Doc No: SC22/WG21/N1648
J16/04-0088

Date: 2004-04-09

Project: JTC1.22.32

Reply to: Attila (Farkas) Fehér

Address: LM Ericsson Oy Ab
Hirsalantie 1
Jorvas 02420

Phone: +358 40 507 8729 (mobile)

Fax: +

Email: attila.f.feher@ericsson.com
whitewolf@mailbox.hu

Motivation, Objectives and Design Decisions

1 Summary

Document status: first draft.

One-liner: Suggestions for clarifications on what proposals should contain in order to make decisions in the committee easier. This paper is a rationale for making another one, containing the specific guidelines we finally come up with. *Not a C++ language extension proposal!*

Problems targeted:

- Foreign features and paradigms
- Brave new features
- Motivation disconnected from the C++ language
- Rules of thumb for designing new features for the language

Proposed changes:

- Require C++ related objectives for each non-trivial/non-obvious proposal (where the motivation may become too generic)
- Require certain objectives to be met by each language extension proposal (e.g.: all interactions with the rest of the language are studied and clarified; the solution is the most generic one which solves the motivating issues; the issue fits into one of the decided language extension categories or, it justifies of adding a new one; etc.)

Major open questions:

- None, yet

The motivator for these suggestions is moment of apparent misunderstanding when I have suggested that the objectives are missing from the Design by Contract proposal. It also aims at making the decision process of the committee easier by explicitly stating what objectives has to be met by all proposals.

2 The Problems

2.1 Introduction

Most language extension proposals are aimed at solving a current issue with the language. They present The Problem, or the Motivating Example, introducing the audience into the reasons of why C++ needs to be changed or extended. That is a very clear path for language extension proposals, which are "natural" to C++ or represent an "obvious" evolution path.

However as the news spread that C++ can be extended again, more and more proposals will come. And some of those may not necessarily represent a natural/obvious evolution path or only seem to do so. We are and will be having proposals for adding proven (but new to C++) paradigms and – what I call – brave new features to the language. There will also be proposals, which intend to solve a concrete problem with the language on an attractive and obvious way, but in fact they only solve a specific issue of a more generic problem.

Another issue is that the committee does not do language design. Committee members and experts do. However – at the end of the story – the committee will do language design by rejecting or accepting a feature – by voting on its specific solutions. This work is far from being simple. The committee has to be able to judge the design decisions made in the proposal within a very limited timeframe.

This proposal also aims at easing the decision work of the committee by giving guidelines to both the authors of proposals and the committee on what objectives a proposal has to meet in order to be considered complete. I propose the introduction of objectives in addition to motivation for certain kind of proposals and mandatory objectives to be explicitly met by each proposal.

The committee work can also become guesswork or Q&A, if the motivations/reasons for design decisions are not clearly documented. Objectives can help with this issue (although not fully solve it). I will also propose (as one of the mandatory objectives to be met) that for any non-obvious design decision the decision process (briefly) has to be documented in the proposal or accessible otherwise.

2.2 Motivation vs. objectives

Motivation for foreign/borrowed features or paradigms[1], new paradigms[2] or brave-new-features[3] can easily get insufficient for introducing a C++ language extension proposal. Objectives, on the other hand, are… well, objective and – by the definition of this paper – about C++. Many objectives will be generally applicable for all proposals. Many of them we already have – implicitly. My suggestion is that we turn them into explicit guidelines spelled out in a paper. Later proposals will prove that they meet these objectives and the ones they set for themselves.

2.2.1 Ideas coming from the outside

For features or paradigms coming from other languages (or other external-to-C++ research) the motivation easily slips into justification of the existence of the feature/paradigm. So (to speak in extremes) it will prove that the feature is good, and then say: because it is good, let's add to C++.

[1] Proven features and paradigms coming from another language; proven is the keyword here for this discussion
[2] For example concepts (example for the term, not insufficient motivation part in a proposal)
[3] For example Daveed's compile-time functions for metaprogramming (example for the term, not insufficient motivation part in the proposal)

Such motivations tend to be not purely technical. They contain technical points, but most in it is references to other implementations, statistical data and marketing. They are not bad; they are needed – former implementation experiences, former research data as well as marketing hype – to sell a feature. But they are insufficient for the scientific work of adding a feature to **this** language.

Even when we have a 100% proven foreign feature/paradigm, adding it to C++ is a different issue than proving its right for existence or proving it success in other languages (or even in proprietary C++ extensions used by a small group). We need to know why and how to make C++ to be able to provide that feature/paradigm the best possible way. This is where the objectives come in: telling how this thing will add to C++, what purposes we want to achieve by adding it – as we may choose different priorities than the original language.

2.2.2 Brand new ideas

The two major dangers with brand new ideas are:

- Failure to generalize them
- Failure to naturally integrate them into the whole of the C++ language

The first danger is general to all extension proposals, but it is easier to miss them with brand new ideas and foreign features.

Example:

> Design by contract can be added to C++ by adding aspect oriented programming as a generic paradigm, but this was not considered by the proposal. Most probably since none of the major languages currently implementing the idea went that way. Maybe we should not either, but we should explicitly **decide** about it rather than just take it in.

The second danger is even more subtle. While an idea can be very well done in itself and it even fits its immediate surroundings it may be still vulnerable. If a brand new idea is introduced into the language it is temping to go to brand new ways – including brand new syntax and semantics. While it may be necessary in some cases, it also can introduce hard-to-solve problems later. I do not say we should not introduce new ideas, syntax or semantics. But while we do so, we should be sure it will not stop us from doing extensions or generalization later.

Examples:

> The concept proposal introduces usage patterns. Usage patterns give brand new semantics to C++ expressions. They are not expressions anymore in the old sense, but patterns in an interface definition. Contracts and conversions come naturally for interfaces. The current semantics does not support very well e.g.: tell that a certain concept requires a promise that size() is always smaller or equal to capacity. I am planning to make a proposal which would allow one concept to be “converted” to another implicitly. The semantics of concepts makes it hard for me to come up with a natural syntax for the involved forwarding.

If we put these all together, it seems to be inevitable that we will need to set some standard objectives which encourage all those who propose extensions to avoid the two dangers listed above – if we want to save us from future trouble in extensions as well as from having too specific features.

2.3 Design in committee vs. design by committee

The language is inevitably designed **by** the committee, since it decides what gets into it and what not. But it is not and cannot be designed **in** the committee. Neither the duration nor the resources of the meetings are appropriate for that. Mailing lists may help, but not all can follow them in details and for complex enough proposals they may prove insufficient. So I conclude that the text of the proposals should contain all the possible help for the committee to be able to judge its design decisions.

Explicit and clear objectives can help the committee to understand the design decisions better.

Explicit and clear mandatory objectives can help the committee to judge the completeness of the proposal in regards to the points we decide to be most important to be present. We have those general objectives, but only implicitly. Making these general objectives or guidelines explicit and mandatory for each proposal can help in making them more complete and thereby helping the ease of the decision process of the committee.

For many proposals the motivation or problem statement already includes the proposal specific objectives. I believe that for some I saw, it did not. By making it to be a requirement to explicitly state the objectives makes it harder to miss the target.

The general objectives (guidelines) and the specific objectives of the proposal will serve two purposes. One is that they can be and should be used as arguments in justifying the design decisions taken. The other is that the proposal should **prove** that these objectives have been met!

Example:

> One should either prove that a certain new feature will not break binary compatibility or prove that it is inevitable. Or prove that it will only do so if the feature is used. Or prove that workarounds can be introduced to keep binary compatibility. Basically it has to be proven that binary compatibility can be kept in a reasonable way or point out that it cannot and it is a price to pay for the feature. But not forget to think about it.

Along these lines I can even imagine anti-proposal documents. Assuming that there is a paradigm or feature, which we predict will be requested; a paper can be created (based on the general objectives) to prove that the feature is not to be considered for C++. E.g.: because it cannot blend well with the existing language or because it would break the “zero overhead rule”. Of course, for such a paper to be taken seriously it should list many guidelines would be broken.

2.4 Proposal specific objectives

In case of a “big enough” proposal it may happen that some promises made are not necessarily fully “compatible” with each other, meaning that we would make different design decisions when aiming at one or the other. For example the Design by Contract proposal in its motivation suggests that it will help with the following areas:

- Documentation, communication
    - Suggests to place the rules into the header; otherwise they never get to the users.
- Testing
    - Checking of contracts granularity is per class, compile time; rules not in the header

- Debugging
    - Checking of contracts granularity is per object, runtime; rules anywhere
- Runtime assertions
    - Contracts are always checked; rules anywhere

The proposal has to be able to strike a balance between the possible gains and the gains we really need in C++. So while a “grand proposal” – like the Design by Contract one – may contain a generic discussion of the proposed paradigm in its motivational part, it should contain the specific objectives for the C++ implementation in another part; basically priorities and goals to be achieved in C++.

Motivation will tell **why** it is a good idea to add something. The objectives will tell **how** it will be added to C++ and why that way. E.g.: what possible benefits we want most.

Since the proposal specific objectives are themselves a sort of design decision, the objectives part should contain the justification for choosing those specific goals and priorities.

2.5 Conclusion

When trying to introduce foreign paradigms or features it is easy to miss the point in the motivation. A good paradigm or a good feature in language X is not necessarily good for C++. A good implementation for a feature in language X may not be good enough for C++. The introduction of the proposal-specific objectives and the mandatory requirements stated by the generic objectives (or guidelines) helps to avoid such oversights.

Designing a new feature for C++ is not a simple task. It is easy to miss one or more viewpoints if they are not stated explicitly. Having a list of major (generic) objectives and guidelines can help both the authors of proposals in making good and complete ones, as well as the committee when it has to decide about them. These rules can form a framework for both the proposal creation process and the proposal evaluation process.

Based on the previous discussion it seems to be inevitable to list our main objectives in an official document for new proposals to use them in the described ways. Such document would not be carved into stone, so the committee would be free to accept proposals not meeting all the requirements if it seems to be The Right Way to Go.

Side note:

> Not all authors will be able to meet all the requirements in their first proposal versions. It is possible that a person will have no clear idea about the implementation issues involved with his proposal. In such a case the committee can form an ad-hoc team to help the original author to complete his proposal.

3 Status

The status of this document is first draft. If it passes the test of the committee there has to be a new document created, which lists the major objectives or guidelines, one of them being that the proposals shall contain their own specific objectives.

4 The Proposal

I propose the creation of a publicly accessible official document, which lists the requirements for language extension proposals. The requirements are not to be in the form of process or procedural decisions but rather as a list of objectives.

The paper would contain:

- The foundation guidelines of the language from the Design and Evolution of C++ book, such as the zero overhead rule, and interpretations
- The requirement of clear proposal specific objectives (priorities etc.)
- The list of the objectives of the ongoing language extension process as stated by Bjarne Stroustrup extended by possible other proposed objectives (e.g.: lib rather than core)
    - People will need to fit their proposal into one or more of these categories or justify adding a new one
- The ways to use these guidelines/objectives in the proposal making as well as in the proposal evaluation process
- The generic quality guidelines we come up with. I propose:
    - The proposed extension has to be generalized up to a feasible level (cost vs. benefits).
        - A rule of thumb is a twisted version of an observation from the Cathedral and the Bazaar. The original “Rule #14” sets the standards for “excellent design decisions”: *Any feature should be useful in the expected ways, but a truly great feature lends itself to uses you never expected.* Hint: templates.
        - An example of such an “extending” feature is in N1611, Implicitly-Callable Functions in C++0x by Walter E. Brown.
    - The proposed syntax and semantics has to “blend well with C++” the best natural way possible.
        - We should avoid introducing a new meaning for an existing term or syntax.
        - We should avoid introducing a new term or syntax for existing semantics.
        - We should avoid adding “exceptions” to the language rules
            - Keeping this rule in effect will ease the learning of the language
            - It also provides a safety net to ensure that a later features (affecting this and other features) can be introduced in the same way (e.g.: design by contract in concepts and in classes)
            - Of course, if the feature is just a brand new idea it may not be possible (or even wise) to make it similar to existing ideas, but the proposed syntax and semantics should still be “natural” in C++
    - The rationale (thinking path) for the design decisions (syntax, semantics, dynamics/mechanics) should be in the proposal if feasible or officially accessible otherwise (like in a companion numbered document)

- Examples to make it easier to understand what these guidelines mean/require

Let me stress that I would avoid at all costs having any process-defining wording in that paper. It should serve as a set of guidelines helping our work, not as a binding administrative document.

5 Open questions, tasks

Straw vote to see if the committee likes the idea of having such a set of objectives laid down in an official paper. And if yes …

Collect additional guidelines what we feel to be important enough to state explicitly.

Make the paper:

- List the different guidelines/objectives with a brief description of their application
- Group them if a proper grouping is possible
- Create examples to help to understand what each objective/guideline will look like in use
- List important exceptions and facilitations (e.g.: easy listing of not applicable objectives, proposal specific objectives can be embedded in motivation etc.)