VMADL: An Architecture Definition Language for Variability and Composition of Virtual Machines

Stefan Marr and Michael Haupt
Hasso-Plattner-Institut, University of Potsdam, Germany
{stefan.marr,michael.haupt}@hpi.uni-potsdam.de

Abstract—High-level language virtual machines (VMs) can be used on a wide range of devices as a basic part of the deployed software stack. As the available devices differ to a large degree in their applications and their available resources, distinct implementation strategies have to be used for certain parts of a VM to meet the special requirements. This paper motivates the need for an architecture definition language for complex software systems like VM implementations. The basic concepts and language constructs of this language, which is called VMADL, are introduced. To motivate further discussions, the benefits of this approach are briefly discussed.

Index Terms—Modularization, Language, Variability, Features, Product Lines, Virtual Machines, Composition.

1 Introduction

In industry and commerce, software is an inherent part of the value chain, not only as part of the business processes but also as part of almost every technical product which is produced. With the rapidly changing business requirements, software needs to be adapted and customized for specific requirements frequently. Moreover, software systems are frequently required to be deployed in different environments, in all of which they are expected to exhibit the same, or very similar, functionality. In many cases, it is not only a matter of enhancing a software product by new features, but it is also necessary to be able to omit features, or even to realize them in different ways depending on the requirements imposed by the deployment environment.

Up to now, this is most often a problem of software evolution and thus, has been solved by solutions specifically designed for each particular software system. A product-line oriented view on software systems seems to be more accurate as it fosters variability while maintaining a consistent perspective on the overall system.

In the field of high-level language virtual machines (VMs) [1], the challenge is to be able to compose a VM from different sets of features for a wide range of application domains. Highly tangled modules [2] have to be interchangeable. Different implementation strategies have to be available to build VMs for systems ranging from resource-constrained systems, e. g., embedded or mobile devices, to standard consumer systems for desktop applications and even large high-performance computing systems.

To achieve this aim, language support for coping with the complex challenges which are inherent to VM implementations is required. Mechanisms to handle cross-cutting concerns which are not manageable by usual decomposition techniques are necessary as well as is support for a more advanced way to describe module interaction than would be possible with languages typically used to implement VMs.

In the next section, we introduce the basic concepts proposed by the *Virtual Machine Architecture Definition Language* (VMADL) [2] to be able to achieve a higher degree of variability and composability in complex software systems such as high-level language virtual machines. In Section 3, our results are briefly discussed and a conclusion is drawn to encourage further discussions and work to assess the ideas and applicability of VMADL.

2 VMADL

The main aim of the Virtual Machine Architecture Definition Language (VMADL) [2] is to provide module descriptions on an architectural level and to enable developers to explicitly express module interactions and, thus, describe the overall architecture more declaratively.

A *service module* defines the service boundaries in terms of an interface and of interactions with other modules. It can comprise several implementation modules for which it provides a combined *bidirectional interface* definition on an architectural level. Typically, it consists of function definitions and additional "points of interests", i. e., pointcut definitions provided for usage by other service modules. Furthermore, relationships with other modules are defined in a structured way to improve the variability and definition of optional dependencies to allow module interchange. VMADL's bidirectional interfaces are similar to, and inspired by, open modules [3] and crosscut programming interfaces (XPIs) [4].

2.1 Language Constructs

At the current stage of development, VMADL incorporates language constructs specifically designed for the modularization of software systems with highly intertwined modules, as usually found in VM implementations.

The basic entity described is a service module which typically relies on the availability of other modules it depends on. The **require** statement expresses this dependency explicitly to make it visible to the developers.

The main body of such a service module definition can hold, e. g., the definition of functions or the public state available to other modules. Interaction among modules is defined here, too, using aspect-oriented means. For consistency, it is expected that such aspects will only make use of functions and pointcuts provided explicitly by module definitions. The **expose** section is used to specify additional "points of interest", i. e., events defined by pointcuts that are exposed to other modules.

Named sections are meant to provide additional structure to a service module definition and can hold the same content as the overall service module definition. In addition, these sections can be used to replace specific parts of a module definition from within another module. This is necessary to be able to adapt parts of very basic modules, e. g., object model definitions.

```
service MyServiceModule {
  require OtherModule;

  // declaration of functions
  // and import or #includes
  #include <module.Foo>
  void MyServiceModule_doSomething();
  ...

  // advices on service module level
  advice execution(OtherModule_doFoo())
        : before() { ... }
  ...

  expose {  /* list of pointcut
     definitions */  }

  for OptionalModule { ... }

  MyNamedSection { ... }

  replace SomeModule.NamedSection { ... }

  startup {
    void initialize_me();
    expose { ... }
  }
  shutdown { ... }
}
```

Listing 1. VMADL Language Constructs

Optional sections are identified by the **for** keyword and the name of the module for which the definitions inside this part are needed. Like for named sections, the same constructs as for the overall service module can be used inside of an optional section.

In addition to the language constructs introduced here, an object definition language [5] might be necessary to be able to refine classes defined in another module. There are cases where a module needs an additional field or method within a class, which could be handled by such a language if the used implementation or aspect language does not already provide a mechanism for inter-type declarations.

3 Conclusion

VMADL was specifically designed for the modularization of virtual machines, but could be used for other software systems as well. It provides constructs for an increased variability and composition of modules in a product-line like manner. Furthermore, it locates architectural concerns like dependencies and interaction between modules in a more explicit and descriptive way to help developers better understand the complex coherences in such software systems.

As far as our work has allowed us to assess the benefits of this approach at the moment, VMADL shows valuable advantages over the pure implementation and the employment of aspect languages. Furthermore, it improves maintainability of the software, since the developers are able to get a better understanding of module interaction and relationships in the system and thus are able to implement changes to the system more efficiently.

This conclusion from the results of our work is subject to evaluation in our ongoing work. We will use software metrics to be able to underpin the results with some basic figures according to complexity and performance. Furthermore, an experiment with students will be conducted to be able to determine the degree to which VMADL supports reasoning about a system's architecture and whether module dependencies are recognized more easily and accurately than without VMADL.

References

[1] J. E. Smith and R. Nair, *Virtual Machines. Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.
[2] M. Haupt, B. Adams, S. Timbermont, C. Gibbs, Y. Coady, and R. Hirschfeld, "Disentangling virtual machine architecture," under review for *IET Journal* special issue on *Domain-Specific Aspect Languages*, 2008.
[3] J. Aldrich, "Open modules: Modular reasoning about advice," in *ECOOP 2005 - Object Oriented Programming: 19th European Conference, Glasgow, UK, July 25-29, 2005. Proceedings*, ser. LNCS, vol. 3586. Springer, 2005, pp. 144–168.
[4] W. G. Griswold, K. Sullivan, Y. Song, M. Shonle, N. Tewari, Y. Cai, and H. Rajan, "Modular software design with crosscutting interfaces," *IEEE Software*, vol. 23, no. 1, pp. 51–60, 2006.
[5] S. Timbermont, B. Adams, and M. Haupt, "Towards a DSAL for Object Layout in Virtual Machines," in *3rd Domain-Specific Aspect Languages Workshop*, 2008, http://dsal.dcc.uchile.cl/2008/_Media/timbermont.pdf.