Task Oriented Programming with Purely Compositional Interactive Vector Graphics

Peter Achten
Radboud University Nijmegen, Netherlands, ICIS, MBSD
P.Achten@cs.ru.nl

Jurriën Stutterheim László Domoszlai
Rinus Plasmeijer
Radboud University Nijmegen, Netherlands, ICIS, MBSD
j.stutterheim@cs.ru.nl,dlacko@gmail.com,rinus@cs.ru.nl

1. Abstract

Task Oriented Programming [24, 29] (TOP) is a paradigm that is designed to construct multi-user, distributed, web-applications. The *iTask system* [28] (iTasks) is a TOP framework that offers three core concepts for software developers.

- *Tasks* which are abstractions of the *work* that needs to be performed by (teams of) human(s) and software components. A task is a value of parameterized type `(Task a)`. The type parameter `a` models the *task value* the task is currently processing. This value can be inspected by other tasks.
- *Shared data sources* (SDS) which are abstractions of *information* that is shared between tasks. A SDS is a value of parameterized type `(ReadWriteShared r w)`. The type parameters `r` and `w` model the *read* and *write* values.
- *Combinator functions* that compose tasks and SDSs into more complex tasks and SDSs and combinations of them.

The iTask system is a domain specific language (DSL) that is shallowly embedded in the strongly typed, lazy, purely functional programming language Clean [27, 30]. When developing an iTask application, the task developer can concentrate on identifying the tasks, the shared data sources, and their interrelation. The iTask system uses generic programming [5, 21] and a hybrid static-dynamic type system [31, 32] to generate all required machinery to create an executable. Among the plethora of concerns, the iTask system automatically generates a graphical user interface (GUI) for any conceivable first order model type. For this purpose the iTask system offers a comprehensive set of data types that model common user interface elements. In this way the task developer needs no working knowledge of JavaScript, HTML 5.0, handling (de)serialization and events. However, this knowledge *is* required whenever the comprehensive set of model types does not cover a particular interface element. This is unfortunate because it breaks the level of abstraction that is offered by the iTask system.

In this paper we show how the level of abstraction of iTask can remain intact when task developers define new user interface elements. This is done in a number of steps:

- We extend iTask with *Images* which are *vector graphics* based renderings. An image of type `(Image m)` is a rendering of a model value of type `m`. Images have a span to specify their dimensionality and local coordinate-system (traditional running from left-to-right and top-to-bottom), but there is no global coordinate system in which they are positioned or global canvas on which they are painted.
- We add *combinator functions* to compose images into more complex images. The absence of a global coordinate system or global canvas allows us to provide only three layout primitives: the *overlay* (placing images on top of each other), *grid* (two-dimensional structured layout), and *collage* (two-dimensional arbitrary layout). Each layout primitive has an optional *host* image that determines a reference span that is used for layout.
- We obtain *interactivity* by integrating images in iTask. Any (composite) image of type `(Image m)` can react to user events and define its behaviour via a pure function of type `(m -> m)` that alters the image's model value. This is in accordance to the philosophy of tasks: behaviour only needs to be defined in terms of how tasks depend on the model value of tasks and images.

The implementation of images and its combinator functions in iTask is based on the Scalable Vector Graphics (SVG) standard [10]. The low-level integration of these images in iTask is structured by means of *editlets*, and the high-level integration is done via the iTask *step* task combinator function.

Compositional Images

The full paper contains a detailed explanation and motivation for the compositional image library. Figure 1 displays the key elements of the API. For this abstract, we state the key properties of the API.

- Think of a basic image as an overhead-projector slide that is infinitely large. This slide can be rotated, scaled, and skewed. A finite portion of the basic image has visual content, the extent of which is defined by its *span*. The *x-span* always extends from left to right, and the *y-span* always extends from top to bottom.
- Think of a composite image as a stack of overhead-project slides. This stack can be rotated, scaled, and skewed. When composing images, their span is used to control their relative location. There are three core image combinator functions: *overlay* to stack images, *grid* to stack images row-by-row or column-by-column, and *collage* to stack images and arrange

them to your liking. The commonly occurring layouts *beside* and *above* are direct specializations of *grid*.

- The layout combinators have an optional *host image* parameter. Think of the host image as the background image relative to which the other images are to be arranged in terms of alignment.
- Images can have *tags*. This is needed when expressing spans in terms of the span(s) of other parts of the image.
- A (composite) image of type `(Image m)` can be made *interactive* by attributing it with a pure function of type `(m -> m)`, thus resulting in a change of image model value. This function is evaluated whenever the user clicks in the image (regardless of the location and transformation of the image).

```
:: Image m      // Opaque type
:: Span         // Opaque type
:: Host  m      :== Maybe (Image m)
:: ImageTag     :== String
:: FontDef      :== String
:: ImageOffset :== (Span, Span)

:: XAlign        = AtLeft | AtMiddleX | AtRight
:: YAlign        = AtTop  | AtMiddleY | AtBottom
:: ImageAlign  :== (XAlign, YAlign)

:: GridDimension = Rows Int | Columns Int
:: GridLayout  :== (GridXLayout, GridYLayout)
:: GridXLayout   = LeftToRight | RightToLeft
:: GridYLayout   = TopToBottom | BottomToTop

:: ImageLayout m :== [ImageOffset] [Image m] (Host m) -> Image m
overlay  :: [ImageAlign]                            -> ImageLayout m
beside   :: [YAlign]                                -> ImageLayout m
above    :: [XAlign]                                -> ImageLayout m
grid     :: GridDimension GridLayout [ImageAlign] -> ImageLayout m
collage  ::                                            ImageLayout m

empty    :: Span Span           -> Image m
text     :: FontDef String      -> Image m
circle   :: Span                -> Image m
ellipse  :: Span Span           -> Image m
rect     :: Span Span           -> Image m

:: Slash = Slash | Backslash

xline    :: Span                -> Image m
yline    :: Span                -> Image m
line     :: Slash Span Span     -> Image m
polygon  :: [ImageOffset]       -> Image m
polyline :: [ImageOffset]       -> Image m

rotate   :: Real      (Image m) -> Image m
fit      :: Span Span (Image m) -> Image m
fitx     :: Span      (Image m) -> Image m
fity     :: Span      (Image m) -> Image m
skewx    :: Real      (Image m) -> Image m
skewy    :: Real      (Image m) -> Image m

px          :: Real             -> Span
ex          :: FontDef          -> Span
descent     :: FontDef          -> Span
textxspan   :: FontDef String   -> Span
imagexspan  :: [ImageTag]       -> Span
imageyspan  :: [ImageTag]       -> Span
columnspan  :: [ImageTag] Int   -> Span
rowspan     :: [ImageTag] Int   -> Span
```

Figure 1. The key elements of the Image API.

Integration in iTask

The integration of interactive, compositional images in iTask concerns the following components:

- The images are mapped to SVG. We face two major hurdles: *(i)* SVG adopts an imperative-style rendering model, so we must take care to unravel the declarative image specifications and paint them in the right order in SVG; *(ii)* text dimensions can only be computed at the client-side of the application, so the layout of images can not be performed entirely on the server-side of iTask.
- To establish the server-client side communication, we use iTask *editlets*.

These will be described in detail in the full paper.

Case studies

We demonstrate the new iTask approach by means of the following case studies:

- a 1-person pocket calculator,
- a 2-person, distributed, *tic-tac-toe* game,
- a 2-person, distributed, *trax* game [1],
- a N-person, distributed, *ligretto* card game.

Related work

Functional programming and GUIs share a long research history [2–4, 6–9, 11–19, 22, 23, 25, 26]. The full paper compares and discusses these approaches in more detail. For this abstract we restrict ourselves to the following observations:

- Regarding compositional images, the work by Henderson [19, 20] has been influential to many compositional approaches, as well as ours. Similar to Henderson's approach, we abstract from absolute location, but we do not from size. In the context of scalable vector graphics, the latter is not an issue because at any time images can be resized to any demanded size.
- Regarding compositional GUIs, Haggis [15] is similar in their approach to layout and transform GUIs. A difference is that Haggis has a monadic flavour: the GUI elements that are to be combined need to be declared *before* their handles can be used to arrange them inside layouts. In our approach, the iTask system 'collects' the offered images in the task specifications.
- Regarding 'completeness', we have not yet made use of all graphics elements that are offered by SVG. Concepts that are currently missing but are intended to be included in the iTask system are Bézier curves, multi-line text blocks, gradients, generalized clipping, and filtering. The layout combinators that we propose were inspired by the Racket image API [14]. The three core layout primitives *overlay*, *grid*, and *collage* of our approach can model them. The current proposal's event model is certainly incomplete as it covers only user-mouse clicks. We expect that extending the model to deal with the usual set of mouse and keyboard events follows the same approach.

Conclusions

In the TOP iTask framework multi-user, distributed, web-applications can be developed on a high level of abstraction because the task developer can concentrate on identifying and specifying the required tasks, information, and how they relate, knowing that the iTask framework can generate a suitable web application. The paper shows how this property can also be satisfied when developing applications that require custom built user interface(element)s. Because images are compositional, the task developer can concentrate on identifying and specifying the required graphical elements, knowing that the image library generates a suitable SVG rendering. Via editlets graphically customized tasks are integrated seamlessly in the TOP paradigm.

References

[1] P. Achten. Why functional programming matters to me. In P. Achten and P. Koopman, editors, *The Beauty of Functional Code - Essays Dedicated to Rinus Plasmeijer on the Occasion of His 61st Birthday, Festschrift*, number 8106 in LNAI, pages 79–96. Springer, August 2013. ISBN ISBN 978-3-642-40354-5.

[2] P. Achten and S. Peyton Jones. Porting the Clean Object I/O library to Haskell. In M. Mohnen and P. Koopman, editors, *Selected Papers of the 12th International Workshop on the Implementation of Functional Languages, IFL '00*, volume 2011 of *LNCS*, pages 194–213. Springer-Verlag, Sept. 2001.

[3] P. Achten and R. Plasmeijer. The ins and outs of Concurrent Clean I/O. *Journal of Functional Programming*, 5(1):81–110, 1995.

[4] P. Achten and R. Plasmeijer. Interactive functional objects in Clean. In C. Clack, K. Hammond, and T. Davie, editors, *Selected Papers of the 9th International Workshop on the Implementation of Functional Languages, IFL '97*, volume 1467 of *LNCS*, pages 304–321. Springer-Verlag, Sept. 1998.

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

[6] M. Carlsson and T. Hallgren. Fudgets - a graphical user interface in a lazy functional language. In *Proceedings of the 6th International Conference on Functional Programming Languages and Computer Architecture, FPCA '93*, Kopenhagen, Denmark, 1993.

[7] K. Claessen, T. Vullinghs, and E. Meijer. Structuring graphical paradigms in TkGofer. In *Proceedings of the 2nd International Conference on Functional Programming, ICFP '97*, volume 32(8), pages 251–262, Amsterdam, The Netherlands, 9-11, June 1997. ACM Press.

[8] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links: web programming without tiers. In *Proceedings of the 5th International Symposium on Formal Methods for Components and Objects, FMCO '06*, volume 4709, CWI, Amsterdam, The Netherlands, 7-10, Nov. 2006. Springer-Verlag.

[9] A. Courtney and C. Elliott. Genuinely functional user interfaces. In *Proceedings of the 5th Haskell Workshop, Haskell '01*, Sept. 2001.

[10] E. Dahlström, P. Dengler, A. Grasso, C. Lilley, C. McCormack, D. Schepers, and J. Watt. Scalable vector graphics (svg) 1.1 (second edition). Technical Report REC-SVG11-20110816, W3C Recommendation 16 August 2011, 2011.

[11] A. Dwelly. Functions and dynamic user interfaces. In *Proceedings of the 4th International Conference on Functional Programming Languages and Computer Architecture, FPCA '89*, pages 371–381, Sept. 1989.

[12] C. Elliot. Tangible functional programming. In *Proceedings of the 12th International Conference on Functional Programming, ICFP '07*, pages 59–70, Freiburg, Germany, 1-3, Oct. 2007. ACM Press. ISBN 978-1-59593-815-2.

[13] M. Elsman and N. Hallenberg. Web programming with SMLserver. In *Proceedings of the 5th International Symposium on the Practical Aspects of Declarative Programming, PADL '03*. New Orleans, LA, USA, Springer-Verlag, Jan. 2003.

[14] M. Felleisen, R. Findler, M. Flatt, and S. Krishnamurthi. A Functional I/O System * or, Fun for Freshman Kids. In *Proceedings International Conference on Functional Programming, ICFP '09*, Edinburgh, Scotland, UK, 2009. ACM Press.

[15] S. Finne and S. Peyton Jones. Composing Haggis. In *Eurographics Workshop on Programming Paradigms in Graphics*, pages 85–101, Maastricht, the Netherlands, 1995. Springer.

[16] P. Graunke, R. Findler, S. Krishnamurthi, and M. Felleisen. Modeling web interactions. In P. Degano, editor, *Proceedings of the 12th European Symposium on Programming, ESOP '03*, volume 2618 of *Lecture Notes in Computer Science* , pages 238–252, 7-11, Apr. 2003.

[17] M. Hanus. High-level server side web scripting in Curry. In *Proceedings of the 3rd International Symposium on the Practical Aspects of Declarative Programming, PADL '01*, pages 76–92. Springer-Verlag, 2001.

[18] M. Hanus. Type-oriented construction of web user interfaces. In *Proceedings of the 8th International Conference on Principles and Practice of Declarative Programming, PPDP '06*, pages 27–38. ACM Press, 2006.

[19] P. Henderson. Functional geometry. In D. Friedman and D. Wise, editors, *Conference Record of the 1982 ACM Symposium on Lisp and Functional Programming*, pages 179–187, Pittsburgh, Pennsylvania, 1982. ACM Press. URL `http://www.ecs.soton.ac.uk/~ph/funcgeo.pdf`.

[20] P. Henderson. Functional geometry. *Higher-Order and Symbolic Computation*, 15:349–365, 2002.

[21] 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.

[22] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson. Arrows, robots, and functional reactive programming. In J. Jeuring and S. Peyton Jones, editors, *Proceedings of the 4th International Summer School on Advanced Functional Programming, AFP '03*, volume 2638 of *Lecture Notes in Computer Science* , pages 159–187. Oxford, UK, Springer-Verlag, 2003.

[23] D. Leijen. wxHaskell: a portable and concise GUI library for Haskell. In *Proceedings of the 2004 ACM SIGPLAN workshop on Haskell*, pages 57–68, Snowbird, Utah, USA, 2004. ACM. . URL `http://doi.acm.org/10.1145/1017472.1017483`.

[24] 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.

[25] F. Loitsch and M. Serrano. Hop client-side compilation. In *Proceedings of the 7th Symposium on Trends in Functional Programming, TFP '07*, pages 141–158, New York, NY, USA, 2-4, Apr. 2007. Interact.

[26] M. Morazán. Functional Video Games in the CS1 Classroom. In R. Page, Z. Horváth, and V. Zsók, editors, *Proceedings of the 11th Symposium on Trends in Functional Programming, TFP '10*, volume 6546 of *LNCS*, pages 166–183, 2010.

[27] R. Plasmeijer and M. van Eekelen. Clean language report (version 2.1). `http://clean.cs.ru.nl`, 2002.

[28] 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.

[29] 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.

[30] J. van Groningen, T. van Noort, P. Achten, P. Koopman, and R. Plasmeijer. Exchanging sources between Clean and Haskell: a double-edged front end for the Clean compiler. In J. Gibbons, editor, *Haskell'10 : proceedings of the third ACM Haskell symposium on Haskell*, pages 49–60. ACM, 2010.

[31] 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.

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