確率的プログラムの AOP による生成

中村 隼也　渡辺 啓介　佐藤 亮介　鵜林 尚靖　亀井 靖高

ソフトウェア工学において，不確かさを包容したソフトウェア開発は重要な研究課題の 1 つである．本論文では，Known-Unknowns（既知の未知）を記述することのできるインターフェース機構 *Archface-U* に確率を導入し，以下を可能とする: 1) 確率的な振る舞いをインターフェースとして記述; 2) インターフェースレベルで確率的モデル検査を実行; 3) 仕様としてのインターフェースからコードへのリファイメントをプログラム変換により実現（確率を含まないコードをインターフェース記述を元に確率を考慮したコードに自動変換）．これにより，モデル検査によって動作保証された確率的プログラムの開発が容易となる．

1　はじめに

ソフトウェア開発の様々な工程で発生する「不確かさ」に我々は対処していかなければならない.「不確かさ」には，その不確かさの存在に誰も気が付いていない状態である「未知の未知」と，その不確かさの存在に気が付いているがどう対処していいかがわからない「既知の未知」の 2 つの種類がある.「未知の未知」に対処することは現状では難しく，ここでは後者の「既知の未知」に焦点を当てて，以降の不確かさは全て「既知の未知」とする．

不確かさの性質の 1 つに確率的な不確かさがある．これは，システムや環境が確率的な振る舞いをすることなどにより発生する不確かさである．近年確率的な挙動で表される不確かさを扱うことのできる言語やツールが注目されている．その例として確率的プログラミングや確率的モデル検査器 PRISM [6] や PAT [8] などが挙げられる．また，Amine M. らは Event-B [1] の非決定性を確率的な選択で置き換えた Probabilistic Even-B を提案している [4].

しかし，確率を考慮したプログラムを作成し，コードレベルで振る舞いの正しさを検証することは難しい．その例として，コードレベルでのモデル検査には状態爆発の問題があることが挙げられる．このような問題に対処するために，不確かさを記述することのできるインターフェース機構 *Archface-U* [5] が確率的な不確かさを記述できるように拡張し，インターフェースレベルでモデル検査を行うことで振る舞いの正しさを検証する手法を提案する．この手法により，モデル検査を行う対象がインターフェースになることでコードレベルのモデル検査より状態爆発する可能性が小さくなる．そして，インターフェースのモデル検査により保証された性質は，インターフェースに従って実装されているコードに対しても保証されるため，コードの動作を保証した確率的プログラムの開発が容易になる．本論文では確率的な不確かさをインターフェースに記述し，それを元に確率的プログラミングを生成する手法について述べる．具体的には，*Archface-U* レベルで確率的モデル検査を行い，振る舞いの正しさが保障されれば，*Archface-U* に記述された確率を元に，確率を含まないプログラムコードから確率を含

Probabilistic Programming by AOP.

Shunya Nakamura, Keisuke Watanabe, 九州大学大学院システム情報科学府, Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University.

Ryosuke Sato, Naoyasu Ubayashi, Yasutaka Kamei, 九州大学大学院システム情報科学研究院, Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University.

むプログラムコードへと自動変換する．本論文では，2 章で関連研究について述べる．3 章では不確かさを包容したインターフェース機構 *Archface-U* を紹介する．4 章では *Archface-U* の型検査と確率モデル検査による振る舞いの検証について述べる．5 章で確率的プログラムの生成について述べ，6 章ではまとめと今後の研究の課題について述べる．

2 関連研究

2.1 確率的モデル検査器 LTSA-PCA

モデル検査とは，検査対象のシステムの振る舞いが，検査したい性質 (プロパティ) を満たすかどうかを判定するものである．LTSA-PCA [9] は確率的振る舞いをするシステムを扱えるようにモデル検査器 LTSA(Labelled Transition System Analyser) [7] を拡張したものである．LTSA-PCA においてモデルの記述は Probabilistic FSP(Finite State Process) という記法を用い，これは LTSA でモデルを記述するための記法である FSP を確率が表現できるように拡張したものである．LTSA-PCA は Probabilistic Component Automata（PCA) [10] としてモデル化されたコンポーネントベースのシステムの確率的な振る舞いをグラフィカルに表示する機能や，システムの故障解析などの機能を有する．モデル検査器 PRISM はマルコフ連鎖 (DTMC) を状態遷移系として扱い，モデルが検査したい性質を満たす確率を求めることができる [6]．LTSA-PCA でシステムが故障する確率などを求める場合は，PCA を DTMC に自動的に変換し，PRISM のようなツールを用いて計算する．本研究では確率を記述したインターフェースのモデル検査のために LTSA-PCA を利用する．これは，インターフェースでのメソッドの振る舞いは，LTSA でモデルを表現するための記法である FSP(Finite State Process) に準拠した文法で記述するためである．

2.2 Probabilistic Event-B

Event-B [1] は要求仕様から段階的にリファインメントを行うことで，詳細なモデルを構成していく手法である．リファインメント毎にそのリファインメントに対する証明責務を証明することによって，抽象モデルに対する正当性を保証する．Probabilistic Event-B [4] は Event-B の全ての非決定的な選択を確率的な選択に置き換えたものである．Event-B の非決定的な選択は，実行可能なイベントの選択，パラメータ値の選択，非決定的な割り当ての 3 つであり，それぞれに対して重みの付与や離散分布などによって非決定的選択を確率的選択に置き換える．Probabilistic Event-B はリファインメントをする際に Event-B の通常の証明責務に加えて，確率に関する証明責務も証明する必要がある．現時点では実行可能なコードを生成する直前のリファインメントにおいてのみ確率を与えることが可能であり，それ以前のリファインメントは非決定的選択は確率に置き換えることができない．確率的な振る舞いをするシステムを対象にしている点では本手法と一致しているが，Probabilistic Event-B は仕様と確率付きプログラムを作成しなければならないが，本手法では確率を記述したインターフェースとプログラムコードがあれば自動で確率付きプログラムコードを生成できるという点で異なる．

```
01: interface component Main {
02:   void actionPerformed(ActionEvent e);
03: }
04:
05: interface component StudentController {
06:   void filterStudent(JTable table);
07:   void colorStudent(JTable table);
08: }
09:
10: interface connector cStudent {}
11:
12: uncertain connector ucStudent extends cStudent {
13:  Student = (Main.actionPerformed ->
14:      {StudentController.filterStudent,
15:       StudentController.colorStudent} -> Main);
16: }
```

図 1　e ラーニングシステムにおける *Archface-U* の記述

3 不確かさを包容するインターフェース機構 *Archface-U*

本章では不確かさを記述可能なインターフェース機構 *Archface-U* について説明する．

3.1 *Archface-U*

Archface-U は，アーキテクチャ設計と実装間の不一致を無くすためのインターフェース機構である．*Archface-U* はプログラム全体の制約を記述することのできるインターフェースであり，*Archface-U* に記述された内容を実装することを強制する．*Archface-U* は不確かさを記述することが可能であり，図 1 にその記述例を示す．この例は，ある大学で用いる e ラーニングシステムにおける受講者一覧機能を記述したものである．いま，受講者のみを一覧表示すべきか，それとも全学生をリスト表示した上で受講者のみを色付表示すべきかが決定できない状況を想定して欲しい．すなわち，どちらの表示方式を採用すべきか判断できないという不確かさが存在する．

Archface-U は *Component-and-Connector* アーキテクチャ [3] に基づいており，*Component* と *Connector* の 2 つのインターフェースからなる．*Component* インターフェースは Java のインターフェースと同様のものであり，実装しなければならないクラスと，そのクラスが持たなければならないメソッドを型，引数を含めて定義する．図 1 の例では，Main クラスは actionPerformed メソッドを，StudentController クラスは filterStudent メソッドと colorStudent メソッドを実装しなければならないという制約を記述してある．また，*Connector* インターフェースでは，FSP に準拠した構文で *Component* 間のインタラクション（メソッド呼び出し関係）を記述する．不確かさはキーワード uncertain で定義することができる．*uncertain Component* や *uncertain Connector* は，不確かさを含まない *Component* インターフェースや *Connector* インターフェースのサブインターフェースであり，記述することのできる不確かさは以下の 2 種類である．
(1) 複数の実装の候補があるが，どの候補を採用するかが分からない．
(2) ある実装について，それを採用するか否かが分からない．
前者は *Alternative* と呼び，複数のメソッドの候補を {} で囲んで記述する．後者は *Optional* と呼び，メソッドを [ ] で囲んで記述する．したがって，図 1 の *uncertain Connector* では actionPerformed において filterStudent と colorStudent のどちらが呼び出されるかが不確かであることを示している．*Alternative* な不確かさの場合は，少なくともどれか 1 つが実装されていなければ型検査によりコンパイルエラーとなる．*Optional* な不確かさの場合は，*Optional* の性質ゆえにそのメソッドを実装しても実装してなくても型検査でコンパイルエラーにはならない．

3.2 重みによる確率記述

これまでの不確かさの記述は，不確かなメソッドが存在していることを示すものであり，実際に呼び出される，または実装されるメソッドはいずれ開発者によって決まるものであった．以降は不確かなメソッドが確率によって振る舞いが振り分けられる，すなわち，設定された確率に従って呼び出されるメソッドが変化する状況についても考える．

Archface-U では，不確かなメソッドが実行される確率を重みによって表現する [12]．図 1 の e ラーニングシステムの例では，受講している生徒の一覧を表示する 2 つのメソッド候補 filterStudent および colorStudent に対しては *Alternative*，すなわち，どちらを最終的に採用するか分からないという不確かさが存在していることを表している．

複数の実装案を比較評価するために，ユーザ毎にそれぞれの実装案を一定の確率に従って切り替えることで，ユーザの好みや傾向等を統計的に検証する手法を A/B テストという [2]．図 1 の 2 つの *Alternative* なメソッドで A/B テストを実施することで，どちらの実装の方がより使いやすいかをテストすることを考える．テストを行うため開発者は両方のメソッドを作成し，それぞれのメソッドに重み w_1，w_2(正の有理数) を設定する．重みを設定したことで，actionPerformed の実行後に filterStudent，または colorStudent が呼び出された場合，確率 $\frac{w_1}{w_1+w_2}$ で filterStudent の処理が，確率 $\frac{w_2}{w_1+w_2}$ で colorStudent の処理が選択され実行される．*Optional* な不確かさの場合は，当該のメソッド m と空の処理 ϕ の *Alternative* とみなし，重み w_m，w_ϕ を設定する．

重みの設定は，*Archface-U* に対するアノテーションを用いる．表 2 に示した 4 種類のアノテーション

表 1　重みと振り分けの確率

インターフェースの種類	可変性の種類	記述	重み	任意の m の実行時の処理
uncertain component	*alternative*	$\{m_1, m_2, ...\}$	$w_1, w_2, ...$	$w_k / \sum_i w_i$ の確率で m_k を実行
uncertain component	*optional*	$[m_1]$	w_1, w_ϕ	$\frac{w_1}{w_1+w_\phi}$ の確率で m_1 を実行，$\frac{w_\phi}{w_1+w_\phi}$ の確率で空の処理を実行
uncertain connector	*alternative*	l -> $\{m_1, m_2, ...\}$ -> n	$m_1, m_2, ...$	もし m が l 中で呼び出されたならば $w_k / \sum_i w_i$ の確率で m_k を実行
uncertain connector	*optional*	l -> $[m_1]$ -> n	w_1, w_ϕ	もし m が l 中で呼び出されたならば $\frac{w_1}{w_1+w_\phi}$ の確率で m を実行，$\frac{w_\phi}{w_1+w_\phi}$ の確率で空の処理を実行

表 2　重みを設定するアノテーション

アノテーション	重みの設定
@ExecForce	可変性において挙げられているメソッドの中で，このアノテーションが指定されたメソッドには重み 1 を，それ以外には重み 0 を設定する．
@ExecIgnore	可変性において挙げられているメソッドの中で，このアノテーションが指定されたメソッドには重み 0 を，それ以外には重み 1 を設定する．
@ExecRatio(double p)	このアノテーションが指定されたメソッドに重み p を設定する．また，可変性においてこのアノテーションが指定されている全てのメソッドの p の合計が p_{sum}，指定されていないメソッドの個数が n であるとき，$1 - p_{sum} > 0$ ならば指定されていないメソッド全てに重み $1 - p_{sum}$ を，$1 - p_{sum} \leq 0$ ならば重み 0 を設定する．
@ExecWeight(double w)	このアノテーションが指定されたメソッドに重み w を設定する．可変性においてこのアノテーションが指定されていないメソッドがあれば重み 0 を設定する．

を使い分けることで，開発者は単に重みを記述するだけではなく，振り分けの割合を直感的に設定したり把握したりすることが可能になる．これらのアノテーションは実行時の振る舞いに影響を与えるため，Exec アノテーションと呼ぶ．Exec アノテーションは，*Archface-U* の *Component* インターフェースのメソッドに対して記述する．*Connector* インターフェースの重みの設定は実装上の問題でまだ実現しておらず，今後サポートしていく予定である．そのため，現時点では *Connector* インターフェースに記述されたメソッドの振る舞いの確率は，*Component* インターフェースで各メソッドに付与された重みを参照することで，

```
01: interface component StudentController {
02:   @ExecWeight(5)
03:   void filterStudent(JTable table);
04:   @ExecWeight(1)
05:   void colorStudent(JTable table);
06: }
```

図 2　e ラーニングシステムにおける *Archface-U* のアノテーションの記述 (図 1 からの変更箇所抜粋)

間接的に設定できるようにしている．前述の e ラーニングシステムの例でアノテーションを指定すると図 2 のようになる．この場合は filterStudent に重み 5 を，colorStudent に重み 1 を設定したことになる．

4 型検査と確率モデル検査による振る舞い検証

まず，現在リリースしている統合開発環境 *iArch-U* [11]（確率を含まない *Archface-U* をサポート）について述べ，その後，確率を含む場合の検証について述べる．

4.1 *iArch-U*

iArch-U は *Archface-U* を用いた開発を支援する統合開発環境であり，不確かさを含む *UML* モデリング，不確かさのマネジメント，*Java* プログラミング，および検証やテスト等の機能を提供する．型検査は *iArch-U* のコンパイラにより行われる．型検査では，*Arcface-U* で記述された振る舞いと *Java* コードの振る舞いにシミュレーション関係があるか否かを検査する．シミュレーション関係が無い場合はコンパイルエラーとなり，開発者は *Archface-U* の記述通りに *Java* コードが実装されるように修正しなければならない．また，型検査だけでなく振る舞いの検証の機能もあり，*Archface-U* の *Connector* の記述を *FSP* に変換し，*LTSA* によりモデル検査を行う．もし，モデル検査で問題が発生しなかったら，型検査をパスした *Java* コードの振る舞いも *Archface-U* の観点では保証される．コードレベルでモデル検査するのではなくインターフェースレベルでモデル検査することで，状態爆発の問題は緩和される．

4.2 確率を含む場合の検証

Archface-U で確率を扱えるように拡張するにあたり，モデル検査器として *LTSA* を確率が扱えるように拡張した *LTSA-PCA* を利用する．また，*Java* コードでは確率を扱える構文を持たないため，*Archfacre-U* の記述からアスペクトを自動で生成し，それを *Java* コードとウィービングする．これにより，確率に基づいて，メソッド実行の振り分けを行うことが可能となった．*Archface-U* に記述された確率的な振る舞いを持つコードが自動で生成されるため，もし，*LTSA-PCA* で正しさが保証されていれば，生成されたコードの振る舞いの正しさも保証される．*Component* インターフェースに記述されたメソッドの確率は，*Connector* インターフェースに記述されたメソッド振る舞いにおいても反映される．この確率付きの *Connector* インターフェースの記述を *Probabilistic FSP(P-FSP)* に変換することで *LTSA-PCA* による検証が可能になる．*LTSA-PCA* の検証によりあるプロパティが満たされる確率が計算され，計算結果が好ましくなかった場合は，開発者は設計の見直しを行うなどして開発を進めていく．

5 確率的プログラムの生成

Archface-U を仕様とみなすと，プログラムコードへのリファイメントはウィービングによって自動化できる．ここでは，*AspectJ* による実装方法について説明する．

図 *2* の *Archface-U* の記述に対して生成されるアスペクトは図 *3* のようになる．図 *3* の *4* 行目から *9* 行目までは処理が置き換わる条件が記述してあり，この条件に合致するとき，元の処理は実行されず *13* 行目から *21* 行目までの処理に置き換わることになる．この例では，*filterStudent* メソッドが *actionPerformed* メソッドにおいて呼び出されることが条件であり，この条件に合致するとき，元々実行されるはずだった *filterStudent* メソッドの処理は実行されず，代わりに *13* 行目から *21* 行目までの処理が実行されることになる．*colorStudent* メソッドについても同様なアスペクトが自動生成される．

13 行目の *weightSwitch* は，n 番目の引数が a_n，引数の和が a_{sum} であるとき，a_n/a_{sum} の確率で n を返すメソッドである．引数は *Archface-U* のアノテーションで設定した重みであり，ここでは *filterStudent*, *colorStudent* のそれぞれに設定した重みが引数となっている．つまり，*13* 行目の処理では設定された重みの比に従って，複数のメソッド候補のうち *1* つを確率的に選択することになる．*Optional* の場合は，当該メソッドか空のメソッドのどちらかを確率に従って選択する処理になる．

最後に，確率的に選択されたメソッドが呼び出されることで一連の振り分け処理が完了する．*15* 行目は *filterStudent* メソッドが選択された処理であり，

```
01: @Aspect
02: public class U {
03:   @Around("call(void com.example.student.StudentController.filterStudent(JTable)) &&
04:           args(table) && !cflow(adviceexecution()) &&
05:           withincode(void com.example.student.Main.actionPerformed(ActionEvent))")
06:   public Object aroundWeightFilter(ProceedingJoinPoint __pjp, JTable table)
07:       throws Throwable {
08:     switch (weightSwitch(5, 1)) {
09:       case 0: return __pjp.proceed();
10:       case 1: com.example.student.StudentController.colorStudent(table); return null;
11:     }
12:     throw new IllegalStateException();
13:   }
14: }
```

図 3　確率を導入するためのアスペクトのコード

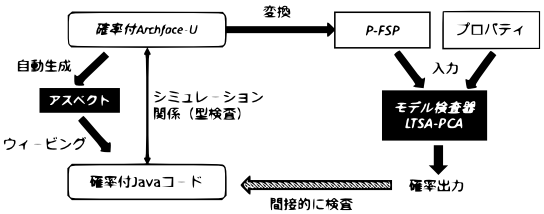

図 4　確率プログラムの自動生成の流れ

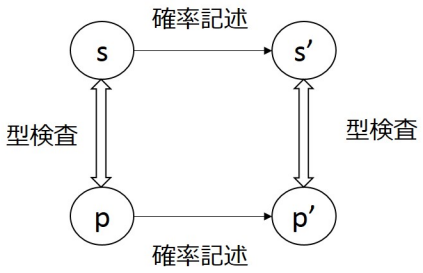

図 5　リファインメント

これは元々の振る舞いと同じである．17 行目は *colorStudent* メソッドが選択された処理であり，これは元々の振る舞いとは別の処理に置き換わることを表す．以上の一連の流れを通すことで，重みを設定した *Archface-U* を元に確率に従って処理が置き換わるプログラムを自動で生成することができる．*weightSwitch* メソッド内では *java.util.Random* クラスにより取得した疑似乱数を用いるが，乱数ジェネレータのシード値，および乱数ジェネレータを振り分けの度に初期化するかどうかを設定することができる．

5.1　プログラム生成の正しさ

本節では，本手法の正しさを示す準備として，部分インターフェース関係やプログラムの模倣関係の表記を導入し，本手法の正しさを表す命題を述べる．これらの形式的な定義，および，命題の証明は今後の課題である．

インターフェースおよびプログラムのメタ変数として，s および p をそれぞれ用いる．あるインターフェース s に対して，プログラム p がそのインターフェースを満たすことを $\models p : s$ と書く．この表記を用いると，本手法の枠組みは次のように述べることができる．本手法は，$\models p : s$ となる s，p が与えられたとき，s に確率を付与したインターフェース s' に対して，$\models p' : s'$ を満たすプログラム p' を *AOP* により自動生成する手法である．このとき，s' の正しさをモデル検査によって確かめられれば，自動生成されたプログラム p' の正しさを確かめられたこととなる．

本手法に関する形式的な定義が正しく与えられた場合，証明すべき命題は次の通りである．

命題 *1* 任意のプログラム p，p'，および，インターフェース s，s' に対して，s' が s に確率を付与したものであり，かつ，$\models p : s$，$p' = \mathbf{Gen}(s', p)$ が成り立つのであれば，$\models p' : s'$ が成り立つ．
ここで，$\mathbf{Gen}(s', p)$ は本手法により自動生成されるプログラムである．

以降では，この命題の予想される証明の概略を述べる．まず，$\vdash p : s' \rightsquigarrow p'$ という型に基づく変換関係を定義する．これは直観的には，プログラム p が s' を満たすようなプログラム p' に変換可能という意味である．このとき，p および s' が与えられた場合，$\vdash p : s' \rightsquigarrow p'$ となる p' が唯一つとなるように定義し，そのような p' を $\mathbf{Gen}(s', p)$ と定義する．ここで，インターフェース上の関数 $\mathbf{ProbToNondet}(-)$ を引数中の確率に関する操作を非決定性の操作に置き換えるものと定義すると，命題 *1* は次の命題の系となる．

命題 *2* プログラム p,p' およびインターフェース s，s' に対して，次を仮定する．

- $\models p : s$.
- $\vdash p : s' \rightsquigarrow p'$.
- $s = \mathbf{ProbToNondet}(s')$.

このとき，次が成り立つ．

$$\models p' : s'$$

この命題は $\vdash p : s' \rightsquigarrow p'$ の導出に関する帰納法によって示すことができると予想する．

6 まとめと今後の課題

本論文では，*Archface-U* に基づいた確率的プログラムの自動生成について述べた．確率をインターフェースに記述することにより，状態爆発を緩和した確率的プログラムのモデル検査が可能となった．これは，インターフェースレベルで確率的モデル検査を行うことで，インターフェースに従って実装されている確率的プログラムにも検査結果が成り立つことによって実現した．また，仕様としてのインターフェース記述からプログラムコードへのリファイメントの自動化などが可能となった．これは，確率を記述したインターフェースからアスペクトを自動で生成し，プログラムコードへウィービングすることによって可能となった．今後の課題として，ツールの面では不確かさを包容した統合開発環境 *iArch-U* にて確率的モデル検査の機能と *AspectJ* を用いた確率的プログラミングの自動生成の実装を行っていくことが挙げられる．特に *Connector* インターフェース上で重みを設定できるようにすることで，より正確に確率的な振る舞いを表現することができる．また，確率の取り扱いやリファイメント等について形式的に記述，および証明をすることで，本手法の正しさを検証していかなければならない．

謝辞 本研究は，文部科学省科学研究補助費基盤研究 *(A)(*課題番号 *26240007)* による助成を受けた．

参 考 文 献

[1] Abrial, J.-R.: Modeling in Event-B: System and Software Engineering, Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[2] Adams, B. and McIntosh, S.: Modern Release Engineering in a Nutshell – Why Researchers Should Care, 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Vol. 5, March 2016, pp. 78–90.

[3] Allen, R. and Garlan, D.: Formalizing Architectural Connection, Proceedings of the 16th International Conference on Software Engineering, ICSE '94, Los Alamitos, CA, USA, IEEE Computer Society Press, 1994, pp. 71–80.

[4] Amine, M., Delahaye, B., and Lanoix, A.: Moving from Event-B to Probabilistic Event-B, Proceedings of the Symposium on Applied Computing, SAC '17, New York, NY, USA, ACM, 2017, pp. 1348–1355.

[5] Fukamachi, T., Ubayashi, N., Hosoai, S., and Kamei, Y.: Modularity for Uncertainty, Proceedings of the Seventh International Workshop on Modeling in Software Engineering, MiSE '15, Piscataway, NJ, USA, IEEE Press, 2015, pp. 7–12.

[6] Hinton, A., Kwiatkowska, M., Norman, G., and Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems, Springer Berlin Heidelberg, 2006, pp. 441–444.

[7] Magee, J. and Kramer, J.: Concurrency: State Models &Amp; Java Programs, John Wiley & Sons, Inc., New York, NY, USA, 1999.

[8] National University of Singapore: PAT : Process analysys toolkit, http://pat.comp.nus.edu.sg/.

[9] Rodrigues, P., Lupu, E., and Kramer, J.: LTSA-PCA: Tool Support for Compositional Reliability Analysis, Companion Proceedings of the 36th International Conference on Software Engineering, ICSE Companion 2014, New York, NY, USA, ACM, 2014, pp. 548–551.

[10] Rodrigues, P., Lupu, E., and Kramer, J.: Compositional Reliability Analysis for Probabilistic Component Automata, Proceedings of the Seventh International Workshop on Modeling in Software Engineering, MiSE '15, Piscataway, NJ, USA, IEEE Press, 2015, pp. 19–24.

[11] Watanabe, K., Ubayashi, N., Fukamachi, T., Nakamura, S., Muraoka, H., and Kamei, Y.: iArch-U: Interface-Centric Integrated Uncertainty-Aware Development Environment, 2017 IEEE/ACM 9th International Workshop on Modelling in Software Engineering (MiSE), May 2017, pp. 40–46.

[12] 渡辺啓介, 深町拓也, 鵜林尚靖, 細合晋太郎, 亀井靖高*:* 宣言的な可変性記述による *A/B* テストの自動化 *[FOSE2016* 推薦レター論文*]*, 日本ソフトウェア科学会誌コンピュータソフトウェア, *2017.*