データベースシステム 93－7
システムソフトウェアと
オペレーティング・システム 59－7
（1993．5．28）

Object Sharing Server における イベント伝達機構の検討

黒沢 貴弘，深澤 寿彦，吉本 雅彦，柴山 茂樹
キヤノン(株)　情報システム研究所

我々は，現在，オブジェクト共有サーバ OSS の設計・実装を進めている．OSS では，分散環境下の複数のアプリケーション間で，イベント伝達や同期など振る舞いに関わる動的な情報も含めて共有することを目指している．本稿では，OSS の機能として，「Injection 機構」と呼ぶ，サーバの要求をクライアントに実行させるインタフェースについて述べる．そして，この Injection 機構という枠組を使って，change notification や，蒸発問題 [5] といった問題も扱えることを示す．OSS の実装は，Mach 2.5 の上で行なっている．Mach の外部ページャを用いて基本的なデータ共有機構を実現し，これをベースとして Injection Manager，Transaction Manager，Schema Manager で OSS を構成する．

A Study on Event Propagation Mechanisms for an Object Sharing Server

Takahiro KUROSAWA, Toshihiko FUKASAWA,
Masahiko YOSHIMOTO, Shigeki SHIBAYAMA
Information Systems Research Center,
Canon Inc.

An Object Sharing Server (OSS) is a server program which provides object sharing facilities in a distributed computing environment. In this paper, we present a notion of "Injection mechanism" as part of the OSS functionality. Using the injection mechanism the OSS is capable of "injecting" essentially dynamic behaviors to client processes, enforcing certain behaviors on such client processes. We also show that some seemingly separate problems are handled in the injection framework. We then outline an OSS implementation on the Mach 2.5 operating system. The OSS has a simple data sharing mechanism in its base using Mach's external pager. Other components, an Injection Manager, a Transaction Manager and a Schema Manager, are built on top of the data sharing mechanism.

1 はじめに

従来のデータベース・アプリケーションの多くは，会計処理や在庫管理に代表されるように，高性能な計算機上にデータを集中的に蓄積・管理しているデータベース管理システム対して処理を要求する形態をとってきた．このようなデータベースでは，単純な数値データがその多くを占めていたため，データベースシステムは，静的なデータの共有のみに貢献すればよく，複雑な値の操作などは，あくまでアプリケーション・プログラムが行なっていた．

一方，ワークステーションなど小型計算機が高性能化し，安価なコンピューティング・パワーを広い分野に適用できるようになった今日，より多様でill-defined なデータもデータベースとして管理したいという要求が高まってきた．また，データベース管理システムのアーキテクチャとしても，ハードウェア環境の変化を反映し，ネットワーク化されたコンピュータ・システムを想定したクライアント・サーバ・アーキテクチャが採り入れられてきている．

このような流れの中で，かつてのように「データベース・アプリケーションがデータベース管理システムに一方的に仕事を依頼する」のではなく，「クライアントとサーバが協調してデータベース処理を進める」作業形態が一般的になりつつある．しかも，そこで扱う対象データは，単純かつ静的なデータばかりではなく，オブジェクトとして能動的な性格を与えられている．このような要求に対応して開発されたのが，オブジェクト指向データベースである [2]．

オブジェクト指向データベースの特徴の一つとして，データベース内の永続オブジェクトは，クライアントのプログラム内では他の揮発性のオブジェクトと同様に操作が可能であることが挙げられる．この特徴は，柔軟なアプリケーションの構築を支援する一方で，いくつかの問題を生み出している．第一に，そのようなアプリケーションでは，従来の（揮発性のオブジェクトのみを扱う）クライアント・サーバのプログラムのように，正しく同期をとりながら並行に動作させる必要がある．第二に，データベースとしてのセマンティクスを考慮したクライアント・サーバのプロトコルを定め，それを一種のフレームワークとして構築する必要がある．不注意な永続オブジェクトの扱いは，データの一貫性を損なう可能性があるからである．

このような問題意識に基づき，我々は，クライアントとサーバとの間で，柔軟なアプリケーションの構築を可能にし，かつ，機能的に多様な要求を満足する基本機能について検討を行なっている．

上記のような考察から，現在，我々は，《サーバ・クライアント・インタラクション》を基本コンセプトとして，分散計算環境下でデータの共有機能を提供する OSS (Object Sharing Server) を設計・実装している．そして，以下に挙げる点から，分散計算環境下での協調型アプリケーションにアプローチしている．

- サーバとクライアントの交信．特に，サーバ側からクライアントに要求を行なえるインタフェースを重視している．
- データベース的な共有セマンティクス．つまり，データ構造のスキーマとしての管理，トランザクション管理などをサポートする．
- さまざまな共有モード．サーバが提供する共有モードは，単一ではなく，効率や分散性などを考慮して，アプリケーションによって使い分ける．

これにより，従来のデータベースシステムを介して静的なデータを共有するように，この OSS を用いることで，複数のアプリケーション間で「振る舞い」に関わる動的な情報を『共有』（イベント伝達や同期など）することを目指している．つまり，同期などを個々の機能として捉えるのではなく，『共有』という概念のもとで統一的に扱うものである．

本稿では，そのために最低限必要になる機能として，「サーバの要求をクライアントに実行させる」インタフェースを議論し，これを OSS において実現するための検討を行う．以下，このインタフェースを，《Injection 機構》と呼ぶことにする．

本稿の構成は，次の通りである．2 章では，Injection 機構 の基本コンセプトについて述べ，3 章で，その実現方法およびイベント伝達機能への適用について検討する．そして，関連研究について 4 章で述べた後，最後に，5 章でまとめ，今後の展開を示す．

2 Injection 機構

この Injection 機構 は，オブジェクト指向データベース管理システム（以下，OODBMS と略す）の

ようにサーバとクライアントとが密にデータを共有する環境を想定し，オブジェクトの振る舞いに付随する各イベントの共有（各クライアントへのイベント伝達をサポートする機構），さらに，クライアントにローカルなリソースを利用する問題 への適用を目的としている．

以下では， Injection 機構 の動機となった いくつかの問題を示す．

- change notification の扱いの欠如

  現在商用化されている Versant [11] , ObjectStore [7] のようなOODBMS では，change notification が十分にサポートされていない．たとえば，ウィンドウ・アプリケーションが，表示しているデータを OODBMS などにより共有している場合，共有されているデータの状態を，表示に常に反映するためには，共有されたデータの変更を，クライアントに通知するための仕組みが必要である．しかし，現状のOODBMS は，この点をほとんどサポートしておらず，クライアント同士が直接通信し合って，共有データの状態変更を通知している．

- 蒸発問題 [5]

  複数のクライアントに共有されたデータが，それぞれのクライアントにローカルなリソースを参照する場合に発生する問題があり，我々は，これを『オブジェクトの蒸発問題』と呼んでいる．この問題の本質は，サーバ内で動作する限りは，解決のための十分な情報を得られないところにある．

  たとえば，X ウィンドウ・システムのアプリケーションにおけるフォント，カラーなどのリソース情報は，ウィンドウ・サーバと通信しているクライアント内の揮発性（ volatile ）の情報であり，それらの情報を含んだデータを共有している OODBMS 単独では知り得ない．その結果，それらのデータを共有している他のクライアントでは，それらのリソース情報（クライアントにローカルな情報）が失われてしまうことがある．

- 所有者問題 [10]

  データベース・アプリケーション，特に，オブジェクトの更新やアクセス制御などに関して，何らかの規約に基づく動作をオブジェクトに

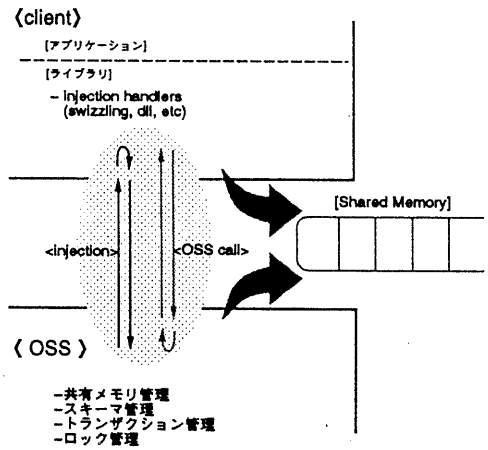

図 1: Injection 機構

『強制』したい場合が発生する．しかし，現在のデータベース・アプリケーションの動作環境では，このような動作（振る舞い）の強制をサポートしておらず，必然的に各プログラマの裁量で実現することになる．これをデータベースシステム側で実現するためには，オブジェクトの振る舞いに対して，網羅的かつ系統的な強制力を行使できる機構が求められる．

このような change notification 機能の実現や，『オブジェクトの蒸発問題』などの解決のために，『サーバの要求をクライアントに実行させる機能』として，Injection 機構 を導入する．そして，各問題に個別に対応するのではなく，Injection 機構 という枠組のもとで一貫した解決を図ってゆく．

2.1 Injection 機構 の構成

以上述べたような問題に対応するため，ここでは，サーバからクライアントへのデータなどの転送を "Injection"[1] と呼び，その機能を OSS に用意する（図1）．

この機能は，OSS とクライアントの間の共有メモリ[2]に単純なデータを書き込むばかりではなく，「実行可能なプログラム・コードを共有メモリに書き込

[1] 注入; サーバが圧力をかけて，その転送を強制することをイメージしている．

[2] 本稿では，密結合型計算機などの物理的な共有メモリばかりではなく，ソフトウェアによって実現される共有メモリ機能も含むものとする．

Injection の種類	クライアントに渡されるデータ (共有メモリを介するデータ)	クライアントで起動される操作 (メッセージ通知される操作)
0.	bulk data	–
1. instance injection	structured data	swizzling
2. method injection	program code (with symbols)	dynamic link and load
3. capsule injection	program code + structured data	dynamic link and load + swizzling
4. event injection	–	raising exception/signal
5. activity injection	program code + structured data	dynamic link and load + swizzling + invocation

表 1: 各 Injection 機構 の比較

みクライアントに (動的に) リンクしたり, または, それをクライアント側で起動したりする」など, クライアント側に振る舞いを『強制』するための機能を含んでいる.

これらの機能には, それぞれに特徴的な性質があるので, 対象となるデータによって, 以下のように分類している. それぞれの機能を 表1にまとめる.

1. instance injection
   構造を持つ静的なデータを, クライアントに提供し共有する. 一般的なデータベースのサービスである.

2. method injection
   共有メモリなどを介して, 実行可能なプログラム・コードをクライアントに送り, それをクライアント側で動的にリンク (dynamic linking and loading) する.

3. capsule injection
   上記, 1 + 2 の機能である. つまり, 構造を持つデータとそれに付随する操作をクライアントに送り, クライアント内で, swizzling , 動的なリンクなどすることにより, 意味のあるデータとなるようにセットアップする.

   たとえば, マルチメディアデータベースのブラウザに対して, 問い合わせにヒットしたデータとその表示操作を inject する場合が考えられる.

4. event injection
   クライアント側で設定したハンドラを, 間接的に起動する. そのため, クライアント内で, シグナルや例外をレイズする.

   たとえば, 協調動作している複数のクライアントの一つがダウンした場合, そのイベントを関係するクライアントに通知し, 各クライアントが適切な処理をするのを期待する場合である.

5. activity injection
   指定した手続きを, クライアント内で直接実行する. そして, この実行に際して, 必要であれば, method injection や capsule injection などと組合せ, 起動するプログラムコード自身も, inject する.

   たとえば, 各クライアントによって, セキュリティレベルを切替える場合, それぞれに異なったセキュリティチェック手続きを, それぞれのクライアントの環境で起動する.

この分類は, クライアント側から見ると, それぞれの Injection に対して, クライアント内に起動される手続きの違いとも言える. この視点からの分類は, 以下のようにまとめられる.

a. 固定された手続き:

   swizzling 手続き, 動的リンク手続きなどのように, 予め固定された手続きを起動する. これらの手続きは, ライブラリとして, クラアント内に用意される.

   これに該当するのは, instance injection, method injection, capsule injection である.

b. 登録された手続き:

   シグナル・ハンドラや update hook などのように, ここで起動される手続きは, クライアント (ユーザ) が, 特定のシグナル, イベントに結び付ける自由度を持つ.

   event injection が, これに該当する.

c. Injection に指定された手続き:

   その Injection によって指定された手続きを起動する. これには, OSS から inject され, ク

ライアントに動的にリンクされた手続きも含まれる．この手続きは，OSS 側が選択権を持ち，逆に，クライアントは，Injection を受けるまで決定することができない．

activity injection が，これに該当する．

2.2 Injection 機構 の特徴

従来，クライアント間の change notification などのアプリケーションに依存した振る舞いは，個々のアプリケーションでアドホックに実現されてきた．たとえば，アプリケーションが直接プロセス間通信を用いるなどである．

しかし，我々のアプローチでは，一連の操作を Injection 機構 として OSS に組み込むことで，これに対応している．すなわち，Injection 機構 を用いることで，敢えてクライアント同士での直接のプロセス間通信を避け，これらの通信を OSS の管理下に置いてしまう．そして，OSS を介することで形式化された通信を用いて，クライアント間の振舞いを記述することを目指している．

2.3 Injection による並行性とその制御

OSS では，Injection 機構 の導入により，一つのクライアント内に複数の実行環境が発生する可能性がある．この場合，クライアントの予想する以上の並行性が発生しうる．しかも，マルチスレッド環境下では，特別な措置を講じていない場合，同一クライアント内の他のスレッドのリソースを保護していないため，それらによる，dirty read/write が可能になってしまう．

この状況を解決するための方法として，我々は，以下の方法を検討した．

- event driven framework：

  Injection を受け取るループをアプリケーションの骨格に据え，イベント駆動型のアプリケーション構成とする．これにより実質的なスレッドを唯一とし，dirty read/write を回避する．

  メリット：実行の単位が一つになるので，並行制御が容易になる．また，クライアント内で，相互排除などのオーバヘッドも小さくなる．

- user committed concurrency：

  マルチスレッド環境下で，その内の一つを OSS からの Injection 受け取り用にリザーブする．こちらでは，クライアントの合意のもと，dirty read/write を許す．

  メリット：各スレッドの実行が，独立性の高いものになる．また，高い並行性が存在するため，並列計算機では，高い実行性能が期待できる．

上記の二つの方法を検討しているが，将来的には，後者が望ましいと考えている．並列処理環境が特別なものではなくなれば，一つのアプリケーション内の複数スレッド間での dirty read/write は，ユーザの管理下に置くのが適切，となると考えられるからである．

また，この Injection により発生する実行環境の並行制御と，トランザクションによる従来からの共有セマンティクスとの関係については，さらに議論する必要があると考えている．

2.4 Injection 機構 の適用例

この Injection 機構 は，OODBMS のようにサーバとクライアントとが密にデータを共有する場合，特に，次に挙げるように，クライアントに何らかの動作を強制する場合や，クライアント毎に動作を入れ換えたいアプリケーションに適用できる．

change notification:

たとえば，協調作業を行なっている複数のクライアントがあり，あるクライアントでのデータの変更が，そのデータを共有している他のクライアントに伝達され，そのデータについての表現を変える（典型的には，その変更をディスプレイに反映，あるいは，変更されたデータにハッチングする）という要求を考える．この場合は，共有されているデータの変更が，OSS に検知された時点で，クライアント側の表現を変更する操作を起動するように event injection あるいは activity injection を行なうことになる．

蒸発問題 [5]:

たとえば，X ウィンドウシステム上の描画データを OSS を用いて共有する場合，描画データのカラー情報などは，ウィンドウサーバと通信するクライアントにローカルな情報（揮発性の情報）であり，そのままでは蒸発してしまう．この情報を別のクライアントでも，正しく参照させるため，個々のクライアントにおいて描画データを activate する時に，揮発性の情報

を回復させるための activity injection を起動する.

所有者問題 [10]:

OSS が，クライアントに応じたセキュリティチェック・レベルの切替えを強制するために，異なったチェック手続きを，それぞれのクライアントに method injection で提供する．あるいは，クライアントの実行環境（たとえば，起動したユーザ）などに応じた有効期間判定などの振る舞いを method injection により強制する.

たとえば，見積書などを共有する OA システムがある場合，次のような関係にある見積書の有効期間とそのアクセス制御に，method injection が適用できる.

ユーザ	期限前	期限後
作成者	Read Only	Read/Write
顧客	Read Only	(none)

このように，これらの問題について，Injection 機構 を適用することで，統一的に解決できると考えている.

3 実現

ここでは，まず，OSS の構成と特徴について述べ，その後，その中での Injection 機構 の実現方法と，その機構を用いた OSS におけるイベントの伝達方法について述べる.

3.1 OSS の構成

現在，OSS は，単純なデータ共有機構の上に，トランザクションやロックなどの一貫性保持モジュール，および，それぞれのデータ型と操作を管理するスキーマ管理モジュールを加えて構成されている（図 2）.

1. データ共有

   OSS は，その名前の由来通り（単純な）データ共有機構をベースにしている．具体的には，Mach[1] の外部ページャ機能 (External Pager Facility) を拡張した構成により，タスク間の memory object (外部ページャの利用単位) の共有機能を提供している．同様のアプローチに，Cricket[8] がある.

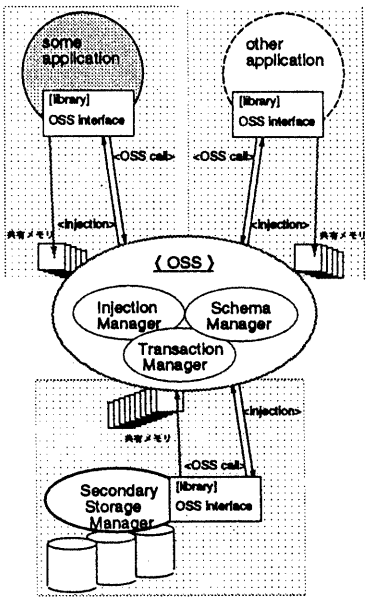

図 2: OSS の構成

2. 一貫性保持

   データ共有に関して，データベース的なセマンティクスを提供するため，デッドロック検知，ロールバックなどにより，トランザクション機能をサポートする.

3. スキーマ管理

   共有エリアにおかれるデータの構造とそれについての妥当な操作を管理する．その他，異なったアーキテクチャ間でデータを共有する際などに必要となるコンバージョンを管理する.

4. イベント検知

   必要となる Injection を起動するため，サーバ内の状態の変化を監視する．具体的には，トランザクションの開始終了やデータのアクセスなど，クライアントからの要求を監視する.

3.2 Injection 機構 の実現

Injection 機構 の動作は，おおまかに以下の操作を決められた手順で行うことによる（図 3）.

1. サーバによる共有エリアへの書き込み

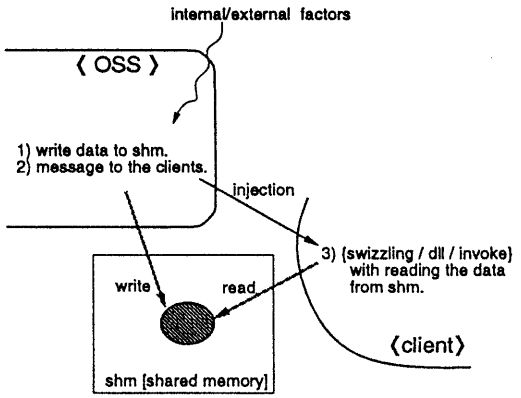

図 3: Injection 機構 の動作イメージ

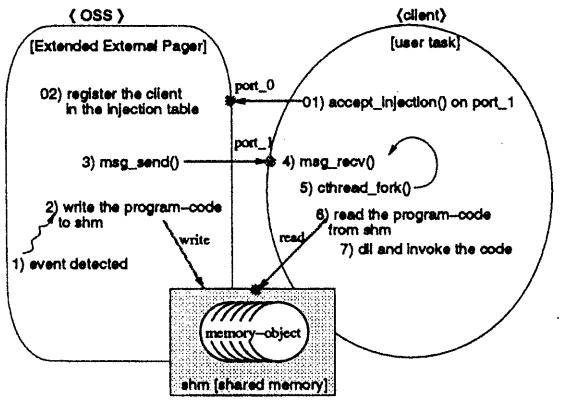

図 4: activity injection の実現

2. クライアントへのメッセージ通知

3. それに対応した操作のクライアントでの実行

ここでは，Mach[1] の機能を利用し，以下のように構成することで，Injection 機構 を実現する．

1. メッセージ通知機能

   Mach の port と message を用いて，OSS とクライアントの通信を行なう．また，クライアント側では，message として送られる Injection を受け付けるためのスレッドを用意し，必要に応じて，それに制御を渡す．

2. データ共有

   OSS が提供する共有エリア，つまり，Mach の外部ページャ機能により用意された共有エリアを利用する．そして，共有した memory object を介して，inject する側 (OSS) と，される側 (クライアント) の間でデータの共有を行なう．

3. クライアント・ライブラリ

   OSS からの Injection によって起動される swizzling, 動的なリンク, signal のレイズ などの固定的な手続きをクライアントライブラリとして提供する．

次に，この構成に基づいた Injection 機構 の具体的な動作を，activity injection を例として，以下に示す（図 4）．なお，ここでのクライアントは，マルチスレッド環境を仮定しており，そのうちの一つが，OSS からの Injection を受け付ける場合の動作となっている．

1. 該当するイベントを検知した段階で，

2. 外部ページャによって共有された memory object へ送り込むプログラムコードを書込み，

3. Mach の `msg_send` を使ってクライアントへ message 化された手続き起動情報を送信する．

4. クライアント側では，`msg_recv` により，OSS からの Injection を待っている．

5. 必要に応じて，スレッドを fork し，次の Injection に備える．

6. そして，OSS と共有している memory object からプログラムコードを読み込み，

7. activity injection に合わせて，読み出したプログラムコードを動的にリンクし，それを起動する．

なお，これに先だって，そのクライアントが Injection を受け取る旨の意思表明（図 4中 01）の `accept_injection` コール）をしているものとする．

これらの機能は，OSS によって定められたプロトコルによって呼び出される．図 5, 図 6に，その一部を示す．

3.3 OSS におけるイベント伝達

OSS では，イベント伝達も，Injection 機構 を用いたアプリケーションの一部と位置付けている．

ここでは，それぞれのクライアント内でも，同じイベントが発生したように振る舞わせるための，

```
- boolean connect(port_t oss, int connect_mode, port_t* shared_mo);
- boolean map_share(port_t shared_mo, int sharing_mode, port_t target);
- boolean register_service(port_t oss, mask_t* accept_mask, service_t service,
                          (int(*))func, int argc, argv_t* packed_argv);
- boolean accept_injection(port_t oss, port_t* wait_port, mask_t accept_mask);
```

図 5: クライアント側プロトコル

```
- void inject_object(port_t client, char* ptr, schema_t type);
- void inject_routin(port_t client, text_t* program, int size, symbol_t* symbols);  // dll
- void inject_signal(port_t client, int code, int subcode);                         // raise signal
- void inject_active(port_t client, symbol_t entry, int argc, argv_t* packed_argv); // invoke
```

図 6: サーバ側プロトコル

イベント伝達方式について a),b),c) のように検討する.

a) OSS によるイベントの監視.

トランザクションのコミットやアボート, あるいは, データのアクセスや更新などのイベントをサーバ内で監視し,「イベントの発生」を検知する.

b) Injection の対象となるクライアント.

基本的には, 予め accept_injection により, Injection の受け入れを指定しているクライアントに対して, Injection 機構 を起動し, イベントを伝達する[3].

c) Injection の対象となる手続き.

イベントの伝達では, それぞれのクライアントに, 同様のイベントを通知すればよいので, event injection を使って, クライアント内にシグナルをレイズすれば十分である[4].

一般に, イベントの伝達には, 即時性が求められることから, OSS 側から, クライアントを起動する Injection 機構 が有効であると考えられる.

3.4 OSS における共有モード

ところで, これまで述べてきたように Injection 機構 では, OSS によるデータ共有機能を, その前提としている. そこで本節では, OSS におけるデータ共有モードの分類 (図 7) を示し, その利用方法について述べる.

1. hard sharing
   複数のクライアント間で, まったく同じメモリイメージを共有する. つまり, サーバによる共有イメージの解釈を行なわない. このため, 共有のためのオーバヘッドが少ない. その反面, ポインタの swizzling などを要する relocatable なデータの共有は難しい.

2. structural sharing
   データの構造を認識した共有を行なう. このため, 必要に応じて, swizzling やクライアントの実行環境に合わせたコンバートを行なうなどして, relocatable なデータを, 異機種分散環境で共有する.

3. semantical sharing
   構造上異なってしまうかも知れない複数のデータを, あるモデルの下で同一のオブジェクトであるように見せ, それらを共有する. (ある意味では) オブジェクトにマルチビュー機能を提供することに相当する.

この共有モードと Injection 機構 を併用することにより, アプリケーションに依存する, より柔軟なデータ共有ができると考えている.

3.5 プロトタイプ

現在, 機能限定した OSS のプロトタイプに, Injection 機構 を作成中である. 実験環境は, 我々の研究所で作成した Stonehigh WS[3] 上の Mach 2.5

[3] この他にも, ある共有データを使っているクライアントについては, 暗黙の accept_injection を発行し, Injection 機構 を起動することも考えられる.

[4] さらに, アクティブ・データベース [4] などにおける ECA ルールの Action に相当するような機能を実現する場合には, activity injection を利用し, クライアントに指定された手続きを起動することも可能だと考えている.

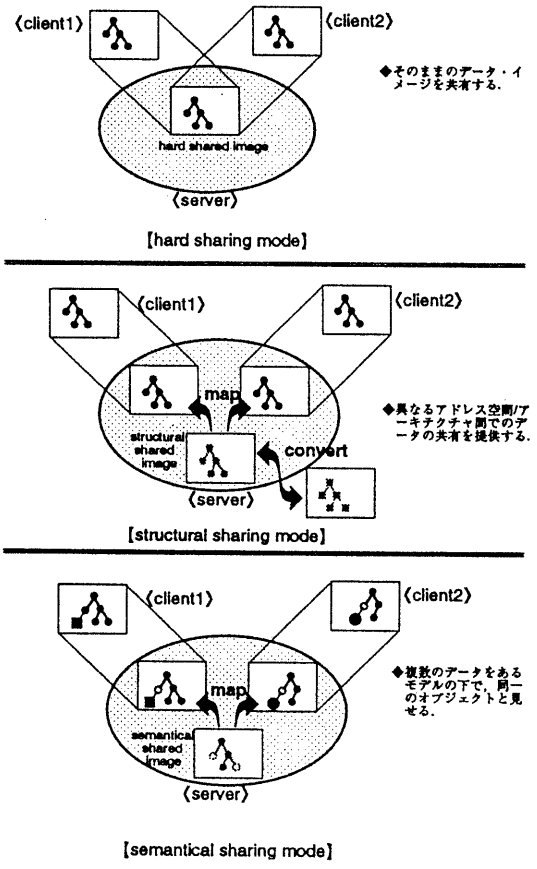

図 7: OSS の共有モード

をベースとした OS[9] を利用している. そして, データ共有のためには, 独自に作成された外部ページャを利用し, ロック, トランザクション, スキーマ管理, それに Injection 機構 用の MIG インタフェースを加えるように改造を施している.

4 関連研究

ここでは, OSS の Injection 機構 のいくつかの側面について, 関連する分野と比較する.

- IPC との比較 (メカニズム面からの比較):

  オペレーティング・システムにより提供される RPC など従来の IPC(Inter Process Communication) と比較し, よりデータベース寄りのアプローチである. つまり, データ共有を前提としたアプリケーションを対象としており, それらの共有メモリを介して, アプリケーションにサーバ (OSS) からの要求を, 強制的に実行させるための機能を含んでいる.

- peer-to-peer のように対等なアプリケーションが通信しあうモデルとの比較 (実現モデル面からの比較):

  ここでは, Injection 機構 の前提として, 敢えてサーバ・クライアント・モデルを用いているが, これは, それぞれのアプリケーションが, 必要以上に肥大化することを抑えるためである. 共有できる機能をサーバに置き, 必要に応じて Injection 機構 を用いて, クライアントに機能追加するモデルを採用している.

- トリガ機構との比較 (利用面からの比較):

  トリガ, アラータなど従来の DBMS で検討されてきたアプローチは, (表現力が小さい) DBMS 側での動作を前提とした記述[5]を要求していた. それに対し, ここで示す Injection 機構 は, ユーザ側のプログラミング環境での動作を前提として記述できる点が, 大きなメリットである.

  また, 従来のトリガ機構を備えたデータベースシステムでも, サーバからクライアントに影響力を持つ例が少ない. それらのデータベース・システムの研究事例では, method をデータベース上にストアして, それをサーバ (デー

[5]その記述には, SQL のような専用言語か, Lisp, Prolog のようなサーバとの一体型言語を強要されることが多い.

タベース・エンジン）内で起動するケースがほとんどである．

5 まとめ

本稿では，「サーバの要求をクライアントに実行させること」を目的としたInjection機構と呼ぶインタフェースについて議論すると共に，現在我々が取り組んでいるObject Sharing ServerにおけるInjection機構の実現について述べた．

Injection機構については，実験のための実装中であり，その動作モデルについても，さらなる検討が必要だと考えている．

また，以下の点についても，OSSの機能拡張あるいは変更を考慮して，さらに検討を進めて行く．

- ここに示した機能を有効に利用するためのプログラミングモデル（application framework）を検討する．たとえば，Injection機構を駆動するためのルール・システムなどが有望である．
- クライアント内で手続きを起動することは，セキュリティ上の問題を引き起こす可能性がある．現状では，この安全性を保証できない[6]．
- OSS上に置かれたオブジェクトに関して，Injection機構を利用するなら，その相手をqueryによって選択的に指定することも考えられる．

参考文献

[1] M. Accetta et al., "Mach: A New Kernel Foundation For UNIX Development," *Proceedings of the Summer USENIX Conference, pp.93-112*, (June 1986)

[2] R. G. G. Cattell, "Object Data Management - Object-Oriented and Extended Relational Database Systems," *Addison-Wesley.*, (1991)

[3] 伊達 厚, 濱口 一正, 出井 克人, 柴山 茂樹, "マルチプロセッサワークステーション"Stonehigh" -コンセプトとハードウェア概要-," 情報処理学会第*45*回全国大会, *6L-02*, (Oct. 1992)

[4] N. H. Gehani, H. V. Jagadish, O. Shmueli, "Event Specification in an Active Object Oriented Database," *SIGMOD pp.81-90*, (1992)

[5] 黒沢 貴弘, 上原 隆平, 吉本 雅彦, 柴山 茂樹, "データベース・アプリケーションにおけるオブジェクトの蒸発問題の検討," *WOOC'92*, (1992)

[6] C. Lamb, G. Landis, J. Orenstein and D. Weinreb, "The ObjectStore Database System," *CACM vol. 34, No. 10, pp. 50-63*, (Oct. 1991)

[7] Object Design Inc., "ObjectStore Reference Manual Release 2.0 for UNIX," (Oct. 1992)

[8] E. Shekita, M. Zwilling, "Cricket: A Mapped, Persistent Object Store," *The Fourth International Workshop on Persistent Object Systems pp.89-102*, (1990)

[9] 鈴木 茂夫, 宮本 剛, 伊達 厚, 岩本 信一, 柴山 茂樹, "マルチプロセッサワークステーション"Stonehigh" -機能分散型OSの設計と実現-," 情報処理学会第*45*回全国大会, *6L-03*, (Oct. 1992)

[10] 上原 隆平, 吉本 雅彦, 黒沢 貴弘, 柴山 茂樹, "オブジェクトの『所有者』問題とC++による実装例," 情報処理学会第*46*回全国大会, *3F-3*, *pp.4-109*, (Mar. 1993)

[11] Versant Corp., "Versant System Reference Manual (release 1.7 for Sun OS)," (January 1992)

[6] しかし，実際上は，Injectionは，サーバ側の要求に従って行なわれるので，それ以前にチェック可能であると思われる．