HyperStation: 分散オブジェクト指向言語の構想

2Q−1

佐治 信之　景山 辰郎
NEC マイコンソフトウェア開発研究所

1 はじめに

我々は, 次世代マルチメディア操作環境として, "*HyperStation*"[1] の研究開発を進めており, マルチメディア情報を含む様々な形態の情報を分散オブジェクトとして表現し, これを作成 / 操作 / 共有する枠組を実現することが重要であると考えている. そして分散オブジェクトが備えるべき要件を以下のように考えている.

- 情報を視覚的に直接操作できるように, 全ての分散オブジェクトが見た目 (ビュー) を持ち, 指示 (イベント) を発行することができること.
- 分散オブジェクトを容易に作成 / 編集できること.
- 複数ユーザが協同作業を行なえるように, オブジェクトを共有する機能を備えること.
- ネットワーク透過なオブジェクト空間中で分散オブジェクトに対して命名 / 参照ができること.
- 一度生成したオブジェクトは, プロセスといった枠組を越えて存在し続けること.

我々は, このような分散オブジェクトを操作する環境として, *HyperShell*[2][3] の研究開発を進めており, この環境を実現するための, 柔軟で表現能力に富んだ言語システム (仮称 "*HyperScheme*") を検討している. この言語システムは, 仕様面では, 仕様の単純性と表現能力の高さ, 拡張性 / 柔軟性 / 安全性, オブジェクト指向システム, 工学面では, 移行及び教育の容易性 (標準), 他言語との結合容易性, デリバリの容易性, を備えることが重要であると考えている.

現在, これら全ての性質を持ち得る言語システムは存在しないが, Lisp 系の言語を基本にこれを拡張することで上記の要求を充たすことを目指している. Lisp 系の言語としては, 小さい言語仕様で高い表現能力を持ち, 世界標準言語の地位を確立している *Scheme*[4] を採用し, さらに, オブジェクトシステムとして, *Common LISP* のオブジェクトシステムである *CLOS*[5] を実装する.

ただし, このような言語で *HyperShell* 全ての環境を構築することは現実的ではなく, 特に, これまでのソフトウェア資産や今後の標準の方向を考慮し, システム記述言語の実質的標準として期待されている C++ をこの言語システムに取り込む. そして, 実行効率を重視した C++ のような静的な言語の特徴を生かしつつ, インクリメンタルなモジュールの追加や置換えを可能にする機能や, C++ をインタプリティブに実行する機能を取り入れることで, 拡張性 / 柔軟性 / 安全性を実現する. C++ の制御は, *CLOS* のメタオブジェクトプロトコル (Meta Object Protocol: MOP) を使用して C++ メタオブジェクトを定義し実現する.

以降本稿では, *HyperScheme* のコア部分の構造, 共有オブジェクトのガードの方法, GUI 制御スクリプト機能に関して述べる.

2 *HyperScheme* コアの構造

HyperScheme コアは, 次の 4 つの基本機能により構成されている.

1. インクリメンタルリンカ (必須)
   Scheme や C++ のプログラムのリンク機構.
2. メモリ管理プリミティブ (必須)
   Scheme や C++ のプログラムとデータの領域管理機構.
3. プロセス間通信プリミティブ (省略可)
   複数の *HyperScheme* コア間の通信を司る機能.[1]
   HyperScheme コアは UNIX[2] のプロセスに対応する.
4. インタプリタとオブジェクトシステム (省略可)
   オブジェクト指向機能を拡張した *Scheme*.

HyperScheme コアは, ある機能をリンクしてその機能を全うする "*HyperScheme* プロセス" として動作する. *HyperScheme* プロセスはプロセス間通信プリミティブを使用して, 分散環境下でプロセスの集合体として協調して動作する.

3 共有オブジェクトのガード

HyperScheme オブジェクトシステムでは, 分散オブジェクトを複数のクライアントが共有する場合に対応する必要があるが, *CLOS* のメソッド結合の方式を拡張して, 分散共有オブジェクトのメッセージ順序制御を実現する.

CLOS では, 標準メソッド結合として, 同名のメソッドに対してメソッド修飾子 (*method-qualifier*) を指定することにより, **around**, **before**, **after**, **primary** の 4 種類のメソッドを定義することができる [5]. これに加えて, *HyperScheme* では, ガード条件を記述する **guard** メソッド (メソッド修飾子:**guard** を指定) を定義することを許す.

以下に *HyperScheme* におけるメッセージ順序制御の仕方を示す.

- オブジェクトの実行は単一スレッドである.
- メソッドキューが 1 本存在する.
- メソッドキューに入れられた実行待ちメソッドが順次取り出され, 標準メソッド結合の実行に先行して,

"*HyperStation*: Concept of Distributed Object-Oriented Dynamic Language *HyperScheme*", Nobuyuki Saji, Tatsuro Kageyama, Microcomputer Software Development Laboratories, NEC Corporation

[1] IPC のプラットフォームとしては OMG の Object Request Broker を想定している.

[2] UNIX オペレーティングシステムは, UNIX System Laboratories, Inc. が開発し, ライセンスしています.

guard メソッドの評価が行われる.
- guard メソッドの評価結果が真になった場合には, 標準メソッド結合を実行し, 真にならない限り, そのメソッドはキューに保留される.
- guard メソッドは, より特定化されていない guard メソッドを, `call-next-method` で明示的に呼び出すことができる.
- guard メソッドは, 真にならない間は, 複数回実行され得るので, 基本的に副作用があってはならない.

4 GUI 制御スクリプト

HyperScheme は, C++ の代表的な GUI である InterViews インタフェースを備え,
- InterViews の C++ 関数の会話的実行
- InterViews のオブジェクトの内視
- オブジェクト指向 GUI スクリプト言語

を実現し, *HyperShell* のスクリプト言語として機能する.

HyperShell では HyperCard 風のユーザインタフェースを実現しており, *HyperShell* のバインダ, シート, ボタンなどのオブジェクトへのイベントメッセージの配信を柔軟に制御するために,

(A) 最初にメッセージが送られたオブジェクトを起点とした, 独立オブジェクト間 (シートとシートなど) のメッセージデリゲーションパス.
(B) 現在メッセージを処理しているひとかたまりのオブジェクトそれ自身.
(C) ひとかたまりのオブジェクトの構成要素オブジェクト.

を個別に扱うことができなければならない.

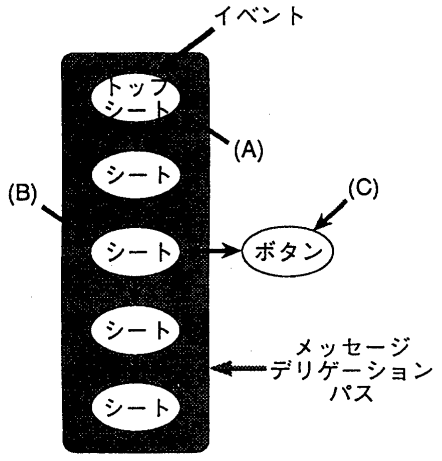

図 1: *HyperShell* のメッセージ配信

(B) はメソッドを実行中のオブジェクトそれ自身としてアクセス可能であり, (C) のメッセージ処理は, メソッド中で明示的に記述するか, *CLOS* のメソッド結合の機能を利用することができる. メソッド結合とは, オブジェクト内の要素部品に対してメソッドを順次適用する機能であるが, 適用するメソッドを実行可能メソッドリスト (*effective method list*: *eml*) として, 各メソッドに隠れ引数を介して渡している.

(A) に該当するオブジェクトとメッセージデリゲーションパスのオブジェクトリスト (*effective object list*: *eol*) を保持する機構は, *HyperScheme* にはない. そこで以下のように対処した.

実行可能メソッドリスト *eml* を渡す隠し引数を利用し, *eol* とペアにして *context* 引数として一般化する. ただし,

eml: クラス継承で関係付けられたメソッドの集合で, 実引数をもとに総称関数内で計算される.
eol: *HyperShell* 空間内の存在位置で関係付けられたオブジェクトの集合で, 総称関数を実行する時点で既に計算されている必要がある.

という違いがあり, *eol* は総称関数に引数として渡される. そこで, 総称関数にキーワード引数:`eol` を暗黙に付加し, ディスパッチャは, この引数を特殊扱いして取り出し, *eml* とペアにして *context* としてメソッドに渡す.

メソッド内でのデリゲーション処理のために, *eml* を扱う `call-next-method` と同様に, *eol* を扱う `delegate-to-next-object` 関数を追加する. この関数は, 現在実行中のメソッドオブジェクトから辿って総称関数オブジェクトを取り出し, これをデリゲート先オブジェクトに再適用する.

HyperShell のイベントセンサは, イベントをメッセージ *mes* に変換するとともに, メッセージデリゲーションパス *eol* を計算し,

(*mes object position* ··· `:eol` *eol*)

を実行すればよい.

5 おわりに

現在, *HyperScheme* 処理系の試作及び *CLOS* の移植作業を行っている. 今後は, マルチスレッド, 例外処理機構, 分散共有オブジェクト, 永続オブジェクトの実装, C++ メタオブジェクトの実装とその有効性の実証, GUI インタフェースの実装, 評価を行ない, 最終的に C++ 操作環境としてまとめてゆく予定である.

参考文献

[1] 濱川 礼 他: 分散オブジェクト指向マルチメディアシステム *HyperStation* – その構想と試作 –, 第 45 回情処全国大会 1B-01, 1992.

[2] 新 淳 他: *HyperStation*: 分散オブジェクト指向シェル *HyperShell*, 第 45 回情処全国大会 6Q-06, 1992.

[3] 岩崎 未知 他: *HyperStation*: 分散オブジェクト指向シェル *HyperShell* のオブジェクト共有方式, 第 45 回情処全国大会 6Q-05, 1992.

[4] Clinger, W., Rees J. (eds.): *Revised*^{4} *Report on the Algorithmic Language Scheme*, 1991.

[5] Steele, G.: *Common Lisp: The Language*, Second Edition, Digital Press, 1990. (bit 別冊 *Common LISP* 第 2 版, 共立出版, 1991)