代数仕様言語CafeOBJのための
拡張可能な前処理系の設計と実装

浅羽 義之 (110002)

北陸先端科学技術大学院大学 情報科学研究科

2003 年 2 月 14 日

キーワード: 代数仕様言語 CafeOBJ, 前処理系, モジュール, 拡張性, ポリシー.

1 本研究の背景と目的

本研究の目的は，代数仕様言語 CafeOBJ の構文を拡張可能にし，CafeOBJ による仕様の記述方法を支援することである．CafeOBJ は代数モデルにより形式的にソフトウェアの仕様を記述するための言語である．CafeOBJ は強力なモジュール機構，型付け，パターンマッチ機能などを備えており高い記述能力を持つが，その反面，仕様の記述が煩雑になる場合がある．そのような煩雑さが仕様の可読性や保守性に与える影響は無視できない．この問題を解決する方法として CafeOBJ の構文拡張が考えられる．CafeOBJ の構文を拡張することで，仕様記述の煩雑さを軽減することが期待できる．しかし，現在の CafeOBJ システムはユーザが容易に構文を拡張できるような機構を提供しておらず，その拡張は困難である．しかし，ユーザが CafeOBJ の構文を拡張したいという要求も多く，その容易な拡張機構が望まれる．さらに，構文拡張のための記述を独立したモジュールとして実現し，これらのモジュールを組み合わせることでインクリメンタルに構文拡張できることが望ましい．しかし，このような構文拡張の実現には，複数の拡張構文の衝突など解決しなければならない問題がある．以上の点を踏まえ，本研究では，CafeOBJ の構文を容易かつインクリメンタルに拡張するための機構を提案する．

2 本研究のアプローチ

本研究では，以下のアプローチにもとづいて CafeOBJ の構文拡張機構を実現する．

- 前処理系の導入による CafeOBJ の擬似的な構文拡張

現在の CafeOBJ システムを本研究の目的に応じて再構成することは現実的ではない．本研究では，現存の CafeOBJ システムそのものの拡張ではなく，前処理系の導入による

CafeOBJ の擬似的な構文拡張を目指す．具体的には，ユーザが前処理系に対して CafeOBJ の拡張構文を定義し，前処理系が拡張構文を CafeOBJ のコードに変換する．つまり，CafeOBJ への構文の追加は前処理系への構文の追加と等価になり，前処理系を拡張可能なアーキテクチャとして実現することになる．このような前処理系を実現するためのアプローチは以下である．

- 拡張構文とその CafeOBJ コードへの変換ルールをモジュール化する (以下，拡張モジュールと呼ぶ)．
- 拡張モジュールを CafeOBJ で記述する．
- 拡張モジュールの利用ポリシーを導入する．

まず，拡張構文とその CafeOBJ コードへの変換ルールを独立したモジュール (拡張モジュール) として実現する．しかし，記述の利便性を考慮し，複数の拡張構文とその変換ルールを 1 つの拡張モジュール中に定義できるようにする．このようなモジュール化の利点として拡張構文の再利用性や保守性の向上があげられる．また，拡張モジュールの記述言語として CafeOBJ を利用することで，ユーザは CafeOBJ を用いて CafeOBJ の構文を (擬似的に) 拡張することができ，さらに，CafeOBJ が提供する強力なモジュール機構なども利用可能となる．このような拡張モジュールをインクリメンタルに追加し，組み合わせることで前処理系を構成する．しかし，前節で述べた問題を解決する必要がある．ここで解決すべき問題とは，複数の拡張構文が衝突することを指す．具体的には，あるモジュール中に定義された拡張構文の名前が他のモジュール中に定義された拡張構文の名前とコンフリクトすることである．本研究では，拡張モジュールの利用ポリシーを導入することでこの問題を回避する．ユーザは使用する拡張構文がどの拡張モジュールに定義された変換ルールを用いるのかをポリシーとして陽に指定する．また，ユーザは仕様記述中のコンテキスト (ブロックのように指定された特定の範囲) という単位で使用する拡張構文の意味 (変換ルール) をポリシーとして指定できる．このようなポリシーの導入により，拡張構文間の衝突を回避し，かつ拡張構文の変換処理を柔軟に制御可能とする．このようなポリシーは拡張モジュールやそれを用いた仕様記述とは独立して記述する方式を採用する．この方式により，ポリシーを変更することで，拡張構文とそれを用いた仕様記述を変更することなく，拡張構文の意味 (変換ルール) を柔軟に変更することが可能となる．

以上のアプローチにもとづく前処理系を用いた CafeOBJ の構文拡張は以下のように実現される．前処理系はその初期状態において CafeOBJ の構文情報を定義する基本モジュールのみを持つ．ユーザは拡張モジュールを記述し，それを前処理系に組み込むことで前処理系の拡張を行う．前処理系はユーザが指定したポリシーにもとづいて拡張モジュールの取り込みを行う．前処理系には拡張モジュールを管理するための機構 (モジュール管理機構) があり，この機構がポリシーにもとづいて拡張構文間の衝突を回避する．

3　前処理系 Espresso の実装と例題の作成

提案したアプローチをもとに前処理系の設計・実装を行った．本研究ではこの前処理系を Espresso と呼ぶ．まず，Espresso の核となる基本モジュールは CafeOBJ を用いて実装した．基本モジュールには，CafeOBJ のパーザおよび拡張モジュールの作成を支援する汎用的な関数群が含まれている．Espresso のモジュール管理機構は Ruby 言語を用いて実装した．モジュール管理機構はユーザが記述した拡張モジュールの利用ポリシーを解釈する必要がある．そこで，ポリシーのための簡易記述言語を設計し，そのパーザを実装した．

次に，Espresso を用いて実際に CafeOBJ の擬似的な構文拡張を行った．この構文拡張は銀行システムの仕様をもとに以下のように行った．まず，CafeOBJ を用いて銀行システムの仕様を記述した．この仕様記述において煩雑な部分に着目し，拡張構文による支援を試みた．この例では，拡張構文を実現する 2 つの拡張モジュールを定義した．1 つは述語を 1 度で複数宣言するための拡張構文を含む拡張モジュールであり，もう 1 つは射影演算の状態が変化する等式を記述するための拡張構文を含む拡張モジュールである．これらのモジュールを Espresso に追加し，拡張構文を用いて銀行システムの仕様を改訂した．その結果，仕様記述のコード量が当初（構文拡張のない CafeOBJ コード）と比較して半分程度になることを確認した．また，これらの拡張モジュールの追加にともない，拡張構文の衝突を意図的に起し，Espresso がその衝突を適切に回避することも確かめた．

4　まとめ

本研究では，CafeOBJ の構文を容易かつインクリメンタルに拡張するための機構を提案し，その機構を拡張可能な前処理系 Espresso として設計・実装した．Espresso では，拡張モジュールの利用ポリシーを導入することで，拡張構文間の衝突回避を実現した．また，コンテキストの概念を導入し，ユーザが拡張構文の変換処理を柔軟に制御可能とした．さらに，Espresso を用いて CafeOBJ の (擬似的な) 構文拡張を行った．この構文拡張は CafeOBJ による銀行システムの仕様をもとに行い，Espresso による構文拡張の有用性を示した．

今後の課題としては，以下があげられる．まず，Espresso を用いて多くの例題を記述し，詳細な実験と考察を行う必要がある．特に，拡張モジュールの数に関する実験を行い，本研究のアプローチの限界について考察する．さらに，これらの実験結果から得られた知見を Espresso の設計に反映させ，Espresso の改良も行う．