

# Learning, teaching, playing with compiler construction - A web based host platform for target virtual machines

Nuno GASPAR

Departamento de Informática,  
Universidade da Beira Interior,  
Covilhã, Portugal  
a17974@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 compilation course. This position paper introduces a proposal for a compilation courses 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 a 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.

## 1 Introduction

This position paper introduces a proposal for a compilation courses support platform. Our focus is the easy installation, multi platform deployment and use of virtual machines (VMs) that are the target architecture in compilation courses.

*The need for such tools.* As argued in [Wai06], the standard computer science curriculum only leaves a small room for compiler courses. Usually only 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 a so short time slot is simply inappropriate. In this context, the choice of a virtual machine as a target architecture for code generation, plays an important role in the success of the course. With its use, the intricacies of a real computer architecture are omitted, letting the undergraduate compilers courses focus on the most important aspects.

In a first contact with such concepts, a carefully designed *GUI* for VM or other auxiliary tools are of paramount importance for a deep understanding

of the involved mechanisms. However the virtual machine *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. Adequately distribute VM (with graphical interface) source code, letting students handle the compilation process and provide them an adequate support can be very painful. Our experience shows that most of the virtual machines used in compilation courses are developed in a Linux environment and used in a Windows environment. This is the case for the two virtual machines that we address here.

*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 the virtual machine's access through a browser and web services calls, we provide a setup independent from the client's operating system. Also there is no longer need to distribute updates, as eventual changes will reflect on any client. Another interesting feature is the ability to gather and share tools, extensions and contributions from the compiler's community. For instance, bytecode optimization tools or new virtual machines can be easily added to the platform and provided to the user.

*Related work.* To the best of our knowledge, there is no similar approach to provide a web environment for the use of VMs. Indeed, a quick look at the compilers tools catalogue [fIT06] 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 virtual machines in the context of compiler courses are LLVM [LA04], Parrot [Fou08], Java [LY99] and CLR [MG00]. For the two last VM, 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 [Lar06] and which success is due, for instance, to the existence of compiler design books like [App97]) or the Intel x86 assembly are also widely chosen as the target of compiler courses homework. In order to conclude, VM [PFC08] and APOO [RM01] are two virtual machines that are used in several Portuguese universities.

All of these execution environments are, obviously, easily deployed over several platforms in their command line form, but except for java/swing implementations, VM graphical interfaces deployment is in fact problematic<sup>1</sup>. 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 extension from the users community) that are not easily covered by this approach.

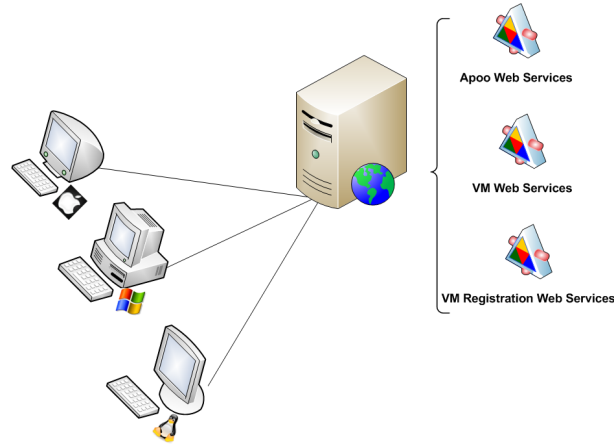
Maybe, the closest tools that we may refer is the C/C++ online compiler for the LLVM virtual machine [CSD08], but one still need to download and install the virtual machine to run the produced code.

---

<sup>1</sup> None of the referred virtual machines have such java implementation.

*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 the section 4.

## 2 Architecture of the Platform



**Fig. 1.** Platform Architecture

The overall architecture of the proposed platform can be found in the figure 1. Basically the user accesses the virtual machines, both graphical and *command line* versions, via his browser. Once the code to be executed is uploaded, the command line version, the most basic, only interacts with the user and presents the result of the execution in a terminal like interface.

The graphical version provides visual information about the internal state of the VM and the changes induced by the execution. The user can either check the virtual machine internal state step by step, or alternatively run it (performing all the execution steps at once). If the virtual machine is able to interpret and perform graphical operations (like drawing geometrical figures) the result will be displayed in a specific area.

The communications between the browser and the platform (the server), between the server and the registered virtual machines are done via web services. A virtual machine can be added to the architecture via a registration process that will be detailed later.

### 2.1 Client Side

The web interfaces are implemented in *Silverlight*. In order to play with it, the user must install a 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 the following table, taken from the official *Microsoft Silverlight* web site, and completed with information available from the *Mono* project web site.

Operating System	IE 7	IE 6	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 virtual machines web services are the means by which all the computation requests are done. The following web services are available for each virtual machine:

- UploadFile. This operation accepts the machine code, and returns a unique session key to the client.
- ExtInstructions. This operation accepts a session key, and returns the initial configuration of the virtual machine.
- Step. This operation accepts a session key and a possible input. The input parameter is ignored in case the instruction pointed by the program counter is not a read instruction. It returns the state of the virtual machine after the step operation.
- Run. This operation accepts a session key and returns the virtual machine's final state.

The server stores the virtual machine 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 virtual machines, we choose to implement from scratch two very distinct specimen. 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, *Apoq*, from *Faculdade de Ciência* of the *Universidade do Porto*.

At this stage, our focus was to prototype the system and get a proof of concept. Thus the integration in the platform was done manually, without the use of the registration web service whose implementation was postponed.

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

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

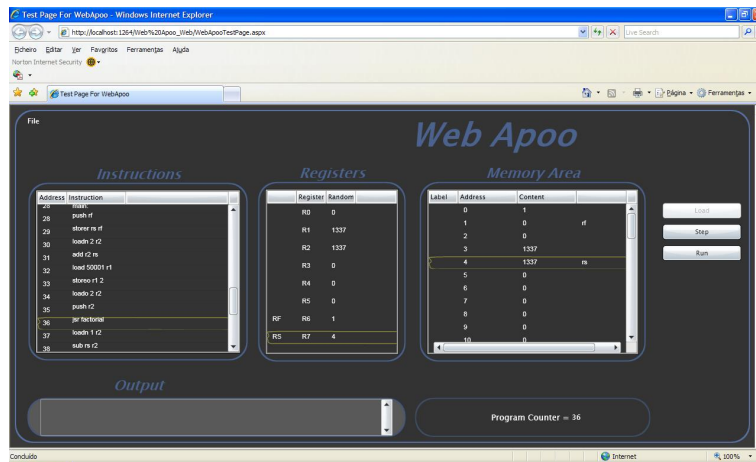


Fig. 2. Web Apoo graphical interface

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

*LRI's VM integration.* Initially developed by Jean-Christophe Filliâtre and Christine Paulin-Mohring, from *Laboratoire de Recherche en Informatique - LRI*, *VM* is a stack based virtual machine. It has an execution stack, a call stack, two heaps and four registers. One interesting feature is its ability to interpret graphical primitives. It is used in *Universidade da Beira Interior*, *Universidade do Minho* and several French universities as a support for compilers course.

As for the last virtual machine, the web version of *VM* works the same way as the original one. The specificity lies in its ability to process graphical primitives. As you can see in figure 3, the graphical client is able, thanks to *Silverlight*, to directly interpret these primitives and displays their result in a specific area.

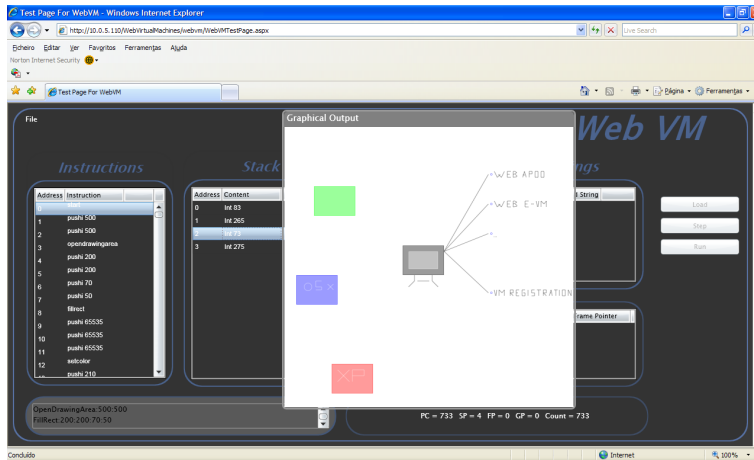


Fig. 3. Web VM graphical interface

*Integrating new virtual machines.* As a result of these first experiments, we were able to understand how to generalize the platform in order to allow the systematic integration of new virtual machines. By specifying what *data* areas are needed, and linking them to a third party virtual machine output, it is possible to achieve this genericity. The implementation of this mechanisms is not yet completed, but despite of this fact, we can already highlight the implementation of the following components:

1. The virtual machine added must provide a *XML* file with the specification of its components (Registers and their type, stacks, memory areas, etc.).
2. Each operation - Load, Step and Run - must produce a *XML* file with the virtual machine's actual internal state (the values stored in the components defined above).
3. If the virtual machine is able interpret graphical instructions, it must produce a *SVG* file with the graphical operations to be interpreted.

Most of the referred virtual machines used in a pedagogical environment are provided with the source code. Integrating them in the proposed platform induce a reasonable amount of changes. For the command line version the changes are quite straightforward. For the graphical version, the main challenging point is the modification of the initialization and *step* functions. When dealing with a virtual machine that already has a graphical interface or a step function, we just need to redirect the information sent by these functions to the interface into an adequate *XML* file.

## 4 Conclusion

We described a proposal for a web based platform for virtual machines 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 multi platform deployment. By simply installing a plug-in, students can interactively see and enjoy the involved mechanisms.

The proposed platform is a work in progress and is developed in the context of an ongoing undergraduate final year project. The platform currently includes the two virtual machines, *Apoo* and *VM* that were manually integrated. The ability to smoothly and automatically add virtual machines is under development.

At this time, the platform has not been tested in a real class environment yet. Although preliminary tests points to a good student's receptivity, a deeper analysis is needed. We plan to make the platform available to the whole community very soon. We will discuss with compilers design teachers in order to let their students to alternatively use the platform. We will include an online form where students can leave their suggestions and point out their platform usage experience.

As future work it is planned the integration of new features to provide more support to students. For instance, provide interactive examples on *loops* translation to machine code.

We conclude by proposing this platform to the compiler 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 courses success.

## References

- [App97] Andrew W. Appel. *Modern compiler implementation in ML: basic techniques*. Cambridge University Press, New York, NY, USA, 1997.
- [CSD08] University of Illinois at Urbana-Champaign. Computer Science Department. Lvm online compiler, 2008. <http://llvm.org/demo/>.
- [fIT06] German National Research Center for Information Technology. The catalog of compiler construction tools, 2006. <http://catalog.compilertools.net/>.
- [Fou08] Perl Foundation. Parrot virtual machine parrot homepage, 2008. <http://www.parrotcode.org/>.
- [LA04] Chris Lattner and Vikram Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.
- [Lar06] James Larus. Spim: A mips32 simulator, 2006. <http://www.cs.wisc.edu/larus/spim.html>.
- [LY99] Tim Lindholm and Frank Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [MG00] E. Meijer and J. Gough. Technical overview of the common language runtime, 2000.
- [PFC08] C. Paulin, J.C. Filliatre, and S. Conchon. Virtual machine for the LRI compiler course, 2008. <http://www.lri.fr/conchon/m1/>.
- [RM01] Rogério Reis and Nelma Moreira. Apoo: an environment for a first course in assembly language programming. *SIGCSE Bull.*, 33(4):43–47, 2001.
- [Wai06] William M. Waite. The compiler course in today's curriculum: three strategies. *SIGCSE Bull.*, 38(1):87–91, 2006.