平成 14 年度　修士論文概要		
横断的性質の分離を実現する 型付オブジェクト指向言語に関する研究	世木研究室	山口陽平
	学籍番号	No. 13417107

1　はじめに

ソフトウェアに対する要求が巨大で複雑になるにつれて、従来の型付プログラミング言語では「記述の反復によって型安全を実現する」か「型安全を捨てて記述の反復を防ぐ」か、いずれかの設計を迫るソフトウェアが現れるようになった。例えばメソッド呼出のロギングを考えるとメッセージ出力の記述は全てのメソッドに分散してしまう。しかし出力されるメッセージの内容はメソッドの引数や戻り値などを知っていないと作れないため、記述の反復を避けるにはメタプログラミングが必要となる。このような現象は横断的性質を従来の型付きプログラミング言語がうまく管理できないために発生する。横断的性質とはソフトウェアを構成する性質の１つで、それはいくつかの性質に関連する。そして横断的性質を他の性質から分離して管理することは横断的性質の分離と呼ばれ重要な研究分野になっている。

横断的性質の分離のための代表的な試みとして、Aspect [1]、Mixin Layers [2]、Mixin Inheritance [3]、Virtual Class [3] などがある。本研究の目的はこれらの方法がどのような横断的性質を型安全に分離でき、どのような問題を持っているのかを明らかにすることである。もう１つ目的はそれらの問題を克服する単純で優れた計算理論を構築することである。その理由は現在の多くの型付プログラミング言語や計算理論がいくつかの不必要な概念を含み、不必要に複雑になっているからである。Martin Abadi と Luca Cardelli も同じような理由で An Imperative Object Calculus を構築した [4]。ソフトウェアの性質は際限なく複雑化し、提案されている手法では型安全に分離できない横断的性質が数多く存在している。本研究はそのような横断的性質の分離と型安全の両立を目指した研究の足がかりとしたい。

2　横断的性質を分離する方法に対する考察

横断的性質を分離する方法として代表的な Aspect、Mixin Layers、Mixin Inheritance、Virtual Class を考察することで、これらの方法は一見大きく異なる方法に見えるが本質的には同じであることが分かる。その本質とは総称 (genecity) である。Aspect は既存のクラスへのメンバ追加やメソッドフックなどをするために総称関数を構成する。Mixin Layers は総称を使いスーパークラスをパラメータ化し、ロールとスーパークラスを分離する。Mixin Inheritance は直列化アルゴリズムによってスーパークラスが確定する Mixin Layers である。Virtual Class はオーヴァーライドできるメンバクラスで、それを使って総称を実現するが、それは総称よりも柔軟である。総称クラスのインスタンス化は一度だけしか行なえず、インスタンスクラスと総称クラスの間にはサブクラス関係がないという問題があるが、Virtual Class の場合はそのような問題はない。そのかわりに Virtual Class では部分型ではないサブクラスを構成できるという問題がある。

Aspect は他の方法と比べ多くの概念を含んでいるが、その多くはメソッドをクラスで表現することで実現できる。

以上の考察から計算理論は Aspect が含む概念やメソッドのようにクラスで代用できるものを含むは必要はない。そして Virtual Class は問題があるが総称よりも単純で柔軟な型システムであり、提案する計算理論に採り入れる価値がある。ただし Virtual Class の問題を解決しなければならない。

3　型なしオブジェクト計算

表 1 に従って作られる項のための型なしオブジェクト計算を定義する。オブジェクト項 $OBJECT$ は名前で識別されるオブジェクトである。構築項 new は評価時にオブジェクト項を構築する。呼出項 $TERM.VARIABLE+$ は $TERM$ を評価した結果得られるオブジェクト項のメンバ $VARIABLE$ に代入されている項を評価する。更新項 $TERM.VARIABLE- = DATA$ は項 $TERM$ を評価して得られるオブジェクト項のメンバ $VARIABLE$ にデータ $DATA$ を評価して得られる項を代入する。データ $DATA$ は $+TERM$ なら呼出時評価で $-TERM$ なら更新時評価である。このような項を使って逐次・反復・分岐・関数などを構成できる。表 2 に逐次オブジェクトの例を示す。オブジェクト $sequence$ のメンバ $execute$ を評価するとメンバ $first$、$second$ の順に評価される。そのため $execute$ を評価する前に $first$ と $second$ に項を代入する必要がある。

$$
\begin{array}{rcl}
TERM & = & OBJECT \mid new \\
 & \mid & TERM.VARIABLE+ \\
 & \mid & TERM.VARIABLE- = DATA \\
DATA & = & MODE\ TERM \\
MODE & = & + \mid -
\end{array}
$$

表 1: 項の構文

さらに、これらの項を評価するための評価規則を定義し、それをJavaで実装し項が評価されるのを確認した。

```
sequence.execute- = +((
  sequence.result- = -sequence.first+
).result- = -sequence.second+)
```

表2: 逐次オブジェクト

4　型付きオブジェクト計算

表3に従って作られる型を使って項を型付けする型付きオブジェクト計算を定義する。この型付きオブジェクト計算の型にはメンバ呼出型$TYPE.VARIABLE+$とメンバ更新型$TYPE.VARIABLE-$という新しい型とVirtual Class に相当するメンバ型$TYPE.PRIMITIVE$がある。メンバ呼出型はメンバに代入されている項を評価して得られるオブジェクト項の型を表し、メンバ更新型はメンバ代入できる項の型を表す。通常の型付き計算理論 [4]では同じ変数の型は入出力ともに同じであるため、varianceの概念が制限されてしまっている。我々の計算理論ではメンバ型をオーヴァーライドの代わりに部分型宣言$TYPE \leqq TYPE$の追加を行なう。オーヴァーライドでは既に存在する部分型関係を壊してしまう可能性があるが、部分型宣言の追加は既に存在する部分型関係を壊さない。さらにこの計算理論はSubtyping PolymorphismでParametric Polymorphismを実現できる。つまり部分型関係だけで総称を実現できる。その例を図4に示す。この例には型$Memory.Element$の値しか呼出・更新できないメモリ型$Memory$がある。その部分型として人間用メモリ$PersonMemroy$を定義し、部分型関係$PerosnMemory.Element <= Person$と$Person <= PerosnMemory.Element$を追加する。つまり$PerosnMemory.Element == Person$であるので$Memory$にはParametric Polymorphismがあると言える。

```
Memory,             Memory.Element,
Memory.element+,    Memory.element-,
Memory.element+ <= Memory.Element,
Memory.Element  <= Memory.element-,
Person,
PersonMemory, PersonMemory <= Memory,
PersonMemory.Element <= Person,
Person <= PersonMemory.Element,
```

表4: Parametirc Polymorphism

$$
\begin{aligned}
TYPE &= PRIMITIVE \\
&\mid TYPE.PRIMITIVE \\
&\mid TYPE.VARIABLE+ \\
&\mid TYPE.VARIABLE- \\
DECLARATION &= TYPE \\
&\mid TYPE \leqq TYPE \\
&\mid OBJECT : TYPE \\
ENVIRONMENT &= DECLARATION \\
&\mid DECLARATION, \\
&\quad ENVIRONMENT
\end{aligned}
$$

表3: 型の構文

5　健全性

型付きオブジェクト計算の健全性を示すためにサブジェクトリダクション定理を証明した。この定理は項の構文に従った正しい項があり、その項を型付け規則に従って型付けできるとき、その項を評価規則によって計算した結果も元の型と同じ型を持つという定理である。

6　まとめ

横断的性質を分離するいくつかの方法について考察し、その特徴や問題を明らかにした。その結果、考察したそれぞれの方法は基本的に同じ特徴を持っていることが分かった。さらに、メンバ型の書き換えではなく、メンバ型に関する部分型関係の追加だけを許すことでVirtual Classの問題を解決した型付きオブジェクト計算を提案した。

今後はこの型付きオブジェクト計算を洗練させ例外の概念を追加したい。また、この計算理論のための実用的な処理系を開発したい。

参考文献

[1] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin：Aspect-Oriented Programming, ECOOP, 1997.

[2] Yannis Smaragdakis and Don Batory：Implementing Layered Designs with Mixin Layers, ECOOP, 1998.

[3] Erik Ernst：Propagating Class and Method Combination, ECOOP, 1999.

[4] Martin Abadi and Luca Cardelli：An imperative object calculus, TAPSOFT, 1995.