Toward Fluent Module Interactions

Tetsuo Kamina
Ritsumeikan University, Japan
kamina@acm.org

Tomoyuki Aotani
Tokyo Institute of Technology, Japan
aotani@is.titech.ac.jp

Hidehiko Masuhara
Tokyo Institute of Technology, Japan
masuhara@acm.org

Abstract

Recent progress on sensor technologies poses challenges on software development such as more interaction with physical environment and context-awareness. This trend makes it difficult to decide the boundaries between changing module interactions. In this paper, we propose a concept of fluent module interactions where we characterize the module interactions in three dimensions, i.e., definition, duration, and scope. Module interactions in any of those dimensions can change dynamically. We also propose a possible extension of existing programming language with fluent module interactions based on context-oriented programming (COP). Then, we derive a future research roadmap for realizing fluent module interactions that covers a wide range of research fields including theory, implementation, engineering, and applications.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features

General Terms Languages

Keywords Context-oriented programming; Dynamic adaptation; ServalCJ; Research roadmap

1. Introduction

With the progress of sensor technologies, computing platforms become more aware of physical environment. More interaction with physical environment requires tighter connection between software and hardware. At the same time, becoming aware of physical environment poses other requirements on software development: context-awareness and autonomous behavior changes with respect to context changes.

To achieve those requirements, it is required to decide the boundaries between the behavior variations. This is not trivial because sometimes such boundaries are changing, or should be decided in a bottom-up manner at runtime. For example, to detect outdoors/indoors situations, we may use a combination of several devices such as GPS and wireless LAN receivers, and atmospheric pressure sensors. Sometimes such devices get an erroneous status and become unavailable, and the method to detect situation switching, which constitutes the boundary between outdoors/indoors behaviors, changes dynamically. Another example is the boundary between taxiing and flying behaviors of an unmanned aerial vehicle. Since the behavior seamlessly moves from one to the other, it is not easy to decide the boundary between them.

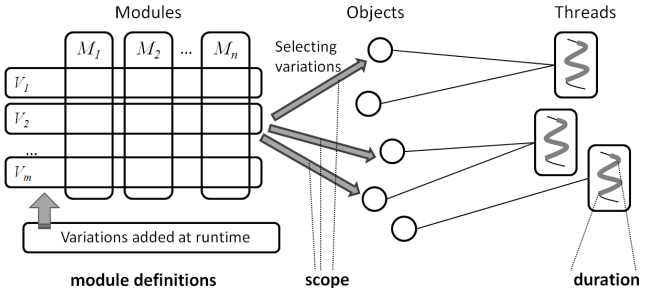

Figure 1. Fluent module interactions. Each module consists of several variations, which can be added at runtime. Each variation can be selected and deselected for particular set of objects at runtime. This selection comprises the definition of modules, and this set of objects indicates the scope of the selected variations. This selection also pertains a particular duration of a particular thread. In fluent module interactions, all of those "boundaries" are dynamically changeable.

In other words, such boundaries looks ambiguous, but in fact, they exist; thus, they should be determined later on and adaptable when contexts are changing. Unfortunately, existing modularization mechanisms such as OOP, AOP, and FOP impose rather fixed boundaries on behaviors, which makes it difficult to realize such late-boundable and changeable behaviors.

In this paper, we propose the concept of *fluent module interactions*, which reactively respond to dynamic context changes and then dynamically change the module interactions. More precisely, we characterize the module interactions in three dimensions: the module definitions, the duration on which the module takes effect, and the scope of the module. Fluent interactions in module definition indicate that module definitions change dynamically. Fluent interactions in module scope indicate that a set of computation units that pertain a specific module changes dynamically. Fluent interactions in duration indicate that the duration of the module effect changes dynamically. The fluent module interactions make it possible to dynamically change the interactions in any of those dimensions, by reactively responding to observed context changes (Figure 1).

As the first step to realize fluent module interactions, we derive requirements for programming languages that support them. Our strategy is based on context-oriented programming (COP) [1] where behavior of classes in object-oriented programming (i.e., the definitions of classes) can change dynamically. In particular, we use ServalCJ [2] as the base language where programmers can explicitly specify the module interactions with respect to scope and duration, and discuss how to extend the language to support flu-

MODULARITY Companion'16, March 14–17, 2016, Málaga, Spain

http://dx.doi.org/10.1145/2892664.2892689

ent module interactions.[1] This discussion raises many issues that should be explored in future in several research fields, including theory, implementation, engineering, and applications.

2. Motivation

To illustrate our idea, we elaborate a motivating example that interacts with physical environment. We consider a pedestrian navigation system that provides variations of behavior for each of outdoors and indoors situations that can be switched dynamically. These situations are detected using GPS, wireless LAN, and atmospheric pressure sensors equipped with a mobile device. There are additional requirements as follows.

Specification of behavior duration should change. Even if some devices become unavailable for some reasons, the system should be able to detect the situations with degraded accuracy; i.e., the method to detect situation switching, which triggers the behavior changes, can change at runtime.

Module definition should change. By paying extra fees, the user can install an extra feature, namely a satellite view, without stopping the system. This feature crosscuts several modules and thus the module definitions should change at runtime.

Specification of behavior scope should change. The user can select another variation, i.e., a static map, simultaneously, which implies that only a limited set of computation units executes the outdoors behavior. Furthermore, the satellite view is running on another computation unit, and should be visible only when the running object also executes the outdoors behavior.

3. A Language To Be Extended

As the first step to realize fluent module interactions, we discuss the design of a programming language that supports them. First, changing module definitions at runtime can be achieved by applying the *layer activation* mechanism in COP [1].

In COP, the behavioral variations added at runtime are implemented using *layers*, which provide the partial definitions for classes that override the original behavior when the layer is composed with the system. In the pedestrian navigation example, we may consider two layers for outdoors and indoors behaviors, namely `Outdoors` and `Indoors`, which override the original behavior (i.e., the static map) by providing the auto-scrolling features (outdoors) and floor plans (indoors) for `Map` with appropriate positioning mechanisms for `Nav`. Below is a skeleton of those layers.

```
layer Outdoors {
  class Map { .. }
  class Nav { .. } }
layer Indoors {
  class Map { .. }
  class Nav { .. } }
```

Those layers can dynamically be *activated* (i.e., composed with the running system). Even though in the original COP, layer activation is hardwired in the base program (using so-called `with`-blocks), recent research shows that layer activation can be characterized in two dimensions, i.e., scope and duration [2], and by explicitly specifying them, the programmer can flexibly specify layer activation. Below is a skeleton of layer activation in ServalCJ [2].

```
contextgroup Boundary {
  subscribers: .. // AOP pointcut
  activate Outdoors .. // temporal term  }
```

[1] We may also consider other COP languages with generalized linguistic features. For example, Korz [5] is another candidate for a base language for the proposed extension.

We specify the duration of `Outdoors` using the `activate` declaration where we can describe when `Outdoors` is activated using a temporal term. This temporal term is a combination of events, which are join-points in AOP, control-flows, and conditions. For example, we can identify the situation switching (i.e., from indoors to the outdoors situation) that activates `Outdoors` as an event using event handlers of wireless communications equipped with the mobile device. Similarly, we can identify the situation switching that deactivates `Outdoors` as another event. We also specify the scope of `Outdoors` using the `subscribers` declaration, which specifies the join-points where the objects on which `Outdoors` is applied are created (or, we can specify the objects within the base program). For example, in the above pedestrian navigation system, the user can select a static map view that provides the different behavior from `Outdoors`; thus, only a limited set of `Map` and `Nav` instances should be included within the scope of `Outdoors`.

Even though ServalCJ explicitly specifies the interactions of context-dependent behavior with respect to its scope and duration, those are fixed. Thus, it does not satisfy the aforementioned requirements. For example, since the specification of `Outdoors` duration is fixed, we need to rebuild the system when the outdoors situation detection changes by some erroneous status of the devices. Furthermore, ServalCJ does not allow us to install new layers dynamically. Thus, to install the satellite view feature, which also require a change of `subscribers` specification for `Outdoors` because the object that executes the satellite view behavior should also executes the outdoors behavior, we also need to rebuild the system.

4. Future Research Roadmap

The above discussion leads us to further research on realizing fluent module interactions. First, runtime installation of layers is necessary. Even though this is achieved by COP languages based on dynamic languages such as JavaScript [3], this still remains as future work in COP based on statically typed languages where extra verification of dynamically installed layer would be necessary. Second, we should consider autonomous changes of module interactions by reacting the observed situation switching (e.g., erroneous status of some devices). By focusing on the data-flows of values obtained from the external world, reactive programming (RP) would help, and how to support reactive layer activation already gains attractions from many researchers [4]. This direction also raises a number of issues in theory and implementation of programming languages, e.g., ensuring constraints when module interactions change reactively and developing efficient compilers. Finally, this focus on fluent module interactions would lead us to new software engineering disciplines and methodologies.

References

[1] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented programming. *Journal of Object Technology*, 7(3):125–151, 2008.

[2] Tetsuo Kamina, Tomoyuki Aotani, and Hidehiko Masuhara. Generalized layer activation mechanism through contexts and subscribers. In *MODULARITY'15*, pages 14–28, 2015.

[3] Jens Lincke, Malte Appeltauer, Bastian Steinert, and Robert Hirschfeld. An open implementation for context-oriented layer composition in ContextJS. *Science of Computer Programming*, 76(12):1194–1209, 2011.

[4] Guido Salvaneschi and Mira Mezini. Reactive behavior in object-oriented applications: an analysis and a research roadmap. In *AOSD'13*, pages 37–48, 2013.

[5] David Ungar, Harold Ossher, and Doug Kimelman. Korz: Simple, symmetric, subjective, context-oriented programming. In *Onward! 2014*, pages 113–131, 2014.