様式 C-19

科学研究費補助金研究成果報告書

平成 23 年　6 月　3 日現在

機関番号：12601
研究種目：若手研究（B）
研究期間：2008～2010
課題番号：20700022
研究課題名（和文）　効率的な高信頼性ソフトウェア開発のためのプログラミング言語の研究

研究課題名（英文）　Programming Language for Effective and Reliable Software Development

研究代表者
紙名　哲生（KAMINA Tetsuo）
東京大学・大学院教育学研究科・特任助教
研究者番号：90431882

研究成果の概要（和文）：ソフトウェアの信頼性を保証しつつ開発効率を上げるために、プログラムを実行する前にある種のエラーが起こらないことを保証できる性質と、プログラムを部品から構成し、さらにそれらの部品も別の部品から組み立てられるような性質の両方において優れた機能を持つプログラミング言語を実現した。また、実行時に部品の結合や切り離しを柔軟に且つ安全に行えるようなプログラミング言語の研究を行い、実行前に安全性の検証を行えるような新たな仕組みを実現した。

研究成果の概要（英文）：To enhance effectiveness of software development while preserving the reliability of products, we developed a new programming language that supports both features of type safety (ability to detect type-relating errors before program execution) and scalable modularity (ability to construct products from software modules, which are also constructed from other fine-grained modules). We also developed a new programming language that supports dynamic module composition and decomposition, and realized a new mechanism for program verification.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
2008 年度	1,100,000	330,000	1,430,000
2009 年度	1,300,000	390,000	1,690,000
2010 年度	900,000	270,000	1,170,000
年度			
年度			
総　計	3,300,000	990,000	4,290,000

研究分野：総合領域
科研費の分科・細目：情報学・ソフトウェア
キーワード：プログラム言語論・プログラミングパラダイム・ソフトウェア工学

１．研究開始当初の背景

(1) 現在、ソフトウェアは大規模なサーバから小型の端末や自動車・家電の制御に用いる超小型のコンピュータに至るまで、日常生活のあらゆる場面に存在する。ソフトウェアの開発コストをいかに下げ、その一方で品質をいかに上げ、開発期間をいかに短縮するかを考えることは、学術的に大きなチャレンジであるだけでなく、社会的要請から見ても非常に重要な問題である。

(2) これまで、この問題に関して非常に多くの研究がなされ、学術的貢献が行われてきた。その中の一つに、プログラミング言語の発達があげられる。現在ではオブジェクト指向言語が広く用いられるようになってきた。これらの言語の中には型安全性の性質が保証されているものも多く、この性質により、バグ

の発見をある程度自動化することができ、ソフトウェアの品質を高めることができる。

(3) 一方、ソフトウェア開発の効率性の観点から見ると、型システムは次の性質を持つことが望ましい。①高い抽象化機構を持ち、様々な場面で再利用できるソフトウェア部品を作れること（再利用性）②既存のシステムに手を加えることなく、新たなシステムを拡張して作れること（拡張性）③モジュールの入れ子関係を任意に拡大できること（スケーラビリティ)。それぞれの性質において、これまで数々の研究がなされてきたが、ソフトウェア開発プロセスの大幅な改善に寄与するために、今後はこれら全ての性質において優れた性能をもつプログラミング言語が必要である。

２．研究の目的

(1) 本研究課題では、これまでの研究成果を統合した新たなプログラミング言語を実現する。とくに、それぞれの言語機構を一つの言語の中で統合した際、それらが干渉し合うときに、いかにしてそれを解決するかを考える。そして、複雑で使いにくい言語ではなく、いかに単純な言語として、且つ産業界における技術移転の障壁を少なくする目的から、広く用いられている既存の言語とあまり違わない言語として実現する。

(2) 本研究課題では、以上の議論を踏まえ、以下の事項に関して研究を進める。

① 再利用性・拡張性・スケーラビリティのための型付計算体系を構築する。とくに、Java のようなオブジェクト指向言語に基づいた計算体系とする。

② ①の結果に基づいて、実用的なプログラミング言語を開発する。実装に関しては新たな実行環境まで含めて開発するのではなく、既存の仮想機械など、標準的なプラットフォームで実行可能な実装方法について検討する。

３．研究の方法

(1) 既存の研究としては、Java の総称型をクラス定義外から参照可能にした型変数参照の機構、nested inheritance、dependent classes、mixin 継承などがある。これらのアイデアを整理し、Featherweight Java (FJ) に基づく形式化を行い、型安全な型システムを構築する。

(2) 標準的な Java プラットフォームでの実行可能性を実現するため、拡張可能な Java コンパイラ、JastAddJ に基づいた言語処理系の実装を行う。

４．研究成果

(1) 内部クラスアクセスの外側（その内部クラスが宣言されてあるクラス）を型変数でパラメータ化できる機構 lightweight dependent classes を提案した。この機構は、

```
class Graph {
  class Edge extends EdgeI<Graph>{}
  class Node extends NodeI<Graph>{}
}

class EdgeI<G extends Graph> {
  G.Node src, dst;
  void connect(G.Node s, G.Node d) {
    s.add(this); d.add(this);
    src = s; dst = d;
  }
}

class NodeI<G extends Graph> {
  Vector<G.Edge> es =
    new Vector<G.Edge>();
  void add(G.Edge e) { es.add(e); }
}
```

のように、内部クラス Node や Edge を型変数上でアクセスできるようにしたものである。この機構により、内部クラスに対する参照が型変数によって相対化されるので、相互再帰的な型の集合を安全に（ダウンキャストの必要なく）拡張することができるようになった。また、内部クラスの実装をクラスの外側に置くことができるため、プログラマは一枚岩の巨大なクラスを作る必要がなく、内部クラス実装の部品化が可能になった。

この機構を実現するためには、文法の拡張だけでなく、型システムの改造、とくに this の型付け規則について大幅な変更が必要となる。本研究では、FJ に基づいた型システムの形式化を行い、型安全性を証明した。

本研究は、研究代表者らによる過去の研究成果である型変数参照の機構と表現力がほぼ一緒であるのに対し、プログラムの書きやすさ・読みやすさに関しては、煩雑な fixed-point クラスの宣言を繰り返し行わなくて済む分、本研究が改善している。また、文法としては Java の内部クラスアクセス(.)の左側に型変数を書けるようにしただけであり、従来研究の dependent classes と比較して、非常に簡潔な言語拡張となっている。簡潔な言語拡張は、処理系の実装のし易さだけでなく、プログラマの学習の負荷を低減することにも貢献する。

(2) Lightweight dependent classes に、

nested inheritance（の簡潔版）を導入する拡張を行った。文法的には、クラス宣言のスーパークラス部に、以下のように型変数でパラメータ化された内部クラスアクセスを記述できるようにした点が拡張されている。

```
class AST {
  class Expr extends ExprT<AST>{}
  class Const extends ConstT<AST>{}
  class Plus extends PlusT<AST>{}
}

class ExprT<L extends AST> {
  String format() { return “” ;}
}

class ConstT<L extends AST>
      extends L.Expr {
  int val;
  String format() { return “” +val; }
}

class PlusT<L extends AST>
      extends L.Expr {
  L.Expr op1, op2;
  String format() {
    return op1.format()+” +”
      +op2.format(); }
}
```

この拡張により、lightweight dependent classes では不可能だった同レイヤ内での継承関係を実現し、さらに継承関係を含めた相互再帰的な関係を持つクラス群を安全に拡張することが可能となった。

一方、多重継承で古くから問題となっているダイアモンド継承の問題が生じたが、本研究ではそれを mixin 継承の機構を導入することによって解決した。よって本研究は、lightweight dependent classes の拡張版としてだけでなく、nested inheritance の簡易版、mixin 継承の応用としても位置付けることができる。これらの成果は、最終的にプログラミング言語 DJ として実装された。

(3) 当初の研究計画にはなかったことだが、時流の流れに従い、ソフトウェア部品の動的結合の仕組みについても研究を行った。ソフトウェア部品を動的に結合できることによって、柔軟で効率的なソフトウェア開発を行うことができる。しかしこのような動的な性質は、しばしば安全性を損なう。本研究では、役割に基づく動的オブジェクト適応モデル Epsilon の型システムを、FJ に基づいて実現し、型安全でない場合がどのような場合であるかを形式的に議論した。またそれに基づいて、Epsilon の型安全な variation である NextEJ を設計した。NextEJ の設計には、近年注目を集めている文脈指向プログラミング（COP）の with 構文のアイデアを借りて、以下のようにオブジェクトの振る舞いを局所的に変更させることができる。

```
Building midtown = new Building();
Person tanaka = new Person();
Person suzuki = new Person();
Person sato = new Person();
bind tanaka with midtown.Guest(),
     suzuki with midtown.Guest(),
     sato with midtown.Security() {
  ..
  sato.notify();
}
```

この例では、bind 構文のスコープ内で、Security が提供する振る舞いを獲得した sato が、Security で宣言されてある notify メソッドを呼んでいる。また、同じく Guest が提供する振る舞いを獲得した tanaka と suzuki が、notify メソッドによってイベントを通知される。

本研究は COP とオブジェクト適応モデルの融合であるとも位置付けることができる。

(4) イベントに基づく COP 言語 EventCJ を実現した。これは、動的結合されるモジュールだけでなく、いつ、どのモジュールが、どの実行主体に結合されるかという仕様も分離して実装できる点に大きな特徴がある。この分離は、モジュール結合の契機となるイベントの宣言と、イベント生成に伴うモジュール結合の規則を宣言することによって行う。

```
declare event GPSEvent(Navi n, int s)
 :after call(void Navi.statusChange(s))
&&target(n)&&args(s)&&if(GPS.AVAILABLE)
 :sendTo(n);

transition GPSEvent:
  WifiNavi switchTo GPSNavi |
  not OnBoard activate GPSNavi;
```

この例では、イベント GPSEvent が、Navi インスタンス n の statusChanged メソッドが呼ばれた時で且つ GPS.AVAILABLE が真だった場合に生成され、n に送られる。n では、transition で宣言された規則に従い、事前条件にマッチすれば GPSNavi モジュールが結合される。

さらに、transition 規則をもとに、モデル検査の分野で広く用いられている Promela 言語

のコードを生成することができる。これによって、各モジュールの排他性や、結合順序に関する時相的な性質を検証することが可能になった。

本研究は、COP 言語に対して、イベントに基づく状態遷移モデルとしてモジュールの結合状態を宣言的に記述する機構を新たに導入した。また、NextEJ 同様インスタンスに対してモジュール結合を行う。宣言的なイベント記述という観点から event-based programming とも関連がある。また、状態遷移モデルがモデル検査へと応用できることから、ソフトウェア工学分野に対しても今後大きなインパクトが予想される。

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計７件）

1. Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara, EventCJ: A Context-Oriented Programming Language with Declarative Event-based Context Transition. In Proceedings of the 10th Annual Aspect-Oriented Software Development Conference (AOSD.11), pages 253-264, 2011. （査読有）
2. Tetsuo Kamina and Tetsuo Tamai, Lightweight Nested Inheritance in Layer Decomposition. In Proceedings of the 2010 International Workshop on Foundations of Object-oriented Languages (FOOL'10), 2010. （査読有）
3. Tetsuo Kamina, Tetsuo Aotani, and Hidehiko Masuhara, Designing Event-based Context Transition in Context-oriented Programming, In Proceedings of the International Workshop on Context-oriented Programming (COP'10), article No.2, 2010. （査読有）
4. Tetsuo Kamina and Tetsuo Tamai, A Smooth Combination of Role-based Languages and Context Activation, In Proceedings of the Ninth Workshop on Foundation of Aspect-Oriented Languages (FOAL 2010), pages 15-24, 2010. （査読有）
5. Tetsuo Kamina and Tetsuo Tamai, Towards Safe and Flexible Object Adaptation, Proceedings of the International Workshop on Context-Oriented Programming (COP09), article No.4, 2009. （査読有）
6. Tetsuo Kamina and Tetsuo Tamai, Lightweight Dependent Classes. Proceedings of the 7th ACM International Conference on Generative Programming and Components Engineering (GPCE'08), ACM Press, pages 113-124, 2008. （査読有）
7. Tetsuo Kamina and Tetsuo Tamai, Flexible Object Adaptation for Java-like Languages, Proceedings of the 10th Workshop on Formal Techniques for Java-like Programs (FTfJP 2008), pages 63-76, 2008. （査読有）

〔学会発表〕（計３件）

1. 紙名哲生, 青谷知幸, 増原英彦. EventCJ: A Context-Oriented Programming Language with Declarative Event-based Context Transition. PPL2011, 2011 年 3 月 11 日.
2. Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara, EventCJ: Realizing Declarative Event-based Context Transition. In AOAsia/Pacific'10, September 24, 2010.
3. 五十嵐淳, 紙名哲生. レイヤー合成のための型理論. 組木シンポジウム --ソフトウェアのコンポジション技術の最前線--, 2009 年 11 月 25 日.

６．研究組織
(1)研究代表者

紙名　哲生（KAMINA, Tetsuo）
東京大学・大学院教育学研究科・特任助教
研究者番号：90431882