The Pharo VM: a statement from the Pharo consortium

This is a statement of the consortium related to the past, current and future effort around the Pharo-VM.

This statement is provided to clarify the situation. The consortium's goal is to build and offer an infrastructure that supports the growth of business around Pharo. Companies will have access to a robust, fast, tested and documented system that can evolve.

The goal of this document is to summarize our actions and show where we are heading at. We evaluated the planned effort for at least a couple of years, probably more.

All the consortium past and current actions have been done in a public way. We have been talking about them in all our public channels, and today we are just summarizing them.
All our efforts are available under the MIT License and hosted under
https://github.com/pharo-project/opensmalltalk-vm.

There have been issues in the past that made us work in a separate repository. These issues persist and they are now part of our history and the basis for the decisions made by the consortium. We did a deep analysis of the situation and took actions to reach our goals. Still, all the results of our actions are open-source and available.

Context:

Investing and improving the Pharo-VM has been voted during the consortium General Assembly in 2019 during the PharoDays that were held at Lille. At that time it was recognized that the VM was a factor of risks: truck factor, lack of documentation, lack of large tests, impossibility to trace builds correctly, the duality of version control systems, ...
Since the goal of the consortium is to sustain the development of business and use of Pharo, actions to remedy such a situation had to be taken.

Following this vote, the consortium put in place several actions for at least a couple of years that are described in details hereafter:

- Controlling the VM build and development process in terms of traceability.
- Improvements around the VM (event, FFI,....).
- Increasing test coverage of the core elements to prepare changes.
- Development of threaded-FFI and idle VM (especially required for Android).
- Documentation and growth of knowledge.

The objectives of these actions are to deliver a better VM that can evolve during the next decades.

Vision.

Our objective is to have a VM:

- Robust.
- Tested.
- Documented.
- With a good level of code quality.
- Validated with modern tools.
- Ready to evolve and tackle new challenges.
- Open to the community.
- Without additional accidental complexity.

Acknowledgments.

The Pharo-VM uses a version of the OpenSmalltalk-VM. We are thankful and recognize all the excellent work that is present in the OpenSmalltalk-VM. We have stated in all our communications and publications the correct acknowledgment and credits to the initial and actual authors. As an example, in the code migration to GIT, we have been more than careful to correctly credit the authors of all the changes - We thank Feenk for the original effort. The current documentation effort of the Spur architecture clearly also states that the consortium is not the original authors

Historical point.

Pharo was originally using its own VM based on the Squeak VM. Inria-RMOD paid Igor Stasenko to build a build system for such a VM. It was using CMake and the goal was that everybody could recompile it. This goal was reached. Then during the merge with the OpenSmalltalk-VM, the consortium lost all its build infrastructure because CMake was discarded.

It should be noted that the Inria-RMOD team paid C. Béra for 6 years to work on the OpenSmalltalk-VM. This led to a new bytecode set, a part of the Spur implementation (GC), fast become, read-only objects and the Sista architecture. The code owned by Inria-RMOD was given to OpenSmalltalk-VM.

Actions

To be able to support the actions presented above and address the problems identified and presented to the consortium members, the consortium engineering team decided to take control of the code base of the VM.

In the last year and a half, a lot of progress has been done in the VM. We have centered our effort into the following axes:

Having a simple reproducible process that can run in many installations producing easy debuggable, embeddable and extensible VM. Integrating with a simpler CI infrastructure that we can control and configure for the desired target platforms:

- Documenting all the processes and the details of the VM, plugins, and frontend.
- Pushing towards the generation of an embeddable event-based VM.
- Unifying the platform-dependent implementations of the plugin and the VM. Simplifying and encapsulated all that cannot be unified and providing strictly the same API and functionality to all of them.
- Building up the support for new development on the VM, the multiplication of experiments in a controlled way and the development of research and productive solutions.

Current VM branch explanation

Our fork of the VM is hosted in github.com/pharo-project/opensmalltalk-vm.

This repository includes different versions of the VM. We have the version corresponding to the released version for Pharo 7 & 8, development versions and experiments.
Each of these versions is in a different branch.

As described in https://github.com/pharo-project/opensmalltalk-vm/wiki/PharoVM-Versions, the Pharo repository contains the following branches:

Pharo: This branch hosts the stable version used by Pharo 7 and Pharo 8. This version only has small improvements to make it compile and it is the latest version of Opensmalltalk-VM that was promoted as stable in Pharo.

dev7: In this branch, we make any change that should be backported to the VM used in Pharo 7 and Pharo 8. Every change that we do that requires a backport to Pharo 7 & 8 is stored in this branch. This branch allows us to build the same VM that is used in production, fixing in this branch any problem or incident.

Cog: This branch is in sync (or tries to be) with the corresponding branch of the Opensmalltalk-VM repository. As this is the main branch in their repository, we use it to fetch changes. This branch has not been modified by us in any case, and no modifications should be pushed here. This branch should be also the starting point of any branch used in a PR to the Opensmalltalk-VM repository.

headless: This is a development version. It removes all events and window handling from the VM and passes it to the image. The image is responsible to open or not a window and to handle the events. We have two different window libraries in the image: SDL and GTK+. By default an image uses SDL. All Pharo 8 and Pharo 9 images are compatible with this VM.
This version can be heavily used and tested. It is working for Windows, OSX, and Linux. Check the documentation page if you want to experiment with it. To get this version please follow the instructions here
(https://github.com/pharo-project/opensmalltalk-vm/wiki/Documentation#how-to-use-the-sdl-support)

idle: This is a development version. It adds an interruptible idle loop to the VM. This modification reduces CPU usage. It does not do an active event checking and the handling of I/O events has been improved. This VM does not present CPU consumption when idle. When a socket or an event arrives the VM is woken up. Also, the asynchronous socket and file implementation have been unified on all platforms.

GC-tests: This branch contains work in progress. This branch has the tests that we have developed for testing the memory model, the garbage collection and the JIT. Our objective is to test all the validations and assumptions we do about the execution of the VM (GC, native code,.... These tests are the base of the documentation process.

feature/cogmt: Experimental version. It implements the multi-threaded interpreter using a global interpreter lock. Only one of the interpreters is running at the time. [Details about the design](https://hal.archives-ouvertes.fr/hal-02379275/document).

experiment/live-typing-stack: This is an experimental branch. In this branch, we have the live typing proposals done by Hernán Wilkinson. This version only has the modifications for the VM without JIT.

experiment/live-typing-cog: This is an experimental branch. The idea of this branch is to implement the live typing extensions to the code produced by the JIT compiler.

Virtual Machine Documentation

We are starting to document the Virtual Machine as a form of a Pharo book.

https://github.com/SquareBracketAssociates/Booklet-PharoVirtualMachine
The contents is limited for now. What is important to understand is that we are reverse-engineering the Virtual Machine logic as executable unit tests and they are committed in the gc-tests branch of this repository. We will continue to extend this book with the knowledge we acquire. It is a slow process but we have regular meeting of test writing and our test suite grows.

There are also other OpenSmalltak-VM sources of documentation

https://clementbera.wordpress.com
Clement is the designer and implementer of the Sista system.

http://www.mirandabanda.org/cogblog/
Eliot is the father of the Cog VM and Spur memory management.

Improvements

Following these objectives, we have performed a lot of improvements. The changes were made while we were learning in the meantime and trying to help other people to learn. Details of the changes are included after.

Some of these improvements are already available in the version in the branch *headless*.
The rest of the changes, mostly the ones related to the event-based are in the branch *idle*.
The idea is to push these VM as the future VM of Pharo 90.
All these changes are openly available in our Github repository if the Opensmalltalk-VM community considers them worthwhile. We are open to help in any porting activity that is initiated from the OpenSmalltalk-VM community. As the past has shown it is likely to have pull requests rejected we refrain on creating them for now.

Details of Enhancements

Code Migration to GIT for Reproducible Builds

Based on the work made by Feenk, we have migrated all the source version control to use Git. We have migrated the full story of changes and given credit to the corresponding authors. This allows us to have a single tool to handle the Smalltalk and the C code in a single repository. This uniqueness of the repository allows us to correctly tag the versions and have reproducible builds.

Versioning of the Third-party dependencies

To improve the stability of the build and to have reproducible builds, we have started versioning all the third-party dependencies. Not having control over the dependencies, and using the ones installed in the build environment produces non-reproducible builds.

Process in Pharo 9

The whole process of generation of the sources, execution of the simulator and the tests are nowadays ported to Pharo 9, but were working in Pharo 8.0. The simulation of the JIT is under redevelopment to use the Unicorn library (Unicorn is a lightweight multi-platform, multi-architecture CPU emulator framework). The idea to avoid having one simulator back-end per architecture.
The build process generates the C code from Slang in each build. We have improved the determinism of the built process, although we need to continue working on it to fix some random errors.
This allows us to use all the latest Pharo tools on the VM code and especially Iceberg. We are still improving the development experience in Pharo 9 as we are working to introduce some extra tools to improve the development experience (debuggers/inspectors/disassemblers/processors simulators).

Controllable CI infrastructure

We have implemented a CI infrastructure that we can control and provide the desired versions of the build and test environment. Now, we have the ability to take the decision of when we want to upgrade a version of the operating system or its dependencies. Also, Ronie Solgado has added the ability to run the process in Github actions, so all the forks can be also tested.

Introducing (back) CMake

We have implemented the full build process using CMake. This allows us to have a more declarative build process, better support for different build systems and IDE integration. IDE integration with Eclipse CDT and XCode, the integration with VisualStudio is on the way.

Benchmarks

We have introduced a series of VM benchmarks in the set of benchmarks run in the Pharo built. These benchmarks are for addressing that none of the changes produce an impact that is not desired. To correctly implement these validations, we should control the integration server. We still have to improve at this level, adding more benchmark scenarios, testing systems, reports, and automation tools.

Code Clean-Up

In the VM there was a lot of dead code and conditional code that was never executed, a lot of this code has been cleaned reducing the size and complexity of the VM and the plugins. We will continue this action.

Improving Logging

The VM now has a consistent portable way of logging messages and error. We have introduced a small logging framework to inform the messages and errors using the correct streams and log support of the target system. Like this, the logging can be activated or deactivated from the command line and provides additional debugging and error tracing capabilities.

VM as a Dynamic Library

In the cleanup and reorganization of the VM code, we have achieved that all the parts of the VM can be loaded as a dynamic library. All the plugins are external plugins and the VM is a single DLL that can be shipped in different applications. Like this, we have divided the VM into two parts a frontend and the VM itself. The Frontend is responsible for looking up the image, and process the parameters of the user and then configure and launch the VM. These changes open the door to have different versions of the VM just by changing the frontend. Allowing not only embedding of the VM but the branding and shipping of products.

Unifying Files & Socket Async Implementations

We have unified all the file and socket implementations, now all the OSes use the same implementation. This reduces the size of the code and eases its modification. Moreover, it is better to introduce optimizations as there is a single point to optimize. These modifications produce an improvement in the performance of Windows file access.

Allowing Multiple image lookup and loading strategies

To allow the development of products and embeddable solutions, we have to provide a way of customizing the image lookup, loading, and saving. We have modified the VM to allow customizable strategies to find the image, loading and saving it. The frontend configures the desired strategy or provides a new strategy following the same interface. With this, it is possible to develop read-only applications, embeddable applications, and applications that download the image if required.

Embeddable Support

Since for certain businesses it is important to provide only one executable file, using the provided support, we have shown that it is possible to embed the VM to run a given application. We have tested getting the image as a resource in the executable file and executing Pharo code from another application.

Queue Based Non-Blocking FFI

We have implemented a new solution for a threaded FFI that allows us to execute non-blocking FFI calls. FFI stands for Foreign Function Interface. A foreign function interface is a popular name for the interface that allows code written in one language to call code written in another language. This solution uses a combination of different strategies to cover more different libraries, as each library requires different types of behavior. This solution uses libFFI and it is loadable as a plugin. The libFFI library provides a portable, high-level programming interface to various calling conventions (https://github.com/libffi/libffi). This allows a programmer to call any function specified by a call interface description at run time. By using this library we have a portable efficient way of doing FFI calls. This library is used extensible by projects like CPython, OpenJDK, Ruby, GHC, and Racket. The VM has been modified to allow the development of FFI as a plugin. This solution also permits the safe execution of callbacks that are initiated from the library and not from the VM thread. This solution is thread-safe. Using this solution the support for GTK+3 has been implemented and it is heavily used by different library bindings.

Event-Based VM

We are working on the development of a real event-based VM. This development is already available (as it has been from the beginning) but it requires some stabilization. This VM version heavily reduces CPU consumption while not executing any Pharo code without affecting the performance during the normal execution.

Non-Blocking FFI using a Global-Interpreter-Lock Experiment

We have recovered the existing code of CogMT. However, this version is far from complete as there are issues in its design as some of the problems are not addressed. Moreover, it presents the same problems that have been extensively documented in the bibliography about the use of global-interpreter-locks. The objective of this task was to document it, add tests and compare it with other solutions. As a result of this task, an article presenting the challenges and comparing the different solutions has been presented at IWST 19. Such an article, as is publicly available, recognizes and credits the original authors of the idea. We have done this to document and preserve the work that has been done opening the door to everyone else working in this subject.

JIT Testing

We are adding tests to all the just-in-time compiler code. We are correctly simulating the execution of all the code and running them in a simulated environment by using Unicorn. The previous simulation was depending on old versions of GDB and Bochs, those versions were inside the repository violating the license of GDB and Bochs and conflicting with the VM license. With this support, we will generate tests for all the bytecodes and different JIT configurations. With this test baseline, we are introducing new documentation and in the near future modifications and the generation of an open-source ARM64 JIT.

GC Testing

We are in the middle of the process of documenting the GC and the memory model. We are doing so by writing many tests on different aspects of the current memory management implementation. This live documentation allows us to not only validate every assumption we produce but to validate every change we made.
Having tests opens the door to a future work of refactorings to improve the quality and clarity of such code. And in the near future the addition of new features.
We consider that tests provide the solid ground that is required to build up the future VM in a more productive and open way.

Documenting

From all the information we are getting we are writing a book about the inner details of the VM. We are documenting everything we find and we reference the tests and the existing code. Like that, we can help others to start from a better point that we started. The book is not only available to everyone, but it is also open-source.

How to Contribute

We are open to receive contributions to the project, the best way of contributing is creating a PR in the Github repository (https://github.com/pharo-project/opensmalltalk-vm). Depending on the modifications it should target any of the branches. The default branch is the headless VM as it is the VM under heavy development.
We also are very thankful to receive issue reports and ideas for improvements as issues in this same repository.

Conclusion

As we have said, this document states the position of the Pharo consortium regarding the status of the VM. Our main objective is to provide a VM that allows users and companies to support their future, giving them a solid ground to perform businesses, research, and teaching.

We are closely coupled to the Pharo community and we want to provide the VM this community deserves. The objectives and plans explained have not been taken in secret, we have done them publicly ever since.

We are heavily working to improve the VM in all the objectives we have stated. We believe that we will start to see the effect of our effort in Pharo 90. The improvements we are pushing are not only to fix specific problems but to relaunch the development of the VM to the future. It is not enough to patch, fix, and improve little parts of the VM. We are pushing it to make it easier to maintain, open to everyone that is willing to learn. We know the VM is far from being simple, but we want to reduce all the complexity that is not needed. We really want an open VM that can support the future of all the community. This is why we are investing in tests because tests are change enablers.

As we would like to remember that being an open-source project is not only publishing the code. A correct open-source community makes the code available with the correct documentation and with the willingness to help newcomers to use, learn and collaborate.

This is a key point in the Pharo community and we need to improve this at the VM side also. We need to break the frontiers and give the tools to the people to learn, helping the growth of everyone. We see a world of equals and not a world where an elite is only allowed to modify some parts. As always we want to democratize the code, the knowledge and why not the fun. Because open-source projects are about freedom.

There is much more room to improve, we are just starting, sometimes faster and sometimes slower. Now if you like Pharo and want to join the VM development do not hesitate. We will systematically consider your submission, give feedback.