwierdzeń (XCO)

W celu rozwiązania problemu ściśle wyznaczonej przez znaczniki czasowe transakcji kolejności wykonywania operacji zatwierdzenia, autorzy pracy [13] proponują zastosowanie metody dynamicznego szeregowania zatwierdzeń (XCO). Propozycja ta, tak jak w przypadku algorytmu XTO, zakłada utrzymywanie wielu wersji dokumentu. Różnica w stosunku do XTO polega na tym, że zamiast przechowywania, na poziomie węzłów, znaczników czasowych transakcji, które dokonały operacji odczytu, wstawienia lub usunięcia węzła, są przechowywane referencje do tych transakcji. Dodatkowo, XCO utrzymuje graf reprezentujący dynamicznie tworzoną kolejność zatwierdzeń $<_{DCO}$. Jest on modyfikowany podczas wykonywania operacji przez poszczególne transakcje. Węzłami w grafie kolejności zatwierdzeń są transakcje ($T_1 \ldots T_n$), istniejące w systemie. Dodanie krawędzi (T_i,T_j), określającej kolejność zatwierdzania, ma miejsce w przypadku detekcji dwóch operacji konfliktowych, wykonywanych przez transakcje T_i, T_j, dla których kolejność w sensie $<_{DCO}$ nie została jeszcze określona (innymi słowy w grafie kolejności zatwierdzeń pomiędzy transakcjami T_i, T_j nie istnieje żadna krawędź). Po zakończeniu transakcji, z grafu kolejności zatwierdzeń usuwany jest węzeł reprezentujący transakcję oraz wszystkie krawędzie z nim związane. Oprócz grafu $<_{DCO}$, XCO utrzymuje graf przydziału zasobów i zależności wycofania. Zależności wycofania są wykorzystywane podczas operacji kaskadowego wycofywania transakcji, które w tym algorytmie również mogą mieć miejsce.

Podobnie jak w przypadku XTO, do określenia akcji podejmowanych przez algorytm zarządzania współbieżnym dostępem do danych jest wykorzystywana tabela rozwiązań sytuacji konfliktowych. Operacje op1 i op2 oznaczają odpowiednio operacje transakcji T1 i T2 realizowane na tym samym węźle, gdzie op1 oznacza operację wykonaną, natomiast op2 operację wykonywaną.

Tabela 15

Tabela rozwiązań sytuacji konfliktowych w algorytmie XCO dla określonej relacji $<_{DCO}$

op1	**op2**	**T1 $<_{DCO}$ T2**	**T2 $<_{DCO}$ T1**	**T1 $<>_{DCO}$ T2**
T	T	brak konfliktu		
D	T	ignoruj (kaskad. wycofanie)	odczytaj	tabela poniżej
I	T	odczytaj (kaskad. wycofanie)	ignoruj	
T	D	zaznacz usunięcie	wycofaj T2	
D	D	przypadek niemożliwy	wycofaj T2	przypadek niemożliwy
I	D	blokuj T2 a następnie zaznacz usunięcie	przypadek niemożliwy	przypadek niemożliwy
X	I	przypadek niemożliwy		

Sposób wyboru rozwiązania sytuacji konfliktowej zależny jest od tego czy w grafie kolejności zatwierdzeń pomiędzy węzłami reprezentującymi transakcje T1 i T2 istnieje już krawędź określająca relację $<_{DCO}$ czy też nie. Jeżeli krawędź pomiędzy transakcjami T1 i T2 istnieje, to w zależności od tego, w jaki sposób relacja $<_{DCO}$ szereguje obie transakcje, podejmowane są określone akcje zgodnie z tabelą rozwiązań sytuacji konfliktowych (tabela 15). Jeżeli krawędź w grafie kolejności zatwierdzeń, pomiędzy transakcjami T1 i T2, nie istnieje, czyli T1 $<>_{DCO}$ T2, wówczas, oprócz wykonania odpowiedniej akcji, algorytm nanosi na graf kolejności zatwierdzeń, krawędź definiującą relację $<_{DCO}$ pomiędzy transakcjami. Wykonywane jest to zgodnie z tabelą rozwiązań sytuacji konfliktowych (tabela 16). Dla przypadków DT oraz IT algorytm ma możliwość wyboru jednego z dwóch różnych wariantów. Autorzy za pomocą testów wyznaczyli wariant, który okazał się najlepszy (został on oznaczony w tabeli akcji za pomocą *)

Tabela 16

Tabela rozwiązań sytuacji konfliktowych w algorytmie XCO dla nieokreślonej relacji $<_{DCO}$

op1	**op2**	**akcja**	**dodanie krawędzi do $<_{DCO}$**
D	T	ignoruj* (kaskad. wycofanie)	T1 $<_{DCO}$ T2
		odczytaj (może prowadzić do DD)	T2 $<_{DCO}$ T1
I	T	odczytaj (może prowadzić do ID)	T1 $<_{DCO}$ T2
		ignoruj*	T2 $<_{DCO}$ T1
T	D	zaznacz usunięcie	T1 $<_{DCO}$ T2

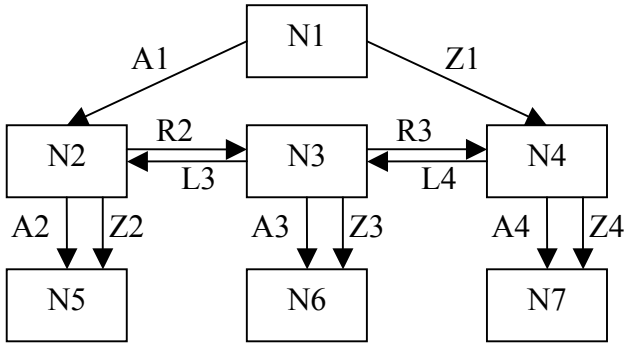
Podobnie jak w przypadku algorytmu XTO, przeanalizujemy poniżej kilka reguł wynikających z tabeli rozwiązań sytuacji konfliktowych algorytmu XCO.

DT – operacja usuwania węzła (op1) poprzedza operację odczytu węzła (op2). Jeżeli T1 poprzedza T2 w relacji $<_{DCO}$, co oznacza, że zgodnie z grafem kolejności zatwierdzeń T1

została uszeregowana przed T2, wówczas op2 ignoruje istnienie usuniętego węzła. Może to w konsekwencji prowadzić do kaskadowego wycofywania transakcji. Jeżeli bowiem transakcja T1 zostałaby wycofana, wówczas oznaczałoby to, że operacja op2 powinna odczytać dany węzeł. W konsekwencji transakcja T2 powinna zostać również wycofana. Jeżeli T2 poprzedza T1, wówczas zaznaczony do usunięcia węzeł zostanie przez operację op2 odczytany. Jeżeli natomiast relacja $<_{DCO}$ transakcji T1 i T2 nie została określona, wówczas, oprócz wykonania odpowiedniej akcji przez operację op2, należy zdefiniować relację $<_{DCO}$ pomiędzy transakcjami i umieścić odpowiednią krawędź w grafie kolejności zatwierdzeń. W tym przypadku możliwe są dwa scenariusze. Zgodnie z pierwszym scenariuszem op2 ignoruje węzeł, jednocześnie definiując relację pomiędzy transakcjami $T1 <_{DCO} T2$ – takie postępowanie grozi koniecznością kaskadowego wycofania transakcji. Drugi scenariusz zakłada, że $T2 <_{DCO} T1$, co powinno zostać odpowiednio odzwierciedlone w grafie kolejności zatwierdzeń, a następnie wykonana powinna zostać operacja odczytu węzła.

DD – operacja usunięcia węzła (op1) poprzedza inną operację usunięcia tego samego węzła (op2). Przypadek, gdy do takiej sytuacji dochodzi dla nieokreślonej relacji $<_{DCO}$ pomiędzy transakcjami T1 i T2 jest niemożliwy, gdyż wcześniej musiała mieć miejsce sytuacja DT, dla której relacja ta została określona zgodnie z wcześniej omówioną regułą DT. W zależności od tego jak została rozwiązana sytuacja DT, należy wybrać odpowiedni sposób postępowania. Jeżeli operacja transakcji T2, w sytuacji konfliktowej DT, zignorowała węzeł, to oznacza, że relacja $<_{DCO}$ pomiędzy transakcjami została zdefiniowana jako $T1 <_{DCO} T2$, w związku z tym przypadek DD nie jest możliwy. Jeżeli natomiast w sytuacji DT doszło do odczytu węzła przez transakcję T2, to odpowiednia krawędź w grafie kolejności zatwierdzeń definiuje, że $T2 <_{DCO} T1$. W takim przypadku należy wycofać transakcję T2, gdyż jej kontynuacja prowadziłaby do nieuszeregowania. Usunięcie węzła przez transakcję T1 nie powinno bowiem mieć miejsca, skoro transakcja T2, poprzedzająca ją w relacji $<_{DCO}$, dokonała jego usunięcia.

	T1	T2
1		sd→N1
2		nthM(1)→N1
3		del→N4
4	sd→N1	
5	nthP(1)→N2	
6		sd→N1
7		nthP(1)→N2
8		del→N2
9	sd→N1	
10	nthM(1)→N4	



Rys. 18. Przykład realizacji dwóch transakcji T1 i T2
Fig. 18. Example of concurrent execution of both transactions T1 and T2

Przykład przedstawiony na rys. 18 ilustruje działanie algorytmu XCO oraz jego zalety w stosunku do algorytmów XTO i OO2PL.

Zastosowanie algorytmu OO2PL doprowadzi w tym przypadku do zakleszczenia. Wynika to z faktu, iż operacje 5 i 8 znajdują się w konflikcie, podobnie jak 3 i 10. Zastosowanie algorytmu XTO powoduje, że w momencie wykonania operacji 8 (usunięcia przez transakcję T2 węzła N2) zachodzi konieczność wycofania transakcji T2. Dzieje się tak dlatego, że transakcja T1 nie może odczytać węzła N2, usuniętego przez transakcję T2 o mniejszym znaczniku czasowym. Dzięki wykorzystaniu XCO obie transakcje mogą bez przeszkód zostać zakończone. W momencie, gdy transakcja T2 wykonuje operację 8 mamy do czynienia z przypadkiem, kiedy należy ustalić porządek $<_{DCO}$ pomiędzy transakcjami, gdyż poprzednie operacje dotyczyły różnych węzłów. Na podstawie tabeli rozwiązań sytuacji konfliktowych zostaje ustalony porządek jako T1 $<_{DCO}$ T2. Ponadto usuwany węzeł jest odpowiednio oznaczany. Dzięki temu przy operacji 10 transakcja T1 będzie mogła bez przeszkód dokonać odczytu żądanego węzła. Jest to bowiem przypadek DT, gdzie operacja usunięcia węzła należy do transakcji T2, natomiast operacja odczytu tego samego węzła, do transakcji T1 – w sytuacji, gdy T1 $<_{DCO}$ T2 (w tabeli akcji T2 $<_{DCO}$ T1 – operacja usunięcia należała do transakcji T2 z naszego przykładu).

Algorytm XCO jest przykładem algorytmu opartego na szeregowaniu transakcji na podstawie dynamicznie tworzonego grafu kolejności zatwierdzeń. Nie wymaga on kolejności zatwierdzania transakcji zgodnej z ich znacznikami czasowymi, tak jak ma to miejsce w przypadku XTO. O kolejności zatwierdzania decyduje tutaj graf kolejności zatwierdzeń, który, w przypadku transakcji działających na różnych dokumentach lub różnych jego fragmentach, nie narzuca żadnego porządku zatwierdzania. Stopień współbieżności algorytmu jest większy niż w przypadku algorytmów OO2PL i XTO. Niestety, podobnie jak w przypadku XTO, podstawową wadą algorytmu jest możliwość wystąpienia kaskadowego wycofywania transakcji.

4.2. Algorytmy zarządzania współbieżnym dostępem oparte na standardzie XPath

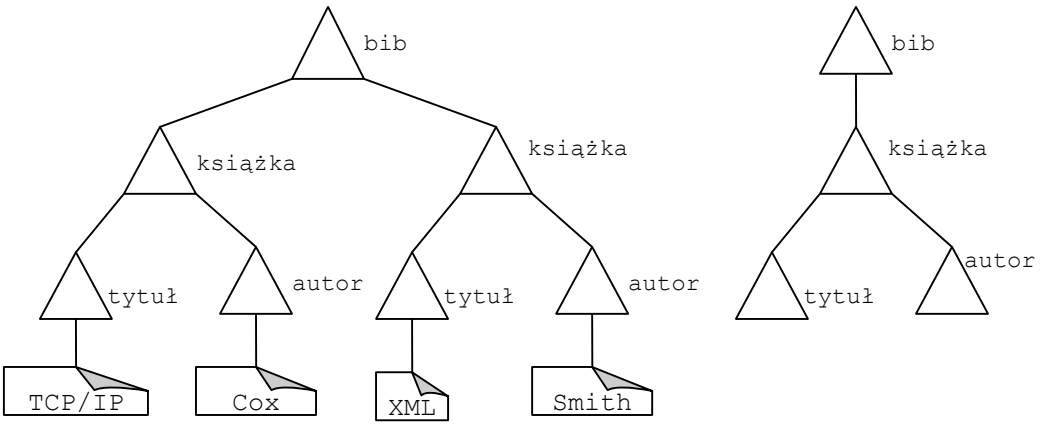
Drugą klasą algorytmów zarządzania współbieżnym dostępem do dokumentów XML są algorytmy oparte na standardzie XPath. Algorytmy te w procesie zarządzania współbieżnym dostępem do dokumentów XML wykorzystują różne techniki rozpoznawania sytuacji konfliktowych. W kolejnych punktach krótko przedstawimy algorytmy zarządzania współbieżnym dostępem oparte na standardzie XPath.

4.2.1. XMLTM

W pracy [9] zaproponowano algorytm XMLTM zarządzania współbieżnym dostępem do dokumentów XML oparty o DataGuide. Algorytm ten wykorzystuje protokół zarządzania

blokadami DGLOCK, będący modyfikacją protokołu blokowania dla acyklicznych grafów skierowanych DAG [11]. Modyfikacja protokołu DAG polega, między innymi, na zastąpieniu struktury grafu przez DataGuide, który lepiej odpowiada strukturze dokumentów XML. DataGuide odzwierciedla budowę dokumentu XML i jest wykorzystywany przez protokół blokowania DGLOCK jako model dokumentu, na którym rejestrowane są blokady zakładane przez poszczególne transakcje.

Na rys. 19 przedstawiono przykładowy dokument XML i odpowiadające mu drzewo DataGuide.



Rys. 19. Drzewo DOM dokumentu XML oraz odpowiadające mu drzewo DataGuide
Fig. 19. DOM Tree of XML document and corresponding DataGuide Tree

Algorytm zakłada, że dostęp do dokumentów XML jest realizowany za pomocą wyrażeń XPath. Uwzględniając budowę wyrażeń XPath, składających się z testów węzła oraz predykatów, zdefiniowano dwa typy ograniczeń: ograniczenia strukturalne i ograniczenia zawartości. Dla przykładu, w wyrażeniu ścieżkowym `/bib/książka/autor[.="Smith"]` wyrażenie `/bib/książka/autor`, składające się z testów węzła, jest ograniczeniem strukturalnym, a wyrażenie `[autor="Smith"]`, będące predykatem, jest ograniczeniem zawartości. Zgodnie z algorytmem ograniczenia strukturalne są traktowane jako ograniczenia na poziomie typu, a ograniczenia zawartości są traktowane jako ograniczenia na poziomie instancji. DGLOCK implementuje uszeregowanie za pomocą dwufazowego protokołu blokowania na węzłach DataGuide. Każde żądanie wykonania operacji na dokumencie XML wymaga założenia odpowiedniej blokady. Są one zwalnianie po zakończeniu transakcji. Poza standardowymi blokadami współdzielonymi (S), wyłącznymi (X) i intencjonalnymi (IS i IX) protokół DGLOCK wprowadza także opisy blokad, składające się z predykatów posiadających postać: $x\ \Theta$ `wartość`, gdzie $\Theta \in \{ =, \in, \neq, \leq, \ldots \}$.

Opis blokady węzła odpowiada predykatom danego węzła występującym w wyrażeniu ścieżkowym. Zadaniem opisów blokad jest ograniczenie zakresu blokady do węzłów spełniających określony predykat.

Tabela 17 przedstawia tabelę kompatybilności blokad dla protokołu DGLOCK.

Tabela 17
Tabela kompatybilności blokad dla protokołu DGLOCK

	IS	IX	S	SIX	X
IS	+	+	+	+	P
IX	+	+	P	P	P
S	+	P	+	P	P
SIX	+	P	P	P	P
X	P	P	P	P	P

W powyższej tabeli + oznacza kompatybilność blokad, natomiast P oznacza, że do zweryfikowania kompatybilności potrzeba jest analiza predykatów występujących w opisach blokad. Jeśli analiza predykatów prowadzi do stwierdzenia, że są one rozłączne, wówczas blokady zakładane na tych samych węzłach traktuje się jako kompatybilne. Jeśli rozłączność predykatów zawartych w opisach nie może być stwierdzona, wówczas zakłada się, że mamy do czynienia z sytuacją konfliktową.

Mechanizm działania algorytmu XMLTM dla żądania dostępu jest następujący:

1. Z wyrażeń ścieżkowych XPath, będących podstawą żądania, ekstrahowane są ograniczenia strukturalne, oznaczone przez E.
2. Wyznaczany jest zbiór N, będący zbiorem wszystkich węzłów w DataGuide, których pozycja jest określona ograniczeniem strukturalnym E. Przy tworzeniu tego zbioru wyróżniane są węzły modyfikowane i odczytywane przez żądanie.
3. Dla każdego węzła $n \in N$ wykonywane są następujące operacje:

A. Jeśli węzeł *n* jest modyfikowany, wówczas zakładane są blokady IX na wszystkich węzłach pomiędzy nim a korzeniem. Następnie na węzeł *n* zakładana jest blokada X. Podczas zakładania blokad jest dopisywany odpowiedni opis blokady.

B. Jeśli węzeł *n* jest odczytywany, wówczas zakładane są blokady IS na wszystkich węzłach pomiędzy nim a korzeniem. Następnie na węzeł *n* zakładana jest blokada S. Podobnie jak w przypadku modyfikacji, przy każdej blokadzie dodawany jest odpowiedni opis.

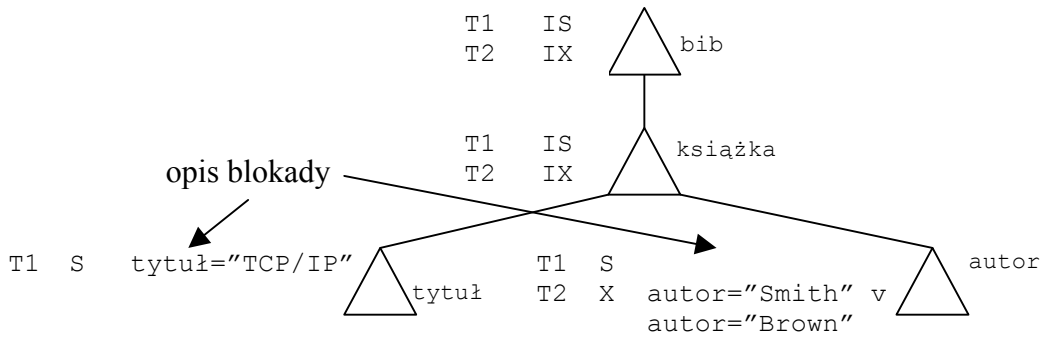
Dla ilustracji działania protokołu DGLOCK rozważmy następujący przykład:

Transakcja T1 odczytuje zawartość dokumentu XML za pomocą następującego zapytania `/bib/książka[tytuł="TCP/IP"]/autor`. Transakcja T2 zmienia zawartość tekstową węzła `autor` z `Smith` na `Brown`. W tym celu wykorzystuje wyrażenie:

```
/bib/książka/autor[.="Smith"].
```

Drzewo DataGuide z zaznaczonymi blokadami ma postać przedstawioną na rys. 20.

W analizowanym przykładzie weryfikacja konfliktu blokad S i X, założonych na węźle `autor` odpowiednio przez transakcje T1 i T2, wymaga weryfikacji opisu tych blokad. Dla transakcji T1 algorytm nie generuje opisu blokady. Oznacza to założenie blokady S na węźle `autor` niezależnie od jego zawartości. W takiej sytuacji opis `autor="Smith"`, umieszczony przy blokadzie założonej przez transakcję T2, nie może zostać uznany za rozłączny z opisem transakcji T1. Oznacza to, że pomiędzy żądaniami transakcji T1 i T2 występuje konflikt.



Rys. 20. Drzewo DataGuide z zaznaczonymi blokadami
Fig. 20. DataGuide Tree with locks

Gdyby transakcja T1 wykonała, dla przykładu, operację odczytu tytułów książek autora o nazwisku `Cox` (wykorzystując ścieżkę `/bib/książka[autor="Cox"]/tytuł`), wówczas na węźle `autor` zostałaby założona blokada S z opisem `autor="Cox"`. W takim przypadku żądanie transakcji T2 może zostać zrealizowane. Wynika to z faktu, iż opis blokady S `autor="Cox"` transakcji T1 oraz opis blokady X `autor="Smith"` transakcji T2 są wzajemnie rozłączne. Autorzy algorytmu zauważają możliwość zwiększenia granulacji blokad przez zakładanie ich na wyższych poziomach drzewa DataGuide. Efektem tych działań byłoby zmniejszenie liczby blokad oraz łatwiejsze zarządzanie blokadami.

4.2.2. Path Lock Satisfiability i Path Lock Propagation

W pracy [6] zaproponowano dwa algorytmy zarządzania współbieżnym dostępem do dokumentów XML, PLS (Path Lock Satisfiability) oraz PLP (Path Lock Propagation), zakładające, że dostęp do dokumentów realizowany jest za pomocą ograniczonych wyrażeń XPath.

W algorytmie przyjęto, że wyrażenia XPath będą opisane poniższą gramatyką

```
P ::= F | F/P | F//P
F ::= . | T | * | @A | @* | τ | σ
```

gdzie, `T` jest zbiorem nazw znaczników, `A` zbiorem nazw atrybutów, τ oznacza funkcję `text()`, która pobiera wszystkie podrzędne węzły tekstowe, a σ oznacza funkcję `string--value()`, która pobiera wartość atrybutu lub węzła tekstowego.

Wyrażenia XPath mogą rozpoczynać się od korzenia dokumentu lub dowolnego innego węzła wcześniej odczytanego przez transakcję. Dlatego też przyjmuje się, że podczas wykonywania operacji przez transakcję przechowuje ona zbiór zmiennych `X = {x1, x2, ...}`, w którym znajdują się węzły, będące rezultatami wcześniejszych zapytań. Zapytanie definiowane jest zatem w następującej formie `X := Q`, gdzie zapytanie `Q` jest definiowane następująco:

```
Q ::= /P | //P | X/P | X//P
```

Oprócz języka zapytań definiowany jest także zbiór operatorów, które pozwalają na modyfikację zawartości dokumentu. Operatory te zostały podzielone na trzy klasy: operatory

atrybutów, operatory elementów i operatory wartości tekstowych. Poniżej przedstawiono operatory dla każdej z klas.

Operatory atrybutów:

- `create-attribute(e-id,a-name,text)` – tworzy atrybut o nazwie `a-name` i zawartości `text` w elemencie o identyfikatorze `e-id`,
- `delete-attribute(a-id)` – usuwa atrybut o identyfikatorze `a-id`,
- `update-attribute(a-id,text)` – zmienia zawartość atrybutu o identyfikatorze `a-id` na `text`.

Operatory elementów:

- `create-element-under(e-id,tagname)` – tworzy pusty element o znaczniku `tagname` jako ostatni podelement węzła o identyfikatorze `e-id`,
- `create-element-before(e-id,tagname)` – tworzy pusty element o znaczniku `tagname` przed węzłem o identyfikatorze `e-id`,
- `create-element-after(e-id,tagname)` – tworzy pusty element o znaczniku `tagname` po węźle o identyfikatorze `e-id`,
- `delete-leaf-element(e-id)` – usuwa element o identyfikatorze `e-id`.

Operatory wartości tekstowych:

- `create-text-under(e-id,text)` – tworzy węzeł tekstowy o zawartości `text` jako ostatni węzeł tekstowy elementu o identyfikatorze `e-id`,
- `delete-text(t-id)` – usuwa węzeł tekstowy o identyfikatorze `t-id`,
- `update-text(t-id,text)` – zmienia zawartość węzła tekstowego o identyfikatorze `t-id` na `text`.

Obecnie przejdziemy do przedstawienia wspomnianych wcześniej algorytmów PLS i PLP.

4.2.2.1. Path Lock Satisfiability (PLS)

W przypadku algorytmu PLS blokada do odczytu jest definiowana jako para `(n,p)`, gdzie `n` jest identyfikatorem węzła, na którym jest zakładana blokada, a `p` jest ścieżką. Istnienie takiej blokady oznacza, że transakcja wykonała zapytanie rozpoczynające się od węzła `n` i wykorzystujące ścieżkę `p`. Blokady zakładane podczas wykonywania zapytań $x_n := q$ wynikają ze zbioru reguł umieszczonych w tabeli 18.

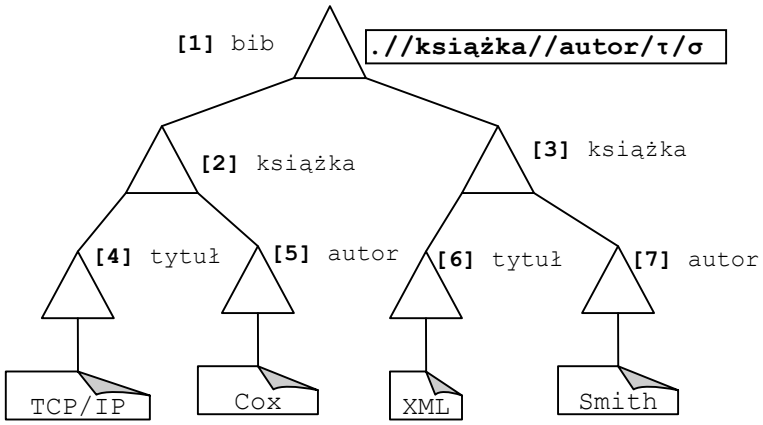
W tabeli przyjęto następujące oznaczenia: `r` oznacza korzeniem dokumentu, x_m oznacza zbiór węzłów będących rezultatami wcześniejszych zapytań, x_n oznacza zbiór węzłów będących rezultatem zapytania `q`.

Tabela 18
Reguły zakładania blokad do odczytu przez algorytm PLS

l.p.	Zapytanie – q	Wymagane blokady – R_q
1	x_m/p	$\{(x,p) \mid x \in x_m\}$
2	$x_m//p$	$\{(x,.//p) \mid x \in x_m\}$
3	/p	{(r,p)}
4	//p	{(r,.//p)}

Aby zilustrować mechanizm zakładania blokad do odczytu przez algorytm PLS rozważmy poniższy przykład.

Załóżmy, że transakcja T1 wykonuje następujące zapytanie q=//książka//autor/τ/σ, pozwalające na odczytanie wszystkich autorów książek. Na podstawie reguły czwartej z tabeli reguł zakładania blokad do odczytu przez algorytm PLS wynika, że transakcja T1 musi założyć blokadę na korzeniu dokumentu w postaci (r,.//książka//autor/τ/σ), co przedstawiono na rys. 21.



Rys. 21. Blokady założone dla zapytania q=//książka//autor/τ/σ przez algorytm PLS
Fig. 21. Locks set for query q=//książka//autor/τ/σ by PLS algorithm

Blokada do zapisu w algorytmie PLS jest definiowana także jako para (n, f), gdzie n jest identyfikatorem węzła, na którym zakładana jest blokada, a f jest elementem ze zbioru F\{.}. Jeśli na węzeł n założono blokadę z wyrażeniem f, wówczas oznacza to, że zostały zmodyfikowane węzły (elementy, atrybuty) bezpośrednio podrzędne w stosunku do węzła n, opowiadające wyrażeniu f lub f jest wartością tekstową σ zmodyfikowanego węzła. Typy blokad do zapisu zależą od operacji modyfikacji żądanych przez transakcje. Tabela 19 przedstawia reguły zakładania blokad do zapisu dla algorytmu PLS.

Tabela 19
Reguły zakładania blokad do zapisu przez algorytm PLS

Operacja	Wymagane blokady
create-attribute(n,a,v)	blokada @a na węźle n – (n,@a)
delete-attribute(n)	blokada @a na rodzicu atrybutu n, gdzie a to nazwa atrybutu n – (parent(n),@name(n))
update-attribute(n,v)	Blokada σ na węźle n – (n,σ)
create-element-under(n,t)	Blokada t na węźle n – (n,t)
delete-leaf-element(n)	Blokada t na rodzicu elementu n gdzie t jest nazwą znacznika n – (parent(n),name(n))
create-text-under(n,v)	Blokada τ na rodzicu n – (parent(n),τ)
delete-text(n)	Blokada τ na rodzicu n – (parent(n),τ)
update-text(n,v)	Blokada τ na n – (n,τ)

W przypadku algorytmu PLS weryfikacja kompatybilności blokad odbywa się za pomocą następujących reguł:

1. Dwie blokady do odczytu są zawsze kompatybilne.
2. Blokada do odczytu (n,p) jest w konflikcie z blokadą dla zapisu (n',f) wtedy i tylko wtedy, gdy n jest węzłem poprzedzającym n' i ścieżka path(n,n')/f spełnia wyrażenie XPath p.
3. Blokada do zapisu (n,f) pozostaje zawsze w konflikcie z blokadą (n,f').

Przez wyrażenie path(n,n') rozumiemy wyrażenie ścieżkowe, składające się z nazw wszystkich węzłów występujących pomiędzy węzłem n a węzłem n' rozdzielonych przez znak '/'. Dodatkowo mówimy, że ścieżka spełnia wyrażenie XPath p, jeżeli jest ona równa jednej ze ścieżek reprezentowanych przez p.

Każda transakcja ma możliwość założenia blokad, wynikających z wykonywanych przez nią operacji, tylko wtedy, gdy blokady te nie są w konflikcie z już istniejącymi blokadami.

Dla ilustracji przedstawmy scenariusz współbieżnej realizacji transakcji T1 i T2. Załóżmy, że transakcja T1 wykonuje wcześniej omówione zapytanie:

q=//książka//autor/τ/σ,

natomiast transakcja T2 próbuje dodać nowy element autor dla książki o tytule XML. W wyniku działania obu transakcji otrzymujemy dwie blokady. Pierwsza blokada transakcji T1 jest blokadą do odczytu i ma postać (r,.//książka//autor/τ/σ), gdzie wyrażenie r oznacza korzeń dokumentu. Druga blokada transakcji T2 jest blokadą do zapisu i ma postać ([3],autor), gdzie wyrażenie [3] jest identyfikatorem elementu książka o tytule XML oznaczonym na rys. 21 przez [3]. Aby zweryfikować istnienie sytuacji konfliktowej, musimy sprawdzić czy ścieżka path(r,[3])/autor spełnia wyrażenie XPath .//książka//autor. W związku z tym, że ścieżka path(r,[3])/autor, mająca postać /bib/książka/autor, spełnia wyrażenie XPath .//książka//autor, mamy do czynienia z konfliktem blokad

i transakcja T2 nie może wykonać swojej operacji. Gdyby transakcja T2 chciała, dla przykładu, dodać nowy tytuł do książki autorstwa Smitha lub zmodyfikować tytuł już istniejący, wówczas do konfliktu by nie doszło. Wynika to z faktu, iż ścieżka `path(dr, [3])/tytuł` nie spełnia wyrażenia XPath `.//książka//autor`.

Algorytm PLS zakłada wykonywanie operacji na dokumencie XML za pomocą uproszczonych wyrażeń XPath. Jest on oparty na protokole blokowania dwufazowego. Weryfikacja sytuacji konfliktowych realizowana jest za pomocą reguł opartych na porównywaniu ścieżek. Wadą algorytmu jest duża złożoność operacji weryfikacji spełniania ścieżek.

4.2.2.2. Path Lock Propagation (PLP)

Algorytm PLP jest kompromisem pomiędzy złożonością weryfikacji spełniania ścieżek a liczbą zakładanych blokad. Sposób działania algorytmu PLP został opisany poniżej.

Blokady dla odczytu są zakładane w sposób następujący. W pierwszym kroku dla danego zapytania $x_n := q$ są zakładane blokady inicjalizacyjne R_q zgodnie z regułami algorytmu PLS. Następnie stosowany jest mechanizm propagacji blokad, składający się z dwóch faz: fazy inferencji blokad oraz fazy propagacji blokad. Fazy te wykonywane są tylko dla blokad do odczytu. W fazie inferencji następuje wywiedzenie kolejnych blokad, które zakładane są na tych samych węzłach. Wywiedzenie to realizowane jest na podstawie reguł przedstawionych na rys. 22.

```
.//p  ➔ p
./p   ➔ p
```

Rys. 22. Reguły interferencji blokad dla algorytmu PLP
Fig. 22. Inference Rules for Read Locks

Faza propagacji, natomiast, powoduje propagację blokad na węzły podrzędne. Jest ona wykonywana na podstawie następujących reguł:

Tabela 20

Reguły propagacji blokad dla algorytmu PLP

Reguła	Blokada rodzica	Typ dziecka	Nazwa dziecka	Blokada dziecka
1	.//p	element	-	.//p
2	t/p	element	t	p
3	t//p	element	t	.//p
4	*/p	element	-	p
5	*//p	element	-	.//p
6	@a/p	atrybut	a	p
7	@*/p	atrybut	-	p
8	τ/p	tekst	-	p

Fazy inferencji i propagacji blokad wykonywane są cyklicznie do momentu, kiedy obie fazy nie doprowadzą do założenia żadnej nowej blokady. Rezultatem takiego działania jest zbiór blokad R_q*. Ponieważ R_q zależy od drzewa dokumentu, jest on wyliczany za każdym razem, gdy drzewo jest modyfikowane.

Ze względu na to, że węzły, na których pojawiają się blokady, są tożsame z tymi, które należy odwiedzić podczas wykonywania operacji zapytania, sam mechanizm propagacji blokad może być stosowany ze stosunkowo małym narzutem. Mechanizm propagacji jest operacją atomową, co oznacza, że albo zostaną założone wszystkie blokady wynikające z zapytania, albo wycofane zostaną blokady wynikające ze wszystkich wcześniejszych faz.

Blokady dla zapisu są w algorytmie PLP zakładane tak samo jak w przypadku algorytmu PLS.

Kompatybilność blokad w przypadku algorytmu PLP jest weryfikowana za pomocą następujących reguł:

1. Dwie blokady do odczytu są zawsze kompatybilne.
2. Blokada do odczytu `(n, p)` pozostaje w konflikcie z blokadą dla zapisu `(n, f)` wtedy i tylko wtedy, gdy `p = f` lub `p=*`.
3. Blokada do zapisu `(n, f)` jest zawsze w konflikcie z blokadą do zapisu `(n, f')`.

Każda transakcja ma możliwość założenia blokad wtedy, gdy blokady te nie są w konflikcie z już istniejącymi.

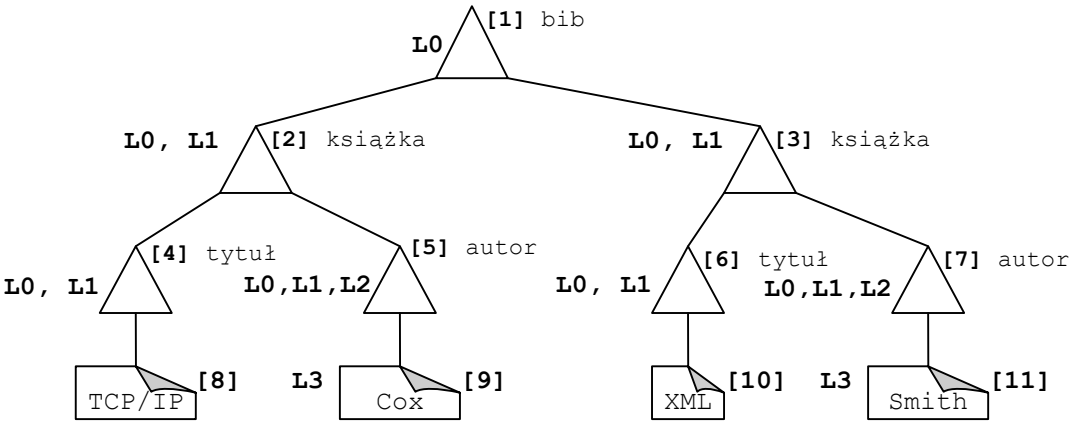
Aby zilustrować działanie powyższego mechanizmu, prześledźmy blokady jakie zostaną założone w wyniku działania algorytmu PLP dla ostatnio omówionego przykładu.

Załóżmy, że transakcja T1 wykonuje zapytanie `q=//książka//autor/τ/σ`. Blokada inicjalizacyjna w tym przypadku to `(r,.//książka//autor/τ/σ)`. Wykonując fazę inferencji uzyskamy blokadę `(r,książka//autor/τ/σ)`. Kolejną fazą jest faza propagacji. W jej efekcie, stosując trzecią regułę propagacji dla węzłów dzieci w stosunku do węzła `r`, uzyskamy dwie blokady `([[2],.//autor/τ/σ)` oraz `([3],.//autor/τ/σ)`. Po zastosowaniu reguły pierwszej, która stosowana jest niezależnie od nazwy dziecka (znacznika elementu), pojawią się kolejne dwie blokady na tych samych węzłach identyczne z blokadami na węźle `r`. Kolejne fazy przedstawia tabela 21 i drzewo dokumentu z zaznaczonymi blokadami przedstawione na rys. 23.

Tabela 21

Blokady założone dla zapytania `q=//książka//autor/τ/σ` przez algorytm PLP

symbol	blokada	po inferencji
L0	`.//książka//autor/τ/σ`	`książka//autor/τ/σ`
L1	`.//autor/τ/σ`	`autor/τ/σ`
L2	`τ/σ`	
L3	`σ`	



Rys. 23. Blokady założone dla zapytania q=//książka//autor/τ/σ przez algorytm PLP
Fig. 23. Locks set for query q=//książka//autor/τ/σ by PLP algorithm

W wyniku realizacji transakcji T2 założona zostanie blokada do zapisu, mająca postać ([3],autor). Aby zweryfikować istnienie sytuacji konfliktowej na podstawie reguł algorytmu PLP, musimy porównać blokady założone na węźle [3]. Z porównania blokady ([3],autor/τ/σ) transakcji T1 oraz blokady ([3],autor) transakcji T2 wynika, że mamy do czynienia z konfliktem blokad i transakcja T2 nie może wykonać swojej operacji.

Algorytm PLP, podobnie jak w przypadku PLS, zakłada wykonywanie operacji na dokumencie XML za pomocą uproszczonych wyrażeń XPath. Jest również oparty na protokole blokowania dwufazowego oraz weryfikacji sytuacji konfliktowych w oparciu o reguły porównujące ścieżki. Jednak w przypadku algorytmu PLP weryfikacja sytuacji konfliktowych jest znacznie mniej złożona, pojawia się natomiast problem propagacji, a dokładniej, konieczność przechowywania dużej liczby blokad. Oba algorytmy podczas szeregowania równoległych transakcji dają te same rezultaty.

4.2.3. XPath-based Concurrency Control (XbCC)

W pracy [5] został przedstawiony algorytm zarządzania współbieżnym dostępem do dokumentów XML, który różni się znacząco od algorytmów prezentowanych wcześniej. Idea tego algorytmu została oparta na porównywaniu zbiorów odczytywanych węzłów przy wykorzystaniu zbioru wersji dokumentu. Mimo iż algorytm zakłada, że dostęp do dokumentu wymaga wykorzystania wyrażeń XPath, to nie jest od tego uzależniony. Autorzy w swoim algorytmie wykorzystują model dokumentu XML, będący standardową reprezentacją wykorzystywaną np. w interfejsie DOM. W modelu tym dokument reprezentowany jest jako drzewo odwzorowujące relacje nadrzędny-podrzędny pomiędzy węzłami. Każdy węzeł w drzewie ma określony typ. Na podstawie typu rozróżniamy węzły elementów, węzły tekstowe lub węzły atrybutów. Jako model transakcji autorzy rozważają zbiór równoległych transakcji $T=\{T_1,\ldots T_n\}$ $(n>=2)$, które równolegle wykonują operacje na jednym dokumencie. Każda z transakcji wykonuje na przemian sekwencje operacji odczytu RS i sekwencje operacji

zapisu WS. Każdy typ sekwencji składa się z następujących po sobie operacji (R, W), których typ jest zgodny z typem sekwencji. Dla przykładu sekwencja operacji odczytu RS składa się ze zbioru następujących po sobie operacji R. Wykorzystywana jest następująca notacja:

- $WS_i(k)$ ($k > 0$) – oznacza k-tą sekwencję operacji zapisu transakcji T_i.
- $|WS_i(k)|$ – oznacza liczbę operacji zapisu w $WS_i(k)$.
- wn_i – oznacza bieżący numer W-sekwencji w transakcji T_i.
- $RS_i(k)$ ($k >= 0$) – oznacza sekwencję operacji odczytu następującą bezpośrednio po $WS_i(k)$. Dla $k = 0$ oznacza początkową sekwencję operacji odczytu.

Operacja odczytu reprezentowana jest przez wyrażenie *Read*(*path*). Operacja zapisu polega na modyfikacji określonego zbioru węzłów odczytanych wcześniej przez operację odczytu. Zakłada się występowanie trzech operacji zapisu: dodawania, usuwania i modyfikacji. Mimo zdefiniowania przez autorów typów operacji zapisu algorytm przedstawiony w pracy [5] nie jest od nich uzależniony.

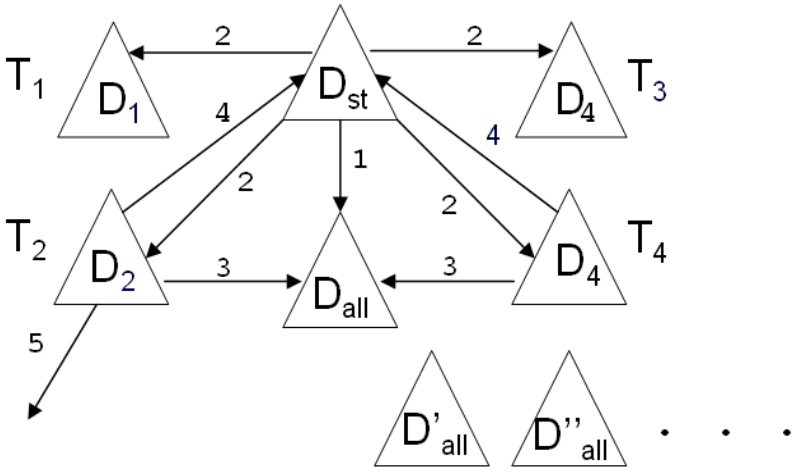
Dla ilustracji wyżej przedstawionego modelu transakcji przyjmijmy, że transakcja T_1 wykonała następującą sekwencję operacji:

```
T1 = {r1, r2, r3, w1, w2, r4, r5, r6, w3, w4, w5, w6}
```

W tym przypadku transakcja T_1 wykonała cztery sekwencje, dwie sekwencje odczytu i dwie zapisu. Dla przykładu do sekwencji $WS_1(2)$ należą operacje: w3, w4, w5, w6, a do sekwencji operacji $RS_1(0)$ operacje: r1, r2, r3.

4.2.3.1. Zarządzanie wersjami dokumentu

Jak wspomniano wcześniej, idea algorytmu została oparta na porównywaniu zbiorów węzłów przy wykorzystaniu zbioru wersji dokumentu. Sposób zarządzania wersjami dokumentu przedstawia rys. 24.



Rys. 24. Zarządzanie wersjami dokumentu w algorytmie XbCC
Fig. 24. Management of document versions in XbCC algorithm

Symbole użyte na rys. 24 mają znaczenie następujące:

D_{st} – wersja dokumentu przechowywana w bazie danych i zawierająca wszelkie zatwierdzone zmiany.

D_i – wersja dokumentu zawierająca modyfikacje transakcji T_i.

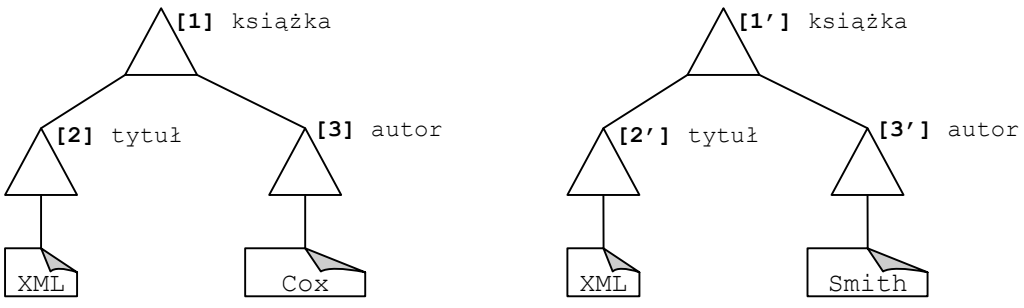
D_{all} – wersja dokumentu zawierająca modyfikacje wszystkich aktywnych transakcji.

Wersje dokumentu D_{st}, D_i i D_{all} są tworzone i utrzymywane zgodnie z poniższymi regułami.

1. W momencie, gdy pierwsza transakcja rozpoczyna pracę nad dokumentem, jest tworzona wersja dokumentu D_{all}, będąca kopią D_{st}.
2. W momencie, gdy transakcja T_i rozpoczyna pracę, jest tworzona wersja D_i, będąca kopią prywatną D_{st}.
3. Prywatna wersja dokumentu D_i transakcji T_i wykorzystywana jest podczas wszelkich operacji wykonywanych przez transakcję T_i. Wszelkie operacje zapisu transakcji T_i odzwierciedlane są na wersji D_i oraz dodatkowo na wersji D_{all}.
4. W momencie, gdy transakcja T_i zatwierdza dokonane przez siebie zmiany, są one propagowane z wersji dokumentu D_i do wersji dokumentu D_{st}.
5. W momencie, gdy transakcja T_i wycofuje dokonane przez siebie zmiany, wersja dokumentu D_i jest usuwana, a z wersji dokumentu D_{all} wycofywane są wszelkie zmiany wykonane przez transakcję T_i. Przywrócenie wartości poprzednich w wersji dokumencie D_{all} może być wykonane na podstawie zawartości wersji dokumentu D_{st}.

Dodatkowo zdefiniowane są dwa pojęcia: równoważności węzłów znajdujących się w różnych dokumentach i połączenia dwóch dokumentów.

Węzły są równoważne, gdy jeden z nich jest kopią drugiego, gdy są kopiami tego samego węzła lub, gdy wykonano na nich te same zmiany. Dwa zbiory węzłów $N = \{v_1,\ldots,v_m\}$ i $N' = \{v_1',\ldots,v_m'\}$ uważa się za równoważne, co oznaczamy przez $N \equiv N'$, gdy odpowiednie ich węzły są równoważne, czyli $v_j \equiv v_j'$ dla $1 <= j <= m$. Równoważność węzłów wymaga identycznej bezpośredniej zawartości węzłów, nie jest wymagana równoważność węzłów podrzędnych. Dla przykładu na rys. 25 węzły [1] i [1'] są równoważne, natomiast węzły [3] i [3'] nie są równoważne.



Rys. 25. Równoważność węzłów
Fig. 25. Equivalence of nodes

Połączenie dwóch dokumentów (oznaczone przez Merge(D_i,D_j)) jest dokumentem, którego zawartość jest utworzona z trzech składowych. Pierwszą składową jest zmodyfikowana przez transakcję T_i część dokumentu zawarta w dokumencie D_i. Drugą składową jest zmodyfikowana przez transakcję T_j część dokumentu zawarta w dokumencie D_j. Trzecią składową jest równoważna część dokumentów D_i i D_j. Zmodyfikowane części obu dokumentów muszą być wzajemnie rozłączne.

Tabela 22 przedstawia tabelę kompatybilności operacji wykonywanych przez transakcje dla algorytmu XbCC.

Tabela 22

Tabela kompatybilności operacji dla algorytmu XbCC

	R	W
R	-	wymaga weryfikacji
W	wymaga weryfikacji	-

Z tabeli kompatybilności dla algorytmu XbCC wynikają dwie operacje odczytu, czyli przypadek R-R – nigdy nie są w konflikcie. Weryfikacji wymagają przypadki R-W i W-R. Przypadek R-W występuje wówczas, gdy transakcja wykonuje operację odczytu i należy sprawdzić, czy nie pozostaje w konflikcie z wykonanymi wcześniej operacjami modyfikacji. Przypadek W-R występuje wówczas, gdy transakcja żąda wykonania operacji modyfikacji i musi być pewna, że nie pozostaje w konflikcie z wykonanymi wcześniej operacjami odczytu. W przypadku wykrycia konfliktu operacji, transakcja, która zamierzała wykonać operację, zostaje zatrzymana do momentu, gdy transakcja konfliktowa zostanie zakończona. Przypadek W-W nie jest rozważany. Wynika to z faktu, iż, aby móc wykonać operację modyfikacji węzłów, muszą one wcześniej zostać odczytane, a zatem dokonana musi zostać weryfikacja przypadku R-W.

4.2.3.2. Algorytm weryfikacji operacji konfliktowych

Algorytm weryfikacji operacji konfliktowych dotyczy dwóch przypadków wynikających z tabeli kompatybilności operacji.

Przypadek R-W polega na sprawdzeniu, czy operacja odczytu nie dotyczy węzłów wcześniej zmodyfikowanych przez równolegle wykonywane transakcje. Do weryfikacji tego przypadku autorzy pracy [5] wprowadzają funkcję Conf(R,*W*), której wynikiem jest wartość logiczna informująca o istnieniu lub braku konfliktu pomiędzy operacją odczytu R i sekwencją operacji zapisu *W*. Operacja R_i=read(path) transakcji T_i polega na odczycie zbioru węzłów znajdujących się na ścieżce zgodnej z wyrażeniem path i znajdujących się w dokumencie będącym kopią prywatną transakcji T_i, czyli D_i. Zbiór węzłów odczytywanych w wyniku operacji R_i=read(path) oznaczamy przez GetN(D_i,path). Przez W_j oznaczmy sekwencję wszystkich wcześniejszych operacji modyfikacji wykonanych przez transakcję T_j. Jeśli operacja odczytu R_i=Read(path) pozostaje w konflikcie z sekwencją operacji W_j to:

$$GetN(D_i,path) \text{ not} \equiv GetN(Merge(D_i,D_j),path)$$

Powyższy wzór oznacza, że jeżeli zbiór węzłów odczytany z wersji dokumentu D_i transakcji T_i nie jest równoważny ze zbiorem węzłów odczytanych z połączonej wersji dokumentu D_i i wersji dokumentu D_j, to operacja R_i pozostaje w konflikcie z W_j. Powyższą weryfikację należy wykonać dla wszystkich transakcji T_j, gdzie $j \neq i$. Autorzy pracy [5] zauważają, że powyższa weryfikacja nie jest wystarczająca. Wynika to z faktu, iż mogą zaistnieć przypadki, w których sytuacja konfliktowa nie ma miejsca pomiędzy operacjami R_i a W_j (Conf(R_i,W_j) = true) oraz R_i a W_h (Conf(R_i,W_h)=true), gdzie W_j i W_h należą odpowiednio do transakcji T_j i T_h, natomiast istnieje konflikt pomiędzy operacją R_i i połączoną sekwencją operacji W_j+W_h (Conf(R_i,W_j+W_h)≠true). Oznacza to, że, aby sprawdzić istnienie konfliktu, należałoby zweryfikować wszystkie możliwe kombinacje transakcji ze zbioru (T-{Ti}), co daje 2^{n-1} weryfikacji. Autorzy udowadniają, że, aby sprawdzić istnienie konfliktu pomiędzy operacją R_i a wszystkimi sekwencjami W_j zbioru transakcji T_j, gdzie $j \neq i$, wystarczy oprzeć analizę konfliktu na wersji dokumentu D_{all}, zawierającej wszystkie zmiany wykonane przez zbiór transakcji T_j. Niestety do wykrycia konfliktu nie wystarcza sprawdzenie równoważności węzłów wynikowych operacji GetN. Konieczna jest także weryfikacja węzłów, których istnienie lub zawartość była wykorzystana do wyznaczenia zbioru węzłów GetN. Sposób wyznaczenia takiego zbioru węzłów, który oznaczymy przez GetS, został przedstawiony w pracy [24], definiującej semantykę wyrażeń XPath.

Ostatecznie, aby sprawdzić istnienie konfliktu pomiędzy operacją odczytu R_i a wcześniejszymi modyfikacjami współbieżnie wykonywanych transakcji, należy zweryfikować warunek równoważności węzłów

$$GetS(D_i) \text{ not} \equiv GetS(D_{all})$$

Powyższy warunek w skrócie oznaczany jest przez Check(D_i,D_{all},R_i) i daje odpowiedź, czy pomiędzy operacją R_i, a wcześniejszymi operacjami W doszło do konfliktu. Niestety weryfikacja powyższego warunku nie odpowiada na pytanie, która transakcja jest w konflikcie z transakcją T_i. Do znalezienia tej transakcji należy, dla każdej z współbieżnie wykonywanych transakcji, zastosować warunek weryfikujący to, czy R_i pozostaje w konflikcie z W_j.

Podsumowując, procedura weryfikacji konfliktów R-W wygląda następująco:

- Sprawdź czy Check(D_i,D_{all},R_i)=ok.
- Jeśli tak, to konfliktu nie ma i transakcja T_i może wykonać operację R_i.
- Jeśli nie, to konflikt występuje. Należy wstrzymać transakcję T_i, następnie znaleźć transakcję T_j, która powoduje konflikt i, gdy T_j zostanie zakończona, kontynuować transakcję T_i.

Przypadek W-R jest nieco bardziej złożony. Rozważmy transakcję T_i, która żąda wykonania operacji W_i. W przyjętym sposobie zarządzania wersjami dokumentu transakcja T_i wykonywać będzie tę operację na wersji dokumentu D*i*. Niech D_i' = GetD(D_i,W_i) będzie stanem wersji dokumentu D_i po wykonaniu na nim modyfikacji W_i. Niech R_j będzie wcze-

śniej wykonaną operacją odczytu współbieżnie wykonywanej transakcji T_j. Weryfikacja sytuacji konfliktowej pomiędzy W_i i R_j polega na sprawdzeniu, czy

$$Check(D_j', Merge(D_j', D_i'), R_j) \neq ok.$$

W powyższym wzorze D_j' jest stanem wersji dokumentu D_j z czasu wykonania operacji R_j. Oznacza to, że, dla sprawdzenia konfliktu operacji W_i z wcześniej wykonanymi operacjami R_j, wymagane są poprzednie stany dla wszystkich wersji dokumentów D_j ($j \neq i$) z momentów wykonania operacji R_j przez transakcje $T_j \in T-\{T_i\}$. Istnieją dwa sposoby uzyskania poprzednich stanów dokumentów D_j'. Pierwszy z nich polega na przechowywaniu wszystkich wcześniejszych stanów, drugi na wygenerowaniu tych stanów przez ponowienie odpowiednich operacji modyfikacji, dla przykładu, w stosunku do stanu początkowego wersji dokumentu D_i. Oba te rozwiązania są niemożliwe do zastosowania w przypadku, gdy będziemy mieli do czynienia z dużą liczbą równoległych transakcji. Dlatego, podobnie jak w przypadku R-W, autorzy proponują wykorzystanie poprzednich stanów wersji dokumentu D_{all}.

Przy wykorzystaniu D_{all} weryfikacja konfliktu operacji modyfikacji W_i transakcji T_i z operacją odczytu $R_j \in RS_j(k)(0 \leq k \leq wn_j)$, czyli operacją odczytu znajdującą się w jednej z sekwencji operacji odczytu mających miejsce przed bieżącą sekwencją operacji zapisu transakcji T_j, wymaga sprawdzenia, czy

$$Check(D_{all}', Merge(D_i', D_{all}'), R_j) \neq ok.$$

W powyższym wzorze przez D_i' rozumiemy $GetD(D_i, W_i)$, czyli stan wersji dokumentu D_i po wykonaniu na nim modyfikacji W_i, przez D_{all}' rozumiemy stany D_{all} z okresu pomiędzy $WS_j(k)$ a $WS_j(k+1)$, czyli pomiędzy seriami modyfikacji transakcji T_j rozdzielonymi przez serię operacji odczytu, do której należy operacja R_j. Taka metoda weryfikacji wymaga przechowywania wszystkich stanów D_{all} lub ich odtwarzania. Autorzy proponują rozwiązanie, będące kombinacją tych dwóch podejść. Polega ono na przechowywaniu kopii dokumentu D_{all} powstałych we wcześniej wyznaczonych momentach czasu nazywanych *s*-punktami i ewentualną częściową regenerację tych kopii w celu uzyskania odpowiednich stanów D_{all}'. Autorzy w swej pracy [5] prezentują metodę wyznaczania *s*-punktów stworzoną specjalnie na potrzeby tego algorytmu.

Algorytm przedstawiony w pracy [5] stosuje oryginalne podejście do problemu współbieżnego dostępu do dokumentów XML. Kryterium decydującym o konflikcie pomiędzy operacjami jest efekt porównania węzłów odczytywanych przez transakcje przy wykorzystaniu wielu wersji dokumentu.

4.2.4. XPath locking protokol (XLP)

W pracy [15] zaproponowano algorytm zarządzania współbieżnym dostępem do dokumentów XML o nazwie XLP. Algorytm ten jest oparty o semantykę wyrażeń XPath [24] oraz

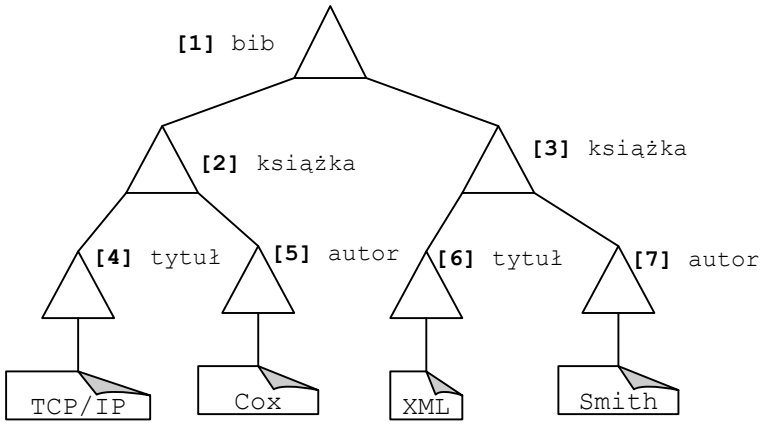
mechanizm zarządzania blokadami. Zanim przejdziemy do omówienia szczegółów algorytmu, wprowadzimy kilka oznaczeń związanych ściśle z semantyką wyrażeń XPath przedstawioną w [24].

Przez *węzły kontekstu kroku* wyznaczania węzłów docelowych $S_{i,j}$ na podstawie ścieżki L_j będziemy rozumieli węzły, od których $S_{i,j}$ startuje. Zbiór takich węzłów oznaczamy przez $C(S_{i,j})$.

Przez zbiór *węzłów pośrednich kroku* wyznaczania węzłów docelowych $S_{i,j}$ na podstawie ścieżki L_j będziemy rozumieli te węzły, które spełniają wyniki funkcji Node-Test w kroku $S_{i,j}$. Oznaczamy je przez $M(S_{i,j})$.

Przez zbiór *węzłów wynikowych kroku* wyznaczania węzłów docelowych $S_{i,j}$ rozumiemy zbiór węzłów $M(S_{i,j})$, spełniających predykat występujący w $S_{i,j}$. Oznaczamy je przez $R(S_{i,j})$. W rzeczywistości $R(S_{i,j}) = C(S_{i+1,j})$.

Aby zobrazować powyższe definicje załóżmy, że ścieżka L_j ma następującą postać: `/child:book/child::książka[position()=2]/child::autor`. Przyjmijmy też, że drzewo DOM, reprezentujące dokument XML, zostało przedstawione na rys. 26.



Rys. 26. Drzewo DOM przykładowego dokumentu XML
Fig. 26. DOM Tree for sample XML document

Tabela 23
Węzły K-kontekstu, P-pośrednie, W-wynikowe dla kroków wyrażenia L*j*

Krok	K	P	W
$S_{1,j}$	/	[1]	[1]
$S_{2,j}$	[1]	[2][3]	[3]
$S_{3,j}$	[3]	[7]	[7]

Ścieżka L_j składa się z trzech kroków $S_{1,j}$=`/child:book`, $S_{2,j}$=`child::[position()=2]`, $S_{3,j}$=`child::autor`, mających ogólną postać wyrażenia `axis::Node-Test[predykat]`. Zbiorem węzłów kontekstu kroku $S_{1,j}$ jest korzeń dokumentu. Zbiorem węzłów pośrednich kroku $S_{1,j}$ jest element `bib[1]`. Spełnia on, wynik funkcji Node-Test. Zbiorem węzłów wynikowych kroku $S_{1,j}$ jest również węzeł `bib[1]`, wynika to z faktu, iż w kroku $S_{1,j}$ nie ma

predykatu, który mógłby ograniczyć liczbę węzłów wynikowych. Omówione wcześniej rodzaje węzłów dla każdego z trzech kroków wyrażenia L_j zostały przedstawione w tabeli 23.

Kolejna definicja dotyczy *węzłów pośrednich ścieżki*. Są to wszystkie węzły, które pojawiły się jako węzły pośrednie kroków składowych ścieżki L_j. Oznaczamy je przez $Np(L_j)$. W naszym przykładzie są to węzły [1], [2], [3], [7].

Przez *węzły docelowe ścieżki* oznaczane przez $Nd(L_j)$ rozumiemy wynik przeszukiwania za pomocą ścieżki Lj. W rzeczywistości $Nd(Lj)=R(S_{|Lj|,j})$, gdzie $|L_j|$ jest długością L_j lub po prostu liczbą kroków ścieżki L_j. W naszym przykładzie jest to węzeł [7].

Przez *transakcję* $T=\{(O_i,L_i)\}$ rozumiemy zbiór par (O_i,L_i), gdzie O_i jest operacją (odczytu, zapisu, dodawania, usuwania) wykonywaną na węzłach wskazywanych przez ścieżkę L_i.

4.2.4.1. Typy blokad

Algorytm XLP wykorzystuje pięć rodzajów blokad: P (pass), R (read), W (write), I (insert), D (delete). Blokady P są zakładane na węzłach pośrednich ścieżki. Blokady R, W, I i D są zakładane na węzłach docelowych ścieżki w przypadku wykonywania na nich operacji odpowiedniego typu. Tabela 24 przedstawia tabelę kompatybilności blokad algorytmu XLP.

Tabela 24

Tabela kompatybilności algorytmu XLP

	P	R	W	I	D
P	+	+	+	+	-
R	+	+	-	+	-
W	+	-	-	+	-
I	+	+	+	-	-
D	-	-	-	-	-

Blokady P są zakładane przed oraz w trakcie realizacji kolejnych kroków wyznaczania węzłów docelowych $S_{i,j}$. Przed wykonaniem kroku $S_{i,j}$ blokady P są zakładane na węzłach $C(S_{i,j})$. W trakcie wykonywania kroku $S_{i,j}$ blokady P zakładane są na węzłach $M(S_{i,j})$. Po zakończeniu kroku $S_{i,j}$ blokady P są zwalniane na węzłach $M(S_{i,j})$-$R(S_{i,j})$, czyli na tych, które nie spełniły predykatów, dodatkowo blokady P zwalniane są na węzłach $C(S_{i,j})$. Blokady P na węzłach $C(S_{|Lj|,j})$, czyli węzłach docelowych ścieżki Lj, zamieniane są na blokady R, W, I lub D w zależności od typu wykonywanej operacji. Blokady P są kompatybilne ze wszystkimi blokadami prócz blokad D, które są zakładane w przypadku usuwania węzłów. Blokady R są zakładane na węzłach $Nd(L_j)$ w przypadku, gdy transakcja T wykonuje operację $(Read,L_j)$. Blokady R i I są kompatybilne, gdyż blokada I zakładana jest w wyniku modyfikacji struktury, a nie zawartości. Blokady W są zakładane na węzłach $Nd(L_j)$ w przypadku, gdy transakcja T wykonuje operację $(Write,L_j)$. Blokady I i D są zakładane na węzłach $Nd(L_j)$ w przypadku, gdy transakcja T wykonuje operację odpowiednio $(Insert,L_j)$ i $(Delete,L_j)$. W przypadku operacji usunięcia węzła są usuwane także wszystkie węzły podrzędne.

4.2.4.2. Protokół XLP

Protokół zakładania blokad XLP jest definiowany przez sześć reguł:

- Dwufazowa reguła blokowania. XML zachowuje protokół dwufazowego blokowania, za wyjątkiem blokad P.
- Reguła kompatybilności. Każdy krok S_i może wymagać założenia blokady odpowiedniego typu. Dopiero po założeniu blokady, krok S_i może zostać wykonany.
- Reguła ziarnistości. Blokady P i I dotyczą tylko węzłów. Blokady R i W dotyczą węzłów wraz z ich zawartością i atrybutami. Blokady D dotyczą węzła wraz ze wszystkimi elementami podrzędnymi.
- Reguła blokowania. Na węzłach $M(S_{i,j})$ w kroku $S_{i,j}$ są zakładane blokady P.
- Reguła zwalniania. Blokady P zwalniane są z węzłów $M(S_{i,j})$-$R(S_{i,j})$ po zakończeniu kroku $S_{i,j}$. Blokady P na węzłach $C(S_{i,j})$ są zwalniane po zakończeniu kroku $S_{i,j}$. Blokady R, W, I i D zwalnianie są dopiero w końcowej fazie transakcji.
- Reguła aktualizacji. Blokady na $Nd(L_j)$ są aktualizowane w stosunku do blokad I bezpośrednio przed wstawieniem węzłów do $Nd(L_j)$. Aktualizacja do blokad R, W oraz D jest realizowana zgodnie z regułą ziarnistości, odpowiednio dla każdego typu blokady.

Aby zilustrować działanie algorytmu XLP prześledźmy wykonanie dwóch transakcji na dokumencie przedstawionym na rys. 26. Załóżmy, że transakcja T1 odczytuje tytuł drugiej książki, natomiast transakcja T2 dodaje kolejnego autora do drugiej książki. Transakcje te są reprezentowane w sposób następujący:

$T1=\{(Read, L_1=$/child:book/child::książka[position()=2]/child::tytuł$)\}$,

gdzie: $L_1=\{S_{1,1}, S_{2,1}, S_{3,1}\}$, $S_{1,1}=$"child::bib", $S_{2,1}=$"child::książka[position() =2]", $S_{3,1}=$"child::tytuł",

$T2=\{(Insert, L_2=$/child:book/child::książka[position()=2]$)\}$,

gdzie: $L_2=\{S_{1,2}, S_{2,2}\}$, $S_{1,2}=$"child::bib", $S_{2,2}=$"child::książka[position()=2]"

Tabela 25 przedstawia przykładowe uszeregowanie tych transakcji.

Tabela 25

Uszeregowanie transakcji T1 i T2 dla algorytmu XLP

T1	**T2**
$S_{1,1}$: Lock-P([1])	
	$S_{1,2}$: Lock-P([1])
$S_{2,1}$: Lock-P([3])	
	$S_{2,2}$: Lock-P([3])
$S_{2,1}$: Unlock([1])	
	$S_{2,2}$: Unlock([1])
	$S_{2,2}$: Upgrade-I([3])
$S_{3,1}$: Lock-P([6])	
$S_{3,1}$: Unlock([3])	
$S_{3,1}$: Upgrade-R([6])	
$S_{3,1}$: Read([6])	
$S_{3,1}$: Unlock([6])	
	$S_{2,2}$: Insert([3])
	$S_{2,2}$: Unlock([3])

Algorytm XLP jest przykładem algorytmu opartego na protokole blokowania dwufazowego. Odstępstwem w tym zakresie jest zarządzanie blokadami P, które są zwalniane w trakcie realizacji transakcji. Mechanizm wcześniejszego zwalniania blokad P ma na celu zwiększenie stopnia współbieżności równolegle wykonywanych transakcji. Algorytm XLP jest silnie oparty o semantykę operacji XPath.

5. Porównanie

Nie ma w chwili obecnej w literaturze światowej opracowania, które zawierałoby porównanie propozycji algorytmów zarządzania współbieżnym dostępem do baz danych dokumentów XML. Należy zaznaczyć, że porównanie takie jest trudne. Algorytmy przedstawiane w literaturze bardzo różnią się od siebie, wykorzystują różne mechanizmy, zakładają różny sposób dostępu do dokumentów XML, zakładają różnego typu ograniczenia i uproszczenia. Nie zmienia to jednak faktu, iż takie porównanie jest konieczne. Wykorzystanie dokumentów XML już dawno wyszło poza sferę repozytoriów, których głównym zadaniem było udostępnianie zbioru dokumentów w trybie tylko do odczytu. Coraz powszedniej stosowane są bazy danych dokumentów XML, których funkcjonalność umożliwia nie tylko odczyt, ale także dodawanie dokumentów oraz ich modyfikację. Współbieżny dostęp do danych to jedna z podstawowych funkcjonalności baz danych. Wybór właściwego mechanizmu, z jednej strony uniwersalnego, dającego możliwość modyfikacji dokumentów XML za pomocą dowolnego z przyjętych standardów, z drugiej zaś dającego dużą współbieżność modyfikacji oraz dużą wydajność, to zagadnienie kluczowe. Punktem wyjścia do porównania algorytmów zarządzania współbieżnym dostępem do baz danych dokumentów XML jest określenie zbioru kryteriów ich oceny. Różny sposób opisu poszczególnych algorytmów powoduje, że zdefiniowanie takiego zbioru kryteriów, który w sposób obiektywny i miarodajny pozwoliłby na ocenę algorytmów, nie jest zagadnieniem trywialnym. W wyniku analizy dotychczasowych rozwiązań przyjęto trzy podstawowe grupy kryteriów. Pierwsza grupa to kryteria porównawcze, odnoszące się do opisu algorytmów i ich cech charakterystycznych. Druga grupa odnosi się do złożoności algorytmów. Trzecia grupa to kryteria oceny jakościowej algorytmów.

5.1. Kryteria opisu algorytmów

Celem tej grupy kryteriów jest zestawienie cech charakterystycznych poszczególnych algorytmów. Do grupy tej zaliczone zostały następujące kryteria:

- zakładana metoda dostępu do dokumentu XML,
- metoda synchronizacji,
- wykorzystywany model danych,

- poziom zakładania blokad ,
- cechy szczególne algorytmu.

Tabela 26 zawiera podstawowe cechy omówionych algorytmów

Tabela 26

Podstawowe cechy algorytmów

Algorytm	**Metoda dostępu**	**Metoda synchronizacji**	**Model danych**	**Miejsce blokad**	**Cechy szczególne**
Doc2PL	DOM API	system blokad	Drzewo DOM+(1)	dokument	
Node2PL	DOM API	system blokad	Drzewo DOM+(1)	poddrzewo	
NO2PL	DOM API	system blokad	Drzewo DOM+(1)	węzły	korzysta z (1)
OO2PL	DOM API	system blokad	Drzewo DOM+(1)	wskaźniki(1)	
XTO	DOM API	znaczniki czasowe	Drzewo DOM+(2)	węzły(6)	
XCO	DOM API	graf kol. transakcji	Drzewo DOM+(3)	węzły(7)	
taDOM	DOM API	system blokad	Drzewo DOM+(4,5)	węzły	trzy rodzaje blokad(8)
XMLTM	XPath	system blokad	DataGuide	węzły	blokady + warunki(9)
PLS	XPath	system blokad	Drzewo DOM	węzły	blokady + ścieżki
PLP	XPath	system blokad	Drzewo DOM	węzły	propagacja blokad PLS
XbCC	XPath	porów. odczy-tów	zbiór wersji dok.	-	porów. serii odczytów
XLP	XPath	system blokad	Drzewo DOM	węzły	wyk. semantykę XPath

Objaśnienia:
(1) – wskaźniki wprowadzone pomiędzy elementami w drzewie XML – (pierwsze dziecko, ostatnie dziecko, poprzednik, następnik)
(2) – dodatkowo utrzymywany jest graf przydziału zasobów i graf zależności, pozwalający na kaskadowe wycofanie transakcji, oraz znaczniki czasowe transakcji na poziomie każdego węzła, które realizowały dostęp do poszczególnych węzłów
(3) – graf przydziału zasobów, graf uszeregowania względem <DCO oraz graf zależności, pozwalający na kaskadowe wycofanie transakcji
(4) – drzewo DOM z nieznacznymi modyfikacjami takimi jak korzenie atrybutów czy dodatkowe węzły ciągów znaków
(5) – dodatkowe tabele dla blokad logicznych
(6) – miejsce przechowywania informacji dotyczących znaczników czasowych transakcji
(7) – miejsce przechowywania informacji dotyczących transakcji
(8) – blokady nawigacyjne, węzłów, logiczne, a także protokoły sterujące liczbą i ziarnistością blokad
(9) – warunki określające zakres nałożonej blokady – określanie konfliktów polegające na badaniu predykatów

Proponowane algorytmy, w zależności od przyjętej przez nie metody dostępu do dokumentu XML, można podzielić na dwie grupy: algorytmy zakładające wykorzystanie DOM API oraz algorytmy zakładające wykorzystanie XPath. Należy zwrócić uwagę, że praktycznie nie występują algorytmy abstrahujące od wykorzystywanej metody dostępu do dokumentów XML. Analizując metodę synchronizacji, można stwierdzić, że większość przedstawionych algorytmów wykorzystuje systemy blokad. Wyjątkiem są algorytmy XTO i XCO, które wykorzystują odpowiednio mechanizm znaczników czasowych transakcji oraz graf kolejności transakcji. Kolejnym wyjątkiem jest algorytm XbCC, który oparty został na oryginalnej metodzie porównywania odczytów z wykorzystaniem wielu wersji dokumentu. Analizując model danych, wykorzystywany przez algorytmy do synchronizacji dostępu, można stwierdzić, że w większości przypadków jest to drzewo DOM. Wyjątkiem w tym przypadku jest

algorytm XbCC, wykorzystujący w tym celu zbiór kopii dokumentu, oraz XMLTM, wykorzystujący DataGuide. Niektóre algorytmy, oprócz drzewa DOM, wykorzystują struktury będące rozszerzeniami drzewa DOM (np. OO2PL, taDOM) lub niezależnymi, dodatkowymi, obiektami takimi jak tabele blokad logicznych wykorzystywane przez taDOM. W przypadku algorytmów opartych na blokadach występują różnice ze względu na miejsce i poziom zakładania blokad. W większości przypadków poziomem tym są pojedyncze węzły dokumentu, choć istnieją algorytmy (np. Doc2PL), gdzie miejsce i poziom przechowywania informacji o blokadach jest inny. Analizując cechy szczególne algorytmów, można zauważyć, że wiele z nich posiada cechy indywidualne. Przykładowo algorytm taDOM wykorzystuje trzy rodzaje blokad, XMLTM przechowuje wraz z blokadami warunki, których porównanie decyduje zajściu o sytuacji konfliktowej.

5.2. Kryteria złożoności algorytmów

Celem tej grupy kryteriów jest porównanie przedstawionych algorytmów w zakresie ich złożoności obliczeniowej i pamięciowej. Niestety, poszczególne algorytmy zostały przedstawione w bardzo różny sposób, często uniemożliwiający precyzyjne wyznaczenie wyżej wspomnianych kryteriów. Do grupy tej zaliczone zostały następujące kryteria:

- złożoność obliczeniowa operacji założenia blokady,
- złożoność obliczeniowa weryfikacji sytuacji konfliktowej,
- złożoność pamięciowa – dotyczy mechanizmu zarządzania współbieżnym dostępem przy dostępie do pojedynczego dokumentu, gdzie przez: T – oznaczamy liczbę równoległych transakcji realizujących dostęp do dokumentu, N – liczbę węzłów w dokumencie, N' – liczbę węzłów w dokumencie z wyłączeniem liści, S – wielkość dokumentu.

Analizując złożoność obliczeniową poszczególnych algorytmów (tabela 27), można stwierdzić, że zależy ona ściśle od wykorzystywanej metody synchronizacji dostępu. Najmniejszą złożonością obliczeniową charakteryzują się algorytmy oparte na metodach wykorzystujących systemy blokad. Różnice, które można zaobserwować, wynikają z zakładanej metody dostępu. Algorytmy, które zakładają, że dostęp do dokumentów XML jest realizowany za pomocą wyrażeń XPath, z reguły charakteryzują się większą złożonością obliczeniową mechanizmów zakładania blokad. Przykładowo, protokół PLP przy zakładaniu blokad wykorzystuje złożony mechanizm propagacji blokad. Analiza złożoności obliczeniowej mechanizmu weryfikacji sytuacji konfliktowych daje podobne wyniki. Dla przykładu, algorytm XMLTM do weryfikacji sytuacji konfliktowych wymaga porównania warunków znajdujących się przy blokadach zakładanych na tych samych węzłach DataGuide. Algorytmem, który ze względu na dużą złożoność obliczeniową weryfikacji sytuacji konfliktowych znacząco różni się od pozostałych jest XbCC. Wynika to z zastosowania zupełnie odmiennej

metody weryfikacji sytuacji konfliktowych, polegającej na porównywaniu wyników zapytań uzyskiwanych w oparciu o różne wersje tego samego dokumentu.

Tabela 27

Złożoność algorytmów

Algorytm	Złożoność obliczeniowa zakładania blokad	Złożoność oblicz. weryfikacji	Złożoność pamięciowa
Doc2PL	1	1	T
Node2PL	1	1	N'*T
NO2PL	1	1	N*T
OO2PL	1	1	4* N*T
XTO	1	1	$(2+1)^{(1)}$*N
XCO	1	1	$3^{(2)}$* N*T
taDOM	1	1	$2^{(8)}$*N*T + $TL^{(8')}$
XMLTM	1	zależne od (3)	zależne od typu dok.
PLS	1	$LP^{(4)}$^4	N
PLP	zależna od $LP^{(4)}$ – propagacja	1	N*$LP^{(4)}$
XbCC	nie dotyczy	R-W = (6); W-R = (7)	S*(T +1+$Nsp^{(5)}$)
XLP	zależna od $LP^{(4)}$	1	N*T

Objaśnienia:
(1) – dwa znaczniki czasowe i status węzła
(2) – dla każdej z trzech operacji (odczytu, usunięcia, wstawienia)
(3) – złożoności warunków określające zakres nałożonej blokady – określanie konfliktów polegające na badaniu predykatów
(4) – LP rozumiemy długość ścieżki występującej przy blokadzie odczytu
(5) – przez Nsp rozumiemy liczbę tzw. s-punktów
(6) – koszt wykonania operacji odczytu na dokumencie Dall i porównania wyników
(7) – koszt wielokrotnego wykonania przeszłych operacji odczytów i porównania wyników – wzór dot. złożoności znajduje się w [5]
(8) – dwa typy blokad: węzłów i nawigacyjne, (8') – dodatkowe tabele do przechowywania blokad logicznych

Analizując złożoność pamięciową poszczególnych algorytmów, należy zwrócić uwagę na to, że jest ona także zależna od wykorzystywanej metody synchronizacji dostępu. Różnice pomiędzy algorytmami opartymi na blokadach wynikają głównie z poziomów zakładania blokad. Dla przykładu, algorytm Doc2PL zakłada blokady na poziomie dokumentów. W związku z tym jego złożoność pamięciowa jest najmniejsza. Pozostałe algorytmy, zakładające blokady na niższych poziomach, mają złożoność pamięciowa odpowiednio większą. Algorytmem, który ze względu na dużą złożoność pamięciową znacząco różni się od pozostałych jest ponownie XbCC. Wynika to z faktu, iż w przypadku tego algorytmu jest przechowywanych wiele wersji dokumentu, między innymi po jednej dla każdej z transakcji działającej na dokumencie.

5.3. Kryteria oceny jakościowej algorytmów

Najważniejszą grupą kryteriów oceny algorytmów są kryteria oceny jakościowej. Do grupy tych kryteriów zostały zaliczone:

- ograniczenia narzucone na założoną metodę dostępu do dokumentów XML w stosunku do zakresu wynikającego z obecnej postaci standardu,

- możliwość wykorzystania innych metod dostępu do dokumentów XML,
- uwzględnienie zależności występujących w dokumentach XML: H – hierarchie, Z – zawartość węzłów, K – kolejność węzłów, R – referencje i identyfikatory, L – zależności logiczne,
- dodatkowe ograniczenia: CA – możliwe wystąpienie kaskadowego wycofywania transakcji, KZ – konieczność zatwierdzania transakcji zgodnie z ich znacznikami czasowymi, FA – możliwość wystąpienia zjawiska fantomu, WSP – znaczący wpływ predykatów na stopień współbieżności, WSS – znaczący wpływ struktury dokumentu na stopień współbieżności.

Tabela 28

Ocena jakościowa algorytmów

Algorytm	Interfejs		Uwzględnione zależności	Dodatkowe ograniczenia
	ogr.	uproszczenia		
Doc2PL	tak	istotne	H, Z, K, R, L	
Node2PL	tak	istotne	H, Z, K	
NO2PL	tak	istotne	H, Z, K	
OO2PL	tak	istotne	H, Z, K	
XTO	tak	istotne	H, Z, K	KZ, CA
XCO	tak	istotne	H, Z, K	CA
taDOM	tak	brak	H, Z, K, R, L	
XMLTM	tak	proste predykaty	H, Z	WSP(1), WSS, FA
PLS	tak	brak predykatów(4)	H, Z	
PLP	tak	brak predykatów(4)	H, Z	
XbCC	nie(3)	brak	H, Z, K, R, L	(2)
XLP	tak	brak	H, Z	FA

Objaśnienia:
(1) – warunki określające zakres nałożonej blokady – określanie konfliktów polegające na badaniu predykatów
(2) – bardzo złożona obliczeniowo weryfikacja zależności konfliktów W-R
(3) – możliwość zastosowania dla dowolnej metody dostępu do dokumentów XML
(4) – uproszczone wyrażenia ścieżkowe np. brak uwzględnienia kroków w postaci `/following-sibling::nazwa_węzła`

Analizując ograniczenia, polegające na założeniu wykorzystania ściśle określonej metody dostępu do dokumentów XML, można stwierdzić, że występują one praktycznie we wszystkich algorytmach. Wyjątkiem jest algorytm XbCC oparty o mechanizm porównywania odczytów. Mechanizm ten może być implementowany w sposób niezależny od metody dostępu. Analiza przyjętych przez algorytmy uproszczeń, dotyczących metod dostępu, pozwala zauważyć, że tylko w trzech algorytmach, taDOM, XbCC oraz XLP, takie uproszczenia nie występują. Istotne uproszczenia występują wśród algorytmów zakładających wykorzystanie interfejsu DOM API. Polegają one na ograniczeniu liczby metod tego interfejsu. Wśród algorytmów opartych na wyrażeniach XPath uproszczenia dotyczą przede wszystkim postaci predykatów. Ma to miejsce np. w algorytmie XMLTM oraz w algorytmach PLP i PLS, które zupełnie nie uwzględniają możliwości użycia predykatów. Analizując złożone zależności występujące w dokumentach XML, można stwierdzić, że często przyjmują one uproszczony model dokumentów XML, w którym ignorowane są złożone zależności, takie jak identyfika-

tory elementów czy referencje. Analiza dodatkowych ograniczeń pozwala wyróżnić algorytmy, w których: możliwe są zjawiska fantomu (XMLTM, XLP), występuje możliwość kaskadowego wycofywania transakcji (XTO, XCO), występują ograniczenia dotyczące kolejności zatwierdzania transakcji (XTO) czy też stopień współbieżności jest zależny od postaci predykatów oraz różnorodności nazw znaczników występujących w dokumencie XML (XMLTM).

Przedstawiony powyżej zbiór kryteriów oceny jakościowej nie wystarcza do pełnej oceny algorytmów. Oceny tej nie można dokonać w oderwaniu od kierunków rozwoju technologii dotyczącej przetwarzania dokumentów XML oraz tendencji rozwoju baz danych dokumentów XML. Porównując algorytmy, należy również uwzględnić: (1) tendencje związane z rozwojem języków zapytań do dokumentów XML, (2) tendencje związane z rozwojem języków modyfikacji dokumentów XML, (3) możliwość wykonywania operacji nie tylko na pojedynczych dokumentach XML, ale także na zbiorach dokumentów, które w bazach danych dokumentów XML nazwane zostały kolekcjami.

Początkowo dostęp do dokumentów XML realizowany był głównie za pomocą metod interfejsu DOM API. W chwili obecnej głównym sposobem dostępu do dokumentów XML są wyrażenia XPath oraz język zapytań XQuery. Język zapytań XQuery wykorzystuje wyrażenia XPath. Daje to możliwość zastosowania większości algorytmów zarządzania współbieżnym dostępem opartych na wyrażeniach XPath również w przypadku użycia języka XQuery. Należy jednak zauważyć, że możliwości zapytań XQuery, takie jak: możliwość definiowania i wykorzystania własnych funkcji, znacząco rozszerzają możliwości wyrażeń XPath. Z drugiej strony, wykorzystywanie wyrażeń XPath czy też języka XQuery nie wyklucza dostępu do dokumentów XML za pomocą interfejsu DOM API. DOM API jest w dalszym ciągu bardzo powszechnie wykorzystywany np. do integracji baz danych dokumentów XML z narzędziami zewnętrznymi takimi jak edytory plików XML. Istnieje zatem potrzeba algorytmu zarządzania współbieżnym dostępem do dokumentów XML, który byłby uniwersalny lub niezależny od użytej metody dostępu.

Podobnie jak w przypadku języka XQuery, który uzupełnia możliwości interfejsu DOM API w zakresie wykonywania zapytań na dokumentach XML, tak w zakresie modyfikacji dokumentów coraz częściej wykorzystywane są języki takie jak XUpdate, wykorzystujące wyrażenia XPath, czy też rozszerzenia języka Xquery, zawierające polecenia modyfikacji. Należy zauważyć, że większość algorytmów, zakładających dostęp do dokumentów za pomocą wyrażeń Xpath, jest zgodna z możliwościami poleceń języka XUpdate. Podobnie jak w przypadku języka zapytań, wykorzystanie poleceń XUpdate nie wyklucza wykorzystywania DOM API. Oznacza to, że również z punktu widzenia modyfikacji dokumentu potrzebny jest algorytm oderwany od ściśle określonego sposobu modyfikacji dokumentu.

Bazy danych dokumentów XML w ostatnich latach przeżywają okres burzliwego rozwoju. W wielu przypadkach dają one możliwość wykonywania operacji zarówno na pojedyn-

czych dokumentach, jak i na zbiorach dokumentów zwanych kolekcjami. Algorytm zarządzania współbieżnym dostępem powinien uwzględniać takie przypadki. W związku z tym, że, dzięki właściwościom dokumentów XML, kolekcja może zostać uznana jako dokument XML, którego węzłami byłyby dokumenty składowe kolekcji, dostosowanie omawianych algorytmów do tej własności baz danych XML jest możliwe do realizacji.

6. Podsumowanie

W pracy przedstawiono algorytmy zarządzania współbieżnym dostępem do baz danych dokumentów XML. Dokonano ich klasyfikacji, zaproponowano zbiór kryteriów oceny, a także przeprowadzono analizę algorytmów w oparciu o wprowadzone kryteria.

Porównanie algorytmów zarządzania współbieżnym dostępem do baz danych dokumentów XML jest trudne. Wynika to z faktu, iż poszczególne algorytmy bardzo różnią się od siebie, wykorzystują różne mechanizmy, zakładają różny sposób dostępu do dokumentów XML, zakładają różnego typu ograniczenia i uproszczenia. Punktem wyjścia do porównania algorytmów było określenie zbioru kryteriów ich oceny. W wyniku analizy przyjęto trzy podstawowe grupy kryteriów: grupę kryteriów porównawczych, grupę kryteriów złożoności algorytmów i grupę kryteriów oceny jakościowej. Wyniki analizy zostały zaprezentowane w rozdziale 5.

Aktualnie prowadzone prace, dotyczące algorytmów zarządzania współbieżnym dostępem do baz danych dokumentów XML, koncentrują się przede wszystkim na rozwoju, modyfikacji i doskonaleniu istniejących już propozycji. Konsekwencją niniejszej pracy będzie porównanie ilościowe wybranych algorytmów oraz zaproponowanie algorytmu, który byłby niezależny od wykorzystywanego sposobu dostępu do dokumentów XML oraz uwzględniałby tendencje związane z rozwojem metod przetwarzania dokumentów XML.

LITERATURA

1. Al-Jadir L., El-Moukaddem F.: F2/XML: Storing XML Documents in Object Databases. OOIS 2002.
2. Buneman P., Grohe M., Koch Ch.: Path Queries on Compressed XML. VLDB 2003: 141÷152.
3. Bernstein P. A., Hadzilacos V., Goodman N.: Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
4. Cellary W., Gelenbe E., Morzy T.: Concurrency Control in Distributed Database Systems. North –Holland 1998 ISBN:0 444 70409 4.

5. Choi E.-H., Kanai T.: XPath-based Concurrency Control for XML Data. In: Proceedings of the 14th Data Engineering Workshop (DEWS 2003), Kaga city, Ishikawa, Japan, March 3-5, 2003.
6. Dekeyser S., Hidders J.: Path Locks for XML Document Collaboration. WISE 2002: 105÷114.
7. Furche T.: Optimizing Multiple Queries against XML Streams. Diploma thesis, Univ. of Munich, 2003.
8. Frick M., Grohe M., Koch C.: Query Evaluation on Compressed Trees. In Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS), Ottawa, Canada 2003.
9. Grabs T., Böhm K., Schek H.-J.: XMLTM: efficient transaction management for XML documents. CIKM 2002: 142÷152.
10. Gottlob G., Koch C., Pichler R.: Efficient Algorithms for Processing XPath Queries. In Proc. 28th VLDB Conf., 2002.
11. Gray J., Reute A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.
12. Haustein M. P., Härder T.: taDOM: A Tailored Synchronization Concept with Tunable Lock Granularity for the DOM API. ADBIS 2003: 88÷102.
13. Helmer S., Kanne C.-Ch., Moerkotte G.: Isolation in XML Bases. Technical report: Reihe Informatik 15/2001, University of Mannheim, Germany 2001.
14. Helmer S., Kanne C.-Ch., Moerkotte G.: Lock-based Protocols for Cooporation on XML Documents. Technical Report: Reihe Informatik 06/2003, University of Mannheim, Germany 2003.
15. Jea K.-F. J., Chen S.-Y., Wang S.-H.: Concurrency Control in XML Document Databases: XPath Locking Protocol. ICPADS 2002: 551÷556.
16. Jiang H., Lu H., Wang W., Xu Yu J.: Path Materialization Revisited: An Efficient Storage Model for XML Data. Australasian Database Conference, 2002.
17. Kanne C. Ch., Moerkotte G.: Efficient Storage of XML Data. Poster abstract in Proc. ICDE: 198. (2000).
18. Liefke H., Suciu D.: XMill: An Effcient Compressor for XML Data. In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2000.
19. Pleshachkov P., Chardin P., Kuznetzov S.: XDGL: XPath-Based Concurrency Control Protocol for XML Data.
20. Runapongsa K., Patel J. M.: Storing and Querying XML Data in Object-Relational DBMSs. EDBT Workshops 2002: 266÷285.
21. Schmidt A., Kersten M. L., Windhouwer M., Waas F.: Efficient Relational Storage and Retrieval of XML Documents. WebDB (Selected Papers) 2000: 137÷150.

22. Shanmugasundaram J., Tufte K., Zhang C., He G., DeWitt D. J., Naughton J. F.: Relational Databases for Querying XML Documents: Limitations and Opportunities. VLDB 1999: 302÷314.
23. Tolani P., Haritsa J. R.: XGRIND: A Query friendly XML Compressor. In Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE), 2002.
24. Wadler P.: Two semantics for XPath. 1999.
25. Win K.-M., Ng W. K., Lim E.-P.: ENAXS: Efficient Native XML Storage System. APWeb 2003: 59÷70.
26. Weikum G., Vossen G.: Transactional Information Systems – Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers, 2002.
27. Zhang N., Ozsu M. T.: Optimizing Correlated Path Queries in XML Languages. Technical Report, University of Waterloo, November 2002.

Recenzent: Dr inż. Marcin Skowronek

Wpłynęło do Redakcji 31 lipca 2006 r.

Abstract

Concurrent and uncontrolled access to XML database systems, like in relational and object-oriented database systems, may lead to data inconsistency. Concurrency control problem was widely considered in the literature [26, 3]. The variety of correctness criteria for concurrency control in database systems and variety of concurrency control algorithms were proposed. The main, commonly accepted concurrency control criterion is conflict serializability [3, 4]. Developed algorithms represent three main approaches: locking, time stamp ordering and optimistic. As we mentioned before, there are only few papers on concurrency control for native XML database systems. Mechanisms used so far in commercially available DBMS are based on locking protocols and offer very low degree of concurrency and, thus, very low processing performance. There is an obvious need to develop new methods of concurrency control for XML database systems, which provide database consistency and acceptable degree of concurrency. These methods should provide acceptable performance taking account specific XML document features and variety of modification methods. There have been only few propositions to solve this problem so far: [13, 14, 12, 9, 6, 5, 15]. These solutions differ in degree of concurrency and assumed modification methods. There is no critical analysis comparing these solutions.

The aim of this paper is a survey of proposed concurrency control algorithms and their critical analysis. The algorithms classification is also presented. The criteria of evaluation are defined and algorithms' analysis is made on the basis of proposed criteria.

Adres

Krzysztof JANKIEWICZ: Politechnika Poznańska, Instytut Informatyki, ul. Piotrowo 2, 60-965 Poznań, Polska, Krzysztof.Jankiewicz@cs.put.poznan.pl