データベースシステム 116－25
（1998. 7. 8）

分散メソッドの提案

有次 正義　吉田 裕介　中村 知久　金森 吉成

群馬大学工学部情報工学科

{aritsugi, yosida, tomo, kanamori}@dbms.cs.gunma-u.ac.jp

データベースアプリケーションは普通分散環境で開発される．本稿では，永続オブジェクトのデータメンバだけでなくそのメソッドもデータベースに保存することを考える．メソッドそのものをデータベースに保存することによって分散環境で柔軟性のあるアプリケーション開発環境を得ることができる．本稿では，メソッドを保存し，ネットワーク上でメソッドを移動し，遠隔サイトでメソッドを実行する，分散メソッドを提案する．また，永続オブジェクトにメソッドを適用する方法として，永続オブジェクトを移動する方法とメソッドを移動する方法がある．データベースサーバが複数ある場合これら二つの手法をうまく組み合わせることによって処理効率が向上する．分散環境での，これら二つの手法の組合わせの処理効率に関する実験を報告する．さらに，メソッドそのものをデータベースに保存することによってメソッドの実装を共有することが可能となる．これにより，他の開発者によって開発されたメソッドを用いてメソッドに対するビューが考えられる．

Proposing Distributed Methods

Masayoshi Aritsugi　Yusuke Yoshida　Tomohisa Nakamura　Yoshinari Kanamori

Department of Computer Science, Gunma University

Database applications are usually developed in distributed environments. In this paper, we try to store not only data members but also methods of persistent objects in databases in order to support flexible application development in distributed environments. By storing methods, moving them over the network, and executing them on remote sites, distributed methods are implemented. Applying methods to persistent objects, we have to move persistent objects to sites that hold methods, or to move methods to sites that store persistent objects. We report experimental results of the two methods in a distributed environment. Storing methods in databases enables us to share them with others in new ways, e.g., we can define views with methods developed by others, and can apply them to data stored in the local and/or remote sites.

1 はじめに

計算機環境の発達により，ワークステーションやパーソナルコンピュータをネットワークで接続した分散環境が一般的になってきた．しかも，接続されている各計算機は比較的高機能で，データベースなどのアプリケーションプログラムは，データベースサーバ上ではなくクライアントサイトで開発されることが多くなっている．このような環境では，データを管理しているサイトと，そのデータに適用するメソッドの開発を行うサイトが異なる場合がある．つまり，ユーザの好きなときに，好きなサイトでメソッドおよびデータベースアプリケーションの開発が可能でなければならない．さらに，独自のアルゴリズムなどを用いた好きなメソッドコードの開発を可能とする柔軟性のある開発環境が望まれる．

メソッドの実装の変更が許される環境においては，実行するメソッドのあるサイトと，それを適用する永続オブジェクトが存在するサイトが異なっている場合の，効率の良いメソッドの処理を考えなければならない．ここで，メソッドの変更が動的に行われるとすると，メソッドを前もって分散配置させておくことはできない．

本稿では，これらを解決する手段として，我々が研究開発中の分散メソッドを提案する．これにより，我々は一つのメソッド／型に対し複数の実装が可能になる．さらに分散メソッドは，ネットワーク中を移動し，他サイトに存在する永続オブジェクトに適用し，その結果を得ることが可能となる．

型とその実装を区別し，一つの型に対し複数の実装を許す方法は既にいくつか提案されている [14, 16, 13]．また，メソッドやプロセスをネットワークを通して移動させ，実行する手法も多数提案されている [17]．これらと我々の研究の違いは，主に次の点である．

- 分散メソッドは，従来の分散オブジェクトデータベースの持つ機能を用いて実現されている．これにより，非常に容易にメソッドを移動させ，実行することが可能になる．
- メソッドにおけるビューが考えられる．

分散環境において，メソッドを，異なるサイトに存在する永続オブジェクトに適用して処理するには，大きく次の二つの手法が考えられる．

- 永続オブジェクトをメソッドのあるサイトへ移動させ処理する．
- メソッドを永続オブジェクトのあるサイトへ移動させ処理する．

本稿では，これらをそれぞれデータマイグレーション，メソッドマイグレーションによる処理と呼ぶ．

分散環境に対応するオブジェクトデータベースはいくつかある [5, 8, 10] が，これらは主にデータマイグレーションによる処理が実装されている．しかし，複数のサーバ・クライアントサイトからなる分散環境におけるデータベース処理の効率化のためには，データマイグレーションだけではなくメソッドマイグレーションによるメソッドの処理をうまく組み合わせて処理することが必要である [6]．本稿では，一つのクライアントが，自サイトで開発した分散メソッドを，分散しているデータベースサーバの管理している永続オブジェクトに適用する際の，メソッドマイグレーションとデータマイグレーションの効率を，実際に分散環境で複数のオブジェクトデータベースを用いてテストした結果を報告し，効率の良い二つの手法の組合わせについて議論する．

さらに，メソッドビューの概念を提案する．メソッドを，従来の永続オブジェクトのように，データベースに保存することによって分散メソッドを実現している．これにより，メソッドの実装を分散環境下で複数のユーザで共有することが可能となる．これにより，オブジェクトの持つデータに対するビューだけでなく，メソッドに対するビューを考えることが可能となる．

本稿の構成は，以下の通りである．2. では分散メソッドの概要を述べる．次に，3. でメソッドマイグレーションとデータマイグレーションについて考察し，実験結果を報告する．4. でメソッドビューについて論じる．5. で関連研究と比較し，6. で本稿のまとめを述べる．

2 分散メソッド

アプリケーションで扱う通常のオブジェクトは，アプリケーションの終了により消滅するが，永続オブジェクトは，そのオブジェクトを生成したアプリケーションが終了しても2次記憶中に保存され，明示的に削除されるまで様々なアプリケーションにより再利用されることになる．その際，永続オブジェクトの持つ属性データは変更されることもある．

このように，永続オブジェクトの寿命が長い場合，時間の経過にともないそのオブジェクトの持つ属性データの値や，オブジェクトの型やスキーマの変更 [3, 12, 2] だけでなく，メソッドの実装が変更

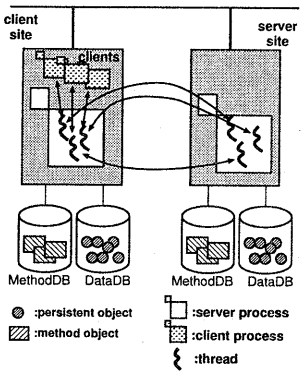

図 1: 分散メソッドのプロセス構造

されることもある．我々は，プログラマが任意の時点で永続オブジェクトの持つメソッドを変更できる分散メソッドを設計・開発中である [18].

図 1に，開発中の分散メソッドのプロセス構造を示す．図 1に示すように，分散メソッドをサポートする各サイトで，そのサイトの保持するデータベースのためのサーバプロセスが動作しており，クライアントは，自サイトのサーバプロセスと通信し，必要なサイトのサーバ同士で通信することによって分散データベース処理を実現している．サーバプロセスは，複数のクライアントプロセス，および他サイトのサーバプロセスからのリクエストのために，スレッドを生成して効率よく処理されるように実装されている．

図 1では，クライアントとサーバサイトは，ともに MethodDB，DataDB と呼ばれる二つのデータベースを保持している．DataDB は，従来の永続オブジェクトを管理しているデータベースのことである．MethodDB は，分散メソッドシステムにより導入されたもので，プログラマにより開発されたメソッドが保存・管理される．このように，永続オブジェクトのもつデータだけでなく，メソッドをデータベース化することにより，ネットワークにつながれた複数のサイトで開発された，より良い実装を持つメソッドを共有することが可能になっている．

我々は，C++により ObjectStore[8] を用いた環境での分散メソッドの実装を行ってきた．データとは異なり，メソッドは，処理するサイトに移動させた後，そのサイトで動くプロセスに動的にロードし，実行しなければならない．この動的ロー

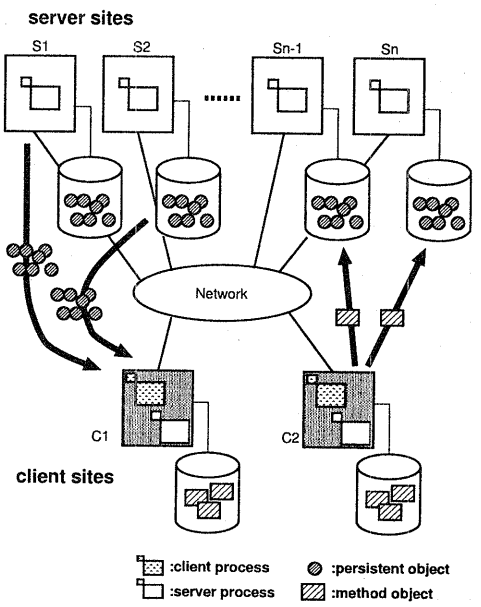

図 2: 分散データベース環境の例

ディングは，C++ を用いる環境では Solaris の提供する`dlopen()` と`dlsym()`[15] を用いて実装している．

C++ を用いた実装では，コンパイラ言語であるための効率的な処理が可能であるが，サイト間の引数や処理結果の受け渡しの実装が困難になる．また，現在の実装では Solaris の動く単一の環境でなければならないなどの制限がある．このため現在では分散メソッドを Java[7] により PSE[9] を用いた環境で実装を行っている．この場合，動的ローディングは，Java の class loader の機能を使う．

3 データマイグレーションとメソッドマイグレーション

1. で述べたように，メソッドを，他サイトの管理する永続オブジェクトに適用して処理するには，データマイグレーションとメソッドマイグレーションの二つが考えられる．3. では，これらについて，それぞれの効率と，それらを組み合わせたときの分散データベース処理の効率について議論する．

図 2に，本稿で用いる分散データベース環境の例を示す．図 2で，サーバサイトの MethodDB とクライアントサイトの DataDB は省略している．

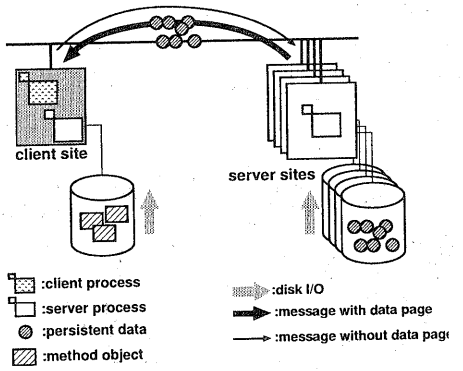

図 3: データマイグレーション

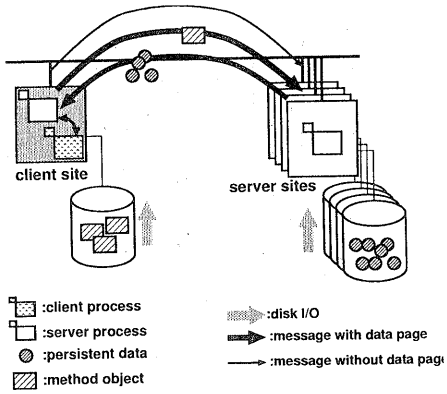

図 4: メソッドマイグレーション

3.1 データマイグレーション

データマイグレーションは，サーバサイトの永続オブジェクトを，適用させるメソッドを持つクライアントサイトに移動させ，クライアントサイトで処理を行う手法である．図 2では，サーバサイト S_1， S_2 とクライアントサイト C_1 間でデータマイグレーションによる処理を行う様子を示している．この手法では，メソッドの処理がクライアントサイトで行われるため，サーバサイトの負荷は少なくてすむが，複数のサーバサイトにある永続オブジェクトを処理する場合には，クライアントの負荷が多くなってしまう．また，処理に必要なオブジェクトのサイズに対し，処理結果のサイズが小さい場合，通信コストが大きくなってしまう（図 3参照）．

3.2 メソッドマイグレーション

メソッドマイグレーションは，クライアントサイトで開発・実装されたメソッドを，処理する永続オブジェクトを管理するサーバサイトへ移動させ，メソッドの処理をサーバサイトで行う手法である．図 2では，サーバサイト S_{n-1}， S_n とクライアントサイト C_2 の間でメソッドマイグレーションによる処理を示している．この手法のコストは，データマイグレーションでの処理のコストと対照的になる．すなわち，メソッドの処理がサーバで行われるため，クライアントサイトの負荷は少なくてすむが，一つのサーバに対し複数のクライアントが処理をする場合，サーバサイトの負荷が増大してしまう．また，処理に必要なオブジェクトのサイズに対し，処理結果のサイズが小さい場合，通信コストが少なくてすむことになる（図 4参照）．

このように，データマイグレーションに比べてメソッドマイグレーション処理の方がコストがかからない場合が存在するが，今までの分散オブジェクトデータベースシステムでは主にデータマイグレーションによるサーバ・クライアント処理が実現されてきた．

3.3 実験

我々は，データマイグレーション・メソッドマイグレーションを組み合わせた分散データベース処理実験を行った．実験には，10Mbps のイーサネットで接続された 4 台の SUN ワークステーションを用いた（表 1）．サイト $S_i(i = 1, 2, or\ 3)$ はサーバサイト，サイト C はクライアントサイトとして用いた．データベースを構築したあとに，高機能なワークステーションをネットワークで接続し，クライアントとして使用する環境を設定し，実験では，クライアントよりも性能の低い S_1， S_2 と，高い S_3 をサーバサイトとして選択した．実験では，これら 4 台の計算機は実験の処理のみを行い，その他の負荷をかけずに行った．

実験のため，我々は `name`， `age`， `ssn`， `salary`， `x` と `y` の属性をもつクラス `Employee` のオブジェクト集合を用いた．この属性の数は，OO1 ベンチマーク [4] の `Part` の持つ属性のうち，リスト構造である `to` と `from` を除いた個数により決定した．属性 `age` は，0 から 99 の範囲のランダムに生成された整数で，実験では，`Employee` オブジェクトは生成順に集合オブジェクトに挿入され，インデックス等は考えていない．

生成した集合オブジェクトは，ObjectStore の

表 1: 実験に用いたワークステーション

Site	S_1, S_2	C	S_3
Type	Ultra1	Ultra30	Ultra30
Clock	167MHz	248MHz	296MHz
Memory	128MB		
Disk	SUN 2.1GB	SUN 4.2GB	
Page size	8192		
Compiler	SPARCompilers 4.1 C++		
DBMS	ObjectStore 5.0.0.0		

提供する集合クラスを用いて，os_Set<Employee*>として実装した．実験では，ある年齢以下のEmployeeオブジェクトの持つsalaryの平均値を求めるメソッドを，サイトCで開発・実装し，サイトS_1，S_2およびS_3で管理されている集合に対し処理した．[1]ここで，各サーバサイトの持つデータベースのサイズはすべて同じに設定した．

本稿ではC++による実装を用いて議論をすすめる．現在のC++の実装では，サーバサイトでメソッドを処理した際の処理結果の転送が困難であるため，本実験では結果の返り値のサイズを0とした．Javaによる実装を考える場合，C++の場合と比較してJavaでのメソッドのサイズは対象となる永続オブジェクトのサイズに比較すると一般的に小さいため，以下の議論のうちメソッドの転送コストを処理結果の転送コストと置き換えればよい．

実験では，移動するデータおよびメソッドのキャッシュへの保存は考慮しなかった．つまり，処理するときはかならずデータおよびメソッドの移動が生じるようにして実験を行った．

図5が実験結果である．図5中の組合わせは，クライアントサイトCと，サーバサイトS_3，S_2，およびS_1の間で，データマイグレーションを行う場合はdを，メソッドマイグレーションによる処理を行う場合はmとして表現されている．例えば，dmdは，CとS_3およびS_1との間の処理をデータマイグレーションで，S_2との間の処理をメソッドマイグレーションでおこなう組合わせを示している．各組合わせの計算は，実験を行ったデータサイズに対して行った．

図5より，従来オブジェクトデータベースが提供してきたデータマイグレーションのみによる分散処理がもっとも効率が悪いことがわかる．今後は，処理コストを評価できるモデルの構築を考えていく予定である．

[1]このメソッドの実装が，メソッドの変更を許す分散メソッドシステムでの典型的なものであるとは考えにくいが，実験における実際のデータ転送量や処理に必要なデータ量などが簡単に求められるため，ここではこれを用いた．

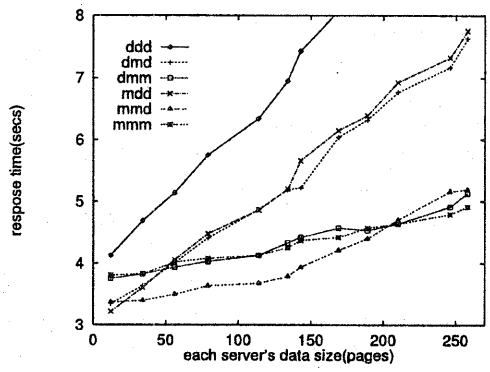

図 5: 測定結果

4 メソッドビュー

分散メソッドによって，1つのメソッドが複数の実装を持つことができる．また，あるサイトで開発したメソッドを別のサイトへ移動させ，実行することも可能である．分散メソッドは，分散データベースによって管理されるため，ネットワークで接続した計算機で分散メソッドの実装を共有することができる．つまり，ネットワーク上のあるサイトで開発・保存されたメソッドの実装を，必要に応じて検索することができる．そして，メソッドの開発・保存を行ったサイトとは異なるサイトでそのメソッドを永続オブジェクトに適用することができる．ここでは，分散メソッドを用いたメソッドに対するビューを提案する．また，分散メソッドを用いたデータベースアプリケーションの構成について考察する．

分散メソッドの概念は，データベースビューの特徴と似ている点がある．関係データベースでは，ユーザはビューを定義することにより，データの形がまるでビューのそれであるように扱うことが可能となる．これと同じように，分散メソッドを用いることにより，ユーザは独自のメソッドの実装を作ることができる．分散メソッドは，ユーザがメソッドをコーディングする一種のビューのように考えられる．つまり，分散メソッド自身が，オブジェクトデータベースでのメソッドビューの特徴を備えていることになる．

多くの研究者がオブジェクトデータベースにビューの仕組みを統合しようと試みてきた [1, 11]. 彼らは仮想クラスとしてビューを定義し，その仮想クラスを一つのクラス階層へ統合することを考えてきた．しかし，永続オブジェクトの持つメソッドに関するビューについては研究されてこなかった．前に述べたように，１つのメソッドに複数の実装を持つことができるという柔軟性はオブジェクトデータベースにとって重要である．なぜなら，永続オブジェクトの寿命は長く，永続オブジェクトに対する操作が時間とともに変わる可能性があるためである．結果として，メソッドに関して定義されたビューが必要になる．

分散メソッドとして，メソッドの新しい実装を書くことでメソッドのビューを実現できる．これは，メソッドビューの一つの定義方法であるが，決して容易ではない．Abiteboul と Bonner[1] が指摘するように，ビューの定義はできる限り容易でなければならない．例えば，“最新のメソッドの実装”, “サイズが最小のメソッド”, “A が開発したメソッド” などである．それらの例を，関係データベースのビュー定義のように表現する機能が提供されるならば，おそらくユーザは容易に例にあげたようなビューを定義できると考えられる．

この３つのメソッドビューの例は，ネットワーク中の全ての MethodDB を管理することによって容易に実現される．メソッドオブジェクトはメソッドがコンパイルされ MethodDB に保存された時刻のタイムスタンプを保持しているので，“メソッドの最新の実装” というメソッドビューの実装は容易である．最新のタイムスタンプを持つメソッドオブジェクトを検索することに，よってこのメソッドビューは作ることができる．第２のメソッドビューは，効率の良さを期待してメソッドの最小の実装を選択するのに使われる．はじめのメソッドビューの例と同様に，このメソッドビューの定義からメソッドオブジェクトを検索するのは容易である．第３のメソッドビューは，開発者 A が開発したコードは他の開発者が開発したコードよりも優れている (頑丈であったり，効率が良かったり) ということを知っているとき，役に立つ．多様なメソッドビューを定義したり扱うことを可能にするために，分散メソッドを MethodDB に格納する際，どんな情報が必要であるかを注意深く検討する必要がある．これは，今後の課題である．

メソッドビューによってメソッドコードをネットワーク中に保存し，共有することができる．データベースアプリケーションは永続オブジェクトを操作するメソッドから成り立っている．メソッドビューの定義によるメソッドをアプリケーションに含めることによって，ユーザはアプリケーションを構築するときにネットワーク中の他の開発者が開発したメソッドを利用することができる．言い換えると，ユーザ自身でメソッドのコードを書くことなる，データベースアプリケーションを構築することができる．さらに，よりよい効率を得るためにはメソッドの実装を改良する必要があるが，優秀な開発者によって開発される優れたメソッドの実装を選択するようなメソッドビューを定義できれば，我々はプログラムのコーディングをせずに効率の良いアプリケーションの開発が可能である．

メソッドビューの実現には，１つのメソッド定義から唯一つのメソッドが選択されることが保証されることが必要となる．これが従来のビューとメソッドビューの異なる点の一つである．これも，今後の課題である．

5 関連研究

プログラムコードを移動させ，実行することが可能なプログラミング言語が多く提案されてきた [17]．それら言語には，異機種分散環境をサポートするものや，セキュリティを強化しているものもある．[2]しかし分散メソッドで取り組んでいる，時間の経過に伴うオブジェクトの振る舞いに関しての要求の変化に答えるための機能は持ち合わせていない．また本稿で述べたように，分散データベース処理の効率化に関しての試みはあまり見られない．

クライアント・サーバアーキテクチャにおけるデータベース処理の効率化は重要である．Franklin 等 [6] が指摘しているように，関係データベースの多くはメソッドマイグレーションによる処理を，オブジェクトデータベースの多くはデータマイグレーションによる処理をサポートしてきたが，処理の効率化のためにそれらを組み合わせることはあまり考えられていない．

Franklin 等 [6] は，Selection 等の単項演算と Join 等の二項演算による処理で，分散環境における処理対象のデータの移動 (data shipping) と演算処理の移動 (query shipping) と，それらの組合わせ (hybrid shipping) による問い合わせの最適化手法を提案している．本研究との主な違いは，我々は単項演算や二項演算として一般的に論じることのできる簡単な処理を扱うのではなく，オブジェクトの持つメソッドを実装するプログラムコードを

[2]我々の研究では，これらは今後の課題である．

ネットワークを通して転送し，他サイトで処理の実行を行う環境での，データマイグレーションとメソッドマイグレーションの組合わせを考慮している点である．

6 まとめ

計算機環境の発達により，ネットワークでつながれた比較的高性能の計算機を使うことが一般的になってきた．その環境での永続オブジェクトの扱いを考えると，オブジェクトの属性データだけでなく，その扱いもまた時間とともに変更することもある．このため，我々は分散メソッドを設計・開発中である．

分散メソッドにより，他サイトの永続オブジェクトに対し，自サイトで開発したメソッドを処理することが可能になる．ネットワークに接続された複数のサイトに存在する永続オブジェクトを扱う際には，データマイグレーションとメソッドマイグレーションをうまく組み合わせることにより効率よく処理することが可能になる．本稿では，分散環境におけるこれら二つの処理の組合わせの効率に関する実験の報告を行った．また，分散メソッドを使った，従来考えてこられなかったメソッドに対するビューについて議論した．

謝辞

本研究の一部は，文部省科学研究費（奨励研究（A） 09780238，重点領域研究 08244101）による．

参考文献

[1] S. Abiteboul and A. Bonner, "Objects and views," Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.238–247, 1991.

[2] M. Aritsugi and A. Makinouchi, "Design and implementation of multiple type objects in a persistent programming language," Proc. COMPSAC, pp.70–76, Aug. 1995.

[3] J. Banerjee, W. Kim, H.-J. Kim, and H.F. Korth, "Semantics and implementation of schema evolution in object-oriented databases," Proc. ACM SIGMOD Conf., pp. 311–322, 1987.

[4] R.G.G. Cattell and J. Skeen, "Object operations benchmark," ACM Trans. Database Syst., vol.17, no.1, pp.1–31, 1992.

[5] M.J. Carey, D.J. DeWitt, M.J. Franklin, N.E. Hall, M.L. McAuliffe, J.F. Naughton, D.T. Schuh, M.H. Solomon, C.K. Tan, O.G. Tsatalos, S.J. White, and M.J. Zwilling, "Shoring up persistent applications," Proc. ACM SIGMOD Conf., Minneapolis, USA, pp.383–394, May 1994.

[6] M.J. Franklin, B.T. Jónsson, and D. Kossmann, "Performance tradeoffs for client-server query processing," Proc. ACM SIGMOD Conf., Montréal, Canada, pp.149–160, June 1996.

[7] J. Gosling and H. McGilton, "The Java language environments: A White Paper," Technical report, Sun Microsystems, 1995.

[8] G. Lamb, G. Landis, J. Orenstein, and D. Weinreb, "The ObjectStore database system," Commun. ACM, vol.34, no.10, pp.51–63, Oct. 1991.

[9] G. Landis, C. Lamb, T. Blackman, S. Haradhvala, M. Noyes, and D. Weinreb, "ObjectStore PSE: a Persistent Storage Engine for Java," The Second Int. Workshop on Persistence and Java (PJW2), Aug. 1997, http://www.sunlabs.com/research/forest/pjw2/.

[10] ONTOS Inc., "ONTOS Object Database Version 3.0 Developer's Guide," 1994.

[11] E.A. Rundensteiner, "*MultiView:* a methodology for supporting multiple views in object-oriented databases," Proc. VLDB, pp.187–198, 1992.

[12] Y.-G. Ra and E.A. Rundensteiner, "A transparent object-oriented schema change approach using view evolution," Proc. Int. Conf. on Data Engineering, pp.165–172, March 1995.

[13] J. Richardson and P. Schwarz, "Aspects: extending objects to support multiple, independent roles," Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.298–307, 1991.

[14] R.K. Raj, E. Tempero, H.M. Levy, A.P. Black, N.C. Hutchinson, and E. Jul, "Emerald: A general-purpose programming language," Software – Practice and Experience, 21(1), pp. 91–118, January 1991.

[15] Sun Microsystems, Inc., "SunOS5.3 Linker and Libraries Manual," 1993.

[16] B. Steensgaard and E. Jul, "Object and native code thread mobility among heterogeneous computers," Proc. ACM Symposium on Operating Systems Principles, pp.68–78, December 1995.

[17] T. Thorn, "Programming languages for mobile code," ACM Computing Surveys, vol.29, no.3, pp.213–239, Sept. 1997.

[18] 吉田 裕介, 中村 知久, 有次 正義, 金森 吉成, "分散環境でのデータ移動とメソッド移動," 電子情報通信学会データ工学専門委員会, 第9回データ工学ワークショップ, March 1998.