DEIM2022 D43-2

Dejimaを用いた自律分散型データ共有・更新システムの実装

清水　敏之†　　加藤　弘之††　　吉川　正俊†††

† 九州大学附属図書館研究開発室　〒819–0395 福岡県福岡市西区元岡 744
†† 国立情報学研究所アーキテクチャ科学研究系　〒101–8430 東京都千代田区一ツ橋 2-1-2
††† 京都大学大学院情報学研究科　〒606–8501 京都府京都市左京区吉田本町
E-mail: †shimizu.toshiyuki.457@m.kyushu-u.ac.jp, ††kato@nii.ac.jp, †††yoshikawa@i.kyoto-u.ac.jp

あらまし　現在，膨大なデータが分散的に生成されており，関連する複数のデータベース間でデータ共有を行う場面が増加している．データ共有を行うことで，より適切なデータ分析を行うことができたり，管理の効率化が期待できたりする場合がある．データの全体像の把握が困難であり，複数のデータ管理主体が自律分散的にデータを管理している場合，局所的な信頼関係に基づいたデータ共有を行い，意図しないデータ共有・更新を行わないようにするために，条件を設定して輸出入するデータをフィルタリングできることが望ましい．本論文では，我々がこれまでに提案してきた自律分散型データ共有・更新システムについて，双方向変換を用いた既存のアーキテクチャである Dejima を用いて実装する方法について議論する．

キーワード　データ共有, 双方向変換, provenance

1　はじめに

近年，データに基づく分析や意思決定の需要の高まりなどから様々な分野でデータ管理が重要視され，管理されるデータも大規模かつ多様になっている．膨大かつ多様な背景を持つデータに対して単一の管理主体のみでデータ管理を行うことは困難であり，関連するデータであっても複数の管理主体（以降ピアと呼ぶ）がそれぞれ独立してデータを管理していることも多い．一方，適切なデータ分析や管理の効率化などのために複数のデータベース間でデータ共有を行いたい場合があるが，それぞれのピアが同じ形式でデータを保持・公開しているとは限らないため，管理主体のデータ保持形式に依存せずに複数ピアによって共同管理が行えるシステムが要求されている [1].

関連するデータが複数のピアによって分散的に管理されている状況では，あるピアにとって関連するデータがどこにどのように存在するかの全体像の把握が困難な場合がある．このような場合でも，把握できる範囲で局所的な信頼関係に基づいてデータ共有を行うことは有用であると思われるが，特にそのような際には，自身のデータの公開（データの輸出）および関連するピアからのデータの取り込み（データの輸入）に関して条件を設定して輸出入するデータをフィルタリングできることが望ましい [2]．さらに，各ピアが同一のデータに対して異なる視点を持っている場合，同一のデータに対して各ピアが異なる値を持ち得る．そのような状況でどのようにデータを協調的に共有していくのかは重要な課題である．複数のピアが自律分散的にデータを管理している場合としては例えば以下のような状況が考えられる．

a）　研究・科学データおよびそのメタデータの共有

専門性の高い研究データや科学データはデータを作成した組織やプロジェクトで管理され，機関リポジトリや分野リポジトリに格納・公開されている場合がある．さらに，データに対してその説明を行うメタデータが付与されていることが一般的になっている．関連するデータであっても異なるピアによって管理されている状況であるといえるが，データを共有することでより適切なデータの利用や分析を行うことが考えられる．また，ピア間でメタデータを共有することで，データの発見性が高まることが期待できる．

この際，データ利用規約の問題や管理上の問題から自身が管理するデータ（もしくはメタデータ）の中で特定のデータのみ共有したい（データを輸出したい）場合や，自身のデータベースの特性を考慮して特定のデータや特定の相手からのみデータを輸入したい場合がある．さらに，組織間のポリシーの違いや分野間の文化の違いから同じデータに対して異なる表現を用いて記述したい場合があると思われる．

b）　個々人が管理しているデータの共有

個々人が様々な情報をウェブサービス等を用いて管理することが一般的になっている．例えば，カレンダーサービスを用いることで予定を管理したり，メモやブックマークの情報をオンラインで管理することも多いだろう．また，研究者が自分の研究に関係する論文等の書誌情報をツールを用いて管理するといった場合も考えられる．

論文書誌情報管理の例では，例えば各研究者が論文に対してキーワードを付与しながら管理し，その情報を他の研究者と共有することで関連研究調査の効率化を行うことが考えられる．この状況において，例えば「後で読む」などといった特定のキーワード（もしくは特定のキーワードが付与された論文書誌情報自体）は共有したくないであるだとか，特定の研究者が付与したキーワード（もしくは特定の研究者が共有している論文書誌情報自体）のみを自身のデータベースに取り込みたい（データを輸入したい）などといったニーズがあると思われる．さらに，他の研究者が付与したキーワードを自身のデータベースに取り

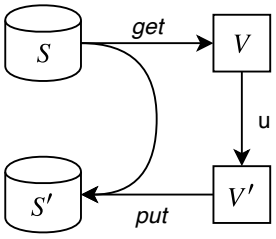

図 1　双方向変換

込んだ後に，そのキーワードを自分の言葉で書き換える（そしてさらにその書き換えたキーワードを共有する）といった状況も考えられる．

我々はこのような状況に対して適切かつ柔軟に対応するために，自律分散型データ共有・更新システムである SKY [3] を提案してきた．本論文では，我々がこれまでに提案してきた SKY について，既存のアーキテクチャである Dejima を用いて実現する方法について議論する．

本論文の構成は以下の通りである．まず，2 節で関連研究を紹介した後に 3 節で SKY の概要を説明する．4 節では実装に利用する Dejima について紹介し，Dejima を用いた SKY の実現方法を具体例を用いて説明する．5 節で残された課題について議論し，最後に 6 節で本論文をまとめる．

2 関 連 研 究

本節では関連研究として，本研究で用いる要素技術である双方向変換について 2.1 節で述べた後，自律分散データ共有に関する関連研究を文献 [4] に基づき 2.2 節で概観する．

2.1 双方向変換

双方向変換 (Bidirectional Transformation, BX) [5] とは，ソースデータをターゲットデータに変換した後，ターゲットデータ上の更新をソースデータに反映させることが可能な計算の枠組みのことであり，データベースのビュー更新問題を解決する手法として注目されている [6], [7].

双方向変換は，図 1 に示すように，順方向変換 *get* と逆方向変換 *put* の対で構成されている．順方向変換 *get* は，ビュー定義であり，ソースデータベース S に対する問合せで，ビュー V を結果として返すものである．その一方で，逆方向変換 *put* は，ビューへの更新をソースデータに反映させるもので，元のソースデータベース S と，u によって更新されたビュー V' の二つを引数として，更新されたソース S' を返すものである．

ソースデータベースとビューの整合性を保証するための主な性質に，GetPut と PutGet がある．[1]

$$\forall\, S, \qquad put\,(S,\, get(S)) = S \qquad \text{(GetPut)}$$

$$\forall\, S,\, V', \qquad get\left(put\left(S,\, V'\right)\right) = V' \qquad \text{(PutGet)}$$

1：双方向変換の性質には他にも様々な種類が議論されている [8].

(GetPut) は，ビューが更新されていなければソースは更新されないことを保証している．その一方で，(PutGet) は，ビューに対する更新の全てはソースに反映されるので，更新されたソースに *get* を適用すると更新されたビューを得ることができることを保証している．

双方向変換の構成要素である *get* と *put* を構築するには二つのアプローチがある．一つは，ビュー定義 (*get*) から *put* を導出する方法 (get_based bx) と，もう一つは，*put* から *get* を導出する方法 (put_based bx) である．ビュー定義は，直感的で書きやすい反面，一般にビュー定義は単射 (injective) でないため，get_based bx では 複数の *put* が候補となりうる．このことが，データベースにおけるビュー更新問題を難しくしている．その一方で，put_based bx では，well_defined な *put* から高々一つの *get* を導出できることが示されている [9] ので，*put* さえ記述すれば良いが，well-defined な *put* の定義が *get* に比べて難しい．最近，この *put* の書きにくさを，データログを用いることで解決する枠組みが提案されている [10].

2.2 自律分散データ共有

自律分散データ共有に関する研究のうち，複数のピア間でデータベースの更新を交換する研究は Collaborative Data Sharing System (CDSS) [2] から始まり，最近では Dejima [11]，BCDS Agent [12], OTMS Core [13] などが提案されている．本研究 (SKY) との比較を図 2 に示す.

CDSS は BX を利用していないが，他の全ての比較対象は BX を利用している．GetPut と PutGet の性質を持つ BX を利用することで，CDSS では問題とされていた更新伝播の副作用問題が解決されている．更新伝播の副作用とは，ある値の更新がピア間のマッピングを通じて自分自身に影響を及ぼすことであり，CDSS ではこの問題に対する一般的な解決手法が提案されていない.

また，CDSS の更新対象は自身のデータだけであるが，他の全ての比較対象は自身のデータだけでなく他から取り込んだデータの更新も可能となっている．他から取り込んだデータの更新を許すことで，同じデータに対する更新衝突が発生する場合がある．この衝突を回避する手法を，SKY では来歴情報を，Dejima では 2 相コミット (2PC) と 2 相ロック (2PL) を，BCDS Agent では first-writer-wins(FWW) を，OTMS Core では operational transformation (OT) [14] を，それぞれ用いて解決している．なお，CDSS では同じ意味を持つデータを取り込む際に，既に当該データを自分自身で所有している場合がある．この時に取り込むデータと所有しているデータ間での衝突回避は来歴情報を用いている.

一貫性については，SKY，CDSS は複数バージョンで対応している一方で，Dejima は strong consistency を，BCDS Agent と OTMS Core は eventual consistency を，それぞれ採用している.

なお，Dejima はピア間の BX の合成が co-tagetial であり SKY は co-sourcial であるが，4 節で述べるように，SKY の構成要素の一つを特別なピアと見なすことでこの特別なピアと通

	データ型	ピア間の BX の合成	連携	一貫性	衝突回避	更新対象
SKY	SQL 表	co-sourcial	緩い	複数バージョン	来歴を利用	自身のデータ+他から取り込んだデータ
CDSS [2]	SQL 表	N/A	緩い	複数バージョン	来歴を利用	自身のデータ
Dejima [11]	SQL 表	co-targetial	きつい	strong	2PC+2PL	自身のデータ+他から取り込んだデータ
BCDS Agent [12]	全て	sequential, split, merge	きつい	eventual	FWW	自身のデータ+他から取り込んだデータ
OTMS Core [13]	木構造	co-sourcial, co-targetial	きつい	eventual	OT-base	自身のデータ+他から取り込んだデータ

図 2　関連する自律分散データ共有との比較表

常のピア間の合成が co-targetial になるため，Dejima を利用して SKY を実現することが可能となる．

また，複数バージョンを管理する手法として近年データベーススキーマの共存に関する研究がいくつか提案されている [15], [16], [17]．これらの研究では，複数のスキーマの共存を考えているが，SKY ではインスタンスとしての複数バージョンの共存を考えている．

3　SKY アーキテクチャ

SKY [3] は我々が提案している自律分散型データ共有・更新システムであり，分散的に管理されているデータを各ピアの共有ポリシーに従って共有するための仕組みである．SKY の構成イメージを図 3 に示す．図 3 では P1, P2, P3 の三つのピアがデータ共有を行っている状況を示している．さらに，各ピアは複数の共有に参加する場合もあり，P2 は P4 とも別のデータ共有を行っている．

SKY では共有に参加するピアが共通してアクセス可能なピアとして共有リポジトリ SR (Shared Repository) を導入し，各ピアが共有したいデータは SR 中の共有テーブル ST (Shared Table) に格納される．各ピアが管理する情報は基礎テーブル BT (Base Table) に格納されており，各ピアは制御テーブル CT (Control Table) を通して輸出したいデータおよび輸入したいデータを制御する．

データの輸出および輸入，つまり BT と CT 間のやり取りおよび CT と ST 間のやり取りは双方向変換を用いることで明確な定義を行う．ST には共有に参加するピアが輸出した全てのデータが格納されており，双方向変換をビュー定義として捉えると，CT は BT のビューであると同時に ST のビューでもある．双方向変換を用いることで BT で発生したデータの更新は CT を通して ST に反映され，ST で発生したデータの更新は CT を通して BT に伝播される．

さらに，特定のピアからのみデータを輸入したいといった輸出入の条件指定を実現するために，BT, CT, ST 中の各データについて来歴 p (provenance) の情報を付加して管理する．輸出入の条件に単純なピア指定だけではなく,「他のピアによる更新が行われていないデータを輸入したい」などといった更新のパターンを用いたい場合もあると思われるため，我々は更新を考慮した来歴 p を以下のように定義することにした．

$$
\begin{array}{rcll}
p & ::= & tid@pid & (* \text{ ground tuple } *) \\
 & | & p * p & (* \text{ join } *) \\
 & | & p + p & (* \text{ duplicates by union or projection } *) \\
 & | & p@pid & (* \text{ updated in pid } *)
\end{array}
$$

tid はタプルを一意に識別する ID であり，pid はピアを一意に識別する ID である．例えば，ピア P1 中に存在している tid が 1 のタプルの p は $1@P1$ となり，そのタプルがピア P2 に輸入されて，P2 で更新が行われたとするとそのタプルの p は $1@P1@P2$ となる。このような p を用いることでそのタプルがもともとどのピアにあり，どのように更新されたかを判別することができ，柔軟な輸出入の条件指定が可能となる．

さらに，ST 中のデータについては受け入れ acc (acceptance) に関する情報を付加して管理する．acc はそのタプルを保持しているピアの pid が格納される．なお，データの再輸入に関する制御を行うため，各ピアがデータを削除した際には ST のデータの acc に “-” を付けて管理することにした．また，更新した際には古いデータは “!” を付けて管理することにした．これらの具体例は 4.3 節で示す．

4　Dejima を用いた実装

4.1　Dejima アーキテクチャ

Dejima [11] は双方向変換を利用したデータ共有アーキテクチャでり，その実装も利用可能である [2]．Dejima の構成イメージを図 4 に示す．図 4 では P1 と P2 がデータ共有を行い，さらに P2 と P3 がデータ共有を行っている状況を示している．

Dejima では各ピアが管理する情報は基礎テーブル BT (Base Table) に格納されており，Dejima テーブル DT を双方向変換を用いて定義し，DT を通して他のピアとデータ共有を行う．図 4 における P1 と P2 のデータ共有では，P1 が P2 と共有したいデータは Dejima テーブルである DT_{12} に格納され，P2 が P1 と共有したいデータは DT_{21} に格納される形になっており，DT_{12} と DT_{21} を同期することでデータ共有を行うアーキテクチャになっている．双方向変換には BIRDS [10] を利用しており，データログを用いて定義する．

Dejima では Dejima テーブルの単純同期によってデータ共有を行うため，共有相手のピアが Dejima テーブルに格納したデータは必ず受け取らなければならない．そのため，信頼に基づいてデータ共有を行う仕組みであるといえる．

2：https://github.com/ekayim/dejima-prototype

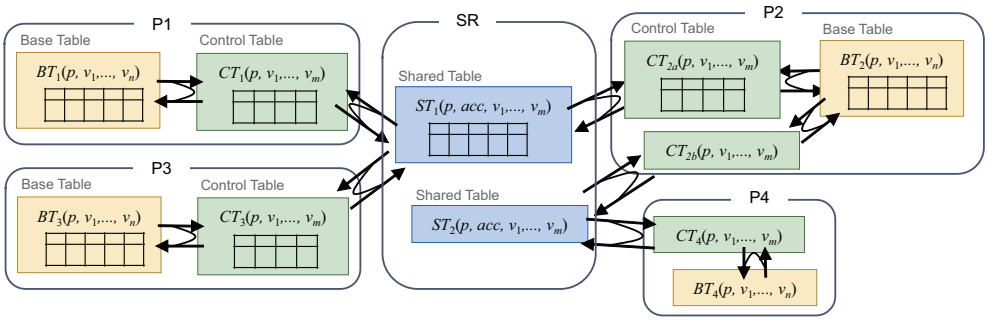

図 3　SKY の構成.

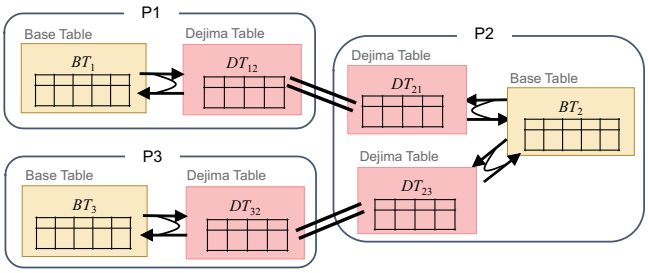

図 4　Dejima の構成.

4.2　SKY と Dejima の関係

図 3 で示した SKY の概念イメージでは CT は BT のビューであると同時に ST のビューでもある形で示していたが，以下のように考えることで Dejima を用いて SKY を構築することが可能だと思われる．

（1）　SR も一種の特殊なピアである．

（2）　BT のビューである CT と ST のビューである CT があり，これら二つの CT が同期されていればよい．

図 3 で示した SKY の構成を Dejima を用いて構築した場合，図 5 のようになる．CT を Dejima テーブルを用いて実装している．Dejima テーブルで構築された CT には各ピアが輸出したいデータおよび輸入したいデータの和集合 (union) が格納されるようになっていればよい．

4.3　実 装 の 例

具体的に P1, P2, P3 の三つのピアがデータ共有を行う例について論文情報をキーワードを付与しながら管理する状況を想定して説明する．各ピアは図 6 のデータを初期データとして保有しているものとし，表 1 の共有ポリシーを持っているものとする．なお，単純化のため BT のスキーマは全てのピアで同一となっているが，双方向変換を用いることでスキーマが異なる場合にも対応可能である．また，Dejima を用いた実装を行うにあたり，p の扱いを容易にするために p の持つ情報の一部を分解格納することにし，tid にタプル ID の情報，org にそのデータを作成したピアのピア ID の情報，lu にそのデータを最後に更新したピアのピア ID の情報をそれぞれ格納することにした．

この状況で SKY を用いてデータ共有を行う場合，表 1 に示した各ピアの輸出ポリシーおよび輸入ポリシーを考慮すると SR 上の ST は図 7 のようになり，また，例えば P1 の BT および CT は図 8 のようになることが期待される．

表 1　各ピアの共有ポリシー

ピア	共有ポリシー
P1	paper が “A” のデータのみ輸出する．P2 が作成したデータのみ輸入する．
P2	全てのデータを輸出する．P1 が作成したデータのみ輸入する．
P3	全てのデータを輸出する．全てのデータを輸入する．

Dejima を用いて SKY を実装するにあたっては，それぞれの双方向変換をどのように用意するかが焦点の一つとなる．各ピアの BT で発生した更新（挿入および削除を含む）が CT を通して ST に伝播されなければならない．また，他のピアから ST にデータが挿入された際にそれが自身の輸入ポリシーに合致するものであれば CT を通して BT に伝播されなければならない．さらに，その際に ST における acc の情報を適切に設定しなければならない．

ST における acc の設定に関しては SR にトリガーを設定することで対処可能だと考えた．輸入条件に合致するタプルが ST に挿入された際に acc の値を自らのピア ID の値に置き換えた当該タプルのコピーを ST に挿入するようなトリガーをピアごとに設定する．SR における P1 の輸入のためのトリガーを図 9 に示す．10 行目に輸入条件を記述している．また，8-9 行目には削除したタプルの再輸入を防ぐための条件を記述している．さらに削除，更新の際の acc の処理のために図 10 のトリガーも設定する．BIRDS は更新を削除と挿入の組合せで扱い，処理の順序として削除の後で挿入が行われることに注意されたい．図 9 および図 10 のトリガーを設置することでデータの削除および更新の際に acc の値を更新し，削除の際には acc に “-” を付与し，更新の際には古いデータの acc に “!” を付与することができる．

各ピアの BT と CT 間に設定する BX としては，CT が輸出条件を満たすタプルと輸入条件を満たすタプルの和集合となるよう記述する．ただし，輸入後に自身が輸入したタプルを更新した場合に更新後のタプルを CT に反映すべきかは輸出条件に依存するため，輸入に関する条件には「自身以外が最終更新者である」という条件も追加する．P1 における BT_1 と CT_1 間の BIRDS による BX 定義を図 11 に示す．下線部に輸出条件と輸入条件を記述することでこの BX 定義を作成できる．

SR における ST と CT 間に設定する BX としては，acc の

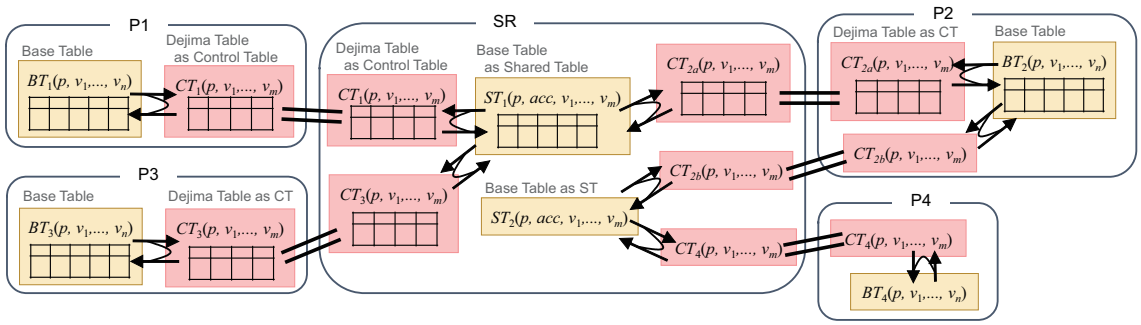

図 5　Dejima を用いた SKY の構成.

BT of P1

p	tid	org	lu	paper	keyword
1@P1	1	P1	P1	A	Data Exchange
2@P1	2	P1	P1	B	Data Cleaning

BT of P2

p	tid	org	lu	paper	keyword
3@P2	3	P2	P2	C	Public Key Encryption
4@P2	4	P2	P2	C	Digital Signature

BT of P3

p	tid	org	lu	paper	keyword
5@P3	5	P3	P3	D	Social Network
6@P3	6	P3	P3	E	Crowdsourcing

図 6　共有前の各ピアの BT

ST

p	tid	org	lu	acc	paper	keyword
1@P1	1	P1	P1	P1	A	Data Exchange
1@P1	1	P1	P1	P2	A	Data Exchange
1@P1	1	P1	P1	P3	A	Data Exchange
3@P2	3	P2	P2	P1	C	Public Key Encryption
3@P2	3	P2	P2	P2	C	Public Key Encryption
3@P2	3	P2	P2	P3	C	Public Key Encryption
4@P2	4	P2	P2	P1	C	Digital Signature
4@P2	4	P2	P2	P2	C	Digital Signature
4@P2	4	P2	P2	P3	C	Digital Signature
5@P3	5	P3	P3	P3	D	Social Network
6@P3	6	P3	P3	P3	E	Crowdsourcing

図 7　共有後の ST

値が対応するピアのピア ID に一致するものが CT に格納されるよう記述する．SR における ST_1 と CT_1 間の BIRDS による BX 定義を図 12 に示す．トリガーによって *acc* の値が適切に設定されるため，こちらの BX は単純に実現できる．

このような BX およびトリガーを設定することで各ピアの BT で発生するデータの挿入，削除，更新が共有ポリシーに基づいて他のピアの BT に反映される．例えば，P2 で P1 から輸入した p が 1@P1 のタプルを削除すると ST は図 13 のようになる．さらに，その後に P1 で 1@P1 のタプルに関し，キーワードを “Data Exchange” から “Data Integration” に更新すると ST は図 14 のようになる．

BT of P1

p	tid	org	lu	paper	keyword
1@P1	1	P1	P1	A	Data Exchange
2@P1	2	P1	P1	B	Data Cleaning
3@P2	3	P2	P2	C	Public Key Encryption
4@P2	4	P2	P2	C	Digital Signature

CT of P1

p	tid	org	lu	paper	keyword
1@P1	1	P1	P1	A	Data Exchange
3@P2	3	P2	P2	C	Public Key Encryption
4@P2	4	P2	P2	C	Digital Signature

図 8　共有後の P1 の BT および CT

5　議　　論

本節では本論文で議論した Dejima を用いた SKY 実装における課題について議論する．

5.1　acc の情報を用いた輸出入の条件指定

acc の情報を用いることで例えば P1 が「P2 と P3 が両方とも輸入しているデータだったら輸入したい」などといった輸入条件を指定することが考えられるが，現状の実装では *acc* を用いた指定に対応していない．

現在の実装では *acc* は ST のみに格納されているが，*acc* を用いた条件指定を実現するためには BT 側にも *acc* の情報を格納する必要があると思われる．格納方法および双方向変換の記述方法が課題である．

5.2　暗黙的に仮定している挙動

SKY は各ピアが自律的にデータを管理するため，いったん輸出したデータは自身のピア内では削除したとしても他のピアが保有していることがあり得る．本実装では削除したタプルは ST の *acc* の値について *pid* に “-” を付けて管理している．この情報を用いて，過去に削除したタプルと同一の *tid* を持つタプルの輸入を防いでいる．4.3 節で示した削除後の更新の例では，P2 が削除した *tid* が 1 のタプルについて P1 がその後に更新を行っているが，P2 にとっては自身が削除したタプルに対する更新であるため更新後のタプルを輸入することは無い．しかし，このように削除したタプルを他のピアが保持しており，

```
CREATE OR REPLACE FUNCTION public.sr_ct1_acc()
RETURNS trigger AS $$
  BEGIN
  IF (NEW.acc != 'P1' AND
      NEW.p NOT IN
        (SELECT p FROM public.bt WHERE acc = 'P1') AND
      substring(NEW.acc, 1, 1) != '-' AND
      (SELECT p FROM public.bt WHERE tid = NEW.tid AND
      acc = '-P1') IS NULL) THEN
    IF (NEW.org = 'P2') THEN
      IF ((SELECT p FROM public.bt
          WHERE tid = NEW.tid AND acc = 'P1')
          IS NOT NULL) THEN
        UPDATE public.bt SET acc = '!P1'
          WHERE tid = NEW.tid AND acc = 'P1';
        RAISE LOG
          'A tuple with acc "P1" updated to "!P1"';
      END IF;
      INSERT INTO bt VALUES
        (NEW.p, NEW.tid, NEW.org, NEW.lu,
        'P1', NEW.paper, NEW.keyword);
      RAISE LOG 'A tuple with acc "P1" inserted';
    END IF;
  END IF;
  RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER sr_ct1_acc_trigger
  AFTER INSERT
  ON public.bt
  FOR EACH ROW EXECUTE PROCEDURE public.sr_ct1_acc();
```

図 9　SR における P1 の輸入のためのトリガー

```
CREATE OR REPLACE FUNCTION public.sr_acc_delete()
RETURNS trigger AS $$
  BEGIN
  INSERT INTO bt VALUES
    (OLD.p, OLD.tid, OLD.org, OLD.lu, '-' || OLD.acc,
    OLD.paper, OLD.keyword);
  RAISE LOG 'A tuple with acc "-" inserted';
  RETURN OLD;
  END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION public.sr_acc_insert()
RETURNS trigger AS $$
  BEGIN
  IF ((SELECT p FROM public.bt
      WHERE tid = NEW.tid AND acc = '-' || NEW.acc)
      IS NOT NULL) THEN
    UPDATE public.bt SET acc = replace(acc, '-', '!')
      WHERE tid = NEW.tid AND acc = '-' || NEW.acc;
    RAISE LOG 'A tuple with acc "-" updated to "!"';
  END IF;
  RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER sr_acc_delete_trigger
  AFTER DELETE
  ON public.bt
  FOR EACH ROW EXECUTE PROCEDURE public.sr_acc_delete();

CREATE TRIGGER sr_acc_insert_trigger
  AFTER INSERT
  ON public.bt
  FOR EACH ROW EXECUTE PROCEDURE public.sr_acc_insert();
```

図 10　SR における *acc* 処理用のトリガー

他のピアによって更新された際に，その更新されたタプルを輸入したい状況もあり得るかもしれない．

また，本実装では同じ *tid* のタプルは各ピアで一つしか保持できないことが前提になっている．そのため，複数のピアで同じ *tid* のタプルが更新された場合には最後の更新を（輸入ポリシーに基づいて）輸入することになる．しかし，同じ *tid* のタプルに対して異なるピアで行われた異なる更新を両方とも輸入したい状況もあり得るかもしれない．このような挙動を切り替えるような実装を行うことが考えらえる．

5.3　補助情報の格納方法

SKY では共有ポリシーを反映したデータ共有を実現するために p および *acc* を補助情報として格納している．現在の実装では p に関する情報は BT に一体化する形で格納しているが，BT と分離して格納することで各ピアのアプリケーション等が SKY を意識することなく BT を参照できるようにすることが考えられる．また，ST における *acc* についても分離して管理することで格納の効率化を行うことが考えられる．

さらに，本実装では p の扱いを容易にするために *tid*, *org*, *lu* の情報を p とは別に格納しているが，ユーザ定義関数を利用することや，p を展開して格納するなど，p の処理方法には検討の余地がある．

5.4　利用者インタフェース

各ピアが BT に対して挿入，削除，更新の操作を行った際に，各ピアは p を適切に処理しなければならないが，そのような処理は各ピアでデータを処理する利用者からは隠蔽されるべきである．5.3 節の議論とも関連するが，p を BT から分離した上でトリガーを利用して p を更新することや，p の処理を踏まえて BT を操作するためのインタフェースを実装することなどが考えられる．

また，各ピアが輸出ポリシーおよび輸入ポリシーを定義するにあたり，データログによる BX 定義やトリガーを直接記述することは不適切であるため，ポリシーを宣言的に記述することでデータログやトリガーが自動生成される仕組みが必要だと考えている．図 11 の BT と CT 間の BX 定義や図 9 の SR にお

```
source bt('p':string, 'tid':string, 'org':string,
   'lu':string, 'paper':string, 'keyword':string).
view ct1('p':string, 'tid':string, 'org':string,
    'lu':string, 'paper':string, 'keyword':string).
% view definition
ct1(P,TID,ORG,LU,PP,K) :- bt(P,TID,ORG,LU,PP,K),
                          PP = 'A'.
ct1(P,TID,ORG,LU,PP,K) :- bt(P,TID,ORG,LU,PP,K),
                          ORG = 'P2', NOT LU = 'P1'.
% update strategies
-bt(P,TID,ORG,LU,PP,K) :- bt(P,TID,ORG,LU,PP,K),
                          NOT ct1(P,TID,ORG,LU,PP,K),
                          PP = 'A'.
-bt(P,TID,ORG,LU,PP,K) :- bt(P,TID,ORG,LU,PP,K),
                          NOT ct1(P,TID,ORG,LU,PP,K),
                          ORG = 'P2', NOT LU = 'P1'.
+bt(P,TID,ORG,LU,PP,K) :- NOT bt(P,TID,ORG,LU,PP,K),
                          ct1(P,TID,ORG,LU,PP,K),
                          PP = 'A'.
+bt(P,TID,ORG,LU,PP,K) :- NOT bt(P,TID,ORG,LU,PP,K),
                          ct1(P,TID,ORG,LU,PP,K),
                          ORG = 'P2', NOT LU = 'P1'.
```

図 11　P1 における BT_1 と CT_1 間の BIRDS による BX 定義

```
source bt('p':string, 'tid':string, 'org':string,
   'lu':string, 'acc':string, 'paper':string,
   'keyword':string).
view ct1('p':string, 'tid':string, 'org':string,
   'lu':string, 'paper':string, 'keyword':string).
% update strategies
-bt(P,TID,ORG,LU,ACC,PP,K) :-
            bt(P,TID,ORG,LU,ACC,PP,K),
            NOT ct1(P,TID,ORG,LU,PP,K), ACC = 'P1'.
+bt(P,TID,ORG,LU,ACC,PP,K) :-
            NOT bt(P,TID,ORG,LU,ACC,PP,K),
            ct1(P,TID,ORG,LU,PP,K), ACC = 'P1'.
```

図 12　SR における ST_1 と CT_1 間の BIRDS による BX 定義

いて各ピアの輸入を制御するトリガーは輸入条件および輸出条件をテンプレートに埋め込む形で容易に生成可能だと思われる．また，図 12 の ST と CT 間の BX 定義や図 10 の *acc* 処理用のトリガーは各ピアの輸出入条件によらず一律に記述可能である．

5.5 評価実験

p および *acc* の情報を用いて各ピアの共有ポリシーを反映したデータ共有を行うにあたり，実シーンにおいて行いたい共有が不足なく実現できるか評価したい．4.3 節で例に挙げた論文情報をキーワードを付与しながら管理するアプリケーションを SKY 上に構築し，実利用を通して評価することなどを検討している．

6 おわりに

本論文では我々がこれまでに提案してきた SKY アーキテクチャを既存のアーキテクチャである Dejima を用いて実装する方法について議論した．SKY で考えているような共有ポリシーを考慮したデータ共有については Dejima の自然な利用では想定されていなかったが，Dejima の利用方法を工夫することで SKY を実現することができた．

ST

p	tid	org	lu	acc	paper	keyword
1@P1	1	P1	P1	P1	A	Data Exchange
1@P1	1	P1	P1	P3	A	Data Exchange
3@P2	3	P2	P2	P1	C	Public Key Encryption
3@P2	3	P2	P2	P2	C	Public Key Encryption
3@P2	3	P2	P2	P3	C	Public Key Encryption
4@P2	4	P2	P2	P1	C	Digital Signature
4@P2	4	P2	P2	P2	C	Digital Signature
4@P2	4	P2	P2	P3	C	Digital Signature
5@P3	5	P3	P3	P3	D	Social Network
6@P3	6	P3	P3	P3	E	Crowdsourcing
1@P1	1	P1	P1	-P2	A	Data Exchange

図 13　削除後の ST

ST

p	tid	org	lu	acc	paper	keyword
3@P2	3	P2	P2	P1	C	Public Key Encryption
3@P2	3	P2	P2	P2	C	Public Key Encryption
3@P2	3	P2	P2	P3	C	Public Key Encryption
4@P2	4	P2	P2	P1	C	Digital Signature
4@P2	4	P2	P2	P2	C	Digital Signature
4@P2	4	P2	P2	P3	C	Digital Signature
5@P3	5	P3	P3	P3	D	Social Network
6@P3	6	P3	P3	P3	E	Crowdsourcing
1@P1	1	P1	P1	-P2	A	Data Exchange
1@P1@P1	1	P1	P1	P1	A	Data Integration
1@P1	1	P1	P1	!P1	A	Data Exchange
1@P1	1	P1	P1	!P3	A	Data Exchange
1@P1@P1	1	P1	P1	P3	A	Data Integration

図 14　更新後の ST

謝辞　本研究の一部は JSPS 科研費 JP17H06099, JP18H04093, JP18K11315 の助成を受けたものです．

文　献

[1] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[2] Grigoris Karvounarakis, Todd J. Green, Zachary G. Ives, and Val Tannen. Collaborative data sharing via update exchange and provenance. *ACM Trans. Database Syst.*, 38(3):19:1–19:42, 2013.

[3] 清水 敏之, 加藤 弘之, and 吉川 正俊. 科学メタデータクリーニングのための自律分散型データ共有・更新システムに向けて. *DEIM 2020*, D8-2.

[4] Yasuhito Asano, Yang Cao, Soichiro Hidaka, Zhenjiang Hu, Yasunori Ishihara, Hiroyuki Kato, Keisuke Nakano, Makoto Onizuka, Yuya Sasaki, Toshiyuki Shimizu, Masato Takeichi, Chuan Xiao, and Masatoshi Yoshikawa. Bidirectional collaborative frameworks for decentralized data management.

In *Proceedings of the Fifth Workshop on Software Foundations for Data Interoperability*, pages 13–51, 2021.

[5] Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. Introduction to bidirectional transformations. In *Bidirectional Transformations*, pages 1–28, 2018.

[6] Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. Relational lenses: A language for updatable views. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 338–347, 2006.

[7] 加藤 弘之, 胡 振江, 日高 宗一郎, and 松田 一孝. 『ソフトウェアサイエンスの基本』シリーズ第 4 回高談闊論：双方向変換の原理と実践. **コンピュータ ソフトウェア**, 31(2):44–56, 2014.

[8] Keisuke Nakano. A tangled web of 12 lens laws. In *Proceedings of the 13th International Conference on Reversible Computation*, pages 185–203, 2021.

[9] Sebastian Fischer, Zhenjiang Hu, and Hugo Pacheco. A clear picture of lens laws. In *Proceedings of the 12th International Conference on Mathematics of Program Construction*, pages 215–223, 2015.

[10] Van-Dang Tran, Hiroyuki Kato, and Zhenjiang Hu. Programmable view update strategies on relations. *PVLDB*, 13(5):726–739, 2020.

[11] Yasunori Ishihara, Hiroyuki Kato, Keisuke Nakano, Makoto Onizuka, and Yuya Sasaki. Toward BX-based architecture for controlling and sharing distributed data. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–5, 2019.

[12] Masato Takeichi. BCDS Agent: An architecture for bidirectional collaborative data sharing. *Computer software*, 38(3):41–57, 2021.

[13] Mikiya Habu and Soichiro Hidaka. Conflict resolution for data updates by multiple bidirectional transformations. In *Proceedings of the Fifth Workshop on Software Foundations for Data Interoperability*, pages 62–75, 2021.

[14] Sergey Sinchuk, Pavel Chuprikov, and Konstantin Solomatov. Verified operational transformation for trees. In *Proceedings of the 7th International Conference on Interactive Theorem Proving*, pages 358–373, 2016.

[15] Kai Herrmann, Hannes Voigt, Andreas Behrend, Jonas Rausch, and Wolfgang Lehner. Living in parallel realities: Co-existing schema versions with a bidirectional database evolution language. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1101–1116, 2017.

[16] Kai Herrmann, Hannes Voigt, Torben Bach Pedersen, and Wolfgang Lehner. Multi-schema-version data management: data independence in the twenty-first century. *VLDB J.*, 27(4):547–571, 2018.

[17] 田中 順平, Van-Dang Tran, 加藤 弘之, and 胡 振江. プログラム可能なスキーマの共存戦略の実現手法. **情報処理学会論文誌プログラミング（*PRO*）**, 14(5):15–33, 2021.