Parametric lenses: change notification for bidirectional lenses

László Domoszlai
Radboud University Nijmegen, Netherlands, ICIS, MBSD
dlacko@gmail.com

Bas Lijnse Rinus Plasmeijer
Radboud University Nijmegen, Netherlands, ICIS, MBSD
b.lijnse@cs.ru.nl,rinus@cs.ru.nl

Abstract

In most complex applications it is inevitable to maintain dependencies between the different subsystems based on some shared data. The subsystems must be able to inform the dependent parties that the shared information is changed. As every actual notification has some communication cost, and every triggered task has associated computation cost, it is crucial for the overall performance of the application to reduce the number of notifications as much as possible. To achieve this, one must be able to define, with arbitrary precision, which party is depending on which data. In this paper we offer a general solution to this general problem. The solution is based on an extension to bidirectional lenses, called *parametric lenses*. With the help of parametric lenses one can define compositional *parametric views* in a declarative way to access some shared data. Parametric views, besides providing read/write access to the shared data, also enable to *observe* changes of some parts, given by an explicit parameter, the *focus domain*. The focus domain can be specified as a *type-based query language* defined over one or more resources using predefined combinators of parametric views.

1. Introduction

Complex applications commonly have to deal with shared data. It is often confined to the use of a relational database coupled with a simple concurrency control method, e.g., optimistic concurrency control [2]. In other cases, when a more proactive behavior is required, polling or some ad hoc notification mechanism can be invoked. At the farther end of the range there are some very involved applications (multi-user applications, workflow management systems, etc.), which are based on interdependent tasks connected by shared data. In the most general case, one has to deal with complex task dependencies defined by shared data coming from diverse sources, e.g. different databases, shared memory, shared files, sensors, etc.

As an example, consider the following case which is based on a prototype we have developed for the Dutch Coastguard [3]; it will be used throughout the paper to introduce the problem, and the concepts of the proposed solution. We have a small database which acts as a source of data of ships: name, cargo capacity, last known position, etc. The positions of the ships are updated repeatedly as the ships move; ships have a transponder on board which send their latest position on a regular basis. As a basic task, we simply want to show the positions of the ships on a map, of which users are allowed to select an area to view, the *focus* of their interest, the *focus domain*. In this setting we can think of map instances and update processes as interdependent tasks that are connected by the data of ships they share. When the position of a ship is updated in the database, the map instances, of which focus domain covers the old or the new coordinates, must be refreshed.

From a theoretical perspective, it would be correct behavior to notify every map instance on ship movement. However, this leads to huge efficiency issues in practice. There are many thousands of ships in the North Sea constantly moving around. Only those map instances need to be refreshed on which area the position of a ship is changed. As every actual notification has some communication cost, and every triggered task has associated computation cost, it is crucial for the overall performance of the application to reduce the number of notifications as much as possible. Thus, we need a notification system which, for efficiency reasons, is as accurate as possible.

As the problem described above is a very common computational pattern, we would like to offer a general, reusable solution. In addition, we would like to solve it *efficiently* enough to be comparable with the ad-hoc solutions.

From the computational perspective, focusing on a specific domain of the underlying data can be achieved by creating and working with one of its abstract views. Lenses [1] are commonly used for creating abstract views. They can be used to support partial reading and writing, for access restriction or to provide a specific view of the data. Lenses enable to define bi-directional transformations. In a nutshell, a lens describes a function that maps the input to an output (called *get*) and backwards (called *put*). The *get* function maps the input to some output, while the *put* function maps the modified output, together with the original input, to a modified input:

$$get \in X \to Y$$
$$put \in Y \times X \to X$$

In our example two kind of abstract views are needed for serving different processes: one to show the ships located in a given area of the map, and another one for the update process, which periodically updates the coordinates of a ship in the database.

The general notification problem is as follows. Given is a set of shared data sources of any type (*A* and *B* in the picture) holding a set of data (D_A, D_B). There are also given some lenses defined on top of the data sources and on each other. These are L_1, L_2, L_3 and L_4 in the picture. One typical question can be, e.g., whether a given update through L_4 affects the L_1 view or not? What about the other way around?

Unfortunately, lens theory does not say anything about how to discover when an update get issued through some lens may effect

IFL '14, October 1–3, 2014, Boston, Massachusetts, US.

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

the data seen through some other. In this paper we present a general extension to lenses as a solution for this general problem. In this extension, called *parametric lenses*, lenses are partially defunctionalized to extract a first-order parameter (the focus domain: ϕ, ψ) that groups a set of similar lenses into a single parametric lens in which the parameter essentially encodes which part of the source domain is mapped to the view domain by the lens. Parametric lenses also return a predicate in the *put* direction. This predicate, called the *invalidation function*, encodes some semantic information associated with the actual focus domain, and enables the engine to decide which domains are affected by a change of data. It tells whether the particular update of focus $p \in \phi$ affects a *get* operation of a given focus $q \in \phi$. With other words, it says whether the value of a previous read of some focus *q* is still valid or not. It returns *true* to indicate that the given focus must be re-read to be up to date.

$$get_F \in \phi \times X \to Y_\phi$$
$$put_F \in \phi \times Y_\phi \times X \to X \times (\phi \times Bool)$$

In our example, the focus domain is a type which enables to specify the area of the map one wants to focus on; the invalidation function then would predicate whether two values of the focus domain, two areas, overlap or not.

Pure parametric lenses cannot be applied to some shared data directly, therefore they attached to the shared data through a *non-pure* abstract interface called *parametric view*. The parametric views are allowed to be composed using predefined combinators. Using these combinators, one is able to specify the focus domain as a *type-based query language* defined over one or more resources. With the query language, one can focus on a specific part of the underlying shared data during reading, writing, or it can be used for notification purposes.

We offer the following contributions in the paper:

1. We introduce parametric lenses as a general extension to lenses which enables to develop efficient notification libraries for them;
2. Parametric lenses are embedded into compositional parametric views which are defined over shared data;
3. The executable semantics, using Haskell [4], of the combinators and an underlying notification engine is provided and explained. The complete Haskell implementation along with the example developed in the paper can be found at https://wiki.clean.cs.ru.nl/File:Pview.zip;
4. Parametric lenses have been introduced in the iTask coastguard case study described above.

References

[1] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM TPL*, 29(3), May 2007. ISSN 0164-0925. . URL `http://doi.acm.org/10.1145/1232420.1232424`.

[2] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, June 1981. ISSN 0362-5915. . URL `http://doi.acm.org/10.1145/319566.319567`.

[3] B. Lijnse, J. Jansen, R. Nanne, and R. Plasmeijer. Capturing the netherlands coast guard's sar workflow with itasks. In D. Mendonca and J. Dugdale, editors, *Proceedings of the 8th International Conference on Information Systems for Crisis Response and Management, ISCRAM '11*, Lisbon, Portugal, May 2011.

[4] S. L. Peyton Jones. *The Implementation of Functional Programming Languages (Prentice-Hall International Series in Computer Science)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN 013453333X.