Editlets: type based client side editors for iTasks

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

The iTask framework is for the construction of distributed systems where users work together on the internet. It offers a domain specific language for defining applications, embedded into the lazy functional language Clean. From the mere declarative specification a complete multi-user web application is generated. Although the generated nature of the user interface entails a number of benefits for the programmer, it suffers from the lack of possibility to create custom UI building blocks. In a precursory work we proposed *tasklets* for the development of custom, interactive web components. However, as tasklets are implemented as a computational element, a *task*, they lack some fundamental properties limiting their usability; these are compositionality and the capability of two-way communication between the clients and the server. In this paper, we introduce *editlets* to overcome these limitations. In addition, editlets also provide a general way to communicate in *edits* instead of exchanging the whole data; it does not just help with reducing the communication cost, but also enables multiple clients to work on the same shared data with minimizing the risk of conflicting updates.

1. Introduction

Task Oriented Programming [5, 7] (TOP) is a paradigm that is designed to construct multi-user, distributed, web-applications. The *iTask system* [6] (iTasks) is a TOP framework that offers a domain specific language embedded in the pure, lazy functional language Clean.

According to the TOP paradigm, the unit of application logic is a task. Tasks are abstract descriptions of interactive persistent units of work that have a typed value. When a task is executed, it has an opaque persistent value, which can be observed by other tasks in a controlled way. In iTasks, complex multi-user interactions can be programmed in a declarative style just by defining the tasks that have to be accomplished. The specification of the tasks is given by a domain specific language (DSL). Furthermore, the specification is given on a very high level of abstraction and does not require the programmer to provide any user interface definition. Merely by defining the workflow of user interaction, a complete multi-user web application is generated, all the details e.g. the generation of web user interface, client-server communication, state management etc. are automatically taken care of by the framework itself.

The iTask system uses generic programming [1, 4] and a hybrid static-dynamic type system [8, 9] to generate the user interface. From the programmers perspective, it is achieved in two levels. In the most basic level, the iTasks engine can be asked to generate user interface for any conceivable first order model type. iTasks uses a predefined set of primitive user interface elements to generate the GUI, a client side *editor*, for the given type, then dynamically creates an associated primitive task. On the higher level, additional user interface elements are generated as tasks are combined together. These elements reflect the actual combinators in use and express the "flow" of the application.

Developing web applications such a way is straightforward in the sense that the programmers are liberated from these cumbersome and error-prone jobs, such that they can concentrate on the essence of the application. The iTask system makes it very easy to develop interactive multi-user applications. The down side is that one has only limited control over the customization of the generated user interface. In real world applications, it is often necessary to develop custom user interface elements to achieve special functionality.

To overcome this limitation, in a previous work we introduced *tasklets* [2], a special primitive task type, for the development of custom, interactive web components. Tasklets are written in Clean and executed in a web browser using a Clean to JavaScript compilation technique [3]. In the browser, they have unlimited access to browser resources through some library functions while on the server they behave like ordinary iTasks tasks.

Using tasklets, we have successfully developed many interactive components for a wide range of applications, but we also experienced certain limitations of the technology. These are the following:

1. Tasklets cannot work with shared data. As an example, it is not possible to create an interactive map, and enable multiple users to make concurrent modifications to that (e.g. add marks). This limitation goes against the main principle of iTasks.
2. There is no way for two-way communication between the client and the server part. Tasklets implement task interface which enables the inspection of task values, the behavior of a task cannot be influenced after its evaluation is started.
3. There is only limited compositionality at task level. Generated editors cannot contain custom elements as they are primitive tasks which cannot contain other tasks.

In this paper we rethought how to create interactive components. We found that attaching the presentation logic to a type has many advantages over our previous approach. The new type of interactive elements are called *editlets* as they work on the lower editor level instead of task level as *tasklets* do. Editlets solve all the

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

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

aforementioned limitations while preserving compatibility: in the most basic use cases they give back the functionality of tasklets.

Editlets also have the property that the client-server communication is done in *edits*, which means that the value of the editlet is communicated through changes instead of exchanging the whole value at every update. In certain cases it does not just reduces drastically the communication cost (just think of a source code editor component), but also allows us to avoid update conflicts when working on shared data.

In this paper we show how editlets can be defined, how they work and interact with the other part of the iTask system. This is done in a number of steps:

1. We extend iTask with editlets. An editlet consists of the type of the value it holds, a type of the edits in use, a description of the behavior of the component on the client side, and the logic of creating and applying edits from and to its current value.
2. We develop a simple, but still realistic example of a drawing applications, where multiple people can work on the same image on the same shared image to give a taste of editlets.
3. We explain the technical background of editles along with additional remarks how they fit the iTasks architecture.

References

[1] A. Alimarine. *Generic functional programming: conceptual design, implementation and applications*. PhD thesis, Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands, 2005.

[2] L. Domoszlai and R. Plasmeijer. Tasklets: Client-side evaluation for iTask3. In *Domain specific languages, summer school*, DSL'13, 2014. Accepted for publication.

[3] L. Domoszlai, E. Bruël, and J. Jansen. Implementing a non-strict purely functional language in JavaScript. *Acta Universitatis Sapientiae*, 3:76–98, 2011. URL `http://www.acta.sapientia.ro/acta-info/C3-1/info31-4.pdf`.

[4] R. Hinze. A new approach to generic functional programming. In T. Reps, editor, *Proceedings of the 27th International Symposium on Principles of Programming Languages, POPL '00, Boston, MA, USA*, pages 119–132. ACM Press, 2000.

[5] B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. PhD thesis, Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands , 2013. ISBN 978-90-820259-0-3.

[6] R. Plasmeijer, P. Achten, P. Koopman, B. Lijnse, T. Van Noort, and J. Van Groningen. iTasks for a change: Type-safe run-time change in dynamically evolving workflows. In *PEPM '11 : Proceedings Workshop on Partial Evaluation and Program Manipulation, PEPM '11, Austin, TX, USA*, pages 151–160, New York, 2011. ACM.

[7] R. Plasmeijer, B. Lijnse, S. Michels, P. Achten, and P. Koopman. Task-Oriented Programming in a Pure Functional Language. In *Proceedings of the 2012 ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, PPDP '12*, pages 195–206, Leuven, Belgium, Sept. 2012. ACM. ISBN 978-1-4503-1522-7.

[8] T. van Noort. *Dynamic Typing in Type-Driven Programming*. PhD thesis, Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands , May 2012. ISBN 978-94-6108-279-4.

[9] A. v. Weelden. *Putting types to good use*. PhD thesis, Institute for Computing and Information Sciences, Radboud University Nijmegen, The Netherlands, Oct. 17, 2007.