The Virtual Virtual Machine Project*

Bertil Folliot

Laboratoire d'Informatique de Paris 6, University Pierre et Marie Curie/CNRS
4, place Jussieu, 75252 Paris Cedex 05, France
{Bertil.Folliot@lip6.fr}

Abstract— **This paper presents the invited talk at the "Simposio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho (SBAC'2000)": the Virtual Virtual Machine approach for highly configurable operating systems. This is a joint work with action SOR – INRIA Rocquencourt. More information about the Virtual Virtual Machine project can be found at:**

http::www-sor.inria.fr/projects/vvm

Keywords— **adaptable and flexible system, emerging applications**

. INTRODUCTION

Nowadays systems and architectures are complex, respond badly to many "emerging" application needs, and are not easily specialisable to support a given application or architecture. This leads to a proliferation of non-reusable "ad-hoc" solutions and place stringent requirements on developers. Our response to this challenge is a systematic approach for software configuration based on a multi-language, hardware independent execution platform, called the Virtual Virtual Machine (VVM) [FOL 98].

The VVM offers both a programming and an execution environment allowing: i) to adapt the execution environment (language + system) to a given domain (for instance: smart card, mobile phone, personal computer or satellite), ii) to be extensible by changing on the fly the execution environment (adding new functionalities, algorithms, or upgrading the hardware), iii) to promote interoperability between applications.

Adaptability, extensibility and interoperability offer new solutions to current emerging applications (embedded systems, virtual world, active networks, active spaces and so on). In the following we briefly present the current state of the VVM project, the Recursive Virtual Machine, the first results obtained in the domain of active networks, and the ongoing work for building a reconfigurable execution environment on board of the french satellite Corot.

. VIRTUAL VIRTUAL MACHINE

Emerging applications are coming from the wider acceptance of distributed computing and the ubiquitous use of "intelligent" devices (mobile telephone, smart card, embedded systems on so on). This kind of applications is characterized by dynamic configurations of heterogeneous interacting parties. This lead to an increasing number and complexity of system components – for data manipulation and storage, communication, fault tolerance, mobility, security and so on. We believe that current standard solution (as CORBA, Java or DCOM) while looking as mature technology respond badly to many emerging application needs and will only introduce more problems later on. Comparizons with other existing approaches like flexible operating systems, specializable virtual machine, meta-object protocol, and language ineroperability are not described here.

As an attempt to answer to the emerging application challenge, we propose a systematic approach for software configuration. It is based on a multi-language, hardware independent execution platform, called the Virtual Virtual Machine (VVM). The Virtual Virtual Machine is a programming and execution environment that is dynamically extensible and tailored to application needs [PIU 00]. To help understand the main principles behind the VVM, let's compare with a classical virtual machine (like the SUN Java Virtual Machine [LIN 99]). A virtual machine approach is a step in the right direction and VMs in general are in increasing use to solve operating system problems. Thanks to the bytecode representation, applications are portable and compact, and the instruction set is dynamically specializable. However the Java VM is still far too rigid. It corresponds to an application domain where there is a high amount of available main memory, limited access to the underlying operating system, and no quality of service. This is why, new virtual machines have to be implemented when the application domain does not correspond to these requirements (for a given architecture, as a smartcard, or for a given software requirement as fault tolerance).

Instead of implementing a new virtual machine for each application domain, the goal of the VVM is to "virtualize" the virtual machine. New specifications of VM adapted for an application domain are loaded on demands in the VVM. These specifications are called VMlets. A VMlet contains

* This work is partially funded by a french RNRT project (Phenix 99S0361), France Telecom R&D (CTI ISA 001B117) and the Paris 6 computer science laboratory (PLERS).

in a high level language the complete description of the execution environment (bytecodes, syntax, operating system services, API and so on). The VMlets are themselves encoded in compact bytecoded programs. They are not juste declarative, but imperative and may alter their own execution environment. For instance, according to the environment where they are loaded they may restrain the visibility of given functionalities. Thus, the VVM can be seen as a VM, executing VMlets, that transform the VM to the one contained in the VMlet (see Figure 1). Bytecoded applications can then be run on this specialized VM.

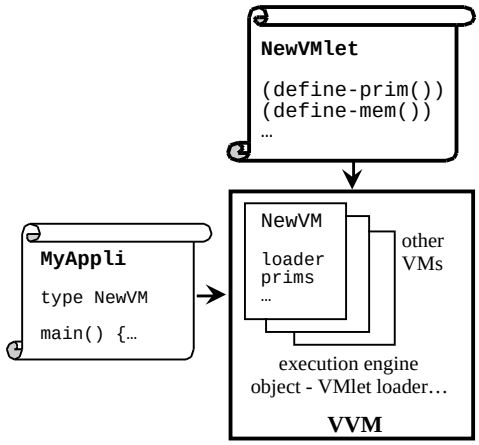

Fig.1 VVM architecture

It is important to notice that after executing the VMlet, the VVM, *becomes* the VM described in the VMlet. Thus, there is a single level of bytecode interpretation. As a result the performance after loading a VMlet in the VVM should be quite similar to the corresponding hand-coded VM. The main advantages are the reduced complexity to program a VMlet and its associated applications, and the extensibility of the VVM allowing to change it on the fly, possibly fundamentally altering its functionality. Moreover the VVM can support several VMlets thus offering a multi-functionality VM and promoting interoperability between applications.

II. THE RECURSIVE VIRTUAL MACHINE

As a first step to the (long) way to the VVM, we realized the Recursive Virtual Machine. The RVM is able to modify its own instruction and primitive sets at runtime. It's a Lisp like interactive language that has been implemented on top of UNIX for several architectures (Pentium, Sparc and PowerPC). By adding or changing instructions sets on the fly, the RVM allows to reconfigure itself to a VM adapted to a given application domain. For now, this application domain should be able to run on top of Unix (we are working on a RVM version based on the OSKit [FOR 97], that will allow the RVM to adapt the operating system itself). Compared to an operating system the RVM looks like a mono-user, mono-application environment. The difficult problems of interoperability between VMlets and the security/verification of the specifications contained in a VMlets are still open.

Though quite limited, the RVM has been used to experiment with the programming of VMlets. For this work, we have considered the application domain of active networks. To simplify, there are two kinds of packets in an active network: data and code packets. Code packets are executed on routers and can manipulate the packets (change, suppress or add new data packets). Among the dozens active network protocols we quoted two: PLAN [HIC 98] and ANTS [WET 98]. In PLAN, each packet contains both the code and the data. In ANTS, there is an initialization phase where the code packets are sent to all the routers. Then, data packets use a code identifier to indicate which code has to handle the packet. Each of these two protocols has advantages and disadvantages - not described here.

To solve with the RVM the active network application domain, we have to define the appropriate VMlet: language, operating system services, API. As language, we didn't consider proposing a dedicated language, and used the Lisp like internal language of the RVM. As system services, we reused the socket primitives of the underlying UNIX operating system (`select()`, `send()`, `receive()` and so on). As API, we have imitated the API of PLAN (`onremote` to send an active packet, `getttl` to get the time to live of the packet and so on). Then, by loading this VMlet, the RVM transforms itself to an active network router that understands PLAN packet.

The main lesson of this exercise is that the VMlet-PLAN is two orders of magnitude smaller than the original implementation (counted as byte of source code). We have implemented an other VMlets that mimics the API of ANTS, and we obtain similar result.

We are in the process of making the two VMlets coexist in order to experiment with interoperability between active networks and to select the most appropriate protocol for *each* packet (i.e. to choose between memory used and communication bandwidth). The next step will be to build an "active active network", where traditional "active networks" contains VMlets packet.

Remember that this work has been done using a limited prototype. Adding change to the underlying operating system, or allowing several VMlets to run simultaneously (next version of the RVM), will allow exploring much more complex application domains. A target one, that is a generalization of the active network work, is what we call "active application". As in active network a packet contains code to manipulate the data, in active application, an application contains code to manipulate the application (both code and data).

III. Reconfigurable execution environment on board satellite

A challenging execution environment is a reconfigurable software environment on board satellite. This application domain represents an extreme case of embedded systems, posing severe constraints as, memory footprint and communications optimizations, real time, limited processing power, fault tolerance and so on. In collaboration with the Space Research department of Paris Observatory we are creating a dedicated version of the VVM to the french satellite Corot (to be launched in 2005). The scientific mission running on this satellite is based on theoretical models that will only be tested during the flight. Thus, the software on board of the satellite should be adaptable according to the physical conditions observed (see [CAI 99] for details on the needed reconfigurations).

Our work is to provide the configuration language (called Corot Configuration Language, CCL) and the associated interpreter, allowing the scientific station on the ground to describe optimized and proved versions of new configurations to be uploaded on the satellite. This is one of the first attempt to propose a systematic and provable way to reconfigure very constrained embedded system. CCL and its interpreter are very tight to the environmental conditions of Corot (2 to 4 non interactive communications by day, 140kbits/day of information in the uplink, limited memory, CPU utilisation for reconfiguration limited to 10% and so on). The next step will be to generalize this work for reconfiguration in severely constrained embedded systems, making the environmental conditions themselves adaptable.

IV. Conclusion

The Virtual Virtual Machine project consists to build a programming and execution environment based on adaptability, extensibility and interoperability. Its main goal is to provide a maximum of support for many essential aspects of the emerging distributed application domains (embedded systems, virtual world, active networks, active spaces and so on). The first prototype, the Recursive Virtual Machine already show the efficiency to write simple execution environment (called VMlet) for active networks. We are now working on several application domains, as "active active networks", "active application" and reconfiguration in severely constrained embedded systems. Other applications are being considered as dynamic aspect-oriented programming and adaptable fault-tolerance.

Acknowledgments

The works presented are being done with the VVM team: Carine Baillarguet, Christian Khoury, Arthur Léger, Fréderic Ogel, and Ian Piumarta.

References

[CAI 99] CAILLAU, Damien; BELLENGER Remy. The Corot instrument's software: towards intrinsically reconfigurable real-time embedded processing software in space-borne instruments. Proceedings of the *4th IEEE International Symposium on High Assurance System Engineering*, Washington DC, USA, Nov. 1999.

[HIC 98] HICKS, Michael et al. PLAN: A Packet Language for Active Networks, Proceedings of the *International Conference on Functional Programming*, 1998.

[FOL 98] FOLLIOT, Bertil; PIUMARTA, Ian; RICCARDI Fabio. A Dynamically Configurable, Multi-language Execution Platform. Proceedings of the *ACM SIGOPS Workshop*, Sintra, Portugal, Sept. 1998.

[FOR 97] FORD, Bryan et al. The Flux OSKit: A Substrate for Kernel and Language Research. Proceedings of the *16th ACM Symposium on Operating Systems Principles*, Saint-Malo, France, Oct. 1997.

[LIN 99] LINDHOLM, Tim; YELLIN Frank. *The Java Virtual Machine Specification*. Addison-Wesley, 1999.

[PIU 00] PIUMARTA, Ian et al. Highly Configurable operating systems: the VVM approach. Proceedings of the *ECOOP'2000 Workshop on Object Orientation and Operating Systems*, Cannes, France, June 2000.

[WET 98] WETHERALL, David; GUTTAG, John; TENNENHOUSE David. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. Proceedings of *IEEE OPENARCH'98*, San Fransisco, USA, Apr. 1998.