Science of Computer Programming 156 (2018) 104–120

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Method safety mechanism for asynchronous layer deactivation

Tetsuo Kamina [a,*], Tomoyuki Aotani [b], Hidehiko Masuhara [b], Atsushi Igarashi [c]

[a] Ritsumeikan University, Japan
[b] Tokyo Institute of Technology, Japan
[c] Kyoto University, Japan

ARTICLE INFO

Article history:
Received 17 February 2016
Received in revised form 15 January 2018
Accepted 21 January 2018
Available online xxxx

Keywords:
Context-oriented programming
Layer-introduced base method
ContextFJ
ServalCJ

ABSTRACT

Context-oriented programming (COP) enhances the modularity of context-dependent behavior in context-aware systems, as it provides modules to implement context-dependent behavior (layers) and composes them dynamically in a disciplined manner (layer activation). We propose a COP language that enables layers to define base methods, while the layers can be asynchronously activated and deactivated. Base methods in layers enhance modularity because they extend the interface of classes without modifying original class definitions. However, calling such a method defined in a layer is problematic as the layer may be inactive when the method is called. We address this problem by introducing a method lookup mechanism that uses the static scope of method invocation for COP; i.e., in addition to currently activated layers, the layer where the method invocation is written, as well as the layers on which that layer depends, are searched during method lookup. We formalize this mechanism as a small calculus referred to as ContextFJ^a and prove its type soundness. We implement this mechanism in ServalCJ, a COP language that supports asynchronous, as well as synchronous, layer activation.

1. Introduction

For several years, context awareness has been a major concern in multiple application areas, and its importance is increasing. For example, with progress in sensor technologies, computing platforms have become more aware of physical environment, and user interfaces have become more adaptable to users' current operations. These interactions with the environment require the ability to change behavior with respect to *context*, such as a specific state of the physical environment or a user's current task. Such dynamic changes in behavior result in complicated system structures and behaviors that are difficult to predict using traditional programming abstractions. To address this difficulty, several context-oriented programming (COP) languages, which have successfully modularized such context-dependent behavior, have been developed [1–9].

COP languages provide language constructs that modularize the variations in behavior that depend on context using *layers*[1] and dynamically activate/deactivate them according to the executing contexts [2,1]. A layer defines *partial methods*, which run before, after, or around a call of a method with the same signature defined in a class, only when the layer is active. These constructs make COP advantageous in terms of modularity, because partial methods can change the original be-

* Corresponding author.
E-mail addresses: kamina@acm.org (T. Kamina), aotani@c.titech.ac.jp (T. Aotani), masuhara@acm.org (H. Masuhara), igarashi@kuis.kyoto-u.ac.jp (A. Igarashi).

[1] In this study, we focus on layer-based COP languages.

https://doi.org/10.1016/j.scico.2018.01.006
0167-6423/

havior by activating layers without changing the base classes, and ensure consistency in dynamic changes using scoping [2] or model checking [5].

Even though several COP languages support only partial methods, *layer-introduced base methods*, i.e., methods in a layer that introduce a new signature and do not override other methods, are also known to be useful in COP [10]. Layer-introduced base methods considerably enhance modularity because they extend the interface of classes dynamically, which makes ensuring type-soundness in COP languages more challenging. Formal calculi have been proposed to support such an extension, e.g., (1) requiring a subordinate layer (a layer that provides base methods) to be activated while the dominant layer (a layer that uses these methods) is executing [10,11], and (2) activating the subordinate layer on-demand when the dominant layer is executing [12].[2]

However, there is an issue in combining these approaches with *asynchronous* layer activation [13,6,14,5,8] in a type-safe manner. Asynchronicity is crucial to application domains where contexts change outside of a program, for example, ubiquitous computing applications and adaptive user interface. The method lookup in existing COP semantics searches all activated layers and the class of a method receiver to dispatch a called method. This semantics does not lead to a problem when layers are activated using `with`-blocks, where the corresponding layer activation is synchronous with the currently executing block. However, it leads to a problem in asynchronous layer activation, where layers are activated and deactivated by external events such as changes in external environment and user operations. These events may occur at any program execution point. Thus, it is possible that the layer that provides a base method to a currently executing method is eventually deactivated, resulting in a method-not-understood error.

In this paper, we propose another method lookup mechanism for type-safe layer deactivation.[3] In this mechanism, methods are searched in the layer *where method invocation is placed and in the layers on which this layer depends*, as well as in currently activated layers. This inclusion of the "static scope" for method lookup addresses the abovementioned problem of method safety. In other words, the proposed approach supports layer deactivation in the "best effort" manner; that is, the deactivation of layers is ensured to the maximum extent, while the deactivation can be canceled when the layers that require these layers are activated. This approach is a natural extension of *loyal strategy* [16], which most COP languages adopt, in that the execution of the required behavior is ensured during the execution of the partial method in the requiring layer.

This approach is applicable when layer deactivation is not a hard requirement. However, it leads to a problem in other cases. For example, we may consider a layer that performs a computation with highly precise results, thereby consuming considerable CPU power. Deactivation of this layer is a hard requirement when a battery is approaching exhaustion because a computation that consumes considerable CPU power should not be performed in this case. However, a layer that is not deactivated may require this high precision layer; thus, it cancels the deactivation.

To resolve this problem, we also introduce an additional mechanism that ensures deactivation of specified layers, i.e., a modifier, `ensureDeactivate`, for layer declarations, which indicates that the deactivation of the declared layer cannot be canceled. To ensure that the methods in layers, which are declared with `ensureDeactivate`, are not called accidentally by other layers that require those layers, layers cannot require the layers declared with `ensureDeactivate`.

We formalize this idea as a small COP calculus referred to as ContextFJ^a, and show that this calculus ensures deactivation of layers with `ensureDeactivate` and is type sound. This calculus is an extension of ContextFJ [10] and notably simple, even though it is sufficiently expressive to represent asynchronous layer activation and layer-introduced base methods.

The proposed mechanism is implemented in ServalCJ, a COP language with a generalized layer activation mechanism [17]. ServalCJ supports asynchronous as well as synchronous layer activation, and per-instance as well as global layer activation. Originally, ServalCJ did not support layer-introduced base methods. As the proposed mechanism ensures the safety of method with asynchronous layer activation, we safely realize layer-introduced base methods in a generalized layer activation mechanism.

The rest of this paper is structured as follows. In Section 2, an overview of COP mechanisms, such as layers, layer activation, and layer-introduced base methods, is provided. In this section, the problem that is addressed in this paper is also identified. In Section 3, we illustrate the proposed method lookup and formalize it as a small calculus, ContextFJ^a. In Section 4, we discuss the problem of layer deactivation cancellation and the proposed solution. In Section 5, we describe the type system of ContextFJ^a and prove its type soundness. In Section 6, the implementation of the proposed mechanism is described. In Section 7, related work is discussed. Lastly, conclusions are stated in Section 8.

2. Layers, layer-introduced base methods, and their problem

We demonstrate a motivating example of an adaptive user interface, which comprises a text editor program that was inspired by the program editor example proposed by Appeltauer et al. [18]. Our example includes class `Editor` (and other classes) to represent an editor view for the user. This user interface provides a menubar (and other widgets), which is displayed by calling `showMenuBar` when the display is refreshed, as shown below.

[2] To be precise, there is a flaw in the proof of type soundness for on-demand activation [12]. To ensure type soundness, we need to modify the reduction of method invocation to enclose the entire method execution within the activation of all required layers.

[3] This paper is an extended version of our previous work [15]. The main differences from the previous work are the `ensureDeactivate` mechanism, a complete set of computation rules and a type system, and the proof of type soundness.