Exhibit 20

Class	Subclass	ISSUE CLASSIFICATION
395	700	

[redacted]

5519867

UTILITY SERIAL NUMBER	PATENT DATE	PATENT NUMBER
08/094673	MAY 21 1996	5519867

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT
08/094,673	07/19/93	395	700	2316

CHRISTOPHER P. MOELLER, LOS ALTOS, CA; EUGENIE L. BOLTON, SUNNYVALE, CA; DANIEL F. CHERNIKOFF, PALO ALTO, CA; RUSSELL T. NAKANO, SUNNYVALE, CA.

CONTINUING DATA********* VERIFIED

N/A

FOREIGN/PCT APPLICATIONS******** VERIFIED

N/A

NOTE-DISCLAIMER
The term of this patent shall not extend beyond the expiration date of Pat. No. 5,179,702

FOREIGN FILING LICENSE GRANTED 08/23/93

Foreign priority claimed 35 USC 119 conditions met	Verified and Acknowledged	Examiner's Initials	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☒ no ☐ yes ☐ no				CA	17	53	7	$1,732.00	P058

KEITH STEPHENS
TALIGENT, INC.
~~10355~~ 10455 N. DEANZA BLVD.
CUPERTINO, CA 95014-2000

OBJECT-ORIENTED MULTITASKING SYSTEM

ISSUE FEE IN FILE

U.S. DEPT. of COMM.-Pat. & TM Office — PTO-436L (rev. 10-78)

PARTS OF APPLICATION FILED SEPARATELY

Applications Examiner

NOTICE OF ALLOWANCE MAILED	Assistant Examiner	CLAIMS ALLOWED	
1-16-96	John A. Chiro	Total Claims: 53	Print Claim: 1

ISSUE FEE		DRAWING		
Amount Due	Date Paid	Sheets Drwg.	Figs. Drwg.	Print Fig.
1250.00	2-5-96	17	17	14

SUPERVISORY PATENT EXAMINER
ART UNIT

Primary Examiner

ISSUE BATCH NUMBER: 563

PREPARED FOR ISSUE

Form PTO-1267 (Rev. 11-92) U.S. Department of Commerce Patent and Trademark Office

DISCLAIMER LABEL

Application No. 08/094,673

A terminal disclaimer has been entered and recorded under 35 U.S.C. 253 in this file.

DO NOT DESTROY

Form PTO-436A (rev. 9/92)

867FH001

6/a Williams 11-1-95

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventors:	Moeller, et. el.
Title:	OBJECT-ORIENTED MULTITASKING SYSTEM
Serial No.:	08/094,673
Filed:	07/19/93
Examiner:	J. Chavis
Art Unit:	2316
Taligent #:	P-58
Date:	October 19, 1995

CERTIFICATE OF TRANSMISSION VIA FACSIMILE

I hereby certify that the following Amendment is being transmitted via facsimile addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231 on October 19, 1995.

AMENDMENT

Greetings:

In response to the Office Action dated 8/8/95, please amend the application as follows:

In the Specification:

Please make the following modifications:

Page 8, line 1, replace "210" with --112--.

Page 17, line13, add a space prior to the italicized "scheduling".

In the Claims:

A

1. (Amended) An apparatus for enabling an object-oriented application, said application including object-oriented statements, to access in an object-oriented manner a procedural operating system by use of said object-oriented statements, said system providing services, including procedural functions saved as executable program logic that are called to access said services [having a native procedural interface], [the]said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer;

(c) a code library, stored in said[the] memory component, comprising means for storing said executable program logic in an object-oriented class library; and means for interfacing said object-oriented application to said procedural operating system utilizing said executable program logic; [computer program logic implementing an object-oriented class library, the object-oriented class library comprising related object-oriented classes for enabling the application to access in an object-oriented manner services provided by the operating system, the object-oriented classes comprising methods for accessing the operating system services using procedural function calls compatible with the native procedural interface of the operating system;]

(d) means, in [the]said computer, for processing said object-oriented statements [contained in the application and defined by the class library] by executing methods from [the]said object-oriented class library corresponding to [the]said object-oriented statements; and

(e) means, in [the]said object-oriented class library, including object-oriented[,] thread classes, for enabling [the]said object-oriented application to access [in an object-oriented manner]said [operating system] services to spawn, control, and obtain information relating to a thread[s] of execution.

2. (Amended) The apparatus of claim 1, wherein [the]said thread classes comprise a first object-oriented class encapsulating information necessary to create a new thread of execution, [the]said first class being an abstract base class, and a second object-oriented class for enabling [the]said application to spawn a new thread of execution on a task by passing a subclass of [the]said first class to an instance of [the]said second class, and for enabling [the]said application to terminate, suspend, resume, and schedule an existing thread of execution, [wherein instances of the]said second class having instances, said instances representing run-time processing entities in [the]said computer.

a1

3. (Amended) The apparatus of claim 1, wherein [the]said object-oriented class library comprises object-oriented[,] task classes for enabling [the]said application to access in an object-oriented manner [operating system]said services to reference and control a task[s], [wherein the] said task[s each] representing[s] an execution environment for at least one thread[s] of execution [respectively] associated with [the]said task[s].

4. (Amended) The apparatus of claim 3, wherein [the]said task classes comprise a first task object-oriented class encapsulating attributes and operations of an existing task, [the]said first task class including protected methods to enable run-time specific subclasses of [the]said first task class to spawn new tasks.

5. (Amended) The apparatus of claim 4, wherein [the]said first task class comprises methods for enabling [the]said application to determine whether a possible task exists, to suspend and resume [a]said existing task, to terminate [a]said existing task, to receive predetermined information relating to [a]said existing task, to identify a thread[s] of execution contained in [a]said existing task, and to perform predetermined virtual memory operations in an address space associated with [a]said existing task.

a'

6. (Amended) The apparatus of claim 5, wherein [the]said object-oriented class library comprises an object-oriented[,] thread class for enabling [the]said application to create a new thread of execution, [and wherein the]said task classes further comprising[e] a second object-oriented task class, derived from [the]said first task class, for enabling [the]said application to spawn a new[,] run-time specific task having a single thread of execution by passing an instance of [the]said thread class to an instance of [the]said second task class.

7. (Amended) The apparatus of claim 1, wherein [the]said object-oriented class library comprises object-oriented[,] synchronization classes for enabling [the]said application to access in an object-oriented manner [operating system] said services to synchronize execution of said thread[s] of execution.

L0

8. (Amended) The apparatus of claim 7, wherein [the]said synchronization classes define counting semaphores for use in synchronizing the execution of said thread of execution, [the]said synchronization classes comprising methods for enabling [the]said application to acquire one or more of [the]said counting semaphores in an exclusive mode or in a shared mode, and to release [the]said counting [acquired] semaphores after said counting semaphores are acquired.

a1

9. (Amended) The apparatus of claim 8, wherein [the]said counting semaphores are recoverable.

10. (Amended) The apparatus of claim 7, wherein [the]said synchronization classes define a monitor lock for use in synchronizing said thread of execution, [the]said synchronization classes comprising methods for enabling [the]said application to acquire and release [the]said monitor lock, and to block on a specified condition once [the]said monitor lock is acquired, [wherein the]said application releasing [relinquishes the]said monitor lock when [the]said application blocks on [the]said specified condition, and after [wherein the]said application is unblocked, due to satisfaction of said specified condition, [and] said application reacquiring [must reacquire the]said monitor lock before resuming execution[after the specified condition is satisfied].

11. (Amended) The apparatus of claim [7]10, wherein [the]said synchronization classes further comprise methods for enabling [the]said application to perform a broadcast operation on a specified blocking condition when [the]said application has acquired [the]said monitor lock, [wherein the]said broadcast operation unblocking[s] all of said threads of execution that are blocked on [the]said specified blocking condition.

61

12. (Amended) The apparatus of claim 7, wherein [the]said object-oriented class library comprises object-oriented[,] scheduling classes for enabling [the]said application to access in an object-oriented manner [operating system] said services to schedule execution of said thread[s] of execution.

13. (Amended) The apparatus of claim 12 in which an actual scheduling priority, a default scheduling priority, and a maximum scheduling priority are associated with [the]said application, [wherein the]said scheduling classes defining[e] one or more scheduling priorities, [the]said object-oriented class library including methods for setting each of [the]said actual, default, and maximum scheduling priorities of [the]said application to one of [the]said scheduling priorities.

14. (Amended) The apparatus of claim 13, wherein [the]said scheduling classes comprise an object-oriented class defining an idle scheduling priority adapted for use with said threads of execution that execute when [the]said computer is substantially idle.

15. (Amended) The apparatus of claim 13, wherein [the]said scheduling classes comprise an object-oriented class defining a responsive scheduling priority adapted for use with highly responsive threads of execution that [which] execute for short time periods, and that[which] block following [the]said short time periods.

867FH167

16. (Amended) The apparatus of claim 13, wherein [the]said scheduling classes comprise an object-oriented class defining an interaction scheduling priority adapted for use with highly responsive threads of execution implementing an interface between a human operator and [the]said computer.

17. (Amended) The apparatus of claim 13, wherein [the]said scheduling classes comprise an object-oriented class defining a long-term scheduling priority adapted for use with threads of execution that [which] execute for long periods of time.

a1

18. (Amended) The apparatus of claim 13, wherein [the]said scheduling classes comprise methods for enabling a task to specify a relative scheduling urgency of [the]said task.

19. (Amended) The apparatus of claim 1[7], wherein [the]said object-oriented class library comprises object-oriented[,] fault classes for enabling [the]said application to access in an object-oriented manner [operating system]said services to process system and user-defined processor faults.

20. (Amended) The apparatus of claim 19, wherein [the]said fault classes comprise a first object-oriented class defining a generic fault, [the]said first class having virtual methods for setting a processor computer program logic and a fault computer program logic to thereby identify [the]said generic fault, [the]said first class representing an abstract base class.

b3

21. (Amended) The apparatus of claim 20, wherein [the]said fault classes comprise a second object-oriented class, derived from [the]said first object-oriented class, comprising non-virtual methods for setting [the]said processor computer program logic and [the]said fault computer program logic in accordance with information specific to a particular fault of a particular processor such that [the]said second class represents a processor-specific fault, [wherein the]said non-virtual methods of [the]said second class overriding[e the] said virtual methods of [the]said first class.

22. (Amended) The apparatus of claim 21, wherein [the]said fault classes comprise an object-oriented class encapsulating information identifying a destination port, a fault message format, and fault types, [the]said object-oriented class comprising methods for enabling [the]said application to specify [the]said destination port, [the]said fault message format, and [the]said fault types, and for enabling [the]said application to instruct [the]said operating system to send messages in [the]said specified fault message format to [the]said specified destination port when [the]said specified fault types occur.

23. (Amended) The apparatus of claim 19, wherein [the]said fault classes comprise a first object-oriented class comprising methods for obtaining and returning a processing state of [a]said thread of execution.

24. (Amended) The apparatus of claim 23, wherein [the]said fault classes comprise a second object-oriented class having methods for enabling [the]said application to receive fault messages and to respond to received fault messages, [the]said fault messages comprising information identifying a faulting task and said faulting task's faulting thread of execution, and/or information of a faulting thread of execution's state, [the]said faulting thread of execution's state being obtained by calling [the]said methods of [the]said first class.

a1

105

25. (Amended) An apparatus for providing an object-oriented interface to a procedural operating system, said system providing services including procedural functions saved as executable program logic that are called to access said services [having a native procedural interface], [the]said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer;

(c) a code library, stored in [the]said memory component, comprising means for storing said executable program logic in an object-oriented class library; means for interfacing to an object-oriented application; and [computer program logic implementing an object-oriented class library, the object-oriented class library comprising related object-oriented classes for enabling an object-oriented application to access in an object-oriented manner services provided by the operating system, the object-oriented classes comprising methods for accessing the operating system services using procedural function calls compatible with the native procedural interface of the operating system; wherein object-oriented statements defined by the] said object-oriented class library defining object-oriented statements, said statements [are] insertable into [the]said application to enable [the]said application to access[in an object-oriented manner the operating system] said services during run-time execution of [the]said application in [the]said computer; and

(d) means, in [the]said object-oriented class library, including object-oriented, thread classes for enabling [the]said application to access in an object-oriented manner operating system services to spawn, control, and obtain information relating to a thread[s] of execution.

867FH171

26. (Amended) The apparatus of claim 25, wherein [the]said object-oriented class library comprises object-oriented[,] task classes for enabling [the]said application to access in an object-oriented manner [operating system] said services to reference and control a task[s], [wherein the]said task[s each] representing[s] an execution environment for at least one thread[s] of execution [respectively] associated with [the]said task[s].

a1

27. (Amended) The apparatus of claim 26, wherein [the]said object-oriented class library comprises object-oriented[,] synchronization classes for enabling [the]said application to access in an object-oriented manner [operating system] said services to synchronize execution of threads of execution.

28. (Amended) The apparatus of claim 27, wherein [the]said object-oriented class library comprises object-oriented, scheduling classes for enabling [the]said application to access in an object-oriented manner [operating system] said services to schedule execution of said thread[s] of execution.

29. (Amended) The apparatus of claim 28, wherein [the]said object-oriented class library comprises object-oriented, fault classes for enabling [the]said application to access in an object-oriented manner [operating system] said services to process system and user-defined processor faults.

67

30. (Amended) An apparatus for providing an object-oriented interface to a procedural operating system, said system providing services including procedural functions saved as executable program logic that are called to access said services, [having a native procedural interface, the]said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer; and

(c) a code library, stored in [the]said memory component, comprising means for storing said executable program logic in an object-oriented class library; means for interfacing to an object-oriented application; and [computer program logic implementing an object-oriented class library, the object-oriented class library comprising related object-oriented, thread classes for enabling an object-oriented application to access in an object-oriented manner operating system thread services to spawn, control, and obtain information relating to threads, the object-oriented classes comprising methods for accessing the operating system thread services using procedural function calls compatible with the native procedural interface of the operating system;

wherein object-oriented statements defined by the] said object-oriented class library comprising object-oriented thread classes, said thread classes comprising methods for accessing said services during run-time execution of said application in said computer, said thread classes having statements, said statements [are] insertable into [the]said application to enable [the]said application to access [in an object-oriented manner the operating system thread] said services to spawn, control, and obtain information relating to threads of execution. [during run-time execution of the application in the computer.]

867FH173

31. (Amended) The apparatus of claim 30, wherein [the]said thread classes comprise a first object-oriented class encapsulating information necessary to create a new thread of execution, [the]said first class being an abstract base class, and a second object-oriented class for enabling [the]said application to spawn a new thread of execution on a task by passing a subclass of [the]said first class to an instance of [the]said second class, and for enabling [the]said application to terminate, suspend, resume, and schedule an existing thread of execution, [wherein instances of the]said second class having instances, said instances representing run-time processing entities in [thie]said computer.

a1

32. (Amended) An apparatus for providing an object-oriented interface to a procedural operating system, said system providing services including procedural functions saved as executable program logic that are called to access said services, [having a native procedural interface, the] said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer; and

a1

(c) a code library, stored in [the]said memory component, comprising means for storing said executable program logic in an object-oriented class library; means for interfacing to an object-oriented application; and[computer program logic implementing an object-oriented class library, the object-oriented class library comprising object-oriented, task classes for enabling an object-oriented application to access in an object-oriented manner operating system task services to reference and control tasks, each of the tasks representing an execution environment for threads respectively associated with the tasks, the object-oriented classes comprising methods for accessing the operating system task services using procedural function calls compatible with the native procedural interface of the operating system; wherein object-oriented statements defined by the] said object-oriented class library comprising object-oriented task classes, said task classes comprising methods for accessing said services during run-time execution of said application in said computer, said task classes having statements, said statements [are] insertable into [the]said application to enable [the]said application to access [in an object-oriented manner the operating system task] said services to reference and control a task, said task representing an execution environment for at least one thread of execution associated with said task, [during run-time execution of the application in the computer.]

-10

33. (Amended) The apparatus of claim 32, wherein [the]said task classes comprise a first object-oriented class encapsulating attributes and operations of an existing task, [the]said first class including protected methods to enable run-time specific subclasses of [the]said first class to spawn new tasks.

a1

34. (Amended) The apparatus of claim 33, wherein [the]said first class comprises methods for enabling [the]said application to determine whether a possible task exists, to suspend and resume [a] said existing task, to terminate [a]said existing task, to receive predetermined information relating to [a]said existing task, to identify a thread[s] of execution contained in [a] said existing task, and to perform predetermined virtual memory operations in an address space associated with [a] said existing task.

35. (Amended) The apparatus of claim 34, wherein the object-oriented class library comprises an object-oriented[,] thread class for enabling [the]said application to create a new thread of execution, and [wherein the]said task classes further comprising[e] a second object-oriented task class, derived from [the]said first class, for enabling [the]said application to spawn a new[,] run-time specific task having a single thread of execution by passing an instance of [the]said thread class to an instance of [the]said second task class.

867FH176

36. (Amended) An apparatus for providing an object-oriented interface to a procedural operating system, said system providing services including procedural functions saved as executable program logic that are called to access said services, [having a native procedural interface, the]said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer; and

a1

(c) a code library, stored in [the]said memory component, comprising means for storing said executable program logic in an object-oriented class library; means for interfacing to an object-oriented application; and [computer program logic implementing an object-oriented class library, the object-oriented class library comprising object-oriented, synchronization classes for enabling an object-oriented application to access in an object-oriented manner operating system synchronization services to synchronize execution of threads, the object-oriented classes comprising methods for accessing the operating system synchronization services using procedural function calls compatible with the native procedural interface of the operating system; wherein object-oriented statements defined by the] said object-oriented class library comprising object-oriented synchronization classes, said synchronization classes comprising methods for accessing said services during run-time execution of said application in said computer, said synchronization classes having statements, said statements [are] insertable into [the]said application to enable [the]said application to access [in an object-oriented manner the operating system synchronization]said services to synchronize execution of threads of execution, [during run-time execution of the application in the computer.]

72

37. (Amended) The apparatus of claim 36, wherein [the]said synchronization classes define counting semaphores for use in synchronizing the execution of said thread of execution, [the]said synchronization classes comprising methods for enabling [the]said application to acquire one or more of [the]said counting semaphores in an exclusive mode or in a shared mode, and to release [the]said [acquired] counting semaphores after said counting semaphores are acquired.

a1

38. (Amended) The apparatus of claim 37, wherein [the]said counting semaphores are recoverable.

39. (Amended) The apparatus of claim 37, wherein [the]said synchronization classes define a monitor lock for use in synchronizing said thread of execution, [the]said synchronization classes comprising methods for enabling [the]said application to acquire and release [the]said monitor lock, and to block on a specified condition once [the]said monitor lock is acquired, [wherein the]said application releasing [relinquishes the]said monitor lock when [the]said application blocks on [the]said specified condition, and after [wherein the]said application is unblocked, due to satisfaction of said specified condition, [and must reacquire the]said application reacquiring said monitor lock before resuming execution.[after the specified condition is satisfied].

40. (Amended) The apparatus of claim 39, wherein [the]said synchronization classes further comprise methods for enabling [the]said application to perform a broadcast operation on a specified blocking condition when [the]said application has acquired [the]said monitor lock, [wherein the]said broadcast operation unblocking[s] all of said threads of execution that are blocked on [the]said specified blocking condition.

73

41. (Amended) An apparatus for providing an object-oriented interface to a procedural operating system, said system providing services including procedural functions saved as executable program logic that are called to access said services, [having a native procedural interface, the]said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer; and

(c) a code library, stored in [the]said memory component, comprising means for storing said executable program logic in an object-oriented class library; means for interfacing to an object-oriented application; and [computer program logic implementing an object-oriented class library, the object-oriented class library comprising object-oriented, scheduling classes for enabling an object-oriented application to access in an object-oriented manner operating system scheduling services to schedule execution of threads, the object-oriented classes comprising methods for accessing the operating system scheduling services using procedural function calls compatible with the native procedural interface of the operating system; wherein object-oriented statements defined by the] said object-oriented class library comprising object-oriented, scheduling classes, said scheduling classes comprising methods for accessing said services during run-time execution of said application in said computer, said scheduling classes having statements, said statements [are] insertable into [the]said application to enable [the]said application to access [in an object-oriented manner the operating system scheduling]said services, [during run-time execution of the application in the computer.]

74

867FH179

42. (Amended) The apparatus of claim 41 in which an actual scheduling priority, a default scheduling priority, and a maximum scheduling priority are associated with [the]said application, [wherein the]said scheduling classes defining[e] one or more scheduling priorities, [the]said object-oriented class library including methods for setting each of [the]said actual, default, and maximum scheduling priorities of [the]said application to one of [the]said scheduling priorities.

a1

43. (Amended) The apparatus of claim 42, wherein [the]said scheduling classes comprise an object-oriented class defining an idle scheduling priority adapted for use with said threads of execution that execute when [the]said computer is substantially idle.

44. (Amended) The apparatus of claim 43, wherein [the]said scheduling classes comprise an object-oriented class defining a responsive scheduling priority adapted for use with highly responsive threads of execution that [which] execute for short time periods, and that[which] block following [the]said short time periods.

45. (Amended) The apparatus of claim 44, wherein [the]said scheduling classes comprise an object-oriented class defining an interaction scheduling priority adapted for use with highly responsive threads of execution implementing an interface between a human operator and [the]said computer.

75

46. (Amended) The apparatus of claim 44, wherein [the]said scheduling classes comprise an object-oriented class defining a long-term scheduling priority adapted for use with threads of execution that[which] execute for long periods of time.

47. (Amended) The apparatus of claim 44, wherein [the]said scheduling classes comprise methods for enabling a task to specify a relative scheduling urgency of [the]said task.

867FH181

48. (Amended) An apparatus for providing an object-oriented interface to a procedural operating system, said system providing services including procedural functions saved as executable program logic that are called to access said services, [having a native procedural interface, the]said apparatus comprising:

(a) a computer;

(b) a memory component in [the]said computer; and

(c) a code library, stored in [the]said memory component, comprising means for storing said executable program logic in an object-oriented class library; means for interfacing to an object-oriented application; and [computer program logic implementing an object-oriented class library, the object-oriented class library comprising object-oriented, fault classes for enabling an object-oriented application to access in an object-oriented manner operating system fault services to process system and user-defined processor faults, the object-oriented classes comprising methods for accessing the operating system fault services using procedural function calls compatible with the native procedural interface of the operating system; wherein object-oriented statements defined by the] said object-oriented class library comprising object-oriented fault classes, said fault classes comprising methods for accessing said services during run-time execution of said application in said computer, said fault classes having statements, said statements [are] insertable into [the]said application to enable [the]said application to access [in an object-oriented manner the operating system fault]said services to process system and user-defined processor faults, [during run-time execution of the application in the computer.]

49. (Amended) The apparatus of claim 48, wherein [the]said fault classes comprise a first object-oriented class defining a generic fault, [the]said first class having virtual methods for setting a processor computer program logic and a fault computer program logic to thereby identify [the]said generic fault, [the]said first class representing an abstract base class.

867FH182

50. (Amended) The apparatus of claim 49, wherein [the]said fault classes comprise a second object-oriented class, derived from [the]said first object-oriented class, comprising non-virtual methods for setting [the]said processor computer program logic and [the]said fault computer program logic in accordance with information specific to a particular fault of a particular processor such that [the]said second class represents a processor-specific fault, [wherein the]said non-virtual methods of [the]said second class override [the]said virtual methods of [the]said first class.

a1

51. (Amended) The apparatus of claim 49, wherein [the]said fault classes comprise an object-oriented class encapsulating information identifying a destination port, a fault message format, and fault types, [the]said object-oriented class comprising methods for enabling [the]said application to specify [the]said destination port, [the]said fault message format, and [the]said fault types, and for enabling [the]said application to instruct [the]said operating system to send messages in [the]said specified fault message format to the specified destination port when [the]said specified fault types occur.

52. (Amended) The apparatus of claim 49, wherein [the]said fault classes comprise a first object-oriented class comprising methods for obtaining and returning a processing state of [a]said thread of execution.

53. (Amended) The apparatus of claim 52, wherein [the]said fault classes comprise a second object-oriented class having methods for enabling [the]said application to receive fault messages and to respond to received fault messages, [the]said fault messages comprising information identifying a faulting task and said faulting task's faulting thread of execution, and/or information of a faulting thread of execution's state, [the]said faulting thread of execution's state being obtained by calling [the]said methods of [the]said first class.

a1

REMARKS

The specification was amended to assure clarity and the claims have been amended to enhance clarity and definiteness.

Rejections under 35 USC § 112, 1st Paragraph

1. After reviewing the M.P.E.P. §§ 706.03(c) and 706.03(z) Applicant has carefully reviewed the Claims and appropriate amendments have been made to particularly point out and distinctly claim the invention in clear and definite terms.

The crux of the invention is an apparatus for enabling or providing an object-oriented application to access a procedural operating system for specific system services such as thread services, task services, synchronization services, scheduling services and fault handling services. The invention allows an object-oriented application to access a procedural operating system with procedural functions which are called to access services provided by the procedural operating system during run-time execution of the application. Specifically, object-oriented statements which access a service provided by the procedural operating system are located and translated into procedural function calls compatible with the procedural functions which are called to access services provided by the procedural operating system. Then, the procedural functions are executed in the computer to thereby invoke the procedural operating system to provide the service on behalf of the object-oriented application. Specifically, the object-oriented application accesses a code library to obtain the executable procedural functions and substitutes them into the application to execute a computer compiled, executable program logic to access services provided by said procedural operating system during run-time execution of the application in the computer. Alternative embodiments utilizing pointers, specialized hardware or a background task are pointed out as alternative translators on page 14, lines 21 to 34. This application applies the above to procedural operating system services to support threads of execution, tasks, scheduling services, synchronization services, and fault support services.

The Examiner has required that the means for performing the invention be "clearly defined interactions specifically indicated between the claimed components." Applicant respectfully reminds the Examiner of *In RE Donaldson Company, Inc.*, 29 USPQ2d 1845, 1848 (Fed. Cir. 1994) that held that "one construing means-plus-function language in a claim must look to the specification and interpret the language in light of the corresponding structure, material, or acts described therein, and equivalents thereof, to the extent that the specification provides such disclosure." This holding was reafferemed *in re Alappat* 31 USPQ2d 1545, 1554 (Fed. Cir. 1994). To further assist the Examiner, the means and structure are sufficiently described in the specification to enable one skilled in the art to practice the invention as follows:

1) Object-oriented Technology — pages 1–4 and associated referenced documents.

2) Code Library Operation — pages 9, and 11–14.

2) Threads — pages 15–16, 29–31 and associated referenced documents.

2) Tasks — pages 15, 17–18, 31–34 and associated referenced documents.

3) Thread synchronization — pages 24–25 and 44–46.

4) Scheduling — pages 25 and 46–47.

5) Faults — pages 25–26 and 47–49.

The details of this processing are initiated in the application on page 1 where a definition of a method in Object-Oriented Technology is provided. Then, a description of how message processing is used to invoke a method is presented at the top of page 2. The specific use of a method in accordance with these Claims is provided on page 9, lines 17–30 with reference to Figures 1–4; pages 11–14 in the Operational Overview of a Preferred Embodiment with reference to Figures 1–4; more details are found in the Wrapper Class Library section commencing on page 27 at line 27 with reference to Figures 1–4 On pages 29–50 of the specification each of the classes of the class library as shown in Figure 4 is discussed with reference to a Booch diagram of the logic flow presented in Figures 5–16. Finally, source code from selected methods is presented on pages 50–55 to

clearly convey to one of ordinary skill in C++ how to make and use the invention in a preferred environment.

The Examiner asserts that only component (c) of claim 1 refers to "applicant's desired results of 'enabling an object-oriented application to access in an object-oriented manner a procedural operating system'."

The claims have been amended to more particularly bring out the structure in the inventive system which accomplishes the objects of the invention. For example, amended Claim 1 component (c) now specifies that a code library is provided in computer memory comprising means for storing program code in an object oriented class library and means for interfacing an object-oriented application to a procedural operating system by using that code. Component (d) includes thread classes within the class library. The class library is included in the code library stored in a computer's memory. These objects, in turn, contain data and logic (OOP methods) and request kernel services by using the contained logic (OOP methods) to execute selective combinations of the kernel commands to provide the requested services. Thus, from amended Claim 1, it is clear that the code library exists in the computer memory and is used to create OOP objects which, in turn, access the kernel directly to provide requested operating system services. Applicant believes sufficient structure has been described in the specification, as noted above, and in the amended claims to instruct one skilled in the art to make the invention. It is believed that this amended recitation clearly indicates how the inventive objects are achieved.

The remaining claims have been rejected for similar reasons as those noted with respect to Claim 1 and similar amendments have been made to Claims 25, 30, 32, 36, 41, and 48 to overcome these rejections. In order to avoid repetition, these arguments will not be repeated in detail.

Rejections under 35 USC § 112, 2nd Paragraph

2 Claims 1-53 have been rejected under 35 U.S.C. §112, second paragraph, for being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. The Examiner

finds the claims to be vague, indefinite and confusing finding these claims to be not clear as to what is being claimed.

To clarify what is being claimed, Applicant has first changed all of the occurrences of terms such as "the method" or "the application" to "said method" and "said application". Further, Applicant has replaced the "native procedural interface" language with by use of said object-oriented statements, said system providing services, including procedural functions saved as executable program logic that are called to access said services. This language is elaborated in the specification on page 7 at the bottom of the page, page 9, lines 23–30, with reference to Figures 1–4, page 11, line 27 to page 12, line 4.

The Examiner has objected to the original recitation of the "code library" in Claim 1 as insufficient to define the invention since a "code library" or data stored in a storage device is not considered to provide a new device. Consequently, the Examiner analogized the code library to data stored in a storage device asserting that if "code stored in a storage device was considered to provide for a new device, then each musical compact disk would be patentable." The examiner concluded that the code library was not entitled to patentable weight.

In response, Applicants have amended the recitation of the "code library". As amended, the code library now includes means for storing executable program logic in an object-oriented class library (described on page 9, line 16 through page 10, line 8) and means for interfacing an object-oriented application to a procedural operating system using the executable program logic contained in the library (described on pages 8–11). The structure in the specification relating to the means for processing object-oriented statements by executing methods from the class library and means for enabling an object-oriented application to access system services clearly do not support the inference that the contents of the "code library" are simply "data" Instead these means are the result of a program executing in a computer. A "code library" contains more than just data. It contains program code that when executed by a computer transforms that computer into a patentable device. In *Alappat*, 31 USPQ2d 1545, (Fed. Cir. 1994), the Federal Circuit has stated "We have held that such programming creates a new machine, because a general purpose computer in effect becomes a

special purpose computer once it is programmed to perform particular functions pursuant to instructions from program software." (*Alappat* at 1558.) Further, memory containing data structures (an organization for storing data in memory) has been found to be patentable (see *in re Lowry*, 32 USPQ2d 1031, (Fed. Cir. 1994)). Thus, the Examiner is mistaken first in analogizing a code library to a musical compact disk, second in equating data to executable code, and third in not recognizing that a computer executing a program can result in a new, useful and non-obvious apparatus.

Claims 1, 25, 30, 32, 36, 41, and 48 have now been amended to specify that the code library comprises classes which enable the application to create and manipulate objects that define and modify threads of execution including attaching threads to tasks, synchronization scheduling and fault services among others. The relationship of classes and objects in object-oriented programming is well-known and the classes used by the inventive system are described in detail in the specification. Consequently, it is believed that the construction and operation of the objects which result from these classes would be apparent to those skilled in the art. Therefore, amended Claims 1–53 properly comply with the requirements of 35 U.S.C. §112, second paragraph.

3. The Examiner has rejected claims 1–53 under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claim 15 of Patent number 5,404,529 and claim 3 of 5,379,432. The Examiner has fundamentally misinterpreted the concept of threads in modern computer operating system terminology and has used a dictionary definition of a thread as "a process that is part of a larger program, with pointers identifying the parent node and utilized for traversing a tree in a tree data structure." Applicant uses thread or thread of execution with the meaning as used in *Inside OS/2* (by Grodon Letwin, ©1988, ISBN 1-55615-117-9 page 69). "The CPU reads each instruction in sequence and executes it. The passage of the CPU through the instruction sequence is called a *thread of execution*." *Object Oriented Design with Applications* (by Grady Booch, ©1991, ISBN 0-8053-0091-0) defines thread of control as "A single process. The start of a thread of control is the root from which independent dynamic action within a system occurs; a given system may have many

simultaneous threads of control, some of which may dynamically come into existence and then cease to exist. Systems executing across multiple CPUs allow of truly concurrent threads of control, whereas systems running on a single CPU can only achieve the illusion of concurrent threads of control." Further, and most importantly, threads are defined and explained in the instant application on page 15-16. "The executable entity in Mach is known as a *thread*." "A thread is always contained in a task, which represents most of the major resources (e.g., address space) of which the thread can make use. A thread has an execution state ..."

Applicant finds that the Examiner was mistaken in stating that claim 3 of '432 "is considered to indicate all the components of the applicant's claim except the 'thread classes'." The Examiner's mistake is when the definition of threads as actually used in the application is applied, then claim 3 of '432 does include the thread classes. Thus, Applicant has included a terminal disclaimer to limit the term of the patent resulting from the instant application to the term of 5,379,432.

However, based on the Examiner's mistaken definition of thread, the Examiner has asserted that "[t]he '529 reference refers to interprocess communication classes, which specifies (for example, in claim 15) that information can be inherited from other classes; this is considered (based on the definition of a thread) to for (sic) the same features a thread." The Examiner has missed one of the essential aspects of object-oriented programming methodology — that of inheritance. This concept of is explained starting on page 2, line 31 through page 3, line 21. Briefly, inheritance allows a subclass to "inherit" data attributes and methods of the subclass' parent class. Thus, a subclass specializes the base class by adding new code that overrides some of the features of the base class, or by adding additional features not available to the base class. The base class provides a significant portion of the functionality needed to implement common portions of a specific programming goal, the subclasses only need to provide the specialized detail required to actually achieve the programming goal.

Using the actual definition of thread, there is nothing involved with the interprocess communication classes of '529 that would suggest to one skilled in the art to combine interprocess communication classes and thread

classes to enable interprocess communication and control between parent and child classes to improve communications between related processes.

However, because the term of Patent No. 5,404,529 extends past the term of 5,379,432, for which a terminal disclaimer has been provided, the issue is moot.

Since all rejections, objections and requirements contained in the outstanding official action have been fully answered or traversed and shown to be inapplicable to the present claims, it is respectfully submitted that reconsideration is now in order under the provisions of 37 CFR §1.111(b) and such reconsideration is respectfully requested. Upon reconsideration, it is also respectfully submitted that this application is in condition for allowance and such action is therefore respectfully requested.

Should any additional issues remain, or if I can be of any additional assistance, please do not hesitate to contact me at (408) 777-5264.

Respectfully submitted,

TALIGENT, INC.

Keith Stephens (#32,632)
Director of Intellectual Property Law
10201 N. DeAnza Blvd.
Cupertino, CA 95014-2233
(408) 777-5264