Science of Computer Programming ••• (••••) •••–•••

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Lub: A pattern for fine grained behavior adaptation at runtime

Steven Costiou *, Mickaël Kerboeuf, Glenn Cavarlé, Alain Plantec

Univ. Bretagne-Occidentale, UMR CNRS 6285, Lab-STICC, F-29200 Brest, France

ARTICLE INFO

Article history:
Received 14 January 2017
Received in revised form 19 June 2017
Accepted 19 September 2017
Available online xxxx

Keywords:
Unanticipated adaptation
Dynamic behavior
Dynamic lookup
Runtime evolution

ABSTRACT

Autonomous systems have to evolve in complex environments and their software must adapt to various situations. Although it is common to anticipate adaptations at design time, it becomes a more complex issue when facing unpredictable contexts at runtime, especially if applications cannot be stopped. We introduce Lub, a pattern designed to extend object oriented languages with fine grained unanticipated adaptations. Lub is based on dynamic instrumentation of the lookup, and allows objects to acquire behaviors from another class than their own. A Pharo Smalltalk implementation of Lub is evaluated through a performance analysis and a running example of a fleet of drones facing unexpected GPS problems. Lub is then discussed from the unanticipated software adaptation perspective.

1. Introduction

The need for dynamic behavior adaptation especially arises in applications that need to run continuously, or embedded in autonomous systems like drones or robots. Adaptations may not be easy to predict, for example because of unexpected events coming from a complex environment in which the system is running, or more simply because of bugs. Sometimes these systems cannot be stopped, or the cost of turning them off for an update or for debugging is too high.

An example could be an autonomous fleet of drones flying on a programmed mission. Their software relies on physical sensors, *e.g.* a GPS, to recover and share the fleet's position. This behavior is defined at compile time and it is usually not designed to be modified at runtime. For instance, if one of the drones loses its ability to process the GPS signal and if it is not programmed to operate without it, the whole fleet will be affected and will be unable to fulfill its mission. Debugging the system would be a very interesting option as recalling the drones means canceling the mission. First we would like to see what is happening in the software. A very basic debug technique is to trace on a log what the system does, if this option is available in the running system. If not, this logging behavior has to be defined, added to the running system (without interruption) and removed once it is not needed anymore. Second, once we figured out what is wrong we would like to adapt the behavior of the software, and *try–cancel–retry* new adaptations through the debugging process. These kinds of changes cannot be anticipated.

Unanticipated software evolution [1] makes possible to dynamically adapt a running software. Applications can be instrumented by touching both functional and non-functional behaviors. This opens perspectives in the debugging activity to understand what is happening in the application and to experiment behavioral variations before applying a patch. Al-

* Corresponding author.
E-mail addresses: steven.costiou@univ-brest.fr (S. Costiou), mickael.kerboeuf@univ-brest.fr (M. Kerboeuf), glenn.cavarle@univ-brest.fr (G. Cavarlé), alain.plantec@univ-brest.fr (A. Plantec).

https://doi.org/10.1016/j.scico.2017.09.006
0167-6423/

though there exist adaptive systems, not many of them cope easily with unanticipated adaptation [2]. A key component in these systems is the adaptation mechanism, which must provide enough flexibility and precision to address the problems of unanticipated behavior adaptation [3].

In this paper, we explore and discuss a pattern named *Lub*. We use it to build an adaptation mechanism designed to answer the needs and constraints of unanticipated behavior adaptation in object-oriented software. With Lub, the developer can specify adaptations to extend or modify the protocol of a given object. Adaptations target classes so that any object can individually acquire behaviors from another class than its own. Lub is designed to be a pattern, therefore it can be implemented in any technology provided the host language has the necessary capabilities discussed in this paper.

The main contributions of this paper are:

- The definition of Lub, an object oriented pattern for unanticipated behavior adaptation. The pattern is described in detail, and we specify how it could extend an object-oriented language.
- A performance analysis of a featherlight implementation of Lub with the Pharo language, to analyze its usability in terms of execution speed, adaptation time and memory overhead.
- The evaluation of this implementation through the case study of a drone simulation. The evaluation shows how adaptations can be specified and how they impact the behavior of the running software.
- The discussion of the Lub pattern with regard to the literature.

The remainder of this paper is organized as follows. In section 2 we describe our motivation for unanticipated adaptation, which we illustrate through a simple example with drones. In section 3, we discuss the difficulties of unanticipated dynamic adaptation. Section 4 defines the Lub pattern and section 5 briefly describes the current implementation. Section 6 shows a performance analysis of Lub and section 7 shows an evaluation of Lub in two use case scenarios. Lub and related works are discussed in section 8. Future works are described in section 9 and we conclude this paper in section 10.

2. Motivating example: unanticipated behavior adaptation in a fleet of drones

This section illustrates the need for unanticipated dynamic adaptation through a simple example. Two drones are moving together, close to each other. The first one (*gps-drone*) has an active GPS while the other one (*follower-drone*) uses a communication channel with *gps-drone* to recover its own relative position. The shared GPS of *gps-drone* is used for navigation by the fleet.

2.1. The GPS loss scenario

Since the environment is highly dynamic and therefore not entirely known, it is not possible to predict everything that could happen during the flight. Our scenario is illustrated in Fig. 1. The fleet enters a secured area where all communications must be encrypted. However both drones enter a stand-by mode because *follower-drone* fails to recover its GPS position from the first drone. But *gps-drone* GPS' signal is fine and the communication between the two drones seems alright. The context change was effective in *gps-drone* but not in *follower-drone*, therefore when *follower-drone* asks *gps-drone* its GPS position, *gps-drone* answers encrypted data that *follower-drone* is unable to decipher. The drones are waiting for the situation to be solved: this is a predefined behavior. To change this behavior and to enable adaptation for this particular context, a simple way is to abort the mission and to update the software offline.

This example may happen in a simulation when designing the software, as the use-case we will show in section 7.2. It can also occur in a real mission with autonomous drones. In both cases, restarting the mission can be very expensive: a simulation may have run for a long time, autonomous drones may have flight over a great distance. Stopping a simulation or recalling the drones each time there is a new context that was not anticipated at design time is a problem. In addition, the drones accumulated dynamic states and data until the problem happened: they *lived* in an unpredictable and complex environment. So there are no guarantees one could reproduce the problem in development mode due to the impossibility to recreate or simulate the exact context and conditions under which it originally happened. This use-case can be classified as a bug and as such it remains unforeseen at design time.

2.2. The unanticipated adaptations

The fleet is stalled because one of the drones failed to adapt to the changing conditions (the entering of the unsecured area). We know exactly what happens here, when we describe our use-case in this paper, but it is not obvious that a human operator would easily understand the problem when it happens. From his point of view, he is facing an unexpected problem with no obvious reason, and a restricted set of options. Either the drones are recalled and their software and data analyzed offline, either the operator has to dynamically update their behavior. The software itself cannot adapt, as we made the assumption in section 2.1 that this particular case was unforeseen at design time.

Please cite this article in press as: S. Costiou et al., Lub: A pattern for fine grained behavior adaptation at runtime, Sci. Comput. Program. (2017), https://doi.org/10.1016/j.scico.2017.09.006