データベースシステム 116－69
（1998. 7. 10）

型付高階モーバイル言語の設計

{大堀 淳， 加藤 岳臣， 橋本 政朋， 吉田 信明}
京都大学 数理解析研究所

概 要

値として扱うことが可能な**第一級の文脈**の概念を用いることにより，柔軟な分散計算を実現する上で必要なプログラムやデータなどの計算資源の移動，分散した計算資源の動的なリンク，さらにプログラムの遠隔実行等の機能を表現可能な高階型付きプログラミング言語の設計が可能である．この洞察に基づき，文脈を値として扱う計算系の理論を基礎とした新しいモーバイル計算のモデルを構築した．現在このモデルに基づく高水準モーバイル言語 COMBO の設計及び実装を計画している．本稿では，文脈計算に基づく高階モーバイル計算について概説した後，COMBO の設計及び実装上の諸問題を論じる．

Desiging a Typed Higher-Order Mobile Language

{Masatomo Hashimoto, Tekeomi Kato, Atsushi Ohori, Nobuaki Yoshida}
Research Institute for Mathematical Sciences, Kyoto University

Abstract

The notion of first-class contexts allows us to represent various advanced features required for flexible mobile and distributed programming, including data and code mobility, dynamic linking of distributed resources, and remote evaluation of a program. Based on this observation, we have developed a new model of mobile programming based on a typed context calculus, which is a typed lambda calculus enriched with first-class contexts, and are designing a higher-order mobile language, COMBO. This paper describes the context-based mobile computation model, and discusses various technical issues in the design and implementation of COMBO.

1 まえがき

近年のインターネットを利用した分散プログラミング技術の急速な普及に伴い，ネットワーク上の種々の資源の共有に関する新たな形態が生まれつつある．共有される資源は，テキストや画像等を含めた静的なデータにとどまらず，検索処理等の高度な計算機能そのものや，さらにそれらを実現するプログラムまでもが共有の対象となっている．これら資源の共有の有用性が広く認識され，ネットワーク上に多くの資源サーバーおよびそれらを利用するためのクライアントプログラムが実装されている．このような分散環境の変化にともない，データベースシステムは，従来のような均一データの集中管理システムにとどまらず，ネットワーク上に分散する種々のデータや処理機能を有機的に合成し利用するための機構を提供する，情報共有に関する制御ソフトウエアとしての役割を果たす必要がある．そのようなソフトウエア実現のためには，従来のデータベース技術に加えて，ネットワーク上の資源共有のためのプログラミング技術を開発する必要がある．このためには，従来のデータのみを受渡しする分散プログラミングに加え，プログラムコードやプログラムの実行をも分散可能にする分散計算モデルを確立する必要がある．

新たな分散計算モデルの必要性を理解するために，あるサイトAに共有大規模データベースがあり，それとは別のサイトBにデータ検索のための共有検索アルゴリズムライブラリがある場合を考えてみよう．これらサイトとは別のサイトにいるユーザが，これら分散資源を有効に利用するためには，Aサイトにあるデータを検索するプログラムを，Bサイトの検索アルゴリズムを利用して作成し，それをAサイトに送って検索を実行し，結果を得るといった処理が可能でなければならない．この例の場合，分散プログラムが交換するのは，単なるデータではなく，検索プログラムであり，そのプログラムも遠隔サイトのライブラリを利用て作成されたものである．さらに，単にデータとしてのプログラムコードを交換するのではなく，そのコードが送られた遠隔サイトにて，必要な資源とリンクし実行し，結果をユーザサイトに返す機能が必要とされている．この例から推測されるように，柔軟な分散プログラミングを実現するためには，

- プログラムやデータの移動
- 遠隔プログラム間の動的なリンク
- 遠隔実行

の機能を統合したプログラムモデルが必要となる．

もちろん，従来のRPC[1]等の技術を使い，これら処理を行なうプログラムを書くことは可能であるが，そのためには，ユーザは通信のための膨大な量の低レベルなコードを書かねばならず，繁雑であるばかりでなく，大規模なシステムのプログラムの信頼性を確保するのが困難である．近年，分散処理に関する新たな方式や言語がいくつか提案されている．これらの研究により，プログラムやオブジェクトの移動やメソッドの遠隔呼び出し[3]，コードの移動およびオブジェクトマイグレーション[4, 2, 7, 8]，遠隔実行[9]等の技術が開発されているが，これらの機能を，資源の動的束縛機構を含め系統的に統合した高水準分散プログラムモデルは未だ構築されていない．

分散資源共有のためのプログラミング言語を実現するためには，以上のような機能を包摂する分散処理のための新しい抽象化の概念を導入する必要がある．さらに，分散処理に内在する複雑さとそれに伴うデバッグの困難さを考えた場合，プログラムに含まれる不整合をコンパイル段階で検出する静的型検査はよりいっそう重要となると考えられる．また，上記の例から推測される通り，高度な分散資源の共有は，処理機能の分散共有という形態をとるため，その系統的なモデルのためには，高階の関数を自由に使用可能な体系が望まれる．従って，分散に関する新しい機構は，高階の型付言語に統合されることが強く望まれる．

以上の一般的な目標の下で，我々は，分散資源利用のための高階の型付き言語COMBOの設計とそのプロトタイプシステムの実装を目指している．その最大の特徴は，「プログラムの文脈」をデータとして扱う計算系を基本とした分散プログラミングの新しいモデルに基づいている点である．「プログラムの文脈」の概念は，プログラムが利用する種々の資源の動的リンクおよびプログラムの実行の概念を含んだ極めて強力で一般性を持つものである．この文脈の概念に，それが作成された場所の属性を付与することにより，遠隔プログラム間の動的なリンク，プログラムやデータの移動とその制御，および遠隔実行の概念を含んだ一般的で強力なモーバイル言語の機能を実現することが可能である．従来分散の対象となっている値や関数などは，文脈の特殊な場合である．しかも，文脈の理論はラムダ計算の型理論の中

で定式化されたものであるため，それを基礎とするモーバイル言語機能は高階型付言語にシームレスに統合することが可能である．本論文では，COMBOの基礎としての文脈計算に基づくモーバイル計算モデルの概要を説明したのち，COMBOを設計実装する上での技術的諸問題を論じ，この言語実現のために実装を開始したプロトタイプシステムの概要を報告する．以下，第2節では，COMBO言語の理論的中核である文脈計算系を概説し，それを用いて動的リンクを行なう機構およびそのための型システムを概説する．さらに，文脈を扱う機能をもつ言語を導入する．第3節では，第2節で述べた言語を分散の概念と統合するために必要な拡張の概要を論じる．第4節では，実際に言語を実装する上での技術的問題点を論じ，現在構築中のプロトタイプシステムの概要を述べる．

2 理論的基礎と核言語

分散環境においては，行なわせたい計算を部分に分割し，それらを幾つかの計算資源に分散させて実行し，それらの結果を統合し，最終的に目的とする結果を得る．このようなパラダイムに適合するプログラミング言語を構築するための基礎理論は，以下のような特質を表現できなくてはならない．

- 計算資源の分割
  行なわせたい計算を分散させるためにはその分割が必須である．従来の分散処理では，分割の対象は静的なデータのみであったが，前節で論じたような高度な分散計算を実現するためには，データのみならず，プログラムをも分割する必要がある．ここで要求される分割は，閉じた関数単位ではなく，より一般的に，本質的に自由変数を含むようなプログラムを単位とすることが可能でなくてはならない．

- 計算の結合と変数の動的束縛
  分割し分散されたプログラムとデータを用いて目的とする計算を遂行するためには，自由変数を含むプログラム単位をデータやさらに別のプログラム単位等の計算資源に結合する必要がある．プログラミング言語の観点からは，この機能は，自由変数の動的な束縛に相当する．

高階の関数を柔軟に扱うことができる高水準言語においては，一般に，プログラムは入れ子構造となっているので，計算の分割とは，ある部分と，それを取り囲む文脈（コンテクスト）との分割と考えることができる．このモデルにおいては，分割されたプログラムの結合は，プログラム部分を文脈に埋め込むことによって実現される．文脈はその言語に含まれる束縛子を当然含み得るので，文脈自体も束縛子としての機能を持つことになる．そこで，プログラム部分を文脈に埋め込むという結合操作は，プログラムの単なる結合のみならず，自由変数の動的束縛をも実現する操作である．コンパイル単位やモジュール等への分割およびそれに対応するリンク等は，この部分と文脈への分割と結合の特殊な場合とみなすことができる．そこで，文脈を操作する機能を含んだ計算系が実現できれば，分散処理に必要な計算の分割と動的束縛を含むプログラムの結合を表現可能となるはずである．

そのような特質をもつ計算モデルとして，**文脈計算** [6] がある．本節では，文脈計算を概観し，次に，それに基づいてCOMBO言語を実現する基礎としての（モーバイル機能を含まない）核言語の構文を導入する．

2.1 型付き文脈計算

文脈の概念は，穴のあいた式と表現できる．文脈の操作は，文脈の穴を他の式あるいは別の文脈で埋める操作である．文脈計算は，計算に関する一般的なモデルであるラムダ計算を文脈の概念で拡張したものである．この計算系は，以下のような式の表す計算を対象とする．

$$e ::= x \mid \lambda x.e \mid e\ e \mid X^{\nu} \mid \delta X.e \mid e \odot_{\nu} e$$

最初の三つの式は，変数，ラムダ抽象，関数適用であり，ラムダ計算の構成要素である．残りの三つが文脈を表現するための式である．X^{ν} は，穴を表す変数，$\delta X.e$ は，e の中の穴 X を抽象し文脈を生成する式，$e_1 \odot_{\nu} e_2$ は，e_1 で表される文脈の穴に式 e_2 を埋め込み，式 e_2 の表す計算を e_1 が表される文脈にて実行するため構文である．関数適用と異なり，この穴埋めにより，e_2 に含まれる自由変数が，e_1 の穴に埋め込まれることによって束縛され得る．穴の抽象と文脈適用における ν は後に導入される名前変換子である．この機構により，文脈の重要な機能であ

る変数の動的束縛が実現される．例えば，名前変換子の機能を取り敢えず無視すれば，

$$(\delta X.(\lambda x.\ X^{\nu_1})\ 3)\odot_{\nu_2}(x+4)$$

において，埋め込まれ実行されるべき式 $x+4$ の x の値は，この文脈の中での x の値である 3 に束縛される．ところが，文献 [6] で分析されている通り，この動的束縛機能を，プログラミング言語のモデルの中核であるラムダ計算にそのまま導入すると，ラムダ計算の基礎であるベータ簡約理論が矛盾をきたす．この矛盾をさけ，ラムダ計算に穴埋めの機構を導入するために，穴埋めの前後に於いて名前の対応を保持する名前変換子を，穴と穴埋めに導入する．例の ν_1 と ν_2 は，この名前変換子である．この機構により，穴埋めによる名前の動的束縛と，関数適用による名前の静的束縛を，矛盾なく統合することが可能となる．この名前変換子を含めると，上記の例は，より一般的に以下のような式ととして表現される．

$$(\delta X.(\lambda y.\ X^{\{y/z\}})\ 3)\odot_{\{z/x\}}(x+4)$$

この式の実行は，以下のように行なわれる．

$$\begin{aligned}&(\delta X.(\lambda y.\ X^{\{y/z\}})\ 3)\odot_{\{z/x\}}(x+4)\\ &\xrightarrow[\text{穴埋め}]{}\ (\lambda y.\ \{y/z\}(\{z/x\}(x+4))\ 3\\ &\xrightarrow[\text{名前変換}]{}\ (\lambda y.y+4)\ 3\end{aligned}$$

文脈計算に於いて，式の持ち得る型は以下のいずれかである．

$$\begin{array}{lll}\tau & ::= & b & \text{基底型}\\ & | & \tau_1 \to \tau_2 & \text{関数型}\\ & | & [\Gamma \triangleright \tau_1] \Rightarrow \tau_2 & \text{文脈型}\end{array}$$

但し，Γ は変数束縛環境を表し，$\{x_1:\tau_1,\ldots,x_n:\tau_n\}$ と書き表す．

最初の二つは，ラムダ計算がもつ通常の型である．最後の文脈型は文脈の構造を記述するために新たに導入された型であり，以下のような直観的意味を持つ．

> Γ で表される変数束縛環境の下で動作する型 τ_1 のプログラムが埋め込まれると，型 τ_2 として動作するプログラムを生成する文脈

文献 [6] では，以上の計算系に対する簡約理論および型理論を構築し，その基本的性質を証明している．

2.2 核言語

前節で概観したように，文脈計算の理論は，動的束縛機構を表現することができるが，ここでは，それに基づきプログラミング言語としての構文を導入する．

先ず，文脈の導入であるが，以下の宣言で行なう．

$$\textbf{context}\ x\ X\ =\ e$$

この宣言により，文脈 $\delta X.e$ が x という名前で保存される．

次に，文脈が提供する束縛を受け取る側のプログラム (コードと呼ぶ) であるが，自由変数を含む式で，文脈適用により束縛されているものに相当する．これを導入するために以下の構文を導入する．

$$\textbf{code}\ \{x_1,\ldots,x_n\}.e$$

ここで，$x_1,\ldots,x_n$ は，式 e に現れる自由変数のうち，動的束縛により束縛する変数を表す．この構文は，前節の理論を用いれば，$\_\odot_{\{x_1,\ldots,x_n\}}e$ と理解することができる．但し，名前変換子 $\{x/x\}$ は $\{x\}$ と書く．コードは，以下の構文で，文脈と組み合わせることにより実際に実行される．

$$\textbf{exec}\ e_1\ \textbf{with}\ e_2$$

ここで，e_1 はコード，e_2 は文脈である．これは e_2 が $\_\odot_{\{x_1,\ldots,x_n\}}e_2'$ と表されるとすれば，$e_1\odot_{\{x_1,\ldots,x_n\}}e_2'$ を表すと理解される．

3 モーバイル言語 COMBO

3.1 核言語へのモーバイル機能の導入

本節では，核言語に導入する分散モーバイル機能について論じる．

前節で論じたように，文脈は分散モーバイル計算を実現する上で必須のプログラムの分割，および結合とその際に必要な変数の動的束縛の機能を表現可能である．そこで，文脈型または文脈へ埋め込むコード型を持つデータを，分散モーバイルの対象となるデータ構造とする．すると，概念的には，我々が実現する分散モーバイル計算系は，文脈およびコードを互いに利用しながら，それぞれ前節で論じた核言語を実行する複数のサイトからなる計算システム系とみなすことでできる．この拡張を，各サイトの言語処理系から見れば，文脈およびコードは，言語の

その他のデータと異なり，それら資源が存在するサイトを同定する位置属性を持ったデータとなる．

以上の拡張を行なうためには，以下の機能が新たに必要となる．

- 大域的環境
  複数の核言語処理系をもつサイトが，これら資源を相互に利用し協調して計算を実行するためには，それぞれのサイトが持つ共有可能な資源に関する情報を保持する大域的環境$GlobalEnv$を共有する必要がある．$GlobalEnv$は，すべてのサイトに共通な資源の名前である資源の識別子から，資源のあるサイト情報と資源の型τの組への関数でモデル化可能である．各サイトは，この大域的環境から識別子を用いて資源の位置と型を検索し，また，新たに生成した分散資源を$GlobalEnv$に登録する．核言語は，識別子を表すは特別な抽象データ型keyおよびそれを用いて$GlobalEnv$を操作する機能を含むように拡張される．

- コードの実行におけるモビリティ属性の制御
  コード及びそれを実行する文脈は，その位置属性が示すサイトに存在する資源であり，従ってその利用のためには資源の移動が必要である．この移動には以下三つの可能性がある．
  1. コードを文脈のあるサイトに送り実行し，その結果を受けとる
  2. 文脈をコードのあるサイトに送り実行し，その結果を受けとる
  3. コードと文脈双方を取り寄せ，それぞれをローカルなデータとして実行する．

  コードやデータの移動に関する柔軟なプログラミングを実現するために，これら三つのモビリティ属性をサポートする．この機構は，従来のモーバイルシステムと違い，サイトを明示的に指定することなく計算資源の移動性をプログラム可能にするものであり，モビリティに関するより抽象的かつ高水準な記述機構を提供している．

3.2 COMBO の文法

以上の機能を表現するために，前節で定義した核言語の文法を以下の様に拡張する．

分散資源の大域的環境へ登録は以下の構文で行なう．

register e_1 **as** e_2

この文を実行すると，e_1で表される文脈もしくはコードが，e_2で表されるkey型データの値を識別子として，大域的環境に登録される．なお，as 以下を省略すると，一意の識別子が自動生成され，その識別子がこの式の実行結果となる．

大域的環境からの資源の検索は以下の構文によって行なう．

lookup e **withtype** τ

この式を評価すると，大域的環境からeが表すkey型の値を識別子として持つ資源を検索する．資源が存在し，その結果の型がプログラムが指定した型τと一致すれば，この式の評価は成功し，τ型の値となる．ここで，τは2節で論じた文脈型かコード型でなければならない．これらの型のデータは一般に遠隔サイトに存在する資源であるため，次節で詳述する通り，それが存在するサイト情報を含む一種のポインターとして実現される．なお，eが表すキーに対応する情報が大域的環境に存在しないか，または，型が異なる場合は，この式の評価は失敗し例外を発生する．

前節で定義したコードe_1の文脈e_2での実行を表す式 **exec** e_1 **with** e_2は，その実行をどこのサイトで行なうかに応じて以下の3種類の分割される．

1. **exec** e_1 **at** e_2
   文脈e_2の実体が存在するサイト上で実行される．すなわち，この式が評価されるとコードe_1の実体がe_2の実体のあるサイトに転送されて適用の計算がなされ，その結果が **exec** e_1 **at** e_2 式の評価を行なったサイトに返送される．
2. **exec** e_1 **via** e_2
   コードe_1が存在するサイト上で実行される．
3. **exec** e_1 **with** e_2
   この式を実行しているサイトにて実行される．すなわち，コードe_1及び文脈e_2が共にこの評価を行なったサイトに転送され実行される．

3.3 プログラム例

以上で導入した構文を用い，簡単なプログラム例を示す．先ず，あるサイトAで，型

```
type BibDBTy =
      [{...,
        search:string×FieldTy→BibTy set,
        print:BibTy set→unit,
        ...} ▷ α] ⇒ α
```

を持つデータベースが以下のように文脈として定義されているとする.

```
val BIB_DB = ⟨context⟩ :  BibDBTy
```

この登録は以下のように行なう.

```
register BIB_DB as BIB_DB_KEY;
```

次に, サイト B で, 型

```
type BibDBLibTy =
      [{...,
        search:string×FieldTy→BibTy set,
        ...} ▷ α] ⇒ α
```

を持つライブラリが以下のように定義されているとする.

```
val BIB_DB_LIB =
    ⟨context⟩ :  BibDBLibTy
```

ここで, このライブラリに於いては, 上のデータベースで提供されている search 関数よりも高速なバージョンが提供されているものとする. この登録も以下のように行なう.

```
register BIB_DB_LIB as BIB_DB_LIB_KEY;
```

これらの資源をさらに別のサイト C で利用する状況を考える. まず, 大域的環境から *key* を用い, データベースとライブラリを確保する.

```
val LIB =
  lookup BIB_DB_LIB_KEY
           withtype BibDBLibTy;

val DB =
  lookup BIB_DB_KEY
           withtype BibDBTy;
```

次に, 以下のようなプログラムを実行する.

```
exec
  code {print}.
    exec
      code {search}.
        print(search("context",TITLE))
    via LIB
at DB;
```

このプログラムでは, search は BIB_DB_LIB にあるものをサイト C で, print に関しては, BIB_DB にあるものをサイト A で実行することを意図している. このように, この言語においては, 動的束縛の有効範囲や実行位置を制御することが可能となっている.

4 COMBO 実装の方針

言語の処理系は一般に, 実行コードを生成するコンパイラと, そのコードを実行するための実行環境に分けることができる. 以下, そのそれぞれについて, 現在実装を開始したプロトタイプシステムの概観とその技術的課題について論じる.

4.1 コンパイラ

コンパイラのターゲットとしては, 実行効率を考えるなら, ネイティブコードが望ましいが, 本プロトタイプシステムの主な目的である COMBO の有用性の確認にとっては, より実装が容易でかつ可搬性の高いバイトコードで十分である. そこで, COMBO 用の抽象機械を設計・実装し, その抽象機械で動くバイトコードへのコンパイラを実装する.

COMBO 言語は, 2 節で概説した文脈計算を表現する核言語をモーバイル機能で拡張したものであり, そのコンパイルも, 核言語のコンパイルにモーバイル機能を実現するための処理を加えて実現される. 核言語のコンパイルは, [5] に述べられている核言語の操作的意味論を実現すればよいので, ここでは詳述しない.

前節で述べた COMBO のモーバイル機能を実現するためには, 核言語を実現するための抽象機械に, 以下の機能を加えれ, コンパイラはこれら機能を使用するバイトコードを生成すればい.

- 大域的環境の操作
  - register(loc,τ,Key)
    loc (で示された位置) にある型 τ のオブジェクトを Key を大域名として登録する. register e_1 as e_2 に対応.

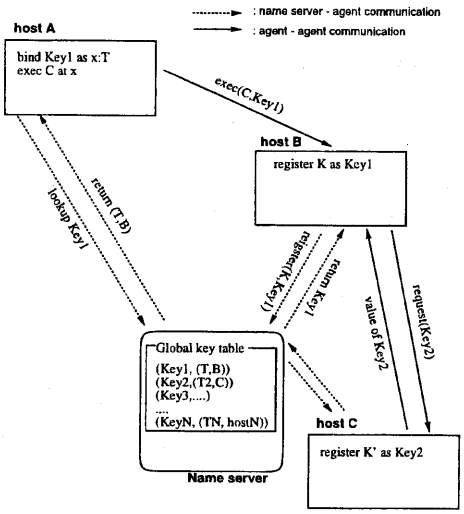

図 1: 実行環境の構造

- `lookup(Key,`τ`)`
  大域名 Key をもつオブジェクトの位置と型を大域的環境から取得し，その型が正しいものであるかどうか，実行時検査をする． **lookup** e **withtype** τ に対応.

- コードの移動
  コードの移動のためには，実行されるサイトの制御と，オブジェクトの送信リクエストの二つが必要になる.

  - `request(loc,Key)`
    loc に大域名 Key をもつオブジェクトの送信を要求する.
  - `exec(loc, C, Key)`
    loc において，大域名 Key のコンテクストの下でコード C を実行する.

4.2 実行環境

実行環境は，大域環境実現するネームサーバとプログラムを各サイト上で実行する実行エージェントとから構成される (図 1).

4.2.1 ネームサーバ

本プロトタイプシステムでは大域環境はネームサーバによって一元管理される．すなわち，ネームサーバは実行環境のすべての大域名とそれに対応する型，及びそのネットワーク上での位置の組のテーブルを保持する.

ネームサーバ–実行エージェント間には，大域的環境に関する抽象機械の機能に対応したプロトコルが存在し，ネームサーバはこのプロトコルによるエージェントからのリクエストを受け付ける．実行エージェントから register リクエストが送られてくると，ネームサーバはリクエストされた名前とその型，位置をテーブルに追加する．lookup リクエストが送られてきた場合には，ネームサーバはテーブルからその名前をさがし，型と位置をエージェントに返す動作をする.

4.2.2 実行エージェント

コンパイルされたプログラムは，ネットワーク上の各サイトの実行エージェントによって実行される．実行エージェントはバイトコード・インタプリタと，ネットワーク処理ルーチンを含んだ実行時ライブラリから構成される．実行時ライブラリは，分散機能を実現するルーチンと，他のエージェントからの要求に応答するためのルーチンからなっている.

実行エージェントは通常各サイト毎に一つずつ存在する．エージェント間には，前述したコードの移動に関する抽象機械の機能に対応したプロトコルが存在し，他のエージェントからのリクエストが来るまで待機している．リクエストを受けとると，エージェントは新しいスレッドを一つ生成し，そのスレッドで exec のリクエストに対しては新規にバイトコード・インタプリタを起動し，送られたコードとコンテクストを読み込んで実行する．また，request に対しては，要求された *key* の値をもとのエージェントに返す操作を実行する.

5 結論

本論文では，文脈を分散の単位とみなすことにより，柔軟なモーバイル言語を実現するため計算の分割および結合に関する機能が高階の型付きプログラミング言語の中で実現可能であることを示し，この

モデルに基づく高階モーバイル言語 COMBO の設計及び実現上の問題点を論じた．現在COMBO 実現の基礎となる，クライアント・サーバ形式で動作する関数型言語の抽象機械の実装実験，および，文脈計算系の実装実験を完了した段階である．今後，第4節で述べたモーバイル機能を実現する実行時機能を実現し，COMBO のプロトタイプを構築する計画である．

参考文献

[1] A.D.Birrell and B.J.Nelson. Implementing remote procedure calls. *ACM Transaction on Computer Systems*, Vol. 2, No. 1, pp. 39–59, Feb. 1984.

[2] A. Black, N. Hutchinson, E. Jul, H.Levy, and L.Carter. Distribution and abstract types in Emerald. *IEEE Transaction on Software Engineering*, Vol. 13, No. 1, January 1987.

[3] Luca Cardelli. Obliq A language with distributed scope. Proc ACM Symposium on Principles of Programming Languages, pages 286 – 297, 1995.

[4] Alessandro Giacalone, Prateek Mishra and Sanjiva Prasad Facile : A Symmetric Integration of Concurrent and Functional Programming *International Journal of Parallel Programming*, Vol. 18, No. 2, pp. 121–160, 1989.

[5] Masatomo Hashimoto. First-class contexts in ML. Submitted for publication. Available from `http://www.kurims.kyoto-u.ac.jp/~masatomo/`.

[6] Masatomo Hashimoto and Atsushi Ohori. A typed context calculus. Available from Research Institute for Mathematical Sciences, Kyoto University, as a preprint no. RIMS-1098.

[7] Kazuhiko Kato. Safe and secure execution mechanisms for mobile objects. In *Mobile object systems: towards the programmable Internet: second international workshop*, Vol. 1222 of *LNCS*, p. 201. Springer, 1997.

[8] Tatsurou Sekiguchi and Akinori Yonezawa. A calculus with code mobility. In *In Proceedings of Second IFIP International Conference on Formal Methods for Open Object-based Distributed Systems*, 1997.

[9] James W. Stamos and David K. Gifford. Remote evaluation. *ACM Transaction on Programming Languages and Systems*, Vol. 12, No. 4, pp. 537–565, October 1990.