**Internal distribution code:**

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

# Datasheet for the decision of 1 July 2015

**Case Number:** T 1211/12 - 3.5.06

**Application Number:** 05106321.2

**Publication Number:** 1615129

**IPC:** G06F9/52, G06F9/54, G06F9/46, G06F9/44

**Language of the proceedings:** EN

**Title of invention:**
Implementation of concurrent programs in object-oriented languages

**Applicant:**
Microsoft Technology Licensing, LLC

**Headword:**
Compiling concurrent object-oriented programs/MICROSOFT

**Relevant legal provisions:**
EPC 1973 Art. 83, 84

**Keyword:**
Claims - clarity (no)
Sufficiency of disclosure - (no)

**Decisions cited:**

**Catchword:**

Europäisches Patentamt
European Patent Office
Office européen des brevets

**Beschwerdekammern**
**Boards of Appeal**
**Chambres de recours**

European Patent Office
D-80298 MUNICH
GERMANY
Tel. +49 (0) 89 2399-0
Fax +49 (0) 89 2399-4465

**Case Number: T 1211/12 - 3.5.06**

# D E C I S I O N
# of Technical Board of Appeal 3.5.06
# of 1 July 2015

| | |
|---|---|
| **Appellant:** (Applicant) | Microsoft Technology Licensing, LLC<br>One Microsoft Way<br>Redmond, WA 98052 (US) |
| **Representative:** | Grünecker Patent- und Rechtsanwälte<br>PartG mbB<br>Leopoldstraße 4<br>80802 München (DE) |
| **Decision under appeal:** | **Decision of the Examining Division of the European Patent Office posted on 21 December 2011 refusing European patent application No. 05106321.2 pursuant to Article 97(2) EPC.** |

**Composition of the Board:**

| | |
|---|---|
| **Chairman** | W. Sekretaruk |
| **Members:** | M. Müller |
| | S. Krischer |

## Summary of Facts and Submissions

I. The appeal lies against the decision of the examining division, with written reasons dispatched on 21 December 2011, to refuse European patent application No. 05 106 321.2 for lack of compliance with Articles 83, 84 and 123(2) EPC. The decision also contains a section entitled "Obiter Dictum" according to which claim 1 of the three pending requests lacks an inventive step over the cited prior art, Article 56 EPC.

II. An appeal was lodged on 29 February 2012 and the appeal fee was paid on the same day. A statement of grounds of appeal was received on 2 May 2012. It was requested that the decision under appeal be set aside and that a patent be granted based on the main or either of the auxiliary requests which were the subject of the decision.

III. With its summons to oral proceedings, the board informed the appellant that, in its preliminary opinion, all three requests did not conform to Article 123(2) EPC and Articles 83 and 84 EPC 1973. Preliminary observations on inventive step were also made.

IV. In response to the summons, with letter dated 1 June 2015, the appellant filed new auxiliary requests 3-5 each of which was an amended version of the pending main and auxiliary requests 1 and 2, respectively, and was intended to address the board's objections under Article 84 EPC 1973 and Article 123(2) EPC.

V. Oral proceedings were held on 1 July 2015 as scheduled. During the oral proceedings the appellant withdrew the main request and auxiliary requests 1 and 2, which made the auxiliary request 3 its main request, and requested

the grant of a patent based on claims 1-13 according to one of the auxiliary requests labelled "3", "4" or "5" as filed with the letter dated 1 June 2015, in combination with the description and the drawings on file.

VI. Claim 1 of auxiliary request "3" reads as follows:

"A computer system employing language extensions in an object-oriented environment (100) that supports concurrency to [sic] object-oriented languages, via message passing to/from services (108) in the object-oriented environment (100), wherein a service executes its own algorithmic thread and does not share state with any code outside the service the system comprising:

a contract component (104) being defined as interface declarations for asynchronous message passing for managing communication between multiple services (108) simultaneously, the contract component including:

a message component (302) specifying a set of messages; and

a protocol component (304) specifying allowable sequences of message exchange;

a component (106) for the multiple services configured to facilitate the handling of multiple messages and multiple message targets, the component comprising:

a compiler component (604) configured to generate a schedule for the messages in accordance with the set of messages and the allowable sequences, wherein the schedule is a runtime object that allows message-oriented code to wait for more than one message in parallel."

VII. Claim 1 of auxiliary request "4" reads as follows:

"A computer system employing language extensions in an object-oriented environment (100) for implementing a service concurrency model for services (108) in an object-oriented environment via message passing to/from services (108) in the object-oriented environment (100), wherein a service executes its own algorithmic thread and does not share state with any code outside the service the system comprising:

a plurality of services (108) in an object-oriented environment (100);

a contract component (104) being defined as interface declarations for asynchronous message passing for managing communication between multiple services (108) simultaneously, the contract component including:

a message component (302) specifying a set of messages; and

a protocol component (304) specifying allowable sequences of message exchange;

whereby said set of messages is usable as an alphabet of a pattern to establish a formal protocol definition of a communication protocol;

a component (106) for the multiple services configured to facilitate the handling of multiple messages and multiple message targets, the component comprising:

a compiler component (604) configured to implement a compilation algorithm to break the source code of the services into pieces to allow parallel waits to occur, wherein places in said source code are broken that could block a current thread of the services thereby allowing multiple points to wait in parallel or allowing the threads context to continue with different computation; and

the compiler component (604) being further configured to generate a schedule for the messages in accordance with the set of messages and the allowable

sequences, wherein the schedule is a runtime object that allows message-oriented code to wait for more than one message in parallel."

VIII. Claim 1 of auxiliary request "5" differs from claim 1 of auxiliary request "4" in that the protocol component is defined as

"... a protocol component (304) identifying an implementation schedule for the set of messages by specifying allowable sequences if message exchange ..."

and in that the second paragraph in the definition of the compiler component refers to the "implementation schedule" and now reads as follows:

"... the compiler component (604) being further configured to generate a schedule for the messages in accordance with the set of messages and the implementation schedule of the contract component, wherein the schedule is a runtime object that allows message-oriented code to wait for more than one message in parallel."

IX. All requests also contain an independent method claim 6 corresponding to the respective independent system claim 1.

X. At the end of the oral proceedings, the chairman announced the decision of the board.

## Reasons for the Decision

*The invention*

1. According to its initial paragraph, the application relates to "language extensions in an object oriented environment to support asynchronous programming through message passing, contracts, and orchestration". The application explains that shared-memory communication as used in certain object-oriented frameworks is "one of the main obstacles to simple support for concurrency" (p. 1, last para. - p. 2, 1st para.) and proposes as a solution to incorporate asynchronous message passing in such object-oriented environment (p. 2, 2nd para.).

1.1 The proposed solution is based on several "services", each running in its own thread, which communicate with each other according to message-based interfaces based on "contracts" (see *e.g.* p. 3, lines 19-20). A contract is defined as "a formal specification of the allowable sequences of invocation of the members of an interface" (*i.e.* of a protocol and its messages; see p. 3, lines 25-26, and p. 26, lines 20-21). Contract declarations in the program code (see *e.g.* p. 10, lines 7-20) are, at run-time, represented by non-deterministic finite state machines (p. 4, 1st para.).

1.2 The contracts are said to be enforced at run-time by "validat[ing] method invocations ... against a contract specification" (p. 3, line 28 - p. 4, line 3) – presumably producing some sort of error message if a method call cannot be so validated. Alternatively, it is disclosed that contracts may be enforced at compile-time (see p. 28, lines 4-5, and p. 2, lines 23-24).

1.3 Another aspect of the invention is referred to in the description as "orchestration", which is said to encompass "the collection of mechanisms for coordinating communication between concurrent services" (p. 9, lines 5-7; p. 37 f.). An "orchestration component" is disclosed comprising what is called a "schedule component" and a "compiler component" (see fig. 6). The compiler is said to "[break] the co-routine-based code down into pieces to allow parallel waits to occur without blocking thread context" (p. 5, lines 5-7). Elsewhere, the compiler is disclosed as producing the schedule component (see original claim 2) as a "runtime object" - with no "source-code manifestation" - that is used to allow message-oriented code to wait for more than one message in parallel" (p. 37, last para. - p. 38, line 2; p. 40, lines 23-24).

1.4 In an appendix, the application contains a scientific paper which appears to provide a theoretical basis for aspects of the present application. The paper refers to model checking to verify that a "message-passing program" conforms to its "contract" and is thus "stuck-free", in that it "cannot deadlock waiting for messages that are never sent or send messages that are never received" (see section 1, lines 1-4). Contracts appear to correspond to the ones described in the application (see Appendix A, fig. 1). The appendix also reports on a study in which a service implementation was automatically detected not to conform to its contract specification in various ways (penult. page).

*Clarity, Article 84 EPC 1973*

2. The board is of the opinion that the wording of claim 1 of all requests is fundamentally unclear, in particular as regards the nature and functionality of the claimed

"schedule" and how the schedule is generated by the claimed "compiler".

3. The board first notes that the term "**schedule**" has no unique established meaning in the relevant art of concurrent and distributed programming languages, environments and compilers. This also applies to the term "**implementation schedule**" as used in auxiliary request 5.

3.1 The "schedule" is claimed as being a "runtime object" (auxiliary request 3, claim 1, penult. line) and disclosed as having no "source-code manifestation" (p. 37, last para.). It is claimed as being generated by a compiler "in accordance with" the contract component, in particular with "the set of messages and the allowable sequences". This language suggests, in conformity with the description, that the schedule is a compiler product meant to aid "enforcement" or "validation" of contracts (see p. 3, 3rd line from the bottom - p. 4, line 3).

3.2 The claims do not specify the desired behaviour of the schedule component at run-time, nor how it is intended to represent, enforce or validate the contract component.

3.2.1 It is noted that a "contract" may be breached by

a) the fact that a message is sent which is not "allowed" at a certain point in time, after a certain sequence of messages, or by the fact that
b) none of the possible messages allowed (and expected) at a particular point in time arrive.

3.2.2 Accordingly, "enforcing" or "validating" a contract can mean different things.

3.2.3 An obvious breach of contract is the sending of a message which is not allowed at a particular point in time. If a service receives such a message, an error message might be produced or an interrupt raised. The service might then process the message nonetheless (if it provides the corresponding method) or it may refuse to process it. In the latter case the sender of the message might be blocked ("stuck") waiting for a response, even though the deviation from the contract was detected.

3.2.4 Another possible breach of contract is that an expected or prescribed message is not received by a service so that the service is blocked waiting for that message, whether or not this is a response message. In this case, it must be defined when this non-compliance is said to occur. The description states that even a long delay of an expected message does not "technically" constitute a breach of contract (p. 64, lines 7-9) but that missing messages constitute a violation of the contract only when the entire schedule is terminated. The board notes that some services are meant to run forever. On the assumption that the "schedule" of such a service never terminates, such a service will, from this perspective, never exhibit a breach of contract even when it is blocked ("stuck") waiting for a message that never arrives.

3.3 Moreover, it may be possible that the compliance of services with a contract can be determined at compile-time (see description, p. 28, lines 4-5). The board notes that this may not be generally possible (for fundamental reasons relating to what is known as the "halting problem") and that the description gives no indication as to the circumstances under which compile-time validation is possible and/or meant to be

performed. Moreover, everything that has been checked at compile-time need not be checked at run-time any more. The skilled person would thus assume that the runtime object "schedule" only performs checks which have not already been carried out at compile-time.

3.4 Claim 1 of all requests does not specify any of the above, be it directly as claimed properties of the schedule or indirectly by way of claimed properties of the "compiler component". As a consequence, it is entirely unclear what the schedule component is meant to do at run-time and thus in what way it is "generated [...] in accordance with the set of messages and the allowable sequences" or, as auxiliary request 5 puts it, "in accordance with the set of message and implementation schedule of the contract component".

4. Claim 1 of all requests specifies that "the schedule component [...] allows message-oriented **code to wait for more than one message in parallel**".

4.1 The board considers that this phrase is ambiguous. Firstly, code comprising multiple threads will typically have, at any point in time, some threads which are ready to proceed and other threads which block while waiting for some message to arrive. The latter threads wait "in parallel". Secondly, any concurrent object must be prepared to process any message corresponding to its methods. Hence, any idle concurrent object providing at least two methods can be said to be waiting "in parallel". Thirdly, according to the claims it is the "schedule", generated "in accordance with" the contract component, which "allows [...] code to wait [...] in parallel"; this appears to suggest that parallel waiting is only allowed to the

extent that the protocol part of the contract component specifies it.

4.2 The wording of claim 1 of all requests leaves open which of the three interpretations is the intended one, and this also renders the claim unclear.

4.3 The board further notes that the inventive contribution which the "parallel wait" feature may possibly make depends significantly on which of these three interpretations is chosen. As suggested above, the possibility of threads waiting in parallel appears to be implicit in any multi-threaded message-based system, the possibility of objects waiting in parallel appears to be implicit in any concurrent object-oriented system, and the possibility for a service to wait in parallel for only those messages which the contract component happens to allow appears to follow from the fact that the specified contract should be "enforced" at run-time. While in the first two cases no dedicated compiler support might be needed at all to "allow [...] waiting in parallel", in the third case such compiler support is needed but not claimed.

5. Auxiliary requests 4 and 5 (penult. para.) refer to a **"compilation algorithm to break the source code** of the services **into pieces** to allow parallel waits to occur, wherein places in said source code are broken that could block a current thread of the services thereby allowing multiple points to wait in parallel or allowing the thread context to continue with different computation".

5.1 This wording leaves open how the "pieces" into which the code is broken are to be determined, whether all places that could potentially block a thread are determined, or only some of them and, in this case,

which ones, how the pieces are to be executed separately, and how they are to cooperate to produce the overall service.

5.2 The board notes in this regard that it is by no means a trivial task to compile sequential program code so that its execution can exploit co-routines, concurrency, or even parallelism.

5.3 In view of this, the lack of any detail regarding the compiler and the compilation algorithm concerns a central feature of the claimed invention which also renders the claims unclear.

6. In its written reply dated 1 June 2015 to the board's summons to oral proceedings, the appellant did not substantially address the board's clarity objections relating to the "schedule", the "compiler", and their relation as claimed (see points 6.6-6.8 of the summons). With regard to the board's objection under Article 83 EPC 1973, the appellant only referred to its submissions in the statement of grounds of appeal which, however, do not explain how the claimed compiler was meant to work.

6.1 Nor did the appellant refer in the oral proceedings to any passages in the description which could elucidate the above clarity and insufficiency problems addressed by the board.

6.2 The appellant argued, however, that the "contract component" or how the compiler generated the schedule from the contract component was not central to the invention and that the contract component could even be deleted from the claims without affecting their inventive merit. Rather, the central contribution of

the invention was that the schedule is generated automatically for the services by the compiler, whereas in the prior art it was necessary for the services to provide and implement their schedules individually and separately.

6.3 The board is of the opinion that without the reference to the contract component "in accordance with" which the schedule is generated, the meaning of the term "schedule" becomes even less clear.

6.4 Moreover, the board considers that the appellant's argument, even if was correct that the central contribution of the invention concerned the way in which a schedule was generated rather than what it did, might be relevant for the inventive step assessment. However, it does not overcome the clarity objection. In the board's view, the appellant's argument even underlines the relevance of the clarity of the term "schedule", since the argument that the invention contributes a new way to provide a "schedule" for services can only be assessed at all if the term "schedule" is clear.

7. In summary, the board concludes that claim 1 of all requests is unclear at least because claim 1 leaves undefined what the "schedule" is meant to do at run-time, how it relates to the "contract component" and how the "compiler" is instrumental in generating the schedule. Therefore, it does not comply with Article 84 EPC 1973.

*Article 83 EPC 1973*

8. The board not only considers the claimed compiler to be unclear, but it is also unable to find in the descrip-

tion any substantial disclosure of how the compiler is meant to work, in particular in view of its opinion that the claimed compiler function is all but trivial even for a person of skill in the art. As already mentioned, in its letter of 1 June 2015 the appellant did not address the specific objection of the board in this regard. Nor did it refer, in oral proceedings, to passages in the description that it considered to provide the pertinent disclosure. Therefore, the board has no reason to deviate from the opinion that it expressed in the summons that the claimed compiler component is not disclosed in a manner sufficiently clear and complete for it to be carried out by a person skilled in the art. Hence, all three requests do not to conform with Article 83 EPC 1973 either.

*Summary*

9. Given their deficiencies under Articles 83 and 84 EPC 1973 as explained above, none of the three requests is allowable. The appeal must therefore be dismissed.

## Order

## For these reasons it is decided that:

The appeal is dismissed.

The Registrar: The Chairman:

D. Hampe W. Sekretaruk

Decision electronically authenticated