自己反映アーキテクチャによる動的適応可能なソフトウェアのための言語

A Language for Dynamically Adaptable Software

天野 憲樹† 渡部 卓雄†
Noriki AMANO Takuo WATANABE

†北陸先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science,
Japan Advanced Institute of Science and Technology

概 要

計算機環境の動的変化に適応可能なソフトウェアアーキテクチャおよびそれを構成するための言語について述べる．

1 はじめに

開放型システムや移動計算機環境の普及にともない，プログラマは設計の段階でそのアプリケーションソフトウェア (以下，アプリケーション) がどのような実行環境で使用されるか，あるいはアプリケーションの実行時にその実行環境がどのように変化するかを完全に予測することができなくなった．こうした実行環境の相違やその動的変化がアプリケーションの実行に与える影響は無視できなくなりつつある．このためこのような計算機環境で使用されるアプリケーションは実行環境に適応できることが望まれる．本稿では実行環境の相違やその動的変化に効果的に対処可能な機構を動的適応可能性と呼ぶ．我々はアプリケーションに動的適応可能性を付加するためのソフトウェアアーキテクチャおよびそれを構成するための言語 LEAD [1] [2] を設計・実装し，その有効性を検証してきた．本稿ではこれまでの研究により得られた知見をもとに LEAD の問題点を述べ，その解決を目指して現在設計中であるオブジェクト指向言語 LEAD++ について述べる．鍵となるアイディアは LEAD にオブジェクト指向の概念を導入することである．これによりアプリケーション中の各処理ごとに，その抽象度に適した動的適応可能性の導入および記述が可能になる．また継承機構の導入によりこうした記述の再利用性が向上する．さらに自己反映性によるアプリケーションの柔軟な適応動作を局所的に実現することが可能になるなど，さまざまな利点が得られる．

2 プログラミング言語 LEAD

我々の目標は *1)* 拡張性の高い動的適応可能なアプリケーションの実現，*2)* 既存言語によるアプリケーションへの動的適応可能性の導入，の 2 点に集約される．本稿で動的適応可能性とは実行環境の相違やその動的変化に応じてアプリケーションの振舞いを変える機構を指す．我々は動的適応可能性を実現するためにアプリケーション中の手続きを実行環境に応じて変化させる手法を考案した．具体的には，アプリケーション中の 1 つの手続きに対して複数の定義を用意し，実行時にその環境に適したものを選択して実行する．この手法は手続き呼び出し時点の実行環境に応じて呼び出す手続きの定義を選択することで実現できるため，*1)* 動的適応可能性の制御機構を単純にし，制御に要するコストを低く抑えることができる，*2)* 手続きの定義を追加するこ

とでアプリケーションの動的適応能力を容易に向上できる，そして *3)* 既存言語に対する親和性が高い（手続き呼び出しの機能を持つ既存言語で記述されたアプリケーションに適用することができる），といった利点がある．本稿ではこのような実行環境に応じて変化する手続きを適応可能手続き，その個々の定義を適応可能メソッドと呼ぶ．適応可能手続きを効果的に実現するには以下の機構が必要となる．

1. 実行環境を構成する要素（環境要素）の状態を取得する機構．
2. 抽象度に応じた環境要素と状態を定義する機構．
3. 実行環境の状態とそれに適した適応可能メソッドを独立したモジュールとして記述する機構．
4. 実行環境の状態に適した適応可能メソッドを選択する機構（適応の機構）．

LEAD は上記の機構を言語機能として提供する．LEAD を用いることでアプリケーションが本来目的とする問題の記述と実行環境に適応するために必要となる適応の記述を分離することができる．この分離によりアプリケーションプログラムの可読性が向上し，保守が容易になるとともに，適応の記述の再利用性，拡張性が高まり，さらにこれを複数のアプリケーション間で共有することが可能となる．

3 LEAD の問題点

LEAD はその実行時システム上に（仮想的な）実行環境が唯一固定的に存在するモデルであるため，アプリケーション中の各処理に適した抽象度の異なる動的適応可能性の導入および記述を行うことが不可能である．しかし同一アプリケーション内でも，各処理ごとにその抽象度が異なることはごく一般的なことであり，この問題は柔軟な動的適応可能性の実現に関して大きな制約となる．

また環境要素は全て同一のアドレス空間上に大域的に定義されるため，その名前の衝突が起こる．さらに LEAD による動的適応可能なアプリケーションは自己反映性 [4] [5] を持ち，それを利用した柔軟な適応動作の実現が可能であるが，これを局所的に実現することができない．これらの問題は動的適応可能なアプリケーションの誤動作につながる．（仮想的な）実行環境を表現するオブジェクトを導入することでこれらの問題を解決することができる．

4 オブジェクト指向言語 LEAD++

我々は現在 LEAD の問題点を解決すべく LEAD に CLOS 的なオブジェクト指向概念 [3] を導入したオブジェクト指向言語 LEAD++の設計を行っている．CLOS 同様，LEAD++では LEAD++を構成する諸言語要素（クラス/総称関数/メソッド等）を LEAD++自身のオブジェクトとして表現するメタオブジェクトを導入する．そして LEAD++による動的適応可能なアプリケーションは LEAD++のメタオブジェクトプロトコル（MOP）[3] を用いて実現される．LEAD++は CLOS から大きな影響を受けているが，アプリケーションの動的適応可能性を実現する上で以下の特徴的な点を有する．

- 環境要素の状態を取得するライブラリ関数群．
- 実行環境をモデル化する環境オブジェクトとそのクラスに関する総称関数群．
- 適応可能手続きと適応可能メソッド．

LEAD++は環境要素の状態を取得するライブラリ関数群を備える．この関数群は *1)* ハードウェア環境の状態を取得する関数群，*2)* ソフトウェア環境の状態を取得する関数群，*3)* 計算機内部のプロセス状態を取得する関数群，そして *4)* 計算機周囲の環境状態を取得する関数群に大別される．個々の関数は実行環境の状態を構成する最小単位の要素である環境要素に対応する．これらの関数群を用いることで，適応可能手続きを実現する上で必要となる実行環境の状態を取得することができ，またこれらの関数群を組み合わせることで抽象度の高い実行環境の状態を表現することが可能となる．

LEAD++では実行環境をモデル化する環境オブジェクトとそのクラスに関する総称関数群が導入される．これらにより（仮想的な）実行環境を表現することができる．つまり環境オブジェクトはアプリケーション中の個々の処理に適した抽象度の実行環境を表現するオブジェクトであり，その総称関数群は環境オブジェクトの表現する実行環境への操作を提供する．またこの総称関数群は抽象度の高い環境要素を定義するものととらえることができる．先に述べた LEAD++のライブラリ関数群は抽象度の低い個々の環境要素の状態を取得するものであり，アプリケーション中の個々の処理の抽象度に合致するとは限らない．そのためこの環境オブジェクトの表現する実行環境の抽象度に適した環境要素およ

びその状態を定義する関数をこのオブジェクトのクラスに関する総称関数として実現する．以上からLEAD で実現不可能であったアプリケーション中の各処理に適した抽象度の動的適応可能性の導入および記述が可能となる．また抽象度の高い環境要素は各環境オブジェクトのクラスに関する総称関数として実現されため，その名前の衝突も回避される．

適応可能手続きと適応可能メソッドもLEAD++ではオブジェクトとして表現される．これらは総称関数とそのメソッドに類似しているが同一ではない．適応可能メソッドのディスパッチは実行環境の状態を表す条件式とそれを評価する戦略 (適応の戦略) によって実現される．また適応可能メソッドにはLEAD++以外の言語で記述された手続きコードや手続き名を記述することができる．

4.1 メタオブジェクトのクラス階層

前節で述べた環境オブジェクト，適応可能手続きそして適応可能メソッドはLEAD++ のメタオブジェクトという形で表現され，LEAD++におけるメタオブジェクトのクラス階層はCLOSのそれに近い形で実現される (図 1)．環境オブジェクトは`standard-env-object` クラスのインスタンスであり，適応可能手続きと適応可能メソッドは`standard-adaptive-procedure` クラス，`standard-adaptive-method` クラスのインスタンスである．`standard-env-object` クラスはハードウェア環境やソフトウェア環境などをモデル化するメタオブジェクトのクラスのサブクラスとして，また適応可能手続きと適応可能メソッドのクラスはそれぞれ，`standard-generic-function` (総称関数のクラス)，`standard-method` (メソッドのクラス) のサブクラスとして実現される．

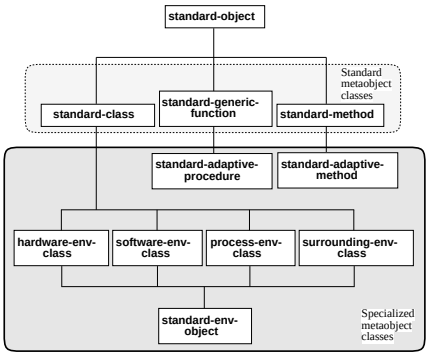

図 1　LEAD++におけるメタオブジェクトのクラス階層

4.2 動的適応可能なアプリケーション

LEAD++による動的適応可能なアプリケーションはアプリケーション (ベースレベル) とLEAD++の実行時システム (メタレベル) からなるメタレベルアーキテクチャ [4] を形成する (図 2)．

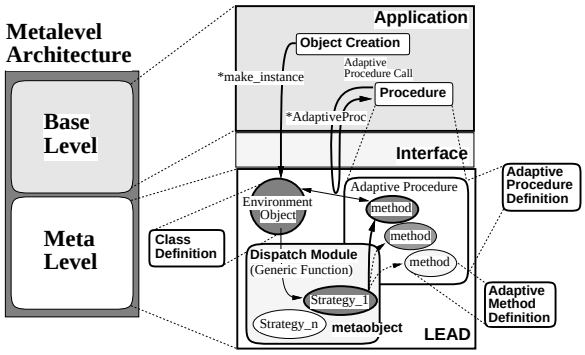

図 2　LEAD++による動的適応可能なアプリケーション

各レベルにおける役割は以下のようになる．

- メタレベル
  1. クラスと総称関数定義
  2. 適応可能手続きと適応可能メソッド定義
- ベースレベル
  1. インスタンス生成
  2. 適応可能手続き呼び出し

まずメタレベル上で環境オブジェクトのもとになるクラスの定義を行う．これによりアプリケーション中の各処理の抽象度に適した環境オブジェクトの生成が可能になる．次にアプリケーション中 (ベースレベル) の手続きを抽出し，適応可能手続きとしてメタレベル上で定義する．適応可能メソッドのディスパッチ条件の記述には環境オブジェクトのクラスに特化した環境要素 (総称関数) を使用する．この条件を評価する方法は適応の戦略として環境オブジェクト中に指定される．ディスパッチを行うモジュールはLEAD++の総称関数として実現され，戦略は個々のメソッドとして実装される．

環境オブジェクトのクラス定義はメタレベル上で行われるが，そのインスタンスの生成はベースレベルから通信インタフェースを介して行う*．アプリケーションはLEAD++の実行支援環境に用意されている通信インタフェースを使用して環境オブジェクトの生成をメタレベルに要求する．この操作によ

* ベースレベルがLEAD++のアプリケーションである場合，通信インタフェースは特に必要ない．

り，メタレベル上に指定されたクラスのインスタンス (環境オブジェクト) が生成される．この操作の返り値としてアプリケーションは生成された環境オブジェクトの識別子を受け取る．そしてこの識別子を引数として適応可能手続きの呼び出しを行う．

適応可能メソッドの選択は適応の戦略にもとづき，引数として渡される環境オブジェクトとその総称関数群が表現する仮想的な実行環境の状態に応じてなされる．この仮想的な実行環境は実際の実行環境の状態を反映している．選択された適応可能メソッドは通信インタフェースを介してアプリケーションに送られ，アプリケーション上で実行される．

アプリケーションは通信インタフェースを介して環境オブジェクト中に指定されている適応の戦略名を動的に変更することができる．この戦略の変更は必然的に適応可能メソッドの選択に反映し，アプリケーションの振舞いに反映する．つまり LEAD++ による動的適応可能なアプリケーションも LEAD によるそれと同様に自己反映性を持つ．しかし LEAD++では自己反映性を利用したアプリケーションの適応動作を局所的に行うことができる．

4.3 LEAD++によるプログラミング

LEAD++による標準的な動的適応可能アプリケーションのメタレベル記述は以下の手順で行う．

1. `standard-env-object` のサブクラスを定義する．
2. 1 で定義したクラスに関し，LEAD++の MOP に従って総称関数とメソッドの定義を行う．
3. 1，2 で定義したクラスと総称関数を使って適応可能手続きと適応可能メソッドの定義を行う．

図 3 は LEAD++による動的適応可能なメーラのメタレベル記述の一部である．このメーラはネットワーク (環境要素) の有線/無線接続に応じてその振舞いを変える．有線接続時は通常の動作を行うが無線接続時には特定の人間からのメールだけを取得する．記述例では CLOS 風の構文を用いている．`def-adapt-proc` と`def-adapt-method` はそれぞれ適応可能手続き，適応可能メソッドを定義するためのマクロとする．また図 4 はこのメーラのベースレベル記述の一部である．`adaptive-proc` は適応可能手続き呼び出しを行うための補助関数とする．この関数を用いることで，適応可能メソッドの選択からその実行までがなされる．

```
;;; 環境オブジェクトのクラス定義
(defclass new-class
          (standard-env-object) ())
;;;new-class の総称関数 (環境要素) 定義
(defmethod network-state ((obj new-class))
  (case (network) ;;;network はライブラリ関数
    (0 'wired)
    (1 'wired)
    (2 'wireless)
    (otherwise 'unknown)))
;;; 適応可能手続きの定義
(def-adapt-proc sample (obj)
  (:documentation "動的適応可能なメーラ")...)
;;; 適応可能メソッドの定義
(def-adapt-method sample ((obj new-class))
  (:dispatch-condition
      ((eq (netowrk-state obj) 'wired)))
  (:procedure-code
      "(defun mailer () ...)"))
(def-adapt-method sample ((obj new-class))
  (:dispatch-condition
      ((eq (netowrk-state obj) 'wireless)))
  (:procedure-code
      "(defun mailer () ...)"))
```

図 3　動的適応可能なメーラの記述 (メタレベル)

```
;;; インスタンス生成
(setq obj (make-instance 'new-class))
;;; 適応可能手続き呼び出し
(adaptive-proc obj '(sample arg))
```

図 4　動的適応可能なメーラの記述 (ベースレベル)

5 今後の予定

現在 LEAD++の詳細な言語仕様と実装方式について検討している．実装のプラットフォームとしては現在の LEAD の実験環境をそのまま使用し，LEAD++の実装は LEAD をベースにそれを拡張する方向で行う予定である．

参考文献

[1] Noriki Amano and Takuo Watanabe. Lead: A language for adaptable applications. In *Proceedings of the 12th International Technical Conference On Circuits/Systems, Computer And Communications (ITC-CSCC'97)*, pp. 825–828, July 1997.

[2] 天野憲樹, 渡部卓雄. モーバイルコンピューティングのための動的適応可能なソフトウエアアーキテクチャ. 日本ソフトウェア科学会 第 13 回オブジェクト指向計算ワークショップ (WOOC'97) 論文集. 日本ソフトウエア科学会第 13 回オブジェクト指向計算ワークショップ (WOOC'97), March 1997.

[3] Gregor Kiczales, Jim des Rivieres, and Daniel G. Bobrow. *The Art of the Metaobject Protocol.* The MIT Press, 1991.

[4] Pattie Maes and Daniele Naddi, editors. *Meta-level Architecture and Reflection.* ELSEVIER SCIENCE PUBLISHERS B.V., 1988.

[5] 渡部卓雄. リフレクション. コンピュータソフトウェア, Vol. 11, No. 3, pp. 5–14, May 1994.