Electronic Communications of the EASST
Volume X (2015)

Proceedings of the Third International Workshop on Open and Original Problems in Software Language Engineering (OOPSLE 2015)

Uncertainty-Aware Programming

Takuya Fukamachi, Naoyasu Ubayashi, Shintaro Hosoai, Yasutaka Kamei

4 pages

Guest Editors: Anya Helene Bagge, Vadim Zaytsev
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Uncertainty-Aware Programming

Takuya Fukamachi[1], **Naoyasu Ubayashi**[2], **Shintaro Hosoai**[3], **Yasutaka Kamei**[4]

[1] fukamachi@posl.ait.kyushu-u.ac.jp, [2] ubayashi@acm.org, [3] hosoai@qito.kyushu-u.ac.jp, [4] kamei@ait.kyushu-u.ac.jp
Kyushu University, Japan

Abstract: Uncertainty is one of the crucial research topics in software engineering, because all of the requirements or design concerns cannot be captured at the early development phase. This paper introduces an idea of *uncertain-aware programming* and discusses the problems to be tackled.

Keywords: Uncertainty, Separation of Concerns, Modularity

1 Introduction

Recently, uncertainty has attracted a growing interest among researchers [ER14, FSC12]. Research themes spread over uncertainty of modeling, model transformations, and testing. Why does uncertainty fascinate the people acting on the cutting edge of the software engineering research? One of the reasons is considered that uncertainty has not been treated as language constructs. To deal with this problem, we started a new research project *model-driven development embracing uncertainty* from April 2014 [jsp14].

This paper introduces our research outline focusing on programming—*uncertain-aware programming* and discusses the problems to be tackled.

2 Uncertainty in Programming

We introduce the key idea by referring the excerpt from our project plan documents.

2.1 Motivating Scenarios

Assume the following situations: 1) uncertain whether a portion of a program is really needed or should be replaced by other code in terms of refactoring; 2) uncertain which algorithm should be adopted to realize performance requirements, and 3) uncertain which code is finally used because of changeable stakeholder requirements.

We have to temporally comment out the target statements to skip an uncertain concern or insert a superfluous *if* statement to be able to select an alternative uncertain choice. After that, we have to compile and test the program. These comments and conditional statements make difficult to understand the program code, because they impede the separation of concerns in terms of modularity. The exploratory modification process may be repeated again and again until all uncertain concerns are fixed. We consider that many programmers have an experience of encountering this kind of problems. If uncertainty can be dealt with modularly, we can add or delete uncertain concerns to/from code whenever these concerns appear or disappear.

2.2 Uncertain-Aware Programming in a Nutshell

We are developing *ucJava*, a Java programming environment, to provide a modular programming style for uncertainty. We consider that an interface mechanism plays an important role in dealing with uncertainty. In *ucJava*, we extend *Archface* [UAL+14] consisting of components and connectors to express uncertainty. First, we show an original *Archface* without containing uncertainty. Next, we show an extended *Archface* including uncertainty. The former and the latter are called *Certain Archface* and *Uncertain Archface*, respectively.

2.2.1 Certain Archface

A component interface is the same with ordinary Java interface and a connector interface defines the message interactions among components. A connector is specified using the notation similar to FSP (Finite State Processes). A *Certain Archface* definition of the *Observer* pattern is shown in List 1.

```
[List 1] -- Java & FSP-like Syntax
01: interface component cSubject {
02:   public void addObserver(Observer);
03:   public void removeObserver(Observer);
04:   public String getState();
05:   public void setState(String);
06:   public void notify();
07: }
08:
09: interface component cObserver {
10:   public void update();
11: }
12:
13: interface connector cObserverPattern (cSubject, cObserver){
14:   cSubject = (cSubject.setState->cSubject.notify
15:               ->cObserver.update->cSubject.getState->cSubject);
16:   cObserver = (cObserver.update->cSubject.getState->cObserver);
17: }
```

2.2.2 Uncertain Archface

As a representative work on uncertainty, a method for expressing uncertainty using a partial model is proposed in [FSC12]. We apply this idea to *ucJava*. A partial model is a single model containing all possible alternative uncertain designs and is encoded in propositional logic. We can check whether or not a model including uncertainty satisfies some interesting properties.

In *ucJava*, uncertainty is introduced modularly by extending a certain *Archface* as illustrated in Figure 1. The symbols $\{\}$ and $[]$ represent *alternative* and *optional*, respectively. This notation can have an expressive power equal to a partial model. For example, if a programmer doubts whether or not `notify` is really needed for understandability, he or she only has to change *Archface* as shown in List 2.

```
[List 2]
01: interface component uSubject extends cSubject {
02:   public void {removeObserver(), deleteObserver()};
03:  [public void notify();]
04: }
```

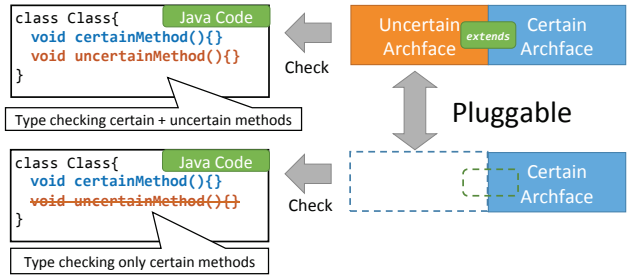

Figure 1: Pluggable Uncertainty

```
05: interface connector uObserverPattern extends ObserverPattern( cSubject, cObserver) {
06:   cSubject = (cSubject.setState-> [cSubject.notify]
07:               ->cObserver.update->cSubject.getState->cSubject);
08: }
```

2.3 Modular Reasoning based on Partial Model

Uncertainty is a target of compilation in *ucJava* whose type checker verifies not only the conformance of a Java program to its *Archface* but also the consistency among components and connectors. The *ucJava* compiler generates a partial model from *Archface* definitions and verifies whether a Java program is an instance of the partial model. In List 2, type check is passed if a Java program corresponds to either of the following: 1) `notify` is defined and is called in the program; 2) `notify` is not defined; 3) both `deleteObserver` and `removeObserver` are defined and either of them is called in the program; or 4) either `deleteObserver` or `removeObserver` is defined and is called in the program. Otherwise, type checker generates an error. In case of List 2, we can continue the development regardless of whether or not `notify` is defined, because there are no inconsistencies in the *Archface* definition. In *ucJava*, original unit test cases are automatically modified using AspectJ to test uncertain methods. The call to an optional method can be skipped and the call to a method defined in the original test case can be replaced by an alternative method.

We can make a program without using comments or conditional statements even if uncertain concerns are contained in the program. We have only to declare an uncertain *Archface*.

3 Research Problems in Uncertain-Aware Programming

Our research project is at the very early stage and many research problems still remain.

3.1 Definition of Uncertainty

We have to clarify not only the difference between uncertainty and other similar concepts such as ambiguity but also the levels of uncertainty. Currently, our community does not have a clear and common definition about uncertainty. In *ucJava*, we consider only the uncertainty that can be expressed by the partial model. Currently, other kinds of uncertainty such as unknown requirements cannot be dealt with by *ucJava*. We consider that the definition of uncertainty is the most important problem.

3.2 Interface vs. Module

It is a difficult question whether an uncertain concern should be described as an interface or a module. The former does not include an instance of uncertain descriptions but only declares an annotation indicating uncertainty. As mentioned in the motivating scenarios, many programmers use comments or conditional statements to temporarily remove or change uncertain concerns. In this case, uncertain concerns still remain in the original code. We consider that temporary code patching is crucial in handling the uncertainty. It is not preferable to define an uncertain concern as a solid module, because a module should be defined in terms of software architecture. We feel that uncertainness is often vague in most cases. In *ucJava*, uncertainty is introduced as a pluggable sub interface.

Aspect-orientation is effective for unanticipated evolution, because structural or behavioral concerns can be modified using inter-type declarations or pointcut-advise mechanism. However, only the aspect-orientation cannot support variability of uncertainness such as alternative or optional. We consider that the problem mentioned here may be resolved by integrating our approach with aspect-orientation.

Although there are difficult problems to be resolved, we believe that the research on uncertainty will become a main stream in the software engineering research community.

Acknowledgements: This research is being conducted as a part of the Grant-in-aid for Scientific Research (A) 26240007 and Challenging Exploratory Research 25540025 by the Ministry of Education, Culture, Sports, Science and Technology, Japan.

Bibliography

[ER14] S. Elbaum, D. S. Rosenblum. Known Unknowns: Testing in the Presence of Uncertainty. In *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. Pp. 833–836. 2014.

[FSC12] M. Famelis, R. Salay, M. Chechik. Partial Models: Towards Modeling and Reasoning with Uncertainty. In *34th International Conference on Software Engineering (ICSE 2012)*. Pp. 573–583. 2012.

[jsp14] Grants-in-Aid for Scientific Research. In *http://www.jsps.go.jp/english/e-grants/index.html*. 2014.

[UAL+14] N. Ubayashi, D. Ai, P. Li, Y. Li, S. Hosoai, Y. Kamei. Abstraction-aware Verifying Compiler for Yet Another MDD. In *29th International Conference on Automated Software Engineering (ASE 2014)*. Pp. 557–562. 2014.