Safer Context (de)Activation
Through the Prompt-Loyal Strategy

Nicolás Cardozo       Sebastián González
Kim Mens
ICTEAM Institute
Université catholique de Louvain
B-1348 Louvain-la-Neuve, Belgium
{nicolas.cardozo,s.gonzalez,kim.mens}@uclouvain.be

Theo D’Hondt
Software Languages Lab
Vrije Universiteit Brussel
B-1050 Brussels, Belgium
tjdhondt@vub.ac.be

Abstract
Context-oriented programming languages provide language abstractions for the dynamic activation and deactivation of behavioral adaptations, based on the system’s context of execution. As contexts are freely activated and deactivated, their associated behavior adaptations are added and removed to and from the system, which may break its consistency with respect to other available adaptations. To manage consistency between adaptations this paper introduces a model for the safe activation and deactivation of contexts. The model consists of two approaches, prompt-loyal for a delayed context (de)activation, and prompt for an immediate context (de)activation.

Categories and Subject Descriptors D.3.3 [Programming Languages]: Language Constructs and Features
General Terms Design, Languages, Reliability
Keywords Context-oriented programming, Consistency management, Behavior adaptation

1. Introduction
Context-Oriented Programming (COP) is a new programming paradigm that introduces language-level abstractions to facilitate the development of software systems that can adapt to highly dynamic execution environments. These systems become aware to their execution context by sensing the surrounding environment. When a particular context arises in the execution environment of an application, specific behavior adaptations for that context are applied. Likewise, if the context is no longer valid, its associated behavior adaptations are disabled.

Dynamically enabling and disabling behavior adaptations may yield unexpected application behavior, because added adaptations may conflict with the ones already deployed in the system, or currently executing adaptations may be removed from it. To avoid these kinds of inconsistencies, there have been two strategies for the dynamic (de)activation of contextual adaptations: loyal and prompt [4]. The loyal approach ensures that the execution of behavioral adaptations finishes in the same context in which it started, preventing other threads from modifying the adaptation configuration from the outside. The prompt strategy, on the other hand, permits the addition or removal of adaptations from external threads (typically, a context manager thread) as soon as their associated context becomes active or inactive. However, such concurrent manipulation may lead to behavioral inconsistencies if not managed properly. For this reason, many context-oriented languages privilege loyalty over promptness [1].

This paper proposes a context management technique that mixes the two strategies for the (de)activation of adaptations. A new strategy called prompt-loyal is introduced to allow context (de)activations that are applied as soon as possible, rather than immediately as in the prompt strategy. The notion of “as soon as possible” is explained in the paper. We claim that the prompt-loyal strategy should be used instead of the strict prompt strategy, because the latter does not provide any kind of guard for the application of adaptations. The guard we propose is precisely the kind of needed fix.

2. Motivation
This section presents a context-aware CityMaps application for mobile devices, which serves as guiding example to describe the main features of our underlying context-oriented language, and to discuss the different alternatives for the dynamic activation and deactivation of context-dependent behavior.
We use Ambience [6] to implement the application and its behavioral adaptations. Ambience is a context-oriented programming language developed in Common Lisp, featuring multimethods and a prototype-based object model similar to that of Slate [8].

2.1 CityMaps Example

The CityMaps application contains static maps of cities, annotated with information such as street names and special spots (hospitals, hotels, public transportation stops). The behavior of the map can be enhanced based on geographical location information coming from a Global Positioning System (GPS). When a GPS is available, the displayed map is centered and zoomed at the current position of the user, depicted by an avatar. To improve user experience in case of GPS disconnection, before reverting to the default (static) display behavior, the application can step into an extrapolation mode. In this degraded mode, the user position is suggested based on the previously observed information (position, direction, speed, and suggested path).

In Ambience, the particular situations in which applications execute are refined as first-class context objects. A context object can delegate to more general contexts if needed. Figure 1 gives an overview of the context relationships for the CityMaps example. Besides individual context objects such as @gps, there are combined context objects such as (@city-maps @gps), representing situations with adaptations that are specific to the given combination.

In such case A is called a subcontext of B. Conversely, B is a supercontext of A.

The base functionality of CityMaps is given by a draw method that displays the map on the screen. The application also provides specific behavior for the @gps context, made available when the device’s GPS is enabled. For such situations, the draw method is refined to fetch the geographical coordinates from the GPS and draw the user avatar on the display, as shown in Snippet 1. Ambience provides a dedicated construct to define context-specific behavior, namely (with-context context behavior). The first block in Snippet 1 defines behavior specific to @gps context; the second block defines behavior specific to the (@city-maps @gps) combined context. In this second specialization, the resend method is used to invoke the default map-drawing behavior of @city-maps context (not shown).

```
(with-context @gps
  (defmethod get-location ()
    (latitude gps-registry) ; estimate latitude
    (longitude gps-registry)) ; estimate longitude)

(with-context (@city-maps @gps)
  (defmethod draw ()
    (resend) ; invoke (@city-maps @gps) draw
    (draw-estimation-tags)))
```

```
(with-context (@city-maps @gps)
  (defmethod draw ()
    (resend) ; invoke (@city-maps @gps) draw
    (draw-estimation-tags)))
```

Snippet 1. GPS-specific method definitions.

When the GPS is unavailable, the CityMaps application provides a fallback mechanism to retrieve an approximate position by means of the behavioral adaptations defined for @degraded-gps context, shown in Snippet 2. This context contributes with functionality to extrapolate the user position based on previously recorded information. Also, the displayed map and user avatar are decorated with a tag signaling that the position maybe less accurate since it has been estimated. Note that the @degraded-gps version of draw is more specific than the @gps version, due to the delegation from @degraded-gps to @gps. Hence, the resend call used in the degraded version invokes the non-degraded one. Within the non-degraded version, the version of get-location that will be invoked corresponds to that of @degraded-gps, which as expected will extrapolate the coordinates.

```
(with-context @degraded-gps
  (defmethod get-location ()
    (latitude gps-registry) ; estimate latitude
    (longitude gps-registry)) ; estimate longitude)

(with-context (@city-maps @degraded-gps)
  (defmethod draw ()
    (resend) ; invoke (@city-maps @gps) draw
    (draw-estimation-tags)))
```

Snippet 2. Degraded GPS method definitions.

To find the applicable methods for a given message, Ambience keeps track of active contexts in a dynamic context delegation graph like the one in Figure 1. Active contexts in the application are represented as a whole by the current context [6]. Applicable methods are first looked up in the current context object, and then in its delegate subcontexts.¹

¹ Lookup order is defined by the C3 linearization [2], but ordering is irrelevant to this paper. We therefore omit the details.
Behavior adaptations become available and unavailable by the dynamic activation and deactivation of their associated context. When a context ctx is activated, the methods defined within the (with-context ctx ...) block become globally available —that is, the contexts are activated in all threads running in the system. Such concurrent context switching can bring problems. Take for example the execution of the draw method for the combined context (@city-maps @gps) in Snippet 1. Suppose that draw has just begun execution, and @degraded-gps is activated before get-location is invoked. The version of get-location that will be invoked is the one for @degraded-gps context, which extrapolates the coordinates. The draw method for @gps context will then continue running till completion, without drawing any tags that warn the user about the estimation. This breaks the requirements of the system, showing the problem of interleaved context activations. To illustrate the analogous case of interleaved context deactivations, consider the execution of draw for (@city-maps @degraded-gps) in Snippet 2. Suppose that @degraded-gps is deactivated just after the execution of the resend method. At this point, the remainder of the execution of draw is compromised, because the message to draw the tags will no longer be understood.

The previous problems motivate the definition of two context-switching strategies already mentioned in Section 1, which we now refine for the sake of the forthcoming discussion:

**Loyalty** The loyal strategy consists in preventing a context activation if one of the methods defined in such context would override a currently executing method of another context —that is, the context cannot be activated if it provides adaptations of methods currently in use. A context deactivation is prevented if a method specific to that particular context is currently executing —that is, the context cannot be deactivated if it is currently in use.

**Promptness** The prompt strategy applies behavior adaptations as soon as a context switch message is received. This means that the behavior for a particular context becomes (un)available immediately, when the context change is sensed.

As illustrated previously, the bare prompt strategy can lead to inconsistencies. On the other hand, the bare loyal strategy might prevent the system from adapting swiftly to context changes. We therefore look for a marriage of both.

### 3. Prompt-Loyal (de)Activation

This section presents the prompt-loyal strategy for context activation and deactivation. The strategy is intended as a middle ground between the loyal and prompt context (de)activation strategies. The main idea of this approach is to process (de)activation messages as soon as possible, but not so soon that they would likely break currently executing methods.

The expected semantics of this strategy is to allow context (de)activations as soon as the message is received, but hiding out its adaptations, until other, currently in use, contexts have finish their pending computations. After such computations have cease, introduced adaptations become available, and extracted adaptations become unavailable automatically. Take for example the activation of the @degraded-gps context while the draw method, defined in the (@city-maps @gps), context is being executed. The @degraded-gps context is activated but its draw method is not yet available. In this case, the map will still be drawn as specified in the (@city-maps @gps) context, that is, with the location given by the GPS and without tags.

### 3.1 Activation Counters

The explicit activation of a context that is already active will not change the composition of the current context graph, but the context management system will record this additional activation by incrementing an activation counter variable of the context object. Upon a deactivation request, the counter is decremented, and if it reaches zero, the deactivated sub-context will be effectively removed from the current context. The rationale behind this mechanism is similar to the retain/release logic of memory management systems based on reference counting. Activation counters were originally introduced in Ambience [5].

### 3.2 Usage Counters

To define the prompt-loyal strategy, context objects are extended with a new usage counter variable to keep track of contexts being used. When a context-specific method begins its execution, the usage counter of that particular context is incremented. Correspondingly, when the method finishes execution, the usage counter is decremented. Note that usage implies activation: if a context is being used, it is necessarily active —if it is inactive, none of its methods would have been found by the method lookup, and therefore could not execute.

Usage counters are analogous to activation counters in that they signal the need for a context. However, whereas activation counters encode the dynamics of context management policies, usage counters encode the dynamics of context-specific method execution.

Given the previous notions, we proceed to explain the internals of the prompt-loyal semantics for the activation and deactivation of contexts, using the CityMaps example.

### 3.3 Context Activation

Context activation can be divided into three different cases, based on the context to be activated and the state of its sub-contexts and super-contexts: (a) if none of the active contexts is being used, and (b) if one of the super-contexts of the context to be activated is being used.
The first case, when none of the active contexts is being used, is straightforward. The activation of the context is processed immediately and since there are no contexts being used, there is no possibility of breaking system consistency. The associated adaptations to the context can be made visible immediately, complying with the prompt-loyal semantics.

The second case, when the active context in use is a supercontext of the context being activated, requires more work to become prompt-loyal. The activation message is processed immediately as in the previous case. However, if the associated adaptations are available immediately, they will be taken first by the method lookup mechanism, than those adaptations of the context currently in use. This may break the assumptions in which the adaptation execution started, a more specific method of a subcontext may yield an inconsistent program behavior. To solve this problem, the method lookup mechanism is modified, to take into account the usage counter, as follows. To avoid the accidental capture of more specific methods that may have been introduced by a subcontext, the method lookup mechanism will retrieve a list composed by the first method for which the usage counter is not zero, and its successors. This situation is shown in Figure 2.

For example, in the CityMaps application, suppose that the (get-location) method is being used in the @gps context and the @degraded-gps is activated. In this situation, the activation counter and usage counter for @gps are both positive, while for @degraded-gps the activation counter is one and the usage counter is zero. In the classic version of the method lookup mechanism for the (get-location) method returns a list for which the first position is the instance for the @degraded-gps and the second position is the one for the @gps. However, with the aforementioned modification, the method lookup retrieves the method defined for the @gps context first, since it is the first context with a positive usage counter.

The execution of the @gps-specific (get-location) adaptation will remain loyal to the situation present when it was called, and the new adaptations will be promptly seen as soon as the method has finished.

Note that the resolution process for the appropriate method in the method lookup mechanism proposed here is consistent with the previous case of context activation. In the case when none of the contexts are being used, the lookup mechanism will retrieve the list of all applicable methods ordered from the most specific to the less specific one, as it originally did.

3.4 Context Deactivation

Context deactivation is also divided into two cases, based on the context to be deactivated: (a) if the context to be deactivated is not being used, and (b) if the context to be deactivated it is being used. For the first case, the deactivation can take place immediately without breaking system consistency because there are no methods being executed for that context. Hence, this deactivation is prompt-loyal.

The case in which the context to be deactivated is being used is more challenging. In this case, the solution consists also in a modification of the method lookup mechanism. The context deactivation message is processed immediately; however, its associated adaptations are not excluded from the lookup mechanism until both the context activation counter and usage counter are zero. This modification to method lookup is shown in Figure 3.

In the CityMaps application, take for example the situation when the @gps context is deactivated while the draw method defined in the (@city-maps @gps) context is being used. In such a case, while the adaptation seems to be inactive for the outside world, internally the calls to the resend and display-avatar methods can still be used, as the two of them will be retrieved by the method lookup mechanism. This execution remains loyal to the execution of the draw method. The adaptations related to (@city-maps @gps) become invisible to method lookup as soon as the draw method finishes.

The new modification to the method lookup mechanism adheres to the modification made previously in the activation case, because the adaptation retrieved by the method lookup corresponds to the context being used, which is the first one with a positive usage counter.

3.5 Supporting Promptness

There are cases in which it is desirable to have a prompt reaction to context (de)activation. In the CityMaps application, take for example the case in which the GPS hard-
ware module gets physically disconnected from the mobile device while in the middle of the first call to the &gps context. Using the prompt-loyal strategy, the execution of the longitude method will still be done in &gps context, as it is currently being used, querying the GPS hardware module for the coordinate. We claim that in such a case the deactivation of the context should occur promptly. In the best case the application would need to actively wait until the GPS hardware module is reconnected, to return the actual longitude position, or in the worst case the application would crash when the longitude method is unexpected. To avoid these issues, we propose to incorporate two constructs in Ambience that provide a prompt context (de)activation strategy, (prompt-activate ctx) and (prompt-deactivate ctx).

The introduction of these constructs needs to comply with the modifications made to the method lookup mechanism in this section. Prompt context activation is realized in two steps. First the context is activated by incrementing its activation counter. Second, to ensure that the introduced method adaptations will be effectively retrieved by the method lookup mechanism, all other adaptations selected by the lookup mechanism, must reset their usage counter back to zero. Similarly, prompt context deactivation is achieved by setting both the activation counter and the usage counter of the context back to zero, ensuring that adaptations associated to that context will not be retrieved by the method lookup mechanism.

However, the prompt strategy may still rise inconsistencies in the system. The responsibility for ensuring that the prompt context (de)activation maintains system consistency is left to the user by providing exception handling blocks such as try/catch.

3.6 Discussion

The introduction of the prompt-loyal strategy proposed in this section contributes to maintain application consistency in the sense that it ensures method adaptations to be completely executed under the conditions in which their execution started (same set of active contexts). Our strategy is not completely loyal because context (de)activations are still allowed, although the introduced adaptations are not visible as long as previous activations are being used. Context (de)activations are prompt in the sense that the moment in which the contextual adaptations are actually introduced occurs as soon as possible and is seamless to the user, thus meeting a middle ground between the two strategies.

However, we recognize that in some situations it would be beneficial to maintain an immediate causal connection between contexts and the detected execution environment, as discussed in Section 3.5. There are contextual situations that are tightly coupled with the physical world, and as physical changes are registered, the application should adapt swiftly. Deferring adaptation execution in a prompt-loyal fashion for these kind of situations can cause inconsistencies, because the observed application behavior will not correspond to the current setting in the physical world.

4. Related Work

The trade-off between the loyal and prompt context activation and deactivation strategies were first identified by Desmet et al [4]. Their definition of loyalty is a specific case of our definition, because it is meant for the layer-based family of context-oriented languages represented by ContextL [3]. The language constructs presented to provide a loyal context (de)activation strategy set under the assumption that layers are local to a single thread. Taking advantage of the dynamic scope of the constructs to activate and deactivate layers, all methods are executed within the specified layers, maintaining their execution loyal. Unfortunately this cannot be applied in Ambience to provide a loyal approach, because the underlying assumption in Ambience is that contexts globally modify the application.

The Subjective-C language [7], introduces two constructs to manage dependency rules between contexts, namely canActivate and canDeactivate. Similar constructs could be used in Ambience to verify if a context can or cannot be (de)activated. The use of such constructs would be used to provide a loyal context (de)activation strategy. Every context (de)activation could be verified by the system against predefined policies, to decide whether it is safe to process the (de)activation message or not.

5. Conclusions and Future Work

This paper presents a model to provide the prompt-loyal strategy for dynamic context (de)activation. The goal of this strategy is to manage system consistency when behavioral adaptations are added and removed as a consequence of the activation and deactivation of their associated contexts. The prompt-loyal strategy provides a delayed context (de)activation for situations in which the ongoing computation of adaptations should finish without being affected by newly added or removed adaptations. New adaptations are installed as soon as used ones become idle.

The prompt-loyal strategy contributes to maintain system consistency, but under certain circumstances the prompt strategy can be needed; in such cases (de)activation occurs immediately to comply with the represented situations of the physical world.

This work is still in its preliminary stages. We are currently introducing the needed modifications for the lookup mechanism and a refinement of the design and implementation to support the prompt-loyal strategy is still needed, as well as corresponding benchmarks.

An additional step to complement the model to support the loyal approach for context (de)activation would be to introduce the notion, in Ambience, of contexts that are local to a thread. With a complete model for safe context
(de)activation, developers could choose whether a context (de)activation is resolved in a loyal, prompt-loyal or prompt fashion, even more choosing the strategy could be based on the context of execution.

Acknowledgments
This work has been supported by the ICT Impulse Programme of the Institute for the encouragement of Scientific Research and Innovation of Brussels, and by the Interuniversity Attraction Poles Programme, Belgian State, Belgian Science Policy. We thank the anonymous reviewers for their comments on an earlier version of this paper.

References