Internal distribution code:

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

Datasheet for the decision of 19 March 2019

Case Number: T 0518/14 - 3.5.06

Application Number: 05111731.5

Publication Number: 1669854

IPC: G06F9/445

Language of the proceedings: EN

Title of invention:
Inter-process interference elimination

Applicant:
Microsoft Technology Licensing, LLC

Headword:
Constructing a process/MICROSOFT

Relevant legal provisions:
EPC 1973 Art. 84

Keyword:
Claims - clarity (no)

Decisions cited:

Catchword:

Europäisches Patentamt
European Patent Office
Office européen des brevets

Beschwerdekammern

Boards of Appeal

Chambres de recours

Boards of Appeal of the
European Patent Office
Richard-Reitzner-Allee 8
85540 Haar
GERMANY
Tel. +49 (0)89 2399-0
Fax +49 (0)89 2399-4465

Case Number: T 0518/14 - 3.5.06

D E C I S I O N
of Technical Board of Appeal 3.5.06
of 19 March 2019

Appellant: (Applicant)	Microsoft Technology Licensing, LLC One Microsoft Way Redmond, WA 98052 (US)
Representative:	Grünecker Patent- und Rechtsanwälte PartG mbB Leopoldstraße 4 80802 München (DE)
Decision under appeal:	**Decision of the Examining Division of the European Patent Office posted on 30 July 2013 refusing European patent application No. 05111731.5 pursuant to Article 97(2) EPC.**

Composition of the Board:

Chairman	W. Sekretaruk
Members:	M. Müller
	G. Zucka

Summary of Facts and Submissions

I. The appeal is against the decision of the examining division dated 30 July 2013 to refuse European patent application No. 05 111 731 for lack of clarity, Article 84 EPC, and insufficiency of disclosure, Article 83 EPC. Auxiliary requests 3-6 were said to share the clarity problem with the main request and were, "thus", not admitted pursuant to Rule 137(3) EPC.

II. Notice of appeal was filed on 9 September 2013, the appeal fee being paid on the same day. A statement of grounds of appeal was filed on 9 December 2013. The appellant requested that the decision be set aside and that a patent be granted on the basis of claims according to a main or six auxiliary requests, respectively, all as filed with the grounds of appeal.

III. In an annex to a summons to oral proceedings, the board informed the appellant of its preliminary opinion that the claims lacked clarity, Article 84 EPC. Objections under Article 83 and 56 EPC were also raised.

IV. In response to the summons, with letter dated 18 February 2019, the appellant filed amended claims according to the main and the six auxiliary requests.

V. Claim 1 of the main request reads as follows:

"A method of constructing a process (140) from load modules (124) stored in a computer storage, the process to be run under an operating system (110), comprising:
creating (320) a typed-code representation of the process from said load modules, wherein the typed-code representation describes all of the code and types of data objects manipulated by the code expressed in said

load modules, the load modules being expressed in a format which describes the code in instructions and the types of data operated on by those instructions, the typed-code representation further describing which data objects are manipulated by each element of code;

generating (326) a processor-executable instruction stream from the typed-code representation; and

constructing the process, wherein the constructed process comprises the generated processor-executable instruction stream."

Claim 1 of auxiliary request 1 differs from that of the main request in that the generating and constructing steps are replaced by the following text:

"... updating (322) said typed-code representation;

generating (326) a processor-executable instruction stream from the typed-code representation, said processor-executable instruction stream including processor-executable instructions to perform a self-examining function; and

constructing the process, wherein the constructed process comprises the generated processor-executable instruction stream,

wherein the typed-code representation is updated before stream generation to add additional processor-executable instructions to the stream to perform the self-examining function, and

wherein the constructed process is configured to examine properties of itself, the generated processor-executable instruction stream included within the constructed process being unalterable once the process is constructed."

Claim 1 of the second auxiliary request reads as follows:

"A computer system having an operating system (110) comprising a process construction architecture (100) for constructing a process (140) from load modules (124) stored in a computer storage, the process to be run under said operating system (110), comprising:

a typed-code representation creator (152) for creating (320) a typed-code representation of the process from said load modules, wherein the typed-code representation describes all of the code and types of data objects manipulated by the code expressed in said load modules, the load modules being expressed in a format which describes the code in instructions and the types of data operated on by those instructions, the typed-code representation further describing which data objects are manipulated by each element of code;

a typed-code representation updater (154) for updating (322) said typed-code representation; and

a typed-code representation converter (156) for generating (326) a processor-executable instruction stream from the typed-code representation,

wherein the process construction architecture is arranged for constructing the process to comprise the generated processor-executable instruction stream, and wherein the operating system is arranged for fixing the executable code running in the process when the process is constructed, wherein once fixed, the process cannot run new executable code."

Claim 1 of the third auxiliary request reads as follows:

"A method of constructing a process (140) from load modules (124) stored in a computer storage, the process to be run under an operating system (110), comprising:

obtaining a process manifest (200) that provides a complete list of the contents ultimately needed to

construct the yet-to-be-constructed process, said process manifest including content definitions defining load modules and a code generator external to the process manifest;

creating (320) a typed-code representation of the process from a first set of one or more load modules specified by the process manifest, wherein the typed-code representation describes all of the code and types of data objects manipulated by the code expressed in said load modules, the load modules being expressed in a format which describes the code in instructions and the types of data operated on by those instructions, the typed-code representation further describing which data objects are manipulated by each element of code;

updating (322) said typed-code representation via a second set of one or more load modules specified by the process manifest, wherein the updating comprises:

inserting additional checks to contents and size of data structures, to verify the integrity of process execution;

adding new types to the created typed-code representation; or

adding new functions for existing types to the created typed-code representation;

generating (326) a processor-executable instruction stream from the typed-code representation using the code generator defined in the process manifest; and constructing the process, wherein the constructed process comprises the generated processor-executable instruction stream.."

Claim 1 of the fourth auxiliary request differs from that of the third auxiliary request in that the updating step reads as follows:

"... updating (322) said typed-code representation by repeatedly invoking a second set of one or more load modules specified by the process manifest until no further updates are required, wherein the updating comprises: ..."

Claim 1 of the fifth auxiliary request differs from that of the third auxiliary request in that the following optimizing step is inserted before the generating step:

"... optimizing (324) the typed-code representation on a process-global basis, including fixing the set of types and converting all code and all type information into a unified typed-code representation; ..."

Claim 1 of the sixth auxiliary request differs from that of the fifth auxiliary request in that the optimizing step reads as follows:

"... optimizing (324) the typed-code representation, including process-global optimization and cross-process optimization, the cross-process optimization being an optimization across multiple communicating processes and comprising making complementary modifications to the communicating processes such as removing marshaling code for the marshaling and unmarshaling of data objects; ..."

VI. Oral proceedings were held on 19 March 2019. At their end, the chairman announced the decision of the board.

Reasons for the Decision

The invention

1. The application relates to the dynamic construction and optimization of "operating-system processes", which are also simply called "processes". The process construction is carried out by a software component called the "process constructor" comprising several optional modules (see figure 1, no. 100; paragraph 30; all references herein being to the application as originally filed) which operate in sequence (see paragraph 95 and figure 3). For the present case, the following modules are of particular relevance: The "process manifest composer" 150 and the "typed code representation creator" 152 and "updater" 154, and the typed-code representation converter 158.

1.1 Process construction according to the invention starts from a "program manifest", which identifies the program constituent components, so-called "extending components" and external interfaces comprising type information (see figure 1, nos. 142 and 140; figure 2; paragraphs 31 and 32; figure 3, no. 312; paragraph 93). From the program manifest and the "load modules" (e.g. DLLs) "of the named constituent components", a "typed-code representation" is created (see paragraphs 7 and 38 *et seq.*), i.e. a program formulated in an "Intermediate Language (IL) that describes all of the code and types of data objects manipulated by the code" (see paragraph 39). It is stated that the IL is "a constrained format that enables analyses" - which in general are disclosed as being difficult (see paragraph 23). The IL could be MSIL ("Microsoft

Intermediate Language"), but also Java Byte Code or typed assembly language (see paragraph 40).

1.2 Next, the typed-code representation is "updated" (or "extended" or "manipulated") on the basis of the extending components of the program manifest (see paragraph 47 *et seq.*).

1.3 Then the typed-code representation is "optimized" (see paragraph 53 *et seq.*) using for instance "constant propagation, code folding, dead code elimination, function inlining and partial function inlining, partial evaluation, and function specialization" (see paragraphs 55, 59 and 60). The typed-code representation is said to enable "cross-process optimization" (paragraph 58). After the update, additional "process-global" analyses may be carried out, for instance "data flow analysis, abstract interpretation" or "model checking" (see paragraphs 61-62). None of these optimizations or analyses are claimed.

1.4 After that, the typed-code presentation is "converted" into a "processor-executable instruction stream" (see paragraphs 64 *et seq.*), inter-process "interferences" are eliminated (see paragraphs 67 *et seq.* and paragraph 98 *et seq.*), and a "fixed identity" is created for each process as "a function of the typed-code representation contained within the process"; the skilled person would understand this as some sort of hash of the process code; see paragraph 78 *et seq.*).

1.5 Once constructed, it is stated that the process is "unalterable" (see paragraph 15) after having been "fixed" or "sealed" (see paragraphs 26 and 90). It is stated that "By sealing the process at creation time, the operating system can provide the process a high

degree of confidence that it can hide sensitive information [...] from hostile parties" (see again paragraph 90).

Main request

2. Claim 1 does not define format or content of the load modules or the process instruction stream, even though both must be assumed to contain some sort of object code. It specifies the load modules to be "expressed in a format which describes the code in instructions and the types of data operated on by those instructions". From the load modules, a "typed-code representation of the process" is created which "describes all of the code and types of data objects manipulated by the code expressed in said load modules".

2.1 Claim 1 does not define how the typed-code representation is created from the load modules.

2.2 On the one hand, claim 1 uses a different term than "typed-code representation" for the content of the load modules ("a format which") and slightly different words for explaining both: The typed-code representation is said to describe "all of the code and types of data objects manipulated by the code expressed", the load module format to describe "the code in instructions and the types of data operated on by those instructions" (emphasis by the board). In the board's judgment, the skilled person would interpret this difference in language as a suggestion that the formats in question are different and that the creating step goes beyond merely combining the content of the load modules.

2.3 On the other hand, the claims do not expressly state whether the "typed-code representation of the process" is any different from the combination of the content of the load modules, let alone how. The difference in terminology aside, the claim language does not exclude the possibility that the load modules already contain a "typed-code representation" of a part of the process.

2.4 The creation of the typed-code representation of the process - and the typed-code representation itself - are central features of claim 1 of the main request, which cover half of the claim language. The fact that claim 1 leaves open whether the creating step differs from a plain combination (e.g. concatenation) of the load module content - and, if so, in what way - renders claim 1 unclear, Article 84 EPC 1973.

2.5 It adds to that lack of clarity that the claim leaves open whether the type information in the typed-code representation is used for generating the processor-executable instruction stream and in what way.

3. The appellant argued that the board's objections do not establish a lack of clarity but merely indicate that the claim language is very broad.

4. The board however opines that a claim is unclear if it cannot be determined whether one of its central features limits the scope of the claim - and how - or is, on a broad reading, virtually redundant.

5. The board therefore concludes that claim 1 of the main request lacks clarity, Article 84 EPC 1973.

Auxiliary requests

6. Claim 1 of the auxiliary requests shares the step of creating a typed-code representation with claim 1 of the main request.

6.1 It adds further steps of modifying the typed-code representation by "updating" it by adding instructions for performing "self-examining functions" or for verifying the "integrity of process execution" or of new types or new functions for existing types. None of these additions can overcome the mentioned lack of clarity of claim 1 of the main request, as none defines the typed-code representation as opposed to the format of the load modules, or its creation or its use in generating the processor-executable instruction stream.

6.2 The board therefore considers that also claim 1 of the auxiliary requests lacks clarity for the same reason as claim 1 of the main request, Article 84 EPC 1973.

General remark

7. In the annex to its summons, the board raised additional clarity objections against a number of other features of the claimed invention, such as "self-examining function" (auxiliary request 1), "unalterable" (auxiliary request 1), "fixing" and being unable to run ("cannot run", auxiliary request 2) or "process manifest" (auxiliary request 3). During the oral proceedings, it was discussed how these features - and the ones added to claim 1 of auxiliary requests 3-6 - had to be construed and whether they had to be considered unclear themselves. However, in view of the above, it need not be decided whether these features - or which of them - are unclear, too, or merely broad as

the appellant argued. It need also not be decided whether, as also suggested in the summons, the claims on the broadest possible interpretation (which might find, for instance, the creating step to be essentially non-limiting) would lack inventive step over common knowledge in art on code generation.

Order

For these reasons it is decided that:

The appeal is dismissed.

The Registrar: The Chairman:

M. Schalow W. Sekretaruk

Decision electronically authenticated