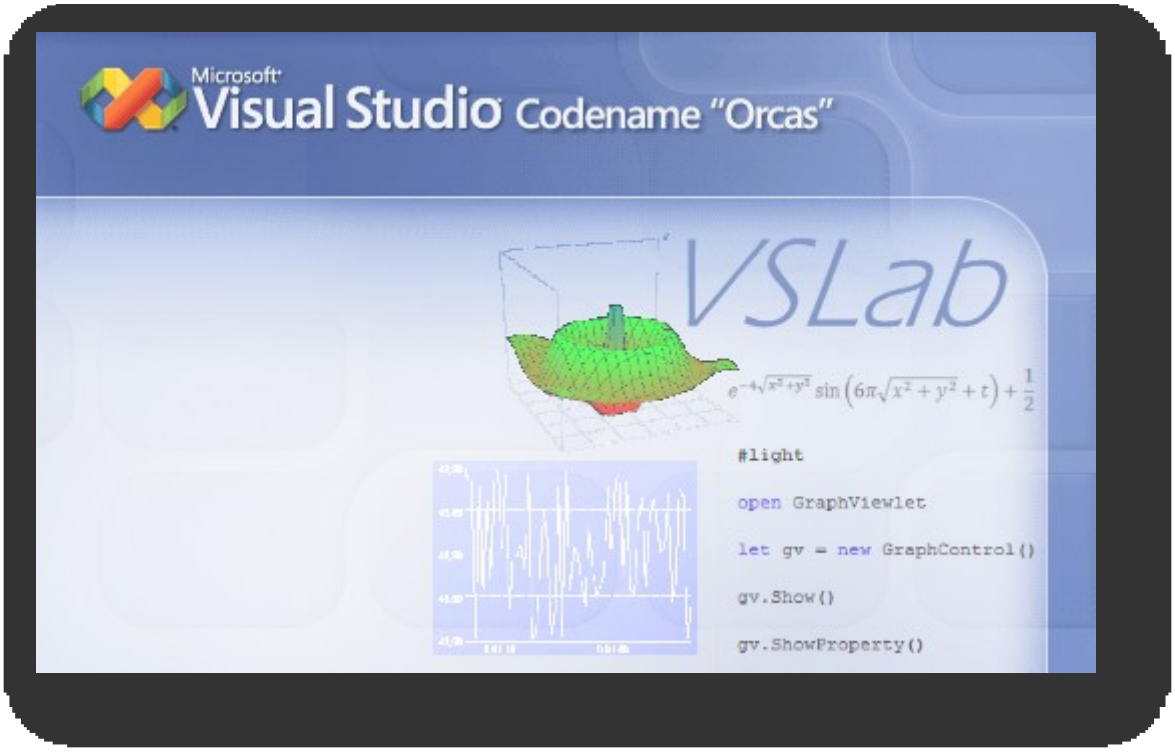
VSLab and MatLab

Whitepaper



In this paper we briefly discuss in what sense VSLab can be compared to MatLab and similar environments, and what are the main differences with those environments.

Author: Antonio Cisternino (cisterni@di.unipi.it), University of Pisa
Version: 1.0
Last update: 6/28/2008 7:11:00 PM

Introduction

When people take a look to VSLab one of the first comments you get is “this is a sort of MatLab”, and it is true, but only in a sense. In this document we explore the relations between VSLab and systems like Mathematica or MatLab to clarify the possible role that VSLab may play in the scenario of scientific computing and laboratory oriented analysis.

First of all it must be defined what VSLab is and the design principles that have driven its development with respect this potential use that of course was in the mind of the designers. VSLab is a complement to F# which is the true magic thing, in particular because of F# interactive that offers the opportunity of evaluating code fragments interactively. The primary purpose of VSLab is to provide the core mechanism for F# interactive to deliver graphics integrated with Visual Studio (and possibly also other features provided by DTE COM infrastructure), allowing toolwindow sharing the state inside a CLR based environment without having to communicate through COM wrappers.

It is not a goal of VSLab to provide a comprehensive set of viewlets and libraries for doing math and data-visualization; it is rather a platform that can be used to implement effectively domain specific languages inside Visual Studio.

MatLab, Mathematica, and neighbors

In scientific environments where data read by a plethora of sensors must be analyzed interactively by programs that require interactive evaluation, a number of environments have emerged to support engineers and scientists in their needs. The anatomy of these environments is essentially the same: a graphical environment featuring a scripting language for manipulating the primitives of the environment interactively and relatively at high level. This is the architecture of many environments also in other systems (think for instance of Microsoft Office and several other environments), but in the scientific environment the needs are mainly typing oriented for input and visualization driven by script statements. Performance, particularly important for data processing, is ensured by highly efficient code developed mostly in C/C++ that is made accessible to the scripting environment so that most of the work is done by compiled code coordinated by the scripting environment.

Once people get used to a tool tend to use it in every occasion, and this has been done with MatLab-like environments, used to code general purpose programs, going beyond the original goals of the tool. In particular the functional nature of these scripting languages, more intuitive for people used to mathematics, has always been considered an important aspect with respect to programming languages used in the general purpose programming environments. The dimension of scripts, therefore, has grown significantly and the overhead of interpretation has become significant, but no-matter how many optimizations you can do the original design of these system has become their limit.

Moreover at the end of the interaction process the user of the system is potentially interested in baking the program and reusing it in a non-interactive fashion. If in origin this baking process was mostly a C rewriting of the program in order to redistribute it, recently the dependency from sophisticated algorithms contained in the runtime of these system has made this approach prohibitively expensive and users of these systems have started to ask the system to those interested in their programs and solutions. Many companies, including hardware-makers have accepted this choice as a fact of life and sell product supporting these runtimes. Although in several scenarios this is a reasonable choice because of the added value provided by

the algorithms shipping with these systems, it often happens that people write simple programs in these environments and pretend to ship them assuming the complicated runtime without any real reason.

VSLab and MatLab*

VSLab has been designed with the idea of supporting the development lifecycle of applications like MatLab, where you have room for annotating ideas and write code fragments which can be tested interactively. Viewlets allow writing data-visualization components providing graphical views of input data, and fill the otherwise significant lack of data manipulation. F# is a functional language, and although it is statically typed, the type inference featured by the language is powerful enough in many cases not requiring mastering the complexity of type annotations for non-programmers.

In VSLab you can:

- Open a .vslab file to write your code fragments
- Evaluate code interactively thanks to F# interactive
- Project data into viewlets and experiment interactively
- Consolidate program fragments into .fs files that can be used for compiling the final result into a standalone application

Thus VSLab provides the infrastructure typical of tools like MatLab but there is no support for algorithms, and we have no plan to do it, there are plenty of mathematical libraries available and that can be easily used from F#. Moreover Viewlets development is easy enough to write your own visualization systems or use those provided by third parties. It can be expected better support for Viewlets in future releases of VSLab, however.

We could have written VSLab using one the various dynamic languages available in the .NET space (for instance IronPython), but the unique mix of features that F# provides has been irresistible for us, since we can provide the same interactivity of dynamic languages, but at the same time the program can be consolidated into a fully compiled application. This is witnessed by the fact that VSLab too has been developed in F# though it has required a significant amount of interop code with system-level technologies such as Windowing and COM (see the whitepaper about VSLab internals).

It not just MatLab

Although shaped with the MatLab-like development process, VSLab is not just a Mat* application targeting only scientists and engineers. I'm starting using VSLab as a system shell or a network monitor, and in many other contexts, because the very large spectrum of class libraries available for the .NET framework.

A nice example is described in the whitepaper about developing a viewlet, in which we show how define a viewlet to monitor CPU and memory load of a VS and F# interactive session.

We have already spotted a number of interesting areas in which VSLab can be used to build interactive systems: natural language research; system shell and resource management; distributed computing just to mention few.

Conclusions

VSLab provides the suitable infrastructure for supporting MatLab-like systems, though it is just a shell and you need also the algorithms in order to obtain a similar system. We expect, nevertheless, that using these core mechanisms people will develop environment tailored for various application as it has happened for packages developed on top of these systems.

The ability to compile the consolidated program into .NET executables ensures performance and no limitation due to a specific runtime and the guarantee to run these programs everywhere without need even for VSLab installed.