A Web-based Host Platform for Pedagogical Virtual Machines

Nuno Gaspar[1] **and Simão Melo de Sousa**[2]

[1] Computer Science Department, University of Beira Interior, Portugal, nmpgaspar@gmail.com
[2] Computer Science Department, University of Beira Interior, Portugal, desousa@di.ubi.pt

Abstract: Difficulties in the multi platform deployment and use of *pedagogical* Virtual Machines can have an annoying impact in the success of a compilers construction course. This paper introduces a compilers construction course support platform that tackles this issue. The proposed platform is web-based, where the virtual machines are remotely accessed through a browser and by the adequate use of web services. Moreover, both command line and graphical user interface versions are supported. Aside portability and maintenance issues, an interesting feature of the architecture design is its ability to easily integrate new virtual machines. As proof of concept we will show some preliminary results on the integration of two very different virtual machines that are used in compiler construction courses in Portuguese universities.

Keywords: virtual machines, pedagogical web platform, compilers, education, computer assisted instruction, teaching methods

1. Introduction

The paper introduces a compilers construction course support platform. Our focus is the easy installation, multi platform deployment and use of virtual machines (VMs) that are the target architecture in a compilers construction course.

1.1 The need for such tools

As argued in [1], the standard computer science curriculum only leaves a small room for compiler construction courses. This is the case, in particular, in Europe with the university reform arising from the Bologna process. Usually only a one-semester long course (two at the best) is dedicated to the teaching of the classical

compilation concepts and tools. Therefore, the teacher must define and adopt pedagogical alternative strategies to the teaching of detailed compilation techniques. Adequately covering all the classical aspects of compilers design in such a short time slot is simply inappropriate. In this context, the choice of a VM as target architecture for code generation plays an important role in the success of the course. With its use, the intricacies of a real architecture are omitted, letting the undergraduate compilers construction courses focus on the most important aspects.

In a first contact with such concepts, a carefully designed *GUI* for the VM or other auxiliary tools are of paramount importance for a deep understanding of the involved mechanisms. However the VM *multi-platform* deployment is not easy and becomes usually problematic for students. It is very common to see in the same classroom the use of three different operating systems. Appropriately distribute VM (with graphical interface) source code, letting students handle the compilation process and provide them an effective support can be hard. Our experience shows that most of the VMs used in compilers construction courses are developed in a Linux environment and used in a Windows environment. This is the case for the two VMs that we address here.

1.2 Our approach

In order to tackle this issue, we propose a web-based VM host platform. By providing a web-based platform, we ease the virtual machine's use, distribution and maintenance. Enabling its access through a browser and web service calls, we provide a setup independent from the client's operating system. Furthermore, the distribution of updates is not required anymore, since eventual changes will reflect on any client.

Another interesting feature is the ability to gather and share tools, extensions and contributions from the users' community. For instance, *bytecode* optimization tools, online tutorials, hints or new VMs can easily be added to the platform.

1.3 Related work

To the best of our knowledge, there is no similar approach to provide a pedagogical web environment for the use of VMs. Indeed, a quick look at the compilers tools catalogue [2] shows that there are tools for almost all the phases of the design of a compiler except for the VMs.

Nevertheless, we may refer that broadly used VMs in the context of compiler construction courses are LLVM [3], Parrot [4], Java [5] and CLR [6]. For the two last VMs, the designed compilers usually target a carefully chosen subset of the java *bytecode* and .NET IL. Let us mention that the MIPS assembly (executed via emulators like SPIM [7] and which success is due, for instance, to the existence of

compilers design books like [8]) or the Intel x86 assembly are also widely chosen as the target of compiler construction courses homework. Finally, *VM* [9] and *Apoo* [10] are two virtual machines that are used in several Portuguese universities.

All of these execution environments are, obviously, easily distributed over several platforms in their command line form, but except for java/swing implementations, VM graphical interfaces deployment is in fact problematic[1]. Opting for a java based implementation is a possible solution for the issue addressed here. But we must refer that a web-based architecture also provides mechanisms and benefits (e.g. upgrades, sharing and integrating extensions from the users' community) that are not easily covered by this approach.

Another tool that we may refer is the *CGI* interface to SPIM [11]. It is very simple and closer to command line interface than to a pedagogical and full featured graphical one. The lack of an internal state visualizer and of step by step interaction makes it unappealing for pedagogical purposes.

1.4 Organization of the paper

Section 2 introduces the main ingredients of the proposed architecture. Some preliminary results are introduced in section 3. Concluding remarks and future work are presented in section 4.

2. Architecture of the Platform

The overall architecture of the proposed platform can be found in the figure 1. The VMs, *WebApoo* and *WebVM*, are both available in graphical and command line interfaces. The user can execute his machine code files in either version. For the last one, the submitted program is executed at once and only the virtual machine's final state (registers, memory cells values, etc) is presented to the user.

1 To the best of our knowledge, none of the referred VMs have such java implementation.

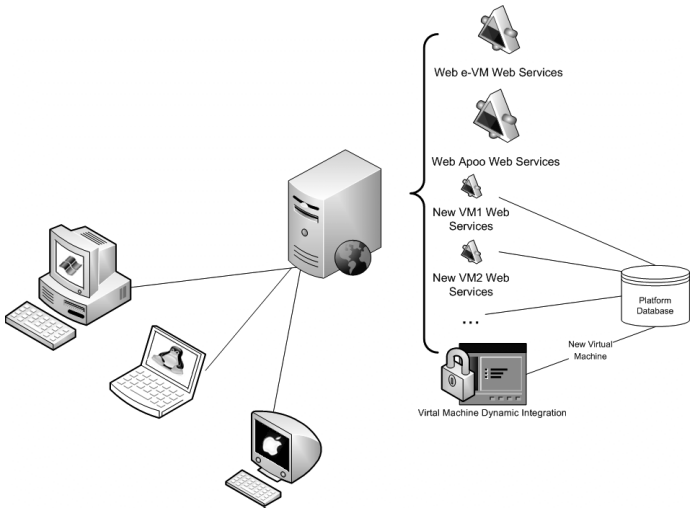

Figure 1 Platform Architecture.

On the other hand, the graphical version allows the user to analyze the program behavior by executing each instruction *step by step*. This permits a better understanding of the VM state changes induced by the execution. Alternatively, the user can just run the program, where the use of *breakpoints* may be of great benefit when debugging.

In the case of *WebVM*, since it has the ability to interpret and perform graphical operations (like drawing geometrical figures), those will be displayed in a *draggable window*.

Concerning the interaction with the VMs, the communications between the clients and the platform (the server) are done via web services. The reply of each request made will affect the virtual machine's interface displayed in the browser. One definite advantage of this kind of architecture is its capacity to be internally restructured without reflecting any change to the client side.

Moreover, besides *WebApoo* and *WebVM*, the platform also includes a mechanism for dynamic integration of new VMs in their executable form. The details regarding this process will be described in section 3.

2.1 Client side

The web interfaces are implemented in *Silverlight* [12]. In order to play with it, the user must install the plug-in in its favorite browser.

Despite being relatively recent, *Silverlight* already offers a nice compatibility between browsers and operating systems. This fact is illustrated by table 1, taken from the official *Microsoft Silverlight* web site, and completed with information available from the *Mono* project web site.

Table 1. Silverlight compatibility

Operating System	IE7	IE6	Firefox 1.5	Firefox 2	Safari
Windows Vista	Yes	-	Yes	Yes	-
Windows XP SP2	Yes	Yes	Yes	Yes	-
Windows 2000	-	Yes**	Yes	Yes	-
Windows Server 2003	Yes	Yes	Yes	Yes	-
Mac OS 10.4.8+ (PowerPC)	-	-	Yes*	Yes*	Yes*
Mac OS 10.4.8+ (Intel-based)	-	-	Yes	Yes	Yes
Linux	-	-	-	Yes***	-

* *Silverlight* 1.0 only; ***Silverlight* 2.0 only; ***via the *Moonlight* – a Mono-based implementation of *Silverlight*

2.2 Server side

On the server side, the web services are the means by which all the computation requests are done. The following web services are available for each VM:

- *UploadFile*: This operation accepts a machine code file, and returns a unique session key to the client.
- *ExtInstructions*: This operation accepts a session key, and returns the initial configuration of the VM.
- *Step*: This operation executes the instruction pointed by the program counter. It requires the user session key and returns the virtual machine's configuration after the execution.
- *Run*: As expected, this operation is a sequence of *steps*. It accepts a session key and performs the *steps* operations until the end of the program.

The server stores the virtual machine's state for each active client connection. The purpose of the session key is to relate the stored states with the clients' requests.

3. Preliminary results

In order to get familiarized with the diversity of target VMs, we first implemented from scratch two very distinct specimens. A stack based virtual machine with graphical primitives, *VM* from *laboratoire de Recherche en Informatique* of the *Université Paris Sud*, and a register based virtual machine, *Apoo*, from *Faculdade de Ciências* of the *Universidade do Porto*.

At this stage, our focuses were to prototype the system and obtain a proof of concept. Thus, the integration in the platform was done manually, without the use

of the integration mechanism whose implementation, use and result are described below.

These first experiments let us understand all the necessary details and considerations to take care of for the systematic integration and use of a VM within a generic platform.

3.1 Apoo integration

Essentially developed by Rogério Reis and Nelma Moreira, from *Universidade do Porto*, *Apoo* is a register based VM. It has a very simple instructions set that mimics almost all the essential features of a modern microprocessor.

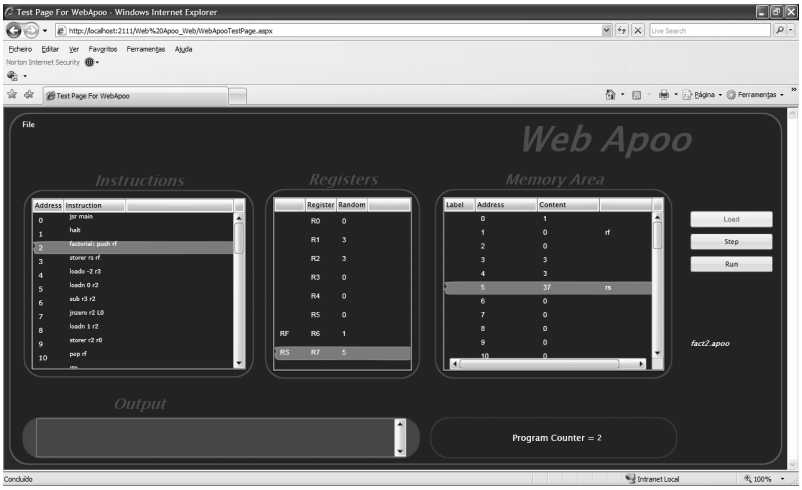

Figure 2 *WebApoo* graphical interface.

The web version of *Apoo* has the same characteristics and allows to do the same operations as in the original one (as illustrated by figure 2). It has a set of general purposes registers, a memory area, a system stack and a program counter register. It allows to check the virtual machine's state, step by step or running it without pauses. Using the command line interface, the user only sees the result of the complete execution.

3.2 LRI's VM integration

Initially developed by Christine Paulin-Mohring, Jean-Christophe Filliâtre and Sylvain Conchon, from *Laboratoire de Recherche en Informatique*, *VM* is a stack based virtual machine. It has an execution stack, a call stack, two heaps and four

registers. It is used in *Universidade da Beira Interior*, *Universidade do Minho* and several French universities as a support for compilers construction courses.

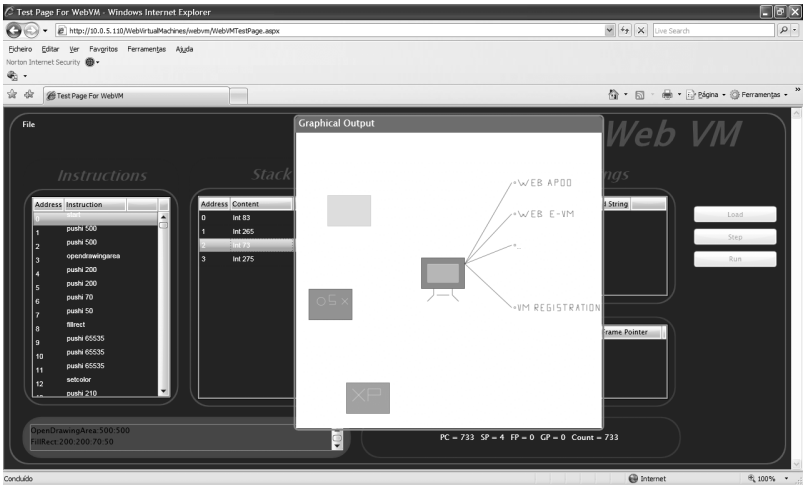

Figure 3 *WebVM* graphical interface.

The web version of *VM* works the same way as the original one. The specificity lies in its ability to process graphical primitives. As shown in figure 3, the graphical client is able, thanks to *Silverlight*, to directly interpret these primitives and displays their result in a *draggable window*.

Alternatively, *WebVM* can also be accessed via a command line interface.

3.3 Integrating new virtual machines

As result of these first experiments, we were able to understand how to generalize the platform in order to allow the systematic integration of new VMs. By specifying what data areas (registers, stacks, etc.) are needed, and linking them to a third party VM output, it is possible to achieve this genericity.

Most of the VMs used in a pedagogical environment are provided with source code. The required changes for their integration were planned in order to be accomplished as easy as possible. The first step is to define a valid VM specification according to the *XML schema* illustrated by figure 4.

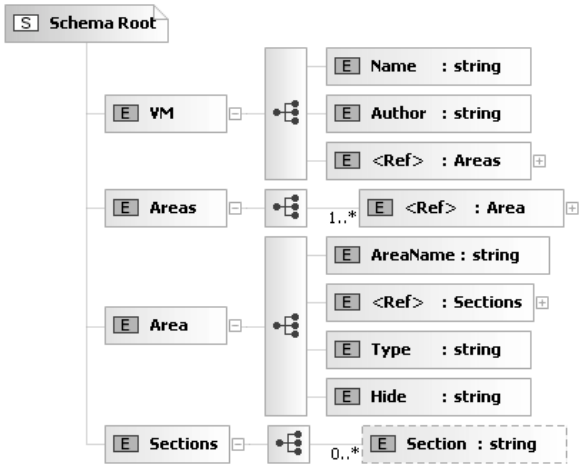

Figure 4 *XML schema*.

Basically, the *name*, *author* and *areas* of the VM need to be provided. Each *area* is composed by its *name*, *sections*, *type* and *hide* attributes.

The value of the *name* attribute will be used as title for the *window* where the current *area* will be displayed. The *type* attribute can only have the values "*Primitive*" or "*List*", meaning that the data to be stored for that *area* requires a single text field or a list of them, respectively. Since there may be situations where it is required to store data that is not relevant to the user, the *hide* attribute determines whether the *area* is meant to be displayed or not. Only the values "*YES*" or "*NO*" are allowed. Finally, the *sections* attribute enumerates the sections that compose the *area*.

As for the virtual machine's executable file itself, the following changes are required:

1) The VM must have two execution modes:
   a. *Load mode* – Receives the machine code file to execute as parameter. It must produce an *XML* file with the virtual machine's initial state (the contents of each *area* defined in its *XML* specification file). The produced *XML* file must have the same name as the one passed as argument. For instance, if invoked by the command *virtualmachine.exe –l asm_file.vm*, the VM must output the file *asm_file.vm.xml*, containing the expected result.
   b. *Step mode* – Receives an *XML* file and a possible user input. The *XML* file holds the virtual machine's internal state, from which all the necessary information required to perform the *step* operation will be loaded from. As result, the VM must overwrite the *XML* file passed as parameter, with its updated

internal state. For instance, the following invocation *virtualmachine.exe –s asm_file.vm.xml [input]* produces an updated *asm_file.vm.xml* file.

2) Any instruction requiring an input must get the expected value from the second command line parameter.
3) If the VM can interpret graphical primitives, it must produce a *XAML* or *SVG* file with the drawing to represent.
4) When reached the end of the program, the VM must produce an empty file.
5) If any error occurs during the execution, it must be described in the produced *XML* file as follows: *<errorMessage>Error Description</errorMessage>*.
6) The VM must run in a Windows environment.

Once the *XML* specification file and the modifications are done, by uploading them to the platform, the VM is stored in the server database and automatically available to the all community.

In order to demonstrate this functionality, we integrated an executable version of *Apoo*. Despite its different and generic layout, it works the same way as the two previously described VMs. After loading the intended machine code file to be executed, the user can either *step* or *run* the program. As illustrated by figure 5, the contents of each *area* are represented in specific *windows*.

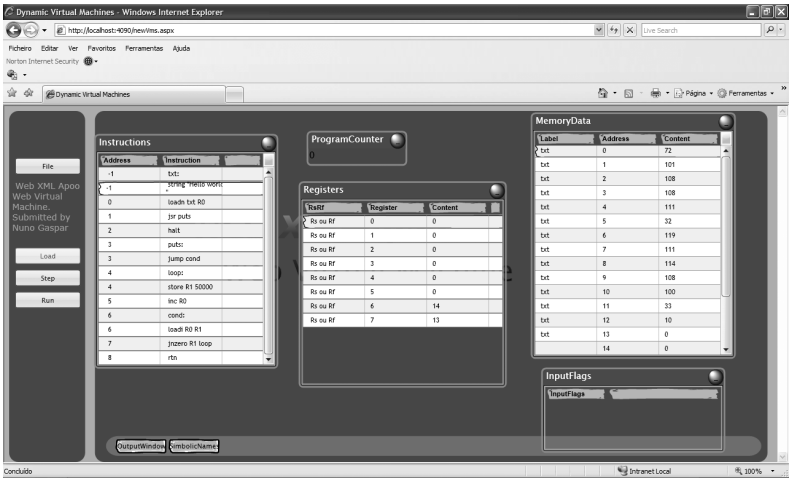

Figure 5 Integrated virtual machine generic layout.

Since the integrated VMs can have many *areas*, each *window* can be moved, resized and minimized. This way, the user can adjust the interface layout according to his will and needs.

4. Conclusion

We described a web-based platform for VMs that provides both command line and graphical interfaces. The use of *Silverlight* for the client produces the same advantages of a local *GUI*, without the issues of muti platform deployment. By simply installing the plug-in, students can interactively see and enjoy the involved mechanisms. Moreover, the ability to automatically add new VMs allows a high degree of reusability and can provide a wide range of code generation targets.

At this time, the platform has been tested in a small environment and not in a real class situation yet. Although preliminary tests points to a good student's receptivity, a deeper analysis is required. In order to get feedback from the community, compilers construction courses teachers have been contacted to let their students to alternatively use the platform. Furthermore, an online form where users can leave their suggestions and point out their usage experience is available[2].

As future work, we plan the integration of new features to provide more support to students. For instance, provide interactive examples on *loops translation* to machine code, optimization techniques or even new VMs.

We conclude by proposing this platform to the compiler construction courses teaching. The benefits of easy multi platform deployment and the possibility to gather and share tools from the community can make an interesting impact on the compilers construction courses success.

References

1. William M. Waite. The compiler course in today's curriculum: three strategies. SIGCSE Bull., 38(1):87-91, 2006.
2. German National Research Center for Information Technology. The catalog of compiler construction tools, 2006. http://catalog.compilertools.net/.
3. Chris Lattner and Vikram Adve. LLVM: A compilation Frameword for Lifelong Program Analysis & Transformation. In Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04), Pablo Alto, California, Mar 2004.
4. Perl Foundation. Parrot virtual machine parrot homepage, 2008. http://www.parrotcode.org/.
5. Tim Lindholm and Frank Yellin. Java Virtual Machine Specification. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
6. E. Meijer and J. Gough. Technical overview of the common language runtime, 2000.
7. James R. Larus. Spim: A mips32 simulator, 2006. http://www.cs.wisc.edu/~larus/spim.html
8. Andrew W. Appel. Modern compiler implementation in ML: basic techniques. Cambridge University Press, New York, NY, USA, 1997.
9. C. Paulin, J.C. Filliâtre, and S. Conchon. Virtual Machine for the LRI compiler course, 2008. http://www.lri.fr/~conchon/m1/.
10. Rogério Reis and Nelma Moreira. Apoo: an environment for a first course in assembly language programming. SIGCSE Bull., 33(4):43-47, 2001.
11. James R. Larus. Ee380 cgi spim, 2001. http://cgi.aggregate.org/cgibin/cgispim.cgi.
12. Christian Wenz. Essential Silverlight. O'Reailly, 2008.

2 See the *RELEASE* web site http://www.di.ubi.pt/~release