成蹊大学理工学研究報告
J. Fac. Sci. Tech., Seikei Univ.
Vol.48 No.1 (2011) pp.9-13

JavaVM 上での非手続オブジェクト転送を可能とする直列化方式の構築

赤井　雄樹[*1]，山口　大祐[*2]，甲斐　宗徳[*3]

An Implementation of Serializing Method for Non-procedural Object Transfer between JavaVMs

Yuuki AKAI[*1], Daisuke YAMAGUCHI[*2], Munenori KAI[*3]

ABSTRACT： In a heterogeneous processing environment in a computer network where multiple PCs running JavaVM on them are connected to, it is very useful to make any class instance movable among any PCs. However, JavaVM can't inherently load and execute unknown classes dynamically which are not in the class path specified at startup of the JavaVM. In this research, we propose and develop a new mechanism to recognize and execute any unknown class dynamically by implementing a hierarchical class loader, special object stream and dispatch mechanism for transferred objects. This proposed technology is useful and indispensable component technology for parallel distributed processing system based on mobile agent system with strong migration mobility. This technology makes it possible to implement various non-stop software services.

Keywords：Java serializing method, Non-procedural object transfer, strong process migration, parallel distributed processing

（Received April 5, 2011）

１．はじめに

１．１．研究背景

現在でも，コンピュータの利用により解決したいが計算量が膨大なため解決が困難な問題はまだ数多く残っている。そのような問題を解くために一つのコンピュータのハードウェアを拡張して高速化していくのは物理的限界の存在やコスト面が大きなハードルとなる。そこでそれらの問題をソフトウェア面から解決する一つの策として並列分散処理がある。これは，ネットワークに接続されている複数のマシンの余剰となっている計算資源を有効に活用し，一台のコンピュータ以上の高速処理を達成するものである。

ネットワークを介した分散処理において，核となる部分の一つにどのようなデータを相互にやり取りするかという事が挙げられる。

ORB，CORBA，Struts 等，一般的に知られるフレームワークがあるが，その基本的な利用はサーバクライアント型である。本研究では，巡航型の実行コードを可能とする為のシステム設計とそれに付随する技術に関する調査・検証・実装方式の提案を行う。

計算資源として多くの PC を用いたいという事と，要求される PC の環境が均一でなくても一つのネットワークとして構成できる事，また計算処理をそのネットワーク内の最適な PC を選択して実行可能である形が好ましい。具体的な例を挙げれば，UNIX 環境のクラスタと Windows 環境のクラスタ上で同じ計算主体が相互に転送可能である形である。このようなヘテロ環境での相互接続を可能にする手段として，JavaVM とその VM 上でのネットワーク処理を用いる事で OS やマシン構成を考慮することなく一般的な PC としてネットワーク構成されていれば計算環境として使う事が可能となる。

JavaVM 上での通信処理であれば，ObjectStream を用いることで透過的にインスタンスを扱うことができる。ローカルの PC とリモートの PC で継続してインスタンス操作が可能なのである。このインスタンスとはクラスから生成されたオブジェクトであり，計算処理中のメンバ変数を保持している。

しかしながら，インスタンス情報はクラス固有であり，そのクラスを構成する情報がローカル PC とリモート PC で異なる場合には，インスタンスを透過的に扱うことが出来ない。例えばローカル PC 上にのみに存在するクラスの情報をリモート PC が知りえない為である。

[*1]：理工学研究科理工学専攻修士学生
[*2]：理工学研究科理工学専攻修士学生
[*3]：理工学研究科理工学専攻教授（kai@st.seikei.ac.jp）

RMI 等の多くの実装ではクラス解決の為に，URI で指定されたサーバなどに動的に問い合わせる事で解決するが，実行開始する PC への物理的なネットワーク通信経路のトポロジによっては明確な遅延になる上，多くのリモート PC から実行開始する PC への不定的かつ大量のリクエストが飛ぶことが想定される。

その為，インスタンスを転送後に復元させるため，必要なクラス情報データ転送の方法を考える必要がある。

本研究で想定しているのは規模が不定で随時増加減少する PC 群からなるネットワークである。その環境下で計算主体としてのクラスインスタンスをどのようにすれば復元できるのかという事に焦点をあて，その実現に必要な機能と JavaVM 環境下での制約や，実現可能な実装を示す。

過去数年にわたり，著者は分散環境における自律型アプリケーションフレームワークの AgentSphere[1)]の開発に携わってきたが，これは AgentSphere が目指すマイグレーション機能を実現するための根底的な技術開発でもある。

１．２．並行研究：モバイルエージェントシステム AgentSphere の概要

ネットワークで繋がれたコンピュータ上で AgentSphere を起動することにより，そのマシンは AgentSphere ネットワークに参加したことになる。AgentSphere は他のマシンから移動してくるモバイルエージェントに活動の場を提供する。また，自身のマシンからのエージェントの起動を行うことができる。エージェントは各マシンの負荷状況を考慮して移動するなど自律的に振舞う。これにより自律型並列分散処理を実現する。

１．３．昨年度までの研究の流れ

2003 年度まで，本研究では既存のモバイルエージェントシステムである AgentSpace を利用し，その拡張を行ってきた。しかし，AgentSpace はエージェントのモビリティに弱マイグレーションを採用しており，実行状態を完全な形のまま移動することができない。そこで，モビリティとして強マイグレーションを採用する独自のシステムである AgentSphere の開発へと移行した。

強マイグレーションを実現する JavaVM を用いたシステムとしては，JavaGo などの先例がある。それらは強マイグレーションを実現しているものの，JavaVM の改変が行われているため JavaVM のバージョンアップが行われるたびに修正を行う必要がある。そこで，本研究では，JavaVM の編集なしに強マイグレーションを実現するための理論研究が行われてきた。それによりコードの変換手法が確立され，コード自動変換器が開発された。そして，2008 年度からはそれを利用して実際にシステムとして運用可能にするための研究が始まり，2009 年度ではユーザインタフェース及びエージェント活動管理機構の拡充を行った。

１．４．本年度研究目的

Java の標準的な実装にはリモートのクラス機能を呼び出す実装は存在するが，受動的に転送クラスを認識し，動的に起動中の VM にプラグインする機構は存在しない。加えて，リモートへのオブジェクトインスタンス転送をおこなう場合に，未知クラスを扱う標準的な設計は存在しない。

本年度は昨年度までの研究段階で限定的な実装であった部分の改修および，拡張を行う。具体的にはクラスの動的更新と差分更新を可能とするクラスローダの設計，遠隔オブジェクトインスタンスの復帰(デシリアライズ)を可能とするオブジェクトストリームの設計，未知クラスの転送を考慮した通信方式の構築を行う。その上で，これらの機能を用いる事で可能となる巡航型クラスの原子転送とオブジェクトデシリアライズ及びメソッド呼び出しの有効な実装を検証する。

２.巡航型クラスの原子転送とオブジェクト直列化

２．１．概要

分散オブジェクトでかつ，そのオブジェクトが巡航型の機能を備えるために必要な機能の一つに未知クラスの受け入れがある。これはローカルとリモートという 2 点間において，ローカルで読みだされ実行したクラスとそのオブジェクトインスタンスをリモートへ転送する場合に，リモートがクラスを動的に認識し，そのオブジェクトインスタンスを復帰(デシリアライズ)させる事を指す。ローカルでの認識クラスをリモートに認識させる事とオブジェクトシリアライズ・デシリアライズの仕組みは一貫しているようで別の理屈で作られており，(Java の標準仕様といえる)この二つの仕組みを連携させる必要がある。

２．２．オブジェクトシリアライズ・デシリアライズ

Java においてシリアライズインターフェースを備えるクラスはシリアライズ可能である。これはインスタンスをバイナリ化する事であり，このバイナリを転送すれ

ば，転送先でデシリアライズする事が可能である。しかしながら，リモート環境下へオブジェクトをシリアライズするのであれば,その構成クラスの情報が必要であり，標準のシリアライズ機能をそのまま利用することは難しい。

インスタンスの送信元であれば，クラスからクラスローダを参照できることから送信用のデータを特定出来る為，拡張は容易であるが，送信先のリモート環境下でオブジェクトを復帰するには予めクラス解決をする必要があり，その実現にはクラスローダ生成・選択の必要がある。

２．３．オブジェクトシリアライズ・デシリアライズ

クラスローダの一般的な設計では検索委譲先として親クラスローダへのリンクを持つとされる(後述)。つまり，ユーザクラスローダを設計した場合，祖先にシステムクラスローダを持つ必要がある。この仕組みは Java の標準 API と実行時カレントディレクトリの Java プログラムのクラスを認識しているクラスローダがシステムクラスローダである為，標準 API のクラス等を扱うクラスをユーザクラスローダで読み込んだ場合に必要であり，すべてのクラスは Object 型継承である以上，必ずこの設計となる。

しかし，最終的にシステムクラスローダへのリンクが可能であれば親子関係のリンク設計に関しては自由度が高く，クラスの動的追加，更新の頻度に応じてマルチレベル化設計が可能である。親側で定義されたデータ構造クラスを用いる事や親側で定義されたクラスを継承するといった使い方を想定した設計が可能である。このクラスローダを 2.2 節でのオブジェクトストリームに組み込む形となる。

また，クラスのロードの際には，そのクラスが用いているクラスの再帰的なロードを行うのだが，実行時の呼び出し順序と，シリアライズ時の呼び出し順序，デシリアライズ時の呼び出し順序は少しずつ異なり，この順序が問題とならないように設計する必要がある。

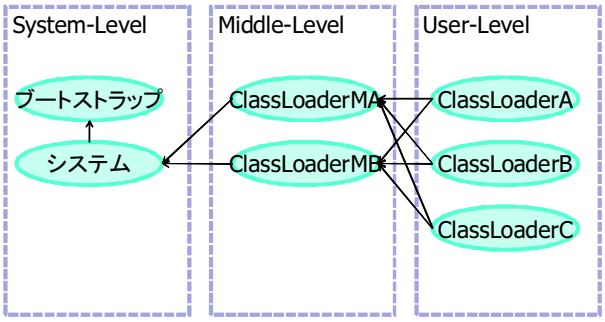

Fig. 02.3 マルチレベルクラスローダ設計例

２．４．オブジェクトシリアライズ・デシリアライズ

多階層化と細分化の為にクラスローダを分割するにあたって，考慮すべき点がある。それはクラスローダ間の委譲関係であり，これはクラスローダの親子関係のように表現される。クラス間継承関係の親子とは異なり，「あるクラスローダがクラス解決を図る前に検索要求を一旦委譲する先」を親と呼ぶ。

クラスローダを設計するに当たり，クラスローダクラスを継承して作成するが，予め用意されているコンストラクタには，その委譲先としての親クラスローダを引数として渡すようになっている。つまり，クラスローダを Java の慣習に則り作成するのであれば，親クラスローダを一つ想定して作るべきである。クラスローダはこの親子関係を用いて再帰的にクラス検索とクラス解決をしており，この標準で用意されているルーチンを使って多階層化を実現する。

クラスローダの検索委譲の例を挙げる。

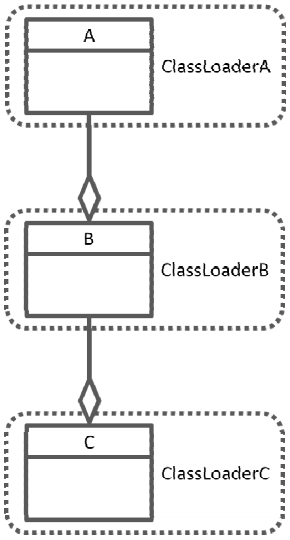

Fig. 02.4-1 クラス集約例

Fig.02.4-1 の例では三つのクラス関係を示す。Class A があり，Class B は Class A をメンバとして持ち，さらに Class C が Class B をメンバとして持つ場合に，Class A は Class B と同じクラスローダか，Class B より親のクラスローダに読まれている必要があり，さらに Class B は Class C と同じクラスローダか Class C より親のクラスローダに読まれている必要がある。

子クラスローダは一つの親クラスローダの参照を持ち，クラス解決の為の検索時に，検索を委譲する先として用いる。Fig.02.4-1 の例では Class C が出現した際に，Class C にはメンバとして Class B が含まれる為，これを解決しなければならない。その際に ClassLoaderC は，

親の ClassLoaderB に検索委譲をし，Class B を解決する。同様に Class A は ClassLoaderA で解決される。この例の場合には集約に対して関連するクラスローダの委譲先が一元的であるため，問題はなく，クラス間の継承関係の場合でも単一継承しか許容されない為，ClassLoader 間の委譲関係は単純に済む。

しかし，集約関係の場合には複雑なクラスローダ親子関係を必要とする場合がある。以下に例を示す。

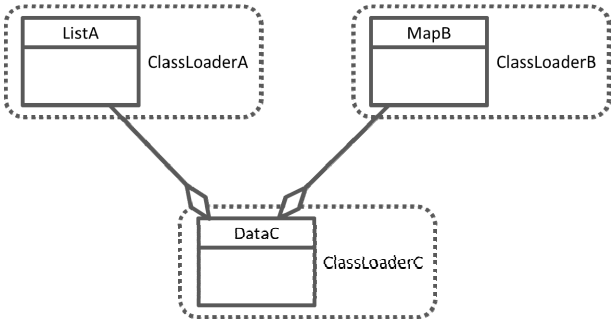

Fig.02.4-2 複数からの集約例

Class ListA と Class MapB というクラスが存在し，それぞれが ClassLoaderA と ClassLoaderB によってロードされているとする。これらは互いに親子関係はないとする。この時 Class DataC をそれらの子にあたる ClassLoaderC で読み込むとする。Class DataC はメンバに Class ListA と Class MapB を持つ。この場合，検索委譲先が一個であると，該当クラスを見つけることができない。

Class DataC は意味的に複数の親を持つ必要がある。クラス間継承であれば Java は単一継承である為，親は唯一で事足りるかもしれないが，メンバに関してはそうであるとは言えない。その為，ClassLoaderC は ClassLoaderA と ClassLoaderB のどちらも検索できなければいけない。それを可能とする為には，クラスローダを複数内部に持つクラスローダを設計する必要がある。

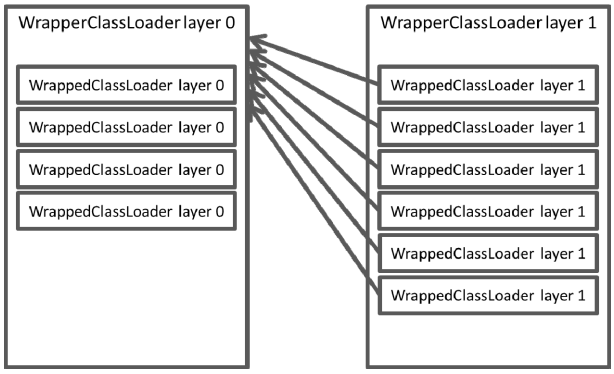

Fig.02.4-3 クラスローダをラップするクラスローダ

Fig.02.4-3 では，クラスローダをラップするクラスローダの設計を示している。2.3 節で示したマルチレベル化の細かな実装例となる。クラスローダは検索委譲先の親を一つだけ持つことが出来るので，クラスローダをラップするクラスローダを委譲先としている。ラップするクラスローダは，ラップされるクラスローダが該当クラスを解決できるかどうかを順に精査していき，可能なクラスローダが見つかった場合はラップされているクラスローダで解決し，見つからなかった場合は更に委譲先へと検索要求をする。

ラップされるクラスローダには，クラスデータのバイナリを取得できる機能を搭載し，この情報を用いてクラスローダのリモートでの再生成を行う。この事は次節で説明する。

２．５．未知クラスの転送を考慮した転送方式

オブジェクトのデシリアライズの為にはクラスのバイナリデータが必要である。この事は 2.2 節，2.3 節及び 2.4 節で説明した。しかし，クラスバイナリデータは単体であるとは限らず，独自実装した構造体のようなクラスを用いている場合がある。その場合，クラスが用いているクラスを含めて転送しなければいけない。転送すべきクラス一覧を生成する方法として，該当クラスローダ自身が解決可能なクラス一覧のバイナリデータをすべて送るという形である。

この設計の場合，厳密にはある特定のクラスにとって不要なクラスのデータさえ一緒に送る事がある。この設計にする理由の一つに，「Java のクラス解決のタイミングは実行時に出現したタイミングである」という事が挙げられる。

対案の設計として，クラスが出現した情報をすべてロギングし，そのクラスデータを一つにまとめて送るという設計が考えられるが，次の想定から却下した。

例として挙げられるのは，頻度の低い例外処理中でのみ使われるクラスが存在した場合，ローカルで例外が発生しない状態のまま転送し，リモートで例外が発生した時にクラス解決が不可能であるというものだ。

オブジェクトのシリアライズに際して，ObjectStream を用いて，オブジェクトインスタンスの情報とクラスデータを一つにまとめて送る事でリモート先での復帰を実現する。しかし，インスタンスからクラスデータを取得し，シリアライズデータに付加する形で転送を行うと失敗してしまう。

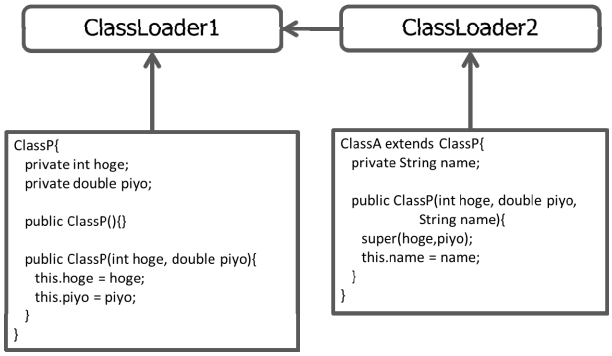

Fig.02.5-1 継承クラス例

インスタンスデータ中でのクラスの出現順序が継承関係などで必要とされる順序と異なるためで，オブジェクト復帰時にクラス解決の再帰呼び出しで失敗する為である。

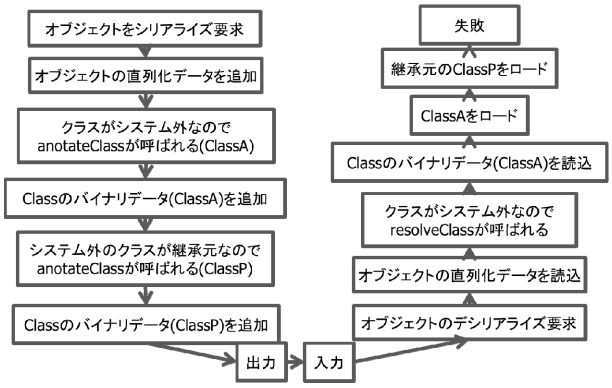

Fig.02.5-2 復帰に失敗する例

その為，インスタンスのシリアライズデータよりも前にクラスのデータを付加する形にデータの並び替えを行い。予めクラスローダ構成を動的に再構築した上でインスタンスのデシリアライズを行う様にし，復帰を可能とした。

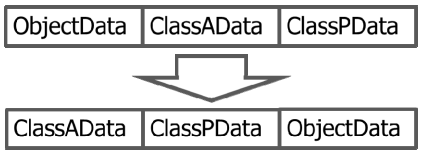

Fig.02.5-3 データ順序の変更

３．おわりに

本研究では，分散オブジェクトの復帰に関するJavaVM 上での現実的な実装手法を提案した。今年度の開発においては，先行研究から続く AgentSphere のコンセプトのうち分散性に関わる部分の開発となり，この分散処理におけるオブジェクトの非手続転送という部分は AgentSphere にとっては，システムの効率性や保守性を謳う為に必須の部分である。ネットワーク分散処理を扱うアプリケーション外にも，クラスローダ単体で見た場合にはオンライン取得からの動的更新自体に実用性があり，さらにオブジェクト転送を可能にする事でクラス構成環境と実行途中のインスタンスオブジェクトを移動できる。その為，分散処理の一つの形として，汎用性と妥当性を考慮した実装ができたと考える。

謝辞

本研究は科研費（基盤研究(C)21500041）の助成を受けたものであることをここに記し，謝意を表します。

参考文献

1） Akai,Y. Wakao,K. Yokouchi,T. Kai, M.: “Development of the strong migration mobile agent system AgentSphere for autonomic distributed processing”, Pacific Rim Conference 2009, pp.582-587