Critique of 'Let Them Fail: Towards VM built-in behavior that falls back to the program'

ANTRANIG BASMAN, Raising the Floor - International, England

We supply a critique of the paper *Let Them Fail: Towards VM built-in behavior that falls back to the program*, suggesting directions in which its work of negotiating the boundary between the responsibilities of VMs and their hosted programs can be continued, and questioning our current disciplinary separation between systems and languages research.

CCS Concepts: • **Software and its engineering → Virtual machines**; **Extensible languages**; **Control structures**; **Error handling and recovery**; *Layered systems*; *Feature interaction*; • **Hardware →** *Emerging languages and compilers*;

ACM Reference Format:
Antranig Basman. 2019. Critique of 'Let Them Fail: Towards VM built-in behavior that falls back to the program'. 1, 1 (July 2019), 2 pages. https://doi.org/10.1145/3328433.3338057

1 INTRODUCTION

Let Them Fail analyses the division of responsibilities between virtual machines, hosted applications and language features, particularly in the context of Smalltalk code running within a Squeak VM. The paper makes welcome contributions in several areas:

- Taxonomy of interaction modes available in determining what execution is appropriate when the client of VM services dispatches an "error"
- Speculations about the role of a VM, its "Sphere of Influence" and style of interaction with running programs
- A recognition of the independent evolution cycles of VM and programs, how this independence might align with different choices of interaction modes

The taxonomy of interaction modes itself is comprehensive and sound, given the particular framing assumptions in the paper. This critique will focus on these assumptions themselves, and consider how the relationship currently embodied in the separation between VMs and their client programs could be more productively reframed.

2 AN ERROR BEGINS A NEGOTIATION

The paper rightly makes the point that invocation of VM primitives should be the start of a more open negotiation than is the case in many traditionally constructed VMs/runtimes in which an invocation found to be in error causes the VM to bail. The paper surveys nine variants of "fallback behaviour" which can be made available within Smalltalk VMs, when a primitive is found to be "unsuccessful". However, waiting for the actual "point of contact" between an application and the VM, as embodied in a function call made by the former to the latter's API, may well not be the most appropriate integration strategy for honoring many kinds of authorial intention.

There are some concrete cases, for example, where this point of contact may simply occur too late — consider a client who wishes to express an intention to the VM's virtual memory system that a particular piece of the implementation has weak requirements for user latency and should be pushed out of main memory if this becomes congested. Waiting for a function call to execute implies that the client must already be executing in main memory, defeating its intention. However, there is a much wider space of authorial intentions where "point of contact" interactions are inappropriate, since these are only available for inspection and interception by other privileged parties.

Further, the very nature of "errors" themselves is called into question by the treatment of [8]. This includes a discussion of the Erlang-inspired "let it crash" model of failure handling, echoing the paper's title itself, but more importantly tries to move beyond the "Algol research programme" model of an error as something which could be excluded from the design of a correct system by means of proof. Instead, one thread of the discussion positions errors not as an end to an interaction, but instead just the beginning of a negotiation between a group of interacting authors on the expected meaning and behaviour of a computational artefact.

3 HELICOPTER VM OR FIRE AND FORGET?

Let Them Fail quietly invokes the notion of an intrusive, intolerant and constantly hovering "helicopter parent" in its metaphor for the relationship between a VM and its hosted processes. It rightly argues that VMs should share more responsibility for the correct operation of the system with the programs they are running. However, it may be that the VM's operation needs to be more tightly integrated with those of its programs and not less. The VM ends up operating an intolerant model for failure because of the necessarily arms-length nature of integration that is possible via APIs. In many important cases it cannot be informed of unwelcome developments or violation of contracts until it is too late.

The "Spheres of Influence" discussion in the paper thus quietly embeds some paradigmatic assumptions:

- That this is a hierarchical relationship of just two parties
- That authorial intents are encoded by expressions directly written in source text under full control of one party

Going beyond the integration model entertained by the paper, a key approach would hoist richer metadata about the expectations of "primitives" out of the VM/OS into an integration domain [4] where these may be brokered against the expectations of invokers,

Author's address: Antranig Basman, Raising the Floor - International, London, England, amb26@ponder.org.uk.

XXXX-XXXX/2019/7-ART $15.00
https://doi.org/10.1145/3328433.3338057

providing a richer and more open structure than can be achieved by relying on *post facto* runtime failures alone.

Continued questioning of the mediating role of the operating system is found in [5], which does indeed seek the "lurking Smalltalk" latent within today's actual operating systems, and considers what might be necessary to mature these binding and integration facilities into a holistic system.

4 AUTHORIAL ROLES AMONGST SYSTEMS AND PROGRAMS

Further developing his thought on how richer networks of metadata may be orchestrated into an integration domain which coordinates the relationships of components interacting at a system level, [6] identifies as latent within Unix some courtesies which are not typically considered part of its specification but in practice are built up to support a breadth of authorial roles (in practice often driven by the debugging role), including:

- An explanation and ontology for the contents of memory
- Facilities for introspection and interception

Smalltalk VMs such as Squeak supply many of these courtesies, but in practice the breadth of roles that can be supported by an API integration model are limited.

As a low-level example of a limitation of embodying the integration relationship as an API separating a 2-party ecosystem composed of "VMs" and "programs", consider the relationship of primitive built-in behaviour and fallback code described in the paper. This fallback process can only be initiated in the case the VM service provider has first signalled an error due to the preferred implementation not being available. This implies a fixed pipeline of choices among the interacting parties — unless the service provider raises an error, the fallback cannot be selected. Considering, for example the case in section 3.6, one could imagine situations where this might not be appropriate. One author has produced a native, fast primitive that is optionally available, and another has implemented a fallback. Now imagine that a third author wants to declare that, despite the fast primitive being available, they nonetheless prefer to activate the so-called "fallback" (perhaps it performs the FFT with greater accuracy). [1] is a useful presentation of modelling authorial networks of this kind, and their economics.

5 LANGUAGE PRIMITIVES AT A SYSTEM LEVEL

What consideration of this pipeline suggests is that what is being modelled is a kind of dispatch, plus a form of method combination — approaches to which there are a huge wealth of in the literature. To start with, the context-oriented programming framework of [7] is an interesting scheme for modelling such dispatch and the potential for plural influences on it — it would be highly interesting to see a treatment considering this at the scope of an entire VM and its contained applications. Other approaches to plural dispatch include the "symmetric dimensions of context" approach of Korz [9] and Bracha's Newspeak [2]. Each of these approaches offers some limited capability to remedy the power imbalance between these participants rather than the "in-code" approach to remedying an already encoded decision to reject execution taxonomised in the paper.

The difficulty is that this wealth of literature lies in a discipline which is considered distinct from the subject–matter of the paper. [3] reflects on the unwelcome historical process by which "systems" work came to constitute a separate discipline from "languages" work, and how these disciplines started to address their respective problems in different ontologies and in different venues. The result is that it becomes hard even for members of the same research group to fruitfully consume each others' work if they lie on different sides of this disciplinary divide.

6 CONCLUSION

The dominant integration schemes for system-level components involve a 2-party "point of contact" idiom in which the parties are connecting by an API embodied as a function call, which may then make a regular return, or signal an error. These schemes are authorially inhibiting and result in systems which fail to put the complete capabilities of the system at the disposal of the user, as a result of the opaque nature of this boundary and its lack of timely operation. We need richer ontologies of live, system-level metadata to describe interacting systems within an integration domain in the sense of [4], dissolving the particular kinds of boundaries separating processes, applications and OS components. In such a scheme, the disconnection between system-level and language-level descriptions of system activity will be healed, allowing both for transparent reasoning about the behaviour of systems as a whole, together with system-level adaptations which can reach into parts of a design traditionally considered the preserve of programming languages.

REFERENCES

[1] Antranig Basman, Clayton Lewis, and Colin Clark. 2018. The Open Authorial Principle: Supporting Networks of Authors in Creating Externalisable Designs. In *Proceedings of the 2018 OOPSLA Companion (Onward)*. 29–43.

[2] Gilad Bracha, Peter von der Ahé, Vassili Bykov, Yaron Kashai, William Maddox, and Eliot Miranda. 2010. Modules as Objects in Newspeak. In *ECOOP*.

[3] Richard P. Gabriel. 2012. The Structure of a Programming Language Revolution. In *Proceedings of the 2012 OOPSLA Companion (Onward)*. 195–214.

[4] S. Kell. 2009. The mythical matched modules: overcoming the tyranny of inflexible software construction. In *Proceedings of the 2009 OOPSLA Companion (Onward)*. 881–888.

[5] Stephen Kell. 2013. The Operating System: Should There Be One? *(PLOS '13)*. Article 8, 8:1–8:7 pages.

[6] Stephen Kell. 2015. Towards a dynamic object model within Unix processes. In *Proceedings of the 2015 OOPSLA Companion (Onward)*. 224–239.

[7] Jens Lincke, Malte Appeltauer, Bastian Steinert, and Robert Hirschfeld. 2011. An Open Implementation for Context-oriented Layer Composition in ContextJS. *Sci. Comput. Program.* 76, 12 (2011).

[8] Tomas Petricek. 2017. Miscomputation in software: Learning to live with errors. *Programming Journal* 1, 2 (2017).

[9] David Ungar, Harold Ossher, and Doug Kimelman. 2014. Korz: Simple, Symmetric, Subjective, Context-Oriented Programming. In *Proceedings of the 2014 OOPSLA Companion (Onward) (Onward! 2014)*. 113–131.