Lisp and Symbolic Functionality in an Excel Spreadsheet: Development of an OLE Scientific Computing Environment

Ronen Gradwohl and Richard Fateman
EECS Department
University of California, Berkeley

August 13, 2002

Abstract

An Excel link to Lisp functionality can provide a window into broader applicability of the spreadsheet interface to the general development of a scientific computing environment. Given the attempt by Microsoft to integrate all its applications via OLE, the same kind of technique can add powerful computation capabilities to other commonly used applications such as word processing and database programs.

1 Introduction

OLE (Object Linking and Embedding) is Microsoft's technology which allows different applications running within Windows to communicate with one another. An OLE server is an object that represents the functionality of an application, and an OLE client accesses that functionality via an interface. Our goal is to allow a user whose primary interaction is with Excel, Word, or any other Office application the power and functionality of Lisp, or, more specifically, a symbolic and numerical computation package implemented in Lisp.

2 The Excel Interface to Lisp

Our work has been based on an Excel interface, since this seems to have the most immediate appeal to computation.

2.1 The Current Implementation

Setting up a Lisp OLE server in the background of an Excel Worksheet can take one or two simple steps, or can be done automatically.

We support two Excel functions that allow access to Lisp functionality. `lispEval(expr)` takes takes the one argument `expr`, a Lisp expression, and evaluates it in the server's Lisp environment. `lispApply(fun,arg1, ... ,argn)`, takes one or more arguments, the first of which is a Lisp function. The following arguments are the values to which that function will be applied. They can contain integers or other values, symbols, cells referring to values within the spreadsheet, and even ranges of cells. `lispApply` passes the function and a list of all the arguments to the Lisp server.

Multi-dimensional Range objects are passed as lists of lists, with the Range object's rows being the nested lists.

2.2 Examples

The data that we need is a superset of the data that is built-in and inherent in Excel's world view, and so we must encode some of it as strings. That is, when we pass an expression to Lisp, we must bypass Excel's view and quote it, as shown in the following basic examples.

If we insert into a spreadsheet cell the following formula: `=lispEval("(+ 1 2)")`, or, equivalently, `=lispApply("+", 1, 2)`, then that cell will evaluate to `3`. Thus `3`, an ordinary Excel integer will be displayed in the table.

If we use this formula: `=lispEval("(format nil ""~s"" abc)")`[1] we will get `abc` in the corresponding cell.

Note that you can define functions within a call to `lispEval`, as in: `=lispEval("(defun foo (&rest r) r)")`. You can then follow this by `=lispApply("foo", 1, 23.43, "x", a4, a1:c7)`, where `a4` refers to the corresponding cell in the spreadsheet, `a1:c7` is a Range object, and `x` is presumably a bound variable with a value in the Lisp system. Finally, `lispApply` can also accept `lambda` function definitions, as in `=lispApply("(lambda(x y z) (foo x (bar y z)))", "x", a4, c7)`.

2.3 Other Interface Possibilities

From a Lisp perspective we would like to get rid of some of the Excel syntax, while still having access to its data. Perhaps we would prefer to have something like `=lisp(+ a1 b2)`, but unfortunately the Excel parser will object to this before ever calling Lisp. How about `=lisp("(+ a1 b2)")`, or perhaps `=lisp("(foo b3:d7)")`? In these cases `a1` and `b2` refer to the values of the corresponding cells in the spreadsheet, and `b3:d7` refers to a range in the worksheet. This syntax is legal Excel and has the advantages of being clear and simple. It does, however, present an inconvenient complexity. Suppose we had the former example in our spreadsheet, and then we changed the value of `a1`. The spreadsheet should then recompute the value of any expression depending on `a1`, as that is some basic functionality of the Excel spreadsheet. But the `a1` is inside a string, which Excel will not notice. This complicates our lives, but is not the worst of it. Suppose we added a row at the top of the spreadsheet, which, in the ordinary semantics of spreadsheets, would change the Lisp expression to add the values of `a2` and `b3`. Should Excel modify the contents of the *string* argument in our Lisp expression? Not only is this difficult, it also has no precedent within Excel functionality. Hence, we must look elsewhere for a practical solution.

The next approach we considered is to implement something similar to other Add-Ins. In particular, Maple's Add-In for Excel [2] suggests that a call to the Lisp server could look like `=lisp("(+ &1 &2)", a1, b2)`. In this example, the `&1` is replaced in context by the second argument, the `&2` is replaced by the third argument, and so on. This pattern-matching approach allows us to use `a1` and `b2` to refer to the values in the corresponding cells in the spreadsheet. Since these names are not hidden inside a string, Excel's normal functionality provides the implementation to re-evaluate. When `a1` or `b2` is modified, the expression gets recalculated. Furthermore, if a row is added at the top of the spreadsheet, the expression automatically gets changed to `=lisp("(+`

[1] The `""` is an escaped `"` character in Excel.

&1 &2)", a2, b3). Excel will determine which cells get recomputed, providing consistency within the spreadsheet's functionality.

A disadvantage of this syntax is that it is ugly, more complicated, and not Lisp-like.

Our implementation of lispEval and lispApply is a somewhat more "Lisp-ish" spin on this syntax.

A simple extension of this work would allow the users access to a Macsyma system running in the Lisp server, which could look like =macsymaApply("integrate",a1, b2), or perhaps =macsymaApply("lambda([x,y], 3*x+sin(y))",a1, b2). Another possible approach is to set a cell, say C3, to the (string) lambda([x,y],3*x+sin(y)) and call from another cell, say D3, the formula =macsymaApply(c3,a1,b2). Naturally a version of =macsymaEval(expr) could also be set up.

3 Future Work

3.1 Additional Features

Since the design of any environment for increasing productivity must place an emphasis on ease of use we considered some additional features for future implementation. For example, instead of having to hit control-i and control-t to initiate and terminate the server, we could install buttons on the toolbar corresponding to these. In fact, it may be even more convenient to only have one button, which then chooses the appropriate action (this presents a limitation, however, in that only one Lisp OLE server could be running within each Excel Workbook). You might also wish to define functions without getting the return value in a spreadsheet cell, so there could be a user form which would permit input or output from a text box. Furthermore, it may be convenient to provide a user form or menu item to initialize or customize the back-end Lisp service by loading files. Thus, the user's interface with the Lisp server could be customized to an application.

3.2 Error Checking with Exceptions

An obvious void in the current implementation is the lack of error checking.

By default, if you attempt to evaluate a faulty Lisp expression, the error will show up in the Lisp server's console, and the spreadsheet will freeze. In order to regain control, you must close the Lisp console, and then start a new server. This should not be necessary. Instead, the server should execute code within an error handler with a suitable versatile design. A simple but not entirely convincing improvement on our implementation is for the Lisp server to execute all code inside (ignore-errors ...) which would return nil from any erroneous (terminating) input or an unsyntactic input. A clean termination of an infinite loop would also be a useful safeguard.

A more elaborate interchange of data would require communication with the VBA function that is accessing the server, and alert the user of the error.

3.3 Application to Microsoft Word

Similarly to Excel, Microsoft Word, Powerpoint, etc. can utilize the Visual Basic macros that link to a Lisp OLE server. Thus, the user may choose to have that word processing application as his main point of interaction, and still have the power and functionality of Lisp. Some uses may include the embedding of Lisp expressions, miniature Lisp consoles, or applications written in Lisp such as computer algebra systems, within a Word document. The overall effect could be similar to that of Scientific Workplace [3], a commercial product which merges a WYSWYG word processor which looks somewhat like TEX with partial access to a computer algebra system (Maple V).

In our combination, the user familiar with Word will have the comfort and advantage of building upon known applications to access the subtleties of a new application.

4 Acknowledgments

This research was supported in part by NSF grant CCR-9901933 administered through the Electronics Research Laboratory, University of California, Berkeley. Ronen Gradwohl was supported by an NSF undergraduate research grant.

5 Appendix

5.1 How to Run the Current Server

These instructions are adapted from the OLE sample code provided by Franz [1], and need be executed only once on any computer. In order to successfully utilize this Lisp OLE server from within Excel or any other Office application, the user must have a registered version of Allegro 6.1 or higher, as well as administrator permissions (since it is necessary to modify the Registry).

First, it is necessary to delete the subfolder `delivery` from the directory containing `deliver.cl`. Next, in a Lisp with a compiler, `:cd` into the directory containing `deliver.cl` and `server.cl`. Now, execute the command `:ld deliver.cl`, and then, if desired, `:exit`. In order to be able to use this server from other applications, you create a reference to it in the Registry. You run the executable `delivery/Register testeval`. Finally, in the Excel spreadsheet, hit `control-i` to initiate the server (note: this might be changed into a toolbar button). Excel is now ready to accept Lisp commands in a particular form, which it will pass on to the Lisp OLE server. Finally, in order to close the server, hit `control-t`.

When the Lisp OLE server is running, it is possible to evaluate any Lisp command from within an Excel cell. In an active cell, type in the function `=lispEval("any-lisp-expression")`. `any-lisp-expression` will be evaluated, and the result returned and displayed in the active cell.

5.2 File and Method Descriptions

The file `server.cl` implements the Lisp server. The notable methods are:

- `defclass test-evaluator`: this class is of type `ole:automaton`, and it defines the interface to the server.

- The `ole:def-automethod` functions define the evaluation methods that can be called on the server. In their current implementations, `evaluation` evaluates the Lisp expression given in `expr`, while `application` applies the function from `fun` to the list `args`.

The file `deliver.cl` creates the application, as well as shortcuts to executables that add and remove references to the server in the Windows Registry.

In order to use the Lisp server from another application, say Excel, it is necessary to start up a new server from a Visual Basic for Applications (VBA) macro or function. This is easily done with the statement `Set lispServer = CreateObject("Franz.testeval.1")`. It is now a simple matter to refer to the elements of this interface, with commands such as `lispServer.expr` and `lispServer.evaluation`.

When the VBA macros and functions are completed, it is convenient to save the code as an Excel Add-In, so the code can be shared. It might be desired to make the code password-protected, which is an option accessed via Tools on the tool bar. To save the code as an Add-In, first name the file by choosing `File->Properties`, and then adding a name to the code. Then, when saving the file, choose to save it as an Excel Add-In, with extension `.xla` rather then the usual extension of `.xls`.

References

[1] Franz Inc. http://www.franz.com.

[2] Maple. http://www.maplesoft.com.

[3] Mackichan software swww.mackichan.com.