Generic Programming
Projects and Open Problems

David R. Musser[1]
Rensselaer Polytechnic Institute
Troy, New York 12180
musser@cs.rpi.edu

Alexander A. Stepanov
Silicon Graphics Inc.
2011 N. Shoreline Boulevard
Mountain View, CA 94043-1389
stepanov@sgi.com

Last updated:[2] August 25, 1998

[1]This work was performed while the author was on sabbatical at the Wilhelm-Schickard-Institut für Informatik, Universität Tübingen.

[2]The most up to date version of this information will be found at http://www.cs.rpi.edu/~musser/gp/pop/index.html.

At the Dagstuhl Seminar on Generic Programming (Section 4.1) held April 27–May 1, 1998 at the Schloß Dagstuhl, Wadern, Germany, a session was held discussing possible projects, with nominations also for open problems. Dave Musser started the discussion by putting up on the blackboard a tree diagram showing a possible taxonomy and filling in a few leaf nodes with some of his own suggestions for projects and open problems. The discussion was then opened for others to add project and problem suggestions or to add interior nodes to the classification tree. Jim Dehnert added a number of STL-related projects from a lengthy list compiled by Alex Stepanov. Due to time limitations not all of Alex's list was discussed during the session, but the full list is incorporated in various places in the taxonomy here, including some projects that are well underway. (Alex's list was more than a year old.) There was a lively discussion in the session as others contributed projects or project areas.[1]

The current state of this list should not, of course, be considered "complete" in any way. It is being made available on the WWW for the Dagstuhl participants and others to make additions or revisions. It will likely continually grow, but hopefully it can be pruned as projects are completed or open problems are solved! (These can be highlighted or maintained separately as a record of progress.)

Suggestions are also sought for improving the level and type of information provided with projects and problems. For example, should there be a rating system to help readers understand the level of difficulty and/or importance attached to a project or problem by its proposer? Originally Alex Stepanov rated the projects on his list according to both difficulty and importance, each on a scale of 1 to 5. Those ratings are currently not included here, but if there is enough interest they could be added, perhaps after translation to an agreed-up scale.

Revision History

July 3, 1998 First made available on the WWW. Prepared based on the authors' pre-meeting notes and additional notes taken by Dietmar Kühl during the Dagstuhl projects session. (In this initial version, the authors are solely responsible for the introductory remarks and comments on the projects and problems.)

[1] So far we haven't attached names to the project/problem suggestions other than the ones we posed ourselves, because of some uncertainty about who suggested them. If you want your name so attached, please let us know. It would be helpful also to send a paragraph describing the project or clarifying the problem.

Contents

Chapter 1

Theory

1.1 Concept Development

Generic programming can be defined as "programming with concepts," where a concept is defined as a family of abstractions that are all related by a common set of requirements. A large part of the activity of generic programming, particularly in the design of generic software components, consists of concept development—identifying sets of requirements that are general enough to be met by a large family of abstractions but still restrictive enough that programs can be written that work efficiently with all members of the family. The importance of STL lies more in its concepts than in the actual code or the details of its interfaces. Currently the most thorough development and exposition of the STL concepts is the Silicon Graphics Inc. STL web site (Section 4.1), where the STL container and iterator concepts are especially well developed and documented. This web site is an example of what might be called a "concept web (Section 4.1)," because many of the pages contain concept definitions and the most important links between pages are those that express concept refinement.

See also Formalization (Section 1.2), where some of the projects deal with concept development in a more formal setting than concept webs.

1.1.1 Project: Expand the set of concepts used to describe STL

[Musser] Compared to the container and iterator concepts, the STL algorithm, function object, and allocator concepts are less developed in the SGI concept web. The goal of this project is to develop more fully these parts of the concept hierarchy. This is a difficult problem, since the algorithm concept in particular deserves careful development because it is even more fundamental than the container or iterator concepts.

1.1.2 Develop new concepts for extensions to STL

For some extensions to STL that have been proposed, the first major step would be to develop appropriate concepts.

Project: Provide STL with concurrency mechanisms (locking)

[Stepanov]

Project: Develop NUMA-iterator requirements

[Stepanov]

> NUMA iterators are a category of iterators that in addition to random iterator requirements provide an ability to access cache lines. They have an affiliated type `LineIterator` that allows a faster traversal within cache lines and a function `cache_line` that takes an iterator range and returns a range of `LineIterators`—the next cache line to be done. There is a prefetch function defined on `LineIterators`. Implement NUMA-iterator versions of important STL algorithms. Make deque iterators into NUMA-iterators. Implement cache line and prefetch for pointers.

This problem, which Alex Stepanov proposed more than a year ago at SGI, was part of the stimulus for the development of hierarchical iterators (Section 2.1.5).

See also the parallel algorithms project (Section 2.1.5), which is predicated on solution of the NUMA-iterator problem.

1.2 Formalization

1.2.1 Semantics

Project: Formalize the semantics of STL

[Musser] Currently, in both the ANSI/ISO C++ standard and the SGI STL concept web (Section 4.1), the semantics of the STL components is given only informally as a set of requirements stated in English. While there are some devices used that aid in achieving completeness and precision, particularly the use of concept refinement in the SGI concept web, these specifications remain incomplete and somewhat vague, depending heavily on reader's (and library implementor's) prior knowledge and "good intentions." STL presents an opportunity for formalists to try their techniques on a library that is important to a large community of programmers.

Two lectures at the Dagstuhl seminar that addressed this problem were those by Gary Leavens (Section 4.1) and Alexandre Zamulin (Section 4.1).

1.2.2 Performance

Project: Formalize the performance requirements of STL

[Musser] In the case of performance requirements, a degree of precision is achieved in the ANSI/ISO C++ Standard and the SGI STL concept web (Section 4.1) through the use of big-O bounds and bounds on operation counts (although there are several errors in stating bounds on operation counts in the Standard, resulting in unachievable requirements). Overall, more detailed requirements should be given, helping users in choosing the best algorithms and data structures for each particular application. See also Dave Musser's Dagstuhl lecture abstract (Section 4.1).

Open Problem: Develop a better way of expressing performance requirements than O-notation

[Musser] Again, see Dave Musser's Dagstuhl lecture abstract (Section 4.1).

1.3 Methodology

The areas listed in this section are rapidly developing and proving to be useful, but further theory needs to be developed to provide a solid foundation for these activities.

1.3.1 Generic Component Synthesis

1.3.2 Generic Traits Methodology

Chapter 2

Practice

2.1 Library Development

2.1.1 Traits Methodology

2.1.2 Const-Correctness Problem

2.1.3 Generic GUI Libraries

2.1.4 ...

There are potentially many other domains for generic library development, but we didn't try to list them during the session at the Dagstuhl Seminar. Some areas were well-represented in work reported in lectures at the seminar, including computational geometry, computer algebra, image processing, and graph algorithms. Dagstuhl participants and others are encouraged to submit projects and open problems in these and other categories they have particular interest in doing or seeing done.

2.1.5 Component Development within the STL Framework

Project: Domain-specific STL-like libraries

Project: Develop container traits

Project: Regular expression search

Project: Iterators for 2-level structures

[Stepanov] Already well along at SGI. Matt Austern's Dagstuhl lecture (Section 4.1) reported considerable progress. See also NUMA iterators (Section 1.1.2).

Project: Design and implement trie-based associative containers

[Stepanov]

Project: Provide an alternative implementation of sorted-associative containers based on skip-lists

[Stepanov]

Project: Provide an alternative implementation of sorted-associative containers based on vectors

[Stepanov]

Project: Design and implement N-ary trees

[Stepanov]

Project: Design and implement a generic version of the Boyer-Moore searching algorithm

[Stepanov] Dave Musser and Gor Nishanov have essentially solved this problem, with a fast generic sequence searching algorithm obtained by combining the Knuth-Morris-Pratt algorithm with a hashed version of Boyer and Moore's skip loop.

Project: Provide versions of all STL algorithms with $O(N \log N)$ or better worst-case time bounds

[Musser and Stepanov] This project is almost complete, with Musser's *introsort* algorithm replacing the original quicksort implementation of *sort*, and the Musser-Nishanov (Section 2.1.5) algorithm replacing the original straightforward sequence search algorithm. The only remaining algorithm with an $O(N^2)$ worst-case time bound is the Hoare *find* algorithm used to implement *nth_element*. This can be replaced by some variant of Musser's *introselect* algorithm, but some experimentation remains to be done.

Project: Provide forward iterator versions for all of the STL algorithms

[Stepanov] For example, a new implementation of stable_sort should be developed that works on forward iterators and improves its cache behavior by using binary-counter instead of parallel-reduction merging.

Project: Add 3-way comparison versions of all STL components that take comparisons

[Stepanov]

Project: Add iterator adaptors

[Stepanov] Examples:

- restrictors (forward, bidirectional)
- const_iterator (*i = const)
- int_iterator
- stride_iterator
- filtering_iterator
- transform_iterator

Project: Define function_traits and re-implement function objects using them

[Stepanov]

Project: Provide parallel versions of important STL algorithms

[Stepanov] This assumes a solution for the NUMA-iterator problem (Section 1.1.2).

2.2 Language Support

2.2.1 Generic Programming Support

Project: design a language suitable for generic programming

[Stepanov]

2.2.2 Generative Programming Support

2.2.3 Requirement Specification within the Language

2.2.4 Cross-Language Support

2.2.5 Language Extensions to Aid Optimization

Project: Design a set of type/function qualifiers that would allow higher order optimizations

[Stepanov]

An example of such an optimization:

```
bool foo(vector<double>& v) {
        vector<double> u = v;
        return reduce(v.begin(), v.end()) == reduce(u.begin(), u.end());
}
```

This should be optimized to:

```
bool foo(vector<double>& v) { return true; }
```

2.3 Tool Support

2.3.1 Open Problem: How can compiler error messages be improved?

[Musser] Compiler error messages are often terrible when generic programs are involved. This issue was mentioned a number of times during Dagstuhl seminar. It is stated as an open problem because it is not clear how it can be effectively solved without more language support than is provided by C++ and other major languages. Without the ability to state and enforce requirements, even syntactic ones, at template declaration time, it is very difficult detect and properly report errors: An error made by a programmer in using a library component may not be seen by the compiler until it is examining code deep within library components. Practical solutions seem to require a language level above that of current major languages, but still taking advantage of commercial compiler technology by using major language compilers for back-end code production (as in, for example, the SuchThat (Section 4.1) project, which is using C++ as a backend).

2.3.2 Debugging Support

This is the error message problem again, but at run time rather than compile time. Current debugging tools are rarely up to the problems of providing useful information while executing generic software components.

2.3.3 Compiler Optimizations

Code Bloat Reduction

When many different instances of a generic component are required in a single application program, using templated generic components can result in large increases in program code size relative to other approaches (such as dynamic typing and inheritance, which however usually do not have as good run-time performance). Although in many cases different instances could share code, current C++ compilers do not perform much or any code-sharing optimizations. This situation should start to improve as the growing popularity of template-based generic libraries highlights the problem, but research projects in this area could help to achieve improvements more rapidly.

2.3.4 Documentation Support

Project: Develop tools for documenting generic software components via concept webs

[Musser] We are awash in tools for creating web sites, but ones that specifically aid in creating and maintaining concept webs would be especially useful for documenting generic software components.

2.4 Usage Patterns

2.5 Benchmarks

2.5.1 Project: Design and implement an STL-mark

[Stepanov] An "STL-mark" is a benchmark that uses most STL components weighted according to their "practical" importance.

Chapter 3

Education

This topic was brought up during the Dagstuhl session by Arturo Sanchez-Ruiz, specifically mentioning the need to develop tutorials on generic programming. Of course, all of the preceding categories are relevant to education to some degree. Development of concept webs (Section 4.1) of generic programming concepts could be an important aid to educators who want to emphasize a generic programming approach. Links from concept definition pages to related projects would be valuable aids to understanding and motivation for deeper exploration. Also relevant to educational goals are the compiler error message problem (Section 2.3.1) and the debugging support problem (Section 2.3.2).

Chapter 4

Resources

4.1 Related Material on the World Wide Web

- Generic Programming Dagstuhl Seminar, Schloß Dagstuhl, Wadern, Germany, April 27–May 1, 1998.[1]
- Matt Austern, Hierarchical Iterators,[2] lecture at Generic Programming Dagstuhl Seminar.
- David Musser, What Kind of Standards Should There Be for Generic Algorithm Performance?,[3] lecture at Generic Programming Dagstuhl Seminar.
- Gary Leavens, Applying Larch/C++ to the STL,[4] lecture at Generic Programming Dagstuhl Seminar.
- D. R. Musser, Concept Webs.[5]
- D. R. Musser and Gor V. Nishanov, "A Fast Generic Sequence Matching Algorithm," submitted for publication, available online.[6]
- Sibylle Schupp, Generic Programming in SuchThat,[7] lecture at Generic Programming Dagstuhl Seminar.
- Silicon Graphics Inc. Standard Template Library Programmer's Guide.[8]
- Alexandre Zamulin, Language Independent Container Specification,[9] lecture at Generic Programming Dagstuhl Seminar

[1] http://www.cs.rpi.edu/~musser/gp/dagstuhl/gpdag.html
[2] http://www.cs.rpi.edu/~musser/gp/dagstuhl/AusternAbstract.html
[3] http://www.cs.rpi.edu/~musser/gp/dagstuhl/MusserAbstract.html
[4] http://www.cs.rpi.edu/~musser/gp/dagstuhl/LeavensAbstract.html
[5] http://www.cs.rpi.edu/~musser/inform/concept-web.html
[6] http://www.cs.rpi.edu/~musser/gp
[7] http://www.cs.rpi.edu/~musser/gp/dagstuhl/SchuppAbstract.html
[8] http://www.sgi.com/Technology/STL
[9] http://www.cs.rpi.edu/~musser/gp/dagstuhl/ZamulinAbstract.html