社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－IOT－1（2）
2008/5/8

分散処理のためのEspace言語の開発

岩川 建彦　小野 智司　中山 茂

鹿児島大学工学部　〒890-0065 鹿児島県鹿児島市郡元 1-21-40

E-mail: {sc100007, ono, shignaka}@ics.kagoshima-u.ac.jp

あらまし 本研究では分散オブジェクトの取り扱いとメソッドの分散実行を言語文法に取り入れることで分散アプリケーションの記述に特化したJava拡張言語Espaceを開発した．Espaceの計算モデルはLindaモデルに準拠しており，様々な計算に対応できる．プログラマはEspaceを用いることで分散アプリケーションの記述を省力化することが可能である．

キーワード 分散処理，Java，コンパイラ，オブジェクト共有空間，Lindaモデル

A Development of Espace: The Programming Language for Distributed Computing

Takehiko IWAKAWA　Satoshi ONO and Shigeru NAKAYAMA

Faculty of Engineering, Kagoshima University　1-21-40 Korimoto, Kagoshima-shi, Kagoshima, 890-0065 Japan

E-mail: {sc100007, ono, shignaka}@ics.kagoshima-u.ac.jp

Abstract We have developed programming language Espace suited to distributed application and based on Java. Espace contains simple syntax for easy use of distributed objects and for distributed parallel method invocations. The computing model of Espace conforms to Linda model, and it can be adjusted to various kinds of computations. Programmers can write codes of distributed applications with reduced cost by using Espace.

Keyword Distributed Computing, Java, Compiler, Object Shared Space, Linda Model

1. はじめに

コンピュータネットワーク環境の発達によって，分散ソフトウェアは普遍的な存在となっている．グループウエアなどのビジネスツールやメッセンジャなどのコミュニケーションツール，あるいはゲームやエンタテインメントにいたるまで，枚挙に暇がない．このような状況の中にあって，プログラマが分散ソフトウェアを記述する機会は今後さらに増加していくと考えられる．一方，分散ソフトウェアを開発するに当たっては，プログラマは通信手順の策定やネットワーク管理，スレッド制御などを強いられるなど，難度が高く煩雑な設計やプログラミングが要求される．

本研究では，通信処理を抽象化し，分散ソフトウェアの開発の省力化を目的とした高級プログラミング言語の仕様を設計し，そのコンパイラと実行環境の実装を行う．実用的なプログラミング言語とし習得を容易にするために，言語仕様はJavaの拡張言語とし，Java用のライブラリが利用できるようにした．文法の拡張は，Javaをベースに容易に分散並列処理による負荷分散を実現できる分散タスク構文，分散オブジェクトを容易に取り扱える分散オブジェクト構文を追加するなど，わずかな変更にとどめており，学習を容易にすると同時に，オブジェクト指向言語の利点を活かせるように配慮した．

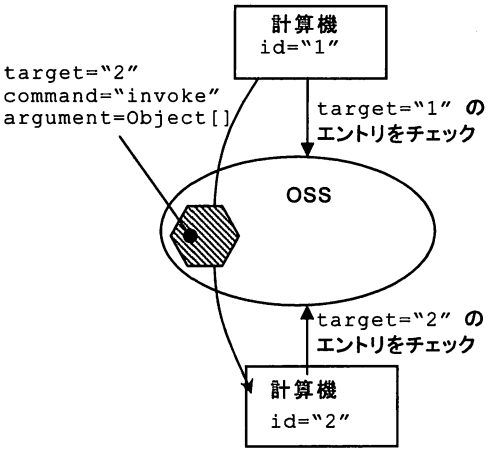
また，分散実行環境の構築を容易にできるようにした．ユーザは分散ユニットとよぶプログラムを起動すれば即座に分散実行環境の構築が可能であり，構築の煩雑さを理由として応用アプリケーションの幅を狭めることはない．

Espaceを利用することで，分散アプリケーション開発を大幅に省力化でき，プログラマはアイデアを即座に形にすることができる[1,2].

2. Espaceの開発

2.1. 分散実行環境の通信手順

分散実行環境はオブジェクト共有空間サーバ（以下



1 OSSを利用したメッセージパッシング

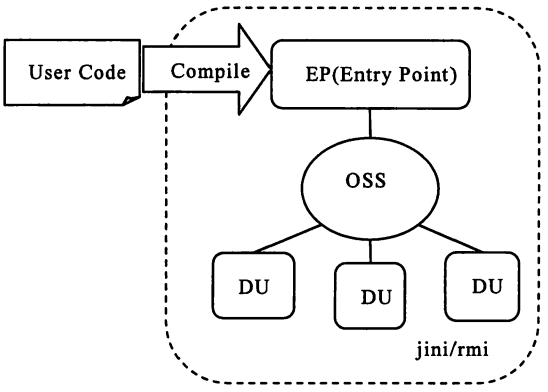


図 2 Espace システム構成

OSS サーバ）を中心として構成しており，OSS サーバ以外のすべての計算機は OSS サーバに接続をする．OSS サーバはオブジェクトを書き込んだり，取り出したりすることのできるサーバであり，オブジェクトのフィールド値を条件にして，条件に従うオブジェクトのみを選択して取得することができる．このとき，読み書きするオブジェクトをエントリとよぶ．Espace で使用するエントリは「宛先」フィールドを持つ．それぞれの計算機は固有の ID をはじめとする特定の識別子を持っており，OSS に自分のもつ識別子が「宛先」となっているエントリが書き込まれるのを待っている．一方，メッセージを送る側は，送りたい相手の識別子を「宛先」フィールドに入れたエントリを OSS に対して書き込むことにより，目的の相手にメッセージの送信が可能となる．模式図を図 1 に示す．なお，識別子はすべてユニークである必要はなく，なんらかのグループを表現するなど，複数の計算機が共通の識別子をもつことも，ひとつの計算機が複数の識別子を持つことも可能である．

Espace が用いるメッセージングプロトコルには以下の 3 種類を定義した．

1.標準メッセージ
2.広範メッセージ
3.状態メッセージ

標準メッセージは，あるひとつの計算機に対してメッセージを送るためのプロトコルであり，エントリはその計算機にメッセージが受け取られた時点で消滅する．広範メッセージは，対応する宛先の識別子をもつ全ての計算機に対してメッセージを送るためのプロトコルであり，全ての対象の計算機にメッセージが受け取られるまでエントリが残される．状態メッセージは上記二つのメッセージとは異なり，特定の宛先を持たない．状態メッセージには名前がついており，名前を指定して OSS よりエントリを取り出し，情報を参照したり，上書きしたりすることができる．

Espace での通信は，すべてこの 3 種類のいずれかに属している．OSS の実現には JavaSpaces サーバを使用した．

2.2. システムの概要

Espace のシステムは OSS サーバと OSS クライアントの二種類から構成される．より詳細には図 2 に示すように，OSS，EP，DU の三種類からなる．

EP とはエントリ・ポイントの略で，ユーザが Espace を用いて記述したソースコードをコンパイルすることで得る実行プログラムである．ユーザが記述するソースコードは Espace システム全体を対象としているが，生成される実行プログラムはシステムの起点として機能するため，このように呼ぶ．EP は起動後，自動的に OSS に接続し，必要に応じて OSS にメッセージを書き込む．

DU はユーザコードによらず，すべての Espace システムで利用する分散ユニット（Decentralized Unit）である．DU は起動後，自動的に OSS に接続し，待機状態となる．EP が起動した後，必要に応じて EP からクラス定義などをロードし，システムの一部となって稼動する．

OSS の実装である JavaSpaces は Jini のサービスとして実装されている．Espace の構成要素である OSS サーバと OSS クライアント（EP と DU）は，Jini のサービス検索機能を使用して接続している．したがって同一

```
public class FrameAllocator implements
WindowListener{
    remote Frame myFrame;
    ...
    public void someMethod(){
      ...
      myFrame.setVisible(true);
      myFrame.addWindowListener(this);
      System.out.println(myFrame.getTitle())
    ;
    System.out.
      println(myFrame.getClass().getName());
  ...
```

図 3 remote フィールドの使用例

LAN 内においては，ユーザは少なくとも接続先の IP アドレスなどの指定をすることなく分散実行環境の構成が行えるため，ネットワーク管理の手間が省ける．

2.3. コンパイラの構成

Espace コンパイラは，入力コードの構文をチェックし，中間コードを生成する構文解析部，変数やクラスなどの名前を解決し，場合によってはエラーを検出する意味解析部，コードの編集を行う翻訳部，中間コードを Java ソースコードに変換するコード生成部からなる．

構文解析部の作成にはコンパイラ・コンパイラ JavaCC を利用した．

2.4. 分散オブジェクト構文

分散オブジェクト構文は，以下の文法からなる．

1. remote フィールド属性
2. role 型
3. 拡張 new 演算子

remote フィールド属性は static フィールド属性に似ている．static 属性をもつフィールドは，あるクラスのすべてのインスタンスで共有される．一方，remote フィールド属性はシステムに参加するすべての計算機で共有される．DU のコードをユーザが書くことはないため，すべての EP で共有されると言い換えてもいい．また，純粋に文法だけをみると static 属性とまったく同様の変数スコープを持つことになる．remote 属性の変数を remote 変数と呼ぶ．

remote 変数に対するメソッドアクセスやオブジェクト割り当ては，Espace コンパイラによって OSS 操作へと変換される．そのために意味解析で得た情報などを利用している．remote フィールドの使用例を図 3 に示す．

図 3 の例の場合，意味解析部において，変数 myFrame が Frame 型であり，remote 属性を持つことがわかる．そして，それぞれのメソッドアクセスのメンバ名や引数リストを構文解析情報から得て，引数リストの型を意味解析によって得る．さらに，そのメソッドアクセスの戻り値が利用されているかどうかを構文解析情報から得ることができる．

Espace コンパイラはこれらの情報を総合して，OSS に書き込む命令を決定する．現時点では，メンバ名，引数リスト，引数型がプリミティブかどうかの情報，および戻り値を返すべきか否か，といった情報を含む命令を使用している．また，myFrame.getClass().getName()のように連続してメソッド呼び出しを行う場合には，getClass, getName のようなメンバアクセス演算子(.)で区切られたそれぞれのメソッド呼び出しについて，引数リストや引数型などの実行に必要な情報を取得し，ひとつの命令としてまとめて発行する．

後述する拡張 new 演算子を使用せず，remote フィールドに値を代入した場合は，その値を分散オブジェクトとして，EP の分散オブジェクトホルダと呼ぶ部分に登録する．分散オブジェクトホルダは，OSS に登録されたオブジェクトに対する命令が書き込まれるのを監視しており，命令が書き込まれた際には OSS より取り出し，実行する．この分散オブジェクトには他の EP からアクセスすることが可能であるため，アプリケーション同士の通信に使用することができる．模式図を図 4に示し，図 4に対応する Espace ソースコードを図 5に示す．

Espace では拡張 new 演算子を用意し，分散オブジェクトをリモート計算機に割り当てることを可能とした．また，拡張 new 演算子と併用し，オブジェクトの割当先を決定するために，新たに role 型変数を導入した．role 型変数は通常のプリミティブ型と同様に宣言し，DU の役割を示すために使用する．それぞれの DU の役割は，必要があれば EP を起動する前に割り当てる．

拡張 new 演算子の文法を以下に示す．

'new' '<' RoleVariable '>' *ClassName*
'(' (Parameter)* ')'

ここで，RoleVariable は role 型変数へのアクセスである．ClassName は割り当てるオブジェクトの型を示す．Parameter は引数である．事前に宣言した role 型変

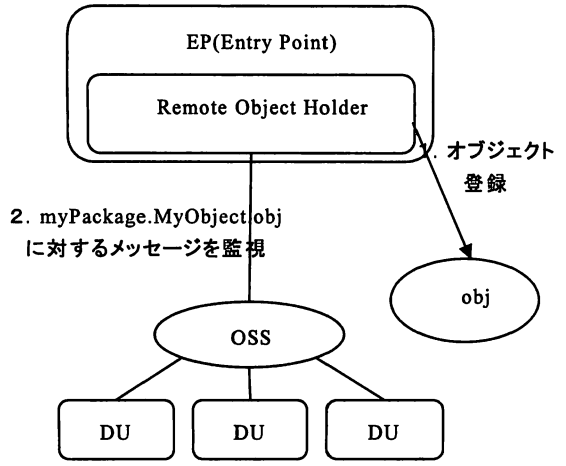


図 4 通常の代入文を使用した場合の remote フィールドの状態

```
package myPackage;
class MyObject{
  remote MyObject obj;
  ...
  obj = new MyObject();
```

図 5 図 4に対応する Espace ソースコード

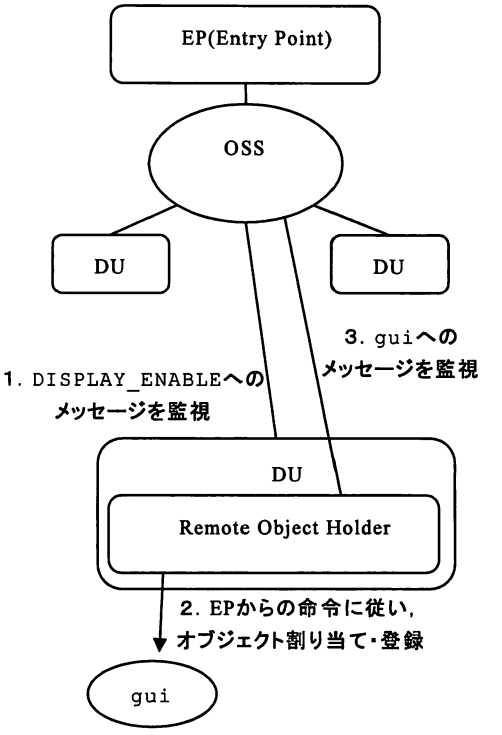


図 6 role 型変数と分散オブジェクト

数を与えることにより，分散オブジェクトの割り当ての際には，割当先の DU の役割を指定することができる．

たとえば，システム全体に 1 台だけプロジェクターに接続された DU が存在するとき，GUI をその DU で表示することを考える．ユーザは，まず“画面表示を行う“という意味合いの role 型変数 DISPLAY_ENABLE を宣言する．

```
role DISPLAY_ENABLE;
```

GUI オブジェクトの割り当て時には，

```
remote MyFrame gui =
     new<DISPLAY_ENABLE> MyFrame();
```

とすることで，DISPLAY_ENABLE の役割を設定された DU において，MyFrame オブジェクトが割り当てられる．模式図をに示す．

role 型変数を宣言した場合は，Espace コンパイラは Java ソースコードのほかに，role 設定ファイルを生成する．このとき生成された role 設定ファイルは次節で説明する DU 管理ツールに読み込まれ，role の設定に利用される．

remote フィールドに対する命令の実行時には，分散オブジェクトの型から実行するメソッドの正確な引数型リストを取得する．引数型にインタフェース型が含まれる場合，はじめに引数オブジェクトそのものを分散オブジェクトホルダに登録しておき，remote フィールドには引数オブジェクトそのものを送るかわりに，対応する引数インタフェースを動的に実装したプロクシインタフェース・オブジェクトとよぶオブジェクトを送る．プロクシインタフェース・オブジェクトのメンバを呼び出した際には，OSS を介して分散オブジェクトとしてホルダに登録された引数オブジェクトに対する命令を書き込むようになっている．つまり，インタフェース型の引数を remote フィールドのメンバに渡す場合，引数自体も通常の remote フィールドと同様に扱われて，分散オブジェクト同士が通信するような構造となる．ただし，分散オブジェクト化された引数オブジェクトは通常の分散オブジェクトのようなスコープは持たないため，当事者（送信側，受信側）以外

のプログラムからのアクセスはできない.

図 5 においては, myFrame.setWindowListener(this) における this がプロクシインタフェースとして扱われる. 意味解析により, myFrame が Frame 型であることが判明する. また, メソッド setWindowLister は引数としてインタフェース WindowListener をとることも解析で判明する. そこで, 引数で渡した this は, FrameAllocator 型としてではなく, WindowListener 型として利用されることがわかるため, プロクシインタフェース発行の対象となる.

2.5. 分散タスク構文

分散タスク構文とは負荷分散を容易に実現するための文法である. 本文法は本研究開始以前から実装を進めていた部分である.

分散タスク構文を利用すると, 複数回におよぶメソッド呼び出しを実際には複数の DU で並列に実行することにより, 負荷分散性能を得ることができる.

ユーザは, 並列実行したいメソッドに espace 属性とよぶメソッド属性を設定する. espace 属性を持つメソッドを Espace メソッドと呼ぶ. その後, distribute 文と呼ぶブロック文で分散並列処理の実行箇所を指定する. distribute 文の文法を以下に示す.

```
'distribute' '{'
        (BLOCK_STATEMENT)*
'}'
```

ここで, BLOCK_STATEMENT とは, ブロック文の内部で宣言できる文である. distribute 文の内部における Espace メソッドアクセスは並列に実行される. distribute 文と espace 属性の利用例を図 10 に示す.

Espace コンパイラは distribute 文を検出すると, distribute 文内部のメソッドアクセスすべてをチェックし, Espace メソッドへのアクセスであるかを調べる. ひとつの distribute 文はひとつの内部クラス宣言 (DB 内部クラスと呼ぶ)に変換され, 当該内部クラスには, Espace メソッドへの引数リストを取得するメソッド (割り当てフェーズ) と, すべての計算が終了した後に, 計算結果を使って distribute 文を実際に実行するメソッド(実行フェーズ)が定義される. つまり, distribute 文に記述した内容は Espace メソッドへのアクセスに関しては Espace メソッドへの入力(引数渡し), Espace メソッドからの出力 (戻り値の取得) という 2 つの内容に分けられ, それぞれ一回ずつ実行される.

割り当てフェーズと実行フェーズの処理の同一性を保持するため, 割り当てフェーズを実行する前に, アクセスする変数すべてを直列化, ないしクローン処理することにより保存し, 実行フェーズの実行時にそれらの変数をロードする.

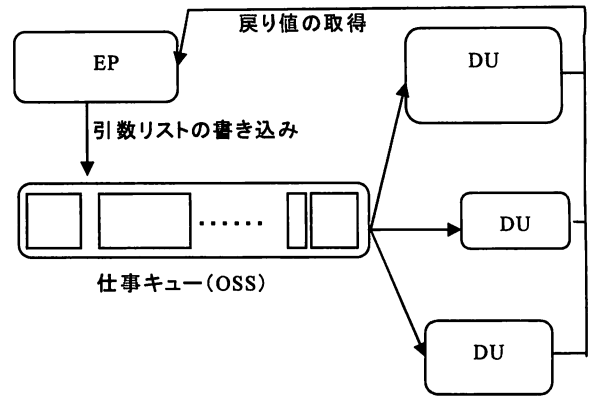


図 7 Espace と Linda モデル

一方, DU 側で実行するための Espace メソッドの定義は, Espace コンパイラによってワーカとよばれるクラスに定義される. ワーカは distribute 文の実行前に, 動的に OSS 経由でロードされる. Espace メソッド内で他の Espace メソッド以外のメソッドを呼び出した場合は, EP 側に引数リストを含むメソッドの実行命令を送信し, EP が戻り値を戻すのを待つ.

distribute 文を実行すると, まずは DB 内部クラスの割り当てフェーズを実行し, 引数バッファに Espace メソッドにあたえるべきすべての引数をためこむ. 次に引数リストを含むメソッドの実行命令を DU に宛てて発行する. DU から計算結果が返されるのを待ち, すべての計算結果が収集された段階で, DB 内部クラスの実行フェーズを実行する.

2.6. Linda モデルと付加機能

この節では, 分散タスク構文の実行時の動作について詳細を述べる. 分散タスク構文実行時には, EP は DU 数を上回る数の命令が OSS に残るように調節し, OSS をタスクのキューとして利用している. この方法は Linda モデル[3]と呼ばれる分散並列処理モデルに従っている. Linda モデルを Espace に適用した際の模式図を図 7に示す.

Linda モデルでは OSS を仕事キューとして使用する. DU それぞれの計算性能が異なっていても, あるいはある程度仕事の粒度にばらつきがあったとしても, それぞれが仕事キューから仕事を取得して実行する作業を繰り返すだけであるため, 仕事それぞれの粒度が十分に細かければ, 結果として仕事が適切に配分される.

ただし, 逆に仕事それぞれの粒度が細かすぎると,

通信セッションの増加がオーバーヘッドとなり，実行性能が低下する．このような性能低下を避けるため，Espaceの分散実行環境システムには計算粒度の自動選定機能を組み込んだ．DU は戻り値を戻す際，計算に要した時間の情報もエントリに含ませる．EP はこの計算時間を観察し，ひとつの命令が十分な計算時間をかけて実行されるよう，命令ごとのメソッド実行数を動的に調節する．

また，計算途中の DU が停止するなどの予期せぬ事態にも全体の計算結果を得ることができるように，命令の発行から一定時間を経ても結果の得られない仕事命令については，再度発行することで結果を得ようとする．

2.7. DU 起動手順

DU の起動には Java Web Start を利用している．本節では，Java Web Start を利用した DU 起動手順を示す．

まず，ユーザは「DU 起動用ウエブサイト」にアクセスする．起動用ウエブサイトには DU の起動を設定された jnlp ファイルへのリンクがある．ユーザがこの jnlp ファイルへのリンクをクリックすると，Java Web Start のローダが起動し，DU が起動する．Java Web Start を使用しない場合は，ここまでの処理を通常のアプリケーションと同様，クラスファイルをダウンロードし，java コマンドを使用して起動することとなる．DU ははじめに Jini テクノロジを利用して OSS を検索し，接続する．

3. おわりに

Espaceは分散オブジェクト構文と分散タスク構文を備え，分散ソフトウェアを容易に記述することができる．分散オブジェクト構文によって分散オブジェクトを通常のオブジェクトのように使用することができ，特に複数の分散オブジェクトがそれぞれ通信するようなアプリケーションでは非常に強力な記述力を得ることができる．また，分散タスク構文によって容易に負荷分散が可能である．

Espace の分散実行環境は Linda モデルに従うためにさまざまな計算資源を有効に活用でき，設定次第でネットワークを使用しない逐次実行型システムに変更することもできるため，柔軟なシステム運用が可能である．ユーザプログラム以外の，システムを構成する要素はすべての Espace システムで共用のものを使用でき，プログラム変更のたびにコードを配信しなおす必要はない．また，システムの起動そのものも容易であるため機動的な運用が可能となり，分散アプリケーションへ特化しながらも，応用アプリケーションの幅を狭めることはない．

近年，携帯電話などの機器でも Java が利用できることなどから，携帯電話を用いた分散計算の研究も行われている[4]．今後は Espace を携帯電話等の機器に対応させることで，応用の幅を広める予定である。

謝辞

本研究の一部は IPA 未踏ソフトウェア創造事業（未踏ユース）の助成を受けました．ここに記して感謝の意を表します．

文　献

[1] 原崇，飯村伊智郎，武田和大，鶴沢偉伸，中山茂: インターネット・ユーザ参加型の分散並列処理のための Espace 言語の開発とその応用; 情報文化学会論文誌, Vol.10, No.1,pp.11–18 (2003)

[2] 岩川建彦，小野智司，中山茂: "分散並列処理プログラミング言語 Espace の開発", システム制御情報学会論文誌, Vol.19, No.7, pp.296-298 (2006).

[3] D. Gelernter: Generative communication in Linda;ACM Trans. Program. Lang. Syst., pp.80–112 (1985).

[4] 小野智司，片山公博，中山茂: "携帯電話上のプライバシを利用した分散組合せ最適化", 電気学会論文誌 C, Vol.127, No.5, pp.793-798