X3J16/96-0057
WG21/N0875
March 8th, 1996
John H. Spicer, Edison Design Group
Michael Ball, Sun Microsystems

Status of the Template Compilation Model - Version 2

1. Introduction

1.1 Sponsors

The following people and/or organizations, as a sign of their support for the proposal made in section 3 of this document, have agreed to have their names included as sponsors of this proposal.

German DIN Working Group
Steve Adamczyk, Edison Design Group
Mike Anderson, Edison Design Group
Mike Ball, Sun Microsystems
Michael Eager, Microtec Research
Bill Gibbons
Sam Harbison, Tartan Labs
Roland Hartinger, Siemens Nixdorf
Mark Immel, Centerline
Jason Merrill, Cygnus
Randy Meyers, Digital Equipment Corp.
Clark Nelson, Intel
Mark Pagel, Cray Research
Tom Pennello, MetaWare
Tom Plum, Plum Hall
Larry Podmolik, STR
Atul Saini, Modena Software
Jonathan Schilling, SCO
Ben Schreiber, Microsoft
Anthony Scian, Watcom
Randy Smithey, Rogue Wave
John Spicer, Edison Design Group
Erwin Unruh, Siemens Nixdorf
Jim Welch, Watcom

1.2 Revision History

Version 1 (N0840/96-0022) was distributed in the pre-Santa Cruz mailing.
Version 2 (N0875/96-0057) was distributed at the Stanta Cruz meeting and in the post meeting mailing. The only difference between version 1 and version 2 is the addition of the sponsors list.

1.3 Problems with the Template Separate Compilation Model

The template compilation model was adopted at the Valley Forge meeting in November of 1994. In January of 1995, I submitted a paper (N0640/95-0040) describing the implementation and usage problems with the new compilation model. In July of 1995, I gave a presentation at a technical session on the problems with the compilation model (N0757/95-0157). To summarize the problems that are described in those documents:

1. The new model is not described in the working paper. Adding such a description would be a difficult and time consuming process (in Tokyo, the core-3 group spent several hours discussing aspects of the new model that must be described in the Working Paper, but this did not even give rise to any firm proposals for how the issues should be addressed, and certainly not Working Paper wording).
2. The new model is radically different than existing implementations and it has never been implemented.
3. Instantiations occur in a synthesized context, which makes it much harder for users to understand and correct errors that occur during instantiation.
4. The compilation model requires context merging (which some choose to describe as a name lookup process that considers multiple contexts). Context merging is a complex process that is difficult to specify (see #5 below).
5. The compilation model cannot be implemented efficiently enough to be usable.

The existence of all of these problems has been acknowledged even by those who continue to advocate the new compilation model (although, of course, they may disagree about the severity and importance of the problems), but no significant progress has been made in resolving any of them. In over a year no progress has been made in incorporating a description of the new model in the Working Paper, and no attempt has been made to describe an implementation that would provide acceptable performance.

2. Alternatives

The committee is trying to send out the second CD. For this to be done, all significant changes to the Working Paper must be completed. Certainly, this includes describing the compilation model.
The alternatives for the committee are:

1. Stick with the new compilation model and take the time to work out the remaining issues and describe the model in the Working Paper. Hopefully this process would include producing a document that describes how the performance problems of the new model can be addressed. Based on the rate of progress at the last meeting, it would probably take at least 2-3 meetings to get this work done (with the added cost of not getting other work done).
2. Adopt an alternate model that codifies existing practice, and can be described easily.

I believe the time has come to choose an alternate model.

3. Proposal for an Alternate Model

The template compilation model adopted in Valley Forge was intended to permit two different source organizations: the one in which templates are defined in separate translation units, and the commonly used model in which templates are explicitly included in the translation units in which they are used. This proposal is to retain only the model in which the templates are explicitly included in the translation units in which they are used (sometimes called the “include everything” model). This model is in widespread use. It is easy to understand and use. Although it is sometimes criticized for its cost, it is much less expensive than the separate compilation model, and techniques for optimizing this method are well understood and widely deployed.
The model eliminates the need to do context merging and, as a result, eliminates the need to define and describe the context merging process. Instead of context merging it makes use of textual inclusion, the rules for which are already present in the working paper.

3.1 Working paper changes

Add the following to the end of section 14.4 [temp.explicit]:

> The definition of a template function or template static data member must be present in any translation unit in which it is explicitly instantiated.

Add the following to section 14.3.2 [temp.point], following paragraph 6:

> If the definition of a template function or template static data member that is neither explicitly instantiated nor explicitly specialized is not present in every translation unit in which it is used, the results are undefined. [Note: it is expected that the violation of this rule will result in a program that fails to link on certain implementations]

The existing section 14.3.1 [temp.linkage] should be removed. This section was already superfluous as the linkage of templates was already discussed elsewhere. 14.3.1 currently reads

> A function template has external linkage, as does a static member of a class template. Every function template shall have the same definition in every translation unit in which it appears.

Add to 14.1 [temp.names] paragraph 6 the following:

> A template with external linkage shall have the same definition in every translation unit in which it appears.

4. Option: location of template definitions may be implementation defined

All existing implementations work by textually including the template definitions at some point in the compilation process. Some implementations require that the definitions be explicitly included; other implementations implicitly include a file containing the template definitions, when it is needed. The proposal above could also be amended to permit the implementation to find and textually include a source file in some implementation-defined way. A number of existing implementations use this technique. This permits the implementation to only include the template definitions when they are needed, but introduces some implementation dependencies that would not otherwise exist.

4.1 Working Paper Wording

If this option is chosen, the following wording should be added to section 14.3.2 [temp.point]. following paragraph 6:

> A template definition is “available” if it has either been explicitly provided in the translation unit or if the implementation is able to implicitly textually include a template definition source file that provides the definition. The means of finding such a template definition source file, and the point at which the file is included, are implementation defined

Add the following to the end of section 14.4 [temp.explicit] (instead of the wording in section 3.1):

> If a template function or template static data member is explicitly instantiated, the definition of the template function or static data member must be available in the translation unit.

Add the following to section 14.3.2 [temp.point], following the paragraph added above (instead of the wording in section 3.1):

> If the definition of a template function or template static data member that is neither explicitly instantiated nor explicitly specialized is not available in every translation unit in which it is used, the results are undefined. [Note: it is expected that the violation of this rule will result in a program that fails to link on certain implementations]

5. Conclusion

The models described here have each been implemented by a number of vendors and are in widespread use. Each has proven to provide an efficient means of providing the template instantiations that are needed.

An instantiation model should not be standardized without extensive use by real users. If a vendor finds a superior means of instantiating templates, other vendors will be pressured to follow suit. At that point, such a model would be a candidate for standardization. Until such time, we should not attempt to standardize a model which is unspecified, unproven, and about which there are so many serious concerns.