Analysis of user requirements

Jacques Rouillard

IMT - Technopôle de Château-Gombert
Phone: (33) 91 05 44 44  Fax: (33) 91 05 43 43

Abstract

The Restandardization process has gone through several phases. The first one was the collecting and the analysis of Users' requirements. This paper shows how the decision process worked, both at an administrative and a technical point of view.

The origins of the requirements, the classification of the analysed requirements are shown. The question of analog requirements is explained.

A paradigmatic controversial example is detailed (global variables), and a list of available documents is given.

1- THE DECISION PROCESS

The VHDL 1076 is an IEEE standard. As such, it has been balloted through a formal process, by IEEE members who volunteered to be on the balloting list.

The restandardization process has to follow the same steps as the initial ballot. As it is explained in another paper, the formal decision of making VHDL a standard belongs to the balloting body.

Nevertheless, before the formal process takes place, a great number of technical decisions and design choices must be made. The final decision (the ballot) will be made on the basis of the LRM only, even if some additional documents may be part of the documentation.

These choices were made by various committees and working groups set up by the DASS. The milestones, from a technical point of view, were:

- the collect of the requirements
- the analysis of requirements, leading to the design objectives.
- the language design
- the documentation (writing of the LRM)

- and it is possible that the validation process induces some more decisions.

At the time of the writing of this paper, some activities are scheduled after the validation phase. For example, the Negative Response is the response of the standardization groups to the comments of the balloting body.

This response has to be elaborated after the initial ballot, and this should be the last technical phase of the restandardization.

2 THE TECHNICAL PROCESS

2.1. Context

There were two levels of modification for VHDL. The first one was simple repair (ISAC issues). But many users wanted to modify VHDL, not only to repair it.

A number of papers, books, or tool-specific literatures were related to subsets, supersets, styles, application-specific packages, non-simulation semantics, etc...

Special interest groups or DASS working groups tackle problems of VHDL for synthesis, formal proof, documentation and other application domains.

Also VHDL is competing with other languages. It is quite natural to maintain the language at a competitive level.

Moreover, the VHDL is derived from the Ada programming language, on which a similar process is running (Ada9x) with sometimes similar issues.

2.2. The Criteria

At the very beginning of the process, a set of "guidelines" were decided either by the VASG or by the initial working groups. These guidelines are (with some short comments):
• Upward Compatibility
VHDL87 descriptions should run on a VHDL92 platform. VHDL92 descriptions should produce obvious error messages when running on a VHDL87 platform.
Although this might seem obvious, many formal and informal requests pushed to remove functionalities (e.g., guarded things) or to change default options (e.g., inertial/transport).

• Preserve Strong Typing
This was the answer to requirements such as: allow a BIT to be a BOOLEAN.

• Keep Separation Between Declaration & Implementation
Most objects in VHDL have this separation: unit declarations and bodies, object declarations and use, etc...
This guideline was set against temptatives to declare "special" signals such as clocks, holding a behavior from their declaration.

• Have a Single Timing Semantics
Should VHDL support new application domains, its semantics should be compatible with the "old" one. It is not clear whether or not this statement is consistent, as it is very difficult to map timing semantics of an application domain onto another one.

• Keep Determinism
VHDL87 as used by a normal designer, is deterministic. This was thought to be a good property. However, several points of VHDL87 can be hacked to make a model non-deterministic.

• Keep Generality
VHDL is not tied to a particular style or technology. VHDL opponents say that this means that VHDL should be usable for still unknown domains.

• Target Circuit to System Description
This is the minimum ambition, as it is already the range of VHDL 87.

• Allow Intermixing of Abstractions
E.g., behavior, data-flow & structure in the same architecture. This was to avoid the definition of "special" design units to solve problems raised by analog-related requirements.

• Keep Concurrency
Most of VHDL statements are truly concurrent. VHDL92 should not introduce features implying an order of execution, nor features forbidding true parallelism.

• Keep or Improve Consistency
VHDL aims at being general and consistent. It was felt a good idea to keep this goal in mind.

• Keep or Improve Portability
The main advantage of VHDL is that VHDL is vendor-independent, and targets portability. Again, this was felt important enough to become a guideline.

• Do Not Address Application Specific Packages
E.g., VHDL as a standard should not include packages written in VHDL. This has been the reason for the deferring of many requirements related to synthesis: they could be satisfied by giving a semantic to a VHDL package.
One can note however that a MATH package has been some time a candidate for inclusion within the LRM.

These guidelines are too short to be complete. For example, a strict upward compatibility would forbid the introduction of new keywords.

Some more general and informal guidelines could also be listed here:

• Let VHDL be a language, not a format.
A language holds concepts and needs a background. This has been the rationale for the discontinuation of requirements such as: allow abbreviations in keywords, allow alternate keywords, alternate syntax, etc...

• Keep the current philosophy of VHDL
This was the rationale for other guidelines, e.g., related to strong typing (discontinued requirements such as relax type checking, merge BIT and BOOLEAN, etc...).

• It is too late to make VHDL simple, do not complicate it too much.
This was the rationale for rejecting requirements such as: allow conditional expressions, allow alternate constructs, introduce as an operator that can already be written as an user-defined function, etc..., and for accepting requirements such as: make VHDL more regular (construct bracketing).

The first raw input of requirements has shown a large variety of abstraction levels, ranging from the complete syntactic description of the desired change, to a very abstract request.
2.3. The Origins of the Requirements

The origins of these requirements were twofolds, some of them coming from individuals, some from working groups.

Another input for the working groups came from the ISAC (Issue Screening & Analysis Committee).

The ISAC is made of implementors, and has raised several hundreds of issues, related to ambiguities (eg the "&" between arrays of arrays), errors in the LRM (eg the package TEXTIO) or general concerns (eg the hard-hard and other approaches, the staticity,...).

Some of these issues were resolved by the ISAC at minimal cost, and had to be revisited within the frame of the restandardization.

Some others included basic reflections on the philosophy of the language.

When resolved by the ISAC, some of the issues were presented and approved by the VASG, then included in the SOVASG (Sense of the VASG), companion document of the LRM.

It was decided that all remaining issues should be inserted in the flow of requirements.

2.4. The Classification

The first work of the requirement analysis group was to bring the whole set of requirements to the same abstraction level, then to classify and categorize. Four buckets were used:

- Design Requirements (should be implemented)
- Design Goals (if time permits)
- Study Topics (not implemented unless championed by somebody)
- Discontinued Objectives (not implemented)

The result of this classification can be found in the Design Objective Document.

On certain topics, the working groups had to make tradeoffs and compromises between contradictory requirements.

Some of these requirements were purely discontinued, some others were merged.

A good example is the couple of requirements

"VHDL shall not include synthesis semantics" and "VHDL shall include synthesis semantics".

The frontier between Design Requirement and Design Goal was quite fuzzy and the decision to implement or not was based on time and available manpower. Even the frontier between some Design Requirements and pure miracles was not so clear.

The Language Design phase should have ideally implemented all of Design Requirements. However, some technical problems were identified only at this point.

Moreover, it happened that the language design for a particular requirement fell completely out of the intent of the submitter. The objective had to be partially redrawn.

2.5. The Analog

Many requirements were related to analog modeling, some of them very detailed (and similar).

After many discussions, it was felt that the restandardization process was short of time, and that the whole issue should be postponed to another standardization effort. Since then, a sub-PAR has been launched.

The result of this is that all analog-related requirements are classified as "discontinued", which does not mean, in this particular case, that they will not be implemented. The primary input of the analog sub-PAR will be the list of analog requirements for VHDL 92.

2.6. A Paradigm: The War of Shared (Global) Variables

This has been the paradigm of a hot issue. No stable consensus was ever found to classify this question: it has been some time in the Study-Topics, then promoted to Design Goal, then partially addressed by the language design team, then submitted to a last-chance examination.

The first action was to rename Global into Shared, but this did not prevent the debate to be passionate.

Shared Variables are required by system-level designers because they can get rid of determinism, and rejected by other-levels designers for the same reason!

Moreover, synthesis and formal verification prefer local information to global one.

It violates at least one guideline (Keep Deterministic), but on the other hand, at least one requirement (provide a Random Generator) has been considered mandatory and violates this guideline; at least one existing feature (non commutative/transitive resolution functions) already violates the guideline in VHDL 87, not to speak of files I/O possibilities:

```vhdl
package body P is

procedure set (GV: String ;A : TheType) is
file F: TEXT is out GV;
begin
Write(F,A);
end;

procedure get (GV: String ;A : out TheType) is
file F: TEXT is in GV;
begin
Read(F,A);
end;
```

Moreover, it happened that the language design for a particular requirement fell completely out of the intent of the submitter. The objective had to be partially redrawn.
Technically speaking, there is no real difficulty to introduce shared variables in the language: variables already exist, the keyword exists, the semantics is clear. Even the case of completely parallel implementation can be taken care of.

A number of circles have been made around the question, and were examined:
- Shared Variables with a mechanism to flag determinism breaking.
- Shared Variables without any check.
- Static Memory within the bodies of packages.
- Constant Variables (variables during elaboration, constants afterwards).
- Protected Variables (against a real concurrent access from several processes).

The first level analysis voted in favor of the first solution, but a closer look showed that the real need (if any) was for plain static memory.

A last examination showed that multi-processor implementations could have trouble if the variable was not protected.

This protection does not guarantee the determinism of the simulation, but only the consistency of the variable. However, the protection mechanism looks like very much an operating system primitive, which might be confusing for the average designer.

3 CONCLUSION OF THE ANALYSIS

It may be interesting, as the language design is finishing, to match actual changes against original guidelines.

Actually most of them have been taken care of, with the notable exception of two:

Compatibility

If we except the introduction of new keywords, for which there was a consensus that it should not be considered harmful, VHDL92 is mostly upward compatible.

Nevertheless, for nearly each LCS, we can find some disputable points.

Typically, those clarifying ambiguous statements of the LRM will cause problems to users of non-conformant platforms (eg library STD should contain only STANDARD and TEXTIO).

Also some constraints have been relaxed (exact evaluation at elaboration-time) and pathological models could be non-portable:

constant C : REAL := ((10.0/3.0) * 3.0);
constant B : BOOLEAN := (C=10.0);

C should be exactly 10.0 in conforming VHDL 87 platforms, whereas VHDL 92 allow the result to be 9.9999...

So B could be TRUE in a VHDL 87 compiler, and FALSE in a VHDL 92 one.

Determinism

Obviously it has been broken in several places: the shared variables are the best example, and the reason is that shared variable were strongly supported by system-level designers (eg need for a RANDOM function).

The new OPEN/CLOSE primitives defined on files have the same effect if used by a VHDL hacker.

Actually, the semantics of the example shown above (section 2.6) was left unspecified by the VHDL 87 LRM, and an implementation was allowed to make difficult the use of file I/Os to emulate global variables. The OPEN and CLOSE primitives now make this emulation feasible by the LRM.

4 AVAILABLE DOCUMENTS

Many working documents have been written: minutes of the working groups meetings, intermediate versions of published documents, many emails.

Such documents are not secret nor copyrighted, and can be found from the chairs of the groups, but they are not aimed at being published as they stand.

Some documents related to the VHDL 92 restandardization have been finalized and published at this date:

- The Requirement Document, raw input of requirements plus the ISAC Issues,
- The Design Objective Document, categorized abstractions of the requirements,
- The LCSs, documents specifying Language Changes
- The draft LRM, proposed new definition of VHDL

All these documents are “constructive” documents, and do not put the emphasis on the rationale of the negative choices, even if they make some references to guidelines. They are mainly targeted at being understood by the following design teams and certainly not by the designer.