

Strategies for Handling the Activity Problem in Runtime Software Evolution by Reducing Activity

Jens Gustavsson
Linköpings universitet, Sweden
jengu@ida.liu.se

1. Introduction

Runtime software evolution means updating software systems while they are running, something that is useful for systems with high availability requirements. A method is *active* if it has been called and not yet finished its execution, i.e. it has an activation record on at least one method call stack. Activity is a problem when making runtime updates, since it must be decided how the activation records of updated methods shall be treated. When inactive methods are updated, it is reasonable to have the system run the new version of the methods next time they are called.

The main goal of this paper is to describe different strategies to reduce the activity and under which circumstances those strategies are viable. To do so we define some properties of active methods in order to facilitate an abstract view on different kinds of activity.

2. Properties of Active Methods

In this section, we propose a set of three properties that distinguish between methods that are highly active for different reasons. The properties have been selected to distinguish between methods that need different strategies for reducing the activity.

An active method is *doing* or *waiting*. *Doing* means that code in method body is executed every once in a while, at least every few seconds. A method that is *doing* is at the top of the stack often. A method that is lower than the top of the stack for long periods of time is *waiting*. If a method takes long or infinite time to finish execution, it is *long*, otherwise it is *short*. A method is *sequential* if at most one activation record of it exists at the same time, otherwise it is *parallel*.

3. Strategies for Reducing Activity

Some of the strategies are based on the fact that breaking up a method in smaller parts will change the activity characteristics of it, while the behavior of the system stays the same. We call this *outlining*.

Outline parts at development time. This strategy is viable for methods that are sequential, long and doing.

Outline non-blocking code at development time. This strategy is viable for methods that are sequential, long and waiting. The strategy is to outline the parts that are not the call that the method is waiting for.

Outline updated part by runtime refactoring. This strategy is viable for methods that are parallel and short. A runtime refactoring is a change at runtime that does not change the observable behavior of the system. This means that it is safe to perform them by letting old and new versions coexist until all old calls have finished executing.

Pause service of a method. This strategy is viable for methods that are parallel and short. The strategy is to stop new threads from entering the method while waiting for all executions of the method to finish. This can be implemented by a runtime refactoring.

Kill threads. This strategy is viable for all methods.

Restart methods. This strategy is viable for all methods. The strategy is to throw activation records from call stacks where the method to be updated are called.

4. Case study

We have created a tool that automatically analyses a running system and finds the properties for any given method. We have performed a case study, where Jetty, an HTTP and servlet server, was investigated.

Table 1: Properties of highly active methods in case study.

Properties	Number of methods
parallel, short, doing	15
parallel, short, waiting	5
parallel, long, doing	1
parallel, long, waiting	7
sequential, long, doing	1
sequential, long, waiting	2

We found that a total of 972 methods were called. 31 of these methods were highly active, i.e. 3.2%. The number of highly active methods with different properties is shown in Table 1. We found examples of all combinations of properties.