Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 8
Tampere University of Technology. Department of Software Systems. Report 8

Antero Taivalsaari & Tommi Mikkonen

Simplifying Interactive Programming with Keywords 'that' and 'those'

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 8
Tampere University of Technology. Department of Software Systems. Report 8

Antero Taivalsaari & Tommi Mikkonen

Simplifying Interactive Programming with Keywords 'that' and 'those'

Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos
Tampere 2009

ISBN 978-952-15-2284-0 (PDF)
ISSN 1797-836X

Simplifying Interactive Programming with Keywords '`that`' and '`those`'

Antero Taivalsaari
Tampere University of Technology
Korkeakoulunkatu 1
FI-33720 Tampere
FINLAND
antero.taivalsaari@tut.fi

Tommi Mikkonen
Tampere University of Technology
Korkeakoulunkatu 1
FI-33720 Tampere
FINLAND
tommi.mikkonen@tut.fi

ABSTRACT
Most object-oriented programming and scripting languages provide a keyword called `self` or `this` that allows applications to refer to the variables and functions of the current object instance. In this paper we introduce two new keywords `that` and `those` to facilitate interactive programming. The key idea behind these keywords is to make it easy to refer programmatically to those objects that are currently under manipulation in the graphical user interface.

Keywords
Graphical user interfaces, dynamic languages, interactive programming, scripting.

1.INTRODUCTION
The software industry is currently experiencing a paradigm shift towards web-based software. Applications that were conventionally written for specific operating systems, CPU architectures or devices are now increasingly targeted to the Web, to be downloaded and executed inside the web browser. Web applications are far more dynamic than conventional desktop applications, e.g., in that they usually require no compilation, binaries or explicit installation.

The emergence of the Web as an application platform has led to a renaissance of dynamic languages. A whole new generation of programmers are growing up with languages such as JavaScript, Perl, PHP, Python or Ruby. The attention that dynamic languages are receiving is remarkable, and is something that has not been witnessed since the early days of personal computers and the widespread use of the BASIC programming language in the 1970s and 1980s. Applications that used to be written using static programming languages such as C, C++ or Java, are now commonly built with dynamic languages such as JavaScript.

One of the key characteristics of dynamic programming languages is support for interactive development. With a dynamic language, software development and testing (and deployment) can occur seamlessly within a uniform graphical user interface, as demonstrated originally by the Smalltalk programming environment [2]. In the context of the Web, it is the web browser that is now used not only for executing applications but also for developing them interactively. Examples of such interactive web-based development environments include Microsoft Popfly [3], Sun Labs Lively Kernel [13], Yahoo Pipes [6] and the Lively for Qt system [8] that we use in the examples shown in this paper.

Most object-oriented programming and scripting languages provide a keyword called `self` or `this` that allows the application to refer to the variables and functions of the current object instance (in Smalltalk parlance: the current message receiver). In this paper, we introduce two new keywords `that` and `those` to facilitate interactive programming. The key idea behind these "demonstrative pronouns" is to make it easy to refer programmatically to those objects that are currently under manipulation in the user interface. The main benefit of the proposed new keywords is that they simplify the interplay between the graphical user interface and source code that is written and evaluated interactively.

2.THAT
Most computer systems today are based on the WIMP (Windows, Icons, Menus and a Pointing device) metaphor invented at Xerox PARC in the 1970s and popularized by the Macintosh computer in the 1980s. A pointing device (such as a mouse or a stylus) is used for choosing the object that the user wants to manipulate (e.g., to move or resize the object).

Our proposal is to associate a keyword called `that` at the source code level with the object that has been manipulated in the user interface most recently. Such a keyword can be extremely useful especially for interactive programming, scripting and debugging. Consider the following example. In Figure 1, we show a starting position in which there are two objects on the screen: a digital clock and a simple gray rectangle. In addition, there is a *source code evaluator* that allows the user to enter source code (in this case: JavaScript source code) and evaluate that code on the fly.

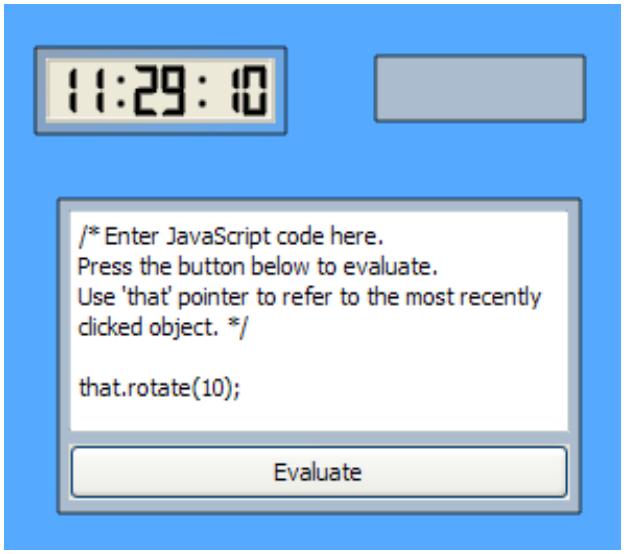

Figure 1. `that` keyword illustrated: starting position

In Figure 2, the user has clicked on the gray rectangle with the mouse – thereby implicitly setting the `that` pointer to point to the gray rectangle – and then pressed the "*Evaluate*" button of the source code evaluator. (Note: We assume that clicking on the code evaluator itself does not change the value of the `that` pointer.) The JavaScript code snippet "`that.rotate(10);`" shown in the evaluator was then executed to rotate the gray rectangle by ten decimal degrees, as illustrated in Figure 2.

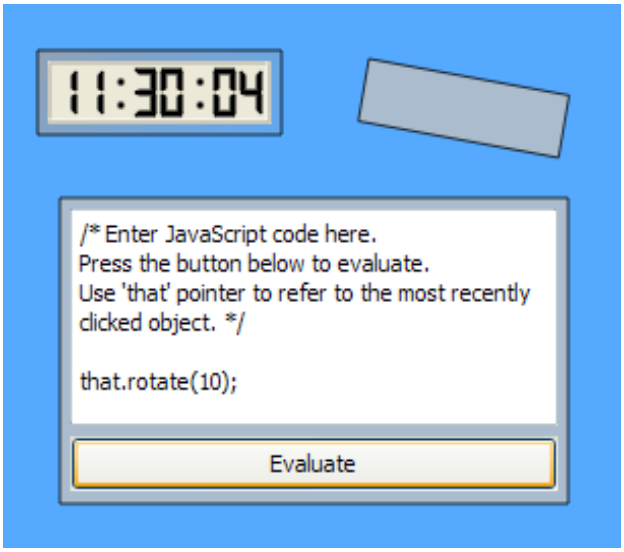

Figure 2. `that` keyword illustrated: the situation after clicking the gray rectangle and then invoking "*Evaluate*"

In Figure 3, the user has clicked on the digital clock and then pressed the "*Evaluate*" button again. As seen in Figure 3, now the digital clock has been rotated as well.

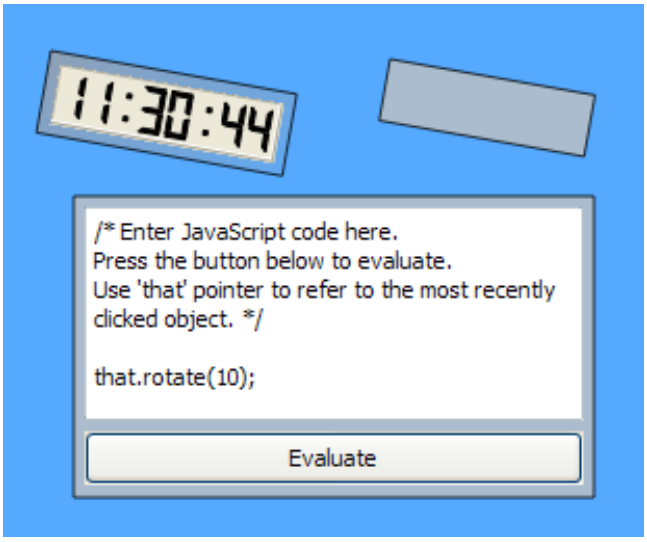

Figure 3. `that` keyword illustrated: the situation after clicking the digital clock and invoking "*Evaluate*" again

The `that` keyword has various use cases. One of the most practical use cases is to utilize the `that` keyword to capture pointers to objects on the screen. This is especially convenient in situations in which the user has constructed a complex object interactively (e.g., a graphical window consisting of multiple panes and other substructures), and the user would then want to pick one of those substructures as a target for interactive scripting. A simple example of such use is shown in Figure 4.

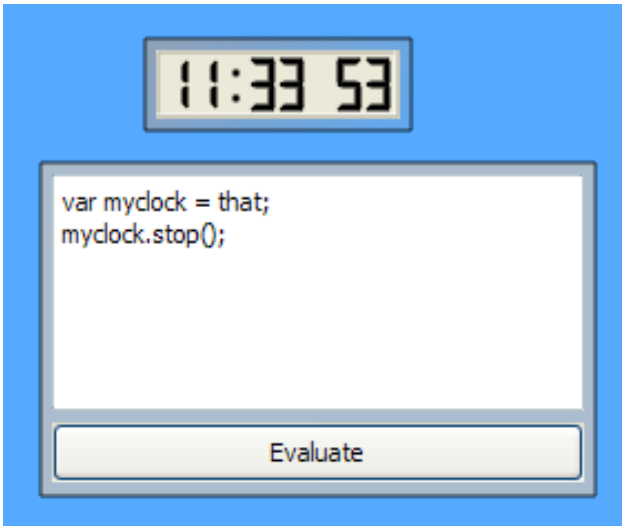

Figure 4. Using `that` keyword to create a programmatical reference to an object

In Figure 4, we assume that the user has clicked on the digital clock object, thereby implicitly setting the `that` pointer to refer to the digital clock. At the source code level, the user then assigns the current value of `that` to a new variable called `myclock`, thereby creating a programmatical reference to the digital clock object. The `myclock` variable can then subsequently be used for manipulating the clock from other interactive scripts and elsewhere in source code. Variable `myclock` will remain pointing to the digital clock even when the value of `that` will later change.

Figure 5 shows a similar example, except that in this case the user has clicked on a substructure of a more complex graphical object in order to manipulate the substructure programmatically. More specifically, in this example the programmer wants to obtain a reference to a map view component that is inside an application that contains various other components and widgets as well.

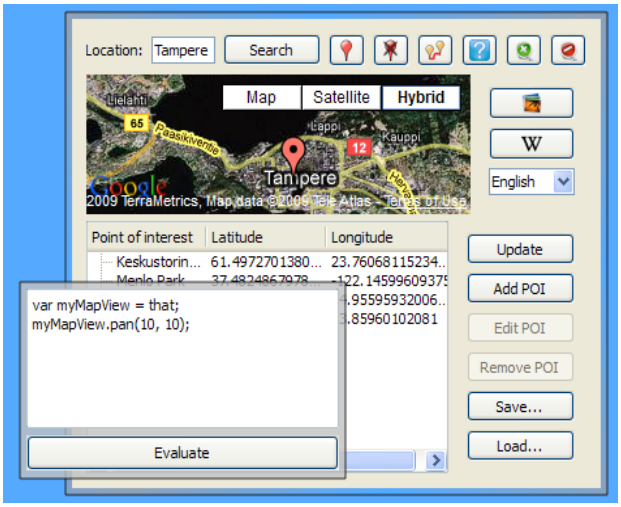

Figure 5. Using `that` keyword to obtain a reference to a substructure of an application

In the example shown in Figure 5, the programmer assigns the current value of the `that` pointer to a variable called `myMapView`, and then subsequently utilizes the `myMapView` variable to pan the map view to the right and down by ten pixels.

Without the `that` keyword, the process of picking up pointers to arbitrary substructures in the graphical user interface can be really tedious. In a typical situation, the user would have to open a separate tool – such as a class browser or object inspector – to visualize the internal structure of the map view component. By utilizing the information displayed by such a tool, the programmer could then manually traverse to the specific location in the application structure, and create a direct pointer to the desired object. Usually, the programmer would have to type a rather long path name (something like `myapp.subcomponent.subsubcomponent.subsubsub.map view`) to programmatically obtain a reference to the desired substructure. The presence of the `that` keyword makes it possible to avoid such hassles altogether.

3.THOSE

An important additional characteristic of a graphical user interface based on the WIMP metaphor is *multiple selection*. For instance, in Mac OS, Linux or Windows applications, the user commonly clicks and then drags the mouse to perform text selection or to select a number of objects in a file browser or graphics editor.

Our proposal is to associate the objects chosen using multiple selection with a keyword called `those`. In a simple implementation, the `those` keyword refers to a collection (such as a JavaScript array) that holds the objects that the user has most recently selected in the user interface, e.g., by "lassoing" (click-dragging around) a number of objects. The user can then iterate over those objects using familiar language mechanisms such as a `for` loop.

In a more advanced implementation, the `those` keyword could be implicitly associated with iteration behavior, so that a message sent to `those` (e.g., "`those.move(10,0);`" to move all the selected objects ten pixels to the right) would be automatically applied to all the selected objects.

The use of the `those` keyword is illustrated in Figure 6. The darker area behind the widgets in Figure 6 represents the range of multiple selection. In this example, we are setting the value of each selected object to 10 by manually iterating over the objects in the `those` collection.

Note that in Figure 6, we use a `for` loop to iterate over all the objects that are currently selected. In a more advanced implementation – with implicit iteration associated with the `those` keyword – the programmer could accomplish the same behavior in a more direct way by writing "`those.setValue(10);`".

Like the `that` pointer, the `those` pointer has various uses in an interactive programming and scripting environment. It can be used for easily applying the same script programmatically to all the objects that the user has pinpointed in the user interface. It can also be used for obtaining a large number of references that can then be copied to other data structures.

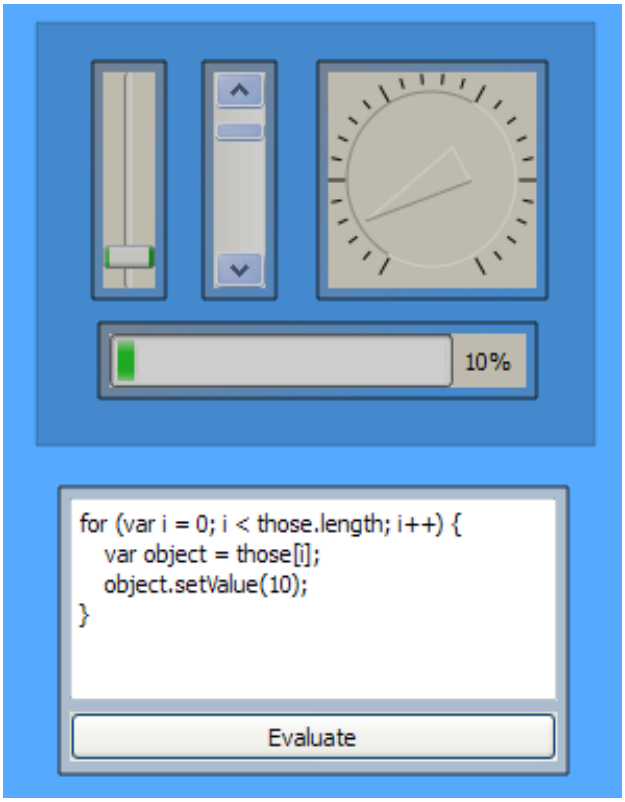

Figure 6. Using `those` keyword to programmatically manipulate objects chosen by multiple selection

Without the `those` pointer, the amount of work that the programmer would have to expend to accomplish the equivalent behavior would be considerably more significant.

4.EXPERIENCES AND COMMENTS

We have implemented `that` and `those` keywords for an interactive, visual JavaScript web application and mashup development environment called *Lively for Qt* (http://lively.cs.tut.fi/qt) [8]. We have used the system in a number of projects, including various international code camps and graduate student projects at the Tampere University of Technology, Finland. In these efforts, we have found the proposed new keywords extremely useful especially during interactive development, testing and debugging, and also during live application demos to large audiences. The keywords serve as a "glue" that bridges the gap between conventional source code editing and visual GUI-driven development. The proposed keywords can also facilitate interactive debugging considerably, allowing the user to effortlessly "grab" pointers to visual objects and their substructures on the screen. Without the `that` and `those` keywords, the interplay between the GUI and source code editing would be much more cumbersome.

Furthermore, the proposed new keywords are helpful in a web programming environment that is running in a mobile device or some other "input-constrained" environment. With the help of the `that` and `those` keywords, the user can easily apply the same script(s) to different objects on the screen without having to spend a lot of time editing and customizing the scripts to use different variable names in the source code.

5.RELATED WORK

Techniques for visual programming and direct manipulation have been studied for decades, starting from Doug Engelbart's work on the NLS system [1] and Ivan Sutherland's work on Sketchpad [12] in the early 1960s. These pioneering activities were followed by the development of the Smalltalk system at Xerox PARC in the 1970s [2] and Ben Schneiderman's early research work on user interfaces based on direct manipulation [10]. Numerous visual programming languages and environments have been proposed over time. For instance, the Fabrik system [4] was one of the first truly "program-by-wire" environments in which programming was performed mainly by visually connecting various graphical elements to each other. Demonstrational user interfaces [9] took visual programming further, sometimes attempting to eschew with conventional source code altogether.

Our work on the `that` and `those` keywords has been inspired by more conventional interactive programming environments in which source code still plays a central role. For instance, Seymour Papert's *Logo* programming language [7] – developed originally in the late 1960s – supported a turtle graphics system in which the location of the graphics cursor ("turtle") on the screen was maintained automatically by the system. The essential commands of the Logo programming language, such as *LT* (for turning the turtle left), *RT* (for turning the turtle right), and *FD* (for moving the turtle forward), rely on the implicitly maintained turtle location and direction information, so that the user can interactively/programmatically control the movement of the turtle in the user interface. Our `that` and `those` keywords are similar in spirit but their use is not restricted only to turtle graphics. Rather, our keywords can be used for commanding any object(s) that are currently under manipulation in the user interface.

Regarding the possibility of creating pointers to interactively created objects in the graphical user interface, our work has been inspired by systems such as Smalltalk [2] and Self [11, 14]. In the Self programming environment, for instance, there is a "drilling" tool that allows the programmer to obtain a "core sample": a visual summary of all those objects that are currently under the mouse cursor, starting from the topmost object to the bottommost graphical object. Similar capabilities are present also in the Squeak Smalltalk environment [5] and in the Sun Labs Lively Kernel JavaScript environment [13]. However, in those systems there is no easy way to convert the obtained information into pointers that could be manipulated at the source code level.

6.CONCLUSION

In this paper we have introduced two new keywords `that` and `those` to facilitate interactive programming. The key idea behind these new keywords is to make it easy to refer programmatically to those objects that are currently under manipulation in the graphical user interface. While the proposed new keywords are not intended for purely visual programming, the new keywords can be very useful in web-based scripting and development environments in which these is a need to manipulate complex, dynamically created objects from an interactive source code or script evaluator.

7.ACKNOWLEDGMENTS

The initial work behind the ideas presented in this paper was completed when the first author was working at Sun Microsystems Laboratories. The authors would like to thank Sun Labs and the Academy of Finland (grant 115485) for their support during this work.

8.REFERENCES

[1] Engelbart, D. C. *Augmenting Human Intellect: A Conceptual Framework*. Summary Report AFOSR-3223, Stanford Research Institute (Menlo Park, California, USA), October 1962.

[2] Goldberg, A. and Robson, D. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.

[3] Griffin, E. *Foundations of Popfly: Rapid Mashup Development*. Apress, 2008.

[4] Ingalls, D., Wallace, S., Chow, Y-Y., Ludoph, F. and Doyle, K. Fabrik: A Visual Programming Environment. In *Proceedings of OOPSLA'88 Conference* (San Diego, California, September 25-20, 1988), ACM SIGPLAN Notices 23, 11 (November 1988), 176-190.

[5] Korienek, G., Wrensch, T. and Dechow, D. *Squeak: A Quick Trip to ObjectLand*. Addison-Wesley, 2001.

[6] Loton, T. *Mashup Case Studies with Yahoo! Pipes*. CreateSpace Press, 2008.

[7] Lukas, G. and Lukas, J. *The LOGO Language: Learning Mathematics Through Programming*. Entelek Press, 1977.

[8] Mikkonen, T., Taivalsaari, A. and Terho, M. Lively for Qt: A Platform for Mobile Web Applications. In *Proceedings of the ACM Mobility Conference 2009* (Sophia Antipolis, France, September 2-4, 2009), ACM Press.

[9] Myers, B. Demonstrational User Interfaces: A Step Beyond Direct Manipulation. *IEEE Computer*, 25, 8 (August 1992), 61-73.

[10] Schneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16, 8 (August 1983), 57-69.

[11] Smith, R.B., Maloney, J. and Ungar, D. The Self-4.0 User Interface: Manifesting a System-Wide Vision of Concreteness, Uniformity and Flexibility. In *Proceedings of OOPSLA'95 Conference* (Austin, Texas, October 15-19, 1995), ACM SIGPLAN Notices 30, 10 (October 1995), 47-60.

[12] Sutherland, I.E. *Sketchpad: A Man-Machine Graphical Communication System*. Ph.D. Thesis, MIT, January 1963.

[13] Taivalsaari, A., Mikkonen, T., Ingalls, D. and Palacz, K. *Web Browser as an Application Platform: The Lively Kernel Experience*. Sun Microsystems Laboratories Technical Report TR-2008-175, January 2008.

[14] Ungar, D. and Smith, R.B. Self: The Power of Simplicity. In *Proceedings of OOPSLA'87 Conference* (Orlando, Florida, October 4-8, 1987), ACM SIGPLAN Notices 22, 12 (December 1987), 227-241.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FIN-33101 Tampere, Finland