分散アプリケーションのためのプログラミング言語開発

― Linda モデルに基づく分散処理言語 Espace ―

１．背景

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おり、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ます。

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ます。

２．目的

本プロジェクトでは通信処理を抽象化し、分散ソフトウェアの開発を容易にすることを目的とした高級プログラミング言語 Espace（エスパース）を開発することを目的とします。

通信処理に関わる処理を言語文法に組み込み、抽象化するために、分散プロセスや分散オブジェクトからなる分散環境をあたかも一つの計算機であるかのように見なしてコーディングできる言語文法、および仕様を規定します。

３．開発の内容

３．１　実行環境の概要

Espace の実行環境はオブジェクト共有空間（以下、OSS）、システムの部品となる分散ユニット（以下、DU）、そしてプログラムの起動を行うエントリポイント（以下、EP）から構成されます。OSS はオブジェクトを書き込んだり、オブジェクトのフィールド値を条件にし、一致するオブジェクトを取り出したりすることのできるサーバであり、OSS には Java での実装である JavaSpace を使用しています。Espace のシステム構成の模式図を図 1 に示します。

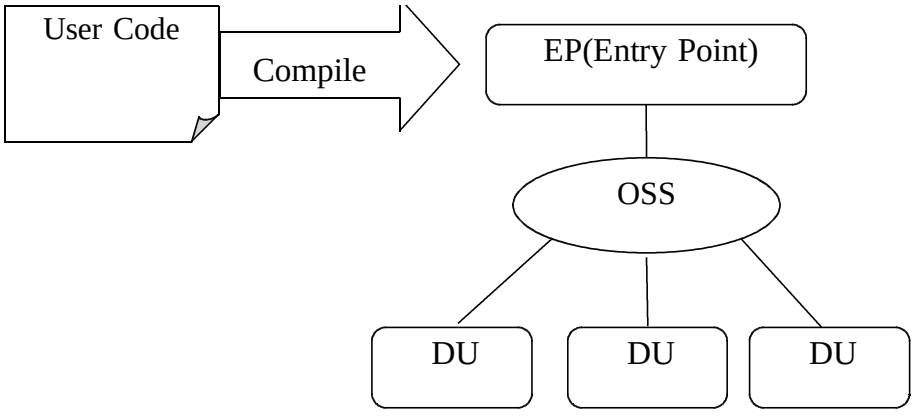

図１　Espace のシステム構成

３．２　Espace コンパイラの作成

本プロジェクトで開発した Espace コンパイラはコンパイラ・コンパイラである JavaCC を利用して作成しました。Espace コンパイラは入力されたソースコードを構文解析・意味解析処理を経た後、OSS 操作を含む Java ソースコードに変換します。

３．３　言語文法（Java に対する拡張）

新たに定義した remote フィールド属性は static フィールド属性に少し似ています。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ますが、 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ます。 remote フィールド属性の利用を含むソースコード例を図 2 に示します。

```
public class FrameAllocator implements WindowListener{
  role DISPLAY_ENABLE;
  remote Frame myFrame;
  ...
  public void someMethod(){
    ...
    myFrame = new<DISPLAY_ENABLE> Frame();
    myFrame.setVisible(true);
    myFrame.addWindowListener(this);
    System.out.println(myFrame.getTitle());
    System.out.println(myFrame.getClass().getName());
    …
```

図 **2**　**remote** フィールドの使用例

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ました。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 型変数を導入しました。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します。それぞれの DU の役割は、必要があれば EP を開始する前に割り当てます。拡張 new 演算子の文法を以下のとおりです。

‘new’ ‘<’ *RoleVariable* ‘>’ *ClassName* ‘(‘ (*Parameter*)* ‘)’

ここで、RoleVariable は role 型変数へのアクセス、ClassName は割り当てるオブジェクトの型、Parameter は引数です。

図 2 に示した例においては、役割"DISPLAY_ENEBLE"を EP 起動前に適切な DU に設定することで、該当の DU に myFrame オブジェクトが配置されます。

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ます。

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します。espace 属性を持つメソッドを Espace メソッドと呼びます。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します。distribute 文の文法を以下に示します。

```
'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す。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ます。distribute 文と espace 属性の利用例を図 3 に示します。

```
public class SimpleDPP{
  ...
  espace int kernel(int argument){
    ...
  }
  void method(){
    int sum=0;
    distribute{
      for(int i=0; i<MAX; i++){
        sum += kernel(i);
      }
    }
  }
}
```

図 3　分散タスク構文利用例

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ます。ワーカは distribute 文の実行前に、動的に OSS 経由でロードされます。したがって、プログラムに変更があった場合などでも、ユーザがインストール作業を再度行う必要はありません。

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ます。この方法は Linda モデルと呼ばれる分散並列処理モデルに従っています。Linda モデルを Espace に適用した際の模式図を図 4 に示します。

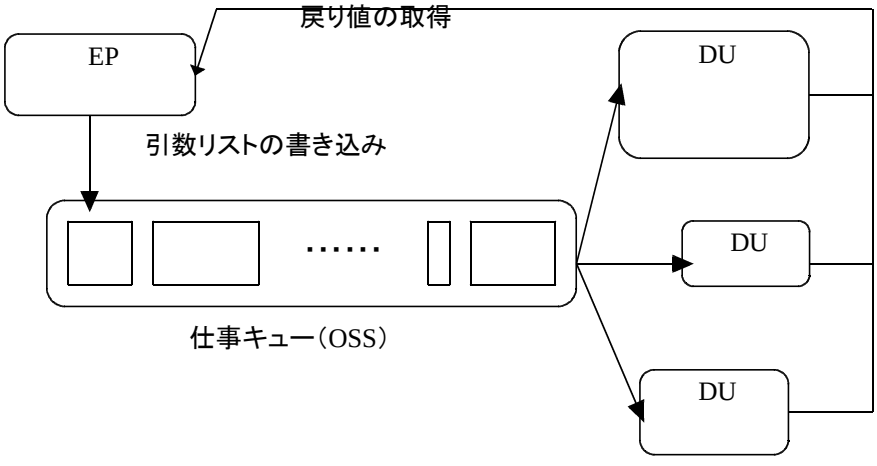

図 4　Espace と Linda モデル

Linda モデルでは OSS を仕事キューとして使用し、DU それぞれの計算性能や、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ます。

３．４　対応機種の拡張

開発期間の後期において、携帯電話などの機種で Espace を実行可能とするための実験や拡張を行いました。この多機種への対応が実現すれば、Espace を用いて様々な機種の利点を統合したシステムを容易に記述できるようになります。今後は調査や実装をさらに進め、多機種への対応を実現する予定です。

４．従来の技術（または機能）との相違

並列処理や分散並列処理の記述を省力化するために逐次実行プログラムと同様の記述を可能とする言語機能はすでに利用されており、一例としてC言語などで利用可能な OpenMP があります（もっとも、OpenMP そのものは**分散**並列処理用ではなく、並列処理に対応するものです。分散化するための技術が別にあります）。Espace の開発コンセプトはいかに容易に分散処理を実現できるかという点にあり、学習の容易さや実行環境構築の容易さを重視しており、既存の技術とはややスタンスが異なります。比較的分散処理プログラミングの経験の浅いプログラマや、アイデアを即座に形にしたいプログラマにとって、特に利用しやすくなっています。

５．期待される効果

Espace は分散処理に対する一連の抽象化により、分散実行環境のアップデートが行われた場合もソースコードが変更不要であるなど、保守性が高くなります。また文法は可能な限りコンパクトになるように、そして Java の言語文法ともなじむような設計をしています。Java のプログラマであれば Espace の導入は容易であり、ソフトウエア開発・運用の大幅な省力化が見込めます。また、分散実行環境の構築が非常に容易で、労力を軽減できるばかりでなく、機動的な運用が可能であるため、開発するアプリケーションの幅が広がります。

６．普及（または活用）の見通し

今後、コンパイラ、実行環境ともに十分なテストを行い、品質を高めた上でシステムを公開し、無料で利用できるようにする予定です。無料とすることで活用を促し、利用者からのフィードバックを得ることで、さらに品質を高め、魅力ある製品とすることで利用を促進します。なお、将来的にオープンソース化への要望が生まれた際には、本プロジェクト自体をオープンソースプロジェクトへ移行する予定です。

７．開発者名（所属）

岩川建彦（鹿児島大学大学院理工学研究科博士後期課程システム情報工学専攻）