科学研究費助成事業　　研究成果報告書

平成　２６　年　　５　月　１６　日現在

機関番号：　１７１０２
研究種目：　基盤研究(B)
研究期間：　2011～2013
課題番号：　２３３０００１０

研究課題名（和文）高信頼ソフトウェアアーキテクチャ構築に関する研究

研究課題名（英文）A Study on Constructing Reliable Software Architecture

研究代表者
鵜林　尚靖（UBAYASHI, Naoyasu）

九州大学・システム情報科学研究科（研究院）・教授

研究者番号：８０３７２７６２

交付決定額（研究期間全体）：（直接経費）　　12,600,000円、（間接経費）　　3,780,000円

研究成果の概要（和文）：アーキテクチャ設計は高信頼ソフトウェアシステムを開発する上で重要な役割を果たす．しかしながら，適切な抽象度をもつ整合性のとれたアーキテクチャを設計すること，その設計内容を正しく実装につなげることは容易ではない．本研究では，このような問題を解決するため，「滑らかな設計抽象化」を提案する．これは設計と実装の間を滑らかに移動した収束点として適切な抽象度が得られるという考え方である．本研究では，滑らかな設計抽象化を実現するため，1)抽象度を指定するためのアーキテクチャインタフェース，2)抽象度を測定するためのメトリクス，3)抽象度を考慮に入れた設計とコードの間のトレーサビリティチェック，を提案する．

研究成果の概要（英文）：Architectural design plays an important role in developing reliable software systems. However, it is not easy to design a consistent architecture having an appropriate abstraction level and implement a program faithful to the design. To deal with this problem, we propose the notion of fluid design abstraction, a design approach in which an appropriate abstraction level can be captured by the convergence of fluid moving between design and implementation. To support fluid design abstraction, we provide the followings: 1) an architectural interface mechanism for specifying abstraction, 2) metrics for measuring an abstraction level, and 3) abstraction-aware traceability check between design and code.

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：　高信頼ソフトウェア開発　ソフトウェアアーキテクチャ　インタフェース　設計抽象化　トレーサビリティ　アーキテクチャ検証

１．研究開始当初の背景

アーキテクチャは高品質なソフトウェアを構築する上で重要な役割を果たす．性能，頑強性，保守性などソフトウェアにおける主要な品質特性はアーキテクチャの良否に大きく依存する．アーキテクチャ設計をいかにうまく行うか，設計した内容をいかにプログラム実装に伝えるかは開発者にとって重要な関心事の一つである．ソフトウェアアーキテクチャ関する研究は 1990 年代以降，アーキテクチャ構築技法，アーキテクチャ記述言語，アーキテクチャパターン，アーキテクチャ検証など，様々な側面からなされてきた．最近では，モデル駆動開発支援ツールの普及により，アーキテクチャ設計のモデリング，設計モデルからのプログラムコードの自動生成が可能になっている．

しかし，開発者の意図を正確に反映し整合性のとれたアーキテクチャを構築すること，そのアーキテクチャを正しく実装につなげることは現在でも容易ではなく依然として重要な研究テーマの一つである．たとえば，Taylor, R. N.らは，ソフトウェア設計における重要研究テーマとして，「アーキテクチャ設計から実装への移行」と「両者の間の頻繁な行き来への適切な支援」を指摘している[1]．

本研究では，このような背景の下，従来にない新たな「高信頼ソフトウェアアーキテクチャ構築手法」の開発に取り組んだ．

２．研究の目的

ソフトウェアアーキテクチャとは，ソフトウェアの基本構造を示すものである．その際，ソフトウェアに求められる機能だけでなく，性能や保守性などの非機能要求についても考慮しなければならない．高信頼なソフトウェアアーキテクチャを構築するには，「アーキテクチャの構造の善し悪しを判断するにはどうしたら良いのか？」「非機能要求はどのようにアーキテクチャに反映したら良いのか？」「設計したアーキテクチャをどうプログラム実装に繋げて行けば良いのか？」などの現実的な諸問題を具体的に解決していく必要がある．

我々は，特にアーキテクチャ設計とプログラム実装の間に横たわる問題の解決を研究の目的とした．その理由は，現在のモデル駆動開発支援ツールは，以下に示す「設計の抽象度」「設計からコードへの洗練」「設計とコードの同期」の３つの側面で十分ではなく，理論的あるいは技術的なブレークスルーが求められていると考えたからである．

課題 1: 設計の抽象度

ソフトウェア開発において抽象化は最重要課題の一つである．Kramer J.は「明快でエレガントな設計やプログラミングができるエンジニアとそうでないエンジニアがいるのは何故であろうか？」と問題提起をした上で，「その答えは抽象化能力にある」と主張している[2]．

設計の抽象度をどのレベルに設定すべきかについては，必ずしも明確な答えがあるわけではない．現状は，開発者の個々の判断あるいはスキルにより，抽象度が設定されていると言ってよい．そのため，様々な問題が現実的に生じている．設計の抽象度が高すぎるとプログラム実装との乖離が大きくなり，設計内容を見ても，プログラムがどのような構造になっているのか理解できない．逆に設計の抽象度が低すぎるとプログラムとの差異が明確でなくなり，設計の意義が薄れてしまう．そのような設計書を読むよりはプログラムコードの方を読んだ方が良い場合が多く，設計として意味を持たない．設計にはソフトウェア構造の全体像とその合理的理由を記載すべきであり，適切な抽象度を持たせるべきである．しかし，「適切な抽象度」とは言っても，それをどのように探し当てれば良いのかは難しい問題である．抽象度を測るメトリクス，メトリクスに基づいて反復的に適切な抽象度を探索するプロセスが必要となる．

課題 2: 設計からコードへの洗練

設計の意図を正確に実装に反映させるのは必ずしも容易ではない．設計に矛盾がないこと，設計から実装への洗練が正しいことが形式に検証できる仕組みが必要となる．

課題 3: 設計とコードの同期

設計とプログラム実装の同期をとるのは容易ではない．設計からプログラムが完全に自動生成されるのであれば同期の問題は発生しないが，現状のモデル駆動開発技術では難しい．現在のモデル駆動開発支援ツールでは，ほとんどの場合，自動生成にはコードに近い設計記述が求められ，上記の「設計は適切な抽象度を持つべき」という条件を満たさない．また，設計が変更になった場合に対応するプログラムが修正され，逆にプログラムが変更になれば設計にもそれが反映される双方向のトレーサビリティが求められる．何らかの形で抽象度を設定し，その設定したレベルでプログラム実装とのトレーサビリティが保証される仕組みが必要となる．

ここで示した３つの問題は，突き詰めると，「設計とプログラム実装の間の境界が曖昧」という本質的な要因に起因する．すなわち，設計として何を記述し，何をプログラム実装に任せるのかについての合意とそれを支援するための技術が未だ確立されていないことが本質的な問題である．

---

[1] Taylor, R. N. and Hoek, A.: Software Design and Architecture --The once and future focus of software engineering, *Proceedings of 2007 Future of Software*

[2] Kramer, J.: Is Abstraction the Key to Computing?, *Communications of the ACM*, vol.50 issue 4, pp.36-42, 2007.

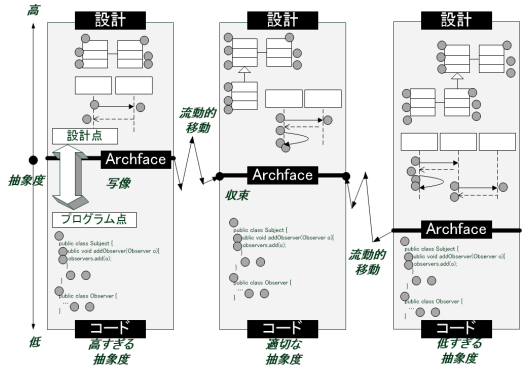

図 1 流動的反復による滑らかな抽象化

３．研究の方法

本研究では，2 節で掲げた 3 つの問題を解決するための手段の一つとして，「滑らかな設計抽象化（Fluid Design Abstraction)」という概念を提案した．これは，設計とプログラム実装の間を流動的に移動（Fluidly Moving）し，その収束の結果として適切な抽象度が獲得される，という考えである．抽象化の視点からのリファクタリングと言える．この場合のリファクタリングは，従来のコードあるいは設計モデルのみのリファクタリングとは異なり，それらを含め一つのソフトウェアを構成する成果物，すなわち設計モデルおよびコードにまたがったリファクタリングである．「滑らかな設計抽象化」という用語を使用した理由は，設計および実装フェーズにおける抽象化は絶対的に固定したものではなく，「設計とコードの間の流動的な移動と共に両者間の最適な組み合わせを柔軟に探索すべき，すなわち抽象度は連続かつ滑らかに変化させていくべきだ」という考えに基づいている．

滑らかな設計抽象化を実現するための機構は， A)抽象度を設定するための「設計モデルとコード間」のインタフェース機構，B)抽象度を測定するためのメトリクス，C)設定した抽象度を保持しつつ設計とコードのトレーサビリティをチェックするための機構，から構成される．2 節の課題 1 は A)と B)により解決される．課題 2 と課題 3 は C)により解決される．インタフェース機構として，我々が提唱する *Archface* を適用した．*Archface*（Architecture と Interface の造語）とは，設計モデルとコードの間に存在するインタフェース機構である．図 1 は滑らかな設計抽象化の概念を図示したものである．

４．研究成果

(1) 滑らかな設計抽象化のための機構

①インタフェース機構 *Archface*

Archface とは，ソフトウェアアーキテクチャが記述可能なインタフェース機構であり，設計モデルとコードの間で共有すべきアーキテクチャ情報（クラスやメソッドの定義，メッセージ送受信など）とそれらの間の制約（継承，メッセージ送受信の順序など）を公開する．これらの情報のことをアーキテクチャ点と呼ぶ．設計モデルは *Archface* で宣言されたアーキテクチャ点を設計点（UML モデル上のメッセージ送受信箇所など）としてモデル化し，一方，コードは同じアーキテクチャ点をプログラム点(メソッド呼び出しなど)として実装する．設計モデルとコードの双方が同じ *Archface* をそれぞれモデル化，実装することにより，両者はアーキテクチャ点の観点でトレーサビリティが保証される．

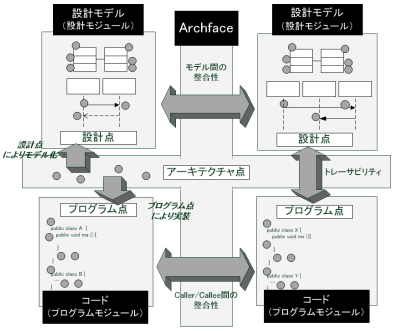

図 2 インタフェース機構 *Archface*

図 2 は *Archface* の概念を図示したものである．図 2 では，設計モデルを設計モジュール，クラス定義等のコードをプログラムモジュールと位置づけている．設計モデルもコード同様，一つのソフトウェアシステムを構成するモジュールであり，両者間の抽象度を見直すことは各々のモジュールの役割を再考するリファクタリングであることが分かる．どのアーキテクチャ点を *Archface* から公開するかにより，設計とコードの間の抽象度が決まる．開発者は *Archface* を通じて抽象度を設定すると共にその抽象度で設計とコード間の同期をとることが可能となる．

② 抽象化指標

抽象化の度合いを示すメトリクスは以下の式で与えられる．

1 - アーキテクチャ点の数 / プログラム点の数

アーキテクチャ点の数がプログラム点の数と一致する場合，すなわち，設計モデルの内容がプログラムコードと同等の場合，指標値は 0 となる．アーキテクチャ点の数が 0 の場合，すなわち，設計モデルが存在しない場合，指標値は 1 となる．通常，指標値はこの間の数値をとる．抽象化指標は開発者が適切な抽象度を見つける際の参考として活用できる．

③ トレーサビリティチェック

図 3 は，*Archface* を介した設計とコードのトレーサビリティチェックの流れを図示したものである．トレーサビリティチェックは，設計モデル上の設計点がアーキテクチャ点を通じてコード上のプログラム点に正しく写像されているか検証することにより行われる．このチェックは設計モジュールとして

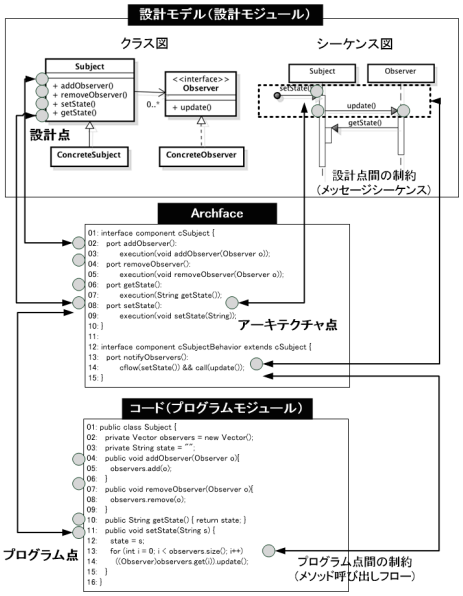

図 3 設計とコード間のトレーサビリティチェック

の設計モデルとプログラムモジュールとしてのコードの双方が同じ *Archface* を満たしているかを検証することにより行われる．この検証は，設計モデル，コード，*Archface* を論理式に変換し充足可能性を判定することにより行われる．充足可能性判定には SMT（Satisfiability Modulo Theories）ソルバーを用いている．

(2) 抽象化洗練プロセス

設計とコードの間でトレーサビリティがとれなくなった場合，両者の不整合を解消する方法として，1) 設計の方が正しいと判断しコードを設計モデルに合わせる，2) コードの方が正しいと判断し設計モデルをコードに合わせる，3) 抽象レベルの設定に問題があると判断し設計モデルの抽象度を変更する，の三つのアプローチがある．1)と 2)の場合は誤りを正すのみなので修正内容は比較的明瞭である．しかし，3)は抽象度の変更であり，必ずしも正解はない．どの値であれば良いのかは，設計とコードの間のバランスにより決定せざるを得ない．基本的に開発者の抽象化能力に負うところが大きい．

明示的にどう抽象化すべきかを指し示すことは難しいにしても，何らかの方法で開発者が適切な抽象度を発見する支援は必要である．反復的な抽象化プロセスが基本となるが，何をトリガーに反復すればよいのか，何をもって収束したと判定をすれば良いのか，については難しい問題である．

この問題を緩和するため，本研究では，モデル検査における CEGAR（Counter Example Guided Abstraction Refinement)にヒントを得た設計抽象化のためのプロセスを提案した．モデル検査は網羅的探索を行うので状態数の爆発が容易に生じてしまう．この問題を回避するには，検査対象を抽象化し状態数を減らす必要がある．CEGAR はモデル検査器から出力される反例をトリガーに抽象化の洗練を行うためのプロセスである．滑らかな設計抽象化では，設計とコードの間でトレーサビリティがとれなくなった場合，エラーが出力される．エラーを解消する手段として前述の三つの選択肢があるが，設計またはコードに明らかな誤りがない限り，3)の抽象化洗練を行うことになる．すなわち，トレーサビリティチェックから出力されるエラーを抽象化の見直しをするためのトリガーとして活用できる．

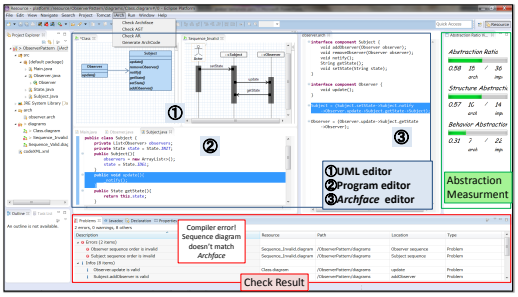

図 4 統合開発環境 *iArch*

(3) コンテキストアウェアシステムへの適用

我々のアプローチの有効性を評価するため，具体的なドメインを設定し，ドメイン特化型のアーキテクチャ点が定義できる仕組みについて研究した．対象ドメインはコンテキストアウェアシステムとした．現在最も注目されているアプリケーション分野の一つであることがその選定理由である．

今回，コンテキストアウェアシステムを対象としたアーキテクチャ記述言語 UML4COP と検証機構 RV4COP を開発した．コンテキストアウェアシステムのアーキテクチャ構造を特徴付けるのに，我々はコンテキスト指向プログラミング（COP: Context-Oriented Programming）におけるイベント群に着目し，これらをアーキテクチャ点として定義した．また，トレーサビリティに関しては，コンテキストアウェアシステムにおける動的なアーキテクチャ変更を考慮し，ランタイム検証（RV: Runtime Verification）の機構を導入した．設計とコードの関係だけでなく実行トレースとの対応についても検証が必要なためである．

(4) 統合開発環境 *iArch*

本研究では，「滑らかな設計抽象化」を支援するため，図 4 に示す統合開発環境（プロトタイプ）*iArch* を開発した．*iArch* は，1)モデルおよびプログラムエディタ，2)設計モデルからの *Archface* 生成機能，3)*Archface* チェッカ（モデルおよびコードに対する型検査），4)抽象度の表示機能をサポートする．

(5) 研究成果の公開

本研究の成果は 5 節に示すように，論文誌や国際会議等で発表した．また，*iArch* につい

ては本研究期間終了後も改良を重ね，実用化に向けた努力をしていく予定である．将来的には，多くの人が利用できるように我々の研究室のホームページから公開したい．

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 19 件）

① Di Ai, Naoyasu Ubayashi, Peiyuan Li, Daisuke Yamamoto, Yu Ning Li, Shintaro Hosoai, and Yasutaka Kamei, iArch: An IDE for Supporting Fluid Abstraction, Proceedings of Companion Publication of the 13th International Conference on Modularity (Modularity'14), 査読有, 2014, pp.13-16

② Di Ai, Naoyasu Ubayashi, Peiyuan Li, Shintaro Hosoai, and Yasutaka Kamei, iArch: An IDE for Supporting Abstraction-aware Design Traceability, Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2014), 査読有, 2014, pp.442-447

③ Naoyasu Ubayashi, Shin Nakajima, and Masayuki Hirayama, Context-dependent Product Line Engineering with Lightweight Formal Approaches (SPLC 2010 Revised Selected Paper), Science of Computer Programming, 78(12), 査読有, 2013, pp.2331-2346

④ 鵜林 尚靖, コンテキストアウェアアプリケーション --ポスト PC 時代の共通問題--, 情報処理, vol.54, no.9, 査読無, 2013, pp.894-897

⑤ Naoyasu Ubayashi, Ai Di, and Yasutaka Kamei, Archface4COP: Architectural Interface for Context-Oriented Programming, 5th Workshop on Context-Oriented Programming (COP 2013) (Workshop at ECOOP 2013), 査読有, 2013, ACM Digital Library

⑥ Naoyasu Ubayashi and Yasutaka Kamei, Design Module: A Modularity Vision Beyond Code --Not Only Program Code But Also a Design Model Is a Module--, 5th International Workshop on Modeling in Software Engineering (MiSE 2013) (Workshop at ICSE 2013), 査読有, 2013, pp.44-50

⑦ Naoyasu Ubayashi and Yasutaka Kamei, UML-based Design and Verification Method for Developing Dependable Context-aware Systems, Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2013), 査読有, 2013, pp.89-94

⑧ Naoyasu Ubayashi and Yasutaka Kamei, UML4COP: UML-based DSML for Context-Aware Systems, 12th Workshop on Domain-Specific Modeling (DSM 2012) (Workshop at SPLASH 2012), 査読有, 2012, pp.33-38

⑨ Naoyasu Ubayashi and Yasutaka Kamei, Archpoint and Archmapping --Bidirectional Traceability between Design and Code--, Korea-Japan Joint Workshop on ICT, 査読有, 2012, paper no.14

⑩ 内尾 静, 鵜林 尚靖, 亀井 靖高, SMT ソルバーを用いたコンテキスト指向プログラミングのためのデバッグ支援[レター論文], 日本ソフトウェア科学会誌 コンピュータソフトウェア, vol.29, no.3, 査読有, 2012, pp.108-114

⑪ Naoyasu Ubayashi and Yasutaka Kamei, Verifiable Architectural Interface for Supporting Model-Driven Development with Adequate Abstraction Level, 4th International Workshop on Modeling in Software Engineering (MiSE 2012) (Workshop at ICSE 2012), 査読有, 2012, pp.15-21

⑫ Naoyasu Ubayashi and Yasutaka Kamei, An Extensible Aspect-oriented Modeling Environment for Constructing Domain-Specific Languages, IEICE Transactions on Information and Systems, vol E95-D no.4, 査読有, 2012, pp.942-958

⑬ Naoyasu Ubayashi and Yasutaka Kamei, Architectural Point Mapping for Design Traceability, 11th Workshop on Foundations of Aspect-Oriented Languages (FOAL 2012) (Workshop at AOSD 2012), 査読有, 2012, pp.39-43

⑭ Naoyasu Ubayashi and Yasutaka Kamei, Stepwise Context Boundary Exploration Using Guide Words, CAiSE Forum 2011, LNBIP 107 proceedings, 査読有, 2012, pp.218-233

⑮ 塩塚 大, 鵜林 尚靖, 亀井 靖高, dcNavi: デバッグを支援する関心事指向推薦システム, 情報処理学会論文誌, vol.53, no.2, 査読有, 2012, pp.631-643

⑯ Naoyasu Ubayashi, Yasutaka Kamei, Masayuki Hirayama, and Tetsuo Tamai, A Context Analysis Method for Embedded Systems --Exploring a Requirement Boundary between a System and Its Context--, Proceedings of the 19th IEEE International Requirements Engineering Conference (RE 2011),

IEEE Computer Society, 査読有, 2011, pp.143-152

⑰ Shizuka Uchio, Naoyasu Ubayashi, and Yasutaka Kamei, CJAdviser: SMT-based Debugging Support for ContextJ*, 3rd Workshop on Context-Oriented Programming (COP 2011) (Workshop at ECOOP 2011), 査読有, 2011, ACM Digital Library

⑱ Masaru Shiozuka, Naoyasu Ubayashi, and Yasutaka Kamei, Debug Concern Navigator, Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), 査読有, 2011, pp.197-202

⑲ Naoyasu Ubayashi and Yasutaka Kamei, Stepwise Context Boundary Exploration Using Guide Words, Pre-proceedings of the 23rd International Conference on Advanced Information Systems Engineering (CAiSE 2011 Forum), 査読有, 2011, visionary paper no.1

〔学会発表〕(計 7 件)

① 艾 迪, 李 沛源, 細合 晋太郎, 亀井 靖高, 鵜林 尚靖, iArch: 滑らかな設計抽象化を支援する IDE [ライブ論文], 日本ソフトウェア科学会 第 20 回ソフトウェア工学の基礎ワークショップ (FOSE 2013), 2013 年 11 月 28 日～2013 年 11 月 30 日, 加賀

② 鵜林 尚靖, モデル駆動開発とドメイン特化言語, 情報処理学会 組込みソフトウェアシンポジウム 2013 (ESS2013) チュートリアル, 2013 年 10 月 16 日, 東京

③ 鵜林 尚靖, 艾 迪, 細合 晋太郎, 亀井 靖高, 滑らかな設計抽象化, 電子情報通信学会ソフトウェアサイエンス研究会, 2013 年 7 月 25 日～2013 年 7 月 26 日, 札幌

④ 鵜林 尚靖, 亀井 靖高, アーキテクチャ点写像による設計・コード間の双方向追跡, 電子情報通信学会ソフトウェアサイエンス研究会, 2012 年 1 月 26 日～2012 年 1 月 27 日, 高知

⑤ 内尾 静, 鵜林 尚靖, 亀井 靖高, SMT ベースの COP デバッグ支援, 日本ソフトウェア科学会 第 18 回ソフトウェア工学の基礎ワークショップ (FOSE 2011), 2011 年 11 月 24 日～2011 年 11 月 26 日, 浅虫温泉

⑥ 鵜林 尚靖, 世界を目指す論文の書き方, 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2011 (SES 2011) チュートリアル, 2011 年 9 月 13 日, 東京女子大学

⑦ 塩塚 大, 鵜林 尚靖, 亀井 靖高, オープンソースリポジトリのバグ修正履歴を再利用したデバッグ推薦の評価実験 [ショート論文], 情報処理学会 ソフトウェアエンジニアリングシンポジウム 2011 (SES 2011), 2011 年 9 月 12 日～2011 年 9 月 14 日, 東京女子大学

〔図書〕(計 3 件)

① Hidehiko Masuhara, Shigeru Chiba, Naoyasu Ubayashi (Eds.), ACM, Aspect-Oriented Software Development, AOSD'13, Fukuoka, Japan, March 24-29, 2013, 220

② Hidehiko Masuhara, Shigeru Chiba, Naoyasu Ubayashi (Eds.), ACM, Aspect-Oriented Software Development, AOSD'13, Companion Volume, Fukuoka, Japan, March 24-29, 2013, 44

③ 鵜林 尚靖, 亀井 靖高 (編集), 近代科学社, ソフトウェア工学の基礎 XIX: FOSE2012 (レクチャーノート・ソフトウェア学), 2013, 248

〔その他〕
ホームページ等
http://posl.ait.kyushu-u.ac.jp/
(POSL は研究室名. Principles Of Software Languages の略)

６．研究組織

(1)研究代表者
鵜林 尚靖 (UBAYASHI, Naoyasu)
九州大学・
大学院システム情報科学研究院・教授
研究者番号： ８０３７２７６２

(2)研究分担者
福田 晃 (FUKUDA, Akira)
九州大学・
大学院システム情報科学研究院・教授
研究者番号： ８０１６５２８２

久住 憲嗣 (HISAZUMI, Kenji)
九州大学・
システムＬＳＩ研究センター・准教授
研究者番号： １０３８０６８

亀井 靖高 (KAMEI, Yasutaka)
九州大学・
大学院システム情報科学研究院・助教
研究者番号： １０６１０２２２

(3)連携研究者
中島 震 (NAKAJIMA, Shin)
国立情報学研究所・
アーキテクチャ科学研究系・教授
研究者番号： ６０３５０２１１