多様な言語システム環境に適応する大域メッセージパッシングアーキテクチャ

斉藤隆之　千葉哲央　前川博俊
(株)ディジタル・ビジョン・ラボラトリーズ

我々は，広域のネットワーク環境での大域並行計算を実現する計算方式を提案している．この方式におけるメッセージパッシング機構は，異なるプログラミング言語や，OS，ネットワーク環境が混在した多様な実行環境で機能できる．そのアーキテクチャは，言語システムやOSに独立な基本機能群と，言語システムに依存した機能群とで構成した．この構成によって，多様な言語システムにおいて，その実行環境を乱すことなく言語システムとOSの機能を活用した上で，メッセージパッシングに基づく大域的な並行計算を行うことが可能となった．本稿では，このアーキテクチャとそれに基づくメッセージパッシング機構のインプリメンテーションについて述べる．

Global Message-Passing Architecture Adaptive to Diverse Language-Systems Environment

Takayuki SAITO, Tetsuhiro CHIBA, and Hirotoshi MAEGAWA
Digital Vision Laboratories Corporation

We have proposed a computing scheme capable of developing a global concurrent computation over extensive network environment. The message passing feature in this scheme may work in execution environment of a variety of different programming languages, operating systems, and communication networks. The message passing architecture consists of basic features independent of languages and operating systems and those adopted in language systems. This architecture achieves a global concurrent computation based on message passing in diverse language-systems environment utilizing functions of language systems and operating systems without interfering with the execution environment of the language systems. In this paper, we describe the architecture and implementation of the message-passing feature.

1. はじめに

我々は，広域のネットワーク環境上で並行計算を行うことのできる大域計算(Global Computation)の方式を提案している[1]．この方式を実現しているメッセージパッシング機構[2]は，広域分散処理を可能とするとともに，プログラミング言語，OS，ネットワーク環境の違いによる多様な実行環境で機能することができる．本稿では，言語システム環境の違いに適応して機能することが可能なメッセージパッシングの方式について述べる．

ネットワークの発展と共に，広域での分散アプリケーションに対する要求が高まってきている．広域の環境では一般に，さまざまなネットワーク方式，OS，プログラミング言語が混在して機能している．したがって，そこではそれら実行環境の多様性に対応して分散アプリケーションを柔軟に構築する技術が求められる．従来，並行分散システムを実現するための技術として，分散処理言語が多数提案されているが[3-7]，それらは単一言語環境に限定されている．Java RMI[8]とHORB[9]は，多様なシステムプラットフォームに対応した分散システム構築機能を提供しているが，それらはやはり特定の言語に限定されたものである．Jini[10]は，分散オブジェクトをサービスの提供者・利用者という観点で緩く結合させることで分散システムを構築しようとしているが，並行性や非同期性のサポートなど広域での並行計算における重要な機能が不足している．COOL[11]は，多言語を扱っているが，密に結合されたネットワーク上での分散オブジェクト環境の提供に対応するものである．ILU[12]は，多言語間の接続を実現しているが，静的な接続の定義に基づくため，柔軟な分散アプリケーションの構築を促すものではない．CORBA[13]は，均一なオブジェクト空間と同期呼び出しを基本とするプログラミングパラダイムを有しており，広域での並行計算を可能とするものではない．

我々の大域計算方式では，広域のネットワーク上に配置されたプログラムモジュールによって計算空間を構成し，それらモジュールの機能が協調して計算を進

める．それらのモジュールは，異なる言語システムとOS の環境によるものでも機能する．このような広域での分散処理は，実行環境の多様性に対応したメッセージパッシング機能を提供するミドルウェアの実現によって可能となった．本稿では，大域計算の方式と，それを実現するためのミドルウェアの要件を示し，多様な言語システムに適応するアーキテクチャを提案する．また，それに基づくインプリメンテーションについて述べる．

2. メッセージパッシングアーキテクチャの要件と構成

ここでは，我々が前提としている大域計算のモデルを示し，多様な言語システム環境に適応できるミドルウェアの要件とその実現方法について論じる．

2.1 大域計算の方式

大域計算とは，広域のネットワーク上に分散した資源を論理的に一体のものとして捉え，ネットワークと分散資源で構成された統合環境上で行われる計算である[1]．この計算方式の特徴は以下の通りである．

(1) 広域のネットワーク環境上に分散した機能モジュール(並行オブジェクト)がメッセージによって互いの機能を呼び出し，そのメッセージの連鎖によって全体の計算が進行する．

(2) アプリケーション上のプログラミングパラダイムとして，分散手続き呼び出し，分散オブジェクト指向計算，大域共有変数の3つを提供する．並行オブジェクトは，それらに対応してそれぞれ，分散手続き(関数)，分散オブジェクト，および大域共有変数という3種類の形態をとる．

(3) メッセージパッシングは，つぎの3方式による．

- 戻り値のない非同期メッセージ送信(Send)
- 戻り値を非同期に得る遅延評価型同期呼び出し(Call)
- 戻り値が返信されるまでブロックする完全同期呼び出し(SyncCall)

また，メッセージに付けられた時間属性と優先度に基づいて，メッセージ評価のスケジューリングを行う[14]．

(4) 並行オブジェクトは，広域環境上でアプリケーションを構成する計算空間において，大域参照で管理する[15]．

並行オブジェクトは，ネットワークノード上にプロセスを生成して管理する．プロセスは，上に挙げた機能を実行環境として提供する(図1)．

2.2 ミドルウェアの要件

異なる言語システムによる実行環境上で，オブジェクト間のメッセージパッシングを実現するには，OS とネットワーク環境の違いも考慮して，次の要件が求められる．

(1) **非同期性，並行性**：適用問題の並行性とメッセージ通信の非同期性に対応するため，並行オブジェクトに対するメッセージ評価の実行は，並行に処理できること．

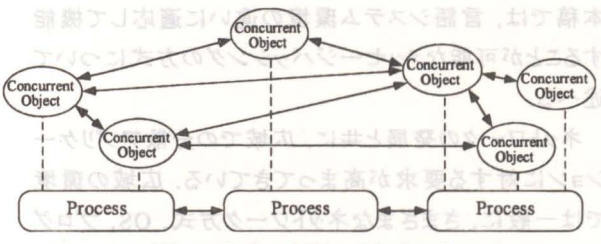
(2) **言語非依存性**：既存の言語に対して処理系の拡張を伴う仕様変更を行うことなく，メッセージパッシングに基づいた大域計算のプログラミング機能を提供できること．また，静的なインターフェース仕様を定義することなく，関数やメソッドを並行オブジェクトの機能として使用できること．

(3) **言語システムに対する非干渉性**：言語システム内のスタックやプロセス管理などの実行環境を破壊しないこと．そして，そのうえでメッセージ通信とユーザ機能のそれぞれの並行処理を可能にすること．

(4) **データ構造変換**：分散処理であることに加えて，異種のプロセッサ，OS，言語間でのメッセージ通信を可能にするため，メッセージの引数と戻り値のデータ構造体は，論理的な共通表現に変換して送信できること．

(5) **スケジューリング**：時間依存の問題に対処するため，メッセージ評価の実行をスケジューリングできること．

(6) **ネットワークの非均質性，多重性**：複数のプロトコルが混在する非均質なネットワーク環境を対象としているので，メッセージ通信経路の違いに対応して通信プロトコルを切り換えて使用できること．



Concurrent Object: Distributed Object, Distributed Function, or Global Shared Variable
Process: Distributed Execution-Environment

図1 並行オブジェクトの実行環境

2.3 アーキテクチャ

機能実現の方式については，次のことを考慮した．

- 多様な言語システムおよび OS への対応の観点から，言語に依存しない基本機能を共通機能としてまとめたい．共通機能は，システム独立な機能レイヤとして位置づけ，さらに，標準的なシステムコール API を用いて移植性を向上させたい．
- 並行実行機能をもつ言語システムは，一般にその内部で並行実行を制御している． また，ガーベジコレクション，関数クロージャの保持と駆動などを行っている言語システムもある．したがって，言語システム内部の実行環境の保全のためには，言語システム側主導でメッセージ送受信処理を制御することが必要である．
- ネットワーク環境の多様性への対応の観点から，通信プロトコル処理機能をプロトコル毎にモジュール化して拡張性を向上させたい．

以上の方針に基づいて構成したメッセージパッシングアーキテクチャを，図2に示す．そのアーキテクチャは，OS と言語システムに依存しないシステム独立機能部と，言語システム内の機能部に区分した．以下に各機能要素を説明する．

2.3.1. システム独立機能部

システム独立機能部は，言語システムと OS から独立した以下の5つの基本機能から構成した．

Message Sender: メッセージの送信処理と戻り値の受信処理を行う．複数のメッセージ送信を並行して行うために，言語システムから並列に呼び出すことができる．

Message Dispatcher: メッセージ受信の制御を行う．言語システムに対する非干渉性を実現しながら，複数のプロトコルでのメッセージ受信を並行して処理するため，言語システム内の Task Dispatcher から並列に起動する．

Communication Media Interface(CMI): メッセージの送信と戻り値の受信を行う通信プロトコル処理モジュールである．プロトコルは，メッセージの送信先プロセスとのネットワーク上の位置関係(ノード内，LAN 内，広域網におけるサイト間など)に対応して複数用意する．CMI は，これらプロトコル毎の独立したモジュール群で構成する．

Message Evaluation Scheduler(スケジューラ): メッセージに付帯した優先度と時間属性に基づいて，メッセージ評価の順序を制御する．OS 機能に依存しない広域のスケジューリングを可能とする．

Reference Resolver: 並行オブジェクトと分散オブジェクトメソッドへの大域参照を管理する．メッセージ送信の際には，その大域参照から，送信先オブジェクトが存在するネットワークノードとプロセス，およびメッセージ送信のための通信プロトコルを特定する．メッセージ受信の際には，並行オブジェクトと分散オブジェクトメソッドのプロセス内での所在を特定する．

2.3.2. 言語システム内機能部

言語システム内には，システム独立機能部と連係する次の機能を設けた．

Message Launcher: メッセージ送信要求をユーザ機能から受けつけ，Message Sender を呼び出して送信処理および戻り値の受信処理を行う．

Task Dispatcher: 言語システムの並行実行機能を使用して，Message Dispatcher にタスクを割り当て，メッセージの受信処理を起動する．メッセージに対する待ち受けタスクの多重度は，Task Dispatcher が制御する．

Function Invoker: ユーザ機能の関数とメソッドを起動するための関数である．Function Invoker とそれによって起動される関数とメソッドの実行は，システム独立部のスケジューラによって間接的に制御される．

Serializer/Expander: メッセージ引数および戻り値のデータ構造を，内部表現と共通表現の間で変換する．

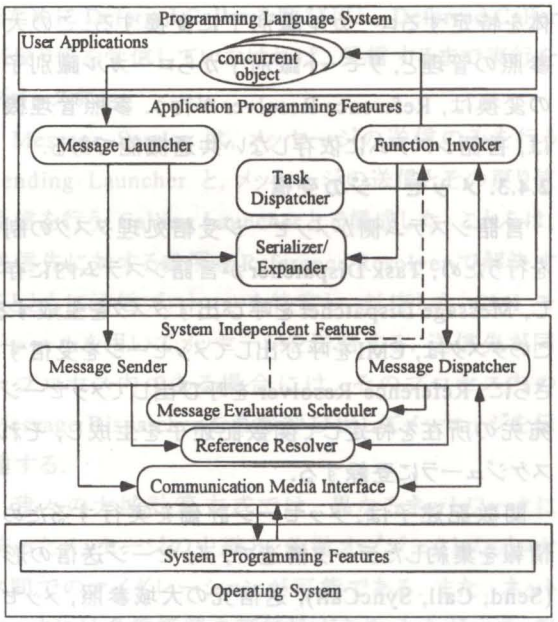


図 2 メッセージパッシング機構のアーキテクチャ

2.4 メッセージ送受信の機構

メッセージ送受信のための主な機能の位置づけと機構の概要を述べる.

2.4.1. メッセージの送信

Message Launcher は, アプリケーションプログラミング言語の言語仕様に適したメッセージ送信 API を提供する. また, メッセージの引数として指定されたデータ構造を, Serializer を呼び出し, 論理的な共通表現に変換する.

Message Sender は, この引数データと送信先への参照を受け, Reference Resolver によって参照の解決を行い, CMI によってネットワークへ送出する. CMI は, OS が提供する通信機能を活用して機能する.

メッセージの送信先がオブジェクトクラスタや任意の並行オブジェクトをまとめたドメイン空間になっている場合には, マルチキャストを行う. マルチキャストは, 送信先の所在に基づくルーティングを行う設計である. ただし, 現在は単一メッセージの繰り返しで実現している.

2.4.2. 参照解決

メッセージの送信先は大域参照で指定する. オブジェクトの大域参照は, ノードのネットワークアドレス, ノード内で有効なプロセス識別子, プロセス内で有効なリモート識別子で構成する. リモート識別子は, 並行オブジェクトまたは分散オブジェクトメソッドに対して割り当てる. メッセージ受信においては, リモート識別子からプロセス内部のオブジェクトまたはメソッドの実体を特定するローカル識別子に変換する. この大域参照の管理と, リモート識別子からローカル識別子への変換は, Reference Resolver が行う. 参照管理機能は, 言語システムに依存しない共通機能である.

2.4.3. メッセージの受信

言語システム側がメッセージ受信処理タスクの制御を行うため, Task Dispatcher が言語システム内に存在し, Message Dispatcher を呼び出すタスクを生成する. このタスクは, CMI を呼び出してメッセージを受信する. さらに, Reference Resolver を呼び出してメッセージの宛先の所在を特定して関数記述子を生成し, それをスケジューラに登録する.

関数記述子は, メッセージ評価を実行するための情報を集約したデータ構造で, メッセージ送信の形式 (Send, Call, SyncCall), 送信元の大域参照, メッセージの宛先である並行オブジェクトまたは分散オブジェクトメソッドのローカル識別子と, 内部表現に変換された引数データの所在, スケジューリングのための情報を保持する. スケジューリングは, 関数記述子をその対象として機能する.

2.4.4. メッセージ評価のスケジューリングとその実行

スケジューリングは, 長期スケジューリングと短期スケジューリングの 2 つの処理ステージに分けて実現した[14]. 評価すべきメッセージの選択は, 長期スケジューリングにおいて, 関数記述子上のスケジューリング情報に基づいて行う. 並行オブジェクトの呼び出しである評価の実行, すなわち, Function Invoker の呼び出しは, 短期スケジューリングによって制御する. スケジューリングは, 時間の制御方法に関するいくつかのポリシーから選択して指定することができる. 長期スケジューリングは OS 独立なミドルウェア機能として, 短期スケジューリングは OS 機能へのインターフェースとして位置づけることができる.

リモート識別子とそれに対応するローカル識別子では, 多重定義関係にある関数とメソッドには同一の識別子を割り当て, Function Invoker が引数並びの違いから関数とメソッドを特定する. 多重定義された関数やメソッドから適切なものを特定する方式は言語依存であり, 引数のデータ型並びの解析と, 適切な関数およびメソッドの選択と呼び出しは, 言語システム内の機構として位置づけた.

3. インプリメンテーション

このアーキテクチャに基づいて実現したメッセージパッシング機構は, アプリケーションプログラミング言語として現在, 実行効率のよい言語として C++, バイトコードインタプリタ形式の言語として Java に対応している. さらに動的機能が豊富な言語である Common Lisp への対応を検討している.

3.1 機能実現の構成

メッセージパッシング機構の構成と, プロセス, 言語システムおよび OS との関係を, 図3に示す. Message Sender と Message Dispatcher からなる Message Passing Operator を, メッセージパッシングのカーネル機能として位置づけた. この Message Passing Operator を中心に, Reference Resolver, スケジューラ, CMI でシステム独立機能部を構成した. システム独立機能部は, 多様な言語システムとの接続を容易にするとともに処理性能を高めるため, C および C++でインプリメントした.

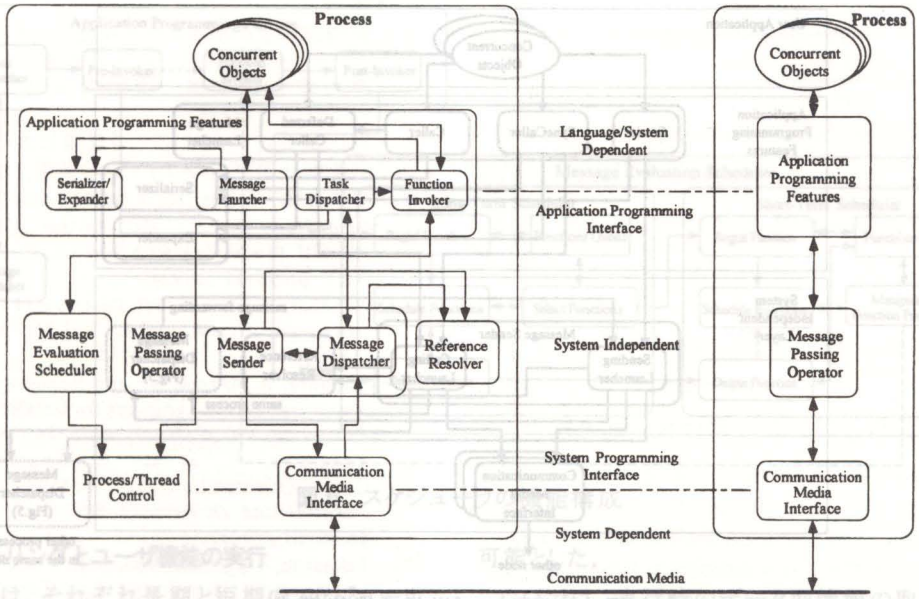


図3 メッセージパッシング機構の機能構成

言語システムとシステム独立機能部は，言語システムが提供する外部機能インターフェースを用いて接続した．Java の場合は JDK1.1 の Java Native Interface[16]を使用した．システム独立機能部は，ダイナミックリンクライブラリとして構成し，機能の差替えを容易とした．

並行実行機能については，C++，Java ともに Solaris，Windows NT のマルチスレッド API を使用した[1]．Java のバイトコードインタプリタには，OS のネイティブスレッドをベースにしたものを使用した．メッセージパッシング機構におけるタスクは，これらスレッド機能によって実現した．現在はマルチスレッド機能をもつ OS の上でインプリメントしているが，シングルスレッドの OS の場合は，システム独立部内でブロックせず，言語システム部へ制御を常に戻す設計である．

3.2 機能構成の詳細

メッセージ送信処理，受信処理の機能構成の詳細と処理の流れを以下に説明する．

3.2.1. メッセージ送信の機能構成と処理フロー

メッセージ送信のための機能モジュールの構成を，図4に示す．

Message Launcher は，3種類のメッセージパッシング方式 Send，SyncCall，Call に対応した Sender，SyncCaller，Caller と，遅延評価型同期呼び出しの送信と戻り値受信を行う Deferred Caller の4つのモジュールで構成した．それぞれのモジュールは，メッセージの引数もしくは戻り値のデータ構造を変換するため，Serializer もしくは Expander を呼び出す．

Sender は，Message Sender を呼び出すタスクを生成して，直ちにユーザ機能に制御を戻す．SyncCaller は，ユーザ機能から呼ばれたタスク内で Message Sender を呼び出すことで同期呼び出しを行う．Caller は，Deferred Caller を呼び出すタスクを生成して制御をユーザ機能に戻す．Deferred Caller は，Message Sender を機能させる．ユーザ機能は，戻り値を取得するために Deferred Caller を呼び出し，Deferred Caller が戻り値を受信していなければ，受信するまで実行をブロックする．

Message Sender は，メッセージの送信のみを行う Sending Launcher と，メッセージの送信とその戻り値受信を行う Calling Launcher とで構成した．これらは，送信先に対する参照を Reference Resolver で解決するとともに通信プロトコルを特定し，対応した CMI モジュールを用いてメッセージを送信する．送信先が同一プロセス内である場合には，そのプロセス内の Message Dispatcher を直接呼び出してメッセージを伝達する．

我々の大域計算方式では，異なるネットワークに跨ったメッセージの中継と，分散オブジェクトのプロセス間でのマイグレーションが可能である．また，ネットワークには多重経路の可能性がある．したがって，戻り値は送信時と同じ通信プロトコルで返信されるとは

[1] 移植性を高めるため，POSIX[17]への対応を考慮したインプリメンテーションにしている．

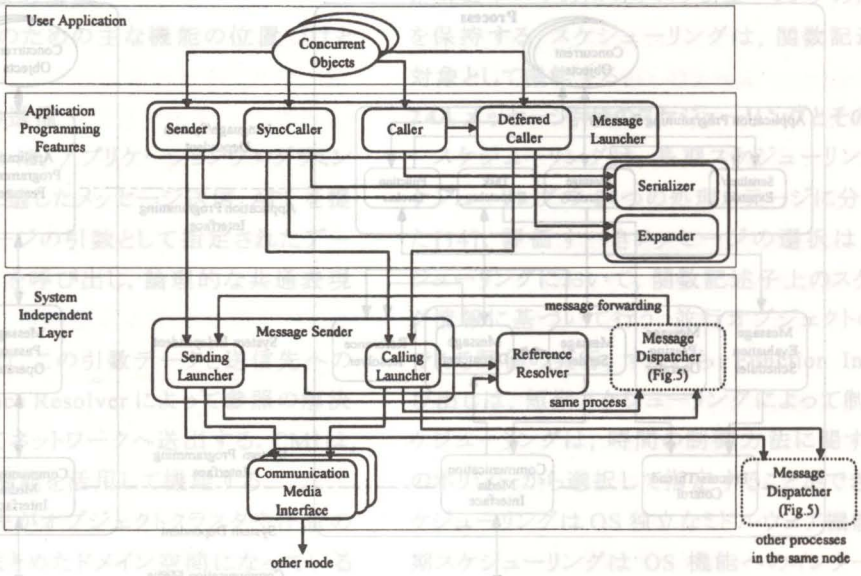


図4 メッセージ送信の機能構成

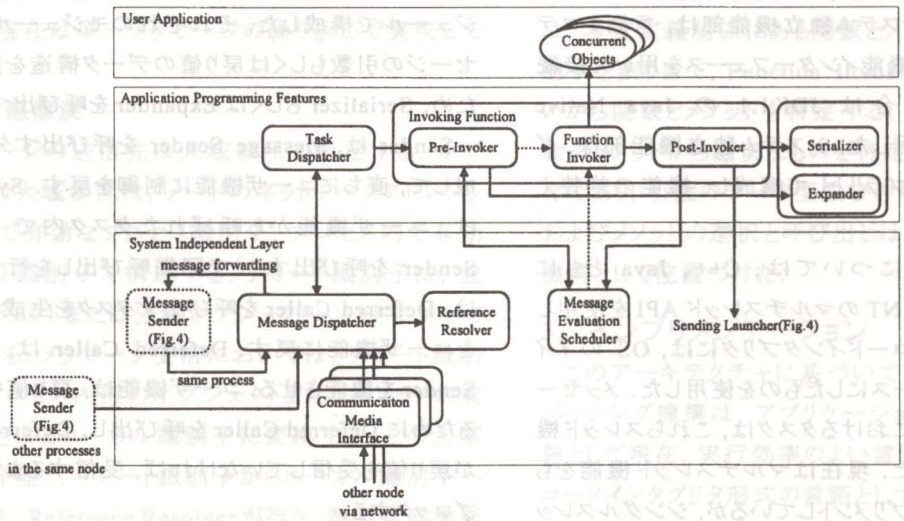


図5 メッセージ受信の機能構成

限らない．戻り値の受信は，全ての通信プロトコルで共通のタスクキューを用いて処理する．

3.2.2. メッセージ受信の機能構成と処理フロー

メッセージ受信のための機能モジュールの構成を，図5に示す．

Task Dispatcherは，Message Dispatcherを起動するタスクを複数生成する．これらのタスクは指定されたCMI を呼び出してメッセージ受信待機を行う．着信したメッセージの宛先に応じて，つぎのように処理が分岐する．

- 宛先がそのプロセス内の並行オブジェクトであれば，関数記述子を生成しTask Dispatcherに制御が戻る．
- 宛先が他プロセスの並行オブジェクトであれば，Message Senderを呼び出してメッセージを転送したのち，再びメッセージ受信に戻る．この転送処理は，メッセージの中継やマイグレーションによる転送に対処するためである．

Task Dispatcher へ制御が戻った場合，Task Dispatcher は，新たなメッセージ受信待ちのためにMessage Dispatcherを呼び出すための新たなタスクを生成する．同時に，Message Dispatcherから受け取った着信メッセージの処理は，Pre-Invokerに移行する．

Pre-Invokerは，メッセージの引数をExpanderによって内部表現へ変換し，その所在を関数記述子に設定して，その関数記述子をスケジューラに登録する．

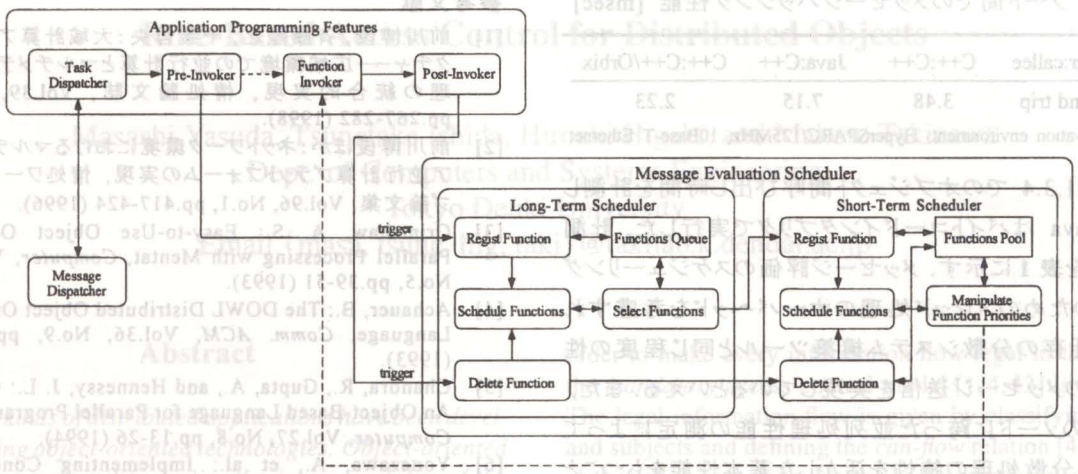


図6 スケジューラの機能構成

3.2.3. スケジューリングとユーザ機能の実行

スケジューラは，それぞれ長期と短期のスケジューリングを行う長期スケジューラと短期スケジューラで構成した．その構成とFunction Invokerとの関係を，図6に示す．

長期スケジューラは，登録された関数記述子をキューに格納し，評価を起動すべきものをその時間属性と優先度に基づいて順次選択する．選択した関数記述子は，キューから取り出し，短期スケジューラに送る．短期スケジューラでは，関数記述子の情報を基にFunction Invokerを実行するためのタスクを生成する．タスクの実行は，OS のスレッド機能を活用して制御する．

Function Invokerは，関数記述子から特定される関数もしくはメソッドを実行し，その実行が終了するとPost-Invoker に処理を移行させる．Post-Invoker は，当該の関数記述子の評価が完了したことをスケジューラに通知する．Post-Invoker は，実行した関数もしくはメソッドの戻り値が要求されていれば，戻り値を Serializer で共通表現に変換したのち，Message Senderを呼び出してメッセージ送信元へ返信する．

3.2.4. データ構造変換

Serializer は，基本データ型とユーザ定義のデータ型からなるデータ構造体を，論理的な共通表現に変換する．データ構造体がオブジェクトグラフの場合には，オブジェクトの参照をトレースしながら共通表現に変換する．Expander は，データの共通表現からメモリ上の内部表現でのデータ構造体に展開する．共通表現には，基本データ型，クラス，構造体などの定義情報を含めることにより，受信側でデータ構造の解析を可能とした．

C++では，実行時のデータ型情報の取得が困難なため，アプリケーションソースコードを解析して言語システム内の機能モジュールのソースコードを生成するプログラムトランスレータを開発した．ユーザ定義のデータ型に対応する Serializer と Expander は，アプリケーションの記述に応じてトランスレータが生成する．Java では，実行時のデータ型情報の取得機能とオブジェクトシリアライゼーション[18]の機能を用いて Serializer と Expander を実現した．[2]

大域参照は，Reference Resolver 内の参照表で管理している．したがって，大域参照を表現するデータ型に対応する Serializer は，Reference Resolver から参照情報を取得し，これを共通表現に変換する．また，Expander は，受信した大域参照の情報を Reference Resolver に登録する．これによって，大域参照もメッセージ上で引数として扱える．

3.3 大域計算フレームワークシステム

このメッセージパッシング機構による大域計算のフレームワークシステムが，10Base-T Ethernet と 156Mbps ATM で接続された Solaris 2.4 /Sparc Station と Windows NT 4.0/IBM PC/AT 互換機上で稼動している．計算方式の評価については[1]で詳述した．ここでは，メッセージ送信の基本性能の測定結果を示す．2台の HyperSPARC/125MHz のノード間で，完全同期呼び出しを用いた並行オブジェクト間メッセージ送信の往復時間を測定した．比較のために

[2] C++では，オブジェクトグラフの循環と合流に対応するため，アドレステーブルによる検出機能を組込む予定である．現仕様では，部分構造が共有されている場合には，送信先で共有部分が別々に複製される．また，循環構造は扱えない．

表 1 ノード間でのメッセージパッシング性能 [msec]

caller:callee	C++:C++	Java:C++	C++:C++/Orbix
round trip	3.48	7.15	2.23

Observation environment: HyperSPARC/125MHz, 10Base-T Ethernet

Orbix1.3.4 でのオブジェクト間呼び出し時間を計測した. Java はバイトコードインタプリタで実行した. 計測結果を表 1 に示す. メッセージ評価のスケジューリングとそのためのスレッド処理のオーバヘッドを考慮すれば, 既存の分散システム構築ツールと同じ程度の性能でのメッセージ送信を実現しているといえる. また, 複数のノードに跨った並列処理性能の測定によって [14], 分散処理の特性を活かした基本性能をもつことを確かめている.

4. まとめ

本稿では, 多様なプログラミング言語システムに適応可能なメッセージパッシング機構のアーキテクチャとインプリメンテーションについて述べた.

このアーキテクチャは, 言語システムと OS から独立したシステム独立機能部と, 言語システム内機能部とで構成した. そこでは, 言語システム内機能部が主導でタスクの制御を行うことで, 言語システムの実行環境を乱すことのない処理を実現した. システム独立機能部は, 複数の通信プロトコル群を備え, 多様なネットワーク形態に対応した. システム独立機能部のメッセージ処理と, 言語システム内機能部でのメッセージ評価の実行は, 並列に機能することが可能であり, メッセージの並行性と非同期性に効率よく対応できる. さらに, メッセージ評価の実行はスケジューリングすることが可能である. 効率のよいメッセージパッシング処理と, 言語システムを含めた多様な実行環境へのその適応性によって, 広域での分散処理に対応したミドルウェア機能を実現した.

このアーキテクチャに基づいて, アプリケーションプログラミング言語として C++と Java に対応した大域計算フレームワークシステムを, Solaris 2.4 ならびに Windows NT 4.0 の上にインプリメントした.

今後, さらに多様な言語への対応によって, 提案したアーキテクチャの有効性を検証する. また, 大域並行計算による広域での応用システムの開発を行い, 提案するメッセージパッシング機構の有効性を検証していく予定である.

参考文献

[1] 前川博俊, 斉藤隆之, 千葉哲央:大域計算アーキテクチャ——広域環境での並行計算とマルチメディア処理の統合的実現, 情処論文誌, Vol.39, No.2, pp.267-282 (1998).

[2] 前川博俊ほか:ネットワーク環境におけるマルチメディア並行計算プラットフォームの実現, 情処ワークショップ論文集, Vol.96, No.1, pp.417-424 (1996).

[3] Grimshaw, A. S.: Easy-to-Use Object Oriented Parallel Processing with Mentat, *Computer*, Vol.26, No.5, pp.39-51 (1993).

[4] Achauer, B.: The DOWL Distributed Object Oriented Language, *Comm. ACM*, Vol.36, No.9, pp.48-55 (1993).

[5] Chandra, R., Gupta, A., and Hennessy, J. L.: COOL: An Object-Based Language for Parallel Programming, *Computer*, Vol.27, No.8, pp.13-26 (1994).

[6] Yonezawa, A., et al.: Implementing Concurrent Object-Oriented Languages on Multicomputers, *IEEE Parallel and Distributed Technology*, Vol.1, No.2, pp.49-61 (1993).

[7] Takashio, K., Shitomi, H., and Tokoro, M.: Constructing Distributed Real-Time Systems with DROL Real-Time Objects, *Proc. 2nd Int'l Workshop on Real-Time Computing Systems and Applications*, IEEE Computer Soc. Press, Los Alamitos, pp.142-149 (1995).

[8] Wollrath, A., Waldo, J., and Riggs, R.: Java-Centric Distributed Computing, *IEEE Micro*, Vol.17, No.3, pp.44-53 (1997).

[9] Hirano, S.: HORB: Distributed Execution of Java Programs, *Proc. 1st Int'l Conf. Worldwide Computing and Its Applications*, Lecture Notes in Computer Science 1274, Springer, Berlin, pp.29-42 (1997).

[10] Waldo, J.: Jini Architecture Overview, Sun Microsystems, Palo Alto; http://www.java.sun.com/products/jini/whitepapers/architectureoverview.pdf (Oct. 1, 1998 現在).

[11] Lea, R. et al.: COOL: System Support for Distributed Programming, *Comm. ACM*, Vol.36, No.9, pp.37-46 (1993).

[12] Janssen, B. et al.: *ILU 2.0 Reference Manual*, Xerox Corp., Palo Alto; ftp://ftp.parc.xerox.com/pub/ilu/2.0a12/manual-html/manual_toc.html (Oct. 1, 1998 現在).

[13] *The Common Object Request Broker: Architecture and Specification*, Wiley, New York (1991).

[14] 前川博俊, 千葉哲央, 斉藤隆之:大域並行計算とそのメッセージ評価スケジューリング, 情処研究報告, 98-OS-77, 98-DPS-87, pp.97-102 (1998).

[15] 小早川雄一, 斉藤隆之, 前川博俊:ネットワークスケーラブルな分散オブジェクト空間管理方式, 情処研究報告, 97-DPS-80, pp.211-216 (1997).

[16] *Java Native Interface Specification*, Sun Microsystems, Palo Alto; http://www.java.sun.com/products/jdk/1.1/docs/guide/jni/spec/jniTOC.doc.html (Oct. 1, 1998 現在).

[17] *Information Technology – Portable Operating System Interface (POSIX) – Part 1: System Application Program Interface API [C Language]*, IEEE, New York (1996).

[18] *Java Object Serialization Specification*, Sun Microsystems, Palo Alto; http://www.java.sun.com/products/jdk/1.2/docs/guide/serialization/spec/serialTOC.doc.html (Oct. 1, 1998 現在).