新しいネットワークソフトウェア基盤と グローバルネットワークへの応用

－ソフトウェアをインターネットへ解き放つ－

加藤　和彦

■ 研究のねらい

これまでソフトウェアはコンピュータという箱の中に閉じこめられたものでした。これは人間社会に例えれば、会社のオフィスから一歩も外には出ず、電話、FAX、手紙のみを使って外部と連絡をとりながら仕事をすることに相当しています。それだけでも限られた仕事はできますが、人間の持つ移動する能力を使うことによって、取引先に出向いていて交渉を行ったり、情報を収集したり、あるいは、専門知識を活かした作業を行うことができます。モバイルオブジェクト・コンピューティングとはまさに、人間が持つ移動能力をソフトウェアに与えようとする技術のことです。

モバイルオブジェクト技術に属すると考えられる技術の一部は、既に実用に供しています。社会で広く利用されるようになったWebシステムのホームページでは、ホームページ記述の記述力をあげるためにコンテンツの一部をスクリプト言語と呼ばれる一種のプログラミング言語を用いて記述することがごく一般的に行われています。あるいは、最近の多くのWebブラウザは、Java仮想機械と呼ばれるソフトウェアを含み、Java言語で書かれたプログラムコードをネットワークを介して自動的に取り寄せ、実行することができます。

私が「さきがけ研究21」で行った研究では、モバイルオブジェクトの機能を出来うる限り制約が少ない形で実現し、さらに、インターネットに代表される、地球規模のオープンな広域ネットワーク環境上で利用した場合の有効性を実証しました。モバイルオブジェクト技術を用いると次のようなことが可能になります。

限られたネットワークバンド幅の有効利用 ネットワークバンド幅（単位時間あたりの情報転送量）は限られています。最近は光ネットワーク等の広帯域のバンド幅の技術も実用的になっていますが、広域ネットワーク環境では、通信線を非常に多くのユーザで共有せねばならないことが多く、1ユーザあたりのネットワークバンド幅は限られています。モバイルオブジェクト技術を用いない、従来のソフトウェア技術— ここではそれを静止型と呼びます—では、すべての通信処理を広域ネットワークを介して行わねばならず、限られたネットワークバンド幅の影響を直接に受けてしまいます。それに対して、モバイルオブジェクト技術を利用すると、オブジェクトの移動時のみ広域ネットワークを利用し（図1参照）、移動はローカルエリアネットワーク

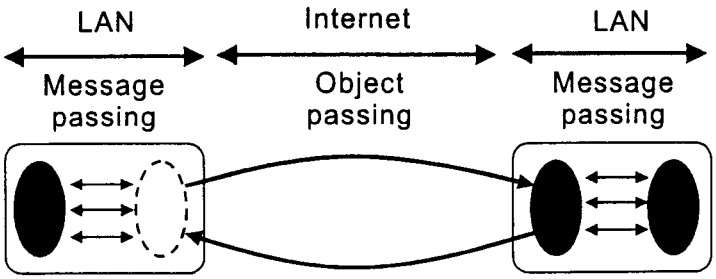

図 1: モバイルオブジェクト・コンピューティング＝オブジェクトパッシング＋メッセージパッシング。

(LAN) 環境の中に入り、高ネットワークバンド幅、低通信遅延、費用の安い通信等の LAN 環境の特性を活かして処理を行うことができます。

通信遅延の隠蔽 LAN 環境では通信遅延は無視できるほど小さいものですが、広域ネットワーク環境では大きな通信遅延を伴います。特に地球規模の通信においては、通信遅延は大きな問題となります。地球の裏側の地点との通信を考えた場合、仮にオーバヘッドが全くない理想的な通信が行えたとしても、1 秒間に最高約 7.5 回のメッセージの往復しかできません。モバイルオブジェクトの技術を用いると（図 1 参照）、広域ネットワークはオブジェクトの転送時にのみ使用し、メッセージパッシングは通信遅延が小さい LAN 環境の中で行うため、実質的に通信遅延を隠蔽することができます。

自律的コンピューティング モバイルオブジェクトにはプログラムコードを含ませることができます。このプログラムコードに、人間操作者になり代わって知的な作業や判断を行うようなプログラミングを行っておけば、モバイルオブジェクトがネットワーク上において自律性をもって移動しながら行動を行わせることが可能になります。

アプリケーションプログラムの動的構成 モバイルオブジェクトは、ソフトウェアを構成するオブジェクトを、ネットワーク環境上にて動的に構成することを可能にします。この機能を積極的に利用すると、広域ネットワーク上にて分散開発されたオブジェクトをネットワークを介して動的に集め、一つのアプリケーションに仕立てあげて実行することが可能となります。

モバイルオブジェクトの研究は、現在世界各地で活発に研究が進められており、上記の特徴はそれらのシステムも同様に有します。これらの特徴に加え、本研究で開発したモバイルオブジェクトシステム PLANET は、他の研究にはない下記のようなユニークな特徴を有しています。

モバイル基盤ソフトウェア (mobile substrate) アプローチ インターネットの社会的な普及により、今や広域ネットワークを介した通信を行う機能は、大型計算機からパーソナルコンピュータ、そして携帯端末に至るまで、ほとんどすべてのコンピュータが備える必須の機能となりつつあります。モバイルオブジェクトの機能が広域ネットワーク環境上で極めて有効な技術とするならば、モバイルオブジェクトのプログラミングはさまざまなプログラミング言語を用いて行えるべきでしょう。ところが他のほとんどのモバイルオブジェクトシステムは、プログラミング言

語層での実現を行っているため、基本的に一つのプログラミング言語のみしかサポートできません。PLANET では、さまざまなプログラミング言語のモバイル化を可能とするために、言語層とオペレーティングシステム（以下 OS と略す）層の間のミドルウェア層でモバイルオブジェクト機能を提供する、モバイル基盤ソフトウェア (mobile substrate) アプローチを採用しました。このアプローチでは、言語処理系層およびアプリケーションプログラム層から見た場合に、さも基盤ソフトウェア（OS）のメモリ管理機能、ファイル管理機能、およびネットワーク通信機能が連携してモバイルオブジェクトをサポートする機能を提供しているかのように見える設計となっています。しかも PLANET 自体は、個々の OS の実装と独立となるように設計されており、複数種類の OS 上で稼動し、オブジェクトを転送できるようになっています。実際、本研究では、C, C++, Tcl のような既存のプログラミング言語にまでモバイルオブジェクト機能を付与できることを実証しました。

自己拡張性 プログラミング言語層内でモバイルオブジェクトを実現したシステムでは、モバイルオブジェクトの機能を使用するすべてのサイトで、その言語のオブジェクトを実行する言語実行系が必要です。PLANET の場合も、PLANET の実行時システムはすべての利用サイトにインストールする必要はありますが、一度インストールしてしまえば、PLANET を用いてモバイル化されたさまざまなプログラミング言語処理系や言語処理系の部品を、サイト毎のインストール作業なしに使用することができます。これは前述のモバイル基盤ソフトウェアアプローチを採用しているためで、言語インタープリタや言語コンパイラ等のシステム構成部品自体をモバイルオブジェクト化して転送および自動インストールができるためです。自動的な自己拡張が可能なシステムと考えることもできます。

標準形式を固定しない異機種間オブジェクト転送 インターネット環境上ではさまざまな種類の計算機プラットフォームが稼動しています。一度作ったオブジェクトをさまざまなプラットフォーム上で稼動させるために標準形式 (canonical form) のオブジェクトを作成しておいて、各プラットフォームへ転送後に、解釈実行 (interpretation) を行ったり、そのプラットフォームでの実行に適した形式に翻訳して実行することが行われます。このような機能を有する他のシステムもありますが、それらは標準形式を一つの形式に固定しています。標準形式は、最上といえるものがあるわけではなく、今後もさまざまな改良が加えられた設計に基づいたものが考案され続けるでしょう。PLANET は自己拡張性を有するために、解釈実行系 (interpreter) や実行時翻訳系 (just-in-time compiler) をモバイルオブジェクトして実現することが出来るため、標準形式を固定せずに異機種間オブジェクト転送を達成できます。これは各プラットフォームへの自動適応機能とも考えられます。

非同期的オブジェクトパッシング・モデル 広域ネットワーク環境では、オブジェクトの送信側と受信側が、異なる管理ポリシーのもとで運営されているのが一般的です。もし、オブジェクトパッシング・モデルとして同期的なものを採用すると、オブジェクトの転送処理が、受信側の状態に依存したものならざるを得ません。PLANET は設計の最初からオープンな広域ネットワーク環境上で利用することを想定し、非同期型のオブジェクトパッシングモデルを採用し、送信者と受信者の独立性を高めています。他の多くのモバイルオブジェクトシステムが、同期型オブ

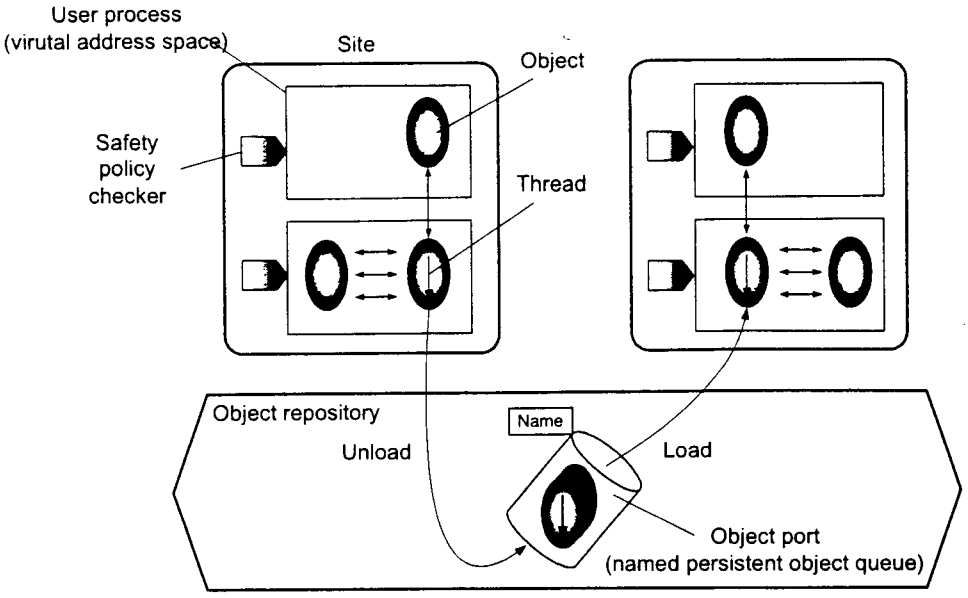

図 2: PLANET の基本システムモデル。

ジェクトパッシングモデルを採用しているのと対照的です。

■ 研究成果

モバイルオブジェクトシステム PLANET

一般にモバイルオブジェクトシステムを実現する場合、下記の処理を行う必要があります。

送信側：中断→整列化→エンコード→送信

受信側：受信→デコード→非直列化→実行の再開

多くのシステムではこれらの処理を、個々のプログラミング言語処理系の機能を拡張することにより実現しています。PLANET では、ミドルウェアレイヤにて、これらの機能をプログラミング言語独立に実装し、個々の言語を PLANET を用いてモバイル化する際の共通ツールとして利用できるようにしました。PLANET が提供するモバイルオブジェクトの基本機能は次の 3 つにまとめられます。

- モバイルメモリセグメント
- オブジェクトリポジトリとオブジェクトポート
- 安全な API 実行

モバイルメモリセグメントは、OS の仮想記憶管理、ファイル管理、そしてネットワーク通信機能を統合的に連携させたもので、このセグメント上に置かれた情報を、**オブジェクトリポジトリ**（以下、

リポジトリ）と呼ばれる一種の分散ファイルシステムを介して、異なるサイトにに転送します（図2参照)。ユーザはユーザプロセス内の仮想記憶上の一部にモバイルメモリセグメントを割り当てます。モバイルメモリセグメント上には、プログラムコード、データ、そして計算の状態（しばしばスレッドと呼ばれる）を置くことができます。モバイルメモリセグメントは仮想記憶空間とは独立な生存期間を持ちます（この性質は永続性(persistency)と呼ばれます)。モバイルオブジェクトの転送時には、モバイルオブジェクトが置かれたモバイルメモリセグメントがユーザ仮想記憶空間から取り外され、オブジェクトリポジトリに転送されます。この際、異機種間転送を行う場合は、モバイルメモリセグメント上の仮想記憶ページごとに標準表現へのエンコード処理が施されます。この一連の取り出し操作を**アンロード**操作と呼びます。

オブジェクトをリポジトリに格納する際には、リポジトリ上に存在する**オブジェクトポート**（以下、ポート）を指定します。ポートへの書き込み権をユーザプロセスが有する場合にのみこの書き込み操作は成功します。一つのポートは一個以上のモバイルオブジェクト（モバイルメモリセグメント）を格納するすることができ、FIFO (first-in-first-out) 順で格納操作と取り出し操作が行われます。オブジェクトポートには、位置独立で、分散システムワイドで一意に定められた論理名が付与されます。リポジトリは物理的には複数のサーバマシンによって構成可能ですが、分散名前管理機構により、ユーザはポートが実際にどのサーバに保持されているかは認識する必要はありません。

オブジェクトを受信する場合には、ユーザプロセスは、そのオブジェクトを含むモバイルメモリセグメントが格納されたリポジトリ上のポートを指定します。このとき、そのユーザプロセスは、そのポートからオブジェクトを取り出す権利を有している必要があります。取り出されたモバイルメモリセグメントは、ユーザプロセスの仮想アドレス空間に割り当てられます。この際、異機種間転送を行う場合は、モバイルメモリセグメント上の仮想記憶ページごとに、標準表現からユーザプロセスが動作するCPUのネイティブコードへのデコード処理が施されます。この一連の受信操作を**ロード操作**と呼びます。

メモリセグメント上には一般にアドレス参照（いわゆるポインタ）が含まれます。メモリセグメントの送受信が行われた場合、異なるアドレス位置にオブジェクトがロードされるのが一般的であるため、アドレス参照情報に関して何らかの調整処理が必要となります。現在のPLANETの実装では、同一メモリセグメント内のアドレス参照について一貫性を保持するように、アドレス調整を自動的に行います。このために、PLANETシステムはメモリセグメントのどの位置にアドレス参照情報があるかを理解している必要があります。この情報は、プログラミング言語処理系から、および、実行中のプログラムそのものから受け取ります（後者は、プログラムのみがその情報を知り得る場合があるため)。異機種間転送を行う場合には、標準表現へのエンコード、デコード処理を述べましたが、この処理を行うためにはメモリセグメント上のデータのデータ型が必要です。論理的なデータ型を保存するようにエンコード、デコード処理を行う必要があるからです。この情報も、アドレス参照情報と同様、プログラミング言語処理系から、および実行中のプログラムそのものから直接的に受け取ります。

現在のPLANETの実装では、モバイルメモリセグメントの転送に、**リモートメモリマップ**技術を用いています。これは仮想記憶管理技術とネットワーク転送技術、そして二次記憶管理技術を統合したもので、CPUが参照したモバイルメモリセグメントの仮想記憶ページをページフォールト例外を用いて実行時に検出し、そのタイミングでユーザプロセスが動くマシン（クライアントマシン）から

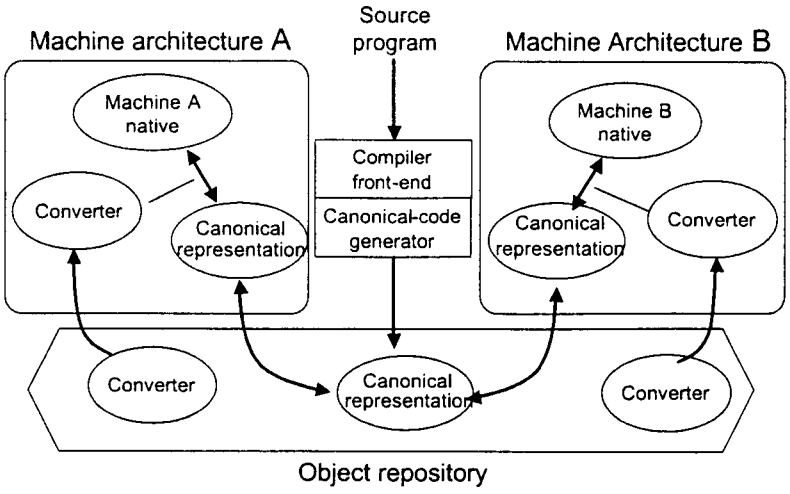

図 3: 異機種オブジェクトモビリティの実現。

リポジトリサーバにページ転送要求を送り、それに対する返答として、ページ内容をサーバが送り返すというものです。サーバにメモリセグメントが格納される際にデータ圧縮処理を施しておき、データページは圧縮された状態で転送され、ページフォールト例外が解除される直前に解凍処理が施されます。以上の機構により、ネットワーク転送量は最小化されています。またこれらの処理はすべてミドルウェアレイヤにて実行され、ユーザプログラムには透明になっています。

安全な API 実行とは、オープンなネットワーク環境上で、外部から飛来したモバイルオブジェクトに含まれるコードを安全に実行するために基本機構です。計算機資源の操作は一般的に API (Application Program Interface) と呼ばれる基本プリミティブの呼び出しによって行われます。プログラムの実行はセーフティポリシーチェッカーと呼ばれるプログラムが API 呼び出しの内容を実行時に検査しながら行います。セーフティポリシーチェッカーに組み込まれたセーフティポリシーに反する API 呼び出しの実行は強制的に停止させられます。

異機種オブジェクトモビリティ：モバイルコンパイラ

広域ネットワーク環境では、さまざまな機種のコンピュータシステムが稼働しているため、モバイルオブジェクトはそれらさまざまな機種上で動作できる必要があります。PLANET では、モバイルオブジェクト機能を用いて、異機種環境上でオブジェクトを動作させる機能を実現しています（図 3 参照)。ソースプログラムはコンパイラのバックエンド部 (canonical-code generator) を用いて正準表現 (canonical representation) に変換され、リポジトリに格納されます。正準形式のオブジェクトがアーキテクチャA のマシンにロードされると、正準形式をアーキテクチャA のネイティブ形式に変換するための変換器 (converter) をリポジトリよりロードします。この変換器自身もネイティブコード形式

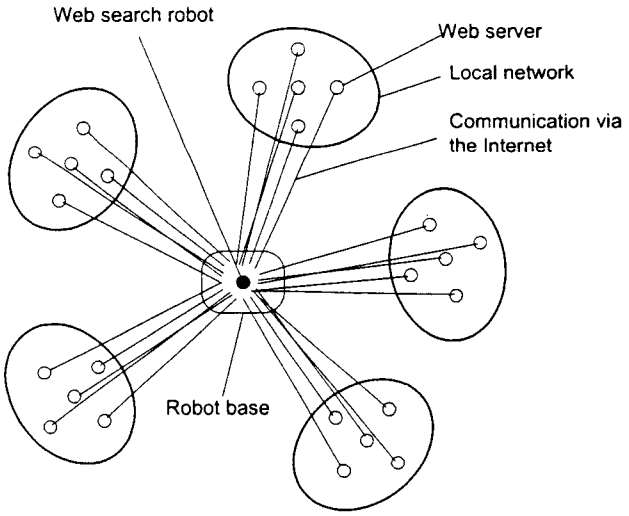

図 4: 静止型 Web サーチロボット。

のモバイルオブジェクトとして実現されています。変換器は正準表現とマシン A ネイティブの間で、プログラムコード、データ、そして計算の状態に関する双方向変換を行う機能を持ちます（ただしプログラムコードは、pure code 方式を用いることにより、正準表現からネイティブコードへの変換を行うのみでよい)。オブジェクトのアンロード時には、この変換器を用いて、ネイティブ表現から正準表現へ変換した後、リポジトリに格納します。マシンアーキテクチャBにロードするときは、上に述べたことと同様のことが、正準表現とアーキテクチャB の間の双方向変換器を用いて行われます。

このアプローチが面白いところは、変換器そのものをモバイルオブジェクトとして扱っているところです。これにより、正準表現を特定のものに固定する必要がなく、さまざまな正準表現を共存させることが可能です。また変換器のバージョンアップや新規導入のような作業も、クライアント側計算機への操作を必要とせず、自動的に行うことができます。

モバイル Web サーチロボット

広域ネットワーク上で動作するモバイルオブジェクト技術を用いたアプリケーションの作成を PLANET を用いて実際に行いました。現行の Web サーチロボットは図 4 に示すようにロボットは静止していて、インターネットを介して Web 情報 (HTML 形式のテキストデータ) を収集します。モバイルオブジェクト技術を用いると図 5 のようにロボットを、Web サーバが存在するサイト、もしくは同サイトが存在する LAN に送り込み、高速で安価な通信が可能なローカル通信もしくは LAN 通信を用いて Web サーバより情報を収集することができます。収集の際に必要ならば、情報フィルタリング等の処理を行い、有用な情報のみを収集することも可能です。収集したデータは一つのファイルにまとめ、圧縮して回収することができます。

この方式の有効性を検証する実験を、筑波大学とスウェーデン国ウプサラ大学の間の実験環境を用いて行いました。モバイル型 Web サーチロボット方式では、筑波大学からウプサラ大学へインター

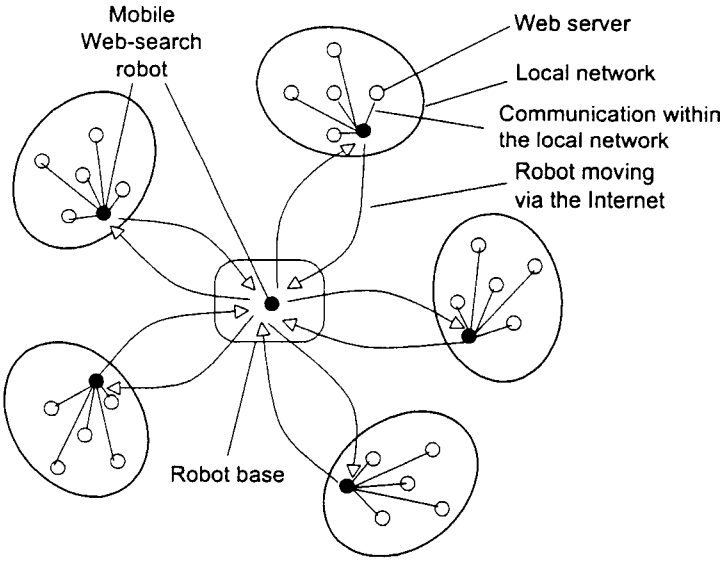

図 5: モバイル型 Web サーチロボット。

ネットを用いてモバイルロボットを移動させ、3MB の HTML 情報を収集後、再びインターネットを用いて筑波大学へロボットを帰還させます。このとき、PLANET のデータ圧縮機構が自動的に動作します。静止型 Web サーチロボット方式では、筑波大学で動作する静止型ロボットがインターネットを介してウプサラ大学より同一の 3MB の HTML 情報を収集します。実験に使用した計算機は、両大学側とも、Sun Ultra-60 (CPU: UltraSPARC-II 360 MHz, main memory: 640 MB, OS: Solaris 2.6) です。実験結果を図 6 に示します。インターネットを介して転送された IP パケット量は、モバイル型ロボット方式が合計 2.79MB（ロボットと HTML テキスト、共に圧縮されている)、静止型ロボット方式の場合が 2.77MB(HTML テキスト）でした。モバイル型ロボットが約 140 秒、静止型ロボットが約 270 秒で動作しています。このように、全く同じ結果を得る仕事を行うのに、モバイル型ロボットの方が約 2 倍高速に動作したのは、次の二つの理由によります。第一に、モバイル型ではロボットを送信するというオーバヘッドが増加しますが、圧縮効果により、送信するデータ転送量を削減できます（より大きな HTML ファイルを収集させるとその効果は顕著になります)。第二に、Web サーバからの情報収集を、低速な日本-スウェーデン間のインターネット回線を使って行うのでなく、モバイル Web ロボットを送って、相手先で高速なローカル通信を行わせるため、収集作業を高速に行えます。

■ 今後の展開

モバイルオブジェクト技術は今後の地球規模広域分散システムにおいて、重要な要素技術の一つであると考えられます。今後予想される研究の展開として、次のような課題が考えられます。

高性能化: PLANET はネイティブコード実行に基づいているため、高速なモバイルオブジェクト実行が可能です。また、リモートメモリマップ技術を用いることにより、計算に必要なオブジェク

トの部分のみを実行時に検出して転送する機構を有しています。さらにこの機構には圧縮技術が組み込まれ、リポジトリに格納する際のディスク容量と通信量を軽減化しています。しかしさらにいくつかの高性能化の余地が残っています。第一に、プリフェッチ技術を組み込み、ストリーム型のデータ転送と統合することにより、デマンドドリブン型のリモートメモリマップ機構をさらに高速化できます。第二に、異機種オブジェクトモビリティの実現において、今回開発したオブジェクト表現の双方向変換技術では、データ領域の実行時解析を容易とするためにコンパイラの aggressive な最適化を行えません。そのような最適化を可能としつつ、かつ、実行時解析が可能となる技術を開発することにより、さらに高速なネイティブコード実行が可能になります。

ワールドワイド OS: 今回の研究で、異機種オブジェクトモビリティを正準表現を固定せずに実現する方式の開発を行いました。この方向性をさらに押し進め、OS が提供する API の相違を吸収するレイヤを実現すると、各サイト上にはローカル OS を走らせながら、その上でオブジェクトを自在に行き来させるワールドワイド OS を実現できる可能性があります。

セキュリティ: 今回の研究では、オブジェクトの実行の際、システムコール呼び出しを検出して、計算機資源へのアクセス制御を行いましたが、セキュリティ関連では、取り組むべき沢山の研究課題があります。第一に、検査を行うためのセキュリティポリシー記述をより高水準に行う方法の開発です。第二に、資源へのアクセス制御のみならず、消費制御を行う方法の開発です。第三に、計算機資源側のみならず、モバイルオブジェクト側のセキュリティを保全する技術の開発です。

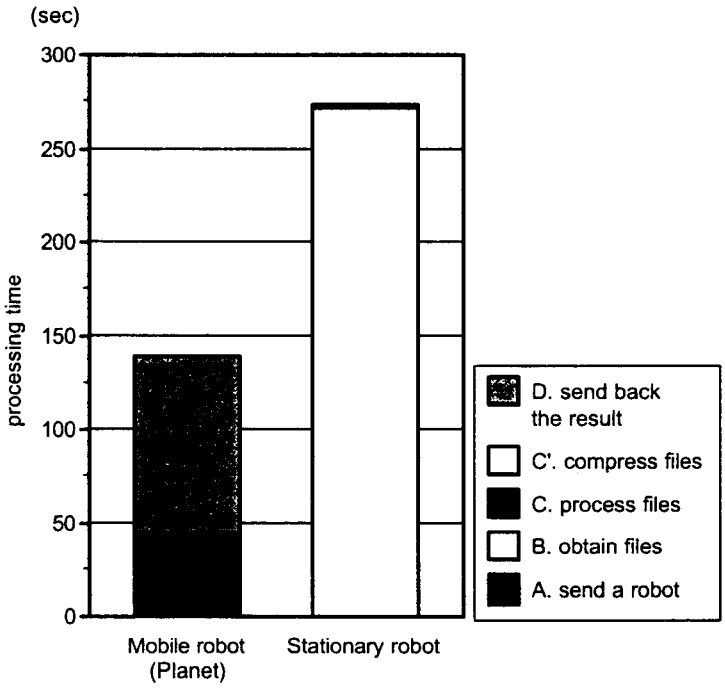

図 6: モバイル Web サーチロボットと静止型 Web サーチロボットの性能比較。筑波大学とスウェーデン国ウプサラ大学の間で 3MB 分の HTML ファイルを転送するのに要した時間（秒）を計測。モバイルロボットは PLANET を用いて実装され、PLANET の自動圧縮機能が用いられている。

アプリケーション: 今後さらに、モバイルオブジェクトの新たなアプリケーション領域を開拓していくことが重要です。有望なアプリケーションの例として、モバイル端末機器を用いた情報環境との融合、地球規模ハイパフォーマンス・コンピューティング、あるいは知的な処理を人間の代理となって広域ネットワーク上で行うエージェント・コンピューティング等が挙げられます。

■ 成果リスト

1. K. Kato, Y. Somcya, K. Matsubara, K. Toumura, H. Abe, An Approach to Mobile Software Robots for the WWW, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 526 548, 1999 July.
2. K. Kato, K. Matsubara, Y. Somcya, K. Itabashi, and Y. Moriyama, Planet: An Open Mobile Object System for Open Network, *Proceedings of IEEE Joint Symposium of First Int. Symposium on Agent Systems and Applications/Third Int. Symposium on Mobile Agents*, pp. 274 275, 1999 October.
3. K. Kato, K. Matsubara, Y. S., K. Itabashi, and Y. Moriyama, Design of an open mobile object system for open networks.*Parallel and Distributed Computing for Symbolic and Irregular Applications: Proceedings of the International Workshop PDSIA '99*, pp. 323 341.World Scientific, 2000.
4. 阿部洋丈, 一杉裕志, 加藤和彦, ソースコード変換技術を用いた Java 言語におけるスレッドのモビリティの実現法, 情報処理学会論文誌 : プログラミング, Vol. 41, SIG2(PRO 6), pp. 29–40, 2000 年 3 月.
5. 松原克弥, 板橋一正, 森山豊, 染谷祐一, 加藤和彦, 関口龍郎, 米澤明憲. 動的双方向変換技術に基づいた異機種オブジェクトモビリティの実現法, 情報処理学会論文誌, Vol. 41, No. 6, pp. 1651–1664, 2000 年 6 月.
6. K. Kato, K. Matsubara, Y. Someya, and M. Yoshida. Mobile substrate: Experiences of middleware-layer object mobility. *Proceedings of 6th ECOOP Workshop on Mobile Object Systems: Operating System Support, Security and Programming Languages*, 15 pages, June 2000.
7. K. Matsubara, K. Kato, T. Maekawa, S. Maeyama, S. Yuta, and A. Haradao, Asynchronous Robot Teleoperation via the Internet: Mobile Agent Approach, *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, July 2000.
8. 吉田雅年, 松原克弥, 加藤和彦. モーバイル・メモリセグメントを用いたモーバイルプログラミング言語の非抽出型実現方式, 情報処理学会論文誌 : プログラミング, 2000. 掲載予定.

Towards New-Generation Network Software

Christian Tschudin

Department of Computer Systems
Uppsala University, Sweden

1 Introduction

Mobile code is a major change in the way we look at computers: As it turns out, computer science has been built on the implicit assumption that a program and the computer that executes it are inseparable. Changing to a new model of program execution where a complete program or parts of it can "move" to another computer requires a re-thinking of current practice (e.g., protocol design), it questions and shows limits of current approaches (e.g., in the area of security), in general it represents a formidable research challenge to computer scientists.

This report comments on the work done by Kazuhiko Kato from the University of Tsukuba. His main research focus has been on middleware i.e., the software layer between operating systems and applications that provides the right network abstraction for distributed applications. He has adopted a mobile code approach for this, thus is working in the field of what is now known as "active middleware".

In active middleware systems, the key advantage of mobile code becomes nicely visible: Late binding means that we postpone decisions to the best possible moment (i.e. run time) when enough information to make such a decision is available. Typically, this aims at either the location where migration is decided based on the task to solve and the current network and system load, or functionality where new code is linked in on the fly. Both aspects are covered in the middleware system "PLANET" built by Kazuhiko Kato and his team. We first comment on the PLANET system before looking at a test and measuring application for WEB searching that was built on top of PLANET.

2 PLANET

The attractive property of PLANET is that it attacks code mobility at the native code level and, this is important for middleware, also addresses heterogeneity. This is in contrast to most other mobile code approaches either based on JAVA, Tcl or another language that build "total mobile code architectures" which limit themselves to their own small world.

In choosing the middleware layer (instead of the application layer), Kazuhiko Kato's system can benefit and extend operating system specific mechanisms, most notably the management of memory. His concept of "mobile memory segments" shows this nicely: From a conceptual point of view it inherits from distributed operating systems research (e.g., MACH) and the performance gain by working at this level, while at the same time offering the essential shared memory high level abstraction to application. His method of lazy on-demand-migration of memory pages, linked with the possibility to redirect them under the control of mobile code, is a convincing case against the usual full transfer approach so often seen in mobile agent systems.

We also point to PLANET's innovative ways of organizing mobile software, namely the use of mobile (!) compilers for mobile code. The basic idea is to provide support for heterogeneous execution environments for mobile code by shipping code translators. This code morphing, we believe, will become an important technique in the future to eliminate the inherent inefficiency of code interpretation. What is promising (also for middleware research) is that this is a development that is paralleled at the hardware level e.g., in Transmeta's Crusoe chip that does code morphing on the fly and at CPU clock speed. PLANET is in a good position to benefit from such developments and already has started to provide the respective abstraction to mobile code developers.

3 "Mobilizing" a WEB Robot

Kazuhiko Kato's other research work we want to comment on is the "mobile web-search robot" which serves as a proof-of-concept for PLANET. The goal was to benefit from the possible parallelization of WEB searching, as well as locality in the Internet, by writing a mobile code based application: The search applications splits itself into several sub-applications that migrate to remote places from where they search a subset of the Internet before returning with their result.

The mobile web robot application shows the elegance that PLANET provides for moving parts of an application to remote locations. At least as important is the fact that PLANET is extensively and carefully compared to other approaches, stationary as well as mobile code based ones. Such a measuring and comparison activity is in fact an essential aspect of any systems research. PLANET performs really well and makes a convincing case in favor of a distributed search approach. Here too, we believe that this work is very well in line with current research efforts, for example efforts to build distributed measuring platforms to "x-ray" the Internet.

4 Summary

The PLANET system, which successfully undertook tedious work in the low levels of operating systems and network code as well as confronted itself with its own run-time behavior, is a convincing and pragmatic piece of systems work. It has a sound conceptual basis and contributes new insights in essential abstractions for building mobile code based applications. The early choice of working with native code is a strategically good one, as many language-centric mobile code efforts will see themselves limited in their scope, while approaches like PLANET already have the experience of how to use mobile code as the "glue" not only inside distributed applications, but also across operating system, middelware and application layer.