BESCHWERDEKAMMERN DES EUROPÄISCHEN PATENTAMTS	**BOARDS OF APPEAL OF THE EUROPEAN PATENT OFFICE**	**CHAMBRES DE RECOURS DE L'OFFICE EUROPEEN DES BREVETS**

Internal distribution code:
(A) [ ] Publication in OJ
(B) [ ] To Chairmen and Members
(C) [ ] To Chairmen
(D) [X] No distribution

Datasheet for the decision of 17 September 2010

Case Number:	T 1714/06 - 3.5.01
Application Number:	02002801.5
Publication Number:	1347375
IPC:	G06F 9/44, G06F 9/46
Language of the proceedings:	EN

Title of invention:
Method and apparatus for parallel distributed compilation

Applicant:
SAP AG

Opponent:
-

Headword:
Parallel make/SAP AG

Relevant legal provisions:
EPC Art. 52(1)

Relevant legal provisions (EPC 1973):
EPC Art. 54(1),(2)

Keyword:
"Novelty (all requests): no"

Decisions cited:
-

Catchword:
-

Case Number: T 1714/06 - 3.5.01

DECISION
of the Technical Board of Appeal 3.5.01
of 17 September 2010

Appellant:

SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf (DE)

Representative:

Schiuma, Daniele Wolfgang
Müller-Boré & Partner
Grafinger Strasse 2
81671 München (DE)

Decision under appeal:

Decision of the Examining Division of the European Patent Office posted 21 June 2006 refusing European patent application No. 02002801.5 pursuant to Article 97(1) EPC 1973.

Composition of the Board:

Chairman: S. Wibergh
Members: R. R. K. Zimmermann
G. Weiss

Summary of Facts and Submissions

I. European application number 02 002 801.5, published as EP 1 347 375 and having the filing date 7 February 2002, is directed to a method and apparatus for parallel distributed compilation.

The examining division refused the application. The decision was announced in oral proceedings on 14 October 2005 and posted in writing on 21 June 2006. According to the grounds for the decision, the application did not comply with the requirements of novelty and inventive step in the light of the following publication (document D1):

Charles J. Fleckenstein and David Hemmendinger: "Using a Global Name Space for Parallel Execution of UNIX Tools", in *Communications of the ACM*, ACM Press New York, 32(1989)September, no.9, pages 1085-1090.

II. By letters dated and received 21 August 2006 and 20 October 2006, respectively, the appellant (applicant) lodged an appeal against the decision and filed a statement setting out the grounds of appeal.

III. In a provisional opinion communicated to the appellant under Rule 100(2) EPC, the Board indicated that it concurred with the examining division concerning lack of novelty. In respect of the auxiliary request the Board raised the objection of added subject matter.

IV. By letters dated 18 November 2009 and 5 August 2010, the appellant filed amended claims and made further submissions in support of its case.

V. In oral proceedings taking place before the Board on 17 September 2010, the matter was discussed with the appellant.

VI. The appellant has requested that the decision under appeal should be set aside and a patent be granted on the basis of claims 1 to 8 filed with letter dated 21 August 2006 (main request) or in the alternative on the basis of claims 1 to 6 submitted at the oral proceedings (new main request) or on the basis of claims according to the auxiliary requests 1 to 4 filed with letter dated 5 August 2010. Claim 1 according to each of these requests reads as follows:

Main request:

"1. Computer-implemented method (400) for controlling a building process of a target program (230), the building process with compiling source code modules (211, 212, 213) into object code modules (221, 222, 223) and linking the object code modules (221, 222, 223) to the target program (230),

the method (400) comprising the following steps:

in repetitions for all modules (401), triggering (410, 01, 07, 13) for each module a pseudo-compiler (325) from a predefined scheduler (310) and acknowledging (420, 03, 09, 15) receipt to the scheduler (310), wherein said pseudo-compiler (325) appears to the scheduler (310) as a compiler and operates like a dispatcher that organizes parallel code processing from a serial schedule;

triggering (430, 02, 08, 14) a plurality of compilers (321, 322, 323) from the pseudo-compiler (325) using the serial schedule to operate the compilers to

independently compile (05, 11, 17) the source code modules (211, 212, 213) to the object code modules (221, 222, 223) in parallel;
acknowledging (440, 19, 20, 21) from the compilers (321, 322, 323) to a synchronizer (335); and
triggering (450, 22/23) a linker (330) from the scheduler (310) when the synchronizer (335) has received the acknowledgements (19, 20, 21) from the compilers (321, 322, 321)."

New main request, claim 1 differs from claim 1 above only in the third and fourth paragraphs (differences underlined):
"1. … in repetitions for all modules (401), triggering (410, 01, 07, 13) for each module a pseudo-compiler (325) from a predefined serial scheduler (310) and acknowledging (420, 03, 09, 15) receipt to the scheduler (310) by the pseudo-compiler (325), wherein said pseudo-compiler (325) appears to the scheduler (310) as a compiler and operates like a dispatcher that organizes parallel code processing from a serial schedule without changing said serial schedule;
triggering (430, 02, 08, 14) a plurality of compilers (321, 322, 323) from the pseudo-compiler (325) using the serial schedule wherein the pseudo-compiler (325) buffers trigger commands from the scheduler (317) in a buffer (326) and forwards the buffered commands as predefined to compilers so as to operate the compilers to independently compile (05, 11, 17) the source code modules (211, 212, 213) to the object code modules (221, 222, 223) in parallel; … ."

Auxiliary request 1, claim 1 differs from claim 1 of the main request only in the third paragraph (differences underlined):

"1. … in repetitions for all modules (401), triggering (410, 01, 07, 13) for each module a pseudo-compiler (325) from a predefined scheduler (310) and acknowledging (420, 03, 09, 15) receipt to the scheduler (310) by the pseudo-compiler (325),
wherein said pseudo-compiler (325) appears to the scheduler (310) as a compiler and operates like a dispatcher that organizes parallel code processing from a serial schedule; … ."

Auxiliary request 2, claim 1 differs from claim 1 of the main request only by amendments of the third and last paragraphs (differences underlined):

"1. … in repetitions for all modules (401), triggering (410, 01, 07, 13) for each module a pseudo-compiler (325) from a predefined scheduler (310) and acknowledging (420, 03, 09, 15) receipt to the scheduler (310) by the pseudo-compiler (325), wherein said pseudo-compiler (325) appears to the scheduler (310) as a compiler and operates like a dispatcher that organizes parallel code processing from a serial schedule without changing said serial schedule;

…

triggering (450, 22/23) a linker (330) from the scheduler (310) when the synchronizer (335) has received the acknowledgements (19, 20, 21) from the compilers (321, 322, 321),
wherein said pseudo-compiler (324) and said scheduler (310) are separate service components."

Auxiliary request 3, claim 1 differs from claim 1 of the main request only by amendments of the third paragraph (differences underlined):

"1. … in repetitions for all modules (401), triggering (410, 01, 07,13) for each module a pseudo-compiler (325) from a predefined scheduler (310) and substantially simultaneously acknowledging (420, 03, 09, 15) receipt to the scheduler (310) by the pseudo-compiler (325), so that for said scheduler it appears that compiling has been completed, wherein said pseudo-compiler (325) appears to the scheduler (310) as a compiler and operates like a dispatcher that organizes parallel code processing from a serial schedule; … ."

Auxiliary request 4:

"1. Computer (900) for controlling a building process of a target program (230), wherein source code modules (211, 212, 213) are compiled into object code modules (221, 222, 223) and object code modules (221, 222, 223) are linked to the target program (230), the computer (900) comprising:

a pseudo-compiler (325) that is triggered (01, 07, 13) from a predefined scheduler (310) and that substantially simultaneously acknowledges (03, 09, 15) to the scheduler (310), so that for said scheduler it appears that compiling has been completed, wherein said pseudo-compiler (325) appears to the scheduler (310) as a compiler and operates like a dispatcher that organizes parallel code processing from a serial schedule;

a plurality of compilers (321, 322, 323) triggered (02, 08, 14) from the pseudo-compiler (325) using the serial schedule to operate the compilers to independently

compile (05, 11, 17) the source code modules (211, 212,213) to the object code modules (221, 222, 223) in parallel, the compilers (321, 322, 323) acknowledging (19, 20, 21) to a synchronizer (335); and
a linker (330) triggered (22/23) from the scheduler (310) when the synchronizer (335) has received the acknowledgements (19, 20, 21) from the compilers (321, 322, 321)."

VII. The appellant's arguments in support of the invention may be summarised as follows:

The amendments requested were disclosed in the application as filed. In particular the pseudo-compiler and the scheduler were disclosed as separate service components at p. 2, line 57 and p. 5, lines 47 to 50 and in figure 3 of the application as filed. The auxiliary requests clarified and emphasised that any conventional serial scheduler could be used without change for compiling source code modules in a parallel process by using the pseudo-compiler of the invention as an intermediary between the serial scheduler and a plurality of compilers.

The term "service component" was used in the claims to refer to functions, processes, routines etc in a very general way, independently of any reference to a particular programming language. When the application referred to a computer program to be executed on a computer, this was not a contradiction to claiming the pseudo-compiler and the scheduler as separate components. All service components - the scheduler, the pseudo-compiler, the compiler, the synchroniser, and the linker – should be considered as callable routines performing specific tasks, possibly returning data after execution.

Furthermore, the invention was novel and inventive over the method of document D1. The functionality and structure of controlling the building process was completely different from the prior art method. The invention was distinguished from the prior art at least by the following three conceptual differences: acknowledging receipt by the intermediary pseudo-compiler to the scheduler, the acknowledgements appearing to the scheduler as if the compilations had been completed, and triggering a plurality of compilers from the intermediary pseudo-compiler using the serial scheduler to independently compile the source code modules in parallel to object code modules.

By using a pseudo-compiler which appeared to the scheduler like a compiler it was neither required to change the scheduler nor to know any details about the scheduler. The compiler/linker interface was mimicked by the pseudo-compiler. By means of the pseudo-compiler as a "middleman" between scheduler and compiler/linker, parallelisation could be achieved in any given build-environment and with any normal serial scheduler-compiler/linker combination, without changing the scheduler or the compiler/linker.

The system architecture which was the subject matter of auxiliary request 4 was characterised by structural components, such as the pseudo-compiler, the plurality of compilers and the linker. Document D1 neither disclosed the structure of the system nor the specific interactions between the components.

In particular, acknowledging receipt was an important feature of the invention. The step of acknowledging receipt meant that information was sent and received which indicated that the compilation had been successfully performed. If the compilation failed, no acknowledgement would be passed or a failure notice would be transmitted. The invocation or call of the OUT operation in document D1, and more generally the switching of the thread of execution from the scheduler to the compiler could not be seen as an acknowledgement in terms of the present invention.

The term "triggering" as used in the claims should be understood as a signal or as an event initiating or starting an action or operation of some kind. The invention comprised two distinctive steps of triggering the compilation of a source code module, namely the step of triggering the pseudo-compiler by the scheduler and the step of triggering one of the parallel compilers by the pseudo-compiler for each module to be compiled. Document D1 did not disclose, neither explicitly nor implicitly, the presence of said second triggering step. The IN and OUT operations merely passed commands and distributed work in a convenient way. Only if the OUT and IN operations were both carried out a compilation was performed, i.e. a compiler was triggered. Invoking the OUT operation only triggered the worker process used for compiling, but not the compiling process itself. Such an operation did thus not appear as a conventional compiler process to the scheduler. The examining division, therefore, was wrong to regard the OUT operation as a pseudo-compiler in terms of the present application.

Reasons for the Decision

1. The appeal, although admissible, has to be dismissed since none of the requests justifies the reversal of the decision under appeal. The main request and auxiliary requests 1 to 4 do not comply with the requirement of novelty (Articles 52(1) EPC and 54(1) and (2) EPC 1973). The "new main request" is not admitted to the proceedings for the reasons given further below.

2. *Main request*

2.1 Claim 1 of the main request is not allowable for lack of novelty in respect of prior art document D1. This document discloses a computer-implemented method, "the parallel *make* utility", for controlling a building process of a target program (see D1, p. 1086, right-hand column, line 47, section "PARALLEL MAKE UTILITY"). The building process includes compiling source code modules, viz. the "files to compile" defined by a variable SOURCES, into object code modules, viz. the files compiled, and linking the object code modules to the target program ("the executable image").

A "master process" (see D1, page 1087, right-hand column, lines 6 ff. and the code at lines 20 to 40) in combination with the tuple-space management of the "Linda support environment" (see D1, page 1086, left-hand column, line 25 to p. 1086, right-hand column, line 46) implements all the functions which claim 1 allocates to service components like scheduler and pseudo-compiler.

2.2 Starting with the pseudo-compiler of the present invention it is first to be noted that this expression is used in a non-standard manner. The pseudo-compiler is not intended to, and does not, generate code in any pseudo or intermediate language. In the light of the description, the pseudo-compiler is rather to be understood as a component interfacing with the scheduler like a single compiler (compiler 320 in figure 2, see A1-application, section 0042) but operating like a dispatcher for parallel code processing.

2.3 In a first program block including a while(more_files)-loop (see the code in D1 at page 1087, right-hand column, lines 20 to 40) the master process scans a *make* file that comprises a linear list of file names for compilation (defined by the variable SOURCES, see the *make* file in D1, p. 1087, the paragraph bridging the two columns). This first program block, therefore, operates as a scheduler assigning files for compilation according to a serial schedule.

2.4 For each name found in the *make* file the master process calls an OUT-routine, which distributes the work to be done by "workers" in parallel processing according to a "pool method" (see D1, p. 1087, right-hand column, lines 6 to 18, and p. 1088, left-hand column, lines 11 to 53 with figure 1). Therefore, the OUT-routine in combination with the Linda support environment meets the claim definition of the pseudo-compiler as "a dispatcher that organizes parallel code processing from a serial scheduler".

2.5 The OUT-routine can be said to "appear" to its master process as a compiler. In fact, the *make* utility of document D1 can be used for serial as well as for parallel compilation without changing the master process or the *make* algorithm, namely simply by changing the number of available worker processes (see figure 1 at p. 1088 of D1). Reducing the number to one for the pool method, i.e. by providing a single worker only, results automatically in a serial processing of source files whereas a number of two or more workers provides for parallel processing. The actual compiling mode is not visible to the master process.

2.6 The OUT-routine "acknowledg[es] receipt to the scheduler" by returning program control to the while loop of the master process.

2.7 Furthermore, the OUT-routine (plus Linda) anticipates the features that the pseudo-compiler is triggered by the scheduler and triggers a plurality of compilers. By calling the OUT-routine, the first program block passes control to the OUT-routine and thus can be said to "trigger" the routine.

2.8 Similarly, by creating a work tuple with the name of the file to compile, the OUT-routine (plus Linda) causes the worker processes to compile the files and in this sense "triggers" the plurality of compilers.

2.9 The term triggering as used in the present claims has to be construed in the light of the application. As described with respect to figures 4 and 5 a "trigger" might be buffered and temporarily stored in a queue (see A1-publication, section 0050 ff.). The same holds for the prior art system since the tuple space functions as a buffer for the compilation requests.

2.10 By setting appropriate flags the workers (compilers) inform the master process that the compilation of the respective file has been completed. The workers thus anticipate the step of acknowledging as defined in the penultimate paragraph of claim 1.

2.11 The master process, more precisely the second program block of the master process, functions as a synchroniser for the essentially independent operation of the workers. By means of the while(num_files)-instruction the second program block determines when all files have been compiled and after exiting the while loop executes the exec(link_command)-instruction. This process synchronises the start of the linking operation for all object modules (files compiled).

2.12 It follows that the *make* utility of document D1 anticipates all the definitions of claim 1 and thus destroys the novelty of the invention.

3. *Auxiliary request 1*

The amendment of claim 1 merely clarifies that the step of acknowledging receipt to the scheduler is done by the pseudo-compiler. As already indicated above (see 2.6) this feature is anticipated by document D1.

4. *Auxiliary request 2*

4.1 Auxiliary request 2 adds to claim 1 that "said pseudo-compiler (324) and said scheduler (310) are separate service components". As indicated in the application (see for example sections 0008, 0020, and 0029), the "separate" components may be parts of a single computer program. In this very sense, the different program blocks of the master process in document D1 are also "separate" components so that the new feature does not distinguish the invention over the prior art.

4.2 Furthermore, according to auxiliary request 2, the pseudo-compiler operates like a dispatcher that organises parallel code processing from a serial scheduler "without changing said serial schedule". As already pointed out above, the master process in document D1 works without any change in a serial mode (only a single worker available) as well as in a parallel mode (two or more workers available). In both modes, the master process scans linearly through the *make* file which stores a linear name list of files to compile, which is indistinguishable from a serial schedule. This feature does not contribute anything new to the prior art.

5. *Auxiliary request 3*

5.1 Claim 1 of auxiliary request 3 reformulates the step of acknowledging receipt as follows (amendment underlined): "<u>substantially simultaneously</u> acknowledging receipt to the scheduler by the pseudo-compiler, <u>so that for the scheduler it appears that compiling has been completed</u>".

5.2 The OUT-routine in document D1 passes control back to the master process without waiting for completion of the respective compiling process. Therefore, compared with the time for compilation, the call and the return occur almost simultaneously. The feature of substantially simultaneous acknowledgement of receipt is thus anticipated by the master process.

5.3 As to the second part of the amendment it is necessary to consult the description. The scheduler as disclosed in the application (figure 3 ff. with the corresponding parts of the description) is actually not informed when the compilation is completed, and it does not use such information in any way. Instead, the compilers directly acknowledge compilation to a synchroniser. The only meaningful function of acknowledging receipt is to inform the scheduler that the request (trigger) has been received error-free and that the pseudo-compiler is ready to receive the next request from the scheduler.

5.4 This function, however, does not distinguish the claimed method over the prior art. The OUT-routine plus Linda (the pseudo-compiler) adds a "work tuple" with the name of the file to compile to the tuple space and immediately returns control to the first program block of the master process (the scheduler). The return to the calling program block informs i.e. acknowledges that the request has been received so that requests for further files to compile can be submitted.

6. *Auxiliary request 4*

6.1 The subject matter of claim 1 is a computer for controlling a building process of a target program comprising a pseudo-compiler, a scheduler, a plurality of compilers, synchroniser, and a linker. As follows from the application, the term computer has to be understood as a computer system or as a distributed network of computer systems (see for example A1-publication, section 0012 ff.).

6.2 The appellant argued that the structure and architecture of the computer claimed was not anticipated by document D1. However, claim 1 does not define the computer by structural details but by functional features, and these are fully anticipated. Document D1 discloses the parallel *make* utility as a computer-implemented process and therefore also the functions of the computer system implementing the various steps of the *make* process. Since claim 1 of auxiliary request 4 is merely a functional reformulation of the method of auxiliary request 3 - actually a one-to-one translation of the method steps into functional features - it follows that essentially the same reasons for lack of novelty apply to both requests.

7. *New main request*

7.1 The "new main request" submitted by the appellant during the oral proceedings is not admitted because it is susceptible of new objections at a late stage of the proceedings without prospects of advancing the case towards grants of a patent.

7.2 In fact, the amendment introducing a "predefined serial scheduler" into claim 1 cannot be derived directly and unambiguously from the application as filed. This problem could be circumvented by interpreting the new feature as a simple reference to a predefined serial schedule. Such an interpretation, however, would not help to restore novelty since document D1 already anticipates a serial schedule for use in a parallel make process (the "*make* file", see D1, p. 1087, line 56 ff.).

8. In summary, none of the requests submitted to the Board for consideration form a valid basis for an allowable appeal.

Order

For these reasons it decided that:

The appeal is dismissed.

The Registrar:	The Chairman:
C. Louca-Dreher	S. Wibergh