Requirements for Object-Oriented Systems Modeling with STEAMS

Gregory D. Peterson
The University of Tennessee
gdp@utk.edu

Phil Wilsey
University of Cincinnati
phil.wilsey@uc.edu

Abstract

Designers face the challenge of specifying and implementing complicated mixed-technology systems. In order to better address mixed-signal designs, the VHDL-AMS and Verilog-AMS languages have been developed. These languages provide powerful capabilities to model and simulate behaviors in both the continuous and discrete time domains. Contemporaneously, the control systems community developed the object-oriented Modelica language to support the specification and continuous time modeling of complex control systems. The STEAMS (SUAVE and Tennessee Extensions for Analog and Mixed-Signal Systems) effort strives to provide an object-oriented systems specification and modeling language that supports both discrete and continuous time behaviors. STEAMS enables the modeling of interacting continuous and discrete time components coupled with the modeling productivity benefits associated with object-oriented techniques. This paper presents the requirements and rationale for the STEAMS language development effort, including modeling deficiencies currently facing the VHDL-AMS user community.

Introduction

Listing a complete set of requirements for object-oriented extensions to VHDL, a rationale for each requirement, and discussing its relative priority is not a simple task. Nonetheless, the object-oriented VHDL group needs to understand the tradeoffs and relative strengths of particular language proposals as they pertain to an independent collection of requirements. In other words, these requirements define the problem the extensions intend to address. This document extends the revised requirements document for the object-oriented VHDL study group of the IEEE Design Automation Standards Committee to support object-oriented extensions to VHDL-AMS.

IEEE VHDL-AMS 1076.1-1999 specifies extensions for VHDL-1076 to add support for the description and simulation of both conservative and non-conservative continuous and mixed continuous/discrete systems. VHDL 1076.1 builds on the VHDL semantic and syntactic foundation, including structural and function decomposition, separate compilation, a powerful sequential notation and a type system.

Because VHDL-AMS is based on VHDL, we first examine the requirements for VHDL and then the additional requirements for VHDL-AMS. These requirements are still very relevant to the VHDL-AMS, as it is based on the VHDL language. During the Department of Defense's Very High Speed Integrated Circuit (VHSIC) program, the VHSIC Hardware Description Language (VHDL) development began to address essentially these needs. A perusal of the DoD Requirements for Hardware Description Languages, dated January 1983, provides interesting insight into the problems originally addressed by the VHDL development effort. These requirements from the beginning of the VHDL development can be loosely categorized into three areas: purpose/domain of use requirements, language capability/expressibility requirements, and usability/implementability requirements. The VHDL requirements from this document include:

Language purpose/domain

- Ability to support high level design/top-down design
- Can be processed via design automation tools for
  - Simulation
  - Synthesis
  - Software development
  - Testing
  - Physical design
- Translatable to other HDLs
- Portable to different operating systems and machines
- Upwardly extensible to interface with high level system description languages (SDLs)
- Enhances reliability

- Enhances maintainability

Language capability/expressibility
- Supports hierarchy
- Supports abstraction
- Supports collections of related objects
- Modularity
- Supports libraries
- Strongly typed with type checking
- No implicit type conversion
- Variant types
- Supports encapsulations
- Supports exceptions
- Communications only via I/O lists

Language usability/implementability
- Efficiency
- Simplicity
- Implementability
- Machine independence
- Parallel processing

This list of requirements is quite remarkable; particularly when one considers that many of the current difficulties or deficiencies with VHDL come from inadequate coverage of these requirements. Additional VHDL-AMS requirements were developed for the extension of VHDL to support modeling of analog and mixed-signal systems. The VHDL-AMS design objectives and design rationale detail additional requirements that were the focus of the 1076.1 language development effort. The primary requirements for the VHDL-AMS development include the following:

- Upward compatibility
- Preserve strong typing
- Separate declaration and functionality
- Declaration before use
- Do not infer functionality from a declaration
- Unification of timing semantics
- Preserve determinism
- Preserve generality
- Technology/methodology independence
- Preserve scope of VHDL (gate to system)
- For continuous part of a model, circuit to system level
- Preserve intermixed abstraction levels
- Analog/digital behavioral/structural in same design unit.
- Preserve concurrency
- Preserve and improve consistency
- Re-use as much as possible
- Define semantics and syntax as understandable as possible
- Preserve and improve portability
- Quality of conformance to standard
- No application-specific packages
- Such packages require separate standardization
- Minimize implementation impact
- Maximize implementation efficiency
- Distinguish between the language, modeling and tool issues/aspects
- We are mainly concerned with language issues
- Ease of use without sacrificing correctness and language design principles
- Constructive rather than propositional definitions

A principal area neglected in all of these requirements is in the area of object-oriented design and programming capabilities. In particular, capabilities such as enhanced reuse, extensibility (inheritance), polymorphism, and late or dynamic binding need to be considered for inclusion in OO-VHDL.

How will one use STEAMS?

The question perhaps most important to consider when gathering these requirements is what domains are expected to use and (hopefully) will be helped by object-oriented extensions to VHDL-AMS. Any number of language constructs or capabilities may be novel or perhaps tremendously useful in obscure problem domains, but the general applicability and practicality of the proposed language changes must be addressed. The problem domains in which one will employ an object-oriented VHDL-AMS should be identified and considered to gain insight into the engineering tradeoffs necessary in listing requirements and relative priorities for the eventual language design. Perhaps the most pragmatic concern should be the economic viability of an enhanced VHDL-AMS and the likely consumer demand.

The authors base their assumptions on the requirements and applicability of object-oriented VHDL-AMS on several resources: numerous discussions; a variety of proposals for object-oriented enhancements to VHDL; the requirements developed for VHDL and VHDL-AMS, other hardware description languages, and object-oriented software languages; references on language definition and implementation; and the authors' personal perspectives.

In the authors' experience with design automation tools, methodologies, and designers, by far the most important consideration for a simulation tool is performance, followed by cost. Issues such as language compliance to standards, the quality of the user interface, and ease of use, although important, do not warrant the level of interest given to raw performance. Given this very pragmatic economic consideration, the performance of object-oriented VHDL-AMS tools must be close, if not superior, to the performance of existing VHDL-AMS design automation tools. Similarly, the tardiness in tool vendors providing support for VHDL-93 demonstrates the importance in considering tool implementation issues.

Creating extensions that co-exist with legacy tools and models minimizes the initial cost of adoption for users and tool vendors. Consequently, STEAMS must support existing VHDL/VHDL-AMS infrastructures. This also is an argument for STEAMS extending VHDL-AMS and not removing or changing language features (or warts), no matter how tempting.

For digital modeling, VHDL-AMS will retain similar characteristics in usage as VHDL. With the widespread use of VHDL and synthesis as a productivity enhancement tool, gate level modeling is no longer a task for humans; synthesizers automatically generate these models. As behavioral synthesis capabilities improve, register transfer level models will become increasingly machine generated. Object-oriented extensions to VHDL focused on gate or register transfer level design will add little benefit to machine generated VHDL code or designer productivity. Similarly, device level behavioral modeling with VHDL-AMS will see less benefit than at higher levels of abstraction, particularly as analog and mixed-signal synthesis capabilities improve. In contrast, inserting object-oriented design capabilities into the specification or architectural definition phase of system or hardware design flow promises significant productivity gains. Targeting specification and architectural definition also positions the technology into the area where designers must update their design methodology, hardware or description languages, and design automation tools. Such an approach maximizes impact and likelihood of success.

Assuming the most promising domain for STEAMS is in system specification or design, the need for better abstraction support becomes clear. The use of VHDL-AMS performance modeling illustrates the need for enhanced abstraction, communications mechanisms, and dynamic creation and destruction of tasks. Not only will this better support the abstract modeling of systems with data-less tokens, it will also narrow the semantic gap between VHDL-AMS and common programming languages and practices.

The underlying requirement for better abstract modeling, support of synthesis, and numerous other requirements is to provide better productivity for the designer. Not only should STEAMS help with creating designs, it should also help in the reuse of designs and in the maintenance of products. As hardware design becomes increasingly complex with shortening design cycle times, designer productivity must improve to keep pace. Borrowing the most effective software engineering techniques related to object-oriented design and programming seems promising as a means of providing the needed productivity improvements. Note that this is not the same as supporting the formats and representations used in systems and software engineering, although support for such representations will improve productivity by enabling easier communications between the systems, hardware, and software engineering teams.

When reading about programming language design, one often sees a number of requirements for a high quality language implementation. Among these requirements are consistency, accuracy, simplicity, completeness, efficiency, and being well-defined. Such requirements should be included because it is almost a tautology that good languages attempt to meet these requirements, although they often seem to be conflicting.

This list of requirements helps to delineate what general capabilities the design automation community needs in STEAMS. The list can be loosely grouped into three categories: existing or emerging design automation infrastructure issues, problem domain or advanced design methodology aids, and general language issues. These requirements are not ordered with respect to priority. A more detailed description of each requirement is included in the appendix. Feedback from the design community is welcome to improve this list of requirements.

Conclusions

Increasing demands on designer productivity may best be met by careful inclusion of object-oriented extensions to VHDL-AMS. The development of these extensions must be carefully considered in the context of existing and emerging design methodologies, design automation tools, and emerging problem areas such as systems specification and architectural design. This document considers previous requirements for VHDL and VHDL-AMS, the potential application domain for STEAMS, and general language design considerations to collect a list of requirements.

References:

[1] J.-M. Berge, W. Nebel, and W. Putzke. "Requirements and Design Objectives for an Object-Oriented Extension of VHDL (OO-VHDL)." White paper for IEEE DASC Object-Oriented VHDL Study Group. June 1996.

[2] Gregory D. Peterson, "Proposed Language Requirements for Object-Oriented Extensions to VHDL." White paper for IEEE DASC Object-Oriented VHDL Study Group. June 1999.

[3] C.A.R. Hoare. "Hints on Programming Language Design." Stanford Artificial Intelligence Laboratory Memo AIM-224 (also STAN-CS-73-403). December 1973.

[4] SUAVE: SAVANT and University of Adelaide VHDL Extensions. See http://www.ashenden.com.au/suave.html

[5] *Ada 95 Rationale: The Language and Standard Libraries.* International Standard ANSI/ISO/IEC-8652:1995. January 1995.

[6] Department of Defense Requirements for Hardware Description Languages. January 1983.

[7] *IEEE Std 1076-1987: IEEE Standard VHDL Language Reference Manual.* Institute of Electrical and Electronic Engineers, Inc. New York, NY. 1988.

[8] *IEEE Std 1076-1993: IEEE Standard VHDL Language Reference Manual.* Institute of Electrical and Electronic Engineers, Inc. New York, NY. 1994.

[9] *IEEE Std 1076.1-1999: IEEE Standard VHDL Analog and Mixed-Signal Extensions.* Institute of Electrical and Electronic Engineers, Inc. New York, NY. 1999.

Requirements and Rationale

1. **Compatibility with VHDL-AMS and legacy models**
   To support the installed base of VHDL-AMS users, models, and tools, object-oriented extensions to VHDL-AMS must maintain compatibility with VHDL-AMS and legacy models to ensure its economic viability. With the exception of new reserved words invalidating existing VHDL-AMS models, compatibility with VHDL-AMS should be provided.
2. **Compatibility with systems engineering/software engineering representations**
   The object-oriented extensions to VHDL-AMS should improve the capability to perform abstract modeling, exchange information with systems engineering or software engineering representations, or migrate functionality between domains such as hardware and software. Given the prevalence of object-oriented design techniques for the systems and software engineering disciplines, STEAMS should be designed with the exchange of design data into or from STEAMS in mind.
3. **Support hardware/software co-design and co-verification**
   By extending VHDL-AMS with object-oriented capabilities, hardware models in STEAMS should be more easily understood by hardware or software engineers familiar with object-oriented design and programming. This narrowing of the "semantic gap" between hardware and software representations should enable better support of hardware/software co-design and co-verification. The language design should be made in consideration of the potential for exchanging descriptions or migrating functionality between hardware and software, for providing better support for modeling/executing (co-simulating) the hardware and software components of an electronic system, and for supporting hardware/software design automation.
4. **Support mixed-technology modeling**
   Models in STEAMS should support mixed technology systems, with multiple interacting energy domains supported.
5. **Support extensions for microwave modeling**
   Current VHDL-AMS related standards efforts within the IEEE Design Automation Standards Committee includes a study group addressing potential extensions to support microwave modeling in VHDL-AMS. Any such extensions should be supported by STEAMS.
6. **Support extensions for partial differential equations**
   A requirement that was not included in the original development of VHDL-AMS was to support partial differential equations. Preliminary discussions have addressed the potential of supporting such capabilities within VHDL-AMS. Any such extensions should be supported by STEAMS.
7. **Supports synthesis**
   Give the need for improved designer productivity and current importance of synthesis, the object-oriented extensions to VHDL-AMS should be designed with automatic synthesis in mind. Language constructs that are difficult to synthesize should be avoided; e.g., unlimited recursion should be avoided and static elaboration should be made possible.
8. **Simulatable**
   Models developed in STEAMS should be simulatable, such that designers can exercise the functionality described in the model for verification. The STEAMS code should provide sufficient information to allow verification of the design by simulation or equivalent tools ranging from abstract functional models through register-transfer to gate level or for analog models of transfer functions through behavioral models to device models.
9. **Upwardly extensible**
   The language should be extensible, with the ability to extend the language to cover new concepts, add new language structures, or support the definition of a design "environment" in specific domains.
10. **Ability to support high level design**
    In order to serve as a design tool for a top-down methodology, STEAMS should allow the designer to deal with elements of the design at a very abstract level at the high levels of system description. The designer should be able to defer decisions on implementation to later design stages.
11. **Implementation neutral representations (including HW/SW neutral)**
    Models developed in STEAMS should not be overly constrained with respect to potential implementations. To the extent possible, the language should enable implementation neutral modeling of hardware behavior. Given the potential for migrating functionality between hardware and software, or analog and digital, the language should support modeling functionality in a hardware/software neutral manner.

12. **Support partial definitions and incremental design (polymorphism, dynamic binding, type genericity)**
    In order to serve as a design tool for a top-down methodology, STEAMS should allow the designer to develop abstract models of behavior that can be extended or refined. The language should support abstract modeling by exploiting common object-oriented design and programming capabilities such as polymorphism, dynamic binding, and type genericity to provide better refinement, better reuse, and to avoid over-specification.
13. **Abstraction (of data, concurrency, communications, timing)**
    The object-oriented extensions to VHDL-AMS should support improved abstraction capabilities for modeling. The language should support more abstract notions of data, concurrency, communications, and timing.
14. **Relaxed timing and typing in controlled manner**
    STEAMS should support interface based design by allowing relaxed timing and typing for abstract modeling of communications and performance modeling. Such relaxation should only occur in a controlled manner.
15. **Improved encapsulation**
    STEAMS should support improved encapsulation to limit access to the implementation details of primary units.
16. **Improved information hiding**
    The object-oriented extensions to VHDL-AMS should support improved information hiding to simplify models, to isolate changes made to models, and to ease in model understanding.
17. **Able to specify interfaces as well as objects/entities**
    The object-oriented extensions to VHDL-AMS should support the capability to define the interface between objects/entities as well as the functionality associated with specific objects/entities. The ability to separate and refine interface functionality will improve abstract modeling capabilities and better support interface based design methodologies.
18. **Improved productivity**
    A primary motivation for extending VHDL-AMS with object-oriented design and programming capabilities is to exploit similar productivity advances in software engineering. STEAMS will help improve designer productivity in electronic product design, implementation, integration, verification, maintenance, and/or upgrade.
19. **Support reuse**
    Reuse is a critical means to achieving improved designer productivity. The object-oriented extensions to VHDL-AMS should support the reuse and refining of existing models.
20. **Provide simulatable specification capabilities**
    In order to define a data set package including all the necessary information for maintaining, reusing, or refining a system, the notion of a simulatable or executable specification was defined. STEAMS should support the development of a simulatable specification by supporting the precise and accurate documentation of electronic systems behavior, a testbench representing environmental behavior, and test vectors.
21. **Documentation**
    Models developed in STEAMS should provide good documentation for the operation of the electronic system being designed. Documentation should be considered a critical aspect of the design process and the design language.
22. **Readable**
    Models developed on STEAMS should be easily readable and understandable. The language will encourage and assist the designer to write clear, self-documenting code. The readability of programs is immeasurably more important that their writeability.
23. **Concurrence**
    The object-oriented extensions to VHDL-AMS should maintain the model of concurrence of VHDL-AMS. Given the inherent concurrent operation of physical hardware components such as gates, STEAMS should maintain the capability of VHDL-AMS to model the concurrent behavior of entities.
24. **Exceptions**
    See codesign references here. There shall be an exception handling mechanism for responding to unplanned error situations detected during simulation. The exception situations shall include errors

detected by hardware, software errors detected during simulation, error situations in built-in operations, and user-defined exceptions.

25. **Dynamic process creation and destruction**
    In order to better support modeling testbenches, integration with software, abstract modeling, and reconfigurable computing, STEAMS should support dynamic process creation and destruction. In the case of modeling hardware behavior on reconfigurable computing platforms, designers find VHDL-AMS inadequate for representing the runtime remapping of functions onto FPGA or FPAA hardware because VHDL-AMS processes can not be dynamically created or destroyed. Similarly, abstract modeling and performance modeling of applications such as communications traffic on hardware systems is unwieldy or impractical with VHDL due to the static existential nature of processes. More expressive and powerful testbench creation and better integration with software systems will be enabled by adding the capability to dynamically create and destroy processes within STEAMS.
26. **Accurate models**
    Models written in STEAMS should be precisely, unambiguously described so that the designer can exactly predict the behavior when executed. Nondeterministic or undefined behavior should be disallowed, except as intended by the designer. Side effects should be minimized, implicit type conversions disallowed, and other language capabilities with nonportable or unpredictable effects avoided.
27. **Completeness**
    The object-oriented extensions to VHDL-AMS should allow for the complete description of hardware. To properly document (and potentially transfer) a given design, each of its design entities must be represented by its complete I/O interface and one its alternative descriptions. If any reference is made to a library description, such description is considered part of the design.
28. **Simplicity**
    The language should not contain unnecessary complexity. It should have a consistent semantic structure that minimizes the number of underlying concepts. It should be as small as possible, consistent with the needs of the intended applications. It should have few special cases and should be composed from features that are individually simple in their semantics. The language should have uniform syntactic conventions and should not provide several notations for the same concept. No arbitrary restriction should be imposed on a language feature.
29. **Efficient**
    The STEAMS language should be efficient with respect to its verbosity and its performance, particularly with respect to simulation. The language design should aid the production of efficient behavioral descriptions. Constructs that have unexpectedly expensive implementations should be easily recognizable by translators and by users. Where possible, features should be chosen to have a simple and efficient implementation in any host machines, to avoid execution costs for available generality where it is not needed, to maximize the number of safe optimizations available to translators, and to ensure that unused and constant portions of programs will not add to execution costs. To the extent possible, language constructs should be implemented and tested to ensure their efficiency.
30. **Clean integration of capabilities**
    The language capabilities of STEAMS should be cleanly integrated, with unique language capabilities kept as orthogonal as possible. Special cases and exceptions should be avoided. General approaches and philosophies of VHDL-AMS should be maintained to the extent possible.
31. **Well defined language**
    The object-oriented VHDL-AMS language should be well defined and precise. The language shall have a complete and unambiguous defining document. It should be possible to predict the possible actions of any syntactically correct description from the language definition.
32. **Extensions are unique and consistent**
    The object-oriented extensions to VHDL-AMS should provide one good way to express every operation of interest; it should avoid providing two or more, thus providing unique instantiations of language features. Similarly, consistency should be applied to the extent possible to provide uniform language syntax and semantics. Exceptions and special cases should be avoided to the maximum extent possible.

33. **Portable**
    Because STEAMS will be used as a means of transmitting design data between engineers, models written in the language must be portable. Portability requires that STEAMS data be deliverable in both machine-readable and in textual form. Language features should be avoided which preclude model development, translation, or execution with particular architectures or operating systems.
34. **Translatable**
    No legal restrictions on the STEAMS should preclude its translation into another HDL, including VHDL-AMS. This requirement is not intended to imply a mandatory preprocessing or other translation capability into VHDL-AMS.
35. **Ease of use**
    The OO-VHDL language should be made easy for designers to use in modeling from abstract to detailed models. To the maximum extent possible, design automation processing of OO-VHDL models should be considered.
36. **Easily learned**
    STEAMS should be defined such that it is easily learned by designers, particularly those with some experience with VHDL-AMS and in object-oriented design and programming.
37. **Ease of compilation/synthesis/optimization**
    Models written in STEAMS should be easily compiled, synthesized, optimized. Baroque language constructs which are difficult or impossible for language processing tools to parse, compile, and optimize should be avoided. The language should have a simple, uniform, and easily parsed grammar and lexical structure. The language should be sufficiently expressive such that a simple straightforward translator will produce straightforward translations of acceptable compactness and efficiency. The language should be sufficiently expressive that most other optimizations can be made into the language itself. The language should be so simple, clear, regular, and free from side effects that a general machine-independent optimizer can simply translate an inefficient program into a more efficient one with guaranteed identical effects, and expressed in the same language.
38. **Implementability**
    STEAMS should be implementable at a reasonable cost and level of effort. The language shall be composed from features that are understood and can be implemented. The semantics of each feature should be sufficiently well specified and understandable that it will be possible to predict its interaction with other features. To the extent that is does not interfere with other requirements, the language shall facilitate the production of translators that are easy to implement and are efficient during translation. There shall be no language restrictions that are not enforceable by translators.
39. **Parallelizability**
    In order to support large simulations, and high performance computing which exploits the concurrent nature of VHDL-AMS, STEAMS should avoid constructs which preclude or handicap the potential for parallel processing. Special consideration should be given to the extent of parallel processing supported within the behavioral description since simulation facilities will naturally support the implicit parallelism within structural descriptions. The parallel processing facility shall be designed to minimize simulation time and space. Processes shall have consistent semantics whether implemented on multicomputers, multiprocessors, or with interleaved execution on a single processor.