様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

# 科学研究費助成事業　　研究成果報告書

平成　３０　年　　８　月　３１　日現在

機関番号：　３２６１９
研究種目：　基盤研究(C)（一般）
研究期間：　2014～2017
課題番号：　２６３３００８９

研究課題名（和文）拡張可能なステートフルアスペクトの設計と実装

研究課題名（英文）Design and Implementation of extensible stateful aspects

研究代表者
福田　浩章（Fukuda, Hiroaki）

芝浦工業大学・工学部・准教授

研究者番号：３０３８３９４６

交付決定額（研究期間全体）：（直接経費）　　3,000,000円

研究成果の概要（和文）：従来ステートフルアスペクト言語では，開発者は正規表現を用いたパターン記述で実行履歴の絞り込みを行う．記述されたパターンはプログラム実行時に評価されるが，パターンをどのように解釈し，実行履歴にマッチさせるのか，というパターンの意味論は予め言語設計者が決定しており，開発者は変更できない．
そこで本研究では，柔軟なパターン記述を行うため，パターン解釈の意味論を開発者が変更可能なステートフルアスペクトの基本要素と織り込み方法(モデル)を提案し，具体的なプログラム言語を用いたシステムを実現した．また．アスペクトの織り込み方法を応用し．非同期処理の複雑な緩和する手法の提案，およびライブラリの実装を行った．

研究成果の概要（英文）：In the traditional Stateful Aspect languages, developers need to specify the pattern using the regular expression. The patter is evaluated when the program runs; however the semantics of the language is predefined by its language designers. Therefore developers cannot modify the semantics if they want.
In this search, we design the new stateful aspect language which enables developers to modify the semantics of the pattern, and it's weaving mechanism. We also provides a prototype implementation of our new stateful aspect language using the practical programming language. Besides, we also propose a mechanism that will avoid the complicated expressions of asynchronous programming reusing the weaving mechanism and provides a prototype library that follows our proposal.

研究分野：　ソフトウェア工学

キーワード：　stateful aspect　aspect oriented　modularity　programming language

## １．研究開始当初の背景

アスペクト指向プログラミング[1][2]は，オブジェクト指向では 1 つのモジュール(クラス)として扱うことが難しかった横断的関心事を，新たなモジュールであるアスペクトにまとめ，ソフトウェアのモジュール化を促進させる技術である．プログラム実行時にはアスペクトはクラスに合成されて(織り込まれ)実行されるが，織込みは「メソッドが呼ばれた時」など，プログラムの現在の実行状態のみが対象となる．そのため,「メソッドが 3 回連続で呼ばれた時」など，実行履歴を織込み対象とするには，開発者自身が履歴の保存や織込みを実行する必要がある．

この問題に対し，プログラムの実行履歴を考慮したステートフルアスペクトの研究も進められており，EventJava[3], HALO[4], tracematches[5]といった，実行履歴を織込み対象として指定できる言語が提案されている．これらの言語では，実行履歴をパターンとして記述して織り込み対象を指定するが，パターンをどのように解釈するのか，という意味論は言語設計者が予め決定しており，開発者が変更することは出来ない．その結果，開発者の直感にそぐわないパターン記述をせざるを得ない．また，複雑なパターンを記述する場合，既存のパターンを組み合わせて記述できることがモジュール化という観点からも望ましいが，既存の言語ではパターンを再利用することが出来ない．すなわち，(1)パターンを再利用でき，(2)パターン解釈の意味論を開発者が変更できる機構が必要となる．以下，ステートフルアスペクト言語の 1 つである tracematches を利用し，具体例とともに既存技術の問題点を述べる．

Tracematches では，他の言語と同様に正規表現でパターンを記述する．例えばエディタアプリケーションにおいて，“ 3 回編集される度に自動で保存する ”というアスペクトは図 1 のように記述する．この記述では，プログラムの実行履歴が edit→edit→edit であった場合に ”Editor.save() ”(アドバイス)が実行されることになる．また，このアスペクトの定義では，プログラムの実行履歴が edit→edit→edit→edit のように，4 回繰り返される場合，アドバイスは 2 回実行されてしまう．これは，2 回目から 4 回目の実行が“ 3 回編集する ”というパターンにマッチしたからである．この振る舞いは，“ 3 回編集する度に ”という意味の解釈をどう扱うかによって変わる．3 回繰り返し実行した後，さらに 3 回繰り返されることが“ 3 回編集する度に ”である，と考える開発者にとっては意図に反することになる．仮に後者の意味としてアドバイスを織り込むためには，tracematches では図 2 のように記述しなければならない．図 2 では，実行履歴にマッチした時に実行されるアドバイス(Edit.save())をシンボル(toggle)として定義し，パターンのマッチングプロセスにおいて敢えて余計なシンボルを割りこませることで 4 回目の edit にアドバイスが織り込まれることを防止している．

さらに，“「3 回編集する度に保存する」を 2 回繰り返したら警告メッセージを出す ”というパターンを記述することを考えると，最も簡単な方法は「3 回編集する」というパターン(このパターンを threeEdit とする)を再利用し，”threeEdit threeEdit ”のように記述することである．しかし，tracematches をはじめとする多くのステートフルアスペクト言語では，パターンの再利用や代入を行うことが出来ない．その結果，開発者は冗長な記述を行う必要があり，再利用性や保守性が損なわれる．

```
tracematch() {
  sym edit after: call {* Editor.edit() }
  edit edit edit {
     Editor.save();
  }
```

図 1

```
tracematch() {
  sym edit after: call {* Editor.edit() }
  sym toggle after: call {* Editor.save()}
  edit edit edit {
     Editor.save();
  }
```

図 2

## ２．研究の目的

本研究 では，(1)パターン解釈の意味論を言語設計から切り離し，開発者自身が変更可能にすること，(2)パターンの記述をファーストクラス(変数への代入や関数の戻り値として利用できること)として扱いパターンの再利用を可能にすること，この 2 つを可能にするステートフルアスペクトのモデルを提案し，具体的なプログラム言語を用いてシステムを実装する．このモデルでは．Open Implementation[6]のガイドラインに則り．ステートフルアスペクトの要素としてパターンとアドバイスの他にパターン解釈の意味論を変更できる要素を加え，実行履歴とマッチングの際に参照することによって開発者が自由にマッチングプロセスを変更できる機構を提供する．

## ３．研究の方法

本研究の目的は，ステートフルアスペクトにおけるパターン記述をファーストクラスにすること，およびパターンの評価ルールを言語設計から切り離し，開発者が変更可能にすることである．ステートフルアスペクトはプログラムの実行履歴にマッチするが，記述されたパターンが実行履歴にマッチするか否かは，既存のアスペクト指向言語におけるポイントカット評価(アスペクトの織込みを行うか否かを決定する評価)を逐次実行することで表現できる．また，tracematches をはじめとする既存のステートフルアスペクト指向言語では，記述されたパターンからオートマトンを生成し評価している．すなわち，プログラムの実行を進める度にオートマトンの状態を遷移し，終了状態に到達したときにマッチと判定する．そして，パターンの評価ルールを変更するということは，オートマトンを実行時に書き換えながら評価していく，ということになる．そこで本研究では，１つの状態を１つの関数として表現し，オートマトンを関数の組み合わせで表現するというアプローチを採る．そして，評価ルールも関数で表現し，評価ルールを表す関数と，状態を表す関数を実行時に柔軟に組み合わせることによって，パターン解釈の意味論を変更可能にする．この手法によって，パターンをファーストクラスとして扱うことも可能になる．

これらの点を踏まえ，次のように研究を進めていく．

まず，ステートフルアスペクトを織り込む環境を構築する．アスペクトの織込み方法には，実行前にコンパイルして織り込む方法，実行時に織り込む方法という２種類あるが，26年度は技術的な検証を行うことに集中するため，実現が容易な実行時の織り込みを採用する．織り込みの具体的な言語としては ActionScript3 を利用し，動的な織込み機構を実現している AspectScheme[7] や AspectScript[8] の方式を参考にする．AspectScript は ActionScript3 と同様に ECMAScript に準拠した JavaScript を対象にしたシステムであることから，その実装方式は非常に参考になると考えている．

次に，前述した織込み環境を利用し，高階関数で表現したパターンを，開発者が同じく高階関数で定義する評価ルールを参照しながらアスペクトをマッチさせるプロトタイプシステムの実装を行う．具体的には，アスペクトを織り込む際，パターンを表す関数を直接評価するのではなく，デザインパターンの１つである Decorator を利用し，パターンの評価を評価ルールで装飾することによって実行時にパターンを書き換えながら実行履歴にマッチさせるというアプローチを採る．なお，このアプローチは[3]でも採用され，有効に機能することが確認されている．なお，このプロトタイプシステムではステートフルアスペクト言語の中で最も複雑なパターン評価の意味論を備えた tracematches の意味論を模擬することを目標とする．

次に，作成したプロトタイプシステムから，パターン評価の意味論を開発者が変更できる機構を備えたステートフルアスペクトに必要なモデルを考案する．具体的には現行のアスペクト指向言語が備えているポイントカット-アドバイスモデルや，[4]においてアドバイスの織り込み方が形式的に提案されたように，ステートフルアスペクトが備えるべき要素(パターン，評価ルール，アドバイスなど)やそれらの性質(引数の型や返り値の型)，およびそれらの評価法や織り込み方法を言語に依存しない形で提案していく．

また，プロトタイプシステムを発展させ，具体的なステートフルアスペクト指向フレームワークを構築していく．具体的には，アスペクト指向言語のデファクトスタンダードである AspectJ を模倣し，AspectJ が備えるポイントカット(e.g. call, execution, cflow)や，アドバイス(e.g. before, after, around)に対応する．なお，実行時に織込みを行うためには代入や関数の実行を監視できる必要があるため，開発者が記述するソースコードを変換するプリプロセッサを実装する．なお，このプリプロセッサの実装には，ActionScript3 のソースコード変換を目的としたライブラリである metaas[5]の利用を考えている．そして，開発したフレームワークを利用し，ウェブアプリケーションで散在してしまう非同期処理などに適用し，有効性を示す．

これらの過程で得た結果は，適宜国際会議や論文誌で発表していく．

## ４．研究成果

本研究では，計画通りに研究を遂行し，まずは ActionScript3 を利用してアスペクトの織り込み機構を実現した．具体的には，関数の呼び出しをクロージャで内包し，関数呼び出しをすべて観測することで実行時にアスペクトの織り込みを実現している．この織り込み機構をベースとし，織り込み時にパターンの評価ルールを関数の入れ子構造で表現したオブジェクトを開発者が与えることで，実行時にパターンの評価ルールを変更することを可能にした．これらの成果を論文誌に投稿し，採択された．これまで提案されたステートフルアスペクト言語がそれぞれの意味論を定義しているのに対し，本研究は開発者自身が作成することができるため，ステートフルアスペクト言語設計において，参考となる モデルとなると考えている．

また，この織り込み機構を実現する過程において，実行スレッドを１つしか持たない言語において問題となる Callback 地獄を解決する方法を考案することができ，織り込み機構を応用したライブラリの研究開発も同時に行った．この機構では，従来 Callback の連鎖で記述せざるを得ない非同期処理を同期的に記

述することを可能にしている．この機構の提案やプロトタイプの実装を数回国際会議で発表し，それらをまとめて論文誌に投稿して採択されている．一方，この研究を進めていくと，ループ処理や処理がジャンプする break や continue といった文が存在するとうまく機能しないことも明らかとなった．そこで，プログラム言語本来の機構である Call/CC とアスペクトを組み合わせることで，プログラムの任意の場所で処理を中断し．非同期処理の結果が受け取った後，中断した処理を再実行する機構を考案し，国際会議での発表を行った．この機構の他にも，現在非同期を扱うための仕組みは様々な提案があり，乱立している状況にある．そこで今後の展開として，Call/CC のような機構を利用した方法ではなく，プログラム言語本来の意味論の定義として非同期を扱うプログラム言語の構文，意味論の定義を行って行く．その結果，これまでライブラリレベルで解決していた非同期処理を，プログラム言語そのものの解釈で扱うことができ，プログラム言語の研究に貢献できると考えている．

<引用文献>


[1] G. Kiczales, et al. Aspect oriented programming, In Proceedings of European Conference on Object-Oriented Programming, 1997.

[2] G. Kiczales, et al. An Overview of AspectJ, vol. 2072. LNCS, June 2001.

[3] P. Eugster and K. Jayaram, EventJava: An extension of java for event correlation. In Drossopoulou, S., editor, Proceedings of the 23rd European Conference on Object-oriented Programming (ECOOP 2009), No. 5653 in Lecture Notes in Computer Science 2009.

[4] C. Herzeel, K. Gybels, and P. Costanza, A temporal logic language for context awareness in pointcuts. In Thomas, D., editor, Workshop on Revival of Dynamic Languages, number 4067 in Lecture Notes in Computer Science, 2006.

[5] Allan, C., Avgustinov, P., Christensen, A. S., Hendren, L., Kuzins, S., Lhot ´ak, O., de Moor, O., Sereni, D., Sittampalam, G., and Tibble, J. Adding trace matching with free variables to AspectJ. In Proceedings of OOPSLA, pp. 345-364, ACM SIGPLAN Notices, 40(11), 2005.

[6] G. Kiczales, J. Lamping, C. V. Lopes, C. Maeda, A. Mendhekar, and G. Murphy, Open implementation design guidelines. In Proceedings of the 19th International Conference on Software Engineering, pages 481-490, Boston, Massachusetts, USA. ACM Press, 1997.

[7] C. Dutchyn, D. B. Tucker and S. Krishnamurthi, Semantics and scoping of aspects in higher-order languages. Science of Computer Programming, 63(3):207-239. 2006.

[8]R. Toledo, P. Leger and E. Tanter, AspectScript: Expressive aspects for the Web. In Proceedings of 9th International Conference on Aspect Oriented Software Development pp. 13-24, 2010.


## 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 2 件）


① Hiroaki Fukuda and Paul Leger, "SyncAS: A Virtual Block Approach to Tame Asynchronous Programming", International Journal of Software Engineering and Knowledge Engineering, Vol. 25, No. 5, pp. 887-907, World Scientific Publishing, 2015.

② Paul Leger, Eric Tanter and Hiroaki Fukuda, An expressive stateful aspect language, Science of Computer Programming (SCP), Vol. 102, pp. 108-141, Elsevier, 2015.


〔学会発表〕（計 9 件）


① 諏訪重貴, 福田浩章, 篠埜 功, Liquid - 非同期一階関数による並行計算体系, 情報処理学会第 80 回全国大会論文集, 2018.

② 佐久間隆平, 福田浩章, 非同期処理の記述を支援するライブラリの提案と実装, 第 198 回ソフトウェア工学研究会研究報告, 2018.

③ 諏訪重貴, 福田浩章, 非同期並列プログラミング言語の意味論と実装, 情報処理学会第 79 回全国大会論文集, 2017.

④ Paul Leger and Hiroaki Fukuda, "Sync/CC: Continuations and Aspects to Tame Callback Dependencies on JavaScript Handlers", to appear In Proceeding of the 32nd ACM/SIGAPP Symposium On Applied Computing, 2017.

⑤ Paul Leger and Hiroaki Fukuda "Using continuations and aspects to tame asynchronous programming on the web", In Proceedings of Foundations Of Aspect-Oriented Languages (FOAL) conjunction with Modularity 2016.

⑥ Wiliam Mateus Boeira da Rocha, Hiroaki Fukuda and Paul Leger "Modular Asynchronous Web Programming: Advantages & Challenges", In BICT 2015 Special Track on Modularization for Practical Software Engineering, pp. 442-445,2015.

⑦ Hiroaki Fukuda and Paul Leger. "Proposals for Modular Asynchronous Web Programming: Issues & Challenges", In Proceedings of First workshop on PERvasive WEb Technologies trends and challenges (PRNET) in conjunction with ICWE2015, Lecture Note in Computer Science, 9396, pp. 91-102, 2015.
⑧ Hiroaki Fukuda and Paul Leger, "A Library to modularly control asynchronous executions", In Proceeding of the 30nd ACM/SIGAPP Symposium On Applied Computing, pp. 1648-1650, ACM Digital Library, 2015.
⑨ Takeshi Azegami, Hiroaki Fukuda and Paul Leger "Towards a Virtual Block Approach to Tame Asynchronous Programming", BICT 2014 Special Track on Modularization for Practical Software Engineering, ACM Digital Library, pp. 239-242, 2014.

## ６．研究組織

(1)研究代表者

福田 浩章（FUKUDA, Hiroaki）
芝浦工業大学・工学部情報工学科・准教授
研究者番号：30383946

(4)研究協力者

Paul Leger（Paul Leger）
Universidad Católica del Norte・Associate Professor